MLC filtering applied to the 3D reconstruction of the left ventricle

O. García[†], A. Susín[‡].

† Departamento de Tecnologías Audiovisuales. Universidad Ramon Llull
Quatre Camins, 2, 08022 - Barcelona,
Tlf: 932.902.442 Fax: 932.902.420. e-mail: oscarg@salleURL.edu

‡ Departamento de Matemática Aplicada I. Universidad Politécnica de Catalunya
Diagonal, 647 3º, 08028 - Barcelona,
Tlf: 934.017.781 Fax: 934.016.069. e-mail: toni.susin@upc.es

Abstract

As presented in previous editions, our method describes the three-dimensional reconstruction of the internal and external surfaces of the human's left ventricle from medical data. The reconstruction is a first process inside a complete VR application that will serve as an important diagnosis tool for hospitals.

The segmentation process has been thoroughly refined in order to provide a robust border detection. Edges must be labeled as internal or external borders in order to allow the posterior dynamic method to perform the 3D reconstruction of the surfaces. The filtering process is based on the Maximum-Likelihood Classification approach. The algorithm rewards an edge depending on its probability to be a surface border. Probabilities are derived from several key characteristics.

Keywords: Tele-medicine, deformable models, virtual reality, image filters, 3D surface.

1 Introduction

3D medical imaging systems have several applications like support on diagnosis or surgery planning. However they can be still improved by auxiliary modules that might improve the reliability of the deductions and, moreover that, benefit other areas like student's training or assisted remote-operation. Typical protocols are based on manual analysis with specific image-processing software, which requires deep medical knowledge and experience. It comes clear that those attributes will be always fundamental but physician's decisions can be easier with a little help from automated procedures.

The algorithms here presented are specifically designed for the medical context of cardiology. In particular, the algorithms are designed for the left ventricle automated reconstruction. The system takes as its input SPECT, *Single Photon Emission Computed Tomography*, imagery. SPECT imagery gives functionality information about the organ with no clue on its anatomy. Thus this data shows the activity inside with no information about shape. Therefore isquemic areas, which are in absence of blood irrigation, won't be shown in the images. That's the case of ventricle areas being affected by a heart attack.

In this edition, we provide an efficient and self-made segmentation algorithm. There exist generic

segmentation procedures [1,2] reliable and accurate. The problem appears due to the concrete nature of the SPECT images, which are very difficult to process without a very precise analysis.

Maximum likelihood classification [3] is based upon the assumption that there exist several models that might be used for classifying items, as pixels or voxels. The class that an item belongs to is determined by calculating which of the models is more likely to describe it. In terms of probability, the item belongs to the class that maximizes its probability term.

We can use this technique for our purposes by building a robust segmentation algorithm that labels voxels if they are considered to be the left ventricle borders. Hence the algorithm distinguishes between internal and external borders, referred to the endocardium and epicardium layers of the human heart.

This algorithm improves our previous implementations [4,5] in terms of reliability, robustness and automation for the complete process directly from the 2D images to the 4D beating ventricle.

2 MLC filtering of the medical images

Classification is the grouping of each pixel (or voxel) to one of the classes, internal or external borders, on the basis of some probability. This probability states if the pixel is likely, or not, to be a part of the border. In terms of border detection, we need to get the internal and external groupings of voxels in order to stop the dynamic surfaces during the reconstruction process.

Given a data slice, there are several generic segmentation methods that we can apply, as explained in section 1. Those methods detect gradient changes that can be used as the first sight to the edges that we are looking for. After applying the segmentation, we can state that every voxel forms part of an edge that can be labeled independently. Then we must define some decision rules in order to classify every edge as belonging to one of the possible classes: external border, internal border or none.

Once the edges have been labeled and classified, we can find the vectorial field that will act as an external force for the reconstruction phase. See figure 1:

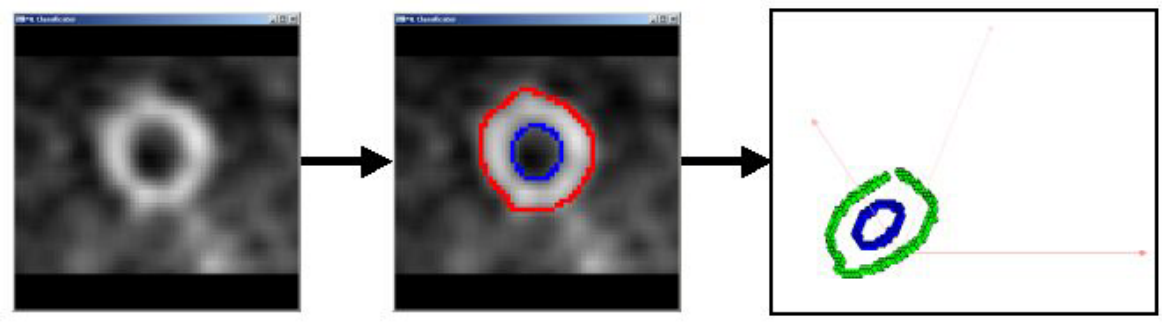

Figure 1. From left to right: a left ventricle data slice; 2D identified borders; borders in 3D space that will be used for getting the 3D vectorial field.

The following sub-sections describe our MLC implementation for the segmentation part. This implementation automates the segmentation, labeling, probability checking and border selection processes.

2.1 Search for the automatic circle

As a prior filter, typical segmentation software for physicians allows them to define a manual circle. This circle acts as a noise-removal tool that deletes everything outside its diameter. It is a key-process for the diagnostic because of its posterior influence on the measures of ejection fraction and wall-thickness [6]. Our intention is to automate the circle search in order to get the smallest one. Once the circle is well fit to data, border detection gets much easier.

As a pre-process to the automated circle determination, we begin by applying a generic segmentation algorithm like [1,2]. After that, we have a first sight at the edges. Those are cut by a very coarse manual circle, as showed in figure 2:

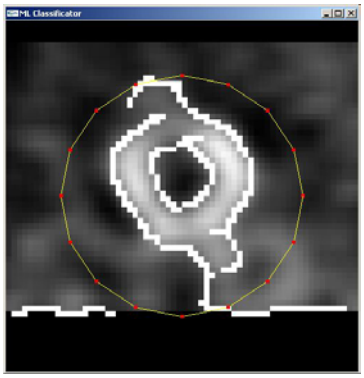

Figure 2. Edges detected by the generic segmentation algorithm (white). The coarse circle (yellow).

Tiny edges, which are considered spurious, are deleted. After that we have a first estimation of the edges that might finally be borders. Now we can begin with the automatic circle search. We split the process in two: finding the right centroid and getting the smallest radius.

2.2 Centroid determination

We must find the best centroid for our automated circle. This centroid needn't to be equal to the geometrical center of the image, although they should be close. We can suppose that because of the medical acquisition procedures, Physicians try to center the images though it is a difficult process, not always reliable.

The centroid determination is performed by the following algorithm (Pseudocode):

```
Begin
   For (every slice s)
      For (every edge e)

         // Find its probability to be circular which means to
         // count the number of voxels equidistant to the edge centroid

         e.centroid = calculateCentroid(e);
         e.probability = countEquidistant(e);

         // Find the bigger edge, in terms of number of voxels
         // where its probability is greater than a user-given threshold
```

```
            If (theBiggestEdge.numberOfVoxels <= e.numberOfVoxels)
            And (e.probability > threshold) Then theBiggestEdge = e;

            // Weigh up its centroid
            // Use a factor taken from a Gaussian function
            // The factor should penalize being far from the center
            // Add this weighted centroid to the average

            factor = gaussian(e.centroid, geometrical_center);
            e.centroid = e.centroid * factor;
            globalCentroid = globalCentroid + e.centroid;
            globalFactor = globalFactor + factor;

         End_For
      End_For

      globalCentroid = globalCentroid / globalFactor;
End_Begin
```

More details are depicted in figure 3:

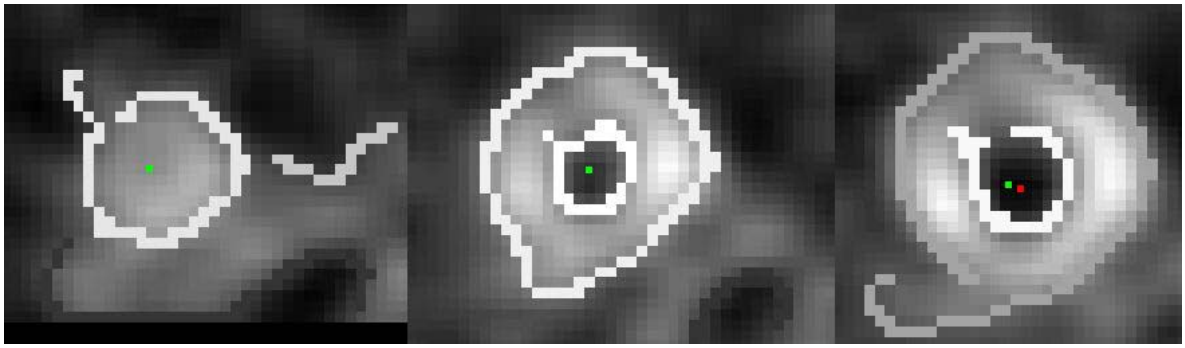

Figure 3.Left & middle: Local centroids for those slices.
Right: Local (green) and global (red) centroids.

Figure 3 shows how every slice contributes with its local centroid to the global, and final, centroid. The global centroid doesn't have to coincide with the geometrical center of the image:

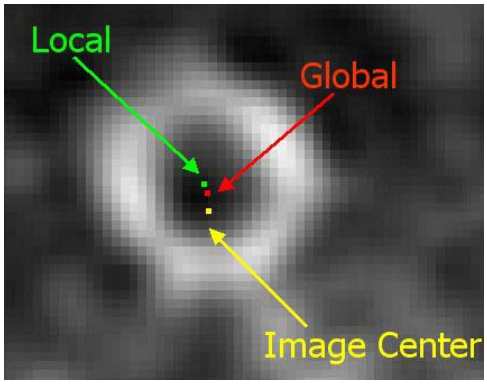

Figure 4.Local centroid for the given slice, global centroid for all the slices and geometrical center of the image.

Figure 4 shows clearly that the centroids are close but not equal. All the local centroids are averaged and contribute to the global one, normally very close to the image center.

2.3 Radius determination

We begin by finding a new set of probabilities. In this case the probabilities of all the edges to be a part of an imaginary circle centered at the global centroid, found in the previous section. Then we run the following algorithm (Pseudocode):

```
Begin
   For (every slice s)
      For (every edge e)

         // Check if its probability is greater
         // than a threshold

         If (e.probability > threshold) then
         Begin
             // Calculate its average distance to the new centroid

             e.distance = averageDistance(e, globalCentroid);

             // Find a Gaussian factor that
             // penalizes the distance to the
             // image center and apply a hard restriction:
             // it must be greater that a threshold

             e.centroid = calculateCentroid(e);
             factor = gaussian(e.centroid, geometrical_center);

             // Check if that distance is greater than
             // the last saved and reject/save it accordingly

             If (factor > threshold)
             And (theSavedDistance < e.distance) Then
             theSavedDistance = e.distance
         End_Begin
      End_For
   End_For

   // Define the radius as the major distance plus a user-given offset

   newRadius = theSavedDistance + offset;
End_Begin
```

After that, we can draw the new circle by using the global centroid and the major distance as the radius, as shown in figure 5:

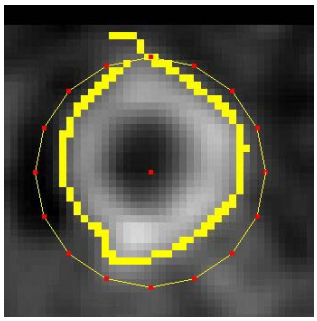

Figure 5.The algorithm finds the "best" edge in terms of distance to the global centroid. The radius is set to be this distance plus a user-given offset.

The process has automated the circle setting. Moreover, the new circle is absolutely well fit to the data, avoiding most of the noise as it is shown in figure 6.

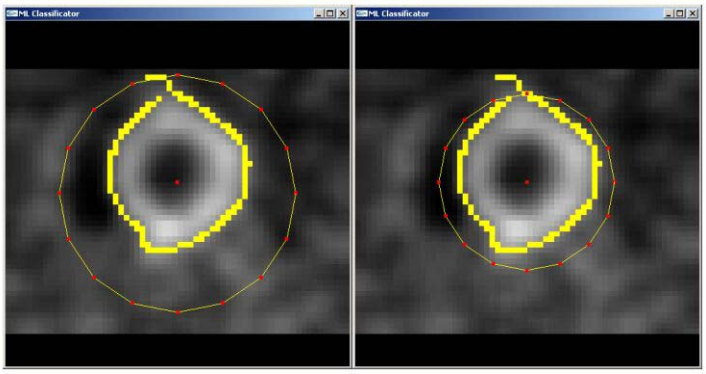

Figure 6.Left: The first, coarse, circle. Right: The second, automated and well fit, circle.

2.4 Finding the division slice

The division slice downwards determines the passage from two (external and internal) to one (external) surface. It is located at the end of the endocardium (inner surface). Therefore this can be considered an anatomical constraint characteristic from the left ventricle, which is the actual data to be recovered.

This constraint is an important feature because it limits the later processes. In order to find the associated slice, we can resort to the fact that the property value (blood irrigation) in the global centroid, raises as we go from top to bottom, as depicted in figure 7.

The property is normalized for all the voxels along the data set. This means that the intensity shown in figure 7 stands for percentage relative to the maximum. Therefore taking a look at different normalized data sets, gives a definitive clue about typical values for the property differences around the division slice.

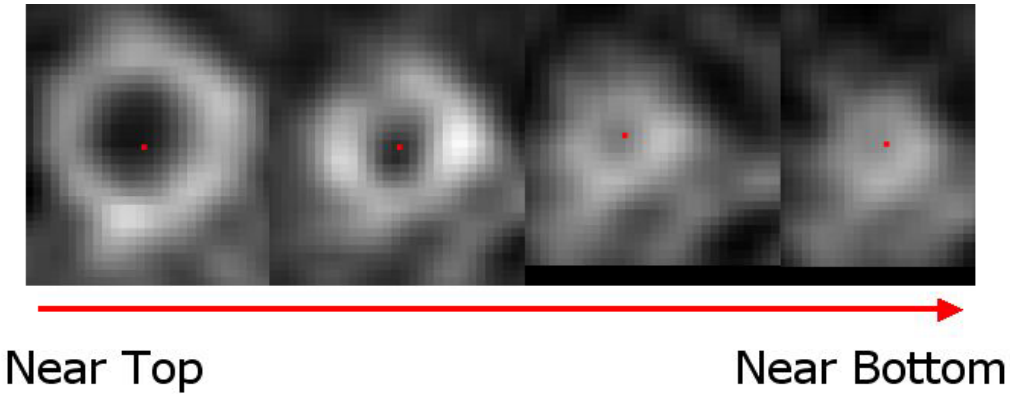

Figure 7.The property at the global centroid grows up (black to white).
Note that the division slice should be between the second and the third image.

Our implementation traverses the slices from top to bottom until the division slice is found. The

property difference is calculated as we move until it gets expectedly big. This high gradient slice determines the endocardium bottom and so, the division slice.

2.5 Labeling the edges

We are now in the last stage of the border classification. Before the definitive labeling, the edges must be treated as follows:

- Use the automatically derived circle to cut off the edges.
- Label the remaining edges with different id's.
- Delete spurious edges.
- Now we can use the division slice to design a two-stage algorithm:

```
Begin
   For (every slice s)
      If (s < divisionSlice) Then

      // Edges under the division slice are marked as external borders

      Begin
        For (every edge e)
            e.externalBorder = True;
        End_For
      End_Begin

      Else
      Begin

        // Calculate their distances to the global centroid

        For (every edge e)
            e.distance = evaluateDistance(e, globalCentroid);
            add(e.distance, arrayDistances);
        End_For

        // Order those distances in a vector
        // Build a new vector containing
        // the differences between consecutive distances

        Order(arrayDistances);
        BuildDifferences(arrayDistances, arrayDifferences);

        // The biggest difference indicates
        // the frontier between internal and
        // external edges, in terms of id's

        frontier = getTheMaximumValueOf(arrayDifferences);

        // label the edges as external or internal borders

        For (every edge e)
            position = GetPosInTheArray(e.id);
            If (position > frontier) Then e.externalBorder = True;
            Else e.internalBorder = True;
        End_For

      End_Begin
   End_For
End_Begin
```

See section 4 for examples on labeling.

3 4D Reconstruction

Time is introduced in the 3D reconstruction process as a fourth dimension to take in consideration. The complete process is presented in figure 8:

Figure 8. The complete reconstruction process, step by step.

The first block is a translation module that converts the DICOM file format (*Digital Imaging and Communications in Medicine*) [7] to our internal voxel implementation. The DICOM file format is widely used by medical equipment, especially by imaging systems like the one used for our tests.

The second and third modules have been extensively documented in the previous sections. Therefore this section addresses the problematic due to the final reconstruction and 4D interpolation of the final meshes.

More information on these can be found at [4,5].

3.1 Vector Field

The vector field is obtained as a Gradient Vector Flow [8,9] derived from the property gradients. In that case, the property value is selected to be the border attribute. A voxel can be assigned one of two possible values: "1" if it is a border or "0" if it is not.

The vector field is calculated twice, for the internal and external borders. The result of the field calculation is the apparition of a vector set that surrounds the borders, making them act as attractors.

The field acts as an external force. This force consists on the minimization of a functional that mixes the information derived from the desired feature gradient with a diffusion term that allows the vector field to be spread out [4,5]. Figure 9 resumes this behavior:

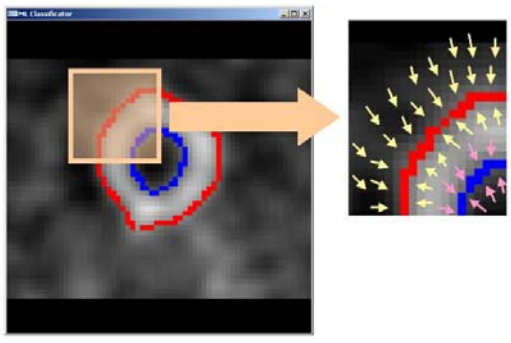

Figure 9. Graphical representation of the behavior for the GVF field.

Note that figure 9 represents the field in 2D. Our implementation finds it in a 3D framework (voxels not pixels) making it more robust and confident with our 3D data set.

3.2 Dual reconstruction process

As with the vector field, the reconstruction process must be calculated twice. The reconstruction is planned as an evolution scheme based on Newtonian dynamics. The model is built from triangles made of particles (vertices) that move due to internal and external forces [4,5]. Figure 10 shows the Newtonian dynamics law:

$$F_i = m_i \cdot \ddot{x}_i \Rightarrow \begin{cases} \dot{x} = v \\ \dot{v} = F_i / m_i \end{cases}$$

Figure 10. Newtonian dynamics law.

Where F_i stands for the external force (check section 3.1), m_i for the mass, x_i for the position, and v_i for the velocity, for the i_{th} particle. The system iterates consecutively until it gets a position of equilibrium in a high percentage of the particles, as figure 11 shows:

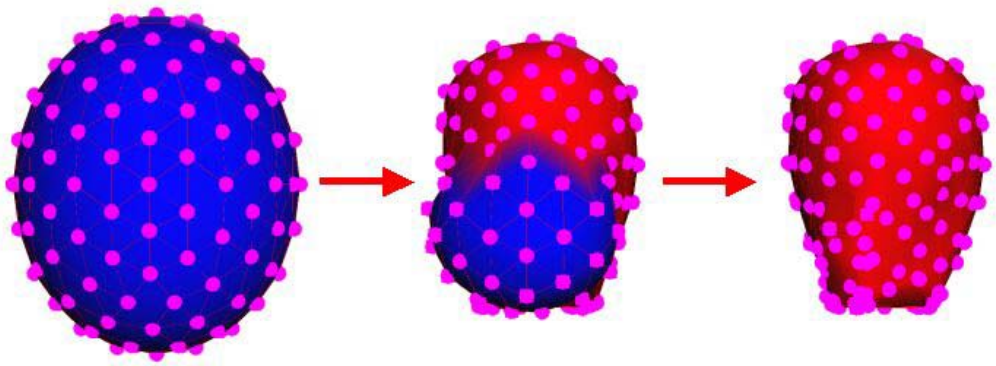

Figure 11. Particles reach their position of equilibrium after several iterations.

In order to solve the system, we need a numerical method. For our tests, we apply an explicit method, a Runge-Kutta 4 solver. It supports both reliable and accurate results because of its low-order error term ($O(h^5)$ where h is the time increase at each iteration). For the simulation shown in figure 11, h was selected to be 0.1 seconds.

Reports on these results can be found at [4,5].

3.3 The smoothing algorithm

Using low-resolution data induces the final result to be poorly detailed. This is mainly caused by the discreet nature of the original data. After the dynamic reconstruction we might need to smooth the final meshes. That's the purpose of the smoothing algorithm. The algorithm is geometrically based which means that it can be always applied with no information from the previous process at all.

The main reasons for smoothing the final meshes are:

- SPECT data usually lacks resolution. Moreover that, for a given slice, the region of interest (ROI) is located on a very small area at the center.
- The distance between slices is about half centimeter depth, a very coarse measure.
- We might lack data voxels due to failures in the reception. Due to the internal basis of SPECT imagery, isquemic areas, which are characterized by their absence of blood irrigation, won't be shown in the images. From a practical point of view, this fact will provoke "holes" to appear in the data. During the reconstruction process, the active mesh might try to enter them.
- Due to the compromise between accuracy and speed, our reconstruction method does not apply internal forces, which means that there's no restriction on curvature or local smoothness. This fact can be partially corrected by the smoothing algorithm.

The algorithm penalizes those vertices that belong to a non-uniform set of triangles. It applies small changes in position to the selected vertices and it never changes the mesh topology.

In order to ensure the correct functionality of the algorithm, the edges must be classified in either CW (Clockwise) or CCW (Counter-clockwise) order. Mesh generation applications doesn't have to ensure this fact so we have to take care of it.

Given a certain neighborhood formed from three triangles, we could talk about two cases: a first case where the sum of the three inner angles is near 180 degrees (half circle), that denotes that the reconstructed surface doesn't present important rugosities; A second case where the sum of the angles is far from the theoretic 180 degrees. When the sum is really tiny we have just found a peak that has to be smoothed.

Smoothing the peak can be released by applying a geometric constraint to the common vertex so that it moves to the centroid of their neighbors. Someone might think that it could be enough to swap the vertex and centroid positions, instead of performing all the displacement calculations. This is not true. The centroid serves as a flag that shows the true direction to take but it is not necessarily the best position to be in. It might be too far, for instance. We want the smoothing algorithm to do its job while applying small changes if possible.

More information about the algorithm and its coding can be found at [10].

3.4 4D Interpolation

Time is introduced in the system when we are dealing with more than a reception of the cardiac cycle for our patient. This is the case of figure 15, for instance.

In those cases, all the process (see figure 8) is repeated for every reception. After that, the evolution through time is implemented as a linear interpolation function between meshes, based on key framing (see figure 12).

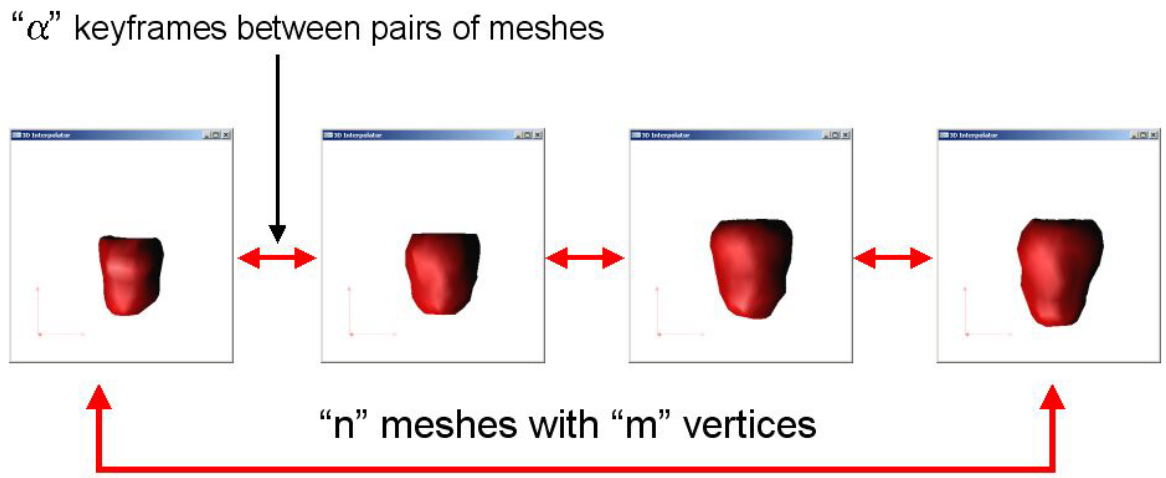

Figure 12.Keyframing introduces the time as the fourth dimension.

For a given vertex, its successive positions can be calculated by linear interpolation as stated in figure 13:

$$\bar{v} = P - Q$$
$$p(\alpha) = Q + \alpha\bar{v}$$
$$\alpha \in [0,1]$$

Figure 13.Linear interpolation between positions P and Q.

Where $p(\alpha)$ gives all the intermediate positions when going from one mesh to the next. The position is adjusted with the α parameter. Values for α range from 0 ($p(\alpha) = Q$) to 1 ($p(\alpha) = P$). The α parameter can be tuned depending on the number of key frames desired between the original meshes. Typical values for our application range from 20 to 30 key frames between successive instants of time (1/20 to 1/30 values for the α parameter).

More information about this module can be found at [4,5].

4 Results

Besides all the results considered through the paper, we also present the segmentation algorithm over a Phantom data set. Details on the characteristics of this model can be sought at [5]. As figure 14 shows:

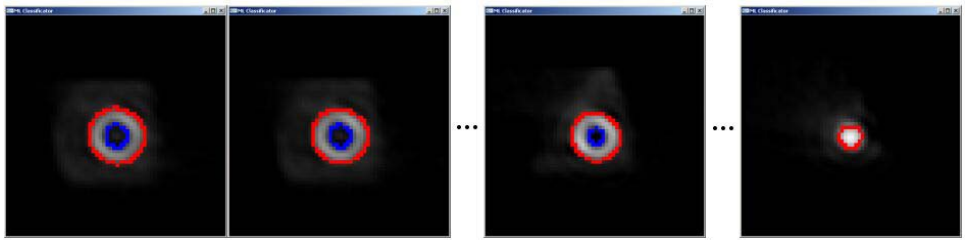

Figure 14. The borders detected over the Phantom data set.

The borders have been correctly detected and labeled as internal (blue) and external (red).

For a real case, an example of the final labeling for a given slice of a complete cardiac cycle, can be observed at figure 15:

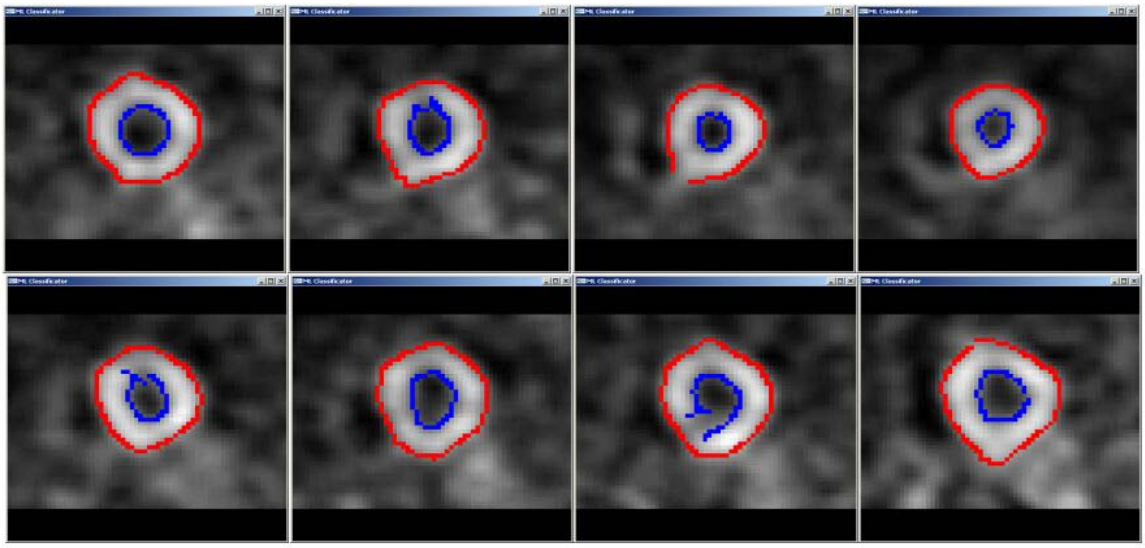

Figure 15. Slice 15 of a complete cardiac cycle data set.
The acquisition consisted on 8 captures ranging from systole to diastole.
Red stands for external (epicardium) and blue for internal (endocardium) surfaces.

For the complete cardiac cycle at figure 15, it must be said that it consists on 16 meshes because of the eight instants of time with two meshes each, internal and external.

In terms of computation time, typical durations for the MLC Filtering process are about half a second long (1,484 seconds for every static instant of figure 15, 23,744 seconds for the 16 meshes). Every instant consists on 94208 voxels (64 x 64 x 23).

The test was held in a Pentium 4 1.7 GHz, 512 MB RAM, with a GeForce 4 Ti 4200 128 MB accelerator card.

5 Conclusions

We have presented a new and robust approach in order to perform the 3D segmentation of the human left ventricle. Our algorithm is specifically designed for SPECT data. In this edition, we have focused our efforts in the automation of the whole process, from the 2D acquisition and filtering, to the interpolation over time. Nevertheless there is still some work to do in terms of adding a new feature, the control of vertical coherence between slices, a factor that will improve the inputs with information on the 3D shape.

The authors wish to thank all the members of the medical applications research group at the IRI-UPC, the Nuclear Cardiology team at Vall Hebrón Hospitals and the people integrating the graphics & VR group at the Multimedia Section of the DTA, at La Salle School of Engineering (Ramon Llull

University).

This work has been partially financed by the TIC2000-1009 project. The first author is granted by an EPSON “Rosina Ribalta” prize (1999).

References

[1] J. Canny. “A Computational Approach to Edge Detection”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, Nov. 1986.

[2] M. Ruzon and C. Tomasi, "Color Edge Detection with the Compass Operator," In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Ft. Collins, CO, V. 2, pp. 160-166, June 1999.

[3] F. Evans, "An investigation into the use of maximum likelihood classifiers, decision trees, neural networks and conditional probabilistic networks for mapping and predicting salinity", Copyright 1997-2003, CSIRO Australia. Available at: “`http://www.cmis.csiro.au/Fiona.Evans/personal/msc/html/toc.html`”

[4] O. García, A. Susín. "Surface and Volume Reconstruction of the Left Ventricle from SPECT data". 1st Ibero-American Symposium in Computer Graphics SIACG 2002, July 2-5. Centro de Computação Gráfica, 4800-073, Guimarães, Portugal.

[5] O. García, A. Susín. "Left Ventricle Volume Values Estimation From 3D SPECT Reconstruction". The 29th Annual Conference of Computers in Cardiology. Memphis, Tennessee, September 22-25, 2002.

[6] Finn Mannting, Puneet K Chandak, Yanina V Zabrodina, B Leonard Holman, “Atlas of Myocardial Perfusion SPECT”, Brigham and Women’s Hospital, Harvard Medical School, Boston MA. 1995-1999. On-line atlas available at: “`http://brighamrad.harvard.edu/education/online/Cardiac/Cardiac.html`”

[7] "The DICOM Standard", School of Psychology, University of Nottingham, University Park, Nottingham, NG7 2RD, UK 2003. Available at: **“**`http://www.psychology.nottingham.ac.uk/staff/cr1/dicom.html`**”**

[8] C. Xu and J.L. Prince, “Gradient Vector Flow: A New External Force for Snakes”, IEEE Proc. CVPR, 1997.

[9] C. Xu and J.L. Prince, “Snakes, Shapes, and Gradient Vector Flow”, IEEE Transactions on Image Processing, pag. 359-369, 1998.

[10] J. Lander et al., "Graphics Programming Methods", Charles River Media, ISBN: 1-58450-299-1, 2003 (to be published in the second quarter of 2003).