18-551 Spring 2000

Group 5

Dallas Baker
Jen Deroy
Kevin Gotze

Virtual Advertising

Final Report

PROBLEM

Advertising is an increasingly important industry in the modern media-driven world, and as technology evolves, it helps advertisers reach their target audience in more and more ways. Ads pop up on Web sites, through sponsorships, and on digital billboards, all to maximize visibility of a company or product. As it is, there is still plenty of wasted advertising, and we feel with the help of technology we can solve one particular problem. Sporting events are broadcasted nationally, yet the ads in the stadium are often oriented to the local market. For example, a Broncos/Steelers game at Mile High Stadium may have ads for a Denver based company, which the fans in Pittsburgh watching on TV are unable to access or benefit from. Or possibly, a Pepsi sponsored event takes place in a stadium in which there are Coca-Cola ads. This sort of counter-productive advertising could easily be remedied with the help of video signal processing. Now that efficient means of implementing this particular technology are available to us, we can superimpose our preferred ads over unwanted ads, like in the case of a CBS broadcast at Times Square. CBS superimposed a CBS logo atop the NBC logo that is on the Times Square TV.

THE SOLUTION

By digitally altering video footage we overlaid our own 'Group 5' ad over a Budweiser ad to demonstrate how we could diversify advertising, making the ad more efficient and effective. Given the original ad, we used color information to find the area of the ad in each frame, then superimposed our ad atop of it. We will explain in more detail how we arrived at this solution in the next section.

PAST PROJECTS

We looked into past projects done by 551 students and we did not find any that really dealt with the issues that we had to work with. This may be due to the inadequacy of past EVM's in terms of memory space and processor speed, but as a result, we had no prior work on which we could build. Even through extensive searching on the Internet, information on Video Processing was mostly theoretical and lacked any coding samples to work off of. Therefore all of the code in our system is completely our own. Hopefully our project will help future groups develop their own Video Processing projects with more ease.

OVERVIEW OF THE PROCESS

PREPROCESSING

We chose to use sports clips in the uncompressed .avi format. The reason behind this was to retain as much image information as possible and to avoid potential headaches involved with decompressing a frame. The problem now was to find a way to take an .avi and break it up into individual frames, then after the changes have been made, package

them back into an .avi. At first Codecs and decoders were explored, in the hopes of coming across something that had been already written that would do the job, but nothing viable surfaced and it was suggested to us that we look into the MSDN library to look for C functions that work with .avi's. We went online to http://microsoft.msdn.com and searched for functions and discovered that there was a library of functions that had what we needed. The libraries used were vfw32.lib and winmm.lib. Using the C functions in the libraries we were able to isolate the video stream of an .avi, remove header information, grab one frame and send it to the EVM, then send it back an write it into a new video stream. The code for this is located in projectpc.c. Our frames ended up as .dib files and since they lacked header information, could not be viewed by any image editor, but using Matlab we could verify that indeed a frame had been isolated and transmitted properly. What we also discovered is that .dibs store the color information in a blue-green-red order rather than a red-green-blue one. The image also is upside-down. We wrote a Matlab script, flip.m, which would correct these problems so the image could be viewed normally. When these .dib's get send to the EVM this problem is corrected, so any filter we used was altered so it would correspond with the .dib images then flip back to normal when sent to the EVM.

Our video clips were 320 x 240 x 3 bytes per frame with 40 bytes header information. Our videos ran at 30 frames per second as well. This means that a single frame contained 230400 bytes of info. We discuss how we handled transfer of this information into the EVM in another section.

PROCESSING THE IMAGE

A number of Algorithms were attempted to find the ad in each frame with varying results. The following sections discuss what methods were used, which ones weren't and why or why not.

CORRELATION

The direct correlation method worked by simply correlating the filter (the ad to be replaced) with the image in the spatial domain. This method as a general finding method was extremely limited, and we did not expect terrific results from it. A slightly rotated ad could throw off the correlation results a great deal, especially if the ad contained a lot of detail. Similarly, if the ad in the image was different from the size of the ad in the filter, the correlation results would be skewed as well. However, we realized that correlation would be useful as the final step in other detection methods, after potential areas had been located, to verify the potential area as actually being an ad.

The correlation suffered from speed problems, as well. A correlation of the entire image with the filter is O(x_image * y_image * x_filter * y_filter), an extremely costly operation. To speed this process, we introduced a sampling factor for use with large-area correlations. A sampling factor of n essentially only correlates every nth pixel of both the image and filter. For a 2X downsampling, reduces the number of points correlated by 16 times, and for 4X downsampling, the number of points is reduced by 256 times-a drop of two orders of magnitude. For very large correlations, 4X downsampling was used, followed by 2X at the 9 points around the best location found, followed by full resolution correlation at the 9 points around and on the best point found at 2X downsampling. This led to large savings in correlation computations. However, since correlations were an

expensive and noise-prone method of detection, large correlations were for the most part avoided.

Further savings in the correlation routine were achieved by changing the method of correlation. At first, a multiplicative correlator was implemented, storing the results, with the best correlation points having the highest values, in an array of ints. Because of the two hardware multipliers on the C67 chip, this method was not as slow as would be expected for O(n^4) multiplications. However, a subtractive correlator was implemented and used for several reasons. Firstly, there was a slight speed improvement in the use of a subtractive correlator, likely due to the fact that there were 4 arithmetic units on the C67, so although the operations took roughly the same time per operation for both methods, the C67 was able to perform more operations simultaneously in the subtractive method. Additionally, after normalizing the subtractive correlation results, the result could be stored in an unsigned char, saving 3 bytes per color per pixel over the int storage of the multiplicative correlation. This is especially important when paging memory, as more data could be paged onto the chip in less time.

In spite of the optimizations done to the correlation, it remained a non-viable method for whole-frame detection of the ad. However, for final validation, coupled with routines to rotate and resize the ad, it remained an important function.

EDGE DETECTION

One method of finding the ad at varying sizes and rotations was the use of edge detection to find potential locations for the ad. This was not a complete method in and of

itself, but needed to be coupled with a verification technique (such as correlation) to do the final testing of the area.

The ad in the image was never subject to rotations past a few degrees. The ads had essentially vertical side edges, and slightly sloped top and bottom edges. By using this relative shape resistance to rotation, we implemented a vertical edge detector that would find vertical edges, assume the edges are sides of the ad, rescale the filter appropriately, and look on either side of the edge for the ad. The edge detector itself was a horizontal Laplacian high pass filter, dimensions 1x3. The Laplacian filter is zero-mean and not sign-dependent, so it found falling edges as well as rising edges. For ease of computation and processing, the high-pass filter output was thresholded into binary output for later stages.

To assemble a cloud of points from the Laplacian filter into a collection of edges, another routine was created. This function went through the binary output of the high-pass filter, looked above each point to determine if it was the top of a line, and then "walked" down the line of 1s to the end of the line. When the end of the line was reached, the function found the length of the line, used the known aspect ratio of the filter to resize the filter appropriately to the length of the edge it found, and correlated on either side of the edge. While not rotation-invariant, the edge-detection routine adapted to the scale of the ad in the image, making it a more robust routine. Additionally, the Laplacian filter was a fast routine, O(n^2). The edge-assembly method was also fast, essentially O(n^2) as well. The ad resizing and correlations remained the costly and slow part of the whole algorithm, but this was limited by correlating over only a few points.

The edge detector suffered from a few problems. Since the Laplacian filter used was small (1x3), it was extremely fast, but subject to noise and low gradients. The ads we looked for did not always have very strong edges, especially when they were far away. The edges frequently fell below the threshold, and weren't detected or passed onto the correlator to be validated. Also, noise on the edge of the ad or a slight slope in the vertical edge of the ad often caused the edge assembler and correlator to get the length of the edge incorrect, causing the ad to be scaled incorrectly, and causing the correlation to fail. With more time, however, the edge detection method could be improved, possibly implemented with a Hough Transform to improve the edge assembly from high pass filter output. A broader or better filter than the [-1 2 -1] filter we used could be applied, to make the high pass filtering produce better output. Broadening the filter runs the risk of increasing the noise in the output, however, this is an acceptable tradeoff for the increase in valid points returned, as the extra noise will be filtered by later steps.

CORNER CORRELATION

Another attempt to use the relatively stable geometry of the advertisement with regards to rotation and scale was a corner correlation. Even as the ad rotates, scales, or skews slightly, the corner will stay as a right angle, and most of the information will not change significantly. Therefore, by selecting only a small region (10x10 in our case) of the filter at the corner, we can find the corners of the ad, derive the geometry of the ad in the image, and replace it more accurately.

Using the top left corner of the filter, the routine correlated the image with the small section of the filter. After the top left corner was found, the area directly

downward from the upper left corner was correlated with a filter made from the lower left corner of the ad filter. When the bottom left corner was found, the height of the ad was computed, the width of the ad was extrapolated from the known aspect ratio of the advertisement, and the whole ad was correlated around the center of where the ad should be, according to the corners.

This method increases the speed of the whole-image correlation by only correlating a small section of the filter, for increased performance results. Additionally, the correlation can use a downsampling factor of 2 for increased speed performance. The corner filter also contains a small section (1 pixel wide line) of the background, to minimize the detection of larger bulk parts of the ad or other corners of the ad, and peak the correlation at the appropriate corner only, minimizing false positives. This information is especially important when detecting the bottom left corner, as the points inside of the ad will produce high correlation results with the corner filter. The bottom left corner detection could also be implemented with a high-pass filter to determine where the ad ends, thus obtaining the height of the ad.

This algorithm was less effective than predicted. The relatively small size of the filter meant that several local areas (such as lights, crowd noise, and players' uniforms) matched the filter, and sometimes produced higher results than the actual corner of the ad. To minimize processing time, the routine was set only to look at the peak point in the image, so these local erroneous points cause a failure of the whole detection algorithm, as suitable bottom left corners or whole ads could be found. Additionally, the small size of the filter meant that noise over top of the corner of the ad or the camera panning such that the upper left corner was out of the frame meant that the ad would be undetected, while

still in frame. With more time, a more flexible detector that used several corners with better handling of invalid points could have been implemented, at the loss of speed performance, but with a significant increase in accuracy and robustness.

COLOR WALK

The algorithm that performed with the best results was partioned into three stages, a binary check of pixels to see if they could be a part of the ad, a "pixel walker" stage which generated geometry information of probable ads, and a final correlation stage which checks that the ad area is actually an ad. In the first stage, the program runs through all of the pixels of the frame and if a particular pixel is a color that is in the ad, the program sets its location to a 1. If the pixel is not a color that is in the target ad, its location is set as 0. In the second stage the program starts at every pixel labeled as a 1, and attempts to “walk” the edges of the ad by moving up and to the right along 1 pixels. To avoid noise overtop of the ad, the routine will skip over a single 0 pixel and continue following 1 pixels. When the walking routines stop, the coordinates are noted, giving us the 4 corners of our potential ad. To save time and processing effort, the program does not return locations that are located within the bounds of another, larger potential ad location. The list of generated co-ordinates is then passed on to a correlation routine, which re-shapes the filter to the size of the potential ad, which is derived from the corner coordinates, and then correlates within this frame to find the maximum correlation. If the correlation peak is high enough, then a replacement is made.

This method was moderately successful. At certain levels of color thresholding only the ad was selected, however the ad would often be ignored. We found that if 2 or

more of the edges of the ad were off of the screen it would be impossible to determine the scale of our potential ad and therefore correlations were poorly done and overlays were problematic. Some problems also arose as a result of color variations in the ad (in the Budweiser ad case, a black shadowing of the lettering, which added contrast to make the words more readable) which would prematurely stop the walkers. This might be fixed with a certain amount of blurring (by making a pixel the weighted sum of itself and its neighbors) before computing the binary map. Excessive blurring, however, may lead to false positives in the potential ad detection phase, and could also lead to inaccuracies in the height and width calculation, leading to poor results in ad overlay.

UNUSED METHODS

A few other methods were contemplated and researched, but not implemented or evaluated. Because of the large correlations we employed (320 x 240 pixel frames and a 55 x 200 pixel filter), a frequency-domain correlator was considered to reduce the number of computations. A full-scale spatial-domain correlator is O(n^4), whereas a frequency-domain correlator is simply O(n^2), with an O(n log n) operation for the fast Fourier Transform. However, shortly into the project we realized that a full-scale correlator was not feasible, for the reasons described above. The correlator algorithm was retained for the limited scope of evaluation at a specific point, and we decided that a spatial correlator would be more efficient in terms of memory usage and computation. The memory usage was particularly notable because the fft routine has to store both the real and imaginary results of the transform, doubling the memory requirements. The frequency-domain correlator for the entire frame also would not solve of the problems

inherent in the spatial-domain correlator, most notably the sensitivity to geometric distortions (such as rotation and scale changes), and the noise factor.

Most other methods were variations on methods we employed. One was the use of a side-edge correlator, similar to the corner correlator we tried. This method was abandoned in favor of the corner correlator because we thought the corner correlator would be less sensitive to scale changes in the filter (as long as the target ad was taller and wider than our corner filter).

Also, a two-dimensional Laplacian high pass filter was considered, from which corner information could be extracted. Based on the poor results of the one-dimensional Laplacian we tried, this method was not implemented either, as it did not appear that it would solve any of the problem we encountered with edge detection. Additionally, it would involve twice as many computations as the one-dimensional filter if it was implemented as a separable filter, and three times as many computations if it was inseparable.

Other methods of parsing the edge-detector output were considered, other than the walk-style method we used. An algorithm of summing the high pass filter output by columns, like the Marlboro pack example in class, was considered, but we found a great deal of crowd, field, and other noise that interfered with any results we would have gotten. A major characteristic of the Marlboro pack example was the fact that the Marlboro pack was the focus of that image, and each result could be linked to a certain feature of the pack. Our frames, however, did not focus on the ad at all, which would have caused the signal to noise ratio to plummet. Background noise in the summation could also not be directly linked to a feature of the ad, as a great deal of similar regular

features in the scenes (crowds, empty seats, other stadium features) could provide similar outputs.

A Hough Transform was considered, but a lack of knowledge on our part and an inability to find examples of actual implementation or base code, as opposed to theoretical conjecture, led to the pursuit of other avenues of line-finding. The efficacy of a full-frame transform after a full-frame filtering is also debatable, in terms of storage and processing time.

DATA TRANSFER

The development of the data transfer routines was initially based on lab3's file transfer structure. However, by the time the project was finished it had changed substantially and boasted a lot more functionality. To handle the data transfer, frames were stored in a variable called *framebuffer* that was eight frames long, which contained memory for a status variable as well as data memory. By implementing an asynchronous transfer as opposed to a synchronous one, we were able to process frames while data was transferring back and forth from the PC and EVM. To achieve this it was necessary to use callback functions and flags to block a second access to the PCI Bus while a transfer was being processed, and to block frame processing code from attempting to work on partially transferred data. The use of separate callback functions for our *getframe* and *sendframe* commands allowed us to keep track of useful statistics such as number of frames received and sent. In order to assure that the PC didn't receive a second PCI request while it was still processing the last one, it was necessary to put small delay loops in the callback functions of about 10,000 instructions.

MEMORY ISSUES

We found that by moving the code from external to internal memory we could improve our code's performance by a factor of 3, going from 3 seconds per frame to about a second per frame. We also observed that this significantly increased the instability of our code (this code crashed the PC on many occasions). We didn't receive the blazing performance enhancement that one would expect from going "on chip" because even though our code sat on the chip, the vast majority of our data was still stranded in the RAM because of its size. Although we didn't attempt to do it, it may be possible to rectify this issue by paging parts of a frame into "on chip" memory at a time.

DEMO/RESULTS

In the end, what we ended up with and what we demoed was a system using the color walk algorithm. The result was not as good as we had hoped, since the system picked up a number of false-positives and did not do a fair job of completely covering the ad all of the time. Our system had problems when the ad was half off the screen, and we decided that for a system of this nature to work, meaning, with no prior knowledge of camera positioning and angles, at least two corners of the ad must be on screen at all times. Our algorithm eventually averaged about 1 frame per second, however the time to complete a specific frame could range from 0.1 seconds to about 5 seconds based on how many potential ad regions were detected and had to be evaluated with a correlation.

POSSIBLE IMPROVEMENTS

Several improvements could be made to the program in general and to specific routines to increase performance, accuracy, and efficiency. Each of the detection routines can be improved and tweaked to provide better performance (or, indeed, any positive performance at all), and the flow of the program could be improved for better coherency and flexibility.

Currently, the dimensions and locations of the images and filters are hard-coded by #defines into the PC code and the C67 code. For more robustness, these should be variables, defined by the user in an actual user interface. This becomes a direct issue when creating a marketable product, rather than a proof-of-concept project as we created this semester.

The asynchronous transfers could be handled by a more complex routine, one that would buffer both incoming and outgoing frames and administrate the depth of the buffers based on the speed of the PCI bus and the computations for each frame. The implementation of this would require more careful memory management, however, as the more complex callback routine for the asynchronous transfers would be interrupting frame computational routines.

Although transfer rates over the PCI bus were not a limiting factor in our project, in future versions, this could become an issue. We measured a peak throughput of about 12 frames/second over the bus with no computation on either side. To bring this to full speed video (30 frames/second), the transfer rate must be increased, or the frame size must be reduced. To this end, PC-side processing could take a significant portion of the load off of the C67. A small motion-detection algorithm on each frame in the AVI

stream could bring improvements by minimizing the area over which the processing routines have to search, bringing significant increases in processing speed, and possibly bringing the EVM side closer to real-time performance. The PC could send only then the changed window or change information about the previous frame, with the EVM returning only the changed information as well, reducing the data transfer on the PCI bus.

Alternatively, another data format could be used. We used uncompressed AVIs to reduce our processing requirement in terms of time and complexity, but a slightly compressed format could bring a reduction in transfers with only an incremental increase in processing. This could become an important step for future projects if significantly faster processors are used (such as the 800 MHz DSP rumored to be in development). A much faster processor, coupled with a better find-and-replace algorithm, would have time to do light decompression on the C67, meaning that a compressed format, such as MPEG, could be used to reduce data transfer loads.

Other information from the AVI format could be used to speed processing as well. Should a later project become hindered by PCI bus speeds, the timing information that can be stored in the AVI stream could be used to simulate real-time performance for displays. If the display system is not capable of 30 frames per second, the AVI display program will only display the frame that is due at that time. Using this information, the program could only find-and-replace on those frames that would end up being displayed. This, unfortunately, could not be used to resave the AVI, as a faster-displaying system would show the gaps in the changed AVI.

Better memory management by paging parts of the large frames into the C67 on-chip memory could greatly reduce computation time. Our early attempts at this were

overly ambitious, and ran into problems with paging too much memory onto the chip, causing other variables to be overwritten, and leading to instability of the entire program, as several program-level status variables (for example, the index of the frame being worked on) were stored on-chip. A more conservative paging routine could provide a performance enhancement, particularly for routines that access large sections of the frame often, such as the correlation function and the first phase of the color walk routine.

In the processing routines, changes could be made to make them work better and more efficiently. An automatic rule creator for the Color Walk algorithm could remove some of the guesswork from the manual rule creation, and better account for the variations of color in the ad. Creating and getting this rule creator to work would be nearly an entire project by itself in pattern recognition and unsupervised classification techniques. Although it could possibly be very computationally intensive, it would fortunately only have to be run once for each ad the user wanted to remove.

The color walk algorithm would also benefit from optimization of the walking strategies it used. The current rudimentary system of up-and-right, with a one-pixel jump, only allows for minor variations in the ad due to shading or shadows, noise or overlay by another object, unnoticed colors in the ad, or geometric distortions. By incorporating an intelligent jumping algorithm, that could keep track of the umber and size of jumps already made and change the jump strategy based on that data (like a moving-average filter), the efficacy of the entire algorithm could be increased. Additionally, modifying the thresholds for the color quality evaluation in the first stage of the routine, or modifying the threshold of the final correlation evaluation, could lead to better results for the detection algorithm. Although some calculation is possible based on

a statistical computation of the color compositions and variations in an average frame, the best thresholds must be determined empirically, by a trial-and-error process. A longer debugging and finalization cycle would be necessary for this step to take place.

Some morphological techniques, particularly a closing operation, implemented by an erosion operation followed by a dilation operation, could smooth the colors and boundaries of the ad in the frame, so that operations such as the color walk and edge detection would better find the ad amidst noise on the boundaries of the ad. The Closing operation would particularly help the edge detection algorithm, by eliminating isolated points (helpful for removing high-frequency crowd noise), and closing gaps in the vertical edges, eliminating noise there.

We ran into problems that appeared to stem from a memory problem regarding some floating-point operations, which hindered our ability to rotate the filter or replacement ads. Addressing this problem would allow for better results from the final correlations and for more realistic replacement of the ad in the frame. Additionally, when the ad was replaced at an angle, noise, such as players or shadows could be picked out from the old ad *in situ*, and overlaid back upon the replacement ad, creating a more realistic appearance.

Also, there are likely several other algorithms we did not think of, realize, or consider feasible that another, later group could implement upon our project to create a better and more marketable system. Although the task required is a difficult and complex one, our project has demonstrated that with appropriate hardware and techniques, the possibility of replacing ads from outdoor scenes is indeed possible.

Improving this possibility is the incorporation of camera or stadium information into the search algorithm. If the location of the camera is known, the program would know precisely where to look for the ad, based on camera movement information. As camera movement in sports broadcasts in generally linear, with no strange angles or movements to consider, a simple frame-motion-detection algorithm could be implemented. With this knowledge, the search function could be completely removed, as the location of the ad would be precisely known. Although this reduces the robustness of the system (the stadium and camera information must be known beforehand, and provided to the system), it would certainly increase the performance, and thus the marketability of the product.

REFERENCES

http://msdn.microsoft.com

http://www.gamedev.net/reference/programming/features/avifile/

http://www.graphics.cornell.edu/~hector/rgbe2avi/avi.html

Lecture notes for 18-798, Image and Video Processing, taught by Professor Tsuhan Chen.

Introduction to Fourier Optics, text for 18-793, Optical Image and Radar Processing.

Lecture notes for 18-793, Optical Image and Radar Processing, taught by Professor David Casasent.

Notes for 18-551

TI Manuals for Memory allocation information.