PRACTICAL IMPLEMENTATION OF XOR – LINKED LISTS IN IMAGE STORAGE

Autori: Igor MARTA, Mihail KULEV
Conducătorştiinţific: dr., conf. univ.MihailKULEV

Universitatea Tehnică a Moldovei
Email:igor_marta@rambler.ru, mkmk@mail.md

Abstract:*Have been considered quadtrees in C/C++ languages and their application for image storing and a way of segmentation of the pixel array, converting quadtree to XOR quadtree, generation of line codes and inverse traversals of quadtree.*

Keywords: *Quadtree, XOR – linked lists, linecodes, rgb color pallet, bitmap image format, pixel array, spatial data structure.*

1. Introduction

The aim was to involve this method of linking nodes into a project which makes use of linked lists or trees that require bidirectional traversals. The choice was computer graphics. There are several methods of storing an image in computer memory. One of those is the polygon decomposition (here: square decomposition), which involves usage of quad trees and linked lists. Also in this method are required bidirectional traversals of the tree which is the place where XOR linking can fit pretty well in order to safe space and to avoid using of an additional memory field in the quad tree structure. Also it was practically useful to see graphically what image the quad trees contain and it was involved in the project some references of image processing. This was made in order to be able to save the pixel matrix in quad tree form and in image form. Also it was interesting to see if after decomposition and saving the image in location code form, the size of memory occupied by the image reduces or increases.

2. Practical implementation of XOR – linked lists in image storage

XOR linked lists are a data structure used in computer programming. They take advantage of the bitwise exclusive disjunction (XOR) operation, here denoted by ⊕, to decrease storage requirements for doubly-linked lists. An ordinary doubly-linked list stores addresses of the previous and next list items in each list node, requiring two address fields:

```
 ...  A       B         C         D         E  ...
         ->  next ->  next  ->  next  ->
           <-  prev<-  prev<-  prev<-
```

An XOR linked list compresses the same information into *one* address field by storing the bitwise XOR of the address for *previous* and the address for *next* in one field:

```
 ...  A         B         C         D         E  ...
        <->   A⊕C   <->   B⊕D   <->   C⊕E   <->
```

When you traverse the list from left to right: supposing you are at C, you can take the address of the previous item, B, and XOR it with the value in the link field (B⊕D). You will then have the address for D and you can continue traversing the list. The same pattern applies in the other direction.

To start traversing the list in either direction from some point, you need the address of two consecutive items, not just one. If the addresses of the two consecutive items are reversed, you will end up traversing the list in the opposite direction.

Quad tree

A **quadtree** is a tree data structure in which each internal node has exactly four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions. The regions may be square or rectangular, or may have arbitrary shapes. This data structure was named a quadtree by Raphael Finkel and J.L. Bentleyin 1974. A similar partitioning is also known as a *Q-tree*. All forms of Quadtrees share some common features:

- They decompose space into adaptable cells
- Each cell (or bucket) has a maximum capacity. When maximum capacity is reached, the bucket splits
- The tree directory follows the spatial decomposition of the Quadtree

Quadtrees may be classified according to the type of data they represent, including areas, points, lines and curves. Quadtrees may also be classified by whether the shape of the tree is independent of the order data is processed.

A node of a point quadtree is similar to a node of a binary tree, with the major difference being that it has four pointers (one for each quadrant) instead of two ("left" and "right") as in an ordinary binary tree. Also a key is usually decomposed into two parts, referring to x and y coordinates. Therefore a node contains following information:

- 4 Pointers: quad[‘NW’], quad[‘NE’], quad[‘SW’], and quad[‘SE’]
- point; which in turn contains:
    - key; usually expressed as x, y coordinates value; for example a name

The key is the first operation, and the properties of XOR:

- $X \oplus X = 0$
- $X \oplus 0 = X$
- $X \oplus Y = Y \oplus X$
- $(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$

The R2 register always contains the XOR of the address of current item C with the address of the predecessor item P: $C \oplus P$. The Link fields in the records contain the XOR of the left and right successor addresses, say $L \oplus R$. XOR of R2 ($C \oplus P$) with the current link field ($L \oplus R$) yields $C \oplus P \oplus L \oplus R$.

- If the predecessor was L, the P(=L) and L *cancel out* leaving $C \oplus R$.
- If the predecessor had been R, the P(=R) and R cancel, leaving $C \oplus L$.

In each case, the result is the XOR of the current address with the next address. XOR of this with the current address in R1 leaves the next address. R2 is left with the requisite XOR pair of the (now) current address and the predecessor.

Image decomposition

A natural gray-level image usually can be divided into different size regions with a variable amount of details and information. Such segmentation of the image is useful for efficient coding of image data. QT decomposition is a powerful technique which divides the image into 2-D *homogeneous* (in the property of interest) regions, i.e., produces the segmentation. The decomposition builds a tree. Each tree node has four children and it is associated with a uniquely defined region of the image. It is obvious that the root is associated with the whole image. QT decomposition can be done either by top-down or bottom-up procedures. In Fig. 1, both top-down and bottom-up QT decomposition procedures are illustrated. It is well known, and also demonstrated by this simple example, that the bottom-up procedure is superior; therefore it is preferred for usage in the suggested algorithm. When QT decomposition is used for image compression, the resulting tree is coded. The coding procedure includes coding of the tree structure information and coding of the leaf information. Let assign “1” to the parent node and ***“0”*** to the leaf. To each leaf a parameter (or parameters) that describe the intensity of the corresponding subimage will also be assigned. Obviously, the image pixels are always leaves, so the tree structure coding can be stopped one level before the bottom level.

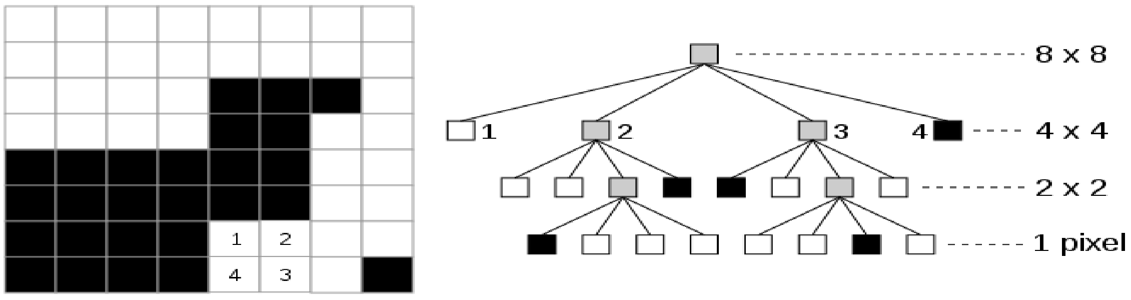

Fig. 1

An image consists of a two-dimensional array of numbers. The color or gray shade displayed for a given picture element (pixel) depends on the number stored in the array for that pixel. The simplest type of image data is black and white. It is a binary image since each pixel is either 0 or 1. The next, more complex type of image data is gray scale, where each pixel takes on a value between zero and the number of gray scales or gray levels that the scanner can record. These images appear like common black-andwhite photographs - they are black, white, and shades of gray. Most gray scale images today have 256 shades of gray. People can distinguish about 40 shades of gray, so a 256-shade image "looks like a photograph". The most complex type of image is color. Color images are similar to gray scale except that there are three bands, or channels, corresponding to the colors red, green, and blue. Thus, each pixel has three values associated with it. A color scanner uses red, green, and blue filters to produce those values. Images are available via the Internet, scanners, and digital cameras. Any picture shown on the Internet can be downloaded by pressing the right mouse button when the pointer is on the image. This brings the image to your PC usually in a JPEG format. Your Internet access software and other software packages can convert that to a TIFF or BMP.

The **BMP file format**, sometimes called **bitmap** or **DIB file format** (for *device-independent bitmap*), is an image file format used to store bitmap digital images, especially on Microsoft Windowsand OS/2 operating systems.

Many older graphical user interfaces used bitmaps in their built-in graphics subsystems; for example, the Microsoft Windows and OS/2 platforms' GDI subsystem, where the specific format used is the *Windows and OS/2 bitmap file format*, usually named with the file extension of .BMP or .DIB.

In uncompressed BMP files, and many other bitmap file formats, image pixels are stored with a color depth of 1, 4, 8, 16, 24, or 32 bits per pixel (BPP). Images of 8 bits and fewer can be either grayscale or indexed color. An alpha channel(for transparency) may be stored in a separate file, where it is similar to a grayscale image, or in a fourth channel that converts 24-bit images to 32 bits per pixel.

BMP FileHeader	Stores general information about the BMP file.
BitmapInformation (DIB header)	Stores detailed information about the bitmap image.
ColorPalette	Stores the definition of the colors being used for indexed color bitmaps.
BitmapData	Stores the actual image, pixel by pixel.

3. Conclusion

So it shows that this method is suitable only when our image has dimensions $2^n \times 2^n$. Also it will be profitable in memory handling when our image has many regions of homogeneous pixel values, it will save only one value in a leaf node instead of saving all those values in a matrix. Main opinion is that bitwise operations are very practical and straight forward; they offer high manipulation of programming units and a very big flexibility of the algorithm. In our case we managed to save one memory field by not using direct linking between nodes, but the XOR linking method.

References

1. Samet H. *Applications of spatial data structures to computer graphics* (AW, 1990);
2. Brian W. Kernighan Dennis M. Ritchie *C programming language*(1972);
3. Dwayne Phillips second edition of *Image Processing in C* (Copyright 1994,).