Computers & Graphics 103 (2022) 61–74

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Dynamic projection mapping on deformable stretchable materials using boundary tracking

Muhammad Twaha Ibrahim *, Aditi Majumder, M. Gopi

Graphics and Visualization Lab, Donald Bren Hall, University of California, Irvine, 92697, USA

ARTICLE INFO

Article history:
Received 24 August 2021
Received in revised form 7 January 2022
Accepted 9 January 2022
Available online 18 January 2022

Keywords:
Dynamic projection mapping
Surface representation
Deformable and stretchable materials

ABSTRACT

We present the first method for dynamic projection mapping on rectangular, deformable and stretchable elastic materials using consumer-grade time of flight (ToF) depth cameras. We use a *B-Spline patch* to model the projection surface without explicitly modeling the deformation. Leveraging the nature of deformable and stretchable materials, we propose an efficient tracking method that can track the boundary of the surface material. This achieves realistic mapping even in the interior of the surface such that the projection appears to be printed on the material. The surface representation is updated in real-time using GPU based computations. Further, we also show that the speed of these updates is limited by the speed of the camera and therefore can be adopted for higher speed cameras as well.

1. Introduction

Projection mapping systems offer a unique way to augment the real-world with virtual data, modifying the appearance of physical objects using digital imagery. These systems enable novel applications such as surgical guidance, visualization, design and entertainment. Unlike projection mapping on static objects, projection mapping on dynamic and deformable objects presents several challenges, including accurate & high-speed reconstruction of the surface geometry, high-speed parameterization of the display surface and low-latency adaptive projection.

In order to avoid accurate reconstruction of the deformable surface, most prior works [1–3] use custom-designed hardware such as high-speed coaxial projector–camera pairs, often using multi-modal capture of IR, color and depth. Other works that reconstruct a deformable surface depend on modeling deformations using expensive optimization computations [4]. They operate at less than video frame rates and cannot address stretchable or elastic materials.

While [5,6] are able to project onto deformable, elastic surfaces without any perception of lag, they use high-speed co-axial projector–camera pairs and also require a precise marker pattern to be printed onto the surface using IR ink. [7,8] perform dynamic projection mapping using multiple projectors, but require the object to be rigid with known geometry.

1.1. Main contributions

In this paper, we propose a general framework for efficiently performing dynamic projection mapping of deformable, stretchable materials that is also backward compatible to non-deformable rigid materials. Specifically, our main contributions are as follows:

1. **Surface geometry representation**: We use rational B-spline patches to model the projection surface without explicitly modeling the deformation. This allows for integrating the modeling and updating of surface geometry, as well as warping of the projection image in a computationally efficient representation that lends itself for efficient GPU-based parallelization.
2. **Boundary-based tracking**: Distortions in the middle of deformable materials are less perceptible than at the boundaries [9]. We leverage this to track only the boundary of the material using simple markers (e.g. black dots) or depth-features with no explicit markers leading to projection mapping with or without markers.
3. **Mapping stretchable materials**: Using B-spline patch-based representations enable constrained projection mapping (e.g. length preserving mapping) to allow realistic mapping on stretchable materials.
4. **Using consumer-grade hardware**: Our system uses a consumer-grade time-of-flight (ToF) based depth camera (e.g. Azure Kinect, Pico Flexx) to achieve real-time mapping of the projected image on the deformable material such that it appears to be printed on the object. Further, we show that the computation speed is limited by the capture

* Corresponding author.
E-mail addresses: muhammti@uci.edu (M.T. Ibrahim), majumder@ics.uci.edu (A. Majumder), gopi@ics.uci.edu (M. Gopi).

https://doi.org/10.1016/j.cag.2022.01.004
0097-8493/

time, demonstrating that our method will be faster with higher speed cameras and projectors.

2. Related work

Projection mapping deals with altering appearances of 3D objects by projecting light on them using projectors. A large body of literature exists on projection mapping on static objects [10–20]. More recently, dynamic projection mapping (DPM), that allows mapping projected light on moving objects, has received much attention [1–8,21–34]. Prior work on dynamic projection mapping can be categorized based on the following: **(a)** *rigid* vs. *deformable* materials, **(b)** projection on *surfaces* vs. *objects*, **(c)** display on *textured* (e.g. using markers or patterns) vs. *texture-less* materials, and **(d)** using *consumer-grade components* vs. *specialized hardware* like a coaxial projector–camera setup.

2.1. Rigid surfaces

When considering projection mapping on dynamic rigid bodies [1–3,7,8,22,25,27–31], a large body of literature focuses only on rigid, *planar surfaces*. [2,3,27] use a *specialized* high-speed projector and camera in a co-axial manner to map the planar surface.

Texture-less surfaces: While [2] uses a high-speed color camera, [3] uses a high-speed depth camera and adjusts the focus based on the captured depth to keep the image focused on the plane as it moves back and forth.

Textured surfaces: [27] uses specialized digital micromirror device (DMD) hardware to project animated content onto *textured*, planar surfaces. The DMD allows projection of special patterns in an imperceptible way between projection frames to recover the four corners of the planar display and track them.

2.2. Rigid objects

Texture-less objects: Projection mapping on moving, rigid, non-planar objects requires computing the orientation and geometry of the objects, also known as shape and pose recovery, in addition to tracking the moving object. [1] uses an expensive, high-speed *coaxial projector–camera pair* to bypass shape recovery and performs projection mapping without any perceptible latency by segmenting the dynamic object from a retro-reflective background.

[22,25,28,29] use dynamic objects of *known* shapes while proposing different tracking methods using different kinds of *consumer-grade components*. [29] detects edges of *known* shapes and tracks them in real-time using a single IR camera to recover the object pose. [28] uses a sparse ICP registration technique to register 2D feature points detected by a standard RGB camera with a *known* 3D shape to track the object. However, the latency is not low enough to allow fast movement of the 3D object.

[7,8,26,30,31] perform dynamic projection mapping on rigid 3D objects using multiple projectors. [7,8] use a tightly calibrated system of 2–3 projectors with a single RGB-D camera to map a dynamic, complex 3D object whose shape has been recovered accurately apriori. Using a light-transport based optimization, key features of the point cloud captured by the RGB-D camera are detected and matched to the high-quality, *known* 3D shape. Though this method handles dynamic objects, it has high latency and therefore, the object can only move slowly. This system is extended in [30] to handle stray illumination due to geometric features and inter-reflections in a content-specific manner. Finally, [26] removes the requirement of tight calibration of the devices apriori by auto-calibrating devices while mapping projected images on the dynamic object.

Textured objects: [25] embeds 3D features as *markers* in the object during 3D printing that provide the features to be tracked in real-time. These embedded markers are made imperceptible during tracking via radiometric compensation.

In summary, all works addressing rigid dynamic objects assume a *known*, accurate 3D shape and cannot be used to project on dynamic, deformable and stretchable materials.

2.3. Deformable surfaces

Texture-less surfaces: [21] embeds retro-reflective markers in a surface that are only visible to an IR camera when an IR light source is shined onto it. They use the marker positions to determine the shape to project onto the surface. Similarly, [35,36] paint markers with IR ink and embed them in various gel-like substances. These markers are visible to an IR camera under IR lighting and are used to determine the deformation of the gel to perform projection mapping. Unlike [21,35,36] are able to handle interactive movement and update the projection in real-time. However, the gel-like substances are small and lie on a flat surface, restricting projection mapping to only those regions.

[10] presents a scalable system that can accommodate any number of projectors and cameras and can handle surfaces with any 3D shape without any prior system calibration. Therefore, the system recalibrates with deformations. However, this system can only achieve fast non-real-time recalibration and therefore cannot handle dynamic objects.

[34,37] use an image of a deformable fabric captured by an RGB camera and use prior techniques [38,39] to process the image properties (e.g. optical flow) to create an enhanced image that alter the perceived motion or stiffness of the fabric. However, these methods work with precise known motions of the deformable surfaces and do not attempt to capture either the motion or the actual geometry of the deformable surface.

[4] addresses deformable surfaces using a projector and RGB-D camera pair. They model the surface deformations (rather than shape) and recover the deformation parameters from the captured point cloud using GPU-based optimizations. However, the deformation model used does not allow stretchable surfaces and the optimizations cannot achieve real-time updates to accommodate fast moving materials.

Textured surfaces: [5,6] achieve projection mapping on dynamic, deformable and stretchable materials using a *specialized*, tightly calibrated, expensive, high-frame rate (∼500fps) projector–IR camera system that tracks dot cluster markers printed in the material with invisible IR ink. This results in conformal projection with imperceptible lag that can handle partial occlusion of the surface as well.

2.4. Deformable objects

Texture-less objects: [24] uses a *specialized*, high-speed projector along with three high speed cameras, all arranged in a co-axial manner using mirrors so that they share the same center of projection. Three NIR light sources of different bands are used to illuminate the object. The three cameras are equipped with color filters to sense each of these three light sources of different bands. The light sources and cameras together use photometric stereo to detect the normals at every pixel location in the camera space. These normals are then used to compute the illumination augmentation required to change the appearance of the object to that of a material with different normals. Due to co-axial arrangements of the projector–camera setup, the correction can be achieved in real time in the camera space irrespective of the dynamic object that is being mapped – either rigid, deformable or fluids – without perceptible lag.

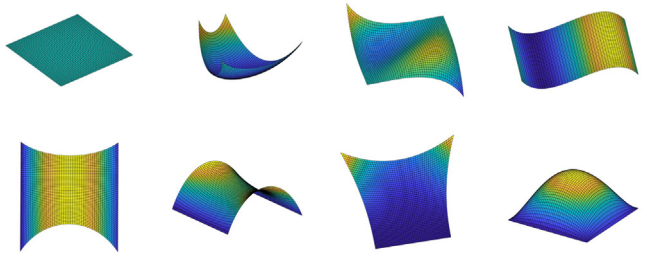


Fig. 1. Various surfaces that can be represented by a cubic B-Spline patch, including (bottom-row) surfaces stretched down the middle, at the upper two corners and outward from the center.

[23,33] specifically address faces. [33] uses a *specialized* co-axial fast projector–camera system along with LED-based lighting to track facial features in IR. The tracked 2D facial features are used to compute the deformation parameters of the 2D mesh of a face. [23] uses multiple projectors and an RGB-D camera in a tightly calibrated system to accurately estimate the 3D mesh of a face. During projection mapping, they update the deformation parameters of the mesh in real-time to register it with the depth camera output. Use of fast cameras in [33] allows imperceptible lag while [23] achieves almost real-time performance with low latency. [32] uses a Kinect in a specific retail dressing room kind of setting to segment multiple users using depth and projects different T-shirt designs on each person. However, the accuracy of projection is extremely low and therefore, the projection does not stick to the objects precisely and flickers continuously.

Prior work on tracking deformable objects in real-time using an RGB-D camera [35,40–42] cannot be used with dynamic projection mapping since the projection on the surface interferes with the tracking. Therefore, IR camera-based tracking is typically used for dynamic deformable objects. Thus, most works that perform dynamic projection mapping on deformable, texture-less objects either use specialized hardware [24,33] or are limited to a specific class of deformable objects (e.g. faces) as in [23].

2.5. Proposed work

Our work is closest to [4] but instead of modeling deformations, we model the surface geometry using a rational B-spline patch (see Fig. 1). We use rational B-spline patches because they **(i)** provide a framework to represent all kinds of smooth, deformable surfaces, including stretchable elastics, **(ii)** offer a convenient way to parameterize the surface, **(iii)** lend themselves to accurate computation on GPU for real-time performance, and **(iv)** provide the flexibility to change the surface representation (e.g. linear to quadratic etc.) at run-time without any changes to the system.

Additionally, unlike [1,2,5,24,27,33], we do not need *specialized* hardware such as co-axial projector–camera pairs, or high-speed, expensive equipment. Instead, we achieve real-time performance using consumer-grade hardware. While consumer-grade hardware is more accessible and affordable, they provide noisier data at lower spatial resolutions and frame-rates. This puts further demands on the system, which must perform projection mapping faster than the hardware frame-rate to avoid lag, from lower resolution data while being robust to noise.

Despite these limitations of consumer-grade hardware, our system is able to achieve the above objectives with a standard ToF depth camera that is augmented with a registered IR camera calibrated together with a projector. Using an IR-Depth camera enables us to track simple markers (e.g. black dots) leading to a marker-based system, or to track depth features leading to a marker-less system. Like [29], we use boundary-based features (e.g. edges or corners), but to determine the deformable, stretchable surface geometry rather than deformation as in [43,44]. Therefore, we present the first general framework for projection mapping of deformable materials, including stretchable elastics, using a single projector–camera system in real-time, operating at speeds up to 45fps. Since we depend on per frame surface representation using B-spline patches, we can handle rigid materials as well.

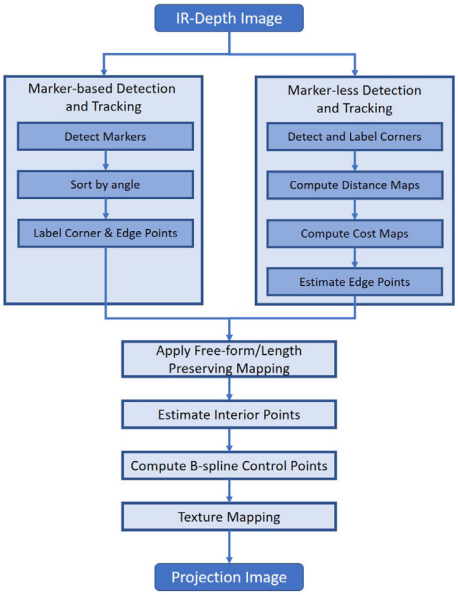


Fig. 2. Pipeline of our system, showing the marker-based and marker-less algorithms.

This paper is an extended version of the conference paper [45]. In [45], the authors use a non-rational bi-cubic Bezier patch that severely restricts the surface shapes that can be modeled accurately. In this work, we use rational B-spline patches, which allow accurate representation of even more complex shapes (e.g. waves, conics) that a non-rational bi-cubic Bezier patch cannot represent. Furthermore, we exploit the projective invariance of rational B-spline patches to achieve better surface conformity and improved our rendering pipeline to decrease the end-to-end latency. Finally, we provide more results and a deeper evaluation of the performance of our system.

3. System overview

The goal of our projection mapping system is to compute a texture-mapping function $\Omega(q) = p$ that maps a texture coordinate $q \in \mathbb{R}^2$ of a target image I_T to a texture coordinate $p \in \mathbb{R}^2$ of a source image I_S such that I_T conforms to the shape of the projection surface when it gets projected:

$$I_T(q) = I_S(\Omega(q)). \tag{1}$$

In this section, we explain how we compute $\Omega(\cdot)$ by tracking the boundaries of a deformable, stretchable display surface with or without markers, in real-time. The complete pipeline of our method is shown in Fig. 2.

3.1. System hardware

Our dynamic projection mapping system consists of a projector, an IR camera and a depth camera (Fig. 3) that have been

Table 1
Important notation used for B-spline patch.

Variable	Definition
H	Number of B-spline patch control points
T	B-spline knot vector
(M_u, M_v)	Number of markers in the horizontal and vertical directions respectively
(n, m)	Degree of B-spline patch
(u, v)	B-spline patch parameters
$\bar{\mathbf{Q}}(u, v)$	3D homogeneous point on the B-spline patch at (u, v)
$\bar{\mathbf{q}}(u, v)$	2D homogeneous point corresponding to $\bar{\mathbf{Q}}(u, v)$
$Q(u, v)$	3D point on the B-spline patch at (u, v)
$q(u, v)$	2D projector pixel corresponding to $Q(u, v)$
$\boldsymbol{N}(u, v)$	B-spline patch basis function evaluated at (u, v)
$\boldsymbol{C}$	$\mathbb{R}^{4\times H}$ matrix of 3D homogeneous B-spline patch control points
$\bar{\boldsymbol{C}}_p$	$\mathbb{R}^{3\times H}$ matrix of 2D homogeneous B-spline patch control points

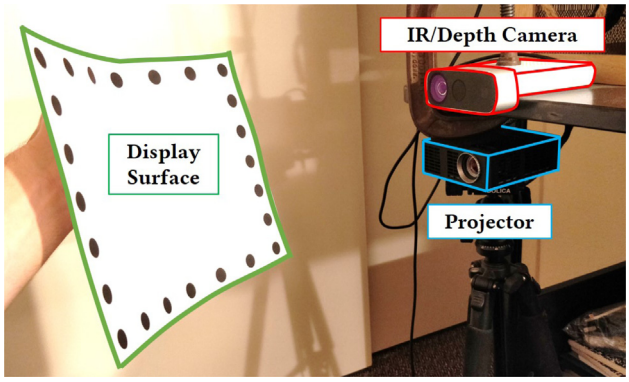


Fig. 3. Our setup: a geometrically calibrated projector (blue) and IR/Depth camera (red) pair along with the display surface (green).

geometrically calibrated for their intrinsic and extrinsic parameters using the software implementation of [46,47]. We assume our projection surface to be a smooth, deformable, elastic rectangle that is visible to the camera without occlusion. Each frame of the IR-depth camera is used to track feature points on the boundary of the deformable projection surface. These features can be explicitly marked using color (e.g. black dots) or by tracking depth features at the boundary of the projection surface. The former results in a system with visible boundary markers (Section 3.3), while the latter yields a marker-less system (Section 3.5). These features are used to compute the parameters of a rational B-spline patch representing the deformable surface shape. The fitted patch is then used to determine the texture-mapping function $\Omega(\cdot)$. Finally, the projection image is warped accordingly and projected onto the display surface in a manner that conforms to the deformable shape.

3.2. B-spline patch based model

A rational B-spline patch of degree (n, m) in d dimensions is defined in parameters (u, v) by a set of $H = (r+1) \times (s+1)$ control points $C_{0,0}, C_{0,1}, \ldots, C_{(r,s-1)}, C_{(r,s)}$ and weights $W_{0,0}, W_{0,1}, \ldots, W_{(r,s-1)}, W_{(r,s)}$ as:

$$\begin{aligned} Q(u,v) &= \sum_{i=0}^{r}\sum_{j=0}^{s} P_{i,j}(u,v)C_{i,j}, \\ P_{i,j}(u,v) &= \frac{N_{i,n}(u)N_{j,m}(v)W_{i,j}}{\sum_{k=0}^{r}\sum_{l=0}^{s} N_{k,n}(u)N_{l,m}(v)W_{k,l}}, \end{aligned} \tag{2}$$

where $W_{i,j} \in \mathbb{R}$, $C_{i,j}, Q(u,v) \in \mathbb{R}^d$. $N_{i,n}(u)$ is the B-spline basis function of degree n. It is defined by a non-decreasing knot vector $T = \{t_0, t_1, \ldots, t_h\}$ as:

$$\begin{aligned} N_{i,0}(u) &= \begin{cases} 1, & t_i \le u < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \\ N_{i,n}(u) &= \frac{u - t_i}{t_{i+n} - t_i} N_{i,n-1}(u) + \frac{t_{i+n+1} - u}{t_{i+n+1} - t_{i+1}} N_{i+1,n-1}(u). \end{aligned} \tag{3}$$

Note that the number of control points r, the B-spline degree n and the size of the knot vector h must satisfy the identity $h = r + n + 1$ for both directions (u, v). Thus, by changing the B-spline degree and/or the size of the knot vector, one can change the number of control points of the B-spline.

In our work, we model our projection surface as an open, uniform, rational B-spline patch of degree (n, m). In an *open uniform* B-spline of degree n, the first $n+1$ knots and the last $n+1$ knots are equal, while the remaining, internal knots are non-decreasing and equally spaced. In a *non-rational* B-spline, the weights of each control point are the same, limiting their ability to represent more complex shapes such as circles, conics and tori. These shapes can be accurately modeled with a *rational* B-spline, where the weights may not be equal. Furthermore, rational B-splines produce correct results under projective transformation, a property we leverage when computing $\Omega(\cdot)$ to render the image for projection. Thus, a B-spline patch can be used to represent various shapes as shown in Fig. 1 and is parameterized by (u, v) in the horizontal and vertical directions respectively (Fig. 4). In matrix form, a rational B-spline patch in 3D can be expressed as:

$$\begin{aligned} \bar{\mathbf{Q}}(u,v) &= \begin{bmatrix} X(u,v)\,W(u,v) \\ Y(u,v)\,W(u,v) \\ Z(u,v)\,W(u,v) \\ W(u,v) \end{bmatrix} = \boldsymbol{C}\,\boldsymbol{N}(\boldsymbol{u},\boldsymbol{v}) \\ \bar{\boldsymbol{C}} &= \begin{bmatrix} X_{00}W_{00} & X_{01}W_{01} & \ldots & X_{rs}W_{rs} \\ Y_{00}W_{00} & Y_{01}W_{01} & \ldots & Y_{rs}W_{rs} \\ Z_{00}W_{00} & Z_{01}W_{01} & \ldots & Z_{rs}W_{rs} \\ W_{00} & W_{01} & \ldots & W_{rs} \end{bmatrix} \in \mathbb{R}^{4\times H}, \\ \boldsymbol{N}(u,v) &= \begin{bmatrix} N_{0,n}(u)N_{0,m}(v) \\ N_{0,n}(u)N_{1,m}(v) \\ \vdots \\ N_{r,n}(u)N_{s,m}(v) \end{bmatrix} \in \mathbb{R}^{H\times 1}, \end{aligned} \tag{4}$$

where $H = (r+1) \times (s+1)$. Here, $\begin{bmatrix} X_{ij}W_{ij} & Y_{ij}W_{ij} & Z_{ij}W_{ij} & W_{ij} \end{bmatrix}$ is a 3D B-spline control point in homogeneous coordinates, and $Q(u,v) = \begin{bmatrix} X(u,v) & Y(u,v) & Z(u,v) \end{bmatrix}$ is a 3D point on the B-spline patch.

Representing the display surface using a B-spline patch lets us express $\Omega(\cdot)$ as a function of the (u, v) parameters. Using the control points $\bar{\boldsymbol{C}}$, we can compute $Q(u, v)$ for any (u, v) parameters. Since our system is geometrically calibrated, we can project $Q(u, v)$ onto the projector image plane to determine the 2D projector pixel $q(u, v)$. This creates a mapping between the projector pixel $q(u, v)$ and the display surface parameters (u, v):

$$\Omega(q(u,v)) = (u,v), \tag{5}$$

i.e. every pixel $q(u, v)$ in $\Omega(\cdot)$ maps to a display surface parameter (u, v). Substituting Eq. (5) into Eq. (1), we get:

$$I_T(q(u,v)) = I_S(u,v). \tag{6}$$

Thus, our goal is to compute $q(u, v)$ for any (u, v) display parameter. We achieve this by computing the control points of the B-spline patch that represent the display surface using a system of linear equations. The key challenge of our work is: **(i)** tracking the deformable, stretchable surface, **(ii)** determining

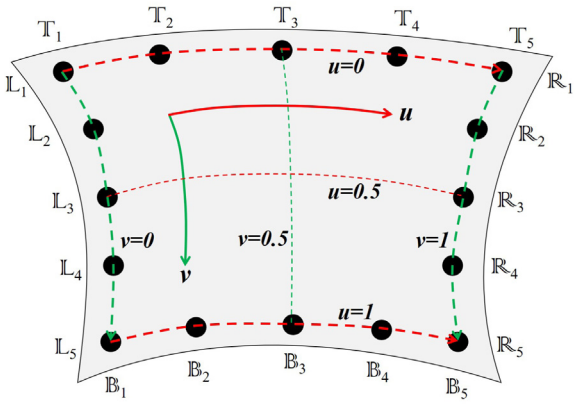


Fig. 4. Illustration of a deformable surface with dots as markers. The iso-parametric boundary curves at $u = 0$ (top), $u = 1$ (bottom), $v = 0$ (left) and $v = 1$ (right) are shown along with the boundary labels for each marker $(\mathbb{T}_i, \mathbb{B}_i, \mathbb{L}_j, \mathbb{R}_j)$. The (u, v) parameterization of the top-left corner is $(0, 0)$ and bottom-right corner is $(1, 1)$.

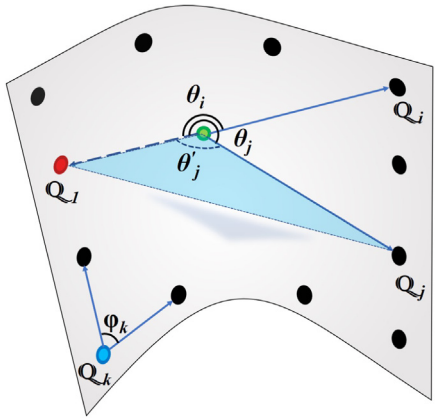


Fig. 5. Determining the top-left corner for a surface that is folded inward. $\mathbb{Q}_{avg}$ is marked in green. Initially, the detections are not ordered. Note that $\theta_i < \theta_j$ when using 2D image coordinates. However, using 3D coordinates, the angle θ'_j between the two vectors spans the plane connecting them rather than along the (u, v) surface. This causes $\theta'_j < \theta_i$ which is incorrect. While 2D image coordinates work better than 3D coordinates when relative ordering of angles is required, 3D coordinates are used to label a corner such as $\mathbb{Q}_k$ (blue marker, bottom-left) for which $\phi_k \approx 90°$, which may not be recognizable in 2D camera image space due to perspective distortion.

the (u, v) parameters of 3D points on the surface, **(iii)** setting up a system of linear equations using these 3D points, **(iv)** solving the system to compute the control points, **(v)** using the control points to compute $q(u, v)$, and **(vi)** perform texture-mapping for projection, all in *real-time*.

A method to track stretchable display surfaces using markers and determining the (u, v) parameters for the tracked points is detailed Section 3.3, while a method for markerless tracking is elaborated in Section 3.5. Setting up the system of linear equations and solving it to compute the control points of the B-spline patch is explained in Section 3.4. Computing $q(u, v)$ and performing texture-mapping is explained in Section 3.6. Important notation used for B-spline patches is explained in Table 1.

3.3. Marker-based tracking of stretchable surface

3.3.1. Marker placement

Our projection surface is a smooth rectangle made of a diffuse, deformable and elastic material. We define the boundaries of our surface with markers such that: (i) four markers specify the four corners of this rectangle, (ii) a fixed number of markers, known apriori, are placed equally spaced along each edge, (iii) including the corners, the top and bottom edges have equal number of markers M_u, and the left and right edges have equal number of markers M_v. We denote markers on the top and bottom edges with $\mathbb{T}_i$ and $\mathbb{B}_i$, $1 \leq i \leq M_u$ and markers on the left and right edges with $\mathbb{L}_j$ and $\mathbb{R}_j$, $1 \leq j \leq M_v$ respectively. Fig. 4 shows an example of our surface, where three markers are placed along each edge in addition to the corner markers. Note that the shape recovery does not depend on the surface being white and will therefore work for textured surfaces as well. However, this work does not focus on photometric corrections for textured surfaces. Therefore, any flat colored surface will yield acceptable projections.

Since the markers $\mathbb{T}_i, \mathbb{B}_i, \mathbb{L}_j$ and $\mathbb{R}_j$ are equally spaced on each edge, we can pre-compute their (u, v) parameters as $(u_i, 0)$, $(u_i, 1), (0, v_j)$ and $(1, v_j)$ respectively, where $u_i = \frac{i-1}{M_u-1}$, $v_j = \frac{j-1}{M_v-1}$. This arrangement gives us clear matches between the boundary markers and their (u, v) parameters on the B-spline patch.

3.3.2. Marker detection and tracking

The markers are tracked every frame and used to compute the 3-D control points of the B-spline patch. We use IR imagery in order to avoid interference caused by the visible projection. The markers, being black, are usually well-visible in the IR camera against the surface. In the first frame, we use a simple blob detector on the IR image to detect the markers. In subsequent frames, each detected marker is tracked using KLT feature tracking [48–50]. This results in fast and robust tracking of the markers. If tracking fails, we retract back to marker detection and repeat the procedure.

3.3.3. Assigning (u, v) parameters to markers

The detection and tracking step provides marker locations in the camera image space. However, this is not sufficient to establish the (u, v) parameters for each marker. This objective is achieved in the marker (u, v) assignment step, detailed as follows.

Sort by Angle: Let I_{IR} denote the IR-camera image and I_{PC} denote the point cloud image from the depth camera. Let $\mathbb{Q}$ denote the set of boundary markers and $\mathbb{Q}_i = \{q_i, Q_i\}$ denote ith marker, where $q_i \in \mathbb{R}^2$ is the 2D location of the marker in I_{IR} and $Q_i = I_{PC}(q_i) \in \mathbb{R}^3$ is the corresponding 3D point in the point cloud image. We compute the centroid of all the detected markers, $\mathbb{Q}_{avg}$, as:

$$\mathbb{Q}_{avg} = \{q_{avg}, Q_{avg}\} = \frac{1}{|\mathbb{Q}|} \sum_{i=1}^{|\mathbb{Q}|} \{q_i, Q_i\} \tag{7}$$

The first step is to sort the detected markers by the angle along the (u, v) surface, denoted by θ, from a reference vector connecting $\mathbb{Q}_{avg}$ to $\mathbb{Q}_1$. Note that the closest approximation of the (u, v) space is available in the 2D IR camera space. Despite perspective projection, the angle of the (u, v) coordinate of a marker from the reference vector will increase monotonically in the IR camera space as we move clockwise from one marker to the next. This invariance can be violated when using 3D coordinates (illustrated in Fig. 5). Therefore, we perform this sorting by angle in the 2-D IR camera space.

Labeling Corner and Edge Points: Sorting by θ provides the ordering of the markers around the surface, with which we can determine the adjacent markers for each marker. Therefore, if we can assign the (u, v) parameters of any one marker correctly, we can determine the parameters for all the remaining ones. So first, we identify the marker that corresponds to the top-left corner and assign $(u, v) = (0, 0)$ to it. We compute, for every marker

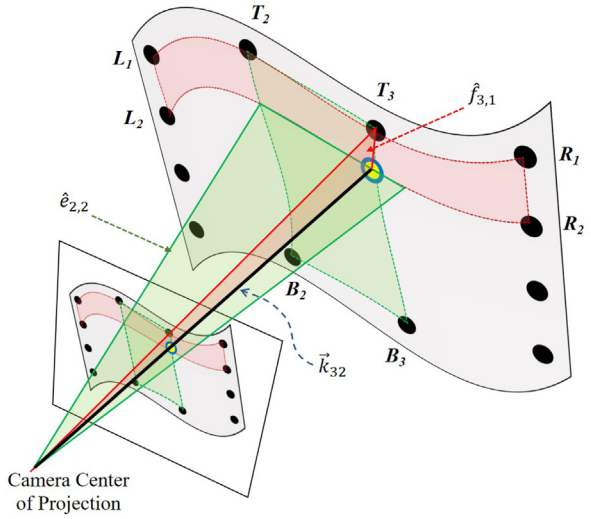


Fig. 6. Estimating interior points of the surface by normal interpolation. Markers used to compute the plane normals are labeled. The green region on the surface is the area scanned by the plane as it is interpolated between $\hat{t}_2^3$ and $\hat{b}_2^3$, while $\hat{e}_{2,2}$ is shown in solid green. The red region is the area scanned by the plane as it is interpolated between $\hat{l}_1^2$ and $\hat{r}_1^2$, while $\hat{f}_{3,1}$ is shown in solid red. The ray $\vec{k}_{32}$ generated by intersecting $\hat{e}_{2,2}$ and $\hat{f}_{3,1}$ is used to estimate the interior point Q_{32} (blue–yellow point).

$\mathbb{Q}_i$, the angle ϕ_i in 3D between the edges connecting it to its two adjacent neighbors $\mathbb{Q}_{i-1}$ and $\mathbb{Q}_{i+1}$ (Fig. 5):

$$\begin{aligned}\phi_i &= \Gamma(\vec{\alpha}_i, \vec{\beta}_i),\\ \vec{\alpha}_i &= Q_{i-1} - Q_i,\\ \vec{\beta}_i &= Q_{i+1} - Q_i,\end{aligned} \tag{8}$$

where $\Gamma(\vec{\alpha}, \vec{\beta})$ computes the angle between vectors $\vec{\alpha}, \vec{\beta}$.

By construction, corner points make an angle close to 90° with their neighbors and edge points make an angle close to 180°, resulting in two clusters of points where the number of points in the cluster $\phi_i \approx 90°$ would be four, corresponding to the four corners of the display. However, due to surface deformations, we may not get exactly four corners. Theoretically, if the stretching distortion is extreme at a corner, the corner marker can have an angle close to 180° between its horizontal and vertical edge neighbors leading to false negatives, and if the folding distortion is extreme, then an edge marker can have close to 90° with its neighbors along the same edge leading to false positives. We note that the stretching distortion is unrealistic and hence we can assume that there are no false negatives. With respect to the folding distortion and hence false positives, we can eliminate false positives and find the correct corners using the invariance that there are alternately $(M_u - 2)$ and $(M_v - 2)$ markers between the corner markers.

Note that any corner can be considered the top-left corner (with $(u, v) = (0, 0)$) as long as it is tracked in subsequent frames and other corners are labeled correctly with respect to the top-left corner. Let the four corners be $\mathbb{C}_k$, $1 \le k \le 4$. To determine the top-left corner, we approximate the distance along the surface between $\mathbb{C}_k$ and its adjacent clockwise corner $\mathbb{C}_{k+1}$ by summing the Euclidean distances between neighboring markers from $\mathbb{C}_k$ to $\mathbb{C}_{k+1}$. This results in two corners with a larger distance (the longer edges of the display) compared to the other two (the shorter edges of the display). From the corners with the two largest distances, the corner closest to the camera origin is labeled as the top-left corner and $(u, v) = (0, 0)$ is assigned to that marker. Then, the (u, v) parameters for all markers are assigned using the cyclic order of markers computed earlier.

All markers are labeled once during the initialization phase. In subsequent frames, each marker and its label is tracked, allowing our projection display to maintain correct orientation even when the surface is rotated upside down.

3.3.4. Estimating interior points

Boundary markers and their (u, v) assignments are not enough to compute the B-spline patch. We need additional 3D points and their (u, v) parameters in the interior of the projection surface where no markers exist. Therefore, we estimate an additional $(M_u - 2) \times (M_v - 2)$ interior 3D points and assign their (u, v) parameters at $(\frac{i-1}{M_u-1}, \frac{j-1}{M_v-1})$, where $2 \le i \le (M_u - 1), 2 \le j \le (M_v - 1)$.

In order to estimate these 3D points, we assume a piecewise linear representation of the boundary curves, which are also iso-parametric (see Fig. 4). Let $t_i, b_i, l_j, r_j \in \mathbb{R}^2$ denote the 2D coordinates of these markers in IR-camera space, and $T_i, B_i, L_j, R_j \in \mathbb{R}^3$ denote their 3D coordinates. Let the markers along each edge be denoted by $\mathbb{T}_i = \{t_i, T_i\}, \mathbb{B}_i = \{b_i, B_i\}, \mathbb{L}_j = \{l_j, L_j\}$ and $\mathbb{R}_j = \{r_j, R_j\}$. We compute the normals of the planes passing through every adjacent pair of these markers and the camera center of projection:

$$\begin{aligned}&\hat{t}_i^{i+1} = \zeta(T_i, T_{i+1}), \quad && \hat{l}_j^{j+1} = \zeta(L_j, L_{j+1}),\\ &\hat{b}_i^{i+1} = \zeta(B_i, B_{i+1}), && \hat{r}_j^{j+1} = \zeta(R_j, R_{j+1}),\\ &1 \le i \le (M_u - 1), && 1 \le j \le (M_v - 1),\end{aligned} \tag{9}$$

where $\zeta(K_1, K_2)$ computes the normal of the plane passing through 3D points K_1, K_2 and the camera center of projection. This leads to $(M_u - 1)$ normals for each of the top and bottom curves, and $(M_v - 1)$ normals for the left and right curves. Note that while the boundary markers are *tracked* every frame, the interior points and their (u, v) parameters are *estimated* for each frame.

We interpolate between each $\hat{t}_i^{i+1}$ and $\hat{b}_i^{i+1}$ to estimate the normals $\hat{e}_{i,j}$ for iso-parametric curves $v_j = \frac{j-1}{M_v-1}$:

$$\hat{e}_{i,j} = \hat{t}_i^{i+1}(1 - v_j) + \hat{b}_i^{i+1} v_j. \tag{10}$$

In Fig. 6, the green region shows the area scanned by the plane as its normal is interpolated between $\hat{t}_2^3$ and $\hat{b}_2^3$, while the interpolated plane $\hat{e}_{2,2}$ at $v_2 = \frac{1}{3}$ is shown in solid green. Similarly, we interpolate between $\hat{l}_j^{j+1}$ and $\hat{r}_j^{j+1}$ to estimate the normals $\hat{f}_{i,j}$ that pass through $u_i = \frac{i-1}{M_u-1}$:

$$\hat{f}_{i,j} = \hat{l}_j^{j+1}(1 - u_i) + \hat{r}_j^{j+1} u_i. \tag{11}$$

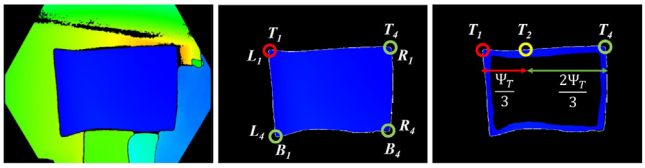
The red region in Fig. 6 shows the area scanned by the plane as its normal is interpolated between $\hat{l}_1^2$ and $\hat{r}_1^2$, while the interpolated plane $\hat{f}_{3,1}$ at $u_3 = \frac{2}{3}$ is shown in solid red.

These interpolated normals are used to estimate the interior 3D points at (u_i, v_j) for $2 \le i \le (M_u - 1), 2 \le j \le (M_v - 1)$. We compute the ray $\vec{k}_{ij}$ by intersecting the interpolated horizontal and vertical planes $\hat{e}_{(i-1,j)}$ and $\hat{f}_{(i,j-1)}$. Fig. 6 shows $\vec{k}_{32}$, the intersection of $\hat{e}_{2,2}$ and $\hat{f}_{3,1}$ as a black line. Since $\vec{k}_{ij}$ passes through the camera center of projection (COP), we find the 2D coordinate q_{ij} by projecting it onto the IR camera image plane. The 3D point at (u_i, v_j) can be determined from the depth camera as $Q_{ij} = I_{PC}(q_{ij})$. This step is performed in every iteration.

Note that whenever a stretch is applied to the display surface, *the distances between markers change in 3D but the distances in* (u, v)*-space do not*. This allows a B-spline patch to model stretches to the surface.

3.4. Computing control points

Once we have determined 3D points on the surface and their (u, v) assignments, we can compute the control points by solving



(a) (b) (c)

Fig. 7. Distance-based marker-less tracking. **(a)** The depth map. **(b)** The segmented point cloud along with the results of corner detection (the top-left corner is highlighted in red). **(c)** The boundary point cloud I_B. T_2 is the point on the top edge that is at a distance $\frac{1}{3}\Psi_T$ from T_1 (red circle) and $\frac{2}{3}\Psi_T$ from T_4 (green circle).

the following system of linear equations for the control points $\bar{\boldsymbol{C}}$:

$$\begin{aligned} \bar{\boldsymbol{Q}} &= \bar{\boldsymbol{C}}\boldsymbol{A}, \\ \bar{\boldsymbol{Q}} &= \begin{bmatrix} X(u_1,v_1) & X(u_2,v_2) & \dots & X(u_F,v_F) \\ Y(u_1,v_1) & Y(u_2,v_2) & \dots & Y(u_F,v_F) \\ Z(u_1,v_1) & Z(u_2,v_2) & \dots & Z(u_F,v_F) \\ 1 & 1 & \dots & 1 \end{bmatrix}, \\ \bar{\boldsymbol{C}} &= \begin{bmatrix} X_{00} & X_{01} & \dots & X_{rs} \\ Y_{00} & Y_{01} & \dots & Y_{rs} \\ Z_{00} & Z_{01} & \dots & Z_{rs} \\ 1 & 1 & \dots & 1 \end{bmatrix}, \\ \boldsymbol{A} &= \begin{bmatrix} \boldsymbol{N}(u_1,v_1) & \boldsymbol{N}(u_2,v_2) & \dots & \boldsymbol{N}(u_F,v_F) \end{bmatrix}, \end{aligned} \tag{12}$$

where F is the number of (u, v) assignments, $\bar{\boldsymbol{Q}} \in \mathbb{R}^{4\times F}$, $\bar{\boldsymbol{C}} \in \mathbb{R}^{4\times H}$, $\boldsymbol{N}(u, v) \in \mathbb{R}^{H\times 1}$ and $\boldsymbol{A} \in \mathbb{R}^{H\times F}$. Note that all the weights W_{ij} are set to 1, making this a non-rational B-spline.

In order to solve Eq. (12), the condition $F \geq H$ must be met. While $F = (M_u \times M_v)$, depends on the number of markers, the number of control points, H, depends on the B-spline degree and the size of the knot vector (see Section 3.2). By changing these two parameters, we can change the number of control points and use different B-spline models even if the number of markers remains the same (see Section 3.6.1 and Fig. 15).

3.5. Marker-less tracking of stretchable surface

In our display, the distance between adjacent markers is the same when no stretch is applied. However, in the absence of explicit markers, the *geodesic* distance between 3D points can be used to assign the (u, v) parameters. A *geodesic* is a curve representing the shortest path between two points on a surface and the *geodesic* distance is the length of that curve. In this section, we outline how we use this concept to create projection displays for marker-less, deformable and stretchable surfaces.

In each frame, we segment out the display surface from the point cloud image assuming there is no occlusion of the surface. This provides a mask that corresponds to the segmented surface (Fig. 7(b)). We extract the boundary mask by taking the difference of the surface segmentation mask with its morphological erosion and compute the boundary point cloud I_B (highlighted in blue in Fig. 7(c)).

While the markers provided explicit brightness features in our marked display, for a marker-less display, the only implicit features are the four corners of the display. We find them using the Harris corner detector [51] on the segmented mask and track them using the KLT feature tracker. Then, we assign the (u, v) parameters to these corners as outlined in Section 3.3.3.

3.5.1. Distance-based edge marker estimation

Recall that the parameters for a marker on the top edge, denoted by $\mathbb{T}_i$, is $(u_i, 0)$, where $u_i = \frac{i-1}{M_u-1}$. For the marker-less display, the goal is to find all $\mathbb{T}_i$s, even though they are not explicitly marked on the surface. From the corner detection step, we already have $\mathbb{T}_1$ and $\mathbb{T}_{M_u}$. Let Ψ_T denote the length of the top edge. Other $\mathbb{T}_i$s are found such that **(i)** they are along the top edge, **(ii)** their *geodesic* distance from $\mathbb{T}_1$ is $u_i\Psi_T$, and **(iii)** their *geodesic* distance from $\mathbb{T}_{M_u}$ is $(1 - u_i)\Psi_T$.

Let $\boldsymbol{G}$ denote the set of 3D points (from the point cloud image) on the geodesic between two points G_1 and G_{M_G}, where M_G is the number of points in $\boldsymbol{G}$. Let G_i denote the ith 3D point on the geodesic. The geodesic distance, $\sigma(\boldsymbol{G})$ of $\boldsymbol{G}$ is:

$$\sigma(\boldsymbol{G}) = \sum_{i=2}^{M_G} \|G_i - G_{i-1}\| \tag{13}$$

However, computing $\sigma(\boldsymbol{G})$ is a time consuming step that cannot exploit GPU-based parallelism. This increases the end-to-end latency of our system and causes a visible lag during projection mapping. To overcome this, we approximate the geodesic distance by computing the *Euclidean* distance between points on the surface instead, which can be computed rapidly on the GPU. The Euclidean distance $\psi(\boldsymbol{G})$ is:

$$\psi(\boldsymbol{G}) = \|G_{M_G} - G_1\| \tag{14}$$

Fig. 7(c) shows the point $\mathbb{T}_2$ on the top edge at a Euclidean distance $\frac{1}{3}\Psi_T$ from $\mathbb{T}_1$ (the top-left corner) and $\frac{2}{3}\Psi_T$ from $\mathbb{T}_4$ (top-right corner).

Searching for 3D points in the boundary point cloud image I_B that are at specific Euclidean distances from the edge corners to assign their (u, v) parameters is slow and cannot exploit GPU-based parallelism. Instead, we compute distance maps that can be efficiently computed on the GPU. Let $S_{TL}, S_{TR}, S_{BL}, S_{BR}$ denote these distance maps containing the Euclidean distance of each 3D point in I_B from the corners top-left, top-right, bottom-left and bottom-right respectively:

$$\begin{aligned} S_{TL} &= \psi(I_B - T_1), \quad S_{TR} = \psi(I_B - T_{M_u}), \\ S_{BL} &= \psi(I_B - B_1), \quad S_{BR} = \psi(I_B - B_{M_u}). \end{aligned} \tag{15}$$

Let $\Psi_T, \Psi_L, \Psi_B, \Psi_R$ denote the lengths of the top, left, bottom and right edges of the display surface. Finally, let $\omega_T(u_i), \omega_B(u_i)$ denote the cost maps of assigning parameter u_i to each 3D point along the top and bottom edges respectively and $\omega_L(v_j), \omega_R(v_j)$ denote the cost maps of assigning parameter v_j to each 3D point along the left and right edges respectively, where $v_j = \frac{j-1}{M_v-1}$. We compute these cost maps as:

$$\begin{aligned} \omega_T(u_i) &= |S_{TL} - u_i\Psi_T| + |S_{TR} - (1-u_i)\Psi_T|, \\ \omega_B(u_i) &= |S_{BL} - u_i\Psi_B| + |S_{BR} - (1-u_i)\Psi_B|, \\ \omega_L(v_j) &= |S_{TL} - v_j\Psi_L| + |S_{BL} - (1-v_j)\Psi_L|, \\ \omega_R(v_j) &= |S_{TR} - v_j\Psi_R| + |S_{BR} - (1-v_j)\Psi_R|. \end{aligned} \tag{16}$$

The markers $\mathbb{T}_i$, $\mathbb{B}_i$, $\mathbb{L}_j$ and $\mathbb{R}_j$ are the points with the minimum cost and are computed as:

$$\begin{aligned} t_i &= \arg\min(\omega_T(u_i)), \quad T_i = I_{PC}(t_i), \\ b_i &= \arg\min(\omega_B(u_i)), \quad B_i = I_{PC}(b_i), \\ l_j &= \arg\min(\omega_L(v_j)), \quad L_j = I_{PC}(l_j), \\ r_j &= \arg\min(\omega_R(v_j)), \quad R_j = I_{PC}(r_j). \end{aligned} \tag{17}$$

In Fig. 7(c), T_2 is the 3D point where $\omega_T(u_2)$ is the least. Note that the cost maps, and the resulting edge markers computed using Euclidean distance will not be as accurate than when using geodesic distance. However, this difference is not visually perceptible in the final projected image. Instead, the lag caused due to computing the geodesic distance is significantly more perceptible. Hence, we trade off accuracy of the edge markers for faster computation time.

Thus, the 3D points of markers along all four edges are computed and the appropriate (u, v) parameters for each of these

markers is assigned. The remaining 3D points i.e. those interior to the surface, are determined by the same method mentioned in Section 3.3.4. Control points are computed as described in Section 3.4.

3.6. Projection mapping

To achieve projection mapping, we need to compute the projector pixel $q(u, v)$ and warp the projection image according to Eq. (6). We sample the display surface densely by evaluating the B-spline patch at different (u, v) coordinates using the control points $\bar{\boldsymbol{C}}$ to get $\bar{\boldsymbol{Q}}(u, v)$, a 3D homogeneous point on the B-spline patch:

$$\bar{\boldsymbol{Q}}(u, v) = \boldsymbol{C}\ \boldsymbol{N}(u, v) \tag{18}$$

Then, this 3D homogeneous point is projected onto the 2D projector image plane using the projection matrix $\boldsymbol{M_p} \in \mathbb{R}^{3\times4}$ to get $\bar{\boldsymbol{q}}(u, v)$, the 2D homogeneous projector pixel coordinates:

$$\bar{\boldsymbol{q}}(u, v) = \begin{bmatrix} x(u, v)\ w(u, v) \\ y(u, v)\ w(u, v) \\ w(u, v) \end{bmatrix} = \boldsymbol{M_p}\ \bar{\boldsymbol{Q}}(u, v) \tag{19}$$

However, applying a projective transformation to a large number of 3D points as in Eq. (19) is a time-consuming operation, increasing the end-to-end latency of our system. This is where we exploit the properties of *rational* B-splines to improve the efficiency of our system. Instead of applying a projective transformation to the 3D homogeneous points $\bar{\boldsymbol{Q}}(u, v)$, we apply the same transformation to the control points $\bar{\boldsymbol{C}}$ to get $\bar{\boldsymbol{C}}_{\boldsymbol{p}}$, the 2D homogeneous control points. Substituting $\bar{\boldsymbol{Q}}(u, v)$ from Eq. (18) into (19):

$$\begin{aligned} \bar{\boldsymbol{q}}(u, v) &= \boldsymbol{M_p}\ \bar{\boldsymbol{C}}\ \boldsymbol{N}(u, v) \\ \bar{\boldsymbol{q}}(u, v) &= \bar{\boldsymbol{C}}_{\boldsymbol{p}}\ \boldsymbol{N}(u, v) \\ \bar{\boldsymbol{C}}_{\boldsymbol{p}} &= \begin{bmatrix} x_{00}w_{00} & x_{01}w_{01} & \dots & x_{rs}w_{rs} \\ y_{00}w_{00} & y_{01}w_{01} & \dots & y_{rs}w_{rs} \\ w_{00} & w_{01} & \dots & w_{rs} \end{bmatrix} \in \mathbb{R}^{3\times H} \end{aligned} \tag{20}$$

The 2D projector pixel is $q(u, v) = [x(u, v)\ y(u, v)]$. Then, we warp the projection image I_S according to Eq. (6) to get I_T, which gets projected.

3.6.1. Improving B-spline approximation

Since the display surface is deformable, it is possible that a B-spline patch with fixed parameters i.e. the degree and number of control points, may not be able to represent it accurately. Thus, we designed our system to automatically change the B-spline parameters if the current model does not accurately register the display surface geometry. We do this by measuring the registration error between the 3D display surface as measured by the depth camera and its B-spline representation using the control points computed in Section 3.4 at different (u, v) values. If the error is beyond a certain threshold, we increase the degree and/or number of control points. It is important to note that the maximum degree and number of control points of the B-spline patch is limited by the number of markers on each edge. Fig. 15 shows projection mapping onto a deformable surface with different B-spline models: planar, quadratic, piece-wise planar and cubic.

3.6.2. Free-form mapping on stretchable surfaces

Typically, the stretch applied to elastic surfaces is isometric i.e. the stretch is applied evenly. This increases the distances between adjacent markers along an edge evenly as well. However, when a non-isometric stretch is applied to the surface, stretching the surface unevenly, the distances between adjacent markers may not be the same. Since we do not constrain the B-spline patch to preserve distances, the projection expands to accommodate the part of the surface that gets stretched. This is known as *free-form* mapping. Fig. 11 (top row) demonstrates the noticeable distortion of the projection when a non-isometric stretch is applied to the middle two boundary points while the corners remain fixed.

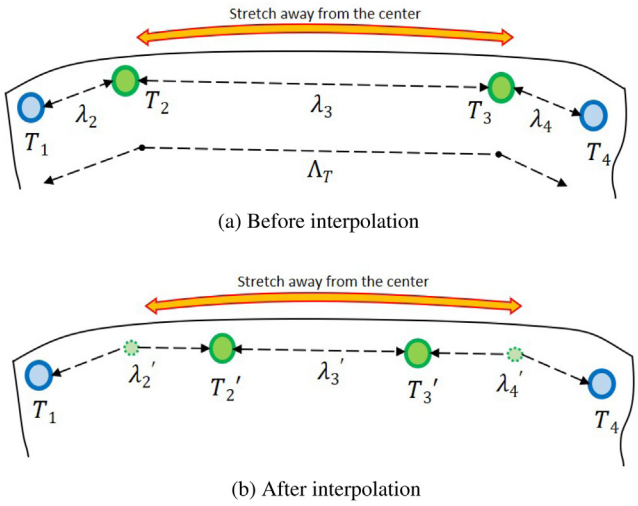


Fig. 8. Performing length-preserving mapping. **(a)** A stretch away from the center causes $\lambda_2 \neq \lambda_3 \neq \lambda_4$. **(b)** By interpolating T_2, T_3 along the line $\{T_2, T_3\}$ to T_2', T_3' respectively, $\lambda_2' \approx \lambda_3' \approx \lambda_4' \approx \frac{\Lambda_T}{3}$, resulting in length-preserving mapping.

While the marker-based implementation lends itself naturally to free-form mapping, to achieve the same for the marker-less case, the edge lengths $\Psi_T, \Psi_B, \Psi_L, \Psi_R$ in Eq. (16) are used to denote the surface edge lengths *at rest*. As a result, when a non-isometric stretch is applied, the distances between edge and corner points remain the same, but the distances between adjacent edge points do not.

3.6.3. Length preserving mapping on stretchable surfaces

However, we can constrain the B-spline patch to preserve distances, enabling a *length-preserving* mapping, even when non-isometric stretches are applied to the surface by moving the 3D point of each marker along each edge (except for the corners) such that the distances between all adjacent 3D points along that edge become equal. Let E_i denote the 3D point of the ith marker on an edge $\mathbb{E}$ with number of markers M_E. Let λ_i denote the distance between adjacent markers $\{E_{i-1}, E_i\}$ and Λ_E the total estimated length of the edge $\mathbb{E}$:

$$\begin{aligned} \lambda_i &= \|E_i - E_{i-1}\|, \quad 2 \leq i \leq M_E \\ \Lambda_E &= \sum_{j=2}^{M_E} \lambda_j \\ \lambda_E' &= \frac{\Lambda_E}{M_E - 1} \end{aligned} \tag{21}$$

Here, λ_E' denotes the distance between two adjacent markers if all E_i's were equally spaced along $\mathbb{E}$. So, if $\lambda_i < \lambda_E'$, E_i is closer to E_{i-1} than if they were equally spaced along $\mathbb{E}$.

To perform length-preserving mapping, the goal is to find, for each non-corner point E_i, a point E_i' along the edge $\mathbb{E}$ such that $\lambda_i' \approx \lambda_E'$, where $\lambda_i' = \|E_i' - E_{i-1}\|$. For e.g. if $\lambda_i < \lambda_E'$, the point E_i is closer to E_{i-1} and should be moved away from E_{i-1} toward E_{i+1}.

Depending on λ_i, each non-corner E_i is either linearly interpolated toward E_{i-1} or E_{i+1}. If $\lambda_i < \lambda_E'$, E_i' is determined by linear

interpolation along the line $\{E_i, E_{i+1}\}$, moving it away from E_{i-1} and increasing λ_i'. Similarly, if $\lambda_i > \lambda_E'$, then E_i' is determined by linear interpolation between $\{E_{i-1}, E_i\}$, reducing its length. This algorithm, that processes each segment between the markers *in sequence*, is summarized in Algorithm 1.

Algorithm 1: Length Preserving Mapping

procedure MakeLengthPreserving(EdgePoints E)
Compute edge lengths
$\Lambda = 0, M = \text{len}(E)$
for $i = 2$ *to* M **do**
  $\lambda_i = \|E_i - E_{i-1}\|$
  $\Lambda = \Lambda + \lambda_i$
end
$\lambda' = \frac{\Lambda}{M-1}$
Update edge points
for $i = 2$ *to* $(M-1)$ **do**
  if $\lambda_i < \lambda'$ **then**
    $t = \frac{\lambda' - \lambda_i}{\lambda_{(i+1)}}$
    $E_i = (1-t)E_i + tE_{i+1}$
  end
  else if $\lambda_i > \lambda'$ **then**
    $t = \frac{\lambda'}{\lambda_i}$
    $E_i = tE_{i-1} + (1-t)E_i$
  end
  $\lambda_i = \|E_i - E_{i-1}\|$
  $\lambda_{i+1} = \|E_{i+1} - E_i\|$
end
return E

Linearly interpolating E_i's between its two neighbors in this way ensures that the resulting E_i''s are approximately equally spaced along $\mathbb{E}$. Fig. 8 shows an example of T_2 and T_3 being moved to new positions T_2' and T_3' using the aforementioned technique. Fig. 11 (left-column) compares the grid squares with and without length-preserving mapping.

Note that the marker-less implementation lends itself naturally to length-preserving mapping when the edge lengths Ψ_T, Ψ_B, Ψ_L, Ψ_R in Eq. (16) are used to denote the *current* surface edge lengths instead of edge lengths *at rest*.

4. Implementation details

Our hardware setup consists of a standard desktop workstation with an Intel Core i7-9700K CPU @ 3.6 GHz with 64 GB of RAM and an NVIDIA GeForce RTX 2080 SUPER GPU. We used OpenCV's CUDA implementation to track the markers, compute the distance maps and cost maps. OpenGL was used for texture-mapping and rendering the final display. Estimating (u, v) assignments for 3D display surface points and computing the B-spline control points is performed on the CPU. Both the marker-based and marker-less systems can run at close to camera frame rates.

In our implementation, we used Optoma DLP Projector. We tested our system using two different cameras: the Microsoft Azure Kinect and the CamBoard Pico-Flexx. Both cameras provide registered IR and point cloud images, while the Azure Kinect additionally provides a registered RGB image as well. However, we only use the IR and point cloud images. The Azure Kinect depth camera resolution is 320 × 288 @ 30fps. The Pico-Flexx provides depth images at 224 × 171 resolution when run at 45fps. Notice the low spatial resolutions of both cameras.

To reduce the run-time of our system, we pre-compute the coefficient matrix $\boldsymbol{A}$ (Section 3.4) once for different combinations of B-spline parameters i.e. degree, knot vector and number of control points. This also allows us to dynamically modify the number of markers as well. Depending on the B-spline model parameters and the number of markers, the corresponding matrix $\boldsymbol{A}$ is used to compute the control points every iteration using linear least-squares. This means that the system must assign the 3D points to the correct (u, v) parameters used to compute $\boldsymbol{A}$. These assumptions allow fast computation of the control points.

From our experiments, for changing the B-spline parameters, the registration error threshold is set to 10%.

Computing distance maps and cost maps (see Section 3.5.1) for the marker-less display is computationally intensive. Thus, we made some important design decisions to speed up the computation without sacrificing accuracy of conforming to the deformable surface.

1. **Down-sizing distance and cost maps:** We compute the distance maps and cost maps using point cloud images with half resolution in both horizontal and vertical directions for the high-resolution Azure Kinect camera. We do not perform this step for the Pico-Flexx camera since it already has low resolution.
2. **16-bit signed integer representation:** We perform calculations on 16-bit signed integer arrays instead of floating-point arrays. If the depth camera provides point cloud data in meters, we convert it to millimeters and then store it as 16-bit signed integer. This gives a big performance boost (∼100ms/frame) without compromising the shape conformity of the final display.
3. **Temporal coherency for localized calculations:** We exploit temporal coherency between successive point cloud images to further reduce the time to compute edge markers. Instead of computing the distance maps and cost maps along all edges, we compute them in local windows (31 × 31) centered on the marker locations in the previous depth frame. Initially, the window centers are linearly interpolated between adjacent corners. When a new depth frame is available, the distance maps and costs maps are computed for 3D points on the edges inside these windows only. These cost map "windows" are used to determine the new edge markers. The window centers are also updated to the location of the new markers in the depth camera.
4. **Noise and jitter removal:** Searching for the 3D point with the minimum cost ω (see Eq. (17)) is a time-consuming step even for small windows. Further, noise in the point cloud image introduces errors in the estimation of the edge markers, resulting in a jittery projection. Although applying a smoothing filter to the point cloud image can reduce noise, it adds extra operations and increases the latency. To avoid the additional time taken by pixel-level search and noise smoothing without adding more operations, we average 3D points inside the windows, weighted by the inverse of their costs ω i.e. a 3D point with a lower cost has a higher weight. This results in faster computation of the edge markers while smoothing out errors in the point cloud image and removing jitter.

5. Results

Fig. 9(b) shows the results of our projection mapping system on a marked and marker-less surface undergoing different non-elastic deformations. Note how the projection conforms to the shape of the surface for all different deformations, especially when compared to Fig. 9(a), which shows the same images without any mapping applied. *Please see the accompanying video for demonstrations on dynamic movement of the surface.*

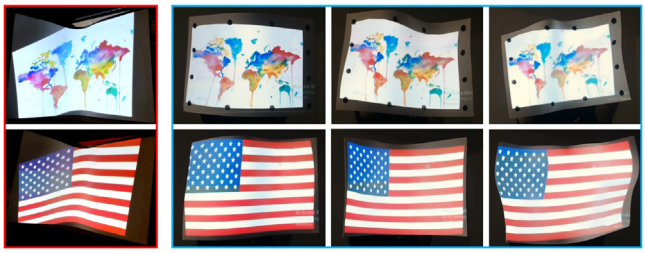


Fig. 9. (a) Projection onto a non-planar surface without any mapping applied. **(b)** Projection mapping onto marker-based (top row) and marker-less (bottom row) surface.

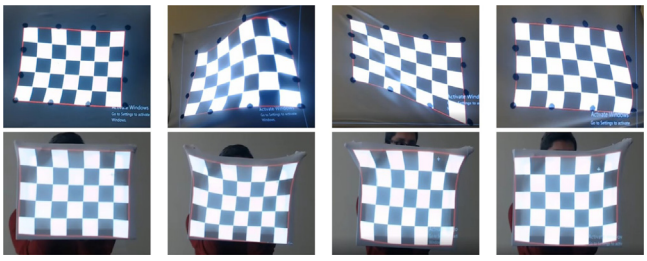


Fig. 10. A checkerboard pattern projected onto (top-row) marker-based and (bottom-row) marker-less display surface. Note that when the surface is stretched, the tiles in the stretched region also expand.

Fig. 10 demonstrates that our method can adapt to stretchable materials for marker-based and marker-less surfaces. Similar effects are also demonstrated for stretches from both directions when projecting a checkerboard pattern (see video).

Fig. 11 shows the difference between free-form mapping and length-preserving mapping. When the two interior points of a boundary edge are stretched away from each other while keeping the corners fixed, in the former the stretch is visually perceptible from the movement of the content. But, with length-preserving correction, the stretch becomes imperceptible giving an impression that the material is not elastic. Note that the length preserving mapping provides perceptually pleasing results for common contents like faces, scenery and buildings (Fig. 11).

Fig. 12 shows some applications of our system. The top row shows our system being used for T-shirt design. A user can project designs of their choice onto the T-shirt and interact with the fabric to see how it will look. The bottom row shows how our system can be used to view volumetric data (e.g. MRI/CT scans of the human body). By moving the display towards and away from the camera, the user can view slices through the volume. Additionally, non-planar, curved cross-sections of the surface can be used to compare different slices at the same time. *The video shows dynamic results.* Fig. 13 shows our system being used for scientific visualization on a markerless, rigid, dynamic surface.

5.1. Performance and evaluation

We evaluated our system by varying different parameters, such as the number of markers along the edges and the number of

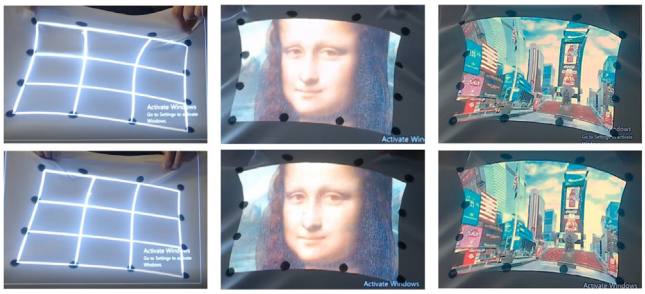


Fig. 11. Free-form mapping (top-row) vs. length-preserving mapping (bottom-row). Note how the vertical grid lines follow the dots in free-form mapping but evenly space out with length preserving mapping. The same stretch is applied to the middle and right columns. Note the distortion caused by the stretch makes the images with free-form mapping look unnatural (e.g. Mona Lisa's forehead). Using length-preserving mapping, the images look less distorted while still conforming to the surface shape.

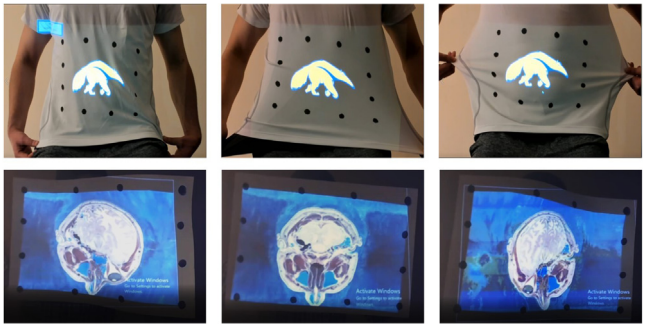


Fig. 12. Two applications of our system. The top row shows our system being used for designing a t-shirt while the user interacts with the fabric. The bottom row shows the user deforming the surface to view volumetric data at curved cross-sectional slices and comparing them.

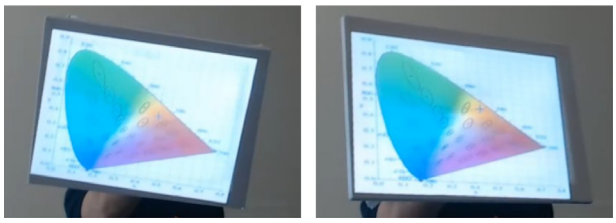


Fig. 13. Marker-less display on a rigid, dynamic surface.

control points, and measuring **(i)** the surface registration between the B-spline patch and the actual surface geometry, and **(ii)** the end-to-end latency. Furthermore, we also tested the accuracy of the internal 3D points for various shapes since that directly affects the B-spline patch that gets computed.

To evaluate surface registration, we compute the distances between the B-spline 3D points and the point cloud image. The misregistered surface area is the ratio of the number of points whose distance is greater than 10 mm to the total number of points. We measured the surface misregistration for three different shapes: a curve, an S-shape and a wave as shown in Fig. 14. Fig. 14(d) shows the effect of increasing the control points on the surface misregistration for all three shapes, while keeping the number of markers constant at $(M_u, M_v) = (8, 8)$. For the curve

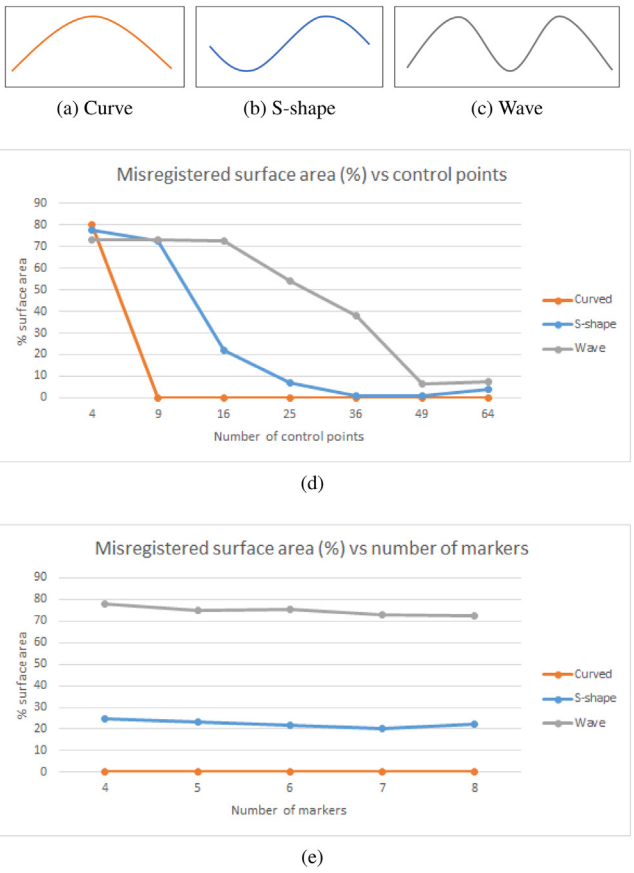


Fig. 14. Effect of various parameters on the surface registration. **(a)–(c)**: the surface profiles used to study the effect on the registration error by **(d)** the number of control points, and **(e)** the number of markers.

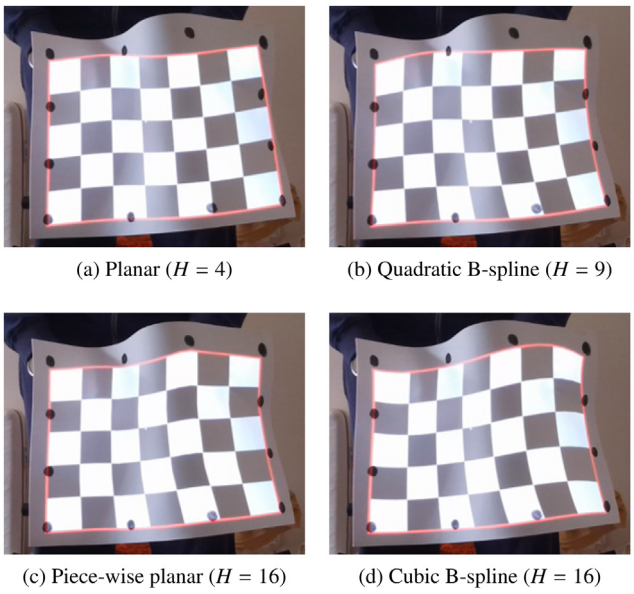


Fig. 15. Projection mapping on a surface deformed as an S-shape while changing the B-spline model. Notice that although **(c)** and **(d)** conform to the surface well and have the same number of control points ($H = 16$), the projection mapping is not identical. This is visible at the top edge, where **(c)** has visible discontinuities compared to **(d)**.

shape, four control points are not sufficient to represent it, therefore we see a high (∼80%) misregistration. However, increasing the control points to 9 or more results in excellent registration since the B-spline patch can represent the shape accurately. We see a similar significant decrease in misregistration for the S-shape: from 80% with 9 control points to 20% with 16 control points. As for the wave, since it is more complex, it requires an even larger number of control points (49) to represent the shape accurately.

Fig. 14(e) shows the effect of increasing the number of markers on the surface misregistration for the three shapes. In this experiment, we modeled the surface using a degree-3 B-spline patch with 16 control points while changing the number of markers. Since 16 control points are sufficient to represent the curved shape, its surface misregistration is close to zero. However, for the S-shape and the wave, we only see a slight decrease in the misregistration as we increase the number of markers. This is because it is the number of control points, rather than the number of samples, that limits the ability of the B-spline patch to represent those shapes. Therefore, to improve surface registration, the number of control points must be increased, an observation that is confirmed by Fig. 14(d).

The surface misregistration can be visually observed as well. Fig. 15 shows projection mapping on an S-shape surface while changing the number of control points. Notice that both planar and quadratic models (Fig. 15(a)–(b)) are not rich enough to adequately represent the surface shape and therefore, do not conform to the boundaries well. This is also confirmed by Fig. 14(d), where they have a high misregistration for the S-shape. However, the piece-wise planar and cubic models (Fig. 15(c)–(d)) are able to represent the shape well and hence, projection mapping conforms to the boundaries. Also notice that while both the piece-wise planar and cubic models have the same number of control points ($H = 16$), they represent two different shapes since they are of two different degrees ($(n, m) = (1, 1)$ and $(3, 3)$ respectively). This is why the projections do not appear identical. The difference is especially visible at the top edge, where the piece-wise planar model has visible discontinuities.

Fig. 16 shows the end-to-end latency, i.e. the time from when a new camera frame is available till the final rendering and display, for both the marker-based and marker-less systems, for different number of edge markers. Note that increasing the number of markers results in an insignificant increase in the latency for the marker-based system, which consistently runs close to 5 ms/frame. On the other hand, the number of markers has a significant effect on the end-to-end latency for the markerless system. This is because increasing the number of markers increases the number of cost maps that need to be computed, a computationally intensive step. Despite that, the markerless system performs faster than the camera frame-rate (33.33 ms) for up to seven markers on each edge. As such, the end-to-end latency becomes limited by the camera frame rate.

Figs. 14 and 16 also reveal an interesting trade-off of our system. Fig. 14 suggests that one should increase the number of control points to achieve better surface registration, but that can only be done by increasing the number of markers. However, the number of markers has a direct impact on the end-to-end latency, as shown in Fig. 16, especially for the marker-less system, which increases at a rate of 4.5ms/marker. Therefore, the number of markers required depends on the demands of the application in terms of speed and accuracy of registration. In general, $5 \leq (M_u, M_v) \leq 7$ and $25 \leq H \leq 49$ leads to excellent performance (in both speed and accuracy) for the marker-based and marker-less systems for a large variety of applications.

Our marker-based system requires the user to place markers on the surface. Therefore, we investigated how accurate the user

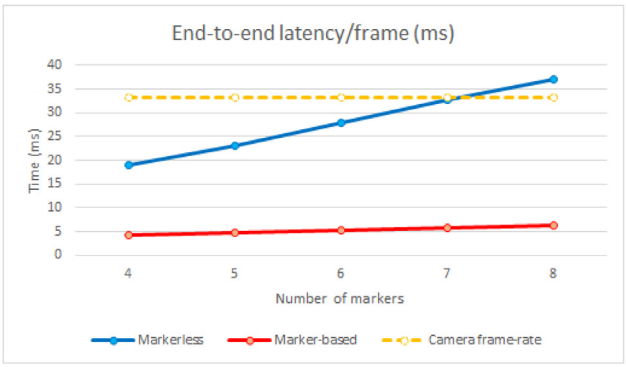
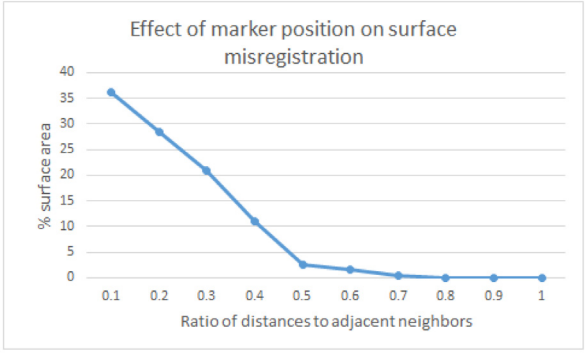
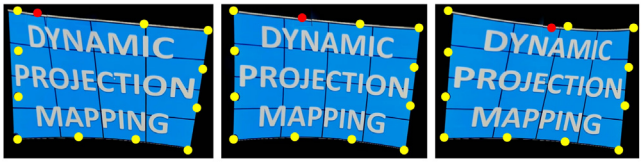


Fig. 16. Effect of the number of markers on the end-to-end latency for marker-based and marker-less systems.



(a) Surface misregistration for various marker positions

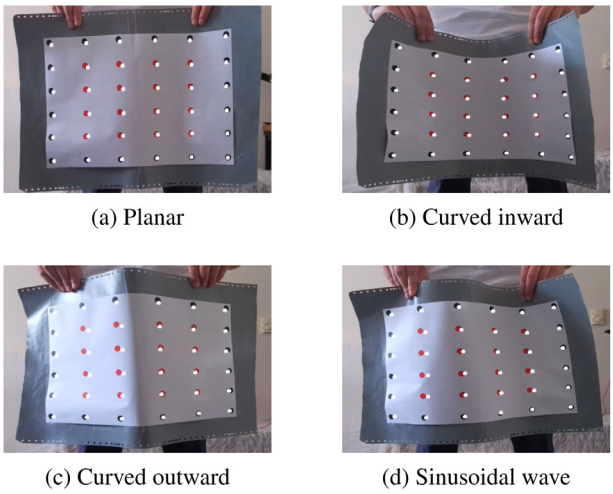


(b) Effect on projection mapping on marker placement

Fig. 17. (a) Effect of marker placement on the surface area misregistration. When the distance ratio (*x-axis*) is 1, the marker is at its ideal position while smaller values mean increasing deviation from the ideal. **(b)** shows the projection when moving the marker T_2 (in red) from its ideal position *(middle)* towards T_1 *(left)* or towards T_3 *(right)*.

must be when placing these markers. For this experiment, the surface had four markers on each edge, was shaped like a curve (Fig. 14(a)) and modeled by a quadratic B-spline (with 9 control points). We moved the marker T_2 towards T_1 and measured the distances between T_2 and its two adjacent neighbors. For each such position, we calculated the surface misregistration as a function of the ratio of the larger distance to the smaller distance. A distance ratio of 1 means T_2 is placed directly between its two neighbors (its ideal position), whereas a value of 0.5 means T_2 is twice as far away from one neighbor compared to the other.

The effect of marker placement on the surface misregistration is shown in Fig. 17(a). Notice that as the distance ratio decreases (i.e. larger deviation from ideal position), the surface misregistration increases. This can be visibly confirmed in Fig. 17(b), which shows the projection becoming quite distorted when T_2 is moved too close to either of its neighbors (left and right) compared to when it is at its ideal position (middle). Fig. 17(a) also shows that



(a) Planar (b) Curved inward

(c) Curved outward (d) Sinusoidal wave

Fig. 18. Error when computing internal 3D points. The red and black markers (ground truth points) are compared with the white overlaid markers (points estimated by our system). The error for each shape is **(a)** 2.59 mm. **(b)** 2.27 mm, **(c)** 4.99 mm, **(d)** 6.59 mm.

even if a marker is placed twice as close to one of its neighbors compared to the other (a ratio of 0.5), the surface misregistration still remains relatively low ($< 5\%$). This means that, for a standard A4-size paper ($21.\times 29.7$ cm), a marker needs to be within $\sim$3 cm of its ideal position to yield good registration. This gives the user significant flexibility when placing markers, even if they are not precisely equidistant from each other.

To measure the accuracy of the internal 3D points, we compute the average error between the points estimated by our system and their ground truth points. The ground truth internal points are obtained by a display surface with the full marker grid printed. We measured this error for different shapes of the surface. As shown in Fig. 18, the largest error is around 6.59 mm, which is not perceivable by viewers.

6. Conclusion and future work

In summary, we present a general B-spline patch-based framework to achieve projection mapping on dynamic deformable stretchable materials using consumer-grade ToF depth camera and the registered IR camera accompanying it. We demonstrate real time performance which has potential to leverage higher speed cameras to reduce any remaining perceptible latency in the system.

In the future, we would like to improve the system to handle occlusion of the markers and the surface for both marker-based and marker-less implementations. One way to implement this is to compute a rigid transform using ICP between successive frames using the visible 3D points and applying that transform to the entire surface before re-computing the shape. The challenge there will be to ensure that the end-to-end latency remains as low as possible despite the additional computation. To handle self-occlusion of the surface, the proposed system can be extended to use multiple projectors, so that light can reach the surface from multiple directions.

The proposed work cannot identify areas undergoing local non-isometric stretches for e.g., stretching only a small part of the surface but not the other. While previous works use markers in the interior of the surface to handle non-isometric stretches, achieving the same without any markers is a challenging task that can be addressed in the future.

Additionally, the proposed framework could be extended to use Non-Uniform Rational B-Splines (NURBs), which would allow representing sharp folds and more complex shapes and deformations. Implementing NURBs will require computing the knot vector and control point weights using non-linear optimizations, unlike the proposed work which only uses linear optimizations.

CRediT authorship contribution statement

Muhammad Twaha Ibrahim: Conceptualization, Methodology, Software, Validation, Investigation, Visualization, Writing – original draft, Writing – review & editing. **Aditi Majumder:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision. **M. Gopi:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2022.01.004.

References

[1] Sueishi T, Oku H, Ishikawa M. Robust high-speed tracking against illumination changes for dynamic projection mapping. In: IEEE VR. 2015, p. 97–104.

[2] Mikawa Y, Sueishi T, Watanabe Y, Ishikawa M. VarioLight: hybrid dynamic projection mapping using high-speed projector and optical axis controller. In: Proc. SIGGRAPH asia 2018 emerging technologies. 2018.

[3] Wang L, Xu H, Hu Y, Tabata S, Ishikawa M. Dynamic depth-of-field projection for 3D projection mapping. In: Extended abstracts of the 2019 CHI conference on human factors in computing systems. 2019.

[4] Steimle J, Jordt A, Maes P. Flexpad: highly flexible bending interactions for projected handheld displays. In: Proc. of the SIGCHI conference on human factors in computing systems. 2013, p. 237–46.

[5] Narita G, Watanabe Y, Ishikawa M. Dynamic projection mapping onto a deformable object with occlusion based on high-speed tracking of dot marker array. In: Proc. 21st ACM symposium on virtual reality software and technology. 2015, p. 149–52.

[6] Narita G, Watanabe Y, Ishikawa M. Dynamic projection mapping onto deforming non-rigid surface using deformable dot cluster marker. IEEE Trans Vis Comput Graphics 2017;23:1235–48.

[7] Siegl C, Colaianni M, Thies L, Thies J, Zollhöfer M, Izadi S, et al. Real-time pixel luminance optimization for dynamic multi-projection mapping. ACM Trans Graph 2015.

[8] Lange V, Siegl C, Colaianni M, Stamminger M, Bauer F. Robust blending and occlusion compensation in dynamic multi-projection mapping. In: Proc. Of european association for computer graphics: short papers. 2017, p. 1–4.

[9] Goldstein EB, Brockmole J. Sensation and perception. 2017.

[10] Tehrani MA, Gopi M, Majumder A. Automated geometric registration for multi-projector displays on arbitrary 3D shapes using uncalibrated devices. IEEE Trans Vis Comput Graphics 2019.

[11] Raskar R, Brown MS, Yang R, Chen WC, Welch G, Towles H, et al. Multi projector displays using camera-based registration. In: Proc IEEE Vis. 1999.

[12] Sajadi B, Majumder A. Automatic registration of multiple projectors on swept surfaces. In: ACM virtual reality and software technology. 2010.

[13] Sajadi B, Majumder A. Auto-calibrating projectors for tiled displays on piece-wise smooth vertically extruded surfaces. IEEE Trans Vis Comput Graphics 2011.

[14] Sajadi B, Majumder A. Markerless view independent geometric registration of multiple distorted projectors on vertically extruded surfaces using a single uncalibrated camera. IEEE Trans Vis Comput Graphics 2009.

[15] Raskar R. Immersive planar displays using roughly aligned projectors. In: Proc. Of IEEE VR. 2000.

[16] Brown MS, Seales WB. A practical and flexible tiled display system. In: 10th pacific conference on computer graphics and applications. 2002, p. 194–203.

[17] Johnson T, Fuchs H. Real-time projector tracking on complex geometry using ordinary imagery. In: IEEE CVPR workshop on projector-camera systems. 2007.

[18] Raskar R, Welch G, Low KL, Bandyopadhyay D. Shader lamps: Animating real objects with image-based illumination. In: Eurographics workshop on rendering techniques. 2001, p. 89–102.

[19] Raskar R, Baar Jv, Beardsley P, Willwacher T, Rao S, Forlines C. ILamps: geometrically aware and self-configuring projectors. In: ACM SIGGRAPH 2006 courses. 2006.

[20] Johnson T, Gyarfas F, Skarbez R, Towles H, Fuchs H. A personal surround environment: Projective display with correction for display surface geometry and extreme lens distortion. In: Proceedings Of IEEE virtual reality conference. 2007, p. 147–54.

[21] Fujimoto Y, Smith RT, Taketomi T, Yamamoto G, Miyazaki J, Kato H, et al. Geometrically-correct projection-based texture mapping onto a deformable object. IEEE Trans Vis Comput Graphics 2014;20:540–9.

[22] Zhou Y, Xiao S, Tang Z, Chen X. Pmomo: Projection mapping on movable 3D object. In: Proceedings Of The 2016 CHI conference on human factors in computing systems. 2016, p. 781–90.

[23] Siegl C, Lange V, Stamminger M, Bauer F, Thies J. FaceForge: Markerless non-rigid face multi-projection mapping. IEEE Trans Vis Comput Graphics 2017;23:2440–6.

[24] Miyashita L, Watanabe Y, Ishikawa M. MIDAS projection: markerless and modelless dynamic projection mapping for material representation. ACM Trans Graph 2018.

[25] Asayama H, Iwai D, Sato K. Fabricating diminishable visual markers for geometric registration in projection mapping. IEEE Trans Vis Comput Graphics 2018;24:1091–102.

[26] Kurth P, Lange V, Siegl C, Stamminger M, Bauer F. Auto-calibration for dynamic multi-projection mapping on arbitrary surfaces. IEEE Trans Vis Comput Graphics 2018;24:2886–94.

[27] Kagami S, Hashimoto K. Animated stickies: Fast video projection mapping onto a markerless plane through a direct closed-loop alignment. IEEE Trans Vis Comput Graphics 2019;25:3094–104.

[28] Resch C, Keitler P, Klinker G. Sticky projections — A new approach to interactive shader lamp tracking. In: IEEE international symposium on mixed and augmented reality. 2014, p. 151–6.

[29] Hashimoto N, Koizumi R, Kobayashi D. Dynamic projection mapping with a single IR camera. Int J Comput Games Technol 2017.

[30] Siegl C, Colaianni M, Stamminger M, Bauer F. Stray-light compensation in dynamic projection mapping. In: SIGGRAPH ASIA 2016 technical briefs. 2016.

[31] Lange V, Siegl C, Colaianni M, Kurth P, Stamminger M, Bauer F. Interactive painting and lighting in dynamic multi-projection mapping. Augment Real Virtual Real Comput Graphics 2016;9769.

[32] Saakes D, Yeo HS, Noh ST, Han G, Woo W. Mirror mirror: An on-body T-shirt design system. In: Proceedings of the 2016 CHI conference on human factors in computing systems. 2016.

[33] Bermano AH, Billeter M, Iwai D, Grundhöfer A. Makeup lamps: Live augmentation of human faces via projection. Comput Graph Forum 2017;311–23.

[34] Punpongsanon P, Iwai D, Sato K. FleXeen: Visually manipulating perceived fabric bending stiffness in spatial augmented reality. IEEE Trans Vis Comput Graphics 2020;1433–9.

[35] Punpongsanon P, Iwai D, Sato K. Deforme: projection-based visualization of deformable surfaces using invisible textures. In: SIGGRAPH asia 2013 emerging technologies. 2013, p. 1–3.

[36] Punpongsanon P, Iwai D, Sato K. Projection-based visualization of tangential deformation of nonrigid surface by deformation estimation using infrared texture. Virtual Real 2015;19.

[37] Kawabe T, Fukiage T, Sawayama M, Nishida S. Deformation lamps: A projection technique to make static objects perceptually dynamic. ACM Trans Appl Percept 2016.

[38] Bouman KL, Xiao B, Battaglia P, Freeman WT. Estimating the material properties of fabric from video. In: Proc. IEEE Int. Conf. Comput. Vis.. 2013, p. 1984–91.

[39] Bi W, Xiao B. Perceptual constancy of mechanical properties of cloth under variation of external forces. In: Proceedings of the ACM symposium on applied perception. 2016, p. 19–23.

[40] Pilet J, Lepetit V, Fua P. Real-time nonrigid surface detection. In: IEEE computer society conference on computer vision and pattern recognition. 2005, p. 822–8.

[41] Leizea I, Álvarez H, Aguinaga I, Borro D. Real-time deformation, registration and tracking of solids based on physical simulation. In: IEEE international symposium on mixed and augmented reality. 2014, p. 165–70.

[42] Ngo DT, Östlund J, Fua P. Template-based monocular 3D shape recovery using Laplacian meshes. IEEE Trans Pattern Anal Mach Intell 2016;38:172–87.

[43] Jacobson A, Baran I, Popović J, Sorkine O. Bounded biharmonic weights for real-time deformation. ACM Trans Graphics 2011;30.

[44] Schaefer S, McPhail T, Warren J. Image deformation using moving least squares. ACM Trans Graphics 2006;25.
[45] Ibrahim MT, Meenakshisundaram G, Majumder A. Dynamic projection mapping of deformable stretchable materials. In: Proc. VRST '20: 26th ACM symposium on virtual reality software and technology. 2020, p. 1–5.
[46] Moreno D, Taubin G. Simple, accurate, and robust projector-camera calibration. In: 2012 second international conference on 3D imaging, modeling, processing, visualization and transmission. 2012, p. 464–71.
[47] Moreno D, Taubin G. Projector-camera calibration / 3D scanning software. 2012, http://mesh.brown.edu/calibration/software.html.
[48] Lucas BD, Kanade T. An iterative image registration technique with an application to stereo vision. In: Proceedings of imaging understanding workshop. 1981, p. 121–30.
[49] Tomasi C, Kanade T. Detection and tracking of point features. Technical report CMU-CS-91-132, In Carnegie Mellon University; 1991.
[50] Shi J, Tomasi C. Good features to track. In: IEEE conference on computer vision and pattern recognition. 1994, p. 593–600.
[51] Harris C, Stephens M. A combined corner and edge detector. In: Alvey vision conference. 1988.