EPCC-SS99-02

Extensions to CAMRA

Mark D. Wilson

Abstract

Describes the process and implementation details for the implementations of the Centerline and Radial methods for calculating the volumes of blood consumed by the heart over a specific period of time from non-invasive magnetic resonance data. [3] Specificially, details decisions and results of the computer based implementation of these techniques.

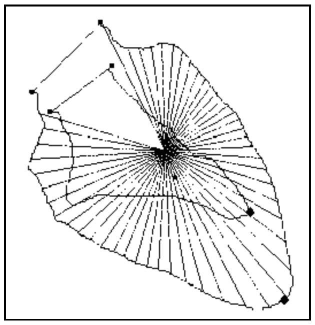
Contents

1 Introduction

The program, in its current form is given a data file that contains the curve data to be processed. The curve data is represented by a relatively small number of points, which are interpolated to provide rough curve data.
The data file is produced using the current version of the Cardiac Magnetic Resonance Analysis software (CAMRA) [1]. This data file is then processed and the data stored in an internal data structure. This internal data structure can then be passed to one of the following functions.

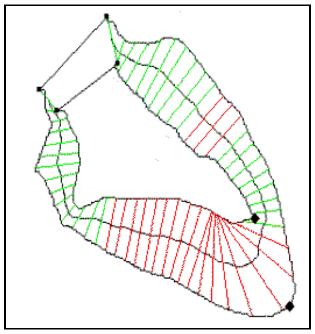
1.1 Radial Method

Simply, the radial method is as follows. Find the centre of both curves. Slice the curves up a specified number of times, from a given offset point. The length of each piece of the 'pie' can be used as an indication of the distance travelled over the period of time. The difference in equivalent slice of the 'pie' can be used to give an indication of the heart wall movement, which in turn can be used to gauge movement of the heart.



1.2 Centerline Method

Following the steps described in [4], the curves are processed so that a 'centreline' can be drawn at a midpoint between the two wall positions. The advantage of this method is that sections can be made perpendicular to the centreline, that provide a more accurate indication of the heart wall movement.



2 Background

Heart disease is one of the major causes of death in Scotland and much of the western world. These deaths can often be prevented with early diagnosis.
By using Magnettic Resonanance Imaging and existing CAMRA software, it is possible to measure the heart wall movement.
The aim of this project is to take the movement information and process it into an easily readable form that can give physicians a better idea of what (if anything) may be wrong with the heart.
This would be achived by drawing a diagram indicating movement for each part of the heart for different cross sections.
Movement of the heart can be calculated in many ways. Two of these ways, which were investigated in the course of the project were:

- The radial method.
- The centerline method.

3 Method

A number of software engineering techniques and practices were applied during the course of the project, from using the advantages of object oriented languages to make code easier to implement, to applying certain methodologies in the development and testing processes.

3.1 Evolutionary Approach

The software implementation methodology was to use the ’Evolutionary’ approach. As the work is focused on proof of concept, rather than for implementing a standardised implementation, the code was implemented in a ’Get something working first - then tidy up the rough edges’ approach, rather than trying to design a complete system from scratch.
This has the advantage that if things are going to go wrong during the development process, they will be found at the time, rather than during the integration stages at the end of the software project.

3.2 Problems

In any software development process, there are problems - bugs and design problems. Particularly when an evolutionary model is being applied to software development. I will highlight some of the difficulties that were encountered, and how they were solved.

3.2.1 Standard Library Implementation

During the initial stages of the project there was the question of ‘how do we parse the camra data files’. Although this problem was fairly straightforward, it would require the use of dynamic data structures as variable numbers of points would be read in from the files and would need to be manipulated within the program.
Instead of implementing linked lists manually, there was a decision to take advantage of the C++ standard libraries - particularly the ’vector’ data type for storing data.
However, this proved more difficult than expected, as the current version of Visual C++ (version 4) didn’t handle these data types very well. After upgrading Visual C++ to version six, and installing the relevant service packs to get around the ’Internal Compiler Error’ messages, it was found that after some effort the standard library functions could be used.

3.3 Spline Representation

The outputted data from the camra software is a series of spline curves. As to test the code initially, and because of the difficulties in finding information explaining how exactly these spline curves should be interpolated, a temporary first order representation was used - the curves were represented in polar coordonates, and interpolated in these polar coordinates. This way there were not straight lines between two provided points, but rather a curve, which is more representative of the kind of shape that we were initially looking at.

3.3.1 The Centerline Method

After looking at examples of the centreline method in action, there was still the question of ’how does it work’. Unlike the radial method that is fairly obvious, the centreline method appeared to use more sophisticated calculation. The aim was to find out how the centreline method actually worked. After researching papers in the Medical Library and obtaining inter-library loans, the paper describing the centreline method was found. This started the remaining stage of the project - implementing the centreline method. The paper didn’t come with copies of source code, so there were difficulties in fully understanding the processes described, and modifications were made to make the algorithm work under a different set of assumptions, such as those provided by the CAMRA software as opposed to the data sources used for the examples in the paper.

3.3.2 Calculating the centre of the circle

One of the stages of the centreline algorithm described in the paper states that given three points along a smoothed curve, the circle containing those three points should be calculated. The problem with this however, is how one should calculate the center of the circle given three points.

If line equations are used to calculate the centre point, there is a lot of algebraic reduction required to perform this task.
It seemed at the time that an easier solution, of using sample code would provide the ability to solve this problem, though this in itself introduced a few problems:
The code file provided didn't include all the specified header files (containing things such as the 'Point' data type). These point types seemed to have additional data fields - as well as just X and Y values to represent a cartesian coordinate, they contained Z, m_x, m_y, m_z to represent the third dimension and appropriate gradients (respectively). The sample code used different data types from the existing code, so 'glue' code was created to allow the transfer of infomation between the two different parts of the program with a minimum of impact on the assumptions made in each part. This way, both pieces of code could continue to be updated and maintained seperately through the previously defined set of interfaces.
After modifications were made, bugs were encountered - there was the problem that the algorithm didn't consistantly return the centre of the circle - it did in some test examples but not all examples.
After encountering these problems, a new piece of code was written from scratch - that resolved some of these problems, using vector geometry rather than the previously used line equations. The remaining problems that exist with this function are directly related to the intersection function, that makes use of line equations.

3.3.3 Problems with line equations

There are problems in the calcluations - If for example there is a large change in the y axis (between two points) and a small change in the x axis (ie. An almost vertical line), the gradient value for the line is very large. This results in very large values being placed in a double type. If the type doesn't contain enough resolution to hold the number, then there will be problems representing the value correctly. As a result the values produced as a result of this calculation will not be correct.
This will also apply if the equation of the line refers to a line that has an angle over 90 degrees. This would have been resolved if more time was available.

3.3.4 Search Algorithm

The paper describes that after a line segment is calculated, care should be taken so that similar points are not calculated again. The paper then goes on to describe how this can be implemented. In this case a class was created that would detect if any of the lines specified already existed. If so, then it would not add a new one (as it is a duplicate). This appeared to produce some problems, so the method described in the paper in detail was used instead. The paper didn't explain why this should be performed over any other method.

3.3.5 Smoothing to 200 points

From the paper descirbing the centerline method, it mentioned that any curve provided to the system should be linearly interpolated to around, say, 200 points. This however proved to be a problem. If the curve initially provided had a lot more (say 1000 points), this would not be

a problem when it came to calculate the circles that travelled through three consecutive points. However, if there were fewer points to begin with, performing linear interpolation to increase the number of points to 200 would cause a problem. If three points were colinear, the theoretical circle that traversed those three points would be of infinite size, meaning it would be impossible to calculate the centre of the circle.
A solution that was used to solve this problem was that, if three points were colinear, then the next available point would be used. ie. one of the colinear points would be ignored until a set of points were found that were not colinear.
Another possible, and pontentially better solution to this problem is to perform spline interpolation on the input curves, increasing the number of points to say, around 1000. As the curve is calulated using spline interpolation it is very unlikely that two consecutive line segments on the interpolated curve would be colinear. Therefore avoiding this problem.

3.3.6 Visualising the Data

There was the question of visualising the data - plotting it in a form that could be readable by physicians. Also, to plot it in a form that would aid debugging.
There was the possibility of designing a specific windows application that would be able to show the data, however upon investigation, it seemed that a lot of additional skills would be required, in terms of MFC programming and much development time.
Due to time restrictions, a quicker, easier to implement solution was chosen - allowing the data to be used by a commercial charting package.
Methods were applied to the classes so that the data contained within them could be output to a comma delimeted file. This way the data files could be easily imported into a graphing packgage, such as Microsoft Excel, and a chart plotted showing the positions of the heart wall and the centerline.

4 Conclusion

Although the results are not fully conclusive, there are aspects of the code that seem to be functioning correctly. Certainly, the project has highlighted many of the pitfalls involved with transferring the algorithm from the paper into a working computer program, in particular, highlighting some of the assumptions that have been made by the authors, but not explained very clearly in the paper. For example, the types of data set used, and numbers of points used in the data sets.

5 Where to Next

- Implementation of the ’Improved Centerline’ method.[2].
- Utilisation of Three Dimensional data to improve the accuracy of the centerline method [5]

A Program Implementation Details

A.1 Data Structures

The program is implemented from scratch in C++. There are some advantages - standard library containers are used to store sets of points that represent the curves. Some classes have been created to represent commonly used pieces of data (such as 'points', curves, circles etc.) and common manipulations on these objects (eg. additon and subtraction of points; division of points by a scalar). There are also classes for special types of data (eg. Polar Coordinates) that support assignment from other data types. (eg. A point can be assigned to a polar coordinate, which results in the polar coordinate class representing the same object that was represented by the cartesian coordinate).

A.2 Methods

A.2.1 Curve Representation

The curve that is provided to the program is a curve that is either drawn by the user by following along the side of the Blood Pool wall with the mouse, using a spline curve. The curve data is then passed into a program as a file, which is then interpreted and interpolated to find radial cross sections.

A.2.2 The Bounding Box Method

To find the centre, a bouding box is drawn around the curve. The centre of the bouding box is calculated, and this value is used to convert the coordinates to polar values.

A.3 Interpolating Points

A.3.1 First Order Interpolation

B Known Bugs

There has not been time to fully complete and test the functions described above. There are a few known bugs that could easily be corrected given more time, and a few bugs that may require more thought or re-design to the code.

- The intersection function does not necessarily return the correct point of intersection for two points that are almost vertical from each other. This is due to a rounding error in the division. As the X value is very small, when the two numbers are divided, the result is higher than that which can be stored in a double data type. This results in error being introduced. A solution would be to say rotate the line segments by 45 degrees, and try to calculate again.

- The raidal method doesn't produce valid results - the center of the shape is determined by the bounding box centre rather than the gravitational centre. The gravitational centre implementation would have been introduced if sufficient time was available.

C Class and Function Summary

C.1 Global Functions

C.1.1 parse curves

```
vector<curve> parse_curves(void)
```

This function reads in all the CAMRA type data curves from the defined data file. All of the curves in the file are interpreted and placed in a 'curve' data structure. All the curve data structures are placed in a 'vector' which is returned. If an error condition is found, an exception is thrown.

C.1.2 distance

```
double distance(point first, point second)
```

This function returns the distance between the two specified points.

C.1.3 divide

```
points_t divide(int npoints, points_t points)
```

Takes a set of points and an integer. The function divides the curve specified by 'points' into 'npoints' points. ie. Perform linear interpolation to 'npoints'.

C.1.4 eatspace

```
void eatspace(char *string)
```

removes any preceeding space characters from the front of the specified string.

C.1.5 interpolate

```
point interpolate(point first, point second, double between)
```

Calculates the point at the given ration between the first and second points

C.1.6 longest

```
point_t longest(point_t a, point_t b)
```

Returns the longest curve out of `a` and `b`.

C.1.7 length

```
double length(Point *a, Point *b)
```

Returns the length of the line segment specified by the two points passed as parameters.

C.1.8 printPoints

```
void printPoints(points_t points)
```

Takes the list of points and prints them out to standard out in excel friendly format.

C.1.9 shortest

```
points_t shortest(points_t B1, points_t B2)
```

Returns the shortest of the two curves

C.1.10 smooth

```
points_t smooth(points_t points, int numavg)
```

Smooths the specified curve by calculating the average of numavg points nearest each point for each point.

C.1.11 takeSmallest

```
point takeSmallest(points_t &points)
```

Returns the smallest point in the data structure provided.

C.1.12 total_distance

```
double total_distance(points_t points)
```

Returns the toatal length of the curve specified.

C.1.13 centerpoints

```
crossects centerpoints(point_t D1, point_t L1, point_t NL1)
```

Takes two smothed curves (D1, L1, NL1), where D1 is the smothed curve provided as input, and L1 and NL1 are the two ’actual’ curves. Using the centerline method, the cross sections at certian points along the curve are calculated and placed in the ’crossects’ class. The crossects class contains a vector of line segments.

C.1.14 newcircle2

```
point newcircle2(point a, point b, point c)
```

A revised version of ’circle’ that uses vector geometry to calculate the centre of the circle specified by the three given points.

C.1.15 TRACE

This function provides the functionality for the error messages to be produced within the program

```
void trace(char* errormessage)
```

C.2 Classes

C.2.1 circle

- circle(point1, point2, point3) Instantiates the circle class, specified by the three points given.
- getCenter() Returns the point that should be at the centre of the circle.
- getRadius() Returns the radius of the circle.

C.2.2 curve

- add_point(double x, double y) Adds a point to the end of the curve data structure.
- assign(int saID, curvekind BP, int TimePoint, curvemethod method) Sets up information particular to that instance of the class.
- center() Performs the default centering function on the curve. (Bounding box)
- center(centermentod METHOD) Allows the program to specify a different center function to be performed on the curve.
- operator =(vector<point> p) Allows a vector<point> to be assigned to the curve allowing conversion between the two sites.
- polarize() Performs polar coordinate conversion on the curve.

- print() Displays debugging information particular to the class.
- read(char *filename) Parses curve data from the specified file.
- set_centrepoint(double x, double y) Allows a specific point to be set up as the center of the curve.
- centrepoint This field contains the point deemed to be the centre of the curve.
- points This is the data structure containing the cartesian coordinates of each point in the curve.
- polarpoints This is the data structure that containst the polar representation of each point in the curve.
- slice This field contains the number assigned to the particular slice that the curve data structure represents.
- thecurvemethod This field contains the type of data representation that is used to represent the curve. (eg. Spline, Point, etc.)
- timepoint This field contains the time point that the curve represents.

C.2.3 point

- operator +(point pt)
- operator -(point pt)
- operator /(double n) Performs the above operations on the point, so that expressions can be produced such as ‘c = a + b’ for the points.
- perpendicular() Converts the point to a point that is perpendicular to the original.
- distance A field used to store distance from a given point which is later used to choose the nearest.
- x The x cartesian value.
- y the y cartesian value.

C.2.4 polar

A class used in general to represent polar coordinates.

- angle This field contains the angle on the plane that the point represents.
- distance This field contains the distance from the origin on the plane that the point represents.

References

[1] M.J. Graves A. Avedisijan, M.D. Westhead. Camra: Parallel application for segmentation of left ventricle (lv) in short axis cardiac mr images. EPCC Internal.

[2] J.H.C. Reiber Craig D. von Land, S. Rohini Rao. Development of an improved centreline wall motion model. Computers in Cardiology, 1990. pp. 687-690.

[3] Sander J. Spanjersberg, BS. F. Paul van Rugge, MD., Ernst E. van der Wall, MD. Magnetic resonance imaging during doutamine stress for detection and localization of coronary artery disease. Circulation, 1994. No. 90,pp. 127-138.

[4] Harold T. Dodge, M.D. Florence H.Sheehan, M.D., Edward L. Bolson, M.S. Advantages and applications for the centerline method for characterizing regional ventricular function. Circulation, August 1996. Vol. 74,No. 2,pp. 293-305.

[5] Martin D. Kiil, BSc. Vincent G.M. Buller, MSc., Rob J. van der Geest, MSc. Assessment of regional left ventricular wall parameters from short axis magnetic resonance imaging using a three- dimensional extension to the imporved centerline method. Investigative Radiology, August 1997. Vol. 32,No. 9,pp. 529-539.

Mark Wilson a student at Univesity of Edinburgh going into his final year of an Honours degree in Computers Science & Artificial Intelligence. Particular interests lie in the implementation of innovative solutions for tackling real world problems.

Supervisor: Armen Avedisijan