Video Puppetry: A Performative Interface for Cutout Animation

Connelly Barnes¹ David E. Jacobs² Jason Sanders² Dan B Goldman³ Szymon Rusinkiewicz¹ Adam Finkelstein¹ Maneesh Agrawala² ³Adobe Systems

¹Princeton University ²University of California, Berkeley

Figure 1: A puppeteer (left) manipulates cutout paper puppets tracked in real time (above) to control an animation (below).

Abstract

We present a video-based interface that allows users of all skill levels to quickly create cutout-style animations by performing the character motions. The puppeteer first creates a cast of physical puppets using paper, markers and scissors. He then physically moves these puppets to tell a story. Using an inexpensive overhead camera our system tracks the motions of the puppets and renders them on a new background while removing the puppeteer's hands. Our system runs in real-time (at 30 fps) so that the puppeteer and the audience can immediately see the animation that is created. Our system also supports a variety of constraints and effects including articulated characters, multi-track animation, scene changes, camera controls, 2¹/₂-D environments, shadows, and animation cycles. Users have evaluated our system both quantitatively and qualitatively: In tests of low-level dexterity, our system has similar accuracy to a mouse interface. For simple story telling, users prefer our system over either a mouse interface or traditional puppetry. We demonstrate that even first-time users, including an eleven-yearold, can use our system to quickly turn an original story idea into an animation.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques-Interaction Techniques; I.3.7 [Computer Graphics]: Three-dimensional Graphics and Realism-Animation

Keywords: real-time, animation, tangible user interface, vision

1 Introduction

Creating animated content is difficult. While traditional handdrawn or stop-motion animation allows broad expressive freedom, creating such animation requires expertise in composition and timing, as the animator must laboriously craft a sequence of frames to convey motion. Computer-based animation tools such as Flash, Toon Boom and Maya provide sophisticated interfaces that allow precise and flexible control over the motion. Yet, the cost of providing such control is a complicated interface that is difficult to learn. Thus, traditional animation and computer-based animation tools are accessible only to experts.

Puppetry, in contrast, is a form of dynamic storytelling that performers of all ages and skill levels can readily engage in. Puppeteers directly manipulate physical objects - puppets - to bring them to life and visually enact the motions in real-time. Even young children commonly manipulate objects (e.g. dolls, paper-cutouts, toy cars, etc.) as they create and tell stories.

Yet, puppet shows typically have lower visual and motion fidelity than animations. Puppeteers and their controls are often visible to the audience. Motions must conform to the laws of physics and are usually less precise than carefully crafted animated sequences. Unlike animations, puppet shows are ephemeral and cannot be viewed repeatedly. Nevertheless, the real-time performative nature of puppetry makes it a powerful medium for telling stories.

In this paper we present a video-based puppetry system that allows users to quickly create cutout-style animations by performing the character motions. As shown in Figure 1 the puppeteer works with familiar media including paper, markers and scissors to create paper-cutout puppets. He or she then physically moves these puppets to tell the story. Our system tracks the motions of the puppets using an overhead camera, and renders them on a new background while removing the puppeteer's hands. In addition, our system applies a variety of special effects, to enhance the richness of the resulting animations. Our system works in real-time (at 30 fps) so that both the puppeteer and the audience can immediately see the animation they are generating. Such real-time feedback is essential to retain the performative nature of puppetry.

Our work is related to motion capture techniques which allow actors to physically demonstrate motions and animate characters in real-time. However, these techniques usually require extensive setup, both in terms of complex and expensive hardware and elaborate character rigging. More recently enthusiasts have used game engines to create animated content called *machinima*. Games offer real-time feedback and their controls are designed to be very easy to learn. However, users are limited to working with pre-designed environments and characters with a narrow range of motions. Thus, this approach significantly limits freedom of expression.

The primary contribution of our work is the interface for creating animated stories via the performance-based direct manipulation paradigm of puppetry. This interface is designed to present a shallow learning curve to users of all skill levels, while placing as few limits as possible on creativity in the design of characters, painting of backgrounds, and telling of stories. Our camera-based input is inexpensive, using hardware that many users already own. Moreover, the effects possible with our system are difficult to achieve with systems such as stop-motion animation or traditional puppetry. Although the resulting animations cannot match the visual quality of a studio production, they are suitable in many contexts, including kids' productions, animatics, or the class of animations such as "South Park" or "JibJab" that explicitly target a "cutout" aesthetic.

We describe an initial study in which we evaluate our system in three ways. First, in a low-level task measuring motor skills, we compare our system against a mouse-based interface and find no significant difference in accuracy. Second, in a mid-level task, users tell a specific story using three different tools — our system, a mouse interface, and traditional puppetry — and evaluate which method they prefer. In this test, all users preferred our system. Third, at a higher level, users draw their own characters and tell their own stories. Our results indicate that even first-time users, including an eleven-year-old, can use our system to quickly turn a story idea into an animation.

2 Related Work

Developing an interactive and easy-to-learn interface that people of all skill levels can use to create animation is a long-standing problem in computer graphics. Here we focus on interface techniques that go beyond standard GUI-based solutions.

Sketch-based animation: Many systems allow users to create animatable objects and demonstrate their motions via sketching [Baecker 1969; Moscovich and Hughes 2001; Davis et al. 2008]. However, while drawing 2D translational motion paths is easy, even simple 2D rotations and scales can be difficult to sketch. Moreover, these systems only allow one object to move at a time and therefore require multiple record-and-playback cycles to build a multi-object *layered* animation.

Several systems explore techniques for sketching articulated figures and their motions. Moscovich and Hughes [2001] describe an IK-based layered animation control system in which users select multiple joints to control different layers of motion. Thorne et al. [2004] develop a gestural interface in which users are limited to working with a library of pre-defined motions. Davis et al. [2003] provide an interface for converting 2D drawings of skeletal keyposes into 3D articulated figure animations. Since users must draw the keyposes this system is not demonstration-based.

Another approach is to combine sketching with simulation techniques. Alvarado and Davis [2001] use physics-based mechanical simulation, while LaViola and Zeleznik [2004] use general math equations to animate sketches. Popović et al. [2003] allow users to draw motion paths for 3D objects and then run rigid-body simulations to compute physically plausible motion that best matches the sketch. While these techniques can generate high-quality motion, the user must give up some control over the resulting animation.

Performance animation: Motion capture systems are designed to interactively transfer an actor's physical performance to a virtual character. Computer puppetry [Sturman 1998] is another form of performance animation in which the performer uses a specialized

input device such as an articulated armature [Knep et al. 1995] to control the character in real-time. Both Oore et al. [2002] and Dontcheva et al. [2003] have developed interfaces for quickly creating and editing layered character animations using either motion capture or tracked props. Despite their ease of use, however, these techniques are out of reach for most users because they require expensive specialized hardware. Moreover, using these systems requires some expertise in building animatable 3D characters and specifying the mapping between the motions of the actor or input device and those of the character.

Video-based animation: In some cases it is much faster to capture video of a real scene than it is to create an animation of the scene. Yet, real-world video does not have the same visual character as most animation. One way to resolve this problem is to apply non-realistic video rendering techniques [Wang et al. 2004; Winnemöller et al. 2006; Bousseau et al. 2007] on the input video. With more sophisticated video processing, such as optical flow calculation or object tracking, it is possible to add non-photorealistic motion cues such as deformations and streak lines to the video [Collomosse and Hall 2005; Collomosse and Hall 2006; Wang et al. 2006]. However, none of these techniques can change the visual content of the video - they only modify its visual style. If the user's hand is in the input video it remains in the output animation. Video cutout [Wang et al. 2005] and user-guided rotoscoping methods [Agarwala 2002; Agarwala et al. 2004] can address this problem as they provide more control over the composition of the output video. Yet, these techniques do not work in real-time and therefore cannot be part of a performance-based interface.

Tangible and multi-touch interfaces: Our work is inspired by tangible and multi-touch interfaces. Tangible interfaces [Ishii and Ullmer 1997] allow users to manipulate physical objects to control the computer. These interfaces usually allow coordinated two-handed input and provide immediate visual feedback about the position of the physical objects. In our system the tangible paper puppets act as the physical interface objects. Moreover, paper is a familiar material with well-known affordances, so most users are immediately aware of the kinds of puppets they can create and the kinds of motions they can perform.

Recently Igarashi et al. [2005] have shown that direct multitouch control is a natural interface for controlling non-rigid deformable characters. Although our paper-based system similarly allows such multi-touch control over the puppets, it also assumes the paper puppets are rigid or articulated models and cannot generate non-rigid animations. Nevertheless, because users of our system work with physical puppets, they benefit from kinesthetic feedback about the position and orientation of the puppets, and can lift puppets off the table to control scale and depth ordering using a more natural affordance than a multi-touch device can provide. Moreover, the table-top and camera interface offers several other benefits relative to multi-touch, such as ease of selection of off-screen characters and components, the ability to scale to large (multi-performer) workspaces, and the relative ubiquity and low cost of cameras.

Video-based paper tracking: Several systems have been designed to recognize and connect physical paper documents and photographs with their electronic counterparts [Wellner 1993; Rus and deSantis 1997; Kim et al. 2004; Wilson 2005]. These techniques use video cameras to capture the movements of the physical documents and track the position and stack structures of these documents. Users can then choose to work with the tangible physical document or the virtual copy. Our system applies similar ideas and techniques to the domain of real-time animation authoring.

Augmented reality: The field of augmented reality has pioneered applications that combine camera-based input, low-level tracking and recognition techniques similar to those we use, and computer-

generated rendering. Indeed, our system falls in the under-explored "augmented virtuality" zone of Milgram's continuum [1994], and demonstrates a novel application in this area: animated storytelling. In contrast with some augmented reality systems, however, we explicitly focus on tracking unmodified paper puppets, as created by novice users. In particular, we do not use visually-coded tags for identification, as is done by some previous systems [Rekimoto and Ayatsuka 2000; Fiala 2005; Lee et al. 2005]. While the use of such tags would simplify tracking and recognition, it would impose unwanted additional burden on users: they would have to generate the tags, attach them to the physical puppets, and register the mapping between the tags and the puppets in the tracking system.

3 System Overview

Our video-based puppetry system is divided into two modules; a *puppet builder* and a *puppet theater*. From a user's perspective the first step in creating an animation is to construct the cast of physical puppets. The puppet builder module then captures an image of each puppet using an overhead video camera and adds it to a puppet database (Section 4). Then as the user manipulates the puppets to perform a story, the puppet theater module tracks the puppet movements and renders them into an animation in real-time.

Internally, the puppet theater module consists of a *tracker*, an *interpreter*, and a *renderer*. The input to the theater module is a video stream of the puppet movements. The tracker computes a mapping between the puppets in each video frame and their corresponding images in the puppet database, as well as a depth ordering between overlapping puppets (Section 5). The interpreter then applies constraints on the tracked transforms and detects non-actor puppets that trigger special rendering effects (Section 6). Finally, the renderer displays the output animation.

4 Puppet Builder

Our puppet builder provides an intuitive interface for adding new puppets to our system. A user first draws a character on paper with markers, crayons, or other high-contrast media. Next the user cuts out the puppet and places it under the video camera. The puppet builder captures an image of the workspace and processes the image in several stages before adding the puppet to a puppet database. For non-articulated puppets, the processing is automatic and takes about 15 seconds per puppet. We describe the puppet building process for articulated characters in Section 6.1.

In the first stage of processing the puppet builder recovers two mattes for the puppet (Figure 2). The *paper matte*, used during tracking, includes the entire piece of paper containing the puppet, while the *character matte*, used for overlap detection and rendering, includes only the region containing the character. To generate the paper matte we perform background subtraction on the input frame and apply a flood-fill seeded from a corner to isolate the paper boundary. We then fill the background with white and perform a second flood-fill to obtain the character matte. This approach



Figure 2: To matte out the puppet from the input frame our system first performs background subtraction. Next it applies a flood-fill technique to isolate the paper matte and then the character matte.



Figure 3: An overview of our tracker. The tracking path (bottom) is computed in real time on each frame, while the identification path (top) is computed as a background task.

is based on the assumptions that the puppet is evenly illuminated and that the character boundary is defined by an edge that contrasts sharply against the white paper background.

One consequence of our approach is that correct mattes are extracted only for characters of genus zero. Although other automated methods such as color range selection could allow for highergenus puppets, these techniques could not distinguish holes from intentionally-white regions in the puppet, such as eyes. Therefore, a user can only produce higher-genus mattes using an image editor.

In the second stage, the puppet builder generates data structures that will be used by the real-time puppet theater module to recognize puppets, track them, and detect overlaps.

5 Puppet Theater: Tracker

Although object tracking is a well-studied topic in computer vision [Trucco and Plakas 2006], our system places two key design requirements on the tracker. First, it must robustly recognize the wide variety of hand-drawn puppets users may create. Second, it must accurately track those puppets in real time. As shown in Figure 3, our approach takes advantage of the complementary strengths of two techniques. SIFT features are used to identify *all* puppets every 7-10 frames. Between SIFT updates, optical flow on KLT features is used to track the movement of puppets in real time.

We have found this combination of techniques to be relatively efficient and robust for the entire tracking pipeline: recognition, frame-to-frame propagation, and depth ordering. We tune it to be conservative in the following sense: it rarely reports incorrect puppet locations, at the cost of greater likelihood of losing tracking. When the tracking is lost, the on screen character simply freezes at its last known position. We have found this to be the most natural response, and users quickly figure out what happened. They then simply hold the character still until new SIFT features are found (typically within a few frames) and then the character jumps to the correct location.

5.1 Identifying Puppets

We recognize puppets using the Scale-Invariant Feature Transform, which selects a set of distinctive feature points from an input image and computes a descriptor based on edge orientation histograms at each point [Lowe 1999]. As noted by Kim et al. [2004], SIFT descriptors are well-suited for matching and recognition tasks because they are *invariant* under translation, rotation, and scale, *robust* to partial occlusion and illumination changes, and *distinctive*.

We compute SIFT features at a subset of video frames, matching each extracted feature to the most similar one in our precomputed database. Given a set of features matching to the same puppet, we apply a Hough transform to prune outlier matches. We then use least-squares minimization to compute a similarity transform that best maps the puppet image from the database to the video frame.

The main drawback of this approach is its computational cost. Even though we parallelize the feature matching, we are only able to execute the identification loop every 7-10 frames, depending on resolution (typically 640×480) and the size of the puppet database.¹ Thus, we only use this pathway to initialize puppet transforms and, as described below, to compensate for accumulated drift in our real-time tracking.

5.2 Tracking Puppets

Our real-time tracking pathway is based on the Kanade-Lucas-Tomasi (KLT) technique [Tomasi and Kanade 1991], which finds corner-like features and determines their motion using optical flow.² We identify tracked features with puppets and maintain puppet transformations as follows.

Identifying features: Newly-found KLT features are associated with a puppet if and only if they overlap the *character matte* of exactly one puppet known to be present. This may occur if we have successfully been tracking the puppet (based on other features), or if we have identified the puppet via the SIFT pathway. Thus, a new puppet will not be tracked until the identification pathway has run, and must remain motionless for up to 10 frames so that the SIFT position corresponds to the true position of the puppet. Similarly, if a puppet is moved while completely occluded by other puppets or the user's hands, we must wait until it is identified in its new position. However, if a puppet is occluded but is not moved, then as soon as it is unoccluded we propagate the puppet's identity from its last-observed position, and immediately resume tracking.

Tracking features: The motion computed by optical flow is used to propagate feature identities from frame to frame (Figure 4, left). We leave unassigned any features that overlap one puppet's *paper matte* and another puppet's *character matte*, since these are likely to be "phantom" points whose motion does not necessarily match the motions of either puppet (Figure 4, right). We use precomputed oriented bounding box (OBB) trees [Gottschalk et al. 1996] to accelerate the point location tests against the character and paper mattes (Figure 5).

Computing puppet transformations: Given the motion of features that have been assigned to a puppet in two consecutive frames, we use the method of Horn [1986] to solve for translation, rotation, and scale. This is used to update the puppet's transformation. Any features not consistent with the recovered transformation are considered outliers, and are marked as unassigned.

Compensating for drift: Because the puppet transformations are maintained by incrementally applying the results of frame-to-frame KLT tracking, they are subject to accumulation of errors (Figure 6). We can eliminate this error by periodically using the SIFT pathway to provide the ground-truth transformation. However, because SIFT requires k frames of computation time, we cannot remediate the error immediately. Instead, we correct the error at frame i + k by starting with the newly-obtained ground-truth position for frame i, and re-applying all subsequent frame-to-frame transforms. This can result in puppets "popping" into position, but we prefer this approach over smoothing with a predictive filter because the SIFT updates are far more reliable than the KLT updates. By using this



Figure 4: Left: a puppet annotated with KLT points. Right: when puppets overlap, the KLT tracker will often find points along their moving boundary. These "phantom" points (yellow) move in arbitrary ways, and are rejected as outliers.



Paper OBB Leaves

Character OBB Leaves

Figure 5: Leaves of the oriented bounding box trees for the paper and character mattes.



Figure 6: Correcting cumulative KLT errors. A SIFT input frame (left) and the current input video frame at the time of SIFT's completion (right). Blue rectangles show the KLT transforms for the puppet. The green rectangle shows the SIFT transform for frame i after k frames of processing time. We concatenate the SIFT transform with the incremental KLT transforms after frame i to produce the combined SIFT and KLT transform (red rectangle). Our system assumes that this combined transform is more correct than the KLT transform (blue) for frame i + k.

scheme, at any given instant the puppet's pose on the screen is always as accurate as possible given the available information.

Withdrawing puppets: As described above, if we are unable to locate any KLT features belonging to a puppet, we assume that it has been occluded and keep track of its last known location. An exception is made if the puppet was near the edge of the video frame – in this case, we assume that it was moved off-screen, and withdraw it from our list of active puppets.

5.3 Resolving Depth Ordering

When puppets partially occlude each other our tracker determines a depth ordering among them so that the virtual puppets can be rendered in the proper depth order. We begin by detecting occlusions: we check for overlap between the character matte OBB-trees of all

¹ We use Lowe's implementation to compute the SIFT features: http://www.cs.ubc.ca/~lowe/keypoints/

² We use the GPU-based KLT implementation by Sinha et al.: http://www.cs.unc.edu/~ssinha/Research/GPU_KLT/



Figure 7: Articulated puppets add an extra dimension of expressiveness not possible with single-body puppets.

pairs of visible puppets, using a 2D version of the separating axis test proposed by Gottschalk et al. [1996].

Once we have detected an occlusion, we must determine which puppet is frontmost. Although we could compare the image contents in the overlap region to the images of both puppets in the database, this would be neither computationally efficient nor robust. Instead we apply a sequence of tests to determine depth ordering:

- 1. **Number of KLT Features.** If one puppet owns more KLT features in the region of overlap, it is likely to be on top.
- 2. **Relative Scale.** A puppet whose image is larger than its initial size (computed by the puppet builder when it was lying on the table) is likely to be closer to the camera and thus on top.
- 3. **Velocity.** It is often difficult to move occluded puppets quickly. Therefore, the faster-moving puppet is likely to be on top.
- 4. **Puppet ID.** If no other test is conclusive, we arbitrarily consider the puppet with lower ID to be on top.

The tests are ordered by decreasing reliability and therefore we terminate as soon as one is deemed conclusive (based on an experimentally-determined threshold).

To maintain frame-to-frame coherence in depth ordering, we initially transfer all conclusive pairwise relationships from the previous frame to the current frame. Thus, the depth ordering can be flipped only if a more reliable test contradicts the previous result.

Given pairwise occlusion information, we perform a topological sort to obtain a global depth ordering among the puppets. Although the occlusion graph can contain cycles, in practice these are physically difficult to produce and are rare in normal interaction. We have therefore not implemented any complex heuristics for resolving occlusion cycles, simply breaking such cycles greedily.

6 Puppet Theater: Interpreter

With our basic tracking system alone, users can render animations of rigid puppets on new backgrounds while removing the puppeteer's hands. We have found that in many cases this is enough functionality to produce an effective animation. However, when we combine the raw transforms output by the tracker with an interpretation module, we can take full advantage of our animation tool as a digital system. The special effects and puppetry techniques made possible by an interpretation phase expand the user's creative horizons without sacrificing ease of use.

6.1 Articulated Puppets

Articulated puppets, composed of multiple rigid segments, physically pinned together at joints, are common in shadow puppetry and other traditional forms [Wisniewski and Wisniewski 1996]. While such articulated puppets are more difficult to create than puppets composed of a single rigid segment, they offer the possibility of much more expressive motions. Pinning the segments together



Figure 8: Multi-track animation. Left: the octopus body movements are recorded in a first track. Middle: several of the legs are recorded next, while playing back the first track for reference. Right: after several more layers are added (including a fish character), the final animation.

creates a physical analog to inverse kinematics. Yet, articulated puppets can also be cumbersome to control — especially when their segments are relatively small. Moreover, the large occlusions between pinned segments can make it difficult to accurately track the segments and compute their depth ordering.

In our video puppetry system there is no need for the segments to be pinned together. While users can work with physically connected segments, they can also separate the segments and work with them independently when more control is needed (Figure 7).

Articulated Puppet Builder: We have extended our puppet builder module (Section 4) to support articulated puppets. After capturing each segment of an articulated puppet using the standard builder, the user specifies the joint positions and depth ordering between pairs of segments using a drag-and-drop interface. As shown in Figure 7 the user drags from the lower segment to the upper segment in the depth ordering. The user then picks a *root* segment which controls the gross translation for the entire puppet.

Articulation Constraints: The tracker independently tracks each segment of the articulated puppet. During the interpretation process, we traverse the articulation hierarchy from the root down to the leaves, translating each segment such that its joint location agrees with that of its parent. Thus, for each segment other than the root, only its tracked orientation affects the rendered output. Whenever a segment disappears from the tracker (typically because it leaves the working area) it is rendered in a default orientation.

Depth Ordering: One drawback of working with disconnected puppets is that rendered segments may overlap while their physical counterparts do not. In such cases, the system must determine a plausible depth ordering. We solve the problem by choosing the topological sort of the occlusion graph (Section 5.3) that favors showing puppets of larger relative scale.

Although our approach only works for tree-structured joint hierarchies, we have found it to be sufficient for many types of puppets. Additionally, our approach gives users the flexibility to work with puppets that are either physically connected or disconnected. In practice we have found that disconnected puppets are often easier to manipulate, as they can be arranged in the workspace so that they do not physically occlude or interact with one another.

6.2 Multiple animators or tracks

The control of complex puppets or multiple puppets can be distributed among several puppeteers, each controlling separate props. The puppeteers might share a common work space, or our system can combine the tracking data from multiple puppeteers working at several identical stations like the one shown in Figure 1, collaborating either in the same room or remotely. Each participant may



Figure 9: Controls and Effects. From left to right: zooming into the scene using "hand" puppets; faux shadows as well as 2¹/₂-D effects such as the foreshortening and occlusion of the boy; rain falling from the cloud as an animation cycle; sprites for each snowflake.

control one or more characters or segments, and each participant sees the results of all characters merged into a single scene.

Just as it is possible to control different characters or segments from multiple locations, our system also allows a puppeteer to control different objects at different times, by recording the motion of one object while another object's motion is playing back. During playback, a new stream of puppet actions may be layered over the "playback" stream. The new layer may either add motions for new characters or refine the motions of articulated characters (for example, where the gross motion of the root node has already been recorded and now the limbs are articulated in the new stream – Figure 8). Motion for articulated characters is refined by overwriting the transformations for all segments tracked in the current layer.

Different streams may be recorded by different people or by the same animator. Each such stream may be thought of as a "track" as in multitrack musical recording, and this framework exposes many similar benefits – the expressive power and immediacy of performative control, the ability to control many "instruments" simultaneously, and the opportunity to perform multiple "takes" of each instrument independently until the artist is satisfied with the result.

6.3 Special Effects

Our system also supports a number of "special effects" that can be triggered by special puppets (rather than by menus or commands, in order to make them easier for novices to use and to remember). Special effects are plug-in modules, each of which can be linked to the appearance of a specific physical "trigger" puppet using a GUI similar to the articulated puppet builder. We envision that special effect modules could be downloaded by users, and even modified by advanced users (using a high-level scripting language). Several of these effects, while simple in a computer-animation system, would be difficult to achieve using stop-motion based cutout animation or pure video animation processes. For example, causing a user-drawn character to automatically walk is simple with our system, though difficult with others.

Scene Changes: To trigger a scene change, the animator simply shows a picture of the desired background to the camera. Since these scenes fill the entire screen, we prefer to scan hand-drawn imagery on a flatbed scanner rather than grab it from the camera. We imagine that a collection of such scenes would appear in a book and that the animator could leaf through the book to select a pre-existing scene. The same mechanism can be used to change the "state" of a character. For example, the burnt knight shown in the lower middle frame of Figure 11 was triggered by placing an image of the character in that state. To control the pan and zoom of the camera within the scene, the animator can hold a pair of "cinematographer's hands" puppets, showing the camera where to crop the scene as illustrated in Figure 9.

 $2^{1/2}$ -D: Some scenes may be pre-designated as $2^{1/2}$ -D environments, thereby invoking several effects. First, characters shrink under uniform scale as they approach the horizon – specified by both a horizon height in the image as well as a parameter indicating the linear term for the scale operation.

Second, mattes (as shown at right) may accompany such scenes indicating various popup elements (e.g., the house and tree in Figure 9). Faux depths for these popup elements, as well as for moving characters in the scene are inferred from vertical height,



under the assumption that all such sprites are rooted to a common ground plane. Rendering these layers with the appropriate depth ordering gives the effect that characters in the scene can go either in front of or behind various popups. While these assumptions are generally brittle, they work for a variety of scenarios, producing "plausible" animations appropriate to the cutout style.

Shadows: Because we have mattes for all the characters, it is easy to generate plausible shadows using the well-known strategy of darkening the background layer by a blurred, warped copy of the character matte onto the background image, seen for example in Figure 9. The direction of the warp is controlled by a "sun" puppet.

Animation Cycles: Several other effects can be implemented using animation cycles. The image of the boy in Figure 9 is actually a series of frames representing a walk cycle. As the boy moves in the image, these frames are advanced at a rate proportional to the rate of movement. Furthermore, user-chosen puppets will automatically flip to face in the direction of motion. Another example of an animation cycles is the rain shown in Figure 9. In this example, the rain falls continuously from the cloud towards the bottom of the screen. This is accomplished by a single (vertically periodic) image that is simply translated per frame. The snowflakes shown in Figure 9 are sprites that fall down the screen with semi-periodic rotation and horizontal drift. As a final example, puppet segments may be identified as "eyes" that will rotate to follow either the nearest character or a specific eye target puppet, as shown in Figure 8.

Automatic Walk Cycles: One of the benefits of our system is that it allows the user to focus on primary character motion, while the computer provides secondary animation. As an example, we automatically provide walk cycles for articulated characters. The user begins by identifying segments that are arms or legs, and setting a resting angle for each. As the character moves, the system applies sinusoidal motions to the limb angles to match feet with the ground. Neighboring arms and legs are offset in phase by 180 degrees, and modifying the extreme angles may cause a character to switch between e.g. a walk, a power-walk, and a run. Examples of automatic walk cycles are shown in Figure 10.

Audio: We detect amplitude peaks in an audio stream, automatically opening a character's mouth as the performer is speaking. An example of this effect can be seen in the accompanying video.

7 Evaluation and Discussion

This section describes our experiences using this system, discusses our observations about what works well in this kind of interface, and offers areas for future investigation. We describe an initial study composed of three types of evaluation. First, a low-level study tests dexterity with the mouse against our puppet system.



Figure 10: Automatic walk cycles. Left: a GUI is used to assign limb types, and set resting and maximal angles. Right: a biped and quadruped automatically walking.

Second, in a mid-level study, users evaluate their experiences in telling a specific story three times: with our system, using a mouse, and videotaping traditional cutout puppetry. Third, at a high level, we report the experiences of people using our system for openended storytelling, both in our lab and in an exhibit at Maker Faire, a public art and technology fair with tens of thousands of attendees. Participants in the first two studies spent a total of about 40 minutes: 5 minutes to explain the system, 25 for the low-level study, and 10 minutes for the mid-level study.

Our current setups mostly use table-top working areas, often around 30×20 cm., as shown in Figure 1, left. Puppets range from 5 to 20 cm. By zooming the camera, we can scale to nearly arbitrary working volumes (limited, of course, by the resolution of the camera and the size and detail of the puppets).

7.1 User Study: Low-level Dexterity

At a low level, we compare the accuracy of our system to a mousebased interface. Users use either puppets or a mouse to follow targets undergoing three types of motion along random paths: one target undergoing translation only, one target undergoing translation and rotation, and two targets undergoing translation and rotation.

The targets are glyphs of the letters F and G, roughly 5 cm tall. As shown at right, the user sees a composite rendering including both the moving target(s), outlined in red, and the current handanimated pose. The user is asked to keep the figure(s) aligned to the target(s) as closely as possible.



To avoid favoring one input method, we generate synthetic motion paths. These are cubic B-splines with random control points in (x, y, θ) space. In the case of two targets, we use identical motion paths constrained to the left and right halves of the screen. Each user follows the same motion path with both mouse and puppets.

For the mouse interface, translation of the mouse controls position, and the scroll wheel controls rotation. Two mice are used when following two targets. Users may practice with each input source as many times as desired, before following the motion path.

For the puppet interface we have found in general that some users prefer to manipulate puppets attached to small sticks, while others prefer to move the paper directly – examples of both styles can be seen in Figures 6 and 8 as well as the accompanying video. While the system works equally well in either case, for this study all users used the direct method.

Results for 8 users are shown in Table 1. The users were graduate students with no previous experience with our system but extensive experience using a mouse. Each user runs two trials for each task, with both the puppet and the mouse (alternating, with the same number of users starting with each condition). The mouse and puppet errors displayed statistically insignificant (p > 0.05) differences under Student's t-test in all conditions, suggesting that neither the mouse nor puppet interface is clearly better. The error values have high variance, which we attribute to variation in user motor skills.

	T only	T+R (1 target)		T+R (2 targets)	
	T (mm)	T (mm)	R (°)	T (mm)	R (°)
Mouse	0.64 (0.34)	· · ·	5.23 (2.48)	. ,	10.22 (3.97)
Puppet	0.83 (0.46)	1.07 (0.38)	3.79 (1.31)	1.87 (0.52)	11.10 (3.68)

Table 1: Comparison of mouse and puppet input accuracy. Values reported are means, with standard deviations in parentheses. Translational error (T) is computed per trial as mean center of mass distance. Rotational error (R) is mean absolute difference in angle. Puppets and mice are calibrated to use equal distances per on-screen translation.

7.2 User Study: Mid-Level Story Task

In this study, users create a story. The user is shown the desired outcome for the story: a lumberjack cuts down a tree, as seen in the video. The user is provided with pre-made characters, and told how to make the story with three different systems: the video puppetry system described in this paper, a mouse interface as described in the previous section, and also a "raw video" system that simply records a traditional paper-cutout puppetry session. With each interface, the user records the story as many times as desired, until satisfied. He or she then evaluates each interface.

The three interfaces are set up as follows. In both our system and the mouse system, the lumberjack and tree are articulated characters. In our system, the animation is done in one layer, but in the mouse system, the user selects one limb at a time, and uses translation and the scroll wheel to orient that limb. Because two limbs move and translate simultaneously in the video, we use one mouse and show the user how to record in two layers.

We conducted this study with the same 8 users as before. All ranked our puppet system as the top preference, and all except one ranked the mouse interface second. When asked to explain their preference for our puppet system, users gave reasons such as:

- Puppets provide more intuitive and more precise control, especially over rotation.
- It is possible to use multiple fingers to control puppets.
- Puppets permit quickly switching the active character.
- The puppets' motion is more fluid and requires less care, resulting in more "cartoon-looking" motion.
- It is useful to physically separate the puppets, to avoid having them bump into each other.

In contrast, users disliked the raw video system because puppets would bump into each other. Moreover, most users strongly disliked having fingers visible in the video. Users typically ranked the mouse as their second choice, as they felt that control was less intuitive than with physical puppets.

We also asked an independent set of 10 users to rate the resulting animations. Each user evaluated 12 of the 24 videos, scoring his or her favorite as 10, the least favorite as 1, and assigning (not necessarily unique) scores to the other animations. The results mirrored the self-selected ratings, with average scores of 6.7, 5.9, and 3.8 for puppets, mouse, and raw video. These results suggest that our system's appeal lies not only in its ease of use, but also in the aesthetic quality of the animations.

7.3 User Experiences: High-Level Storytelling

Much of the animation shown on the accompanying video was performed by the authors of this paper. However, roughly 100 users have tried our system at Maker Faire, and roughly 30 people have tried our system at other events. Animations from two of our test users appear on the video and in Figure 11, one by a professional puppeteer ("The Balloon") and the other by an 11-year-old boy



Figure 11: Several short stories created using our puppetry system.

("Knight meets Dragon"). Here we make some general observations from these experiences:

- As advertised, the system is easy to learn and use. All users were able to control the onscreen puppets with minimal instruction because the interface is so transparent. We believe that a significant factor contributing to this ease of use is the fact that people manipulate paper puppets that largely resemble their onscreen counterparts, leading to a natural interface.
- Our system preserves many of the performative qualities of puppeteering — the immediate feedback of seeing one's own performance and the ability to tune and respond in real time and the quality of the resulting animation has the feeling of "straight ahead" rather than "pose-to-pose" animation [Thomas and Johnston 1981].
- The time to create a typical story is dominated by the time to draw the characters. The largely automatic matting process takes a few seconds, while adding controls for articulated characters takes an additional few tens of seconds. While animation is performed in real time, users often try a few "takes" before they are happy with the result. Furthermore, for layered animation, each layer adds another multiplier on the performance time. Nevertheless, performance generally takes less time than drawing, and *far* less time than stop-motion, key-frame or other offline animation paradigms.

7.4 Limitations

Unfortunately, our system is not capable of handling every puppet or action a user may wish to animate. For example, in order for a puppet to contain sufficient detail for robust tracking, it typically needs to occupy a significant portion of the video frame's area, so very small puppets do not always track well.

Our system currently has almost 100 puppets in its database, and some kids seem to delight in putting many of them in front of the camera at once. With more than 5 or 6 puppets on camera simultaneously, it becomes difficult to manipulate puppets without creating near-total occlusions that interfere with tracking. Although performance remains real-time, the recognition portion of the pipeline slows down with many puppets.

Additionally, puppets cannot be moved too quickly, as the KLT algorithm assumes that the optical flow between frames is well approximated by small displacements without rotation. Although we offer alternatives via interpretation, such as per puppet render scaling, a better tracking system could enable a more natural user experience.

These limitations arise from the fact that our system's hardware can only process so many pixels per second. Increasing the volume of pixels processed could alleviate both of these issues. More pixels per frame would allow for smaller puppets or a larger workspace, because smaller scale details would become trackable. More frames per second would reduce inter-frame displacements, improving the correctness of KLT's assumption. Thus, as technology improves, these limitations will become less significant to the user.

7.5 Conclusion and Future Work

We have demonstrated a novel interface for creating cutout style animations by manipulating paper puppets. The system relies on easy to use components for creating puppets, combined with a natural interface for controlling them. To implement these controls, we introduce a novel use of known computer vision algorithms that is designed to handle the particular real-time requirements of our application in spite of occlusions common in this setup. It is arguable that this kind of interface might instead be constructed via a multi-touch display. However, for the foreseeable future, cameras are likely to be substantially cheaper and more ubiquitous than such displays. Moreover, we believe that the tactile, proprioceptive experience of physically manipulating these puppets contributes to both the ease of use and the quality of the resulting animation.

We show that this kind of interface is appropriate for real-time creation of the kinds of cutout stop-motion animations that people already make. However, we believe that it would be useful in other applications as well, for example in creating low-budget animatics during the planning stages for high-budget productions, or for creating productions in styles with deliberately low animation fidelity (such as JibJab), which are currently created in key-frame oriented systems such as Flash.

This system suggests a number of areas for future work, both to further reduce the learning curve for users and to extend the range of animations that can be created:

Automated articulation inference: We believe it would be possible to more automatically infer the segmentation and articulation of a character by placing the segments on the table in several poses. This would address the general goal of providing a single camerabased interface for animation.

Audio: Our system can respond to audio events and, for example, trigger a mouth animation cycle. However, treating sound as a firstclass component in the system and providing tools to layer audio effects over the animation would provide for a more complete authoring system. Furthermore, source separation techniques could be applied to identify multiple performers controlling different objects with their voices simultaneously.

Editing: Currently the animator can only "edit" an animation by re-recording it. In order to author longer animations a facility for tracking a sequence of scenes and the ability to edit these scenes or animation layers within them will go a long way towards expanding this system as a tool for authoring longer stories.

Acknowledgments

We would like to thank the many amateur (and professional) puppeteers of all ages who have tried the system and offered suggestions, in particular Thomas (age 11), Lydia Greer, and Helene van Rossum (hvanrossum.com). We thank the paper reviewers and the Tiggraph gang for their helpful suggestions. This work was supported in part by Adobe Systems and the NSF grants IIS-0511965, CCF-0347427, CCF-0643552 and IIS-0812562.

References

- AGARWALA, A., HERTZMANN, A., SALESIN, D. H., AND SEITZ, S. M. 2004. Keyframe-based Tracking for Rotoscoping and Animation. ACM Trans. Graphics (Proc. SIGGRAPH), Vol. 23, No. 3, 584–591.
- AGARWALA, A. 2002. SnakeToonz: A Semi-Automatic Approach to Creating Cel Animation from Video. In Proc. NPAR, 139–146.

- ALVARADO, C., AND DAVIS, R. 2001. Resolving Ambiguities to Create a Natural Sketch Based Interface. In *Proc. IJCAI*, 1365– 1371.
- BAECKER, R. 1969. Picture-Driven Animation. In Proc. Spring Joint Computer Conference, 273–288.
- BOUSSEAU, A., NEYRET, F., THOLLOT, J., AND SALESIN, D. 2007. Video Watercolorization using Bidirectional Texture Advection. ACM Trans. Graphics (Proc. SIGGRAPH), Vol. 26, No. 3, 104:1–104:7.
- COLLOMOSSE, J., AND HALL, P. 2005. Video Paintbox: The Fine Art of Video Painting. *Computers & Graphics, Vol. 29, No. 6*, 862–870.
- COLLOMOSSE, J., AND HALL, P. 2006. Video Motion Analysis for the Synthesis of Dynamic Cues and Futurist Art. *Graphical Models, Vol. 68, No. 5-6*, 402–414.
- DAVIS, J., AGRAWALA, M., CHUANG, E., POPOVIĆ, Z., AND SALESIN, D. 2003. A Sketching Interface for Articulated Figure Animation. In *Proc. SCA*, 320–328.
- DAVIS, R. C., COLWELL, B., AND LANDAY, J. A. 2008. K-Sketch: A "Kinetic" Sketch Pad for Novice Animators. In *Proc. SIGCHI*, 413–422.
- DONTCHEVA, M., YNGVE, G., AND POPOVIĆ, Z. 2003. Layered acting for character animation. *ACM Trans. Graphics (Proc. SIGGRAPH), Vol. 22, No. 3,* 409–416.
- FIALA, M. 2005. ARTag, a Fiducial Marker System Using Digital Techniques. Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, Vol. 2, 590– 596.
- GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. In *Proc. SIGGRAPH*, 171–180.
- HORN, B. K. P. 1986. Closed-Form Solution of Absolute Orientation using Unit Quaternions. *Journal of the Optical Society of America, Vol. 4*, 629–642.
- IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-Rigid-as-Possible Shape Manipulation. ACM Trans. Graphics (Proc. SIGGRAPH), Vol. 24, No. 3, 1134–1141.
- ISHII, H., AND ULLMER, B. 1997. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In *Proc. SIGCHI*, 234–241.
- KIM, J., SEITZ, S., AND AGRAWALA, M. 2004. Video-based Document Tracking: Unifying Your Physical and Electronic Desktops. In *Proc. UIST*, 99–107.
- KNEP, B., HAYES, C., SAYRE, R., AND WILLIAMS, T. 1995. Dinosaur Input Device. In *Proc. SIGCHI*, 304–309.
- LAVIOLA JR, J., AND ZELEZNIK, R. 2004. MathPad 2: A System for the Creation and Exploration of Mathematical Sketches. *ACM Trans. Graphics (Proc. SIGGRAPH), Vol. 23, No. 3*, 432–440.
- LEE, G., KIM, G., AND BILLINGHURST, M. 2005. Immersive authoring: What You eXperience Is What You Get (WYXIWYG). *Communications of the ACM, Vol. 48, No.* 7, 76–81.
- LOWE, D. G. 1999. Object Recognition from Local Scale-Invariant Features. In *International Conference on Computer Vision*, 1150–1157.
- MILGRAM, P., AND KISHINO, F. 1994. A Taxonomy of Mixed Reality Visual Displays. *IEICE Transactions on Information and Systems, Vol. 77, No. 12*, 1321–1329.

- MOSCOVICH, T., AND HUGHES, J. 2001. Animation Sketching: An Approach to Accessible Animation. Tech. Rep. CS04-03, Brown University CS Department.
- OORE, S., TERZOPOULOS, D., AND HINTON, G. 2002. A Desktop Input Device and Interface for Interactive 3D Character Animation. *Graphics Interface, Vol.* 2, 133–140.
- POPOVIĆ, J., SEITZ, S., AND ERDMANN, M. 2003. Motion Sketching for Control of Rigid-Body Simulations. ACM Trans. Graphics, Vol. 22, No. 4, 1034–1054.
- REKIMOTO, J., AND AYATSUKA, Y. 2000. CyberCode: designing augmented reality environments with visual tags. *Proceedings* of DARE 2000 on Designing augmented reality environments, 1–10.
- RUS, D., AND DESANTIS, P. 1997. The Self-Organizing Desk. Tech. Rep. PCS-TR97-305, Dartmouth University CS Department.
- STURMAN, D. 1998. Computer Puppetry. IEEE Computer Graphics and Applications, Vol. 18, No. 1, 38–45.
- THOMAS, F., AND JOHNSTON, O. 1981. *Disney Animation: The Illusion of Life*. Walt Disney Productions, New York.
- THORNE, M., BURKE, D., AND VAN DE PANNE, M. 2004. Motion Doodles: An Interface for Sketching Character Motion. *ACM Trans. Graphics (Proc. SIGGRAPH), Vol. 23, No. 3*, 424–431.
- TOMASI, C., AND KANADE, T. 1991. Detection and Tracking of Point Features. Tech. Rep. CMU-CS-91-132, Carnegie Mellon University, Pittsburgh, PA, USA.
- TRUCCO, E., AND PLAKAS, K. 2006. Video Tracking: A Concise Survey. *IEEE Journal of Oceanic Engineering*, Vol. 31, No. 2, 520–529.
- WANG, J., XU, Y., SHUM, H., AND COHEN, M. 2004. Video Tooning. ACM Trans. Graphics (Proc. SIGGRAPH), Vol. 23, No. 3, 574–583.
- WANG, J., BHAT, P., COLBURN, R., AGRAWALA, M., AND CO-HEN, M. 2005. Interactive video cutout. ACM Trans. Graphics (Proc. SIGGGRAPH), Vol. 24, No. 3, 585–594.
- WANG, J., DRUCKER, S., AGRAWALA, M., AND COHEN, M. 2006. The Cartoon Animation Filter. ACM Trans. Graphics (Proc. SIGGRAPH), Vol. 25, No. 3, 1169–1173.
- WELLNER, P. 1993. Interacting with Paper on the DigitalDesk. Communications of the ACM, Vol. 36, No. 7 (July), 86–97.
- WILSON, A. 2005. PlayAnywhere: a compact interactive tabletop projection-vision system. In *Proc. UIST*, 83–92.
- WINNEMÖLLER, H., OLSEN, S., AND GOOCH, B. 2006. Realtime Video Abstraction. ACM Trans. Graphics (Proc. SIG-GRAPH), Vol. 25, No. 3, 1221–1226.
- WISNIEWSKI, D., AND WISNIEWSKI, D. 1996. Worlds of Shadow: Teaching with Shadow Puppetry. Teacher Ideas Press.