Puppet Animation with Textured Billboards

Anthony Chansavang
Coupang
South Korea
anthony.chansavang@gmail.com

Henry Johan
Fraunhofer Singapore
Nanyang Technological University
Singapore
henryjohan@ntu.edu.sg

Hock Soon Seah
School of Computer Science and Engineering
Nanyang Technological University
Singapore
ashsseah@ntu.edu.sg

Rall Hans-Martin
School of Art, Design and Media
Nanyang Technological University
Singapore
Rall@ntu.edu.sg

***Abstract* — We present a system for key frame flat articulated puppet animation. We show the main user interface components and features specifically crafted for this type of animation. In particular, the puppet rigging system is designed for intuitive animation of flat articulated characters without explicitly defining a skeleton hierarchy. Useful features, including automatic in-betweening, inverse kinematics control and animation retargeting, make it easy to create a cutout animation. The system is built upon a game engine, effectively taking advantage of its 3D rendering engine, scene management system, graphical user interface, and allowing real time interaction with animations in final render quality.**

Keywords—Articulated Puppet Animation, Auto In-betweening, Inverse Kinematics, Animation Retargeting

I. Introduction

Shadow puppetry is an ancient performative art that usually uses flat articulated figures as puppets. The puppet master uses his hands to control rods or strings connected to the puppet. Originated from Asia, shadow puppetry is popular in many cultures, including Indonesia and China. In traditional animation [1], cutout refers to a style of animation in which characters, props and backgrounds are flat. The viewer is facing those elements and the animation is usually restricted to one plane. This style is traditionally inspired by shadow puppetry, which differs from the cutout animation that it is a performative art focused on articulated characters.

We developed a system for users to intuitively create cutout animations with flat articulated characters. 2D animation software such as Animate CC and Toon Boom and 3D animation software such as Maya and 3D Studio Max are powerful tools for creating animation. As they cover the entire production pipeline from modeling to final animation, their UIs need to handle complex functions. It is sometimes not trivial to achieve a particular style of rendering, and it requires many hours of learning to master these systems.

By limiting to the cutout style we provide a simpler interface, intuitive tools and an easy workflow for this particular style of animation. The user can focus on artistic aspects. The system does not provide drawing/modeling functionalities, and is instead dedicated to animation. By building upon a game engine, we ensure real time interaction and facilitate the implementation of our features.

We developed our system alongside an animation production to get quick feedbacks for improvements. The animation is inspired by the Indonesian shadow puppetry called Wayang Kulit, from which we derived the name of our system: WaKu Engine (shortened WaKu). This paper discusses the technical design and implementation of our system, compared to an earlier paper [2] describing the aesthetic choices based on the affordances of animation software.

We present our proposed workflow and set of tools. Section II describes the related work in the field. Section III analyzes the different components of our system. In Section IV, we provide some results of the application of our system and conclude with topics for future research in Section V.

II. Related Work

A. Puppetry and Cut Out Animation

Ghani [3] describes the design for a game character inspired by Wayang Kulit. He uses a video game as his medium; we instead use animation to convey an original story in Wayang Kulit style. We also opted to render the puppets in a cutout style for a closer match to the real puppets whereas Ghani modeled his character in 3D.

Talib et al. [4] propose an interactive virtual shadow puppet play using texture mapping and blending techniques in Adobe Flash (now, Animate CC). This is done in a 2D environment while our setting is in a 3D environment, which allows us to have a 2.5D system.

More recently, Huang et. al [5] propose a method to transform scanned data of 3D human bodies to 2D puppet figures for shadow puppets. Our method gives artists the flexibility to design and create their own puppets.

Barnes et al. [6] propose a system for cutout animation, which focuses on performative animation in real time, where a user manipulates physical materials made of paper. A video-based interface captures the motions of the puppets and renders them on a new background with the puppeteer's hands removed. This method presents an alternative to traditional puppetry. However, the focus on real time animation and the computer vision techniques involved make it difficult to animate complex scenes with precise motions.

Reeth [7] uses 2.5D modeling techniques for the creation of computer animation sequences which have the look and feel of traditional hand-drawn animation productions. 2D characters and scenery can freely be positioned in 3D space. Characters are designed using Bezier curves. Since this system focuses on a hand-drawn animation look and feel, the techniques used for designing and animating characters are not suited to cutout animation.

B. Animation Systems and Game Engine

Commercial animation packages such as Animate CC, Toon Boom, Maya or 3D Studio Max pioneered the area of key frame animation interfaces and are the main inspiration for the design of our user interface. Several helpful features common to key frame animation systems such as onion skins (i.e. ghost images of previous key frames), or many operations on the timeline are adopted in our system. However, since they are designed as generic animation systems and often also provide modeling capabilities, their interfaces are very complex and difficult to learn. Moreover they lack the dedicated tools useful for cutout animation.

Game enthusiasts use game engines to create animations called Machinima. These animations are however limited to pre-designed contents and therefore do not offer enough creative freedom. We used the GF Engine [8] for the development of our system, allowing us to focus our efforts on features specific to flat articulated puppet animation. The GF Engine can easily be extended to implement various 3D interactive applications. It provides 3D visualization and scene management using the open source 3D graphics engine Ogre, user interface using the Qt library and physics simulation with NVIDIA PhysX.

III. WaKu Animation

There are many similarities between shadow puppetry and traditional cutout animation. In shadow puppetry, puppets are articulated links of flat rigid parts. Puppeteers manipulate puppets via rods attached to various parts of the puppets. In traditional cutout animation, shapes are cut out from different materials. These shapes are then moved in small steps for each frame to be captured by a camera. We can generalize our system to be a key frame cutout animation system in which the puppets are articulated characters.

Our approach is based on modeling the scene in 3D space, simulating a theatre stage. The organization is similar to a multiplane camera with textured billboards representing the backgrounds and characters, see Figure 1. The 3D space is organized as a tree structure with each layer being attached to a node that controls its position and orientation with respect to origin of the 3D space. The depth order of the layers is automatically solved and post-processing effects naturally fit into the rendering pipeline of shaders.

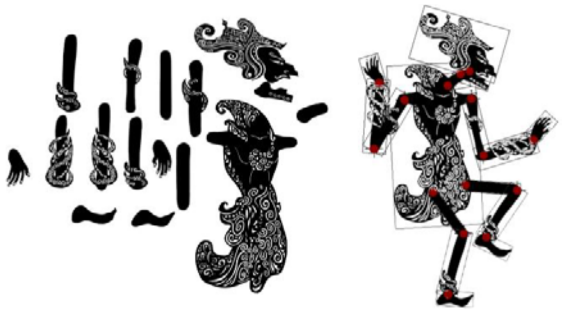
Figure 1: Puppet rigging by loading the textures of the puppets' parts into the system and applying them to billboards. Then, the user assembles the puppet. Red circles indicate the joints' positions.

A. Articulated Character Animation

Our system handles forward and inverse kinematics control. In this section we refer to a rendering frame as the rendering output of the 3D engine's graphics pipeline, and therefore not a frame from the produced animation.

Given a pivot joint and a pressed part of the puppet, forward kinematics computes which body parts should be rotated, see Figure 2.

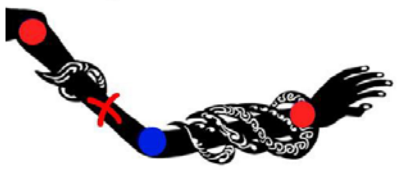
Figure 2: The user selects a pivot (the blue joint) and presses the body part indicated by the red cross. The forearm and the hand will be rotated since they are not connected to the pressed part other than through the pivot joint. The rest of the body parts will remain fixed.

For animators, inverse kinematics (IK) is a more intuitive way of animating an articulated character and is similar to controlling a puppet. The joints on our puppet model only have 1 DOF, i.e. rotation and in most cases the number of joints involved is very limited. Hence, we do not require advanced IK methods, such as those in [9][10]. Instead, we use a naive optimization strategy, with an iterative solver that allows us to control the amount of computation done per rendering frame. Based on our observations on the way most people manipulate puppets, we define constraints and priorities on the joints, depending on the dragging position, making the control more intuitive and allowing us to do without a skeleton hierarchy. The user enables several joints to be allowed to rotate in the IK. We define the dragged part as the part of the puppet's body being dragged by the user. A dominant joint is defined as follows. Let J be the subset of joints connected to the dragged part (green and yellow joints in Figure 3, left) and p_j be the position of a joint j. We define T_g and D_g as being the target position and the dragging position, respectively, in the global coordinate system. The dominant joint j is defined as:

$$j \in J, \max_j \left\| p_j - D_g \right\|_2 \qquad (1)$$

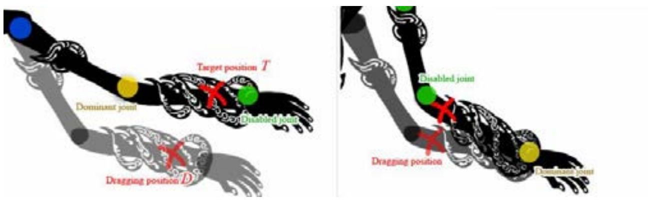

Figure 3: Constraints and priorities defined in regards to the dragging position. The user enabled the three joints of the arm. The green joints are automatically disabled according to the dragging position (the angle between their two linked parts will remain constant). The blue joint can rotate but is given a lower priority than the dominant joint.

This is illustrated in Figure 3. Next, we disable the joints connected to the dominant joint through the dragged part, i.e. these joints will not be able to rotate although their positions can change. We iterate through the remaining user-enabled joints, ordered from near to far by their distances to the dominant joint, and we rotate each joint to minimize the distance between T and D. The rotation angle θ_j from D_j to T_j with respect to joint j is defined as:

$$\theta_j = \cos^{-1}\left(\frac{D_j \cdot T_j}{\|D_j\|_2 \|T_j\|_2}\right) \qquad (2)$$

By reiterating this process, D converges towards T. We spread the computation across multiple rendering frames by limiting the number of iterations to N per rendering frame and the algorithm stops when no rotation of the enabled joints can decrease the distance between D and T or if the user stops the dragging. This results in a smooth motion of the puppet. In practice, setting N to 10 gives a comfortable feel to the interaction. In most cases animators need only to enable a few joints, e.g. 3 joints (shoulder, elbow and wrist) for the arm. The computation is therefore not intensive.

B. Automatic In-betweening

We use the information in our rigged puppet model to calculate its interpolation. The puppet is a graph with the body parts being the vertices and the joints, the edges (the leaves of the graph are always vertices, i.e. body parts). We assume that the puppet is a connected graph, i.e. there is a path from any vertex to any other vertex in the graph.

Let k_1 and k_2 denote the key frames between which we are interpolating, and k a frame between k_1 and k_2. Each joint j is linked to two body parts α_j and β_j. At each key frame k_n we know the positions g_{j,k_n} of each joint j in the global coordinate system, and the relative positions, p_{α_j,k_n} and p_{β_j,k_n}, of α_j and β_j with respect to j (here we consider the center of the billboard as the position for the body parts). We define $q_{\alpha_j,k}$ and $q_{\beta_j,k}$ as the quaternions representing the rotation for parts α_j and β_j, respectively, from k_1 to k with respect to j (determined by Slerp [11]). A fixed joint is defined as a joint for which either: $g_{j,k_1} = g_{j,k_2}$ or $p_{\alpha_j,k_1} = p_{\alpha_j,k_2}$ and $p_{\beta_j,k_1} = p_{\beta_j,k_2}$. Our goal is to find $g_{j,k}$, $p_{\alpha_j,k}$ and $p_{\beta_j,k}$ for all joints j at frame k.

(a) We start by looking for a fixed joint j' to linearly interpolate $g_{j,k}$ if necessary and set $p_{\alpha_j,k}$ and $p_{\beta_j,k}$ by Slerp[11]. If no fixed joint is found, we start by linearly interpolating the body part of the puppet ω for which $\left\| g_{\omega,k_1} - g_{\omega,k_2} \right\|_2$ is the smallest. This first step is to ensure that the puppet's position is consistent across all this sequence of frames.

(b) Let us assume that $g_{\alpha_j,k}$ has been determined. We can deduce the position of the joint $g_{j,k}$ and therefore the position of the second linked part in the global coordinate system, $p_{\beta_j,k}$, using the following equations:

$$g_{j,k} = g_{\alpha_j,k} - q_{\alpha_j,k} \times p_{\alpha_j,k_1} \qquad (3)$$
$$p_{\beta_j,k} = q_{\beta_j,k} \times p_{\beta_j,k_1} \qquad (4)$$
$$g_{\beta_j,k} = g_{j,k} + p_{\beta_j,k} \qquad (5)$$

Since the skeleton is a connected graph, by knowing the position of one part as described in (a), we can solve the positions of all parts by iteration of the equations in (b). This algorithm effectively preserves the structure of the puppet's skeleton, i.e. distances between the joints.

C. Animation Retargeting

Some common animation sequences like walk cycles or jumps differ only by a few variations for morphologically similar characters. Animators can therefore save a lot of time by reusing generic animations. A generic animation file will contain a series of key frames, and for each key frame, the rigged puppet model information. Figure 4 shows an example application of our retargeting method.

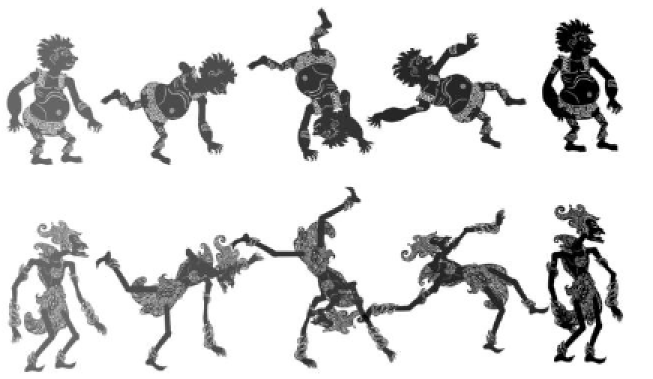

Figure 4: Retargeting of a cartwheel animation to a topologically similar character. The generic animation was created using the character on top. No editing was done to the second character at the bottom.

For our purposes, in most cases we want to reproduce the "feel" of an animation rather than the exact position of the character. For this reason, we retarget an animation according to the relative orientation of parts with respect to their joints. However we still need to consider the differences of positions of the parts from one key frame to another in order to ensure consistency in the motion, otherwise issues as shown in Figure 5 will arise. Hence, we define two joints on two different characters as similar if they link the two same body parts (one joint can only link two body parts together).

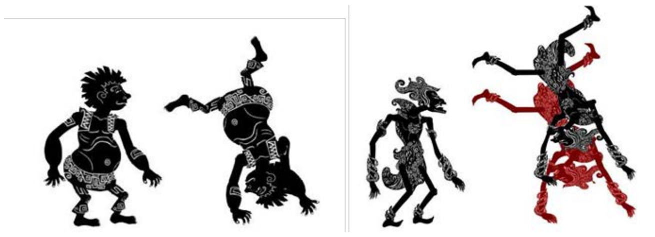
Figure 5: This is what happens if we only consider the orientations of the joints. The original animation is illustrated with the character on the left. When retargeting to the second character, the posture of the puppet is correct but the global position from key frame 1 to key frame 2 is inconsistent (appropriate position highlighted in red).

We start by manually calibrating the character's pose to be similar to the first key frame of the animation. The rest of the algorithm is similar to the interpolation in Section IIIB with a few modifications.

(a) For each key frame of the generic animation, we start by looking for a fixed joint to set its global position and the positions and orientations of its linked parts. Similarly to the interpolation, if no fixed joint is found, we start by the part of the puppet with the smallest distance between its positions in the current and previous key frames. Again, this first step is to ensure that the global position of the puppet will be consistent across the animation.

(b) We use the positions of the already set parts to determine the position of other joints they are linked to and deduce the positions and orientations of all the parts linked to those joints. We iterate process (b) until all the joints are solved.

IV. Results

Our system has been used to create an animation inspired by the Indonesian shadow puppet art of Wayang Kulit. The complete scenes, some of which are shown at the top of this paper, have been animated and rendered with high resolution textures. Feedbacks from our animators show that WaKu is a fast to learn and intuitive system capable of rendering high quality cutout animations. As the user interface is much clearer and simpler, a professional animator can familiarize with our system within a few hours.

V. Conclusion and Future Work

The fast growing development of computer aided animation has led to more and more powerful generic systems. While most usages only require a subset of these systems capabilities, users have to cope with increasingly complex interfaces, when they should be focusing on artistic aspects. We have presented a system dedicated to cutout animation of articulated puppet. Our system improves the workflow and speed of production of such animations due to the dedicated tools. We found that using a game engine to develop our system presents many benefits from a developer's point of view. We took advantage of the entire rendering pipeline, scene management and interactivity capacities. In the future we could even make use of the integrated physics engine for procedural animations, e.g. collisions, or a more performative animation system by simulating in real time the physics of the puppets. We hope to extend our system to make it even more intuitive with the use of novel interfaces, e.g. multi-touch screen. Finally we consider different possible spin-off applications like cutout animation for games, or interactive virtual puppetry using physical simulations.

Acknowledgment

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IDM Futures Funding Initiative as well as by the Ministry of Education Academic Research Fund.

References

[1] J. Krasner, "Motion Graphic Design and Fine Art Animation: Principles and Practice". Focal Press, 2004.

[2] D.K. Jernigan, A. Chansavang, H. Martin-Rall, H.S. Seah, D. Lim, H. Johan, "Aesthetic Affordances: Computer Animation and Wayang Kulit Puppet Theatre." Animation Practice, Process & Production, 3(1/2):195-217, Dec 2013.

[3] D.B.A. Ghani, "Seri Rama: Converting a Shadow Play Puppet to Street Fighter". Computer Graphics and Applications, IEEE 32(1): 8-11. 2012.

[4] A.Z. Talib, M.A. Osman, K.L. Tan, S. Piman, "Design and Development of an Interactive Virtual Shadow Puppet Play." International Conference on Arts and Technology ArtsIT: Arts and Technology, 118-126, 2011.

[5] X.F. Huang, S.Q. Sun, K.J. Zhang, T.N. Xu, J.F. Wu, B. Zhu, "A Method of Shadow Puppet Figure Modeling and Animation." Frontiers of Information Technology & Electronic Engineering, 16(5): 367–379, May 2015.

[6] C. Barnes, D.E Jacobs, J. Sanders, D.B. Goldman, S. Rusinkiewicz, A. Finkelstein, M. Agrawala, "Video Puppetry: A Performative Interface for Cutout Animation". ACM Trans. Graph, 27(5), 2008.

[7] F.V. Reeth, "Integrating 2 1/2D Computer Animation Techniques for Supporting Traditional Animation". In Proceedings of the Computer Animation (Washington, DC, USA, 1996), CA '96, IEEE Computer Society, 118-125, 1996.

[8] H.S. Seah, H. Johan, C.K. Quah, N.M. Wardhana, T.Y. Lim, D.W.S. Ong, "GF Engine: A Versatile Platform for Game Design and Development". In International Conference on Computer Animation and Social Agent, Workshop on Serious Games and Simulation, pages 147-165. Springer, 2014.

[9] D. Tolani, A. Goswami, N.I. Badler, "Realtime Inverse Kinematics Techniques for Anthropomorphic Limbs". Graph. Models Image Process. 62(5): 353–388, Sep 2000.

[10] M. Fêdor, "Application of Inverse Kinematics for Skeleton Manipulation in Real-Time". In Proceedings of the 19th Spring Conference on Computer Graphics Pages 203-212, 2003.

[11] K. Shoemake, "Animating Rotation with Quaternion Curves". SIGGRAPH Comput. Graph. 19(3):245–254, Jul 1985.