Texture compression with adaptive block partitions

Leonid Levkovich-Maslyuk
The Keldysh Inst. of Appl. Math.,
4, Miusskaya Sq.,
Moscow, Russia,125047
(7-095)-250-78-56
levkovl@spp.keldysh.ru

Pavel Kalyuzhny
Lomonosov Moscow State University
Russia, Moscow, GSP119899, MSU,
Computer Science Faculty
(7-095)-939-01-90
kaluzhny@graphics.cs.msu.su

Alexander Zhirkov
Lomonosov Moscow State University
Russia, Moscow, GSP119899, MSU,
Computer Science Faculty
(7-095)-939-01-90
zh@graphics.cs.msu.su

ABSTRACT
We present image compression method based on block palletizing. Image block is partitioned into four subsets, and each subset is palletized by 2 or 4 colors from the quasioptimal local palette, constructed for the whole block. Index map for the whole block, being the union of index maps for subsets, is thus only 1 or 2 bits deep, while the local palette may consist of 8 or even 16 colors. The local palette has a specific geometrical configuration in RGB color space, determined by only 2 colors. These two colors are stored explicitly, and the rest are reconstructed at the decompression stage. Compressed block consists, essentially, of the index map, palette description and partition description. This format allows fast access to randomly chosen pixels, and high reconstruction quality for compression ratios from 8 to 12, which is useful for texture storage in 3D graphics applications where real-time *decompression* is crucial.

Keywords
Texture compression, image compression, fast rendering, local palletizing

1. OUTLINE OF THE METHOD
To avoid complex geometrical computations, it is common to use textures in 3D real-time graphics. In this context, textures are appropriate images (e.g., photos of a building façade) that are 'glued' upon a 3D 'wire frames'. Texture compression methods oriented at this class of applications must allow fast access to arbitrary chosen texels (pixels of the texture image). Mostly for this reason, virtually all texture compression techniques are block-based. For example, the well known S3TC algorithm [1] uses 4-by-4 image blocks, and the compressed block consists of a local palette (four colors) and the index map. The index map is a 4-by-4 array of 2-bit words, each word being index of the palette color replacing the original color of the corresponding pixel. Substantial idea behind S3TC is to represent the local palette as a line segment in RGB color space, optimally approximating total of 16 colors in the block. Only the endpoints of this segment are stored, while the other 2 colors are reconstructed at decompression stage. This allows to achieve compression ratio 6 for true color images with 24 bit color depth.

We suggest a different compression method, based on two main ideas. The first idea is to use more complex palette geometries than line segment, in order to represent more colors, closer adapting the palette to the original color distribution in the block. However, as the number of colors increase, the index map is becoming too large because indices must contain more bits. This makes compression ratio impractically low.

The second idea allows to cope with this problem. It is independent palletizing of block subsets by small (2 or 4 colors) "subpalettes" of the palette for the whole block. For example, 8-color palette would normally require 3 bit indices. But if a subpalette of 2 colors is used for each subset, index map is only 1 bit deep, i.e. becomes 3 times smaller.

However, in this case one has to store additional information: subpalettes description and partition description. Subpalettes can be specified by fixing an order of the palette entries, so that the first two entries are the first subpalette, e.t.c. For example, in case of the 8-color palette and four 2-color subpalettes we need a string of eight 3-bit words. Another way is to specify a small number of fixed subsets of the palette as possible subpalettes. Both approaches were implemented in our algorithms.

Since it is impossible to compactly represent an *arbitrary* block partition, we used a fixed collection (dictionary) of partitions. In this case, partition is specified by its index in the dictionary. We worked with 8-by-8 blocks, and partitioned them into 4 subsets. It turned out that even this very restricted collection of 256 'typical' partitions allowed to obtain good reconstruction quality. In this case partition is specified by 8-bit word.

Compression of each block is performed in 2 steps. At the first step, a quasioptimal palette for the whole block is constructed. At the second step, an optimal partition together with an appropriate collection of subpalettes is determined so that the total block distortion is minimal. Thus, the compressed block consists of: index of the selected partition; palette description; subpalettes description; index map.

Decompression of a texel is performed by: determination of the containing block (trivial calculation), reconstruction of the palette (essentially, a sequence of linear interpolations and equally simple operations), finding the partition subset containing the texel (simple masking operation), and output of the palette entry according to the index map.

Now let us consider some important details of the algorithms based on this approach.

2. DETAILS OF THE ALGORITHMS

2.1 Palettes geometry

We implemented two algorithms, (denote them A12 and A8), with compression ratios 12 and 8, respectively. A12 uses 8-color palette for the block, and 2-color subpalette for each of the four block subsets. A8 uses 16-color palette for the block, and 4-color subpalette for each of its four subsets. Both algorithms use the same dictionary of 256 block partitions.

Figure 1 shows the palette geometries for A8 and A12.

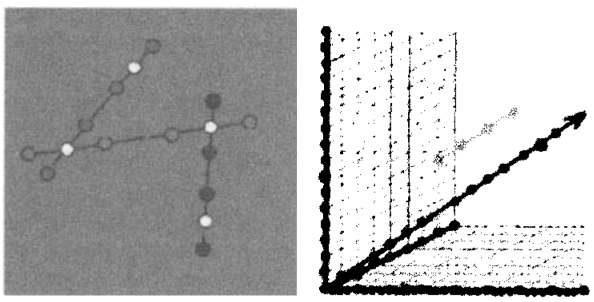

Figure 1. A8 (left) and A12 (right) palette

For A8 palette, white circles are the base colors that are stored. They are vertices of a tetrahedron in RGB color space. Sets of points shown by green, red, blue and white circles, each form a 4-color subpalette. Each subset of the chosen partition is palletized by one of these subpalettes. The A12 palette consists of two segments (four points on each) in the color space RGB. One of the two segments is shown in light-brown in Fig.1. Starting point of each segment is stored in RGB(4,5,3) format. The endpoint has the same (U,V) components (in the standard YUV color space), but its Y-component is different, and is defined by a 4-bit number (Y axis is shown, conventionally, in black in Fig.1). Hence, the palette is determined by $2\cdot(4+5+3+4)=32$ bits. Subpalettes can be arbitrary 2-color subsets of the palette.

2.2 Examples of partitions

Block partitions are obtained by discretization of 'typical' quadratic functions level sets. Figure 2 shows how an original 8-by-8 pixel block (a) is compressed by A8 (b) and by A12 (c). These pictures are examples of the partitions from the dictionary.

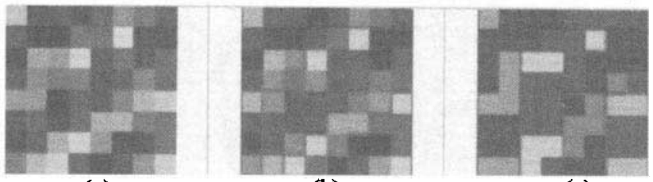

Figure 2. (a) Original image block. (b) A8-compressed block. (c) A12-compressed block.

Subsets boundaries are shown in green. Fidelity of reconstruction decreases with the increase of compression ratio, but the informative features of the block remain visible.

2.3 Compression and decompression processes

A12 compression and decompression flowcharts for a single block

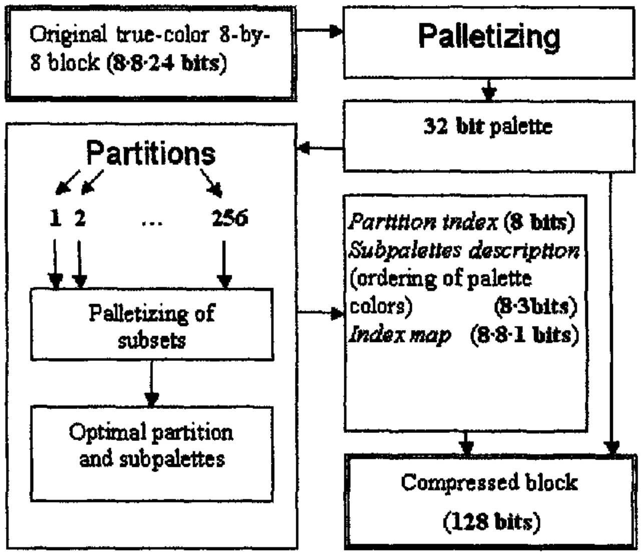
Original true-color 8-by-8 block (8·8·24 bits)
Palletizing
32 bit palette
Partitions
1 2 ... 256
Palletizing of subsets
Optimal partition and subpalettes
Partition index (8 bits)
Subpalettes description (ordering of palette colors) (8·3bits)
Index map (8·8·1 bits)
Compressed block (128 bits)

Figure 3. A12 Compression flowchart.

are shown in Figures 3 and 4, respectively.

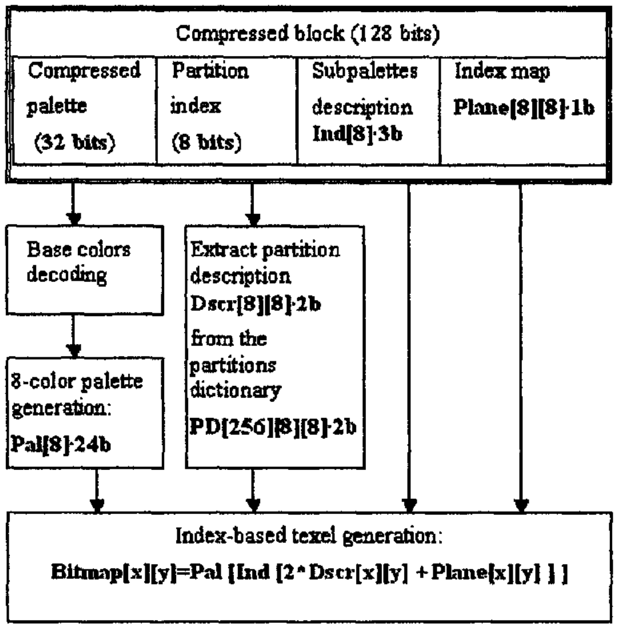
Compressed block (128 bits)
Compressed palette (32 bits)
Partition index (8 bits)
Subpalettes description Ind[8]·3b
Index map Plane[8][8]·1b
Base colors decoding
8-color palette generation: Pal[8]·24b
Extract partition description Dscr[8][8]·2b from the partitions dictionary PD[256][8][8]·2b
Index-based texel generation:
Bitmap[x][y]=Pal [Ind [2*Dscr[x][y] +Plane[x][y]]]

Figure 4. A12 Decompression flowchart.

3. TEST RESULTS

We have performed extensive testing of the algorithms A12 and A8 on a collection of images of various nature, with the aid of various distortion measures. In all cases, A8 algorithm (8 times compression) outperformed S3TC (6 times compression). A12 (12 times compression) has shown somewhat lower reconstruction quality than S3TC. Visual quality can be subjectively estimated as 'good' for A12 and 'excellent' for A8. PSNR values presented in the Table 1 give a quality estimate for two test images in terms of PSNR. "Portrait" image (Figure 5), turned out to be the most difficult among the 'natural' images for all the three algorithms A8, A12 and S3TC. "Lena" image was included here because of its popularity.

It should be noted that for *any* compression method based on reduced palette for small blocks, it is easy to construct artificial images for which visual distortion will be very noticeable. Indeed, any pattern composed of blocks with colors uniformly distributed in RGB color space will suffer drastic distortion after palletizing. Fortunately, this is practically never the case for images one meets in applications.

PSNR computations for RGB color images were performed for the distortion of standard intensity Y=0.3R+0.59G +0.11B.

Table 1. PSNR values for the three algorithms

	A12	S3TC	A8
"Portrait"	28.4	32	33.6
"Lena"	35	38.95	40.9

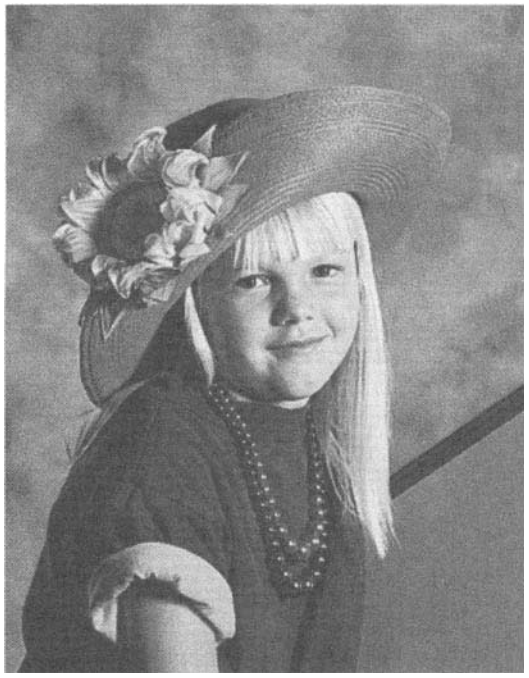

Figure 5. Test image "Portrait"

4. CONCLUSION

A method for image compression based on palletizing of small block subsets by subpalettes of compactly stored local palette was developed. Two algorithms using this method have compression ratios 12 and 8 for true color images, and allow fast random access to distinct pixels. High quality of reconstructed images suggests that the algorithms can be useful for texture storage in 3D graphics applications where fast decompression is important. Ideas of the work can be further developed by optimizing the dictionary of block partitions (with the aid of, for example, coding theory techniques), and by improving palette geometry in color space.

5. AKNOWLEDGMENTS

We are grateful to Yuri Bayakovski (Lomonosov Moscow State University) for helpful discussions, and to Jim Hurley and Alexander Reshetov (Intel Corporation) for their attention to the work. The work was supported by the Intel Corporation Research Contract.

6. REFERENCES

[1] S3TC DirectX 6.0 Standard Texture Compression, Savage 3D white papers (1998).