Title
Streaming Aerial Video Textures

Journal
Scientific Visualization: Advanced Concepts, Dagstuhl Follow-Ups, 1

Authors
Co, Christopher S.
Duchaineau, Mark A.
Joy, Ken

Publication Date
2010

Peer reviewed

Streaming Aerial Video Textures

Christopher S. Co [1], Mark A. Duchaineau [2], and Kenneth I. Joy[3]

[1] Google, Inc.

[2] Lawrence Livermore National Laboratory
Livermore CA

[3] Institute for Data Analysis and Visualization,
Computer Science Department,
University of California, Davis

Abstract. We present a streaming compression algorithm for huge time-varying aerial imagery. New airborne optical sensors are capable of collecting billion-pixel images at multiple frames per second. These images must be transmitted through a low-bandwidth pipe requiring aggressive compression techniques. We achieve such compression by treating foreground portions of the imagery separately from background portions. Foreground information consists of moving objects, which form a tiny fraction of the total pixels. Background areas are compressed effectively over time using streaming wavelet analysis to compute a compact video texture map that represents several frames of raw input images. This map can be rendered efficiently using an algorithm amenable to GPU implementation. The core algorithmic contributions of this work are methods for fast, low-memory streaming wavelet compression and efficient display of wavelet video textures resulting from such compression.

1 Introduction

Aerial systems are being devised that deploy billion-pixel cameras, providing high-resolution wide-area video at several frames per second. These new cameras produce data streams that are a factor of one hundred larger than previously deployed in aerial imaging systems. In this setting, the main challenges are twofold:

- to transmit this huge pixel stream to the ground over available wireless bandwidths, at best about 20 megabits per second; and
- to display this huge image stream visualized over 3D terrain models, by extending the best known view-dependent display optimization techniques to handle data that is not only large spatially, but large temporally.

Current state-of-the art static image and video compression methods are at best capable of a factor of 100 compression while keeping high image quality to perform automated mover detection and analysis. An additional factor of 10 to 100 compression is needed. At the same time, the process of compression must occur before transmission to ground due to restricted bandwidth, payload storage, and power available on the aircraft.

To achieve the thousand to ten thousand times compression needed, alternatives to conventional image and video compression strategies are needed. We make two observations about wide-area aerial video that enable some specialization in compression techniques:

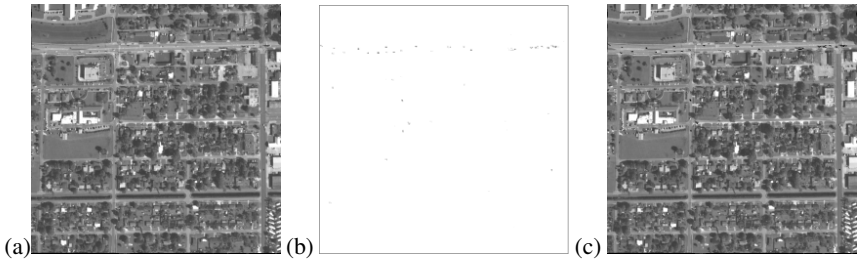


Fig. 1. For aerial images, such as the one shown in (a), we achieve high levels of compression by considering two portions of the images. The foreground portion shown in (b) is comprised of moving objects in the image and makes up only a tiny fraction of the image. The background portion shown in (c) varies slowly and smoothly over time, making it ideal for aggressive compression.

1. images are taken repeatedly over the same area, and
2. the primary interest is for moving objects.

With two assumptions, the strategy pursued here is to compress foreground and background imagery separately, see Figure 1. Moving objects comprising the foreground make up a tiny fraction of the pixels in the image and can be transmitted directly. Background imagery varies slowly and smoothly, making it ideal for compression.

The background imagery is compressed in a streaming fashion with the aid of 1D wavelet analysis performed on a per-pixel basis. Using temporal wavelet analysis, the resulting *wavelet images* effectively represent several frames of raw input images. Since movers only occupy a tiny fraction of the overall imagery, and background imagery is reduced significantly to low temporal rates, and the overall compressed size can easily be a factor of 10 to 100 smaller than using existing state-of-the-art still-image and video compression on the image stream, while meeting the special accuracy requirements for analysis.

To minimize memory usage when compressing a large time sequence of huge images, streaming variants on wavelet analysis are used. The key idea is to view wavelet lifting diagrams as dependency graphs, and to effectively "parse" the diagrams as soon as images are taken. Overall this reduces the total memory footprint.

The system consists of several components—image segmentation, temporal compression, and rendering. In this paper, we focus on the description of an image segmentation approach where methods are derived from similar techniques used in the visualization and computer graphics community. Section 4 describes the image segmentation methods. The wavelet-based streaming compression algorithm is discussed in Sections 5, 6 and 7. Section 8 addresses the replacement of foreground movers by static elements. Section 9 illustrates the efficient rendering of the compressed information using a GPU-based algorithm. Sections 10 and 11 illustrates results obtained with this system.

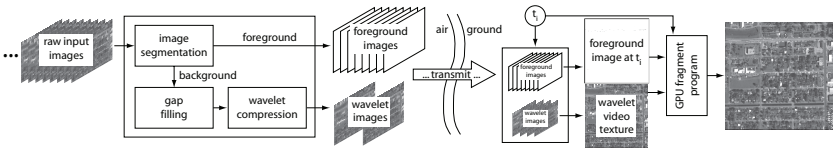


Fig. 2. Overview of the complete compression system. A raw input stream of images are segmented into foreground and background images. Moving objects comprising the foreground make up a tiny fraction of the pixels in the image and are transmitted directly. Background images are compressed aggressively using wavelet-based techniques, greatly reducing the number of frames that need to be transmitted. After transmission, wavelet images are "decompressed" efficiently in the renderer for a user-specified time point t_i, and foreground pixels are overlayed on top.

2 Related Work

There is an extensive volume of work on segmenting foreground from background in computer vision [1]. Algorithms for object recognition and video tracking are most related to our work. In particular, the goal of video tracking is to locate moving objects in time. While the literature has primarily focused on improving the speed and accuracy of locating a few moving objects, relatively little research has been devoted to locating a large number of moving objects efficiently. Consider that aerial image sequences may contain hundreds of moving vehicles.

Wavelets have been used in computer graphics and visualization for a number of applications, including multiresolution analysis, variational modeling, and compression [2]. B-spline wavelets have been combined with subdivision surface techniques to represent geometric models at multiple levels-of-detail [3, 2]. They compute wavelet transforms using a lifting approach [4] that divide wavelet analysis into smaller, more efficient lifting operations.

Streaming techniques treat the data as a continuous stream of information. These algorithms operate on restricted contiguous portions of data which are currently "in focus." Because streaming methods operate on a fixed amount of data at a time, they are often faster, more efficient, and more scalable than "global" techniques that require an entire data set to reside in main memory [5]. Recent research has focused on streaming algorithms for simplifying and compressing 3D geometric models [5]. The availability of online media has increased the need for streaming video encoding and decoding methods [6]. Several codecs for streaming video exist, however many of these methods focus on only streaming content delivery. Compression and encoding of the content is often considered a preprocess.

In our work, we combine image segmentation and wavelet methods to compress long sequences of huge aerial imagery in a streaming fashion. We analyze pixel intensity values over time to separate pixels that represent moving foreground objects from stationary background pixels. Background pixels form a slowly varying image sequence that we compress with the aid of wavelet analysis. We use streaming evaluation of the wavelet transformation to allow both compression and delivery to occur as soon as possible in a streaming fashion while keeping the memory footprint low. Wavelet encoded

images compactly represent large numbers of images. They can be rendered efficiently making them suitable for incorporation into state-of-the-art large terrain, large texture viewers, such as the one developed by Hwa et al. [7, 8].

3 System Overview

Given a raw input stream of huge images, our system outputs foreground images and wavelet-compressed background images. In the first stage of processing, raw input images are segmented into foreground and background images by analyzing changes in pixel intensity over time. Holes in the background images are filled in preparation for wavelet compression. The background images are passed to a wavelet compression engine comprised of several levels of wavelet transform. We refer to these output background images as *wavelet images*. The number of wavelet images output is a constant factor smaller than the number of images input to the system. Wavelet images and foreground images constitute a compressed set of information that is suitable for transmission. After transmission, consecutive sets of wavelet images are used to render larger video sequences using a texture-mapping approach. Foreground images are overlayed on top of the reconstituted backgrounds. We refer to the sets of wavelet images as *wavelet video textures*. Figure 2 illustrates our complete system.

4 Image Segmentation

Segmentation of moving objects is performed in two phases, detection and completion. In the detection phase, the goal is to have at least a single pixel of positive detection per moving object, not including paralax motion of buildings, trees, or other tall structures. In the completion phase, pixels for the entire object are determined. Extra "guard pixels" surrounding the object are also labeled as part of the mover to obtain a conservative segmentation. Both phases of this algorithm could be accomplished with well known but expensive search and correlation strategies. For real-time processing on small sensor platforms, fast, local computations are desired.

For detection, two approaches were tested. The first approach uses two buffered frames to perform detection. Pixels are declared likely movers in frame f if (1) their value $v_f(x,y)$ is different from all the values within one pixel in frame $f+1$, (2) this frame-to-frame difference has a high gradient magnitude, (3) the gradient magnitude at frame f is high, and (4) the gradient directions in the two cases above are parallel to one another (either the same or exact opposite directions). This is formulated by defining a unit gradient vector

$$g = \frac{\nabla v_f(x,y)}{||\nabla v_f(x,y)||}$$

and defining

$$d_f(x,y) = \min_{dx,dy \in [-1,1]} |v_f(x,y) - v_{f+1}(x+dx, y+dy)$$

along with

$$h = \frac{\nabla d_f(x,y)}{||\nabla d_f(x,y)||}$$

and

$$q = (h' \cdot g')(||g|| - g_{m\min})||h||$$

This consistently finds some pixels per mover, but is somewhat sensitive to noise, and does not completely eliminate paralax-induced detections of tall building edges.

The second detection approach uses five frames. The median value is obtained for a pixel over these frames. The detection is then the difference between the current frame's value and the least different value from a small neighborhood of the next frame. A small amount of "soft erosion" is performed on the results. This was found to be very reliable at detecting some pixels per mover, and was not as sensitive to noise or to building edge paralax.

After detecting a set of pixels from each mover, a conservative region of pixels is determined around each complete mover. Using the results of the five-frame detection, there are good starting points to seed a search for all mover pixels. We start with the core pixel of each connected component, defined as the last pixel to be deleted for the component by repeated erosion operations that are restricted to not break components into two pieces. From this starting point, we flood fill to all pixels that are near pixels with frame-to-frame differences with magnitudes above a specified threshold.

5 Wavelet Analysis Reviewed

The lifting approach [4] for discrete wavelet analysis uses a 1D input signal F represented by a set of samples f_i uniformly spaced in time at points t_i. F is decomposed into *scaling-function coefficients*, or *s-values*, and *wavelet coefficients*, or *w-values*. In the lifting approach, this decomposition is computed by relabeling signal values as

$$s_i = f_{2i} \qquad \text{and} \qquad w_i = f_{2i+1}$$

followed by a sequence of alternating *s-lift* and *w-lift* operations. The s-lift and w-lift operations are defined as
s-lift(a, b):

$$s_j = aw_{j-1} + bs_j + aw_j \qquad \forall j \qquad \text{and} \tag{1}$$

w-lift(a, b):

$$w_j = as_j + bw_j + as_{j+1} \qquad \forall j. \tag{2}$$

Therefore, *s*-lifts compute new s-values by computing a weighted sum of a given s-value with its neighboring w-values. Likewise, *w*-lifts compute new w-values by computing a weighted sum of a given w-value with its neighboring s-values. The order and number of lifting operations and the values of the weights a and b determine the type of wavelet used. In our work, we use uniform cubic B-spline wavelets [3, 2]. This transform is defined by the sequence of three lifting operations *s*-lift(- 1/4, 1), followed by *w*-lift(-1, 1), and then *s*-lift(3/8, 2). Figure 3 illustrates the lifting approach to wavelet analysis.

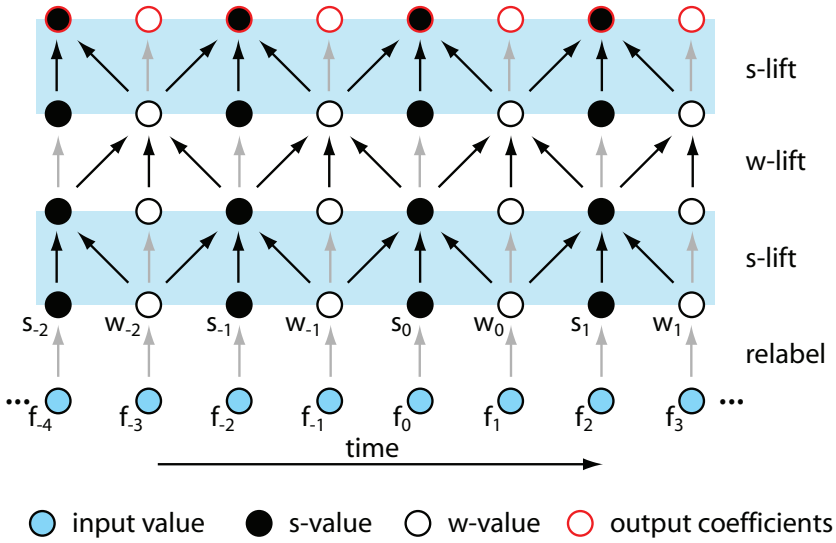


Fig. 3. Illustration of cubic wavelet analysis. Input values shown as blue circles are relabeled as s-values and w-values. Three alternating s- and w-lift operations compute the final cubic wavelet coefficients outlined in red. Black arrows show which values contribute to intermediate values between lifting operations. Gray arrows illustrate which values remain the same between operations.

6 Streaming Wavelet Analysis

Values resulting from wavelet analysis can be computed in a streaming fashion. We view the stages of the lifting operations as a dependency graph, where nodes represent computed values and directed edges represent how terms are combined to produce a computed value. From this perspective, the relationship between input signal values and output values reveals that the first s- and w-values can be computed after the first four input signal values have been received, see Figure 4a. Further investigation reveals that subsequent pairs of s- and w-values can be computed as soon as pairs of input signal values are received, see Figure 4b. Only three internal results from previous computations need to be maintained between updates to compute two new output values. Thus, values from wavelet analysis can be produced in a streaming fashion, using a small memory footprint to track the computation state. For the purposes of this discussion, we call the computational engine that performs streaming wavelet analysis a *stream analyzer*.

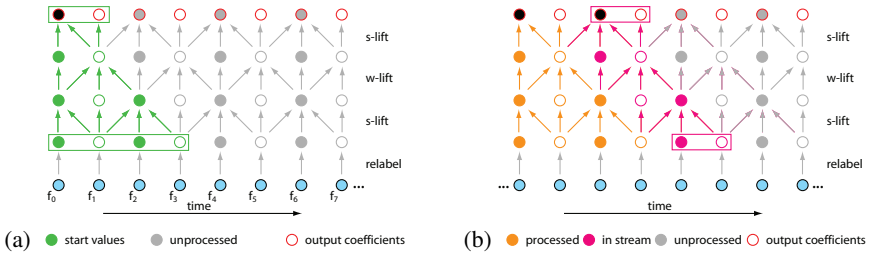


Fig. 4. Illustration of streaming cubic wavelet analysis. The beginning of the computation shown in (a) requires four initial values shown here in green. These values produce one s-value and one w-value. Post-initialization computations are shown in (b). Here, pairs of input values produce pairs of output values. The pink circles represent stream values currently being computed, and orange circles represent those computed from the last stream update. Note that the new output values depend on three values from the previous computation, and so three values must be maintained in memory across stream updates. This dependence is visualized here as pink arrows emanating from orange circles. Gray circles represent values that have not yet been processed.

7 Wavelet Compression

Compression is achieved by discarding w-values and performing several levels of wavelet analysis. Discarding w-values achieves a factor of two reduction in data size. The s-values from one level of analysis are used as input to the next level. Since each level of wavelet analysis achieves a factor of two reduction in data size, k levels of such nested wavelet analysis achieve a factor of 2^k reduction in data size. We achieve streaming wavelet compression by cascading k stream analyzers together such that s-value output from one stream analyzer is used as input to the next stream analyzer. Figure 5 illustrates a sample wavelet compression system constructed from three stream analyzers.

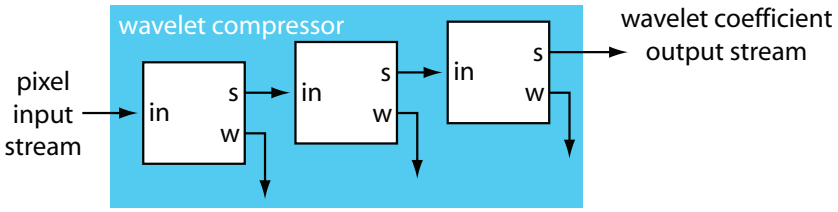


Fig. 5. Illustration of a streaming wavelet compression system. A pixel value stream is input to the system, which computes a smaller output stream of wavelet coefficients. The system is constructed from three stream analyzers. Here, each box represents a stream analyzer that performs wavelet analysis, producing s- and w-values. The s-values are used as input to subsequent stream analyzers, and w-values are disregarded. Each level of wavelet analysis reduces the stream by a factor of two. Thus, this example system prduces an output stream that is a factor of $2^3 = 8$ smaller than the input stream.

8 Filling Gaps

This wavelet analysis requires a uniform spacing of samples over time. Image segmentation removes foreground pixels, creating gaps in the stream passed to the wavelet compressor, and we must fill these temporal gaps in order to perform wavelet compression.

We adopt a simple strategy to fill in the gaps created by foreground pixels. The buffer keeps track of the last value received as well as the point in time it was received. Let f_{last} be the last pixel value received and t_{last} be the point in time it was received. Let f_{curr} be the newly arrived pixel value and t_{curr} be the current point in time. A gap exists if $t_{curr} - t_{last} > 1$. Missing values corresponding to time values $t_{last+1}, \ldots, t_{curr-1}$ are computed using linear interpolation between f_{last} and f_{curr}. We note that the goal here is to provide noise-free input to the wavelet compressor, and so linear interpolation suffices to maintain a stable input signal and is more desirable than filling gaps with zeroes. Figure 6 illustrates the gap filling process.

9 Rendering Compressed Video Images

Our system performs gap filling and wavelet compression for each pixel of each background image, producing output images whose pixel values correspond to wavelet coefficients. We refer to the output images containing wavelet coefficients as *wavelet images*.

Each set of four consecutive wavelet images defines a single four-channel texture that compactly represents several frames of the raw input video. The four intensity values of each pixel of this image define a uniform cubic B-spline curve. Evaluating these per pixel B-spline curves at time t_i provides the pixel intensities comprising an image that approximates frame t_i of the raw input video. We refer to the four-channel textures as *wavelet video textures* and note that they can be rendered efficiently using a GPU fragment program.

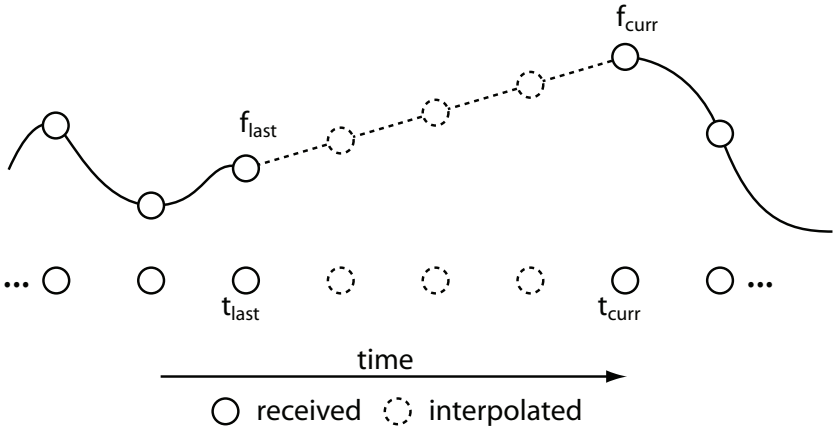


Fig. 6. Illustration of gap filling. Gaps are detected easily by comparing the arrival time of the current pixel value against the arrival time of the last pixel value. Missing values are linearly interpolated using the current and last pixel values. Solid circles represent received values and dashed circles represent interpolated values.

Let I_u be the set of wavelet images. Each I_u represents 2^k frames of the original input video. At time t_i, the four wavelet images necessary to construct a wavelet video texture are I_{u-1}, I_u, I_{u+1}, and I_{u+2} where $u = i/2^k$. Given the wavelet video texture and a specific point in time, a GPU fragment program renders the appropriate background image by evaluating a uniform cubic B-spline. Once the compressed background image has been rendered, foreground pixels are overlayed onto the result.

10 Discussion

Performing streaming wavelet compression is advantageous for several reasons. It keeps the memory footprint small, which is critical, as the process is performed on a per-pixel basis. Since three internal values per pixel need to be kept between stream updates, 3×2^k values are needed per pixel to perform cubic wavelet compression. Streaming allows values to be returned as soon as possible, enabling results to be displayed sooner despite the extra computation. Our compression saves bandwidth in that a single wavelet image need only be sent once for every 2^k raw input images.

Our wavelet encoding provides a few distinct advantages. The smooth nature of our compression naturally removes undesirable noise found in the original image sequence. The encoding is amenable to hardware-accelerated rendering through the use of texture mapping and programmable GPUs, enabling rendering efficiency. The encoding results in a stack of wavelet image textures, which can be incorporated into state-of-the-art terrain visualization systems capable of managing and rendering large textures [7, 8].

Naturally, streaming compression impacts the latency between when images are captured and when they are displayed. This latency is most significant at the initialization of the compression system, where $3 \times 2^k - 2$ frames are necessary before the first

wavelet image is produced. Subsequent wavelet images are produced for every 2^k raw input images, and three additional images are necessary before the first wavelet video texture can be rendered. Thus, the initial cold-start latency is $(3 \times 2^k - 2) + (3 \times 2^k) = 6 \times 2^k - 2$ frames.

Our approach to compression has some limitations. As with many other vision algorithms, our image segmentation procedure is sensitive to reflections off of shiny objects such as bodies of water or metal roof tops. Tall structures such as skyscrapers appear to move as the vantage point of the sensors shifts. Both of these issues cause fluctuations in pixel intensity over time making the problem of identifying foreground movers even more difficult and will be addressed in future work.

11 Results

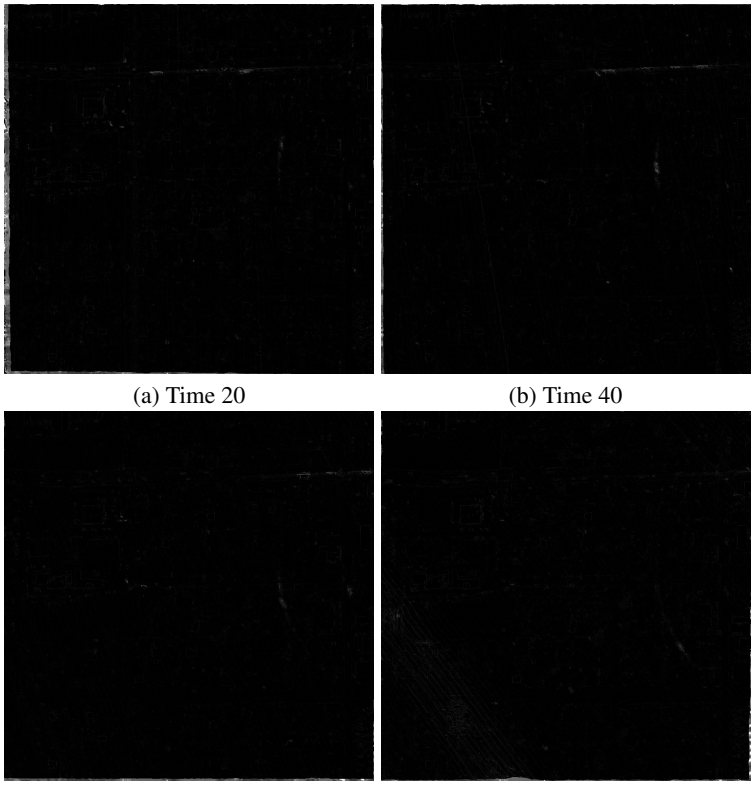
We have tested our method on a 100 frame sequence consisting of 1 megapixel images. The total raw input size was 100 MB. Our test machine was a 2.8 GHz P4 PC with 2 GB of memory to perform streaming wavelet compression on the sequence. In our implementation, we use $k = 4$ levels of wavelet analysis for compression, achieving a 16:1 compression ratio.

Figure 7 shows difference images between our compressed result and the original images. They demonstrate that our compression scheme works relatively well, but is sensitive to foreground pixels mislabeled as background pixels. The difference images also illustrate that undesirable noise in the original image sequence do not exist in the compressed result. We note that some images show portions of the original images toward the borders of the images. This is due to image registration, since some frames simply do not contain pixel information contained in other frames.

12 Conclusions and Future Work

We have described a streaming compression algorithm designed for huge time-varying aerial imagery. By treating foreground and background imagery separately, our system is able to achieve high levels of compression. We have developed a streaming formulation for wavelet analysis that satisfies several requirements of high-resolution aerial photography: computational efficiency, low-memory footprint, compression suitable for transmission in low-bandwidth environments, streaming update of recently captured images, and high-quality approximation to slowly-varying background images. The wavelet images resulting from this compression are combined creating wavelet video textures that compactly represent large numbers of raw input frames and that are inexpensive to render. As our compression method is a streaming technique, it scales well to larger input image sizes and is insensitive to the duration of the image sequence.

As future work, we will investigate ways to achieve further compression. After all pixels are processed in the 1D temporal streaming transform, the wavelet images can be compressed spatially using a number of image compression techniques, including those based on biorthogonal wavelets. We hope to develop compression techniques tailored for fast-varying foreground imagery. Although the compression we have presented is highly effective, it depends on a good segmentation of background from foreground.



(a) Time 20 (b) Time 40

(c) Time 60 (d) Time 80

Fig. 7. Difference images for select time steps of a 100 frame data set show two things: (1) the error introduced by cubic wavelet approximation is low and (2) some pixels corresponding to moving objects are not classified as foreground.

This prerequisite motivates further development of fast and effective segmentation techniques.

Acknowledgments

This work was supported by Lawrence Livermore National Laboratory under contract B556671. We thank members of the Visualization and Graphics Group of the Institute for Data Analysis and Visualization (IDAV) at UC Davis for their support in this project.

References

1. Forsyth, D.A., Ponce, J.: Computer Vision: A Modern Approach. Prentice Hall (2003)
2. Stollnitz, E.J., DeRose, T.D., Salesin, D.H.: Wavelets for Computer Graphics: Theory and Applications. Morgan Kaufmann, San Francisco, CA (1996)
3. Bertram, M., Duchaineau, M.A., Hamann, B., Joy, K.I.: Generalized B-spline subdivision-surface wavelets for geometry compression. IEEE Transactions on Visualization and Computer Graphics (2004)
4. Sweldens, W.: The lifting scheme: A new philosophy in biorthogonal wavelet constructions. In Laine, A.F., Unser, M.A., Wickerhauser, M.V., eds.: Wavelet applications in signal and image processing III. Volume 2569 of Proceedings of SPIE. (1995) 68–79
5. Isenburg, M.: Compression and Streaming of Polygon Meshes. PhD thesis, University of North Carolina, Chapel Hill (2004)
6. Richardson, I.: Video Codec Design: Developing Image and Video Compression Systems. John Wiley & Sons, Ltd. (2002)
7. Hwa, L.M., Duchaineau, M.A., Joy, K.I.: Adaptive 4-8 texture hierarchies. In: Proceedings of IEEE Visualization 2004, Los Alamitos, CA, IEEE, Computer Society Press (October 2004) 219–226
8. Hwa, L.M., Duchaineau, M.A., Joy, K.I.: Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. In: IEEE Transactions on Visualization and Computer Graphics. Volume 11. (July-August 2005) 355–368