8th International Conference on DEVELOPMENT AND APPLICATION SYSTEMS
Suceava, Romania, May 25 – 27, 2006

A NEW APPROACH TO VIDEO COMPRESSION USING THE 3D-DCT ALGORITHM

Radu RĂDESCU, George PĂUNA
"Politehnica" University of Bucharest
1-3, Iuliu Maniu Blvd, sector 6, Bucharest
rradescu@atm.neuro.pub.ro

Abstract. *The present paper proposes an application in video signal compression based on a new variation of the 3-Dimensional Discrete Cosine Transform (3D-DCT). The described application, called RME, was conceived in Microsoft Visual Studio 6.0, using Visual C++ 6.0, and ported into Microsoft Visual Studio .NET 2002 and Visual C++ 7.0. RME implements an original version of the 3D-DCT compression algorithm, making a series of improvements to the standard algorithm. The main improvement is the creation of time-variable-length 3-dimensional blocks having other dimension than the standard one (8×8×8). The other improvements refer to the way of reducing the information bits when a file is saved. The application uses several external libraries to obtain some facilities: using BMP files as input files and adopting hardware conversion of the color space The object-oriented structure of the application is detailed and an example of the encoding-decoding procedure is presented. The parameters of the compression (compression ratio, duration of encoding and decoding) are computed*
Keywords: *video processing, compression algorithm, encoding-decoding procedure, 3D-DCT, multimedia.*

Introduction

During the last years, the video signal has been increasingly used in its digital format [1], due to the extent of the Internet. The symmetrical multimedia applications (e.g., video conference, video telephone, etc.) need the following steps of the video image processing:

- capturing and the real-time compression on the source equipment;
- transmission on the network;
- decompression and real-time playing on the receiver equipment.

The asymmetrical multimedia applications (e.g., video-on-demand, interactive television, etc.) need the following steps of the video image processing:

- storing on the server in a compressed format;
- transmission on the network;
- decompression and real-time playing on the receiver equipment.

In the case of interactive multimedia applications [2], [3], the image is usually recorded in a compressed format on CD-ROM and afterwards decompressed and played in real-time.

The 3D-DCT Algorithm

The 3D Discrete Cosine Transform (DCT) [4] is used in video compression when the basis principle is the correlation between the adjacent pixels in a frame (spatial correlation), and the correlation between the pixels of the same position in adjacent frames (temporal correlation). The principle is based on the idea that a video sequence could be seen as a tri-dimensional block of parameters W, H, and T.

W and H represent the width and the height of a frame, respectively, measured in number of pixels, and T represents the length of the video sequence, measured in number of frames (usually, T = the length in seconds × 30 or 25). This tri-dimensional block, representing the entire video sequence, is divided in smaller blocks of dimension 8×8×8, which are independently encoded. The encoding is performed using the 3D-DCT, as follows. The 3D-DCT is defined like:

$$F_{u,v,w} = \frac{1}{8} \cdot C_u C_v C_w \cdot \sum_{x=0}^{7} \sum_{y=0}^{7} \sum_{z=0}^{7} s_{x,y,z} \cdot D(x,u) \cdot D(y,v) \cdot D(z,w), \quad (1)$$

where:

- $s_{x,y,z}$ is the value of an image element (pixel);
- $F_{u,v,w}$ is the corresponding 3D-DCT coefficient;
- C_u, C_v, C_w are constant values ($C_u = C_v = C_w = 0.7071$ for $u = v = w = 0$, and $C_u = C_v = C_w = 1$ for all $u > 0$, $v > 0$, $w > 0$).

The function D has the value:

$$D(a,b) = \cos \frac{(2a+1)b\pi}{16}, \quad (2)$$

where $a \in \{x, y, z\}$ and $b \in \{u, v, w\}$.

After the coefficient computation, a non-uniform quantization is applied, speculating the limitation of the human eyed, which is less sensitive at high frequencies. Similarly, the inverse 3D-DCT is defined as:

$$s_{x,y,z} = \frac{1}{8} \cdot \sum_{u=0}^{7} \sum_{v=0}^{7} \sum_{w=0}^{7} C_u C_v C_w \cdot F_{u,v,w} \cdot D(x,u) \cdot D(y,v) \cdot D(z,w). \quad (3)$$

The block scheme of the encoder and the decoder is represented in Figure 1.

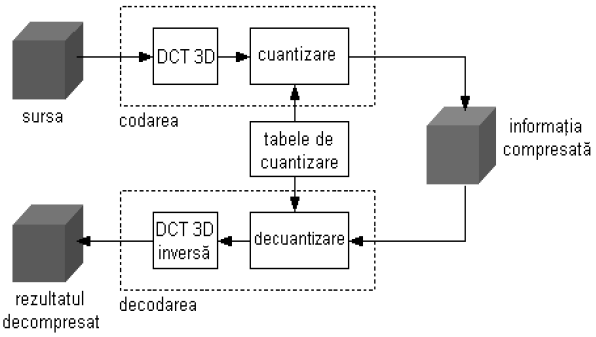

Figure 1. The block scheme of the encoder and the decoder for the 3D-DCT.

The source video signal is divided in video cubes of dimension 8×8×8 and then the above mentioned algorithm is applied. The quantization tables must be known both by the encoder and the decoder, in order to perform the decompression of the video signal.

The RME Application

RME is an application conceived in Microsoft Visual Studio 6.0, using Visual C++ 6.0. Afterwards, the application was ported into Microsoft Visual Studio .NET 2002 and Visual C++ 7.0.

The application uses several external libraries for some facilities. These facilities are:

- Open GL (GLaux), in order to use BMP files as input files;
- DirectX (DirectDraw), in order to benefit the hardware conversion of the color space: YUV → RGB, if available (if not available, the conversion is performed software, user-transparent) [5].

The application implements an original version of the 3D-DCT compression algorithm, making a series of improvements to the standard algorithm. The main improvement is the creation of time-variable-length tri-dimensional blocks having other dimension than the standard one (8×8×8). The other improvements refer to the way of reducing the information bits when a file is saved.

The structure of the RME application

The RME application has an object-oriented structure, with the following classes implemented.

RAdaptiveCODEC is the main class of the application. It implements both the encoding (AddFrame(...)) and the decoding at high level (RetrieveFrame(...)).

RArea is a class which implements the transforms performed on an 8×8 pixels area within a frame. It implements the color space transforms, both YUV → RGB and RGB → YUV (needed at reading AVI files, because they use Big Endian memory layout). 8×8 values for Y and 4×4 values for U and V, respectively, are stored in the YUV space (these values are obtained by average operation). It also implements different post-processing effects: dots randomly spread out on the image and reducing the abrupt transitions between adjacent areas by edge-average operation for the pixel values (reducing the blocking effect, specific to DCT-based algorithms). The 2D-DCT for each area is performed here, and the coefficients are used in the RAreaBlock class in order to compute the 3D-DCT. The quantization of the DCT coefficients is performed too.

RAreaBlock implements a list (a block of RArea class) and deals with saving the information on the disk, loading a list of RArea class from the disk, and encoding the information with the 3D-DCT.

RAreaBlockListElement implements a list of blocks from the RArea class.

RFrame implements a frame from the video sequence and contains a table from the RArea class of dimension W×H. The frame can be read from a BMP file using the CreateFromBitmap(...) function or it can be constructed from a memory region using the CreateFromRawData(...) function. The MakeAreaNeighbours(...) function creates the links within the RArea class in order to implement the effect of reducing the abrupt transitions between adjacent areas by edge-average operation for the pixel values.

RIFileBuffer implements the reading from the file, which is performed by large blocks, in order to avoid accessing the file repeatedly for short read operations.

RIndicatorBlock implements a marker in the file, which stores information regarding a specific block from the RArea class.

RIndicatorMarker implements a marker in the file, which stores information regarding a specific position within the file or a specific placement within the file.

RIStream permits reading of a certain number of bits from a file (not necessarily a multiple of 8). Thus, if it is necessary to save a variable having values between 0 and 2n – 1, this value is saved on n bits, reducing the number of bits when n is not a multiple of 8.

RMovie implements the encoding the decoding function for the video sequences at the highest level.

RMovieDescriptor deals with the characteristics of the encoded video (format, version, frame dimensions, frame rate).

ROFileBuffer implements the writing from the file, which is performed by large blocks, in order to avoid accessing the file repeatedly for short write operations.

ROStream allows the reading of a certain number of bits from the file (not necessarily a multiple of 8). Thus, if it is necessary to save a variable having values between 0 and 2n – 1, this value is saved on n bits, reducing the number of bits when n is not a multiple of 8.

RWindow deals with the display on the screen of a frame within the video file. It uses WinAPI in order to create a window and afterwards to draw in that window the frame content. It also displays some information about the frame (for instance, the number of the current frame).
RWindowX is similar to the RWindow class, with the difference that it uses the DirectDraw technique (from the DirectX package) in order to display the frame content.

The Operating Mode of the Application

In order to encode a video file, the RMovie class is instantiated and afterwards a function within the class is called, CreateFromAVI(...) or CreateFromBitmaps(...), which accepts as input file an AVI file and or sequence of BMP files, respectively. Using the input file, it creates a file containing the encoded video information, implementing the 3D-DCT.

One can remark the use of the RMovie class in order to encode an AVI file and to decode a file in the application-specific format RME, respectively.

The function CreateFromAVI within the RMovie class instantiates the RWindow class or the RWindowX class (depending on the selected display method) and obtains each frame from the AVI file using the CreateFromRawData(...) function of the RFrame class. Afterwards, it transforms the pixels from the RGB color space into the YUV color space using the YUV_TransformFromBGR(...) function of the RFrame class, it displays them and finally sends them to the AddFrame(...) function of the RAdaptiveCODEC class. The encoding window is presented in Figure 2.

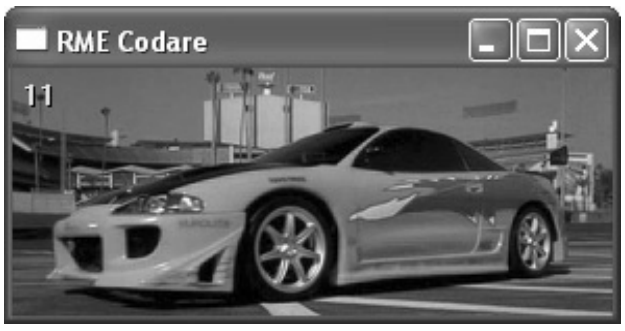

Figure 2. The encoding window of the RME application.

The PlayFromFile function within the RMovie class performs the decoding and displays on the screen the file compressed using the 3D-DCT. Each frame is obtained using the RetrieveFrame(...) function of the class RAdaptiveCODEC. The decoding window is presented in Figure 3.

Figure 3. The decoding window of the RME application.

The AddFrame function of the RAdaptiveCODEC class adds to the block list each RArea from the frame received as parameter. If a block detects a major change in the video information, it decided that the current block is finished and it starts a new block. In the same time, it checks whether the already finished blocks can be saved on the disk. This check is performed by the CheckBlocksAndSaveToStream function.

The CheckBlocksAndSaveToStream function is very important because it deals with the writing of the block in the same order they will be needed at the reading operation (an extra search is not necessary at the decompression). Thus, it saved the least recently performed block although it is not necessarily the last finished one. The SaveFirstToStream function saves the first block after it is encoded. The Encode function deals with the encoding. It calculates the 3D-DCT coefficients, using a pre-computed matrix, in order to optimize the computing speed. First, the 2D-DCT coefficients for each RArea are calculated. There is also a version of integer computing, which is faster, but it is less precise. This function and its pair, DecodeCurrent, take most of the processing time, as it could be observed in the percentage histogram represented in Figure 4.

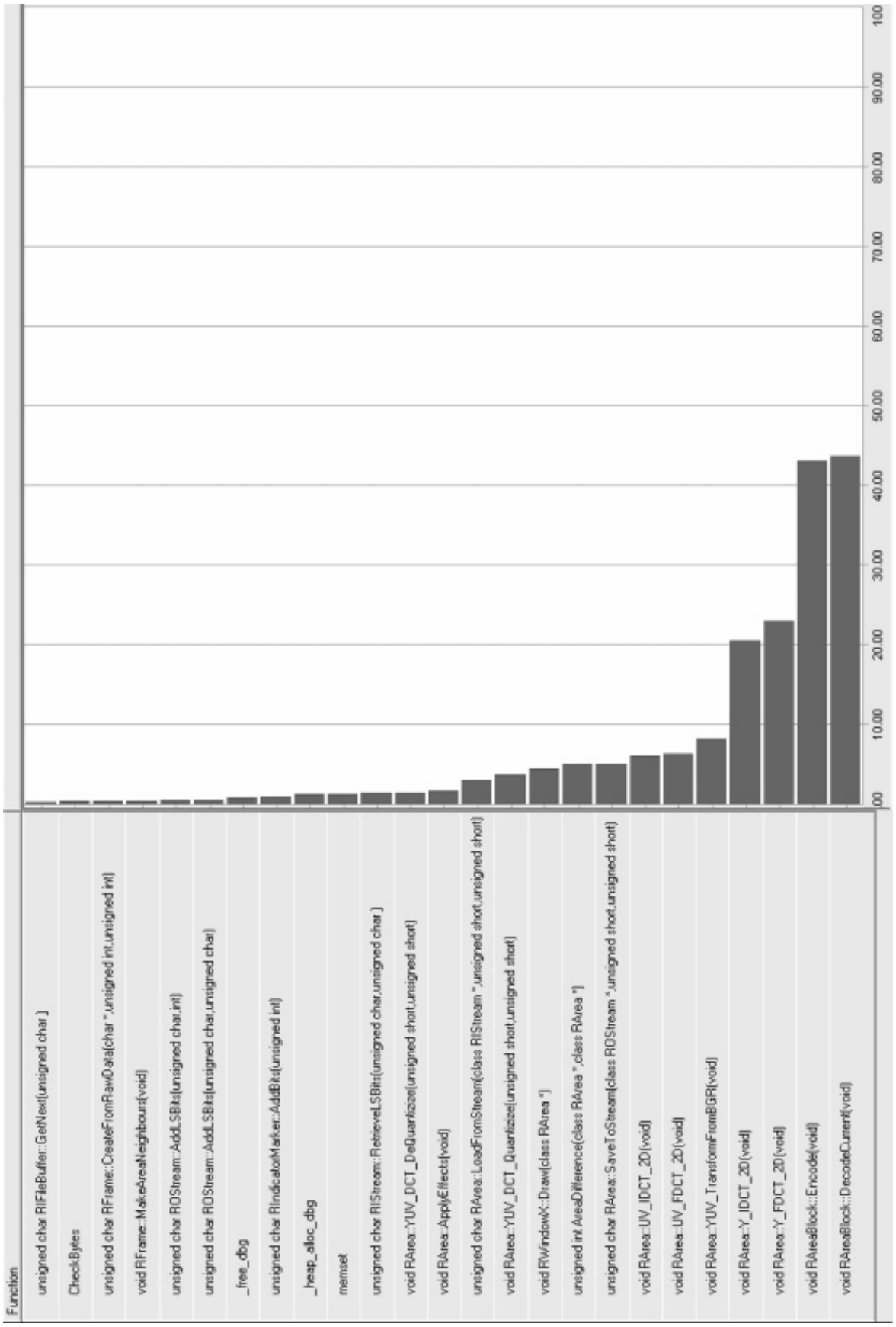

Figure 4. The duration of the called functions within the encoding-decoding procedure [%].

The pair function of the AddFrame function is called RetrieveFrame. It gets from the block lists all the needed RAreas in order to reconstruct the current frame.

At the end of the encoding and decoding procedure a window is displayed, containing information about the duration of the encoding and decoding process, and the achieved compression ratio (see Figure 5).

Figure 5. The window containing the parameters of the encoding-decoding process.

Conclusions

The paper proposes an application in video signal compression based on an original version of the 3D-DCT compression algorithm, making a series of improvements to the standard algorithm. The main improvement is the creation of time-variable-length 3-dimensional blocks having other dimension than the standard one.

The other improvements refer to the way of reducing the information bits when a file is saved. The application uses several external libraries to obtain some facilities: using BMP files as input files and adopting hardware conversion of the color space. Compared to the MPEG standard [6], the use of the compression algorithm proposed here has the benefit of the most economical manner of using the information bits. Multiple multimedia applications of this algorithm could be devised.

References

[1] Society for Imaging Science and Technology: http://www.imaging.org/resources/jpegtutorial/index.cfm
[2] Video Compression: MPEG-4 and Beyond: http://www.cse.ohio-state.edu/~jain/cis788-99/ftp/compression/index.html
[3] Video Compression Tutorial: http://www.wave-report.com/tutorials/VC.htm
[4] Video Compression using Three-Dimensional Discrete Cosine Transform: http://www.dip.ee.uct.ac.za/~marcs/
[5] Color Video Signals Compression based on 3D-DCT Transform: http://www.elektrorevue.cz/clanky/03009/english.htm
[6] Rădescu, R. (2003) *Compresia fără pierderi – metode şi aplicaţii*, Bucureşti, Editura Matrix Rom.