Animation for Novices

Eric Chu
Stanford University
echu508@stanford.edu

Kurt Loidl
Stanford University
kloidl@stanford.edu

Sean Rosenbaum
Stanford University
rosenbas@stanford.edu

ABSTRACT
We present a simple animation system driven by hand-gestures. The user gestures with his hands to interact with a 3D model to define animation key-frames. Extraneous features are notably absent from our system. Techniques for implementing the gestures are listed, along with solutions to some challenges faced when using vision input with graphical applications like ours.

Author Keywords
animation, modeling, hand gestures, computer vision

ACM Classification Keywords
[H5.1]: Multimedia Information Systems

INTRODUCTION
We developed a simple animation system for 3D models. The user interacts directly with a model having predefined joints to define a sequence of key-frames in an animation. Most animation systems, including Maya and Studio Max, are extremely difficult to work with. Our goal was to make our animator simple enough for novice users, rather than target professional artists who are already reaping the benefits of more sophisticated systems after having invested an enormous amount of time learning them. We believe computer vision is pivotal to achieving this goal.

Our hypothesis was that it is easier for novice users to interact directly with a model using their hands. In addition, we theorized that users would be more productive because they would be more adept with their hands. Consequently, we implemented camera-driven hand-tracking and gesture recognition.

We also developed a simple graphical user interface. Only a select few options are available to the user to virtually eliminate the steep learning curve in most animation systems. We also strove to occupy a minimal amount of screen "real-estate" by UI widgets to keep the user's focus on the model he is interacting with. Finally, our graphical components had to accommodate two-handed vision input.

RELATED WORK
3D animation is often done in modeling software like Maya and 3D Studio Max. These applications are difficult to learn and have many more features than a novice requires. High-quality animations frequently use live motion capture [2], which is complicated, usually expensive, and requires the actor to be physiologically similar to the 3D model. Oore introduced a puppeteer-like approach where recorded video sequences of a stick held in each of the user's hands controls a model [1]. However, puppetering is not a universal skill and it does not extend well to arbitrary models and sophisticated animations. Similarly, novel interfaces for model deformation and texturing using haptic devices has been explored in [3], [4], and [5]. While they may provide tactile feedback, they must be learned and are generally cost-prohibitive outside of a commercial setting. Our analysis of the related work motivated our decision to use key-frames and moveable limbs, like most modeling software do, to construct an animation.

APPLICATION ARCHITECTURE
The Object-Oriented Graphics Rendering Engine (Ogre) [8] manages the state of the virtual world and renders the scene. We maintain separate modules for the computer vision input, custom graphical user interface, and the animator in order to facilitate group development and good programming methodology. The tracked hands and identified gestures are forwarded to the core Ogre component, which propagates events to our GUI widgets. In turn, the widgets interface with the animation module and change Ogre's state using callback functions.

HAND-TRACKING AND GESTURE RECOGNITION
We hoped to implement a couple of basic hand gestures in three-dimensions. The first was a simple "pinching" gesture for clicking; the second a bimanual "zoom" command; and the third, a bimanual "rotate" command. To accomplish these goals we had to consider two factors–how to track the hands and how to determine the input. We also wanted to consider the added constraint of simplicity and functionality over a robust gesture vocabulary.

While we initially hoped to track hands in three-dimensions, the added hardware–such as a second camera for stereo vision–complicated the problem without adding extra func-

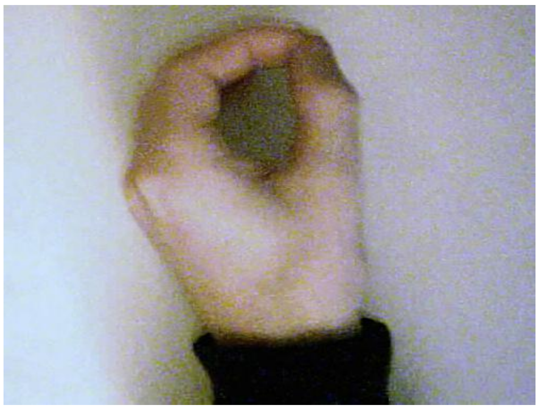
Figure 1. The “pinch” gesture.

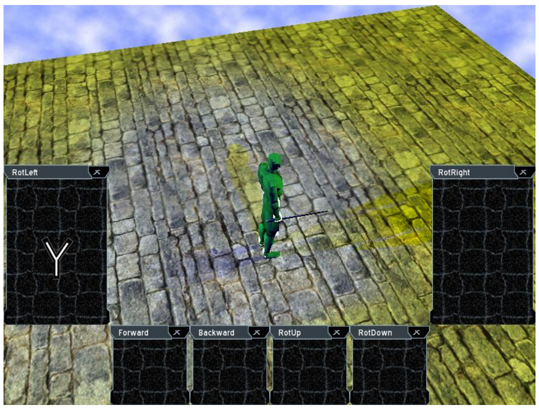

Figure 2. Region-driven interface.

tionality. Hence, we kept hand tracking simple by restricting it to two-dimensions. Furthermore, to track hands reliably with background segmentation, we wanted to put the user’s hands in front of a solid background where the skin colors would stand out significantly. To do this persuasively without having the person stand in front of a big blue screen, we used an overhead camera looking down at the user’s hands, which are placed over a solid, dark blue background. We track the position of both of the user’s hands along the plane perpendicular to the camera by selecting the lighter colors in the scene and defining the “cursor position” as the centroid of the hand contours. Thus, we make the simplistic assumption that any object appearing in the scene of a lighter color than the dark blue background must be the user’s hands. Tracking exactly the centroids of these objects will return the hand positions.

Once we were able to track hand positions, we had to decide on a natural hand gesture to accomplish our three types of inputs. To mimic a mouse click, we implemented the pinch detector by Andy Wilson [6]. It was simple, efficient and, most of all, gave us the desired functionality. In OpenCV [7], we were able to easily detect “children” contours within the hand “contours” and consider those as “pinching” gestures; then we were able to assign the “click” command to such a gesture.

In addition, we use gestures for rotating and zooming the virtual view. The user rotates about the axis perpendicular to the ground plane by pinching with both hands and displacing them vertically. The sign and magnitude of the displacement determines whether we rotate clockwise or counterclockwise and the number of degrees, respectively. The user brings together (horizontally) his two pinched hands to zoom in, or moves them apart to zoom out. To calculate the zoom amount, the distance between the two hands was used. Furthermore, a line of “zoom” was stored such that the user had to move his hands in a motion close to this line in order to complete the “zoom” command. When the bimanual motion deviated from this line, a “rotate” command is interpreted. The angle of rotation is determined by the amount of deviation from the previous line defined by the location of the two hands. With the appearance of a second hand on the screen, however, we had to differentiate between the right and left hands for the pinching gesture. To get around this, we created handedness-neutral cursors that appeared neither left-handed nor right-handed and displayed one on the screen whenever a hand appeared. If OpenCV had an efficient hand-tracking function, it would have been easier for us to implement bimanual commands.

In all, OpenCV provided excellent functionality for implementing these gestures. Functions related to color segmentation, background subtraction, and contour detection were extremely helpful. Unfortunately, OpenCV did not include hand-tracking functions or arm-removal functions. Because our cursor tracked the centroid of the user’s hands, the appearance of an entire arm, which is skin colored, displaces the centroid. To get around this, we had users wear long, black sleeves. But if OpenCV could automatically remove “arms” and only detect hands, we would not have had to provide external hardware.

GRAPHICAL USER INTERFACE

We initially implemented a region-driven interface. When a pointer entered one of the regions reserved for a feature, the application would activate it. Unfortunately, we found this interface to be very cumbersome. First, it obstructed our view of the world. Second, pointers often mistakenly strayed into these regions, which lead to unintended side-effects.

Since then we reworked the user interface using buttons and scrollbars. Only a few critical features are supported by the graphical interface. The scrollbar on the left side of the screen lets the user rotate along one of the axes parallel to the ground plane to move vertically. Either one of his hands may pinch and drag this scrollbar. This pinch and drag interaction is natural for people to do. An options window is hidden on the right side of the screen, which comes into view whenever one of the pointers moves next to it, thus preserving precious screen-space. It stores several buttons for playing the skit (animation sequence), recording the last key-frame, and clearing the skit. The user pinches over these buttons to press them. The window at the top also slides down when a pointer moves near it; here previews for all the key-frames are displayed. All of these components were intentionally

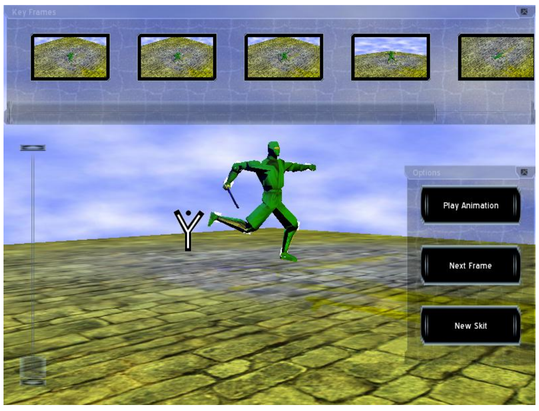

Figure 3. Current interface with the key-frame preview and options window visible.

made large so they would be easy to interact with in spite of the error and latency inherent in vision input.

Given more time, we would further improve the layout of the components. During the demonstration we observed many people accidentally popping open the options window. One alternative is to use hand-gestures in place of those buttons. Another possibility is requiring the user to pinch and drag the window open. The problem is, if we use too many GUI components, the application begins to feel like the gestures are implemented as an afterthought. On the other hand, if we use too many gestures, we will inundate the recognition system, making it more difficult to accurately identify them and for the user to remember them. This was a serious problem we grappled with which lead to us integrating zoom and rotate gestures, yet use widgets for other features. In this case, the latter solution should be tried before experimenting with new gestures.

We also discovered that the key-frame preview was not that important to users. People tended to construct short animation sequences, which they remembered on their own. We originally planned to let users reorder key-frames by pinching and dragging their preview frames. A key-frame could also be loaded by dragging its preview into the world. We opted not to do this because these features were not critical to a simple animation system, though we certainly feel that adding them would not complicate it.

The GUI implementation was more laborious than we expected. Crazy Eddie's GUI System (CEGUI) [9] is a powerful GUI library compatible with Ogre, but it was intended for only one pointing device. In particular, the event system, which the programmer has control over, only propagates events for one pointer. Therefore, you cannot have two cursors displayed at once or even interact with scrollbars or other components that are thrown off by events pertaining to another pointer. As a result, we implemented wrapper classes to handle multi-pointer interaction with the underlying CEGUI components. We track and propagate pointer events on our own, as well as render our own cursors. At this point CEGUI components are only used for rendering the widgets.

ANIMATION

Each limb can be rotated along its predefined axis of rotation at its parent joint by pinching one hand over it and moving the hand. The joint positions are computed with forward kinematics. A set of these limb rotations are stored with each key-frame. Ogre interpolates between the angles of each of the joints in the key-frame list to playback the entire animation.

Because our hand-tracking is less precise than other pointing devices, it is often difficult to pinch a limb unless the view is zoomed in on it. We could later resolve this by locking the pointers onto the closest limb when it is in proximity to the model, or by assigning limbs a larger selection radius.

Animations are arguably better and more efficiently made with inverse kinematics. We initially planned to integrate this, rather than forward kinematics, but found it much more difficult to implement in Ogre. Since it is tangential to our project goals, we decided to defer its development.

CONCLUSION

We successfully demonstrated a simple animation system driven by hand-gestures. Novice users are able to create skits with little to no prior experience modeling or animating 3D models. We believe the challenges we faced are applicable to many applications using vision input and for designing graphical applications in general.

ACKNOWLEDGMENTS

We would like to thank Monzy for his insightful comments and everyone who attended the demonstration for their interest and great feedback.

REFERENCES

1. Sageev Oore, Demetri Terzopoulos, Geoffrey Hinton. A Desktop Input Device and Interface for Interactive 3D Character Animation. Graphics Interface, 2002.
2. Basmah Daham Jassem. Real-time Gesture Controlled 3D Animation. MSc in Distributed Multimedia Systems at University of Leeds, 2005.
3. SensAble Technologies. http://www.sensable.com
4. Thomas Massie. A Tangible Goal for 3D Modeling. IEEE Computer Graphics and Applications, 1998.
5. Mark Foskey, Miguel A. Otaduy, and Ming C. Lin. ArtNova: Touch-Enabled 3D Model Design. http://www.cs.unc.edu/ geom/ArtNova, 2001.
6. Andrew D. Wilson. Robust Vision-Based Detection of Pinching for One and Two-Handed Gesture Input. Symposium on User Interface Software and Technology (UIST), 2006.
7. Open Source Computer Vision Library (OpenCV). http://www.intel.com/technology/computing/opencv
8. Object-Oriented Graphics Rendering Engine (Ogre). http://www.ogre3d.org/
9. Crazy Eddie's GUI System. www.cegui.org.uk