Research

iMedPub Journals
http://www.imedpub.com/

American Journal of Computer Science and Information Technology

2019
Vol.7 No.1:32

DOI: 10.21767/2349-3917.100032

A Novel Approach to Vector-based Video Compression through Motion Tracking

Kanav Kalucha* **and Abhi Upadhyay**

Mission San Jose High School, California, United States

***Corresponding author:** Kanav Kalucha, Mission San Jose High School, California, United States, E-mail: kanav.kalucha@gmail.com

Received Date: January 02, 2019 **Accepted Date:** January 21, 2019 **Published Date:** January 28, 2019

Citation: Kalucha K, Upadhyay A (2019) A Novel Approach to Vector-Based Video Compression through Motion Tracking. Am J Compt Sci Inform Technol Vol.7 No.1: 32

Copyright: ©2019 Kalucha K, et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract

As the age of technology progresses, the demand for video with higher resolution and bitrate increases accordingly, and as video compression algorithms approach near-theoretically perfect compression, more bandwidth is necessary to stream higher-quality video. Higher pixel resolutions do not change the fact that scaling individual frames using bilinear or bi cubic filtering naturally causes the video to lose detail and quality. In addition, as the number of pixels per frame continues to increase, so does the file size of each frame and ultimately the file size of the fully rendered video. Over time, as file size increases and required bandwidth increases, the cost of hardware systems multiplies, requiring a solution for further compression. Using core concepts of computer vision and Bezier models, this project proposes a method of converting pixel-based frames into a graphical vector format and applying motion tracking methods to compress the rendered video past current compression techniques. The algorithm uses the canny operator to break down pixel-based frames into points and then obtains Bezier curves through taking the matrix pseudo inverse. By tracking the motion of these curves through multiple frames, we group curves with similar motion into "objects" and store their motion and components, thus compressing our rendered videos: adding scalability without losing quality. Through this approach, we are able to achieve an average compression rate of 88% over industry-standard compression algorithms for ten sample H.264-encoded animation videos. Future work with such approaches could include modeling different lighting or shading with similar Bezier splines as well as bypassing pixel-based recording altogether by introducing a method to record video in a directly scalable format.

Keywords: Vector-based video; Higher resolution; Bezier models; Graphical vector; Algorithm;Animation videos

Introduction

When recording any sort of media, specifically videos, all technology today records in a pixel format. The pixel-based format has a myriad of issues we identified and attempted to mitigate when implementing our approach to video compression. When scaled up, pixels do not maintain the image quality and cohesion, and so existing technology, such as focusing and sharpening based on edge and contrast detection, is unsatisfactory [1]. The current, easiest solution to scaling is to simply take pictures with a high enough resolution, but this merely compounds the problem further, as these images still cannot be scaled to even larger screens. As the resolution accelerates, so does the memory required to store the image, and at extremely high resolutions, certain screens such as older phones and laptops do not have enough pixel density to fully display each pixel, simply leaving wasted memory [2]. Traditional video suffers from these same issues plaguing pixel-based images, while also introducing the problem with FPS (frames per second), or bitrate. A high FPS means much smoother playback at the expense of processing power, while a low FPS could mean choppy playback ruining the video. A variable bitrate that could be controlled by the user is not possible, yet would lend utility [3]. Finally, the largest problem with the status quo is of course data storage. As Seagate predicts, by 2020, the world will have produced six zettabytes of data it cannot easily store, and according to Cisco, most of that data being transferred over the internet in the form of video, with YouTube and Netflix accounting for a large chunk [2]. Thus, the demand for greater resolution continues to strain the existing bandwidth, leading us to seek a solution with a high compression rate coupled with adjustable resolution and bitrate.

With our solution, users will be able to input any simple animated video and receive a compressed version of that video encoded with our codec. When processed and compressed, the video we produce would be more detailed and theoretically, infinitely scalable without quality loss and a variable bitrate. We plan on converting every frame of the simple animated video into a vector graphic-based format. These vectors can be resized to any resolution and retain their quality as they are simply a collection of mathematical equations representing the lines and colors in an image [4].

The compression would be efficient because storing vector frames would reduce the file size compared to pixel-formatted frames greatly, as vector graphics store lines and contours, not individual pixels. Next, it would be compressed further as we plan on tracking the motion of different objects in the video and modeling that too with a mathematical equation. From there, we would only have to store the vector of the object and the

change in its motion rather than a separate vector for each individual frame, saving space and processing power.

With our solution, we begin the end of low quality, bloated animated videos and replace it with a more versatile and compressible format.

Methods

We started our project by initially creating multiple sample animations and exported them as a regular video format. The animations included just a simple circle moving across the screen in different ways, whether it is straight, diagonally, or following a curved path.

From these test videos, we began by using the well-known OpenCV library to generate a set of points to represent each frame. We did this specifically by applying mainly the Canny operator over each frame, which would produce a result as shown in **Figure 1** below.

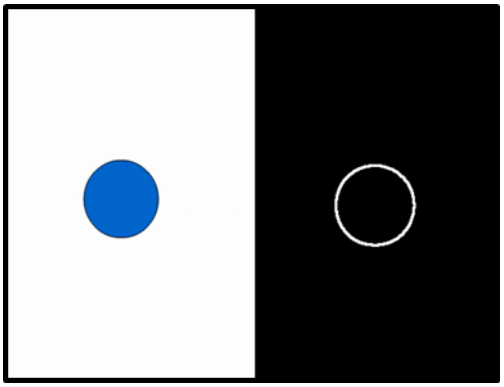

Figure 1: Result after applying the Canny operator.

After taking in a set of points for each object, we were able to calculate the distance between any 2 points and differentiate between 2 different Bezier curves if their endpoints were a certain threshold away. From here, we took the set of points for each curve and fitted a cubic Bezier curve to it by working backwards and generating the control points for each curve. This was done through taking the pseudoinverse of a matrix of points and dividing it by multiple t (linear interpolation) intervals (Pastva). At this point, we now have access to the full Bezier curve for each object in each frame.

The next step included parsing this Bezier curve into an SVG or vector format. Fortunately, the modern day SVG formatting allows us to directly model Bezier curves by encoding it in XML. A sample of the code is shown above in **Figure 2**. We have now successfully completed the first portion of our project by converting all frames of a video into a vector based format.

```
<svg>
    <path d="M211.61 222.49 C
    231.32 207.80, 230.78
    170.30, 213.16 " stroke="
    black" fill="#0064ca"/>
</svg>
```

Figure 2: Sample of our SVG formatting

As for the second portion, motion tracking, we were able to take the set of vector curves received after applying the Canny operator and matrix pseudoinverse on one frame and track individual curves across frames because their change in position was under a predefined threshold. Then, we grouped sets of vectors that moved similarly into "objects" and took the weighted average of all the points in each object to obtain an average point for that specific object in that specific frame. We were then able to model the motion of that averaged point for each object over the next multiple frames using another Bézier curve through a least-squares regression. The motion of the average point we tracked for one of our test cases, including the whole set of points in the beginning frame and the end frame of the constant motion is shown as a graph below in **Figure 3**.

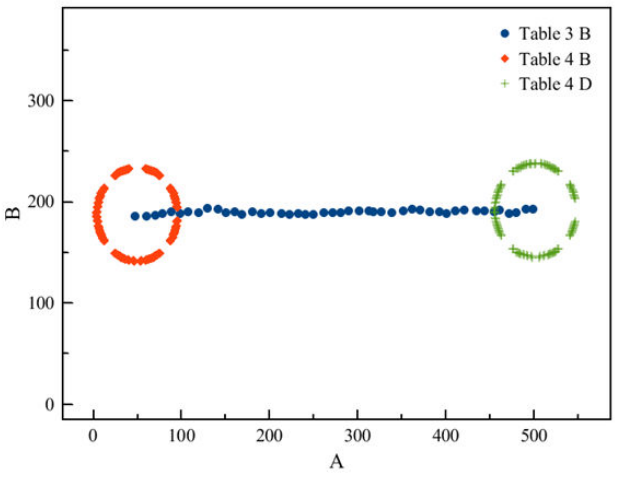

Figure 3: Graph displays motion of the average point of an object moving in a constant direction.

Taking in all this data, we then formatted a codec that could represent the full video, including the motion of each object, in correspondence with their respective SVG files, modeled with the Bézier curve. The codec simply stores the SVG files for the frames it cannot track motion in, and then stores the first frame with a trackable object, followed by the motion line for the object over frames *i* to *j*. A final sample of our codec is shown in **Figure 4**, modeling a trackable object from frame 13 to frame 53, with a motion line given as a cubic Bezier curve.

```
-12.SVG
-13.SVG
        13, 53
        168 239, 194 239,
        211 222, 245 267
-54.SVG
-55.SVG
-56.SVG
```

Figure 4: *Our final codec formatting.*

Results

Compiling and analyzing the output from our compression process for our test cases, we found that the compressed file we generated had a size of around ~25 kb, compressed down from a H.264 video of ~216kb. This is a compression rate of around 88% from the current day standard for pixel-based video. It is, however, important to note that our test cases involved a simple ball that was easily modeled with Bezier curves moving in different directions. We've shown a sample (**Figure 5**) of the quality of video we're able to achieve with the compressed vector version and we can see that this sample has a much higher quality compared to the original low-quality frames encoded with H.264.

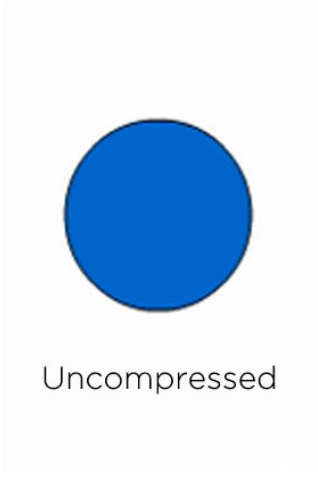

Figure 5: *Depiction of quality of identical frames, before and after compression*

Based on our process for obtaining compressed video, we noticed that simply converting frames of a pixel-based video into a vector format already compressed the video by ~69%. When applying our motion tracking algorithm to full vector based frames, we were able to compress the fully vector formatted video another ~61%, resulting in our final product, compressed around 88% overall.

With this data, we were able to generate a breakdown of how much storage each portion of our final product based on our first test case. Each vector formatted frame took up around 1kb of storage, whereas each pixel formatted frame took up around 3kb of storage. The structure for our codec, which held information about the motion of different objects totaled to around 4kb of storage, all for our first test case. This meant that we were saving space on every single frame and allowed us to compress these videos at an 88% compression rate.

Next, considering processing power, the complexity and runtime of our algorithm depends on the complexity of the video we're dealing with. Constructing Bezier curves from a collection of points requires taking the pseudoinverse of a matrix of points, which is possible in $O(n^{\omega})$(Keller-Gehrig). This must be done for m points per frame, where m is the average number of points per frame. Assuming the number of vectors per frame is linear with respect to the number of points per frame, with n frames, the algorithm has a complexity of $O(n \cdot m \cdot m\omega)$, and using Le Gall's upper bound of 2.3729 on ω, the matrix multiplication constant, we have an overall complexity of around $O(nm^{3.3729})$(Le Gall).

We now have the ability to convert simple animated videos into a vector based format and compress them using motion tracking. However, from this point, we would need to begin development on video playback software so that our codec could be unpacked and could playback each SVG frame as a regular video would. This would include simply looping through each frame at a specified frame rate, and moving objects based on the Bezier curve that represents the motion of that object. We also noted that since each object follows its own motion tracking line over several frames, we could sample the line as often as we'd like, meaning we could control the bitrate, or the FPS of the video.

Overall, for our test cases, we were able to achieve a compress rate of ~88%, a lot larger than what we originally expected at 55% compression.

Discussion

Our work with the compression of vector-based videos has led us to believe that compression of current-day video is still possible to great extents, even as we approach the theoretical limit of pixel-based compression [2]. However, our work has only been tested in-depth with simple animated videos. In the future, we would have to conduct more research about how we can incorporate this approach with complex animations that involve multiple objects and complicated motion paths. We would also have to think about how the lighting on and shading of these different objects affects our results [1]. Lastly, we have come to the conclusion that it's best to convert animations into a vector-based format rather than convert real life videos into these formats. It is difficult to convert a real life still into multiple shapes and objects with finite colors and clear contours. It may be possible in the future to completely bypass the pixels in the first place by introducing a method of recording video directly in a scalable format rather than pixels, which could potentially be its own research project.

© Under License of Creative Commons Attribution 3.0 License

Acknowledgements

We'd like to thank the MIT THINK team for providing mentoring and guidance to turn our proposal into a finished project.

References

1. Patterson JW, Taylor CD, Willis PJ (2012) Constructing and rendering vectorised photographic images. J Virtual Reality Broadcasting 9:3.
2. Athow D (2015) The data capacity gap: why the world is running out of data storage. IT Proportal 9.
3. Ortega A, Ramchandran K (1998) Rate-Distortion Methods for Image and Video Compression: An Overview. IEEE Signal Process Mag 23-50.
4. Hussin M, H0ussain MZ, Saddiqa M (2015) Circular Approximation by Trigonometric Bézier Curves. Int J Mathematical, Computational, Physical, Electrical Computer Engineering 9-1.