Repurposing of Video Footage

CS750 Final Project
Fall 2003

Dave Cardon
Heather Ipson
Lauralea Otis
Mike Smith

<u>Introduction</u>

Standard video editing is limited by the constraints of existing footage. For example, conventional editing does not allow editors to easily adjust character timing, panning of the camera or removal of unwanted background and foreground objects. Typical solutions to these problems require expensive post-production changes or possible re-filming of footage. We propose a system that provides editors more flexibility in addressing these types of editing problems.

In our system, we separate individual frames of video into background and foreground layers. We use the term "background" to mean the elements of the video sequence that remain stationary over time and "foreground" to mean the elements of the video sequence that change over time. After extracting these layers, the user may recombine them and mix them with other sequences to create novel video footage.

<u>Previous Research</u>

Some previous research that we looked at includes:

- Image Mosaics
  R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphic and Applications*, 16(2):22-30, 1996.

  Image Mosaics involves automatically compositing video frames creating a large panoramic image of arbitrary shape and detail. Video mosaics can be used with planar scenes, panoramic scenes, and scenes with depth variation. The basic algorithm is based off of a simple idea, the alignment of different pieces of a scene to a larger picture of the scene. Image warping and blending can help to align the images. Szeliski uses local image registration to find motion, by minimizing the sum of the squared intensity errors over all pixel pairs. Although this technique works well locally, Szeliski found that using a more global approach such as hierarchical matching or phase correlation was much more accurate.

- Optical Flow
  Another way to calculate motion is by finding the optical flow of a video sequence. Optical Flow describes how pixel intensity changes over time. It is often used to determine the motion of small regions or pixels in a scene. For more information, consult Dr. Barrett's notes on "Motion"

- Layers
  J. Wang and E. Adelson. *Representing moving images with layers*. IEEE Trans. on Image Processing, 3:625-638, 1994.

  Wang and Adelson present a method for representing images that move (video sequence) in sets of overlapping layers. Each layer includes both intensity and transparency. The layers are ordered by depth and follow the rules of compositing. Velocity maps show how the layers change over time. The layered technique is flexible.

Our Solution

Our solution to the problem can be split into several parts. These include extracting and fixing frames from video footage, creating motion vectors between pairs of consecutive frames, creating a large composite background image of temporal color modes, and using this background image to determine the foreground and background pixels in each frame of the sequence.

Image Preprocessing

The main video sequence that we use in our experiments is a short animated sequence from "Cat' s Don' t Dance". We decided to begin experimenting with cartoon animation and then to progress to live footage if time permitted. We chose cartoon animation because the characters are generally well defined and the camera motion is often very simple. We used standard video editing software to digitize the video sequence and represent it as individual frames. Each image had a small section of black along its borders and these pixels caused difficulty when creating motion vectors. To fix this problem we removed 10 pixels from the left and right borders of each of the images in the sequence.

Motion Vectors

We chose to use image registration to find the motion between consecutive frames. This is a fairly simple approach, but we wanted to do the simple thing first and then progress to more advanced methods. We also argued that it should be sufficient for our cartoon sequence.

Our algorithm for image registration involves sliding the second image across the first image in each pair of images and finding the position for which the sum of the differences in pixel intensities of the images in a minimum. The resulting position is the offset of the second image from the first. We call this offset the motion vector.

In general, this algorithm produces good results for a couple of reasons. First, we limit the type of camera motion to panning. Second, we are dealing with sequences where the background is similarly lighted throughout and has distinct colors allowing image registration to correctly detect motion.

Although the results of the algorithm are fair, there were several problems that we ran into during the implementation. First, we found that we were getting false minimums when doing image differencing due to a black border on each frame. After ignoring the borders of each frame, the motion results were much more accurate.

Another problem is that shifting and differencing each image against it' s neighbor for every possible overlapping position is very time consuming. A speed up we implemented is to create a threshold within which the image can move. For example, if it is predetermined that the camera motion between two consecutive frames image will not exceed 20 pixels, then we only have to search for the best match within a 20 pixel radius. Adding a search radius threshold speeds up the algorithm considerably but requires prior knowledge about the video sequence that the user may or may not have.

Below is an example of the motion that is detected between two sequential frames. The offset between the first frame and the second frame is 19 in the x-direction and 0 in the y-direction.

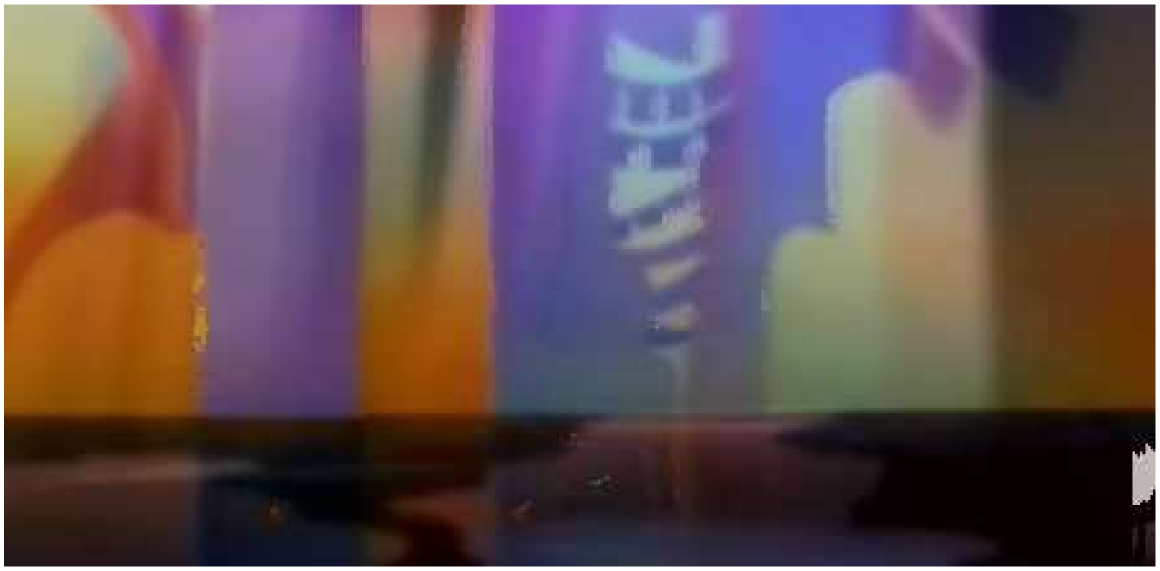
Enhancements that could be made to the algorithm include the following:

- Using hierarchical matching. This approach uses a scale space to first register smaller versions of each frame. Motion estimates from these smaller coarser images are then used as initial guesses for the motion at finer levels. This helps speed up the matching as well as avoiding local minimum.
- Using sub-pixel interpolation. In our version of the algorithm we can only match at the pixel level. However, motion between the consecutive frames of video are seldom on pixel boundaries. Adding sub-pixel interpolation would allow more accurate motion estimation between frames. We also believe that it will help reduce aliasing around the edges of objects in the frames.
- Accounting for camera zoom and rotation. Adding this would allow us to handle more complex camera motion in video sequences.

Composite Background Image of Modes
The motion information for each frame is used to create a composite background image of temporal intensity modes. The basic idea is to use all the frames in the sequence to determine a background pixel intensity for every pixel of every frame. It is helpful to think of the panning video footage as shots of a much larger composite image. We lay each of the frames on top of each other—lined up according to their motion estimation—and analyze the temporal intensity change of each pixel in the composite image.

The first method we tried was to make a histogram of the colors that are seen at each pixel of the composite image. The color that is seen the most often is then set to be the color for the background. The reason that this should work is because each pixel should usually be its background color and it will only be different for a small amount of time when an object is in front of it. This method wasn't completely successful because the pixels tend to have slight variations in color over time.

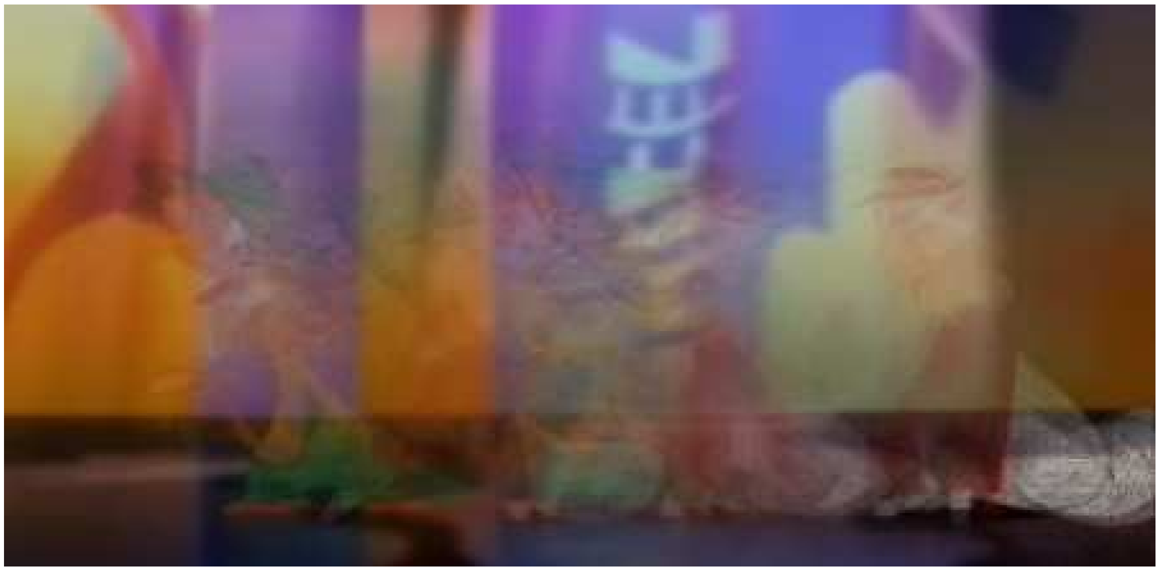
The next method we tried was to keep a vector of different colors for each pixel. If a color being added to the vector is within a certain threshold of a color that is already in the vector, then it is averaged it into that pixel color. After all frames have been analyzed, the color in the vector with the most values averaged into it is the color chosen for that particular pixel in the composite image. In other words, the most dominant mode of the pixel colors is chosen as the color for the composite image. The following is the final composite image produced from this method.



This is a good composite image, but there are still several places where the noise can be seen. In an attempt to reduce the noise of the image, we blurred the image. The blurred background has less noise and is seen below.

Although the blurred background is more aesthetically pleasing to look at, it doesn't obtain better results for what we are trying to accomplish. This is because along the edges it creates pixel values that don't match well with pixels in the original frames. One method for fixing this would be to do a median filter on the composite image. We tried to implement a "cheat" for the median filter, by averaging the separated color planes in the kernel and then choosing the pixel with the closest Euclidean distance to the average as the replacement pixel. Unfortunately this seemed to increase noise rather then decrease it. If there where more time, it would be interesting to try a true median filter on the color planes individually or do a median filter on the intensities, but keep the corresponding r, g, b value.

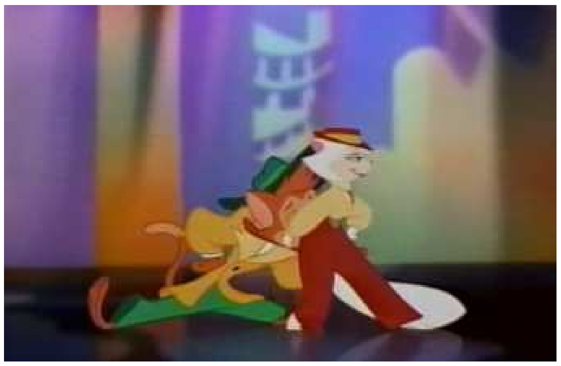
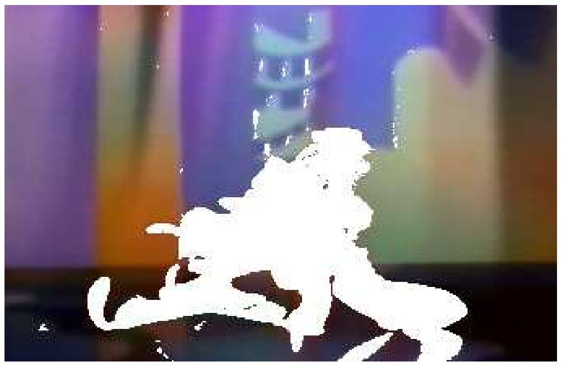
Another method we tried for creating a composite image was to set each pixel in the composite image to the mean of all the possible pixel values. The results of this method are seen in the image below.



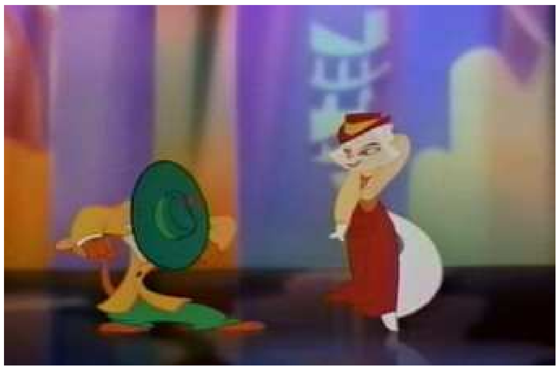
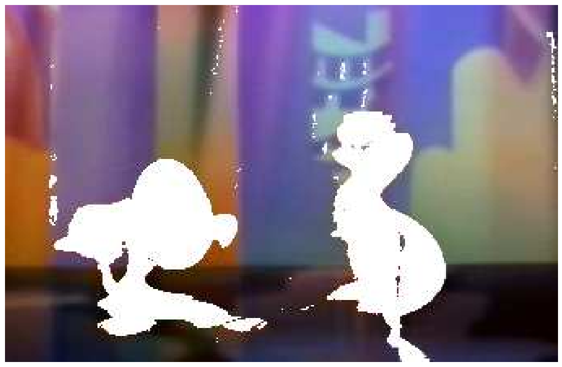
The image created from the means isn't very useful for creating a composite image because you can see the characters at all the places they have been. However, the mean image is useful for verifying that the motion vectors are correct as everything seems to be lined up properly.

Separating Foreground and Background

The next step is to use the composite image of modes to separate the background from the foreground in each of the frames. This is done by going through each of the pixels in the original frames and seeing how closely the original pixel matches its corresponding pixel in the composite background image. If the original pixel color is within a certain Euclidian distance in color space from the composite background pixel color, then the pixel is considered to be a background pixel and the original pixel color is kept in the frame. If the original pixel color is not within a certain distance from the composite background pixel color, then the pixel is considered to be a foreground pixel and is set to be white in the original frame. This creates a white mask where the foreground pixels are located. The following is an example of an original frame and the resulting frame after separating the foreground and the background.

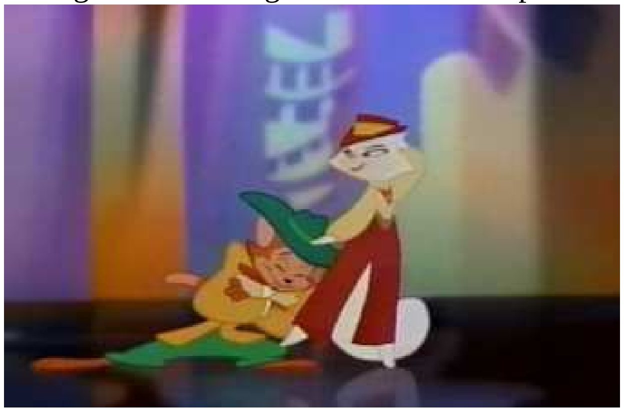
Here is another example for a different frame.

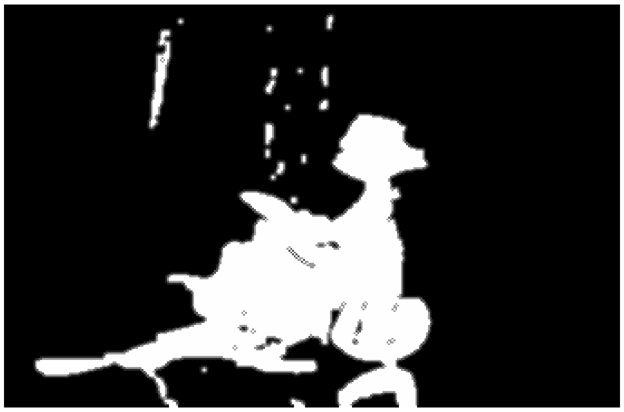
The separation does a good job at recognizing the characters in the frames. However, there are several places where background pixels are classified as foreground pixels. There are also some small groups of background pixels that should actually be part of the foreground. This is most common when the color of the character is very close to the color in the background.

Approximating the Covered Background

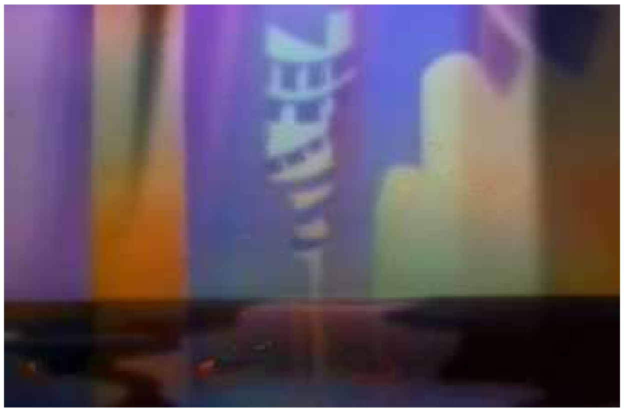
Now that we have a mask for the foreground we can pull the foreground into a separate layer and fill the space in with elements from the background. The background may be filled by using values in the composite image. However, we mentioned before that the color for each pixel changes slightly over time. To maintain consistency between the filled pixel colors and the original background colors, we look before and after a given frame for color values at a given pixel location. We average the first eight non foreground pixel colors that we find and use the average for the background color of the pixel in the frame.



Original frame



Mask for the frame



Calculated Background

One nice side affect of this method is a smooth temporal blending between these masked pixels. The biggest problems occur where the foreground masks are not accurate. Even in these cases however, the gradual change in pixel intensity due to the averaging described above makes the artifacts more subtle and smooth.

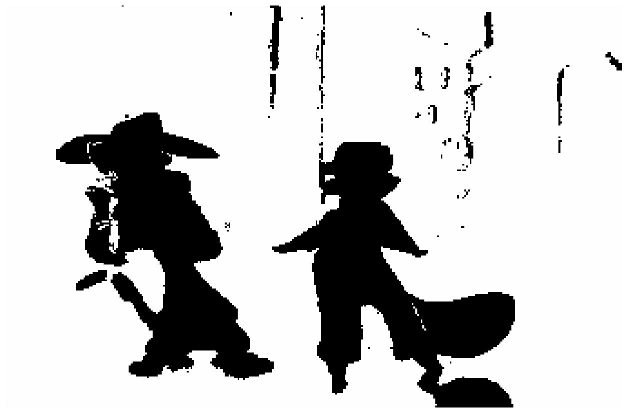
Improving the Foreground

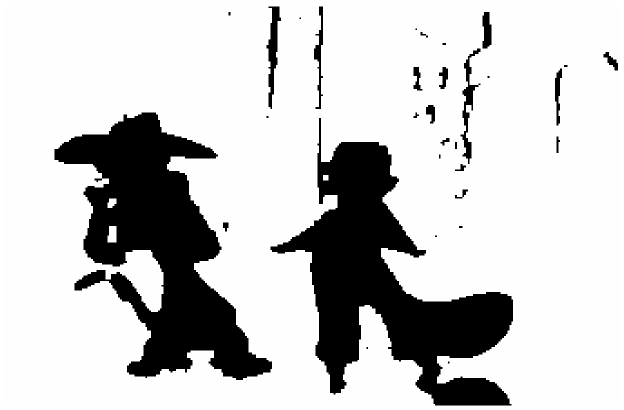
The foreground images extracted directly from the mask have very hard edges and many unwanted gaps. There are also many unwanted artifacts from the background that were classified as foreground. Automatic processing of these foreground masks is a difficult task that must be done effectively to have nice foreground "sprites" to use in video editing.

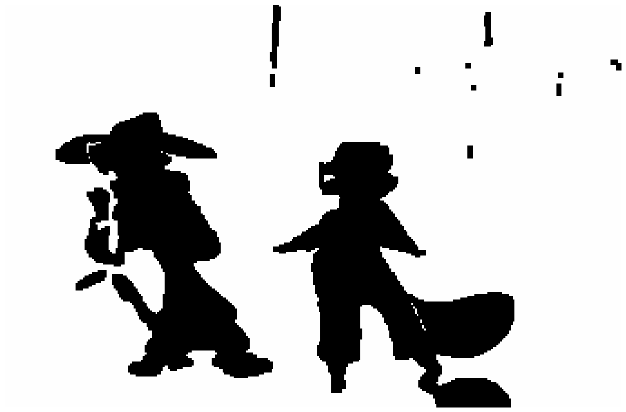
We tried a variety of methods that included looking at neighboring pixels for similar colors to determine if a certain pixel was for sure a background pixel or a foreground pixel. These methods did not make any significant improvements over the initial segmentation. The main problem is once again when the colors of a foreground object match well with the background. The initial paradigm that we set up to solve stated that colors similar to the most common background color should be classified as background. In the case that a foreground color is similar to the background, the paradigm is not very useful in classifying pixels.

The final method that we use consists of binary erosions and dilations on the foreground mask to help fill gaps and or remove small connected components. We played around with erosions and dilations and found that simple open or close operations gave nice results as shown below. The open operation gets rid of a lot of unwanted noise, but increases the gaps in the characters. The close operation helps fill gaps in the characters, but increases the size of the unwanted noise. (The black and white images have been inverted to save a little on ink)

Original image


Mask


Result of a 3x3 close operation

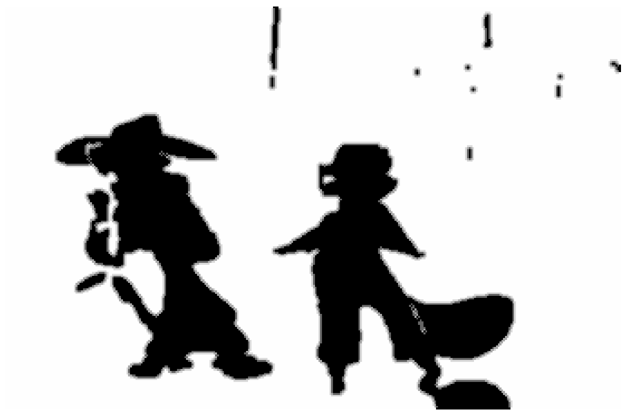

Result of a 3x3 open operation

To get rid of unwanted artifacts, a connected component method may be employed to find small components that are not desirable in the animation. Also, the proximity of smaller components to larger components seems like it would be a good indicator of whether or not they should be kept as part of a larger component that happened to be broken up during segmentation. We can also take advantage of user input stating that there are at most two significant objects in the scene. It is not unreasonable to ask this of a user for scenes involving a few characters that remain on the screen. We feel that adding a connected component cleanup operation will improve the results significantly.

To avoid sharp jagged edges in compositing, we smooth the mask. If we use a standard 3x3 box filter on the mask image, the resulting smoothed edges are too large and noticeable after compositing with another image. Because of this, we use the following filter:

$$p(x,y) = \frac{[\, p(x+1, y) + p(x-1, y) + p(x, y+1) + p(x, y-1) \,]}{[4 + (1020 - p(x+1, y) - p(x-1, y) - p(x, y+1) - p(x, y-1)) / 255]}$$

p(x, y) is the pixel in question and we are using 255 as white and 0 as black. This filter looks a little odd, but after examining it you will find that it merely gives more weight to the black surrounding a pixel while doing no damage to a pixel surrounded completely by white. The results of this smoothing can be seen in the picture below. Notice how the blurred edges are kept very small and close to the objects.


Result from smoothing the mask obtained by the open operation.

Real Time Play Back

The final test for our project is to play back a scene with new camera motion after changing the background. We are able to do this in real time while allowing the user control over the camera motion associated with the new background in the scene. It is a simple extension to add multiple sprites or foregrounds to the scene during playback.

Conclusion

We were able to do pretty much what we set out to do. We are happy with our results, but also see room for significant improvements, many of which are noted in this report. As a group we had a pretty fair division of labor with the exception of our group leader Dave Cardon, who ended up just standing over us with a bull whip making us code until our fingers bled. Dave did make up for it however by discussing many core ideas with us and offering some great insights.