J Sign Process Syst (2017) 88:453–462
DOI 10.1007/s11265-016-1182-x

CrossMark

Embedded Hypercube Graph Applied to Image Analysis Problems

Eduardo Sant'Ana da Silva[1] · Helio Pedrini[1]

Received: 4 October 2015 / Revised: 5 May 2016 / Accepted: 12 September 2016 / Published online: 7 October 2016
© Springer Science+Business Media New York 2016

Abstract Hypercubes have interesting geometric and topological properties with applications in several different fields, such as computer networks, information retrieval, data fusion, social networks, coding theory and linguistics. In this work, we present and discuss the use of hypercubes in some image analysis problems. Hypercube graphs take advantage of high dimensional features to provide low-cost image transformations. The downsampling of an image is performed as a pixel permutation, with no need for interpolation and, consequently, addition and multiplication operations. The hypercube graph is employed on demand one edge at once, such that there is no memory usage to traverse the image. Experimental results demonstrate the effectiveness of hypercubes as a powerful space representation both in terms of computational time and memory requirements.

Keywords Image analysis · Hypercube graph · Gray code · Edge detection · Multiscale decomposition · Logic units

1 Introduction

In this work, we introduce a methodology for solving a set of image processing problems based only on simple logical operations. Such restriction allows its implementation on a variety of devices not only based on a central processing unit (CPU) computation, as well as in graphics processing unit (GPU), field-programmable gate array (FPGA) or even simple application-specific integrated circuits (ASIC).

Before presenting the proposed methodology, it is needed to define our computational model to perform complexity analysis that should be far in any chosen architecture to implement it. In our computational model, we considered the most basic operation as the access of a pixel from an image data source, that could be a sensor, a file, a service or any mean that provides that information as $O(1)$.

A hypercube Q_n is a graph with 2^n vertices, where its vertices are binary n-vectors and two vertices are adjacent in the hypercube if the Hamming distance between them is equal to 1. We start embedding the image data source on a hypercube by mapping the image 2D coordinates to a Binary Gray Code axis. This way, all pixels that are neighbors in the image data source are neighbors on the hypercube as well (the opposite is not valid) and each node belonging to the hypercube represents a pixel in the image. Using this approach, one can benefit from the multidimensional properties of the hypercube as the projection in different vector spaces. Vector space projections usually involve calculations that require complex arithmetic units that are beyond the scope of the presented framework. However, one can perform such operations on the hypercube using only simple logic units.

In order to take advantages of the hypercube topology, it is necessary the employment of an efficient way to construct it entirely or partially. Therefore, we use the algorithm proposed by Silva et al. [42] to construct independent spanning trees [50] over the hypercube. As we are not working with spanning trees directly, we can disregard the memory consumptions since we process each edge (composed of a pair of pixels in image) once. Actually, Silva et al. [42] algorithm builds a directed hypercube and its edge generation has no

✉ Helio Pedrini
helio@ic.unicamp.br

[1] Institute of Computing, University of Campinas, Campinas, São Paulo 13083-852, Brazil

repeated edge even without marking the visited edges in the hypercube case (in the case of the spanning trees, it requires a bit per vertex). Thus, the complexity for the construction of the proposed technique is constant in terms of memory, that is, $O(1)$.

As the technique complexity regarding required memory requirements and computational resources remain low, it is still necessary a simple operation that performs projections onto other vector spaces. For this purpose, it is only used a transformation operation that maps each coordinate of the hypercube to a vector space $D-1$ or $D+1$. This operation is the one-bit shift right of each hypercube coordinate in the case of downsampling an image $w \times h$ to $w/2 \times h/2$ and logical shift to the left to increase the resolution, where w and h are the width and height of the image, respectively. The increase in resolution requires some work regarding interpolation. The number used in the shift right logical operation generates 2^n image samples and, although the images are similar, they are merely redistributions of the original image pixels.

With the proposed technique, it is possible to generate low resolution images samples with the complexity of the sample size, i.e. $w \times h$ sample pixels no matter the original resolution of the image is. Regarding other multiscale methods, only considering them as one access per pixel, the complexity is $O(n)$. Thus, if we have an image of, for instance, 512 × 512 pixels and we need a 64 × 64 pixel sample image, then 262,144 pixel accesses will be required against 4,096 used in the proposed method. As it has no interpolation or change on the information of the pixels besides its repositioning on the image, some characteristics as histogram are quite preserved. Our results show that the proposed method preserves the geometric features of the image better than other multiscale methods.

The proposed method can be applied to a variety of problems. It can provide image search gains by using the low resolution samples of the image before going through the entire image at original resolution. It allows reduction on the processing time in real-time applications, such as vision-aided vehicle driving systems or robot path-planning systems, which require to process thousands of images per second.

As main contributions of this work, we provide a simple way to perform preliminary image analysis in real-time systems on very limited computing power devices, and provide new perspectives on how to traverse images based on the desired goal. We provide satisfactory results with simple operations which can contribute to an initial sorting of images before applying more expensive traditional methods from a computational point of view. Our operations are based upon simple logic units and simple pixel access using some coordinate transformation functions providing fast and effective results. To the best of our knowledge, there is no previous work that applies transformations to images by simply translating and rotating hypercube coordinates.

The remainder of the paper is organized as follows. Section 2 briefly describes some concepts related to hypercube graphs and their applications. Section 3 presents the methodology proposed in this work. Section 4 describes and discusses the experiments and results. Section 5 concludes the paper with final remarks and directions for future work.

2 Background

Before presenting the methodology, it is necessary to define some terms that are used throughout this work, as well as some conventions proposed here.

A graph G is an ordered pair of disjoint sets (V, E) such that E is a subset of $V^{(2)}$ of unordered pairs of V. Let G be a graph, then $V = V(G)$ is the vertex set of G and $E = E(G)$ is the set of edges. An edge x, y joins the vertices x and y and is denoted by xy. Then, xy and yx refer to the same edge. If $xy \in E(G)$, then x and y are adjacent vertices of G, and the vertices x and y are incident to the edge xy. Two edges are said adjacent if they have exactly one common endvertex [4]. A path is a sequence of vertices $v_0, v_1, v_2, ..., v_l$ describing the route of the vertex i towards the vertex j. A cycle in a graph is a subset of edges that represents a path such as that the first vertex of the path corresponds to the last vertex.

A graph G has a Hamiltonian path if, and only if, there is a path in G such that each vertex is visited once. A Hamiltonian cycle is a cycle that forms a Hamiltonian path.

The hypercube, also called n-cube and usually denoted as Q_k, is a k-connected graph with a special property where each vertex and its adjacent differ in only one bit in their binary representations, as illustrated in Fig. 1.

An m-bit Gray code, G_m, denotes a sequence of all binary numbers of m-bits. G_1 is defined as (0, 1) and for $m > 1$, G_m is defined recursively in terms of G_{m-1} as $(0G_{m-1}, 1G_{m-1}{}^r)$, where $G_{m-1}{}^r$ is the reverse order G_{m-1} and $0G_{m-1}$ $(1G_{m-1}{}^r)$ is the advance fixing of each binary number G_{m-1} $(G_{m-1}{}^r)$ as 0 (1) [21].

For instance, a binary number sequence (000, 001, 011, 010, 110, 111, 101, 100) is a Gray code of 3 bits representing a path in a hypercube of 2^3 vertices (3-cube), starting with vertex 000 and finishing at vertex 100.

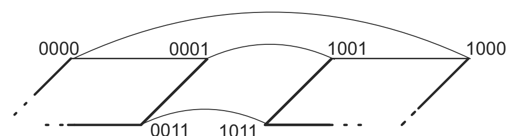


Figure 1 Two vertices are adjacent in the hypercube if their symbols differ exactly in a single coordinate.

Figure 2 Mapping an image of 8×4 pixels to the hypercube Q_5.

4 lines (2 bits)

decimal	gray code
--	--
0	00
1	01
2	11
3	10

8 columns (3 bits)

decimal	gray code
--	--
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)

00000	00001	00011	00010	00110	00111	00101	00100
01000	01001	01011	01010	01110	01111	01101	01100
11000	11001	11011	11010	11110	11111	11101	11100
10000	10001	10011	10010	10110	10111	10101	10100

tuple	gray code
(0,0)	00000
(0,1)	00001
(0,2)	00011
(0,3)	00010
(0,4)	00110
(0,5)	00111
(0,6)	00101
(0,7)	00100
(1,0)	01000
(1,1)	01001
(1,2)	01011
(1,3)	01010
(1,4)	01110
(1,5)	01111
(1,6)	01101
(1,7)	01100
(2,0)	11000
(2,1)	11001
(2,2)	11011
(2,3)	11010
(2,4)	11110
(2,5)	11111
(2,6)	11101
(2,7)	11100
(3,0)	10000
(3,1)	10001
(3,2)	10011
(3,3)	10010
(3,4)	10110
(3,5)	10111
(3,6)	10101
(3,7)	10100

The Hamiltonian property of the hypercube is strongly related to the theory of Gray codes [22, 43, 48]. Hypercubes are often used in networks [3, 28, 46] due to their valuable properties in fault tolerant systems, their symmetry and logarithmic distance between the vertices available in commercial implementations such as iPSC [23] and nCUBE [35]. Several algorithms have been proposed to embed important topologies on the hypercube such as rectangular meshes [6, 25], trees [13, 20] and pyramids [29, 51]. For additional information about hypercube properties, the reader may consult [18].

Hypercubes are not a novel topic in the computer field. Cover [9] used them to give the upperbounds on the number of polynomially separable Boolean functions. Pease [36] employed the hypercube topology to explore the possibility of its use as a large scale array of microprocessors as computational facility due to the high degree of parallelism.

Some hypercube algorithms have been employed to address problems in coding theory [26], linguistics [16], computer system design [19], image analysis [7, 10, 31, 33, 38, 52], pattern recognition [24, 34, 37, 40] and computational geometry [8, 30, 32, 41]. For additional information about properties and applications of hypercube graphs, the reader can refer to [2, 12, 18, 39].

Graph representation has also been utilized in several image processing problems. Eshera and Fu [15] described a semantic-synthatic model based on attributed relational graphs to understand the content of images. Wu and Leahy [49] presented a graph theoretic approach to data clustering, such that the problem is formulated as the

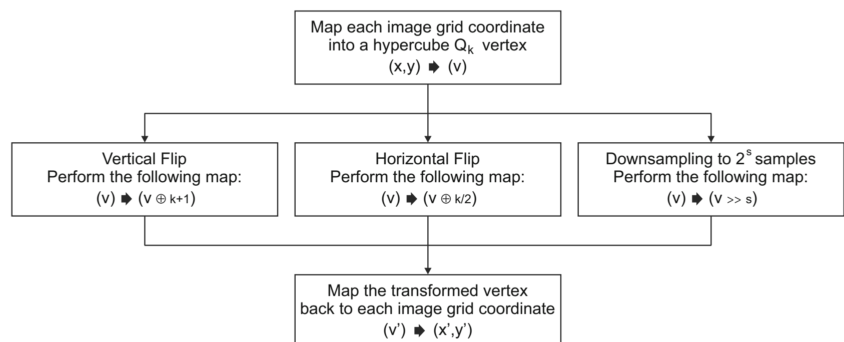


Figure 3 Diagram with the main stages of the proposed methodology.

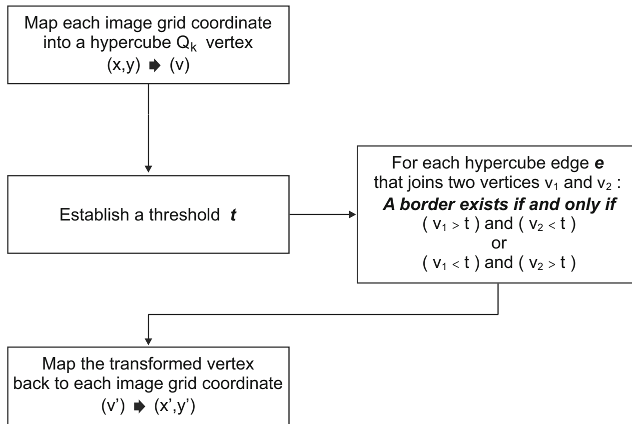


Figure 4 Diagram with the main stages of edge detection methodology.

optimal partitioning of an undirected graph into a number of subgraphs. The clustering algorithm is applied to segment images.

Trémeau and Colantoni [44] developed structures and algorithms based on region adjacency graphs to segment color images, improving other segmentation approaches such as region growing or watershed transformation. Uyttendaele et al. [45] presented methods for automatically creating image mosaics from a set of panoramic photographs through a graph of regions of difference between input images.

Methods based on graph cuts construct a specialized graph for the energy function to be minimized [27]. These methods have been applied to a variety of image processing problems, such as image restoration [11], image segmentation [47] and medical imaging [5]. Elmoataz et al. [14] presented a non-local discrete regularization approach on weighted graphs for image and manifold processing.

The majority of the research work on image processing using hypercubes developed in the 1980 and 1990 decades reports the application of algorithms with n-cube or hypercube architecture present in supercomputers such as iPSC [23] and nCUBE [35]. In contrast, the hypercube dimensions used in our method are based on the image size, $O(\log(w * h))$, such that for an image of 4096×4096 pixels, for instance, the hypercube has 22 dimensions. Our algorithm for constructing the hypercubes does not need any memory in the construction stage and the edges are used on demand and discarded right after their use.

3 Methodology

This section describes the methodology developed for the image manipulation by simply mapping the original image coordinates to the hypercube topology. Two operations are analyzed: edge detection and multiscale decomposition.

The edge detection is performed locally using only two vertices at a time that represent two neighbor pixels in the original image through a very simple approach in which a border exists if an active pixel and an inactive one are neighbors in the original image.

Figure 5 Translation of the coordinates through 1-bit logic shift right operation.

tuple	gray code	gray code >>1	tuple	tuple	gray code	gray code >>1	tuple
(4,0)	110000	11000	(2,0)	(0,0)	000000	00000	(0,0)
(4,1)	110001	11000	(2,0)	(0,1)	000001	00000	(0,0)
(4,2)	110011	11001	(2,1)	(0,2)	000011	00001	(0,1)
(4,3)	110010	11001	(2,1)	(0,3)	000010	00001	(0,1)
(4,4)	110110	11011	(2,2)	(0,4)	000110	00011	(0,2)
(4,5)	110111	11011	(2,2)	(0,5)	000111	00011	(0,2)
(4,6)	110101	11010	(2,3)	(0,6)	000101	00010	(0,3)
(4,7)	110100	11010	(2,3)	(0,7)	000100	00010	(0,3)
(5,0)	111000	11100	(2,7)	(1,0)	001000	00100	(0,7)
(5,1)	111001	11100	(2,7)	(1,1)	001001	00100	(0,7)
(5,2)	111011	11101	(2,6)	(1,2)	001011	00101	(0,6)
(5,3)	111010	11101	(2,6)	(1,3)	001010	00101	(0,6)
(5,4)	111110	11111	(2,5)	(1,4)	001110	00111	(0,5)
(5,5)	111111	11111	(2,5)	(1,5)	001111	00111	(0,5)
(5,6)	111101	11110	(2,4)	(1,6)	001101	00110	(0,4)
(5,7)	111100	11110	(2,4)	(1,7)	001100	00110	(0,4)
(6,0)	101000	10100	(3,7)	(2,0)	011000	01100	(1,7)
(6,1)	101001	10100	(3,7)	(2,1)	011001	01100	(1,7)
(6,2)	101011	10101	(3,6)	(2,2)	011011	01101	(1,6)
(6,3)	101010	10101	(3,6)	(2,3)	011010	01101	(1,6)
(6,4)	101110	10111	(3,5)	(2,4)	011110	01111	(1,5)
(6,5)	101111	10111	(3,5)	(2,5)	011111	01111	(1,5)
(6,6)	101101	10110	(3,4)	(2,6)	011101	01110	(1,4)
(6,7)	101100	10110	(3,4)	(2,7)	011100	01110	(1,4)
(7,0)	100000	10000	(3,0)	(3,0)	010000	01000	(1,0)
(7,1)	100001	10000	(3,0)	(3,1)	010001	01000	(1,0)
(7,2)	100011	10001	(3,1)	(3,2)	010011	01001	(1,1)
(7,3)	100010	10001	(3,1)	(3,3)	010010	01001	(1,1)
(7,4)	100110	10011	(3,2)	(3,4)	010110	01011	(1,2)
(7,5)	100111	10011	(3,2)	(3,5)	010111	01011	(1,2)
(7,6)	100101	10010	(3,3)	(3,6)	010101	01010	(1,3)
(7,7)	100100	10010	(3,3)	(3,7)	010100	01010	(1,3)

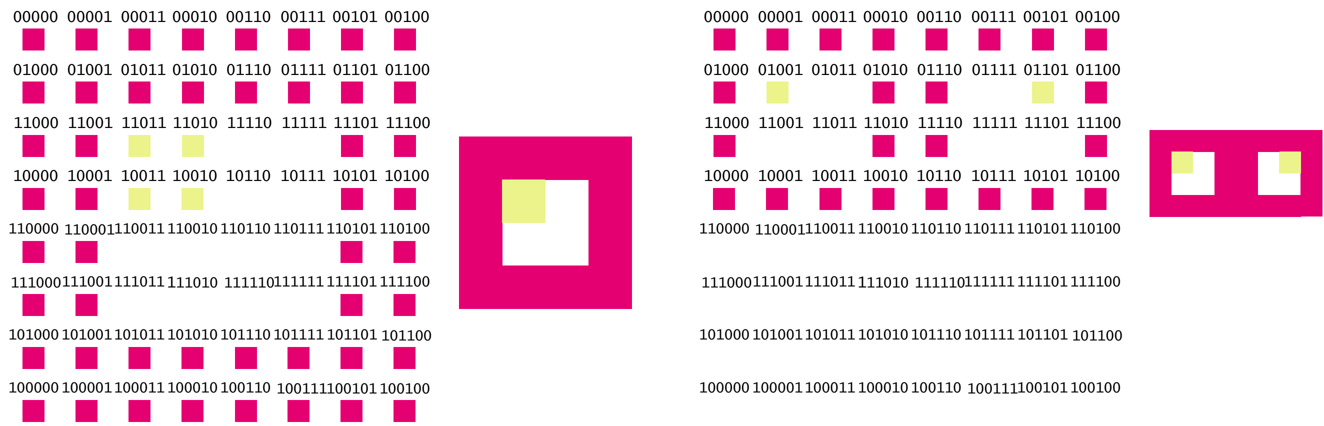


Figure 6 Original image and the result of the translation of the coordinates using 1-bit logic shift right operation.

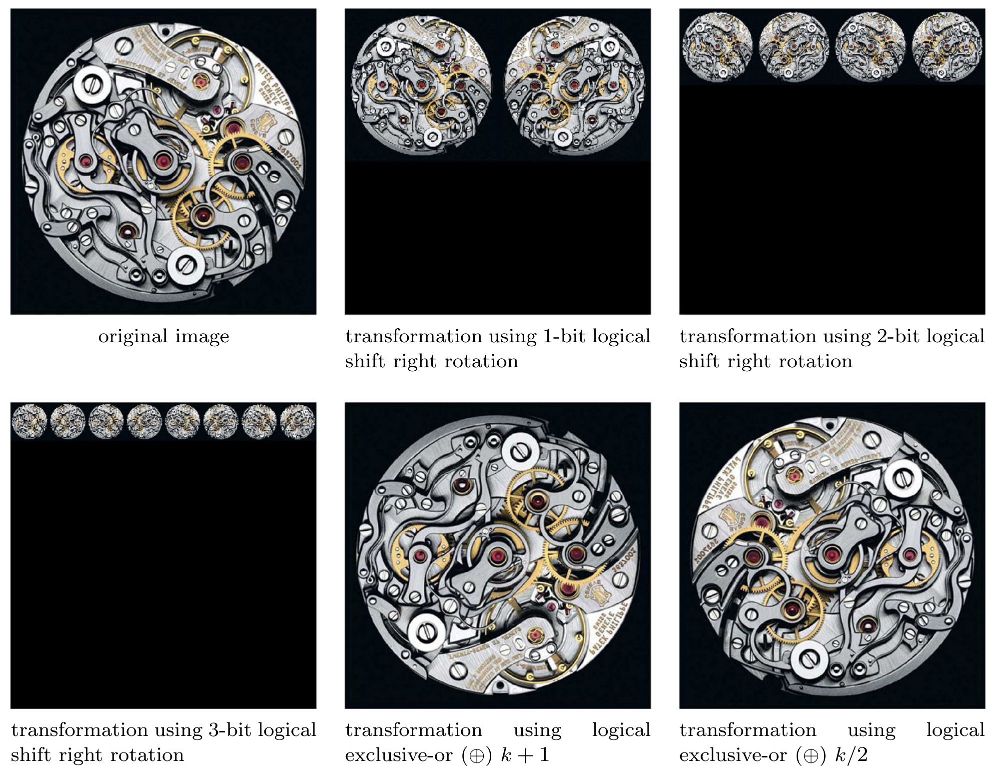


Figure 7 Results for multi-resolution and horizontal/vertical flips on an image.

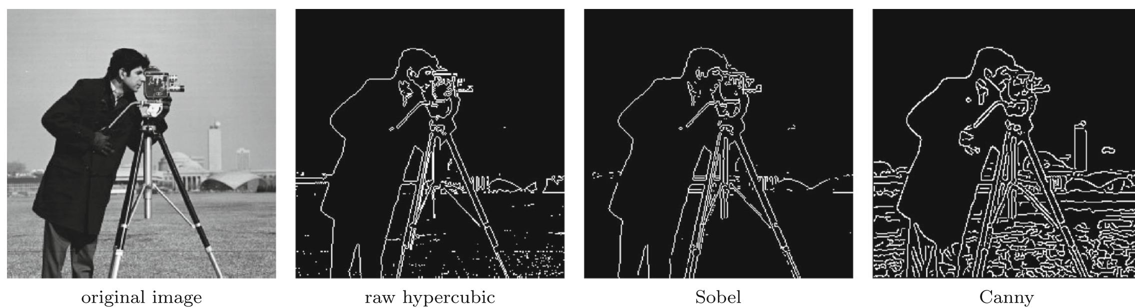


Figure 8 Results for different edge detection methods.

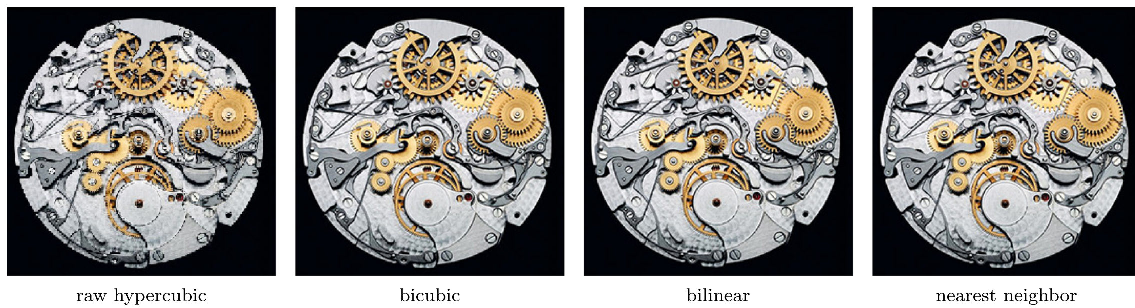


Figure 9 Results for different image resampling methods.

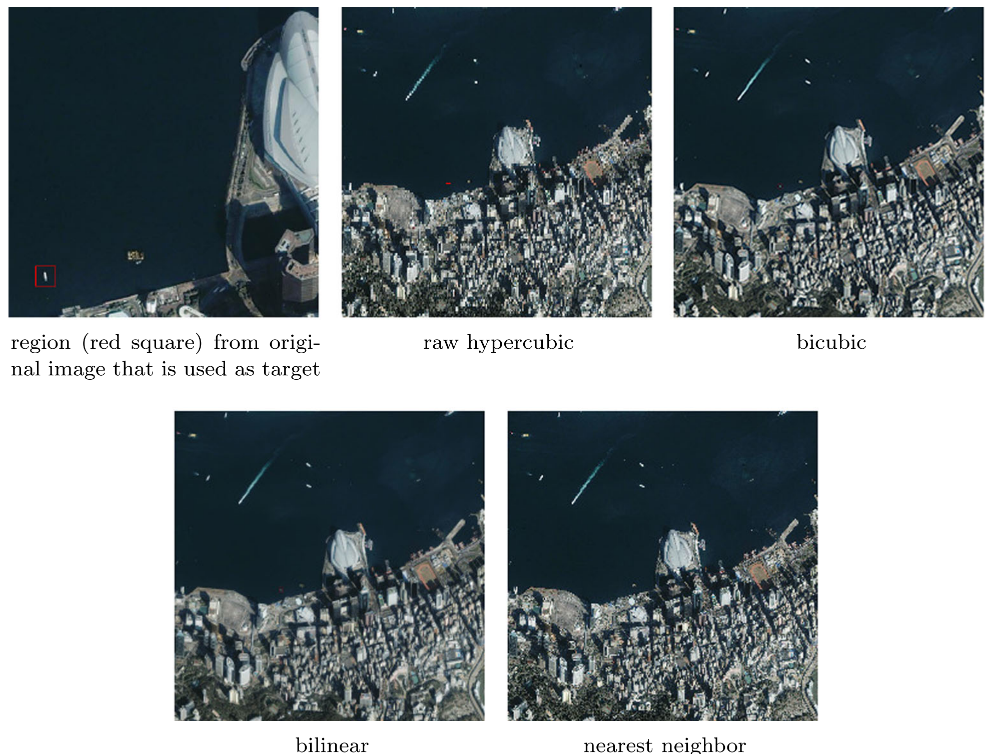
Differently, the multiscale decomposition translates each pixel of the image to a new coordinate, such that the decomposition can be performed by just accessing the pixels of the original image without any other computation involving the generation of new values. Furthermore, all the pixels present in a multiscale decomposition are present in the original image with values of identical band intensities. Such approach provides a more accurate analysis of the original image through the processing of its multiscale decomposition samples. For instance, one can perform a search for a specific color in an image via its decomposed samples, which reduces the required processing time.

The following describes the steps needed by the methodology proposed in this work. Initially, an image is mapped to the hypercube topology, such that the mapping is performed as follows: each hypercube coordinate is obtained by concatenating the bits representing the X-Y image coordinates using their equivalent Gray code, resulting in a binary representation of the hypercube vertex. Figure 2 shows the mapping of an image with 8×4 pixels to the hypercube Q_5. After that, one can handle the image operations based on the hypercube topology, almost disregarding the image grid representation.

Figures 3 and 4 show schematic diagrams representing the proposed methodology. Our approach provides a proper level of abstraction since it simplifies the traditional way to deal with usual image processing problems. For instance, the edge detection is handled by using only a hypercube edge, i.e., a threshold is established and an edge is present if one of the vertices is below and the other is above the threshold established.

Algorithm 1 is used to navigate in the image as a hypercube. Initially, the vertex set is traversed (Line 2), the neighboring dimensions are explored (Line 3), edge

Figure 10 Results for different image resampling methods.



repetitions are avoided (Line 4) and, finally, the edges are generated (Line 5).

Algorithm 1 Algorithm for construction of the hypercube edge set.

```
  input : k (2^k is the hypercube order)
  output: hypercube edge set
1 v: number of vertices (2^k) bitwise operators OR: ∨,
  and XOR: ⊕
2 for n ← 0 to v do
3 |   for b ← 1 ; i ← 1 to k do
4 |   |   if n ≠ n ∨ b then
5 |   |   |   edge ←Edge(n,n ⊕ b);
6 |   |   end
7 |   end
8 end
```

We provide some factors of multiscale decomposition by the simple translation of the original coordinate space to another one. For instance, to generate an image with half of the original scale, we transform the hypercube by shifting all vertices by 1 bit to the right-hand side, causing the translation of the pixels to a new vectorial space, as illustrated in Fig. 5.

The number m used in the shift transformation operation results in 2^m multiscale decompositions of the image. Figure 6 illustrates a decomposition resulting in two sample images, each one with the half of the original resolution.

4 Experimental Results

This section presents the experimental results obtained by applying the proposed methodology to a number of images. The experiments were performed on an AMD FX-6300 3.5 GHz processor and 8 GB of memory. The algorithms were implemented in Java programming language.

Figure 7 shows the results for different logical operations on an input image. It is possible to observe the multiple resolutions and horizontal/vertical flips obtained only by coordinate translations with simple operations.

Results for edge detection are shown in Fig. 8. A comparison among the method based on hypercube graphs and two traditional image edge detection methods (Sobel and Canny) [17], is performed. A threshold value of 80 was used in the Sobel method. A standard deviation of 0.45 and an upper and lower threshold of 50 and 110, respectively, were used in the Canny edge detector. Our method used a threshold value of 90. The proposed method was able to capture fine details present in the image.

Figure 9 shows the results for different resampling methods, including the nearest neighbor, bilinear and bicubic interpolation techniques. The proposed method is referred here to as raw hypercube due to the fact that we purely apply the transposition of pixels without any interpolation or any other modification. This means that the signal information of the original pixel is retained as one can see by the lack

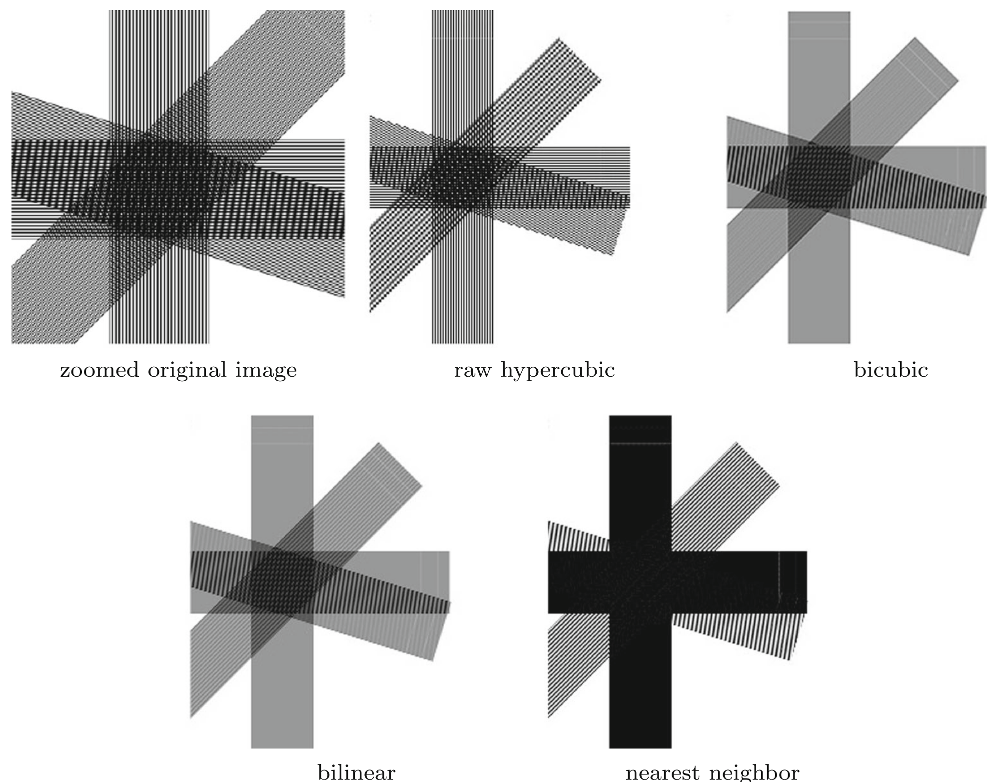
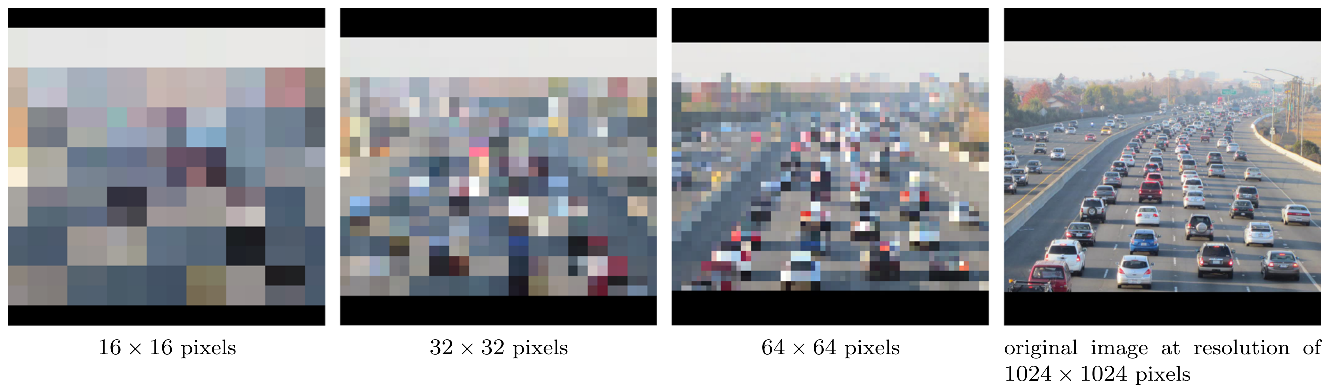


Figure 11 Results for different image resampling methods.



16 × 16 pixels | 32 × 32 pixels | 64 × 64 pixels | original image at resolution of 1024 × 1024 pixels

Figure 12 Example of multi-resolution search using the hypercube for different image resolutions.

of smoothing on the resulting image. Furthermore, the final image quality obtained by such a simple method is relevant.

Figure 10 shows the results for different resampling methods with a specific target. The square border in the original image is one pixel thick. Our method preserves the original pixel color information that facilitates search for a particular color belonging to the target object.

Other results for resampling methods are shown in Fig. 11. It is possible to observe that our method preserved the geometric characteristics of the original image, whereas none of the other tested techniques maintained the one-pixel separation between the parallel lines present in the original image. This may be advantageous in the search for images by texture since the information in the original image texture patterns are preserved in the subsamples.

Figure 12 shows the results for a multi-resolution search using the hypercube. In this experiment, the purpose is to search for a specific blue car at the bottom center of the image. The search starts at a resolution from 1×1 to $w \times h$, where w and h are the width and height of the image, respectively. The car was found in the 3,035-th pixel of the image with resolution of 64×64 pixels. Therefore, the total cost was $2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 3,035 = 3,098$ accesses. On the other hand, a sequential search on the entire image found the pixel with cost of 422,660 accesses.

The number of operations required for a pixel interpolation is a commonly used strategy for assessing complexity cost [1]. The metric includes the operations demanded both for the convolution and for the calculation of the basis function. The convolution operation of an $m \times m$ mask requires m^2 multiplications and $m^2 - 1$ additions. Table 1 reports the number of operations per pixel for the nearest neighbor, bilinear and bicubic interpolation methods, as well as for the proposed hypercubic approach. Our method has cost 0 for both operations since it uses the original pixel values.

Table 1 Number of operations per pixel required for different interpolation methods.

Operation	Interpolation Method			
	Nearest	Bilinear	Bicubic	Hypercubic
Addition	2	16	22	0
Multiplication	0	18	29	0

5 Conclusions and Future Work

In this work, we presented and evaluated an approach to solving a number of image analysis problems using the hypercube graph. We were able to satisfactorily deal with different problems in a simple and flexible way.

We performed transformations on the images with only the granularity of the hypercube vertices and edges. The strategy for generating the edge set is easily parallelized and the transformations hereby described are restricted to simple logic operations that simplified the final hardware design.

As directions for future work, we intend to propose different logical operations and apply the method to other image analysis problems.

Acknowledgments The authors are grateful to FAPESP - São Paulo Research Foundation (Grant 2011/22749-8), CNPq (Grant 307113/2012-4) and CAPES for their financial support.

References

1. Amanatiadis, A., & Andreadis, I. (2009). A survey on evaluation methods for image interpolation. *Measurement Science and Technology*, *20*(10), 1–9.

2. Asratian, A.S., Denley, T.M., & Häggkvist, R. (1998). *Bipartite Graphs and their Applications* Vol. 131. Cambridge: Cambridge University Press.
3. Bhuyan, L., & Agrawal, D. (1984). Generalized hypercube and hyperbus structures for a computer network. *IEEE Transactions on Computers*, *C-33*(4), 323–333.
4. Bollobás, B. (1998). Modern Graph Theory, vol. 184 Springer Science & Business Media.
5. Boykov, Y., & Funka-Lea, G. (2006). Graph cuts and efficient ND image segmentation. *International Journal of Computer Vision*, *70*(2), 109–131.
6. Chan, T., & Saad, Y. (1986). Multigrid algorithms on the hypercube multiprocessor. *IEEE Transactions on Computers*, *C-35*(11), 969–977.
7. Chen, C., Lee, S., & Cho, Z. (1990). A parallel implementation of 3-D CT image reconstruction on hypercube multiprocessor. *IEEE Transactions on Nuclear Science*, *37*(3), 1333–1346.
8. Chepoi, V., & Hagen, M.F. (2013). On Embeddings of CAT (0) Cube Complexes into Products of Trees via Colouring their Hyperplanes. *Journal of Combinatorial Theory Series B*, *103*(4), 428–467.
9. Cover, T.M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE Transactions on Electronic Computers (3), 326–334.
10. Cypher, R., Sanz, J., & Snyder, L. (1989). Hypercube and Shuffle-Exchange algorithms for image component labeling. *Journal of Algorithms*, *10*(1), 140–150.
11. Darbon, J., & Sigelle, M. (2006). Image restoration with discrete constrained total variation Part I: Fast and exact optimization. *Journal of Mathematical Imaging and Vision*, *26*(3), 261–276.
12. Day, K., & Tripat, A. (1994). A comparative study of topological properties of hypercubes and star graphs. *IEEE Transactions on Parallel and Distributed Systems*, *5*(1), 31–38.
13. Deshpande, S., & Jenevin, R.M. (1986). Scalability of a binary tree on a hypercube. International Parallel and Distributed Processing Symposium, 661–668.
14. Elmoataz, A., Lezoray, O., & Bougleux, S. (2008). Nonlocal discrete regularization on weighted graphs: a framework for image and manifold processing. *IEEE Transactions on Image Processing*, *17*(7), 1047–1060.
15. Eshera, M., & Fu, K.S. (1986). An Image Understanding System using Attributed Symbolic Representation and Inexact Graph-Matching. IEEE Transactions on Pattern Analysis and Machine Intelligence (5), 604–618.
16. Firsov, V. (1965). Isometric embedding of a graph in a boolean cube. *Cybernetics and Systems Analysis*, *1*(6), 112–113.
17. Gonzalez, R.C., & Woods, R.E. (2006). *Digital image processing*. NJ: Prentice-Hall, Inc.
18. Harary, F., Hayes, J.P., & Wu, H.J. (1988). A survey of the theory of hypercube graphs. *Computers & Mathematics with Applications*, *15*(4), 277–289.
19. Hayes, J.P. (1976). A graph model for Fault-Tolerant computing systems. *IEEE Transactions on Computers*, *100*(9), 875–884.
20. Ho, C., & Johnsson, S. (1989). Spanning balanced trees in boolean cubes. *SIAM Journal on Scientific and Statistical Computing*, *10*(4), 607–630.
21. Horowitz, E., Sahni, S., & Rajasekaran, S. (2007). Computer Algorithms/C++ silicon press.
22. Hussain, Z., Bose, B., & Al-Dhelaan, A. (2014). Generalized hypercubes: Edge-disjoint hamiltonian cycles and gray codes. *IEEE Transactions on Computers*, *63*(2), 375–382.
23. Intel 17455-03 (1985). IPSC User's Guide. Portland, OR, USA.
24. Jeulin, D. (2014). Random tessellations generated by boolean random functions. *Pattern Recognition Letters*, *47*, 139–146.
25. Johnsson, S.L. (1987). Communication effect basic linear algebra computations on hypercube architectures. *Journal of Parallel and Distributed Computing*, *4*(2), 133–172.
26. Kautz, W. (1958). Unit-Distance Error-Checking Codes. IRE Transactions on Electronic Computers (2), 179–180.
27. Kolmogorov, V., & Zabin, R. (2004). What Energy Functions can be Minimized via Graph Cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(2), 147–159.
28. Kuo, C.N., Chou, H.H., Chang, N.W., & Hsieh, S.Y. (2013). Fault-Tolerant Path embedding in folded hypercubes with both node and edge faults. *Theoretical Computer Science*, *475*, 82–91.
29. Lai, T.H., & White, W. (1990). Mapping pyramid algorithms into hypercubes. *Journal of Parallel and Distributed Computing*, *9*(1), 42–54.
30. Leighton, F.T. (2014). Introduction to Parallel Algorithms and Architectures: Arrays·Trees· Hypercubes Elsevier.
31. Martín-Herrero, J. (2007). Anisotropic diffusion in the hypercube. *IEEE Transactions on Geoscience and Remote Sensing*, *45*(5), 1386–1398.
32. Mathew, K.A., & Östergård, P.R. (2015). On Hypercube Packings, Blocking Sets and a Covering Problem. *Information Processing Letters*, *115*(2), 141–145.
33. Mendez-Rial, R., & Martin-Herrero, J. (2012). Efficiency of Semi-Implicit schemes for anisotropic diffusion in the hypercube. *IEEE Transactions on Image Processing*, *21*(5), 2389–2398.
34. Mudge, T. (1987). An analysis of hypercube architectures for image pattern recognition algorithms. In *Proc. SPIE 0755, image pattern recognition: Algorithm implementations, techniques, and technology, pp. 71–83. International society for optics and photonics*.
35. nCUBE V1.0. (1990). *NCUBE* Vol. 6400. Beaverton: Processor Manual.
36. Pease, M.C. (1977). The Indirect Binary n-Cube Microprocessor Array. *IEEE Transactions on Computers*, *C-26*(5), 458–473.
37. Raju, S.V., & Govardhan, A. (2013). Overlapped text partition algorithm for pattern matching on hypercube networked model global journal of computer science and technology 13(4).
38. Ranka, S., & Sahni, S. (2012). Hypercube Algorithms with Applications to Image Processing and Pattern Recognition Springer Science & Business Media.
39. Saad, Y., & Schultz, M.H. (1988). Topological properties of hypercubes. *IEEE Transactions on Computers*, *37*(7), 867–872.
40. Schuld, M., Sinayskiy, I., & Petruccione, F. (2014). Quantum walks on graphs representing the firing patterns of a quantum neural network. *Physical Review A*, *89*(3), 032,333.
41. Shankar, R.V., & Ranka, S. (1992). Hypercube algorithms for operations on quadtrees. *Pattern Recognition*, *25*(7), 741–747.
42. Silva, E., Guedes, A., & Todt, E. (2013). Independent Spanning Trees on Systems-on-chip Hypercubes Routing. In *International conference on electronics, circuits and systems, vol. 75, pp. 93–96*.
43. Skiena, S. (1990). Hypercubes. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica 148–150.
44. Trémeau, A., & Colantoni, P. (2000). Regions adjacency graph applied to color image segmentation. *IEEE Transactions on Image Processing*, *9*(4), 735–744.
45. Uyttendaele, M., Eden, A., & Skeliski, R. (2001). Eliminating ghosting and exposure artifacts in image mosaics. In *IEEE*

Computer society conference on computer vision and pattern recognition, vol. 2, pp. II–509. IEEE.
46. Wang, E., Wang, H., Chen, J., Gong, W., & Ni, F. (2014). A Novel Node-to-Set Node-Disjoint Fault-Tolerant Routing Strategy in Hypercube. In *Advanced computer architecture, pp. 58–67. Springer*.
47. Wang, X., Li, H., Bichot, C.E., Masnou, S., & Chen, L. (2013). A Graph-Cut Approach to Image Segmentation using an Affinity Graph based on l_0- Sparse Representation of Features. In *20Th IEEE international conference on image processing, pp. 4019–4023. IEEE.*
48. Wu, R.Y., Chen, G.H., Fu, J.S., & Chang, G.J. (2008). Finding cycles in hierarchical hypercube networks. *Information Processing Letters*, *109*(2), 112–115.
49. Wu, Z., & Leahy, R. (1993). An Optimal Graph Theoretic Approach to Data Clustering: Theory and its Application to Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*(11), 1101–1113.
50. Yang, J.S., & Chang, J.M. (2014). Optimal independent spanning trees on cartesian product of hybrid graphs. *The Computer Journal*, *57*(1), 93–99.
51. Ziavras, S.G., & Siddiqui, M.A. (1993). Pyramid mappings onto hypercubes for computer vision: Connection machine comparative study. *Concurrency: Practice and Experience*, *5*(6), 471–489.
52. Ziavras, S.G., & Sideras, M.A. (1995). Facilitating High-Performance image analysis on reduced hypercube (RH) parallel computers. *International Journal of Pattern Recognition and Artificial Intelligence*, *9*(4), 679–698.

Eduardo Sant'Ana da Silva is currently a software architect and consultant. He received his B.Sc., M.Sc. and Ph.D. degrees in Computer Science from the Federal University of Parana, Brazil. His research interests include image processing, machine learning, software engineering, computer networks.

Helio Pedrini is currently a professor in the Institute of Computing at the University of Campinas, Brazil. He received his Ph.D. degree in Electrical and Computer Engineering from Rensselaer Polytechnic Institute, Troy, NY, USA. He received his M.Sc. in Electrical Engineering and his B.Sc. in Computer Science, both degrees from the University of Campinas, Brazil. His research interests include image processing, computer vision, pattern recognition, machine learning, computer graphics.