Computers **2014**, *3*, 58-68; doi:10.3390/computers3020058

OPEN ACCESS

computers

ISSN 2073-431X

www.mdpi.com/journal/computers

Article

Adaptive Video Transmission Using Residue Octree Cubes

Adrian Enache and Costin-Anton Boiangiu *

Department of Computer Science, Faculty of Automatic Control and Computer Science, University "Politehnica" of Bucharest, Splaiul Independentei 313, 060042 Bucharest, Romania; E-Mail: adrian.enache@cti.pub.ro

* Author to whom correspondence should be addressed; E-Mail: costin.boiangiu@cs.pub.ro; Tel.: +40-762-609-111.

Received: 4 April 2014; in revised form: 4 July 2014 / Accepted: 7 July 2014 / Published: 15 July 2014

Abstract: This paper proposes a method of transmitting video streaming data based on downsampling-upsampling pyramidal decomposition. By implementing an octal tree decomposition of the frame cubes, prior to transforming them into hypercubes, the algorithm manages to increase the granularity of the transmitted data. In this sense, the communication relies on a series of smaller hypercubes, as opposed to a single hypercube containing the entire, undivided frames form a sequence. This translates into increased adaptability to the variations of the transmitting channel's bandwidth.

Keywords: video; streaming; adaptive; cube; hypercube; octree; upsampling; downsampling; entropy coding; codec

---

1. Introduction

Modern video content distribution requirements include a minimum of two metrics. The transmission needs to use low bandwidth and have low latency.

Usually, the latency problem is hard-imposed by the underlying network transmitting media, but the software used to send and receive the media content can also be responsible for introducing extra delay. This software algorithm may add latency due to the processing it does (e.g., coding/decoding, compression/decompression).

The minimal bandwidth limitation is an issue strictly pertaining to software, and while the transmitting media has an upper limit, it does not impose a lower bound, like in the case of latency.

The bandwidth problem can be mitigated by implementing proper algorithms for coding and compression before sending the media data to the clients.

1.1. Introduction to Similar Systems

A video streaming service must be able to offer low latency, high quality, extensions, like ease of navigation, subtitles, chapters, *etc.* In order to manage the complexity of the system, it must be split into layers [1,2], our attention being aimed towards the application layer—the coding and decoding of the transmitted data through the transport layer.

The application layer consists of the video container, the codec and the program that renders the results. The video container is a meta-file containing information, like aspect ratio, frame rate, key-frames and duration, with reference to the coded data. Popular containers are AVI, MP4, MOV and FLV, with MP4 and MOV evolving to be technically the same and AVI being more limited than the rest.

The video codec is responsible for packing the video data in as little information as possible and recovering the original data while rendering. It is not mandatory for the compression to be light-weight, because it only takes place once, on a dedicated machine (though portable video cameras can benefit from a low complexity compression). On the other hand, the decompressing takes place on all of the systems on which the video must be displayed; thus, even low power devices must be able to cope with the complexity of the algorithm.

The most used video codec today is H.264 (ex: YouTube), an improved version of MPEG. It uses three types of frames—I (Initial, all information in an image), P (Predicted, difference between the predicted and actual frame) and B (Bi-Predicted, constructed from I and P or O and P frames). Each frame is composed of 8 × 8 pixel tiles, which are compressed with the discrete cosine transform and are tracked inside larger 16 × 16 pixel macro-blocks, in order estimate the movement and to improve the P and B frames. The High Efficiency Video Coding standard (HEVC/H.265) uses tiles of variable size (1–64 pixels), intra-frame predictions (from neighboring pixels in a frame), more accurate motion estimation, deblocking for diminishing compression artifacts, along with numerous extensions, like 3D depth data.

The extension for H.264, Scalable Video Codec (SVC) splits the video data into a base video and multiple residues, intending to add an adaptive quality feature to the codec. It includes temporal (removing of frames), spatial (losing resolution, obtained through layers) and quality scalability.

The drawbacks of this mechanism are the complexity and single-frame image quality. To exemplify: at the transition of surveillance cameras from Motion JPEG to MPEG2, the recording device was observed to drop frames, and at the expense of maintaining a high quality video recording of the static scenes, the single important frames lacked the needed details in order to distinguish a subject. If someone wants to freeze a H.264 picture for forensic investigation purposes, his/her best option is to step to the closest I frame, possibly missing the important event. Furthermore, more advanced network cameras and high-performance monitoring stations are required in order to be able to meet the current demands of 30 fps and high resolution acquisitions, a purpose which cannot be easily achieved with wireless devices. However, this problem has been partially handled by the profile settings: the baseline uses simpler algorithms and no B frames, resulting in a low quality video at the

same bandwidth; the high setting uses high-performance and high-complexity algorithms. One of the big problems that affect MPEG encoders is rapid motion, as motion deduction cannot correctly estimate the positions of the blocks. This has been improved in H.264, by using more reference frames and, thus, smaller distances between fast moving frames, but at the cost of processing power (every reference frame has to be tested for block movements) and file dimensions. These are unacceptable in a system where a minute percentage of the frames contain the desired information. MPEG variants are designed to give a sense of quality while playing, by predicting, but when a person's face needs to be observed, no single frame contains enough information, because of the high compression.

Motion JPEG and MxPEG encode each frame using temporal coherence and the JPEG image codec in order to resolve the single frame quality issue. Motion tracking does not take place, so fast moving scenes do not represent a problem. Another advantage comes with Windows-based file systems like New Technology File System (NTFS) and File Allocation Table (FAT) because they are optimized to work with many small files rather than a single large file. As a file becomes larger, the system ends up fragmenting it, the performance decreasing with time for single, large file-based devices. However, all of these come at the cost of bandwidth and memory, typical videos being around 80% larger than their H.264 counterparts. That is why they are seldom used at the needed speed of 30 fps. They are also unreliable on variable bandwidth networks, because they can only be sent at the quality at which they exist on the disk.

1.2. Hypercube Base Algorithm

This paper continues and generalizes the research presented in [3], proposing a method for compressing a video sequence by grouping multiple frames in a cube and splitting it into multiple cube residues (of different resolutions), a "hypercube". The work conducted in this paper introduces the finer hypercubes mentioned in the "future work" section of the previous article and includes the implementation and testing of the codec.

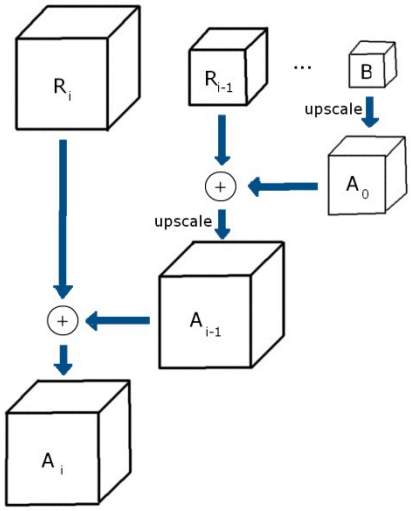
The idea is an extension of the Laplacian pyramid [4] for images to video. A frame cube is a 3D texture created from a sequence of frames taken from a video source. Frame cubes can be downsampled and re-upsampled multiple times (at different scales) in order to compute frame cube residues (which are the difference between the original and interpolated cubes).

As a quick overview, a "frame hypercube" is a series of incrementally smaller frame cube residues and a base frame cube. As seen in Figure 1, the reconstruction of the original frame cube is done by reversing the process, repeatedly upsampling the base cube and adding the subsequent residue level. The reason behind this representation is that the residue frame cubes contain very little information (the difference between scales), in the form of small values, centered at zero [5,6]. Thus, after a remapping of the value interval, they are better suited for entropic compression methods (e.g., Huffman coding).

The real advantage of this scheme is that it can be used for adaptive transmission and reception: transmitting the hypercube slices (*i.e.*, frame cube residues at different scales) is done starting from lower sizes going to higher sizes. A consequence of this strategy is that, if a drop in bandwidth occurs, even though a client may receive an incomplete number of frame cube residues, the client software is still able to synthesize a lower-quality frame cube and use that to present the frames [7].

The method and results for this technique are presented in more detail in [3].

Figure 1. Algorithm to synthesize initial full-quality frame cube A_i. Residue cubes are marked R; the starting base frame cube is marked B.



1.3. Octree Base Algorithm

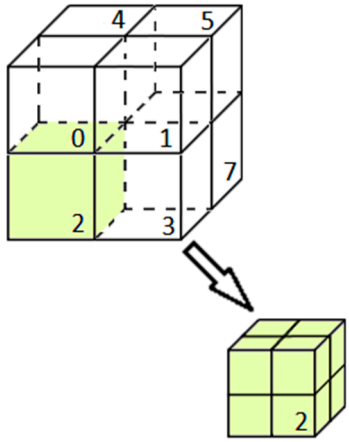
The main optimization, which is the topic of this paper, is splitting the initial frame cube using an octree algorithm. The reason for doing this is the need to increase the granularity of the transmitted data. In this sense, the octree splitting algorithm works in the following way.

Given the starting full-quality frame cube, it is no longer converted directly into a frame hypercube, but it can be split into eight sub-cubes, as depicted in Figure 2. The positions of the sub-cubes are fixed in order to remove complexity from the reconstruction algorithm. As such, the origin is in the top, left, front sub-cube, and we have:

- Cube 0 in top, left, front
- Cube 1 in top, right, front
- Cube 2 in bottom, left, front
- Cube 3 in bottom, right, front
- Cube 4 in top, left, back
- Cube 5 in top, right, back
- Cube 6 in bottom, left, back
- Cube 7 in bottom, right, back

The splitting into sub-cubes is recursive, and the end-condition can be expressed as a function of the sub-cube size and a metric determined before each splitting. The metric chosen in the current implementation is based on the standard deviation. It is calculated as a mean between the standard deviations of each slice in the given frame cube. While there may exist better metrics, the reasoning behind choosing this particular metric is that it measures the "uniformity" of the pixel colors inside the given frame cube [8]. This means that, the more uniform the color is the less information will be wasted in the compression stage. Since this algorithm generates a tree structure (octree), the cubes that can no longer get split will be called leaf-cubes.

Figure 2. Octree fixed structure. Second sub-cube (highlighted) is recursively split.



The algorithm used for the construction of the octree generates a set of leaf-cubes. Each of these can now be fed through the same algorithm presented in [3], in order to create a hypercube of resolutions. Therefore, at the end of the octree construction, the software will have a list of leaf-cubes represented in hypercube form. Since the initial transmission order was implicit (due to the fact that there was only one hypercube, so the base and residue layers would be sent in succession), the problem remaining is that there are multiple hypercubes of different sizes that need to be prioritized.

The chosen prioritization scheme is based on a round-robin transmission from a sorted set of hypercubes. The sorting is done in order of importance: the temporal depth and, afterwards, the actual 2D sizes of the frame images. Using this scheme, the server can use a severely limited bandwidth in order to get a minimum amount of information to the client, to the point that, for hypercubes that get to transmit no data, the client just replaces them with a blank frame cube or the most recent available cube.

Figure 3. Degradation process in a dynamic scene: (**a**) 90%, loss of insensible detail (Peak Signal-to-Noise Ratio (PSNR) 36.9 dB); (**b**) 55%, uniform areas start losing temporal information, causing regional motion blur (PSNR 22.3 dB); (**c**) 30%, spatial information loss becomes noticeable (PSNR 14.0 dB); (**d**) 25%, example of temporal and spatial information loss (PSNR 18.0 dB).

(**a**)

(**b**)

Figure 3. *Cont.*

(**c**)

(**d**)

The raw presented technique does, however, have a drawback. If the frame hypercubes are not received in full resolution, there will be a noticeable border between them and the complete sub-sequences (Figure 3b). For correcting this issue, a mechanism of compensation needs to be involved. In this project, this was done by artificially increasing the size of the leaf-cubes, therefore getting some overlapping data between them. This also raises the lower bandwidth limit. The overlapping of the hypercubes now allows the client software (which knows the relative positions between the octree nodes) to blend into the 3D edges between the frame cubes. This means that frame cubes of higher quality (computed as the ratio between the received number of hypercube slices and the full quality number of slices) get to blend into the neighboring lower quality cubes. If they have the same quality factor, the blending occurs simultaneously on both sides. The blending algorithm is quite straightforward, and it assumes that the blending is done (either linearly with equal weights or non-linearly with variable weights) on each of the three axes using Equation (1).

$$P(x) = C_1(x) * w(x) + C_2(x) * (1 - w(x)) \quad (1)$$

where:

- x is the distance from the edge on the respective axis/plane;
- $P(x)$ is the computed pixel color;
- $C_1(x)$ and $C_2(x)$ are input colors from both cubes at distance x from the edge;
- $w(x)$ is the computed weight w.r.t. x. If used as a linear blending function: $w(x) = 0.5$.

2. Complexity Discussion

Table 1 shows the times measured for each step of the process. Since the conversion between video frames and the custom octree format was done offline, the only interesting time is the one on the last column. This time specifies the computing time needed for a client to get all of the full-quality leaf hypercubes and process them into frames. The software has made use of the OpenCV framework for the scaling operations and residue calculations (residues that are afterwards compressed, before storing/transmitting).

Table 1. Profiling per octree cube size.

Depth (Frame Count)	Total Frame Time (ms)	Load Image (ms)	Frame Cube (ms)	Octree (ms)	Save (ms)	Load (ms)	Synth (ms)	Render Total (ms)	Remaining Buffer (Frame Time − Render Total) (ms)
96	3197	535	24	1994	3795	782	284	1066	2131
64	2131	384	18	1272	2368	497	198	695	1436
48	1598	303	12	953	2065	489	143	632	966
32	1066	224	9	564	1237	254	71	325	741

It is important that this processing time be no higher than the time it would take to actually render the frames, because the client would not have time to buffer the data, causing stuttering, e.g., the movie used here was rendered at 30 fps, which translates into 33 ms per frame. For x frames, this means that there is a time buffer of $33 \times x$ milliseconds. As seen in the Table 1, the algorithm mainly takes about a third of this time buffer.

3. Experimental Section

In order to obtain the results depicted in this chapter, we used these parameters:

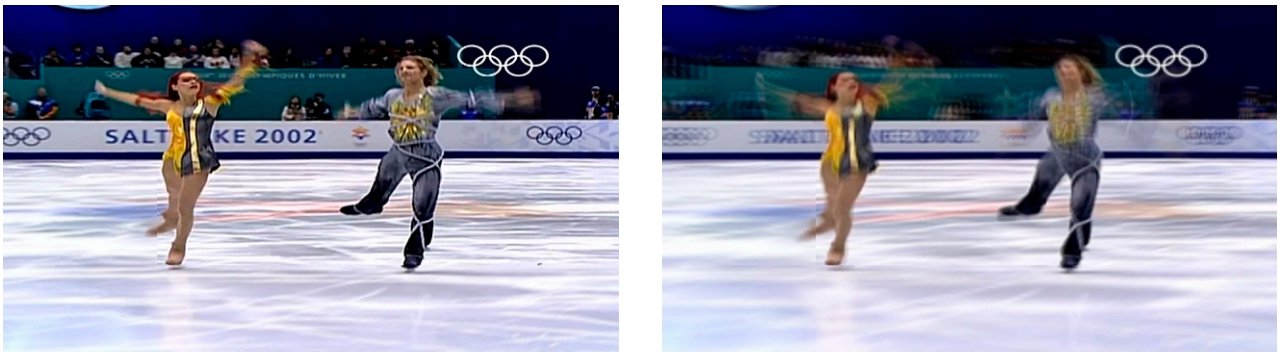
- octree sub-cube threshold size for end-condition (in pixels) = $8 \times 8 \times 4$;
- octree sub-cube threshold standard deviation for end-condition = 60.0;
- octree sub-cube division factor (on all three axes) = 2.0;
- frame cube depth (in frames) = 32;
- hypercube base cube minimum size (in pixels) = $4 \times 4 \times 2$;
- hypercube maximum level of residues (resize operations) = 5;
- hypercube resize factor (on all three axes) = 2.0.

We used two classes of input videos for the tests: dynamic and static scenes. The algorithm copes well in both cases. When the video sequence is very mobile, at low compression rates, there is little to no difference. As the compression gets stronger, because of combining information on the time axis of the cubes, a natural motion blur is generated. This actually helps the psychological process of interpreting the motion in the video, as opposed to current frame-based technologies, which merely jump frames or delete residues when data cannot be acquired in time. At very strong compression ratios, spatial interpolation becomes evident and unpleasant.

Because of the exemplified process, even though the PSNR estimates moderate quality, the inherent motion blur introduced generates a more satisfying result (Figure 3). Figure 4 offers an example where even a single frame at a compression just above the noticeable decimation in the spatial information (40%) is pleasant, the difference from the original being a very strong motion blur. Another proof for the inappropriate estimation of the PSNR is again Figure 3: at 30% quality the PSNR is 14 dB, while at 25%, it raises back to 18 dB; the paradox being caused by the diminishing of the blur when the scene becomes less dynamic.

For static scenes (Figure 5), even at extreme compression ratios, details remain evident, the only problem being a not so good region of interest detection.

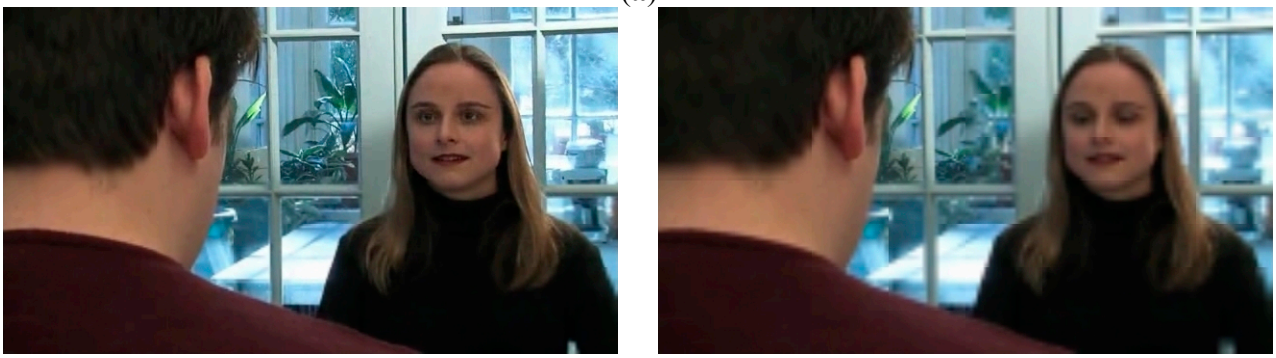
Figure 4. 100% (**a**) *vs.* 40% (**b**); minimal boxing artifacts can be noticed. The resulting motion blur helps the physiological process of interpreting movement.



(**a**) (**b**)

Figure 5. Compression example using a static scene: (**a**) 100%, (**b**) 40%, (**c**) 17%; (b, c) Static scenes can be strongly decimated without much loss of information; (c) standard deviation-based defined regions of interest are poorly detected: the plant in the background kept details, while the actor's face is unclear.

(**a**)



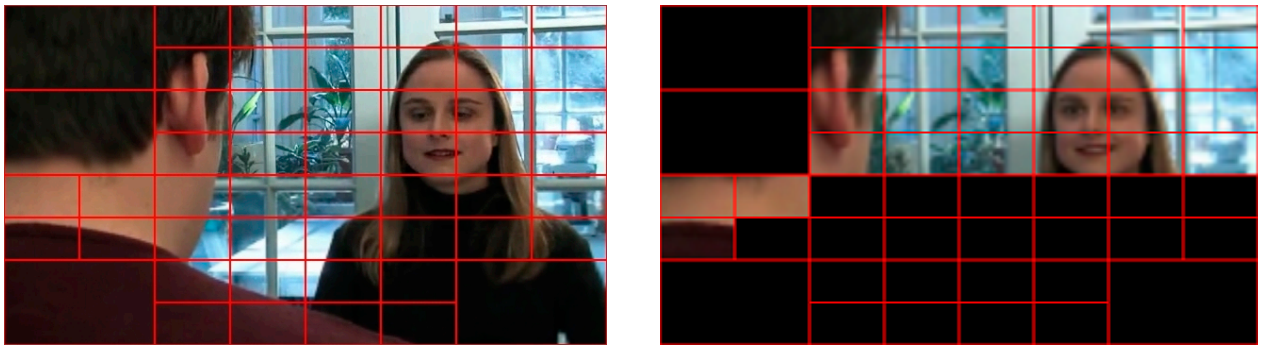
(**b**) (**c**)

If the bandwidth drops drastically, only the regions of interest are being transmitted, the rest having the possibility of being interpolated (Figure 6).

Usage of the standard deviation as an importance measure for a region in the frame, for the octree splitting end-condition, has proven to offer moderate results. For the example in Figure 3, ice regions are correctly determined as unimportant, but in the example in Figure 5, the ear in the foreground and plant in the background are marked as more important than the face of the actor.

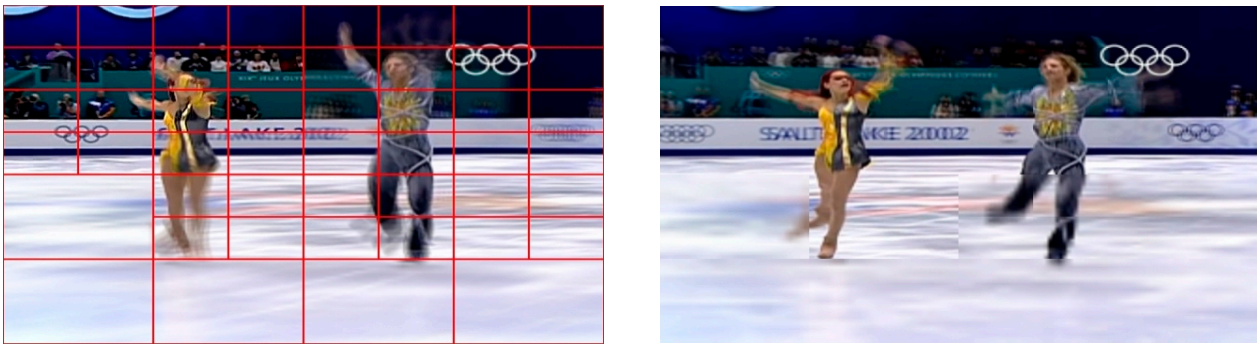
In Figures 7 and 8, the tiling artifacts at the borders of the cubes are clearly evident. They can be diminished by overlapping the cubes by a few pixels and gradually going from one cube to the other.

Figure 6. (**a**) Octal tree generation example; (**b**) at very low bandwidth rates, only regions of interest are being sent.



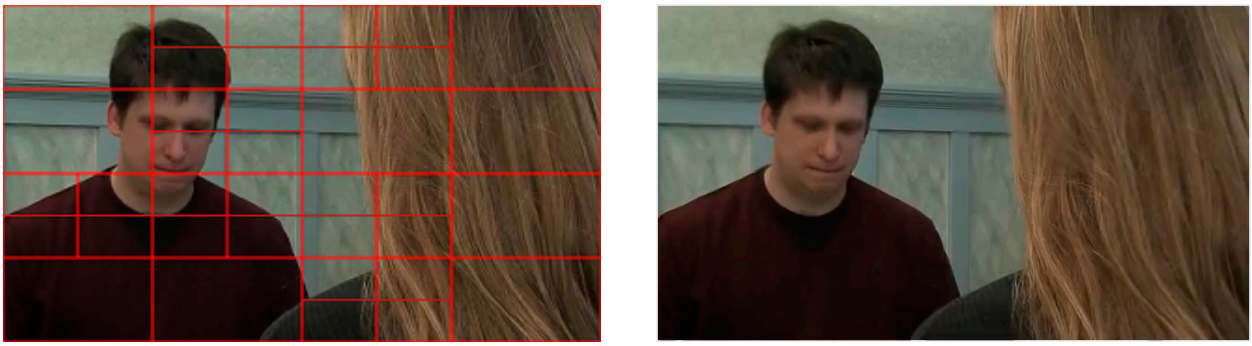
(**a**) (**b**)

Figure 7. (**a**) Octal tree generation example; (**b**) uniform regions are more strongly encoded; tiling artifacts appear.



(**a**) (**b**)

Figure 8. (**a**) Octal tree generation example; (**b**) standard result for blending overlapping areas.



(**a**) (**b**)

4. Conclusions

The proposed algorithm is a significant improvement over the original hypercube method described in [3]. The original method was meant for splitting the set of movie frames into frame cubes and directly applying the pyramidal decomposition on them, whereas the optimization presented in this

paper takes those frame cubes and splits them in a recursive octree fashion, each of the octree leaf being considered a frame cube in itself (it can be considered that the algorithm presented in [3] is a specialization of the current optimization, having an octree with a depth of one; the only leaf being the starting frame cube). Therefore, it offers both more bandwidth adaption, due to the higher granularity of the transmitted data (we have smaller octree leaf cubes that go through the pyramidal decomposition/priority transmission), and provides more control over the process; the only downside being the increased computation time, which remains within reasonable margins.

Compared to modern codecs, it introduces undisturbing artifacts in strongly-encoded, highly-dynamic scenes and is able, at moderate compressions, to offer good quality, single-frame snapshots.

5. Future Work

Future work will be aimed firstly at correcting currently-observed issues, secondly at improving the algorithms, with some ideas detailed below, and finally, at performance analysis and comparison with standard methods.

The two observed issues are the blocking artifacts at the edges of the resolution trees and the wrongly-detected regions of interest. The first problem can be diminished by using a better overlapping technique for neighboring blocks. For the second, we will investigate different statistics and heuristics. One idea is to estimate how different a region is than the rest of the image. Studies have been undertaken in this domain [9,10] and we will try to include one that offers a favorable balance between complexity and speed.

An improvement to be implemented is the scaling of the hypercube relative to the distance between key-frames.

After that, a further generalization of the proposed algorithm is again possible: to ensure a full coverage of the video 3D space with hypercubes at arbitrary resolutions and with arbitrary downsampling-upsampling ratios. The new method will be more demanding in terms of computational effort, but we expect to give an even better control over the overall data processing and the removal of the blocking effect by enabling adequate (and allowing different) sizes for neighboring blocks to overlap. The regions of interest must also be calculated in a different manner, and the artifacts at the edges of the cubes must be diminished by overlapping neighboring blocks.

After all, the proposed improvements will be integrated in the current demonstrator, it will be necessary to create objective metrics for performance investigation of the proposed solution in different situations and for comparing it with state-of-the-art techniques.

Acknowledgments

The authors would like to thank Mihai Zaharescu and Andrei Tigora for their great ideas, support and assistance with this paper.

Author Contributions

This paper was the result of collaboration between the two authors. The research theme and idea were proposed by Costin-Anton Boiangiu and further refined by Adrian Enache. Adrian Enache was

mainly involved in the demonstrator application development and Costin-Anton Boiangiu in fine tuning and testing. Both authors contributed to the writing of the paper, literature review, test scenarios and discussion of the obtained results. Both authors have read and approved the final version of the manuscript.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Yang, H.-J.; Lin, H.-C.; Wang, Y.-D.; Kuo, L.-H. Designing and constructing live streaming system for broadcast. In Proceedings of the 2010 American Conference on Applied Mathematics (AMERICAN-MATH'10), Harvard University, Cambridge, MA, USA, 27–29 January 2010.
2. Soliman, H.H.; El-Bakry, H.M.; Reda, M. Real-time transmission of video streaming over computer networks. In Proceedings of the 11th WSEAS International Conference on Recent Researches in Communications, Electronics, Signal Processing and Automatic Control, Cambridge, UK, 22–24 February 2012; pp. 51–62.
3. Enache, A.; Boiangiu, C.-A. Adaptive video streaming using residue hypercubes. In Proceedings of the 12th WSEAS International Conference on Circuits, Systems, Electronics, Control & Signal Processing (CSECS '13), Budapest, Hungary, 10–12 December 2013; pp. 173–179.
4. Burt, P.J.; Adelson, E.H. The Laplacian Pyramid as a Compact Image Code. *IEEE Trans. Commun.* **1983**, *31*, 532–540.
5. Kim, W.-S.; Kim, H.M. Residue sampling for image and video compression. In Proceedings of the Visual Communications and Image Processing 2005, Beijing, China, 12 July 2005; Volume 5960, pp. 12–18.
6. Song, X.; Neuvo, Y. Image compression using nonlinear pyramid vector quantization. *Multidimens. Syst. Signal Process.* **1994**, *5*, 133–149.
7. Vandendorpe, L.; Macq, B. Optimum quality and progressive resolution of video signals. *Ann. Télécommun.* **1990**, *45*, 487–502.
8. Boiangiu, C.-A.; Olteanu, A.; Stefanescu, A.V.; Rosner, D.; Egner, A.I. Local thresholding image binarization using variable-window standard deviation response. In Proceedings of the 21st International DAAAM Symposium, Zadar, Croatia, 20–23 October 2010; pp. 133–134.
9. Osberger, W.; Rohaly, A.M. Automatic detection of regions of interest in complex video sequences. In Proceedings of the Human Vision and Electronic Imaging VI, San Jose, CA, USA, 20 January 2001; Volume 4299.
10. Kapsalas, P.; Rapantzikos, K.; Sofou, A.; Avrithis, Y. Regions of Interest for accurate object detection. In Proceedings of the International Workshop on Content-Based Multimedia Indexing, London, UK, 18–20 June 2008; pp. 147–154.