Live Texturing of Augmented Reality Characters from Colored Drawings

Stéphane Magnenat, Dat Tien Ngo*, Fabio Zünd, Mattia Ryffel, Gioacchino Noris, Gerhard Rothlin, Alessia Marra, Maurizio Nitti, Pascal Fua, *Fellow, IEEE*, Markus Gross, Robert W. Sumner*

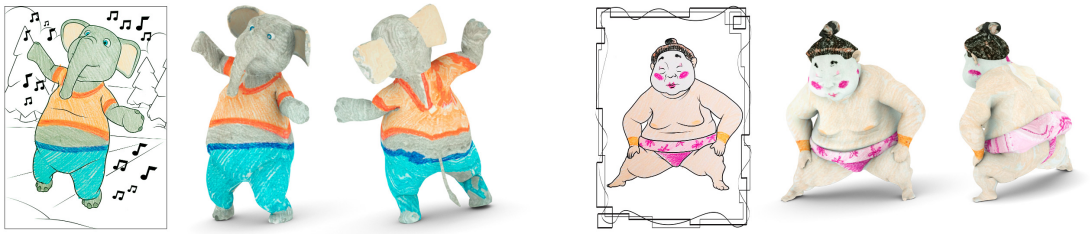


Fig. 1: Two examples of our augmented reality coloring book algorithm showing the colored input drawings and the captured texture applied to both visible and occluded regions of the corresponding 3-D characters.

Abstract— Coloring books capture the imagination of children and provide them with one of their earliest opportunities for creative expression. However, given the proliferation and popularity of digital devices, real-world activities like coloring can seem unexciting, and children become less engaged in them. Augmented reality holds unique potential to impact this situation by providing a bridge between real-world activities and digital enhancements. In this paper, we present an augmented reality coloring book App in which children color characters in a printed coloring book and inspect their work using a mobile device. The drawing is detected and tracked, and the video stream is augmented with an animated 3-D version of the character that is textured according to the child's coloring. This is possible thanks to several novel technical contributions. We present a texturing process that applies the captured texture from a 2-D colored drawing to both the visible and occluded regions of a 3-D character in real time. We develop a deformable surface tracking method designed for colored drawings that uses a new outlier rejection algorithm for real-time tracking and surface deformation recovery. We present a content creation pipeline to efficiently create the 2-D and 3-D content. And, finally, we validate our work with two user studies that examine the quality of our texturing algorithm and the overall App experience.

Index Terms—Augmented reality, deformable surface tracking, inpainting, interactive books, drawing coloring

1 Introduction

Coloring books capture the imagination of children and provide them with one of their earliest opportunities for creative expression. However, given the proliferation and popularity of television and digital devices, traditional activities like coloring can seem unexciting in comparison. As a result, children spend an increasing amount of time passively consuming content or absorbed in digital devices and become less engaged with real-world activities that challenge their creativity. Augmented reality (AR) holds unique potential to impact this situation by providing a bridge between real-world activities and digital enhancements. AR allows us to use the full power and popularity of digital devices in order to direct renewed emphasis on traditional activities like coloring.

In this paper, we present an AR coloring book App that provides a bridge between animated cartoon characters and their colored drawings. Children color characters in a printed coloring book and inspect their work using a consumer-grade mobile device, such as a tablet or smart phone. The drawing is detected and tracked, and the video stream is augmented with an animated 3-D version of the character that is textured according to the child's coloring. Fig. 1 shows two 3-D characters textured with our method based on the input 2-D colored drawings.

Accomplishing this goal required addressing several challenges. First, the 2-D colored drawing provides texture information only about the visible portions of the character. Texture for the occluded regions, such as the back side of the character, must be generated. Naïve approaches, such as mirroring, produce poor results since features like the character's face may be mirrored to the back of their head. In addition, without special treatment, texture mapping will exhibit visible seams where different portions of the parameterization meet. Second, our method targets live update, so that colored changes are immediately visible on the 3-D model as the child colors. Thus, the texturing challenges must be solved with a very limited computation budget. Third, the pages in an actual printed coloring book are not flat but exhibit curvature due to the binding of the book. As a result, tracking algorithms and texture capture must be robust to page deformation in order to properly track the drawings and lift texture from the appropriate 2-D regions. Finally, the practical consideration of authoring costs requires an efficient content creation pipeline for AR coloring books.

Our coloring book App addresses each of these technical challenges. We present a novel texturing process that applies the captured texture from a 2-D colored drawing to both the visible and occluded regions of a 3-D character in real time while avoiding mirroring artifacts and artifacts due to parameterization seams. We develop a deformable surface tracking method that uses a new outlier rejection algorithm for real-time tracking and surface deformation recovery. We present a content creation pipeline to efficiently create the 2-D and 3-D content used in App. Finally, we validate our work with 2 user studies that examine the quality of our texturing algorithm and the overall experience.

- *S. Magnenat, M. Ryffel, G. Noris, G. Rohlin, A. Marra, M. Nitti, M. Gross, and R. Sumner are with Disney Research Zürich, Switzerland. E-mail: sumner@disneyresearch.com (Coloring book App and texture synthesis).*
- *T.D. Ngo and P. Fua are with Computer Vision Lab, EPFL, Switzerland. E-mail: dat.ngo@epfl.ch (deformable surface tracking).*
- *F. Zünd, M. Gross and R. Sumner are with Computer Graphics Laboratory, ETH Zurich, Switzerland. (user studies and evaluation).*

Manuscript received 18 Sept. 2014; accepted 10 Jan. 2015. Date of Publication 20 Jan. 2015; date of current version 23 Mar. 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.

2 Related work

2.1 AR creative tools for children

A prominent early example of the combination of virtual content with a book is the MagicBook [4]. It uses large markers and virtual-reality glasses to view 3-D content based on which page of the book is open. The glasses are opaque and hence there is no blending of virtual content with any real content in the book itself. Scherrer and colleagues [34] have shown how well-integrated AR content can improve the user experience of books, but the book itself is still a static element. Recently, Clark and Dünser [11] explore the use of colored drawings to texture AR elements. Their work uses a pop-up book metaphor, providing animated augmented pop-ups that are textured using the drawing's colors. Their paper also shows two 3-D models colored according to the drawings, and a commercial product derived from this work called colAR[1] shows many more. However, the paper does not include details of the texturing process. In addition to colAR, three other coloring book products, Crayola Color Alive, Paint My Cat and Disney's Color and Play[2], use AR in the context of coloring. Although no details of the texturing process are provided, we suspect that these products use manually-edited correspondences between triangles in UV space and drawing space. The use case for these products targets line art printed at home, which avoids the issue of curved pages due to a book's binding. For example, the colAR FAQ urges users to ensure that the printed pages are "lying flat." On the contrary, we support augmentation on deformed surfaces and propose an efficient content creation pipeline that provides a mostly automatic method to generate appropriate UV mappings. In addition, we describe quantitative user studies which contribute to the anecdotal observations offered by Clark and Dünser [11].

2.2 Texture generation for 3-D meshes

In the context of our App, a 2-D colored drawing provides information about the texture of the portions of the 3-D model that are visible in the drawing. Determining the texture for the occluded regions involves filling in the remaining portions of the model's texture map, which is an *inpainting* problem. Inpainting addresses the task of filling a portion of an image from other parts of the image. Methods can be split into two categories: diffusion-based and patch-based [18]. The former consist of propagating local information around the boundary between the known and the unknown parts of the image, under smoothness constraints. These methods work well for filling small unknown regions surrounded by known regions. However, they require many iterations and may exhibit artifacts when filling larger regions [18]. Methods in the second category copy patches from the known regions to the unknown ones until the unknown regions are filled completely. The order of filling is critical for the visual result [13], but even the best filling order can lead to discrepancies, especially when an unknown region lies in the middle of a known region [22]. Indeed, in that case there will likely be non-matching patches in the center of the unknown region, creating visual artifacts. Recently, techniques such as graph cuts have been applied to alleviate this problem. The resulting algorithm produces good visual results but takes about one minute for a typical image [22]. Recent work on video inpainting [19] achieves real-time performance, but uses a desktop processor and fills only a small area of the image. A modern approach exploits the fact that local structures are typically repeated in the image and therefore structural priors can be captured and used for reconstruction [21]. These global methods work well for filling many small holes, but are not designed to fill larger areas.

Although these methods work well for image processing applications, they are not designed for real-time performance on mobile devices. In the context of texturing meshes from colored drawings, the critical element is that the generated texture is continuous across silhouette boundaries and texture seams. Therefore, we express texture generation as a static problem whose aim is to create a mapping, for every point of the texture, to a point in the drawing. Our proposed algorithm is inspired by both diffusion-based and patch-based methods, as it both extends the known parts of the image to the unknown ones, and copies pixels know regions to unknown ones.

2.3 Deformable surface tracking

Reconstructing the 3-D shape of a non-rigid surface from monocular images is an under-constrained problem, even when a reference image of the surface in its known rest shape is available. This is the problem we address in the context of live texturing from colored drawings, as opposed to recovering the shape from sequences as in many recent monocular Non-Rigid Structure from Motion methods such as [16, 17].

Given correspondences between a reference image and a live input image, one can compute a 2-D warp between the images and infer a 3-D shape from it, assuming the surface deforms isometrically [2, 10]. The reconstruction has the potential to run in real time because it is done in closed form and point-wise or by linear least-squares. However, the accuracy of the recovered shape is affected by the quality of the 2-D warp, which does not take into account the 3-D deformation properties of the surface. In our coloring book application, this problem is more severe because a large part of the surface is homogeneously blank, making 2-D image warping imprecise.

An alternative is to go directly from correspondences to a 3-D shape by solving an ill-conditioned linear-system [31], which requires the introduction of additional constraints to make it well-posed. The most popular added constraints involve preserving Euclidean or Geodesic distances as the surface deforms, which is enforced either by solving a convex optimization problem [8, 27, 32] or by solving sets of quadratic equations [33, 23] in closed form. The latter method is typically implemented by linearization, which results in very large systems and is no faster than minimizing a convex objective function, as is done in [32].

The complexity of the problem can be reduced using a dimensionality reduction technique such as principal component analysis (PCA) to create morphable models [12, 5], modal analysis [23], free form deformations (FFD) [8], or 3-D warps [14]. One drawback of PCA and modal analysis is that they require either training data or specific surface properties, neither of which may be forthcoming. Another is that the modal deformations are expressed with respect to a reference shape, which must be correctly positioned. This requires the introduction of additional rotation and translation parameters into the computation, preventing its use in live AR applications.

The FFD approach [8] avoids these difficulties and relies on parameterizing the surface in terms of control points. However, its complex formulation is quite slow to optimize as reported in [7]. The work of Ostlund et al. [25] takes inspiration from the Laplacian formalism presented in [35] and the rotation-invariant formulation of [36] to derive a rotation-invariant regularization term and a linear subspace parameterization of mesh vertices with respect to some control vertices. This technique leads to the first real-time 3-D deformable surface tracking system as reported in [24], which can run at 8–10 frames per second (FPS) on a MacBook Pro 2014 laptop. However, the high memory consumption and still heavy computation prohibit it from running in real-time on mobile devices.

To the best of our knowledge, there have been no reports so far describing a real-time deformable object tracking system on mobile devices. The presented contribution is an improvement upon previous work [24]. We propose a new outlier rejection mechanism, reformulate the reconstruction energy function to gain speed while not sacrificing accuracy, as well as rely on frame-to-frame tracking to gain frame rate and only apply the feature detection and matching periodically to retrieve back lost tracked points and accumulate good correspondences. These together allow real-time tracking on a tablet.

3 Problem formulation

Our method for live texturing an AR character from a colored drawing updates the texture of the 3-D character at every frame by copying pixels from the drawing. To do so, we create a UV lookup map, that, for every pixel of the texture, indicates a pixel coordinate in the drawing. As the drawing lies on a deformable surface, the later procedure operates on a rectified image of the drawing. We split this process into two separate

[1] http://colarapp.com/

[2] http://www.crayola.com/qr/package/2015/coloralive; http://www.paintmyzoo.com; http://disneystories.com/app/disney-color-and-play

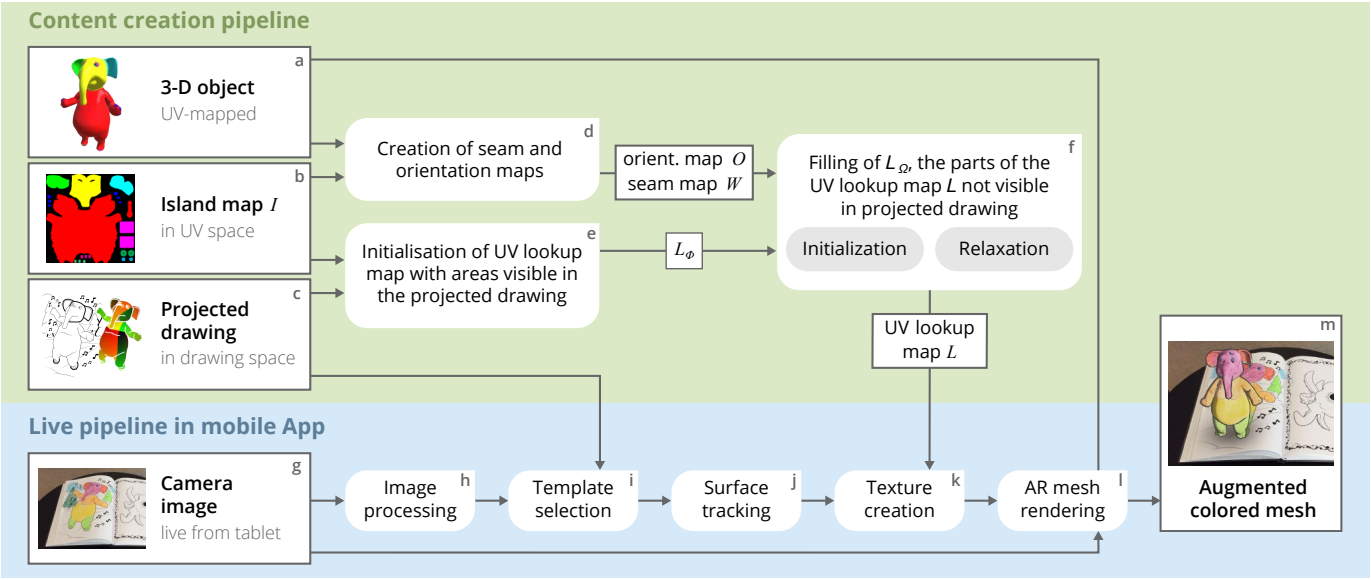


Fig. 2: The static content creation and the in-App live surface tracking, texturing, and rendering pipelines.

pipelines, as shown in Fig. 2. A static *content creation* pipeline creates a lookup map from the work of artists. This pipeline needs to be run only once. A *live* pipeline tracks the drawing and overlays the augmented character on top of it, with a texture created from the drawing in real time using the lookup map.

3.1 Content creation pipeline

This process aims at finding suitable lookup coordinates on the drawing for the parts of the mesh that are not visible in the drawing, given a UV-mapped mesh and its projection as a drawing. We want to find a lookup map that generates a texture that is continuous over the boundary between hidden and visible parts of the mesh in the drawing and over seams. We also want to fill hidden parts with patterns similar to the ones on visible parts, with minimal distortions between them.

3.1.1 Production process and variables used

The artist produces (1) a UV-mapped mesh (Fig. 2a); (2) an island map I in UV space that defines regions that shall receive similar coloring (Fig. 2b); and (3) a drawing, which is a combination of a view of the mesh through an edge shader – for printing – and a mapping from coordinates on this drawing to points in the UV map (Fig. 2c).

Based on these three items, the following objects are created: (1) the wrapping seams W^i for island I^i, mapping some pixels of I^i together; (2) the orientation map O in the UV space, giving a local orientation of the projected mesh structure; and (3) the lookup map L, indicating for every pixel of the character texture which drawing pixel to read. $L = L_\Phi \cup L_\Omega$, where L_Φ are regions visible in the drawing (source regions) and L_Ω are regions not visible in the drawing (target regions).

3.1.2 Lookup map initialization and relaxation

The most-challenging part of the content creation pipeline is the creation of L, based on the mesh, the projected drawing, and the island map I. To do so, W, O (Fig. 2d), and L_Φ (Fig. 2e) are first created. Then, for every island I^i, a first approximation of L^i_Ω is generated by copying coordinates from L^i_Φ. This approximation is very rough and violates the continuity desideratum, in particular across seams. Therefore, the next step is to enforce this continuity (Fig. 2f).

We can frame this problem in a similar way as the one of energy minimization in a spring system. Assuming that every pair of neighboring points of L – including across seams – are connected by a virtual spring, relaxing all springs will lead to fulfill the continuity constraints. To do so, we have to minimize the total energy of the system:

$$\sum_{(p,q)\in N_L} k_{p,q}(||L[q]-L[p]||-1)^2 \qquad (1)$$

for all pairs p,q which are either direct neighbors or seam neighbors in L; $k_{p,q}$ being a constant specific to the pair (p,q) that compensates the texel density in the UV map, which is not constant compared to the density in the model space. This correction is necessary for the generated texture motifs to be unstretched.

3.2 Live pipeline

Given a set of colored drawing templates (Fig. 2c), the App takes the live camera stream as the input (Fig. 2g), detects (Fig. 2i) and tracks (Fig. 2j) in 3D one of the templates appearing in the camera image. In this work, we explicitly account for the fact that the drawing paper may not be flat and may change its shape during the coloring, which is a common situation for books. Once the drawing shape is tracked in 3D, accurate color information from the colored drawing can be retrieved for our texturing algorithm, unlike other existing works [11] that can only deal with planar book pages. The augmented character is then overlaid on the input image by using the retrieved colors and the 3-D pose of the drawing (Fig. 2m).

3.2.1 Image processing

Given the camera image stream of the colored drawing, we want to process the input image so that the colored drawing appearance is as close to the original template as possible (Fig. 2h). Our approach achieves this by exploiting the fact that it is a line art drawing. This step is necessary because the appearance of the drawing changes significantly due to the coloring.

3.2.2 Template selection

After the input image is processed to be close to the original line art drawing templates, the system automatically detects which template is appearing in the camera stream (Fig. 2i). The selected drawing template is used as the template image in our template-based deformable surface tracking algorithm and later for drawing the augmented character.

3.2.3 Deformable surface tracking

Allowing the page to deform creates many challenges for the tracking algorithm since the number of degrees of freedom in deformable surface tracking is much higher than that in rigid object tracking. Our deformable surface tracking (Fig. 2j) builds upon previous work [24]

and makes it fast enough to run in real time on mobile devices and robust enough to handle colored line art drawings. We will formulate the shape reconstruction problem as shown in [24].

We represent the drawing page by a pre-defined 3-D triangular mesh with vertex coordinates stacked in a vector of variables $\mathbf{x} = [\mathbf{v}_1; \ldots; \mathbf{v}_{N_v}]$. The drawing page in its rest shape corresponds to the triangular mesh in the flat rectangular configuration with the paper size as dimensions. To make the template appear similar to the one captured by the camera, we synthetically render a reference image in the camera view in which the template image occupies 2/3 of the camera view.

To reduce the problem dimension, instead of solving for all mesh vertex coordinates, we use a linear subspace parameterization that expresses all vertex coordinates as a linear combinations of a small number N_c of control vertices, whose coordinates are $\mathbf{c} = [\mathbf{v}_{i_1}; \ldots; \mathbf{v}_{i_{N_c}}]$, as demonstrated in a previous work [25]. There exists a linear relation $\mathbf{x} = \mathbf{P}\mathbf{c}$, where $\mathbf{P}$ is a constant parameterization matrix.

Given point correspondences between the reference image generated above and an input image, recovering the colored drawing shape in this image amounts to solving a very compact linear system

$$\min_{\mathbf{c}} \quad \|\mathbf{M}\mathbf{P}\mathbf{c}\|^2 + w_r^2\|\mathbf{A}\mathbf{P}\mathbf{c}\|^2, \text{ s.t. } \|\mathbf{c}\| = 1\,, \tag{2}$$

in which the first term enforces the re-projection of the 3-D reconstructed mesh to match the input image data encoded by matrix $\mathbf{M}$, the second term regularizes the mesh to encourage physically plausible deformations encoded by matrix $\mathbf{A}$, and w_r is a scalar coefficient defining how much we regularize the solution. This linear system can be solved in the least-square sense up to a scale factor by finding the eigenvector corresponding to the smallest eigenvalue of the matrix $\mathbf{M}_{w_r}^T\mathbf{M}_{w_r}$, in which $\mathbf{M}_{w_r} = [\mathbf{M}\mathbf{P}; w_r\mathbf{A}\mathbf{P}]$. Its solution is a mesh whose projection is very accurate but whose 3-D shape may not because the regularization term does not penalize affine deformations away from the reference shape. The initial shape is further refined in a constrained optimization that enforces the surface to be inextensible

$$\min_{\mathbf{c}} \quad \|\mathbf{M}\mathbf{P}\mathbf{c}\|^2 + w_r^2\|\mathbf{A}\mathbf{P}\mathbf{c}\|^2, \text{ s.t. } C(\mathbf{P}\mathbf{c}) \le 0\,, \tag{3}$$

giving the final 3-D pose of the drawing in the camera view. $C(\mathbf{P}\mathbf{c})$ are inextensibility constraints that prevent Euclidean distances between neighboring vertices from growing beyond their geodesic distance in the reference shape.

3.2.4 Texture creation and mesh rendering

Once the 3-D shape of the colored drawing has been recovered in the view of camera, the mesh is re-projected onto the image plane. This re-projection defines a direct mapping between the pixels on the original drawing template and the pixels on the image of the colored drawing. We then can generate the texture for the character mesh using the lookup map L (Fig. 2k). Using the live view as the background image for the 3-D scene, and using proper parameters for the virtual camera, we can render the augmented character in the 3-D pose of the page using the generated texture from the drawing (Fig. 2l).

In Sec. 4.2, we will present in detail how we tackle the steps described above and compare to [24] our improved method to achieve a robust, accurate, and real-time tracking system for colored drawings.

4 Implementation

4.1 Generation of the UV lookup map L

4.1.1 Initialization of visible parts of the UV lookup map

To create the visible part L_Φ of the UV lookup map:

1. An artist prepares the texture-mapped mesh and the island map I. Each island has a unique color, unused boundary pixels are marked as well.
2. An optional mask can be used to exclude some pixels of L_Φ. This allows, for example, ignoring the printed black outlines in the coloring book. The artist is responsible for deciding which pixels to exclude.
3. A *unique color* UV texture is generated, in which every pixel has a color that encodes its position.
4. The mesh is rendered using the unique color texture, producing an image where each pixel encodes the UV position that was used to create it.
5. L_Φ is computed by traversing each pixel in the rendered image, computing its position in the UV space, and recording its destination at that location.

4.1.2 Creation of the seam map

The seam map W is created from the input mesh and the island map I. The resulting seams describe which points in the UV space correspond to the same points on the mesh. Only seams within a UV island are considered.

1. The mesh is transformed into a graph in UV space; each vertex is mapped to one or more nodes having exactly one UV coordinate and an island index.
2. All sets of nodes that correspond to the same vertex are added to a candidate set. Edges connecting nodes that are part of the candidate set are added to the graph.
3. A set of pairs of seams is extracted by traversing these edges starting at the candidate points, while ensuring that a seam pair consists of corresponding nodes and has the same topology.
4. Seam edges are rasterized so that not only vertices, but all points on seams are linked together in W.

4.1.3 Creation of the orientation map

The orientation map O encodes a locally-consistent orientation derived from the edges of the input mesh in UV space.

1. The mesh is transformed into a graph in UV space, where each vertex is mapped to one or more nodes.
2. For each node, we normalize the directions of connecting edges and cluster them into direction groups. If there are two clusters, we store the average directions, otherwise we ignore the node.
3. The orientation is made consistent by traversing the graph and by rotating the directions of nodes.
4. The projected mesh in UV space is rasterized to assign to each pixel the orientation of the nodes of the corresponding face.

4.1.4 Creation of an initial complete UV lookup map

The algorithm for creating an initial lookup map is shown in Algorithm 1. It takes a two-step approach. First, it creates an initial mapping by copying coordinates from the known region (source) into the unknown one (target) in a way that maintains continuity constraints at the border between these regions. Second, it removes discontinuities at seams by expressing neighboring constraints as a spring system, whose total energy must be minimized.

The creation of the initial mapping works by copying coordinates from one side of the border to the other side. The algorithm first propagates a gradient from the border of the source region into the target region. The generation of that gradient uses the E* algorithm [28], which is a variation of A* on a grid with interpolation. Then, starting from points close to the source, for every point in the target the algorithm counts the distance to the source, by tracing a path following the gradient. Once in the source, the algorithm continues to trace the path in the already-mapped area until it has run for the same distance as it did in the target. If the tracing procedure encounters the end of the already-mapped area, it reverses the orientation of the path. This procedure leads to rough copies of the source area being written into the target area.

Algorithm 1 shows the most common execution path. In addition to this normal path, the island I and the mesh can be arbitrarily pathological: some areas in Ω might be unconnected to Φ or there can be saddle points in the gradient. Therefore, the algorithm needs a procedure to recover from exception cases; this is represented by FixBrokenPoints(). While iterating over points in Ω, the algorithm collects all points that fail to reach a valid point in Φ, and stores by island i them for further processing. Then, for every of them it checks whether one of its neighbor is valid, and if so, it copies its mapping. For the remaining points, which typically belong to unconnected regions in Ω^i, it groups them in connected blobs and tries to copy a consistent

Algorithm 1 The initial creation of the lookup map

procedure GENERATELOOKUP(I,L_Φ,W,O)
  ▷ First approximation
  $L \leftarrow \emptyset$
  for i **in** enumerate(I) **do**
    $G \leftarrow$ generate gradient for island i
    $L^i \leftarrow L^i_\Phi$ ▷ initialize with source
    for p **in** sorted(Ω^i, G) **do** ▷ for points in target
      $d, p' \leftarrow 0, p$
      while $p' \notin L^i_\Phi$ **do** ▷ until source reached
        $p' \leftarrow$ descend G from p'
        $d \leftarrow d+1$ ▷ count distance in target
      end while
      $v \leftarrow$ incoming direction ▷ enter source
      while $d > 0$ **do** ▷ trace same dist. as in target
        if $p' \notin L^i$ **then** ▷ unknown mapping
          $v \leftarrow -v$ ▷ reverse direction
        else
          $L^i[p] \leftarrow L^i[p']$ ▷ copy mapping
        end if
        rotate v using $O[p']$ ▷ follow source orientation
        $p', d \leftarrow p'+v, d-1$
      end while
    end for
    $L \leftarrow L \cup L^i$ ▷ merge lookup for this island
  end for
  FIXBROKENPOINTS()
  ▷ Relaxation
  $e \leftarrow \infty$ ▷ iterative relaxation
  for c **in** range($c_{\max}$) **do**
    $L, e' \leftarrow$ RELAXLOOKUP(L, W)
    if $e' > e$ **then** ▷ if error increases...
      break ▷ ... stop early
    end if
    $e \leftarrow e'$
  end for
  return L
end procedure

mapping based on the center of the largest region in Φ^i. If some pixels cannot be copied, the algorithm assigns a point from Φ^i.

The algorithm then solves Eq. 1 by performing iterative relaxation. For each point, it attaches a spring to its neighbors (4-connectivity) and, if this point is on a seam, to the point on the other side of the seam (using the seam map W). The spring constant is adjusted to account for the distortion in the UV map. The algorithm iteratively relaxes all springs, using a simple iterative gradient descent method. The relaxation stops if the error does not diminish for a continuous number of steps. In our experiments, we set this number to 8; we set the maximum number of iterations $c_{\max}$ to 100,000 but the algorithm typically stops early after 4,000–20,000 steps.

4.2 Deformable surface tracking

In this section, we describe our algorithm to detect and track in 3D a possibly non-flat deformable colored line drawing paper. We rely on wide-baseline feature point correspondences between the reference image and the input image. For this, we propose to use BRISK [20] in place of the memory-intensive Ferns [26] used in [24]. Since many of the correspondences are erroneous, we propose a new outlier rejection algorithm, which is faster and more robust than the one used in [24]. We reformulate the reconstruction energy function to gain speed while not sacrificing accuracy. We rely more on frame-to-frame tracking to gain frame rate and only apply the feature detection and matching once every N frames to retrieve back lost tracked points and accumulate good correspondences. A mask indicating the region of interest in which to look for feature points is also constructed to speed up the feature point detection and limit gross outliers.

(a) (b) (c)

Fig. 3: Image processing: (a) Input image of the colored drawing. (b) Luminance channel in HSV color space. (c) Resulting line draw image.

4.2.1 Image processing

In this step, we apply image processing techniques to the input image so that the appearance of the colored drawing becomes as close to the original template as possible. Because in the next step of the pipeline we describe feature points by binary descriptors using intensity comparisons, this step is especially important to be able to better detect which drawing template is selected and to provide more inlier wide-baseline feature point correspondences between the input image and the reference image.

In the coloring book context, we want to remove colors and keep the black lines in the image of the colored drawing. Therefore, we transform the input image from RGB to HSV color space, in which only the luminance channel is used because it captures most information about the original black line draws. Line draws are more visible in the HSV luminance channel than in the gray scale image. We then apply adaptive thresholding to the luminance image to get a binary line draw image. Small noisy connected components are removed and Gaussian smoothing with standard deviation $\sigma = 1$ pixel is used to remove the staircase effect of binarization. This process is demonstrated in Fig. 3.

Our color removal procedure is completely automatic without having to adjust parameters for different capturing scenarios, in contrast to [11], where the thresholds must be adjusted for different lighting conditions and for different cameras.

4.2.2 Template selection

To detect which template appears in the camera, we compare the processed input image with a set of given templates. We detect a sparse set of feature points and use a voting scheme to determine which template is currently visible and should be selected for further steps.

Later in the shape reconstruction phase, we also need to establish feature point correspondences between the selected template and the input image. In our problem context, it is necessary to choose a suitable feature point detection, description, and matching method that has reasonable memory and CPU consumption, and provides good quality matches. We have tested a number of methods including Ferns [26], BRIEF [9], ORB [30], FREAK [1], BRISK [20], SURF [3], and found BRISK to be the most suitable choice for our application. BRISK detects scale and rotation invariant feature points, the descriptors are binary vectors, which can be compared very quickly using NEON instructions widely available in mobile processors nowadays, and the matching quality is good enough for our purpose. Clark et al. [11] uses SURF in their coloring book application, which we found too computationally intensive. Differently, in the context of deformable surface tracking, [24] extracts scale and rotation invariant interest points as maxima of the Laplacian and then uses the Ferns classifier [26] to do the matching. However, Ferns is so memory intensive that it is not suitable for our real-time tracking on mobile devices.

We detect about 500 feature points on each generated reference image, and extract their binary descriptors. Doing so allows all templates to have an equal chance of being selected in the voting scheme. We automatically adjust parameters to extract between 300–2000 features points on the processed input image. The idea of our voting scheme is that each feature descriptor in the input image will vote for one template which has the closest descriptor according to the Hamming metric.

Although comparing two binary descriptors can be done very quickly using NEON instructions, performing brute-force search for the nearest

neighbour in the template descriptor database is prohibitive. We instead project all binary descriptors to a low d-dimension space ($d = 7$ in our settings) using a pre-generated random projection matrix. K nearest neighbours ($K = 100$ in our settings) in this low dimension space can be searched very quickly using k-d trees. It also scales well with respect to the number of templates. We use the *libnabo* library [15] to perform K-nearest-neighbour search. We then go back to the original binary space and select the nearest descriptors among these K candidates. A Hamming threshold is used to filter truly good matches. The template with highest votes is selected.

4.2.3 Outlier rejection

Once the template has been selected, wide-baseline correspondences are established between the template and the processed input image. This matching step for only two images is done very quickly using brute-force search in the Hamming metric. We obtain a set of putative correspondences, and outliers must then be removed.

Outlier rejection in 3D: The method in [24] eliminates erroneous correspondences by iteratively solving Eq. 2, starting with a relatively high regularization weight w_r and reducing it by half at each iteration. The current shape estimate is projected on the input image and the correspondences with higher re-projection error than a pre-set radius are removed. This radius is then reduced by half for the next iteration. Repeating this procedure a fixed number of times results in a set of inlier correspondences.

This outlier rejection procedure can reject up to about 60 percent of outliers as reported in [24]. One of its main limitations is that it solves for a 3-D mesh, which involves depth ambiguity and a larger number of variables compared to a 2-D mesh. We propose to perform outlier rejection in 2D similarly to [29] but our algorithm can work with both regular and irregular meshes and is much faster thanks to the linear subspace parameterization.

Outlier rejection in 2D: We represent the 2-D deformable surface by a 2-D triangular mesh and use the regularization matrix $\mathbf{A}$ mentioned in Sec. 3.2.3 on the x and y components to regularize the mesh.

Unlike [29], which requires a regular 2-D mesh and uses the squared directional curvature of the surface as the smoothing term, our regularization term can work on both regular and irregular meshes. We solve for a 2-D mesh, which is smooth and matches the input image. The linear subspace parameterization $\mathbf{x} = \mathbf{Pc}$ still works on a 2-D triangular mesh and is used to reduce the complexity of the problem. We solve the following optimization problem:

$$\min_{\mathbf{c}} \quad \rho(\mathbf{BPc} - \mathbf{U}, r) + \lambda_s^2 \|\mathbf{APc}\|^2 \,, \tag{4}$$

where $\mathbf{c}$ represents 2-D control vertices, $\mathbf{A}$ is the regularization matrix, $\mathbf{B}$ represents the barycentric coordinates of the feature points in matrix form, and $\mathbf{U}$ encodes the feature point locations in the input image. Further, ρ is a robust estimator whose radius of confidence is r and is defined as

$$\rho(\delta, r) = \begin{cases} \frac{3(r^2-\delta^2)}{4r^3} & -r < \delta < r \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Instead of introducing a viscosity parameter α and iteratively solving two coupled equations with a random initial solution as in [29], we solve Eq. 4 directly using a linear least squares approach with a big starting radius of confidence and reduce it by half at each iteration. The result of this iterative process is a both robust and very fast outlier rejection algorithm.

Note that our 2-D outlier rejection does not prevent us from tracking the surface in 3D. We use the obtained set of inlier correspondences to track and reconstruct the surface fully in 3D.

4.2.4 Surface reconstruction by detection

Once outlier correspondences are eliminated, we solve Eq. 2 only once and scale the solution to give initialization for the constrained optimization in Eq. 3. The works in [25, 24] formulate the inequality constraints $C(\mathbf{Pc}) \leq 0$ as equality constraints with additional slack variables whose norm is penalized to prevent lengths from becoming too small and the solution from shrinking to the origin. They solve a complex optimization problem involving extra variables and hard constraints:

$$\begin{aligned} \min_{\mathbf{c},\mathbf{s}} \quad & \|\mathbf{MPc}\|^2 + w_r^2 \|\mathbf{APc}\|^2 + \mu^2 \|\mathbf{s}\|^2, \\ \text{s.t.} \quad & C(\mathbf{Pc}) + \mathbf{s}^2 = 0 \,. \end{aligned} \tag{6}$$

In contrast, we use soft constraints that allow the edge lengths to slightly vary around their reference lengths. We obtain a simpler optimization problem with fewer variables and still arrive at sufficiently accurate reconstructions for AR purposes. We further use a motion model to temporally regularize the solution. Since the tracking frame rate is high, a linear motion model is enough. We solve

$$\begin{aligned} \min_{\mathbf{c}} \quad & \|\mathbf{MPc}\|^2 + w_r^2 \|\mathbf{APc}\|^2 + \lambda^2 \|C(\mathbf{Pc})\|^2 \\ & + \gamma^2 \|\mathbf{c}_{t-2} - 2\mathbf{c}_{t-1} + \mathbf{c}\|^2 \,, \end{aligned} \tag{7}$$

in which $\mathbf{c}_{t-1}$ and $\mathbf{c}_{t-2}$ are solutions to previous two frames.

Using the linear motion model, we can predict the 3-D pose of the drawing in the next frame and create an occlusion mask where the surface projection should be in the next input image. This technique helps to speed up the feature point detection and matching. It also improves the robustness of the system because gross outliers are limited.

4.2.5 Surface reconstruction by tracking

In the tracking mode, we make use of the fact that both the surface shape and the camera pose change only slightly between two consecutive frames. We use the motion model to predict the shape for the current frame and use the result to initialize the reconstruction.

Similar to [24], we track inlier correspondences from frame to frame on grayscale images using the standard Lukas-Kanade algorithm [6]. This step brings a great help that allows the system to track under extreme tilts and large deformations, successfully.

We rely on frame-to-frame tracking to gain frame rate and only apply the feature detection and matching once every $N = 10$ frames to retrieve back lost tracked points and accumulate good correspondences. This simple technique turns out to be efficient for our problem.

4.3 Interactive coloring book

The interactive coloring book App is built using the Unity game engine[3] and runs on Android and iOS. It uses Unity to access the camera through the `WebCamTexture` object, and fetches the pixels into the main memory. These are then passed to a C++ library implementing the deformable surface tracking algorithm. This library tells Unity whether a drawing template is detected and if so returns the 3-D shape of the possibly non-flat colored drawing in the camera coordinates. The App then rectifies the image of the colored drawing in the canonical fronto-parallel pose. It does so by projecting the vertices of the triangular mesh representing the drawing 3-D shape into the image plane, and using these projected points as texture coordinate to draw a rectified grid with the camera image as texture, in an offscreen render target. Finally, the App renders the camera image using a screen-aligned quad, and overlays the corresponding animated 3-D character. The virtual camera for the 3-D scene is positioned in function of the location of the tracked surface in the real world. The character is rendered with a texture filled by getting colors from the the rectified camera image through the lookup map L.

5 Evaluation

5.1 Texture synthesis

5.1.1 Performance

Our artist required less than one hour per drawing to create the inputs for the algorithm. Then, the computational cost is about half an hour on a MacBook Pro 2014 2.5 GHz. The implementation currently uses Cython, and could be accelerated massively, should it use the GPU.

[3] `http://unity3d.com`

Fig. 4: Completion of the texturing of 3-D models from a drawing, for different models (M1–M3) and methods (A, P, C, L).

5.1.2 Method

We conducted a web-based quantitative user study to evaluate the subjective performance of our lookup-based texture generation method (L), in comparison to three other methods: completion of the texture by a professional artist (A); projection of the drawing to the model from the point of view of the drawing (P); and completion using the Photoshop CS6 *content-aware* filling tool, by applying it to each island separately (C). Methods L and P share the same run-time component: copying pixels from the drawing to the texture using a lookup map, but they differ in how this map is generated. Method C synthesizes pixels in function of the existing patterns, and is thus considerably slower. Finally, method A is a manual operation that takes about 20 minutes when applied by a professional artist.

The participants were presented with 36 screens, each showing an image of a 2-D drawing along with two images of a 3-D model, each textured with 2 different methods. A short text explained that the coloring of the 3-D models were automatically generated from the 2-D drawing and the participant was asked to choose the image that looked aesthetically more pleasant. This was repeated for 3 different models, 2 colorings and 4 methods; all combinations were presented in random order. Fig. 4 shows a selection of these combinations. We collected the number of total votes for each method n_A, n_P, n_C, and n_L for each participant, interpreted as dependent variables (DV). Additionally, we collected the age, gender, and we asked the question whether the participant plays video games at least once a month in average (to measure the participant's general expertise with animated content), as independent variables (IV). The user study was disseminated through professional and personal contacts and by announcement on reddit's *gamedev*, *augmentedreality*, *computervision*, and *SampleSize* channels. We collected the results over a duration of two days.

5.1.3 Results

A total of 314 persons completed the study: 84% male and 16% female, 74% with expertise and 26% without expertise. The median age was 28 (min: 18, max: 78); we use the median statistics as the number of votes is a discrete space. A Friedman test was run to determine if there were differences in n_A, n_P, n_C, and n_L. Pairwise comparisons were performed with a Bonferroni correction for multiple comparisons. The number of votes for the methods were statistically significantly different

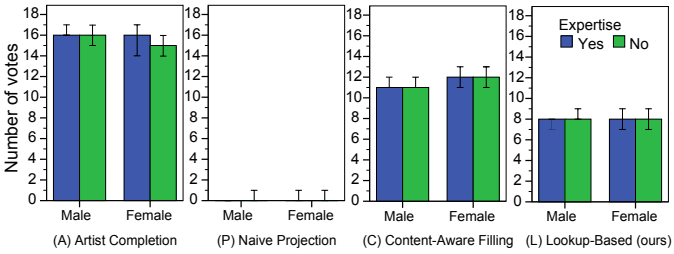


Fig. 5: Results from the user study to evaluate our texture synthesis method. The median and the 95% confidence interval are shown for the number of votes for the compared methods (A, P, C, L).

($X^2(3) = 816.07, p < 0.001$). Post hoc analysis revealed statistically significant differences ($p < 0.001$) among all pairwise combinations of methods. Fig. 5 plots the median and the 95% confidence interval of n_A, n_P, n_C, and n_L, ordered by gender and expertise.

The preferred completion method is the drawing by a professional artist (A). The second preferred method is the content-aware filling of Photoshop CS6 (C), probably because it tends to generate more smooth textures than ours (Fig. 4, M1-C vs M1-L). However, it does not handle seams, which can lead to dramatic results in some cases (Fig. 4, M2-C). On the contrary, our method has more graceful degradations (Fig. 4, M2-L). The naive method P is consistently disliked by everyone. A Chi-Square test revealed that no significant association between all pairwise combinations of IV and DV could be observed ($p > 0.05$), that is, no influence of age, gender, or expertise on the number of votes could be confirmed. This is consistent with the strong agreement of participants and shows that our method, albeit a bit worst than Photoshop's content-aware filling (which is not real-time), is far better appreciated than a naive approach.

Our proposed texture generation algorithm was necessary to meet the real-time requirements of our application. However, being lookup-based, it has some limitations, as being unaware of the users' coloring at run-time. Nonetheless, it provides superior results to a naïve approach, comparable to much slower offline methods. We conclude that it is good enough for real-time application in a coloring book.

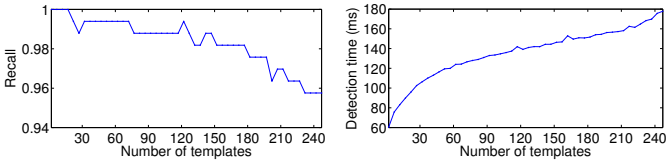


Fig. 6: Scalability of the template detection algorithm. Left: recall measure. Right: average detection time.

5.2 Deformable surface tracking

In this section, we experimentally validate the robustness, accuracy, and computational efficiency of our methods and show that they are suitable to our coloring book application on mobile devices.

5.2.1 Template selection

We captured an image sequence of colored drawings in normal settings of book coloring and input it into the template selection module to test how accurately we can detect a template if it appears in the camera. To study the scalability of our detection algorithm, we vary the number of templates from 2 to 250, which is much more than the number of pages a coloring book usually has. We define the recall measure as the ratio of correct detection among the total number of test input images. In our case, false positives are not very severe because these false alarms will be rejected in the outlier rejection step. As shown in Fig. 6, our template selection algorithm can detect the template accurately and its computational performance scales well with the number of templates. Since we detect the template only once before passing the result to the tracking module, this detection can be run across several frames.

5.2.2 Robustness

We demonstrate the robustness of our outlier rejection scheme described in Sec. 4.2.3 for planar surfaces and compare it to [24, 29]. To evaluate our approach, we used the same image from the standard paper dataset as in [24], in which the surface undergoes large deformations. We used both the corresponding Kinect point cloud and SIFT correspondences to reconstruct a mesh that we treated as the ground truth. Similar to [24, 29], we then produced approximately one million sets of correspondences by synthetically generating 10 to 180 inlier correspondences spread uniformly across the surface, adding a zero mean Gaussian noise ($\sigma = 1$ pixel) to the corresponding pixel coordinates to simulate feature point localization errors, and introducing proportions varying from 0 to 100 % of the randomly spread outlier correspondences.

We ran our proposed outlier rejection algorithm with 20 regularly sampled control vertices on each one of these sets of correspondences and defined a criteria to assess the effectiveness of our outlier rejection scheme. The criterion for success is that at least 90 % of the reconstructed 3-D mesh vertices project within 2 pixels of where they should. Fig. 7 depicts the success rates according to this criteria as a function of the total number of inliers and the proportion of outliers. The results reported in Ngo et al. [24] and Pilet et al. [29], which have similar experiment settings as ours, are included for comparisons. Note that unlike us and Ngo et al. [24], Pilet et al. [29] does not add Gaussian noise to the input correspondences. Nevertheless, it takes our method approximately only 80 inlier correspondences to guarantee that the algorithm will tolerate up to 0.95 ratio of outliers with 0.9 probability. The algorithm decays nicely with respect to the number of inlier input matches. It is still 50 % successful given only 50 inlier matches and a 0.95 outlier ratio. Compared to [24, 29], our rejection algorithm requires fewer inlier correspondences to detect the surface and can handle a larger portion of outliers.

Fig. 8 demonstrates the robustness of our outlier rejection algorithm on real images. The algorithm can filter 37 inlier matches out of 422 putative ones. As a result, accurate color information can be retrieved for the texture generation algorithm (see the virtual characters in Fig. 8).

Pilet et al. [29] reports the total execution time of 100 ms per frame in a 2.8 GHz PC including feature point detection and matching time. Our outlier rejection algorithm is similar except that we solve a much smaller least squares problem. For the outlier rejection algorithm alone, on a MacBook Pro 2014 2.6 GHz laptop, it takes Ngo et al. [24] about 25 ms per frame while it only takes our method about 2 ms per frame on the same input correspondences.

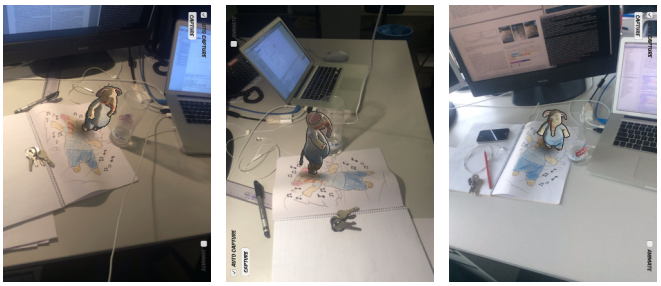


Fig. 8: Screenshot from the App showing the robustness of our outlier rejection algorithm.

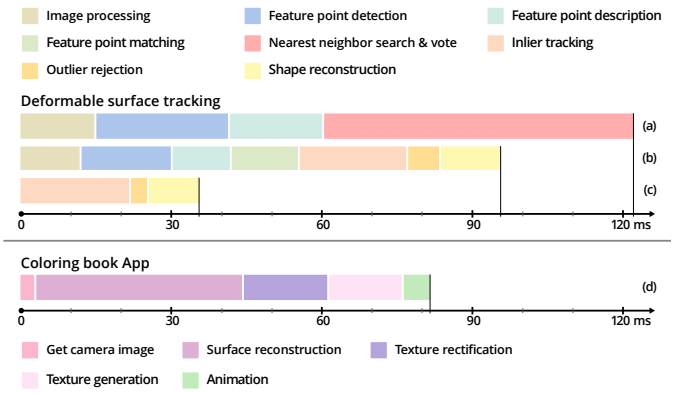


Fig. 9: Timing measurements of major steps of the deformable-surface tracking algorithm and the coloring book App. (a) Template selection. (b) Surface reconstruction in the detection plus tracking mode. (c) Surface reconstruction in the tracking mode. (d) Coloring book App.

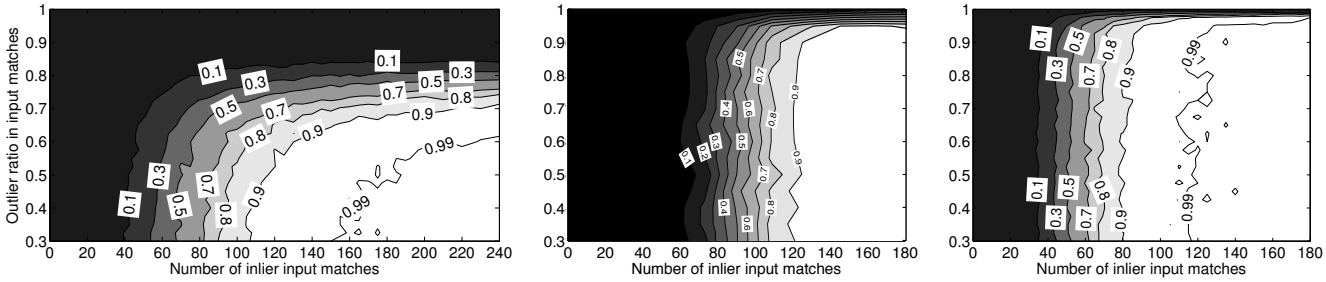
5.2.3 Reconstruction accuracy

We compared the reconstruction accuracy of our method described in Sec. 4.2.4 against [24] which is representative of the current state-of-the-art in template-based deformable surface reconstruction. We used the same dataset as for the Robustness evaluation (see preceding section). This dataset provides 193 consecutive images acquired using a Kinect camera [37]. The template is constructed from the first frame, and 3-D reconstruction is performed for the rest of the sequence using the image information alone. Both methods use 20 regularly-placed control vertices. We used the Kinect point cloud and SIFT correspondences to build ground truth meshes, and compute the average vertex-to-vertex distance from the reconstructed mesh to the ground truth mesh as the measure of reconstruction accuracy. The average reconstruction accuracy of [24] is 2.83 mm while our average reconstruction accuracy is 2.74 mm, which is comparable. However, on a MacBook Pro 2014 2.6 GHz laptop, it takes [24] about 70 ms per frame to solve the optimization problem in Eq. 6, while it takes our method only 3.8 ms to solve the optimization problem in Eq. 7.

5.2.4 Timing

In this section, we look in detail at the timing of the major steps (see Sec. 4.2) of our deformable surface detection and tracking algorithm on an iPad Air on which we prototyped our App. We used a sample coloring book with 3 drawing templates and measured the average time spent on each step. The results are shown in Fig. 9.

In the template selection mode, the total time spent per frame is 122 ms (Fig. 9a). These measurements were averaged over 480 frames. In the surface reconstruction by detection plus tracking mode, both



(a) Ngo et al. [24] (b) Pilet et al. [29] (c) Ours

Fig. 7: Probability of having at least 90 % of mesh vertices re-projected within 2 pixels of the solution as a function of the number of inlier matches and proportion of outliers, on the x-axis and y-axis respectively. The lines are level lines of the probability.

surface reconstruction by detection and frame-to-frame surface tracking are performed. The reconstruction time per frame is 96 ms, averaged over 50 frames (Fig. 9b). Once the colored drawing is properly tracked, using a mask indicating the region of interest to look for feature points decreases the time spent on feature point detection and description because only a small region of the input image is processed and disturbing feature points are filtered out. In the surface reconstruction by tracking mode, image processing, feature detection, description, and matching are not required. The average total reconstruction time per frame is 36 ms (Fig. 9c). These measurements were averaged over 250 frames.

Most of the time of our algorithm is spent on feature point detection, description, and matching. We perform surface reconstruction by detection once every 10 frames, and perform surface reconstruction by tracking on the rest of the frames. On average, it takes the reconstruction 41 ms per frame. Our approach has some limitations: the computational cost varies for different frames and we have not fully exploited the tracking context yet. To avoid feature point detection, description and matching, a future work is to incorporate active search to locally look for correspondences in tracking mode. Doing so, we believe our deformable surface tracking method will have a performance closer to its planar rigid object tracking counterpart.

5.3 Interactive coloring book

5.3.1 App performance

The App runs at about 12 FPS on an iPad Air (first generation). Fig. 9(d) shows the breakdown of the time for one frame, averaged over 600 measurements. Most of the time, 41 ms, is spent on tracking the deformable surface. Rectifying the texture to match the drawing template takes 17 ms, while creating the texture using the lookup map L takes 15 ms. The latter step could be strongly optimized by doing it through a shader. Finally, acquiring the image from the camera and rendering the scene takes 9 ms.

Since, in the context of a coloring book, the deformation of the drawing paper is relatively small, we experimentally observed that 20 control vertices out of total 110 vertices are enough. The corresponding App frame rate is 12 FPS. The frame rate decreases to 9, 6, 5, 3 FPS with 40, 60, 80, 110 control vertices, respectively.

Our approach relies on feature points, whose discriminativeness in line art drawings is not as high as in natural textures. It thus requires the drawing to have sufficient detail. To reliably track the whole page, we also rely on feature points detected around the character, for instance in decorations printed in margins. Occlusions due to the hand of the user are handled well because besides tracking feature points from frame to frame, which causes drifts and point losses, we also periodically perform feature points detection and matching to retrieve back lost tracked points, correct drifts, and accumulate good correspondences. The frame rate of 12 FPS, however, is not enough to handle sudden motions. In future work, optimisation of implementation would improve the frame rate and provide better user experience.

5.3.2 End-user evaluation, method

We conducted a user study to assess the subjective performance of our App. Each participant was asked to choose one out of three templates (see Fig. 4) and color it with a set of colored crayons. There was no time limit and multiple subjects were allowed to sit together, chat, and color their chosen template. An iPad Air running our App was available and the participants were asked to occasionally inspect their coloring through the App. Upon finishing coloring, the participants were asked to answer the following questions: Tablet Usage: *How often do you use a tablet in average?*, Drawing Skill: *Does this augmented reality App app affect your drawing skills?*, Drawing Motivation: *Does this augmented reality App affect your motivation to draw?*, Connection to Character: *Does this augmented reality App affect your feeling of connection with the cartoon character?*, and Would Use App: *If this App was available for free, would you use it to draw characters?*. Additionally, we recorded their age, gender, and their comments. In the analysis we treated Tablet Usage, age, and gender as IV and Drawing Skill, Drawing Motivation, Connection to Character, and Would Use App as DV.

5.3.3 End-user evaluation, results

We recorded answers from a total of 40 participants, 38 % male and 62 % female. The median age was 23 (min: 18, max: 36). While our App is designed to be used by children, due to delays with ethical approval for the experiment, we did not quantitatively test it with this age group. Despite this limitation of our study, informal tests showed a strong appreciation and engagement. In a further work, additional testing with children is desirable to quantify these. Fig. 10 shows a histogram of the answers, for the different questions. Chi-Square tests revealed that no statistically significant association could be observed between any pairwise combination of IV and DV ($p > 0.05$).

The majority of participants (60 %) felt that the App would increase their motivation to draw coloring books, while a large minority (40 %) said that their motivation was not affected. This shows that while the App should be made more engaging, it already has a positive effect on motivation. The App is perceived as having a minor pedagogical value, as 20 % of the participants said that it affected their drawing skills positively. Almost all participants (80 %) felt that the App increased their feeling of connection to the character, except for 2 people who actually felt the contrary. This may be due to imperfect tracking for their actual drawings, ruining the experience. A majority (75 %) of the participants would use this App, if it was available for free. About a third of the participants gave additional comments, mostly a short sentence of appreciation. Two participants noted that the texture synthesis was not perfect, but found it impressive nevertheless. One suggested to add the possibility to control the virtual character. These results show that the proposed solution is mature and well appreciated, and contributes to enrich the coloring experience.

6 Conclusion

In this paper we have demonstrated an augmented reality App that allows children to color cartoon characters in a coloring book and examine an animated 3-D model of their drawing. We concluded from a user study that this App strongly improves the sense of connection with the character, and motivates people to draw more. This App is built upon two novel technical contributions in the field of (1) real-time

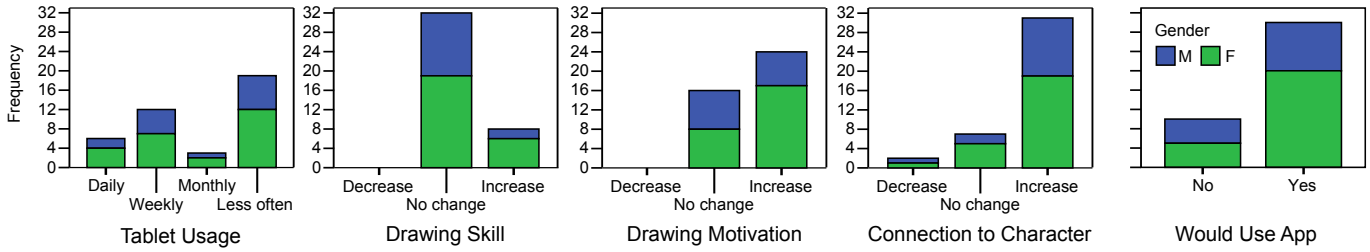


Fig. 10: Histograms of the answers to the questionnaire evaluating the end-user appreciation of the AR coloring book App.

texture synthesis and (2) deformable surface tracking. For the first, we formulated the problem of generating a texture in real time and proposed a solution by creating a lookup map to copy existing pixels from the drawing. We proposed a novel algorithm for artists to generate this map automatically. For deformable surface tracking, we provided a new template selection and a new outlier rejection mechanism, as well as a lighter problem formulation, which together allow real-time tracking on a mobile device. While we have used these contributions to enable our AR coloring book, we believe they have the potential to impact many more applications in the field of visual computing.

Acknowledgments

The authors thank Julien Pilet and Oliver Wang for their contributions and the reviewers for their insightful comments. T.D. Ngo was supported in part by the Swiss National Science Foundation.

References

[1] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast retina keypoint. In *CVPR*, pages 510–517, 2012.

[2] A. Bartoli, Y. Gérard, F. Chadebecq, and T. Collins. On template-based reconstruction from a single view: Analytical solutions and proofs of well-posedness for developable, isometric and conformal surfaces. In *CVPR*, pages 2026–2033, 2012.

[3] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In *ECCV*, pages 404–417, 2006.

[4] M. Billinghurst, H. Kato, and I. Poupyrev. The MagicBook - moving seamlessly between reality and virtuality. *IEEE Computer Graphics and Applications*, 21(3):6–8, May 2001.

[5] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *SIGGRAPH*, pages 187–194, August 1999.

[6] J.-Y. Bouguet. Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm. *Intel Corporation*, 5:1–10, 2001.

[7] F. Brunet, A. Bartoli, and R. Hartley. Monocular template-based 3D surface reconstruction: Convex inextensible and nonconvex isometric methods. *Computer Vision and Image Understanding*, 125:138–154, 2014.

[8] F. Brunet, R. Hartley, A. Bartoli, N. Navab, and R. Malgouyres. Monocular template-based reconstruction of smooth and inextensible surfaces. In *ACCV*, pages 52–66, 2010.

[9] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. In *ECCV*, pages 778–792, 2010.

[10] A. Chhatkuli, D. Pizarro, and A. Bartoli. Stable template-based isometric 3D reconstruction in all imaging conditions by linear least-squares. In *CVPR*, pages 708–715, 2014.

[11] A. Clark and A. Dunser. An interactive augmented reality coloring book. In *IEEE Symposium on 3D User Interfaces*, pages 7–10, March 2012.

[12] T. Cootes, G. Edwards, and C. Taylor. Active appearance models. In *ECCV*, pages 681–685, 1998.

[13] A. Criminisi, P. Pérez, and K. Toyama. Region filling and object removal by exemplar-based image inpainting. *Image Processing, IEEE Transactions on*, 13(9):1200–1212, 2004.

[14] A. Delbue and A. Bartoli. Multiview 3D warps. In *ICCV*, pages 675–682, 2011.

[15] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics*, 3(1):2–12, 2012.

[16] J. Fayad, L. Agapito, and A. Delbue. Piecewise quadratic reconstruction of non-rigid surfaces from monocular sequences. In *ECCV*, pages 297–310, 2010.

[17] R. Garg, A. Roussos, and L. Agapito. Dense variational reconstruction of non-rigid surfaces from monocular video. In *CVPR*, pages 1272–1279, 2013.

[18] C. Guillemot and O. Le Meur. Image inpainting : Overview and recent advances. *IEEE Signal Processing Magazine*, 31(1):127–144, Jan 2014.

[19] J. Herling and W. Broll. High-quality real-time video inpaintingwith PixMix. *IEEE Transactions on Visualization and Computer Graphics*, 20(6):866–879, 2014.

[20] S. Leutenegger, M. Chli, and R. Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *ICCV*, pages 2548–2555, 2011.

[21] J. Liu, P. Musialski, P. Wonka, and J. Ye. Tensor completion for estimating missing values in visual data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):208–220, Jan. 2013.

[22] Y. Liu and V. Caselles. Exemplar-based image inpainting using multiscale graph cuts. *IEEE Transactions on Image Processing*, 22(5):1699–1711, May 2013.

[23] F. Moreno-Noguer, M. Salzmann, V. Lepetit, and P. Fua. Capturing 3D stretchable surfaces from single images in closed form. In *CVPR*, pages 1842–1849, 2009.

[24] D. Ngo, J. Ostlund, and P. Fua. Template-based monocular 3D shape recovery using laplacian meshes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015. Accepted for publication.

[25] J. Ostlund, A. Varol, D. Ngo, and P. Fua. Laplacian meshes for monocular 3D shape recovery. In *ECCV*, pages 412–425, 2012.

[26] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):448–461, 2010.

[27] M. Perriollat, R. Hartley, and A. Bartoli. Monocular template-based reconstruction of inextensible surfaces. *International Journal of Computer Vision*, 95(2):124–137, 2011.

[28] R. Philippsen. A light formulation of the E* interpolated path replanner. Technical report, Autonomous Systems Lab, Ecole Polytechnique Federale de Lausanne, 2006.

[29] J. Pilet, V. Lepetit, and P. Fua. Fast non-rigid surface detection, registration and realistic augmentation. *International Journal of Computer Vision*, 76(2):109–122, February 2008.

[30] E. Rublee, V. Rabaud, K. Konolidge, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *ICCV*, pages 2564–2571, 2011.

[31] M. Salzmann and P. Fua. *Deformable Surface 3D Reconstruction from Monocular Images*. Morgan-Claypool, 2010.

[32] M. Salzmann and P. Fua. Linear local models for monocular reconstruction of deformable surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):931–944, 2011.

[33] M. Salzmann, F. Moreno-Noguer, V. Lepetit, and P. Fua. Closed-form solution to non-rigid 3D surface registration. In *ECCV*, pages 581–594, 2008.

[34] C. Scherrer, J. Pilet, P. Fua, and V. Lepetit. The haunted book. In *ISMAR*, pages 163–164, 2008.

[35] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Symposium on Geometry Processing*, pages 175–184, 2004.

[36] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Transactions on Graphics*, 23(3):399–405, 2004.

[37] A. Varol, M. Salzmann, P. Fua, and R. Urtasun. A constrained latent variable model. In *CVPR*, pages 2248–2255, 2012.