Progressive Image Compression Using Binary Trees

Denis V. Ivanov, Eugene P. Kuzmin, Sergey V. Burtsev

Mathematics and Mechanics Dept., Moscow State University

Moscow, Russia

Abstract

In this paper, we present an approach of using binary tree structures for efficient progressive image compression. General ideas of progressive transmission and image approximation are discussed. We also present a specific technique, which provides compression ratios comparable to the ratios of the best algorithms known in the literature. Besides, it is computationally simple and specially designed for fast progressive reconstruction of image sketches. Practical results are also present in this paper.

Keywords: *Image, Lossy, Lossless, Progressive Image Compression, Binary Tree*

1. INTRODUCTION

Progressive Image Compression refers to image compression techniques that allow both: (1) original image reconstruction without loss of any details; and (2) construction of picture approximations (sketches) with the accuracy level depending on the amount of data available.

Lossless compression is highly important for images obtained at a great cost, such as space or medical images. In this case, even marginal loss of data may destroy some details required during further processing, or add artifacts that lead to erroneous interpretation.

However, frequent incremental visual inspections of images may also be necessary. By the term "image inspection" we mean here fast extraction of image sketches with desired levels of accuracy. The speed of this operation depends on the amount of data that should be retrieved from the storage for sketch reconstruction, but with data transferring through potentially many network connections, it may not be fast enough. Nevertheless, in the case of non-progressive compression, neither an approximation nor the original picture is available until the whole image data is retrieved.

For these reasons, many applications require an image compression method that would perform:

- Lossless image compression with good compression ratios in order to use data storage more efficiently;
- Reconstruction of an image approximation if only a beginning part of a data stream is available. Obviously, it is preferable to obtain good-looking sketches by extraction of just small parts of the compressed data.

Typically, users had to choose different methods depending on whether the compression or fast inspection is desired.

Commonly used lossless compression techniques, such as GIF or PING, show very good performance, but all they can offer in order to provide progressive decompression is an interlaced order for pixel transmission. For example, the top-left pixels of 8x8 blocks can be stored in first pass; they are then followed by the pixels missed to produce blocks of 4x4, and so on. Having received the first part of data, the decoder can reconstruct an image consisting of relatively large blocks filled with the colors of their top-left pixels. This approximation, obviously, has insufficient visual perception quality for multicolor images of high frequency.

For lossy compression, but fast extraction of an image, some other methods, such as JPEG or wavelets, may be used. These techniques provide very good compression ratios, but the original image may not be reconstructed completely and significant details may be lost.

This paper presents a new and efficient technique that successfully achieves both objectives - progressive image transmission and (ultimately) lossless compression. It is based on the hierarchical principle of a binary tree data structure and provides compression ratios comparable to the best algorithms known in the literature.

Some compression methods also use hierarchical structures in order to perform progressive compression. One of the best known algorithms is the S (S+P) transform [1], which assumes that, in the general case, the entropy of an image can be decreased by applying special transformations (non-linear, but still reversible). Thus, an image with lower entropy may be more efficiently compressed with entropy coding methods, such as Arithmetic or Huffman coding. However, this technique performs progressive reconstruction on a block basis regardless of the approximation quality. To solve this problem and increase the compression ratio, a set of partitioning schemes [2] may be applied. This modification (in the case of lossless compression) produces very good results, but still requires entropy encoding and some computational effort for sketch visualization.

The technique presented in this paper is characterized by the following properties:

- Lossless compression ratios are comparable to the best ratios known in the literature.

- Fast and simple compression/decompression algorithm.
- Image approximations can be progressively updated in parallel to data extraction. When all the data is received, the entire original image will have been already constructed, this while previous (non-complete) versions of the image are used in the interim.
- As each moment passes, the decoder can estimate, without additional computations, the difference between the currently reconstructed sketch and the original, in terms of mean square error (MSE) or PSNR.
- No entropy coding or other post-processing is required.

This paper is organized as follows. The next section describes how binary tree structure can be used for raster image representation. It presents the general concept of tree usage for efficient compression, storage, and progressive reconstruction. Section 3 presents the, so-called, '3-ranged binary tree' technique, which, by our estimation, is mostly efficient in terms of compression ratios and visual quality. This section along with Appendix A includes some practical results in comparison to other methods. The conclusion of the paper is in Section 4.

2. BINARY TREE USAGE

Here we describe how the binary tree data structure can be used in order to represent raster images. For simplicity of explanation, we consider only GrayScale (8 bits/pixel) images; however, all the following assertions can be easily adapted to TrueColor (24 bits/pixel) images considering each color component separately.

2.1 Images in 3D space

Let us consider the color (luminance) attribute of each pixel as it's third coordinate in 3D space (the other two dimensions being screen space X and Y). Thus, an image may be represented by graph of the function

$C(x,y) = ColorOfPixel(x,y)$, (see Figure 1).

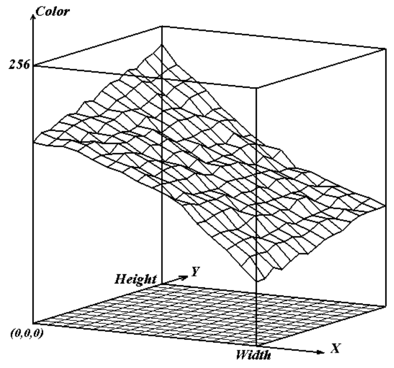


Figure 1. GrayScale 8bpp image considered a surface in 3D space.

Obviously, considering 8bpp images, the color of the pixels does not exceed 256; therefore, the whole image is sure to be lying inside the parallelepiped

$$\pi = [0, Width] \times [0, Height] \times [0,255].$$

2.2 Binary tree construction

2.2.1 Contraction scheme

Suppose we have a finite set of contraction operators defined on the segment [0,1]. Adding the identity operator to this set, we denote it with Ψ. Thus,

$$\Psi = id \cup \{\varphi_i, i = 1...N \mid \forall i, \varphi_i([0,1]) \subset [0,1]\}.$$

Considering any segment [*a,b*] and denoting it's linear transformation to [0,1] with *l*

$$l : l([a,b]) = [0,1],$$

we can define $\varphi_{i,i=0...N}$ on [*a,b*] by the formula

$$\varphi_i([a,b]) = l^{-1} \circ \varphi_i \circ l([a,b]).$$

Thus, we obtain the set of operators which map any segment [*a,b*] into the segment [*c,d*]⊆[*a,b*]. However, Ψ is required to have the following property in order to be used for image representation with a binary tree structure.

Definition 1. The union of the identity operator and a finite set of contraction operators defined on [0,1] is called a *contraction scheme* if

$$\forall \varepsilon > 0, \forall P \in [0,1], \exists i_1 ... i_M : \\ P \in \varphi_{i_1} \circ ... \circ \varphi_{i_M}([0,1]) \quad \vee \quad Diameter(\varphi_{i_1} \circ ... \circ \varphi_{i_M}([0,1])) < \varepsilon$$

where *Diameter*(·) refers to the length of the segment.

The simplest example of a contraction scheme is one corresponding to the binary subdivision method of point localization. It is the following

$$\Psi = \begin{cases} \varphi_0([0,1]) = [0,1] \\ \varphi_1([0,1]) = [0, 1/2] \\ \varphi_2([0,1]) = [1/2, 1] \end{cases}$$

Another example of the contraction scheme will be considered in the next chapter.

2.2.2 Binary tree construction

We discuss here a general procedure that is used to construct the binary tree, which represents a raster image. In the beginning, we are given an image considered as a surface *C*(*x,y*) in 3D space (Figure 1), and a contraction scheme Ψ.

Each node of a tree corresponds to a parallelepiped

$$\Pi = [x, x + width] \times [y, y + height] \times [c, c + depth]$$

which complies with the two following properties:

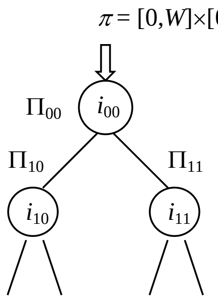
(1) $\Pi \subseteq \pi$, where π is the parallelepiped containing the whole image (see Figure 1);

(2) $C([x, x + width] \times [y, y + height]) \subset \Pi$, i.e. the part of the image which corresponds to the XY rectangle of Π lies inside Π.

Initially, the root of the tree corresponds to *π*. For each node, we perform the following steps:

- Check the *depth* of Π. If it is equal to 1, we stop construction of the tree branch, because according to condition (2) we are guaranteed that all points of the surface within the XY rectangle of Π belong to it. Therefore, the flat region of the surface has been accurately approximated.
- Select a contraction operator from Ψ (not the identity one, if possible), so that if it is applied to the [*c*,*c*+*depth*] segment (see Figures 2,3), the resulting parallelepiped complies with (1) and (2) requirements. The identity operator does not change Π, so it may be chosen in any case. An identifier for selected operator is stored in the node.
- Construct children. If the *width* or *height* of Π is greater than 1, we can split Π into two equally sized parallelepipeds bisecting the X or Y side depending on which is greater (see Figures 2,3). These two parallelepipeds, corresponding to the children, are processed in the same manner.

Thus, the algorithm of tree building is recursive, generating children at each node, and applying the same procedure to them. It cyclically bisects the bounding parallelepipeds in the X and Y directions and contracts it's color side whenever possible. The process of subdivision is stopped if the original image is localized and accurately approximated.



$$\Pi_{00} : [0,W]\times[0,H]\times\varphi_{i_{00}}([0,255])$$

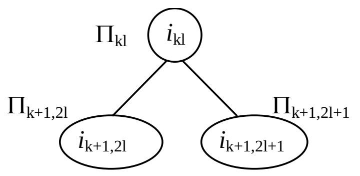
$$\Pi_{10} : [0,W/2]\times[0,H]\times\varphi_{i_{10}}\varphi_{i_{00}}([0,255])$$

$$\Pi_{11} : [W/2,W]\times[0,H]\times\varphi_{i_{11}}\varphi_{i_{00}}([0,255])$$

$$\Pi_{20} : [0,W/2]\times[0,H/2]\times\varphi_{i_{20}}\varphi_{i_{11}}\varphi_{i_{00}}([0,255])$$

$$\Pi_{21} : [0,W/2]\times[H/2,H]\times\varphi_{i_{21}}\varphi_{i_{11}}\varphi_{i_{00}}([0,255])$$

Figure 2. Binary tree construction. Starting from the root.



$$\Pi_{kl} : [x, x+w]\times[y, y+h]\times\prod_{i=0}^{k}\varphi_{i,\left\{ l/2^{k-i} \right\}}([0,255])$$

$$\Pi_{k+1,2l} : \left[x, x+ w/2\right]\times[y, y+h]\times\prod_{i=0}^{k+1}\varphi_{i,\left\{ 2l/2^{k+1-i} \right\}}([0,255])$$

$$\Pi_{k+1,2l+1} : \left[x + w/2, x+w\right]\times[y, y+h]\times\prod_{i=0}^{k+1}\varphi_{i,\left\{ 2l+1/2^{k+1-i} \right\}}([0,255])$$

Figure 3. Binary tree construction. General case (bisecting in X direction).

Because Ψ is the contraction scheme (see Definition 1), any image may be represented by the tree constructed by this procedure. Indeed, if the algorithm reaches the 1×1×[*c*,*c*+*depth*] parallelepiped, it can stop bisections in X and Y directions, and just split the [*c*,*c*+*depth*] until the pixel color is localized.

More formally, the algorithm performs the following:

```
Node* GenerateNode( Π ) {
   if ( depth <= 1 ) return NULL;
   node = new Node;
   node->index = FindContraction(Π);
   Π = ApplyContraction(Π,node->index);
   Parallelepiped leftΠ=Π, rightΠ=Π;
   if ( Π.width > Π.height ) {
      leftΠ.width /= 2;
      rightΠ.width /= 2;
      rightΠ.x += rightΠ.width;
   }
   else if ( Π.height > 1 ) {
      leftΠ.height /= 2;
      rightΠ.height /= 2;
      rightΠ.y += rightΠ.height;
   }
   node->left = GenerateNode(leftΠ);
   node->right = GenerateNode(rightΠ);
}
```

As a result, a binary tree having one number (the index of a contraction operator) is stored (see Figure 4). Generally, it may not be height-balanced, because branch generation stops whenever the original image is localized by corresponding parallelepiped.

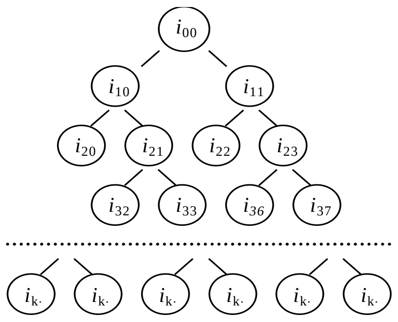


Figure 4. Binary tree as image representation.

2.3 Binary tree storage

Each node of a binary tree representing a raster image contains one value, which is the index of a contraction operator from the contraction scheme Ψ. These values along with the parent-children relationship allow precise reconstruction of the initial picture. Obviously, predecessors should be stored (extracted) before their ancestors. However, it may be achieved in different ways.

2.3.1 Tree with pointers

The simplest form of binary tree storage is writing node values followed by two pointers to the children. Although this approach provides both the node value and the parent-children relationship, it is very inefficient due to the necessity of storage space for 2 pointers per node. Because the contraction scheme typically consists of a small number of such operators, and, indexes require much less space then pointers, we consider this approach inefficient for storing the binary tree of an image.

2.3.2 Layer-by-layer storage

Another approach is based on the pyramidal form of a tree. The tree may be stored layer by layer, starting from the root (Figure 5). If the decoder knows which nodes are leaves, it can reconstruct the tree from this stream.

This approach is more efficient than the previous one, because it stores all the most valuable node information without any pointers. It is very simple and efficient. However, there exist other schemes that may be used more efficiently in terms of progressive visualization.

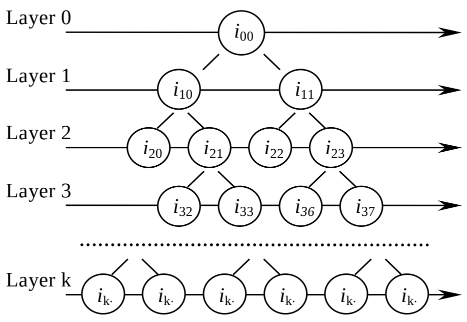


Figure 5. Layer-by-layer tree storage

2.3.3 Volume sorting

Because the original image is being approximated by parallelepipeds in 3D space, we can consider the volume of these parallelepipeds as a good estimation of the quality of the approximation.

$$V(\Pi) = width \times height \times depth\,.$$

Two parallelepipeds of equal volume, but different *depths* are considered identical in terms of visual quality. This is because the original image is guaranteed to be reconstructed with better accuracy with the larger XY rectangle than with the smaller one. Therefore, we can define a strict ordering relationship on nodes judging them by the volumes of the corresponding parallelepipeds. Denoting the node which corresponds to Π_{ij} with N_{ij}, we can write

$$\{N_{mn} > N_{kl}\} \Leftrightarrow \left\{\begin{matrix} (V(\Pi_{mn}) > V(\Pi_{kl}))\vee \\ (V(\Pi_{mn}) = V(\Pi_{kl}) \wedge m < k)\vee \\ (V(\Pi_{mn}) = V(\Pi_{kl}) \wedge m = k \wedge n < l) \end{matrix}\right\}$$

Having sorted all of the nodes, we can then store them in the following manner:

- The root goes first;
- Amongst all non-terminal nodes, whose children have not been stored yet, find the first (largest) node and store (transmit) its children (see Figure 6);
- Repeat the previous step until all nodes are stored (transmitted).

This approach provides better overall visual quality if only the beginning part of the data stream is available, because the mean square error (MSE) is more evenly distributed.

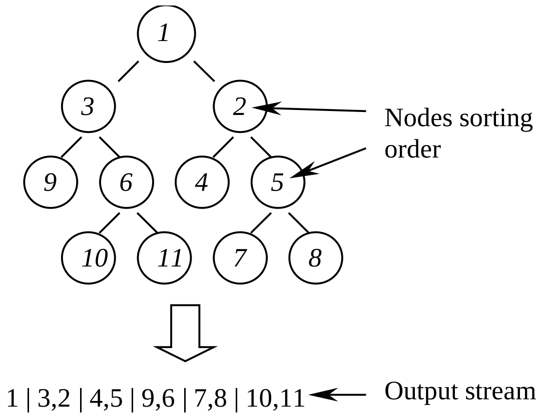


Figure 6. Output data stream considering volume sorting.

2.4 Progressive image reconstruction

The process of image reconstruction is the inverse of the process of tree construction. All these storage schemes start the data stream with the root value. The decoder initializes its initial rectangle with π (parallelepiped containing the whole image's surface) and begins the process of contraction in the color dimension and bisection in X and Y dimensions forming each node's children.

The correct order of operations is provided by the following properties of the storage schemes:

- The root always transmitted first;
- At every moment, the decoder determines which node is being transmitted by analysis of the parallelepipeds that have been already constructed; therefore, no additional information is required – the coder and decoder simply use the same algorithm for node extraction;
- Children of a node are always transmitted after their parents, providing the correct order of contractions.

Progressive visualization may be implemented in the following manner. Let us consider a decoder having received the beginning part of a data stream, so it reconstructs just the top part of a tree. Enumerating all leaf nodes of the reconstructed part of a tree with

$$\Pi_i = [x_i, x_i + w_i] \times [y_i, y_i + h_i] \times [c_i, c_i + d_i], i = 1...M ,$$

we can state that

$$\bigcup_{i=1}^{M} \left([x_i, x_i + w_i] \times [y_i, y_i + h_i] \right) = [0, Width] \times [0, Height],$$

where *Width* and *Height* refer to the width and height of the initial image, respectively. It should be emphasized, that by the leaf nodes of the reconstructed part of the tree, we mean it's nodes which are terminal in the full tree, or, which do have children in the full tree, but they have not been transmitted.

Besides, considering condition (2) in 2.2.2 we can deduce that

$$C([0, Width] \times [0, Height]) \subset \bigcup_{i=1}^{M} \Pi_i \text{ , i.e.}$$

the surface of an image belongs to the union of parallelepipeds corresponding to the leaf nodes. Therefore, by filling each [x_i, x_i+w_i]×[y_i,y_i+h_i] rectangle with the middle value equal to $c_i+d_i/2$, the decoder produces an approximation of an image. If the next portion of nodes is received, the decoder simply shifts the colors at the rectangles which are affected by new information. Thus, starting from the middle value of [0,255], which is spread over the whole image, the decoder, receiving node values of the tree, constructs more and more accurate approximations of an image. Having received the whole tree, it produces the precise reconstruction of the original.

3. 3-RANGED BINARY TREE

In this chapter, we present a contraction scheme that, by our observation, provides a very efficient progressive compression on images of different types. Efficiency is achieved in both compression ratio and computational complexity.

3.1 3-Ranged binary tree introduction

Guided by three desired properties, such as

- fast and high-quality image approximation;
- simple computations;
- efficient and compact storage;

we proposed to use the following contraction scheme:

$$\Psi_3 = \begin{cases} \varphi_0 : \varphi_0([0,1]) = [0,1] - \mathit{identity} \\ \varphi_1 : \varphi_1([0,1]) = [0, 1/2] - \mathit{bottom} \\ \varphi_2 : \varphi_2([0,1]) = [1/4, 3/4] - \mathit{middle} \\ \varphi_3 : \varphi_3([0,1]) = [1/2, 1] - \mathit{top} \end{cases}$$

Except for the identity operator, this scheme includes 3 operators which halve the length of the of the segment. The first one selects the bottom "half" of a segment, the second selects the middle "half", and the third contracts the segment into its top "half" (see Figure 7).

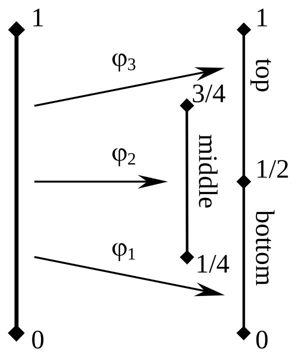


Figure 7. Contraction operators of the 3-ranged scheme.

Because the approximation of an image starts from the segment [0,255], which, in fact, can be considered as [0,256], all contraction operators may be applied using additions and bitwise shifts. Therefore, coding and decoding algorithms are very fast, very efficient, and they allow for hardware implementation.

Figure 8 shows the process of building a 3-ranged tree using a 1-dimentional function, for illustration purposes. In the case of a 2D surface, we just use the XY rectangles instead of segments, and the process of children generation results in bisecting the greater of the X or Y dimension.

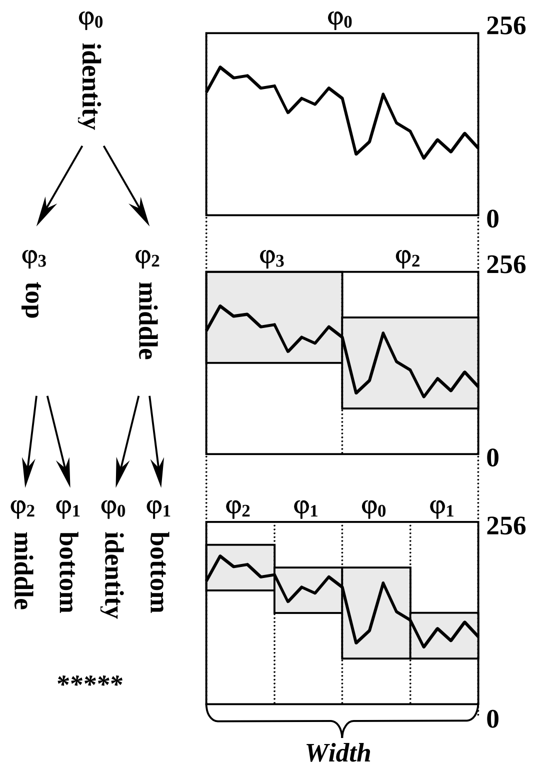


Figure 8. 3-ranged bynary tree construction.

3.2 Compact storage of the 3-ranged tree

As we have already proposed, the 3-ranged binary tree, which represents a raster image, may be stored in a layer-by-layer or volume-dependent order. However, for each node, we store the index of a contraction operator from the contraction scheme Ψ, which is used for tree generation. In the case of a 3-ranged tree, the contraction scheme Ψ_3 consists of 4 operators.

Having researched binary trees, constructed for images of several types, we propose to use two techniques in order to obtain high compression ratios, and, simultaneously, to fulfill all the desired conditions, such as progressive transmission. These techniques are tree limitation and multi-length node encoding.

3.2.1 Tree truncation

Our research showed that it is more efficient to construct a tree up to a certain level, which, however, varies from image to image. Before storing the tree, we can specify two parameters – the minimal parallelepiped's *depth* and the minimal XY rectangle's area, which is *width*×*height*. These parameters may also be layer number or minimal parallelepiped's volume. Having specified these limitations, we construct the tree branches until one of the minimal parameters is reached. To produce lossless compression, we can store residual parts of images belonging to the parallelepipeds, considering these parts as separate images. Typically, much less than 8 bits are required for each pixel, because the encoder and the decoder have all the parameters of the leaf parallelepipeds, and they assume that the corresponding image part lies entirely within them. For example, if a leaf parallelepiped is 2×2×16 in size, 4 bits for each pixel are required.

Thus, the storage of an image consists of two parts – the tree constructed up to a certain level, and the residual (see Figure 9), which provides for lossless compression. Generally, a tree is constructed up to the level which provides good visual quality, so all properties of progressive transmission are fulfilled.

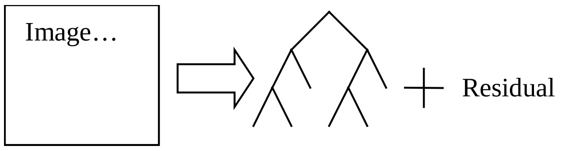


Figure 9. Tree with limits + residual storage scheme.

There are three good reasons for dividing image storage to the tree part and the residual:

- It is possible to achieve an optimal balance between compression ratio and progressive visualization.
- By our estimation, the residual, coded with the number of bits required, may not generally be encoded by entropy coding methods with better ratios. The residual may be considered noise in most cases.
- Having reconstructed the approximation from the tree, the decoder knows how the original differs from the reconstructed image.

3.2.2 Multi-length node coding.

As mentioned above, the residual may not be compressed with better ratios by entropy coding methods. However, these methods may be efficiently applied to the tree part. We propose a static Huffman encoding technique to store the tree nodes more efficiently.

From the histogram of node frequencies, a Huffman tree may be constructed. Then, all node indexes are substituted by unique prefix codes, which can be read one by one from the continuos data stream. By our estimation, the codes, presented in Table 1, are appropriate for most images.

Contraction operator	**Code**
φ_0: *identity*	0
φ_1: *bottom*	110
φ_2: *middle*	10
φ_3: *top*	111

Table 1. Codes of the 3-ranged tree nodes.

3.3 Practical results

We have chosen the well-known image of Lena as a test sample for analyzing visual characteristics and for comparing our compression ratios to other techniques. This image is a grayscale 8bpp image of 256×256 pixels in size.

The most suitable tree limits for this picture are 16 for the parallelepiped's *depth* (color range) and 4 for the XY rectangle area. The algorithm stops building tree branches whenever one of these parameters is reached. The remainder is stored as-is, just using the information about the number of bits required for each pixel. The tree is stored using Huffman encoding, as shown in Table 1.

With these limitations, the results for Lena are as following.

3.3.1 Image storage

Table 2 shows the storage requirements for the original Lena image and it's compressed version using a 3-ranged binary tree, constructed up to a certain level, and the remainder.

Image	**Size (in bytes)**
Original Lena (unpacked)	65536 (100%)
Tree (limits ÷ *depth* = 16, *area* = 4)	7332 (11%)
Residual	39515 (60%)
Tree + Residual	46847 (71%)

Table 2. Storage requirements for different parts of the image of Lena compressed with 3-ranged binary tree.

3.3.2 Compression ratio comparison

Table 3 shows the comparison of compression ratios obtained by packing the image of Lena with 3-ranged binary tree techniques and other well-known lossless compression schemes, including the best ones known in the literature.

Compression Technique	**Size (in bytes)**
SPIHT [2] (entropy coding included)	42140 (65%)
3-ranged binary tree	46847 (71%)
PING (interlacing, LZW-based)	47383 (72%)
Unpacked	65536 (100%)
PCX (RLE)	73088 (112%)
GIF (LZW)	74206 (113%)

Table 3. Comparison of compression ratios for the image of Lena.

3.3.3 PSNR of reconstructed sketches

Here we present the graph of PSNR (Peak Signal to Noise Ratio) characteristics of the difference between the original Lena image, and a variety of it's approximations, reconstructed from short initial sequences of compressed data. The visual quality of these images may be inspected in Appendix A.

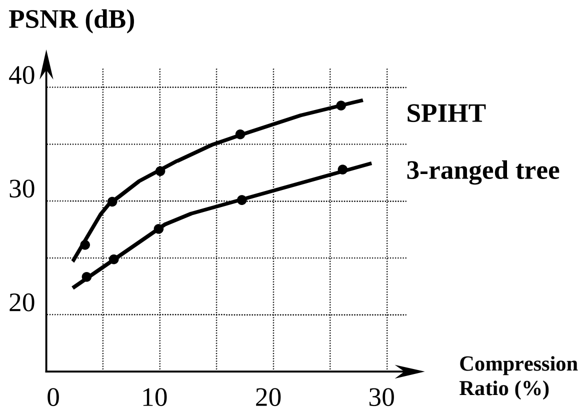


Figure 10. Comparison of PSNR for progressive visualization with 3-ranged tree and SPIHT

It should be emphasized, that the SPIHT technique was designed to minimize the mean square error (MSE) when only the beginning part of a message is available; therefore, it may be considered optimal in this sense. However, this method does not allow transmission and reconstruction in parallel, besides it is relatively more complicated.

4. CONCLUSION

In this paper, we have presented the basic concept of binary trees in order to obtain progressive compression of raster images. We also presented a specific technique, the 3-ranged binary tree compression, which is very efficient in terms of visual quality, compression ratio and computational complexity. Some practical results are also included.

The 3-ranged binary tree technique provides lossless compression with ratios, which are among the best known in the literature. We also estimate that our technique provides better compression ratios then commonly used algorithms such as PCX, GIF and PING.

The binary tree technique was designed to perform progressive transmission of compressed images. Image approximations may be constructed by a decoder in parallel with data reception, because, when the next portion of data becomes available, the decoder just improves the corresponding image parts, making it looking better. Besides, the decoder can automatically estimate the quality (PSNR) of the sketches obtained, without having the original or any additional information. For these reasons, this technique may be efficiently used for remote image inspections and observations.

The algorithm presented here is very efficient in terms of computational complexity. It may be implemented using additive and bitwise shifting operators, which are applied to pixel groups. Thus, MMX instructions may be used. Hardware implementation is also possible.

We suggest that the 3-ranged binary tree technique be used for efficient lossless image compression with fast extraction of image approximations, as well as, for lossy compression. It successfully achieves both objectives; and considering this property along with it's computational simplicity, we estimate that our compression method may be useful for many purposes.

5. ACKNOLEDGMENTS

This work was performed by the Computer Graphics Group (Dept. of Math, Moscow State University) as a part of a research program performed in accordance with the Research Agreement between Department of Mathematics and Mechanics of Moscow State University and Intel Technologies, Inc. We would like to thank Mr. Jim Hurley and Mr. Alexander Reshetov (Intel Technologies, Inc.) for their careful reading and constructive criticism.

6. REFERENCES

[1] Amir Said and William A. Pearlman. *An Image Multiresolution Representation for Lossless and Lossy Compression.*

[2] Amir Said and William A. Pearlman. *A new Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees, 1996.*

[3] Marcelo J. Weinberger, Gadiel Serroussi, and Guillermo Sapiro. *LOCO-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm, 1996.*

[4]Manfred Kopp. *Lossless Wavelet Based Image Compression with Adaptive 2D Decomposition.*

Author(s):

Denis V. Ivanov, Dept. of Math, MSU.

Dr. Eugene P. Kuzmin, Dept. of Math, MSU.

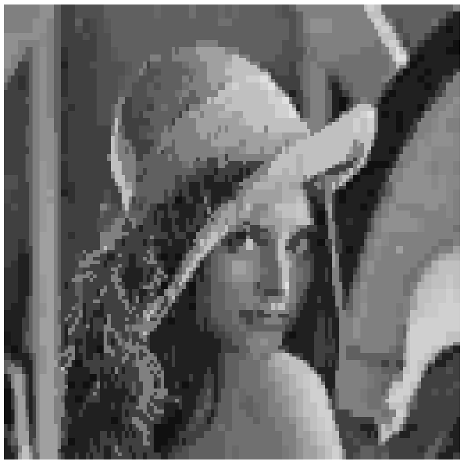
Dr. Sergey V. Burtsev, Dept. of Math, MSU.

Address: Mathematics and Mechanics Dept., Moscow State University, Vorobyovy Gory, Moscow, Russia, 119899

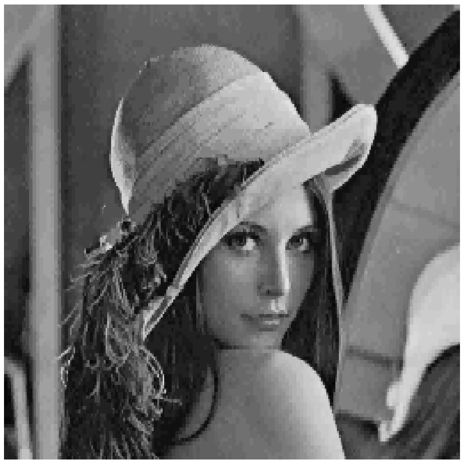
E-mail: csl@online.ru

Appendix A

Original Image of Lena (8bpp)



0.24 bpp (3%, 27.10 dB)



1.36 bpp (17%, 36.03 dB)