

## **TESI DOCTORAL**

**Títol:** **General Dynamic Surface Reconstruction:  
Application to the 3D Segmentation of the  
Left Ventricle**

**Realitzada per** **Oscar García Pañella**

**En el Centre** **Escola Tècnica Superior d'Enginyeria  
Electrònica i Informàtica La Salle**

**I en el Departament** **Tecnologies Audiovisuais**

**Dirigida per** **Dr. Antoni Susín Sánchez**

**I tutoritzada per** **Dr. Gabriel Fernandez**

## **Abstract**

This thesis describes a contribution to the three-dimensional reconstruction of the internal and external surfaces of the human's left ventricle. The reconstruction is a first process fitting in a complete VR application that will serve as an important diagnosis tool for hospitals. Beginning with the surfaces reconstruction, the application will provide volume and interactive real-time manipulation to the model. We focus on speed, precision and smoothness for the final surfaces. As long as heart diseases diagnosis requires experience, time and professional knowledge, simulation is a key-process that enlarges efficiency.

The algorithms and implementations have been applied to both synthetic and real datasets with differences regarding missing data, present in cases where pathologies and abnormalities arise. The datasets include single acquisitions and complete cardiac cycles. The goodness of the reconstructions has been evaluated with medical parameters in order to compare our results with those retrieved by typical software used by physicians.

Besides the direct application to medicine diagnosis, our methodology is suitable for generic reconstructions in the field of computer graphics. Our reconstructions can serve for getting 3D models at low cost, in terms of manual interaction and CPU computation overhead. Furthermore, our method is a robust tessellation algorithm that builds surfaces from clouds of points that can be retrieved from laser scanners or magnetic sensors, among other available hardware.

The author is granted by an EPSON "Rosina Ribalta" award. This work has been also financed by the TIC2000-1009 project.

## Acknowledgements

There is a huge list of people that has contributed to this research work, either in one way or another. I hope to mention all of them and I apologize from beforehand if I miss any of the names.

I wish to thank all the members of the medical applications research group at the IRI-UPC, the Nuclear Cardiology team at Vall d'Hebrón Hospitals and the people integrating La Salle School of Engineering (Ramon Llull University).

It has been a pleasure to work with such a good professionals like Pere Brunet, Alvar Vinacua, Isabel Navazo, Jordi Regincós, Josep Oriol Esteve, Eva Monclús, Daniel Marín, Lyudmila Rodriguez, Máximo Mero and the rest of the crew at the Technical University of Catalonia.

Doctors Santiago Agudé and Jaume Candell provided us with the datasets that we used for all the experiments and no testing would have been possible without their help.

My colleagues at La Salle School of Engineering have been giving me continuous support. Everyone encouraged me when needed. I'd like to mention Elisa Martinez whose help regarding the image processing area has been basic for me; Josep M<sup>a</sup> Garrell and brother Miquel Angel Barrabeig for the help on the formal aspects of the thesis; Elisabet Golobardes, Xavier Vilasis, and Anna Barjau for their monitoring in the PhD courses; Gabriel Fernandez and Antonio Pastor for letting me spend hours and hours writing this document; Josep M<sup>a</sup> Duque for the advises on the *VRML* standard; brothers Josep Martí and Daniel Cabedo for their interest in my progress; brother Carles Giol who printed so many publications for me; August Climent for his help when publishing in the *Input* magazine and of course all my companions at the department: Diana López, Claudia Dalmau, Daniel Barrabino, Alex Arañó, Sergi Villagrasa, Tomàs Cayuela, Jesús Liarte, Marc Llibre, Jaume Duran, Nacho Aso, Vicenç Garcia, Richard H. Cuello, David Fonseca, Eloi Castells, Manuel Lucas, Antoni Cabello, Oscar Fernandez, Albert Garcia, Xavier Ginesta, Lambert Malivern, Sergi Salas, Avelino Gonzalez and the rest of the people integrating the technical section. Thanks also to Josep Piqué, Juan Carlos Olabe and brothers Francisco V. Martin and Louis Althaus, among the rest, for the wonderful reception at CBU (*Christian Brothers University*) in Memphis.

I can't forget my colleague of promotion Josep Lluís Anguera, who has contributed with the testing and programming of the *discreet contour deformation model*; Mark Ruzon for his code for the *Compass operator*; David Baraff for the slides; Ken Hoge of the Texas Heart Institute for letting me use their images; Jeff Lander for publishing the smoothing algorithm in his book; Petia Radeva and Debora Gil for testing their *ACC* paradigm onto our images; Jordi Amatller for the cloth modeling framework that we started with; Ken Tabb for the *snake* code; Gina Deibel for the images of the atlas; Jerry L. Prince for the *GVF* vector field; Mike Heath for the *Canny Edge Detector* code; Dr. Ulrich Neumann and the people of the IMSC at USC (University of Southern California) at Los Angeles, for the six months that I spent with them.

Very special words for Ramón Ollé, Epon Iberica's CEO, whose award has supported some of the expenses of my research.

I am absolutely grateful to my advisor during all this adventure, Dr. Antoni Susín Sánchez. His optimism during all those years made me overcome the problems and his strong knowledge on the area has possibilited all the material within this document. Nothing would have been possible without him.

As a final statement, thanks to my wonderful parents Montse and Jesús, to Chelo, to my beautiful sisters Mariona and Laura, to my little dog Dark and to the woman that changed my life several years ago, the woman that I will respect and admire for the rest of my life, my precious Eva.



# Index

<b>List of tables</b> .....	<b>vi</b>
<b>List of figures</b> .....	<b>viii</b>
<b>1 Introduction</b> .....	<b>1</b>
<b>1.1 Objectives</b> .....	<b>1</b>
<b>1.2 State of the art</b> .....	<b>2</b>
1.2.1 Deformable models .....	2
1.2.2 Models in cardiac image analysis .....	8
1.2.3 Soft tissue modeling applied to other contexts .....	13
<b>1.3 Structure of this document</b> .....	<b>15</b>
<b>2 2D Processing</b> .....	<b>16</b>
<b>2.1 Manual vs. automatic</b> .....	<b>16</b>
<b>2.2 Tested operators</b> .....	<b>17</b>
2.2.1 Robert operator .....	17
2.2.2 Canny operator .....	19
2.2.3 Compass operator .....	20
2.2.4 Sobel operator .....	21
2.2.5 Comparative between operators .....	22
<b>2.3 Border labeling</b> .....	<b>28</b>
2.3.1 Can we avoid the labeling?.....	28
2.3.2 Labeling by the vector field sign .....	29
2.3.3 Labeling by the gradient.....	30
2.3.4 Case-based classification.....	30
2.3.5 Radial-circumferences-based algorithm .....	32

2.3.6	MLC filtering.....	34
<b>2.4</b>	<b>Search for the automatic circle .....</b>	<b>37</b>
2.4.1	Centroid determination.....	39
2.4.2	Radius determination.....	40
<b>2.5</b>	<b>Finding and refining the division slice.....</b>	<b>41</b>
<b>2.6</b>	<b>Bounding Box based filtering .....</b>	<b>44</b>
<b>2.7</b>	<b>Summary .....</b>	<b>45</b>
<b>3</b>	<b>3D Deformation models .....</b>	<b>47</b>
<b>3.1</b>	<b>Newtonian dynamics .....</b>	<b>47</b>
3.1.1	Particle systems .....	48
3.1.2	Phase Space .....	49
<b>3.2</b>	<b>Internal forces.....</b>	<b>50</b>
3.2.1	Stretch force.....	51
3.2.2	Shear force.....	52
3.2.3	Bend force .....	52
3.2.4	Local curvature force.....	53
<b>3.3</b>	<b>External forces.....</b>	<b>58</b>
3.3.1	Snakes.....	58
3.3.2	Active nets and Topologic active nets.....	59
3.3.3	Radial energy based potential.....	62
3.3.4	Gradient vector flow .....	65
<b>3.4</b>	<b>Deformable models.....</b>	<b>68</b>
3.4.1	Discreet contour deformation model.....	69
3.4.2	Plain deformation model .....	71
3.4.3	Spring-mass deformation model.....	72

3.4.4	Restricted spring-mass deformation model .....	73
3.4.5	Free deformation model .....	74
<b>3.5</b>	<b>Summary .....</b>	<b>74</b>
<b>4</b>	<b>Numerical implementation.....</b>	<b>76</b>
<b>4.1</b>	<b>ODE based methods .....</b>	<b>76</b>
<b>4.2</b>	<b>Explicit methods .....</b>	<b>79</b>
4.2.1	Euler’s method.....	79
4.2.2	The midpoint method .....	81
4.2.3	The Runge-Kutta 4 method .....	83
4.2.4	Adaptative stepsize .....	85
<b>4.3</b>	<b>Implicit methods .....</b>	<b>87</b>
<b>4.4</b>	<b>Implementations of the deformable models.....</b>	<b>88</b>
4.4.1	Implementation of the plain deformable model .....	89
4.4.2	Implementations of the spring-mass, restricted spring-mass and free deformable models .....	90
4.4.3	Comparative between numerical implementations.....	90
<b>4.5</b>	<b>Implementation of the external force .....</b>	<b>95</b>
<b>4.6</b>	<b>Summary .....</b>	<b>97</b>
<b>5</b>	<b>Model geometry .....</b>	<b>99</b>
<b>5.1</b>	<b>Classification of geometrical models.....</b>	<b>99</b>
<b>5.2</b>	<b>Quality of the triangulated mesh .....</b>	<b>102</b>
<b>5.3</b>	<b>In-to-Out vs. Out-to-In reconstruction.....</b>	<b>106</b>
<b>5.4</b>	<b>Our model and the Marching Cubes algorithm .....</b>	<b>108</b>
<b>5.5</b>	<b>The smoothing algorithm.....</b>	<b>111</b>
5.5.1	About the input mesh.....	112

5.5.2	Motivation .....	112
5.5.3	A prior question: the edge arrangement .....	114
5.5.4	The Algorithm .....	115
5.5.5	Computing the angle and the projections .....	116
5.5.6	The displacement vector .....	117
5.5.7	Conclusion .....	117
<b>5.6</b>	<b>Volume specified by a surface mesh .....</b>	<b>118</b>
<b>5.7</b>	<b>4D Interpolation .....</b>	<b>120</b>
<b>5.8</b>	<b>Summary .....</b>	<b>121</b>
<b>6</b>	<b>Applications and results .....</b>	<b>123</b>
<b>6.1</b>	<b>Human's Left Ventricle reconstruction .....</b>	<b>124</b>
6.1.1	First steps .....	124
6.1.2	Evolution to the final solution .....	128
6.1.2.1.	Test model: the Phantom Volume .....	128
6.1.2.2.	GVF parameters for the Phantom volume .....	131
6.1.2.3.	The stopping mechanism .....	134
6.1.2.4.	Distance-to-data measures .....	136
6.1.2.5.	Volume results .....	142
6.1.2.6.	CPU time measures .....	145
6.1.3	Missing-data results .....	148
6.1.4	Complete cardiac cycle .....	151
6.1.5	Pathological cases .....	153
6.1.6	Other explored approaches .....	155
6.1.6.1.	Automatic contouring and tessellation of the 3D shape .....	155
6.1.6.2.	Flexible volumetric model of the left ventricle .....	158

6.1.6.3. Anisotropic Contour Completion applied to left ventricle imagery .....	159
<b>6.2 Generic reconstructions .....</b>	<b>163</b>
6.2.1 Low-polygon modeling and mipmeshing.....	163
6.2.2 Automatic tessellation .....	164
6.2.3 Results on automatic meshing.....	166
<b>6.3 Summary .....</b>	<b>175</b>
<b>7 Conclusions .....</b>	<b>176</b>
7.1 General structure of the system .....	176
7.2 2D Processing.....	179
7.3 3D Deformation models .....	180
7.4 Numerical implementation .....	182
7.5 Model geometry .....	184
7.6 Applications and results.....	186
7.7 Publications.....	188
<b>8 Future Work.....</b>	<b>189</b>
<b>9 Bibliography .....</b>	<b>191</b>
<b>Appendixes .....</b>	<b>200</b>
<b>A SPECT imagery .....</b>	<b>200</b>
<b>B The left ventricle .....</b>	<b>203</b>
<b>C Quantitative parameters of the left ventricle .....</b>	<b>206</b>
<b>D The Phantom volume .....</b>	<b>208</b>
<b>E The DICOM file format .....</b>	<b>211</b>

## List of tables

Table 2.1. Cases to take into consideration when refining the division slice.....	42
Table 4.1: Comparison between numerical schemes.....	91
Table 4.2: First comparison of the explicit methods.....	94
Table 4.3: Second comparison of explicit methods.....	95
Table 5.1: Classification of geometrical models.....	100
Table 5.2: Initial triangulated meshes.....	103
Table 5.3: Quality of the triangles after the simulation.....	104
Table 5.4: Final quality of the triangles attending to the reconstruction method.....	105
Table 5.5: Artifacts in internal reconstructions.....	107
Table 5.6: Comparison of results attending to the initial mesh (Scaled Sphere vs. Marching Cubes).....	110
Table 6.1: GVF vector field study for the Phantom volume (I).....	131
Table 6.2: GVF vector field study for the Phantom volume (II).....	133
Table 6.3: The GVF vector field parameters study for the Mayo Phantom volume....	133
Table 6.4: Simulation parameters associated to the results shown in figure 6.10.....	135
Table 6.5: Simulation parameters associated to the measures in distance to data. For those simulations $\Delta t = 0.00025$ s., iterations = 2000. (Spring-mass deformation model).....	136
Table 6.6: Simulation parameters associated to the measures in distance to data. (Free deformation model).....	137
Table 6.7: Mesh rendering and simulation parameters associated to several measures in distance to data.....	141
Table 6.8: Volume comparison according to the integration method.....	143
Table 6.9: Volume comparison in the Phantom Mayo dataset.....	144
Table 6.10: Volume comparison in internal reconstructions.....	146
Table 6.11: CPU times for the GVF computation on two different Phantom datasets	147
Table 6.12: CPU times and relative errors depending on the number of iterations.....	147
Table 6.13: CPU times associated to the latency of the whole pipeline.....	148
Table 6.14: Recovered surfaces from partial data.....	150
Table 6.15: Complete cardiac cycle with external surfaces (top) and internal surfaces (bottom).....	152
Table 6.16: Obtained volumes for the eight temporal instances.....	152
Table 6.17: Volumes for the eight temporal instances, applying the discreet contour deformation model.....	158
Table 6.18: Characteristics for the general 3D reconstructions.....	167

Table 6.19: General reconstructions. Test 1, the Venus model.....	169
Table 6.20: General reconstructions. Test 2, the rabbit model.....	170
Table 6.21: General reconstructions. Test 3, the screwdriver model .....	171
Table 6.22: General reconstructions. Test 4, the vase model .....	172
Table 6.23: General reconstructions. Test 5, the Moai model.....	173
Table 6.24: General reconstructions. Test 6, the squirrel model.....	174

## List of figures

Figure 2.1: Results of the Robert operator in medical images .....	18
Figure 2.2: Two resolutions for the same data slices .....	18
Figure 2.3: Results of the Robert operator in medical images .....	19
Figure 2.4: Smoothing masks for doubling the resolution .....	19
Figure 2.5: Left, Original SPECT image. Middle, Robert operator filtered image. Right, Canny operator filtered image .....	20
Figure 2.6: The Sobel operator mask.....	21
Figure 2.7: Convolution mask for $S_x$ and $S_y$ .....	21
Figure 2.8: Filtered medical images. Top, Sobel operator, middle and bottom, Canny with different thresholds.....	22
Figure 2.9: First test and first sequence comparing the Canny and Compass operators	24
Figure 2.10: First test and second sequence comparing the Canny and Compass operators .....	25
Figure 2.11: Second test and first sequence comparing the Canny and Compass operators .....	26
Figure 2.12: Second test and second sequence comparing the Canny and Compass operators .....	27
Figure 2.13: Evaluation of the vector field directly from the image.....	28
Figure 2.14: The sign of the vector field when following a horizontal (left) and vertical (right) trajectory in a data slice.....	29
Figure 2.15: Voxels marked by sign (left) and gradient (right) .....	30
Figure 2.16: Case-based classification of the borders .....	31
Figure 2.17: An actual patient's ischemic left ventricle.....	32
Figure 2.18: The radial-circumferences-based algorithm put to the test.....	33
Figure 2.19: Borders labeled by the radial-circumferences-based algorithm.....	34
Figure 2.20: From left to right: a left ventricle data slice; 2D identified borders; borders in 3D space that will be used for getting the 3D vector field .....	35
Figure 2.21: The borders detected over the Phantom data set (blue for internal and red for external) .....	36
Figure 2.22: Slice 15 of a complete cardiac cycle data set.....	36
Figure 2.23: Filtering by the property value.....	37
Figure 2.24: The filtering circle avoids noise plus an incorrect labeling of the borders	37
Figure 2.25: Using the circle for the property cutting off .....	38
Figure 2.26: Cutting off and smoothing the data.....	38
Figure 2.27: Edges detected by the generic segmentation algorithm .....	39



Figure 2.28: Local centroids for those slices (left & middle); Local (green) and global (red) centroids (right) .....	39
Figure 2.29: Local centroid for the given slice, global centroid for all the slices and geometrical center of the image.....	40
Figure 2.30: The algorithm finds the “best” edge in terms of distance to the global centroid.....	40
Figure 2.31: The first coarse circle (left); the automated and well fit circle (right).....	41
Figure 2.32: The property at the global centroid grows up (black to white).....	41
Figure 2.33: The division slice refinement process.....	42
Figure 2.34: Case 7 of the division slice refinement process .....	43
Figure 2.35: Case 1 of the division slice refinement process .....	44
Figure 2.36: Filtering with bounding boxes .....	45
Figure 3.1: Particles reach their position of equilibrium after several iterations .....	48
Figure 3.2: Particle attributes .....	48
Figure 3.3: A complete particle system.....	49
Figure 3.4: A deformable cube.....	50
Figure 3.5: Local coordinates for a triangle .....	51
Figure 3.6: Internal shear force.....	52
Figure 3.7: Internal bending force .....	53
Figure 3.8: Local curvature forces.....	53
Figure 3.9: Local curvature $c_i$ .....	54
Figure 3.10: Tangential and Radial unit vectors $tu_i$ and $ru_i$ .....	54
Figure 3.11: Positive and negative local curvatures .....	55
Figure 3.12: Curvature vectors $c_i$ for some typical situations .....	56
Figure 3.13: Internal forces $f_{in,i}$ .....	57
Figure 3.14: Breaking links and defining new external nodes .....	61
Figure 3.15: Components of the external force over the contour.....	63
Figure 3.16: Clustering phenomena .....	63
Figure 3.17: Radial component of the external force .....	64
Figure 3.18: Graphical representation of the behavior for the GVF field .....	66
Figure 3.19: Correct (left) and wrong (right) GVF vector fields.....	66
Figure 3.20: Different operators and signs for the GVF vector field evaluation.....	67
Figure 3.21: Two resolutions for the same contour.....	70
Figure 3.22: Resampling method .....	70
Figure 3.23: Two views of the damping map for a left ventricle medical dataset .....	72
Figure 3.24: A damped spring connects two particles .....	72

Figure 3.25: Restricted spring .....	73
Figure 3.26: Evaluating the orthogonal-to-the-gradient-vector component of the spring force .....	73
Figure 4.1: The initial value problem .....	77
Figure 4.2: The solver interface .....	77
Figure 4.3: Euler’s method .....	79
Figure 4.4: Euler’s inaccuracies and instabilities .....	80
Figure 4.5: Euler’s accuracy depends strongly on the stepsize .....	80
Figure 4.6: The Midpoint Method .....	82
Figure 4.7: Values of the derivative in the midpoint method .....	83
Figure 4.8: Slope evaluations in the Runge-Kutta 4 method .....	83
Figure 4.9: The slope is evaluated four times in the Runge-Kutta 4 method .....	84
Figure 4.10: Adaptive stepsize in a simulation framework .....	85
Figure 4.11: Final distances from the particles to the data to recover .....	92
Figure 4.12: Evolution of the distances to the data to recover .....	92
Figure 4.13: Degenerations on the final meshes due to the resolution increase .....	94
Figure 4.14: Wrong (left) and correct (right) results for the GVF external force .....	97
Figure 5.1: C structure for a triangulated mesh .....	101
Figure 5.2: Top view of the poorly results due to failures in the topology .....	101
Figure 5.3: Measure of the degeneration in triangles .....	102
Figure 5.4: Graphical view of the quality for the final triangles .....	105
Figure 5.5: Internal mesh reconstruction .....	106
Figure 5.6: Three cases of the Marching Squares algorithm .....	108
Figure 5.7: Cross sections of the recovered surfaces for the Phantom volume .....	109
Figure 5.8: Reconstruction of an artificial left ventricle, with absence of data .....	110
Figure 5.9: Graphical evolution of the vertices attending to their distance to real data .....	111
Figure 5.10: Top-Left) Voxelized data to recover. Top-Right) Initial mesh before the reconstruction. Bottom-Left) Illuminated final mesh. Bottom-Right) Wire frame final mesh .....	112
Figure 5.11: Four portions of the data slices corresponding to medical imagery from the left ventricle .....	113
Figure 5.12: Final mesh with (right) and without (left) applied smoothing algorithm .....	113
Figure 5.13: Left) The neighbor projections onto the vertex plane. Top-Right) The director vectors for the plane containing the vertex. Bottom-Right) The $(\lambda, \mu)$ plane coordinates for one of the neighbors .....	115
Figure 5.14: Evaluating the smoothness of a triangle neighborhood .....	115
Figure 5.15: Peak correction by vertex displacement .....	116

Figure 5.16: The smoothing algorithm in the context of terrain generation.....	118
Figure 5.17: Computing the contributions on volume for a surface triangle .....	119
Figure 5.18: Keyframing introduces time as a fourth dimension .....	120
Figure 5.19: Keyframing in a vertex-by-vertex manner.....	121
Figure 6.1: First trials in the reconstruction process .....	124
Figure 6.2: Initial tests regarding the vector field .....	125
Figure 6.3: First reconstructions with the plain deformation model .....	126
Figure 6.4: Vector field associated to real data (two slices).....	126
Figure 6.5: Reconstruction of the external surface in a real dataset.....	128
Figure 6.6: Top: Long-axis view of the Phantom dataset; bottom: the artificial cover	129
Figure 6.7: Reconstruction of the Phantom external surface .....	130
Figure 6.8: Vector field for a gray level dataset.....	131
Figure 6.9: Vector field itinerary for the Mayo Phantom volume.....	134
Figure 6.10: Oscillations around the data borders.....	134
Figure 6.11: Distance results for the plain deformation model.....	138
Figure 6.12: Filtered data of the Phantom dataset.....	142
Figure 6.13: Partially filled Phantom dataset .....	149
Figure 6.14: The Phantom Mayo dataset.....	149
Figure 6.15: Data and GVF field for the 32% loss dataset.....	150
Figure 6.16: Internal and external labeling for the complete (left) and partial (right) datasets .....	151
Figure 6.17: A complete reconstruction process .....	153
Figure 6.18: Perfusion borders (left); Ventriculography borders (middle); Final reconstructed mesh (right).....	153
Figure 6.19: Digestive activity perturbing initial datasets.....	154
Figure 6.20: Occlusions perturbing initial datasets .....	154
Figure 6.21: Template contour for the discreet contour deformation model.....	156
Figure 6.22: Segmentation of a whole dataset by the discreet contour model .....	156
Figure 6.23: 3D recreation of the external surface by the new tessellation method.....	157
Figure 6.24: The cardiac cycle retrieved by the discreet contour deformation model .	157
Figure 6.25: Volumetric model of the left ventricle.....	159
Figure 6.26: ACC as a preprocess for SPECT imagery segmentation .....	160
Figure 6.27: Borders are correctly labeled even if incomplete edges are retrieved .....	161
Figure 6.28: External and internal surfaces for the ACC operator.....	162
Figure 6.29: Mipmeshing .....	164

Figure 6.30: A tessellation from a cloud of points .....	165
Figure 6.31: A real time application linked with a magnetic sensor .....	165
Figure 6.32: A 3D reconstruction evolution graph.....	166
Figure 7.1: General structure of the whole system (first version).....	176
Figure 7.2: General structure of the whole system (second version) .....	177
Figure 7.3: General structure of the whole system applied to generic reconstructions (second version).....	178
Figure 7.4: Automatic contouring and tessellation of the 3D shape with the discreet contour deformation model .....	178
Figure A.1: Acquisition planes in SPECT imagery.....	201
Figure A.2: Images from the horizontal (first and second rows) and vertical (third and fourth rows) planes .....	201
Figure A.3: SPECT images from pathological cases .....	202
Figure B.1: Anatomy of the left ventricle .....	203
Figure B.2: The cardiac cycle into two phases.....	204
Figure B.3: The conduction system.....	205
Figure C.1: The cardiac volume evolution for an actual's patient data.....	206
Figure D.1: The Phantom volume geometry .....	208
Figure D.2: The walls of the Phantom volume.....	209
Figure D.3: Filling of the Phantom volume.....	209
Figure E.1: The DICOM file format structure.....	211

# 1 Introduction

Access to a 3D model obtained from patient's data can have several applications like support on diagnosis, surgery planning, student's training or even remote-operation. A first approximation to the problem would be using a manual process with specific image-processing software [82] though it would require deep medical knowledge and experience.

The aim of this project is to reconstruct the three-dimensional internal and external surfaces of the human's left ventricle. The reconstruction is a first process fitting in a complete VR application that will serve as an important diagnosis tool for hospitals. Beginning with the surfaces reconstruction, the application will provide volume and interactive real-time manipulation to the model. We focus on speed, precision and smoothness for the final surfaces. As long as heart diseases diagnosis requires experience, time and professional knowledge, simulation is a key-process that enlarges efficiency.

## 1.1 Objectives

This thesis is embedded inside a global project where a computerized system is intended to improve the analysis of the cardiac data defined by specialists.

When detecting a cardiac abnormality, there are different informations to take into consideration: morphology, functionality and muscle irrigation plus vascular structure. In order to determine the pathologies, it is compulsory to perform quantitative and qualitative analysis for each of those informations. A better diagnosis can be given to the patient if the related data are presented in an effective, accurate and understanding manner for the physician. Once the diagnosis has been determined, specialists define the patterns regarding the treatment of the patient. Then it comes clear that a simulation-oriented tool can be quite useful in order to practice quirurgical operations and predict their consequences. The virtual environment must be close to reality in the sense that it should give real sensations to the physicians, when manipulating a reconstructed heart.

Innovation has allowed the development of these simulators: powerful software and hardware systems plus accurate image acquisition technologies are key factors in that sense, as pointed out by Ezquerro, Navazo, Morris and Monclús in [29]. As they suggest, any medical imaging process should pass through:

- *Data creation* in terms of acquisition with the corresponding hardware and software.
- *Preprocessing* in terms of discretization into voxels, contrast enhancement or filtering of the initial dataset with appropriate modules.
- *Analysis* in order to extract relevant information by segmentation, feature detection, registration or labeling, among others.
- *Synthesis* in order to provide physicians with scientific visualization of the concluded results.
- *Interaction* such as manipulation of the models inside an immersive environment.

The processes depicted within this document are fit inside the analysis and synthesis stages. Our first aim is to reconstruct the external (epicardium) and internal

(endocardium) surfaces of the human's left ventricle (LV) preventing physicians from manually and costly procedures. The system takes SPECT (Single Photon Emission Computed Tomography) cardiac images as its input. Those images are hardly to handle for several reasons:

- Initial data to recover lacks resolution. Typical resolutions for the data model are of  $64 \times 64 \times 20$  or  $128 \times 128 \times 20$  voxels.
- The region of interest doesn't expand all along the slices. For a given slice, it is typically located on a small centered area occupying  $20 \times 20$  pixels at much.
- The images are characterized by very coarse depth measurements.
- It is common to lack data voxels due to failures in the capture process.
- Isquemic areas, with poor or absence of blood irrigation, are not shown in the images. That means "holes" in the data that the algorithms must deal with.

Besides the LV application, our methodology is suitable for generic reconstructions in the field of computer graphics. Our reconstructions can serve for getting 3D models at low cost, in terms of manual interaction and CPU computation overhead. As a second utility, we present our method as a robust tessellation algorithm that builds surfaces from clouds of points that can be retrieved from laser scanners, for instance.

## **1.2 State of the art**

Deformable models are the kernel of our algorithms and applications. Those models are suitable for retrieving solutions in several fields like telemedicine, material analysis, cloth modeling, fluid simulation or hair animation.

Next paragraphs review the contributions regarding deformable models, which have been especially remarkable in the last fifteen years.

### **1.2.1 Deformable models**

First attempts were made in the field of computer vision. Terzopoulos, Witkin and Kass [47] presented a constrained deformable model whose goal was to recover the shape and motion of a 3D free-form flexible object from its images. Objects were modeled as elastically deformable bodies subjected to the laws of continuum mechanics. Constraints were applied as forces to the objects and were divided into intrinsic and extrinsic. Intrinsic constraints such as surface coherence and symmetric regularity were enforced in models ensuring a certain axial symmetry. Extrinsic constraints were defined from the object's profile, also known as the occluding contour, or from a human operator.

A very interesting contribution from that paper was a comparison between deformable and conventional models. The authors emphasized several characteristics that would justify the use of deformable entities. Deformable models are active, dynamic, distributed, physical, and based on controlled constraints within a broad coverage while conventional are mainly geometrical, passive, kinematical and based in very strict constraints that provide a narrower coverage.

The models presented in [47] accommodated several deformations like bending, stretching and shearing in an elasticity framework. They also provided a method for

combining stereo and motion in order to recover object's shape according to depth measurement. Note that this method, essentially, considers objects in front of a strong background. The numerical scheme was based in an implicit version of the Euler solver.

This model was extended to support generalized cylinders enhanced with deformation parameters to control the main axis and the walls of the underlying cylinder [94]. A natural evolution of this scheme consisted in the case of deformable superquadrics. Solina and Bajcsy [88] proposed it and stated the ambiguity of these surfaces in the sense that different sets of parameters can retrieve the same superquadric. Their model was based in the Levenberg-Marquardt method for nonlinear least squares minimization. Parametric deformations such as tapering and bending were applied to the model. In fact they introduced a new deformation for modeling cavities. They stated the importance regarding the ordering of deformations since matrix multiplication is not commutative.

Terzopoulos and Metaxas continued this paradigm by proposing a methodology that would satisfy the requirements of reconstruction and recognition simultaneously [95]. The paper combined membrane Splines with parameterized superquadric ellipsoids for creating a new family of models, deformable superquadrics. The main characteristic of those was their ability to deform both globally and locally. Their behavior was governed by rigid and nonrigid dynamics expressed through a set of Lagrangian equations of motion. Several degrees of freedom were defined like a translation vector, a quaternion-based rotation and a scale among others. Besides that, the user would interact with the model by using mouse control. Those interactions included viewpoint selection, initialization and changes in the global parameters. Their formulation involved numerically stable equations of motion that were integrated using an explicit Euler solver. External forces were divided into short-range forces, obtained from gradients, and long-range forces, based on distances between data points and the model. They suggested the calculation of central moments of the data for the seek of initialization (initial values at the beginning of the simulation).

Weiss [102] performed shape reconstruction from an incomplete dataset providing a method that would handle shapes with smooth and sharp areas without the use of any threshold. The concept of local refinement was applied by building a mesh denser in rapid variation parts. This adaptation was done automatically by applying a new optimization principle to a spring model. However there were undetermined parameters like the stiffness of the springs. Some general principles were presented regarding surface consistency, translational and rotational invariance, dimensionlessness, handling different scales of variations, a wide domain of applicability, computational efficiency and consistency with human perception. The method looked for the minimal overall curvature for a surface built from springs whose energy was defined in terms of stretch, bend and external forces (image attraction).

A similar approach was presented by Wang et al. [101]. In that case they applied Hamilton's principle. This principle states that the motion of an elastic grid follows a path from an initial state to a final equilibrium form that minimizes an action integral that balances kinetic versus potential energy. The model derived the potential energy from shape constraints and the grid deformation. This deformation was modeled as a grid of springs to ensure consistency in position, orientation and curvature. A kinetic energy term was also associated to the whole system. It is also interesting to note their

statement attending the importance of a friction term that would help dissipating the kinetic energy. Derivatives were computed by central differences in the spatial domain and a backward difference scheme for the temporal domain. The numerical part was solved by using an implicit Euler formulation. A snake accomplished for the boundary model by attaching it to the grid of springs.

Snakes (active contours) are a very well known paradigm [46]. Basically consist on curves that move inside an image-domain due to the action of internal and external forces. The forces are defined so that the snake will conform to a desired final configuration, related to some boundary or specific feature in the image. Several extensions to this methodology have been presented through time. See section 3.3 for more details.

Cohen and Cohen [17] used an inflation force to expand the snake model and leave it from stopping at spurious edges and artifacts. This strategy made the snake less sensitive to noise and initial conditions.

Xu and Prince [104, 105] presented a variation based on parametric active contours, the Gradient Vector Flow external force (GVF), which has been extensively used within this project. They stated two key problems related to active contours: the need for the initial contour to be close to the dataset and the difficulties when progressing into concave regions. The GVF formulation has been extensively documented and tested in this project. See section 3.3.4 for further comments.

The same authors presented an extension to the GVF field in [106]. In there they generalized the GVF when finding difficulties for forcing a snake through thin boundary indentations, for instance. In order to solve that, they introduced a varying weighting function in the energy functional, instead of a constant term. In addition, they decreased the smoothing effect near strong gradients by a second weighting function. In fact, they applied weighting functions to the smoothing and data terms respectively, instead of constant regulation parameters where a dynamic tuning is not possible.

McInerney and Terzopoulos [58] superpose a simplicial grid onto the image and evolved a deforming snake model iteratively. The model leads to interesting results thanks to its ability to flow into very complex shapes with significant branches. Moreover, the model was designed so that it could change its topology dynamically. In fact they embedded the snake paradigm into the field of simplicial domain decomposition.

The snakes were defined discrete as a set of nodes interconnected by springs. Those springs would not remain constant during the simulation providing the model with a better adaptability to the image features. An inflation force was applied to push the model towards the image edges until exerting the external forces near them. The system was integrated by using an explicit Euler method.

It is important to note that the system would react differently attending to the snake colliding behaviors. It could collide with itself or with another snake inducing the system to disconnect or reconnect nodes accordingly. In order to solve the ambiguity problem related to the recovery of an object made of several independent parts, they



distributed uniformly a set of small circular snakes over all the image area. These snakes would expand or shrink at their own until recovering all the features.

The same authors presented an extension of the previous model by their ACID technique (Affine Cell Image Decomposition) in [61]. They described the so-called T-surfaces from the previous T-snakes. The framework was extended to 3D by using a space decomposition into tetrahedral cells with the Coxeter-Freudenthal triangulation method. They applied this methodology to several medical scenarios like a human vertebra phantom (Computed Tomography, CT imagery), the left-ventricular chamber and aorta (CT imagery) or the vascular system of the brain (Magnetic Resonance, MR imagery). It is also important their discussion about thresholding and region growing, another techniques in the same field that can be very difficult to tune in an effective way. Thresholding is very noise-sensitive and region growing is difficult to control because it does not follow any image edge information, does not apply any noise-suppression technique and provides no local control for the segmentation.

Hamarmeh and McInerney [39] presented a method that addressed several points regarding shape deformations applied to medical analysis. The deformable shapes were modeled using a mass-spring scheme. The connectivity was based on the medial axis of the object. They used operator or statistics-based deformations to control the deformation at different locations and scales. Furthermore, principal component analysis retrieved feasible deformations. The selected numerical scheme was also an explicit Euler method. The spring actuation allowed intuitive deformations like radial and directional bulging, bending, tapering and scaling. Rotation and translation was implemented by means of external forces. Scaling was derived from muscle actuators.

The muscle actuation lead to the design of a deformable organism for automatic medical image segmentation. The latter authors plus Terzopoulos defined this new paradigm in [40] according to the studies of Artificial Life presented in [96, 97]. As a matter of fact, copying it with the enormous quantity of cases available in medical analysis is a hard task. In that sense, they introduced this new paradigm that utilized contextual knowledge to perform a correct analysis in regions with poor or absence of data.

The authors define a deformable organism actuated by muscles. The organism search for the most stable structures first and then uses neighboring information and prior knowledge to determine the boundary regions of interest. The deformable “worm” has its own motor system controlled by shape deformation actuators and motor controllers. Moreover, sensory organs are defined regarding several sensitive parameters like image intensity, image gradient magnitude and direction. One important point to note is their use of the Canny edge detector [14] like in our approach, as it will be deeply explained in the next chapters.

Staib and Duncan [90] stated that it is better to use the shape information as specifically as possible. Like in our approach, they find difficult to use generic edge detectors only because those tools do not find necessarily the edges corresponding to the boundaries of the object. They ignore model-based information and higher order organization of the image.

This methodology consisted in applying flexible constraints modeled like probabilistic deformable models. Boundary finding was defined like an optimization problem and solved by a posteriori objective function.

More work in the field of probability applied to image feature identification can be retrieved from McCulloch et al. [57]. They proposed a probability density function to identify features in an image taken from a class, given a template image. They demonstrated their method in MR images of the brain. Their probabilistic model tries to match the positions of features in the template with those existing in the image.

Lobregt and Viergever [54] presented a mass-spring snake scheme where the internal energy of the model depends on local contour curvature. We make use and have extended this approach in chapter 6. The internal and external forces are conveniently weighted in order to adapt the method to the imagery. As a clear contribution, they defined methodologies in order to avoid shrinking of closed models and clustering or gathering of vertices in the corners of the model. By checking the edge lengths at regular intervals, vertices can be added or removed when needed, providing the model with an adjustable local control, useful in unfeasible areas of the image. More information concerning this method can be seen in chapter 3.

Another variation to the snake paradigm can be seen in [4] where Mosquera et al. presented a model where topological changes are possible. The structure adjusts itself in order to adapt to local features, segment several objects concurrently and find holes inner to the objects. More details can be seen in section 3.3.2 of this document.

Bro-Nielsen [13] presented the families of active nets and cubes, following the tradition of the snake paradigm and evolving it to two and three dimensions respectively. As he stated, the advantage of active nets compared to active contours is the large number of internal nodes. Besides that, in the case of active cubes the interior is defined so that there is information about the object beneath its surface. An internal energy term controls the first and second-order smoothness of the net. There is a controlled continuity stabilizer that ensures  $C^1$  (first derivative) continuity at the joint points. For the external energy, any operator based on the image intensity might be used. The equilibrium state is reached by locally minimizing the energy term. For that purpose, Bro-Nielsen used an improved version of the Greedy algorithm. Moreover, this methodology provided several ways for cutting the nets if needed by the datasets.

Active cubes were modeled like elastic volumetric dice whose interior was defined. Computation times seemed to be large so that interior nodes were only allowed to move inside a slice but not between slices. This decision reduced the CPU times by a factor of 2. This methodology provided nice results in simulated operations where the physician would cut the model or move interior bones realistically.

Joukhadar et al. in [45] presented a solution for penalty method's two major drawbacks: determining the visco-elastic parameters and finding the best stepsize. In their paper, they provided a discussion regarding the differences between impulse and penalty-based methods when dealing with collisions.

As a matter of fact, they state that all solids can be considered as deformable objects with near-infinite rigidity. Then the behavior of a rigid scheme should be similar to the

behavior of a deformable model where rigidity is assumed to be extremely high. Several connectors (springs) were defined for that purpose (linear, torsion and joint). They also automated the subdivision of a rigid object into several components plus the computation of masses. This is especially important if we are to ensure that mass and inertia are respected globally and locally. The system was solved by an explicit scheme based on a second-order limited development with adaptive stepsize. A clear implementation of this methodology is their *Robot $\Phi$*  system [44].

Montagnat and Delingette [69, 70] demonstrated the equivalence between registration-based deformation and the application of a global force to a deformable model. From that statement, they proposed a constrained deformation scheme submitted to global, external and internal forces. All the forces were weighted by what they called the locality parameter. This parameter would control the degrees of freedom for the model and the shape variation accordingly, allowing the model to evolve from a coarse to a fine execution automatically. For the implementation they used their simplex meshes paradigm [23]. The external force was computed as a vector directed along the normal direction proportional to a distance measurement (from a vertex to the dataset). Tests on medical images and generic clouds of points were presented. In the case of medical images, contour information was provided by the gradient (Sobel masks) and the property in the voxels. As they pointed out, it is important to note the importance of knowing the gray level range of interest associated to the features to segment.

Cotin, Delingette and Ayache [21] presented a real-time computation of elastic deformation of soft tissue for surgery simulation and planning. They applied this technology to the concrete case of the liver. The segmentation of the images, CT data, was performed by a contrast enhancement stage followed by an edge detection algorithm and a simple thresholding technique. 2D contour extraction was then used to generate a set of 2D binary images that would be stacked as to get a 3D version of the dataset. Simplex meshes were also used in order to model the surface. Their use was justified by their decimation properties regarding the quantity of triangles in the final mesh, much lower than with other isosurface techniques like the Marching Cubes algorithm [55].

As long as the model involved real-time interaction, its interior had to be modeled somehow. Decomposition into tetrahedral elements was selected to be the best solution. Soft tissue deformation was approximated by 3D linear elasticity. An improvement of the physical model allowed simulating quasi-nonlinear elasticity. They also reduced the computation time by applying a preprocessing stage that took advantage of the linearity and the superposition principle. Thanks to that, some elementary deformations could be precomputed. As a conclusion, one can say that the appreciation of the results was essentially qualitative which can be considered enough if we take into consideration the lack of knowledge in the field of soft tissue behavior in the abdomen.

Koch, Gross et al. [48] presented a facial surgery simulator using a finite element method (FEM) scheme. They applied this scheme because no constraints on computational cost were applied so that it was accepted. Their system was coupled to a commercial modeling and animation software. As its input, they took photogrammetric CT scans of the patient. As a relevant contribution, they presented an automatic process for extracting the stiffness parameters of the soft tissue. Those were derived from segmentations of the dataset.

The evolving paradigm was typical in the sense that a global energy term would be minimized all along the surface under the presence of external forces. They employed a nonlinear, globally  $C^1$  continuous patch based on triangular shape functions for the facial surface. The external forces were computed by connecting the surface with the skull by means of nodal springs.

It is also interesting that they defined the whole set of boundary conditions by using texture maps on the facial surface. The system is then adapted to user's convenience in the sense that the texture map can be generated by commercial software.

For more details regarding deformable models read the surveys provided by McNerney and Terzopoulos [60], Terzopoulos and Fleischer [93] and Montagnat et al. [71]. Metaxas also wrote a very complete analysis in his book [65].

### **1.2.2 Models in cardiac image analysis**

Several attempts have been made in the last years in order to build deformable models suitable for the analysis of cardiac imagery. Better diagnosis procedures and training sets depend from that area of research. Segmentation of the left ventricle is a key process that allows the physicians to compute diagnostic information such as the ejection fraction ratio (see appendix C) or the wall motion analysis. This section summarizes several results on the segmentation of cardiac imagery. Details on the implementation of the methods can be seen in section 1.2.1.

Staib and Duncan [90] analyzed transaxial MR images of the ventricular wall within their system. In there, the endocardial (inside) and epicardial (outside) walls of the LV were delineated from  $256 \times 156$  images. They also presented an effective approach to temporal sequence analysis applied to a cardiac motion sequence, also from MR images ( $256 \times 256$ ). They inferred the motion of the left ventricle boundary by segmenting several frames. The problem is solved for the first frame of the sequence and subsequent frames are initialized with the previous boundary that was found, assuming small changes between frames.

McNerney and Terzopoulos [59] efforts in that area have been huge. Their "balloon" technique was applied to the segmentation of the LV surface and the tracking in a dynamic volume in order to estimate nonrigid LV motion over the cardiac cycle. They used CT data of a canine heart for those tests. The volume data consisted in 16 acquisitions for the entire cardiac cycle, where each instant of time was built from 118 slices of  $128 \times 128$  pixels. Each slice was 0.9 mm thick so that each voxel represented  $0.729 \text{ mm}^3$  of volume in the space. Typical experiments involved initial meshes of 20 triangles that were sequentially refined until 5120 triangular elements for the final mesh.

The user must supply the approximate center for the images as a first step. The user then specifies the initial size and position of the model in the 2D image. Initial resolution level can also be manually defined. While the system iterates, the user can interact with it, correcting possible loss of important features. The process is completed when the user decides so, by visual inspection. Like in [90], the final mesh for an instant was used as the initial for the next reconstruction. That ensures less errors and faster convergence

rates because the subsequent meshes will have to deform slightly only in order to reach equilibrium.

Mullick and Ezquerro [75] presented a new methodology suitable in the area of registration. The idea behind their system was to automatically find the orientation of the LV from SPECT data. We have dealt with this kind of images in this project. See appendix A for more details.

Their algorithm finds an accurate and fast delineation of the LV long-axis, which is presented like a 3D curve. The methodology begins by the segmentation and continues by applying topological goniometry (analysis of the normal vectors associated with the polygons) to the dataset. They experimented on both Phantom (see appendix D) and actual patient's data with good results. Maximum errors of 4.47 degrees in the horizontal angle and 0.23 degrees in the vertical angle were reported for the Phantom dataset. Their method was extensively used with real data. 124 consecutive patient datasets including normal and pathological cases were treated and compared to results derived from physicians. Only 8 of the cases failed the tests while the rest were correctly quantified. It is important to note that the test involved only 30 seconds per dataset.

Metaxas et al. [79] analyzed the motion of the LV from tagged MR slices. They presented a new family of parameterized deformable primitives suitable for complex shapes and, at the same time, a few number of parameters involved. The parameters were modeled like 6 functions (DMPF or deformable models with parameter functions) as opposed to being constant which ensures local control. Added to that, they applied global translation and rotation. Their model can be easily generalized although they focused in twisting and axis offset deformations, typical in the LV.

The model iterates in a Lagrangian scheme of motion and stops when forces equilibrate because there is no inertia at all. As a simplification, no stiffness was added to the global parameters of the system. The tagged dataset provides correspondence over time between individual points. This fact allows the recovery of the LV twisting motion. As in previous approximations to the problem, a recovered model is used as the initial shape for the next reconstruction.

The datasets in their study comprised 400 material points each, describing the motion of the LV during the systole period. They treated two normal and two abnormal hearts. The normal heart motion consisted in 5 acquisitions. For the first test, they derived a magnitude of contraction between 20 and 25% in the radial direction. The system retrieved a total displacement of the long-axis of 18 mm (typical lengths of the LV are 75 mm). Then the measured longitudinal contraction was 24%. The twisting motion was determined to be around 18 degrees. For the second heart, the overall contraction was 25% and the twisting angle during systole was quantified to 20 degrees.

The abnormal hearts were pathological cases where the patients diagnosis had been hypertrophic cardiomyopathy. This disease is manifested by bigger hearts that do not pump as they should. For those cases, they delivered a radial contraction of 15-20% and a longitudinal contraction of 7%. The twisting motion was quantified to 27 degrees. All those results lead to interesting comparisons between the normal and abnormal cases.

Bardinet, Cohen and Ayache [8] presented four different approaches to tracking surfaces in a sequence of cardiac images. From the tracking they inferred quantitative parameters useful for physicians such as the variation of volume, the wall thickness during a cardiac cycle, the ejection fraction and the twist component of the LV. They used Nuclear Medicine and X-Ray CT images.

They refined a superquadric model by means of a parametric deformation. For a given set of 3D points, they begin by fitting data with a superellipsoid and then refine this with Free Form Deformations (FFDs) that retrieve complex deformations defined by a small number of points. They use boxes of 6000 points that are resampled to 130 points which stands for a compression rate of 47.

One of their experiments consisted on the simultaneous deformation of two surfaces in order to recover the endocardium (1500 points) and the epicardium (4500 points) of the LV. As long as the FFD is a volumetric deformation, this is possible for the system. Final models consisted in 130 points and retrieved compression rates between 23 (two FFDs) and 46 (one FFD). Simulating both surfaces separately or simultaneously presented similar errors. For the thresholding of the images they set a level of 40% of the histogram maximum, which seems to be a reasonable value for SPECT imagery of the LV myocardium. It is also interesting to note the use of artificial caps at the base of the heart in order to get two separated and closed surfaces.

Regarding the temporal sequence of the cardiac cycle, the authors suggested four tracking strategies: fitting the model independently to each image, independent representation with a reference deformation, recursive representation and recursive representation with a unique deformation. The strategies were different according to the use of the continuity of the motion basically. Tracking the entire cardiac cycle took 25 minutes on a DEC Alpha 300 machine.

Rückert [83] used geometrically deformable models (GDMs) and templates (GDTs) to segment and track the heart in 2D and 3D cardiac MR images. In his dissertation he points out that although MR images are high-resolution data, their quality can be severely affected by artifacts due to respiratory and cardiac motion or even blood flow. He also depicts that ischemic (with poor or absence of blood irrigation) areas appear as dark regions in MR images.

For the tests the system was input with 256 x 256 x 8 pixels images where each image plane had a thickness of 10 mm and a pixel size of 1.17 x 1.17 mm. As a first step, they select manually a volume of interest of 128 x 128 x 8 pixels centered on the ventricles, left and right, of the heart. The segmentation is somehow slow for the first image but faster for the rest (subsequent simulations use the previous as their initial state). Reported time was 15-30 minutes for the first slice and 1-2 minutes for the rest (Sun Ultra 2 workstation).

Montagnat and Delingette [70, 72] segmented the LV from an MR image by using FFDs applied over a sphere (simplex mesh) made of 800 vertices. Their system seems to be very robust in the sense that tests with different datasets are also successful (brain, liver, blood vessels, etc.).

They also processed a 4D SPECT database consisting in healthy and pathological patients. The mean deformation time for all models was 2 minutes and 34 seconds (700 vertices meshes). The simulations took 3 stages of 20 iterations each in order to perform a coarse-to-fine evolution. The 3D models were quantitatively analyzed in their paper.

Laading, McCulloch et al. [49] proposed a method for modeling gated cardiac SPECT imagery by tracking anatomical points within the organ's volume. From the tracking they built a composite image specifically designed according to the heart's motion prior knowledge. Their dataset consisted of 16 acquisitions of  $64 \times 64 \times 16$  voxels each. Voxel's size was 7.1 mm, a very coarse length. The heart was contained in a region of interest measuring  $16 \times 16 \times 16$  voxels. As their system uses a reference image, gate 8 (mid-diastole) was used to play this role. Typical computation times were 3 minutes per gated image (DEC 433au workstation). From the simulation they derived an overall volume change graph among other measurements that can be very effective when trying to detect function abnormalities of the heart.

Paragios [78] proposed a level set method to segment MR cardiac images. Like Rückert [83], Paragios stated some difficulties associated to MR imaging of the heart. Among others, the presence of papillary muscles in the endocardium and some difficulties when trying to separate the myocardium from other structures apart from the heart. He used the GVF paradigm [104, 105] as the external force, like we have done in this project. He confirmed that this vector field does not give much information about the distance but provides the system with an optimal path to the cardiac borders.

Paragios presented a model whose forces were perfectly tailored for the LV. A curvature-driven term, a boundary-driven bi-directional force, an intensity-driven region force accounting for the homogeneity of the cardiac regions and an anatomy-driven constraint related to the relative positions of the endocardium and epicardium. The author stated that the next step would consist on completing the system with an automatic tuning module that would select the value for all the involved parameters in a better way.

Bonciu, Weber and Nguyen [10] presented a new acquisition system using a fast rotating 2D ultrasound probe used to reconstruct the deformations of the LV. The probe acquires successive conic sections of the LV during one cardiac cycle at a maximum speed of 8.7 r.p.s. The acquisition rate is 48 images per second. Thanks to these high rates, the system is capable to acquire all the needed planes in a single cardiac cycle (commonly 0.7 seconds).

The initial data is sparse and irregularly spaced due to the inner characteristics of the discrete acquisition of the probe. Their model retrieves the missing data by an iterative interpolation method based on the 3D Fourier transform.

The system was tested to store 100 consecutive conic images of  $508 \times 508$  pixels (gray level images). After a segmentation process, there were 3000 measurement points available. The reconstruction process was applied to a grid sampling of  $64 \times 64 \times 64$ . The ejection fraction parameter was estimated for this patient. Its value was 0.416, which agreed with physician's estimations.

Legrand et al. [53] describe each part of the heart as a different set of triangles. The edges of the triangles are modeled like springs that are compressed and stressed according to heart's motion. The system tessellates the first image of the set and defines the regions there. Then it tracks automatically for the regions in the rest of the images following an energy measure. Comparisons between hand drawn regions and the automatically derived ones have been presented with differences that are under 10%.

Sachse et al. [85] have modeled the electro-mechanics in the LV by a fusion of different models (anatomical, electrophysiological, intercellular current flow and force-based). A 3D realistic model of the LV is presented. This model allows the simultaneous simulation of electrical excitation propagation, force development and deformation. It uses deformable grids and the FEM method.

Cardot et al. [18] have developed a program for automatic contour detection in gated SPECT imagery. Recent tests were applied to 110 explorations with each consisting in 8 images per cardiac cycle. They use the center of mass of the images as a value to be compared with data in order to check its belonging to the heart tissue. In fact we have developed something similar in this thesis. The visual analysis of images superimposed to their contours retrieved a 92% of good labelings, 3% were suitable and 5% were catalogued as incorrect due to digestive uptake. Some results on digestive-affected pathological hearts are also presented within this document. See chapter 6 for further reading.

Neumann et al. [28] are also presenting some of the latest results in terms of segmentation, now over MR images. They begin by applying a bottom-up multi-scale analysis that retrieves the gray level appearance of the structures adjacent to the endocardial border. Then those can be easily removed avoiding confusion. In a second refinement phase, the method builds a statistical model of the radial gray level profiles from the estimated border center of mass, similarly to the results presented in [18] and in our approach. Their system evolved correctly in 91% of the 150 MR images chosen. Similar tests were presented by Ordás, Boisrobert and Frangi in [77] where the model runs over an active shape model framework applied to 110 cases with promising results. Spreeuwers and Breeuwer extract the epicardium and endocardium boundaries from Short-axis MR images simultaneously by using coupled active contours [89].

Echocardiograms are also a suitable technique for imaging the LV. In that sense, several approaches have been proposed lately. Morales et al. [74] applied anisotropic filtering and active contours to contrast enhanced echo images of the LV. LV volumes were measured in 16 patients (5 normal and 11 abnormal). GVF was used as the external force. Hang et al. [41] used the GVF vector field also in their new geometric deformable model. It combines the geodesic active contour model and the GVF. Promising results may arise in the future.

Radeva et al. [33] automatically segment the U-shape presented by the LV in the echo contrast images so that myocardial perfusion (MP) can be quantified objectively. They begin by applying an anisotropic filtering process to enhance contours according to their magnitude and orientation. The deformable model is based on the active shape model and signed distance potential field in order to achieve robust results. The algorithm was tested in 180 images taken from 4 different cardiac datasets. The results were compared



with the expert segmentation guidelines given by physicians. The maximum difference was 12 pixels in 95% of the images (a mean difference of less than 4 pixels).

### **1.2.3 Soft tissue modeling applied to other contexts**

Deformable models can be applied to other contexts such as cloth modeling and fluid simulation, to mention some of them. As long as our system began as a cloth-modeling tool [5] we present an overview of the existing contributions in this area. Simulators in this category can be useful in several areas like virtual human modeling, material analysis and fabric design among others. We will focus in cloth modeling as it introduced us to the development of our final solutions.

Breen, House and Wozny [11] defined a theoretical model representing the microstructure of woven cloth with a particle system. By testing cloth samples in a real environment they derived the parameters necessary for their simulations. As they stated, each material is characterized by its own parameters. Moreover that, trained eyes would recognize one from another by the way the fabric waves, for instance. That statement justified their studies in the sense that designers and engineers are quite interested in characterizing the draping properties of cloth.

Their approach took the following premise: if we model the low-level structures of a material and we aggregate their interactions accordingly, correct macroscopic behavior will emerge. Geometric relationships were modeled between neighboring particles as energy functions. Some of the behaviors involved thread collision, stretching, bending and trellising. Note that these forces have been extensively used in section 1.2.1 as well. They derived energy equations for the 100% cotton, the 100% wool and for some samples made both of polyester and cotton. Qualitative and quantitative conclusions were reported on 1 m x 1 m samples of cloth draping inside a 0.5 m x 0.5 m x 0.5 m volume cube.

Simulations consisted of three phases: evaluating the dynamics of each particle, performing an energy-minimization to enforce constraints between particles and correcting their velocity.

Baraff and Witkin [7] described a cloth simulation technique based on stability even when taking large stepsizes. The system coupled a technique for enforcing constraints on particles with an implicit integration method (backwards Euler method). Cloth samples were modeled like triangular meshes with a very straightforward definition about internal forces (stretch, shear and bend). Due to the sparse characteristic of the involved matrices, the simulation framework resulted significantly faster than previous approaches.

Their simulations included clothes relaxing from an initial deformed state onto several characters with complex geometry. Some local adjustment needed to be applied for a better behavior of the animations. Tests with different stiffness for the bending force demonstrated the speed of the simulator, with variances in the running times under 5% while stiffness parameters were ranging from 0.1 to 1000.

Synthetic authors can be dressed with complex deformable clothes as stated in [15] by Carignan, Yang and Thalmann. 2D cloth samples are designed and fit in 3D characters.

They work by tailoring the cloth samples from piece to piece. As a major contribution, they design a new paradigm regarding the handle of collisions among the cloth elements themselves or between a cloth element and a rigid object such as the synthetic avatar. They also discuss the reduction of the parameters in order to get a usable interface for the final user. The automated parameter definition is defined onto the mass evaluation for each node, the resistance to stretching and bending, the dissipation rate for the energy related to stretch and the stepsize. Optimizations about collision response and numerical integration were also presented by Thalmann and Volino [99, 100].

Eberhardt, Weber and Strasser used particle systems to obtain realistic draping behavior [27]. They achieved a fast implementation with a general approach. They introduced several optimization techniques by combining techniques of computer algebra, theoretical physics, numerical mathematics, and ray tracing, in the context of computer graphics. Their system simulates the dynamic behavior of cloth over time.

Provot's [81] model was adapted to the particular stiff properties of textiles by using dynamic inverse procedures. They showed that if a concentration of high stresses occurs in a surface, the local deformation becomes unrealistic. The only solution so far was to increase the stiffness associated to the springs but this decision increases dramatically the cost of the algorithm. They studied the case of a sheet hanging from its two corners, subjected to the action of gravity. Their simulations were held by an explicit Euler solver.

Choi and Ko [16] continued working in the same line as the previous approach and presented a semi-implicit cloth simulation technique that retrieved very realistic animations, regarding wrinkles for instance. They also stated that although the damping term is inherently necessary for stability, it degrades the realism of the simulated cloth movement. For the seeking of realism, they proposed a method including both artificial damping and material intrinsic damping but no fictitious damping. As presented in their paper, their method would be considerably simpler than the methodology presented in [11].

Two interaction models were defined accounting for stretch and shear (type 1) and flexural and compression resistance (type 2). One can point out the concept of buckling (failure in a rigid material) which stands for a crash in rigid-body animation but for success when draping textile fabrics. Fabrics do not break or collapse since they pass the unstable state and reach equilibrium. In their paper they describe a solution for the post-buckling instability that may lead the system to undesirable results. Reported animations involved resolutions between 1 and 2 cm. Results on human motions and fabrics dropped over a solid box were presented.

Cordero and Matellanes [19] have presented recent experiments with their cloth animation system based in a particle system consisting in masses and springs. The model is easy to implement and retrieves realistic animations. The main contributions are the control of the orientation of the particles and the elongation functions. External forces include gravity and friction.

Finally, another important contribution is the work of Fedkiw et al. [12]. Their research has overcome several major problems related to the simulation of folds and wrinkles. Their novel contributions include a hybrid explicit/implicit numerical scheme, a

physically correct bending model and a dynamic constraint mechanism that preserves folds and wrinkles while treating collisions in an efficient manner, among others.

Their solver combines the flexibility of explicit schemes when handling nonlinearities with the high convergence ratios that characterize implicit methodologies. The system handles three types of collisions, with level-set methods: self-collision, cloth-object collision and cloth-character collision. Some of their animations have been used in movies like Terminator 3 and Harry Potter and the Chambers of Secrets.

### **1.3 Structure of this document**

This chapter has introduced an overall view of our goals. Deep explanations on all the related topics are presented in the following chapters.

Chapter 2 presents our major contributions to the bidimensional processing of our initial datasets. Bidimensional processing is needed in order to mark the data boundaries of our imagery. From the 2D data slices, we are able to build a 3D voxel data set that has been correctly labeled.

Chapter 3 describes the deformation models that have been tested and compared, in order to provide our 3D reconstruction method with a reliable, accurate and fast solution.

Chapter 4 is devoted to the underlying numerical theory related to the simulation of our deformable models and the solvers used in order to implement them. There are several solvers available, different in terms of efficiency, robustness and speed of evaluation. We analyzed them and explain the reasons for our final choice.

Chapter 5 presents the possibilities regarding the selection of a geometrical model suitable for our specific needs. Depending on the type of representation we restrict the complexity of the shape and we must choose one or another methodology.

Chapter 6 presents several results related to our research in 3D reconstruction as a modeling tool. We are mainly focused with the reconstruction of the LV, in normal and abnormal cases, but we also demonstrate that our technique can be applied in other fields that require generic reconstructions. We present several examples on this where our method can be used as a tessellation algorithm or in a low polygon-rendering framework.

From that point, we present our final conclusions regarding the achievement of our initial goals. We also point out some future lines that might be taken into consideration for the seeking of completeness.

The final appendices are intended to guide the author in several topics which are related to this project like SPECT imagery (appendix A), the LV physiomy and functionality (appendix B), medical parameters of interest (appendix C), the Phantom volume as a synthetic model for testing (appendix D) and the DICOM format for the images (appendix E).

## **2 2D Processing**

Bidimensional processing is needed in order to mark the data boundaries of our imagery. From the 2D data slices, we must build a 3D voxel data set that has to be correctly labeled.

From the labeling of the borders, we derive an external force, a vector field (see chapter 3), which pushes our particle mesh to its final configuration, recovering the 3D desired shape.

This chapter presents our major contributions to the bidimensional processing of our initial datasets:

- An automatic scheme designed for labeling the data borders.
- A comparison between several generic edge detectors, used as the first step of our algorithm.
- Five different techniques developed specifically for the edge labeling: the vector sign method, the gradient method, the case-based method, the radial-circumferences method and the MLC method.
- An automated process intended to find an automatic circle that first filters the data, in the context of the left ventricle reconstruction (see appendix A).
- An automated process intended to find the division slice, in the context of the left ventricle reconstruction (see appendix A).
- A new approach that applies vertical coherence and bounding boxes in order to filter the entire dataset from top to down.

### **2.1 Manual vs. automatic**

Access to a 3D model obtained from patient's data can have several applications like support on diagnosis, surgery planning, student training or even remote-operation. A first approximation to the problem would be using a manual process with specific image-processing software like in the segmentation of the visible human data set [82]. In this huge project, both the human male data set and partially the female data set have been segmented by interactive manual procedures. They have used a commercial tool plus some other specific routines included in their packages.

The package works using NURBS curves as its basis functions. Those functions are used to interpolate points previously marked by the physician. The program allows the user to freely edit and delete the curves. It must be said that NURBS curves belong to the Spline class of primitives, widely used for their virtues [2]:

- Local control of shape.
- Smoothness and continuity ( $C^2$  continuity in the case of Splines ensures high local smoothness).
- Evaluation of the derivatives at the joint points.
- Stability.
- Ease of rendering. NURBS are included in the GL and GLU graphical libraries [38].

Nevertheless, automatic or semi-automatic segmentation is an open research task for the biomedical and engineering research community that is giving results one after another. It still poses several difficulties that must be overcome in order to avoid manual processes that require deep medical knowledge and experience.

This is the main goal of the segmentation algorithms designed and tested in this project. To provide the diagnosis tool with an automatic border detector that only requires some parameterization and less manual adjustments.

## 2.2 Tested operators

General-purpose edge detectors have been extensively used as a first part of our automatic border detection algorithm. Input imagery must be treated in an optimum manner as follows:

- Important edges cannot be missed because of their major contribution to the border detection. Spurious responses must be avoided.
- The distance between the actual and detected position of the edges has to be minimal.
- Multiple responses to a single edge must be minimized.

From the images we derive an edge map. The gradient of the resulting edge maps has vectors pointing toward the edges of the data to recover (vectors orthogonal to data borders). Moreover, these vectors have large magnitudes only near the edges. The same magnitudes decay to zero in homogeneous regions.

We have tested several operators in order to fulfill these requirements.

### 2.2.1 Robert operator

This operator consists on evaluating  $\nabla I$ , where  $I$  is the intensity (gray level) of the image. For a given pixel whose intensity value is  $I(x,y,z)$ , we derive its laplacian value as equation 2.1 shows:

$$\begin{aligned}\frac{dI}{dx} &= \frac{I(i+1, j, k) - I(i-1, j, k)}{2\Delta x} \\ \frac{dI}{dy} &= \frac{I(i, j+1, k) - I(i, j-1, k)}{2\Delta y} \\ \nabla^2 I &= \frac{d^2 I}{dx^2} + \frac{d^2 I}{dy^2}\end{aligned}\tag{2.1}$$

Some results are presented in figure 2.1.

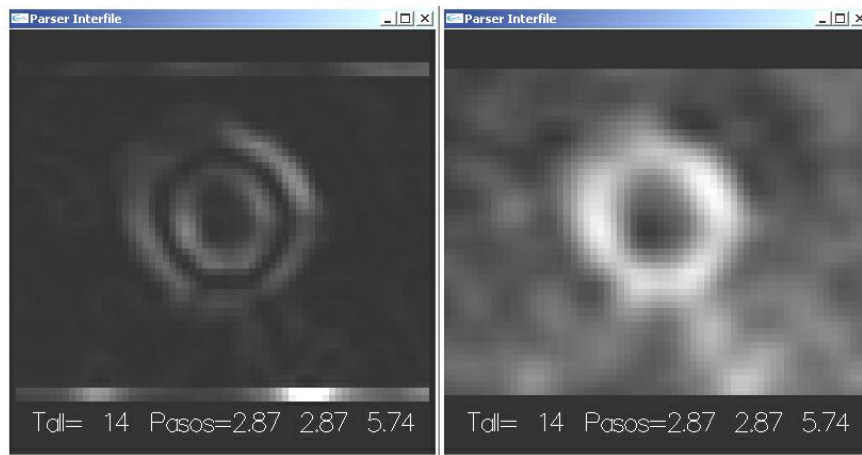


Figure 2.1: Results of the Robert operator in medical images.

As figure 2.1 shows, the map can be relatively effective because it fails to find the high gradient areas in some regions of the image. If we take into consideration that image in figure 2.1 is one of the best cases we can deal, it comes clear that we might miss important features of the image, especially if it is low contrasted due to irregularities in the captures.

Moreover that, medical images within the scope of this project have very low resolutions (64 by 64 pixels) which definitely difficult the operator task if it is not robust enough. Attempts were made on doubling the resolution as showed in figure 2.2.

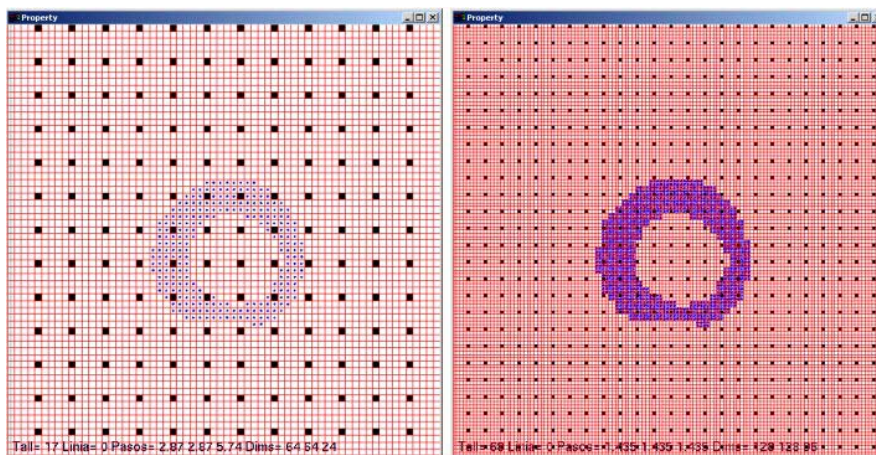


Figure 2.2: Two resolutions for the same data slices.

Doubling the resolution of a data slice from 64 x 64 to 128 x 128 pixels does not really improve the derived edge map. In fact it might be poor because of the smoothing process that it implies. This process can delete the high frequency information (edges) that the image contains.

In figure 2.3, we show how the image gets smoothed and consistently, the edge map loses precision. The case shown in figure 2.3 is exactly the same as presented before, in figure 2.1. The unique difference is the resolution: 64 by 64 pixels in the first case, 128 x 128 in the second, doubled by the program.

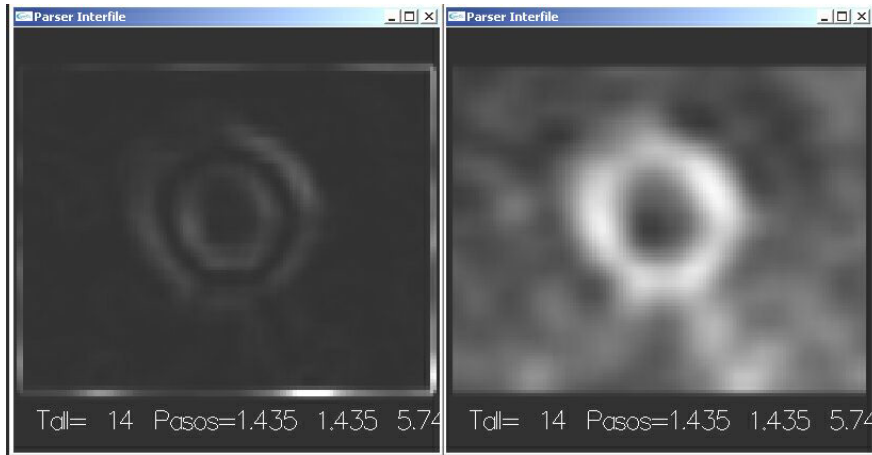


Figure 2.3: Results of the Robert operator in medical images, with doubled resolution.

For the doubling procedure, we applied the masks shown in figure 2.4.

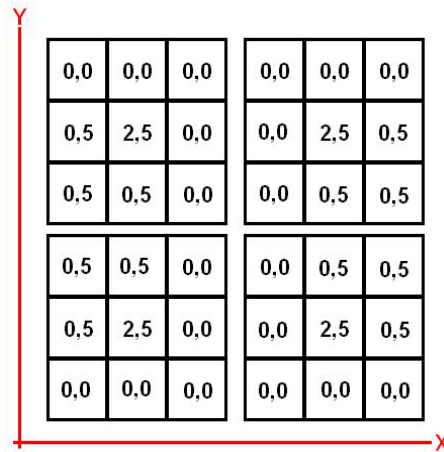


Figure 2.4: Smoothing masks for doubling the resolution.

As figure 2.4 depicts, the central pixel is weighted by a factor of 2.5 in front of the 1.5 factor used for the vicinity pixels. After adding the values, the result is correctly averaged dividing by 4.

### 2.2.2 Canny operator

Within this project, the Canny operator [14] took importance in two stages:

- As a first approximation for the external and internal data borders.
- As a previous generic segmentation process preceding our final MLC robust border labeling (see section 2.3.6 in this chapter).

In the first attempts, we chose this operator as a first approximation for the border labeling because of its robustness even when the image presents irregularities in the intensities. In that sense, it was more effective than the Robert operator.

When marking the borders, we noted that there exist several local-maxima that cannot be rejected by property filtering. In some cases, discreet nature of data and its loss makes it difficult to mark the frontiers by the sign of the associated vector field (see

chapter 3). Then we finally adopted the Canny-Edge Detector as a sophisticated segmentation filter that might improve the border labeling.



Figure 2.5: Left, Original SPECT image. Middle, Robert operator filtered image. Right, Canny operator filtered image.

In figure 2.5 there's a clear difference between the results of both operators. Canny offers the best edge detection for the SPECT (see appendix A) images that we must treat.

The Canny algorithm consists on the following steps:

- Smoothing the image with a Gaussian filter to reduce noise and tiny artifacts.
- Traverse all the pixels in order to determine the gradient magnitude and direction. Its direction has to coincide with the maximum change in intensity.
- If the gradient magnitude at one pixel is larger than the one in its two neighbors (vicinity in the gradient direction), mark the pixel as an edge. Otherwise the pixel belongs to the background.
- Remove weak edges by applying hysteresis thresholding.

However it can be seen that when the direction of a gradient slope at a ridge point is not normal to the ridge contour, the ridge point is not selected to be an edge. The typical implementation of this algorithm misses some edges that can be recovered by the revised Canny edge detector [24] algorithm. This algorithm enlarges the classification with a third member: the minor edge. Then pixels can be labeled to be major edges, minor edges or part of the background.

### 2.2.3 Compass operator

The Compass operator [84] represents an important step in the state of art of edge detectors because of an important aspect: it does not smooth images in the usual sense. As the previous operator algorithm shows, smoothing serves as a method for removing noise or introducing scale into an image. However the smoothing process can remove edge and corner information besides noise.

The Compass operator algorithm generalizes smoothing to vector quantization, which seems to be a more detailed description of part of an image. Important features of this detector are:



- It allows arbitrary distributions of pixel values. On the contrary, most edge detectors assume that the edge or corner divides an image window into two constant regions.
- This model can be used over any image type (binary, gray scale, color, multi-spectral).
- The treatment of color is more intuitive than in other models.
- The detection of edges and the detection of corners can be treated in a unified framework.

The name of the operator, "compass", comes from the fact that it uses a circular window divided in half by a "needle" that spins until finding the orientation that maximizes the gradient. Then standard techniques like non-maximal suppression or hysteresis thresholding are suitable for extracting edges.

### 2.2.4 Sobel operator

The Sobel operator [31] avoids having the gradient calculated by using a 3 x 3 neighborhood mask. For a given pixel [i, j], it uses the mask of figure 2.6.

a0	a1	a2
a7	aij	a3
a6	a5	a4

Figure 2.6: The Sobel operator mask.

Then the magnitude of the gradient can be evaluated like equation 2.2 illustrates:

$$|G| = \sqrt{s_x^2 + s_y^2} \quad (2.2)$$

Where the partial derivatives  $s_x$  and  $s_y$  can be found by the expressions of equation 2.3:

$$\begin{aligned} s_x &= (a_2 + 2a_3 + a_4) - (a_0 + 2a_7 + a_6) \\ s_y &= (a_0 + 2a_1 + a_2) - (a_6 + 2a_5 + a_4) \end{aligned} \quad (2.3)$$

$s_x$  and  $s_y$  can be implemented with the convolution masks presented in figure 2.7.

-1	0	1	1	2	1
-2	0	2	0	0	0
-1	0	1	-1	-2	-1

Figure 2.7: Convolution mask for  $S_x$  and  $S_y$ .

The convolution masks of figure 2.7 respond maximally to vertical and horizontal edges, always relative to the pixel grid. There's one mask for each of the two perpendicular orientations. The masks can be applied separately in order to get different measures for every orientation.

The Sobel operator smooths the input image to a greater extent that means that it is less sensitive to noise than others. The operator also produces high output values for similar edges.

### 2.2.5 Comparative between operators

We will show several images treated with the Canny, Compass and Sobel operators. Those, specially the first two, are the ones considered to be robust enough for the application. It must be said that within our application, the main purpose of the generic edge detector is to provide a first clue of the edges that must be taken into consideration for the posterior border labeling.

As a first comparison, we present figure 2.8.

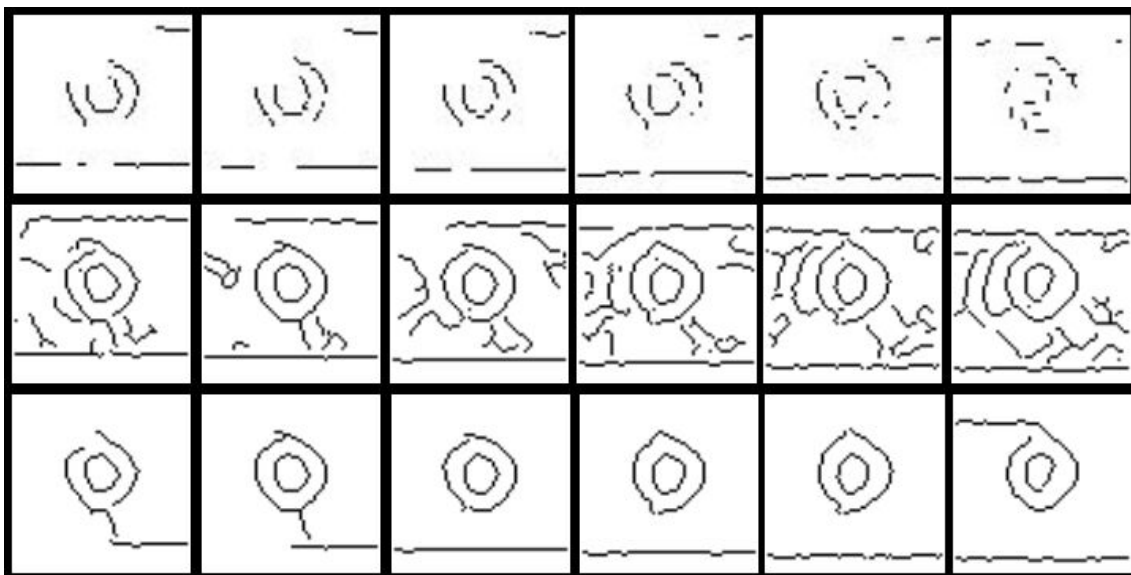


Figure 2.8: Filtered medical images. Top, Sobel operator, middle and bottom, Canny with different thresholds.

As figure 2.8 shows, it comes clear that Canny offers better results. The Sobel operator is not as robust as the first in front of noise. In order to avoid the impulsive noise, it is better to use high thresholds. For the Canny detector we get the best results with a value of 1.8 for the  $\sigma$  and 0.3-0.8 for the low and high thresholds respectively.

If we compare the Canny edge detector against the Compass operator we conclude that both can be useful with a suitable tuning of the input parameters. We present several tests as a comparison between both operators (figures 2.9 to 2.12). All the tests were made with left ventricle SPECT images from actual patient's data.

It comes clear that the more we increase the parameter  $\sigma$ , the more edges are missed. We must find a trade-off between the limits (0.2 to 5.0) for the best results. Low values

give very complete but noisy edge maps. High values miss the borders that we are precisely interested in maintaining. Medium rows edge maps are good enough for our purposes ( $\sigma = 1.8$ ).

The first test image (healthy heart) is a pretty good input because of the extent of its ROI (**region of interest**), a 30 x 30 pixels mask. This fact ensures a good basis for the operators. Nevertheless the second test image (pathological heart) has the same resolution (64 x 64 pixels) but half the extent for the ROI (12 x 12 pixels mask). That's a tough scene for the operators to do their job.

If we take a look at figure 2.11 (second test, first sequence, fourth row and so) we see that the internal edge disappears. In fact, we can hardly maintain the external edge after applying the operators with a high  $\sigma$ . Then we infer that for tiny ROI's we should use lower  $\sigma$  values though we might have more noisy edge maps.

When keeping the  $\sigma$  value constant and varying the thresholds, it seems that Canny offers best results in terms of noise absence.

The commented figures (2.9 to 2.12) are presented in the following pages.

**First test – First sequence**

- Slice 14 from a healthy heart. Resolution 64 x 64 pixels. The ROI takes about a 30 x 30 pixels region.
- The  $\sigma$  parameter goes from 0.2 to 5.0 and the thresholds remain constant with values of Low = 0.05 and High = 0.1.
- Left corresponds to Canny, right to Compass.

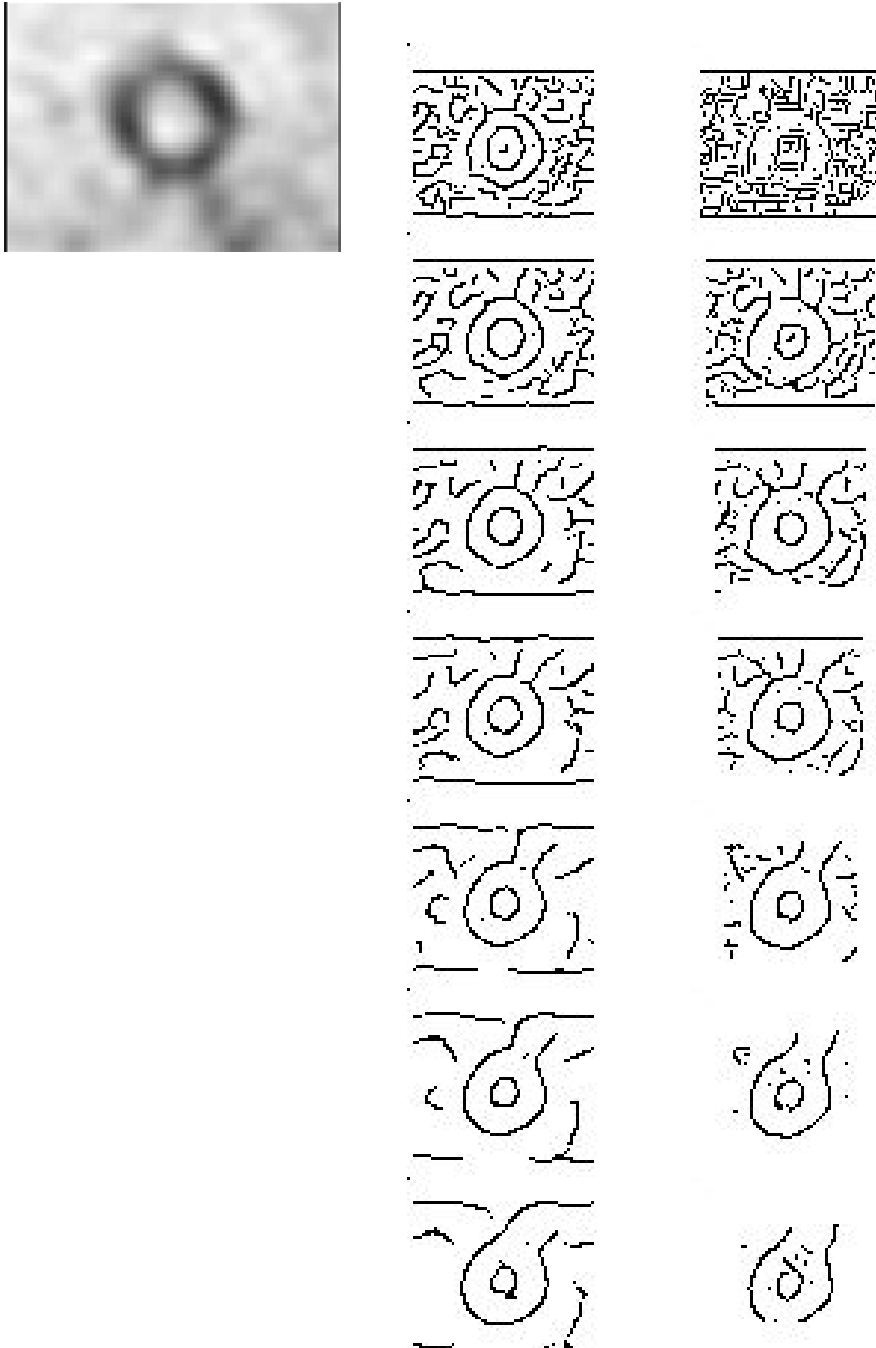


Figure 2.9: First test and first sequence comparing the Canny and Compass operators.

## First test – Second sequence

- Same image from the previous test.
- The  $\sigma$  parameter remains constant with a value of 2.2 and the thresholds take three different values:
  - First row  $\Rightarrow$  Low = 0.1 and High = 0.7.
  - First row  $\Rightarrow$  Low = 0.3 and High = 0.6.
  - First row  $\Rightarrow$  Low = 0.3 and High = 0.7.
- Left corresponds to Canny, right to Compass.

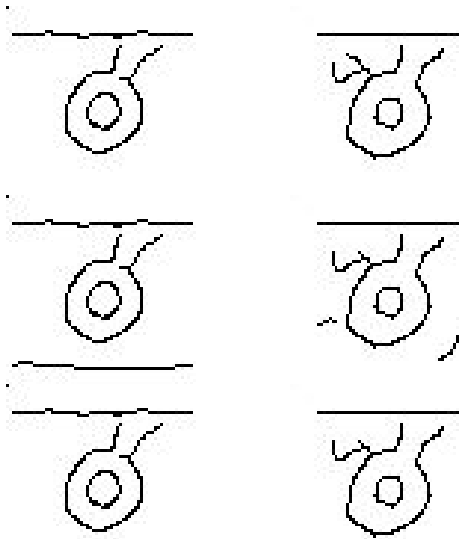


Figure 2.10: First test and second sequence comparing the Canny and Compass operators.

## Second test – First sequence

- Slice 14 from a pathological heart. Resolution 64 x 64 pixels. The ROI takes about a 12 x 12 pixels region.
- The  $\sigma$  parameter goes from 0.2 to 5.0 and the thresholds remain constant with values of Low = 0.05 and High = 0.1.
- Left corresponds to Canny, right to Compass.

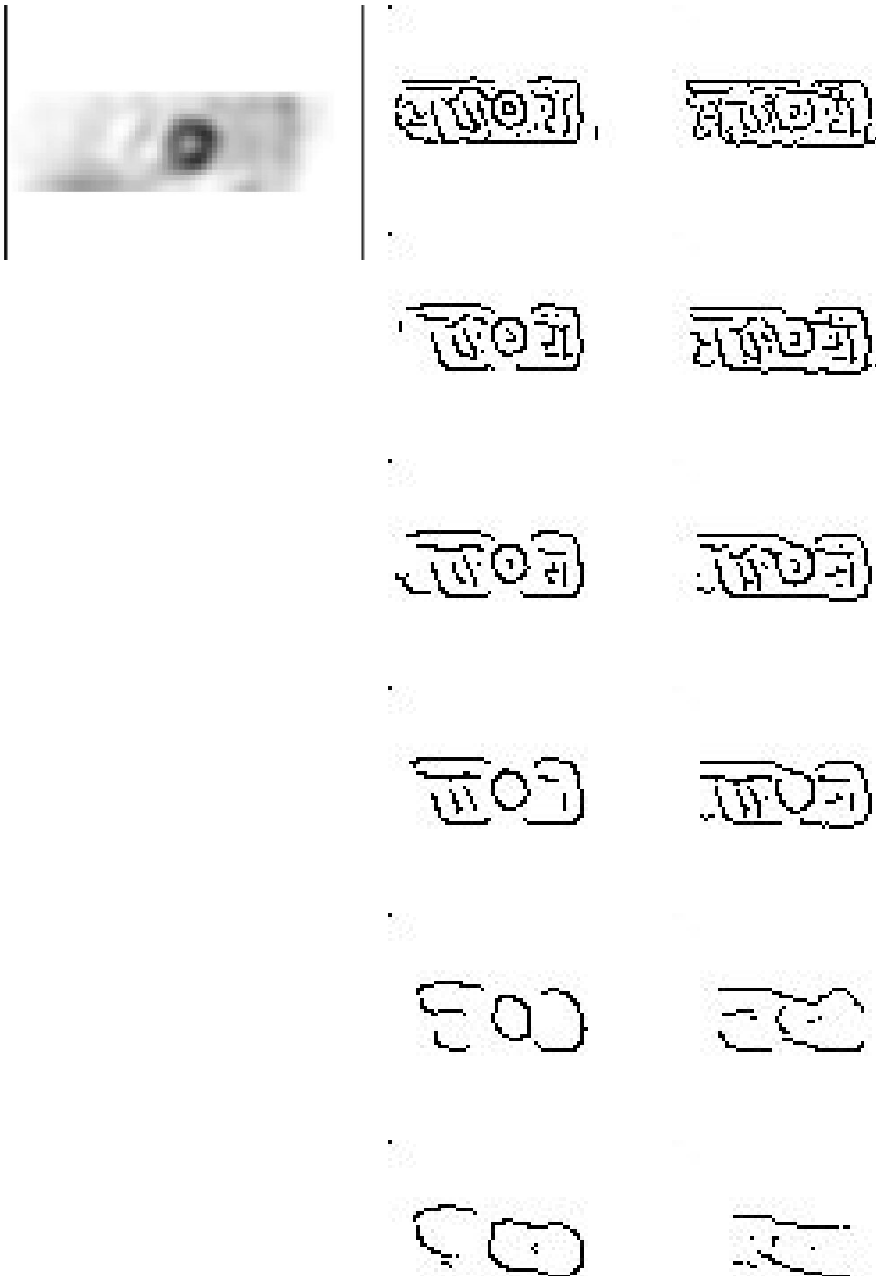


Figure 2.11: Second test and first sequence comparing the Canny and Compass operators.

## Second test – Second sequence

- Same image from the previous test.
- The  $\sigma$  parameter remains constant with a value of 2.2 and the thresholds take three different values:
  - First row  $\Rightarrow$  Low = 0.1 and High = 0.7.
  - First row  $\Rightarrow$  Low = 0.3 and High = 0.6.
  - First row  $\Rightarrow$  Low = 0.3 and High = 0.7.
- Left corresponds to Canny, right to Compass.

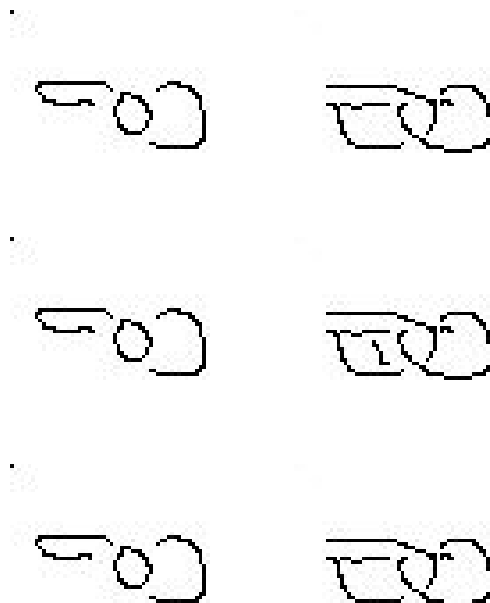


Figure 2.12: Second test and second sequence comparing the Canny and Compass operators.

## 2.3 Border labeling

As told before, the main objective of the present work is to reconstruct the internal and external surfaces of the human's left ventricle. Doing so implies identifying which part of the data belongs to the organ and which does not.

Then data will be labeled as “in” or “out”, referenced to the organ. In fact, the generic segmentation algorithms explained in the previous section, detect contours that will be catalogued, or not, as borders.

This section addresses the different techniques used for this process.

### 2.3.1 Can we avoid the labeling?

As a first approximation, one might think that a vector field can be evaluated directly from the images, with no needed labeling. This is true but not always as optimum as we may need.

Depending on the resolution of the input imagery and the ROI dimensions, the vector field derived is basically poor detailed. This effect happens because of the tiny dimensions of regions like the inner volume between both the surfaces, in some actual patient's data. If we take a look at figures 2.11 and 2.12 (second test for the Canny and Compass operators), we can see that the ROI is a 12 x 12 pixels mask. If we understand that for every pixel, or voxel in 3D, we will have an associated gradient vector, we see that the lack in space and resolution causes the vectors to be not enough. Figure 2.13 depicts this problem.

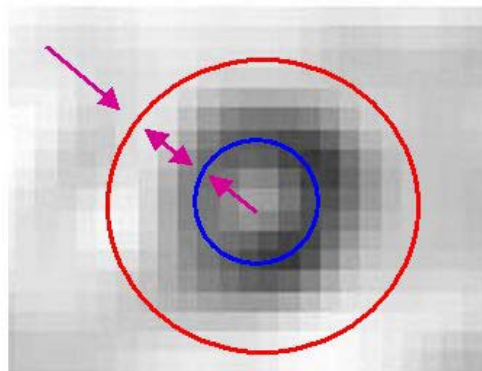


Figure 2.13: Evaluation of the vector field directly from the image. The theoretical surfaces have been manually edited as circles for a better understanding.

Figure 2.13 shows three cases: a first case where the vectors outside the external surface (red) must point inwards; a second case where the vectors between both the surfaces (red and blue) must point inwards and outwards; and a third case with the vectors inside the internal surface (blue) pointing outwards.

It comes clear that we have a very small area for placing our vectors, especially in the second and third cases. Only a few pixels allow constructing only a few vectors which means lack of information for the 3D meshes, about recovering the surfaces.



Moreover that, with this approximation the external mesh can be deformed from outside to inside but the internal one, has to be placed inside (a really small area) in order to follow the vector field that “inflates” it, until recovering the shape. It would be better to perform the internal reconstruction from outside to inside too, avoiding the placing and scaling of the initial mesh in such a small area.

As long as most of the images are characterized by small ROI’s, we decline to use this approximation and we conclude that the border labeling is necessary.

### 2.3.2 Labeling by the vector field sign

Despite what we said in the previous section, in some cases it can be useful to begin by evaluating the vector field for the images for labeling the borders after that.

In that sense it is possible to track the sign of the vector field in 2D, marking the changes that should be associated to the borders.

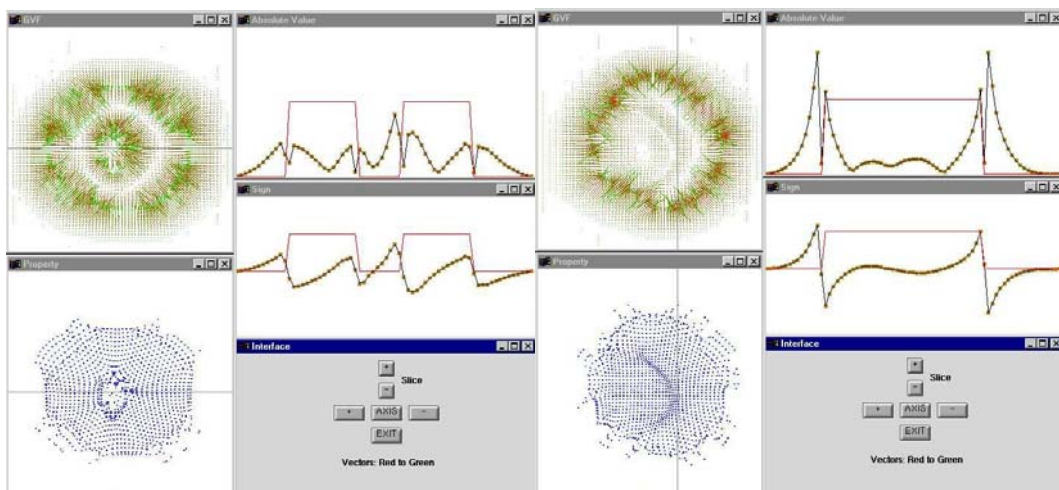


Figure 2.14: The sign of the vector field when following a horizontal (left) and vertical (right) trajectories in a data slice.

Both snapshots of the application shown in figure 2.14 are composed from five different views, for the given data slice:

- The vector field rendered in a top view of the slice. Note that there is a thin line showing the trajectory followed (top-left).
- The data to recover in a top view of the slice. Note that there is a thin line showing the trajectory followed (bottom-left).
- The absolute value of the vector field along the trajectory line. The “peaks” define higher or lower magnitudes for the vectors in that area (top-right).
- The sign of the vector field along the trajectory line. In the case of the horizontal trajectory, the sign is rendered as positive if the horizontal component of the vector points to the right. For the vertical trajectory, the sign is positive if the projected component of the vector over the vertical points to the top (middle-right).
- The interface of the application (bottom-right).

Figure 2.14 shows how the borders can be detected by high changes in the vector field sign. Following the trajectories in both the horizontal and vertical directions, allows the

application to mark the pixels (voxels) where the borders should be placed. Figure 2.15 shows the results of this process.

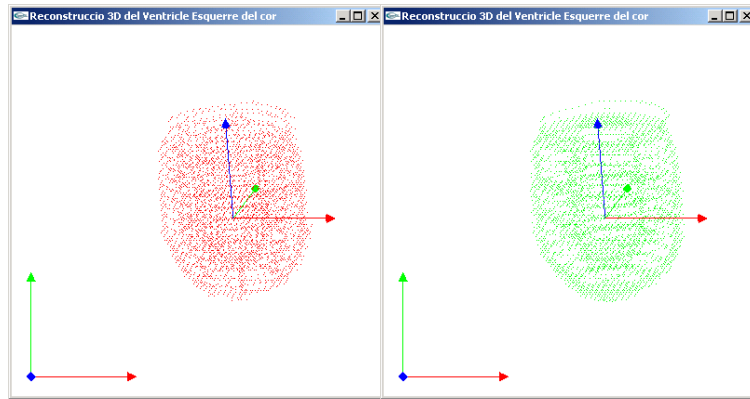


Figure 2.15: Voxels marked by sign (left) and gradient (right).

Figure 2.15 shows two processes: marking by sign and by gradient. We are discussing the first one; the second will be analyzed in the next section.

Although the process works quite well for the case presented in figure 2.15, it can be poorly effective if the ROI does not have a minimal dimension. As we commented in the previous section, this cannot be ensured.

Moreover that, the borders can be labeled in some cases but there's no way to distinguish if they are internal or external. In the following sections we present several attempts on this classification.

### 2.3.3 Labeling by the gradient

If we take a look at the absolute values shown in the two snapshots of figure 2.14 (top-right little windows), we will note that the magnitude tends to be nearly zeroed in the borders. In fact this means that both sides of the border are high gradient areas where the magnitudes tend to be at their maximum values.

Labeling consequently allows the application to obtain results like the green voxels showed in figure 2.15, where the big dimension of the ROI are permitted this process to be satisfactory.

Again this process can be poorly effective if the ROI does not have a minimal dimension and there's no way to distinguish if the labeled borders are internal or external. Follow the next section and the rest for several attempts on this classification.

### 2.3.4 Case-based classification

This is a first attempt on labeling the borders as internal or external, according to a previous case-based classification. It can be easy to explain if we look at a practical example, like the one shown in figure 2.16.

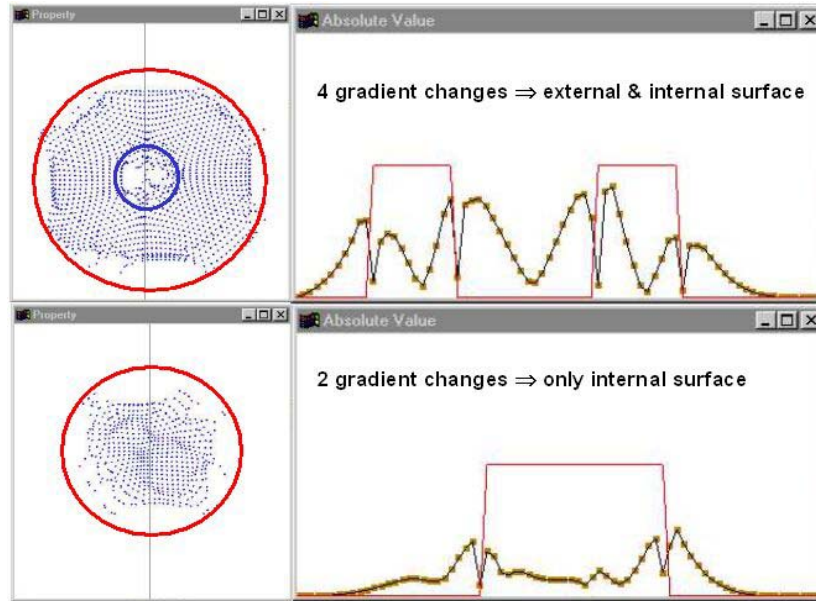


Figure 2.16: Case-based classification of the borders.

Figure 2.16 illustrates an attempt for classification of the borders between external and internal. If we do so, we can force the meshes to stop at the outside or inside borders although the tiny dimensions of the ROI do not cooperate with the vector field creation.

Then although having a low quality vector field, we can use it for the marking and then evaluating a second vector field, once we know the borders. A better description follows:

1. Evaluate the vector field, directly from the images.
2. Trace trajectories in every slice, and classify the near-zero gradient pixels (voxels) as external or internal borders. Every slice has to be inside one of the known cases, like the two slices shown in figure 2.16 (see Appendix B for a better understanding of the left ventricle).
3. Once the borders are labeled, evaluate two different vector fields: one that goes from outside to the external borders; another field that goes from outside to the internal borders (note that this approximation solves the problem on “inflating” the internal mesh, as explained in section 2.3.1).
4. With the two vector fields perform the reconstruction of the internal and external surfaces (see chapter 3).

This process works perfectly in the ideal case, where no ischemic areas exist (areas in absence of blood irrigation presented as holes in the data) but not in the contrary. It is hardly to achieve a classification algorithm that takes into consideration all the possibilities.

Figure 2.17 shows an actual patient’s left ventricle data set. Although the left slice looks well, the right one (close to the apex, bottom part of the ventricle) presents ischemia, as indicated by the blue arrow.

This slice is not a catalogued ideal case, because although there are two borders to recover, the vertical trajectory along the direction indicated by the blue arrow will find only two gradient changes (not four as figure 2.16 showed before).

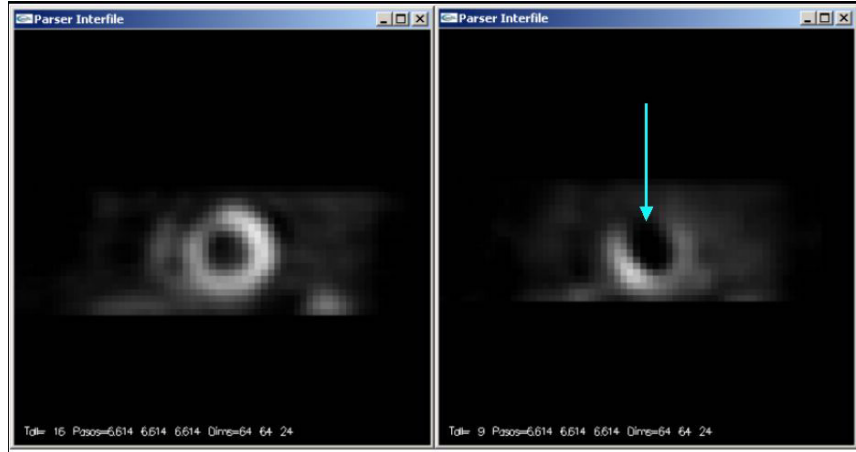


Figure 2.17: An actual patient's ischemic left ventricle.

That lack of control on the existent cases for very patient and data set forced us to take another solution to the problem.

### 2.3.5 Radial-circumferences-based algorithm

In order to solve all the previously related problems and from the knowledge of the left ventricle's shape (see appendix B), we derived a radial-circumferences-based algorithm as a pattern based method.

The algorithm assumes that the internal and external borders can be considered basically circular contours. The algorithm also starts from the fact that the physician has defined a manual coarse circle, like a first noise filtering tool (more information on this manual circle and its automation is given in the following sections).

The algorithm relies on the effectiveness provided by a generic edge detector, for instance the Canny edge detector [14], and the symmetry of the left-ventricle's images.

The steps are showed next:

- Apply a generic edge detector to all the slices.
- Delete all the segmented contours outside the manual circle defined by the physicians.
- Then for each data slice:
  - Create an initial circumference with null radius.
  - Increase the circumference radius iteratively by a defined parameter.
  - For each of the circles:
    - Save the number of changes in sign. A change in sign means that a voxel that was outside the previous circle, is inside the present one.
  - Identify the circles that provided the maximum number of changes in sign. Those circles are marked if provided more changes of sign than a given parameter. This parameter is a percentage of the total amount of voxels segmented by the generic edge detector.
- Find the slice of change that defines the beginning of the left ventricle's apex. This can be achieved by tracking all the slices vertically, from top to bottom in the Z axis,

while looking for a change in the number of circles previously identified (we must go from two circles to one).

- For each data slice, classify the segmented voxels as internal or external borders depending on their distance to the identified circles and the vertical position of the slice.

A visual explanation of this algorithm is given at figure 2.18.

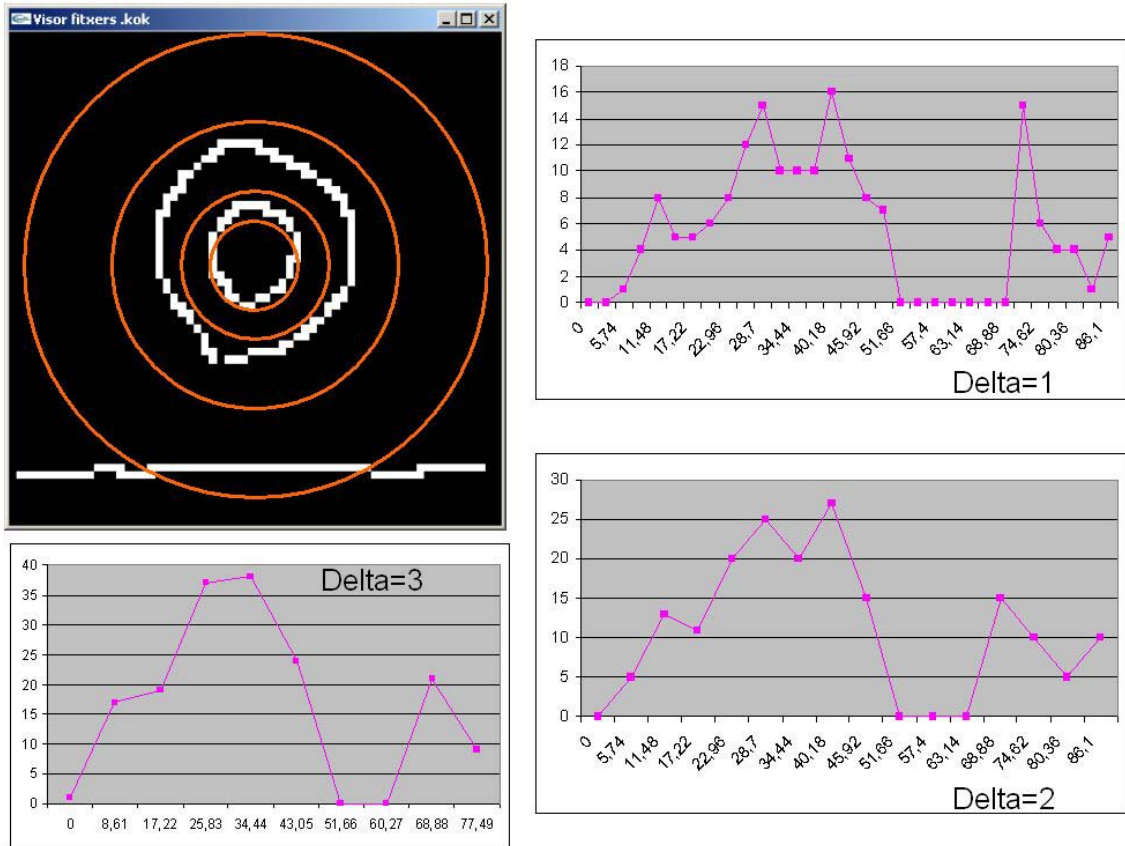


Figure 2.18: The radial-circumferences-based algorithm put to the test.

The circles can be started from the geometrical center of the image (only in very ideal cases) or from a user-defined center for the image (a tedious task if we test the application from a Usability [76] point of view). Right before the beginning we see that having a center that corresponds to the centroid of the image is important. Let's assume that we have this center for all the slices, just as a manual or automatic (see section 2.4.1) previous process.

Figure 2.18 shows one of the slices to be labeled. Several circles are defined and the amount of changes in sign is calculated and rendered in the graphs. Three graphs are presented, corresponding to three different rises for the increasing radius. As one can derive from the images, the rise begins being equal to the side of a voxel ( $\delta = 1$ ) and goes until two ( $\delta = 2$ ) and three times this side ( $\delta = 3$ ).

It is a compromise to find the optimal rise for the radius because we need to go as faster as possible (increasing the rise) without losing borders (decreasing the rise).

The orange circles drawn over the slice depict what we see in one of the graphs, in particular the graph that corresponds to the smaller rise ( $\Delta = 1$ ). The graph shows four peaks clearly. Those peaks correspond to the changes in sign of the four rendered circles.

The first circle overlaps some of the segmented voxels of the inner contour (first tiny peak); the second circle contains the entire inner contour (second peak); the third circle contains both contours (third peak). It must be said that all the calculus are made relative to the previous which means that we only take in consideration the “new” voxels in terms of changes in sign; the fourth circle has “touched” some noise that notably increases the amount on changes of sign (fourth peak).

That one is a typical case where the noise fools the algorithm, if not removed first. However that fact that we know the pattern to recover and that this noise is very far from the center can give us a clue about the reliability of the fourth peak.

A complete example over one of the phantom volumes used in this study is presented in figure 2.19.

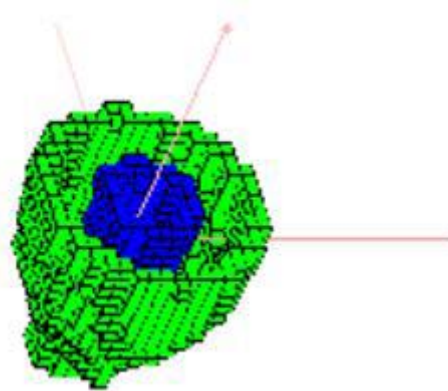


Figure 2.19: Borders labeled by the radial-circumferences-based algorithm.

As a matter of fact, this algorithm relies too much in the ideal characteristic of symmetry for those images. Unfortunately this can be hardly achieved which drive us to our final approach, discussed next.

### 2.3.6 MLC filtering

Classification is the grouping of each pixel (or voxel) to one of the classes, internal or external borders in our particular application, 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 have deeply explained that we do 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 2.2. 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 vector field that will act as an external force for the reconstruction phase. See the borders in figure 2.20.

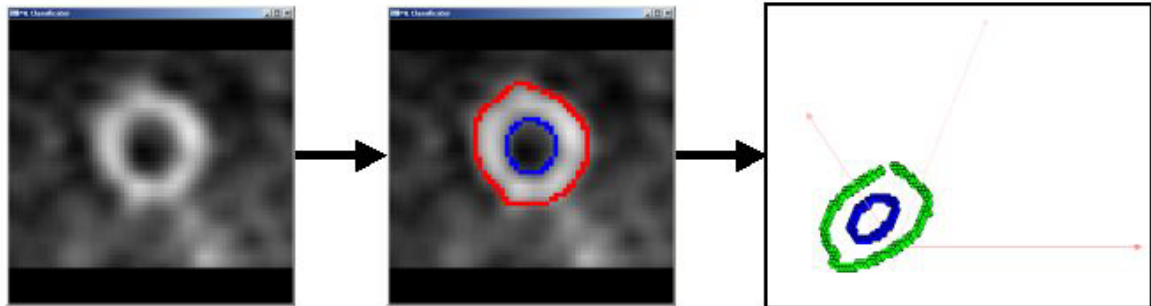


Figure 2.20: From left to right: a left ventricle data slice; 2D identified borders; borders in 3D space that will be used for getting the 3D vector field.

Our MLC implementation classifies the contours depending on their probability to belong to an external border, an internal border or none. As a precondition, our implementation assumes that there is a previous process that finds automatically:

- The smallest circle that, in terms of center and radius, can be used as the first filtering tool for all the slices. This circle should be more accurate than the manual one defined by the physicians. See section 2.4.
- The slice of change (division slice) that marks the beginning of the left ventricle's apex. The slice is calculated first and refined after as explained in section 2.5.

Let us focus on the algorithm. Those are the steps:

- Use the automatically derived circle to cut off the edges.
- Label the remaining edges with different ids.
- Delete spurious edges.
- First pass: use the first division slice to design a two-stage algorithm where edges under the division slice are marked as external borders. For the rest of the edges:
  - The algorithm calculates their distances to the global centroid.
  - It orders those distances in a vector and builds a new one containing the differences between consecutive distances.
  - The biggest difference indicates the frontier between internal and external edges, in terms of ids.
  - Then the edges can be labeled as external or internal accordingly.
- Refine the division slice as explained in section 2.5, by using the previously labeled edges.
- Second pass: use the refined division slice to again, applying a two-stage algorithm where edges under the division slice are finally marked as external borders. For the rest of the edges:
  - The algorithm calculates their distances to the global centroid.

- It orders those distances in a vector and builds a new one containing the differences between consecutive distances.
- The biggest difference indicates the frontier between internal and external edges, in terms of ids.
- Then the edges can be finally labeled as external or internal.
- Use vertical coherence by applying a final mask to all the slices. This filtering process is explained in section 2.6.

This algorithm is definitely characterized by its robustness because it performs all the calculations automatically, with less need of symmetry assumptions. The filtering circle, the centroid and the division slice are found automatically so that there's no need to assume that the original images are centered or that the physician will provide a very accurate initial circle.

Figure 2.21 presents the segmentation algorithm over a phantom data set. As it shows, the borders have been correctly detected and labeled.

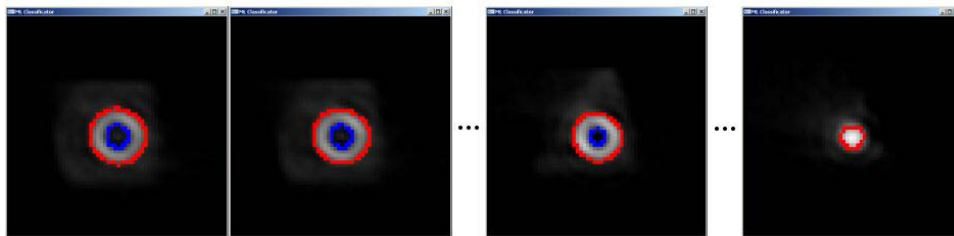


Figure 2.21: The borders detected over the Phantom data set (blue for internal and red for external).

For a real case, an example of the final labeling for a given slice of a complete cardiac cycle can be observed at figure 2.22.

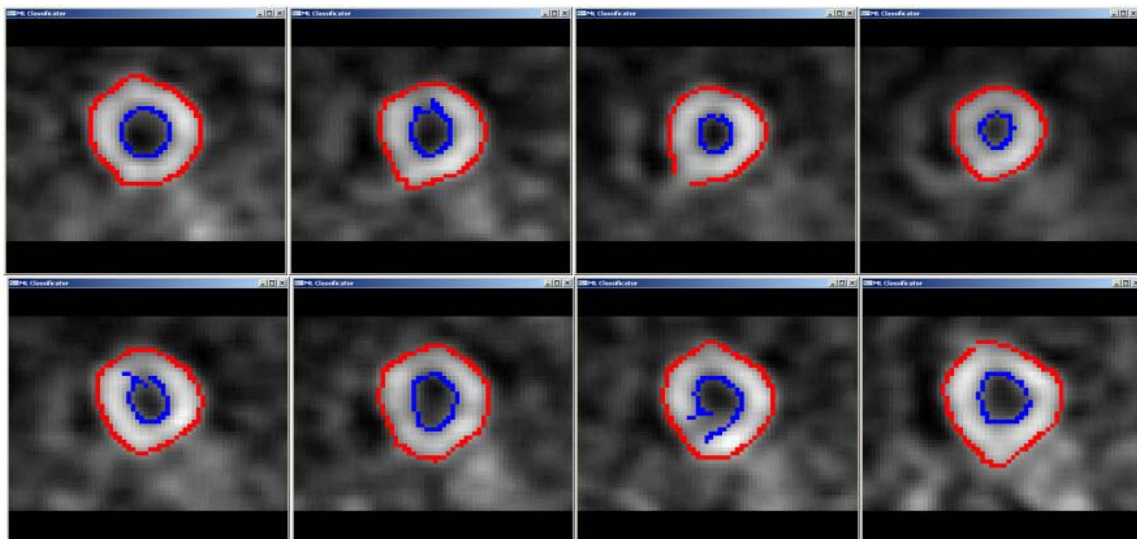


Figure 2.22: Slice 15 of a complete cardiac cycle data set.

The acquisition consisted on 8 captures ranging from systole to diastole. It must be said that the complete process used 16 meshes because of the eight instants of time with two meshes each, internal (endocardium) and external (epicardium).



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).

## 2.4 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 (see appendix C). 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.

We made a first attempt in order to avoid the initial filtering circle. It consisted on applying a property filter to the slices, as shown in figure 2.23.

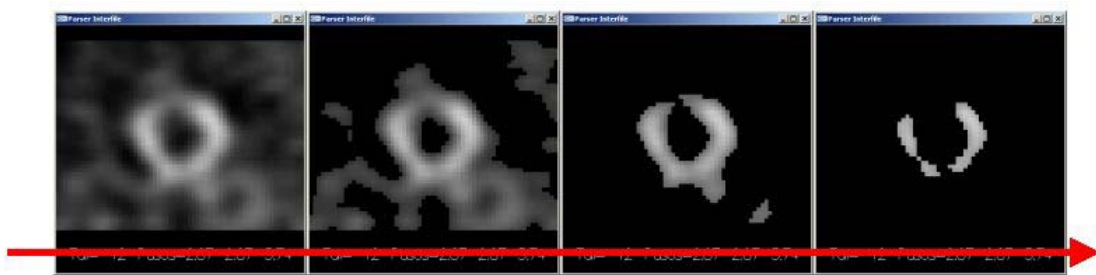


Figure 2.23: Filtering by the property value.

Although it might seem a good strategy, it is not. The reason is that the filtering eliminates noise but also useful data. This is due to the fact that some noisy areas have the same intensity values as the actual data that we need to recover.

Then it is compulsory to use a filtering circle, either manual or automatic, in order to “clean” our SPECT images.

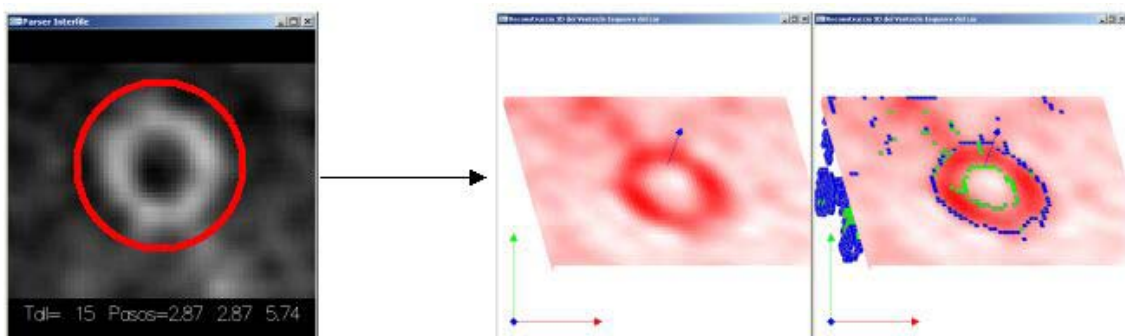


Figure 2.24: The filtering circle avoids noise plus an incorrect labeling of the borders.

Figure 2.24 shows the importance of the filtering circle. If not used, we can conclude with an incorrect labeling of the segmented borders. This fact is showed in the right images of this figure, where several isolated voxels are labeled as external (blue) or internal (green) borders when they shouldn't.

Nevertheless we must be careful when deciding “what” has to be filtered by the circle. If we proceed by cutting off all the data outside, we might have an added problem, as stated in figure 2.25.

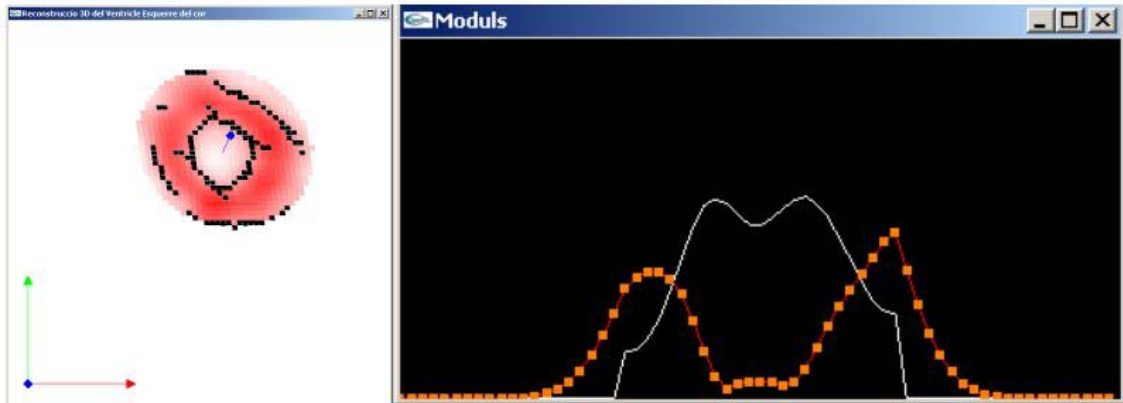


Figure 2.25: Using the circle for the property cutting off.

Sections 2.3.3 and 2.3.4 show that we can use gradient-based strategies for the labeling of the borders. If we filter by property previously to that, we might create false high gradient regions, easy to confuse with the real borders. This effect is clearly pointed out in figure 2.25 where the property for a given section of the data slice is drawn as a white thin line and the gradient is shown by the dotted orange curve. The gradient is maximum at the cut limits although those do not correspond to the data borders.

A partial solution might be smoothing the cut data, as depicted in figure 2.26. There we have, from left to right, the manual filtering circle, the original data with the superimposed circle, the smoothed data after the cutting off and the original data after the cutting off.

If we smooth only the surroundings, where the circle has actuated, we can avoid the high gradient changes that appeared in figure 2.25.

However what if we can automate all the filtering, even the design of the circle? Is it possible to filter with no property cutting at all? This is what we show in the next paragraphs.

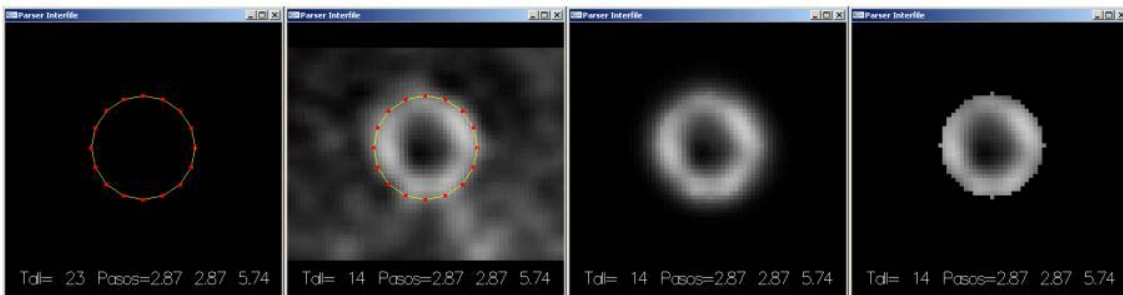


Figure 2.26: Cutting off and smoothing the data.

As a pre-process to the automated circle determination, we begin by applying a generic segmentation algorithm like [14, 84]. After that, we have a first sight at the edges. A very coarse manual circle cuts those off, as figure 2.27 shows.

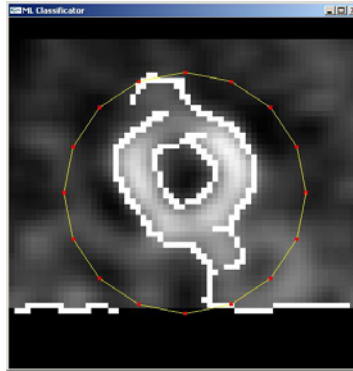


Figure 2.27: Edges detected by the generic segmentation algorithm (white).  
The coarse circle (yellow).

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.

#### 2.4.1 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 steps (for all the edges in a given slice):

- The algorithm begins by finding the probability of the edge to be circular, which means to count the number of voxels equidistant to its own centroid.
- Besides that, the algorithm compares the edge with the biggest found so far (in terms of number of voxels), swapping them if current edge's probability is greater than a given threshold and it is formed from more voxels.
- For this edge, the algorithm weighs up its centroid by using a factor taken from a Gaussian function. The factor should penalize being far from the center. The weighted centroid is added to the average count.

More details are depicted in figure 2.28.

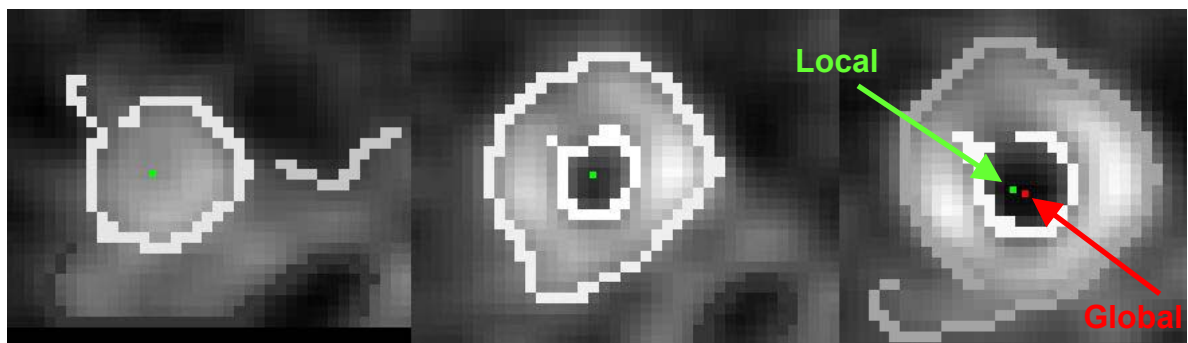


Figure 2.28: Local centroids for those slices (left & middle); Local (green) and global (red) centroids (right).

Figure 2.28 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.

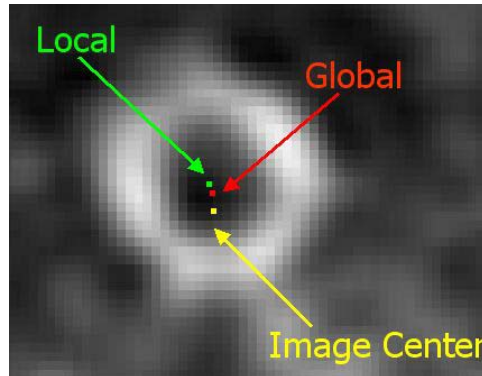


Figure 2.29: Local centroid for the given slice, global centroid for all the slices and geometrical center of the image.

Figure 2.29 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.

#### 2.4.2 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 for all the edges in a given slice:

- The algorithm checks for the edge's probability to be greater than a defined threshold.
- If so, it calculates its average distance to the new centroid and it finds a Gaussian factor that penalizes its distance to the image center. Then it applies a size restriction: it must be greater than a defined threshold, in order to avoid spurious edges.
- The algorithm checks if that distance is greater than the last saved and rejects/saves it accordingly.
- As a final step, the radius is defined as the major distance plus a user-given offset.

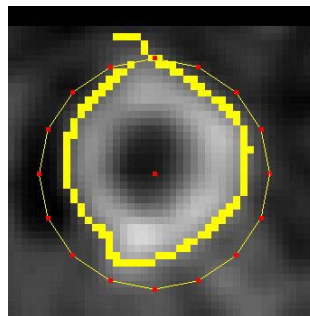


Figure 2.30: The algorithm finds the “best” edge in terms of distance to the global centroid. The radius is set to be this distance plus a user-given offset.

We can draw the new circle by using the global centroid and the major distance as the radius, as shown in figure 2.30.

The process has automated the circle setting. Moreover, the new circle is absolutely well fitted to the data, avoiding most of the noise as it is shown in figure 2.31.

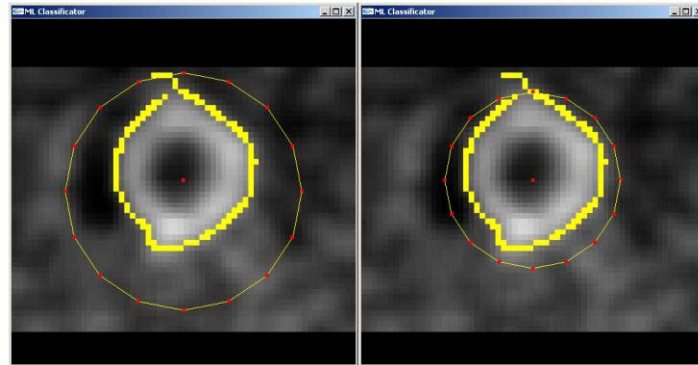


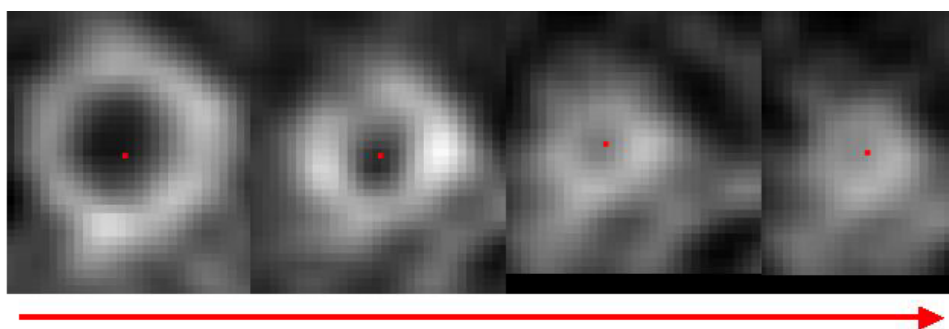
Figure 2.31: The first coarse circle (left); the automated and well-fit circle (right).

## 2.5 Finding and refining 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 grows up as we go from top to bottom, as depicted in figure 2.32.

The property is normalized for all the voxels along the data set. This means that the intensity shown in figure 2.32 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.



Near Top

Near Bottom

Figure 2.32: 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 differences in property are tracked until they get expectedly big. This high property slice determines the endocardium bottom and therefore the division slice.

The division slice is refined for the second pass of the MLC algorithm (see section 2.3.6). This is an adjustment that has to be made in order to ensure that the division slice is correct.

The refinement consists on taking advantage of the vertical coherence that should characterize our data set. If we are solid with our knowledge of the left ventricle, we must know that there are several cases that are very unlikely to happen. Those cases can be derived from the labeled borders in the slices. Take a look at figure 2.33.

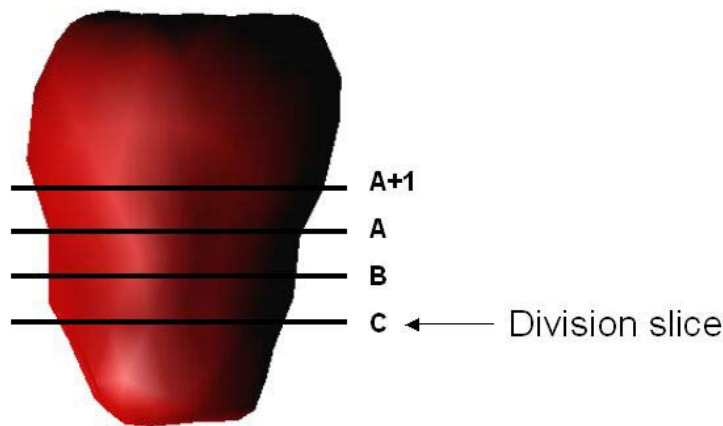


Figure 2.33: The division slice refinement process.

In figure 2.33, C is the division slice that we found before the refinement. A + 1, A and B are the slices immediately over it. The technique examines the labelings in the first three slices (A, B and C) to see if everything seems to be coherent. If necessary, it examines the labeled borders of slice A + 1.

Examining the labelings means checking if the slice has got one (external) or two (external and internal) borders.

Slice/Case	1	2	3	4	5	6	7	8
A	1	1	1	1	2	2	2	2
B	1	1	2	2	1	1	2	2
C	1	2	1	2	1	2	1	2

Table 2.1. Cases to take into consideration when refining the division slice.

Table 2.1 shows the eight cases that arise. For every case, we can see the amount of borders in slices A, B and C. If there's only one border, we understand that it has to be external because we are in the apex area; if not we must have two borders, where one is internal and the other is external. Let us examine each case separately:

1. Very unlikely to happen because it means that the apex is formed from lots of slices. This fact is totally uncommon. Notify it to the user.
2. Not possible if the quality of the data is within some certain limits. We have two borders when beginning the apex and only one in the top of it! Notify it to the user.



3. Neither A or B are bad catalogued slices. We must check A + 1:
  - a. If A + 1 has got two borders, the division slice is still considered to be C because we consider that A was bad catalogued with a unique border when it should have two.
  - b. If not, we do have a not possible situation if the quality of the data is within some certain limits. Notify it to the user.
4. Not possible if the quality of the data is within some certain limits. Notify it to the user.
5. B is the final division slice.
6. Not possible if the quality of the data is within some certain limits. Notify it to the user.
7. C is the final division slice so no changes have to be made.
8. Not possible if the quality of the data is within some certain limits. Notify it to the user.

Figure 2.34 shows an actual case, labeled as 7.

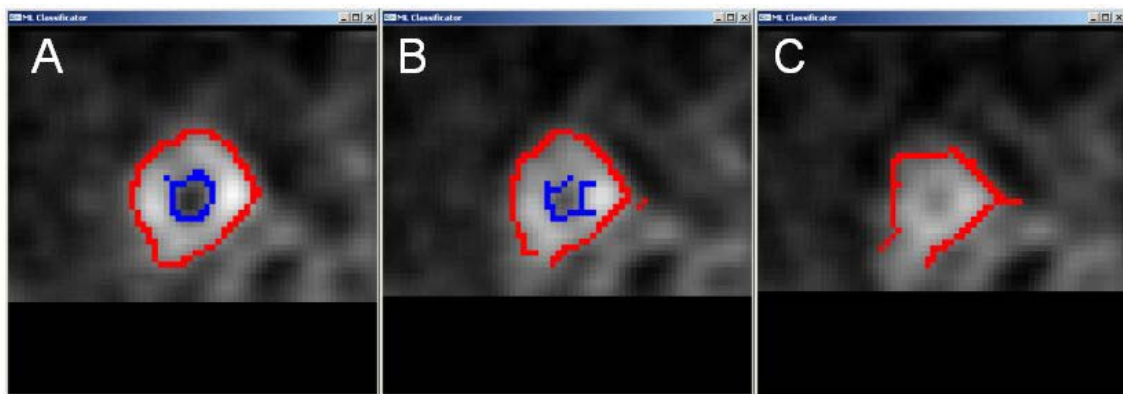


Figure 2.34: Case 7 of the division slice refinement process.

The algorithm outputs 2 2 1 for the slices A B and C. As the images show, it is clear that slice C contains one unique external border while A and B contain two borders, one external and the other internal. Then the division slice was correctly labeled at the first process and no corrections have to be made.

However problems in the generic edge segmentation algorithm can force the apparition of very uncommon cases, like 1. In figure 2.35 we see this effect.

The algorithm identifies C as the division slice, in terms of the property value in the centroid, as explained. The selection is correct attending to visual inspection but when the refinement stage begins, it detects that slices A and B only have one border which means, according to the previous catalogue, that we are in very unlikely to happen case 1.

Trying to find the cause for this, we see that the problem is due to the poor resolution of the ROI, especially in the inner surface area (2 pixels wide in slice C, 4 to 6 pixels wide in slices A and B). With such a low resolution the generic edge detectors cannot detect edges properly. If the generic edge detector does not provide a contour, it is impossible to label it as internal or external. We just miss this information and there's nothing to do even by the MLC algorithm.

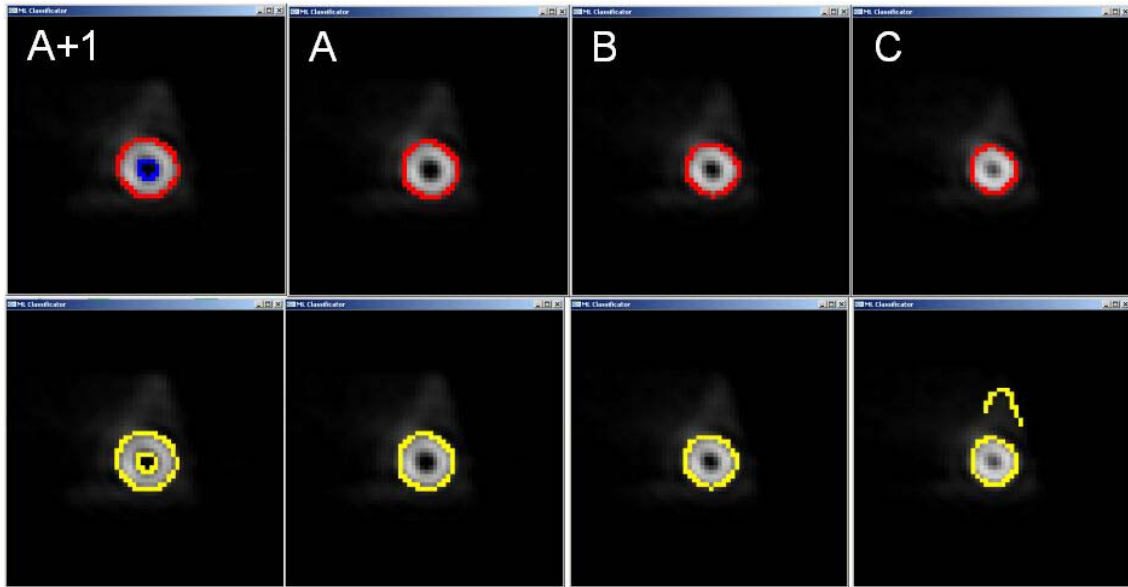


Figure 2.35: Case 1 of the division slice refinement process.

The Canny edge detector contours are rendered in yellow in the second row of the images in figure 2.35. Note how there are no contours in the inner areas, except for the slice  $A + 1$ , where the resolution begins to be suitable. Besides that, the algorithm has perfectly removed a noise contour in the slice C.

## 2.6 Bounding Box based filtering

As a final stage, after labeling the borders by using one of the previously explained algorithms, it is compulsory to attend noise removal, especially if the later 3D reconstruction can be affected.

There are several contours that might “survive” and that are not associated to the real data borders. Besides that, those contours might be inside the automatic circle and big enough to avoid the consideration of spurious.

The Bounding Box based filter is designed for taking ride of those. The filter relies totally in one of the slices, the slice that contained the biggest external contour that was used for finding the automatic circle (see section 2.4.2). It assumes that this is the biggest slice in terms of useful data, which should be true. This slice must be located far away from the apex region, near the top. Then it should contain two borders, perfectly defined. We will name this slice, the big slice.

The filter evaluates the bounding boxes of the borders in the big slice. Those bounding boxes are used to filter the upper and lower slices as follows:

- From the big slice to the division slice: we traverse all the slices filtering with the bounding boxes (internal and external) of the actual slice all the subsequent slices.
- From the big slice to the top: all the slices are filtered with the bounding boxes (internal and external), of the big slice.
- From the division slice to the bottom: we traverse all the slices filtering with the bounding box (only external) of the actual slice filter all the subsequent slices.



When talking about using a bounding box for filtering, we are referring to what is depicted in figure 2.36.

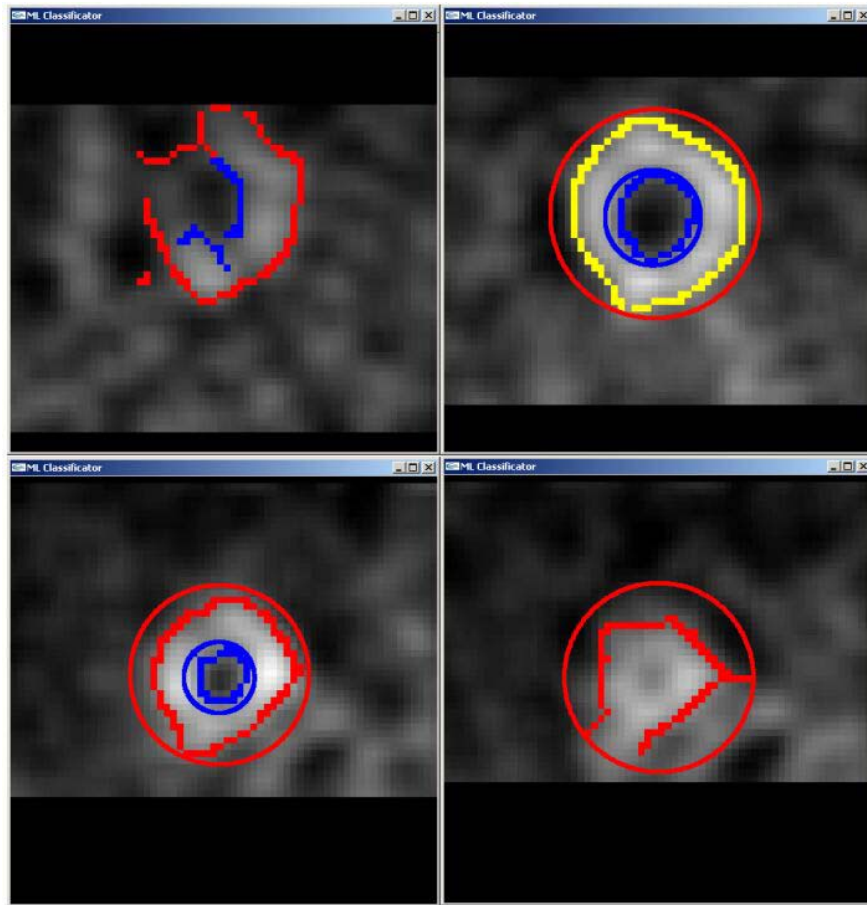


Figure 2.36: Filtering with bounding boxes.

As figure 2.36 illustrates, all the borders have their associated bounding box. In the showed example, we can see:

- One of the final slices (top-left), very near the top of the left ventricle that has been filtered with the big slice bounding box.
- The big slice (yellow) with its associated bounding box (red) and the internal border and bounding box (blue) (top-right).
- One of the slices located between the big slice and the division slice (bottom-left). The figure shows the associated bounding boxes (red and blue), used to filter all the subsequent slices.
- The division slice (bottom-right) with its associated external border and bounding box (red).

The algorithm acts as a final noise removal tool, just preceding the vector field calculation, explained in the next chapter.

## 2.7 Summary

In chapter 2, we have presented the 2D image processing that precedes the rest of blocks. The images must be treated in order to detect the internal and external borders

that allow the posterior 3D reconstruction, with the resulting recovery of important parameters like the inner volume or the ejection fraction (see Appendix C).

We began by explaining the need to automate all the 2D processing as much as possible, in order to improve speed, accuracy and robustness.

Next, we have presented the generic edge detection algorithms tested besides their utilities inside the whole application. Those algorithms are a preprocessing step needed to segment the best candidates to be the final labeled borders of the data to recover.

After the edge detection step, we detect and mark the borders for all the slices in the data set. Several strategies have been tested and intensively compared before selecting our major contribution to this task, the MLC algorithm applied to the labeling.

As a part of the MLC implementation, we have automated the evaluation of the best circle that fits data, in terms of calculating its center and radius. This filtering circle is used as a noise-removal tool that serves as a classification mechanism that scores the best contours to be labeled as borders.

We also detect the slice of change automatically. The slice of change defines the beginning of the apex for the left ventricle. Above the apex, we should have two borders referred to the endocardium and epicardium of the organ. In the apex, we only have one border, the epicardium.

As a final filter, we have implemented a bounding box based filter that ensures vertical coherence between all the data slices, checking for singularities to be repaired.

Chapter 3 describes the 3D deformation models that we have tested, implemented and selected for the 3D final reconstruction.

### 3 3D Deformation models

Given an initial data set, a deformation model can be used to recover the geometry that best fits it. Besides that, the deformable model can recover and animate the shape, motion and interaction of the geometrical object [71, 65].

The reconstruction of the left ventricle fits inside the category of deformable objects. Simple shapes or rigid geometries cannot correctly model those objects. Moreover that, deformable objects are characterized by nonrigid transformations that describe their motion trajectories, differently from the motions associated to rigid objects.

This chapter describes the deformation models that have been tested and compared, in order to provide our 3D reconstruction method with a reliable, accurate and fast solution.

Our major contributions are:

- Using particle systems in a Newtonian evolution scheme as a 3D segmentation tool.
- Testing several methodologies available in terms of internal forces designed for 2D contours and 3D surfaces. Different internal forces will lead to different deformation models.
- A comparative between several external force schemes that serves us to justify the reason to use the GVF approach.
- The implementation and exhaustive analysis of five different deformation models.

#### 3.1 Newtonian dynamics

Our reconstruction method is planned as an evolution scheme based on Newtonian dynamics despite of the deformation model used [6]. Our initial meshes are built from triangles and vertices (particles) that move due to internal and external forces. Equation 3.1 shows the Newtonian dynamics law:

$$F_i = m_i \cdot \ddot{x}_i \Rightarrow \begin{cases} \dot{x} = v \\ \dot{v} = F_i / m_i \end{cases} \quad (3.1)$$

Where  $F_i$  stands for the total force,  $m_i$  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 3.1 shows.

The mass is adjusted for every particle and the forces are defined as internal and external. Both forces drive the system to its equilibrium state. In order to solve it, we need a numerical method. This method or solver, depends on the deformable model applied, and can be formulated in terms of implicit or explicit equations. Reports on the models can be found at section 3.4. The numerical methods are discussed in chapter 4.

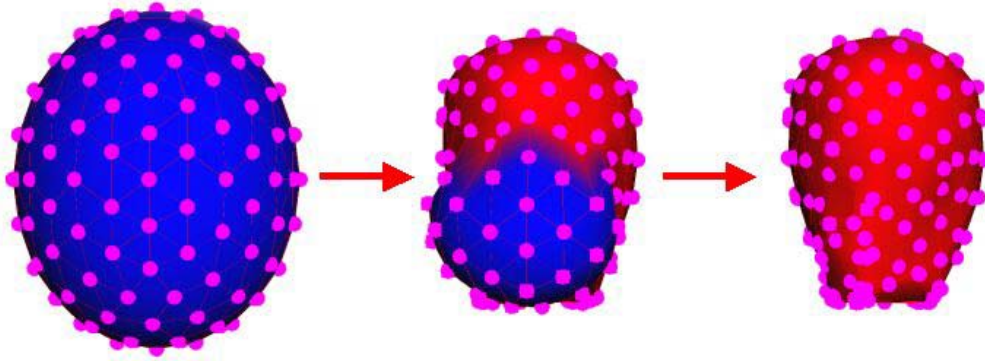


Figure 3.1: Particles reach their position of equilibrium after several iterations.

### 3.1.1 Particle systems

Particle systems are totally related to Newtonian schemes. Because of their simplicity, particles are especially suitable for modeling, simulation and animation. We can specify very interesting behaviors for particle-oriented objects, especially in the case of deformable models. As an example we might say that it is possible to build an object by connecting several particles with simple damped springs. Then we can apply forces for all the particles and let the physical laws do the rest of the job.

Particles have mass, position, and velocity and they should react according to some existing forces. Then particle simulation involves the particles themselves and the entities that apply forces to them.

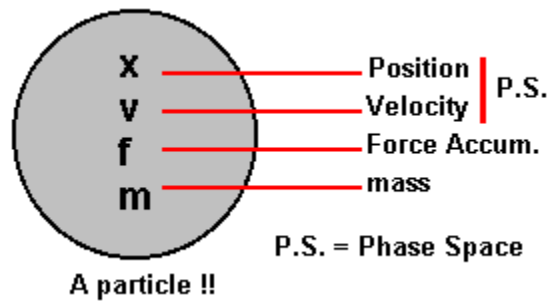


Figure 3.2: Particle attributes.

Figure 3.2 states that a particle is identified by several physical parameters like its position-velocity (named phase space as next section explains) at every instant of time, and its mass (a constant value). It is also important to define a force accumulator in order to add all the forces that act over that particle.

The concept of particle system can be easily associated to the concept of geometrical mesh, as described in chapter 5. It is common to create several particle systems associated to all the different entities that we need to simulate. Then a particle system is formed from a set of particles, connected or not. A general particle system structure might be defined as follows:

- Id.
- Total quantity of particles involved.
- Total quantity of springs involved (if it applies).

- Total quantity of existing forces.
- List of particles.
- List of springs (if it applies).
- List of forces.
- Starting time (optional because this is a parameter related to the dynamic scheme, see chapter 4).
- Stopping time (optional because this is a parameter related to the dynamic scheme, see chapter 4).
- Step size (optional because this is a parameter related to the dynamic scheme, see chapter 4).

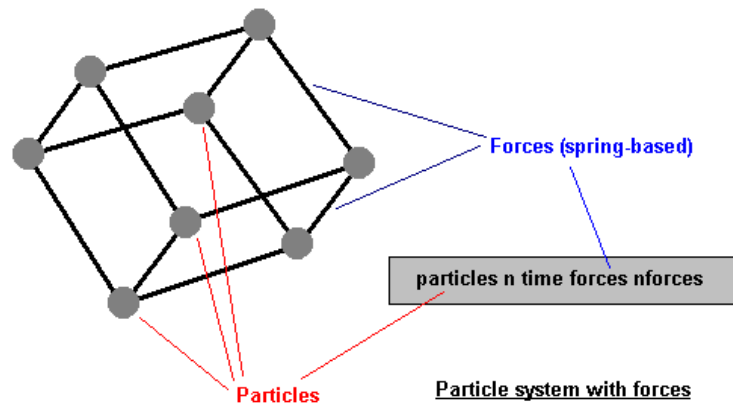


Figure 3.3: A complete particle system.

Figure 3.3 shows a spring-based particle system (see section 3.4.3) where the particles are linked by springs that act as a constraining force.

As we present extensively in chapter 4, simulating a particle system involves the implementation of a numerical scheme that begins the simulation for the given start time and iterates until it reaches a given condition. Each step takes time after step size seconds. All the particles vary their associated phase space vector at each step according to the existing forces that act on them.

### 3.1.2 Phase Space

The Newton law states that a particle's acceleration can be obtained from its associated mass, plus the accumulated force acting on it. This idea can be expressed by a second time derivative as stated in equation 3.1. To handle a second order ordinary differential equation we can convert it to a first-order one by adding some extra variables, as the velocity  $v$ , for instance. In that case we get a couple of new equations:

$$\begin{aligned} v' &= f/m \\ x' &= v \end{aligned} \tag{3.2}$$

For a given particle, we can concatenate its position and velocity in a vector (both magnitudes have three components in a three-dimensional space). Then this vector will be formed by six components. This vector is called phase space [103] and can be expressed like equation 3.3 states:

$$\left[ \dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{v}_1, \dot{v}_2, \dot{v}_3 \right] = \left[ v_1, v_2, v_3, \frac{f_1}{m}, \frac{f_2}{m}, \frac{f_3}{m} \right] \quad (3.3)$$

A system with  $n$  particles can be described with  $n$  vectors like the one shown in equation 3.3. If we concatenate all of them we will have a  $6n$ -dimensional phase space vector describing all the entire system. We could think in simulating a huge particle-oriented object as a point moving through a  $6n$ -space.

We will have to recalculate all the values for this vector at each new iteration. All the included positions and velocities will vary depending on the existing forces and interactions.

### 3.2 Internal forces

Internal forces control the inner cohesion and stability of the model. For a given topology, we have several connections between particles that must ensure the stability of the solution.

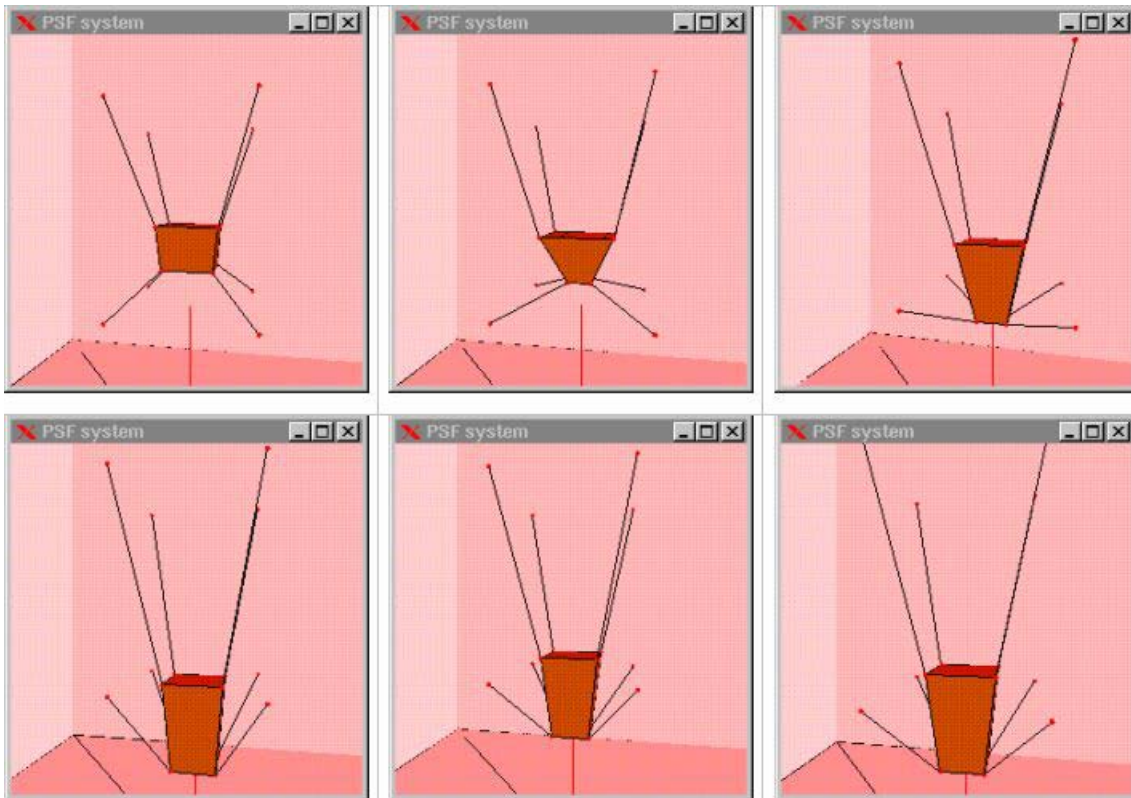


Figure 3.4: A deformable cube [34].

Figure 3.4 shows a cube modeled like a deformable entity. In that sense, the cube is formed from eight vertices linked by 12 springs and rendered like 12 triangles. The cube hangs from eight fixed vertices that act as anchors for the system. In that simulation the internal forces are represented by the springs.

We see that the cube gets deformed due to the action of the gravity (an external force in that case) and the soft characteristic of the springs. Strong springs would avoid the deformation and produce a different final simulation where the cube would act more or less as expected if it was a rigid object. Then it is clear that the springs are acting as the internal forces that define the behavior of the particle system in terms of elasticity.

In a general framework of planar deformation, three internal forces must be taken into account: stretch, shear and bend are introduced in the following subsections.

### 3.2.1 Stretch force

The stretch force [7] controls the magnitudes associated to the sides of a given triangle. We have several triangles in a mesh, that can be characterized by three vertices (or particles). We consider a map  $W$  between a local and a world coordinate system as figure 3.5 suggests.

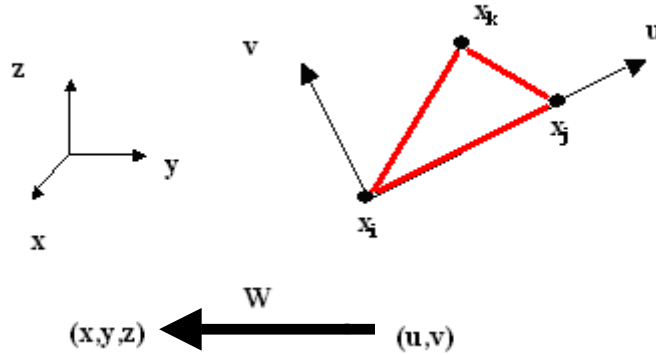


Figure 3.5: Local coordinates for a triangle.

From figure 3.5, it can be easily demonstrated that:

$$\begin{aligned}
 \Delta x_1 &= x_j - x_i & \Delta x_2 &= x_k - x_i \\
 \Delta u_1 &= u_j - u_i & \Delta u_2 &= u_k - u_i \\
 \Delta v_1 &= v_j - v_i & \Delta v_2 &= v_k - v_i
 \end{aligned}
 \tag{3.4}$$

Then a correct lineal approximation for the reference's change can be:

$$\begin{aligned}
 \Delta x_1 &= W_u \cdot \Delta u_1 + W_v \cdot \Delta v_1 \\
 \Delta x_2 &= W_u \cdot \Delta u_2 + W_v \cdot \Delta v_2
 \end{aligned}
 \tag{3.5}$$

And the derivatives are:

$$(W_u \quad W_v) = (\Delta x_1 \quad \Delta x_2) \begin{pmatrix} \Delta u_1 & \Delta u_2 \\ \Delta v_1 & \Delta v_2 \end{pmatrix}^{-1}
 \tag{3.6}$$

With this system it is possible to formulate two conditions that will prevent both sides,  $\Delta x_1$  and  $\Delta x_2$ , from deformations:

$$C(x) = a \begin{pmatrix} \|W_u(x)\| - b_u \\ \|W_v(x)\| - b_v \end{pmatrix} \quad (3.7)$$

We obtain the increments and decrements of every edge's magnitude by using the differences in position between the two related particles.

The definition of  $C(x)$  ensures that the stretching is proportional to the triangle's area  $a$ , with a typical value of 1 for the constants  $b_u$  and  $b_v$ . This factor implies that the restriction will try to maintain every edge's magnitude as it was at the beginning of the simulation. Moreover that, it is possible to weight the restriction's importance within the global term, by using a constant parameter. This parameter must be adjusted depending on the final elasticity desired for the system (see section 3.4.2 forward in the text).

### 3.2.2 Shear force

This force [7] acts on the inner triangle's angle. As we can state from figure 3.6.

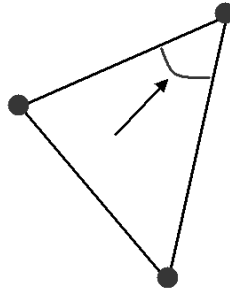


Figure 3.6: Internal shear force.

$$C(x) = a W_u(x)^T W_v(x) \quad (3.8)$$

Figure 3.6 and equation 3.8 show that the inner angle is defined between vectors  $W_u(x)$  and  $W_v(x)$ . The restriction tries to maintain its value as orthogonal as possible, avoiding then massive degenerations.

### 3.2.3 Bend force

It controls the bending movement on the mesh surface [7]. This force is defined between pairs of adjacent triangles, as figure 3.7 shows.

Now the restriction tries to force the angle between both sides to be inside a concrete interval. The normal vectors  $n_1$  and  $n_2$  and the shared edge's direction  $e$ , are used for its computation. The restriction is just the angle's value that we need to maintain.



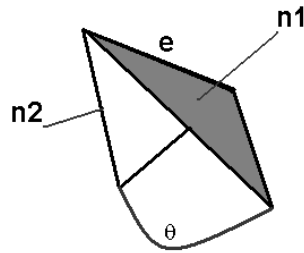


Figure 3.7: Internal bending force.

$$\begin{aligned}
 \sin \theta &= (\mathbf{n}_1 \times \mathbf{n}_2) \cdot \mathbf{e} \\
 \cos \theta &= \mathbf{n}_1 \cdot \mathbf{n}_2 \\
 C(x) &= \theta
 \end{aligned}
 \tag{3.9}$$

### 3.2.4 Local curvature force

Those forces are related to the minimization of the local curvature in order to balance the action of the external forces that model the contour to follow all the variations that the energy related to the image demands [3, 54]. It is important to note that in this case we are dealing with curves while in the previous subsections we were defining forces related to triangulated surfaces.

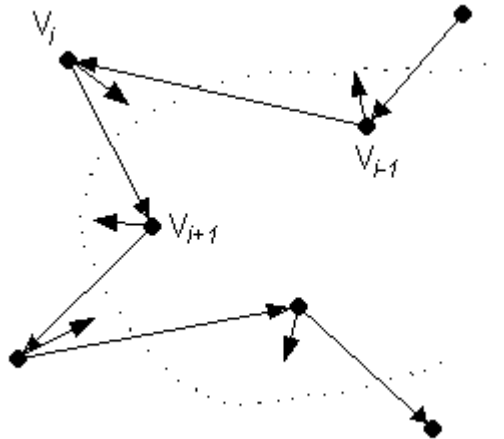


Figure 3.8: Local curvature forces.

Let's begin by defining the local curvature concept in a discrete model. The local curvature must tend to zero when traversing a segment between a pair of vertices. However it is not clear how to define the curvature in the vertices themselves because in these points, we do not have a derivative due to a first order discontinuity.

An approximated solution might be to define the curvature as the difference between the directions associated to the edges that share this vertex. As figure 3.9 states, vector  $d_i$  stands for the segment between two consecutive vertices and its direction is perfectly defined by the unit vector  $du_i$ .

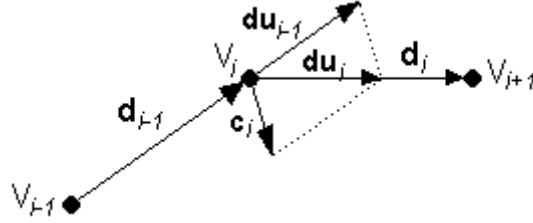


Figure 3.9: Local curvature  $c_i$ .

According to the previously cited definition, the local curvature  $c_i$  in  $V_i$  is described by (see figure 3.9):

$$c_i = du_i - du_{i-1} \quad (3.10)$$

Then the local curvature is a vector (with magnitude and direction) that allows us to derive the angle between both edges. In fact, the vector's magnitude only depends on this angle; it is never related to the magnitudes of the edges that share the vertex.

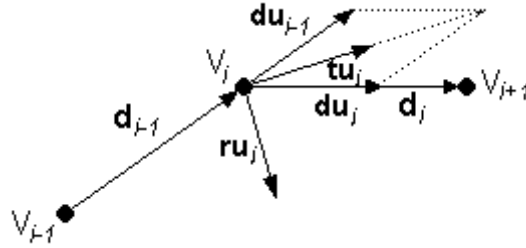


Figure 3.10: Tangential and Radial unit vectors  $tu_i$  and  $ru_i$ .

The method also defines the tangential and radial directions local to the vertex position. In order to get those, we can use the unit vectors  $du_i$  that represent the directions associated to the edges ( $d_i$ ). Defining the tangential unit vector  $tu_i$  implies using the normalized sum of the unit vectors associated to the edges that share a vertex (equation 3.11).

$$tu_i = \frac{du_i + du_{i-1}}{\|du_i + du_{i-1}\|} \quad (3.11)$$

The unit and radial vector  $ru_i$ , equals the vector  $tu_i$  after a rotation of  $\pi/2$  radians:

$$ru_i = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot tu_i \quad (3.12)$$

The vectors  $tu_i$  and  $ru_i$  represent a local coordinate system in the position of the vertex  $V_i$ . This coordinate system can be very helpful when evaluating the internal and external forces applied at that position.

This paradigm is suitable for open and closed contours. Let's assume that the amount of vertices is  $n$  and that we are working with a closed contour. The first vertex  $V_0$  and the last one  $V_{n-1}$  are connected so that  $V_0$  is surrounded by two neighbors:  $V_1$  and  $V_{n-1}$ . If only the contour was opened,  $V_0$  and  $V_{n-1}$  would not be connected and they would be

surrounded by a single neighbor ( $V_1$  and  $V_{n-2}$  respectively). This situation requires a special treatment in order to evaluate  $ru_i$ ,  $tu_i$  and also  $c_i$  in these points.

In the positions associated to the first and last vertices, we define the locally tangential direction as the direction of the first or last edge, respectively. It means that:

$$tu_0 = du_0 \tag{3.13}$$

And

$$tu_{n-1} = du_{n-2} \tag{3.14}$$

The longitude of the curvature vector will tend to zero for both sides:

$$c_0 = c_{n-1} = 0 \tag{3.15}$$

If we describe the curvature vector  $c_i$  in terms of the local coordinate system  $(r, t)$ , we can observe that  $c_i$  is parallel to  $ru_i$  (the direction can be the same or opposite). Then  $c_i$  is a vector contained in the  $r$  edge of the local coordinate system and its magnitude can be described by the dot product  $c_i \cdot ru_i$ .

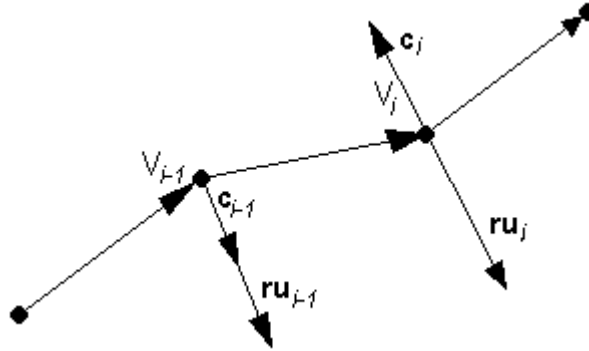


Figure 3.11: Positive and negative local curvatures.

According to this definition, the magnitude  $c_i$  of the curvature vector can be positive or negative (figure 3.11).

$$c_i = (c_i \cdot ru_i) \cdot ru_i \tag{3.16}$$

Now that the local curvature has been defined as a unique dimension variable in the local coordinate system, let us define the internal forces that will restrict the deformation process. In order to understand clearly the contribution of such forces, let us consider a situation where there's no external force applied. If we take a look at the previous figures we can figure out that the internal forces and the curvature vectors should have the same orientation. Nevertheless, it is not as easy as defining the internal forces to be equally proportional to the curvature vectors.

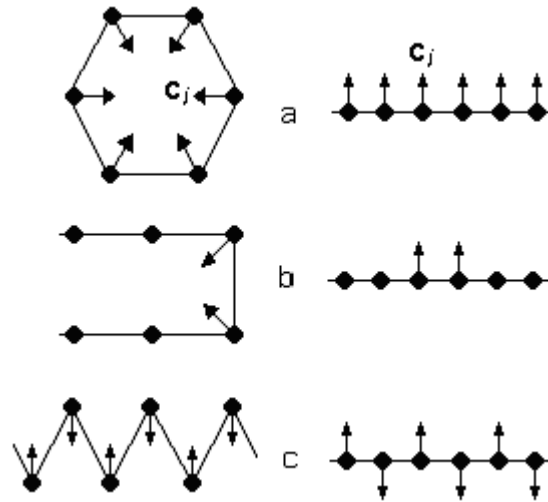


Figure 3.12: Curvature vectors  $c_i$  for some typical situations.

We can understand that by examining the left images in figure 3.12, any closed shape will deform until getting the minimal curvature at every point, in absence of external forces. That means that the final shape will be close to a symmetric polygon, similar to a circle if it is formed from enough vertices.

As soon as we get to this situation, the deformation process would never stop. The vertices would keep moving to the geometrical center, as figure 3.12(a) shows. The shape would shrink until being reduced to a single point.

Miller et al [66, 67, 68] studied the same problem and solved it by introducing a second force based on a stretching spring that maintained the distance between consecutive vertices within some predefined values. As explained in section 3.4.1, vertices too close receive a repulsion force that stops the shrinking. The stopping depends on the weighting factors associated to the internal forces. Then it can be seen that we have a tradeoff when balancing the elastic force against the local curvature force.

Unfortunately, this situation is not always possible in presence of external forces. Then in order to avoid the introduction of a second internal force and the tradeoff associated to the balancing of the weighting factors, another solution can be adding a modification to the local curvature force. We must also take into consideration that the internal force must not reduce the local curvature if it remains constant by itself.

Figures 3.12(b) and (c) show some typical shapes that we might get in a discreet contour model. Figure 3.12(b) represents a contour that rotates  $\pi$  radians and figure 3.12(c) shows an example where the curvature values are alternated. The problem that arises if we have the situation of figure 3.12(a) has been previously documented. The shape will reduce to a single point in the absence of external forces.

Figure 3.12(b) presents a similar situation. If we apply a force proportional to the curvature of its vertices, we will get the reduction of the vertical segment so that the local curvature will remain intact.

Figure 3.12(c) does not present any problem and it has been included because it presents a typical situation where the local curvature must be reduced. Any solution to the

problem of the shrinking should not affect the correct operation in contours where we need to reduce the curvature.

Right side of figure 3.12 shows the same shapes but expressed in the local coordinated system  $(r, t)$ . We observe that:

- The internal forces  $f_{in}$  that are applied over the vertices  $V_i$  should have the same radial direction that the curvature vectors. That implies that the internal forces can be obtained from the curvature vectors by modifying their magnitude.
- In order to introduce the local curvature while not affecting the constant curvature areas, the magnitudes of the internal force vectors must be zero in those areas.

We can accomplish both if we consider the sequence  $c_i \cdot r_{ui}$  along the contour as a scalar discrete function that depends on the position  $i$ , and we use the convolution of this function with the discrete filter  $k_i$ . This filter stands for a representation of a sequence of the magnitudes associated to the vectors of the internal force  $f_{in,i}$ :

$$f_{in,i} = (c_i \cdot r_{ui}) \otimes k_i \quad (3.17)$$

The first condition gets accomplished if we take  $r_{ui}$  as the direction of  $f_{in,i}$ :

$$f_{in,i} = f_{in,i} \cdot r_{ui} \quad (3.18)$$

The second condition depends on the accurate selection of the  $k_i$  filter coefficients. If we need that the convolution with a constant sequence equals zero, we have to define  $k_i$  so that it blocks the continue component. There are several filters that accomplish this condition. A simple one can be a uniform filter where three coefficients are different from zero:

$$k_i = \{\dots, 0, 0, -1/2, 1, -1/2, 0, 0, \dots\} \quad (3.19)$$

Where a value of 1 is applied to the  $i$  position and the value  $-1/2$  stands for the positions  $i-1$  and  $i+1$ . It must be said that the optimum filter for this application might be adaptive.

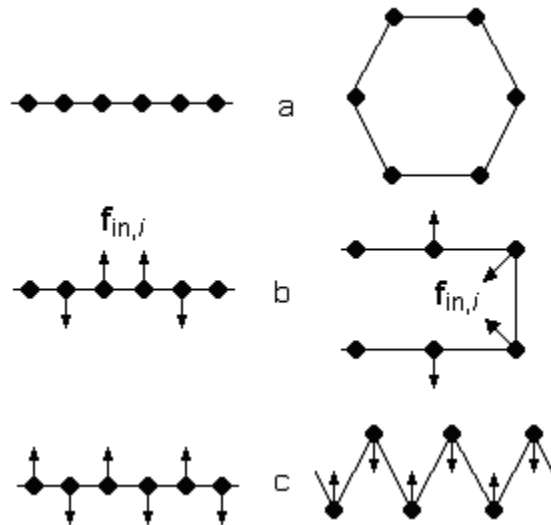


Figure 3.13: Internal forces  $f_{in,i}$ .

Figure 3.13 shows the internal force vectors related to the shapes of figure 3.12. The left side of the figure shows the internal forces in local coordinates  $(r,t)$ ; the right side shows the shapes and internal forces in Cartesian coordinates. The constant curvature of figure 3.13(a) induces that the internal forces tend to zero in all the vertices so that the shrinking problem gets solved. The rotation of  $\pi$  radians in figure 3.13(b) will not be deformed so that it will appear more natural. The closed ending of this contour, where the rotation takes place, will get shortened and wider due to the curvature reduction. The alternated curvature of figure 3.13(c) will be also shortened effectively because of the convolution. The result from the convolution of the filter  $k_i$  with an alternate signal is the same signal multiplied by a constant.

As a conclusion, the definition of the internal forces  $f_{in}$  produces, in absence of external forces, the deformation effects desired.

### 3.3 External forces

External forces attract particle systems to the data set. The simulation must be correctly balanced in terms of internal against external forces. Both terms will lead the system to an equilibrium situation that should coincide with the solution that we are looking for.

This section describes the different existing external forces according to their definition characteristics.

#### 3.3.1 Snakes

The snake model was created as a first attempt to describe a 2D parametric deformable model [46] suitable for several applications like edge detection, segmentation or motion tracking. This model or contour moves inside an image domain trying to minimize the energy functional of equation 3.20.

$$E = \int_0^l \frac{1}{2} \left[ \alpha |x'(s)|^2 + \beta |x''(s)|^2 + E_{ext}(x(s)) \right] ds \quad (3.20)$$

In equation 3.20,  $x(s)$  is the contour or model,  $s$  is its associated parameter and belongs to the interval  $[0, 1]$ ,  $\alpha$  and  $\beta$  are the weighting constants that control tension and rigidity respectively and  $x'(s)$  and  $x''(s)$  stand for the first and second derivatives of  $x(s)$  with respect to  $s$ .

The  $E_{ext}$  term stands for the external potential function and it is derived from the image. There exist several possibilities for this term. Some are developed next, in equations 3.21 to 3.24.

$$E_{ext}(x, y) = -|\nabla I(x, y)|^2 \quad (3.21)$$

$$E_{ext}(x, y) = -|\nabla(G_\sigma(x, y) * I(x, y))|^2 \quad (3.22)$$

$$E_{ext}(x, y) = I(x, y) \quad (3.23)$$

$$E_{ext}(x, y) = G_{\sigma}(x, y) * I(x, y) \quad (3.24)$$

Each of the energies is used depending on the concrete scenario.  $I(x, y)$  stands for the image property value on every pixel (voxels of a data slice if working in 3D),  $\nabla$  is the gradient operator and  $G_{\sigma}(x, y)$  is a Gaussian function with a standard deviation  $\sigma$ .

Equations 3.21 and 3.22 are especially suitable for gray level images while equations 3.23 and 3.24 are best fit for black and white scenarios.

Minimizing the functional of equation 3.20, implies satisfying the Euler equation 3.25.

$$\alpha x'(s) + \beta x''(s) - \nabla E_{ext} = 0 \quad (3.25)$$

Which can be compared to an equilibrium equation where tension and rigidity are treated like internal forces in front of the external energy potential.

Equation 3.25 is solved by adding the time variable  $t$  to the system. Then it can be treated by a dynamic scheme that discretizes the equation and solves it iteratively.

Snakes have three main disadvantages. First of all, the contour has to be initialized very close to the data to recover. Secondly, it has difficulties when trying to follow image maps that contain boundary concavities, like in the left ventricle case. The third problem is related to our final goal, the recovering of the inner and outer surfaces of the left ventricle. Isquemic areas, which mean in absence of blood irrigation, do not appear in the images. This effect provokes the apparition of holes that are an added problem for the contour or surface.

### 3.3.2 Active nets and Topologic active nets

Active nets can be described as bidimensional snakes. Active nets are bidimensional parametric representations moving in a bidimensional space. Then its representation can be derived from the one-dimensional snake formulation, extending it to the bidimensional space.

$$v(x_1, x_2) = [y_1(x_1, x_2), y_2(x_1, x_2)]^T \quad (3.26)$$

$$E(v) = \int_0^1 \int_0^1 E_{int}(v(x_1, x_2)) + E_{ext}(v(x_1, x_2)) dx_1 dx_2 \quad (3.27)$$

As stated in equations 3.26 and 3.27, active nets are characterized by an energy function that describes the equilibrium between internal and external energies.

It is necessary to filter the image in order to ease the localization of objects. The information about borders must be maintained in order to allow the shape adjustment. This tradeoff can be reached by classifying the nodes between internal and external and treating them differently in terms of the image function applied [13].

Let us assume that the interesting features are characterized by low gray levels and that the background tends to have high gray levels. Then the image function that finally defines the external force, can be defined separately for the internal and external nodes.

$$f[I(v(x_1, x_2))] = \begin{cases} h[I_n(v(x_1, x_2))], & \text{internal nodes} \\ h[I_{max} - I(v(x_1, x_2))], & \text{external nodes} \end{cases} \quad (3.28)$$

where  $I_{max}$  is the maximum intensity value;  $I(v(x_1, x_2))$  is the intensity value for the node  $v(x_1, x_2)$ ;  $h$  is a normalization function and  $n$  is the window size for the average evaluation.

Then the internal nodes will minimize their energy when being inside the object while the external nodes will perform the same behavior when positioned in the background. This process allows to track the objects by the internal nodes and to stop at the borders due to the external ones.

Active nets work when objects are quite simple but some problems arise when irregularities exist, like large concavities. In order to solve those complex cases, topologic active nets have been introduced [4].

Topologic active nets provide with:

- A first initialization step where the object is located and its shape detected in order to automatically center and dimension the initial net. This step allows obtaining a uniform distribution of the internal nodes inside the object that offers valuable information about its internal structure. This information can be used in later processes to detect internal concavities.
- A second step where the net is topologically altered in order to fit the object perfectly.

Those improvements make the topological active nets suitable for locating more complex shapes, tracking several objects simultaneously and detecting concavities in the internal structure of the objects.

In the first active nets, the external nodes act as anchors that restrict the movements of the net to the interior of the object. With fixed external nodes, the mobility of their internal neighbors is limited. Moreover that, the internal distribution of nodes is totally conditioned by the instant in which the external nodes reach the borders. In order to avoid this effect, it would be desirable that all the external borders reach the object's contour nearly simultaneously. That might be possible if the initial net is centered with respect to the object and if the distance between nodes is equal in both dimensions.

Using a first sparse net that covers the entire image solves this problem. This net moves freely and detects all the objects, their positions and their approximate dimensions. Then this information is used for associating a dense net adapted to every object in the image.

If the object is characterized by irregularities or concavities or even if we have several objects in the image, the complexity of the process increases. It could be treated by



locally modifying the topology of the net when necessary. In other words, being able to break links between external nodes. Breaking links in difficult areas means more external nodes that stand for a better adjustment.

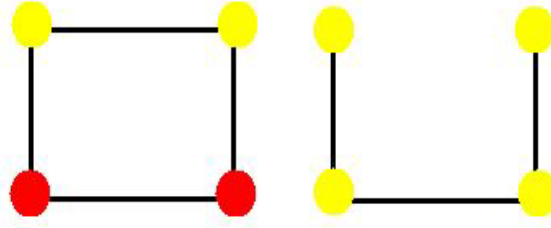


Figure 3.14: Breaking links and defining new external nodes.

Figure 3.14 shows a typical situation where two internal nodes (red) change into external nodes (yellow) when a link is broken.

Breaking links implies detecting first the external nodes that are poorly positioned. This detection can be achieved by the gradient distance evaluation limited to the directions that point inside the net. A new term can be added that certainly describes the quality of the adjustment, as equation 3.29 shows.

$$f[I(v(x_1, x_2))] = \begin{cases} h[I_n(v(x_1, x_2))], & \text{internal nodes} \\ h[I_{max} - I(v(x_1, x_2))] + \xi[G_{max} - G(v(x_1, x_2))], & \text{ext. nodes} \end{cases} \quad (3.29)$$

The new term represents the image gradient.  $G_{max}$  is the maximum gradient value,  $G(v(x_1, x_2))$  is the gradient value in the position of the node  $v(x_1, x_2)$  and  $\xi$  is a weighting factor that balances gradient and intensity information.

With this new term the external nodes vary their behavior. When located in the background, far from the borders, their energy function will be dominated by the intensity term and their behavior will be as before. Nevertheless, when approaching the borders of the image, the gradient term will increase its effect making the external nodes to be attracted by the high gradient values of the image. Then the external nodes will lie on the borders and not only close to them.

When positioned, the external nodes can be classified depending on their gradient distance. Distances near zero will characterize well-positioned nodes while distances over a defined threshold will be symptom of an incorrect position.

Once the external nodes have been labeled, it is necessary to select a link to be broken. The selected link must maximize the gradient distances sum of all the labeled nodes that form part of it. After breaking the link, the net evolves until reaching a new stable state.

It is important to note that the breaking process helps on the smoothness of the solution but also when an image presents several objects to be recovered. If we use a single initial net with an image formed from several objects, it will be automatically broken as explained, until being split in independent nets, one per object in the scenery.

As a final feature of the topological active nets algorithm, we can describe its ability when finding internal concavities. These tasks can be achieved by accessing the information that the internal nodes store about the internal distribution of the object.

When an internal node is located in a concavity, it presents high energy values compared to those nodes located over the object. By analyzing these energies, especially those relatively high, it is possible to obtain information about the concavity presence and situation.

Once the concavity has been detected, the inner behavior of the net performs the correct adjustment by breaking links if necessary. The internal nodes inside the concavity evolve to be external nodes that must adjust the borders. Those new external nodes must expand, not contract, and this is achieved by a new energy term. See equation 3.30 for more details.

$$f[I(v(x_1, x_2))] = \begin{cases} h[I_n(v(x_1, x_2))], & \text{internal nodes} \\ h[I_{max} - I(v(x_1, x_2))] + \xi[G_{max} - G(v(x_1, x_2))], & \text{ext. nodes} \\ h[I_{max} - I(v(x_1, x_2))] + U[Dist_{min} - G(v(r, s))], & \text{conc. ext. nodes} \end{cases} \quad (3.30)$$

Where the first row stands for internal nodes, the second row for external nodes and the third row for external nodes related to concavities.  $U[Dist_{min} - G(v(r, s))]$  stands for the thresholded gradient distance image. This new term guides the new external nodes to the borders of the concavity, by expanding themselves.

Topologic active nets can be useful when tracking objects in an image. Unfortunately, this strategy might fail our purposes of tracking the entire left ventricle despite it is formed from an object or from several of them. These nets will track several apparently unrelated objects separately and we need to group them. Moreover that, the 3D reconstruction method described in section 3.1 depends on the calculation of an external force related to external and internal borders. This classification is not provided by this methodology.

### 3.3.3 Radial energy based potential

When working with local curvature internal forces, like those explained in section 3.2.4, a suitable external force can be derived from a distribution of an external potential energy [3, 54]. This distribution can be associated to several characteristics of the image. One of the characteristics that best fits a good behavior, is the gray level and the gradient magnitude at every pixel (or voxel if working in a 3D framework).

If we need the model to follow the maximal gradient path through the image, we can use its magnitude and define an energy distribution proportional to its value. The implementation of the deformation process will try to drive all the vertices through the minimal energy areas. Moreover that, it implies that the way to follow is described by minimal energies (valleys) so that we must invert the potential energy distribution.

The force field can be described like equation 3.31 shows.

$$f_{im} = -\nabla E_{im} \quad (3.31)$$

If we apply this force to all the vertices in the model and we consider no internal forces at all, the final result will consist on the contour connecting points with local energy minima, following a valley through the external energy distribution.

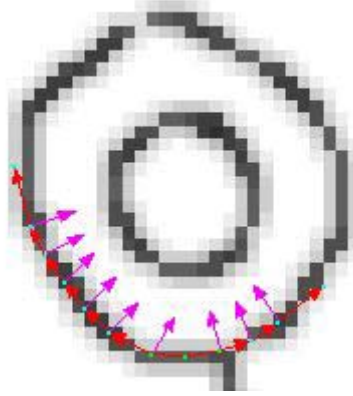


Figure 3.15: Components of the external force over the contour.

Besides that, we might encounter that the force  $f_{im}$  does not only have an orthogonal component to the local direction of the contour, but a tangential component that follows the contour trajectory (figure 3.15). This component might be important in terms of magnitude. If no restrictions are applied in the curvature, for instance when no internal forces exist, the final situation might be characterized by the apparition of clusters, as depicted in figure 3.16.

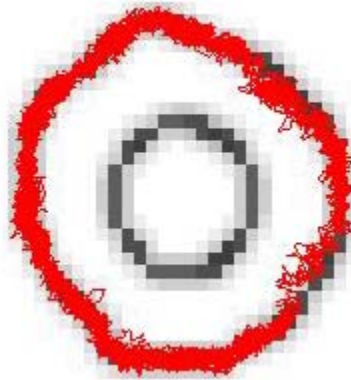


Figure 3.16: Clustering phenomena.

As a first seek to the problem it is possible to add a stretching force that avoids the vertices to be too close from each other, avoiding then the clustering. This force must be adjusted locally for each vertex depending on the external force at its position. We must take into consideration that the external force is different in every pixel of the image. Moreover that, it changes from a slice to another. Besides that, the external force derivation is related to the acquisition procedure and to the posterior processing that can be applied (filtering). Those reasons make the adjustment of the stretching force a critical process. The stretching force should not be too weak (because it would not act as a restriction) or too strong (because it would alter the deformation process).

Unfortunately this local tuning results in a very complex problem that drives us to find another solution: using only the radial component of the external force  $f_{im}$ . The basic reason for doing so is that the vertex displacement along the path that the contour follows does not contribute to the deformation of the model in order to capture the real data.

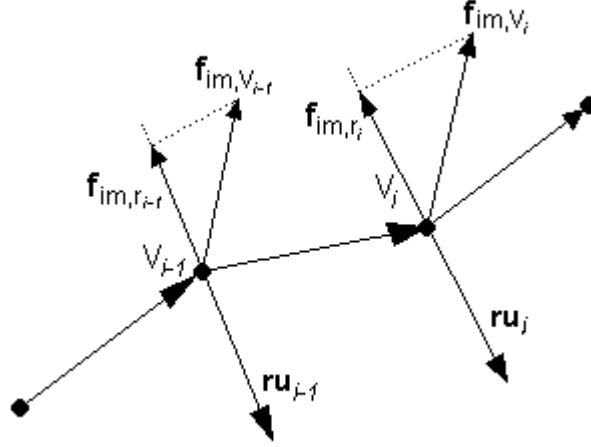


Figure 3.17: Radial component of the external force.

If we denote this radial component of  $f_{im, V_i}$  as  $f_{im, ri}$ , we can see that its magnitude can be evaluated by the dot product  $f_{im, V_i} \cdot r_{ui}$ .

$$f_{im, ri} = (f_{im, V_i} \cdot r_{ui}) \cdot r_{ui} \quad (3.32)$$

The sum of the radial local forces  $f_{im, ri}$  and  $f_{im, i}$  gives the resultant force that accelerates all the vertices through the contour while making them move radially.

In terms of energy, it might be useful to allow the user to exert some kind of control over the contour. That might be possible by defining a second external energy distribution called  $E_{user}$ . This distribution would be added to the first defined energy and would allow the user to alter the energetical landscape by defining new valleys, for instance.

$$E_{ex} = E_{im} + E_{user} \quad (3.33)$$

$$f_{ex} = -\nabla E_{ex} \quad (3.34)$$

$$f_{ex, ri} = (f_{ex, V_i} \cdot r_{ui}) \cdot r_{ui} \quad (3.35)$$

Then the generation of the external associated force would be identical, as stated by equations 3.33 to 3.35.

This paradigm has been used for the tracking of the left ventricle, as presented in chapter 6. However there is still the need to find an external energy term that attracts the contour or the surface to the original data. That's the reason for using the vector field described in next section.

### 3.3.4 Gradient vector flow

The Gradient vector flow [104] is a vector field where the external force consists on the minimization of a functional that mixes the information derived from the image-intensities gradient with a diffusion term that allows the field to be spread out.

This vector field tries to solve two key difficulties related to parametric deformable models:

- Avoiding the need for the initial model to be close to the data to recover. Usual systems are likely to diverge into wrong results if the initialization is far away from the data. The GVF increases the capture range of the external force field so that it is able to guide the model towards the desired position.
- Solving the classical difficulties when progressing into boundary concavities.

The vector field that acts as the external force is obtained by minimizing the functional in equation 3.36.

$$\varepsilon = \iiint \mu \sum_{i \in \{x, y, z\}} (u_i^2 + v_i^2 + w_i^2) + |\nabla I|^2 |V - \nabla I|^2 dx dy dz \quad (3.36)$$

The functional consists on two well-differentiated terms. On the left the diffusion term that spreads the field when variations on intensities are negligible. On the right the property term, that domains the expression if variations are important. The  $\mu$  parameter will control the balance between both terms.

In fact, minimizing the functional in equation 3.36, can be translated into solving equation 3.37.

$$\alpha x'(s, t) + \beta x''(s, t) + v(x) = 0 \quad (3.37)$$

Where the time variable  $t$  has been added to the system and  $v(x)$  stands for the GVF vector field instead of the energy term in section 3.3.1. The vector field can be derived solving the Euler equations of equation 3.38.

$$\begin{aligned} \mu \cdot \nabla^2 u - (u - I_x)(I_x^2 + I_y^2 + I_z^2) &= 0 \\ \mu \cdot \nabla^2 v - (v - I_y)(I_x^2 + I_y^2 + I_z^2) &= 0 \\ \mu \cdot \nabla^2 w - (w - I_z)(I_x^2 + I_y^2 + I_z^2) &= 0 \end{aligned} \quad (3.38)$$

Taking a glance at the equations allows us to infer that in low-variation areas the second term becomes negligible due to the null-value of the gradient.

The numerical implementation of this external force uses a finite-differences scheme complemented with an Explicit-Euler method that follows the temporal evolution. Details on its implementation can be found in chapter 4.

For the left-ventricle reconstruction, the property value is selected to be the border attribute. Given a data slice, 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 field that surrounds the borders, making them act as attractors.

The field will act as the external force. Figure 3.18 resumes this behavior.

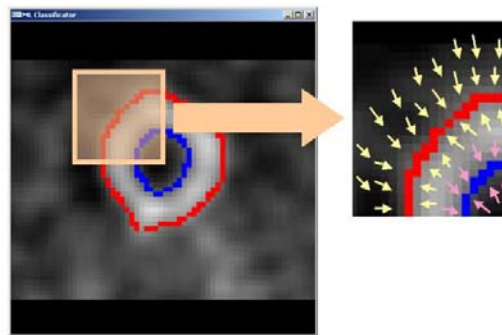


Figure 3.18: Graphical representation of the behavior for the GVF field.

Note that figure 3.18 represents the field in 2D. Our final implementation finds it in a 3D framework (voxels but not pixels) making it more robust and confident with our 3D data sets.

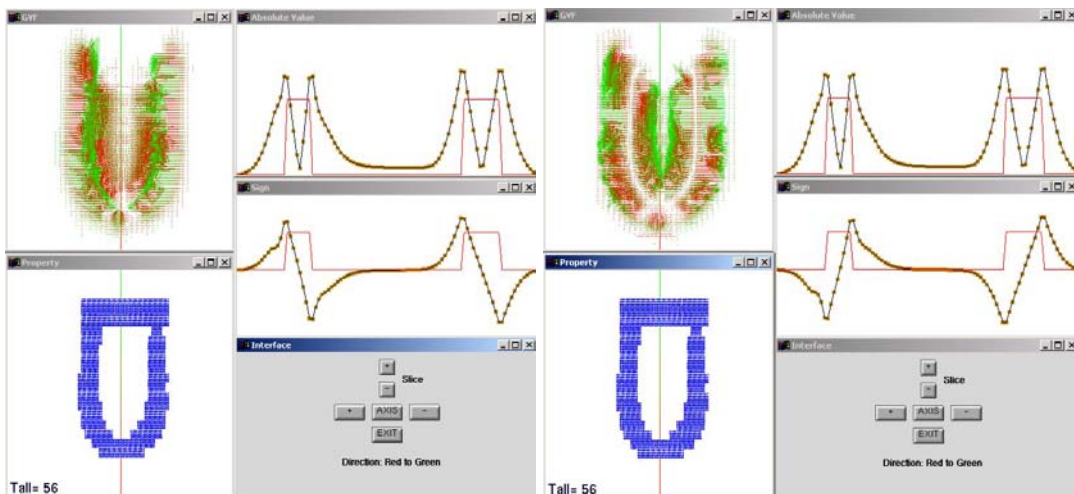


Figure 3.19: Correct (left) and wrong (right) GVF vector fields.

Figure 3.19 shows two GVF representations in 3D. The initial data set is exactly the same but the fields were computed with a difference in sign. The vectors are rendered in red-green, the data set is the blue world of voxels and the magnitudes and signs for the vector field are shown in the top right viewports (for a given trajectory marked as a thin vertical line in the left). The intensity values are marked as a red thin line (right

viewports). Note the magnitude valleys and the changes in sign around them, in the top right viewports.

If we extend the analysis regarding the changes in sign and the operators used for the field evaluations, we obtain the results of figure 3.20. There six different cases are analyzed. All the tests were made over the same data set, a medical volume called Phantom (see Appendix D):

- Tests 1, 2 and 3 are quite similar. All of them use the gradient of the edge map as their input. Their differences are only related to the values of some of the parameters ( $\mu = 0.4$  for test 1,  $\mu = 0.2$  for tests 2 and 3; 200 iterations for tests 1 and 3, 100 iterations for test 2).
- Test 4 used the intensity value with its sign reversed, as its input. Regardless of the magnitudes, that seem correct, the sign of the derived field is wrong (it goes from inside to outside when it should be the contrary). The simulation took 100 iterations with  $\mu = 0.2$ .
- Test 5 is exactly the same as the previous but when using the original intensity with no changes in sign at all. Here the results are perfect in terms of magnitude and sign for the vector field.
- Test 6 is the reversed version of test 5. Here all the values have been reversed showing that the high gradient changes in intensity have become valleys instead of high peaks. Although it would be manageable in terms of magnitude, signs are not correct.

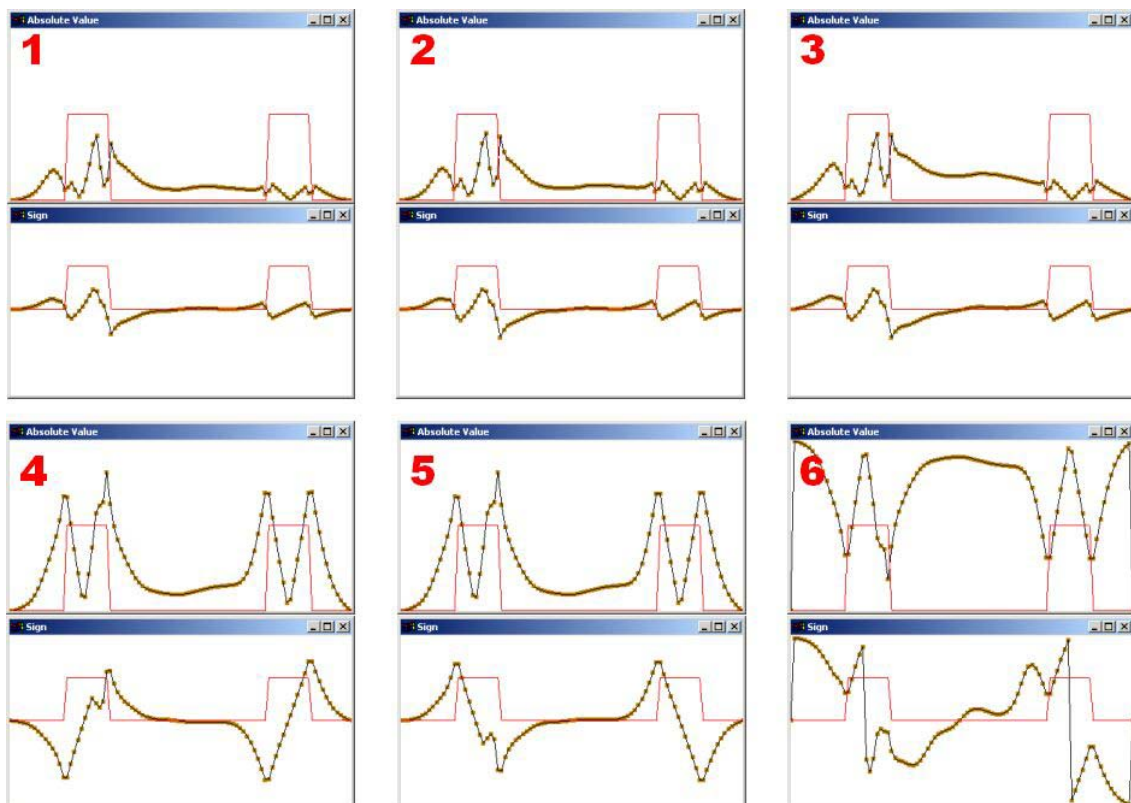


Figure 3.20: Different operators and signs for the GVF vector field evaluation.

It is important to note that depending on the final application, a variation of the derived field can be a good solution. Several strategies based on reversing values or tracking signs can be used in order to get different final results from the same initial vector field.

The same authors presented a generalization of the GVF vector field called Generalized GVF or GGVF [106]. This evolution tries to solve several difficulties that still appear although using the GVF vector field as the external force. It still has difficulties when used at thin boundary indentations. The main formulation differences between GVF and GGVF can be seen at equations 3.39 and 3.40.

$$v_{t-GVF} = \mu \nabla^2 v - (v - \nabla f) |\nabla f|^2 \quad (3.39)$$

$$v_{t-GGVF} = g(|\nabla f|) \nabla^2 v - h(|\nabla f|) (v - \nabla f) \quad (3.40)$$

In both equations,  $v_t$  is the partial derivative of  $v$  with respect to  $t$ ;  $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$  is the Laplacian operator and  $f$  is an edge map of the starting set of images or world of voxels (see that the process can be applied in 3D if we use several slices piled up).

In equation 3.40 we see that  $\mu$  and  $|\nabla f|^2$  have been substituted by the weighting functions  $g(|\nabla f|)$  and  $h(|\nabla f|)$ . Those functions apply to both terms in the equation, the smoothing and the data term. Because of the dependency of the weighting functions from the gradient of the edge map, this formulation ensures that the weights are spatially varying which means locally adjusted.

The weighting functions should ensure that the vector field is smooth when situated far from the edges but coherent with  $\nabla f$  when near them. This fact implies that  $g(|\nabla f|)$  must get smaller when  $h(|\nabla f|)$  gets larger. Then near large gradients the smoothing will be negligible allowing the vector field to be basically equal to the gradient of the edge map.

Many formulations are allowed for the weighting functions. The authors of GGVF proposed those shown in equations 3.41 and 3.42.

$$g(|\nabla f|) = e^{-(|\nabla f|/K)} \quad (3.41)$$

$$h(|\nabla f|) = 1 - g(|\nabla f|) \quad (3.42)$$

Those functions ensure that the vector field conforms to the gradient when edges are strong while varying smoothly if far away. The parameter  $K$  controls the tradeoff between both behaviors.

### 3.4 Deformable models

We have tested five deformable models within our application:



- **Discreet contour deformation model**, characterized by the weighting of the applied forces in order to ensure a correct stabilization.
- **Plain deformation model**, where each of the triangles in the mesh has its own elasticity forces. Deformation will be characterized by the added action of three forces: stretch, shear and bend.
- **Spring-mass deformation model**, where the only internal force is stretch, defined between pairs of particles following Hook's equation.
- **Restricted spring-mass deformation model**, where spring-forces are only allowed in the normal direction of the derived vector field.
- **Free deformation model**, where the only existing force is the external one, derived from the data set. There's no connectivity between particles and topology must be maintained using a smoothing algorithm (see chapter 6 for more details), apart from the evolution scheme.

The following paragraphs describe the quoted models.

### 3.4.1 Discreet contour deformation model

As partially introduced in sections 3.2.4 and 3.3.3, a discreet contour deformable model applies to 2D contours driven over imagery. However the method can be easily extended to 3D if using surfaces instead of contours.

The force that acts on a concrete vertex is a weighted combination of internal and external forces [3, 54]:

$$f_i = w_{ex} \cdot f_{ex,ri} + w_{in} \cdot f_{in,i} \quad (3.43)$$

Where  $f_{ex,ri}$  and  $f_{in,i}$  stand for the external and internal forces. The weighting factors are  $w_{ex}$  and  $w_{in}$ . If we emphasize the external forces, the model will follow the image characteristics with more precision. If we enlarge the internal forces, the final result will be clearly smooth. As a result of the forces, a vertex moves by Newtonian dynamics (see section 3.1) until it reaches the condition of stability, stated in equation 3.44.

$$a_i = v_i = 0 \quad (3.44)$$

In order to avoid oscillations between two minimal energy states, the model can be completed with a damping component  $f_{damp}$ , proportional to the vertex's velocity.

$$f_i = w_{ex} \cdot f_{ex,ri} + w_{in} \cdot f_{in,i} + f_{damp,i} \quad (3.45)$$

$$f_{damp,i} = w_{damp} \cdot v_i \quad (3.46)$$

Where  $w_{damp}$  stands for the weighting factor associated to the damping force and  $v_i$  is the vertex velocity. The damping weighting factor is negative in order to act in the opposite direction of the velocity.

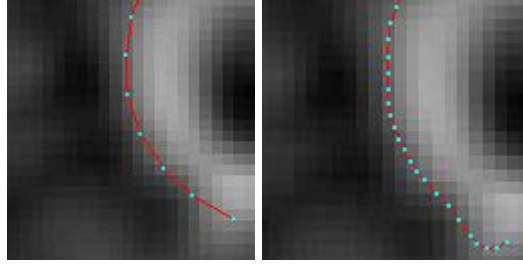


Figure 3.21: Two resolutions for the same contour.

In this method, the distance between a vertex and its neighbors defines the final resolution that it can get, as stated in figure 3.21. Tiny details of the external energy distribution can be lost between consecutive vertices if distances are too big. In order to avoid that, we need the model to be locally adjusted while not affecting its global state in an important manner. An approach that can be taken consists on allowing variations within certain limits by periodically resampling the contour.

Let's assume that  $l_{des}$  is the edge magnitude that we define to be the standard. From this value we extract  $l_{min}$  and  $l_{max}$  that represent the lower and upper limits that we permit. Then the resampling procedure consists on two steps:

- Firstly, we have to check if there's any edge whose magnitude is under the  $l_{min}$  value. If so, this edge has to be removed from the model in the sense that the pair of vertices that define it are substituted by only one (figure 3.22(a)).
- Secondly, we check if there's any edge whose magnitude is over the maximum allowed ( $l_{max}$ ). Segments that fit this requirement are split in two, as illustrated in figure 3.22(b).

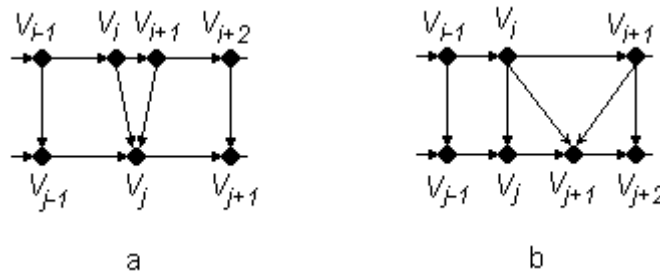


Figure 3.22: Resampling method.

As previously explained, the values for  $l_{min}$  and  $l_{max}$  are derived from the parameter  $l_{des}$ . The condition that must be satisfied is:

$$l_{max} > 2l_{min} \tag{3.47}$$

All the values that validate this condition are permitted. The values depend strongly on every application. In the case of SPECT medical imagery, satisfactory tested values are given in equations 3.48 and 3.49.

$$l_{min} = 0.5 \cdot l_{des} \tag{3.48}$$

$$l_{max} = 1.5 \cdot l_{des} \tag{3.49}$$

For the new inserted vertex, we need to specify a velocity and an acceleration value that can be calculated from the averaged values of the two vertices substituted (figure 3.22(a)) or from its new neighbors (figure 3.22(b)). This is necessary in order to maintain continuity in the dynamical situation of the model.

### 3.4.2 Plain deformation model

In the plain deformable model [7], the deformation forces are defined for each of the triangles in the initial mesh. The deformation is characterized by the action of three different internal forces: stretch, shear and bend.

Each of the forces is implemented like a restriction condition that the system has to maintain, as equation 3.50 summarizes.

$$C(\mathbf{x}) = 0 \tag{3.50}$$

We can derive an energy function  $E_c(\mathbf{x})$  and a force, from the restriction.

$$E_c = \frac{k}{2} C(\mathbf{x})^T C(\mathbf{x}) \tag{3.51}$$

$$f_i = -\frac{\partial E_c}{\partial x_i} = -k \frac{\partial C(\mathbf{x})}{\partial x_i} C(\mathbf{x}) \tag{3.52}$$

$$J = \frac{\partial f}{\partial \mathbf{x}} \tag{3.53}$$

$$J_{ij} = \frac{\partial f_i}{\partial x_j} = -k \left( \frac{\partial C(\mathbf{x})}{\partial x_i} \frac{\partial C(\mathbf{x})}{\partial x_j}^T + \frac{\partial^2 C(\mathbf{x})}{\partial x_i \partial x_j} C(\mathbf{x}) \right) \tag{3.54}$$

In equations 3.51 and 3.52,  $k$  is a constant that weights the restriction  $C(x)$ . If we consider that the restrictions are triangle-based,  $C(x)$  will depend on a few particles. Then our system will be sparse in terms of force representation. Each force element,  $f_i$ , is an  $\mathfrak{R}^3$  vector. The force derivatives matrix  $J$  belongs to  $\mathfrak{R}^{3n \times 3n}$ . Each of its elements  $J_{ij}$  is a 3 x 3 matrix.

For stability adjustment, every force has an associated damping term, defined in equation 3.55:

$$f_d = -k_d \frac{\partial C(\mathbf{x})}{\partial \mathbf{x}} \dot{C} \tag{3.55}$$

This term is opposed to the force and proportional to its velocity, understood as the first derivative of the restriction. In fact the damping constant can be locally adjusted by using a certain spatially varying function or by building a damping map, as depicted in figure 3.23.

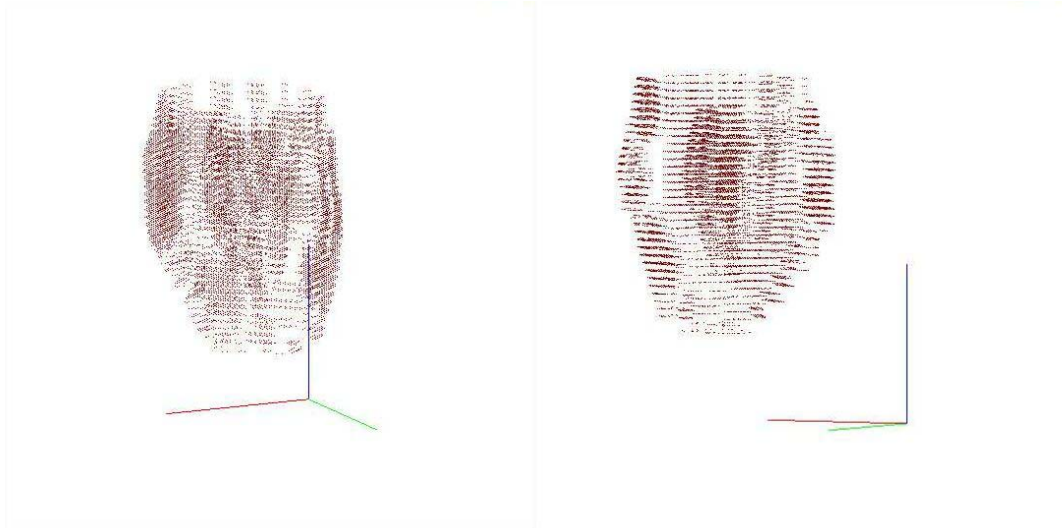


Figure 3.23: Two views of the damping map for a left ventricle medical dataset.

Figure 3.23 shows several slices of medical imagery piled up forming what is called a world of voxels. Here every voxel stores the damping factor that will be used for a particle if it enters it. The value for the damping factor can be associated to a voxel depending on several reasonings. The voxels represent a human left ventricle and have been labeled according to their probability to be part of the muscular tissue. That means that voxels with high probability will be associated a larger damping factor so that particles stop in them when trying to reconstruct the organ in 3D.

### 3.4.3 Spring-mass deformation model

In this model the deformation is stretch-based, as stated in Hook's law. Then the deformation is characterized by the stretching force between pairs of particles, connected by springs.

Figure 3.24 and equation 3.56, show the stretch force between two particles  $a$  and  $b$ , as defined by Hook's law.

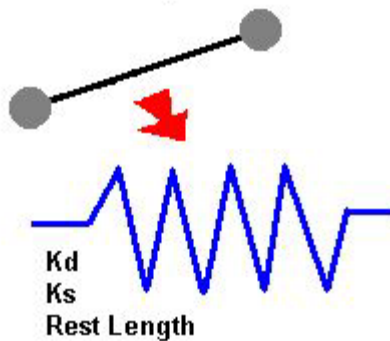


Figure 3.24: A damped spring connects two particles.

The force that the spring applies to the particles is described by equation 3.56:

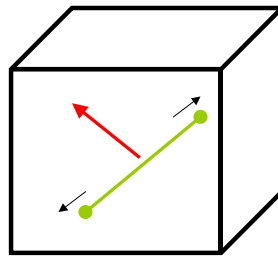
$$f_a = - \left[ k_s (|l| - r) + k_d \frac{\dot{l}}{|l|} \right] \frac{l}{|l|} \quad (3.56)$$

$$f_b = -f_a$$

Where  $k_s$  is the spring constant,  $k_d$  its associated damping factor,  $l$  its elongation and  $r$  the distance between particles in equilibrium.

### 3.4.4 Restricted spring-mass deformation model

Designed as a variation from the previous model, the restricted spring-mass deformation model adds a movement restriction to the springs. Their stretch deformation is only allowed in some specific directions. For instance, for the left-ventricle reconstruction application, the springs can be only allowed to deform in the direction orthogonal to the vector field. Figure 3.25 summarizes this behavior.



Voxel  $i, j, k$

Figure 3.25: Restricted spring.

For a voxel at some given position  $i, j$ , and  $k$ , that contains two particles joined by a spring (green) and a gradient vector (external force), the restricted model evaluates the component of the stretch force that is orthogonal to the gradient vector (red). This force component will be added to the force accumulators of both particles.

This model distinguishes between two kinds of motion:

- Tangent to data, performed by the restricted springs (internal forces).
- Normal to data, performed by the gradient vectors (external forces).

The problem arises when the particles are really close to the data borders. There, the gradient vectors tend to disappear (their magnitude tends to be zero) and then the component determination might become unstable. Figure 3.26 shows the vector projection that allows the component to be evaluated.

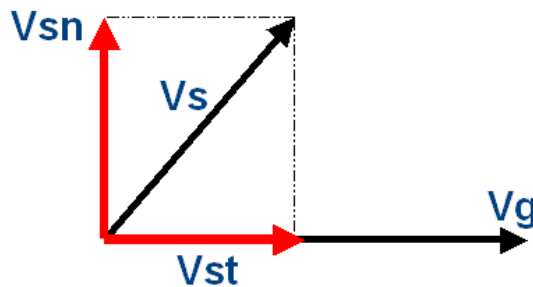


Figure 3.26: Evaluating the orthogonal-to-the-gradient-vector component of the spring force.

Where  $V_s$  is the force vector associated to the spring,  $V_g$  is the gradient vector,  $V_{st}$  is the component of  $V_s$  tangential to  $V_g$  and  $V_{sn}$  is the component of  $V_s$  normal to  $V_g$ . If  $V_g$  tends to be zero, there's no projection and the evaluation is unstable. We can treat this behavior as a separate case, where springs near data borders are released from the restriction and therefore treated as normal.

The projection equations are described in chapter 5.

### **3.4.5 Free deformation model**

The free deformation model can be understood like a simplification of the spring-mass deformation model. It consists on leaving the external force as the unique force within the system. Then the particles have no real connectivity and can be treated separately.

The free deformation model is characterized by:

- High convergence rates, because of the small amount of numerical evaluations needed.
- It doesn't provide control over the mesh topology. Therefore it becomes important to apply some reinforcement strategies in order to ensure the required smoothness (see chapter 5 for comments on the smoothing algorithm).
- Failures in the case of data absences, for instance in areas of poor blood irrigation for the left's ventricle reconstruction application. If the volume to recover presents "holes", particles won't have any restriction that avoids them from penetrating to the interior. However the smoothing algorithm also helps in this question (see chapter 5 for comments on the smoothing algorithm).

The plain and spring-mass deformation models provide control mechanisms over those effects (the damping factors associated to the internal forces, for instance).

There we have a tradeoff then: robustness of the solution against high convergence rates when recovering data.

## **3.5 Summary**

In chapter 3 we have been through the deformation models that have been analyzed and tested. All of them share the evolution paradigm, a Newtonian scheme. We have also described their internal and external forces.

The Newtonian scheme defines the underlying dynamics related to the reconstruction of the left ventricle. Particle systems modeled like geometrical meshes try to find their equilibrium state. This state of minimum energy corresponds to the 3D recovery of the original shape.

In order to rule the system, several forces have to be defined. The first classification states that internal and external forces must be introduced.

Internal forces ensure stability, curvature requirements and cohesion. This document describes the stretch, shear, bend, and local curvature forces that have been extensively tested with good results.

External forces drive the system to the original data that has to be recovered. There are several options available like the traditional snakes, the active nets, the topologic nets, and so on. We have ended selecting the radial energy based potentials and the GVF and GGVF vector fields like the best paradigms that completely fit our needs.

This chapter also presents the theoretical concepts related to the five deformable models that have been used. The discreet contour, plain, spring-mass, restricted spring-mass and free deformable models have been used with success. Using one or another depends on several constraints like accuracy, precision, final smoothness and speed on the calculations.

We need to use a computer in order to perform our simulations in a fast manner. Chapter 4 explores the numerical methods available in order to implement the technologies that we need. Several possibilities are presented and tested extensively.

## 4 Numerical implementation

Simulations describing physical reactions require computing over the time. The interval of simulation has to be discretized into sequential steps. In order to compute the state of the virtual scenery at every step, we need a numerical method, or solver, that evaluates several derivative values. The system iterates from the beginning,  $t_0$ , until the final instant,  $t_{max}$ , by using a defined stepsize for every cycle of the simulation.

There are several solvers available, different in terms of efficiency, robustness and speed of evaluation. This chapter describes the underlying numerical theory related to the simulation of our deformable models and the solvers used in order to implement them.

This chapter presents the following contributions:

- A deep analysis of all the numerical methods that have been implemented, according to a first classification: explicit vs. implicit schemes.
- An adaptative stepsize alternative to dynamically adjust the simulator.
- Explanations on the implementations of the deformable models presented in chapter 3.
- A comparative test between the numerical implementations of the deformable models.
- Explanation on the implementation of the external force.

### 4.1 ODE based methods

The Ordinary Differential Equation (ODE) methods are based on a differential equation that describes the relationship between a function and each of its derivatives. Satisfying it means checking some other conditions as well. The behavior of the system can be described like equation 4.1 shows:

$$x' = f(x, t) \tag{4.1}$$

Where  $f$  is a known function dependent on  $x$  and  $t$ ,  $x$  is the state of the system and  $x'$  is its time derivative. We can have several types of forms for our derivative functions.

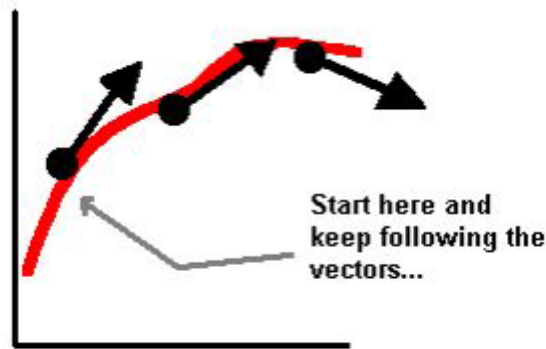
Most of the situations that we might need to simulate can be classified like initial value problems [103]. In those some kind of initial state is given, like a particle's position  $x_0$  for a given time  $t_0$ .

$$x(t_0) = x_0 \tag{4.2}$$

From the relation stated in equation 4.2, we need to be capable of following the system's behavior along the time.

If we try to visualize this problem graphically, we can think in a "vector field" (see figure 4.1). In the two-dimensional case, at any point  $x$  the function  $f$  will give us a two-dimensional vector. In a dynamical context, we could say that the vector at  $x$  is the velocity that the free point  $p$  must have if ever moves through  $x$ .





The derivative function forms a vector field.  
Figure 4.1: The initial value problem. [34]

Wherever we initially deposit the point  $p$ , the “stream” at that position will seize it, like the ocean would act in a particle of sand. We provide the initial position of  $p$  and the ocean, the trajectory function  $f$ , does the rest. The trajectory determined by  $p$  through  $f$  forms an "integral curve" of the generated vector field.

Something that needs to be taken into consideration is that  $p$ 's velocity depends on where it is but also on when it reaches that position. Depending on the moment,  $p$  can perceive one or other derivative, in the same physical space.

A numerical method or solver is intended to allow the calculation of an ODE based simulation into a computer. ODE equations are essential for modeling physical situations like chemical reactions, resistor-capacitor-inductance circuits or spring-mass systems besides other problems related to ecology or economics, for instance.

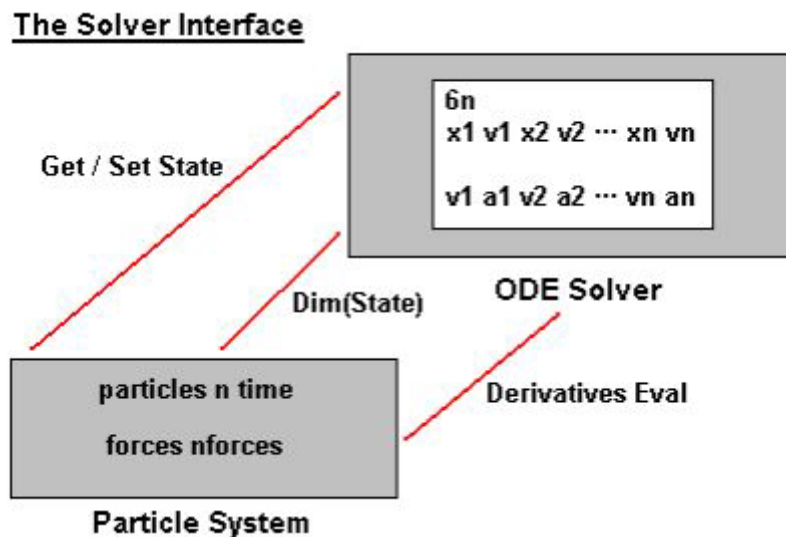


Figure 4.2: The solver interface. [34]

Let us assume that we are working in a spring-mass framework. There a particle system moves according to the application of several forces. Particles have mass and then, by following Newtonian dynamics (see chapter 3), perceive an acceleration that converts into velocity and position while time passes.

Figure 4.2 summarizes the interface between the particle system and the solver. The system might be formed from  $n$  particles and  $n$  forces, either internal or external. The ODE solver gets the state of the system, changes it if time passes and sets it again. As long as the particle system is formed from  $n$  particles and every particle is characterized by its position, velocity and acceleration, the solver must treat with a large amount of values (in fact a  $6n$  vector containing the 3D position and velocity of all the particles leads to the calculation of a  $3n$  vector of 3D accelerations). From the forces and the masses, the solver evaluates the accelerations and the derivatives associated to the system.

Problems like the spring-mass framework involve second-order equations where a particle moves in a general force field [62]:

$$\frac{d^2x}{dt^2} = f\left(x, \frac{dx}{dt}, t\right) \quad (4.3)$$

In the ODE equation 4.3,  $x$  stands for a particle's position,  $\frac{\partial x}{\partial t}$  is its associated position derivative or velocity, and  $t$  is the time variable. Second-order equation 4.3 can be reduced to an equivalent set of two first-order equations without loss of generality:

$$\begin{aligned} y &= (y_1, y_2) \\ y_1 &= x \\ y_2 &= \frac{dx}{dt} \\ \frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= f(y_1, y_2, t) \end{aligned} \quad (4.4)$$

Where a set of two new variables,  $y_1$  and  $y_2$ , has been introduced. We can rename the equations for the seek of clearness, according to a Newtonian scheme:

$$\begin{aligned} \frac{dx_i}{dt} &= v_i \\ \frac{dv_i}{dt} &= \frac{F_i}{m_i} \end{aligned} \quad (4.5)$$

Here  $x_i$  stands for the  $i_{th}$  particle's position,  $v_i$  is its velocity,  $F_i$  is the sum of forces acting on it and  $m_i$  is its mass. It comes clear that solving the system implies solving two first-order equations where forces and masses contribute to the evaluation of an acceleration value. This value is used for getting the changes in velocity and position when a discrete instant of time ( $\Delta t$ ) passes.

Solving this kind of equations in a discrete manner, like in a computer for instance, is the goal of a solver. Different solvers will be presented within the next sections

according to the classification on explicit or implicit methods, which is related to the kind of schemes used to defining them.

## 4.2 Explicit methods

ODE solvers in this category are formulated in an explicit manner. It means that their equations can be directly solved with no need from iterative processes. Explicit methods are classically simpler than implicit methods because of the clearness of their inner equations.

Scoring in terms of accuracy, precision, smoothness, robustness or speed depends on the complexity of the method. Several possibilities are presented next.

### 4.2.1 Euler's method

Euler's method has been always stated as the simplest one. Let us assume that our initial value for the position  $x$  is:

$$x_0 = x(t_0) \tag{4.6}$$

And that our estimate value one step after is:

$$x(t_0 + h) \tag{4.7}$$

Where  $h$  is the stepsize parameter (the increment of time referred sometimes as  $\Delta t$ ).

As figure 4.3 shows, Euler's method computes a new step using the derivative direction at the beginning of the interval.

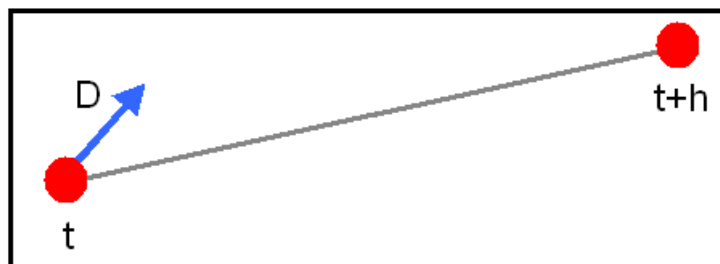


Figure 4.3: Euler's method.

So that:

$$x(t_0 + h) = x_0 + h\dot{x}(t_0) \tag{4.8}$$

Instead of having a real vector field, Euler provides us with a polygonal path for every point  $p$ .

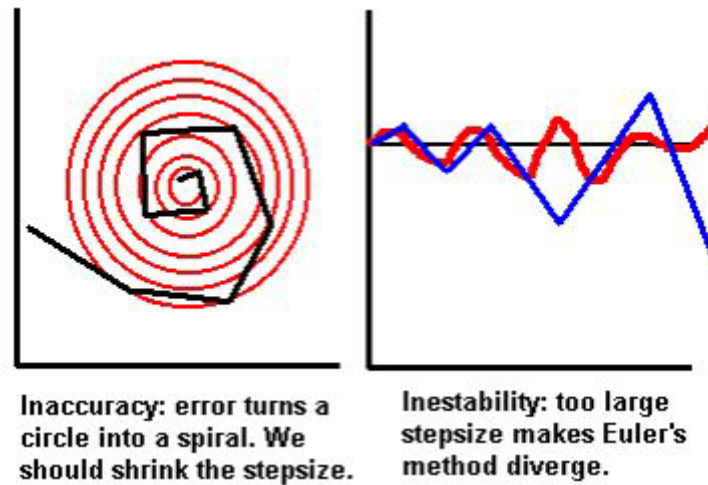


Figure 4.4: Euler's inaccuracies and instabilities. [34]

It is a simple method but not accurate at all [103]. Suppose that we have a 2D function  $f$  whose integral curves are concentric circles. A point  $p$  governed by  $f$  should orbit forever around its circle. Instead of that, using Euler's method we will move the point on a straight line that will make it appear in a circle with larger radius. Its trajectory will achieve the form of a spiral. This behavior cannot be avoided. See figure 4.4 for more details on this effect.

Moreover that, Euler can get unstable. For instance, in the case of Hook's law:

$$f = -kx \tag{4.9}$$

One can find that a point  $p$  following it must decay to zero. That will happen for small enough stepsizes but not in other cases. If  $h > 1/k$  we have a solution that never stops. Instead of that, it oscillates around zero forever.

It comes clear that Euler's accuracy depends strongly on the stepsize selected, as figure 4.5 depicts. Tiny stepsizes offer good results in terms of trajectory (red) while large stepsizes lead to wrong results (magenta).

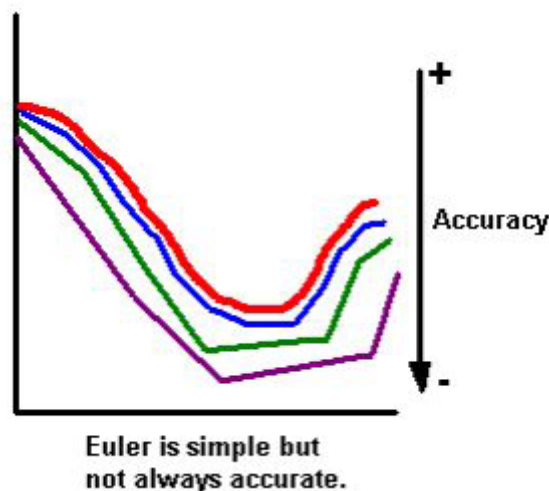


Figure 4.5: Euler's accuracy depends strongly on the stepsize. [34]

Euler's method takes one evaluation of the derivative per step, with short steps if we want to keep accuracy and stability (it always depends on the simulation frame).

There are more robust cost-per-step methods that even evaluating four or five steps per iteration, can greatly outperform Euler's efficiency allowing the steps to be much longer.

Let's look at the Euler's error more closely. If we assume that our trajectory function  $x(t)$  is smooth enough, we can describe it in terms of its associated Taylor series. We can express the value at the end of the step as an infinite sum of terms involving the derivative value at the beginning:

$$x(t_0 + h) = x(t_0) + hx'(t_0) + \dots \tag{4.10}$$

We get the Euler's method formula by truncating the Taylor series right after the second term. This means that Euler would be correct in the case that all derivatives beyond first were zero or either negligible.

The error is dominated by the  $\frac{h^2}{2} x''(t_0)$  term so that we have an absolute error of  $O(h^2)$ .

The error that we accumulate over an interval from  $t_0$  to  $t_1$  depends linearly on the stepsize  $h$ .

Someone could think that it shouldn't be a big problem. We could select a suitable stepsize, a small enough one. In practice, lots of steps will be required in order to avoid error accumulation.

### 4.2.2 The midpoint method

In section 4.2.1, we showed that the Euler method uses two terms of the Taylor series for the trajectory to approximate. For the Midpoint method we take another term of the Taylor series in order to keep the maximum error at  $O(h^3)$  [103]:

$$x(t_0 + h) = x(t_0) + hx'(t_0) + \frac{h^2}{2} x''(t_0) + O(h^3) \tag{4.11}$$

Our derivative function depends on  $x(t)$  and  $t$ . For simplicity in the explanation we will derive this method avoiding the dependency on  $t$ . Then we will use the expression:

$$x' = f(x(t)) \tag{4.12}$$

Instead of

$$x' = f(x(t), t) \tag{4.13}$$

Equation 4.14 summarizes the chain rule:

$$x'' = \frac{df}{dx} x' = f' f \tag{4.14}$$

We will avoid evaluating  $f'$  because it would be complicated and expensive and truthfully, not really necessary. Instead we can approximate the second-order term just in terms of  $f$  and use it in equation 4.11. It should look like:

$$f(x_0 + \Delta x) = f(x_0) + \Delta x f'(x_0) + O(\Delta x^2) \tag{4.15}$$

We introduce  $x''$  by choosing:

$$\Delta x = \frac{h}{2} f(x_0) \tag{4.16}$$

And then:

$$f(x_0 + \frac{h}{2} f(x_0)) = f(x_0) + \frac{h}{2} f(x_0) f'(x_0) + O(h^2) = f(x_0) + \frac{h}{2} x''(t_0) + O(h^2) \tag{4.17}$$

Where  $x_0 = x(t_0)$ . If we multiply both sides by  $h$  and rearrange:

$$\frac{h^2}{2} x'' + O(h^3) = h[f(x_0 + \frac{h}{2} f(x_0)) - f(x_0)] \tag{4.18}$$

And then if we take the right hand side and we put it into equation 4.11:

$$x(t_0 + h) = x(t_0) + h(f(x_0 + \frac{h}{2} f(x_0))) \tag{4.19}$$

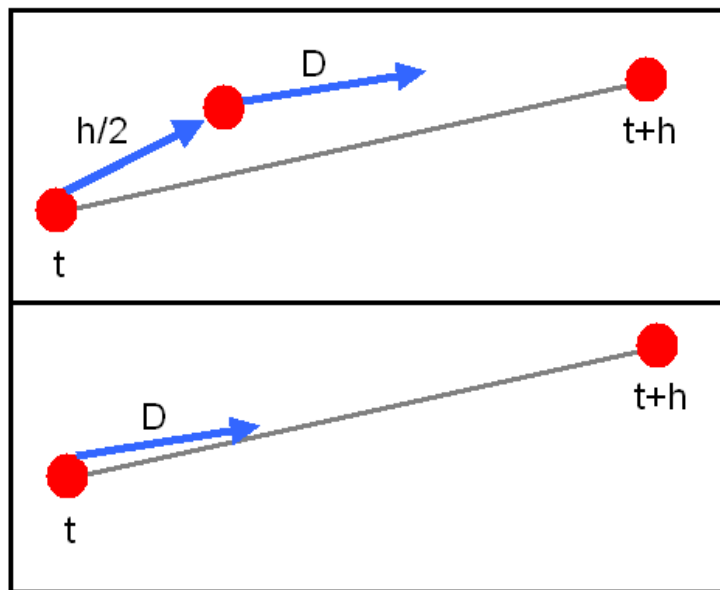
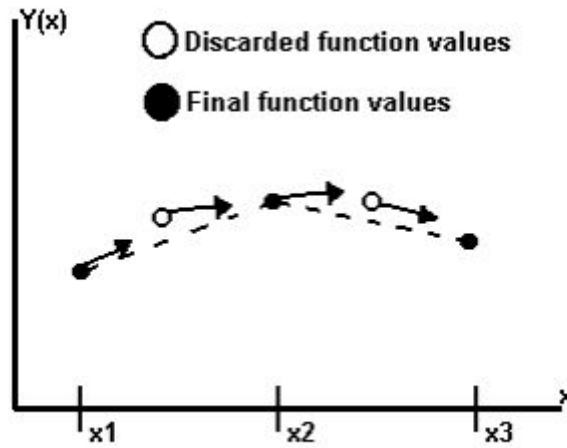


Figure 4.6: The Midpoint Method.

Equation 4.19 evaluates an Euler step for performing a second derivative calculation at a midpoint between both ends of the interval. Then the midpoint method is a second-order solution method where the second derivative, evaluated at the midpoint of the interval, is finally used to calculate the whole step.

The midpoint method associated error is  $O(h^3)$ . In that sense it is important to note its major precision over the Euler method. Nevertheless it requires two evaluations of the derivative function per step, as figure 4.7 shows.



The Midpoint method.

Figure 4.7: Values of the derivative in the midpoint method. [34]

We could follow adding more terms to our Taylor series in order to minimize the error term. That's the case analyzed at section 4.2.3, the Runge-Kutta methodology.

### 4.2.3 The Runge-Kutta 4 method

The Runge-Kutta 4 method is an extension of the previously described midpoint method [103]. Figure 4.8 summarizes its philosophy.

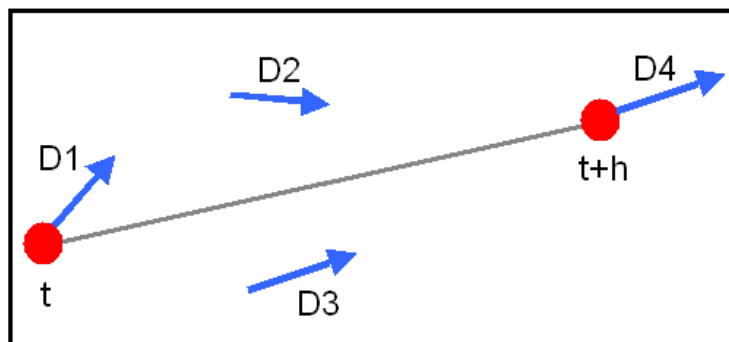


Figure 4.8: Slope evaluations in the Runge-Kutta 4 method.

This method requires four evaluations of the slope per iteration. Nevertheless the stepsize can be greater than in the midpoint method (at least twice as large) while keeping the same accuracy. The local error term for the fourth-order Runge-Kutta method is  $O(h^5)$  whereas the global error is  $O(h^4)$ .

The overview of the algorithm's behavior is not really complex. As in the other methods, we input the values of the independent variables and we get out new values that are stepped by a stepsize  $h$  (which as in the other methods can be either positive or negative). We also must supply the derivatives function for evaluating the right-hand side of the expressions and the starting values for all the derivatives.

The Runge-Kutta 4 method treats every step in a sequence of steps in identical way. Prior behavior of a solution is not used in its propagation. Especially the first affirmation makes it very easy to incorporate this algorithm into relatively simple "driver" schemes.

To check the accuracy we can increase the step-number, repeat the integration and compare results at the end. In fact this is usually done when comparing results between this method and high order Runge-Kutta methods (fifth, sixth and so on). Those comparisons can be useful to advantage in determining a suitable size for the stepsize  $h$ .

Greater complexity results from having to compare terms of a higher order. We must solve more equations per step (11 equations and 13 unknowns) but we reduce the error term. Equations 4.20 to 4.24 describe this classical methodology.

$$k_1 = hf(x_0, t_0) \tag{4.20}$$

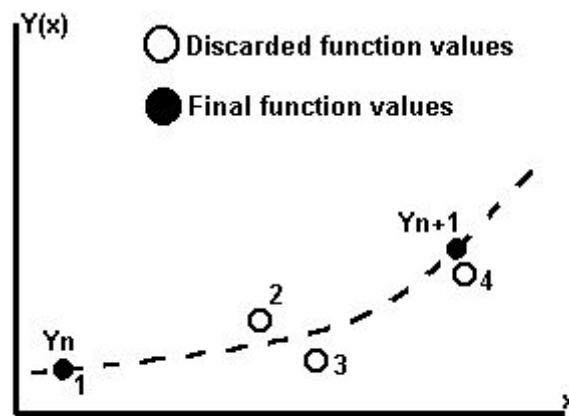
$$k_2 = hf\left(x_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}\right) \tag{4.21}$$

$$k_3 = hf\left(x_0 + \frac{k_2}{2}, t_0 + \frac{h}{2}\right) \tag{4.22}$$

$$k_4 = hf(x_0 + k_3, t_0 + h) \tag{4.23}$$

$$x(t_0 + h) = x_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \tag{4.24}$$

Note how the slope is evaluated four times, corresponding to four different positions within the step. Then the values are averaged with different weights attending to their relevance on the final result. Figure 4.9 shows this behavior graphically.



**Fourth-order Runge-Kutta method.**

Figure 4.9: The slope is evaluated four times in the Runge-Kutta 4 method. [34]



Implementing the Runge Kutta 4 method can be done by successively applying the simple Euler method, for every slope evaluation. From that knowledge we can take the simple Euler equations and solve the derivative term sequentially at different positions in order to find the  $k_1$  to  $k_4$  coefficients. Then we can average them by simple addition (see equations 4.20 to 4.24).

#### 4.2.4 Adaptive stepsize

Changing the stepsize can exert some adaptive control over the progress of the simulation. We need to achieve an accuracy at the same time that we must save as much computational effort as possible [103].

The gains in efficiency can be important. We should select a stepsize or another depending on the present situation. A small stepsize in the case of going through a rough path or a larger stepsize if traversing a very flat and homogeneous space. It is not the same to simulate a bouncing football over a concrete floor than a golf ball running across a grass field.

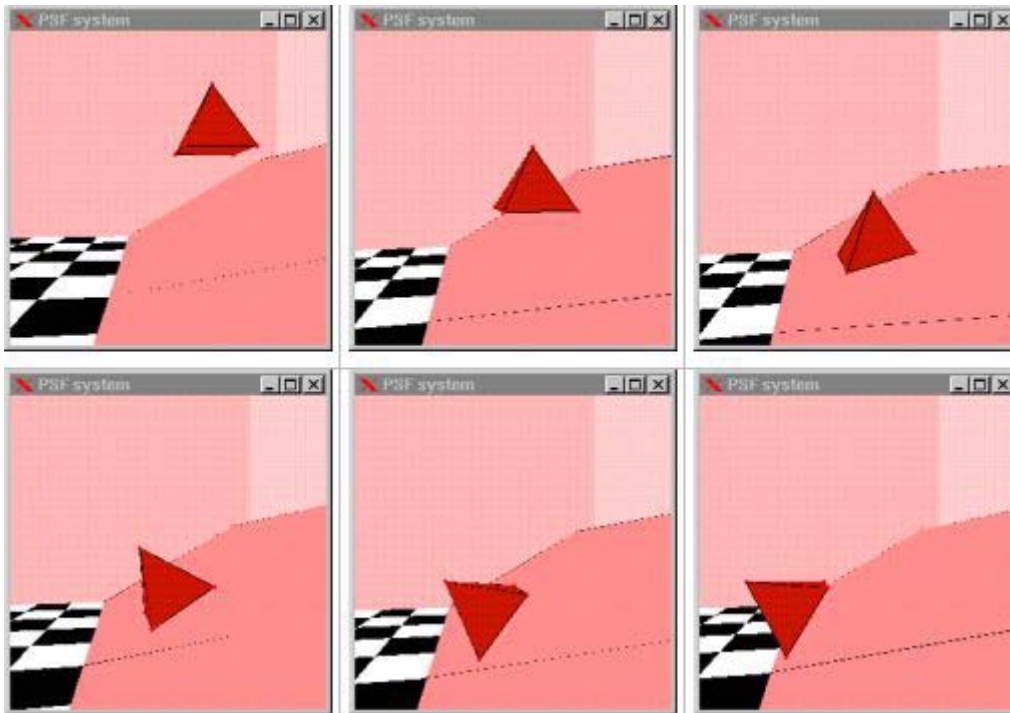


Figure 4.10: Adaptive stepsize in a simulation framework [34].

Figure 4.10 shows a particle system being simulated in real time. The sequence consists on a bouncing pyramid that collides against a plane. The stepsize can be larger when no collision exists because of the relative simplicity of the acting forces, only external gravity in that case. When the pyramid collides, large stepsizes might spoil the simulation that has to be run by following tiny instants of time in order to be reliable.

For this simulation, the initial stepsize was selected to be 0.04 seconds. From that moment the underlying solver automatically adapted it.

It comes clear that we want to choose the stepsize  $h$  as large as possible but we also need to maintain the error term between some desirable bounds. By adapting our

stepsize at each iteration, we inform the system about the kind of "scenery" that is being traversed.

Implementing the adaptative stepsize into our system requires information about performance and a reliable estimation of the truncation error. The computation of all these values supposes a clear computational overhead, but the investment pays the effort.

One of the techniques intended to adapt the stepsize is called step doubling. It is a very straightforward technique. It consists on taking every step twice. The first step is taken as a full step and the second step is built from two half steps. Let us assume that the exact solution for an advance from  $x$  to  $x + 2h$  is  $y(x + 2h)$  and that the two approximated solutions are  $y_1$  (one step  $2h$ ) and  $y_2$  (two steps of  $h$  each). Then we can define an error term like:

$$\Delta_e = y_2 - y_1 \tag{4.25}$$

We can control this difference in order to keep our desired degree of accuracy. We adjust  $h$  at every new iteration.

Now that we know approximately our committed error, we need to consider how to keep it within some desired bounds. We need to find some kind of association between  $\Delta_e$  and  $h$ . We can assume that a stepsize  $h_1$  produces an error  $\Delta_{e1}$ , while a stepsize  $h_0$  gives an error  $\Delta_{e0}$  that can be estimated like equation 4.26 shows.

$$h_0 = h_1 \left| \frac{\Delta_{e0}}{\Delta_{e1}} \right|^{0,5} \tag{4.26}$$

For the Euler method.

$\Delta_{e0}$  is our desired accuracy. If  $\Delta_{e1}$  is larger than  $\Delta_{e0}$ , the equation is telling us how much to decrease the stepsize when we retry the current, and failed, step. Otherwise, if  $\Delta_{e1}$  is smaller than  $\Delta_{e0}$ , then the equation tells us how much we can increase the stepsize for the next step.

In order to provide an example, let us suppose that we have defined an internal accuracy of  $10^{-3}$ . If the present error is  $10^{-6}$ , we can increase our stepsize  $h$  accordingly:

$$\left| \frac{10^{-3}}{10^{-6}} \right|^{0,5} h = 31,6h \tag{4.27}$$

On the other hand, if our error is  $10^{-2}$ , we are not maintaining our accuracy within its desirable bounds. Then we should reduce the stepsize:

$$\left| \frac{10^{-3}}{10^{-2}} \right|^{0,5} h = 0,316h \tag{4.28}$$

Those examples give a clue on the tradeoff between the stepsize and the committed error.

When controlling the stepsize, we can define a unique accuracy for the whole system or a vector of accuracies, one for each of the ODE's in the system. All the equations will have to be within their respective bounds. In such cases we can rescale the stepsize attending to the requirements of the "worst-offender" equation.

Another problem arises when having to select the accuracy value. It strongly depends on the concrete application and it has to be defined accordingly.

More complex methods like the Runge-Kutta-Fehlberg (RKF-45) method provide mechanisms that adapt the stepsize internally with no need from other strategies like the step doubling [91].

### 4.3 Implicit methods

Implicit solvers are based on algebraic formulas that need to be solved. Iterative algorithms or even explicit solvers usually support these kinds of methods.

It is common that implicit methodologies are more complicated than explicit methods while ensuring a better convergence in a few iterations.

We will comment the implicit version of the Euler method. Starting from the same philosophy as the simple Euler method, the backwards or implicit Euler method makes some decisions about the derivative value. It assumes some kind of average between the derivatives at the beginning and at the end of the interval.

If we generalize the average, we get the formulation of equation 4.29:

$$Y(t + \Delta t) = Y(t) + [(1 - \lambda)f(Y(t)) + \lambda f(Y(t + \Delta t))]\Delta t \quad (4.29)$$

Where  $f(Y(t))$  equals  $\frac{dY}{dt}$  and  $\lambda$  is a constant parameter that belongs to the interval  $[0,1]$ .

It is important to note how using a linear interpolation averages the derivatives at the beginning and at the end. If  $\lambda=0$  the equation equals the simple Euler method formulation. When  $\lambda=1$  the equation is called the implicit version of Euler's method.

These kinds of formulations require a prior knowledge of the data that we are precisely looking for. We do not know  $Y(t + \Delta t)$  at the beginning of the interval but we can overcome this problem by using additional derivative information:

$$\frac{dY}{dt} \approx f(Y_0) + (Y - Y_0)\nabla f_{Y=Y_0} \quad (4.30)$$

And if we substitute this approximation into equation 4.29:

$$\Delta Y = [f(Y_0) + \lambda \Delta Y (\nabla f)_{Y=Y_0}] \Delta t \quad (4.31)$$

And solving for  $\Delta Y$ :

$$\Delta Y \left[ \frac{1}{\Delta t} I - \lambda (\nabla f)_{Y=Y_0} \right] = f(Y_0) \quad (4.32)$$

Where  $I$  stands for the identity matrix. Computing the update for  $Y$  requires solving a linear system.

If we apply equation 4.32 to a spring-mass framework we can get ride of some simplifications because the force depends only on the positions of the particles and not on their velocities (unless we use a damping factor).

We end up having that:

$$\Delta x A = v \quad (4.33)$$

Where

$$A = \left[ \frac{1}{\Delta t} I - \lambda H \right] \quad (4.34)$$

And  $H$  is the Hessian matrix associated to the spring energy, like equation 4.35 shows:

$$H = \frac{\partial^2 E}{\partial x_i \partial x_j} \quad (4.35)$$

In fact this method is faster than simple Euler method. Part of the reasoning is related to the sparse nature of matrix  $A$ , which is three-diagonal:

$$A = \begin{pmatrix} a_0 & b_0 & & & 0 \\ b_0 & a_1 & b_1 & & \\ & b_1 & a_2 & & \\ & & & \dots & \dots \\ 0 & & & b_{n-2} & a_{n-1} \end{pmatrix} \quad (4.36)$$

This matrix can be solved easily in time proportional to  $n$  [98] by using a factorization method that uses a lower and an upper triangular matrix.

#### 4.4 Implementations of the deformable models

Several deformable models were presented in the previous chapter. Besides that, several numerical methods have been presented in the preceding sections. The numerical implementation of such deformable models uses some of the paradigms mentioned before.

#### 4.4.1 Implementation of the plain deformable model.

For details on the plain deformable model, see section 3.4.2 of chapter 3. When implementing this deformable model we must take into consideration that:

- The model is built in a spring-mass framework.
- Several forces (three internal and one external) are defined. This fact forces us to use a very stable, robust and precise numerical scheme, specially when dealing with the internal forces that model the inner elasticity of the mesh (see sections 3.2.1, 3.2.2 and 3.2.3 of chapter 3).
- We are always referring to a Newtonian dynamics approach.

In order to ensure the stability of the solution we use an implicit numerical scheme, the Backwards Euler method [7]. Then we must personalize the general methodology described to fill our needs, a spring-mass framework with particles involved.

The variations in position and velocity can be written like:

$$\Delta x = x(t_0 + h) - x(t_0) \tag{4.37}$$

And

$$\Delta v = v(t_0 + h) - v(t_0) \tag{4.38}$$

Where  $h$  stands for the stepsize. Then the system can be written like:

$$\begin{pmatrix} \Delta x \\ \Delta v \end{pmatrix} = h \begin{pmatrix} v_o + \Delta v \\ M^{-1} f(x_o + \Delta x, v_o + \Delta v) \end{pmatrix} \tag{4.39}$$

The method evaluates the forces at the next time step. In order to find those forces, it uses a first order *Taylor* approximation:

$$f(x_o + \Delta x, v_o + \Delta v) = f_o + \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial v} \Delta v \tag{4.40}$$

The derivatives  $\partial f/\partial x$  and  $\partial f/\partial v$  are evaluated in  $(x_o, v_o)$ . Substituting them in equation 4.39 we obtain:

$$\begin{pmatrix} \Delta x \\ \Delta v \end{pmatrix} = h \begin{pmatrix} v_o + \Delta v \\ M^{-1} (f_o + \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial v} \Delta v) \end{pmatrix} \tag{4.41}$$

And knowing that:

$$\Delta x = h \cdot (v_o + \Delta v) \tag{4.42}$$

We finally find:

$$\Delta v = h \cdot M^{-1} \left( f_o + \frac{\partial f}{\partial x} h(v_o + \Delta v) + \frac{\partial f}{\partial v} \Delta v \right) \quad (4.43)$$

We group terms and we finally obtain a linear equation system that serves us to find  $\Delta v$ :

$$\left( I - hM^{-1} \frac{\partial f}{\partial v} - h^2 M^{-1} \frac{\partial f}{\partial x} \right) \Delta v = hM^{-1} \left( f_o + h \frac{\partial f}{\partial x} v_o \right) \quad (4.44)$$

This is a linear equation system with  $3n$  unknowns. The final solution is the velocity increment for all the particles in the system. From this knowledge, we can calculate the new position for all the particles like:

$$x = x + \Delta x \quad (4.45)$$

Where

$$\Delta x = h \cdot (v_o + \Delta v) \quad (4.46)$$

In order to perform a step, we must evaluate the forces and their derivatives (related to position and velocity).

The implicit characteristic of the method allows us to use bigger stepsizes, having to solve a linear equation system at each iteration. Those systems are typically sparse and then are suitable to be solved by an iterative method like the conjugate gradient method [5, 7].

#### **4.4.2 Implementations of the spring-mass, restricted spring-mass and free deformable models**

The spring-mass, restricted spring-mass and free deformable models share the Newtonian approach with the plain deformable model. Moreover that, all of them are methods based on the simulation of a particle system. Nevertheless, the difference appears when talking about the internal forces implied. Those methods employ only one (spring-mass and restricted spring-mass deformation models) or none (free deformation model) internal force (see sections 3.4.3, 3.4.4 and 3.4.5 of chapter 3). It means that their numerical evaluations are not as critical as in the previous model. In fact similar results in terms of robustness and accuracy can be achieved in these by explicit methods, like the Runge Kutta 4 method (section 4.2.3).

#### **4.4.3 Comparative between numerical implementations**

This section presents several experiments as a comparison between the implicit and explicit methods used within the scope of this project.

In the first test, there is a common simulation experiment that consists on the 3D reconstruction of the external surface of a phantom volume (see appendix D for more details). This volume is used for testing the quality of the reconstruction because of its known measurements.

The common conditions for the experiment are described next:

- Initial reconstruction mesh built from 642 particles and 1280 triangles. See chapter 5 for more details on the geometrical mesh.
- The external GVF force was weighted by a factor of 25. See chapter 3 for more information regarding this external force.
- A damping factor of 0.25 was associated to the external GVF vector field.
- The discretized phantom volume was formed from 8616 peripheral voxels. Those voxels were used as the borders where the geometrical mesh had to stop.

Table 4.1 summarizes the results of the simulations depending on the deformation model employed. Note that the plain deformation model was simulated by a backwards Euler method. The other deformation models were simulated by using an explicit Runge-Kutta 4 method.

The stepsizes used in the implicit method can be considerable bigger than in the explicit simulations. The selected stepsize is then associated to the internal forces simulated and to the mesh resolution (the resolution can vary the internal forces magnitude). The free deformation model is able to drive a high percentage of particles to a distance less than one voxel long from the real data to recover.

The parameters *K Stretch*, *K Bend*, *K Shear* and their associated damping factors were tuned by using typical values used for most of the simulations. Those values are also written in table 4.1.

	<b>Plain (1)</b>	<b>Plain (2)</b>	<b>Spring-mass</b>	<b>Free</b>
<b><math>\Delta t</math></b>	0.01 seg.	0.05 seg.	0.05 seg.	0.05 seg.
<b>% final particles less than 1 voxel away</b>	77 %	74.5 %	73.6 %	97 %
<b>K Stretch</b>	10	10	10	----
<b>K Stretch Damping</b>	1	1	1	----
<b>K Bend</b>	10	10	----	----
<b>K Bend Damping</b>	1	1	----	----
<b>K Shear</b>	1	1	----	----
<b>K Shear Damping</b>	0.1	0.1	----	----

Table 4.1: Comparison between numerical schemes.

The results of table 4.1 can be completed by the graphical representations of figures 4.11 and 4.12.

Figure 4.11 shows the dispersion of the particles around the zero-distance line (red line). The particles are rendered like blue points with a positive (over) or negative (under) distance from the real data. It is clear that the best approximation is D, which corresponds to the simulation that uses the free deformation model and the explicit Runge-Kutta 4 method. In that experiment 97 % of the particles ended up being less than one voxel away from the data to recover.

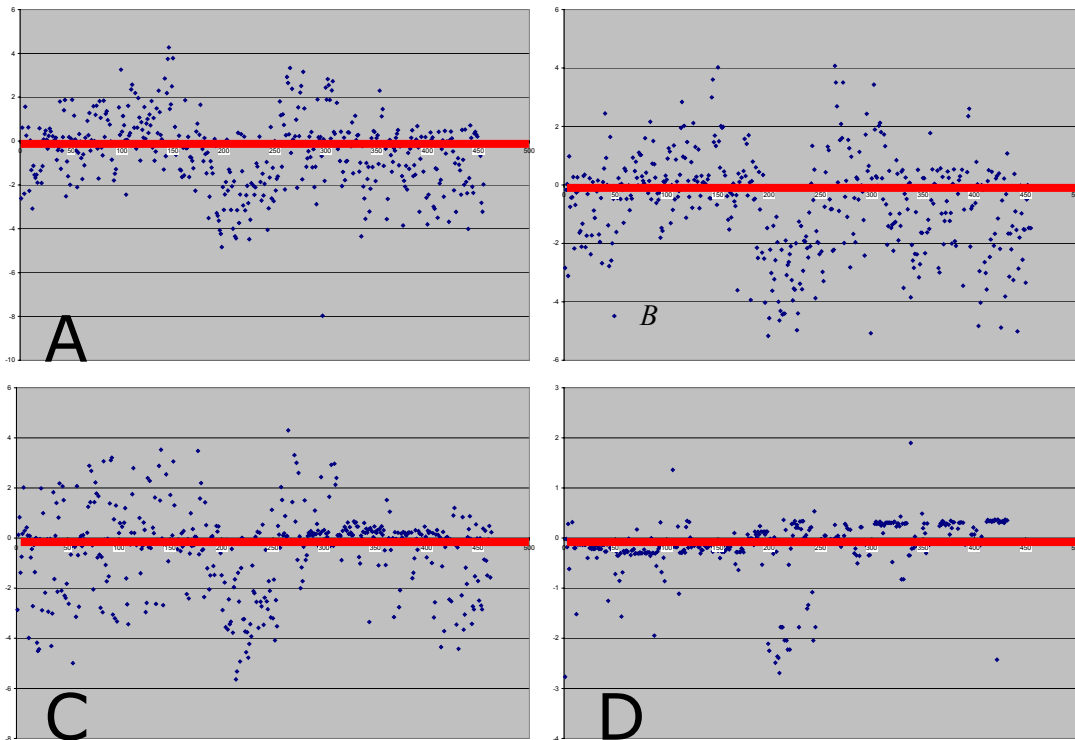


Figure 4.11: Final distances from the particles to the data to recover. A) Plain deformation method with  $\Delta t = 0.01$  s.; B) Plain deformation method with  $\Delta t = 0.05$  s.; C) Spring-mass deformation method; D) Free deformation method.

Figure 4.12 shows the evolution of the distances while time passes.

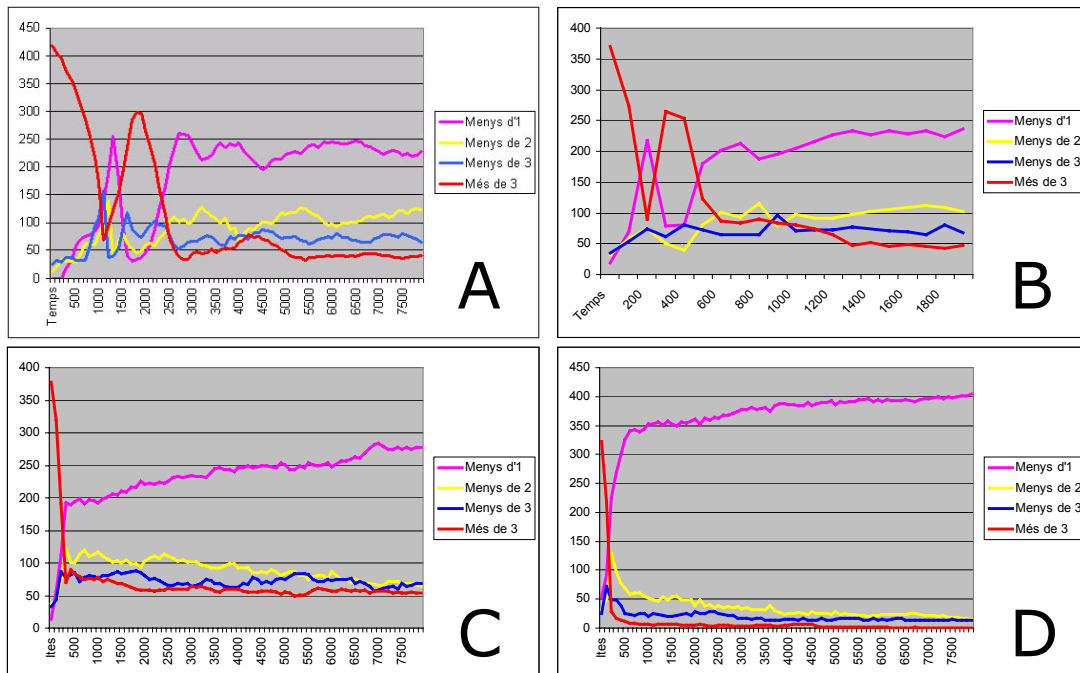


Figure 4.12: Evolution of the distances to the data to recover. A) Plain deformation method with  $\Delta t = 0.01$  s.; B) Plain deformation method with  $\Delta t = 0.05$  s.; C) Spring-mass deformation method; D) Free deformation method.



It is important to note the evolution of the particles more than three voxels away from data (red curve) and less than one voxel close (magenta curve). At the beginning there are several particles too far from the zero-distance region but as long as the system iterates, the red curve decreases at the same time that the magenta curve increases. Another conclusion to take is that the oscillations of the particles around the zero-distance region in C and D are less than in the other two simulations. At the same time, C and D get maximum levels of proximity in only a few iterations.

As a conclusion we might say that the plain de formation model is too complex for the 3D reconstructions that we are dealing with. In that sense it is not compulsory to use an implicit numerical method because the other deformation models make use of less internal forces or even none. The explicit methods give less control over smoothing parameters but are fast and efficient enough, optimizing distances and avoiding the apparition of oscillations.

Next paragraphs present two more experiments, now based on the explicit methods. All the tests were performed using a spring-mass deformation model (only stretch internal force). The parameters for the spring-mass deformation model were  $KS = 10.0$  (the weighting parameter for the stretch force),  $KSD = 1.0$  (its associated damping factor),  $KGVF = 10$  (a multiplying factor for the external force),  $KGVFD = 0.1$  (its associated damping factor). The global mass for the entire particle system was 1 kg.

The durations are expressed in seconds. All the tests were held on a Pentium III PC, with 256 MB RAM and an accelerator card (ATI Radeon 32 MB).

The voxelized data to recover consisted in a 64 x 64 x 23 world of voxels, with dimensions 2,87 mm in X and Y, 5,74 mm in Z.

Table 4.2 presents the results for the first test. In that case, the smoothing algorithm (see chapter 5) was not enabled. The idea of the experiment was to find out the maximum stepsize allowed depending on the method and on the initial mesh. The tested meshes were three: simple, medium and complex. Their characteristics are described next:

- Simple mesh: 162 vertices and 320 triangles.
- Medium mesh: 642 vertices and 1280 triangles.
- Complex mesh: 2562 vertices and 5120 triangles.

The mesh resolution (see chapter 5) affects definitely the internal force. The stretch force (see section 3.2.1 of chapter 3) depends on the elongations of the springs, which are directly related to the links between vertices. More resolution means more links and that means more internal forces to evaluate at every step. Besides that, the initial elongations get shortened as the resolution increases, and this initial elongations are the ones that the stretch force tries to maintain.

Let's start looking at the maximum stepsizes according to the mesh resolution. It is clear that as long as the resolution increases (from simple to complex), the stepsize has to be reduced in order to avoid divergences. All the solvers show the same behavior. On the other hand, the duration of the entire simulation increases if the resolution gets larger. More complicated meshes lead to longer durations because of the larger amount of vertices and links to be processed.

Solver	deltaT	Sim. Time	Volume	Mesh
Euler	0,1	0,07	60338,9	simple
Euler	0,031	1,732	66979	medium
Euler	0,007	22,697	69114,9	complex
Midpoint	0,12	0,02	62054,5	simple
Midpoint	0,031	1,265	68304,6	medium
Midpoint	0,007	46,191	67917,3	complex
RK4	0,172	0,06	62308,2	simple
RK4	0,043	1,273	67850	medium
RK4	0,01	18,342	69007,5	complex

Table 4.2: First comparison of the explicit methods.

In terms of the stepsize from a fixed mesh, the Runge Kutta 4 method allows the higher stepsize, followed by the Midpoint and Euler methods.

As a final conclusion, one can notice that the final volumes for the recovered mesh are quite similar if we fix the mesh while varying the employed solver. The simple mesh gives the most reliable volume because of the absence of degenerations. Medium and complex meshes are too detailed for the coarse quality of the voxelized data. If the triangle area is not similar to the distance between data slices, the final surface shows what is called a cuberille effect [42].

The volumes are referred to the internal cavity (endocardium) of the human left ventricle. Those measures are very important for the correct diagnosis of a patient. For each of the reconstructions, the information on volume gives the amount of blood present inside the ventricle at that moment.

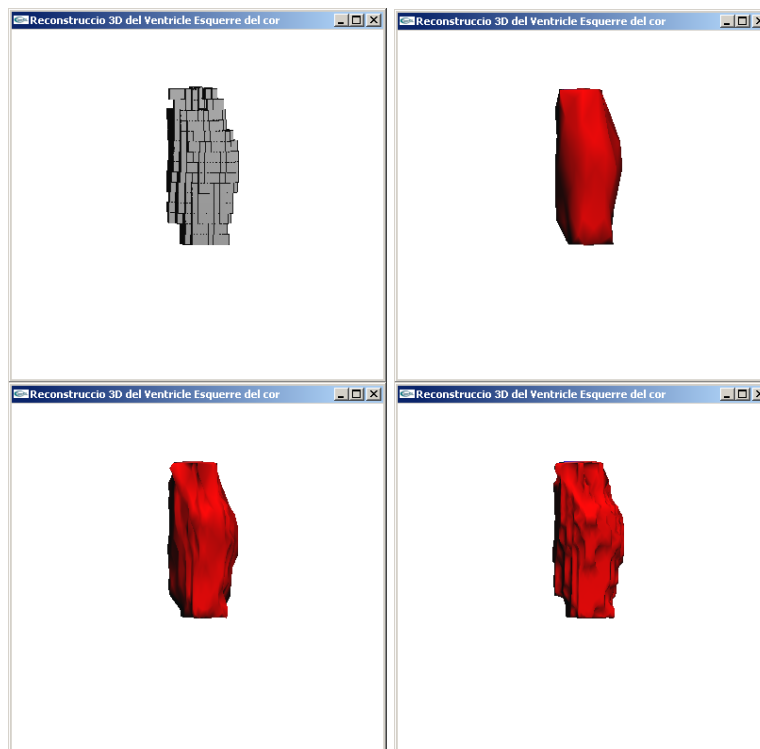


Figure 4.13: Degenerations on the final meshes due to the resolution increase. From left to right and top to bottom: the voxelized data to recover, the simple mesh, the medium mesh and the complex mesh.

Figure 4.13 shows the voxelized data and three of the final reconstructions, attending to the three resolutions of the meshes. The data is based on an actual's patient left ventricle imagery. More concretely, the meshes reconstructed the internal surface of the organ.

Table 4.3 shows the results for the second test, regarding also explicit methods. In that case, all the tests were performed using the simple mesh and the stepsize was fixed to 0,1 seconds. The borders where related to the external surface of a real left ventricle dataset.

<b>Solver</b>	<b>Sim. Time</b>	<b>Relax</b>
Euler	0,05	on
RK4	0,13	on
Euler	0,12	off
RK4	0,17	off

Table 4.3: Second comparison of explicit methods.

The smoothing algorithm was only enabled for the first two simulations. When the smoothing is disabled, the simulation durations tend to increase. Peaks appear because of the absence of the smoothing effect, and errors arise. There also differences between the methods due to the differences in evaluations that every solver performs at each iteration (one evaluation for the Euler method and four evaluations for the Runge Kutta 4 method). There is a tradeoff between the final accuracy and the duration of the simulation. In that sense the Runge Kutta 4 method gives the best results.

If the smoothing algorithm is enabled the durations decrease. As long as the smoothing algorithm corrects peaks that appear during the simulation, the reconstruction performs more fluently and quickly. The smoothing effect is especially present in the Euler method, which improves better in terms of duration. The RK4 method is refined by performing more steps per iteration, the results are better by themselves and the smoothing becomes less intensive.

## 4.5 Implementation of the external force

In order to solve the GVF external force presented in section 3.3.4, we define a temporal evolution that will guide the system until it reaches its stationary form.

We use a finite differences scheme complemented by an explicit Euler's method in order to follow this temporal evolution [104].

First of all, the variable  $t$  is introduced in the system as a parameter. Its evolution is planned as follows:

$$\begin{aligned}
 \frac{\partial u}{\partial t} &= \nabla^2 u(x, y, z, t) - b(x, y, z) \cdot u(x, y, z, t) + c^1(x, y, z) \\
 \frac{\partial v}{\partial t} &= \nabla^2 v(x, y, z, t) - b(x, y, z) \cdot v(x, y, z, t) + c^2(x, y, z) \\
 \frac{\partial w}{\partial t} &= \nabla^2 w(x, y, z, t) - b(x, y, z) \cdot w(x, y, z, t) + c^3(x, y, z)
 \end{aligned}
 \tag{4.47}$$

where the expressions of  $b$ ,  $c^1$ ,  $c^2$  and  $c^3$  are:

$$\begin{aligned}
 b(x, y, z) &= \left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2 + \left(\frac{\partial I}{\partial z}\right)^2 \\
 c^1(x, y, z) &= b(x, y, z) \frac{\partial I}{\partial x} \\
 c^2(x, y, z) &= b(x, y, z) \frac{\partial I}{\partial y} \\
 c^3(x, y, z) &= b(x, y, z) \frac{\partial I}{\partial z}
 \end{aligned} \tag{4.48}$$

If we discretize the expressions, we obtain the final implicit equations. For the first component of the vector field,  $u$ , we have that:

$$\begin{aligned}
 u_{i,j,k}^{n+1} &= (1 - \mathbf{b}_{i,j,k} \cdot \Delta t) u_{i,j,k}^n + r (u_{i+1,j,k}^n + u_{i,j+1,k}^n + \\
 &\quad + u_{i,j,k+1}^n + u_{i-1,j,k}^n + u_{i,j-1,k}^n + u_{i,j,k-1}^n - 6u_{i,j,k}^n) + \\
 &\quad + c_{i,j,k}^1 \Delta t
 \end{aligned} \tag{4.49}$$

where:

$$r = \frac{\mu \Delta t}{\Delta x \Delta y \Delta z} \tag{4.50}$$

There are analogue expressions for the rest of vector components ( $v$  and  $w$ ). Because of the finite differences scheme, the convergence to the solution restricts the maximum stepsize applicable (Courant-Friedrichs-Lewy condition). Then we must select a stepsize that satisfies the condition:

$$\Delta t \leq \frac{\Delta x \Delta y \Delta z}{4\mu} \tag{4.51}$$

Figure 4.14 shows two examples (divergence on the left and convergence on the right) for the method. In the left simulation, the central object is a voxelized sphere. Prior to the 3D reconstruction of its external surface, the GVF external force is computed. If the stepsize is not tuned according to equation 4.51, the results can be incorrect as shown. Note how the vectors are totally unorganized, especially in the four corners of the world of voxels. On the contrary, the right simulation was performed according to equation 4.51 and the results were correct as expected.

Satisfying the Courant-Friedrichs-Lewy condition is not a big issue. It defines a top value that depends on the smoothing parameter  $\mu$ . We normally begin by selecting the value for  $\mu$  depending on the smoothness that we need for the vector field. Take into consideration that  $\Delta x$ ,  $\Delta y$  and  $\Delta z$  are fixed values of the world of voxels. The voxels are created from medical images with a fixed measurement for the pixels. Then we only

need to use a suitable, which means under the top defined by equation 4.51, value for  $\Delta t$ .

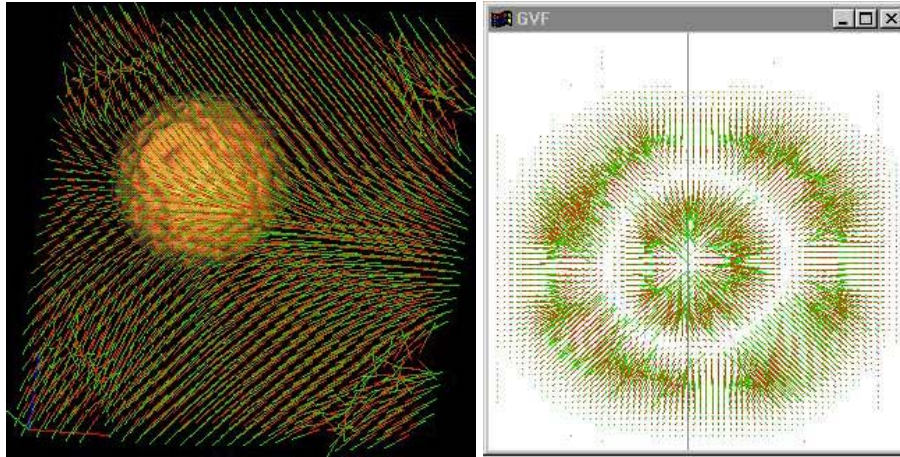


Figure 4.14: Wrong (left) and correct (right) results for the GVF external force.

## 4.6 Summary

This chapter shows the need of solvers in order to perform discrete simulations in a computer. The state of our virtual scenery, let us say the trajectory that the particle system has to follow, has to be simulated step by step by solving several derivatives. We choose an initial stepsize and let the system evolve from the beginning.

Particle systems can be simulated by Newtonian dynamics and these physical laws are represented by second-order ordinary differential equations. Second-order ODE's are frequently split in two first-order ODE's that are solved by several methodologies available. Our simulations are classified as initial value problems, where the initial position of all the particles is known for a given time. From that value we proceed to the calculation of the whole trajectory.

Numerical methods can be explicit or implicit. Explicit methods are typically simpler and easy to solve. Nevertheless several steps must be taken per iteration in order to ensure high degree accuracies. Implicit methods ensure a better solution but must be supported by iterative algorithms or even by other explicit solvers. This is due to the fact that implicit methods are based in an algebraic formula that has to be solved.

This chapter describes several explicit (simple Euler method, Midpoint method and Runge-Kutta 4 method) and implicit (Backwards Euler method) methods. It also presents a summary on adaptive stepsizes, useful for sceneries where the external conditions change continuously, making an adaptation necessary in order to speed the process while maintaining overall accuracy.

The implementation of the deformable models presented in chapter 4 is also described. The method is chosen depending on the existing internal forces for every model. More complex models lead to the need of better solvers. The plain deformable model is implemented by an implicit Backwards Euler method. The spring-mass, restricted spring-mass and free deformation models are implemented by a Runge-Kutta 4 method.

The methods are compared using real data and practical results. The study concludes that implicit solvers better simulate deformation models where several internal forces apply. However, less force intensive models like the spring-mass based or the free deformation model, can achieve good results with an explicit solver where the stepsize has been accurately selected.

The external force implementation is also demonstrated. It can be obtained by an iterative algorithm that uses a finite differences scheme complemented by an explicit Euler solver.

Chapter 5 introduces the geometrical representation used within this project. In order to perform the simulations, a geometrical surface must be defined in terms of resolution, initial size and topology. Next chapter presents several options available and illustrates the reasons that drove us to select our triangulated mesh.

## **5 Model geometry**

The geometry of the model is one of the main restrictions to the available options. Depending on the type of representation we restrict the complexity of the shape and we must choose one or another methodology. Using triangles, quads or hexagons, explicit or implicit functions, continuous or discrete approximations is a tough decision that has to be taken conscientiously.

This chapter presents the possibilities that have been analyzed and compared in order to select a suitable geometrical model for our specific needs.

### **5.1 Classification of geometrical models**

The geometry of the model describes the shape parameters that among others, like the deformation parameters, permit a realistic simulation. A basic classification distinguishes between continuous or discrete surfaces.

Continuous surfaces are characterized by an implicit or explicit equation while discrete morphologies are built from other entities like vertices or particles in a mesh. Table 5.1 summarizes the classification [71].

Attending to the classification in table 5.1, our geometrical model can be considered as an hybrid of several categories. In fact, our model can be described as a discrete triangulated mesh that is treated as a particle system in terms of evolution. It can be simulated by several deformation patterns like the ones described in section 3.4 of chapter 3. One of the patterns is the spring-mass deformation scheme, also summarized in table 5.1.

A triangulated mesh can be easily handled by code and most of the accelerator cards in the market are specifically designed for it in terms of overall performance of the computation and the rendering. Moreover that, meshes are easy to map into particle systems where each of the vertices can be treated like a particle in a direct way.

As long as our deformable model is based on a Newtonian scheme, that is built from particle systems, and taking into consideration the hardware functionalities of the accelerator cards in typical workstations, we decide to use the geometrical model described before.

For the seek of clearness, let us present a suitable data structure that maintains a discrete triangulated mesh. A set of crossed lists (vertices, edges, triangles) does the job. Figure 5.1 shows the code expressed in C language.

As figure 5.1 shows, every entity has complete access to the others. It is clear that a vertex can access the edges and triangles it belongs to; every edge is composed by two vertices and belongs to several triangles; every triangle is defined by three vertices and three edges. This structure is then bi-directional in terms of access.

<b>Category</b>	<b>Explicit representation</b>
Defined by a parameter vector ( $q$ ) More parameters imply more complexity The parameters control local and global deformation	
<b>Types</b>	<b>Characteristics</b>
<b>Polynomial functions</b>	Where their continuity can be controlled at different levels like at the derivative level ( $C^1$ ) or at the smoothness level ( $C^2$ )
<b>Superquadrics</b>	Typically closed surfaces Defined by a symmetry axis No good for complex shapes
<b>Modal decomposition</b>	The <i>LOD (Level Of Detail)</i> is controlled dynamically by a decomposition in several frequency harmonics The more modes, the best detail
<b>Category</b>	<b>Implicit representation</b>
Characterized by the zeros set of a certain function $f$	
<b>Types</b>	<b>Characteristics</b>
<b>Algebraic surfaces</b>	Can be used to reconstruct unstructured sets of points in the space Not necessarily closed Require the computation of distances between vertices and data which are typically difficult
<b>Superquadrics</b>	Implicit versions of the explicit surfaces commented before
<b>Hiperquadrics</b>	An extension of Superquadrics acting as a refinement Must be homeomorphic to a sphere
<b>Level Sets</b>	Embedded in a higher dimension space Poor interaction and computationally expensive
<b>Category</b>	<b>Discreet meshes</b>
Several vertices connected with each other	
<b>Types</b>	<b>Characteristics</b>
<b>Discreet contour</b>	Vertices connected
<b>Triangulation</b>	A discreet contour with some kind of neighborhood relation between the vertices Topological constraints can be defined
<b>Spring-mass</b>	Every vertex acts as a particle defined by its mass, position and velocity Particles are joined together by links that act as physical springs
<b>Simplex meshes</b>	With constant connectivity between vertices (three neighbors per vertex) Based on hexagonal shapes Dual to triangulations
<b>Category</b>	<b>Particle systems</b>
Based on Newtonian evolution schemes Forces and energies maintain the cohesion of the entire model	

Table 5.1: Classification of geometrical models.

Besides that, it can be easily embedded into a particle system scheme where vertices are particles and edges might be treated like springs. In fact it is only necessary to add some parameters to the structure like the associated mass and velocity for a vertex (we already save the position) and the rest length and the associated constants for an edge that acts like a spring.

It is important to note that the triangulation ensures the robustness of the mesh. Vertices are joined together via their topology relationship. If no topology or external mechanism is provided, we might obtain poorly results after the reconstructions. Figure 5.2 shows an actual example.



```

typedef struct{
    int id; // Vertex ID
    float position[3]; // XYZ values for the vertex
    int triangleCounter; // It belongs to "triangleCounter"
                        // triangles
    int *triangles; // Indexes of the triangles
    int edgeCounter; // It belongs to "edgeCounter" edges
    int *edges; // Indexes of the edges
    float normal[3]; // Normal vector associated to the vertex
}vertex; // A vertex

typedef struct{
    int id; // Edge ID
    int vertexs[2]; // Two vertices for this edge
    int triangleCounter; // It belongs to "triangleCounter"
                        // triangles
    int *triangles; // Indexes of the triangles
}edge; // An edge

typedef struct{
    int id; // triangle ID
    int edges[3]; // Three edges for this triangle
    int vertexs[3]; // Three vertices for this triangle
    float normal[3]; // Normal vector associated to the
                    // triangle
}triangle; // A triangle
    
```

Figure 5.1: C structure for a triangulated mesh.

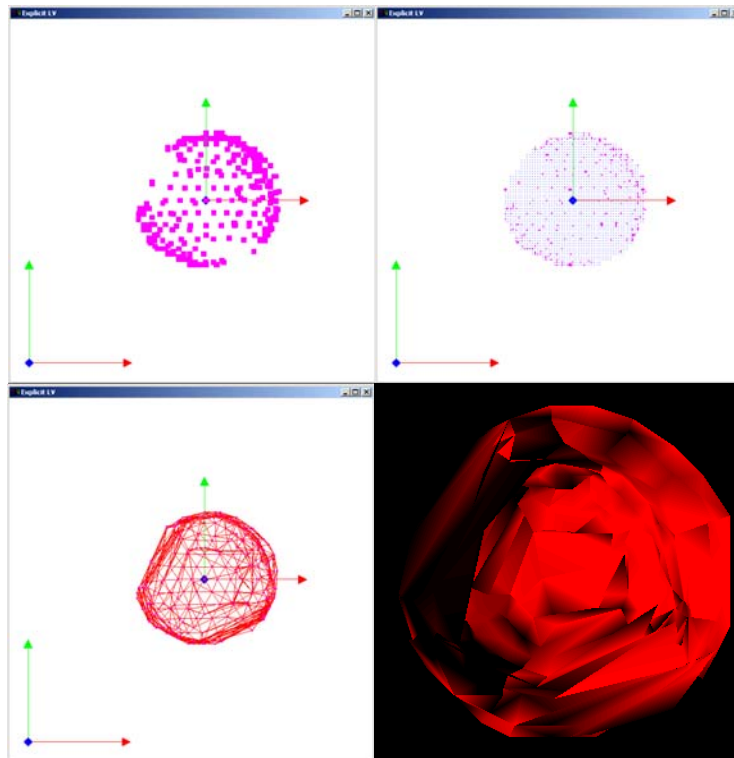


Figure 5.2: Top view of the poorly results due to failures in the topology.

In figure 5.2, from left to right and top to bottom: the vertices after the reconstruction, the vertices superimposed to the dataset, the triangles associated to the vertices and a final render using flat shading.

Figure 5.2 shows the results after a reconstruction using the free deformation model (see section 3.4.5) where no topology relationship was provided between vertices. Moreover that, there were no external mechanisms controlling the simulation and ensuring its quality. External mechanisms might be stopping processes that would ensure that the vertices stop when reaching the dataset or smoothing algorithms like the one presented in section 5.5.

The vertices of this reconstruction ran freely trying to track the dataset. All of them reached it but did not stop when they should. Due to the tangential components of the external vector field (see section 3.3.4), the vertices (particles) kept moving. That behavior resulted in an unorganized cloud of points with different densities depending on the region. If we render their associated triangles (for rendering purposes only, no present in the simulation) we perceive the low quality of the final mesh, totally degenerated.

## 5.2 Quality of the triangulated mesh

The final 3D model must contain information on surface and volume. As a second process which is not within the scope of this project, it will be necessary a tetrahedralization process between both the internal and external surfaces (see chapter 6) in order to allow real-time interaction with the model. This process must associate internal triangles with their external neighbors. Then it is imperative to ensure their quality by avoiding possible degenerations.

The degree of degeneration for a triangle can be easily computed. A good measure of this is the quotient between its area and the magnitude of the biggest edge (aspect ratio). If this quotient is near to zero, the triangle might be presenting degenerations. Figure 5.3 shows two examples on this. The blue triangle on the left has got a degeneration measure of 1.5 while the red triangle on the right, which is definitely more degenerated, has got a measure of 0.25, clearly nearer to zero than the other.

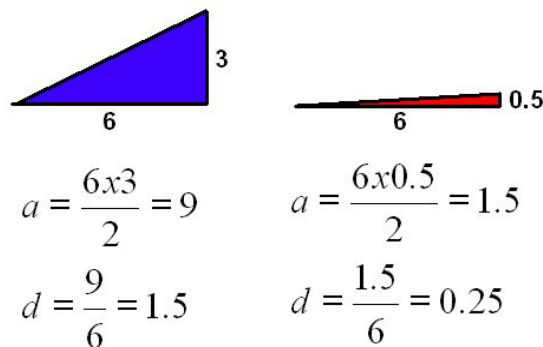


Figure 5.3: Measure of the degeneration in triangles. Values  $a$  and  $d$  stand for area and degeneration respectively.

It becomes clear that the quality of the triangles in the final mesh is important. Moreover that, this needed quality determines in some way the optimal size for the initial triangles of the mesh. A very detailed mesh will recover the data very closely but will fail in terms of final smoothness and degeneration of the triangles. On the contrary, if the size of the initial triangles is greater or equal to the separation between the data slices (the vertical size of a voxel), we will ensure a good aspect ratio for them.

For creating the initial mesh, we use a sphere centered at the origin. We then evaluate the center of mass and the bounding box of the data to recover. These parameters are used in order to determine the suitable transformations that will make the sphere surround the data. These transformations are a translation and a scaling. After their application to all the vertices of the centered sphere, we do have the initial mesh ready for the simulation.

We have used three different spherical meshes attending to their differences in resolution (number of triangles). Table 5.2 shows their characteristics.

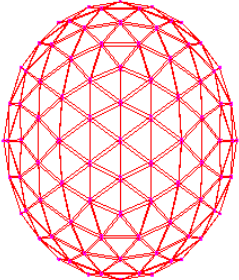
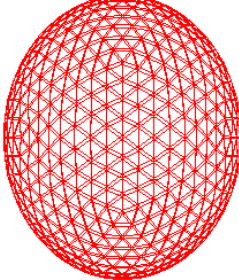
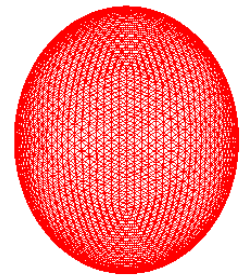
Mesh #	1	2	3
Render			
# vertices	162	642	2562
# triangles	320	1280	5120
Averaged areas / degeneration	133.38 / 6.95	33.82 / 3.49	8.48 / 1.75
Max. area / degeneration	183.87 / 8.28	47.85 / 3.96	12.2 / 1.99
Min. area / degeneration	111.8 / 6.25	28.18 / 3.12	7.05 / 1.55

Table 5.2: Initial triangulated meshes.

Areas are expressed in  $\text{mm}^2$ . The format # / # corresponds to the area value followed by the normalized area or aspect ratio, as commented before. It can be seen that the triangles of the first mesh are much bigger than the other. There are no degenerations for the three meshes in their initial situations as the numerical values depict.

The amount of triangles is static so that no one will be created nor destroyed during the simulation. That means that the topology of the geometrical model is not altered.

Table 5.3 shows the same quantities that table 5.2 but after the reconstruction. This test consisted on the recovering of a Phantom dataset (see appendix D) by using the simple (1), medium (2) and complex (3) meshes, attending to their resolutions. All the reconstructions where performed by using the free deformation model (see section 3.4.5). The 3D graphics show the meshes with flat (left) and smooth (right) shadings.


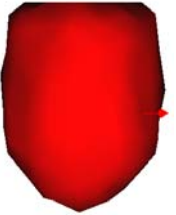


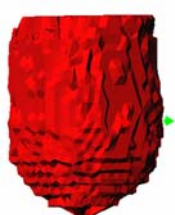
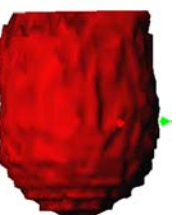
	Flat	Smooth	Averaged areas / degeneration	Max. area / degeneration	Min. area / degeneration
1			84.03 / 5.1	170.02 / 7.73	23.15 / 1.99
2			22.5 / 2.57	116.4 / 5.24	0.82 / 0.09
3			6.24 / 1.29	72.38 / 4.95	0.0007 / 0.0005

Table 5.3: Quality of the triangles after the simulation.

Let us examine them case by case:

- In the first mesh, the final result does not present any degenerations although the minimum area has been reduced considerably (from 111.8/6.25 to 23.15/1.99).
- In the second mesh, we can perceive how the degenerations begin to appear. Looking at the minimum areas closely we see that we have changed from 28.18/3.12 to 0.82/0.09).
- In the third mesh the degenerations are very important as we can derive from the differences between the minimum and maximum areas. The renders show clearly the cuberille effect in the reconstruction due to the tiny initial triangles and the discreet nature of the dataset.

We can conclude that, as expected, the first mesh gives the best final reconstruction. This mesh is smooth enough for our purposes and presents no degeneration at all.

In order to check if there is some kind of dependence between the final quality of the triangles and the reconstruction method, table 5.4 shows some results.

Figure 5.4 shows all the areas. Real values are rendered in blue while aspect ratios are rendered in magenta. The reference to take is the initial mesh (upper-left corner). If there are important differences between the initial mesh and the analyzed case, we can conclude that we have important variations between the areas which can be understood like tiny and huge triangles in the same mesh (it might be a symptom of degeneration).

Reconstruction method	Averaged areas	Max. area	Min. area
Initial mesh	33.82 / 3.49	47.85 / 3.96	28.18 / 3.12
Spring-mass	28.7 / 3.15	41.9 / 3.7	17.8 / 1.4
Free	28.5 / 3.14	40.8 / 3.46	21.4 / 2.6
Restricted Spring-mass	29.24 / 3.18	45.85 / 3.77	21.0 / 2.65
Plain	25.77 / 2.46	90.32 / 5.92	0.76 / 0.14

Table 5.4: Final quality of the triangles attending to the reconstruction method.

All the reconstructions were performed using the second mesh of table 5.2 (1280 triangles). We present a graphical representation of this results in figure 5.4.

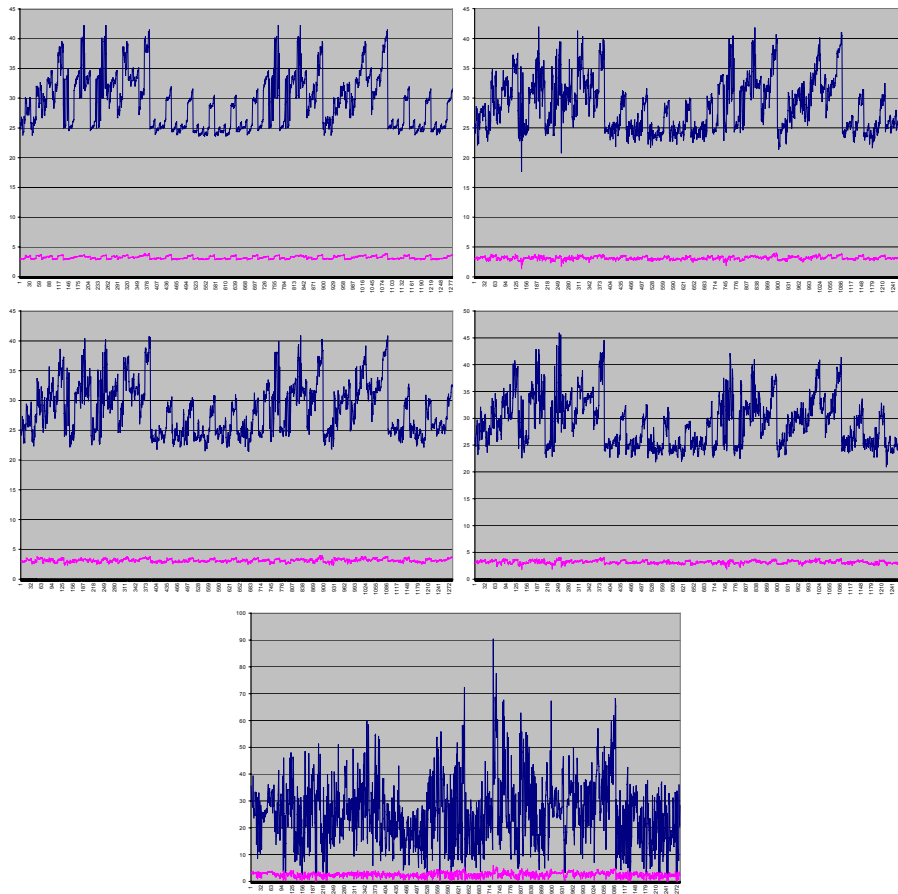


Figure 5.4: Graphical view of the quality for the final triangles. From left to right and top to bottom: initial mesh, Spring-mass mesh, free mesh, restricted Spring-mass mesh and plain mesh.

Note the poor quality of the mesh in the bottom case, when using the plain deformation method. It is not an exaggerated case but it contrasts with the rest which are mainly equal. We conclude that the final quality of the triangles depends more on the initial dimensions of the mesh, not hardly on the reconstruction method.

### 5.3 In-to-Out vs. Out-to-In reconstruction

The reconstructions provide two surfaces, one internal and the other external. The external surface can be clearly obtained by a simulation from the external initial surface that surrounds the dataset and evolves in a Out-to-In manner. The problem arises when dealing with vertices in a triangulated mesh that should evolve in a In-to-Out scheme in order to recover the internal surface. Figure 5.5 helps explaining this question.

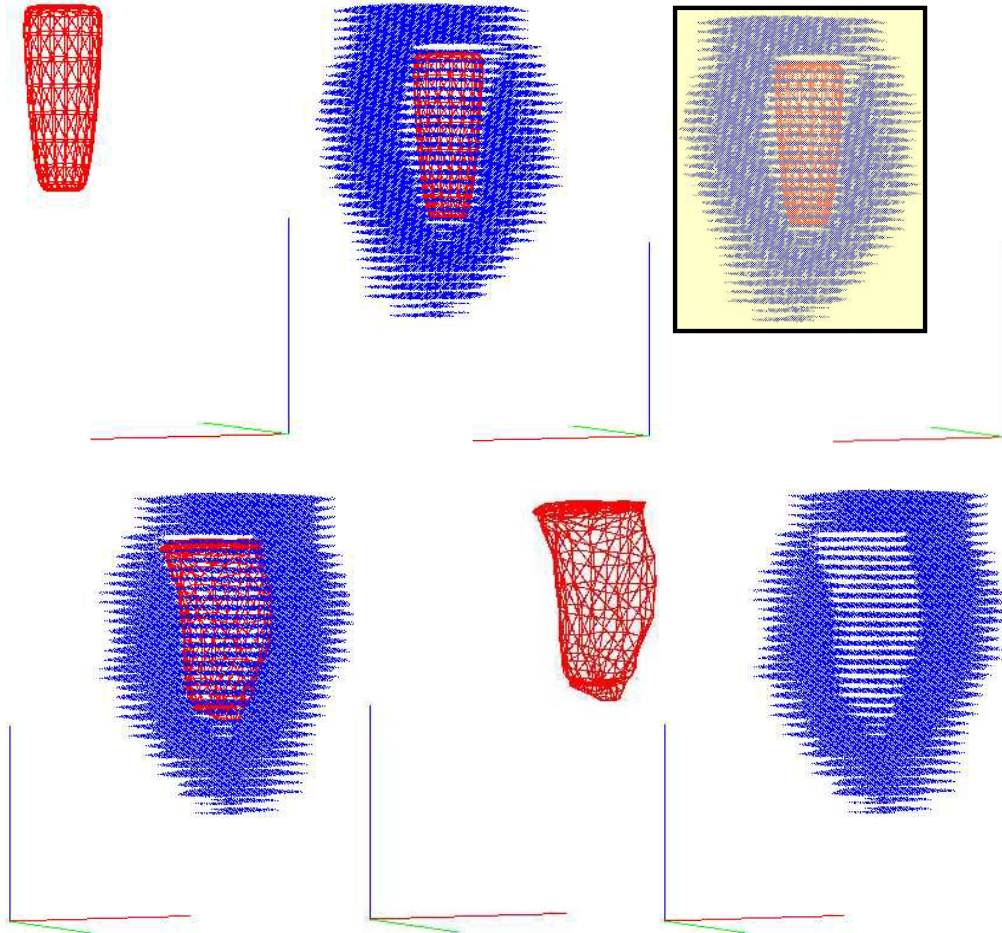


Figure 5.5: Internal mesh reconstruction.

The first row of figure 5.5 shows an actual patient dataset (blue) and the initial internal mesh that will try to reconstruct the internal cavity (red). The bounding box of the dataset is rendered as a yellow square. The second row shows the final results after the reconstruction of the internal cavity.

In order to initialize the mesh we have two possibilities:

- Doing it by hand, which is the case of figure 5.5. This methodology slows down the whole process which we want to be as automatic as possible.
- Translating and scaling the mesh accordingly. For the internal mesh, the translation is easy and corresponds to the position of the center of mass of the dataset. The problem arises when having to scale it. The bounding box of the dataset gives clues



on the initial dimensions for the external initial mesh but no information on the inner cavity.

As long as we want to use the second approach, we need an alternative methodology for the inner mesh initialization. Let us deduce a second reason for this.

When using the In-to-Out approach, we can have several reconstruction artifacts. Table 5.5 shows two experiments about the inner surface retrieval. The only difference between both tests is the resolution of the initial mesh (320 triangles for the first test and 1280 for the second).

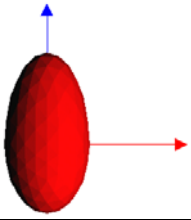
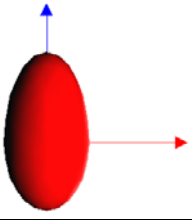
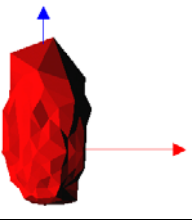
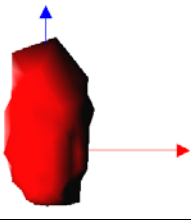
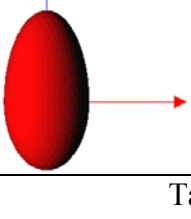
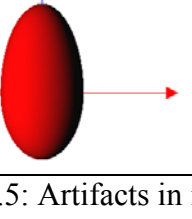
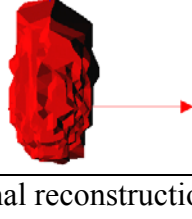
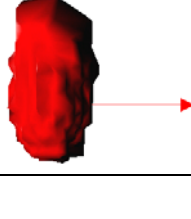
Test	Initial mesh (flat)	Initial mesh (smooth)	Final mesh (flat)	Final mesh (Smooth)
1				
2				

Table 5.5: Artifacts in internal reconstructions.

Although the initial mesh has been carefully positioned inside the cavity of the dataset, the final results should be much better. Those are much detailed in the second test but we still perceive artifacts in the recovery. The differences in the sizes of the triangles are important. We can see by simple visual inspection that the top area of the mesh is poorly detailed while the bottom zone has been covered by lots of tiny triangles.

There is a clear reason for this, regarding the quality of the external force inside the dataset (see section 3.3.4). The external force can not be derived correctly there because of the lack of physical space. Besides that, the vertices in the mesh are not uniformly attracted making the final triangles to be very different. Among all of these, triangle degeneration can be best controlled when decreasing than when increasing its area.

We see that the In-to-Out approach presents several problems related to the initial adjustments and posterior evolution. Those corrupt our intentions to automate the process while obtaining a robust final result. In order to overcome this, we propose to use the Out-to-In approach even if reconstructing the internal surface. That means beginning as in the external reconstruction case but using a second external force, derived only in the vicinities of the internal borders. More details on these can be found in chapter 3.

## 5.4 Our model and the Marching Cubes algorithm

The Marching Cubes algorithm [55] has been extensively used in the context of medicine among others. It was developed by Lorensen and Cline in 1987 and it is always used as the first 3D segmentation algorithm to compare with.

The Marching Cubes algorithm extracts the surface that corresponds to a defined level of property in the given dataset. We might name it as an isosurface because of the equal values of property everywhere inside its domain. It works in a voxel basis, like in our approach, by classifying the voxels attending to the property values at their associated vertices. Once the voxels have been classified, the internal topology can be identified and the vertices can be interpolated in order to get the triangle mesh that best satisfies our needs.

Figure 5.6 summarizes its behavior in a 2D case, for a better understanding. It is the so-called Marching Squares algorithm [2], a simplification of the Marching Cubes approach.

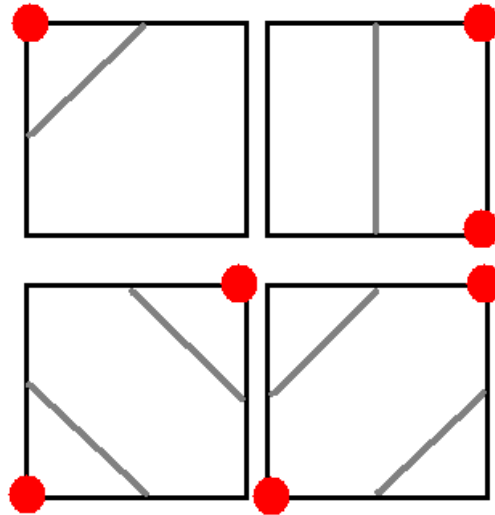


Figure 5.6: Three cases of the Marching Squares algorithm.

Figure 5.6 shows in red the vertices that verify the property value that we are looking for. That is that their associated property value is over (or under if we prefer) some desirable bounds. Blank vertices do not accomplish this fact. Then it is easy to see that the contour (or surface in 3D) that separates the vertices over the threshold from the vertices under it, must be the gray line. The top cases are pretty clear but the bottom row presents an ambiguity: the vertices are the same but the contour might have two different representations. In fact, this is the main problem of the Marching Squares, and by extension, the Marching Cubes algorithm: the existence of ambiguities that might induce the apparition of holes.

Moreover that, the Marching Cubes algorithm generates an elevated amount of triangles that must be simplified afterwards in order to provide a quick and real-time response to user interaction, for instance.

Several attempts have been presented in order to solve these defects. A well-known one is the Discretized Marching Cubes approach of Montani et al. [73]. This approach



focuses on the reduction of generated triangles besides the elimination of some ambiguities. In fact, the algorithm completes the original cases with several more that avoid the apparition of holes in the final surface.

However there are still a couple of problems hard to handle by this kind of algorithms:

- First of all, the existence of holes in the original dataset. If the data to recover is incomplete, those algorithms will recover an incomplete surface.
- Secondly, we are mainly interested in the recovering of two different, and separated, surfaces. Those algorithms would recover a unique surface if we take into consideration that the left ventricle acquisitions are opened at the top. See figure 5.7 for more details on this.

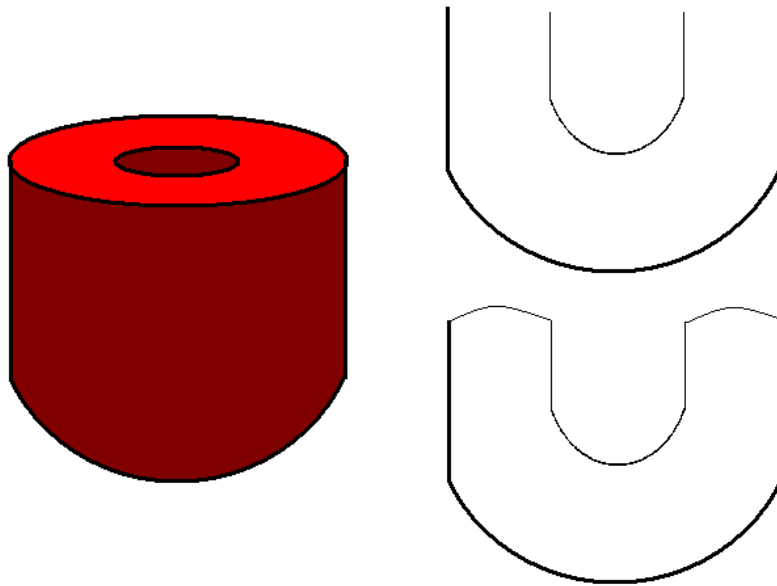


Figure 5.7: Cross sections of the recovered surfaces for the Phantom volume (see appendix A).

Figure 5.7 shows the cross sections of the surfaces recovered by two different methodologies. In the first case (top-right), the method finds two different surfaces which are unrelated to each other. The second case (bottom-right) consist on a method like the Marching Cubes algorithm, that finds a unique surface that wraps up the entire dataset.

If the original data presents holes or let us say regions with absence of acquisition, the Marching Cubes algorithm does not provide the results that we are looking for.

Figure 5.8 shows an artificial dataset where there is a clear absence of data in the middle region. However our methodology “interpolates” the blank space due to the topology restrictions that exist between the vertices. The edges and then the triangles “save” this situation from a clear failure. Isosurface based algorithms like the Marching Cubes algorithm would find two different surfaces, one at the top and the other at the bottom of the dataset.

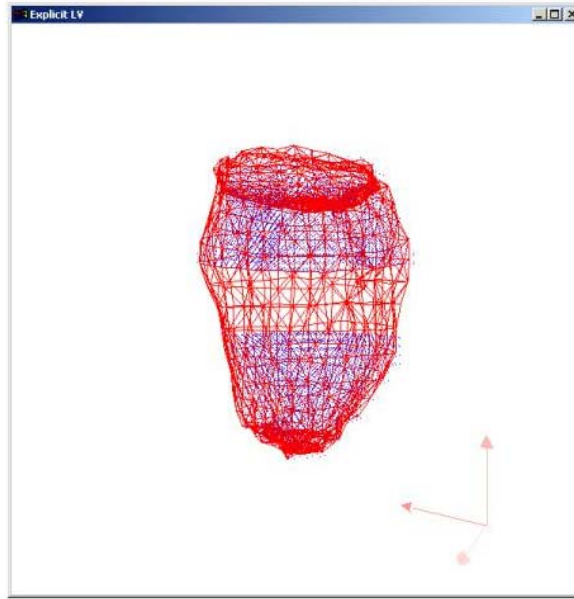


Figure 5.8: Reconstruction of an artificial left ventricle, with absence of data.

There is still a final reason for avoiding the use of isosurfaces. It is very difficult to know the exact value that has to be taken as a property threshold. Moreover that, this value might be different from one acquisition, or dataset, to another. Different patients have different tissue characteristics and different blood densities that make the obtained images to be different, in terms of gray level. We should work attending to gradient changes and not to absolute property values.

Despite of all the reasons stated before, we performed a test that consisted on using a Marching Cubes isosurface as the initial mesh for our reconstruction method. Given a complete dataset, we defined a threshold value that was used to obtain an isosurface. This isosurface was used as the initial mesh that would adapt to the data. The final results were compared to the same reconstruction by using a transformed sphere as the initial mesh. Table 5.6 presents the results and figure 5.9 shows some conclusions graphically.

Mesh	Max. initial area	Min. initial area	Averaged initial area	Max. final area	Min. final area	Averaged final area	% less than one voxel far
Sphere	41.8	24.6	29.95	43.23	23.14	29.92	85%
MC's	20.5	0.25	10.5	22.12	0.2	10.8	83%

Table 5.6: Comparison of results attending to the initial mesh (Scaled Sphere vs. Marching Cubes)

It is clear that the initial meshes are different in the sense that their areas differ. Moreover that, the distance between the maximum and minimum areas is much bigger in the isosurface than in the transformed sphere. After the simulation, the values are basically maintained within the same magnitudes. The reconstructions were equally successful because nearly the same percentage of vertices ended up being less than one voxel far from the data to recover.

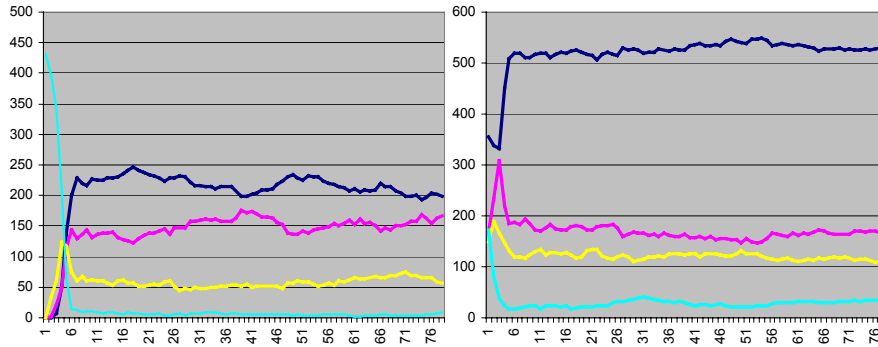


Figure 5.9: Graphical evolution of the vertices attending to their distance to real data.

Figure 5.9 shows the evolution of the vertices in terms of their distance to the real dataset. The left graphic stands for the simulation using the transformed sphere while the right graphic shows the results of the simulation using the isosurface. Both simulations were performed using the spring-mass deformation method with stepsizes below 0.001 seconds that ensured their convergence. No automatic stopping mechanism was used and no smoothing algorithm was applied. The blue line stands for vertices less than one voxel far from data, the magenta line for less than two voxels, the yellow line for less than three voxels and the cyan line for voxels far away from three voxels.

A first analysis shows that just at the beginning, the sphere has no vertices near the dataset while the isosurface begins with more than 300 vertices really close to data. This is due to the proximity of the isosurface to the dataset and demonstrates that the property threshold was correctly selected.

Both simulations present a quick and big decay in the cyan line which means that the vertices came close to data in a very fast way. Nevertheless the isosurface ends up with more vertices far away (cyan line) although it gets more vertices really close to data than the sphere. As expected, the enormous amount of vertices that starts the simulation very close to data helps in that question.

## 5.5 The smoothing algorithm

We are dealing with the reconstruction of low-resolution data, where the final results can be poorly detailed because of its discreet nature. We use an algorithm that is applied to a world of voxels in order to “cover” the low-resolution data, getting the shape. After this reconstruction process, we might need to smooth the mesh. We propose an easy-to-code smoothing algorithm for 3D reconstructed surfaces. The algorithm has been recently published in [50].

The smoothing algorithm is geometrically based. There is no dependency on the reconstruction method previously used. Thus, the algorithm can be always applied without any information from the previous process.

The algorithm can be useful in all contexts where a mesh needs to be smoothed: in terrain generation, fluid simulation, automated reconstruction applied to virtual surgery, etc.

### 5.5.1 About the input mesh

The algorithm takes as its input a mesh to be smoothed. In order to facilitate the coding, it is highly recommended to build an intelligent data structure in the sense that every item should be able to access the others easily. It might be enough to use a set of crossed lists where every triangle has access to its edges and vertices; every edge has access to its vertices and to the triangles it belongs to, and every vertex has access to the edges and triangles it belongs to. Figure 5.1 shows a suitable structure.

The algorithm is performed in a vertex-by-vertex manner. This means that we will need to go from vertices to edges and triangles.

We need to take in consideration the contour conditions at the corners and borders of the mesh. The algorithm assumes that we are working with a closed and convex mesh. If this is not the case, the programmer will have to control the singularities separately.

### 5.5.2 Motivation

Figure 5.10 presents a reconstruction example in our context of a medical application, the reconstruction of the human's left ventricle of the heart. In this case, our reconstruction method (see section 3.1) has been applied to an initial dataset.

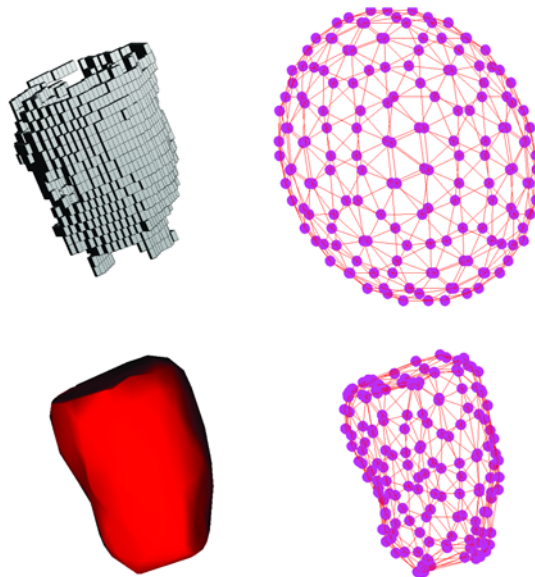


Figure 5.10: Top-Left) Voxelized data to recover. Top-Right) Initial mesh before the reconstruction. Bottom-Left) Illuminated final mesh. Bottom-Right) Wire frame final mesh.

Reconstruction methods present several problems that make it difficult to get a final “smoothed” mesh. First of all, initial data to recover lacks resolution. Voxels are formed from several superimposed slices (images). In that sense, typical resolutions for the data model are small, for example  $64 \times 64 \times 20$  or  $128 \times 128 \times 20$  voxels. Moreover, the region of interest (ROI) does not expand all along the slices. For a given slice, the ROI is typically located on a small central area, occupying  $20 \times 20$  pixels at most. See the data slices in figure 5.11.

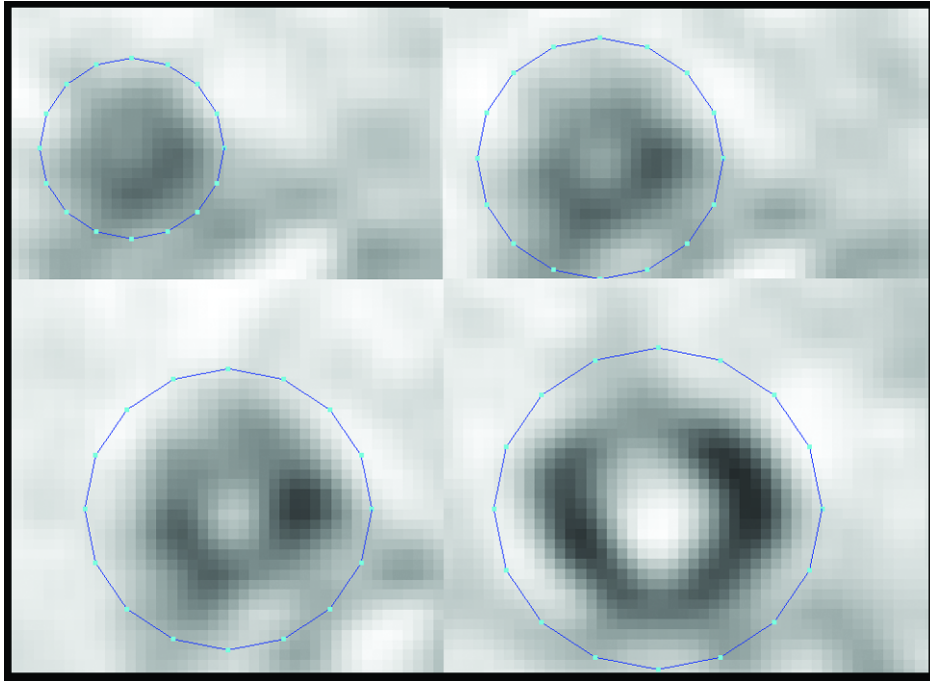


Figure 5.11: Four portions of the data slices corresponding to medical imagery from the left ventricle. The ventricle (ROI) is indicated by a circle.

In figure 5.11, each of the slices is about half a centimeter deep. This is a very coarse measure and increases the difficulties for smoothing the final result.

It is common that we lack data voxels due to failures in the capture process. In our case, we are working with Single Photon Emission Computed Tomography (SPECT) imagery (see appendix A) for getting the data from the left ventricle of the human heart. Ischemic areas, with poor or absence of blood irrigation, will not appear in the images, that means that some “holes” are expected in the dataset that the mesh will try to cover with a surface. This is another cause of errors in the meshing (peaks).

Due to the compromise between accuracy and speed, some reconstruction methods do not apply internal forces, which means that there is no restriction on curvature or local smoothness. This is the case of the free deformation model explained in section 3.4.5.

Due to all these factors, we need some kind of alternate control designed to avoid the “peaks” that appear. The mesh in figure 5.12, illustrates the smoothing comparison.

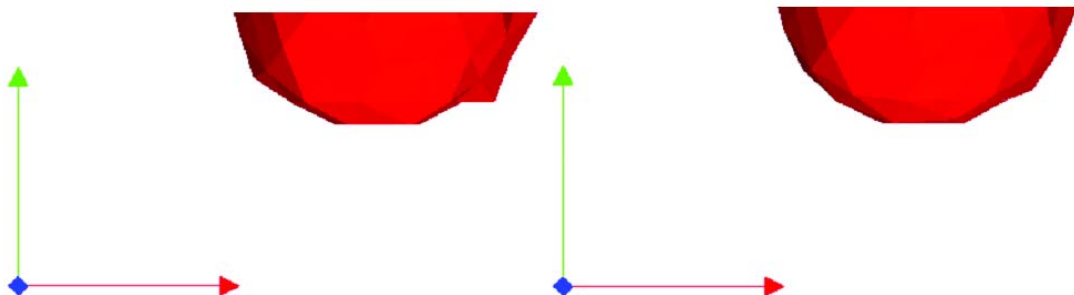


Figure 5.12: Final mesh with (right) and without (left) applied smoothing algorithm.

Figure 5.12 shows the “cuberille effect” due to the voxelized data. This effect becomes especially problematic if the resolution of the voxels is not large enough.

### 5.5.3 A prior question: the edge arrangement

In order to ensure the correct functionality of the algorithm, the edges must be classified in either clockwise (CW) or counter-clockwise (CCW) order. Mesh generation applications do not have to ensure a convention, so we have to take care of it.

This convention applies to the list of edges of every vertex. Every vertex holds information about the edges it belongs to. Those edges must be arranged because we will need to compute the angle between pairs of them as a part of the algorithm.

This task can be accomplished as follows:

- First of all, we must estimate the normal vector associated with each vertex.
  - Begin by calculating the normal vector associated with each triangle (from the plane that contains it).
  - Average all the normal vectors of the triangles that contain a given vertex.
- For each of the vertices:
  - Calculate the plane that contains the vertex. The plane is defined by the vertex’s normal vector.
  - Identify all the neighbor vertices by looking at the list of edges.
  - Project all the neighbor vertices over the plane.
  - Create the director vectors of the plane:
    - $V_1$  from the vertex to the projection of the first neighbor.
    - $V_2$  from the cross product of  $V_1$  and the normal vector.
  - Identify the plane coordinates  $(\lambda, \mu)$  of the neighbor vertices:
    - First derive a vector from the vertex to the neighbor of interest.
    - $\lambda$  equals the projection of the vector over  $V_1$ .
    - $\mu$  equals the projection of the vector over  $V_2$ .
  - Now each of the edges (vectors from the vertex to the neighbors) can be labeled with an angle derived from  $\lambda$  and  $\mu$  (see equation 5.1).

$$angle = \tan^{-1}\left(\frac{\mu}{\lambda}\right) \tag{5.1}$$

- Order the angles from the minimum to the maximum  $\Rightarrow$  ordering the edges.

Take a look at figure 5.13 for a better understanding of the edge arrangement.

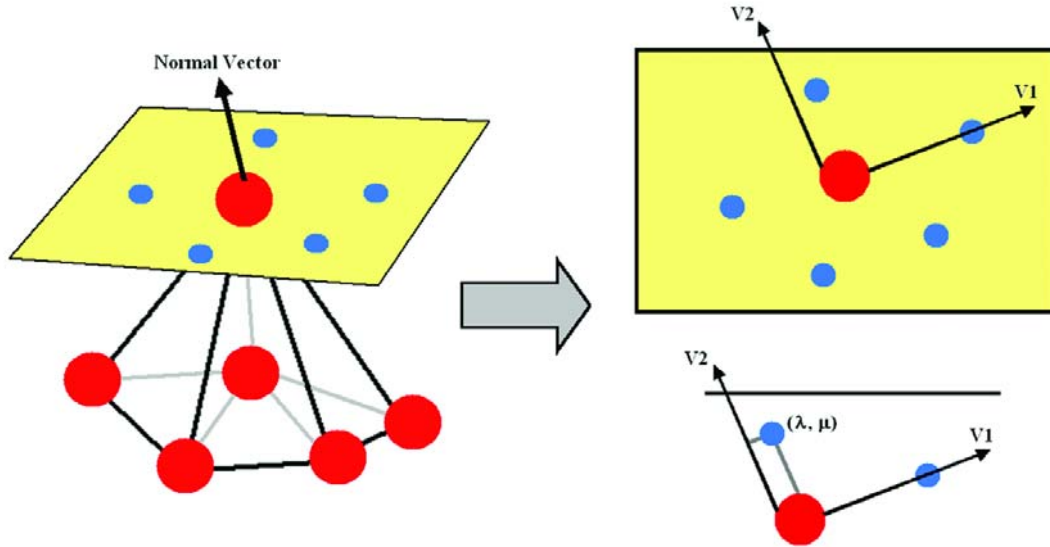


Figure 5.13: Left) The neighbor projections onto the vertex plane. Top-Right) The director vectors for the plane containing the vertex. Note that  $V_1$  goes through the first selected neighbor. Bottom-Right) The  $(\lambda, \mu)$  plane coordinates for one of the neighbors.

#### 5.5.4 The Algorithm

The algorithm penalizes vertices that are inside a non-uniform triangle set. We can see how this looks graphically in figure 5.14.

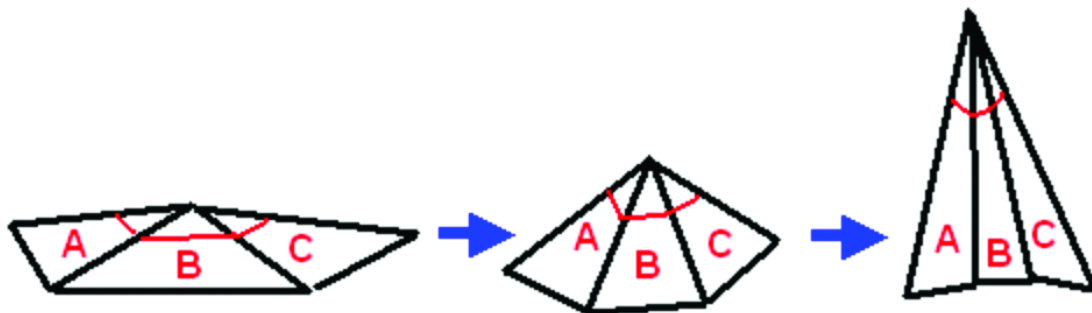


Figure 5.14: Evaluating the smoothness of a triangle neighborhood.

Figure 5.14 shows a neighborhood formed from three triangles A, B and C. The first case (left), where the sum of the three inner angles is near 180 degrees (half circle), denotes that the reconstructed surface does not present important creases. If we go from left to right, the sum of the angles decreases, far from the theoretic 180 degrees. When the sum is really tiny, we have just found a peak to be smoothed.

Smoothing the peak of the third case implies acting on the triangles until we get a situation similar to the one in the first case. We will solve the problem by applying a geometric constraint to the common vertex so that this vertex moves to the centroid of its neighbors. Figure 5.15 shows the displacement direction applied to the common vertex in order to smooth the peak.



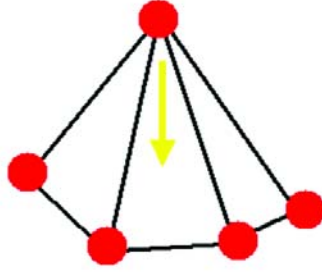


Figure 5.15: Peak correction by vertex displacement.

Then for each of the vertices in the mesh:

- Evaluate the global angle  $\theta_T$  from the contributions of all the triangles that contain the vertex. For a given triangle  $i$ , the angle is denoted as  $\theta_i$  (see equation 5.2).

$$\theta_T = \sum_i \theta_i \quad (5.2)$$

Where  $i$  goes from 1 to  $n$  (if  $n$  is the number of triangles in the neighborhood).

- Compute the error term by subtracting 360 degrees from  $\theta_T$ . If the result is bigger than our selected maximum tolerance, the algorithm will perform the vertex displacement. If not, the algorithm will skip this vertex (see equation 5.3).

$$\varepsilon = \theta_T - 360^\circ \begin{cases} \varepsilon > tol \Rightarrow disp \\ \varepsilon < tol \Rightarrow skip \end{cases} \quad (5.3)$$

- Then if the error term is bigger than the tolerance:
  - The algorithm computes the neighborhood centroid.
  - The displacement vector, from the vertex to the centroid, is also calculated.
  - The common vertex is moved along the displacement vector until  $\varepsilon < tol$ .
- Else  $\Rightarrow$  skip this vertex and go through the rest.

For the mesh in figure 5.10, the tolerance was tuned to be 0.2.

Although it seems like 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, which 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 only small changes if possible.

### 5.5.5 Computing the angle and the projections

Computing the angle between pairs of edges can be easily achieved by using the dot product. We can treat the edges as vectors. Given two vectors  $\vec{u}$  and  $\vec{v}$ , we can compute the cosine of their angle  $\theta$  as equation 5.4 shows.



$$\cos \theta = \frac{\bar{u} \cdot \bar{v}}{|\bar{u}| |\bar{v}|} \quad (5.4)$$

Similarly, the projection of  $\bar{u}$  over  $\bar{v}$ , can be found like shown in equation 5.5.

$$p_{\bar{u}} = |\bar{u}| \cos \theta \quad (5.5)$$

### 5.5.6 The displacement vector

The neighborhood centroid is evaluated by the average of the positions contributed by all the vertices sharing a triangle with the common one (see equation 5.6).

$$C = \frac{\sum_j p_j}{m} \quad (5.6)$$

Where  $m$  is the number of vertices in the neighborhood,  $j$  goes from 1 to the  $m$  neighbors,  $C$  stands for the centroid and  $p_j$  is the position for a given vertex apart from the common one. Then we can derive the displacement vector  $\bar{v}$  as shown in equation 5.7.

$$\bar{v} = C - V \quad (5.7)$$

Where  $C$  and  $V$  are the positions for the centroid and the common vertex respectively. The movement along the displacement vector is accomplished by simple linear interpolation (see equation 5.8).

$$\begin{aligned} d(\alpha) &= V + \alpha \bar{v} \\ \alpha &\in [0,1] \end{aligned} \quad (5.8)$$

In the previous equation,  $d(\alpha)$  will give the new position for the common vertex. The position is adjusted by using the  $\alpha$  parameter. Values for  $\alpha$  range from 0 ( $d(\alpha) = V$ ) to 1 ( $d(\alpha) = C$ ). As long as the algorithm moves the vertex until the tolerance is satisfied, the  $\alpha$  value must be set by the programmer. If working in the context of voxels, a suitable value can be any fraction of the smaller side of the voxel.

For the mesh presented in figure 5.10, the  $\alpha$  value was defined to be the smaller side of the voxel divided by a defined constant of 20.

### 5.5.7 Conclusion

The technique penalizes (and reallocates) those vertices that belong to a non-uniform set of triangles. The algorithm is intended to apply small changes in position, never changing the mesh topology.

Before running the algorithm, the edges must be correctly arranged. This task can be accomplished by the software used for exporting the geometry or by the algorithm, as shown in the text.

The tuning parameters for the algorithm are the tolerance ( $tol$ ) and the alpha ( $\alpha$ ) values. This text gives some clues on how to adjust them. Nevertheless, those depend on the specific application.

Figure 5.16 shows the algorithm in a very different context: the generation of synthetic terrain. This area is related to several applications like videogames, military and space simulators or weather software.

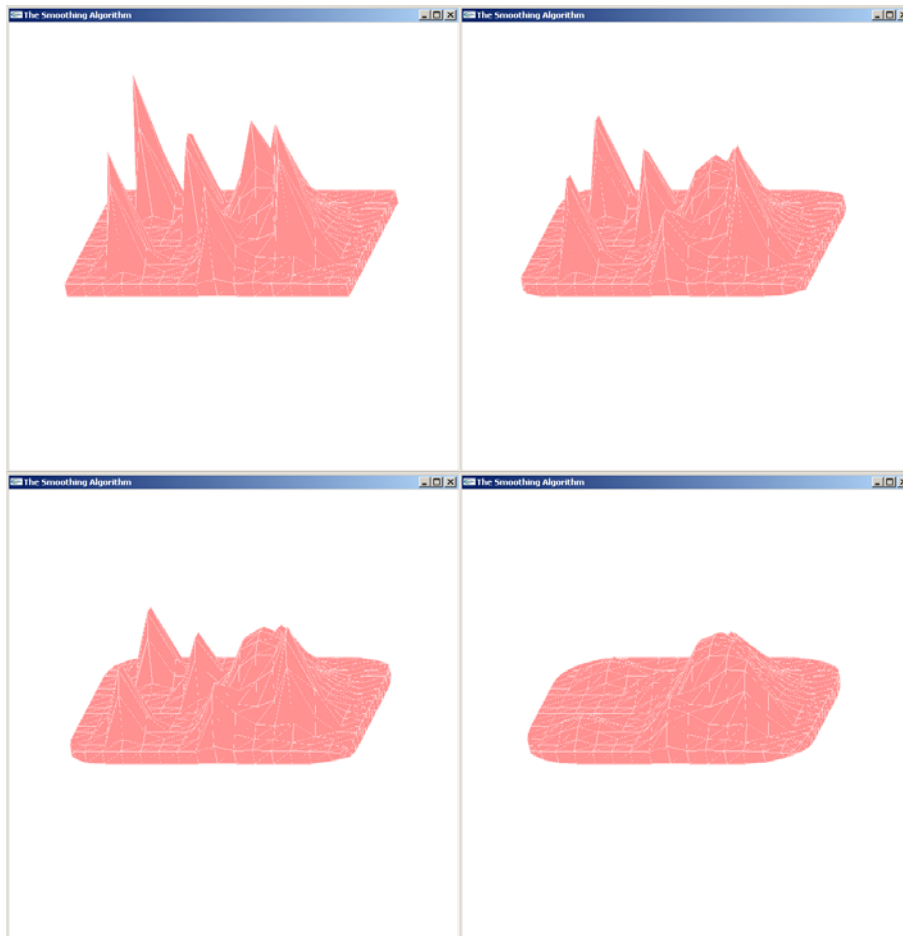


Figure 5.16: The smoothing algorithm in the context of terrain generation. From top to bottom and left to right, the terrain mesh is smoothed in order to eliminate the peaks.

## 5.6 Volume specified by a surface mesh

We might need to evaluate the volume that lies inside our closed surfaces in order to compare it with the volume that we expect to recover, for instance. In fact, the final goal of our simulations is to provide physicians with several medical values that have influence in their diagnostics and depend strongly on volume evaluations (see appendix A).

The volume determined by a closed mesh can be obtained by following the Gauss theorem as equation 5.9 states.

$$V = \frac{1}{3} \int_{Surface} (x \cdot n_x + y \cdot n_y + z \cdot n_z) ds \quad (5.9)$$

Where  $x, y, z$  and  $n_x, n_y, n_z$  stand for a given vertex position and its normal vector coordinates respectively.

This theorem evaluates the contribution of every triangle to the global volume. Figure 5.17 shows its underlying basis.

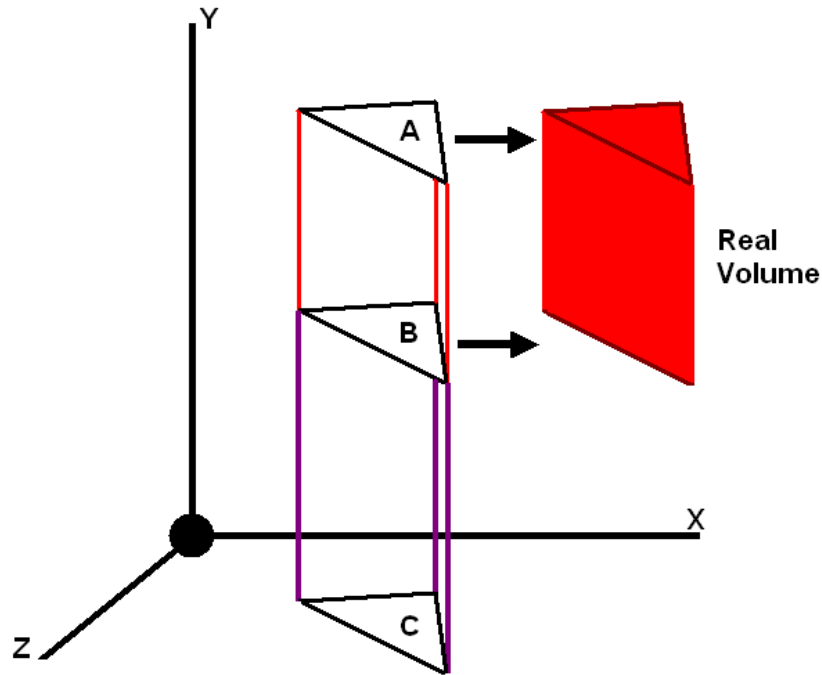


Figure 5.17: Computing the contributions on volume for a surface triangle.

Let us assume that A is a triangle that belongs to our surface. If we project it into the  $-Y$  direction, we will find two different intersections, B and C. B is the projection that lies on the bottom area of the surface. C is the projection onto the XZ plane. We need to get the volume that lies within A and B (the red volume). In order to do that, we can compute the volume related to the prism BC and subtract it from the volume associated to the prism AC. The volume of a prism is easy to evaluate since it can be obtained by an integration within the limits of its associated triangle (at the top). The orientation of the triangle can be obtained from its normal vector.

For a numerical implementation of the Gauss theorem we might follow this pseudocode:

- **Volume** = 0.0;
- For each triangle in the mesh (the triangle is formed from 3 vertices Triangle[i], Triangle[j] and Triangle[k]):
  - ➔ Evaluate its associated normal vector  $N = (n_x, n_y, n_z)$ :
  - ➔ For l from 1 to 4:
    - If l = 1,  $w = -27.0 / 96.0$ ;
    - Else  $w = 25.0 / 96.0$ ;

- $x = (M[1,1]*Triangle[i].x + M[1,2]*Triangle[j].x + M[1,3]*Triangle[k].x) / 15.0;$
- $y = (M[1,1]*Triangle[i].y + M[1,2]*Triangle[j].y + M[1,3]*Triangle[k].y) / 15.0;$
- $z = (M[1,1]*Triangle[i].z + M[1,2]*Triangle[j].z + M[1,3]*Triangle[k].z) / 15.0;$
- **Volume = Volume + w \* (x\*n<sub>x</sub> + y\*n<sub>y</sub> + z\*n<sub>z</sub>) / 3.0;**

Where M is the matrix that equation 5.10 shows.

$$M = \begin{pmatrix} 5 & 5 & 5 \\ 11 & 2 & 2 \\ 2 & 11 & 2 \\ 2 & 2 & 11 \end{pmatrix} \quad (5.10)$$

## 5.7 4D Interpolation

Time is introduced in the system when we are dealing with more than a unique acquisition of the dataset. For instance, for the reconstruction of the left ventricle we might be interested in the entire cardiac cycle of an actual patient. That is the case of figure 2.22 in chapter 2, where physicians provided us with several acquisitions for a given patient.

In those cases, all the reconstruction process is repeated for recovering the two surfaces associated to each reception. After that, the evolution through time is implemented as a linear interpolation function between meshes, based on keyframing [52]. Figure 5.18 gives some clues on this.

“ $\alpha$ ” keyframes between pairs of meshes

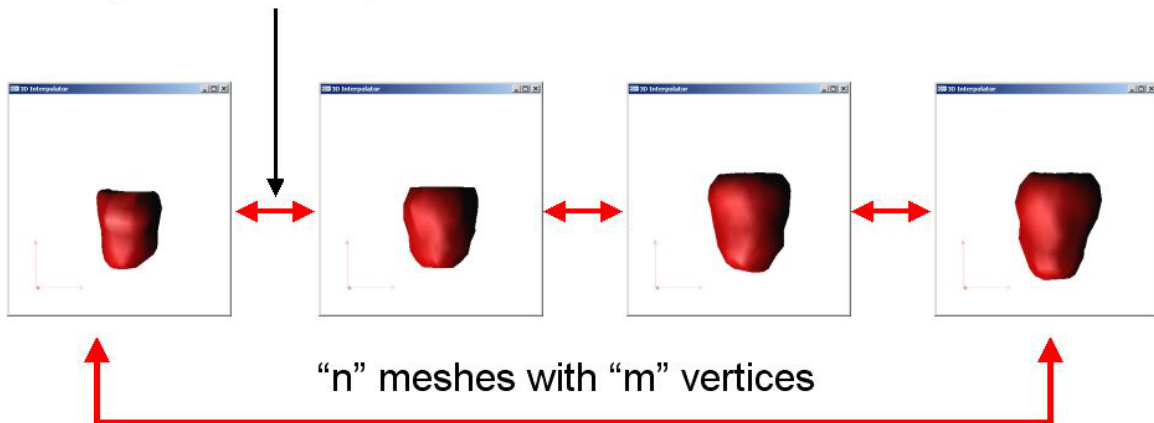


Figure 5.18: Keyframing introduces time as a fourth dimension.

The idea is to define several keys, or master surfaces, positioned at the beginning, at the end and at several positions between them. Those keys correspond to the internal and external surfaces that we have reconstructed by using our method ( $n$  meshes mean  $n$  keys). Once the keys are defined, we must create automatically the surfaces that fit within them in order to perform an animation as realistic as possible ( $\alpha$  keyframes). If

we create enough keyframes the animation will look smooth and fluent. This principle was used by Walt Disney [25] artists from the beginning of their super productions. Two or three very good artists would draw several keys and twenty or more less experienced people would emphasize the action by using keyframes.

The association between meshes is performed vertex by vertex (in figure 5.18 every mesh contains  $m$  vertices). Because all the reconstructions have been made by using the same initial mesh, the amount of vertices remains constant so that they can be directly linked.

For a given vertex, its successive positions can be calculated by linear interpolation as stated in equation 5.8 of section 5.5.6. Figure 5.19 shows the process graphically. We already know their initial (start) and final (stop) positions and we know several positions between as well (all those are colored in red). The interpolation creates the transitions (colored in blue).

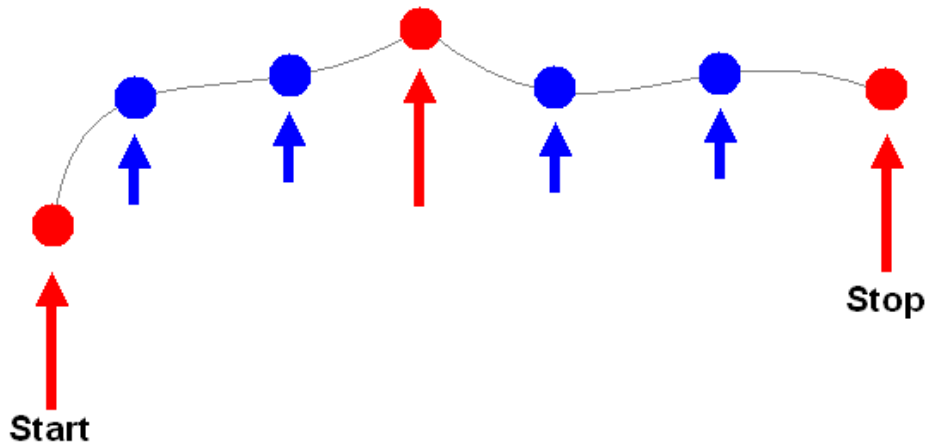


Figure 5.19: Keyframing in a vertex-by-vertex manner.

Typical values for our application range from 20 to 30 key frames between successive instants of time.

It must be said that the tracking of the motion in the left ventricle is not within the goals of this project. Other publications like [79] present accurate methodologies that permit the tracking of its characteristic twisting behavior. Our implementation provides motion but does not show the changes due to the twist movement.

## 5.8 Summary

The geometrical model conditions the operation of the whole process. The use of one or another primitive, the neighborhood relations between them and the importance of a design well fit to the evolution scheme are very important decisions to take.

This chapter begins by classifying the existing geometrical models in a general way. From all the classes available, we decide to use a discreet model based on a triangulation mesh where the vertices can be directly mapped as the particles of our evolution scheme. Moreover that, triangulated meshes have been extensively used and are a suitable model for the graphics hardware that we use nowadays. Those meshes ensure the topological constraints that we must validate.

Once the decision has been taken, we put our triangulated meshes into the test. We determine the degeneration of the triangles after several simulations in order to select the best initial mesh, in terms of quantity and size of the triangles. We also conclude that the reconstruction methods presented in chapter X do not affect the quality of the final triangles too much.

When dealing with the reconstruction of the internal left ventricle cavity, we analyze the best path to take. There are two options attending to the radial direction of the simulation: In-to-Out or Out-to-In evolutions. By testing the system, we decide to use the Out-to-In approach in order to ensure correct results that avoid the apparition of artifacts and peaks in the final mesh.

Our model is compared with the well known Marching Cubes algorithm [55] in order to point out their differences and deduce the reasons that make the second an unsuitable tool for our particular needs.

As an important apportation, we present a self-made smoothing algorithm. This algorithm detects and smoothes out the peaks that can appear in the mesh during and after a simulation. It is a general purpose tool that can be used in several fields like Telemedicine, Terrain generation or Space and Air force simulation.

For the Telemedicine application, which is the main goal of this study, we present the numerical scheme that has to be followed in order to compute the volume specified by a surface mesh. Several measurements needed by the physicians make extensive use of this.

As a final apportation, we present the integration of time into our simulation system. Geometrically based, our interpolation method is based in the use of an old technique called Keyframing [51]. It allows us to present fluent animations of objects in movement that have been reconstructed by our algorithms.

Chapter 6 presents the results of our research. Those are related to the reconstruction of the left ventricle and also to other generic cases, demonstrating then that our implementations and algorithms can be broadly applied.

## **6 Applications and results**

This section presents several results related to our research in 3D reconstruction as a modeling tool. We are focused with the reconstruction of the human's left ventricle, as stated in the majority of this document, but we also demonstrate that our technique can be applied in other fields that require generic reconstructions. In section 6.2 we present several reconstruction examples where our method can be used as a tessellation algorithm and also for obtaining different resolution meshes for some objects.

This chapter is structured as follows. We begin by showing our first steps in reconstructing 3D meshes by applying external forces to them. Firstly we apply a generic external force. Then it is changed into the GVF vector field. From these experiments, we retrieve several reconstructions of spherical datasets followed by our first left ventricle.

After that, we evolve to the final solution. We state the need of testing a known volume in order to find valid values for all our parameters. We use the Phantom volume because of its absorption and transmission characteristics similar to the ones related to the left ventricle. We test binary and gray level datasets obtained from this volume to get specific information about the GVF parameters, the stopping mechanism, the final distance to data, the recovered volume and the computational cost. We also apply our algorithms to missing-data datasets. The results are useful for analyzing the performance of the algorithm in pathological cases where ischemic areas, due to the absence of irrigation, induce holes in the data.

After the Phantom experiments, a complete cardiac cycle is reconstructed. The dataset comes from an actual's patient acquisition. Several comparisons are shown regarding the methodologies previously described. Besides that, pathological cases are also treated by our algorithms with very successful results.

We also depict other explored approaches such as automatic contouring and tessellation of the 3D shape by using the discreet contour deformation model. This methodology is applied to SPECT imagery as well.

A flexible volumetric model for the left ventricle is also summarized for the seek of completeness. This model is not within the scope of this project but it is being defined by other members of the research group as the next stage of the whole process. Real time interaction with the organ requires this type of modeling.

Our images are also tested within the Anisotropic Contour Completion context. This is a very novel technique that has been published recently. The technique takes into consideration the local orientation of the contours which is an absolutely well fit assumption for SPECT images of the left ventricle.

Finally, we perform generic reconstructions in the context of low-polygon modeling, tessellation and mipmeshing. Those are well-known techniques in the context of computer graphics that optimize the rendering of a synthetic scene. We show that our algorithm can be applied to other contexts apart from the left ventricle 3D reconstruction.

## 6.1 Human's Left Ventricle reconstruction

Access to a 3D model obtained from patient's data can have several applications like support on diagnosis, surgery planning, student's training or even remote-operation. A first approximation to the problem would be using a manual process with specific image-processing software though it would require deep medical knowledge and experience.

Our method automates the manual processes related to the evaluation of SPECT images (see appendix A) by the physicians. From the images we derive a 3D model that can be used as a key tool in order to get a robust diagnostic. We will show the evolution of our approach trying to justify the final method adopted in each step.

### 6.1.1 First steps

This project of research began by using the DRAPS cloth simulator [5] as a tool that would allow applying forces to a certain particle system. As depicted in figure 6.1, a piece of cloth built from several particles is hanging from a cylinder. All the particles were suitable of being affected by an external force. This external force was selected to be of the form  $F = (-x, -y, -z)$ .

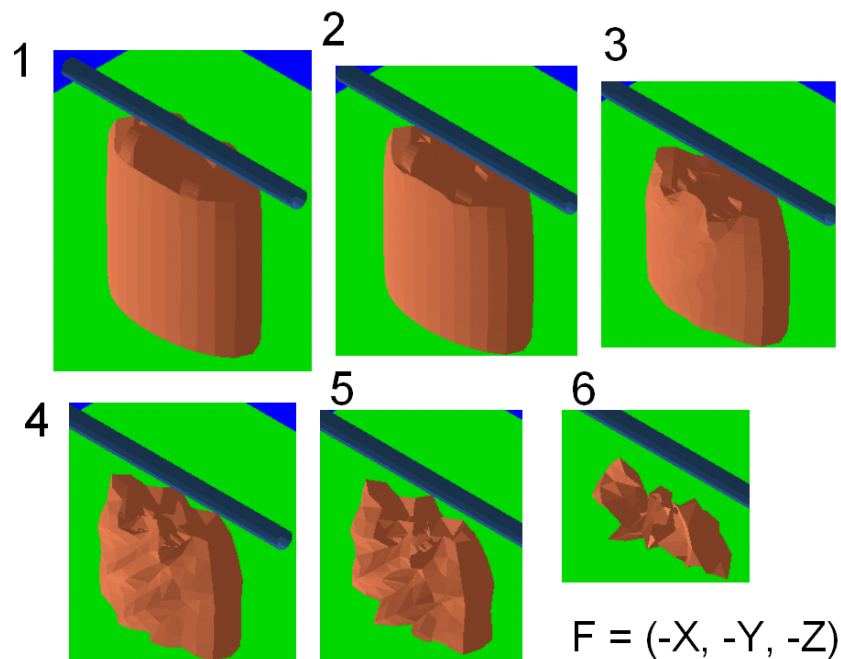


Figure 6.1: First trials in the reconstruction process.

Figure 6.1 shows the evolution of the simulation. All the particles building the piece of cloth are affected by an external force that attracts them to the coordinate system center. As long as there is no internal restriction, the initial shape vanishes and the particles get accumulated in a unique position.

That test was not really related to the aim of the project in the sense that no Left Ventricle data was part of it. Nevertheless it revealed that the DRAPS cloth simulator might be an interesting tool to use, at least as our initial framework platform.



As explained in chapter 3, the external force that we use for the 3D reconstruction method is based on a vector field. First tests involved also putting into operation the field and checking its diffusion variations attending to the  $\mu$  parameter. Our first scenario consisted on two concentric spheres acting as the internal and external surfaces to be recovered. In figure 6.2, the intensity values along the set are presented in a binary format where 0 stands for no property at all and 1 for property values over a concrete threshold value.

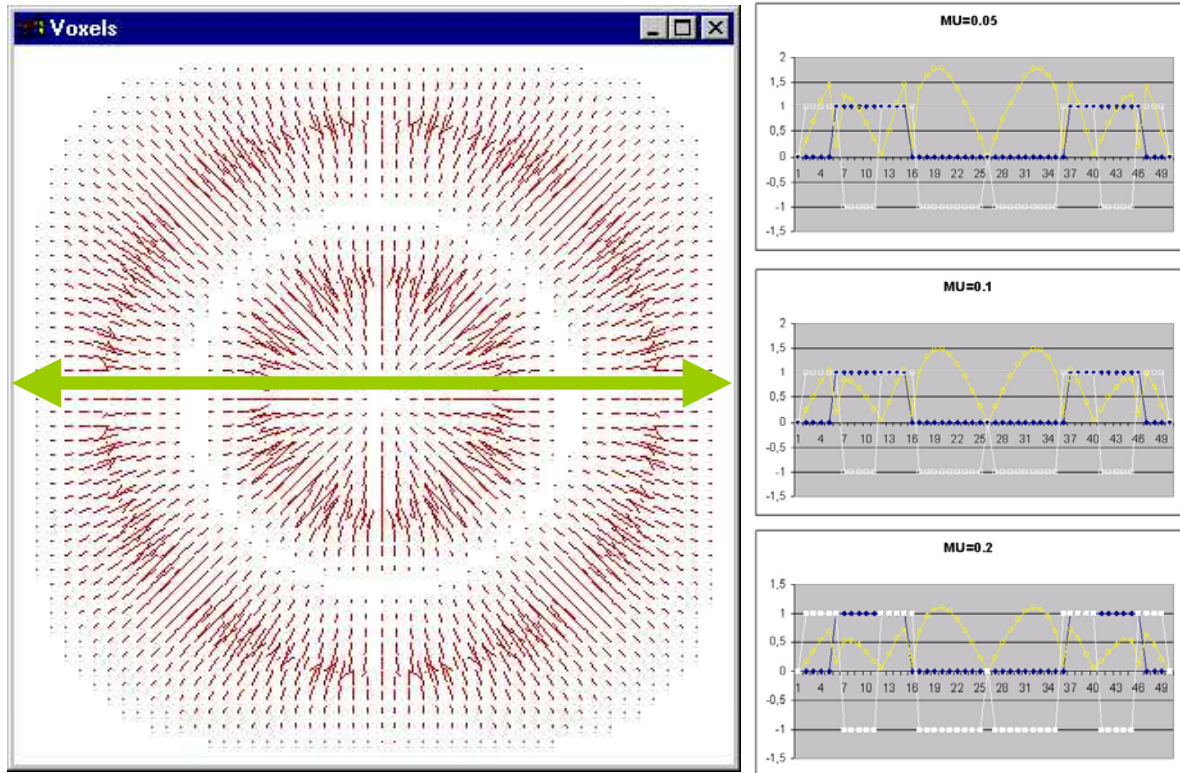


Figure 6.2: Initial tests regarding the vector field.

Figure 6.2 (left) shows the vector field for a data slice. The vectors clearly identify both the surfaces to recover, internal and external. Both surfaces are characterized by a null potential.

Figure 6.2 (right) shows three different graphs which correspond to a horizontal itinerary crossing the central area of the slice (green double arrow). The magnitudes of the vectors are presented in yellow; their associated sign is presented in white; and the intensity values are rendered in deep blue. The itinerary consists on 50 voxels with two high intensity regions (10 voxels wide). The vector field becomes null in the borders of those regions. Moreover that, it also changes in terms of sign. The change has to exist because even if we come closer from a side or from another, we must end stopping at a null gradient voxel.

Moreover that, figure 6.2 shows that for higher values of  $\mu$ , the vector field gets smoothed. One can perceive that in those cases the energy gets spread out within the voxels, never concentrated at some maximal areas. Then the global amplitude decreases.

Using the vector field of figure 6.2 and the plain deformation model (see section 3.4.2), we proceeded to the recovery of both the spheres. The initial meshes were cubes. Figure 6.3 shows the final results after the reconstruction process.

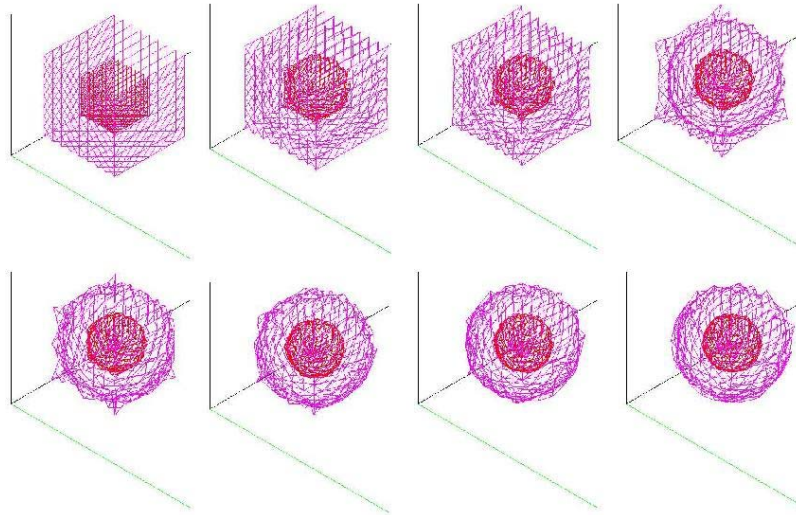


Figure 6.3: First reconstructions with the plain deformation model.

Both spherical surfaces are recovered although it can be seen that there is a clear lack of smoothness.

Using the plain deformation model and a real dataset (from an actual patient's left ventricle), we performed the first non-synthetic simulation. Figure 6.4 shows the vector fields associated to two slices of this first test on a real dataset.

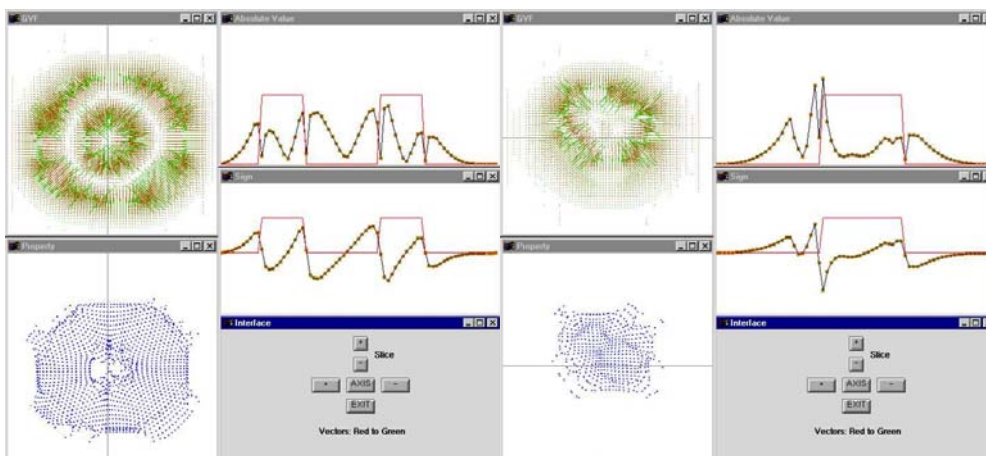


Figure 6.4: Vector field associated to real data (two slices).

In figure 6.4, left and right slices, the intensity values are binary (bottom-left window with the blue samples). The vector field is also shown (top-left window with the red to green vectors) besides the itinerary along a tiny line, vertical in the left and horizontal in the right slice. The magnitudes (top-right) and signs (middle-right) are associated to the

itineraries. Figure 6.4 (left) shows a slice with two null gradient regions (two red peaks) whilst figure 6.4 (right) shows a unique peak. This is due to the fact that on the first case, the slice is located on the top of the apex region (where there are two surfaces, the epicardium and the endocardium) but on the second case, the slice is located in the apex (only epicardium). See appendix B for more details on the left ventricle anatomy.

We can see the maximum magnitudes around the internal and external vicinities. On the other hand, note the change of sign on the field vectors. It is positive when the vectors go from left to right (or from top to bottom in the left slice of figure 6.4) and negative in the rest of the cases. The magnitude of the vectors was scaled by a factor of 100 for a better observation.

The vector field is not entirely uniform (see the magnitude differences on the highest peaks of the right slice in figure 6.4), this suggest us that the energy operator might be changed in order to get better results. In fact, we should use the operator of equation 6.1 when working with several gray levels and the operator of equation 6.2 if using binary data.

$$E_{ext}(x, y, z) = -|\nabla I(x, y, z)|^2 \tag{6.1}$$

$$E_{ext}(x, y, z) = I(x, y, z) \tag{6.2}$$

For this first example on a real dataset, the *SPECT* (see appendix A) images were embedded in a world of voxels characterized by a resolution of 68x75x58 voxels and spatial steps of 1.435 mm (X), 1.435 mm (Y) and 2.871 mm (Z) for each voxel. It is important to note that those were the dimensions of the entire world of voxels. The region of interest (ROI) was definitely smaller. The temporal step was selected to satisfy the Courant-Friedrichs-Lewy condition as shown in equations 4.50 and 4.51. In fact it was also divided by a factor of 2. The  $\mu$  parameter was taken as 0.4.

Figure 6.5 shows the final reconstruction of the external surface. The reconstruction used the vector field of figure 6.4. The initial mesh was built from 1396 particles (2788 triangles). Figure 6.5 shows how the final mesh is completely stuck to the external border of the voxels. The temporal average per simulation frame was 2.22 seconds (K7 Athlon 600 MHz, 128 Mbytes RAM). The stepsize for the reconstruction of the particle system was selected to be of 0.01 seconds.

For the simulations regarding the internal surface of this dataset, see figure 5.5.

Those simulations allowed us to derive some conclusions:

- It is difficult to select the amount of particles for the initial mesh. Too many will slow down the process and not enough might produce low resolution reconstructions with a poor quality.
- It is not obvious to choose a stepsize value. It must be as higher as possible while avoiding divergences.
- The weighting constants associated to the internal forces are difficult to adjust. It is important to find the required tradeoff between flexibility and stiffness.

- There are oscillations at the borders produced by the discrete nature of the dataset. This behavior must be smoothed or eventually eliminated.
- Some kind of strategy must be applied in order to minimize the duration of the simulation (when using the plain deformation model). A unique iteration is not computationally expensive but the application needs several for reaching convergence.

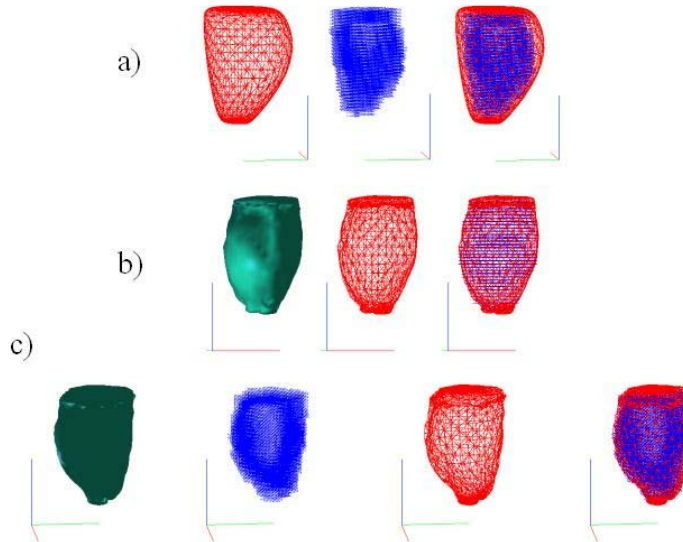


Figure 6.5: Reconstruction of the external surface in a real dataset; a) Initial mesh and dataset to recover; b) and c) Final mesh after the reconstruction.

Those topics were addressed by using a synthetic test model in order to perform the needed tests and measures: the Phantom model. Section 6.1.2 gives details on these.

## 6.1.2 Evolution to the final solution

It is basic to test a known volume in order to infer the correct values for all our parameters. We also need a dataset which should be characterized by similar parameters to those related to the left ventricle tissue. In order to demonstrate the reliability of our assumptions, we present specific results about the GVF parameters, the stopping mechanism, the final distance to data, the recovered volume and the computational cost.

### 6.1.2.1. Test model: the Phantom Volume

In order to measure the reliability of the system from an analytical point of view, it is compulsory to test the algorithms on a well-known dataset. We decided to use a medical test volume which is frequently related to the imaging hardware used by physicians: the Phantom volume.

The Phantom volume presents absorption and transmission characteristics similar to the ones related to a certain tissue. It can then be used to simulate human tissue like the left ventricle muscle. Details on its characteristics and measures can be seen in appendix D.

Figure 6.6 (top) shows a long-axis cut of the binary dataset associated to the Phantom volume. From the images of the Phantom we can build up a world of voxels that feeds

into our reconstruction module. Figure 6.6 (top-left) shows the vector field built from the intensity data in figure 6.6 (top-right).

Note that:

- For those first experiments, the smoothing algorithm wasn't still planned. For that reason, an artificial cover had to be added to the dataset in order to stop the internal and external meshes (figure 6.6 (bottom)).
- The initial dataset is voxelized which enlarges the view of its discreet nature. This is an undesirable characteristic if we are willing to retrieve a final smooth surface.

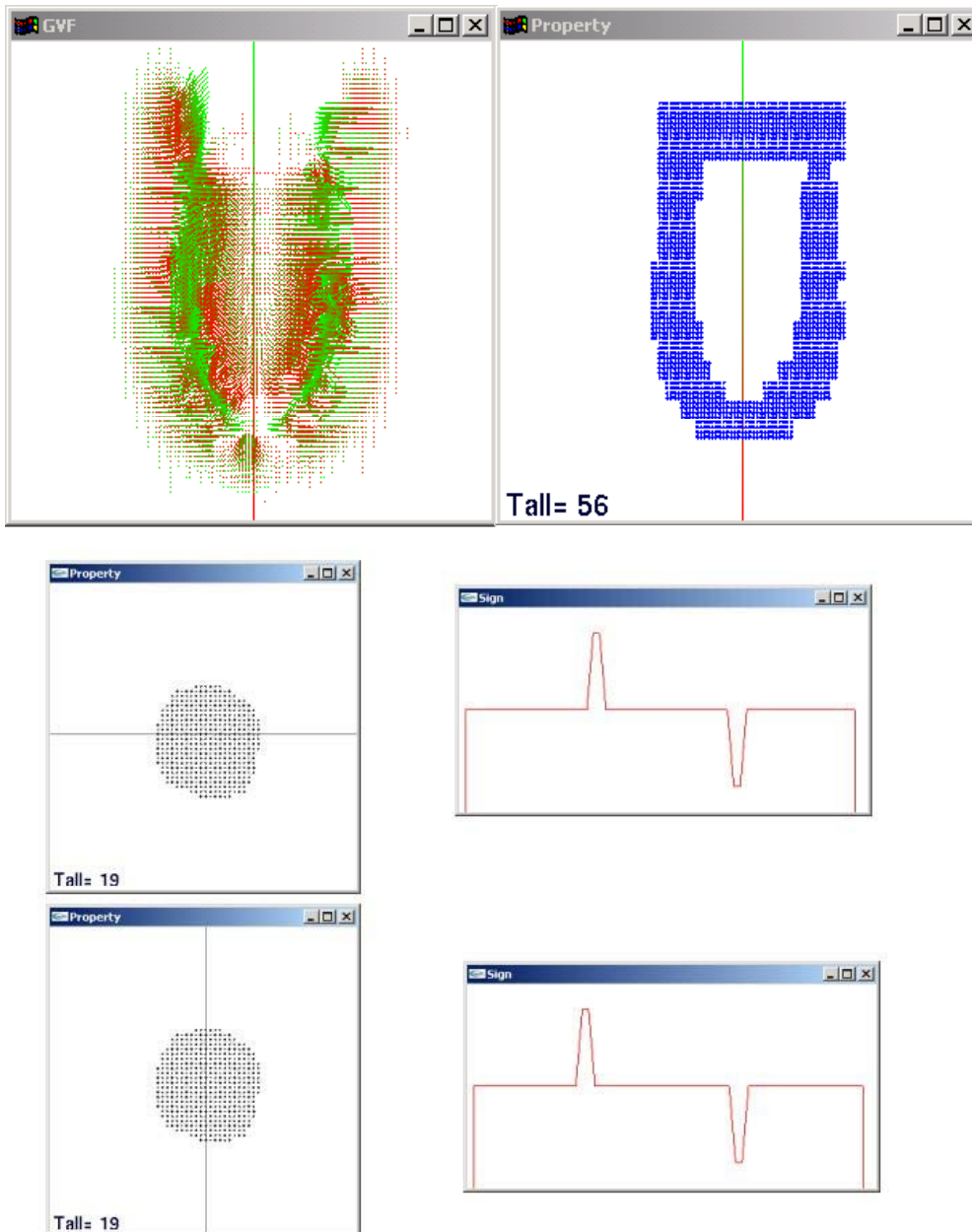


Figure 6.6: Top: Long-axis view of the Phantom dataset; bottom: the artificial cover.



At this point, two strategies were defined in order to minimize the discreet nature of the dataset:

- In reconstruction time, check every particle's position, determine the voxel where it lies and affect its force accumulator by a vector field obtained from the average of the vector fields associated to its neighbors (neighbors in its direction of movement).
- Subdivide the initial dataset generating more voxels. Then evaluate the vector field in this "new and enlarged" dataset. Section 2.2.1 finds out the reasons that made us avoiding this second strategy.

Figure 6.7 shows the final result after the reconstruction of the external surface of the Phantom volume. We used the plain deformation model with a temporal stepsize ( $\Delta t$ ) of 0.05 seconds, a GVF weighting constant of 25 and a damping factor of the 25%. The simulation took 2.8474 seconds per frame (averaged).

In figure 6.7, the final mesh is superimposed to the initial dataset. Note that in this simulation there wasn't stopping mechanism and no measures of reliability where taken (measures on distance to data, triangle quality or final volume).

All the reconstructions presented so far were based on binary datasets. It must be said that we also performed several tests over datasets based on gray-level values. In fact those datasets are more realistic with the real data that physicians provide because blood irrigation is not imaged as a unique tone.

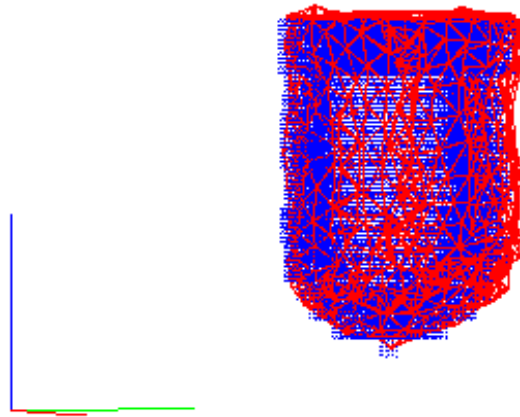


Figure 6.7: Reconstruction of the Phantom external surface.

Figure 6.8 presents a vector field obtained from a dataset based on gray level intensities.

Note the non-binary nature of the intensities by looking at the red thin lines of figure 6.8 (top-right windows). Instead of sudden changes of property from the zero to the one level, we have intensity slopes that make the transitions continuous. The only thing to take into consideration is the energy operator applied, as depicted before in equations 6.1 and 6.2.

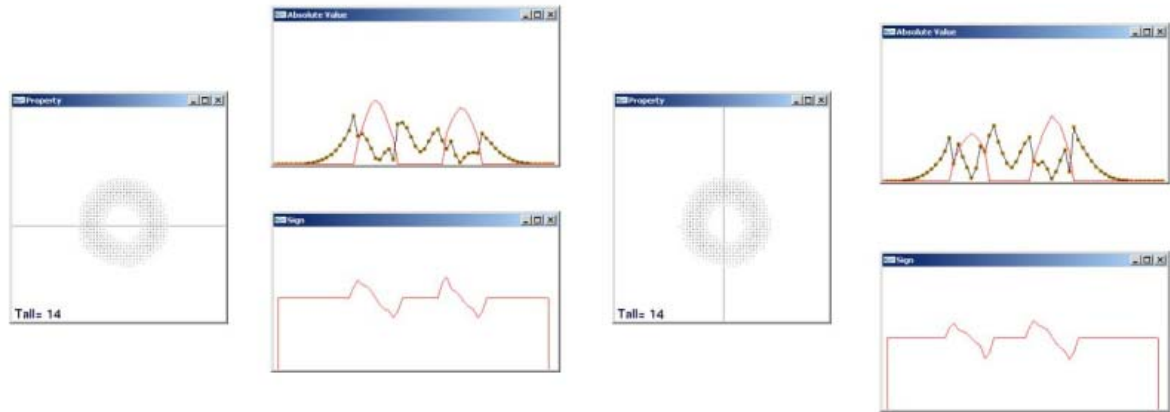


Figure 6.8: Vector field for a gray level dataset.

### 6.1.2.2. GVF parameters for the Phantom volume

Tables 6.1 and 6.2 present several vector fields evaluated for the Phantom volume. The objective of this test is to examine the variability of the solution depending on the vector field parameters.

GVF Calculation $\mu = 0.15$ $F = \text{Laplacian of Intensity}$ $16\mu$ as the denominator of the Courant-Friedrichs-Lewy condition			
	Regular data	Doubled data	
<50 iterations			<50 iterations
>50 iterations			>50 iterations

Table 6.1: GVF vector field study for the Phantom volume (I).

Let us examine table 6.1. In there, all the parameters are equally defined except for the amount of iterations and the data preprocess. The left column shows fields derived from the original data, with no preprocess at all. The right column shows the same experiments over preprocessed data (this preprocess consisted on doubling the resolution of the world of voxels like explained in section 6.1.2. After the doubling, the vector field was found).

In the first experiment (first row), the GVF algorithm was run under 50 iterations. For the second experiment, the algorithm was left executing for more that 100 iterations.

Table 6.2 shows similar experiments where different parameters have been altered, in that case two variables, the  $\mu$  parameter and the denominator of the Courant-Friedrichs-Lewy condition (see section 4.5).

In all the cases the images correspond to a horizontal itinerary along slice number 10. The itinerary traverses the slice horizontally for  $y = 29$  (or  $y = 58$  for the doubled data).

Let us examine the tables in order to retrieve several conclusions:

- It seems clear that there's no need to iterate for more than 50-100 cycles. The results are pretty defined from the 50 iteration.
- Doubling the initial data doesn't help too much as expected. In fact the results are basically equal or worse, in terms of magnitudes.
- The election of the  $\mu$  parameter is less significant than it might seem. In fact it is a matter of convergence or divergence as we can see from the tests. If the test converges, the final solution will vary between some tiny bounds.
- The denominator must be as smaller as possible in order to guarantee a major stepsize. However we must be careful in order to avoid divergences.

Can we use the same parameters for all our datasets? We would be glad if possible but the answer is not obvious. As a third test, we used the same parameters again a second Phantom volume, the Mayo Phantom volume.

The Mayo Phantom volume was considerably smaller in terms of ROI. Nevertheless its spatial dimensions (XYZ) were bigger than for the previous (5.388 mm 5.388 mm and 5.388 mm against 2.87 mm 2.87 mm and 5.74 mm).

We placed the new dataset into two different simulations. Table 6.3 depicts the results.

As expected, higher spatial dimensions induce higher stepsizes that must be corrected by a suitable denominator. If its value is not big enough, the system can diverge. Figure 6.9 shows the vector field derived for the convergence case. Note that the ROI is considerably smaller than in the cases analyzed in tables 6.1 and 6.2.



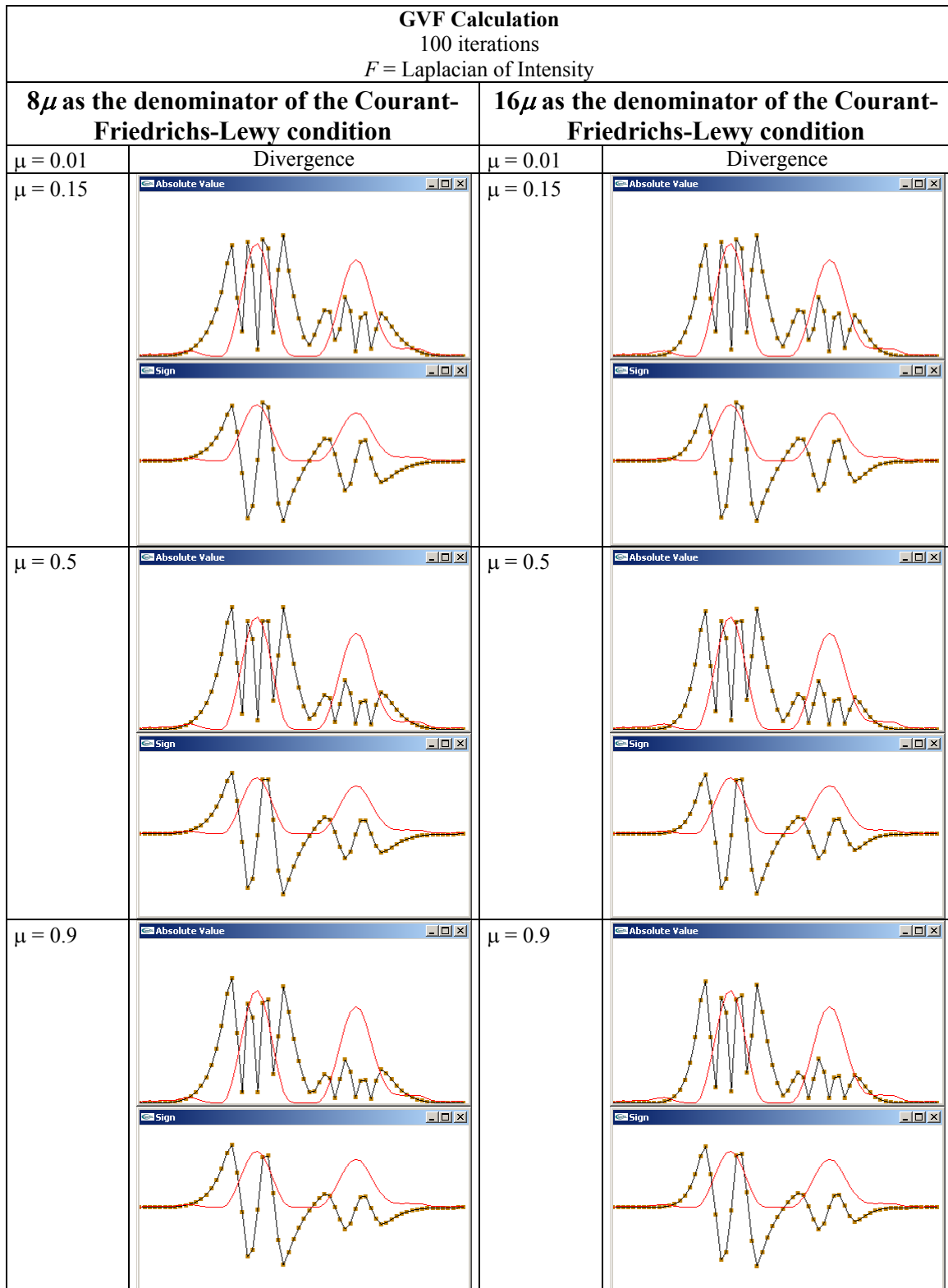


Table 6.2: GVF vector field study for the Phantom volume (II).

$\mu$ parameter	Iterations	Denominator	Result
0.15	50	$8\mu$	Divergence
0.15	50	$16\mu$	Convergence

Table 6.3: The GVF vector field parameters study for the Mayo Phantom volume.

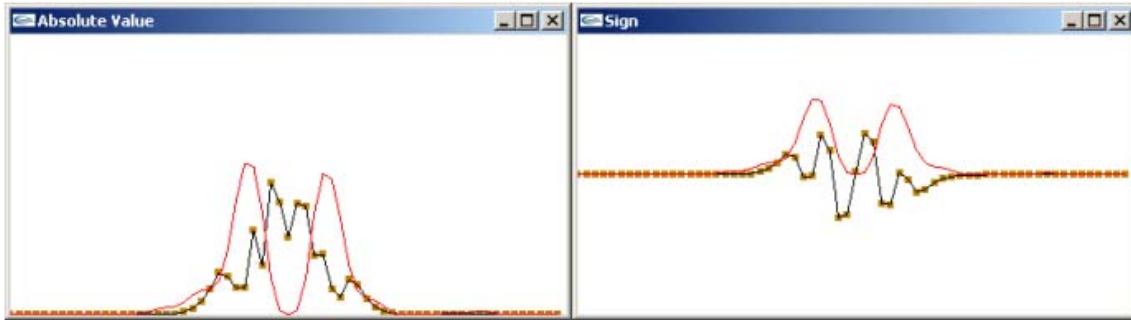


Figure 6.9: Vector field itinerary for the Mayo Phantom volume.

We may present a final conclusion for this section: most of the parameters can be tuned via typical values although big distortions in the dimensions of the dataset might require some kind of adjustment.

### 6.1.2.3. The stopping mechanism

If we take a look at the behavior of our particles around the data borders, we will detect undesirable oscillations. The particles do not stop immediately if no one forces them to. Residual components of the external force might induce movement to a basically quiet particle system. Figure 6.10 depicts this behavior graphically.

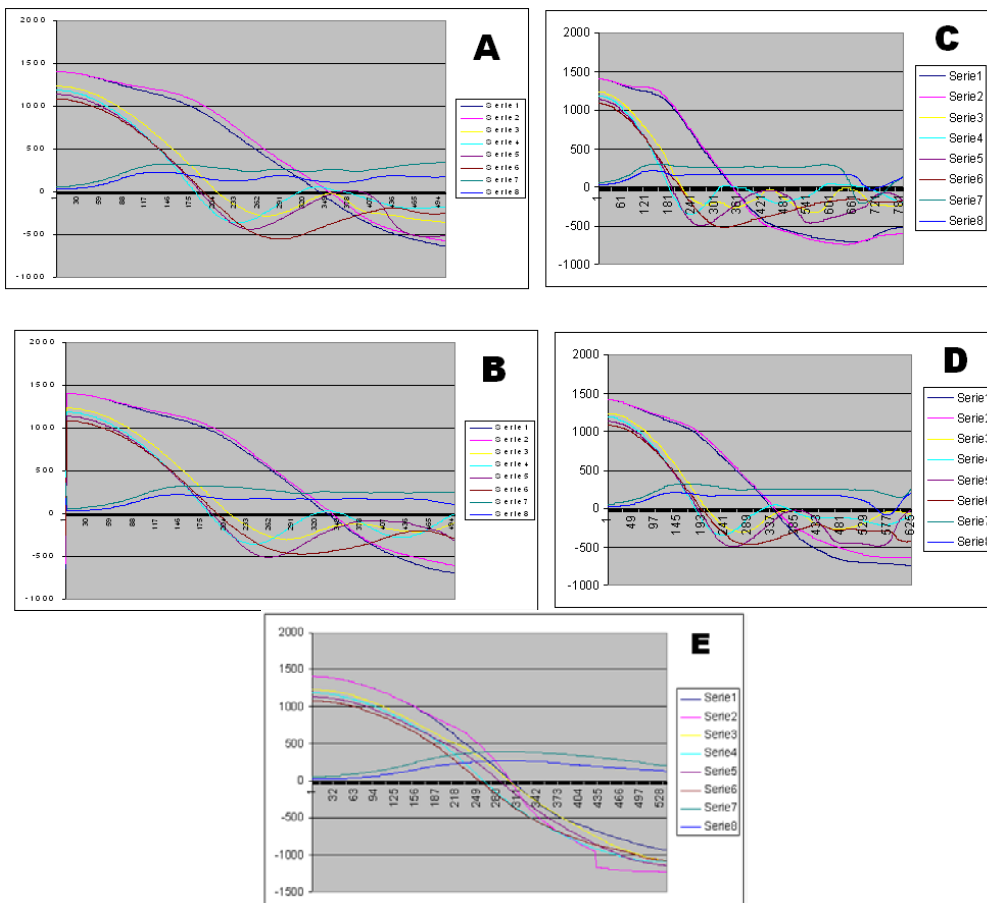


Figure 6.10: Oscillations around the data borders.

The parameters of each test can be seen in table 6.4. All the simulations were performed by using the plain deformation model (see section 3.4.2) with a stepsize of 0.05 seconds. All the reconstructions used the second mesh of table 5.2 (1280 triangles). As explained before, figure 6.10 shows the oscillation effect that characterizes a particle's motion when being near the vector field boundaries. In this case, we measured the distances of the particles to the well-known volume of the Phantom dataset. The black line stands for the zero distance position. It can be seen that several particles get close to the boundaries but never stop due to their oscillation motion. It comes clear that we need some kind of stopping mechanism. We found no big differences when varying the simulation parameters. In fact, even avoiding the use of a multiplying factor (this factor increases the speed of the simulation by scaling the vector field vectors), like in test E, does not ensure that oscillations will not occur.

<b>Test</b>	<b>Stretch</b>	<b>Stretch Damping</b>	<b>Shear</b>	<b>Shear Damping</b>	<b>Bend</b>	<b>Bend Damping</b>	<b>Multiplying Factor</b>
<b>A</b>	10	1	0.5	0.05	5	0.5	50
<b>B</b>	10	1	0.5	0.05	10	1	50
<b>C</b>	20	2	0.5	0.05	10	1	50
<b>D</b>	10	1	2	0.2	10	1	50
<b>E</b>	10	1	0.5	0.05	5	0.5	1

Table 6.4: Simulation parameters associated to the results shown in figure 6.10.

As a first approximation to the stopping mechanism, we associated a damping value to each voxel in the dataset. This value was directly related to the magnitude of the vector field in the voxel. We have an example in figure 3.23 (see chapter 3).

The more magnitude we have, the more damping we add. This control “breaks” the particles when closer to the data borders that we are expecting to reconstruct. It does not imply a real computational effort to the algorithm because it is evaluated as preprocess from the intensity data. This method smoothes the oscillation effect but it does not reject it completely.

As a definitive stopping mechanism, we proposed to use a technique that would mark the voxels belonging to the data borders. This is not an obvious task because we do not have this information from beforehand. We need to process that dataset in order to get the answers that we are looking for. See section 2.3 for a deep explanation on the tested methodologies

Once the voxels have been marked as belonging to one of the three categories available (external border, internal border or none), we can apply several stopping mechanisms:

- If using the free deformation model, we can stop the particles when entering a marked voxel. In this case, the particles are unrelated to each other which mean that we are neither adding nor extracting energy to the whole system. Nevertheless this process is not accurate at all since the particle will not get totally relaxed in the position that it should but in the boundary of the voxel that it just entered. Moreover that, it is easy to contribute to the apparition of peaks in the final mesh. We are somehow introducing a cuberille effect to the mesh.
- In the case of models with topology like the plain deformation model or the spring-mass deformation model, we can ease the oscillation effect by hardening

the damping factor in the vicinities of the border voxels. That would be an approximation similar to the one illustrated in figure 3.23. If we detect that a high percentage of particles is subjected to a minimal increment of motion, we can stop the system literally.

- It would be definitely desirable to add a smoothing post-process that once the particles are stop, can control the smoothness of the generated triangles. This algorithm would delete the characteristic peaks that might appear, as explained before. It is an hybrid process, built from a dynamic module followed by an algebraic one (see section 5.5).

#### 6.1.2.4. Distance-to-data measures

In order to test the reliability of the final reconstruction, several tests were performed on the Phantom dataset, regarding the final distance between the particles and the border voxels. The tests were performed by using the free and spring-mass deformation models. Tables 6.5 and 6.6 show the parameters and results related to these experiments.

$k_s$	$k_{sd}$	$k_{gvfd}$	Pond.	% < 1 voxel
10	1	-0.1	10	82.28
50	1	-0.1	10	81.95
100	1	-0.1	10	81.84
10	1	-0.25	10	66.55
10	1	-0.5	10	61.27
10	1	-0.1	100	88.56
0.1	0.01	-0.1	1	59.51
1000	0	-0.25	100	88.77

Table 6.5: Simulation parameters associated to the measures in distance to data. For those simulations  $\Delta t = 0.00025$  s., iterations = 2000. (Spring-mass deformation model)

In tables 6.5 and 6.6,  $k_s$  stands for the stretch constant;  $k_{sd}$  for the associated stretch damping factor;  $k_{gvfd}$  for the external force's (vector field) damping factor; Pond. is the weighting factor that multiplies the vector field and  $\Delta t$  is the selected stepsize. The last column specifies the percentage of particles that ended-up being less than one voxel far from the data borders. It must be said that the maximum accuracy achievable is given by the dataset precision, which is of one voxel.

We see that for the spring-mass deformation model (table 6.5), we have more than 80% of the particles close to the dataset in the 62.5 % of the simulations. In fact the worst results appear if the weighting factor is higher or if the stretching constants are negligible. In those cases there is a clear lack of cohesion in the model plus an acceleration that might corrupt the final mesh.

Nevertheless the model that we are really interested in is depicted in table 6.6, the free deformation model. In that case, 80% of the reconstructions ended up with more than 75% of the particles close to the data boundaries. In fact, 70% of the reconstructions had a 90% of particles attached closely. It is clear that the worst results are characterized by:

- Small number of iterations, surely not enough for the system to converge if the stepsize is not big enough (rows 4, 5 and 6).

- Too high weighting factors for the vector field (row 13).

$k_{gfd}$	Pond.	$\Delta t$ (sec.)	iterations	% < 1 voxel
-0.1	10	0.00025	2000	87.12
-0.1	10	0.1	400	92.36
-0.1	10	0.01	3400	91.93
-0.1	10	0.01	1000	66.36
-0.1	10	0.01	200	38.99
-0.1	10	0.01	400	35.29
-0.1	25	0.0005	7700	100
-0.1	10	0.005	7200	90.99
-0.1	25	0.005	3700	90.35
-0.1	10	0.01	3000	93.13
-0.1	25	0.0005	15000	90.05
-0.25	25	0.001	1800	75.19
-0.25	100	0.00025	2000	36.41
-0.1	10	0.005	1900	98.69
-0.1	10	0.01	900	97.93
-0.1	25	0.005	700	98.02
-0.25	25	0.01	1000	98.47
-0.25	25	0.001	1800	77.57
-0.1	10	0.01	3600	91.18
-0.1	10	0.1	400	92.36

Table 6.6: Simulation parameters associated to the measures in distance to data.  
(Free deformation model)

The free deformation model is more independent from the stepsize than the spring-mass deformation model. It offers better reconstruction results in terms of final distances with bigger stepsizes and less iterations (see the last row of table 6.6). On the other hand it lacks cohesion but we can address it with the smoothing algorithm (see section 5.5).

Mesh rendering and additional comments on some of the tests in tables 6.5 and 6.6 are given in table 6.7. We may get several conclusions by analyzing them:

- Tests 1, 2, 3 and 4. Performed with the Phantom Mayo dataset. The initial mesh is very detailed and close to data. In fact it was obtained from an execution of the Marching Cubes algorithm (see section 5.4). The dataset is binary and we doubled it in order to get a much bigger workspace. There was no neighbor interpolation for the vector field. Note that more than 70% of the particles end up being less than one voxel far from the dataset. Note also that more complex deformation models require smaller stepsizes and much more CPU time. Besides that, the degeneration in areas is especially present in the constrained stretch deformation model.
- Tests 1 and 4. Note the final percentages (71.4% vs. 81.1%) due to the differences in the initial mesh. In fact test 4 is also much better in terms of degeneration. The reasoning seems to reverse when talking about CPU times (17.4 vs. 99.65) although the stepsize of test 1 doubles the stepsize of test 5.
- Tests 5 to 9. Performed with the Phantom Hebrón dataset. Note that there are also differences regarding the data. It is gray level based and normalized (between 0 and

1). For all these tests, the simulator applied the free deformation model. The initial mesh is the translated and scaled sphere, suitable for avoiding degenerations. In fact those defects are only significant if the resolution of the mesh is quite large only (test 8). The distance percentages are always above the 90% although the degenerations are totally avoided if the resolution of the mesh is of the same order than the distance between slices (test 9).

- Tests 8 and 9. Very different magnitudes on the CPU times. These differences are directly related to the amount of particles that the system has to update at each new iteration (2562 vs. 162).

Results on tables 6.5, 6.6 and 6.7 where obtained from a Pentium II PC 266 MHz equipped with 256 MB RAM and a nVidia RIVA TNT II graphics card (32 MB).

More results regarding the distances have been presented through the text. See table 4.1 and figures 4.11 and 4.12.

The plain deformation model is implemented, but we are no longer using it as an experimental tool. It is a very slow model because the stepsizes must be tiny, according to the internal forces that apply (see chapter 3). Moreover that, it is very difficult to tune the parameters taking into consideration the tradeoff between internal rigidity (never too high because we are interested in motion) and external force (that can be weighted as we have already seen). Figure 6.11 gives some results regarding the distance-to-data scoring for two reconstructions.

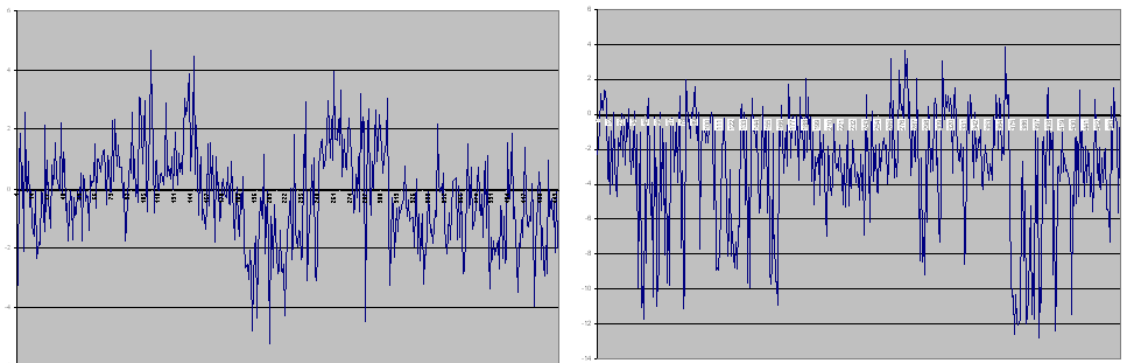


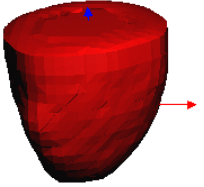
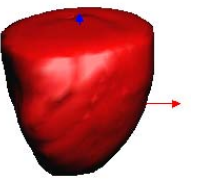
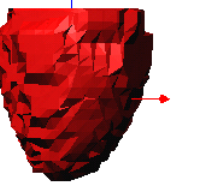
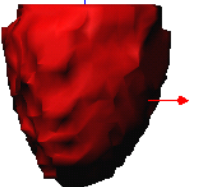
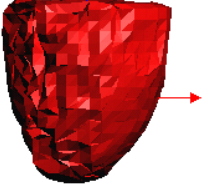
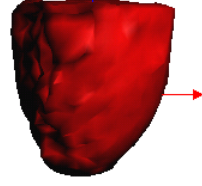
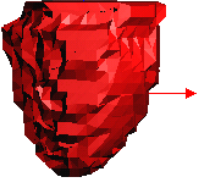
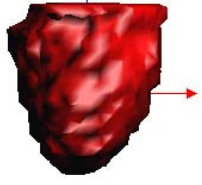
Figure 6.11: Distance results for the plain deformation model.

The parameters for the simulations in figure 6.11, where tuned like:

- For the left simulation: Stretch = 1.0, Stretch Damping = 0.1; Shear = 0.1, Shear Damping = 0.01 and Bend = 1.0, Bend Damping = 0.1.
- For the right simulation: Stretch = 100.0, Stretch Damping = 10.0; Shear = 1.0, Shear Damping = 0.1 and Bend = 50.0, Bend Damping = 5.0.

In both cases, the stepsize was 0.05 seconds. As shown, the parameters are suitable for the simulations to converge. Besides that, the parameters are quite different in both simulations but as figure 6.11 shows, the qualitative results are far from being optimal. The amplitudes (blue) show the distances between a particle and the zero-distance line (black). The left simulation is better but it is still far away from the results that the other models retrieve.

General Dynamic Surface Reconstruction

Test code	Mesh (Flat shaded)	Mesh (Gouraud shaded)	Geometry	Model	KS / KD	KGVF / KGVFD	$\Delta T$ (sec.)	Areas			CPU Time	% < 1 voxel	Comments
								Average	Max.	Min.			
--			Initial mesh: - 1236 Particles. - 2465 Triangles. - Initially, 52.8 % < 1 voxel Average areas is 10.504499 and normalized 1.459838 Max area is 20.545576 and normalized 2.696347 Min area is 0.251531 and normalized 0.329984										
1				Free	0.0 / 0.0	10.0 / -0.1	0.01	9.44 / 1.34	28.6 / 3	0.043 / 0.024	17.4	71.4 %	- Binary - Doubled - Phantom Mayo - No neighbor interpolation
2				Spring-mass	1000.0 / 0.0	100.0 / - 0.25	0.00025	10.48 / 1.46	20.38 / 2.67	0.23 / 0.3	267.05	71.5%	- Binary - Doubled - Phantom Mayo - No neighbor interpolation
3				Restricted Spring-mass	1.0 / 0.0	25.0 / -0.1	0.0005	10.01 / 1.4	23.08 / 2.7	0.0022 / 0.0021	448.7	72.16%	- Binary - Doubled - Phantom Mayo - No neighbor interpolation

4			Free	0.0 / 0.0	10.0 / -0.1	0.005	17.9 / 2.26	45.7 / 3.79	4.51 / 1.10	99.65	81.1 %	- Binary - Doubled - Phantom Mayo - No neighbor interpolation
--		Initial mesh: - 642 Particles. - 1280 Triangles. Average areas is 33.816044 and normalized 3.497494 Max area is 47.854534 and normalized 3.965818 Min area is 28.186029 and normalized 3.121821										
5			Free	0.0 / 0.0	25.0 / -0.1	0.0005	22.9 / 2.6	73.7 / 4.4	0.7 / 0.07	861.3	90.05 %	- Gray levels - Normalized - Phantom Hebrón - No neighbor interpolation
6			Free	0.0 / 0.0	10.0 / -0.1	0.005	22.49 / 2.57	116.34 / 5.23	0.79 / 0.088	485.4	90.9 %	- Gray levels - Normalized - Phantom Hebrón - No neighbor interpolation





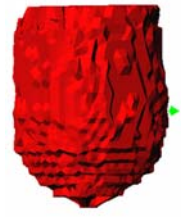


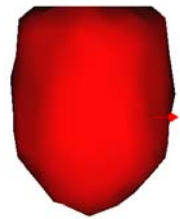
7				Free	0.0 / 0.0	10.0 / -0.1	0.01	22.5 / 2.57	116.4 / 5.24	0.82 / 0.09	217.4	91.2 %	- Gray levels - Normalized - Phantom Hebrón - No neighbor interpolation
8			More refined initial mesh: - 2562 pa. - 5120 tri. - Areas: Average: 8.48 / 1.75 Max: 12.2 / 1.99 Min: 7.05/ 1.55	Free	0.0 / 0.0	25.0 / -0.1	0.005	6.24 / 1.29	72.38 / 4.95	0.0007 / 0.0005	513.53	90.35%	- Gray levels - Normalized - Phantom Hebrón - No neighbor interpolation
9			Less refined initial mesh: - 162 pa. - 320 tri. - Areas: Average: 133.38 / 6.95 Max: 183.87 / 8.28 Min: 111.8 / 6.25	Free	0.0 / 0.0	25.0 / -0.25	0.01	84.03 / 5.1	170.02 / 7.73	23.15 / 1.99	124.3	93.13 %	- Gray levels - Normalized - Phantom Hebrón - No neighbor interpolation

Table 6.7: Mesh rendering and simulation parameters associated to several measures in distance to data.

### 6.1.2.5. Volume results

The tests regarding the recovered volume have been comparing the final reconstruction with the theoretical and real volumes associated to the Phantom dataset. In all the cases we are referring to the external surface of the model to recover.

The theoretical volume can be obtained from the geometry associated to the Phantom model (see figure 5.7):

$$\begin{aligned} Vol_{theor} &= Vol_{HalfSphere} + Vol_{Cylinder} = \\ &= \frac{1}{2} \cdot \frac{4}{3} \pi r^3 + \pi r^2 h = \frac{1}{2} \cdot \frac{4}{3} \pi (35)^3 + \pi (35)^2 \cdot 55 = 305301.4 (mm^3) \end{aligned} \quad (6.3)$$

The volume value does not include the artificial cover that is added to the voxelized data. This artificial cover increments the global volume in  $20425 \text{ mm}^3$ .

Besides the theoretical volume, we can evaluate the volume associated to the acquired imagery. It is a matter of counting the voxels that contain a certain amount of property, let's say those that pass a defined threshold. It is necessary to perform a prior test, a noise filtering (deleting intensities below 20% of the maximum). Then we obtain the voxels depicted in figure 6.12.

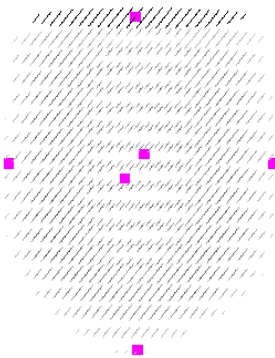


Figure 6.12: Filtered data of the Phantom dataset.

If we quantify the volume associated to the labeled voxels (those that correspond to the borders), we obtain  $368782 \text{ mm}^3$ . Then there is a difference of 12% between the volume that the images retrieve and the theoretical one. We understand that this difference is due to the quality of the caption (it is a discreet process at a very low resolution).

In order to compare the data volumes with the ones obtained from the reconstruction, we need a process in order to compute the volume associated to a triangulated mesh. For that purpose, we use the algorithm described in section 5.6.

The tests in table 6.8 were performed with the free deformation model and the smoothing algorithm active.

Following the previously described reasoning, we should compare the reconstructed volumes with the dataset volume. In this way note that the third column of table 6.8 shows the percentage of error that exists between the reconstructions and the real data.

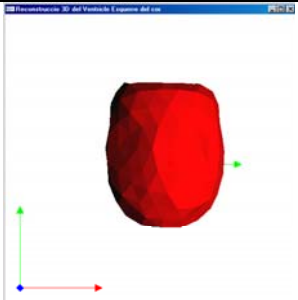
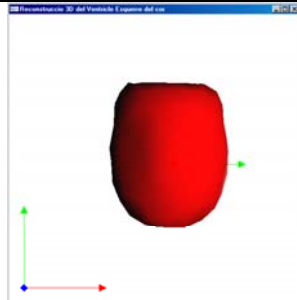
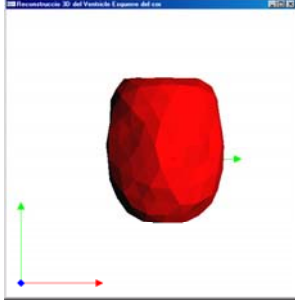
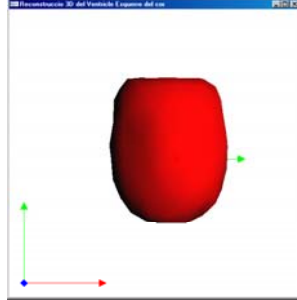
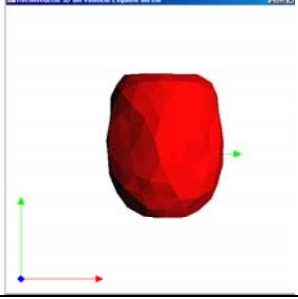
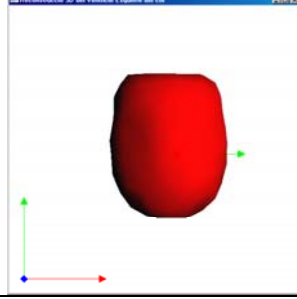
Recovering the external surface of the Phantom dataset					
Mesh (Flat shaded)	Mesh (Gouraud shaded)	Error (%)	$\Delta t$ (sec.)	Time (sec.)	Solver
		1.44	0.1	0.322	Euler
		0.68	0.12	0.14	Midpoint
		1.53	0.17	0.111	RK4

Table 6.8: Volume comparison according to the integration method.

The three methods retrieve good results:

- **Euler:** recovers  $363467 \text{ mm}^3$  vs.  $368782 \text{ mm}^3$  (real data). There is a relative difference of 1.44 % between both measures.
- **Midpoint:** recovers  $366253 \text{ mm}^3$  vs.  $368782 \text{ mm}^3$  (real data). There is a relative difference of 0.68 % between both measures.
- **Runge-Kutta-4:** recovers  $363134 \text{ mm}^3$  vs.  $368782 \text{ mm}^3$  (real data). There is a relative difference of 1.53 % between both measures.

We see that in all the cases the relative differences in volume are below 2%. It is important to note that as the method gets complicated (Euler  $\Rightarrow$  Midpoint  $\Rightarrow$  RK4), the stepsize can be increased (0.1 sec.  $\Rightarrow$  0.12 sec.  $\Rightarrow$  0.17 sec.). Then the overall performance in terms of speed gets improved (0.322 sec.  $\Rightarrow$  0.14 sec.  $\Rightarrow$  0.111 sec.).

The tests in table 6.8 were ran under a Pentium III PC (800 MHz), with 256 MB of RAM and a 3D ATI Radeon graphics card (32 MB).

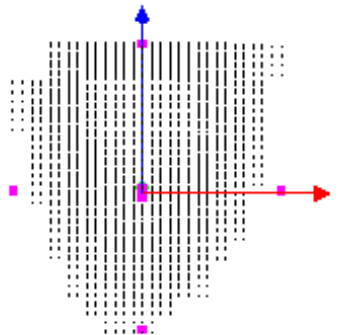
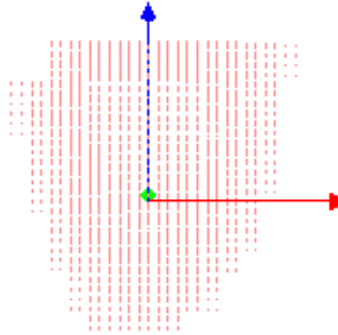
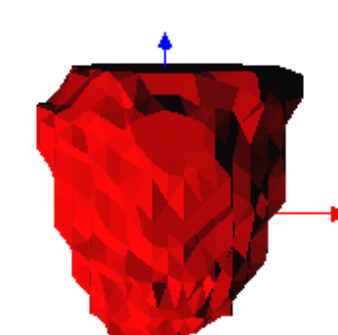
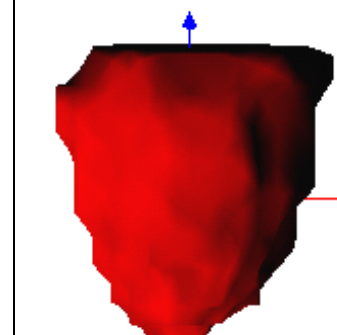
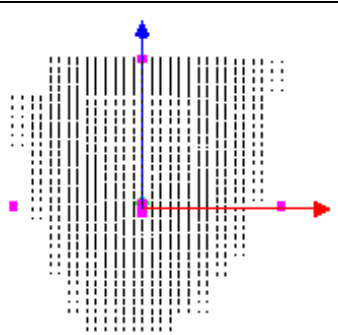
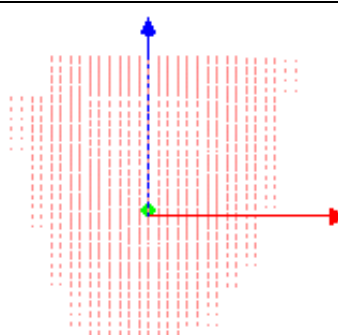
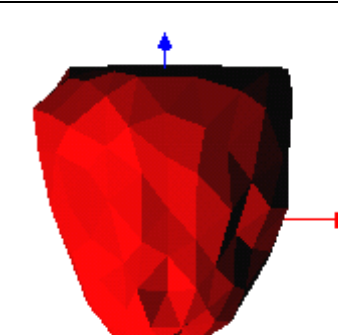
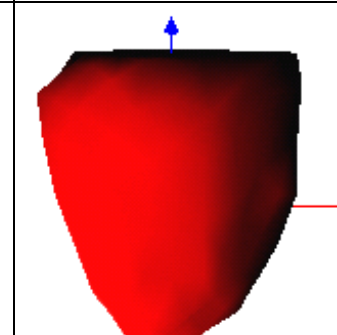
Data voxels	Border voxels	Mesh (Flat shaded)	Mesh (Gouraud shaded)	Mesh volume (Including cover)	Data volume (without cover)	Data volume (with cover)
				257597 mm <sup>3</sup>	207408.35 mm <sup>3</sup>	246825,33 mm <sup>3</sup>
				251309.12 mm <sup>3</sup>	207408.35 mm <sup>3</sup>	246825,33 mm <sup>3</sup>

Table 6.9: Volume comparison in the Phantom Mayo dataset.

For the seeking of completeness, table 6.9 presents two volume comparisons performed on a different dataset (the Phantom Mayo). This dataset is much smaller than the regular Phantom dataset (Phantom Hebrón). This fact makes the reconstruction a complex task, especially when finding the vector field for the ROI of the data.

Both tests used the free deformation model ( $\Delta t = 0.01$ ) and a very coarse initial mesh (162 particles and 320 triangles). Note how the differences between the recovered volume and the real one are minimal (relative errors of 4.36% and 1.82% respectively).

As a final test, we present two experiments regarding the reconstruction of the internal cavity of the Phantom Hebrón dataset. The simulation conditions were exactly the same as described for table 6.9. The volumes are of course smaller because we are dealing with the internal surfaces. The reconstructions were also successful retrieving relative errors of 8.36% and 1% respectively.

Note the initial and final meshes superimposed to the data voxels in the second test. These tests corresponded to an In-to-Out reconstruction paradigm (see section 5.3).

Results on tables 6.9 and 6.10 were obtained from a Pentium II PC 266 MHz equipped with 256 MB RAM and a nVidia RIVA TNT II graphics card (32 MB).

#### **6.1.2.6. CPU time measures**

Those measures are referred to the CPU time employed for all the stages of the reconstruction process. Nevertheless it must be said that mostly of the time is used in the GVF vector field generation. Some values regarding the reconstruction stage have been described previously (see table 6.7).

Table 6.11 shows two tests regarding the vector field evaluation in different Phantom datasets. In both cases, the vector field was derived using the same parameters ( $\mu = 0.15$  and 200 iterations). It is important to note that the performance in terms of CPU time is mainly governed by the total amount of voxels to evaluate, not by their spatial size.

First row shows a dataset with  $64 \times 64 \times 23$  voxels which stands for a global amount of 94208 voxels that must be evaluated. On the other hand, the second row shows a dataset formed by  $128 \times 128 \times 46$  voxels (doubled resolution) that stands for 753664 voxels. The second execution (1247.6 seconds) is basically one order of magnitude above the first (128.03 seconds).

Here we are the tradeoff then. More resolution on the images ensures a better border labeling but by paying a major computational effort when finding the vector field.

The tests were performed with a Pentium II PC, 266 MHz equipped with 256 MB RAM and a nVidia Riva TNT II graphics card (32 MB).

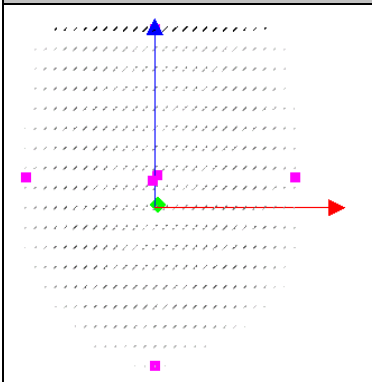
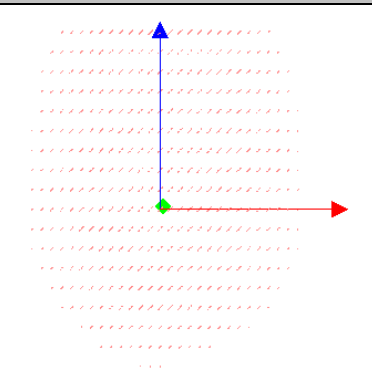
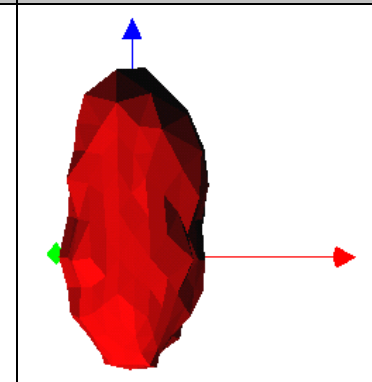
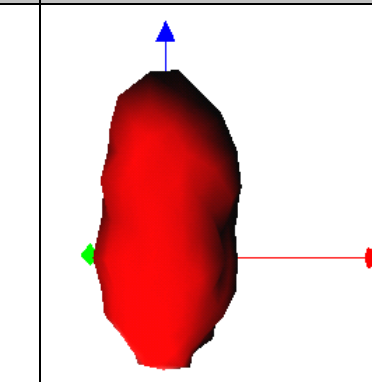
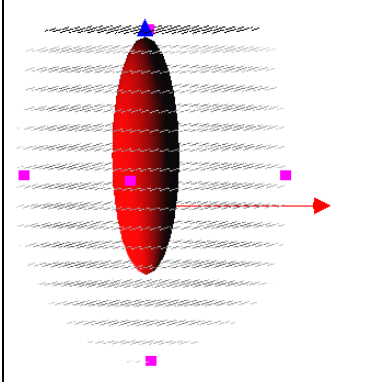
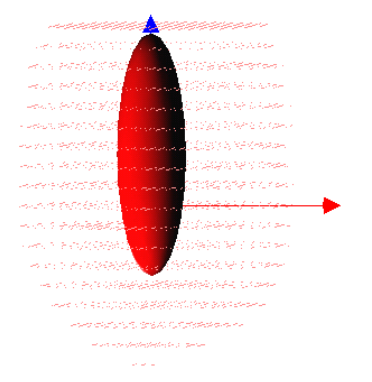
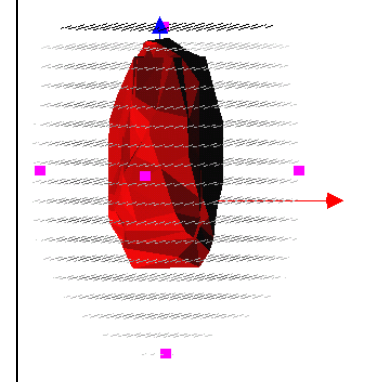
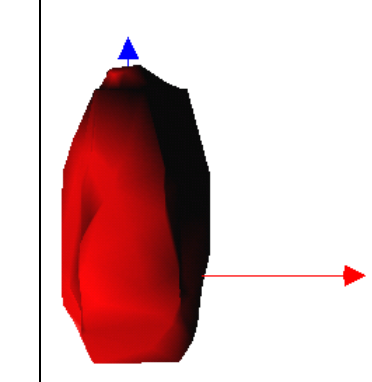
Data voxels	Border voxels	Mesh (Flat shaded)	Mesh (Gouraud shaded)	Mesh volume mm <sup>3</sup>	Data volume mm <sup>3</sup>
				39251.8 mm <sup>3</sup>	42835.5 mm <sup>3</sup>
				43259.9 mm <sup>3</sup>	42835.5 mm <sup>3</sup>

Table 6.10: Volume comparison in internal reconstructions.

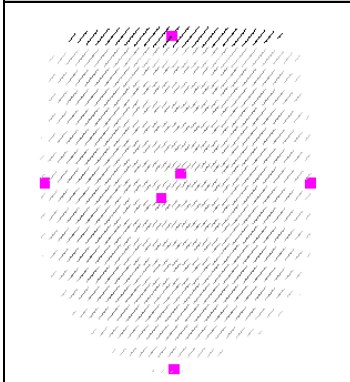
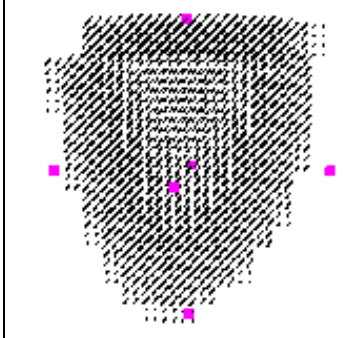
Data	Resolution (# voxels)	Spatial Dimensions for a voxel (mm)	CPU Time (sec.)
	64 (X) 64 (Y) 23 (Z)	2.87 (X) 2.87 (Y) 5.74 (Z)	128.03
	128 (X) 128 (Y) 46 (Z)	2.694 (X) 2.694 (Y) 2.694 (Z)	1247.6

Table 6.11: CPU times for the GVF computation on two different Phantom datasets.

Table 6.12 presents a second test that shows the differences in terms of number of iterations used for the vector field calculation ( $\mu = 2$ ). In there, six different executions are shown. Each of the executions ran onto the same dataset but by using a different number of iterations. The dataset was generated from an actual patient's acquisition (64 x 64 x 23 voxels) that was labeled according to our MLC implementation (see section 2.3.6).

Test	GVF Solver CPU Time (sec.)	Averaged relative error (%)	Iterations
1	30.334	0	200
2	21.119	12	150
3	14.49	27.6	100
4	10.795	36.7	75
5	7.34	48.2	50
6	3.414	61.4	25

Table 6.12: CPU times and relative errors depending on the number of iterations.

The reference test is the first one (200 iterations). This amount of iterations is considered big enough to ensure that the vectors have been correctly found. All the averaged relative errors are then computed between the examined cases (tests 2 to 6) and the reference (test 1). It can be seen that decreasing the number of iterations (from 200 to 25) decreases the required CPU time (from ~30 to ~3 seconds) but also the

quality of the evaluation. The relative error reaches averaged values over the 60% in the case of using 25 iterations only. It comes clear that there's a tradeoff between the final quality and the number of iterations which stands for the involved CPU cost. Note that the averaged relative error was computed using the magnitudes, not the single components, of the vectors. The PC consisted on a Pentium III with 256 MB RAM and an ATI Radeon 32 MB graphics card.

As a final test regarding the computational overhead in terms of time, we present an estimation of the global latency that stands for the whole pipeline in table 6.13. In this test we used a complete cardiac cycle consisting on 8 captures of 64 x 64 x 23 voxels each. Times are expressed in seconds. The whole process is presented for a single external mesh, for the eight meshes of the external surface and for the 16 meshes of the internal and external surfaces.

Type	Canny edge detector <sup>1</sup>	MLC filter	GVF <sup>2</sup>	3D Reconstruction <sup>3</sup>	Interpolation <sup>4</sup>	TOTAL
Single mesh (external)	0.37	0.41	28.7	2.63	--	32.08
Cardiac cycle (8 instants for the external)	2.96	3.28	229.6	21.04	0.12	257
Cardiac cycle (8 instants for both surfaces)	5.92	6.56	459.2	42.08	0.24	514

<sup>1</sup>  $\sigma = 1.8$ , thresholds 0.3 and 0.7  
<sup>2</sup> Time includes the solver plus all the auxiliary processes; 150 iterations;  $\mu = 2$   
<sup>3</sup> Free deformation model, smoothing algorithm active  
<sup>4</sup> 30 keyframes between original meshes

Table 6.13: CPU times associated to the latency of the whole pipeline.

The most complete reconstruction takes 514 seconds to be done (between 8 and 9 minutes) in a regular PC (Pentium III, 256 MB RAM, ATI Radeon 32 MB graphics card). As expected the GVF evaluation takes most of the time (nearly the 90% of the global latency) which drives us to a conclusion: the method which is explicit might be implemented in an implicit manner in order to accelerate the process.

### 6.1.3 Missing-data results

When a patient has had a heart attack some of the areas of his heart become ischemic and, because of the absence of blood irrigation, the data that we obtain can have some missing zones. Evaluating recovering results with partial missing data require using different fillings of the Phantom volume. We made some recovering test experiments with 10%, 32% and 53% percentages of missing volume data, always referred to the 100% of the total. Figure 6.13 shows a graphical view of the voxelized version of a partially filled Phantom.



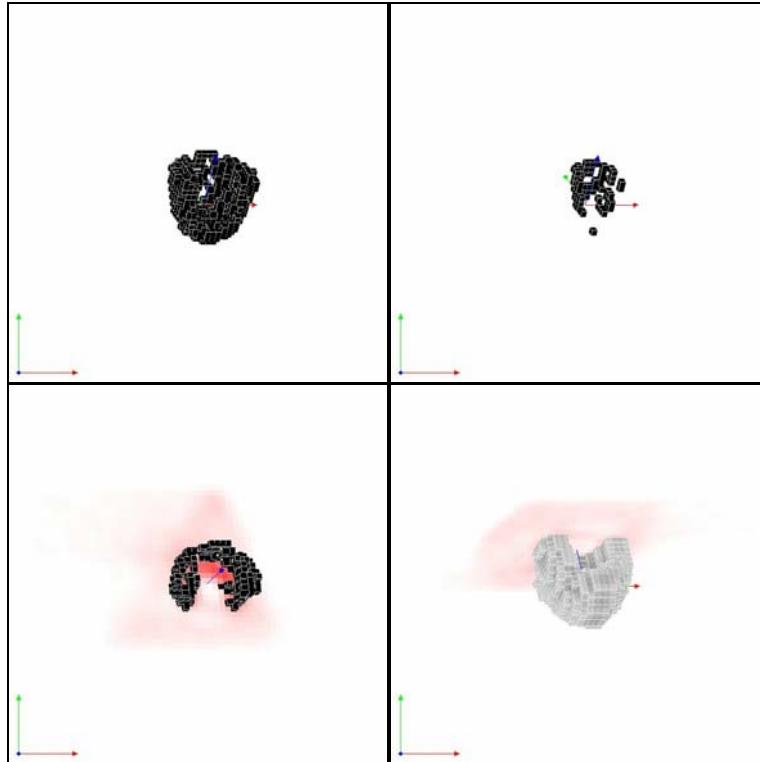


Figure 6.13: Partially filled Phantom dataset.

For this test, we used the Phantom Mayo dataset which is characterized by an overall volume of  $265501 \text{ mm}^3$ . The images corresponding to an acquisition of this Phantom, with no loss, are shown in figure 6.14.

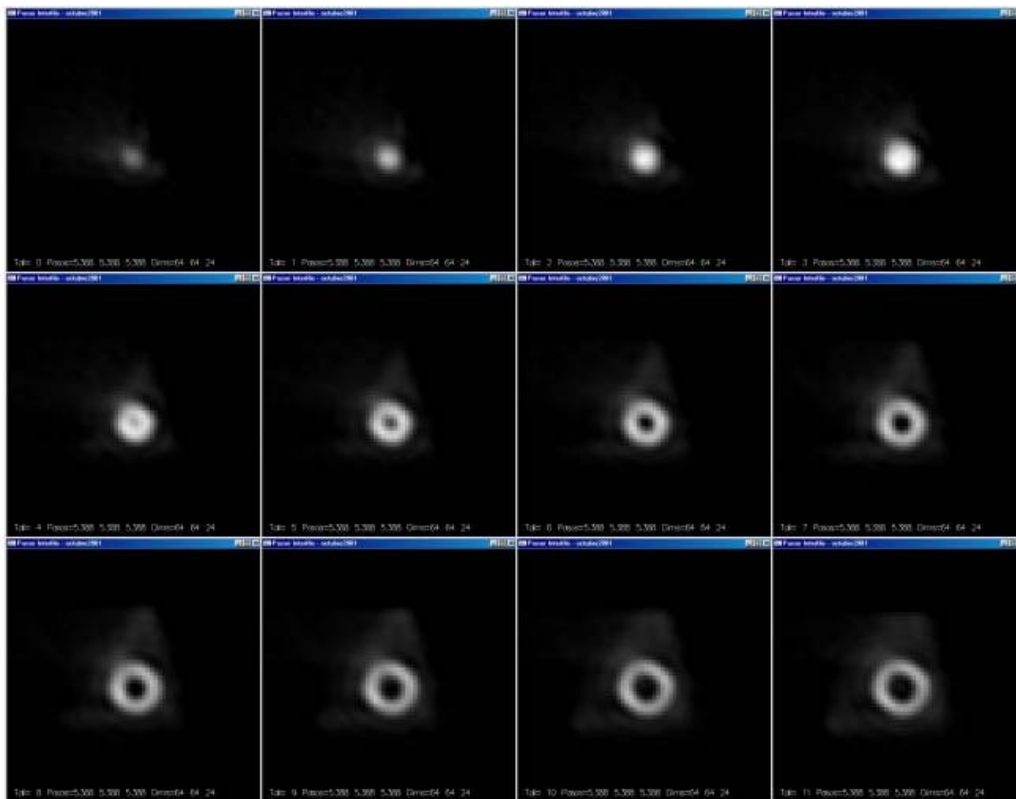


Figure 6.14: The Phantom Mayo dataset.

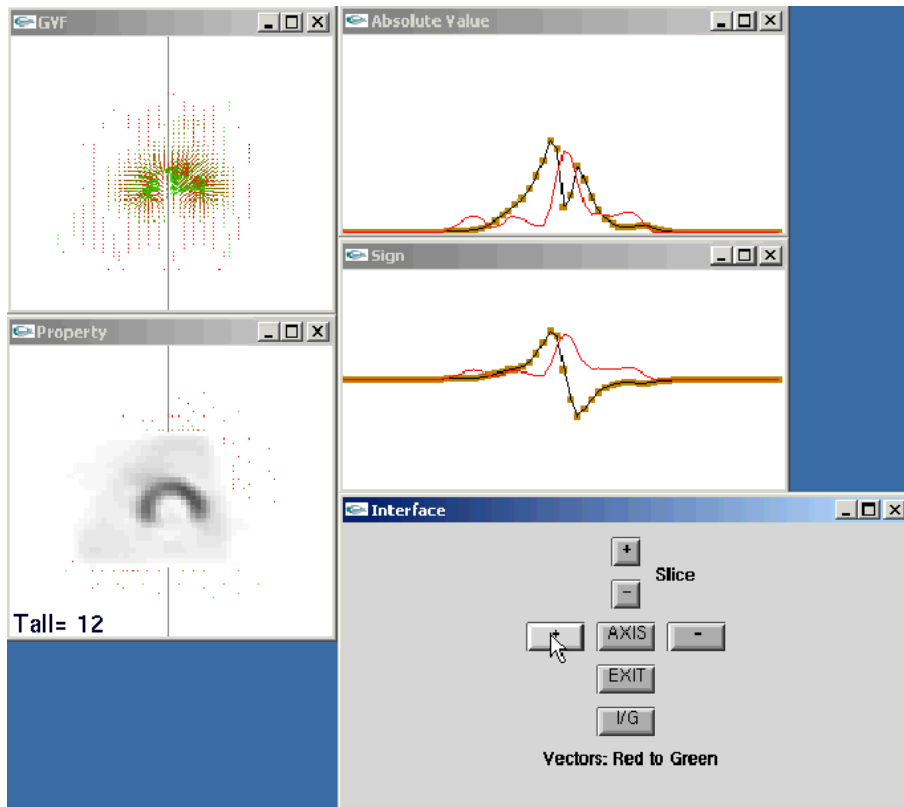


Figure 6.15: Data and GVF field for the 32% loss dataset.

Figure 6.15 shows an example for one of the original datasets, the 32% loss pack of the Phantom Mayo. Note the simulation of lack in blood irrigation in the data window (bottom-left), where the slice is unexpectedly open in terms of property, no longer circular. Table 6.14 shows the results.

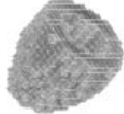




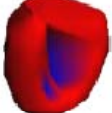
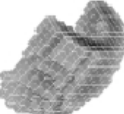

		<b>Volume=265501 mm<sup>3</sup></b> <b>% of total=100%</b>
		<b>Volume=250944 mm<sup>3</sup></b> <b>% of total=94.5%</b>
		<b>Volume=188360 mm<sup>3</sup></b> <b>% of total=70.9%</b>
		<b>Volume=159970 mm<sup>3</sup></b> <b>% of total=60.2%</b>

Table 6.14: Recovered surfaces from partial data.

The first column shows the initial data to be recovered; the second column depicts the final meshes and the third column points out the recovered final volumes (absolute values and percentage of total).

As we can see, the best recovering is the third one because it gives a percentage of missing volume of 29.1% against the 32% of the real data emptied. This represents a relative percentage error of 0.09. For the other test examples we obtained relative errors of 0.45 and 0.24 respectively.

Figure 6.16 shows some graphical results regarding the labeling, well-fit to the data.

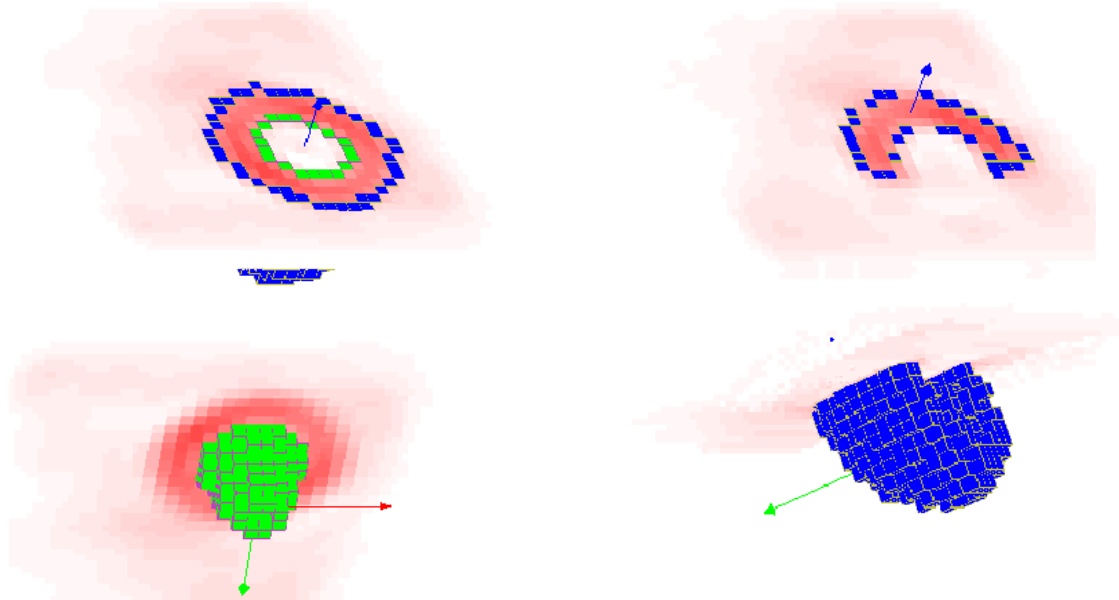


Figure 6.16: Internal and external labeling for the complete (left) and partial (right) datasets.

#### 6.1.4 Complete cardiac cycle

Table 6.15 presents a complete cardiac cycle reconstruction, recovered from actual patient's data. The cycle is formed by eight temporal acquisitions. Each data set consists on  $64 \times 64 \times 24$  voxels, with spatial resolutions of 2.87 mm (X), 2.87 mm (Y) and 5.74 mm (Z).

The meshes were generated using the free deformation model with the RK4 explicit scheme, using a GVF balancing constant of 10, a damping factor of 1% and a stepsize of 0.1 seconds.

One can notice the changes in volume as the organ beats (from systole to diastole), here noticeable as the sequence defines a complete cardiac cycle. Internal mesh, external mesh and wall volumes ( $\text{mm}^3$ ) are presented in table 6.16. Then ejection fraction (see appendix C) can be calculated for this ventricle as equation 6.4 shows. In there all the volumes are internal. As it is showed, its value is inside the interval 50% - 70% which states for a non-pathological situation. Moreover, physician's 2D software gave an ejection fraction value of 53% for this case which is really close to our computation.

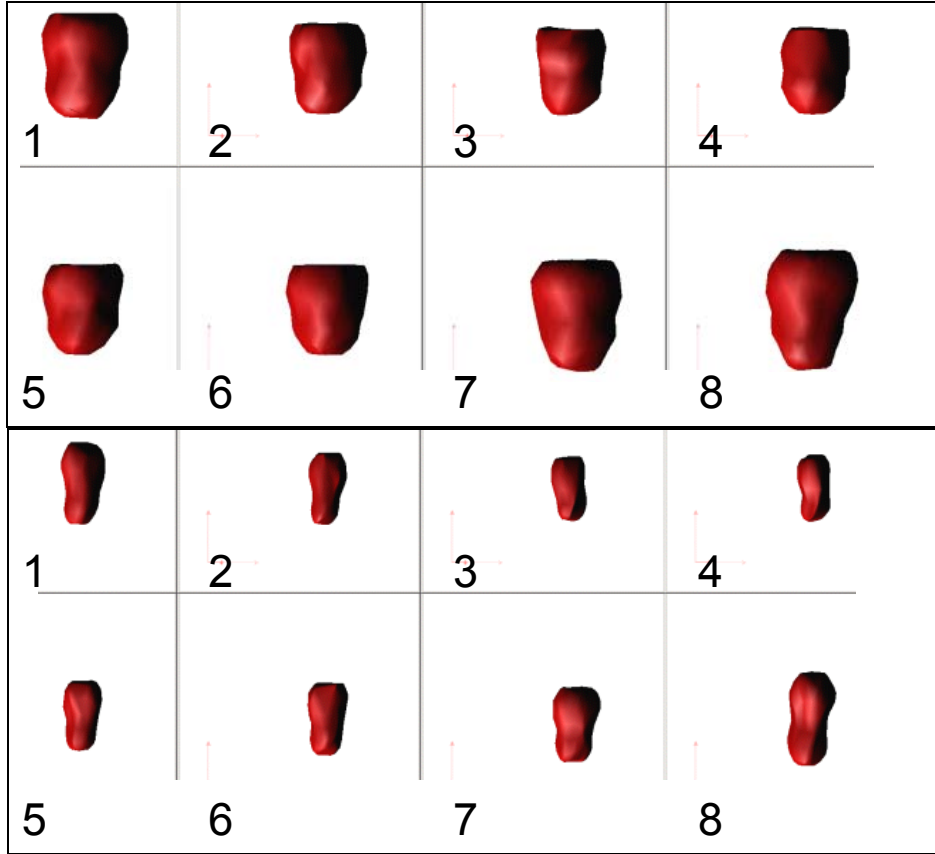


Table 6.15: Complete cardiac cycle with external surfaces (top) and internal surfaces (bottom).

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>Int.</b>	74741	45677	36523	32956
<b>Ext.</b>	443762	316804	254112	250131
<b>Wall</b>	369021	271127	217589	217175
	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
<b>Int.</b>	47237	55825	63133	79581
<b>Ext.</b>	299779	345830	451525	473608
<b>Wall</b>	252542	290005	388392	394027

Table 6.16: Obtained volumes for the eight temporal instances.

$$\begin{aligned}
 EF &= \frac{\text{DiastoleEndVolume} - \text{SystoleEndVolume}}{\text{DiastoleEndVolume}} \\
 &= \frac{79581 - 32956}{79581} = 0.586 \Rightarrow 58.6\%
 \end{aligned}
 \tag{6.4}$$

Figure 6.17 shows the complete process for one of the reconstructions in this cardiac cycle. In this case the external and internal surfaces of the first dataset on the cycle.

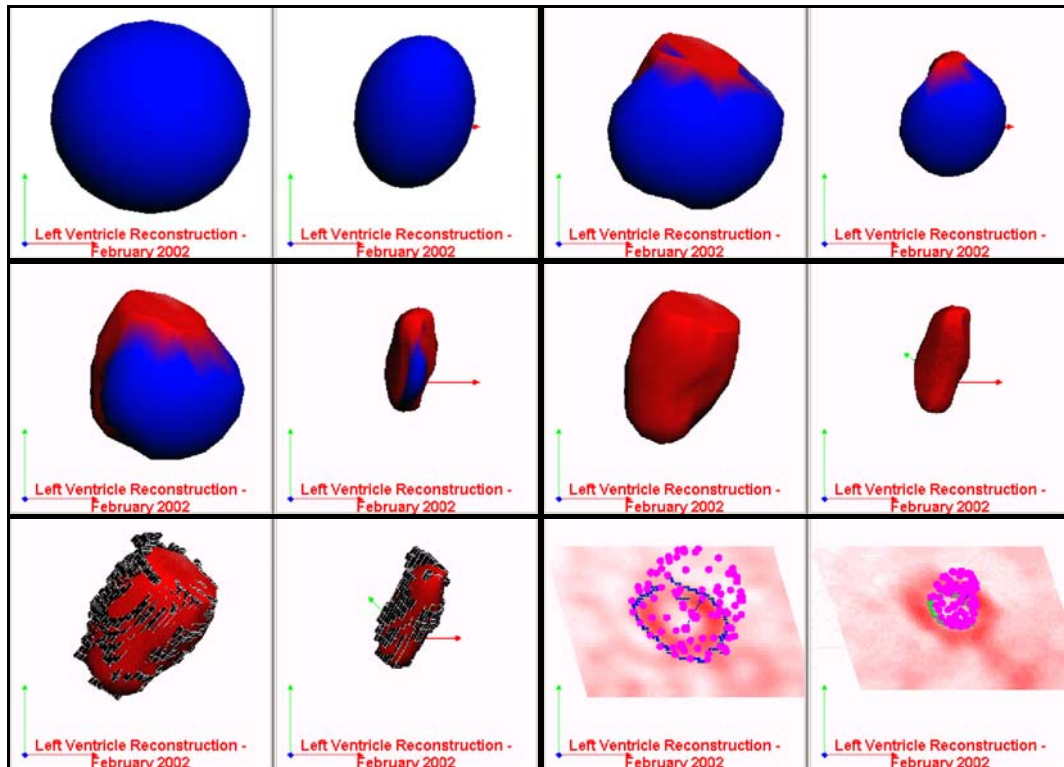


Figure 6.17: A complete reconstruction process.

The sequence shows first the initial meshes in blue. The reconstruction begins and the meshes turn red as long as the particles reach the marked voxels borders. Finally the figure shows the border voxels (black) superimposed to meshes, the particles over a data slice, and some particles over the external (blue) and internal (green) frontiers.

### 6.1.5 Pathological cases

In some pathological cases, it can be interesting to use data from nuclear ventriculography (see appendix A) instead of using standard perfusion imagery. If the volume of ischemic tissue is really severe, the detection of the inner borders can be better achieved by this way, like shown in figure 6.18.

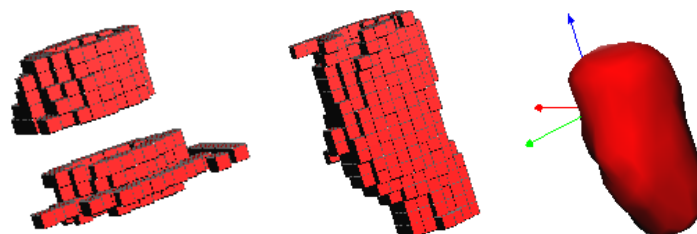


Figure 6.18: Perfusion borders (left); Ventriculography borders (middle); Final reconstructed mesh (right).

As shown in figure 6.18 for the internal surface, data borders detected from the perfusion data present an important lack of consistency. If we take a look at the same test done over the ventriculography, the borders are much better labeled. Then the final surface can be perfectly recovered for the case.

We also present six pathological cases that have been treated with the proposed algorithms. All of them were simulated using the Runge-Kutta 4 solver, the free deformation model and the final smoothing process. The tests were performed in a Pentium III PC with 256 MB RAM and a 32 MB accelerator card (mobility ATI Radeon).

Figure 6.19 shows two datasets of ventricles originally affected by digestive activity in their apex. However, the algorithm can distinguish correctly the left ventricle borders. Note how the surfaces are well-fit to the voxelized data to recover. The simulations took 0.56 (top) and 0.45 (bottom) seconds.

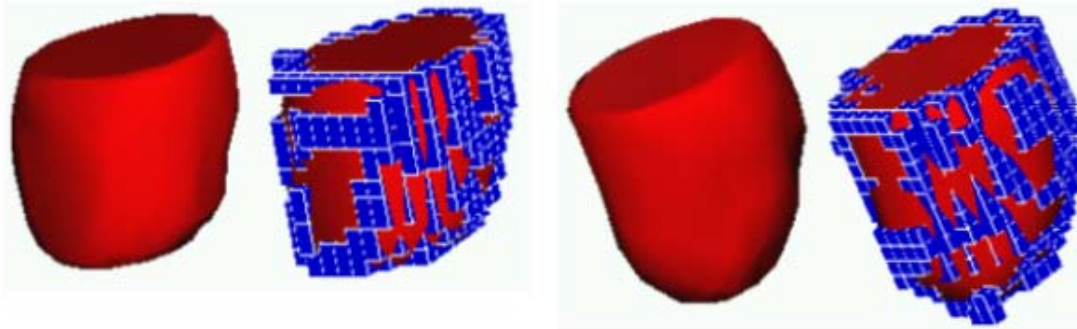


Figure 6.19: Digestive activity perturbing initial datasets. For the two presented cases: Left, final external surfaces; Right, final surfaces over the dataset to recover.

Figure 6.20 shows four more cases. Their respective initial datasets presented occlusion defects located at several places. The defects were classified by physicians as inferior (1), apical (2), side (3) and vast anteroapical (4). The simulation times took 0.32, 0.45, 0.44 and 0.68 seconds respectively.

Note how the lack of information in the original datasets influences the final reconstructed surfaces (cases 3 and 4). The final smoothing process avoids the apparition of creases along the mesh.

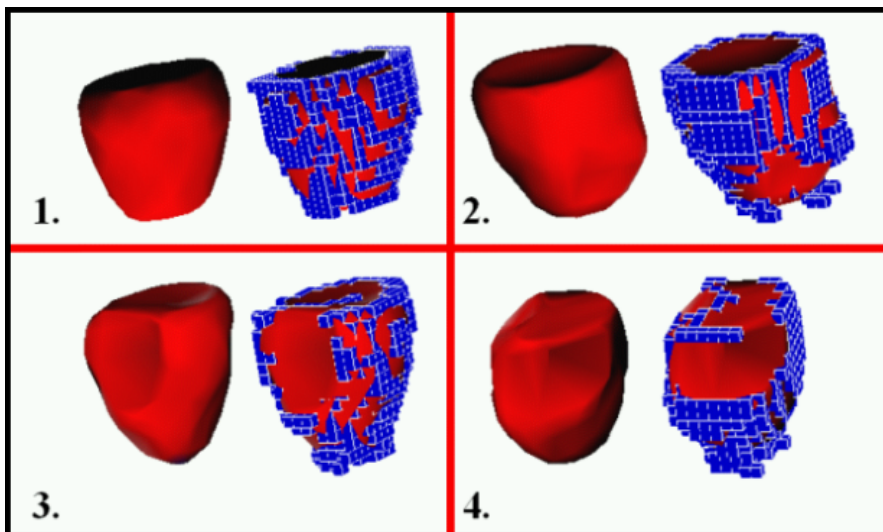


Figure 6.20: Occlusions perturbing initial datasets. For each case the final external surface is rendered in the left and mixed with the dataset to recover in the right.

### **6.1.6 Other explored approaches**

There are other approaches to explore such as automatic contouring and tessellation of the 3D shape by using the discrete contour deformation model. This methodology is applied to our SPECT datasets in order to get a more complete feedback.

We also point out the research that is being done in flexible volumetric modeling for the left ventricle, by other members of our research group. This is the next stage of the whole process and it is required in order to allow real time interaction with the organ.

We finally test our imagery with the Anisotropic Contour Completion operator. This is a very novel technique that has been published recently. The technique takes into account the local orientation of the contours to be closed, which fits well with cardiac images of the left ventricle.

#### **6.1.6.1. Automatic contouring and tessellation of the 3D shape**

The discrete contour deformation model (see sections 3.2.4, 3.3.3 and 3.4.1) can be extended to a 3D framework. Moreover that, we can use this method to derive 3D surfaces of the left ventricle walls. It can be used as a new tessellation method.

The process consists on finding a first contour by the usual method. This contour will be used as an initial template for the rest of the slices. The process is described below:

- Begin by finding the 2D contour for the best slice (in terms of property clarity).
- Use this contour as a template for the neighbor slices (top and bottom).
- Repeat the process for all the slices, using the immediate contour as the initial template for the actual contouring.
- Once the system has found all the contours, tessellate a 3D mesh along them.
- Close the boundaries of the mesh in the top and bottom contours. This will ensure that volume evaluations are correct.
- Smooth the mesh if needed.

Figure 6.21 presents the template contour for a test using actual patient's data (external surface tessellation). In this case we used the same instant of the cardiac cycle that figure 6.17 shows. The initial contour was found in slice 16, considered to be representative enough in terms of quality and clarity.

From that initial contour, we segmented the whole dataset by applying the previously described algorithm. Figure 6.22 shows the final result.

Once the whole dataset has been segmented in 2D, we apply our tessellation strategy in order to build the 3D surface that we are looking for. Figure 6.23 shows the final render in either its wire frame (left) and shaded (right) versions.



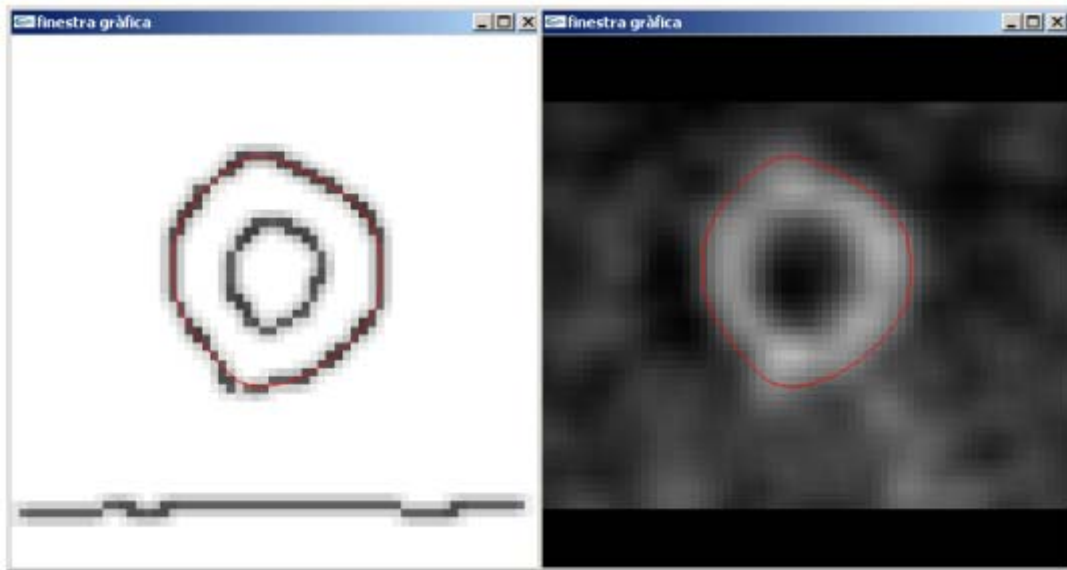


Figure 6.21: Template contour for the discrete contour deformation model.

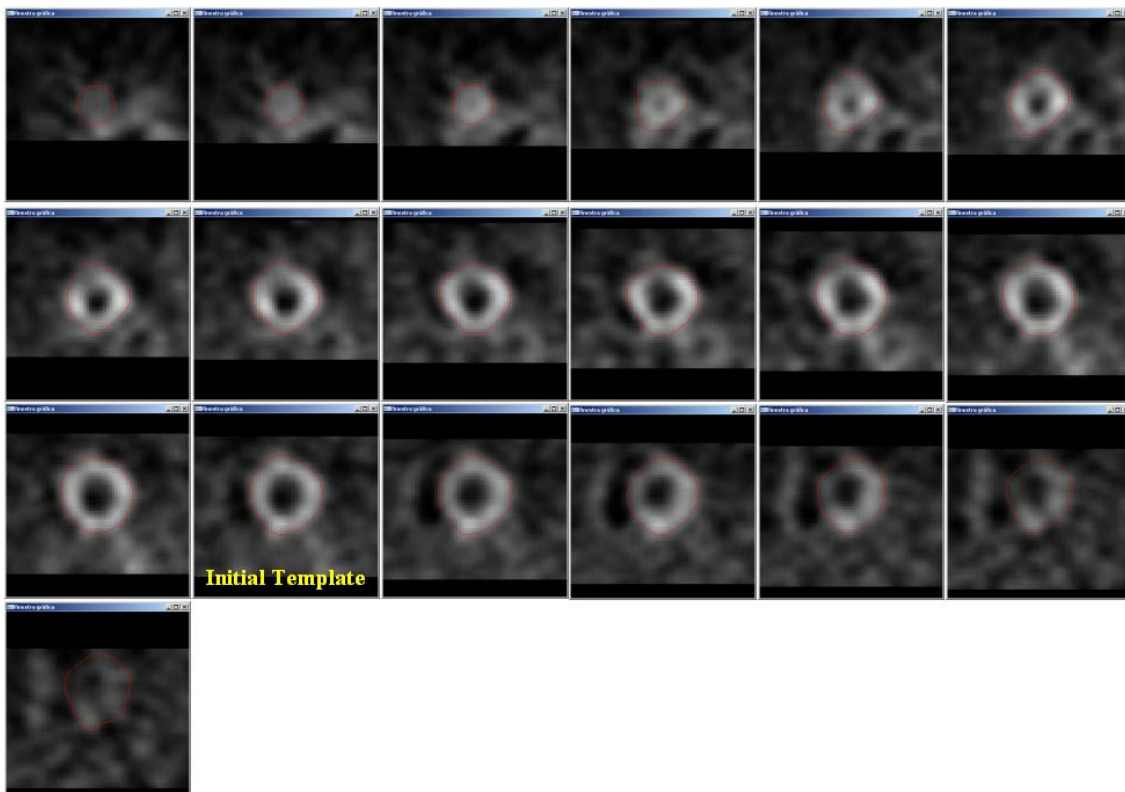


Figure 6.22: Segmentation of a whole dataset by the discrete contour deformation model.

We can apply this methodology to the reconstruction of the cardiac cycle, like in section 6.1.4. Figure 6.24 shows the external and internal meshes retrieved for the same dataset but when using this approach.



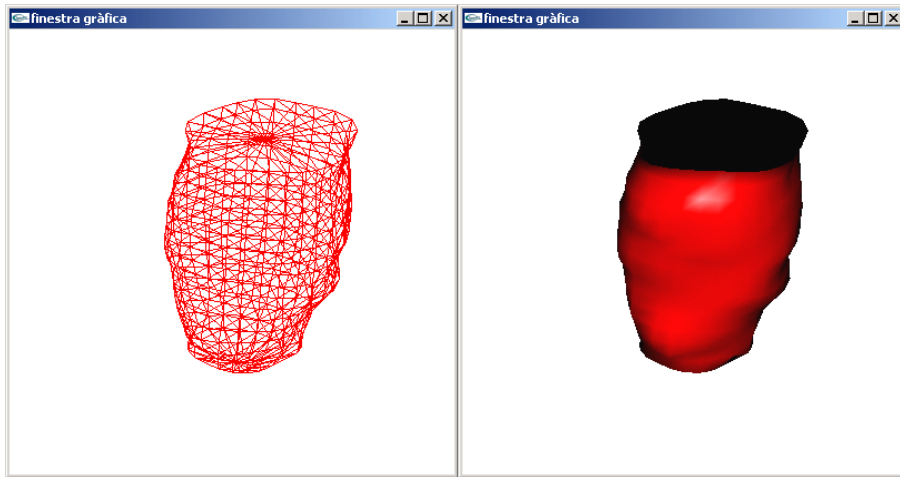


Figure 6.23: 3D recreation of the external surface by the new tessellation method.

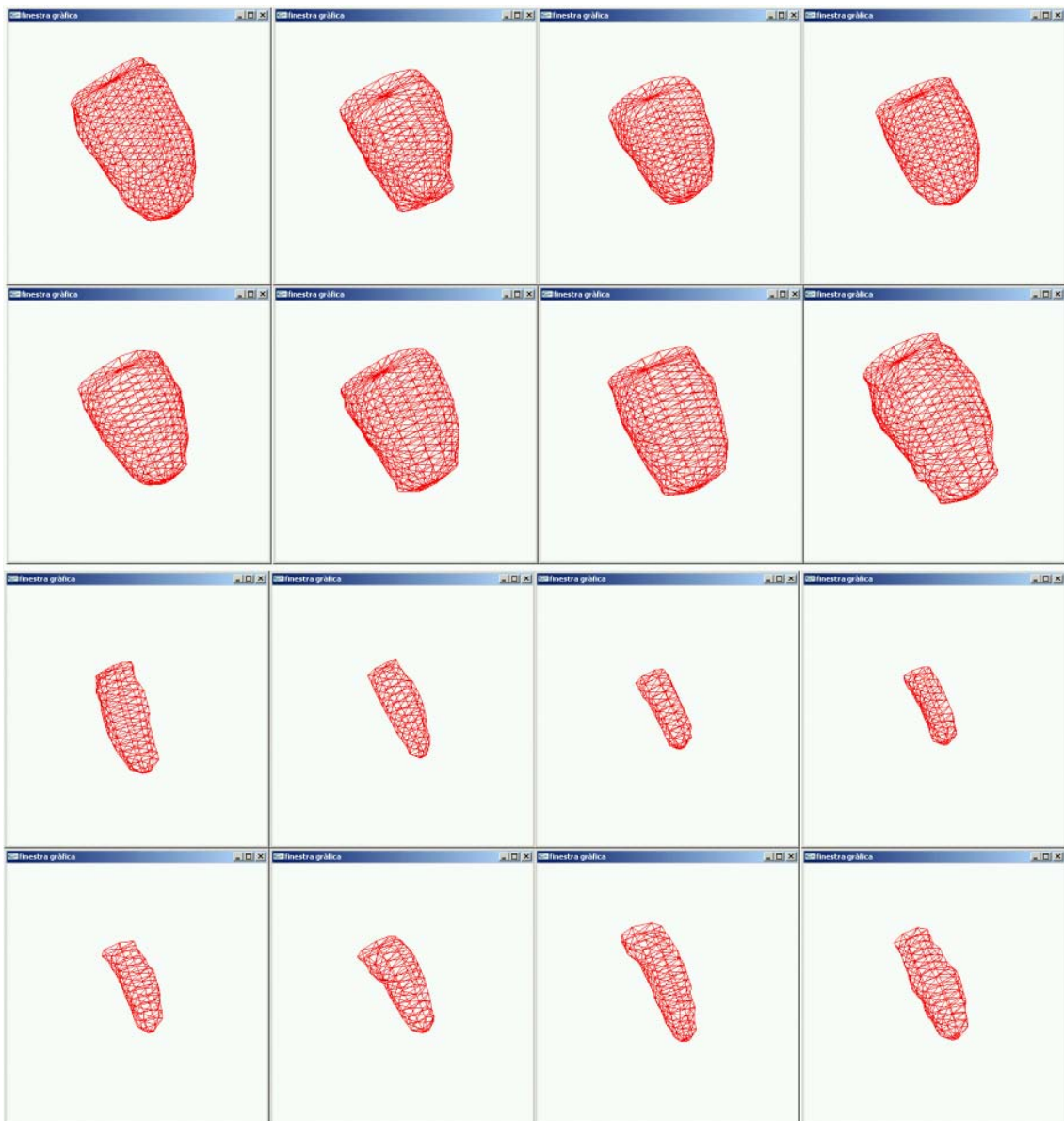


Figure 6.24: The cardiac cycle retrieved by the discreet contour deformation model.

Where rows one and two correspond to the external surfaces and rows three and four to the internal ones. We can evaluate the volumes and the ejection fraction like in table 6.16 and equation 6.4. The results can be seen in table 6.17 and equation 6.5 respectively.

All the volumes are inferior in absolute terms but the ejection fraction is practically equal. In that sense, we can state that this test retrieved lower volumes but as long as this fact is consistent in all the reconstructions, the ejection fraction, which is a relative parameter, can be considered as good as the previous one. The extension of the discreet contour deformation model to 3D is being researched so far. In that sense, those are promising results.

We find the volume by calculating the area of each contour. We subtract the area of the first contour to these (Green's theorem) and we multiply the result by the thickness between slices. We add all the partial volumes.

Then we have to evaluate the contributions of all the lateral triangles that link the different contours. The contribution for a triangle is calculated like the area that it projects multiplied by the slice thickness and divided by a factor of 2. Depending on the sign of the normal vector, this contribution is added (positive normal) or subtracted (negative normal).

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>Int.</b>	49973	29964	20849	18929
<b>Ext.</b>	349349	264170	215068	201387
<b>Wall</b>	299376	234206	194219	182458
	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
<b>Int.</b>	27440	40689	47080	49034
<b>Ext.</b>	247078	291363	322970	373602
<b>Wall</b>	219638	250674	275890	324568

Table 6.17: Volumes for the eight temporal instances, applying the discreet contour deformation model.

$$\begin{aligned}
 EF &= \frac{DiastoleEndVolume - SystoleEndVolume}{DiastoleEndVolume} \\
 &= \frac{49973 - 18929}{49973} = 0.621 \Rightarrow 62.1\%
 \end{aligned}
 \tag{6.5}$$

### 6.1.6.2. Flexible volumetric model of the left ventricle

In order to interact with the organ in real-time, like in a surgery simulator, we need to add volume to our surfacing solution. This task is accomplished by a module that receives our surfaces as its inputs generating a volumetric model of the left ventricle

[64]. Although this algorithm is not within the scope of this project, we consider necessary explaining it in a few words for a better description of the complete VR tool.

The volumetric model can be found by using a model that mixes two different paradigms: the Finite Element and Mesh Free Methods [43]. This hybrid approach constructs a multi-resolution model that can be used for real time interaction purposes. These simulations are characterized by the dynamic interaction between the deformable model and some possible external forces acting on it. The dynamic behavior of the volumetric model is based on linear elastic mechanics.

In fact the model consists on several layers that start with a very coarse mesh that is being refined into a more complex representation. The computational accuracy is always related to the amount of elements (tetrahedra in the case of volume) and their associated sizes. If the user applies an external force to the model, the algorithm activates the refined mesh in the appropriated area. The rest of layers are used to animate the model according to their distance from the force location. When thinking about obtaining the different multiresolution meshes, it is easy to select one of two choices available: a refinement of the previous level mesh or a completely new and independent geometry.

It is necessary to work with several meshes which results on the increase of the complexity of the data structure. We can overcome that by using a single and coarse mesh while applying another multiresolution methodologies to the deformable model. The idea behind this is to refine the zone where the forces are acting and to use a Mesh Free Method (MFM) in order to compute the elastic reaction of the model.

Figure 6.25 shows two views for the volumetric model of the left ventricle. Left side shows the external surface as we have dealing in this project. Right side shows the innerities of the volumetric model, where two multiresolution meshes have been built.

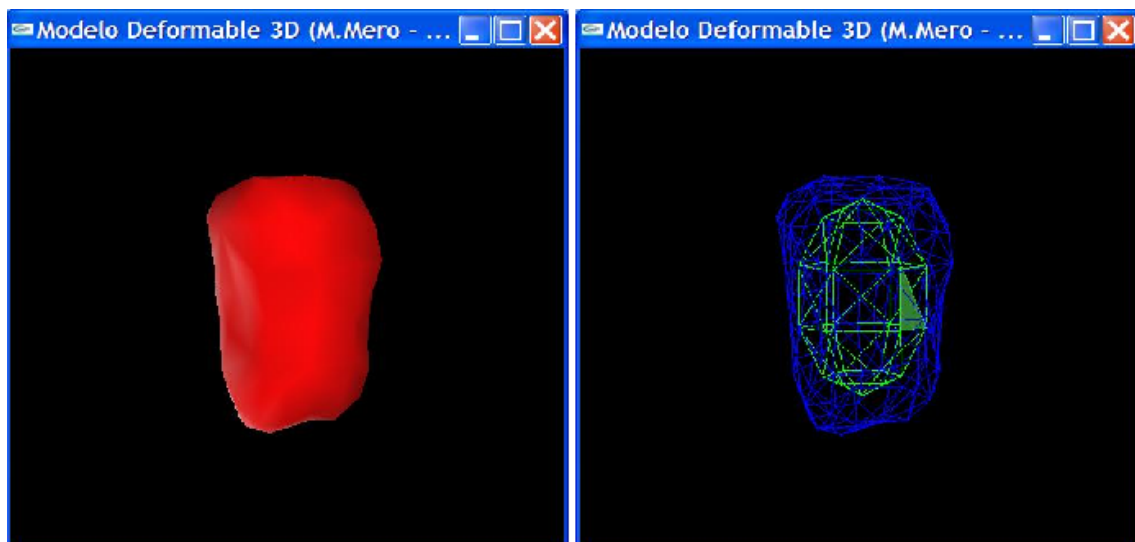


Figure 6.25: Volumetric model of the left ventricle.

### 6.1.6.3. Anisotropic Contour Completion applied to left ventricle imagery

Anisotropic Contour Completion (ACC) is a novel application of the diffusion tensor for anisotropic image processing presented by Gil, Radeva and Vilariño [37]. The

technique takes into consideration the local orientation of the contours to be closed. This fact can be exploited in our case, where the images to segment are circular-based. It can be interesting to add an extra knowledge of the images to the system, by using this operator.

In fact the method recovers a closed model of the curve that can be refined by other means such as ours. The idea behind the method is to use the initial image as an initial heat distribution, in our case the irrigation values of the SPECT images. With these initial values a diffusion heat process is started using a local metric tensor, based on the principal component analysis, chosen to drive the heat diffusion in the tangent direction of the level contours. This way a more noise robust approach to the contour detection can be achieved. For our noisy images this is a very valuable property.

The technique has been tested in a cardiac sample in order to complete our results. After applying the novel operator, a generic edge detector is used (Canny Edge Detector). The results for all the slices in this dataset are shown in figure 6.26.

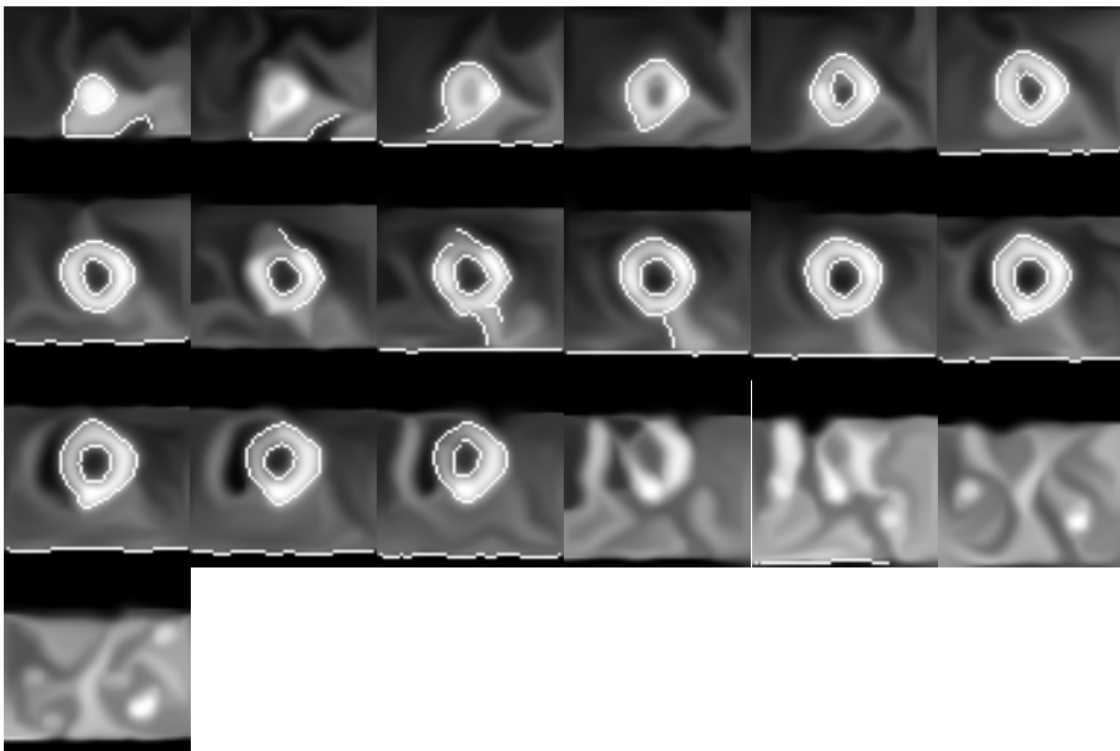


Figure 6.26: ACC as a preprocess for SPECT imagery segmentation.

The test retrieves clean and closed contours in the majority of the images. Spurious edges do not appear making it easier for our MLC implementation to operate. Nevertheless, there are some incomplete contours, for instance in slice 8 (second row, second image in the left), where the contour is not closed around the cardiac tissue. Fortunately our implementation can handle these situations by applying vertical coherence to the bounding box filtering stage. The bounding boxes do not suffer from bad alignments even if this type of incomplete contours appear.

Figure 6.27 shows the results after applying our MLC implementation to slice 8 of the dataset in figure 6.26. The Canny edge detector is shown in yellow and the internal and

external labeling are rendered in blue and red respectively. Note that the labeling is correct and the bounding boxes are well-aligned.

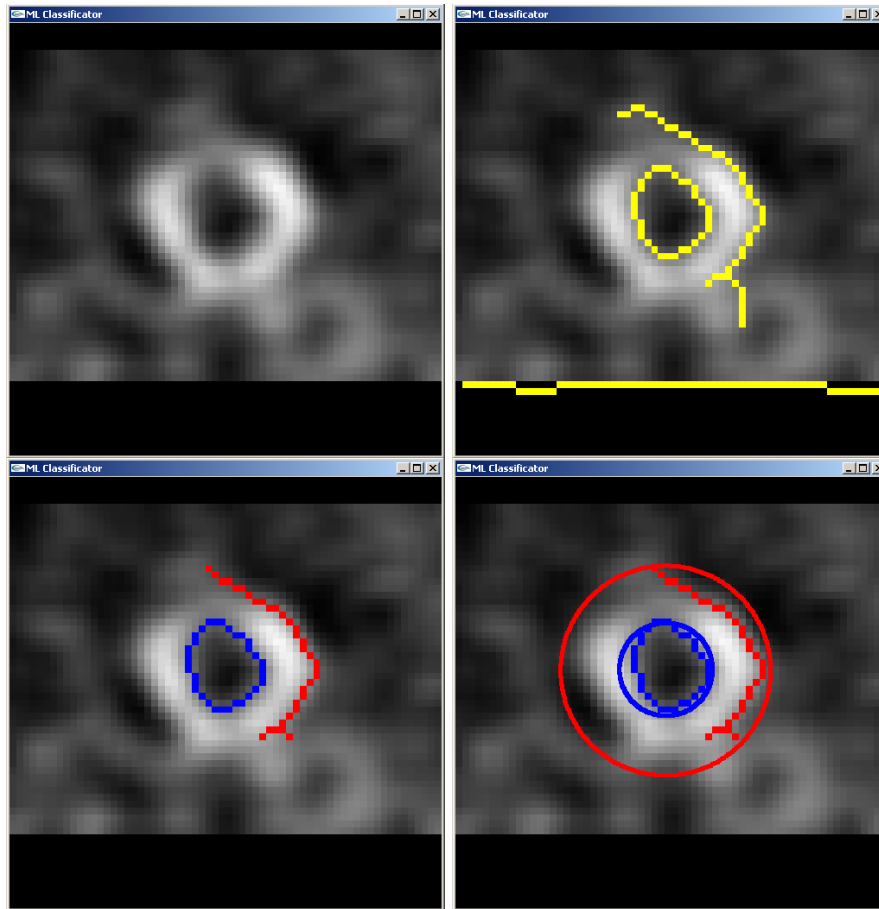


Figure 6.27: Borders are correctly labeled even if incomplete edges are retrieved.

If we mark the labeling for the whole dataset in order to perform a 3D reconstruction, we obtain the surfaces in figure 6.28. The GVF evaluations took 44.18 / 42.561 seconds for the internal / external borders respectively (64 x 64 x 23 dataset). The 3D reconstruction took 0.935 seconds for the external surface and 0.887 seconds for the internal. Tests were executed in a Pentium IV 1.7 Ghz, 512 MB RAM, GeForce 4 FX with 128 MB graphics card.

The resulting volumes are of 56166.5 mm<sup>3</sup> (internal) and 325142 mm<sup>3</sup> (external). These measurements are slightly smaller than those shown in section 6.1.9 (for the first dataset of the cardiac cycle which corresponds to this case). Differences range from 25 to 28%. The main reason for this is that some slices have no contouring at all (besides slice 8, see the last four slices in figure 6.26) and then do not contribute to the overall volume.

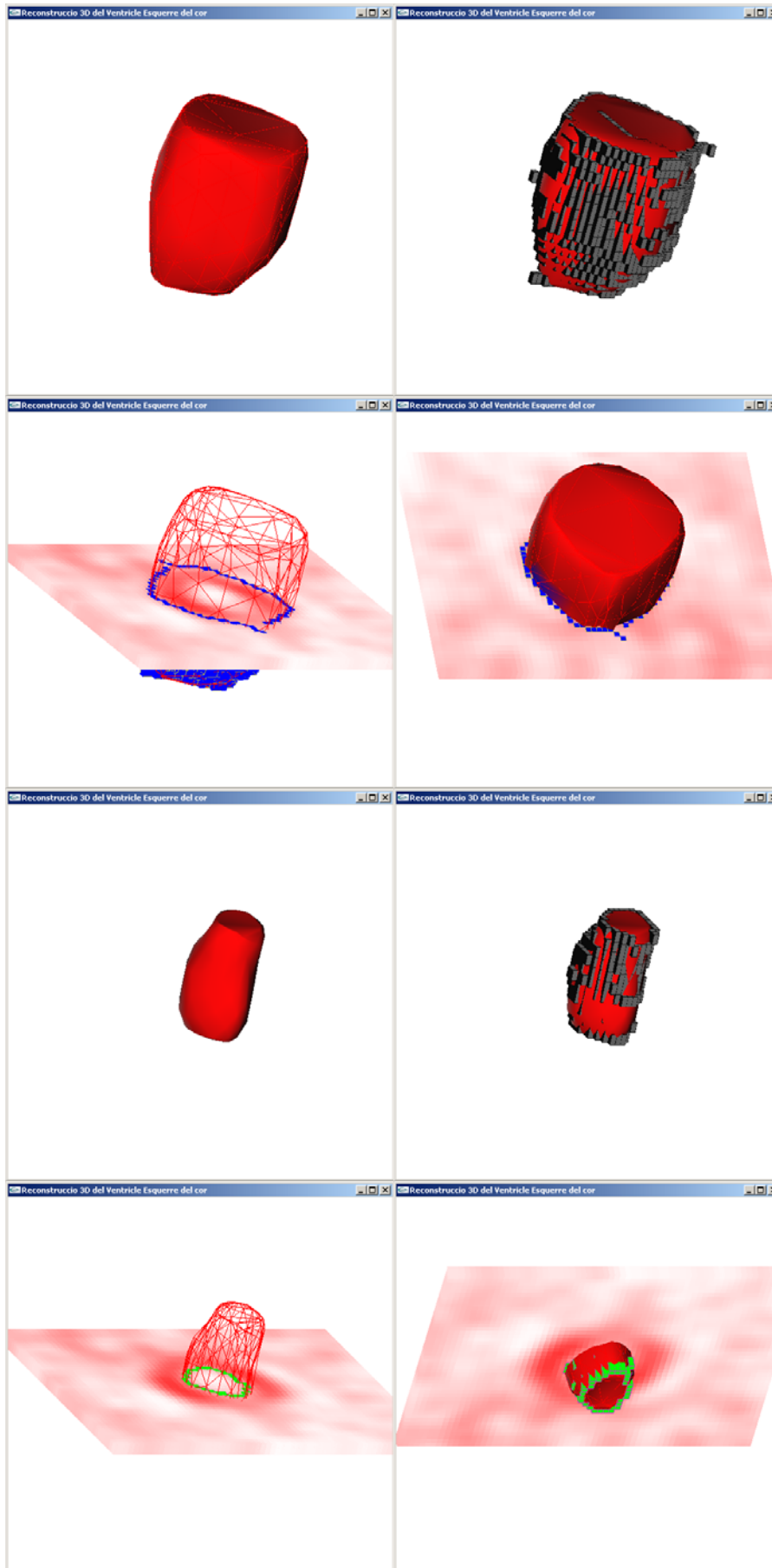


Figure 6.28: External and internal surfaces for the ACC operator.

## **6.2 Generic reconstructions**

Our methodology can be extended to other sort of problems in a generalist way. 3D reconstruction can be interesting in other fields such as terrain generation, procedural modeling, architecture or animation, among others.

This section presents several results that demonstrate how our reconstructions can serve for getting 3D models at low cost, in terms of manual interaction and CPU computation cost.

### **6.2.1 Low-polygon modeling and mipmeshing**

When dealing with real time engines, low polygon modeling is a key process that ensures real time response. If an object is far away from the viewer, we will not perceive its detailed shape. Then it is convenient to use a reduced version from its geometry in order to ensure better CPU costs. We build several versions of the geometry from different meshes that represent the same object with more or less polygons. This is what the literature calls mipmeshing [30]. When rendering the object, we must check its distance to the camera, or viewer, and select the appropriated mesh. There is a clear tradeoff between the amount of polygons and the visual quality the we must offer. No quality loss should be perceived by the final user.

Nevertheless, this methodology presents several disadvantages:

- The user, or the programmer as a preprocess, must consider the distance intervals associated to the levels of detail.
- The distance determination does not take into account the field of view. If this parameter changes during the animation, due to changes in the camera initializations for instance, the distance intervals will no longer be useful. The sizes of the object that the user sees depend strongly on the distance to it and on the field of view.
- If we move away from the object, the system will produce a change in the level of detail that may be visually perceived by the user. An abrupt change when substituting a model by a reduced version of itself.

Even with the disadvantages stated, this technique is very useful if we take into consideration that no changes in the field of view occur in most of the cases. Moreover that, the transition problem can be solved by implementing an interpolation algorithm that evolves from one mesh to another. As a final statement, we can determine when to change the model if we track the bounding box associated to the object. We can project this bounding box onto the camera frame and see what is the approximated size of the object, once projected. Then it is easy to select the correct model to display automatically.

Figure 6.29 presents a mipmeshing example. In this case, the animation presented 200 simultaneous hammers that had to be rendered and animated in real time. With the application of this technique, the system runs in real time in a medium PC (Pentium III, 256 MB RAM and a 32 MB Ati Radeon accelerator card).

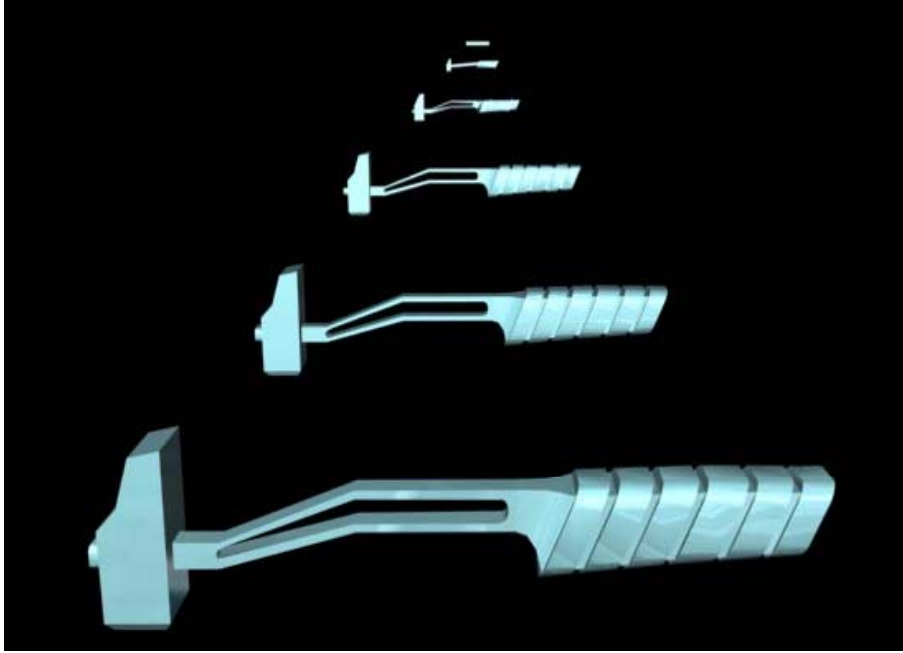


Figure 6.29: Mipmeshing [30].

There are six levels of detail that can be used for six distance intervals during the animation. The hammers were modeled with 1830, 1084, 798, 420, 85, and 12 triangles respectively (beginning from the closer model). It can be seen that using lots of polygons is not necessary if the distances are large because the size of the object prevents us from appreciating the complexity of the geometry.

Our methodology can be used to automatically build the different levels of detail for a 3D model. It is a matter of selecting the required amount of polygons in order to initialize a mesh and apply the 3D reconstruction method. We reduce the modeling time considerably because we can avoid manual operations.

### 6.2.2 Automatic tessellation

As a second utility, we present our method as a robust tessellation algorithm that builds surfaces from clouds of points. When modeling real life objects such as vases or dishes, it is common to use 3D scanners and magnetic sensors [1, 9]. These devices output thousands of points for the desired object avoiding costly manual processes in a very rudimentary manner. These points or “clouds of points” must be tessellated (a tessellation consists on the generation of triangles from a collection of vertices) if we want to use them inside a VR environment or in any sort of animation production.

Figure 6.30 illustrates how a generic tessellation algorithm works. From a cloud of points we need to derive triangles in order to end up a mesh that the system will render realistically. Applying realism to a scene consists on shading it consistently according to some predefined lights plus texturing it, among other stages. This task can be accomplished if using triangles but there’s no way if we are given points only. Triangles allow us to compute normal vectors that are a key value for lighting computations.

Figure 6.31 shows a real time application that animates according to the position and orientation of a magnetic sensor. In the scene, the blue box stands for the coordinate origin and reference frames for the scene and the real sensor. The pink box is the virtual



representation of the sensor that moves freely guided by the user. There is a yellow plane associated to the sensor. This plane cuts interactively a model, rendered in brown.

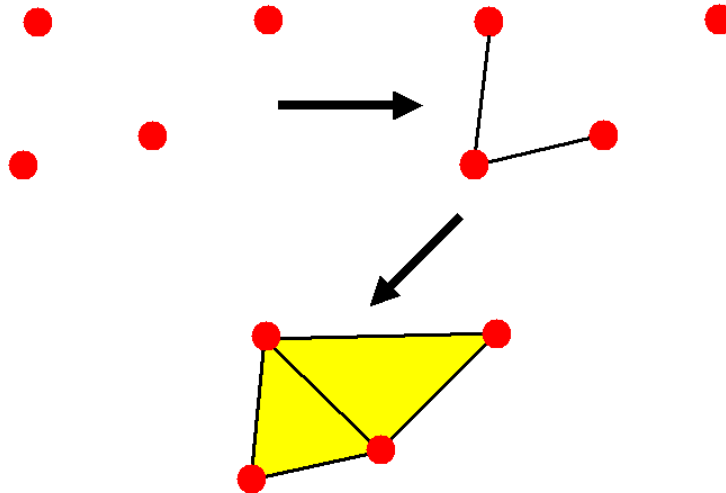


Figure 6.30: A tessellation from a cloud of points.

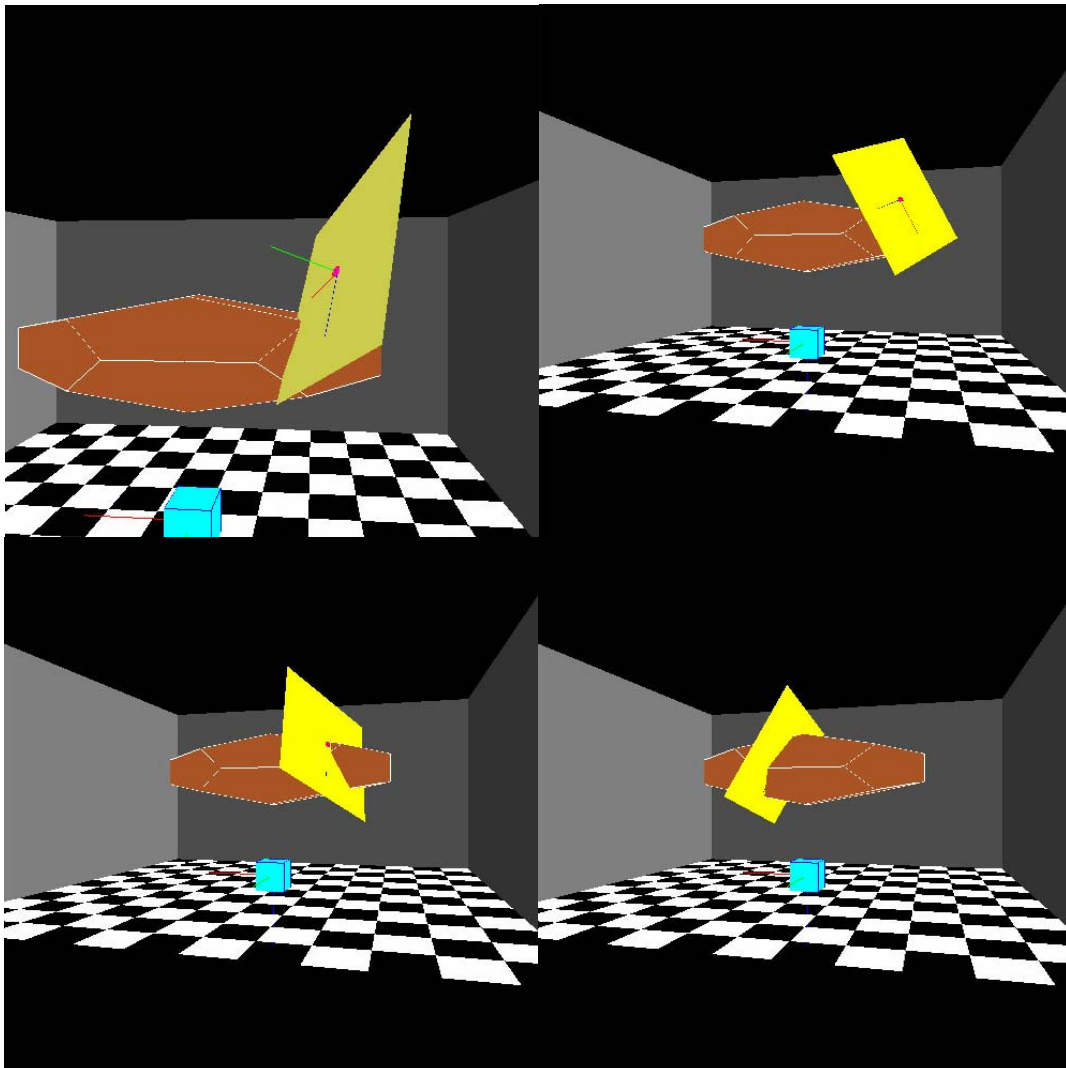


Figure 6.31: A real time application linked with a magnetic sensor.

Our algorithm automatically finds a surface associated to the cloud of points that the sensor or scanner delivers. Once the mesh has been computed, it can be exported and used inside an animation framework in order to be shaded, textured and finally rendered.

### 6.2.3 Results on automatic meshing

This section presents several results related to the explanations in sections 6.2.1 and 6.2.2. The models were selected to be very different in order to demonstrate the robustness of the method. All the tests were performed by using three different meshes according to three predefined levels of detail.

Figure 6.32 presents one of the reconstructions as it evolves in time. Note how the differences between the initial mesh (an ellipsoid) and the final model (an screwdriver) do not suppose a problem for the method.

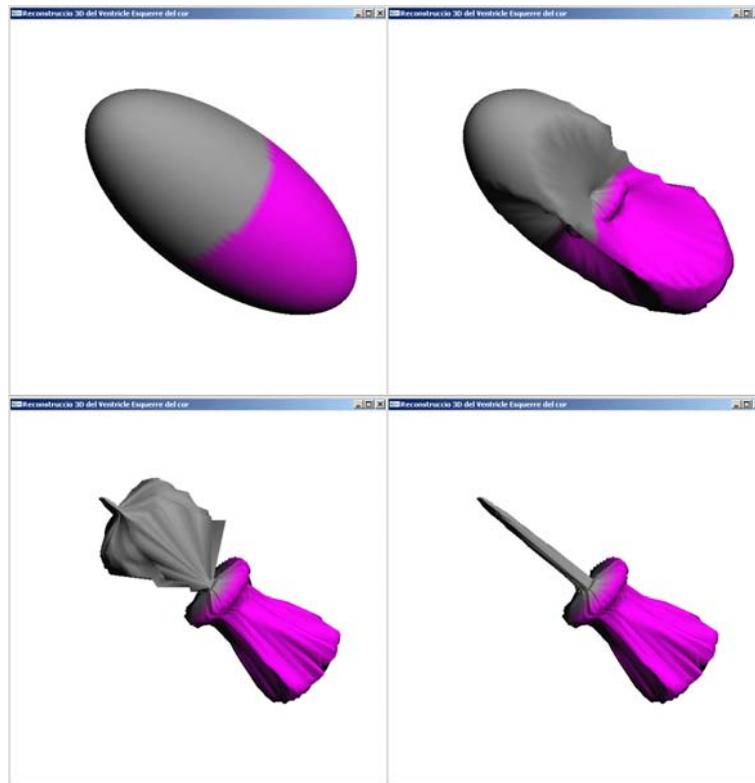


Figure 6.32: A 3D reconstruction evolution graph. Time goes from top to bottom and from left to right.

For all the tests, their associated clouds of points were voxelized to a resolution of  $62 \times 62 \times 62$  voxels. The voxels were treated as data borders for the GVF evaluation. This evaluation took 200 iterations for each test ( $\mu = 2.0$ ). The averaged CPU cost associated to the GVF evaluation was 95.14 seconds. The reconstruction was performed by using the free deformation model and the Runge-Kutta 4 solver with no smoothing algorithm. The simulation parameters were:

- Scaling factor of 0.1 for the GVF vector field.
- Damping factor of 0.0 for the GVF vector field (except for tests 5 and 6 where we applied a value of  $-0.1$  due to the tiny magnitudes of the original clouds).

- A stepsize of 0.001 seconds.

All the tests were performed on a Pentium III PC with 256 MB RAM and a 32 MB Ati Radeon accelerator card.

Table 6.18 presents several characteristics for the reconstructed models. In there, the meshes are catalogued as in table 5.2 (see chapter 5). The amount of triangles for each mesh is 320 (mesh 1, low detail), 1280 (mesh 2, medium detail) and 5120 (mesh 3, high detail). Quantities are always referred to those used in virtual reality environments that require less computations in order to achieve real time responses. Photo realistic productions related to movies for instance, require much larger quantities (millions of triangles) which are not within the scope of this project.

Test	Model	Amount of original points	Amount of original triangles	GVF time (sec.)	Time for mesh 1 (sec.)	Time for mesh 2 (sec.)	Time for mesh 3 (sec.)
1	Rabbit	67038	134074	93.84	--	78.587	300.187
2	Screwdriver	27152	54300	96.93	--	89.097	201.381
3	Vase	68097	136192	94.09	--	50.272	114.517
4	Venus	134345	268686	97.79	--	46.111	107.24
5	Moai	10002	20004	94.52	1.332	26.456	57.277
6	Squirrel	9996	19992	93.64	1.343	7.548	69.001

Table 6.18: Characteristics for the general 3D reconstructions.

Mesh 1 was only applied to tests 4 and 5. The overall computation cost for the medium and complex levels of detail were of 49.68 and 141.60 seconds respectively. Note then how the system retrieves a complex tessellation for the model in less than three minutes. The variations on the timings are related to the complexities of the models to reconstruct in terms of existing concavities. Those cases are more difficult to deal with and require higher computation costs.

If we compare the potential amount of triangles related to a cloud of points against the complex mesh that we retrieve, we can see the huge difference. For instance, with the Venus model (test 4). The original sculpture was scanned and this process retrieved 134345 points. This quantity leads to 268686 triangles by a typical tessellation algorithm. With our algorithm we build a final complex mesh consisting in 5120 triangles which stands for the 1.9 % of the total (reduction of the 98.1 %). In the case of the medium approach the final mesh stands for the 0.47 % of the overall geometry (reduction of the 99.53 %).

We provide a reduced level of detail for the model that is quite close to the original but consisting on a small percentage of its total geometry. This fact demonstrates the utility of our methodology in order to perform automatic mipmeshing for a given cloud of points.

Tables 6.19 to 6.24 show the graphical results of the analyzed tests. In tables 6.19 to 6.22 we show (from top to bottom and from left to right): a final render of the triangles retrieved by a typical tessellation algorithm, the original cloud of points, the voxelized

borders, two views of our final mesh and the mesh superimposed to the voxels for the medium (second row) and complex (third row) levels of detail.

Tables 6.23 and 6.24 show the results of tests 5 and 6. In that case the first row corresponds to the voxels and the simple mesh. The second and third rows show the medium and complex final meshes (with and without the voxels).

Note the accuracy when comparing the voxelized version of the data against the recovered mesh, especially when dealing with the complex mesh (more triangles ensure a better reconstruction that takes into account more details).

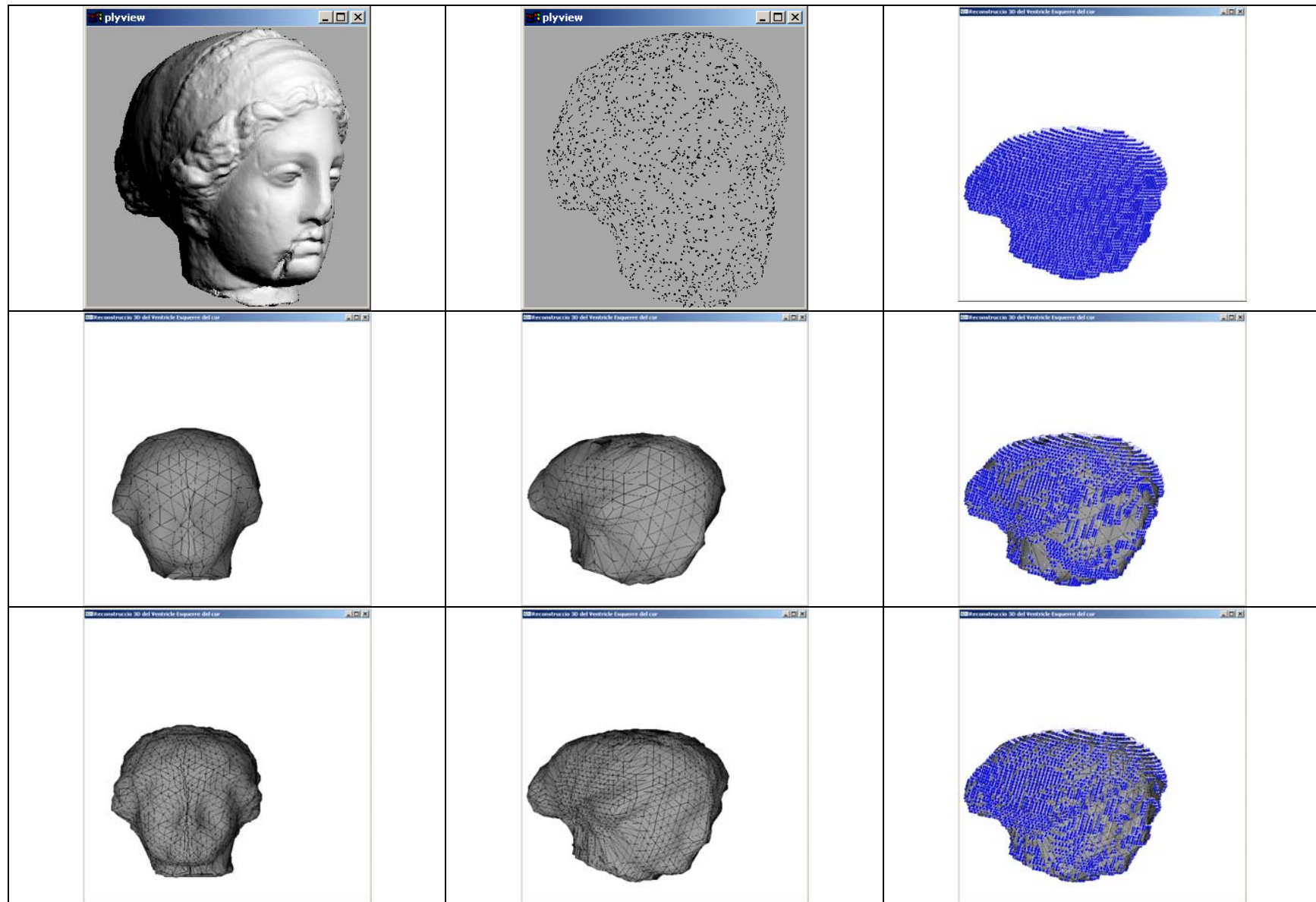


Table 6.19: General reconstructions. Test 1, the Venus model.

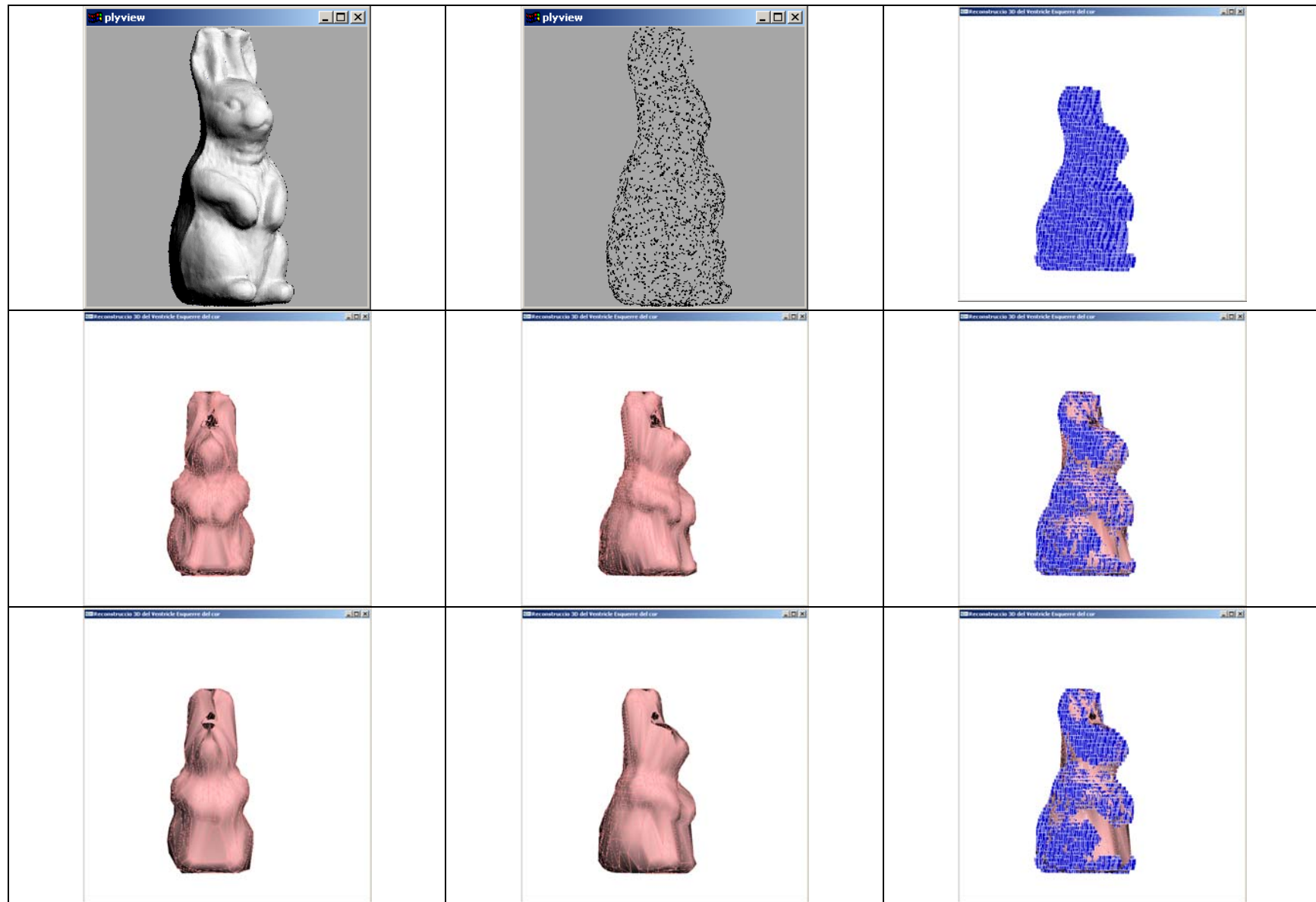


Table 6.20: General reconstructions. Test 2, the rabbit model.

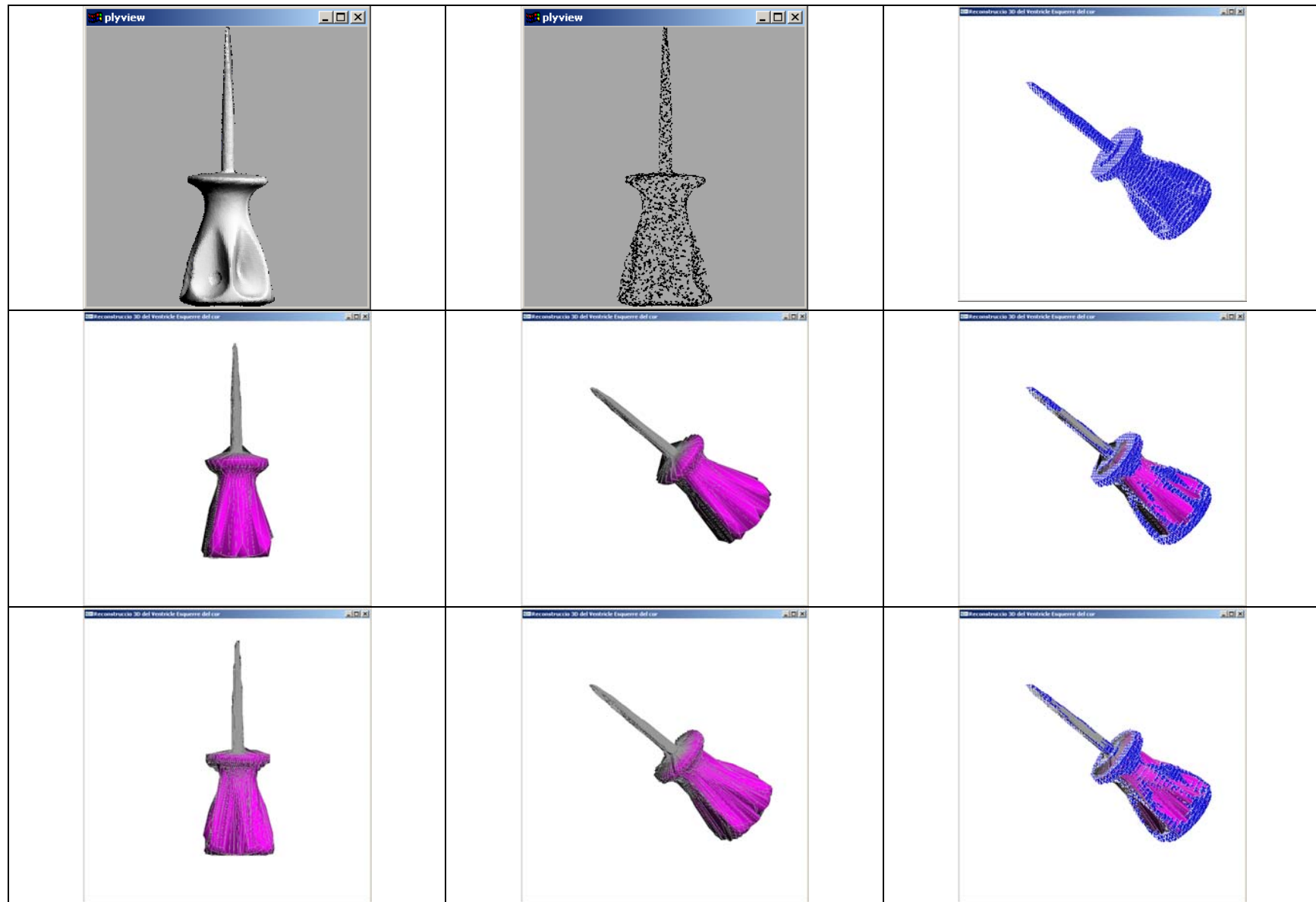


Table 6.21: General reconstructions. Test 3, the screwdriver model.

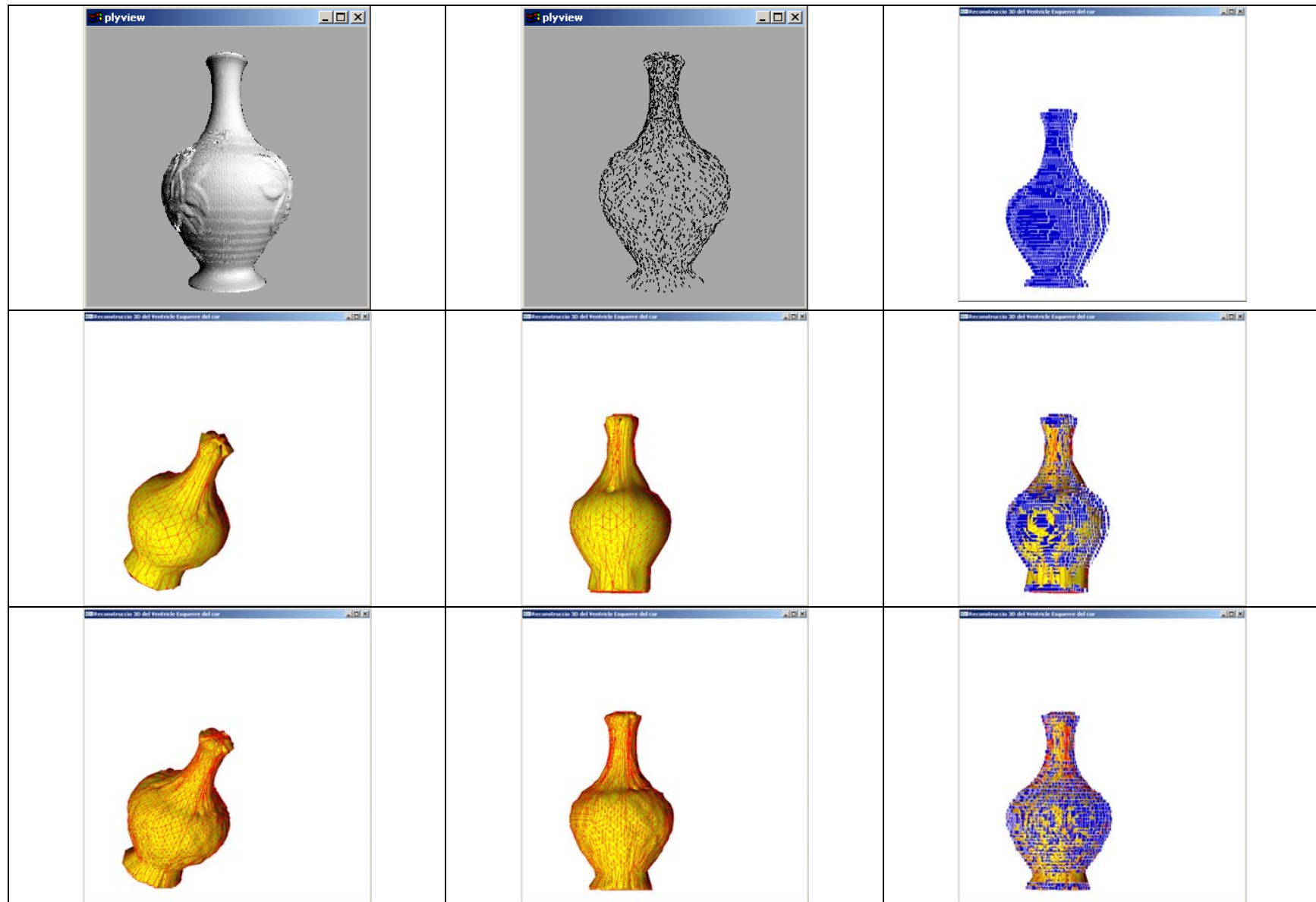


Table 6.22: General reconstructions. Test 4, the vase model.



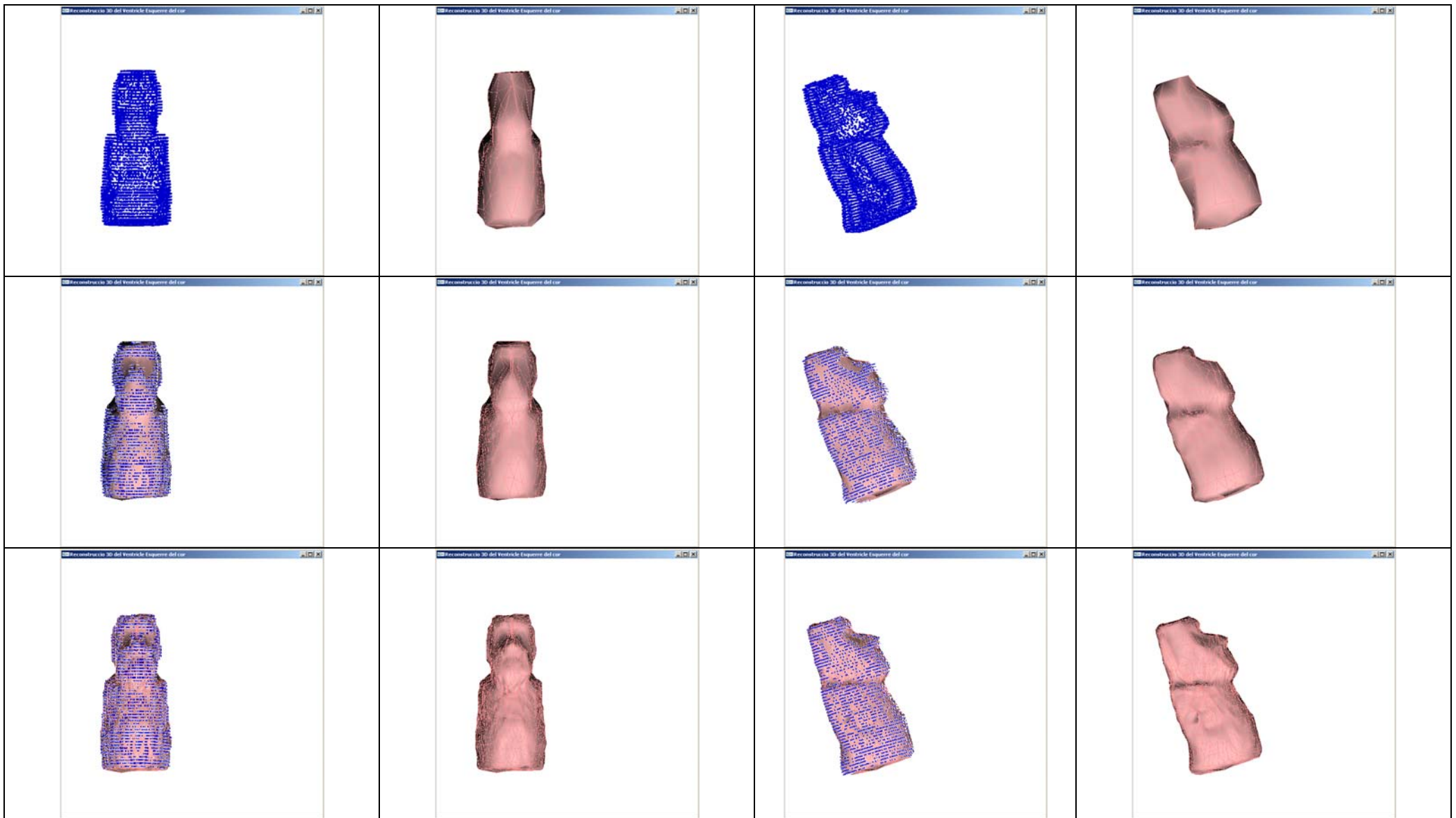


Table 6.23: General reconstructions. Test 5, the Moai model.

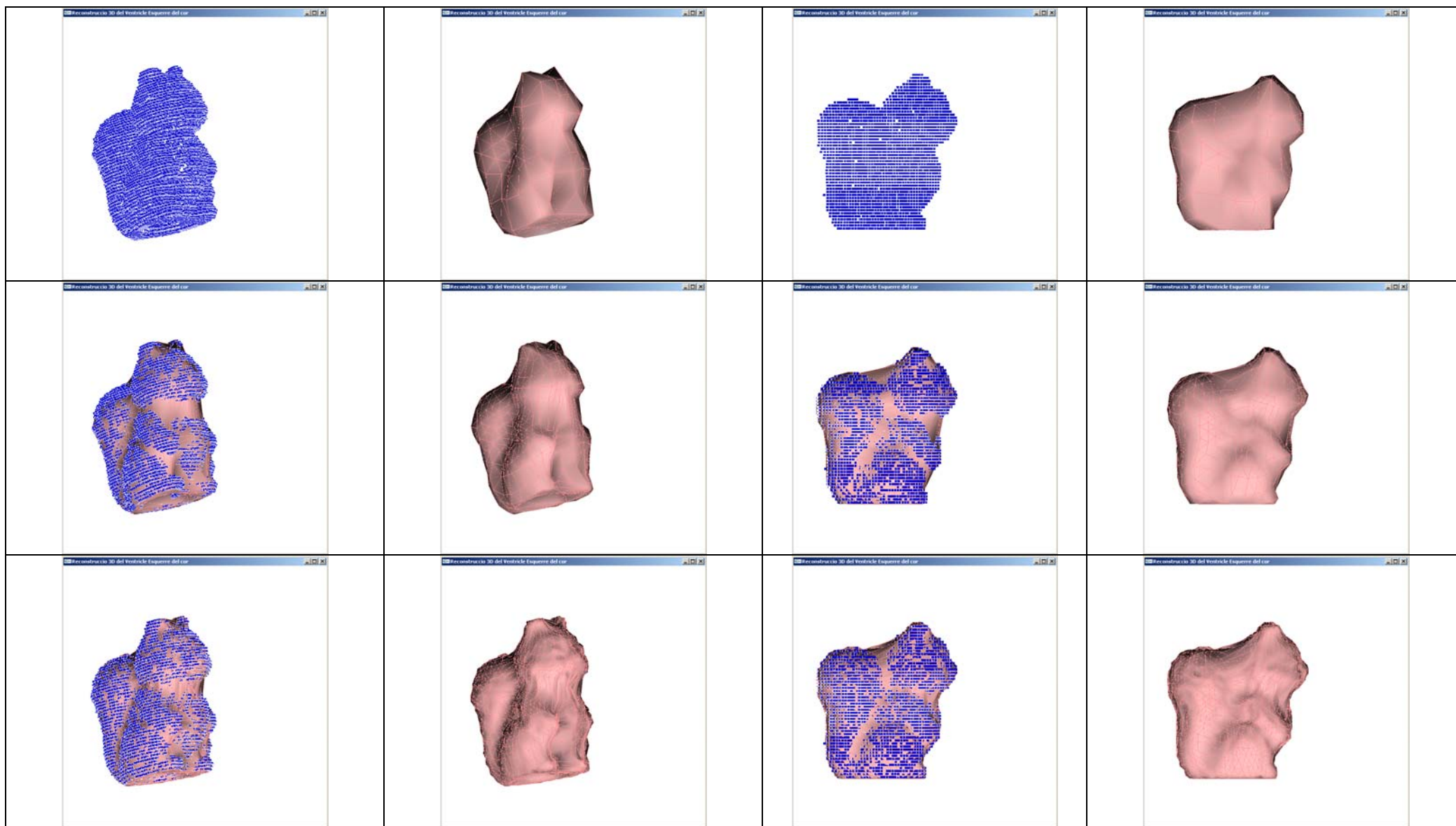


Table 6.24: General reconstructions. Test 6, the squirrel model.

### **6.3 Summary**

We have presented several results related to the three-dimensional reconstruction of the internal and external surfaces of the human's left ventricle from actual SPECT data. In that context, the reconstruction is a first process fitting in a complete VR application that will serve as an important diagnosis tool for hospitals.

We began by performing several reconstructions in a real time environment in order to achieve the desired accuracy for the GVF vector field. We experimented with simple and well-known scenarios where the final result had to fit our predefined requirements of distance to data.

After the first experiments, we reconstructed our first left ventricle, taken from an actual patient's dataset obtained from the physicians. The algorithm worked well in terms of the final mesh but it was too slow for our purposes. In order to improve our methodologies we needed a test volume with a known geometry and dimension. Moreover that, it was important that its characteristics of absorption and transmission were similar to the ones related to the ventricular tissue. We needed the Phantom volume.

We present results on the Phantom volume regarding all the topics: the GVF vector field, the stopping mechanism, measures about the distance to data, the recovered volume, the CPU cost, reconstructions with missing-data datasets, etc.

Once the technology was deeply tested, we were ready to use it with real datasets related to several patients. Experiments on healthy and pathological ventricles have been presented. The experiments show results through the entire pipeline (2D segmentation, border labeling, field evaluation, 3D reconstruction and evolution in time).

After getting the surfaces, we present the volumetric model of the left ventricle. This process is not within the scope of this project but there is active research on this topic inside our group and we give a first sight on the results, provided by other members.

We also demonstrate how our methodologies can be used as automatic contouring and tessellation algorithms, via the discreet contour and free deformation models. We apply them in the context of low polygon modeling and mipmeshing, very common techniques within the 3D community. Those technologies are used to perform generic reconstructions of different models that demonstrate the flexibility of our algorithms.

Next chapter presents the conclusions regarding all the topics explained within this document.

## 7 Conclusions

This thesis is embedded inside a global project where a computerized system is intended to improve the analysis of the cardiac data defined by specialists. The aim of this particular project is to reconstruct the three-dimensional internal and external surfaces of the human's left ventricle. Beginning with the surfaces reconstruction, the application provides volume and interactive real-time manipulation to the model. We focus on speed, precision and smoothness for the final surfaces. As long as heart diseases diagnosis requires experience, time and professional knowledge, simulation is a key-process that enlarges efficiency.

Besides the LV application, our methodology is suitable for generic reconstructions in the field of computer graphics. Our reconstructions can serve for getting 3D models at low cost, in terms of manual interaction and CPU computation overhead. As a second utility, our method is a robust tessellation algorithm that builds surfaces from clouds of points that can be retrieved from laser scanners, for instance.

### 7.1 General structure of the system

Figure 7.1 shows the first implementation of the system.

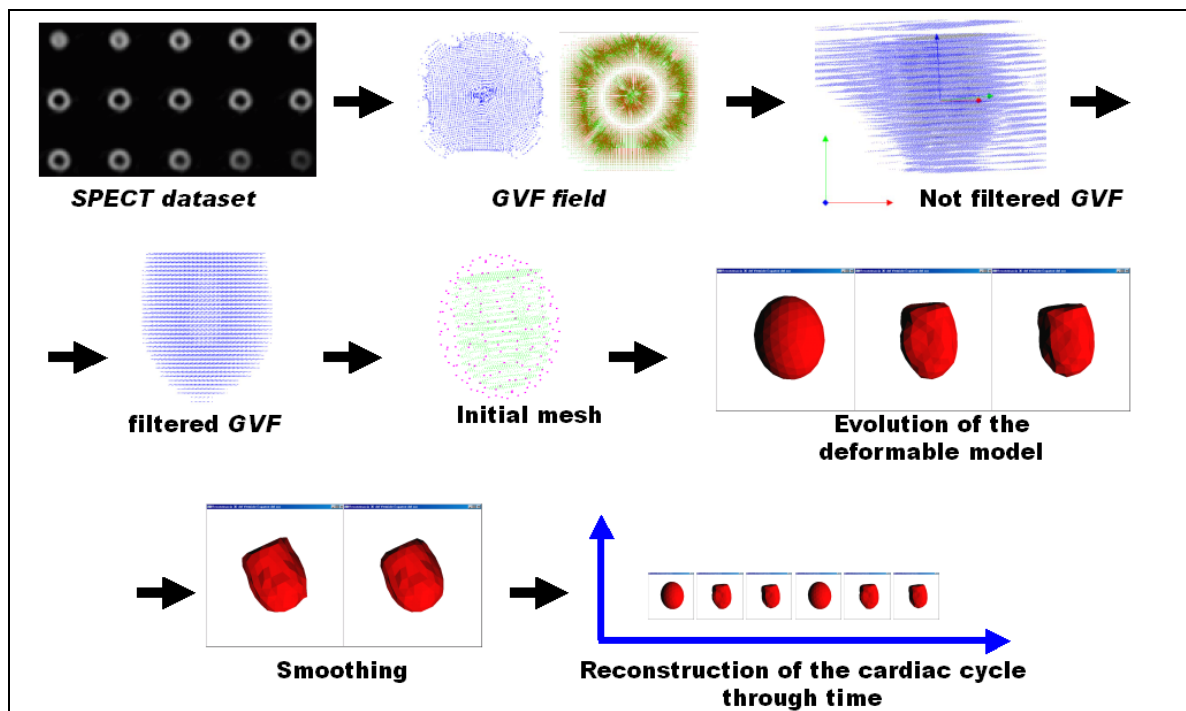


Figure 7.1: General structure of the whole system (first version).

The main features were:

- Automated evaluation of the GVF vector field.
- Three deformable models available: the plain deformation model, the mass-spring deformable model (standard and restricted) and the free deformation model.
- Two numerical schemes: explicit and implicit.
- Three implementations of the explicit solver: Euler, Midpoint and Runge-Kutta 4.

- Smoothing algorithm available in simulation time and as a post process.
- Several alternatives applied to the stopping mechanism of the mesh: dynamic damping, voxel labeling according to the sign of the GVF field and voxel labeling according to the gradient maxima.
- Exhaustive analysis of the Phantom volume. First tests with real datasets.
- Experiments on final distances to data, quality of the tessellation and final recovered volumes.
- CPU cost measurements.

Nevertheless by that moment there were several topics to achieve that drove us to a redefinition of the structure (see figure 7.2).

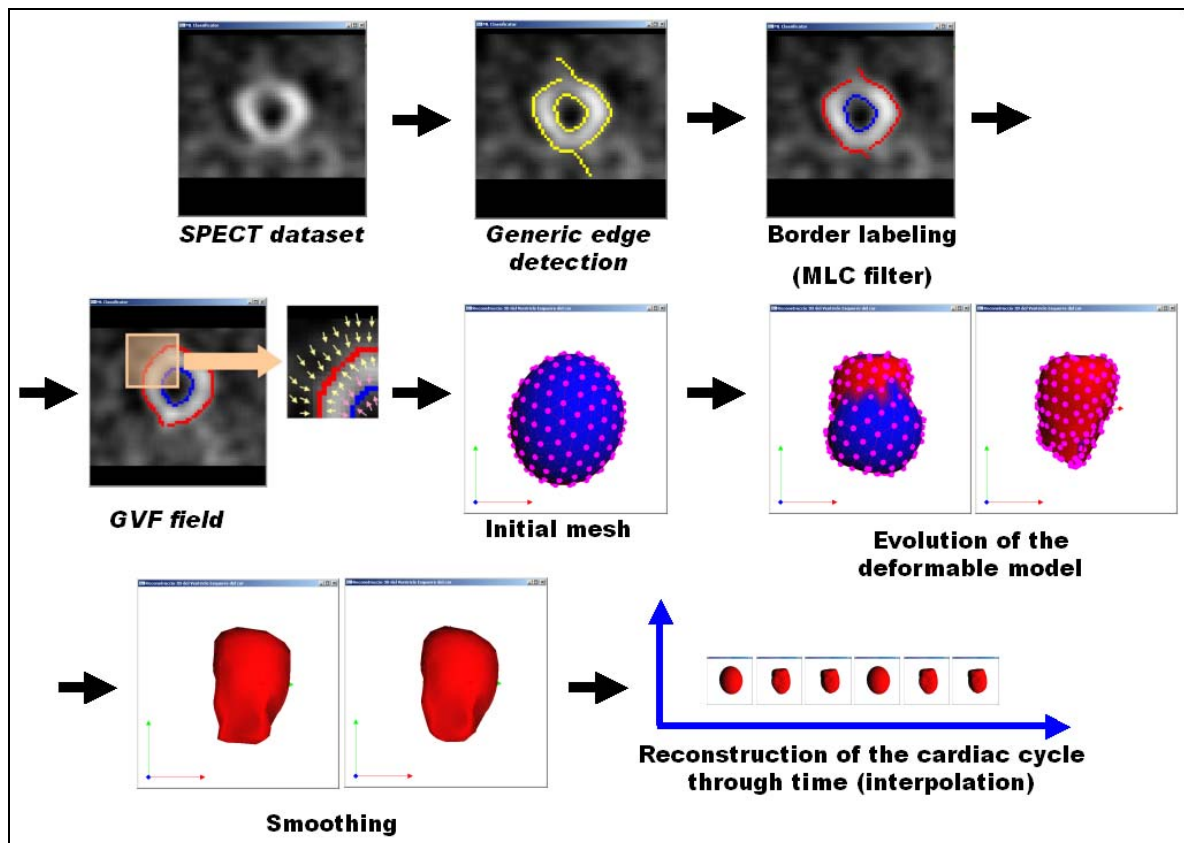


Figure 7.2: General structure of the whole system (second version).

The topics were:

- Solving the problems due to the lack of data when ischemia is present.
- Reconstructing the whole cardiac cycle.
- Implementing the transition between different keyframes of the cardiac cycle.
- Creating the final animation, from systole to diastole.
- Demonstrating the “kindness” of the system if applied to other datasets such as those obtained from scattered data (from laser scanners or VR sensors).
- Presenting the complete algorithm as a new alternative in the fields of low polygon tessellation applied to mipmeshing, for instance.

Figure 7.3 shows the structure of the system if applied to the generic reconstruction of a scanned real object (a cloud of points).



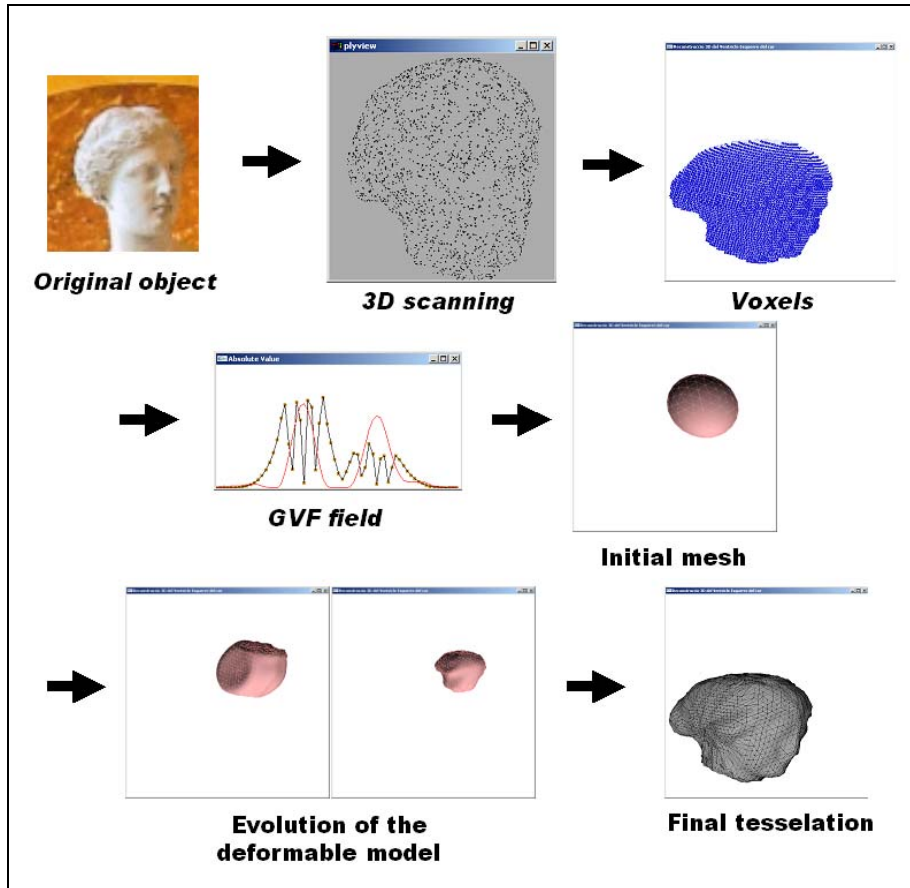


Figure 7.3: General structure of the whole system applied to generic reconstructions (second version).

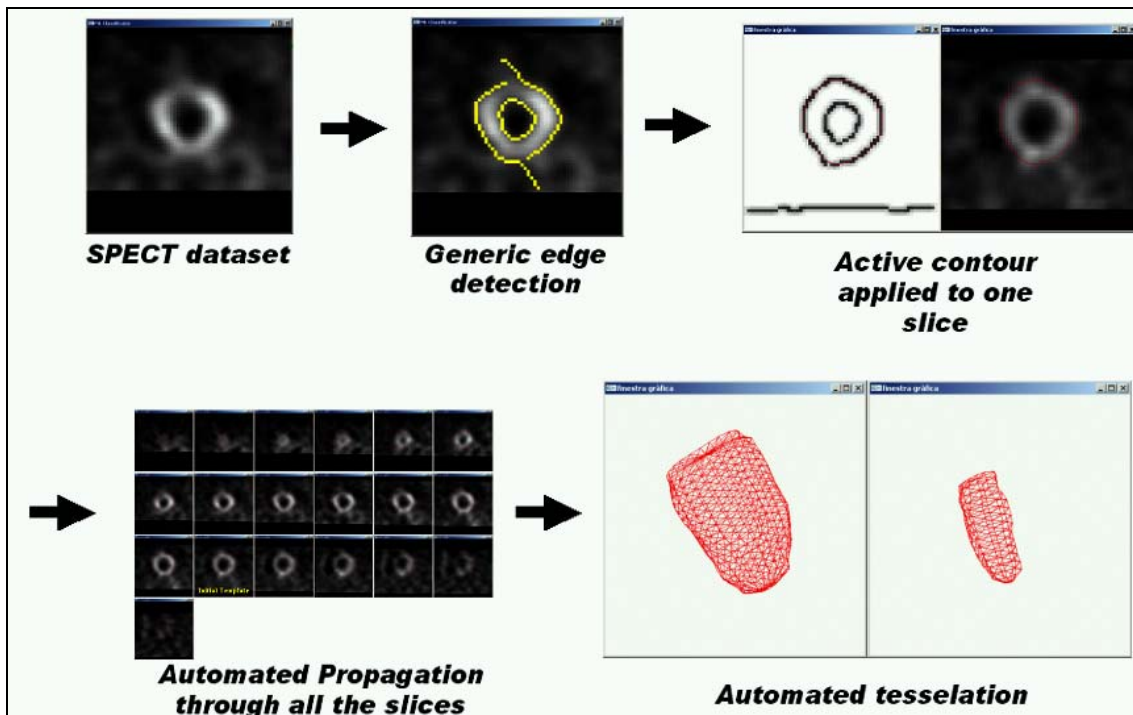


Figure 7.4: Automatic contouring and tessellation of the 3D shape with the discrete contour deformation model.

We have also extended the discreet contour deformation model (see sections 3.2.4, 3.3.3 and 3.4.1) into a 3D framework. In fact, it is possible to use this method as a tessellator to derive 3D surfaces of the left ventricle walls. The structure of this parallel system is depicted in figure 7.4.

Next sections present the conclusions regarding all the topics analyzed in the chapters of this document.

## 7.2 2D Processing

3D segmentation involves the evaluation of an external force, a vector field that pushes the meshes to the data boundaries. Prior to that, 2D processing is needed to mark the internal and external borders of the left ventricle. The borders are contained in slices that will be piled up in order to find the vector field. We want to provide the diagnosis tool with an automatic border detector that only requires some parameterization and less manual adjustments.

Firstly, we have presented a comparison between several generic edge detectors, used as the first step of the process and applied to cardiac images. The *Robert* operator, the *Canny* edge detector, the *Compass* operator, *Sobel* masks and *Anisotropic Contour Completion* have been tested. In fact, we focused deeply in *Canny* and *Compass* operators (see figures 2.9 to 2.12) and we concluded that both can be useful with a suitable tuning of the input parameters. Those serve as a first segmentation tool that retrieves contours without any knowledge of the shape that we are willing to recover. Anyway, we take advantage of this knowledge later with our self-made filters.

Attempts were made on doubling the resolution of the images although we had to reject them because of their smoothing effect. This effect neglects high gradient areas that we are interested in.

We have automated the scheme designed for labeling the data borders. Five specific techniques (section 2.3.2 and so on) have been developed: the vector sign method, the gradient method, the case-based method, the radial-circumferences method and the MLC method. One might think that a vector field can be evaluated directly from the images, with no needed labeling. This is true but not always optimum because depending on the resolution of the input imagery and the ROI (region of interest) dimensions, the derived vector field is basically poor detailed (see figure 2.13).

The *vector sign method* requires applying first the GVF (*Gradient Vector Flow*) vector field to the input 2D images. It tracks the sign of the vector field in 2D, marking the changes that should be associated to the borders. Although the process works quite well for some cases, it can be poorly effective if the ROI does not have a minimal dimension. Moreover that, the borders can be labeled in some cases but there's no way to distinguish if they are internal or external. We do need to reconstruct both surfaces separately and we have to separate borders in two categories.

The *gradient method* is based in the fact that both sides of a border are high gradient areas where the magnitudes tend to be at their maximum values. However, this method presents the same problems stated in the vector sign method.

The *case-based method* is a first attempt on labeling the borders as internal or external, according to a previous case-based classification. If we do so, we can force the meshes to stop at the outside or inside borders although the tiny dimensions of the ROI do not cooperate. From a first GVF evaluation, we mark the borders. Once this is done, we evaluate a second vector field used in the 3D reconstruction and evaluated on the borders, not over the property itself. This method is independent from the dimensions of the ROI but it is hardly to achieve a classification algorithm that takes into consideration all the possibilities when ischemia and pathological datasets are present.

The *radial-circumferences method* assumes that the internal and external borders can be considered basically as circular contours. The algorithm also starts from the fact that the physician has defined a manual coarse circle, like a first noise filtering tool. After that, a generic edge detector is applied. As a matter of fact, this approach relies too much in the characteristic of symmetry for those images what causes it to fail in some pathological cases. Moreover, noise can fool the algorithm.

The *MLC method (Maximum Likelihood Classifier)* groups pixels to one of the classes, internal or external borders, on the basis of some probability. We need to define some decision rules in order to classify them accordingly. As a precondition, our implementation assumes that there is a previous process that finds *the smallest circle* that can be used as the first filtering tool for all the slices and *the division slice* that marks the beginning of the left ventricle's apex. Both parameters can be obtained automatically within our application, which is a major contribution in terms of usability and system management for physicians.

We also exploit the concept of *vertical coherence* between slices in order to refine the division slice finding and the bounding box filtering that have been designed as a final stage that removes noise. Thus, we rely on 3D information rather than deciding in a 2D basis only.

This algorithm is definitely characterized by its robustness because it performs all the calculations automatically, with less need of symmetry assumptions. All cases can be treated with a minimal amount of parameters to be tuned.

### 7.3 3D Deformation models

Deformation models are used to recover the geometry that best fits them, among other utilities. The reconstruction of the left ventricle fits inside this category because it is necessary to use a complex model in order to recover all of its detail and shape. Simple primitives or rigid geometries cannot model those objects correctly.

We use particle systems in a Newtonian evolution scheme as a 3D segmentation tool (see section 3.1). In that context, we have experimented with several methodologies available in terms of *internal forces* (section 3.2) designed for 2D contours and 3D surfaces. We have successfully implemented *stretching* (controls the magnitudes associated to the sides of a given triangle), *shear* (acts on the inner triangle's angle), *bending* (defined between pairs of adjacent triangles) and *local curvature control* (minimization of the local curvature).



In terms of *external forces* (section 3.3), we present a comparison between several paradigms that serves us to justify the reason to use the GVF approach. We analyzed classic *snakes*, *active nets*, *topologic active nets*, *radial energy based potentials* and the *gradient vector flow* vector field. In fact, chapter 1 provides with a deep explanation on these methodologies among many others.

Snakes have three main disadvantages. First of all, the contour has to be initialized very close to the data to recover. Secondly, it has difficulties when trying to follow image maps that contain boundary concavities, like in the left ventricle case. The third problem is related to our final goal, the recovering of the inner and outer surfaces of the left ventricle. Ischemic areas, which mean in absence of blood irrigation, do not appear in the images. This effect provokes the apparition of holes that are an added problem for the contour or surface

Active nets work when objects are quite simple but some problems arise when irregularities exist, like large concavities. In order to solve those complex cases, topologic active nets were defined. Those are suitable for locating more complex shapes, tracking several objects simultaneously and detecting concavities in the internal structure of the objects. Unfortunately, this strategy might fail our purposes of tracking the entire left ventricle despite it is formed from an object or from several of them. These nets will track several apparently unrelated objects separately and we need to group them. Moreover, the 3D reconstruction method depends on the calculation of an external force related to external and internal borders. This classification is not provided by this methodology.

When working with local curvature internal forces a suitable external force can be derived from a distribution of an external potential energy: the Radial based Energy potential. This distribution can be associated to several characteristics of the image like the gray level and the gradient magnitude at every pixel or voxel. This paradigm has been used for the tracking of the left ventricle, as presented in chapter 6. However there is still the need to find an external energy term that attracts the contour or the surface to the original data. That's the reason for using the vector field described in next paragraph.

The Gradient vector flow is a vector field where the external force consists on the minimization of a functional that mixes the information derived from the image-intensities gradient with a diffusion term that allows the field to be spread out. This vector field solves two key difficulties: it avoids the need for the initial model to be close to the data to recover and it performs effectively within boundary concavities.

For the left-ventricle reconstruction, the property value is selected to be the border attribute. Given a data slice, 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. As a result of the field calculation a vector field that surrounds the borders arises, making them act as attractors (see figure 3.18). We have extended this method to 3D as a clear contribution of the present document.

We have also implemented five different deformation models (section 3.4), all of them fully functional: the *discreet contour deformation model* that weights the applied forces in order to ensure a correct stabilization. In fact we have also extended it to a 3D

framework as another contribution (see chapter 6); the *plain deformation model*, a complex scheme where each of the triangles in the mesh has its own three elasticity forces; the *spring-mass deformation model*, a well-known one where the only internal force is stretch, defined between pairs of particles; the *restricted spring-mass deformation model*, which can be considered also as a contribution, where spring-forces are only allowed in the normal direction of the derived vector field and finally, the *free deformation model*, where the only existing force is the external one, derived from the dataset. In that case, there's no connectivity between particles and topology must be maintained using a smoothing algorithm that we have developed entirely (chapter 5).

For the damping factors necessary in a simulation framework, we have also introduced the concept of damping maps (figure 3.23) that link a damping value with a voxel attending to its quality of data border.

## 7.4 Numerical implementation

For computing the state of the virtual scenery at every step, we need a numerical method, or solver, that evaluates several derivative values. There are several solvers available, different in terms of efficiency, robustness and speed of evaluation.

We have discussed explicit vs. implicit schemes which find a solution for an ODE (*Ordinary Differential Equation*) based problem. *Explicit schemes* (section 4.2) are characterized by equations that can be directly solved with no need from iterative processes. Those are classically simpler than implicit methods because of the clearness of their inner equations. *Euler's* method, *midpoint* method and *Runge-Kutta 4* method are some of the possibilities available. All of them are implemented within our simulator.

Euler's accuracy depends strongly on the stepsize selected. Tiny stepsizes offer good results in terms of trajectory while large stepsizes lead to wrong results. We can select a suitable stepsize, a small enough one, but then, lots of steps will be required in order to avoid error accumulation.

The midpoint method is a second-order solution method where the second derivative, evaluated at the midpoint of the interval, is used to calculate the whole step. It outperforms the precision over the Euler method although it requires two evaluations of the derivative function per step. If we add more terms to our Taylor series in order to minimize the error term, we get the Runge-Kutta formulation.

The Runge-Kutta 4 method requires four evaluations of the slope per iteration. Nevertheless the stepsize can be greater than in the midpoint method (at least twice as large) while keeping the same accuracy. It treats every step in a sequence of steps in identical way. Prior behavior of a solution is not used in its propagation. Especially the first affirmation makes it very easy to incorporate this algorithm into relatively simple "driver" schemes.

Implementing the Runge Kutta 4 method can be done by successively applying the simple Euler method, for every slope evaluation. From that knowledge we can take the simple Euler equations and solve the derivative term sequentially at different positions in order to find its coefficients.

We also provide with an explanation of adaptative stepsize (the step doubling technique), which provides with dynamic correction of the time step according to the restrictions at each instant. Changing the stepsize can exert some adaptative control over the progress of the simulation. The gains in efficiency can be important. We should select a stepsize or another depending on the present situation.

*Implicit schemes* (section 4.3) are based on algebraic formulas. Those methodologies are usually more complicated than explicit methods while ensuring a better convergence in a few iterations. We have implemented the implicit version of the Euler method, also called the *backwards Euler* method. It makes some decisions about the derivative value. In fact, it assumes some kind of average between the derivatives at the beginning and at the end of the interval. The method can be easily solved if we take into account the sparse characteristic of the involved matrices.

Explanations on the *implementations of the deformable models* are also within the scope of this document (section 4.4). The plain deformation model is implemented by an implicit scheme due to its complexity in terms of internal forces. The spring-mass, restricted spring-mass and free deformable models share the Newtonian approach with the plain deformable model while being simpler in terms of forces. Because of that, we decided to implement them by an explicit solver. Similar results in terms of robustness and accuracy can be achieved in these models by an explicit method like the Runge Kutta 4 solver.

We also offer a comparative test between all the implementations (section 4.4.3). The *first test* consists on the 3D reconstruction of the external surface of a Phantom volume (see appendix D). We observe that the stepsizes used in the implicit method can be considerable bigger than in the explicit simulations. The selected stepsize is then associated to the internal forces simulated and to the mesh resolution (the resolution can vary the internal forces magnitude). The free deformation model is able to drive a high percentage of particles to a distance less than one voxel long from the real data to recover.

The plain de formation model is far too complex for the 3D reconstructions that we are dealing with. In that sense it is not compulsory to use an implicit numerical method because the other deformation models make use of less internal forces or even none. The explicit methods give less control over smoothing parameters but are fast and efficient enough, optimizing distances and avoiding the apparition of oscillations.

After concluding that, we experimented more on the explicit methods. The *second and third tests* were performed using a spring-mass deformation model (only stretch internal force). The idea of the experiments was to find out the maximum stepsize allowed depending on the method and on the initial mesh. We also analyzed the impact of the smoothing algorithm on the durations.

The mesh resolution (see chapter 5) affects definitely the internal force. The stretch force (see section 3.2.1 of chapter 3) depends on the elongations of the springs, which are directly related to the links between vertices. More resolution means more links and that means more internal forces to evaluate at every step. Besides that, the initial

elongations get shortened as the resolution increases, and these initial elongations are the ones that the stretch force tries to maintain.

Looking at the maximum stepsizes according to the mesh resolution shows clearly that as long as the resolution increases (from simple to complex), the stepsize has to be reduced in order to avoid divergences. All the solvers show the same behavior. On the other hand, the duration of the entire simulation increases if the resolution gets larger. More complicated meshes lead to longer durations because of the larger amount of vertices and links to be processed.

One can notice that the final volumes for the recovered mesh are quite similar if we fix the mesh while varying the employed solver. A low-resolution mesh gives the most reliable volume because of the absence of degenerations. More complex meshes are too detailed for the coarse quality of the voxelized data.

Attending to the smoothing algorithm, we conclude that when it is disabled, the simulation durations tend to increase. Peaks appear because of the absence of the smoothing effect, and errors arise. There are also some differences between the methods due to the different evaluations that every solver performs at each iteration. On the contrary, if it is enabled, the durations decrease. As long as the smoothing algorithm corrects peaks that appear during the simulation, the reconstruction performs more fluently and quickly. The smoothing effect is especially present in the Euler method, which improves better in terms of duration. The Runge Kutta 4 method is refined by performing more steps per iteration, the results are better by themselves and the smoothing becomes less intensive.

The document also explains the implementation of the external force (section 4.5). In order to solve the GVF field, we define a temporal evolution that will guide the system until it reaches its stationary form. We use a finite differences scheme complemented by an explicit Euler's method in order to follow this evolution. It is especially important to follow the Courant-Friedrichs-Lewy condition (see equation 4.51) when defining the stepsize in order to avoid abnormal results and divergences of the field.

## 7.5 Model geometry

The geometry of the model must be selected carefully because it restricts the complexity of the shape that can be obtained. There are several possibilities available. In order to select one, we present a *deep study* in section 5.1.

A first classification of the models drives us to four main categories: *explicit and implicit representations*, *discreet meshes* and *particle systems*. Explicit representations are basically defined by a parameter vector that controls local and global deformation. On the other hand, implicit representations are characterized by the zeros set of a certain function. Discreet meshes are based on several vertices connected with each other while particle systems, in a geometrical point of view, define forces and energies that maintain the cohesion of the entire model.

Inside these categories we find several subcategories such as *polynomial functions*, *Superquadrics and Hiperquadrics*, *modal decomposition*, *algebraic surfaces*, *Level Sets*, *discreet contours*, *triangulations* and *Simplex meshes*, among others. Our model can be described as a discreet triangulated mesh that is treated as a particle system in

terms of evolution. A triangulated mesh can be easily handled by code and most of the accelerator cards in the market are specifically designed for it in terms of overall performance of the computation and the rendering. Moreover that, meshes are easy to map into particle systems where each of the vertices can be treated like a particle in a direct way.

It is important to note that the triangulation ensures the robustness of the mesh. Vertices are joined together via their topology relationship. If no topology or external mechanism is provided, like the smoothing mechanism, we might obtain poorly results after the reconstructions.

Once the paradigm is selected, we need to know the resolution needed in terms of number of triangles in the mesh. It is important because we must avoid degenerations while reconstructing. As a second process which is not within the scope of this project, it will be necessary a tetrahedralization module between both the internal and external surfaces (see chapter 6) in order to allow real-time interaction with the model. This process must associate internal triangles with their external neighbors and degenerations might spoil the tetrahedralization.

We have tested three different meshes attending to their differences in resolution (section 5.2). We observed that the lower the resolution, the better the reconstruction. Our datasets are really coarse and simple meshes are smooth enough for our purposes while presenting no degenerations at all. More complex meshes present degenerations in some of the final triangles due to their tiny dimensions according to the magnitude of the dataset.

We also tried to find out the dependences between the final quality of the triangles and the reconstruction method. Our conclusions state that the final quality of the triangles depends more on the initial dimensions of the mesh and not on the reconstruction scheme.

When reconstructing the internal mesh, we demonstrate the reliability of an Out-to-In (deflating the initial mesh) approach over the In-to-Out scheme (inflating it). In the In-to-Out scheme (see section 5.3), the external force can not be derived correctly there because of the lack of physical space. Besides that, the vertices in the mesh are not uniformly attracted making the final triangles to be very different. Among all of these, triangle degeneration can be best controlled when decreasing than when increasing its area.

We also offer a complete comparison between our approach and a very well-known technique, the Marching Cubes algorithm (section 5.4). Within the analysis of cardiac imagery, this technique presents several ambiguities when reconstructing an isosurface, generates an elevated amount of triangles not suitable for real-time applications, does not take into account the existence of holes in the dataset and only recovers a unique surface while we are interested in two, internal and external. In fact, it needs a certain threshold to be defined, which is a handicap within our imagery where this value might be different from one acquisition, or dataset, to another.

In section 5.5 we present one of our major contributions: the smoothing algorithm. The smoothing algorithm is geometrically based. There is no dependency on the

reconstruction method previously used. Thus, the algorithm can be always applied without any information from the previous process. Besides that, it can be applied in all contexts where a mesh needs to be smoothed like in terrain generation or in fluid and water simulation.

In that sense, we number the reasons for using it: lack of resolution in initial datasets, tiny ROI's, very coarse depths for the slices, absence of data due to failures in the capture or ischemia, etc. Due to all these factors, we need some kind of alternate control designed to avoid the “peaks” that appear. The algorithm penalizes vertices that are inside a non-uniform triangle set, presenting important creases. We solve the problem by applying a geometric constraint to the common vertex in a triangle set, so that this vertex moves to the centroid of its neighbors. The algorithm then is intended to apply small changes in position, never changing the mesh topology.

As a final contribution, we present the integration of time into our simulation system by using interpolation with the help of Keyframing. It allows us to present fluent animations of objects in movement that have been reconstructed by our algorithms.

## 7.6 Applications and results

As stated within the whole document, we are deeply focused on the reconstruction of the human's left ventricle but we also demonstrate that our technique can be applied in other fields that might require generic reconstructions. Due to the extension of the results, we summarize the conclusions regarding our contributions in the following list. In fact there are more experiments and comparisons that have been described previously within this chapter.

- Experiments on the *tuning of the  $\mu$  parameter* regarding the GVF evaluation. First experiments where committed on synthetic scenarios.
- *A first reconstruction* of the left ventricle that driven us to several conclusions:
  - It is difficult to select the amount of particles for the initial mesh.
  - It is not obvious to choose a stepsize value.
  - In the plain deformation model, the weighting constants associated to the internal forces are difficult to adjust.
  - If no stopping mechanism is supplied, there are oscillations at the borders produced by the discreet nature of the dataset.
  - Explicit methods retrieve short simulation durations while implicit schemes require longer ones.
- *A complete analysis of a test volume: the Phantom volume*, which helped us with several decisions.
  - Averaging the forces of the neighbors when affecting a particle does not provide with much better results.
  - Binaryzing data is not suitable because a threshold must be selected. It is better to work onto the property itself.
  - It seems clear that there's no need to iterate for more than 50-100 cycles when finding the GVF field.
  - Doubling the initial data doesn't help too much. In fact the results are basically equal or worse, in terms of magnitudes.

- The election of the  $\mu$  parameter is less significant than it might seem. We need the system to converge and the interval of available values is wide enough to fix this value for all the simulations.
- In fact, most of the parameters can be tuned via typical values although big distortions in the dimensions of the dataset might require some kind of adjustment.
- Oscillations exist, nearby the data borders, and then we must stop the particles by using some kind of strategy. We propose several mechanisms like a damping map or an automatic marking of the voxels belonging to the data borders in order to stop there (we selected this option).
- Regarding distance-to-data measurements, the maximum accuracy achievable is given by the dataset precision, which is of one voxel width.
- The free deformation model is the fastest and best in terms of final distances (70% of the reconstructions had a 90% of particles attached closely).
- In volume measures, all the solvers present relative differences in volume below 2% (between real and synthetic data).
- Computation cost tests retrieved that most of the time (90%) is dedicated to the GVF evaluation.
- There is a tradeoff between the final quality and the number of iterations applied.
- The most complete reconstruction of an entire cardiac cycle takes 514 seconds to be done (between 8 and 9 minutes) in a regular PC (Pentium III, 256 MB RAM, ATI Radeon 32 MB graphics card).
- When recovering known datasets, partially emptied, the best recovering gives a percentage of missing volume of 29.1% against the 32% of the real data emptied. This represents a relative percentage error of 0.09. For the other test examples we obtained relative errors of 0.45 and 0.24 respectively.
- We reconstruct *the entire cardiac cycle* from actual patient's data by using two different methodologies. In addition to that, we derive interesting medical parameters which were compared to those found by physicians using their standard 2D software.
- We present *several reconstructions regarding abnormal datasets*. In those, the real left ventricle presented pathologies such as digestive activity in the apex or occlusion defects located at several places (inferior, apical, side and vast anteroapical).
- We demonstrate that the discreet contour deformation model can be extended to a 3D framework in order to use it as *a new tessellation method*.
- We test our imagery with the *Anisotropic Contour Completion operator*, a very novel technique that takes into account the local orientation of the contours to be closed, which fits well with cardiac images of the left ventricle.
- Our methodology can be extended to other sort of problems in a generalist way. We demonstrate how our reconstructions can serve for getting 3D models at low cost, in the context of *low-polygon modeling* and *mipmeshing*, and as a second novel *automatic tessellation* algorithm. We show successful reconstructions of such different objects like a rabbit, a squirrel, a screwdriver and a Greek sculpture, among others.

## 7.7 Publications

J.Amatller, O.García, and A. Susin. "*Modelo Dinámico para la segmentación automática de imágenes 3D*". Proc. CEIG2000, June 28-30 2000 Castellón de la Plana (Spain), p. 355-370.

O.García, A. Susin, I.Navazo. "*Segmentación Automática mediante un modelo dinámico. Aplicación a la reconstrucción del ventrículo izquierdo*". Segundas jornadas de investigación en Ingeniería Biomédica, Noviembre 2-3 año 2000 Sitges (Spain).

A. Susín, O. García. "*Modelo Dinámico para la Reconstrucción del Corazón Humano*". XVII Congreso de Ecuaciones Diferenciales y Aplicaciones, VII de Matemática Aplicada. Setiembre 24-28, año 2001. Salamanca (Castilla y León).

A. Susín, O. García. "*Segmentación Automática del ventrículo izquierdo del corazón mediante un modelo dinámico*". Revista INPUT de Ingeniería-Arquitectura La Salle, número 23, Diciembre 2001, páginas 110-113.

O. García, A. Susín. "*Left ventricle's surface reconstruction and volume estimation*". Terceres jornades de recerca en enginyeria biomèdica, Vic 13 i 14 de juny de 2002, Xarxa Temàtica en Enginyeria Biomèdica, Generalitat de Catalunya.

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.

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.

O. García, A. Susín. "*3D reconstruction of the left ventricle*". Poster session. Forum ETSEIB 2003, UPC. Barcelona, Spain. March 2003.

O. García. "*An Easy-to-code Smoothing Algorithm for 3D Reconstructed surfaces*". Published in the book "Graphics Programming Methods", edited by Jeff Lander, editorial Charles River Media (July 2003). ISBN: 1-58450-299-1. Pages 139 to 146.

O. García, A. Susín. "*MLC filtering applied to the 3D reconstruction of the left ventricle*". Proc. CEIG2003, July 2-4 2003 La Coruña (Spain).

O. García, A. Susín. "*Vertical Coherence Applied to Spect Imagery in the 3D Reconstruction of the Left Ventricle*". The 30th Annual Conference of Computers in Cardiology. Thessaloniki, Greece, September 21-24, 2003.

*Co-worked in:*

A. Susín, I. Navazo, A.Vinacua, and P.Brunet. "*Dynamic Recognition and Reconstruction of the Human Heart*". Proc. 15th International Conference on Pattern Recognition, September 3-8 2000 Barcelona (Spain).



## 8 Future work

This research work was intended to be a complete solution regarding cardiac image segmentation and diagnosis. In fact we have also demonstrated that it can be generalized in order to retrieve reconstructions of other kinds of datasets. Although the job is done, we think that there are always some adjustments that might enrich the application.

Our approach has been applied to several datasets, including synthetic and actual's patient data. Plus to that, several experiments have demonstrated the validity of the method in normal and pathological cases.

Anyway, it would be desirable to apply the methodology to a higher population of cases in order to test its reliability onto more pathologies. It comes clear that it works for several abnormalities, which might be enough if we see this application like a complement for traditional diagnosis, but a wider study taking into account uncommon cases would be also beneficial.

In addition to that, there are other imaging modalities that might tolerate our processing. Other organs such as the liver or the prostate would take advantage of our tools. An additional study on the rest of imaging methods might allow us to some refinements in order to adapt our algorithms in a suitable way. Then this research would be useful for determining a better diagnosis to other diseases, related to different organs.

In fact other imaging techniques such as *Tagged Magnetic Resonance*, retrieve information regarding the tracking of some data points along time. This information can be used for determining the evolution of the surface in a 4D framework. This is not possible with *SPECT* imagery and it was not within this work, but it might be a valid area of interest and application. Then more complicated motions like the twisting movement of the left ventricle would be possible to simulate. If no correspondence between data points us supplied, like in our case, there is no way to obtain this.

The *ACC (Anisotropic Contour Completion)* paradigm [37] is a novel technique that takes into consideration the local orientation of the contours to be closed. We have shown some experiments in chapter 6, where the operator is used as a preprocess to the cardiac images. It takes advantage of their circular-based shape. As long as this technique is still under prior research, it would be interesting to follow its evolution and keep using it in order to improve our results. It seems coherent to do this since this operator adds an extra and prior knowledge of the shape. In fact we do so with our *MLC* technique as explained in chapter 2. Cardiac images are characterized by circular or elliptical shapes. We think that this fact can be fully exploited by future improvements of the *ACC* methodology.

We have shown a way to extend the *discreet contour deformation model* to a 3D framework. We think that we can go further and use it as a complement to our particle system reconstruction system, regarding the stopping mechanism. The 2D contours retrieved by this model collide with the voxels of the slice where they are lying in. This information might be stored and used in order to refine the stopping mechanism, making the particles go further within a voxel, until colliding with the contour. We do retrieve the maximum precision allowable, that of a voxel, but this modification would

ensure a more accurate position for a given particle, while maintaining precision. This would be true if the contours were accurately adjusted to the data borders.

As stated in chapter 3, the *GVF* is a valid scheme in terms of external force. Anyway we showed that its calculation is computationally expensive, at least compared to the rest of the modules in the whole process. In order to reduce this time, we propose the implementation of an implicit solver, instead of the explicit scheme presented in literature and used by our system. An implicit integrator allows higher stepsizes while maintaining or even improving the accuracy of the result. Those methods can be solved in a few iterations if implemented in a clever way.

In order to reduce reconstruction times, especially when reconstructing the entire cardiac cycle for a patient, we suggest to use the mesh reconstructed previously as the initial for the actual instant. If we assume that differences between instants are not considerable, we can do this in order to begin the simulations being closer to the final solution. Of course this assumption depends on the amount of acquisitions for a dataset. More acquisitions might lead to better reconstructions because the differences between meshes should be smaller, ideally negligible. In fact there is no need to change any of the programming if we want to do this, we only need to load the previous mesh with our actual application and use it for the reconstruction, instead of the spherical meshes that we have shown so far.

Evolution through time is based in *linear interpolation*. There are other possibilities to experiment with like cubic curves for instance. *Bezier* curves or *B-Spline* segments are some of the possibilities available. Those allow the user to control the joint points between instants, ensuring continuity and derivability, besides smoothness for instance. Depending on the curve, we can ensure  $C^0$ ,  $C^1$  or  $C^2$  continuities which stand for position, first derivative and second derivative (smoothness) completeness at the joint points. Nevertheless this improving might be coupled with the knowledge of successive positions for some data points, if we want it to be effective. This statement has been explained before in this chapter.

Our application minimizes the manual adjustments required by clinical workers. Anyway it would be desirable to detect automatically the first and last useful slices within the dataset. Instead of asking the physician, the application would detect the first and last noisy slices and reject them. This issue happens when the acquisition process, via medical hardware, tracks an area larger than the space where the ventricle lies in. We might think in a module that would limit the vertical region of interest. This can be corrected in acquisition time but it is not always possible because it is not easy at all.

## Bibliography

- [1] D. Addleman et al. "Cyberware". Copyright © 1999 Cyberware. All rights reserved.  
<http://www.cyberware.com>
- [2] E. Angel. "Interactive Computer Graphics: A Top-Down Approach with OpenGL" .  
Second Edition". 020138597X. 613 pp, Cloth AUG 06, 1999 Addison-Wesley.
- [3] J. L. A. Colom. "Model discret de contorn dinàmic". PFC DTA-PFC-861/EE E.S.  
Electrònica La Salle. 2003.
- [4] F.M. Ansia; C. Mariño, M. Penedo and A. Mosquera. "Mallas Activas Topológicas"  
en Congreso Español de Informática Gráfica; Quirós, R.; Regincós J. and Hernández, L.  
(Eds.); 45-58, Coruña, 2003.
- [5] J. Amatller, O. Garcia and A. Susín. "Modelo Dinámico para la segmentación  
automática de imágenes 3D". Ed. Joan R., Navazo I., Quirós R. CEIG 2000, X  
Congreso Español de Informática Gráfica. Univ. Jaume I, Col. Treballs d'Informàtica i  
Tecnologia n.3, pp 355-370, 2000. (ISBN 84-8021-314-0)
- [6] D. Baraff, "Analytical methods for dynamic simulation of nonpenetrating rigid  
bodies", Computer Graphics (Proc.SIGGRAPH), vol.23, No.3, pp.223-232, July, 1989.
- [7] D. Baraff and A. Witkin. "Large steps in cloth simulation". In Proceedings of  
SIGGRAPH, pages 43--54, 1998.
- [8] E. Bardinet, L. D. Cohen and N. Ayache. "Analyzing the deformation of the left  
ventricle of the heart with a parametric deformable model", INRIA Sophia Antipolis,  
France, technical report number RR-2797, February 1996.
- [9] E. Blood, J. Scully et al. "Ascension Technology Corporation". © 2003 Ascension  
Technology Corporation. All Rights Reserved. P.O. Box 527 Burlington, VT 05402,  
USA.  
<http://www.ascension-tech.com/>
- [10] C. Bonciu, R. Weber and C. Léger. "4D reconstruction of the left ventricle  
during a single heart beat from ultrasound imaging". Image and Vision Computing,  
Vol. 19 (6) (2001) pp. 401-412.
- [11] D. E. Breen, D. H. House and M. J. Wozny. "Predicting the drape of woven  
cloth using interacting particles". In SIGGRAPH 94 Conference Proceedings, Annual  
Conference Series, pages 365--372, Orlando, FL, USA, July 1994. ACM SIGGRAPH.
- [12] R. Bridson, S. Marino and R. Fedkiw. "Simulation of Clothing with Folds and  
Wrinkles", ACM SIGGRAPH/Eurographics Symposium on Computer Animation  
(SCA), edited by D. Breen and M. Lin, pp. 28-36, 2003.
- [13] M. Bro-Nielsen: "Active Nets and Cubes", IMM Tech. Rep 94-13, 1994.

- [14] J. Canny. “*A Computational Approach to Edge Detection*”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, Nov. 1986.
- [15] M. Carignan, Y. Yang, N. Magnenat-Thalmann and D. Thalmann. “*Dressing animated synthetic actors with complex deformable clothes*”. In Computer Graphics (Proc. of ACM Siggraph 92), ACM, 99-104.
- [16] K. Choi and H. Ko. “*Stable but responsive cloth*”. In ACM Transactions on Graphics (SIGGRAPH 2002), volume 21, 2002.
- [17] L.D. Cohen and I. Cohen. “*Finite element methods for active contour models and balloons for 2D and 3D images*”. IEEE Trans. Pattern Analysis. Machine Intelligence, 15(11), 1131-1147. 1993.
- [18] L. Comas, P. Berthout, R. Sabbat, O. Blagosklonov, J. Verdenet, M. Baud and J.C. Cardot. “*Automatic Contours Detection in Myocardial Gated Single-Photon Emission Tomography*”. IEEE Computers in Cardiology 2003. September 21-24, 2003. Thessaloniki, Greece.
- [19] J. M. Cordero and J. Matellanes. “*Animación realista de tejidos usando un modelo basado en masas y resortes*”. XIII Congreso español de informática gráfica CEIG 2003, A Coruña 2,3 y 4 de Julio de 2003. p.303-316.
- [20] S. Cotin, H. Delingette, J-M. Clément, V. Tasseti, J. Marescaux, and N. Ayache. “*Geometric and Physical Representations for a Simulator of Hepatic Surgery*”. In Medicine Meets Virtual Reality IV, Interactive Technology and the New Paradigm for Healthcare, pages 139-151, January 1996. IOS Press.
- [21] S. Cotin, H. Delingette and N. Ayache, “*Real Time Elastic Deformations of Soft Tissues for Surgery Simulation*”, IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 1, January-March, 1999, pp. 62-73.
- [22] M. Daher and R. Little. “*LeadTools*”. Copyright © 2002 LEAD Technologies, Inc.  
<http://www.leadtools.com/SDK/Medical/DICOM/dicomovr.htm>
- [23] H. Delingette. “*General object reconstruction based on simplex meshes*”. Research Report 3111, INRIA, Sophia Antipolis, France, February 1997.
- [24] L. Ding and A. Goshtasby. “*On the Canny edge detector*”. Computer Science and Engineering Department. Wright State University.
- [25] “*Disney on-line*”. © 2003 Disney Enterprises, Inc. All rights reserved.  
<http://disney.go.com>
- [26] S. B. Dove. “*Dental Diagnostic Science*”. The University of Texas Health Science Center at San Antonio. March 2002. Copyright UTHSCSA 1996-2002 All rights reserved.  
<http://ddsdx.uthscsa.edu/DICOM.html>

- [27] B. Eberhardt, A. Weber and W. Strasser. “*A fast, flexible, particle-system model for cloth draping*”. IEEE Computer Graphics and Applications 16, 5 Sept. 1996. 52-59.
- [28] H. EI-Messiry, H.A. Kestler, O. Grebe, H. Neumann. “*Segmenting the Endocardial Border of the Left Ventricle in Cardiac Magnetic Resonance Images*”. IEEE Computers in Cardiology 2003. September 21-24, 2003. Thessaloniki, Greece.
- [29] N. Ezquerro, I. Navazo, T.I. Morris and E. Monclus. “*Graphics, Vision, and Visualization in Medical Imaging: A State of the Art Report*”, in Eurographics’99, pp. 21-80, 1999.
- [30] O. Fernandez Barracel. “*Sistema extensible de representación de animaciones 3D en tiempo real*”. E. T. Informàtica de Sistemes. TFC DTA-TFC-850/IS (2001).
- [31] B. Fisher, S. Perkins, A. Walker and E. Wolfart. “*Sobel Edge Detector*”. Department of Artificial Intelligence, University of Edinburgh UK. ©1994.  
<http://www.cee.hw.ac.uk/hipr/html/sobel.html>
- [32] R. N. Fogoros, “*Heart Disease / Cardiology*”, ABOUT, The Human Internet 2001.  
<http://heartdisease.about.com/library/glossary/blglejectionfraction.htm>
- [33] J. Garcia and P. Radeva. “*Contrast Echography Segmentation and Tracking by a Trained Deformable Model*”. IEEE Computers in Cardiology 2003. September 21-24, 2003. Thessaloniki, Greece.
- [34] O. García. “*Dynamics simulation of deformable objects*”. PFC S0257 E.S. Electrònica La Salle. 09/11/1998.
- [35] O. García, A. Susín. “*MLC filtering applied to the 3D reconstruction of the left ventricle*”. Proc. CEIG2003, July 2-4 2003 La Coruña (Spain).
- [36] S. Gibson, J. Samosky, A. Mor, C. Fyock, E. Grimson, T. Kanade, R. Kikinis, H. Lauer, N. McKenzie, S. Nakajima, H. Ohkami, R. Osborne, A. Sawada. “*Simulating Arthroscopic Knee Surgery Using Volumetric Object Representations, Real-Time Volume Rendering and Haptic Feedback*”. (Technical Report TR96-19), (1996).
- [37] D. Gil, P. Radeva and F. Vilariño. “*Anisotropic Contour Completion*”. Proceedings of the International Conference on Image Processing 2003. September 14-17, Barcelona, Spain.
- [38] “*GL and GLU graphical libraries*”. OpenGL The Industry’s Foundation for High Performance Graphics. 1992.  
<http://www.opengl.org/>
- [39] G. Hamarneh and T. McInerney. “*Physics-Based Shape Deformation for Medical Image Analysis*”, Proc. of IS&T/SPIE's 15th Annual Symposium on Electronic Imaging, Science and Technology, Santa Clara, CA, January, 2003.

- [40] G. Hamarneh, T. McInerney, and D. Terzopoulos, “*Deformable Organisms for Automatic Medical Image Analysis*”, to appear in Medical Image Computing and Computer-Assisted Intervention, MICCAI 2000, Utrecht, The Netherlands, 14-17 October 2001.
- [41] X. Hang, N.L. Greenberg and J.D. Thomas. “*A Geometric Deformable Model For Echocardiographic Image Segmentation*”. IEEE Computers in Cardiology 2002. September 22-25, 2002. Memphis, Tennessee, USA.
- [42] G. T. Herman and H. K. Liu. “*Three-dimensional display of human organs from computer tomograms*”. Computer Graphics and Image Processing, 9(1):1 - 21, 1979.
- [43] A. Huerta and S. Fernandez. “Enrichment and coupling of the finite element and meshless methods”. International Journal for Numerical Methods in Engineering , 48:1615 –1636,2000.
- [44] A. Joukhadar and C. Laugier. “*Dynamic Simulation: Model, Basic algorithms, and Optimization*”.Conference Paper, Laumond, J.-P. and Overmars, M. (Eds), In Algorithms For Robotic Motion and Manipulation, pp. 419-434, A.K. Peters Publisher, (1997).
- [45] A. Joukhadar, A. Deguet and C. Laugier. “*Towards Realistic Dynamic Simulation: Deformations and Collisions Models*”. Conference Paper, In Proc. of the Workshop on Dynamic Simulation: Methods and Applications, pp. 21-31, Grenoble (FR) (September 1997) Workshop held in association with the IEEE-RSJ Int.\ Conf. on Intelligent Robots and Systems.
- [46] M. Kass, A. Witkin and D. Terzopoulos, “*Snake: Active contour model*”, Int. J. Computer Vision, Vol. 1, pp. 321-331, 1987.
- [47] M. Kass, A. Witkin, and D. Terzopoulos, “*Constraints on deformable models: Recovering 3D shape and nongrid motion*”, Artificial Intelligence, 36 (1988), pp. 91--123.
- [48] R. M. Koch, M. H. Gross, F. R. Carls, D. F. von Bren, G. Frankhauser, and Y. I. H. Parish, “*Simulating Facial Surgery Using Finite Element Models*”, Proceeding of the 23 Annual Conference on Computer Graphics, New Orleans, LA USA, August 4-9, 1996, pp 421-428.
- [49] J.L. Laading, C. McCulloch, V.E. Johnson, D. Gilland and R.J. Jaszczak. “*A hierarchical feature based deformation model applied to 4D cardiac SPECT data*”. Lecture Notes in Computer Science: Information Processing in Medical Imaging. 266-279 (Springer-Verlag, Berlin; 1999).
- [50] J. Lander et al. “*Graphics Programming Methods*”. Charles River Media; Book and CD-ROM edition (July 2003). ISBN: 1584502991 (pages 139-146).
- [51] K. Lango. “*Organized Keyframing and How it Works*”. 2001-2002.  
[http://www.keithlango.com/popThru/popThru.html#organized\\_keys](http://www.keithlango.com/popThru/popThru.html#organized_keys)

- [52] John Lasseter, "*Principles of Traditional Animation Applied to 3D Computer Animation*", Computer Graphics, pp. 35-44, 21:4, July 1987 (SIGGRAPH 87).
- [53] L. Legrand, C. Bordier, A. Lalande, P. Walker, F. Brunotte. "*Magnetic Resonance Image Segmentation and Heart Motion Tracking With An Active Mesh Based System*". IEEE Computers in Cardiology 2002. September 22-25, 2002. Memphis, Tennessee, USA.
- [54] S. Lobregt and M. A. Viergever. "*A discrete dynamic contour model*". IEEE Trans. on Medical Imaging, 14(1):12-24, March 1995.
- [55] W. Lorensen and H. Cline, "*Marching Cubes: A High Resolution 3D Surface Construction Algorithm*", Proc. SIGGRAPH'87, pag. 163-169, 1987.
- [56] F. Mannting, P.K. Chandak, Y. V. Zabrodina and B. L. Holman, "Atlas of Myocardial Perfusion SPECT", Brigham and Women's Hospital, Harvard Medical School, Boston MA. 1995-1999.  
<http://brighamrad.harvard.edu/education/online/Cardiac/Cardiac.html>
- [57] C. C. McCulloch, J. K. Laading, and V. E. Johnson. "*Image feature identification via bayesian hierarchical models*". In The American Statistical Association Proceedings of the Section on Bayesian Statistical Science, Anaheim, California, 1997.
- [58] T. McInerney, D. Terzopoulos. "Topologically adaptable snakes", Proc. Fifth International Conf. on Computer Vision (ICCV'95), Cambridge, MA, June, 1995, 840-845.
- [59] T. McInerney and D. Terzopoulos. "*A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4D image analysis*". Comp. Med. Imaging and Graphics, 19:69-83, 1995.
- [60] T. McInerney, D. Terzopoulos. "*Deformable models in medical images analysis: a survey*". Medical Image Analysis 1(2) 91-108 (1996).
- [61] T. McInerney, D. Terzopoulos. "*Topology Adaptive Deformable Surfaces for Medical Image Volume Segmentation*". IEEE Transactions on Medical Imaging, Vol. 18(9). (1999) 100-111.
- [62] S. McMillan. "*Overview of Ordinary Differential Equations*". B.A. (Mathematics), 1977, Cambridge University. Ph.D. (Astronomy), 1983, Harvard University.  
[http://einstein.drexel.edu/courses/Comp\\_Phys/ODE/](http://einstein.drexel.edu/courses/Comp_Phys/ODE/)
- [63] W. McNeely, K. Puterbaugh, and J. Troy. "*Six degree-of-freedom haptic rendering using voxel sampling*". Proc. of ACM SIGGRAPH, pages 401--408, 1999.
- [64] M. G. Mero, A. Susín. "*Deformable 3D Objects for a VR medical application*". 3es. Jornades de Recerca en Enginyeria Biomèdica. Lacroix D., Ginebra M.P. ed pp.264--269 (2002) Xarxa Temàtica en Enginyeria Biomèdica (ISBN 84-699-8705-4). Vic (Spain).

- [65] D. Metaxas, "*Physics-based Deformable Models: Applications to Computer Vision, Graphics and Medical Imaging*", Kluwer Academic Press, 1997.
- [66] J.V. Miller, D.E. Breen and M.J. Wozny. "Extracting geometric models through constraint minimization". Rensselaer Polytechnic Institute, Tech. Rep. No. 90024, 1990.
- [67] J.V. Miller. "On GDMs: Geometrically deformed models for the extraction of closed shapes from volume data". Rensselaer Polytechnic Institute, Tech. Rep. No. 90051, 1990.
- [68] J.V. Miller, D.E. Breen, W.E. Lorensen, R.M. O'Bara and M.J. Wozny. "Geometrically deformed models: A method to extract closed geometric models from volume data". *Comput. Graphics*, vol. 25, no. 4, pp. 217-226. 1991.
- [69] J. Montagnat and H. Delingette, "*Volumetric medical image segmentation using shape constrained deformable models*", Proceedings of the Second International Conference on Computer Vision, Virtual Reality and Robotics in Medicine, 1997, pp. 13-22.
- [70] J. Montagnat, H. Delingette. "Globally constrained deformable models for 3D object reconstruction". *Signal Processing*, 71(2) (1998).
- [71] J. Montagnat, H. Delingette, N. Scapel, and N. Ayache. "Representation, Shape, Topology and Evolution of Deformable Surfaces". Application to 3D Medical Image Segmentation. Technical report RR-3954, INRIA, 2000.
- [72] J. Montagnat and H. Delingette. "Space and time shape constrained deformable surfaces for 4D medical image segmentation". In *Medical Image Computing and Computer-Assisted Intervention (MICCAI'00)*, 2000.
- [73] C. Montani, R. Scateni, R. Scopigno, "Discretized Marching Cubes", In *Proceedings of Visualization '94*, IEEE Computer Society Press, pp. 281-287, 1994.
- [74] M-A. Morales, V. Positano, O. Rodriguez, M. Morelos, M. Passera, M. Lombardi, D. Rovai. "Tracking The Left Ventricle In Contrast Enhanced Echocardiography By Anisotropic Filtering And Active Contours Algorithm". *IEEE Computers in Cardiology 2002*. September 22-25, 2002. Memphis, Tennessee, USA.
- [75] R. Mullick and N.F. Ezquerria. "Automatic determination of LV orientation from SPECT data". *IEEE Trans. Med. Imag.* 14, 1 (March 1995) 88-99.
- [76] J. Nielsen. "useit.com: Jakob Nielsen's Website". Nielsen Norman Group, 48921 Warm Springs Blvd., Fremont, CA 94539, USA.  
<http://www.useit.com/>
- [77] S. Ordás, L. Boisrobert and A.F. Frangi. "Cardiac MRI Segmentation Using 2D-ASMs with Optimal Invariant Features". *IEEE Computers in Cardiology 2003*. September 21-24, 2003. Thessaloniki, Greece.



- [78] N. Paragios. "A Variational Approach for the Segmentation of the Left Ventricle in MR Cardiac Images". IEEE Workshop on Variational and Level Set Methods (VLSM'01), July 13 - 13, 2001 Vancouver, Canada. (Siemens Corporate Research)
- [79] J. Park, D. Metaxas, A. A. Young, and L. Axel, "Deformable Models with Parameter Functions for Cardiac Motion Analysis from Tagged MRI Data", IEEE Trans. on Medical Imaging. Vol. 15, No. 3, pp. 278--289, 1996.
- [80] G. Picinbono, J.C. Lombardo, H. Delingette, and N. Ayache. "Improving Realism of a Surgery Simulator: Linear Anisotropic Elasticity, Complex Interactions and Force Extrapolation". The Journal of Visualization and Computer Animation, 2001. accepted (also available as INRIA research report RR-4018).
- [81] X. Provot. "Deformation constraints in a mass-spring model to describe rigid cloth behavior". In Graphics Interface'95. 147-154.
- [82] D. Quackenbush, P. Ratiu, and J. Kerr, "Segmentation of the Visible Human Data Set", The Visible Human Project Conference, pag. 69-70, 1996.
- [83] D. Rueckert. "Segmentation and Tracking in cardiovascular MR Images using Geometrically Deformable Models and Templates". PhD thesis, Imperial College of Science, Technology and Medicine, University of London, London, 1997.
- [84] 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.
- [85] F.B. Sachse, G. Seemann and C.D. Werner. "Modeling of Electro-Mechanics In The Left Ventricle". IEEE Computers in Cardiology 2002. September 22-25, 2002. Memphis, Tennessee, USA.
- [86] C. Scarfone, "Single Photon Emission Computed Tomography (SPECT)", Duke University Medical Center, 1995.  
<http://www.bae.ncsu.edu/bae/courses/bae590f/1995/scarfone/>
- [87] J. Shewchuk. "An introduction to the conjugate gradient method without the agonizing pain". Technical Report CMUCS-TR-94-125, Carnegie Mellon University, 1994.  
<http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.ps>
- [88] F. Solina and R. Bajcsy. "Recovery of parametric models from range images: The case for superquadrics with global deformations". IEEE Transactions on Pattern Analysis and Machine Intelligence, 1990.
- [89] L.J. Spreeuwiers and M. Breeuwer. "Simultaneous Extraction of Epi and Endocardial Boundaries from Short-Axis MRI Cardio Images Using Coupled Active Contours". IEEE Computers in Cardiology 2003. September 21-24, 2003. Thessaloniki, Greece.

- [90] L. Staib, J. Duncan. “*Boundary finding with parametrically deformable models*”, IEEE Transactions On Pattern Analysis and Machine Intelligence, 14(11) 1061-1075 (1992).
- [91] A. Susín. “*Càlcul Numèric i Simulació*”. Assignatura Càlcul Numèric i Simulació (ETSEIB-UPC).  
<http://www-ma1.upc.es/~susin/contingut/files/apuntsCN.pdf>
- [92] T. Taxt, A. Lundervold, J. Strand and S. Holm. “*Advances in medical imaging*”. Proceedings, 14th International Conference on Pattern Recognition. 505-508 1998-08. 14th International Conference on Pattern Recognition. Brisbane, Australia.
- [93] D. Terzopoulos and K. Fleisher. “*Deformable models*”. In The Visual Computer, pages 306--331, 1988.
- [94] D. Terzopoulos, A. Witkin and M. Kass. “*Symmetry-seeking models for 3D object reconstruction*”. Int. J. Comput. Vision, vol. 1, no. 3, 1988.
- [95] D. Terzopoulos and D. Metaxas, “*Dynamic 3-D Models with Local and Global Deformations: Deformable Superquadrics*”, IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 13, No.7, pp. 703-714, 1991.
- [96] D. Terzopoulos, X. Tu and R. Grzeczuk. “*Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World*”. Artificial Life 1(4) (1994).
- [97] D. Terzopoulos. “*Artificial Life for Computer Graphics*”. Commum. ACM 42(8) (1999) 32-42.
- [98] Various authors. “*Numerical Recipes in C++*”. Numerical Recipes Software (ISBN 0-521-75033-4). Published by Cambridge University Press (CUP, with U.K. and U.S. websites).  
<http://www.nr.com/>
- [99] P. Volino and N. Magnenat-Thalmann. “*Implementing fast cloth simulation with collision response*”. In Proceedings of the Conference on Computer Graphics International (CGI-00), 257-268.
- [100] P. Volino and N. Magnenat-Thalmann. “*Comparing efficiency of integration methods for cloth animation*”. In Proceedings of the Conference on Computer Graphics International (CGI-01).
- [101] Y.F. Wang and J.F. Wang. “*Surface Reconstruction using Deformable Models with Interior and Boundary Constraints*”. IEEE Trans. Pat. Anal. Mach. Intel., 14(5):572--579, May 1992.
- [102] I. Weiss. “*Shape Reconstruction on a Varying Mesh*”. IEEE Trans. on patt. Anal. Machine Intell., vol. 12, no. 4, pp.345-361, 1990.

[103] A. Witkin and D. Baraff. “*Physically Based Modeling: Principles and Practice*”. Online Siggraph '97 Course notes.

<http://www-2.cs.cmu.edu/~baraff/sigcourse/index.html>

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

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

[106] C. Xu and J.L. Prince, “*Generalized gradient vector flow external force for active contour*”, Signal Processing - An International Journal, Vol. 71, No. 2, pp. 131-139, 1998.

## A SPECT imagery

Our system takes as its input SPECT, *Single Photon Emission Computed Tomography*, images. SPECT, gSPECT (*gated SPECT*) and PET (*Positron Emission Tomography*) datasets are inside the category of *Nuclear Medicine* data. Those images give functionality keys about the organ and do not describe its anatomy. Therefore, data shows the activity being held in terms of the amount of useful tissue, never giving a clue about shape. From this knowledge it becomes clear that ischemic areas, it means 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.

Chris Scarfone [86] points out how this technology was born. The first ECT (*Emission Computed Tomography*) device was the MARK IV developed by Edwards and Kuhl. This system consisted of several banks of sodium iodide (NaI) photon detectors arranged in a rectangular shape around the patient's head. It comes clear that in this case, the images to acquire where from the brain. The first commercial imaging device had 32 photon detectors and was called the Tomomatic-32. Many developments were presented in the early to mid 1970s.

Early applications of ECT resulted in diagnostically unusable images. That issue caused this technique to be partially forgotten. It was not until the introduction of x-ray CT by Hounsfield and Cormak, applied to nuclear medicine ECT, that this imaging modality entered the medical imaging practice. Segmentation algorithms designed for x-ray CT had to be modified for ECT to take into account particular effects of photon attenuation and scatter within the body and limited mechanical and electrical detector response. Then ECT images allowed qualitative and quantitative image analysis and therefore clinical use.

The acquiring of SPECT images involves lots of parameters including: attenuation, scatter, uniformity and linearity of detector response, geometric spatial resolution and sensitivity of the collimator, intrinsic spatial resolution and sensitivity of the Anger's camera, energy resolution of the electronics and system sensitivity among others. The calibration and monitoring of these parameters falls under a certified nuclear medicine technician or a medical physicist. Among all the parameters, collimation is responsible from determining the system's spatial resolution and sensitivity (amount of photons detected per second). System resolution and sensitivity are key parameters related to the performance of a SPECT system.

The acquiring process begins with the introduction of a marker by intravenous injection. The marker acts as a transportation for two radioactive atoms (Technetium-99m and Thallium-201). Those emit Gamma rays that will be captured by the receptor, conforming then the final image. When mixed with blood, the liquid goes through the ventricle filling all its volume and giving the possibility to evaluate its internal volume, via the acquired images.

The acquisition planes are horizontal and vertical, equally spaced. Typically acquisition procedures involve 64 planes normal to the symmetry axis (X and Y) and 24 planes following it (Z). Figure A.1 shows the planes.

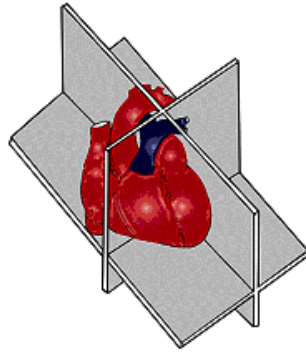


Figure A.1: Acquisition planes in SPECT imagery [56].

Final data results on low-resolution images attending to its spatial resolution (3mm x 3mm x 5mm). That will be a big issue, especially when recovering the internal surface due to its tiny discrete dimensions. In figure A.2, the bright areas correspond to tissue with blood irrigation.

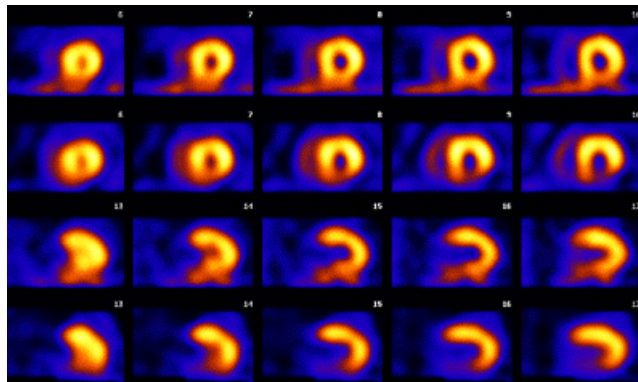


Figure A.2: Images from the horizontal (first and second rows) and vertical (third and fourth rows) planes [56].

SPECT images are difficult to process for several reasons:

- The datasets lack resolution.
- The region of interest doesn't expand all along the slices. On the contrary, it is confined into a small region nearby the geometrical center.
- The images are characterized by very coarse depth and spatial sizes.
- It is common to lack data voxels due to failures in the capture process.
- Data of interest is not necessarily aligned with the geometrical center of the image.
- Isquemic areas do not appear in the images.
- Noise emerges in the form of other structures attached to the data of interest such as the right ventricle or digestive activity affecting the left ventricle visualization.

Figure A.3 shows two actual pathological cases (64 x 64 pixels each slice). First, note the small region of interest for the left ventricle section, which is more or less centered in the image. There are ischemic areas emphasized by the red circles in the second row. This absence of data must be corrected by our segmentation algorithms in order to get acceptable measurements. Note also the noisy structure in the second case on the right (blue circle).

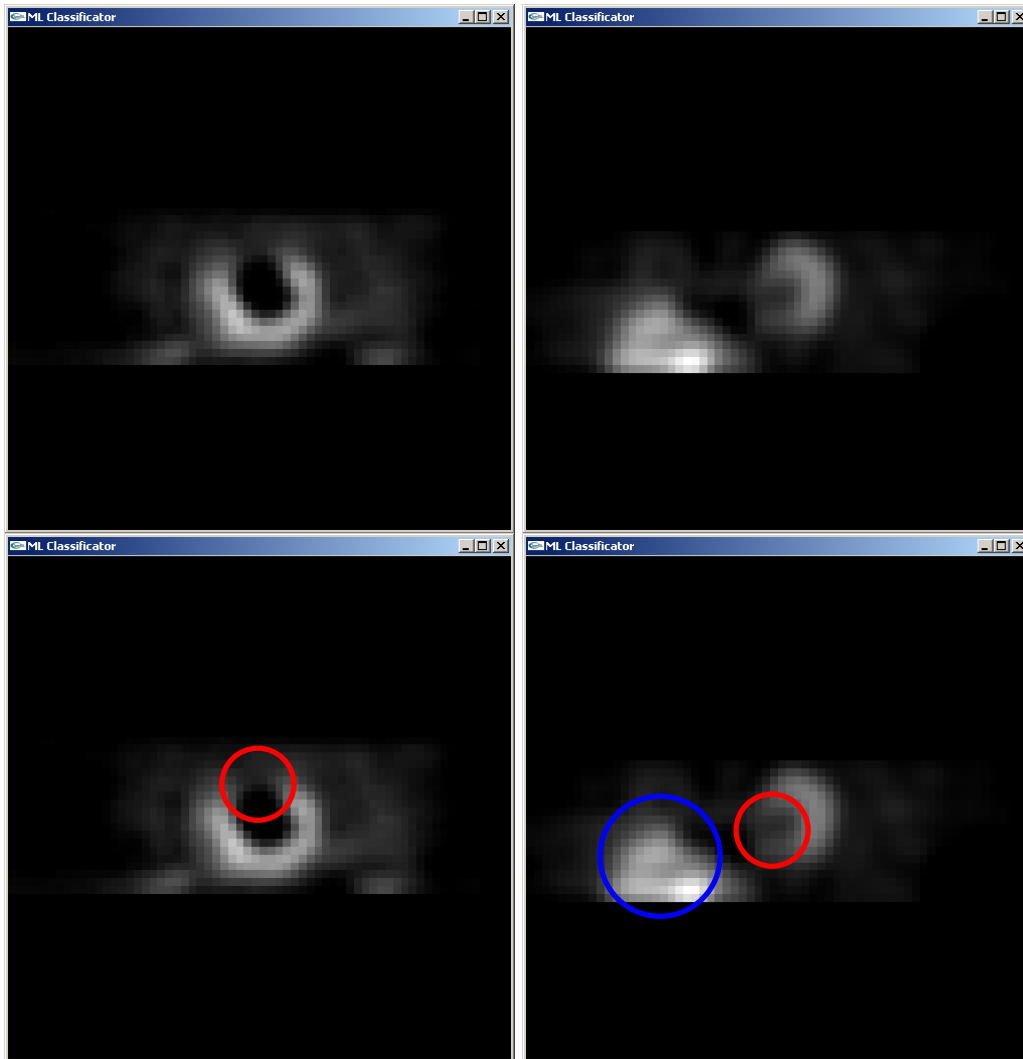


Figure A.3: SPECT images from pathological cases.

All these issues must be taken into account by the 2D processing module, which has to mark the data boundaries of the left ventricle. Once having the boundaries, we can build a 3D voxel dataset as an input for our 3D reconstruction method.

Figure 6.18 of chapter 6 shows a different type of cardiac test: *Nuclear ventriculography* (MUGA or RNV). It also uses radioactive tracers but in this case in order to mark heart chambers. Then the process is non-invasive because heart structures are not touched by instruments. Like in SPECT tests, common isotopes include technetium and thallium. The isotopes attach to red blood cells and enter the circulation system. The images can be synchronized with an electrocardiogram. Abnormal results indicate a myocardial infarction, diseases in the coronary artery or in a heart valve, among other disorders.

See [29, 92] for a complete analysis on medical imaging modalities.

## B The left ventricle

Rückert summarizes the cardiovascular system functionality in [83]. We will focus on the left ventricle function inside the heart system. The human heart consists of four chambers called atrias and ventricles (see figure B.1). The basic function of ventricles is to act as a pump that forces blood through the blood vessels. This function can be achieved because of the muscular tissues that are part of the ventricle walls. This muscular tissues compound the *myocardium* cavity whose surfaces are called *endocardium* (inside) and *epicardium* (outside). The atrias can contract also but only as a reservoir which is filled by the blood that returns through the veins to the heart.

The blood flows through the superior and inferior vena cava into the right atrium. It contracts and causes the blood to enter the right ventricle that expels it to the lungs. After being saturated with oxygen, the blood flows back into the left atrium whose functionality is to empty it into the left ventricle by opening the mitral valve. From the left ventricle, which is the largest and strongest chamber, the blood is ejected with high pressure into the aorta in order to supply the tissue of the entire body. The valves are intended to control the phases regarding the cardiac cycle plus preventing back flows of blood.

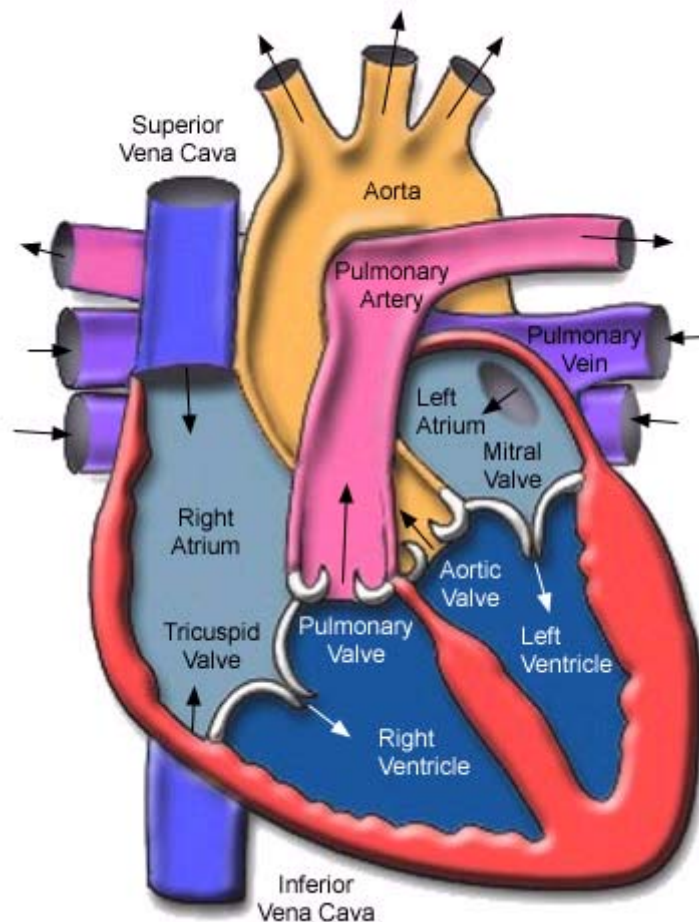


Figure B.1: Anatomy of the left ventricle.

(Image copyright 1996 Texas Heart Institute, [www.texasheartinstitute.org](http://www.texasheartinstitute.org))

A complete cardiac cycle is characterized by two phases: contraction and relaxation (see figure B.2). The contraction phase is called *systole* and it is also divided into two steps:

a first stage where the valves close, the ventricular muscle starts contracting and the pressure increases; a second stage where the valves open due to the high pressure levels and the ejection starts.

The relaxation phase is called *diastole* and it can be divided into two stages as well. In the first stage, all the valves close and the ventricular pressure decreases surprisingly fast. During the second stage, the valves separating atria and ventricles open and the ventricles are filled with blood. The pressure in the ventricles increases a little bit and the cycle starts again.

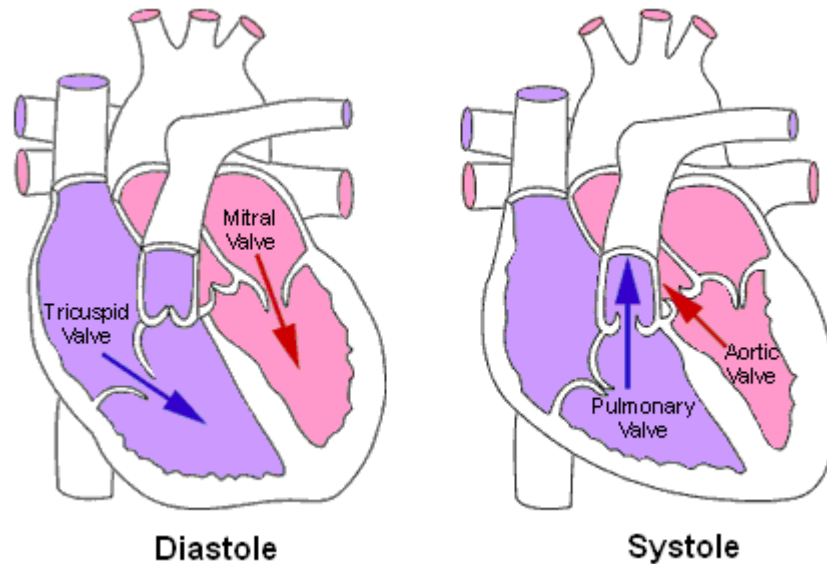


Figure B.2: The cardiac cycle into two phases.

(Image copyright 1996 Texas Heart Institute, [www.texasheartinstitute.org](http://www.texasheartinstitute.org))

The conduction system is responsible from generating the electrical impulses that cause the heart to beat (see figure B.3). These impulses begin in the sinoatrial (SA) node, in the top of the right atrium. Impulses from the SA node cause the atria to contract. After that, the signal goes through the atrioventricular (AV) node. In there, the signal is checked and retrieved to the muscle fibers of the ventricles, allowing contraction. The beating frequency depends on the rate followed by the SA node but also on other parameters such as physical demands, stress, or hormonal factors associated to every concrete person.

We can conclude that as long as the left ventricle is the strongest chamber in the heart, it is responsible from delivering oxygenated blood to the entire body through the circulatory system. The role of the left ventricle is extremely important and this is the main reason that forces physicians to pay an especial attention to it. Heart attacks can be located at several places but specialists are especially concerned with those in the left ventricle, because of the high degree consequences that might arise. If ischemia is placed within the left ventricle, its pumping power decreases and the volume of blood that has to be ejected is not maintained as it should.



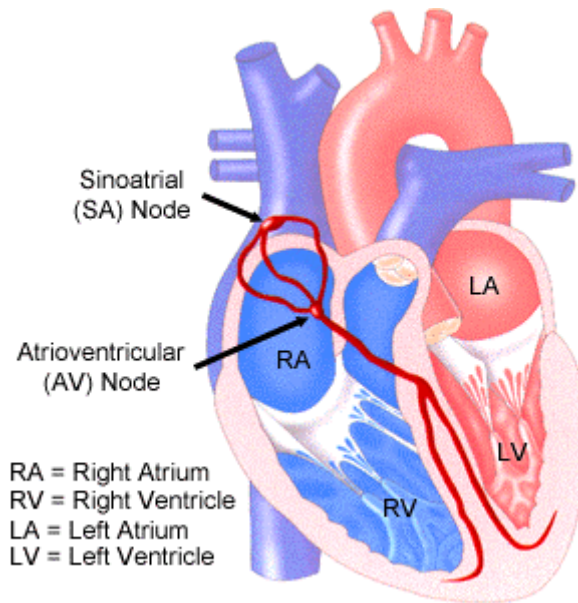


Figure B.3: The conduction system.

(Image copyright 1996 Texas Heart Institute, [www.texasheartinstitute.org](http://www.texasheartinstitute.org))

*Copyright notice*

1. The Texas Heart Institute is pleased to grant you conditional permission to use the copyrighted graphics in this appendix from our website at:

<http://texasheartinstitute.org> (and mirror site <http://www.tmc.edu/thi/>)

2. No commercial gain may be derived from the sale or distribution of this publication by the author or the publisher.

3. Texas Heart Institute reserves all rights and does not grant permission for any subsequent republication, commercial or otherwise, of these images in any media (print, web, video, etc.).

4. Permission to reproduce these images is contingent upon acceptance of these terms. No rights to any further use beyond this one time are hereby granted.

5. The authors are: Department of Scientific Publications, Texas Heart Institute: <http://texasheartinstitute.org/scipub.html>

## C Quantitative parameters of the left ventricle

There are several parameters relevant for diagnosis. Physicians detect cardiovascular diseases by analyzing them. Then it is important to build a tool that offers qualitative results, in the sense that a physician can see the reconstructed left ventricle, but also quantitative analysis based on the recovered surfaces and volumes.

The *cardiac volume* is a key parameter for diagnosis purposes. The cardiac cycle can be summarized into three stages [83]:

- A first stage of contraction where the volume decreases in a very fast manner.
- A second stage of relaxation where the volume increases quickly.
- A third stage of relaxation where the volume increases slowly until reaching its maximum and beginning to decay for the next cycle.

Figure C.1 shows the three phases for the cardiac cycle presented in section 6.1.4. The graph depicts the three phases in a clear manner. The time axis corresponds to the 8 instants of the acquiring process. The volume axis stands for the internal volumes of the left ventricle (myocardium) in  $\text{mm}^3$ . This is a dataset from a healthy patient.

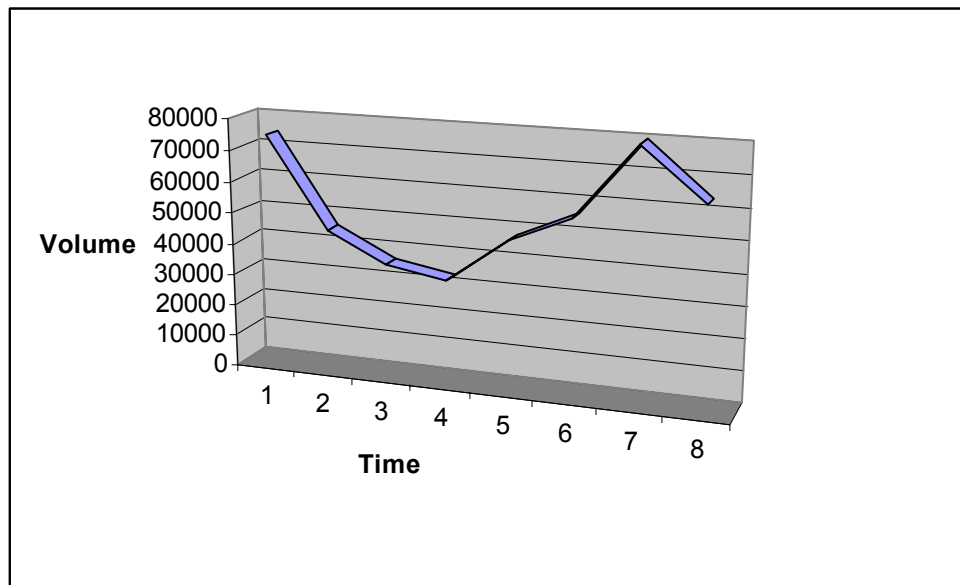


Figure C.1: The cardiac volume evolution for an actual's patient data.

The different issues regarding the evaluation of the volume inside a mesh are presented in section 5.6 of this document.

Quantifying the *ventricular wall thickness* is also possible within our system. This parameter stands for the volume between the epicardium and the endocardium. Measurements on the wall thickness are also possible because we are working into a mathematical context applied within a computer graphics framework. As long as we build two surfaces made from triangles, measurements on distances can be achieved by ray casting techniques that couple pairs of polygons in order to retrieve their relative distance.

The *ejection fraction* (EF) is the fraction of blood that the left ventricle does pump out per beat [32]. A value of 50% indicates that the ventricle ejects half of its volume for each contraction. For a healthy patient, this value has to be within the 50% and 70%.

This value can be obtained as equation 6.4 shows. We rewrite this equation here for a clear and complete summary.

$$EF = \frac{DiastoleEndVolume - SystoleEndVolume}{DiastoleEndVolume} \quad (C.1)$$

Equation C.1 shows the EF evaluation method. It is important to point out that all the volumes are internal. If we apply this equation to the healthy patient of figure C.1, we obtain the results in equation C.2.

$$\begin{aligned} EF &= \frac{DiastoleEndVolume - SystoleEndVolume}{DiastoleEndVolume} \\ &= \frac{79581 - 32956}{79581} = 0.586 \Rightarrow 58.6\% \end{aligned} \quad (C.2)$$

Medical software derived an EF value of 53%, which stands for a percentage error of 5.6 % with regard to our application.

These three parameters are considered fundamental by physicians. As shown, their computation is possible by using discrete versions of the underlying mathematical framework.

## D The Phantom volume

In order to measure the reliability of medical hardware used for retrieving medical images it is necessary to use a medical volume with known characteristics: the Phantom volume. There are several volumes available from different vendors and hospitals. All of them present absorption and transmission characteristics similar to the ones related to a certain tissue. In our case, cardiac tissues like the left ventricle muscle.

Figure D.1 shows the geometry for a Phantom volume. It can be seen that it is similar to the left ventricle while being more symmetric of course. Anyway, it is enough for testing functionality, performance and the involved parameters.

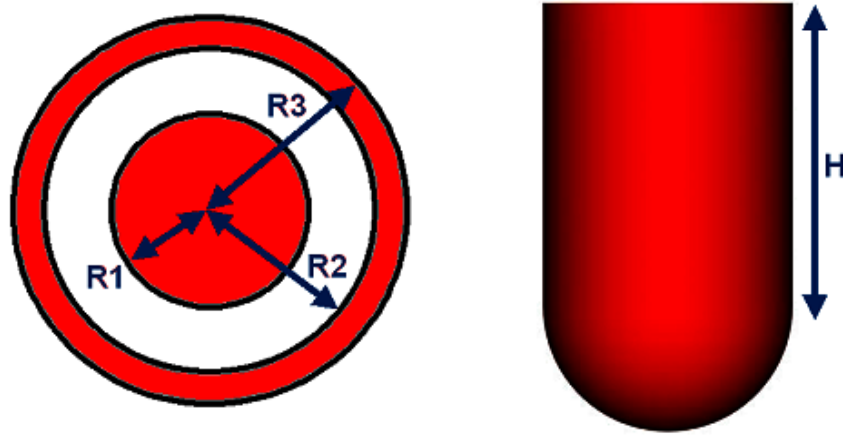


Figure D.1: The Phantom volume geometry.

Where  $R_1 = 20$  mm,  $R_2 = 35$  mm,  $R_3 = 45$  mm and  $H = 55$  mm. Those measures lead to internal and external volumes of  $89794.5$  mm<sup>3</sup> and  $305301.4$  mm<sup>3</sup> respectively. The walls of the Phantom can be described with equations D.1 and D.2. Equation D.3 corresponds to the global volume evaluation.

$$\begin{aligned} (x - x_{cm})^2 + (y - y_{cm})^2 &= r^2 \\ 0 - z_{cm} &\leq z - z_{cm} \leq H - z_{cm} \end{aligned} \quad (D.1)$$

$$\begin{aligned} (x - x_{cm})^2 + (y - y_{cm})^2 + (z - z_{cm})^2 &= r^2 \\ -r - z_{cm} &\leq z - z_{cm} \leq 0 - z_{cm} \end{aligned} \quad (D.2)$$

$$\begin{aligned} GlobalVolume &= Volume_{HalfSphere} + Volume_{Cylinder} = \\ &= \frac{1}{2} \cdot \frac{4}{3} \pi r^3 + \pi r^2 H \end{aligned} \quad (D.3)$$

In equations D.1, D.2 and D.3,  $(x_{cm}, y_{cm}, z_{cm})$  is the center of mass of the cylinder,  $H$  is the height of the cylinder,  $r$  stands for the radius which depending on the wall must be equal to  $R_1$  or  $R_2$ .

Two cylinder-sphere sets are necessary in order to compound the test volume walls, as figure D.2 shows.

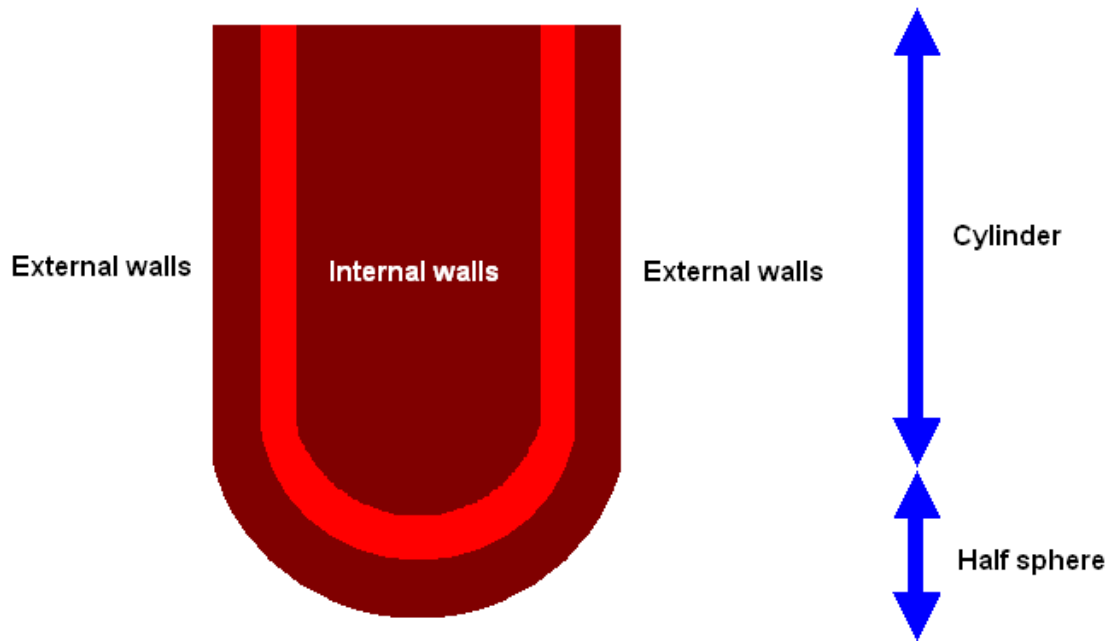


Figure D.2: The walls of the Phantom volume.

The Phantom volume can be filled in different ways for simulating several abnormalities. For that purpose, it is covered and hardly screwed in the top while maintaining two openings. Figure D.3 shows this.

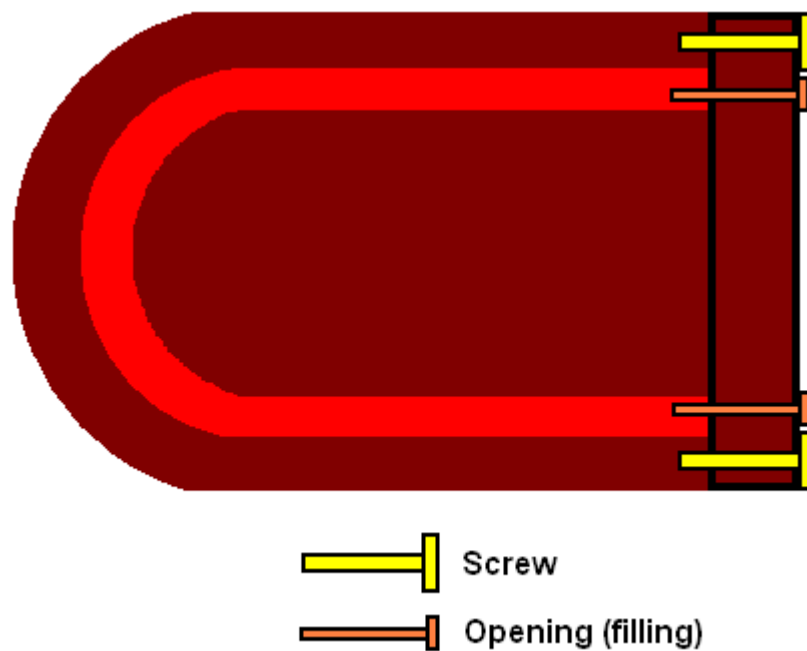


Figure D.3: Filling of the Phantom volume.

Note that there is another peripheral called Phantom in the Computer Graphics/Robotics area. For clarity reasons, let us state the differences between them. In Robotics, the Phantom is a robot arm used for manipulation purposes. The arm is attached to a real-time virtual reality application in order to “touch” the objects in the synthetic scene. In fact, we are also using a Phantom robot arm within this project for the physicians to

interact with the reconstructed left ventricle. Some examples can be seen in [20, 36, 63, 80].

## E The DICOM file format

DICOM stands for the *Digital Imaging and Communications in Medicine* standard. It describes a way of interconnectivity in terms of formatting and exchanging medical images. In fact, it also describes the interface related to the transfer of data in and out of an imaging device. Several imaging modalities make use of this standard: CT, MR, PET, Nuclear Medicine, Ultrasound, X-ray, CR, digital radiography, digitized film, video capture and HIS/RIS information among others.

The first draft of the DICOM standard dates from 1984. It was defined by the American College of Radiology (*ACR*) and National Electrical Manufacturers Association (*NEMA*). Version 1.0 was published in 1985 and version 2.0 in 1988. The DICOM Standard Committee was formed in 1996. The committee was created to extend the standard to support all forms of biomedical imaging. The standard was definitely placed as a way of reinforcing communications in order to advance diagnosis and treatment decisions.

The standard specifies a set of protocols to be followed by devices, the syntax and semantics of commands and information that must be supplied with each implementation.

The basic file structure is divided into a header (a 128 byte File Preamble, followed by a 4 byte DICOM prefix) and a dataset. The dataset is also divided into data elements. Data elements can be standard and private. Figure E.1 shows this structure.

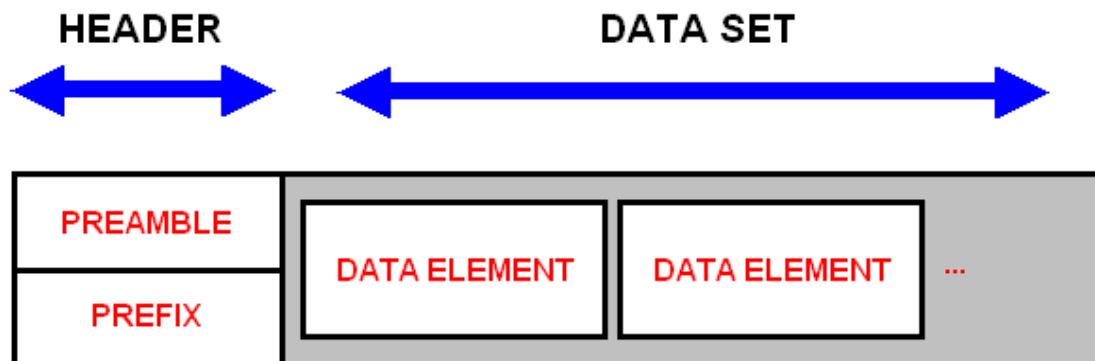


Figure E.1: The DICOM file format structure.

When operating the DICOM file format, we must be aware of the byte ordering. It is different depending on the hardware (Big Endian or Little Endian). Little Endian byte ordering encodes the least significant byte first while Big Endian byte ordering encodes first the most significant byte. Translation from an ordering to the other can be achieved easily.

Note also that DICOM supports the use of JPEG Image and Run Length Encoding (RLE) Compression methodologies. More information regarding the DICOM file format can be seen in [22, 26].