3D Fractal Compression for Real-time Video

Alexandre Chabarchine
Ural State Technical University
Department of Automation and Information Technology
Mira St. 32
620002 Ekaterinburg
Russia
saa@rtf.ustu.ru

Reiner Creutzburg 1,2
1 Fachhochschule Brandenburg
University of Applied Sciences
Department of Computer Science and Media
P. O. Box 2132
D-14737 Brandenburg
Germany
creutzburg@fh-brandenburg.de

2 Tampere University of Technology
Digital and Computer Systems Laboratory
P. O. Box 553
FIN-33101 Tampere
Finland
creutzbu@cs.tut.fi

April 7, 2001

Abstract

A method of image sequence encoding is presented, based on fractal compression of 3D data blocks without searching. A simple fractal transform scheme is used that only depends on one real valued coefficient. We have developed a software codec for the Windows95 operating system, that carries out video streams in real-time. The method can be further developed for the application to videoconferencing, videophone or streaming video over Internet.

1 Introduction

At present the computer and information technologies play the important role in many fields of the human life. Image processing is very often becoming a significant part of modern computer application. High-quality digital images and video without encoding are represented by huge data arrays.

Different compression methods are used to decrease the amount of data. The fractal compression is known to provide good representation of natural images with a small amount of data [2]-[10]. The fractal compression is highly related to iterated function systems (IFS) which are used for image approximation and compression. Furthermore, from the theoretical point of view the fractal transform is related to the wavelet transform approach that is well-known in image and video compression [4, 10, 11].

The problem of generating a code from the original image is called the inverse problem for image approximation. Traditional methods of fractal compression use a search to solve the inverse problem.

The presented method of fractal image compression is based on the idea of fractal compression without any search [9]. A video sequence can be presented
as a 3D array of pixels. Therefore, video encoding can be implemented as 3D data encoding.

2 Theory

Various methods of image compression are based on elimination of any form of redundancy. In particular fractal methods eliminate the form of redundancy as a self-similarity. The self-similarity is considered to be a property of all natural objects and their images [11], and elimination of the form of redundancy can extremely reduce the amount of the encoding data.

Fractal image encoding is based on the hypothesis, that it is possible to find the local self-similarity of parts for any image.

Existing algorithms of fractal compression use the following scheme [10]. The image that has to be encoded is divided into a set of nonoverlapping blocks, so-called 'ranges'. A larger block (so-called 'domain') is found for each of the ranges. The domain approximates the range using a simple transform (usually affine):

\[ F = \bigcup_{i=1}^{N} F_i = \bigcup_{i=1}^{N} w_i(F), \]  

where the \( w_i \) are affine and contractive. Then, the image is represented only by the coefficients of the simple transforms which encode the image.

To decode the encoded image we use the principle of contractive mappings [1]. There is only one fixed point of the contractive mapping, the attractor. In our case the attractor is the decoded image. By starting the iteration process from any initial image, we can attract the image that was encoded.

3 Implementation

Let the pixels have grayscale levels from 0 to 255. The frame of the model is represented by a two-dimensional array of 64 to 64 pixels. Let the pack consist of 16 frames. Therefore, one pack has a size of 64 kilobytes. The pack is divided into 16 cubes, as presented in fig. 1 left. Then we will divide any cube into 8 subcubes, as presented in fig. 1 right.
ness transform, and \( < d > \) denotes the average level of brightness in the domain. If we fix the coefficient \( a \) as a constant 0.6, then we can calculate the coefficient \( b \) as the average level of brightness in the possible range. The encoding process for the cube starts from dividing of the cube into the 8 subcubes. Considering each subcube we assume that a transform from every cube to each subcube exists. The assumption is tested by substitution of the average brightness level in the subcube \( < r > \) as \( b \) coefficient in (2). Let \( M \) be the maximal error for decoding, that is calculated as the maximal difference from real brightness and calculated brightness. If the maximal difference for the considered subcube is less than \( M \), then we assume that the subcube is at the end of a branch of the octree, else we divide the subcube into 8 subsubcubes and continue the encoding process.

4 Experimental results

The encoding of one pack on computer with a simple processor Pentium-60 is executed from 2 to 4 seconds (4-8 frames per second).

The decoding program must execute simultaneously the two following processes: decoding of the pack and viewing of the decoded pack. On a computer with processor Pentium-60 the time of decoding of one pack can vary from 2 to 5 seconds (3-8 frames per second), and the speed of the demonstration, at the same time, should remain constant.

The program of decoding and demonstration is developed for the Windows-95 operating system (fig. 3), which has advanced possibilities for the execution of parallel processes in one program and the synchronization of parallel processes.

The program starts two parallel processes: the process \( A \) and the process \( B \) (fig. 4). The process \( A \) executes reading from file and decoding of a pack, and the process \( B \) shows already the decoded pack of frames.

The synchronization of processes is proceeded as follows. As shown in fig. 2, at the beginning of the program the process \( A \) is loading and decoding the first pack, proceeding 5 iterations. At the same time the process \( B \) expects the decoding end. The period \( T_0 \) of decoding the first pack is stored. Outcoming from value \( T_0 \) the program calculates the duration of time for one frame \( t_k = T_0/15 \). The process \( B \) views the frames of the pack, decoded in the previous step, with period \( t_k \). At the same time the process \( A \) is reading and decoding the following pack. At the end of the "film", when all packs have processed and process \( B \) begins to show the frames from the last packs, the process \( A \) can again start to read the first pack of the "film". Thus, the "film" is closed in a ring without appreciable delay between the last and the first frames.

The speed of the reproduction depends on the productivity of the computer and the size of the window, but by software it is limited to about 16 frames per second (1 pack per second). At testing on computer
Table 1: Dependence of the encoding results on the threshold $M$

<table>
<thead>
<tr>
<th>$M$</th>
<th>encoding time, sec</th>
<th>decoding time, sec</th>
<th>rate</th>
<th>quality of result</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4.1</td>
<td>3-5</td>
<td>4.7</td>
<td>excellent</td>
</tr>
<tr>
<td>24</td>
<td>5.4</td>
<td>3-5</td>
<td>6.9</td>
<td>good</td>
</tr>
<tr>
<td>32</td>
<td>5.8</td>
<td>3-5</td>
<td>10.2</td>
<td>not bad</td>
</tr>
<tr>
<td>40</td>
<td>7.1</td>
<td>3-5</td>
<td>15.3</td>
<td>bad</td>
</tr>
<tr>
<td>48</td>
<td>8.5</td>
<td>3-5</td>
<td>23.2</td>
<td>very bad</td>
</tr>
</tbody>
</table>

with processor Pentium-60 the speed of frames was within the limits of 3-8 frames per second.

The Table 1 shows the dependence of the coding speed, the decoding speed and the compression rate on the threshold $M$. An acceptable quality is reached if $M < 32$ (for brightness values from 0 to 255).

Figure 5: Increasing image resolution

Since a fractal compression allows an increase of the image resolution without granularity, we can get more then 16 frames inside of one pack, using the same encoding data (fig. 5). In near future our work will be concentrated on the optimization of the algorithm and on using images with higher resolution.

References


