# FPGA dedicated hardware architecture of 3D medical image reconstruction: marching cubes algorithm

BOURAOUI MAHMOUD[1,2], NADIA NACER[1], MANEL MILI[1],MOHAMED HEDI BEDOUI[1]
[1]Laboratory of Medical Imaging and Technology – Faculty of Medicine at Monastir – University of Monastir - TUNISIA
[2]National Engineering School at Sousse – University of Sousse – TUNISIA
Bouraoui.mahmoud@eniso.rnu.tn, nd.nacer@gmail.com; Medhedi.bedoui@fmm.rnu.tn

*Abstract:* - Generally, the implementation of various treatments of medical images, especially the 3D reconstruction, should support a real-time execution of the algorithm on the architecture: It is to meet the constraints of latency and circuit space while minimizing the resource consumption.
We are interested in this project for a 3D reconstruction of medical images on a platform based on an FPGA. We have as an objective the specification and hardware implementation of the reconstruction algorithm, the Marching Cubes.

*Key-words:* - FPGA, Implementation, 3D, 2D Images, Reconstruction, Marching Cubes, Medical image.

## 1 Introduction

The 3D reconstruction consists in providing a volumetric representation of an object from some 2D images that ensure a description of the 3D volume. The involvement of these algorithms is important in various fields, especially in medicine and biology.

The Marching Cubes algorithm is the most used for the isosurface 3D reconstruction [1]. This algorithm was designed by William E. Lorensen and Harvey E. Cline in 1987 [2] to generate a 3D model for an interesting anatomical structure. For that, it uses a threshold (characteristic value of anatomical structure for studying a human organ, which will be defined by the medical expert; each organ has its proper threshold) and previously-segmented images [3] [4].

We have known that the application implementation in processing and the 3D reconstruction images must often respect real-time execution, while minimizing the resource consumption when targeting systems with a low cost. These later are able to integrate the maximum processing algorithms.

Our objective is to make the hardware design FPGA-implementation of a fast and robust 3D reconstruction by defining a specific hardware architecture for the Marching Cubes algorithm. In this article, we describe the modeling and description of the architecture of the various parts of the Marching Cubes algorithm for the hardware implementation on the FPGA. We apply the approach to a high-level design. We start with the specification and description of the system and we end with the implementation of an FPGA of Altera (Cyclone) through simulation and synthesis. Finally, we report and discuss the results of this implementation.

We present in the first part of this article the approach to 3D reconstruction from parallel slices (2D images) segmented using the Marching Cubes algorithm. In the second part we present the methodology used for its implementation on an FPGA circuit, the dedicated design architecture and the results of synthesis and implementation.

## 2 Marching Cubes algorithm

The flowchart in Fig. 1 presents the overall operation of the Marching Cubes algorithm.

### 2.1 Step 1: Voxel Extracting

Two successive and adjacent images provide 8 pixels forming a voxel. A system of coordinates of the eight vertices of the cube must be generated to be used later to create the triangles of the surface in this logical cube, determining the intersections of the surface with the cube and then going to the next cube (Fig. 2).

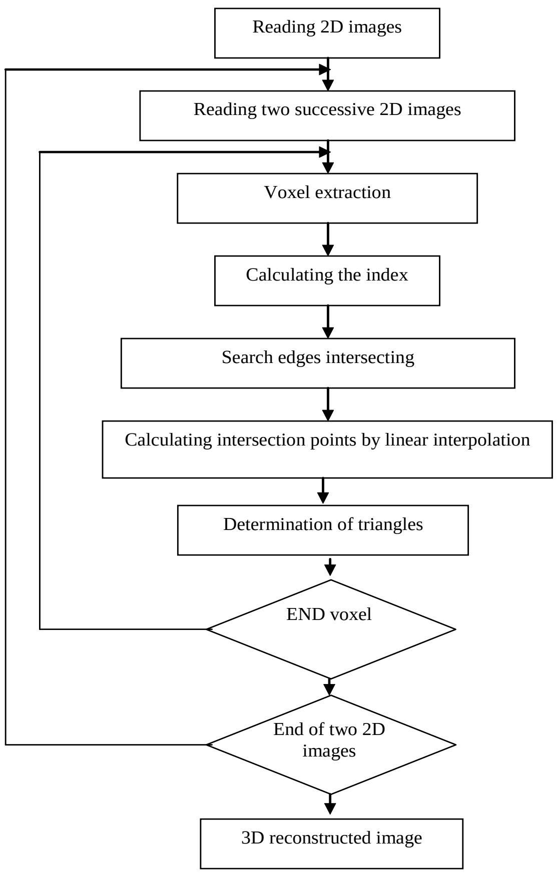
Reading 2D images
Reading two successive 2D images
Voxel extraction
Calculating the index
Search edges intersecting
Calculating intersection points by linear interpolation
Determination of triangles
END voxel
End of two 2D images
3D reconstructed image

Fig. 1: Flowchart of the Marching Cubes algorithm

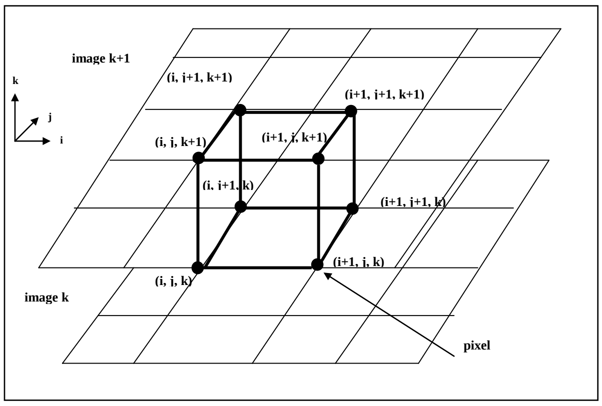

Fig. 2: Voxel extraction

Once the voxel is defined, it should be noted that it is necessary to choose a convention for numbering the vertices of the cube and edges. For our algorithm, we have chosen the convention of Paul Bourke shown in Fig. 3 [6].

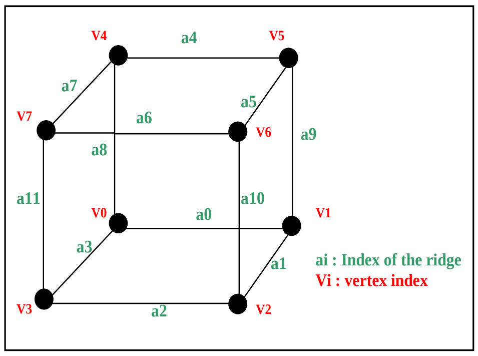

Fig. 3: Numbering vertices and edges

## 2.2 Step 2: Calculation of the index

First, we set a threshold or iso-value density and we browse the cube by a cube space. For each cube, we distinguish:

• "Internal point": any vertex whose density (gray level) is below the threshold (iso-value).

• "External Point": any vertex whose density (gray level) is above the threshold (iso-value).

Once these models are distinguished, we can recognize them from an index created according to their vertices. This index is a byte where each bit is associated with a vertex; the bit is set to "1" if the point is internal to the surface; otherwise, it is "0".

The surface intersects the edges of the cube when the two vertices forming the edge have opposite signs (one is "0" the other is "1").

Fig. 4 shows, from left to right, the index creation illustration and a concrete example of the index formation. In this example, we have a cube whose vertices 6 and 3 are considered internal points on the surface; hence the index will have a value of 24H.

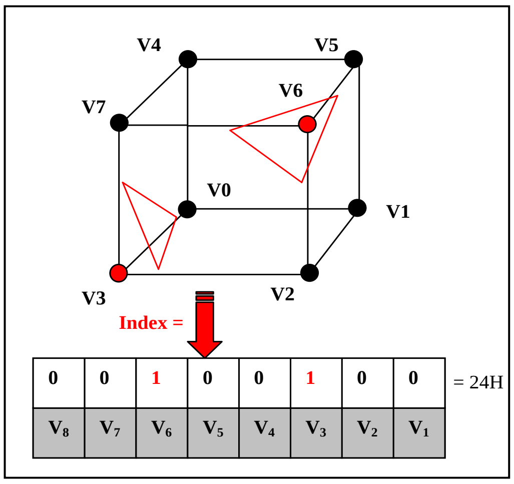

Fig. 4: The index creation

This index will be used later as a pointer in the Tri-Table and Edge-Table [1]. These two tables allow defining all the intersections on the edges of the cubes and the topologies of the triangles to draw [1] [5].

### 2.3 Step 3: Determining edges intersecting

Once the index is calculated, we can access an array, denoted as the Edge-Table, which contains all the possible topologies inside a cube and tells us which edges cut the surface [1][5]. It just reads the corresponding value of the cube_index in the Edge-Table index, which is encoded in 12 bits: if the ith bit is 1, the surface intersects the ith edge of the cube; otherwise, it does not cut it. In particular, if the line number cube_index of the Edge-Table is "0", then the surface does not cut the cube.

If the vertex number 3, for example, is below the selected iso-value and all the other vertices are above this value, then a triangle cuts edges 2, 3 and 11 (Fig. 5) [1][5].

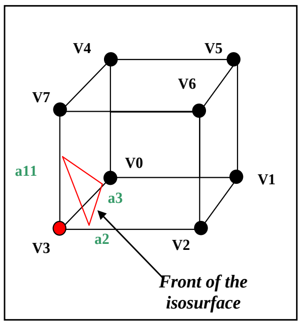

Fig. 5: Example of the surface intersection with the cube [5]

The exact position of the vertex of the triangle depends on the relationship between the iso-value and the values of the vertex 3-2, 3-0 and 3-7 respectively (Figure 5) [1][5].

The cube_index is: 8H = "00001000", which is in the table edges to [8H] = "100000001100". Bits 2, 3, and 11 are ‘1‘and the others are ‘0‘; i.e, the edges 2, 3 and 11 are cut by the iso-surface. Then, we have to determine the positions of the intersections on these edges [5].

### 2.4 Step 4: Determining the intersection points by a linear interpolation

The intersection points are calculated by a linear interpolation [1][5][6]. Indeed, let P1(X, Y, Z) and P2(X, Y, Z) be the coordinates of the two vertices of the edge intersected and V1 and V2 be the scalar values of the corresponding intensities at each vertex, then the coordinate of the intersection point P(X, Y, Z) is given by the following formulas:

$$\begin{cases} P_X = P_{1X} + \dfrac{(isovalue - V_1)}{(V_2 - V_1)} * (P_{2X} - P_{1X}) & (1) \\ P_Y = P_{1Y} + \dfrac{(isovalue - V_1)}{(V_2 - V_1)} * (P_{2Y} - P_{1Y}) & (2) \\ P_Z = P_{1Z} + \dfrac{(isovalue - V_1)}{(V_2 - V_1)} * (P_{2Z} - P_{1Z}) & (3) \end{cases}$$

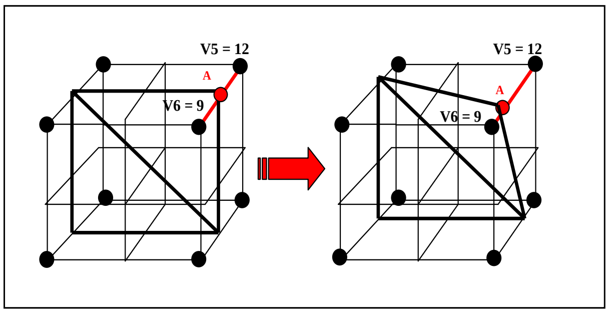

Fig. 6: Illustration of linear interpolation

In Fig. 6, the iso-value is 10. Before interpolation, the point A is in the middle of the ridge although Mathematics 10 is not equidistant between 9 and 12. After interpolation, the point A is twice close to 9 than to 12.

### 2.5 Step 5: Determining the triangles inside the cube

This step of the algorithm is to create triangles inside the found surface. To determine these triangles, use the index calculated as entered in the table of configurations (Tri-table) which will allow us to define all the intersections on the edges of the cubes. Each line in the table describes a triangle topology. Therefore, these triangles allow the surface representation of the volume to be determined. Such a process is called triangulation [1][6].

## 3 Implementation of the Marching Cubes algorithm on FPGA

### 3.4 Hardware design and description of the Marching Cubes algorithm

We divide the Marching Cubes algorithm into two modules named respectively: managing memory and triangles calculation (Fig. 7).

The access time to the 4 pixels of each image, to form a voxel (8 pixels), is greatly predominant relative to the calculation time for each cube. The execution time for calculating 4 pixels is proportional to the memory access number performed by the computing operation of the 4 pixels.

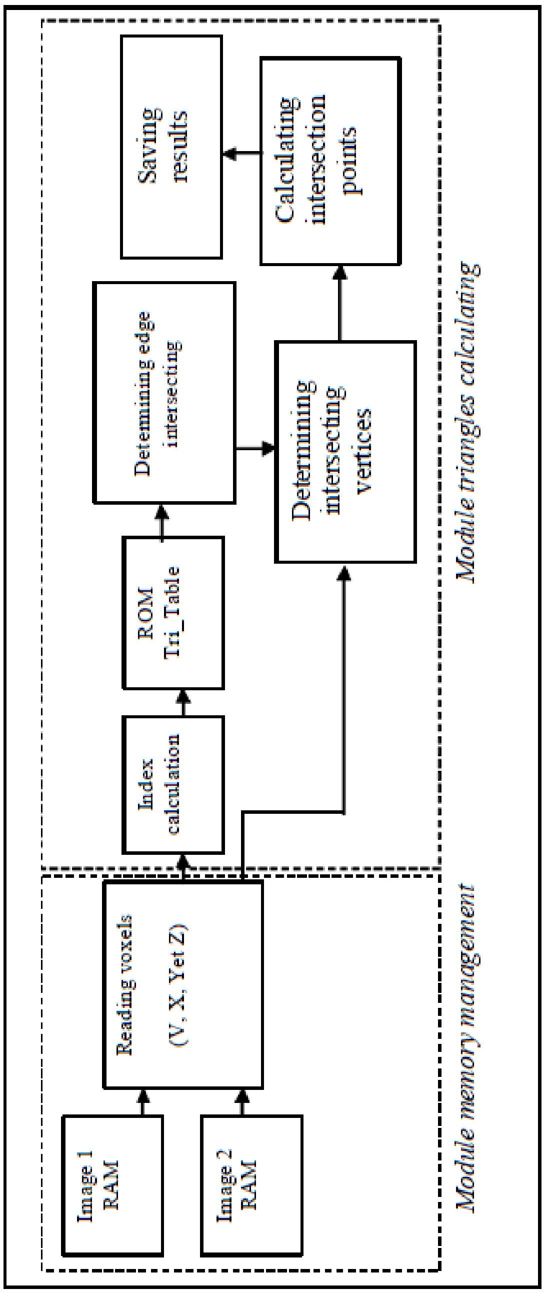
Saving results
Calculating intersection points
Determining edge intersecting
Determining intersecting vertices
ROM Tri_Table
Index calculation
Module triangles calculating
Reading voxels (V, X, Y et Z)
Image 1 RAM
Image 2 RAM
Module memory management

Fig. 7: Block diagram of the Marching Cubes algorithm

### 3.1.1 Module memory management

The memory management system allows you to manage two RAMs to read both successive images and do a Voxel-by-Voxel scan. At the end, eight pixel values are generated corresponding to each voxel. Fig. 8 shows the architecture of the memory management module.

The image format used is (64 x 64). To store two images, we have designed two dual-port RAMs with a size of (64 x 64 x 8)bits. The coordinates (x, y, z) and the pixel intensities are coded in 8 bits. Z represents the image number which belongs to the pixel being processed. That is why an address generator is designed for the management of two memories storing the two images for reading the pixel intensities. This module allows reading the intensities of 8 pixels (4 pixels in each image) then storing their respective coordinates (x, y, z) in registers.

These eight intensity values will then be the inputs of a comparator that can compare against a threshold called the iso-value (image segmentation in real time). An index will be generated by the comparator, which is noted as the cube_index and coded in 8 bits. The coordinates (x, y, z) of each pixel are used in the second part of the algorithm for calculating the coordinates of the intersection points and the creation of the triangles of the surface.

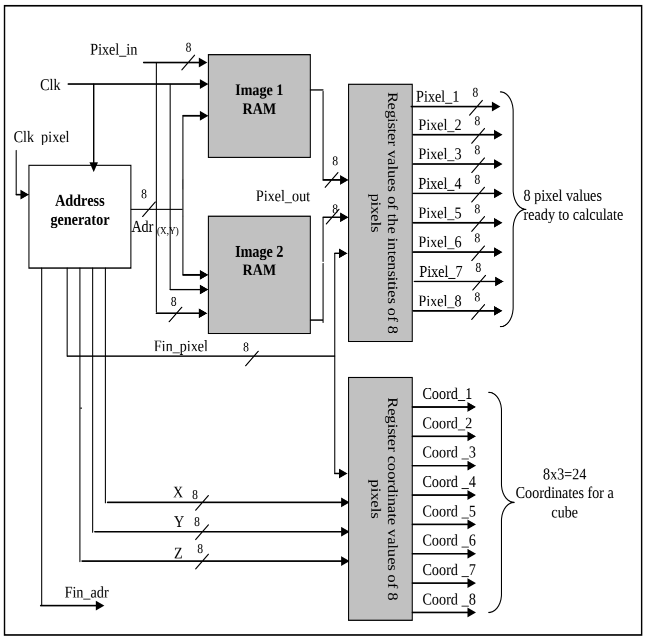

Fig. 8: Architecture of the module memory management

We have developed a comparison module for calculating the cube index. Using this index, we can access a Tri_Table, which contains all the possible topologies inside a cube and tells us the number of edges that cut the surface in the ordered construction triangles from each three intersection points. To do this, we have used a ROM to record all the intersection cases (256 topologies), already listed. Going through this triangles table, for a line of 16 elements, the edge numbers are obtained on which we must calculate the intersection points to construct triangles.

For each generated intersected edge number, it must first find the vertices that form the edge and their contact information and then proceed to calculate the intersection points on each edge using a linear interpolation. Also, the intersection points calculated for each edge have three values (X, Y, Z). For this first recording is done in three registers; each stores a value according to these coordinates. It will then be divided into three data RAMs for each configuration so that they can be used to construct the surface.

The architectures of the different modules of the Marching Cubes algorithm are modeled in VHDL using modelsim and the Altera Quartus II software (Fig. 9).

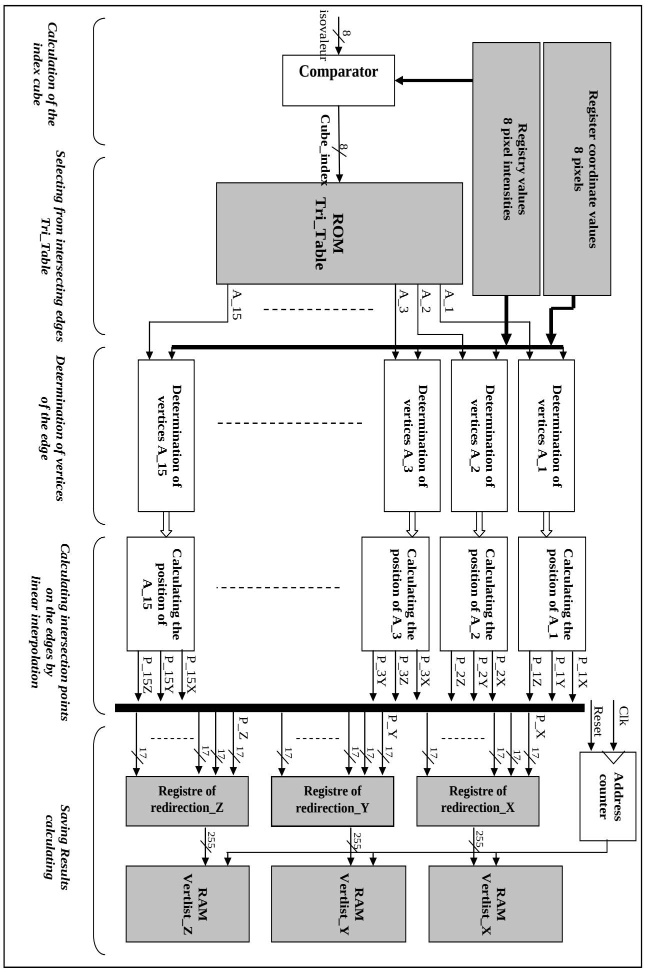
Calculation of the index cube
Selecting from intersecting edges Tri_Table
Determination of vertices of the edge
Calculating intersection points on the edges by linear interpolation
Saving Results calculating
isovaleur
8
Comparator
Cube_index
8
ROM Tri_Table
Registry values 8 pixel intensities
Register coordinate values 8 pixels
A_15
A_3
A_2
A_1
Determination of vertices A_15
Determination of vertices A_3
Determination of vertices A_2
Determination of vertices A_1
Calculating the position of A_15
Calculating the position of A_3
Calculating the position of A_2
Calculating the position of A_1
P_15Z
P_15Y
P_15X
P_3Y
P_3Z
P_3X
P_2Z
P_2Y
P_2X
P_1Z
P_1Y
P_1X
P_Z
P_Y
P_X
17
Clk
Reset
Address counter
Registre of redirection_Z
Registre of redirection_Y
Registre of redirection_X
255
RAM Vertlist_Z
RAM Vertlist_Y
RAM Vertlist_X

Fig. 9: Architecture of the triangles computing module

## 4 Results of the synthesis and implementation on FPGA

We implement the architectures of the Marching Cubes algorithm on the FPGA circuit Altera Cyclone family. Table I shows the main characteristics of this circuit.

Table 1: characteristics of FPGA circuits

| FPGA Circuit | LE number | Bits select RAM | PLL |
|---|---|---|---|
| **Cyclone EP1C20** | **20,060** | **294,919** | **2** |

### 4.1 Results of the synthesis and implementation on the FPGA Cyclone

To test our application we have spent an implementation on an FPGA of the Altera family. We have used a Nios development Kit, Cyclone Edition Altera firm. This card is built around a Cyclone EP1C20 FPGA. This development environment mainly offers a 50MHz clock, 8Mbits flash memory, static RAM 1Mbit, and 16 Mbit SDRAM.

Table II: Results of the synthesis and implementation of the Cyclone FPGA circuit modules of the Marching Cubes algorithm

| Modules of *« Marching Cubes »* | Blocs of *« Marching Cubes »* | Number of EL | Memory (bits) |
|---|---|---|---|
| **Memory management** | Clock divider | 23 | 0 |
| | Generating pixel location | 95 | 0 |
| | Conversion coordinate address | 17 | 0 |
| | Storage memory (RAM) | 0 | 128 x 2 |
| | **Total** | **135** | **256** |
| **Calculating triangles** | Calculating the index | 24 | 0 |
| | Edge selection (Tri_Table) | 96 | 16,384 |
| | Determining the vertices | 404 | 0 |
| | Calculating the intersection points | 17,430 | 0 |
| | Saving results | 0 | 255 x 3 |
| | **Total** | **17,954** | **17,405** |
| **Marching Cubes** | **Total** | **18,089** | **17,369** |

The data in Table II show the results of the synthesis and implementation in terms of number of the ELs and memory space used for the various modules of

the Marching Cubes algorithm. These results are given for one voxel.
From Table II, the number of logic elements used for the implementation of the Marching Cubes algorithm on the FPGA Cyclone EP1C20 is important. It occupies almost the entire circuit (20,000 EL). The memory space of the circuit EP1C20 Cyclone can be considered sufficient for processing the images of small sizes (up to 8 x 8), where as for large images, the space required is very large and we need to change the circuit or find other optimization solutions to reduce the size of the application. In this application, we have used the resources memory of the FPGA development card beforehand. For the new FPGA circuit the architecture is very developed (important resources memory and electronics); there will be no limit for the implementation of this application.

### 4.2 Performance of the implementation of the Marching Cubes algorithm on the FPGA Cyclone

The data in Table III show the synthesis result (result obtained in the synthesis of the Altera Quartus tool) number of ELs, memory space and execution time required to implement the Marching Cubes algorithm on the FPGA for different image sizes as inputs. The circuit is operating with an internal 50 MHz timing.

Table III: Synthesis results of the hardware implementation of the Marching Cubes algorithm on FPGA

| | | *« Marching Cubes »* | | |
|---|---|---|---|---|
| **Image number** | **Image Size** | **Number of(EL)** | **Memory (Mbits)** | **Execution time (µs)** |
| Un Voxel | Un Voxel | 18,089 | 0.017 | 0.085 |
| **24** | 64 x 64 | 18,089 | 70.955 | 1,826.2 |
| | 128 x 128 | 18,089 | 287.04 | 31,533 |
| | 256 x 256 | 18,089 | 1,156.44 | 127,121 |

### 4.3 Discussion and performance of the proposed system

The high-level design of the integrated circuits appears to be an extremely powerful tool to explore the architectures and perform the 3D reconstruction systems of medical images in real time. In an FPGA, prototyping allows validating the description. The use of a high-level synthesis reduces the design time, exploring several architectural alternatives. However, the integration of a medical imaging application in a programmable integrated circuit (FPGA) is connected to the satisfaction of the performance designer from a cost, power consumption, or flexibility point of view.
We note that a hardware implementation of the reconstruction algorithm of the 3D Marching Cubes has given good results in terms of execution and space recovery time, which is the main characteristics of the FPGA (parallelism and speed performance) that has contributed much to reduce the processing time of the reconstruction system reads, where most of the algorithm modules have an architecture of a parallel operation.
To apply the Marching Cubes algorithm of 24 images with a size of 256 x 256 we need 1,156.44 Mbps, which requires the use of FPGAs with an internal memory above 1GB or use an external memory FPGA which will penalize the execution time (increase of the external memory access time).

### 4.3 The 3D image reconstructed result

We used 24 images (64 * 64) scintigraphic of the heart. The 24 images are previously segmented (Fig. 10). We have stored the 24 images in an external memory on the FPGA development board. The Fig. 11 shows the reconstructed 3D image after the execution of the Marching Cubes algorithm in the FPGA circuit.

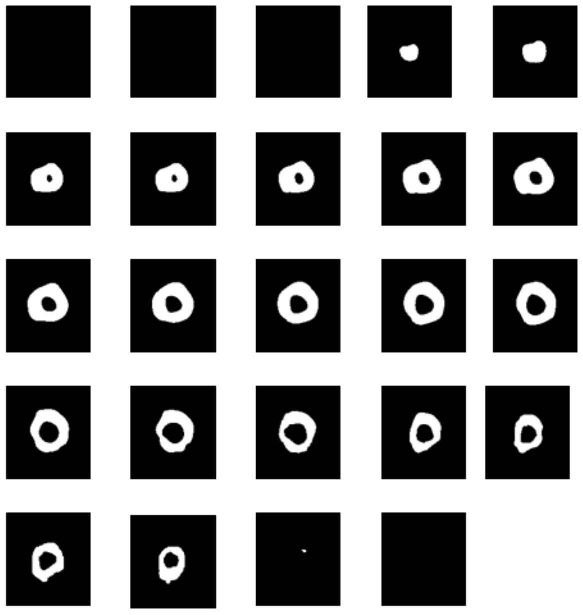

Fig. 10: 24 images (64 * 64) scintigraphic of the heart

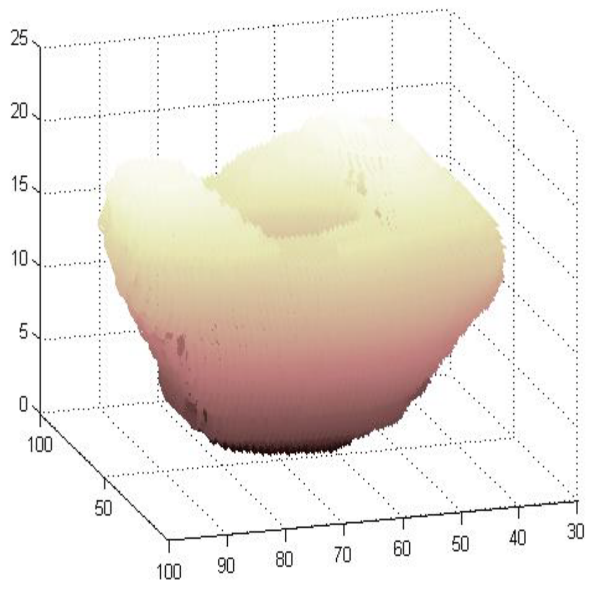

Fig. 11: The reconstructed 3D image of the heart

## 5 CONCLUSION

In this paper, we have introduced the modular design of the Marching Cubes algorithm for hardware implementation on an FPGA programmable circuit. The necessary steps have been taken, namely the system, simulation, synthesis and implementation on an FPGA specification.
The developed Marching Cubes algorithm has been divided into two major modules and each module is divided into several modules. The first module allows the management of cuts and the generation of voxel-by-voxel data. The second allows us to perform the necessary calculations and record the results in memory.
The time required to run the application on a physical medium processing is greatly reduced. This is due to the fact that the FPGA circuits fully exploit the potential parallelism of a given algorithm. In addition, the results of the synthesis and implementation on the FPGA Cyclone generation has shown the extended capabilities in terms of speed of the achieved system.

*References*

[1] William E. Lorensen and Harvey E. Cline. Marching cubes: “A high resolution 3D surface construction algorithm”. *Computer Graphics*, 21(4), July 1987, pp. 163–169.

[2] A. Lopes, K. Brodlie, Improving the Robustness and Accuracy of the Marching Cubes Algorithm for Isosurfacing. *IEEE Transactions on Visualization and Computer Graphics*, 9(1), 2003, pp. 16-29.

[3] Timothy S. Newman_, Hong Yi , “A survey of the marching cubes algorithm”, *Elsevier, Sciencedirect, Computers & Graphics*, 30, 2006, pp. 854–879.

[4] V. Gelder, J. Wilhelms, Topological considerations in isosurface generation. *ACM Trans Graph*; 13, 1994, p: 337–75.

[5] Paul Bourke, Polygonising a scalar field Also known as: "3D Contouring," "Marching Cubes," "*Surface Reconstruction*;" May 1994,

[6] Adriano Lopes and Ken Brodlie, « Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing » . *IEEE Transactions on Visualization and Computer Graphics*, 9(1), 2003, pp.16.29.