EULERIAN VIDEO MAGNIFICATION WEBCAM APPLICATION

Spencer Yue

The University of Texas at Austin

ABSTRACT

The goal of this project is to reproduce the results of Wu et al. in a real-time web application which takes input from the user's web camera [1]. The original method of spatial filtering step using a Laplacian pyramid is replicated without modification, as is the IIR time-filtering method for motion magnification. The ideal DCT time-filtering mode for color magnification is adapted to use only a window of the last 2 seconds of input to accommodate the real-time requirement. In reproducing these results as a web application, it is hoped that the widest possible audience can be exposed to the ideas and possibilities presented by this impressive video filtering technique. The project application website can be visited at https://spenceryue.github.io/videomag/.

Index Terms— video, magnification, WebRTC, WebAssembly, real-time

1. INTRODUCTION

Video filters are commonly used in consumer applications for entertainment and recreational purposes. Often in these contexts there is not a strong underlying need to apply a video filter, but rather filters are merely considered a "feature" or "enhancement." More functionally-oriented video filtering algorithms remain largely unnoticed or inaccessible to the public due to barriers in technical literacy or simply a lack of "presence" for the work. As a first step in exploring the possible applications of a new and capable algorithm, focus should be placed on exposing the idea to as many creative minds from varying domains as possible, since rarely will an idea mature in isolation. The Eulerian Video Magnification algorithm is one instance of such a functionally-oriented and saliently capable video filtering technique which would benefit from a more accessible representation for the public viewing.

There are immediate applications which would benefit from the algorithm's capabilities. These include the applications suggested by the original authors through their choice of demo videos: baby monitoring, pulse detection, and visualization of sound in instruments [1]. Natural additions to these areas include analysis of muscle movements for athletes, validating videos of famous people speaking to see if a video has been doctored by checking for a regular pulse (see Suwajanakorn et al. for a convincing example of a synthesized video of Obama speaking), and revealing moderately fast motions on devices where slow-motion capture is not possible [2].

One of the more intriguing potential applications, however, is in revealing human micro facial expressions, a concept from the domain of psychology. These fleeting signs of hidden emotion are as relevant to psychologists as they are to law enforcement and common people. Instead of relying on a more capable camera which supports recording at high frame rates (see Polikovsky et al.), Eulerian Video Magnification enables the direct visualization of micro facial expressions in typical consumer-grade hardware, such as a webcam [3]. This project's website demonstrates this capability in the microexpression.mp4 sample video. (Note that the video clip is not of a real-life scene but instead is of actors replicating micro facial expressions in a physically plausible manner.)

This project primarily seeks to build a better "presence" for the original Eulerian Video Magnification (henceforth EVM) work in an accessible, interactive online web application. One of the claims in in Wu et al. is that the Eulerian Video Magnification algorithm can run in real-time [1]. However, the accompanying MATLAB code provided by the authors did not exercise or demonstrate this capability and instead only processed pre-recorded video segments in an offline fashion. This project hopes to make evident that the techniques to implement EVM are simple, efficient, and capable of enabling real-time applications on devices with limited hardware and constrained processing power. The theoretical basis for the algorithm is also relatively straightforward, which would enable those potentially interested in applying the algorithm to more easily reason about the algorithm's limitations and applicability to a given context. As an auxiliary contribution, this project also explores the feasibility and limitations of porting native desktop programs to the browser using the WebAssembly web standard.

1.1. Related Work

There are several existing repositories on GitHub devoted to reproducing the results of EVM. (A quick Google search of "Eulerian video magnification GitHub" will return them.) The majority are desktop applications and a couple are Android applications (for detecting pulse only). This application differs by targeting the web platform which removes a layer of encumbrance for the interested as no assumption of user device is assumed and installation is unnecessary.

3. SUMMARY OF EVM ALGORITHM

As this project does not attempt to contribute to the theory of the original algorithm, only a brief review of the derivation will be given.

The premise of the analysis is that changes (in motion and color) in a video can be modeled as translations of cosines of the component spatial frequencies from frame to frame. The goal of EVM is to increase the corresponding phase shift of these component cosines associated with a translation.

Using the terminology of the source paper, given an image frame I and our model assumption of translational motion, we have $I(x, t) = f(x + \delta(t))$ for some displacement $\delta(t)$ with $\delta(0) = 0$. We desire synthesizing the result $\hat{I}(x, t) = f(x + (1 + \alpha)\,\delta(t))$ for some positive amplification factor α. (We can let x and $\delta(t)$ be two-dimensional vectors or scalars in the analysis without special consideration.)

Taking a first order Taylor approximation of the frame in space, we have $I(x, t) \approx f(x) + \delta(t)\frac{\partial f(x)}{\partial x}$ (*i.e.* equation 2 in the source paper). Observing that from the time perspective, the $f(x)$ term is part of the DC component, we can filter it out with a bandpass time-filter to get $B(x, t) = \delta(t)\frac{\partial f(x)}{\partial x}$ (equation 3). Next, we add the scaled $\alpha B(x, t)$ back to $I(x, t)$ to get an approximation for desired result:

$$\hat{I}(x, t) \approx I(x, t) + \alpha B(x, t) = f(x) + (1 + \alpha)\,\delta(t)\frac{\partial f(x)}{\partial x}$$

(equations 4 and 5). Now taking $f(x)$ to be a component cosine with spatial frequency ω, we can derive an upper bound for α by comparing the above approximation for $\hat{I}(x, t)$ to the actual $\hat{I}(x, t)$ expected by initial assumptions. (Take $\beta = 1 + \alpha$ as in the source paper.)

$$f(x) + \beta\,\delta(t)\frac{\partial f(x)}{\partial x} \approx \hat{I}(x, t) = f(x + \beta\,\delta(t))$$

$$\Rightarrow \cos(\omega x) - \beta\,\delta(t)\sin(\omega x) \approx \cos(\omega x + \beta\,\omega\,\delta(t))$$

$$\Rightarrow \cos(\omega x) - \beta\,\delta(t)\sin(\omega x) \approx \cos(\omega x)\cos(\beta\,\omega\,\delta(t)) - \sin(\omega x)\sin(\beta\,\omega\,\delta(t))$$

Comparing terms in the last equation, we see that we would like $\cos(\beta\,\omega\,\delta(t)) \approx 1$ and $\beta\,\delta(t) \approx \sin(\beta\,\omega\,\delta(t))$. From here, reasoning that the sine is the limiting error and holds so long as the argument satisfies $\beta\,\omega\,\delta(t) \leq \frac{\pi}{4}$, we arrive at the final bound (after substituting $\lambda = \frac{2\pi}{\omega}$):

$$(1 + \alpha)\,\delta(t) \leq \frac{\lambda}{8}$$

The bound on the amplification factor is applied on a per-component frequency basis to the video frame's component cosines.

With this the idea for EVM entirely is summarized. The simplicity of the algorithm ideas makes it potentially more accessible and useful to a more general audience.

4. SPATIAL FILTERING: LAPLACIAN PYRAMID

One of the other secondary contributions of this project is a JavaScript implementation of the Laplacian pyramid. To my knowledge (based on a Google search for "Laplacian pyramid JavaScript," the first two pages of search results) there does not exist another publicly available full implementation of this image processing technique in JavaScript. The perceptualdiff project includes a partial Laplacian pyramid implementation in JavaScript, but it has several shortcomings: It does not include the down-sampling and up-sampling steps, has a hard-coded blur kernel, and does not include the reconstruction subroutine [4]. In contrast, this project includes all the Laplacian pyramid features included in the matlabPyrTools MATLAB library used by the source paper authors in their demo code [5].

The Laplacian pyramid is an intuitively attractive algorithm which can be essentially summarized as a process which separates an image into different scales by subtracting each successive layer's blurred version from itself and down-sampling. The difference of Gaussians with standard deviation differing by a factor of approximately 1.6 closely approximates the Laplacian of Gaussians commonly used for edge detection, hence the name of the technique.

Reconstruction is done by starting from the deepest layer of the pyramid and performing an up-sample and blur before adding to the previous layer, repeating until the first layer is reached. As an implementation detail (this is also mentioned in the documentation of matlabPyrTools), since some information is permanently lost in the down-sample step during construction, the actual sequence of operations to produce one pyramid level should be implemented as:

(1) blur **I** to get **J**

(2) down-sample **J** to get **K**

(3) up-sample and blur **K** and subtract from **I** to get the new pyramid level **L**.

Repeat from step (1) with **I** := **K** until the input **I** is smaller than the blur kernel.

While matlabPyrTools includes the option to use different blur kernels on the down-sample and up-sample steps, the EVM source authors only used the default 5 element binomial blur kernel. This project allows for the user to change the blur kernel size interactively to allow for additional freedom of user experimentation.

In this project, a fixed down-/up-sample factor of 2 in each direction is used, which leads to no more than 4/3 the memory usage of an original video frame per pyramid. Multiple pyramids are allocated in memory at application initialization either for direct use in a current frame's spatial filtering step or to store previous pyramid results for the time filtering step.

5. TIME FILTERING

The source paper uses three different time filters: the difference of two IIR exponential filters, a first-order Butterworth filter, and the ideal DCT-based filter. Only the difference of IIR exponential filters and the DCT filter are implemented in this project since the results of the IIR and first-order Butterworth filters were experimentally comparable, and both served the same purpose of motion magnification (though the source paper mentions that IIR filters can also be used for color amplification). The modification of windowing over approximately the last 2 seconds of input was made to the DCT to try to accommodate the real-time requirement.

5.1. Difference of Exponential IIR Filters

The exponential filter is a simple low-pass filter with the following difference formula:

$$y[n] = \text{decay} * y[n - 1] + (1 - \text{decay}) * x[n]$$

where y is the filtered signal and x is the input signal. Typically, the coefficient in front of x is denoted α and $(1 - \alpha)$ is used as the coefficient for y (*i.e.* the reverse of how the equation above is written). In any case, the coefficients are nonnegative and sum to one. Given a target cutoff-frequency, the decay coefficient can be calculated as:

$$\text{decay} = e^{-2\pi\, f_c / \, f_s}$$

where f_c is the cutoff frequency and f_s is the sampling rate. (The cutoff frequency by convention is where the input signal power gain is half.)

In this project, the decay for the two IIR exponential filters is recalculated per rendering cycle based on the estimated frame rate (FPS) in order to provide more accurate results than assuming a fixed sampling rate. All time filters accept f_{low} and f_{high} parameters to specify the desired passband. In the case of the difference of exponential filters, these parameters correspond to the cutoff frequencies of the two IIR exponential filters to be differenced.

5.2. Windowed-DCT Ideal Filter

The ideal DCT time filter performs the DCT on a window of the past inputs and zeros out undesired coefficients. Conceptually, the DCT is no more than the DFT aided by the simplification that with a real input signal the DFT coefficients indexed by $i = 0, \ldots, N - 1$ will be conjugate pairs about the index $(N - 1) / 2$ if N is even and N/2 if N is odd. For convenience we always pick N odd so that every coefficient besides DC has a conjugate pair.

For the DCT on a window of the last N frames (where N is odd), we use N frame-sized arrays to hold the coefficient data for the DCT. The first array stores the DC values, the next $(N - 1) / 2$ arrays store the real component of the correspondingly indexed DFT coefficients, and the final (N – 1) / 2 arrays store the imaginary component of the first DFT coefficients (those at indices $i = 1, \ldots, (N - 1) / 2$). This representation (keeping the real and imaginary components of half the DFT coefficients) allows for a simple $O(N)$ update formula for each pixel rather than an $O(N \log N)$ or $O(N^2)$ re-computation of the coefficients at each new time step.

The update formula which follows can be geometrically understood by imagining in the complex plane for a given DCT coefficient "rotating the wheel" of vectors in the dot product by one angular-speed turn while replacing the oldest vector's magnitude by that of the newest value. There is an analogy to a wheel because the dot product is between the signal vector and the N^{th} roots of unity which lie in a "wheel" around the unit circle spaced equally by the angular speed.

Let $Y[i]$ represent the i^{th} DFT coefficient of the length-N real-valued signal y indexed from $k = 0$ as the oldest to $k = N - 1$ as the latest. $y[N]$ is the newest value yet to be processed. Then to update $Y[i]$ for a time step, we use:

$$Y[i] := (Y[i] - y[0] + y[N])\, W_i^{-1}$$

where $W_i = e^{-2\pi i j / N}$. The explicit matrix form of this update rule is as follows:

$$\begin{bmatrix} Y[i] \\ Y[N-i] \end{bmatrix} = \begin{bmatrix} \cos\omega_i & -\sin\omega_i \\ \sin\omega_i & \cos\omega_i \end{bmatrix} \begin{bmatrix} Y[i] - y[0] + y[N] \\ Y[N-i] \end{bmatrix}$$

where $\omega_i = 2\pi i / N$. This update is only performed on the half of DFT coefficients which are kept, but since real and imaginary components are stored, all N frame-sized arrays are updated. Values of $y[k]$ from $k = 0$ to $k = N - 1$ must also be stored and shifted by one index-place to the left for use in future update computations.

The inverse DCT is used to recover $y[k]$ with the following formula:

$$y[k] = \frac{Y[0]}{N} + \frac{2}{N} \sum_{i=1}^{(N-1)/2} \begin{bmatrix} \cos \omega_{ik} & -\sin \omega_{ik} \end{bmatrix} \begin{bmatrix} Y[i] \\ Y[N-i] \end{bmatrix}$$

where $\omega_{ik} = 2\pi ik/N$. The filtering step occurs during recovery by treating $Y[i]$ as zero if $i < N(f_{low}/f_{sample})$ or $i \geq N(f_{hig}\ \ /f_{sample})$.

6. RESULTS

The results of this project can be measured in two ways: (1) based on the requirements listed in the initial project proposal and (2) based on the demonstration results alone.

6.1. Proposal-Based Evaluation

First, in considering just the goals listed in the proposal, this project was remarkably successful for its defined scope. Each of the goals targeted in the initial proposal was satisfied.

The initial challenge was to acquire webcam input and display it using the WebRTC and Canvas Web APIs. Mozilla Developer Networks documentation and tutorials were invaluable for this and all subsequent web-application-relevant aspects of the project. One web development lesson encountered at this point concerned the Cross-Origin-Resource-Sharing (CORS) mechanism for accessing cross-domain content from JavaScript. This browser security feature prevented the Canvas API from rendering content from static resources from my local file system. The solution was to serve the project directory from a local web server. The Python files in the project repository are devoted to this task.

The next major hurdle was setting up the Emscripten compiler, learning about the WebAssembly memory model, and interfacing with Emscripten-compiled C functions from JavaScript. Emscripten is a compilation tool built on top of the LLVM compiler which allows porting programs written in a variety of languages to be run in JavaScript. The tool outputs JavaScript-compatible programs in either the asm.js or (the newer) WebAssembly format. Because Emscripten-compiled programs are still managed by the browser and not the user's operating system, it was necessary to gain some intuition about how memory is represented and accessed via JavaScript. Fortunately, this proved straightforward as memory is simply represented as a compile-time configurable number of bytes allocated in a JavaScript TypedArray, which can be accessed from both the JavaScript and the program perspectives as a contiguous array.

Another notable difficulty was achieving real-time frame rates. Initially, both with and without using WebAssembly, performing the pyramid construction spatial filtering step lowered frame rates to between 5 and 10 FPS. WebGL was briefly entertained as an alternative to both WebAssembly and JavaScript implementations due to the promise of faster speeds on the GPU, but it was suspected that WebAssembly had not yet reached its peak performance, so this alternative was dismissed.

Indeed, a solution to performance bottlenecks was found after combined use of several debugging tools including assertions, browser developer tools, and memory corruption validation checks. Assertions helped detect invalid array index accesses, which do not cause fatal errors or warnings in JavaScript but do negatively affect performance. Similarly, memory corruption validation in the form of allocating sentinel bytes following calls to malloc() and setting them to known values such as 0xdeadbeef helped detect array index violations on the WebAssembly side. The largest performance gains, however, came from using the Chrome Developer Tools performance profiler to identify bottlenecks at the functional level. The identified solution was to remove all dynamic allocations of intermediate arrays during the application's execution, instead allocating all arrays once at initializing and resizing these arrays based on changing needs. These combined efforts brought the spatial filtering computation up to around 50 FPS.

One other significant unexpected difficulty arose in completing this project. The source authors relied on the matlabPyrTools library for the crucial spatial filtering step for their demo program in MATLAB, but no such library existed for C. MATLAB C/C++ automatic code generation tools were briefly considered. However, it proved the most straightforward (and educational) to simply read and reimplement this library than to imagine ways to save work.

Finally, the project was successfully formatted with HTML and CSS and deployed to the web with GitHub Pages. Considerable effort was devoted to ensuring the application would be visually appealing and the intuitive to control so that a visitor would find her experience on the site memorable.

6.2. Judging Algorithm Performance

The IIR time-filtering mode replicated the results of the source paper perfectly and showed great applicability when used on live video captured from the webcam. Performance is above 30 FPS (tested in Chrome browser on a laptop running Windows 10 with 7th Generation Intel i7 CPU). The IIR filtering mode produces unambiguous live motion magnification results for a variety of lighting conditions with little to no tuning. The source authors mentioned being able to use IIR filters to design a color magnification filter as well. However, they did not mention any details of how this would be done, nor was a supplementary demonstration provided.

Testing the feasibility of using non-ideal filters such as the exponential IIR and higher-order Butterworth to perform real-time color magnification would be an interesting topic for future study. (The source authors did not have any demonstrations using Butterworth filters higher than first-order and for purposes other than motion magnification.)

The ideal DCT filter implementation in this project performed with inconsistent results. For known lighting conditions such as in the source paper demo videos, parameters can be tuned to give results which replicate those provided by the original authors. (See face.mp4 on the project website for a good example.) For live input, the project can fail to produce the desired color magnification for unintuitive reasons. After much experimentation tuning the existing exposed parameters available for the user's control in the application consistent results could not be obtained. Parameters unexposed to the user were also varied including the DCT window length, the lag of the filtered signal (whether to recover the most recent value or some value from several samples in the past), the extension of the width of the passband (beyond the user-specified f_{low} and f_{high} values), and the amplification factor and the pyramid blur level to use for spatial filtering. While real-time results could not be consistenly obtained, the success in reproducing the source demo videos indicates that this project faithfully replicates the color magnification capabilities of EVM as specified by the source authors. However, further experimentation is necessary as results are still too unreliable to be of use to the casual user.

The interactive performance of the ideal DCT filter implementation is below the 30 FPS goal but still appears somewhat smooth. (Frame rates hover around 15 FPS in the same testing conditions as mentioned before.) WebAssembly has been disabled for this implementation (without control by the user) due to unaccounted for differences from the JavaScript implementation, likely arising from different levels of floating point precision in the two execution environments. Interactive performance does not improve dramatically even with WebAssembly turned on as the bulk of computation is still at the spatial filtering rather than time filtering step.

7. CONCLUSION

This project exposes the Eulerian Video Magnification algorithm of Wu et al. to the public in the form of a widely accessible web-platform interactive webcam application [1]. It achieves real-time performance for motion magnification and replicates results of the source work faithfully. Nearly all the tunable algorithm parameters have been exposed through the carefully designed user interface for direct experimentation. Attention has been given to ensuring the algorithm incorporates changes to parameters in an online fashion (with exception of the ideal DCT filtering mode which requires rebuffering the input for 2 seconds). This project establishes a presence in the public view for the functionally-oriented and widely applicable EVM video filtering algorithm. Possible applications such as identifying micro facial expressions and detecting falsified recorded statements from famous speakers are hinted at by the choice of demo videos on the application website. Finally, all the source code is made publicly available on GitHub for the future educational benefit of anyone interested in replicating these results using Web platform technologies like JavaScript and WebAssembly [6].

11. REFERENCES

[1] Hao-Yu Wu, Michael Rubenstein, Eugene Shih, John Guttag, Frédo Durand, and William T. Freeman. "Eulerian Video Magnification for Revealing Subtle Changes in the World," *ACM Transactions on Graphics*, ACM, New York, Volume 31, Issue 4, Article 65.

[2] Supasorn Suwajanakorn, Steven M. Seitz, and Ira Kemelmacher-Shlizerman. "Synthesizing Obama: learning lip sync from audio," *ACM Transactions on Graphics*, ACM, New York, Volume 36, Issue 4, Article 95.

[3] Senya Polikovsky, Yoshinari Kameda, and Yuichi Ohta. "Facial micro-expressions recognition using high speed camera and 3D-gradient descriptor," *3rd International Conference on Imaging for Crime Detection and Prevention (ICDP 2009)*, London, pp. 1-6.

[4] perceptualdiff (GitHub repository)
https://github.com/ablage/perceptualdiff

[5] matlabPyrTools (GitHub repository)
https://github.com/LabForComputationalVision/matlabPyrTools

[6] videomag (GitHub repository)
https://github.com/spenceryue/videomag