Animato: 2D Shape Deformation and Animation on Mobile Devices

Stefan Messmer[1,2] Signe Fleischmann[1] Olga Sorkine-Hornung[2]
[1]kindergarten.io [2]ETH Zurich

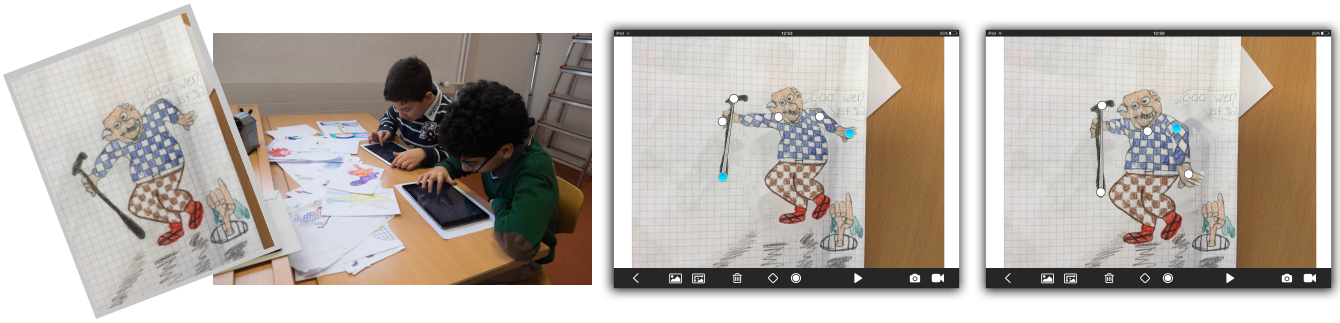

Figure 1: *Novice users animate their own drawings using our app. Left: input drawing and two of our testers working collaboratively with the app on a short animation movie. Right: two frames showing the input character being interactively deformed.*

Abstract

We present Animato, an interactive app for the animation of 2D shapes using an intuitive multi-touch interface and a novel collaborative multi-user mode. We describe our implementation of a state-of-the-art nonlinear shape deformation method on iOS devices, overcoming their hardware limitations. Informal testing with users shows that even novices have an easily accessible entry point to computer animation and are quickly able to record animation performances using our app thanks to the direct and intuitive multi-touch interface and the fast shape deformation algorithm.

Keywords: computer animation, shape deformation, skinning, mobile devices, multi-touch, collaborative interfaces

Concepts: •**Computing methodologies** → **Animation;**

1 Introduction

Computer animation is a popular form of art and entertainment, with expressive characters fascinating audiences of all ages worldwide. Yet, creating own animations is challenging and largely inaccessible: while sketching and drawing is relatively easy for lay users, bringing drawings to life with computer animation typically requires using complicated software with a steep learning curve, attainable only by trained artists. Recent research efforts on shape deformation [Jacobson et al. 2016a] have produced algorithms that enable interactive deformation and posing of articulated shapes via a click-and-drag mouse interface: arbitrary points (or "handles") can be attached to a given shape and dragged around using the mouse, yielding intuitive and visually pleasing deformations of the shape that can further serve as animation keyframes. However, since these methods involve sophisticated numerical optimization, they were originally implemented on desktop computers with ample memory, CPU and GPU capacities. Additionally, manipulating a shape rendered on the screen using a mouse is indirect and imposes a cognitive load, plus only one control handle can be manipulated at a time, such that "live puppeteering" of a character is nearly impossible. These factors make the barrier for the widespread use of desktop animation techniques relatively high.

In this work, we implement a state-of-the-art shape deformation method on the mobile iOS platform and take advantage of the multi-touch display capabilities for a direct shape manipulation interface. Additionally, we propose a new, collaborative interface that enables multiple users to animate a shape together by simultaneously manipulating it on several mobile devices. We discuss our solutions to overcoming the hardware limitations of the mobile devices while preserving the realtime frame rate of the deformation algorithm. We test the features of our app with novice users, including school children, confirming that our system provides an easy, intuitive and playful introduction to animation. Novice users are able to create their own animations quickly and naturally, which places our approach at an advantage over current desktop systems.

2 App workflow and user interface

Advantages of direct shape manipulation with multi-touch interfaces have already been demonstrated on large collaborative displays [Igarashi et al. 2005]. Our app, called Animato, brings these advantages to small touch screens of mobile devices, thereby significantly increasing accessibility and potential impact. The pipeline of Animato has four main steps, and users can go back and forth between them (see Fig. 2):

1) Input image segmentation. The user selects an image of the desired shape from the photo library, or directly shoots it from the app, and scribbles some strokes inside and outside the shape to be animated. The app automatically segments the foreground shape and shows the result by superimposing a semi-transparent colored mask. The user can refine and rectify the segmentation by adding more strokes. Once the segmentation is satisfactory, the user presses the "next" button to continue to the second step. At this stage, the foreground is extracted and the background is inpainted to create a coherent background canvas. It is also possible to load a different background altogether.

2) Rigging. The textured shape is displayed, and the user may tap on it to place control handles in form of point handles and skeletal

SA '16 Symp on Mobile Graphics and Interactive Applications, December 05-08, 2016, Macao
ISBN: 978-1-4503-4551-4/16/12
DOI: http://dx.doi.org/10.1145/2999508.2999528

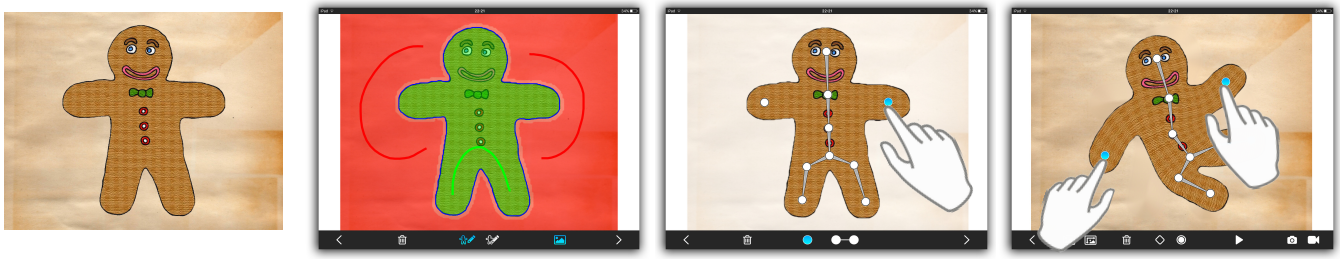

Figure 2: *The workflow of our app Animato. In single-user mode, all steps are performed on the same device. In multi-user collaborative mode, the "master" user performs steps 1 and 2, while step 3 (animation) is performed simultaneously by all users, each manipulating a different set of control handles and watching the combined animation on their device. See Sec. 2 for more details.*

bones. Point handles are useful for controlling extremities and soft parts of the shape, while skeletal controls represent rigid parts, such as legs. The user can toggle between point handle mode and skeleton mode in the UI at the bottom of the screen. When in skeleton mode, tapping on the shape creates a new bone segment that connects between the currently tapped point and the previously selected point. To add a new bone that connects to a different joint, the user just needs to tap on that joint to select it, and then tap on a new location to create the endpoint of the new bone. Once the user is satisfied with the created control structure, she can press the "next" button in the UI, and the app performs the binding of the controls to the shape and all necessary precomputation for the subsequent realtime deformation.

3) Deformation and animation. The user activates one or multiple control handles (point handles or skeleton nodes and bones) by tapping on them and then drags them around with the fingers. The shape interpolates the displaced handles and deforms to follow the user interaction in a smooth, plausible and intuitive way. The user can switch to recording mode in the UI, in which case the transformations of the handles are continuously saved to record the realtime animation performance. The user can also record partial animation performances with the available fingers and combine recordings: whenever the recording mode is entered, the previously saved animation is continuously played back and the currently active interaction handles are newly recorded and combined with the existing animation.

4) Exporting the result. The user can export the recorded animation to a movie, or just save the current deformation result as a single image to the photo library of the device.

The app also offers gesture UI for zooming and translation, such that the user can explore various parts of the shape in better detail and also translate its global positioning. The gestures are the familiar iOS two-finger pinching/spreading for zoom and dragging a finger outside the shape for translation (cf. Fig. 3).

Collaborative mode. In this mode, one mobile device is assigned as the master device, and its user selects the shape to animate and performs steps 1 and 2 above (segmentation and rigging). Then, multiple users can join the deformation and animation session with the app in collaborative mode on their devices. They can either just watch the animation on their device or actively manipulate the interaction handles and the positioning of the shape. The handles grabbed by a user are marked as "occupied", so that other users may choose to manipulate other handles. The resulting animation is updated in real time on all devices. For example, one user can animate the arms, another one the legs of a character, and a third user can be in charge of its global translation in the scene (see Fig. 3 for an example). Each user may choose her own preferred zoom level without affecting the display for other users.

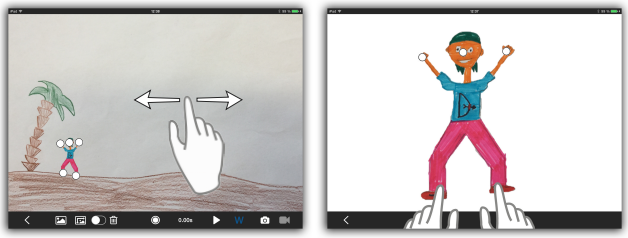

Figure 3: *Gestures and collaborative mode. One user animates the global position of the character on the background canvas (left), while another simultaneously animates the character's legs on a separate device (right).*

3 Algorithms and implementation

We have chosen iOS as our platform and implemented Animato on the iPad, with an adaptation to the iPhone currently underway. The segmentation, rigging and realtime deformation steps of the workflow require solving challenging optimization problems, even more difficult on mobile platforms, which have limited memory and computing resources. Below we present our algorithmic choices and system design to tackle this challenge.

Input image segmentation. If the loaded image contains an alpha mask, we can use it directly to identify the foreground shape to be animated. Otherwise, we rely on the user to draw a few scribbles inside and outside the desired shape and feed these as constraints to GrabCut segmentation [Rother et al. 2004] implemented in OpenCV [Itseez 2016]. To provide the user with immediate preview of the segmentation, the input image is downsampled by a power of 2 (to a resolution smaller than 512×512 in the current implementation). This way, the user can refine the segmentation by adding more scribbles and receive prompt feedback. The final segmentation is then computed on the fine resolution using the result of the coarse segmentation as the prior (this takes a few seconds), and the largest foreground component is selected as the shape to animate. In order to use the rest of the input image as a background canvas for the animation, we perform inpainting inside the contour of the foreground shape using the simple method of fast marching cubes [Telea 2004] available in OpenCV. Although it provides very

basic results, the running time is extremely fast, unlike the more sophisticated patch based inpainting methods, which unfortunately take minutes on the iPad if run out of the box. Finally, we compute a triangle meshing of the foreground component.

Rigging. The result of the previous step is a textured triangle mesh of the shape to be animated (plus a background canvas). After the user specifies all the control handles (points and bones), the app performs precomputations necessary for the realtime deformation in the next step. For efficiency, Animato uses linear blend skinning (LBS) for deforming the shape, namely, each vertex $\mathbf{v}$ of the mesh is transformed by the formula

$$\mathbf{v}' = \sum_{j=1}^{m} w_j(\mathbf{v})\, \mathbf{T}_j \mathbf{v}, \tag{1}$$

where $w_j(\mathbf{v})$ are scalar per-vertex weight functions associated with each control handle j and $\mathbf{T}_j$ are affine transformations of the handles, supplied for each frame during the animation stage. Homogeneous coordinates are used for the vertices to enable linear representation of translations. The skinning weights w_j are fixed per choice of mesh and deformation handles, and so they are precomputed in the rigging stage. We use the bounded biharmonic weights of [Jacobson et al. 2011], which require solving a sparse quadratic programming problem whose size is proportional to the number of mesh vertices, n.

LBS alone requires tedious input from the user, because in order to create visually pleasing shape deformations, not just translations but also rotations must be specified for each control handle, precluding the simple dragging interface we would like to use. We therefore use the FAST method (Fast Automatic Skinning Transformations) of [Jacobson et al. 2012], which automatically computes the rotational parts of the $\mathbf{T}_j$'s by minimizing the nonlinear as-rigid-as-possible (ARAP) energy of the deformed mesh. Evaluating the energy on the mesh requires performing $O(n)$ singular value decompositions for local rotation optimization, which is too expensive, but the FAST method approximates the energy by clustering the local rotations, such that only several dense matrices of size $O(m)$ are required at the rigging stage (where $m \ll n$); these are then used for the optimization iterations during the deformation stage. For details please refer to [Jacobson et al. 2012].

To implement FAST in iOS, we rely on Eigen [Guennebaud et al. 2010] for linear algebra computations and libIGL [Jacobson et al. 2016b] for mesh structures and geometry processing. Although libIGL contains a reference implementation of FAST, we cannot employ it directly, as it is not optimized for the specifics of mobile devices. Instead, we reimplement the algorithm using NEON vectorization, the SIMD of the ARM architecture, to get good performance on the iPad and iPhone.

Deformation and animation. During the realtime deformation and animation stage, the vertex positions are updated by the LBS formula in Eq. (1), and the resulting textured mesh is continuously displayed. The LBS computation is implemented using a specific OpenGL shader to enable the performance of 60 frames per second, which results in smooth animations and is also the native refresh rate of the iPad screen. The transformations $\mathbf{T}_j$ in Eq. (1) are computed on the CPU. They partially consist of the translations directly specified by user manipulation of the handles; their remaining degrees of freedom are automatically computed by the iterative FAST optimization, where each iteration involves singular value decompositions of $O(m)$ 2×2 matrices and matrix-vector multiplications of size $O(m)$. Since the number of handles m is typically low, these computations are fast and memory-efficient.

Collaborative mode. Sharing the animation experience between several devices requires high frame rate of the deformations (60 fps) and very low latency, such that all users are well synchronized. The frame rate is guaranteed by our choice of deformation algorithm and its implementation, as described above, and the low latency must be ensured by the data exchange framework and protocol. We employ Apple's MultiPeer framework to connect several iOS devices over WiFi and create a stream to exchange data packets. We choose WiFi over Bluetooth, since the latter underperformed in our tests, with fluctuating bandwidth and latency of more than 70 ms, which is visually noticeable. When using a WiFi access point, a network between the devices is created using Bonjour, which automatically discovers all devices that use the same service. Since all data is routed through the access point, and since in our setting one user has to prepare the animation by segmentation and rigging, it is natural to employ the client/server communication model, where the server is the device that prepares the scene. The server is in charge of sending the scene information and the precomputed data to each client when it connects, and synchronizing all clients by continuously sending them the full state of the current animation. Each client sends updates about its contribution to the animation to the server. The measured latency using WiFi is only about 30 ms, which is not perceivable.

The complete state of the animation in each frame is defined by the m transformation matrices $\mathbf{T}_j$ and the model view matrix, i.e., the position and size of the animated shape w.r.t. the canvas. Our exchange protocol can be sketched as follows:

Client: (1) look for server by starting Multipeer Browser; (2) if a server is found, ask for connection; (3) if the connection succeeds, receive background image, mesh, texture and precomputed matrices of the FAST algorithm; (4) send updates of changed transformations and their handle indices; (5) retrieve transformation updates from the server and incorporate the changes; (6) repeat (4) and (5).
Server: (1) look for incoming connections by starting the Multipeer Advertiser; (2) accept connection and send the background image, mesh, texture and precomputed matrices for FAST; (3) receive partial updates for handles and incorporates them; (4) send out the complete set of all transformations (needed to synchronize all connected clients) for every frame; (5) repeat (3) and (4).

Thanks to our choice of shape deformation algorithm based on direct manipulation handles, the amount of data that defines the current state is low. A back-of-the-envelope calculation shows that for a setup with 4 handles, a bandwidth of approximately 40 KB/s is required to obtain the target frame rate of 60 fps.

4 Evaluation and discussion

In the process of developing the app, our colleagues tested its various features, and we received very positive feedback. However, since most our colleagues come from a computer graphics background, we decided to conduct an informal user study with true novice users, namely, school children (see Figs. 1, 4). We visited a class of third graders (9 years old) in primary school. The children were asked to use their own drawings for background canvases and characters, which they created on paper a week in advance in painting class. For the single user mode, 4 children at a time worked on 4 iPads. The task was to create a short sequence with their character walking around on a background by setting up two handles on the legs. Next, the children were free to create any animation by combining multiple sequences and also adding more interaction handles. We also tested the collaborative mode, where the children worked in groups of 2 or 3, with one user moving the character around on the canvas, and the others articulating the extremities.

Observing the children revealed that they had an easy access to Animato, as most were already familiar with the iPad, so that the barrier to trying a new program was very low. It was quite natural for them

Figure 4: *School students in third grade performing an informal user study of Animato using their own hand drawn characters and background images. The students tested both the single-user and collaborative multi-user modes and successfully created short animation movies.*

to quickly get their drawings into the app by photographing and segmenting them using the in-app UI. The children needed some explanation and help when setting up the handles. Overall, everyone succeeded in creating short animation sequences with their own characters, which would have been difficult to impossible for children using a desktop computer – and they had fun using the app. The collaborative mode also worked well and enabled the creation of more complex and expressive animations. At the same time, we learned that certain interactions are challenging when multiple users are involved, namely, it is difficult to synchronize the timing of the various components of the animation. For example, creating a coherent walking sequence when one person controls the arms and the other controls the legs is difficult without a "conductor" mechanism. Using a metronome or music for rhythm helps a lot in such situations, and we would like to investigate effective modes of multi-user interaction in future research.

The hand-sized touch screen offers the advantage of direct connection between the touched locations and the animated shape, and it is easy to trace smooth curves by moving the fingers. This way, realtime "acting" performance of the animation becomes possible, which is much faster and more intuitive than keyframing. This is in stark contrast to the standard mouse-and-keyboard, single-touch interface, where the movement of the mouse is not directly equal to that of the handles on screen, and the translation between the two is cognitively taxing. On the other hand, touch sensing on the iPad and the iPhone is less precise than working with the mouse, and the fingers occlude parts of the animated scene, so that designing detailed, precise deformations is more difficult on such devices.

5 Conclusions and future work

We presented our design of an iPad app that enables animating 2D shapes using the multi-touch interface by a single user or in a collaborative mode. We have tested the app with school children, who successfully created short animated movies of characters that they had drawn themselves. The direct connection between finger movements and the animated shape, as well as the multi-touch ability to manipulate several handles at once make recording a realtime performance of an animation easy and efficient. We conclude that our app provides a playful and easy entry point for novice users into computer animation, and the state-of-the-art shape deformation algorithm we implemented could be helpful for experienced users as well when creating rough concept sketches of animations.

In the future, we are interested in exploring the addition of dynamics to the animations, potentially exploiting the sensors available in mobile devices, such as the accelerometer and the orientation sensor. Another important extension is the animation of 3D shapes. While the deformation algorithm itself readily works in 3D, the user interface poses significant challenges: the direct connection between the shape geometry and the fingers manipulating it is lost due to perspective projection, and at the same time manipulation precision is more important in 3D. Solving these challenges on easily accessible mobile devices would help introduce the public to creative expression in 3D, and we leave this as exciting future work.

Acknowledgements

We are grateful to Alec Jacobson, Daniele Panozzo and Alexander Sorkine-Hornung for illuminating discussions and support, and to Bettina Sigrist and her students for volunteering their time to the user studies. This work was funded in part by the SNF grant 200021_162958 and the ERC grant iModel (StG-2012-306877).

References

GUENNEBAUD, G., JACOB, B., ET AL., 2010. Eigen v3. http://eigen.tuxfamily.org.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3, 1134–1141.

ITSEEZ, 2016. OpenCV: Open source computer vision library. https://github.com/itseez/opencv.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4, 78:1–78:8.

JACOBSON, A., BARAN, I., KAVAN, L., POPOVIĆ, J., AND SORKINE, O. 2012. Fast automatic skinning transformations. *ACM Trans. Graph. 31*, 4, 77:1–77:10.

JACOBSON, A., DENG, Z., KAVAN, L., AND LEWIS, J. 2016. Skinning: Real-time shape deformation. In *International Geometry Summit 2016, invited course*.

JACOBSON, A., PANOZZO, D., ET AL., 2016. libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/.

ROTHER, C., KOLMOGOROV, V., AND BLAKE, A. 2004. "GrabCut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph. 23*, 3, 309–314.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, vol. 1148 of *Lecture Notes in Computer Science*. Springer-Verlag, 203–222.

TELEA, A. 2004. An image inpainting technique based on the fast marching method. *Journal of Graphics Tools 9*, 1, 23–34.