Gestural VJ Software

6.866 Final Project

Justin Manor
manor@media.mit.edu

Introduction

The goal of this project was to create a prototype software application for video editing that employs a vision based interface. Current realtime video authoring packages are typically controlled by standard keyboard, mouse, or MIDI inputs, and often rely on complex GUI's to keep track of available media clips and their states. As an alternative I built a system which integrates video editing and effects software, vision based gesture tracking methods, and a simple pair of painted gloves.

Motivation

The relentless increase in processing power of personal and laptop computers has made it possible to emulate many audio and video hardware systems in software. The functionality of a whole rack of audio synthesizers [4,6] or a video-editing station [5] are quite commonly encapsulated into a single program, affordable by consumer enthusiast standards. A great number of these software packages available are designed explicitly for the realtime creation of sound and or visuals for live performance.

There are literally thousands of "laptop performers" around the world creating music and visuals for live audiences and more are appearing every day. Concert goers are beginning to expect high-quality video to accompany live music, and large projection screens are becoming standard in a great number of musical venues. In the past few years VJ's, or video jockeys, have gone from novelty sideshows to headlining acts in many cities.

Current interfaces for commercially available realtime performance software typically consist of a graphical user interface and computer inputs such as keyboard, mice, and MIDI devices. The GUI is for the performer to keep track of available media objects and processes, and is not present in the visual feed displayed to the audience. By pressing keys or moving sliders with a mouse, users toggle or change parameters of the audio instruments and video clips being output by the system. MIDI control is also a common feature, via piano style keyboards or other specialized "knob boxes".

Because the software interfaces of prevalent performance packages are quite complex and often hidden from the audience, the performer and her actions lack a clear and engaging dynamic to observe. Correlation of the software performer's physical actions to audiovisual events by the spectators is very difficult because all key presses and mouse drags look very much the same. When the performer becomes de-emphasized in this way, there is a loss of liveness to the performance because the audience focuses on the media alone, and not the fact that it is being created, and maybe improvised, on the spot. A gestural interface would keep the artist in the foreground and give the audience clues about her intention through body language.

The prototype gestural VJ software I have created attempts to address these issues, and takes a few pointers from a recent Hollywood film.

Inspiration

Steven Spielberg's recent sci-fi drama movie *Minority Report* takes place in the year 2054 and the main character is a police detective named John Anderton (Tom Cruise). Part of his job is to search vast video databases quickly to find subtle clues about murder cases. The system uses a vision based hand-tracker to follow the user's gestures and allow realtime search and manipulation of video clips.

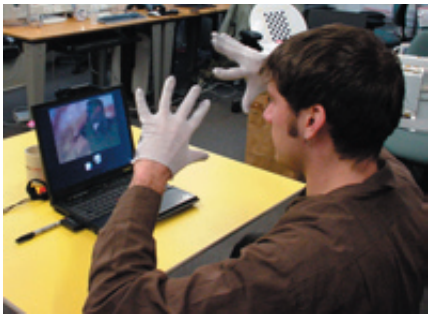
The interface roughly resembled what it would be like to use two computer mice at once, but with a few extra degrees of freedom. One handed operations included moving, opening, and deleting clips. Two handed operations were used for resizing the video and stepping through long series of clips. Position as well as orientation of the hands was recovered and intuitive gestures were mapped to common video editing functions. For example, rotating the hand clockwise scrubbed the video forward, and left rotation rewound.

Tom Cruise in *Minority Report*

One of the reasons the fictional interface was so convincing on screen was because it was designed by MIT alum John Underkoffler specifically for the movie. John was the science and technology advisor for the film, and his decade long stint at the Media Lab gave him a good idea what the cutting edge of new interfaces is and where the field might be headed.

My System

I developed video editing software on my laptop, which has a small USB camera that attaches to the top of the screen. The user sits or stands in front of the laptop and swivels the camera to aim the lens at themself, taking care that the field of view will contain the desired range of arm motions. You can see me testing the system in the picture below.



Me interacting with Gestural VJ Software

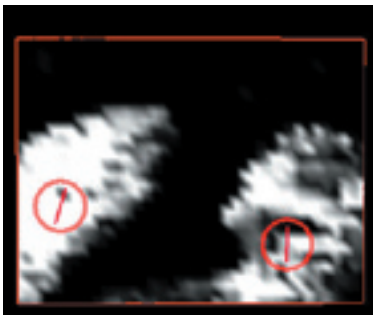
For this prototype I kept the complexity of the interaction quite low and only allowed control of two variables at most at any time. The aspects of video that I experimented gestural control of were: video opacity, audio volume, playback speed, playback timeline position (scrubbing), clip size, and clip rotation.

The software was written in C++ and uses Microsoft's DirectShow libraries to read camera input and video files. The camera data was sampled at very low resolution (40 by 30 pixels) for speedy calculations and to smooth out noise. I used a hierarchy of images moments to provide a coarse summary of what the camera observed, quite similar to Freeman et. al's methods of video game control [7].

During initial development, I was using only moment calculations of image intensity, and I found that getting repeatable results was quite difficult with lighting and clothing variations. I switched to analyzing the amount of motion in the scene and control became much easier. By subtracting a running average of the scene from the current frame, a measure of the scene change is available. This motion-map is then input into the moment calculations.

The system always assumes there are two hands present and calculates two image motion moments, one on the left and one on the right. To maximize the amount of movement my hands generated as seen by my software, I painted a pair of gloves with a checkerboard pattern to make them visually busy. This caused even small movements of my hand to register as large changes in frame-to-frame intensity. Below are pictures of the gloves and an example motion map that is generated every frame.

Visually busy gloves



What the program sees

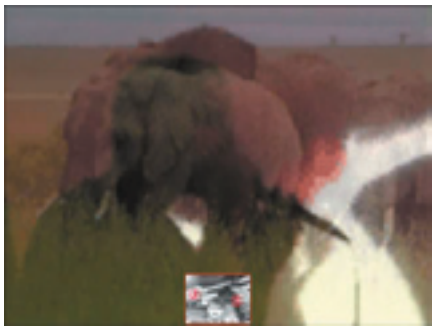
The red circles denote the center of mass of motion on each side of the scene. The line inside the circle represents the orientation of that mass. Below are more examples of camera snapshots with the intensity information displayed instead of the motion, to illustrate the basic hand tracking that is taking place.

Rough motion centroid position is recovered, as well as orientation.

The Fader

The first editing tool I built was a simple video fader. Two video clips were playing at the same time and the user controlled the intensity of each with the vertical position of his two hands, as if they were on large slide potentiometer faders. If both hands were raised, both video clips played at a high volume and on top of each other. If one hand was fully raised and the other fully lowered, only one video was playing.

Right hand raised - Video 1 mostly.



Both hands up - both videos play.

Left hand raised only - Video 2 max

The Scratcher

This mode allowed the user to control the playback position and speed of two clips. Speed was determined by the orientation of the hand, with zero tilt being normal speed. Rotating a hand in place was analogous to adjusting a playback speed knob. Translation of the hands 'scratches' the video, similar to the way a vinyl DJ can scratch a record. I extended the analog DJ metaphor by designing the software to increment (or decrement) the absolute frame number it was displaying in proportion to the speed of circular hard motion. The amount of area swept out by each hand every frame set the speed of the scratch, and the direction of the circular motion (clockwise or counterclockwise) decided whether it was a pushing the video forward, or pulling it back.

The Stretcher

I chose rotation and scaling of video to demonstrate two-handed gestures. The relative positions of the hands were measured and used to set the clips' size and tilt. When the hands are close to together, the image is compressed, and as they are separated, the clip stretches out. The natural pantomiming of squeezing and stretching gestures translates into those actions.

Hands press in - video squashed

Hands pull apart - video stretched

The relative heights of the hands controlled rotation of the video playback. When both of the user's hands were at the same vertical position, the video clip was not tilted. But when the hands approached opposite corners of the camera field, the video playback would lean in the direction of rotation. In this way the natural motion of rocking or twisting hands or even the whole body would translate to very similar behavior in the playback video.

Clockwise twisting of the body produced clockwise video twist

Same for counterclockwise

What Worked, What Didn't

The fact that the whole thing ran on a two year old laptop was very encouraging. The machine was processing three video feeds simultaneously, with one (the camera) modulating the other two (arbitrary movie clips). Granted, it was not hi-fidelity output; the video clips were 160x120 pixels running at fifteen frames per second, but the software should scale nicely to a desktop with a dedicated graphics card.

As far as being an engaging interface to witness, I think the system was a success. Anyone who saw me testing the software was immediately drawn into the experience. The controls were never exquisitely tuned, but causality was obvious to people and most wanted to try it for themselves. The easiest interactions for people to grasp were the visually oriented controls, like rotating and resizing the video. I believe that the visual feedback loop if more powerful than an audio feedback loop in the case of gestural control. Informal user testing found that people could easily get lost in audio space when trying to use the Video Scratcher. Too many parameters were available to adjust, and untrained users quickly became frustrated.

Image motion moments worked quite well for continuous control of state variables, but proved to be a poor solution for discreet selections. Motions like turning knobs and sliding faders translate well into exaggerated gestures and are detectable. I found it hard to come up with effective switch metaphors and corresponding hand gestures to make hard state changes. Perhaps methods like orientation histograms or even microphone input would provide a natural solution.

References

[1] Cohen, Philip et al. Multimodal Interaction for 2D and 3D Environments. *IEEE Computer Graphics and Application*, 1999.

[2] Rokeby, David. Personal Website : *http://www.interlog.com/~drokeby/home.html*. 2002.

[3] Cycling 74 company website, makers of MAX/MSP. *http://www.cycling74.com*. 2002.

[4] Native Instruments website, creators of Reaktor. *http://www.nativeinstruments.de*. 2002.

[5] NATO.0+55+3d software website. *http://www.eusocial.com/*. 2002.

[6] Propellerhead website, makers of Reason. *http://www.propellerheads.se/*. 2002.

[7] Freeman, William et al. Computer Vision for Interactive Computer Graphics. IEEE Computer Graphics, May/June 1998.

[8] Corradini, Andrea. Multimodal Speech-Gesture Interface for Handfree Painting on a Virtual Paper Using Partial Recurrent Neural Networks as Gesture Recognizer. *Proceedincs of the Internation Joint Conference on Artificial Neural Networs*, Vol III. 2002.