From Biomedical Images to Geometric Models

Chand T. John[2], Scott L. Delp[1]
[1]Departments of Mechanical Engineering, Bioengineering, and Orthopaedic Surgery
[2]Department of Computer Science

Introduction

Advances in biomedical imaging have resulted in an abundance of data describing the form of biological structures across an enormous range of physical scales, from molecules to entire organ systems. Interpretation of this vast quantity and variety of data is greatly enhanced by the construction of 3D geometric models that allow one to measure and visualize key features of biological structures. Common biomedical image processing operations include *registration* of multiple images, *segmentation* of structures of interest, *visualization* of the data, *measurements* of structural features, and construction of *3D models* of anatomical and biological structures. Slicer [4] is a free, open-source, extensible and customizable application which contains a number of automated image processing algorithms. However, it lacks many standard tools for manual segmentation, which are needed in almost all research settings. We developed four algorithms to facilitate the implementation of manual tools that will boost Slicer's utility in biomedical research.

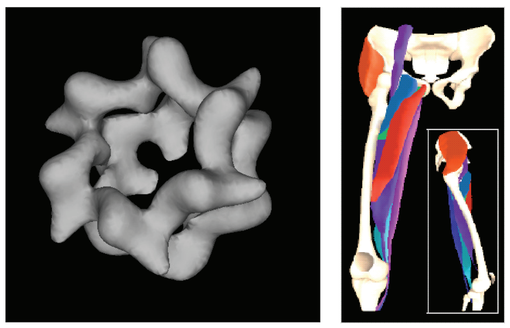

Figure 1. 3D geometric models of complex biological structures [2][1].

Methods

Slicer 2.6 was developed using Tcl 8.4.5/Tk 8.4.5 and Microsoft Visual C++ .NET 2003 in Windows XP. We developed algorithms for the following tasks:

1. Inserting intermediate control points in user-defined control polygons
2. Selecting and moving points by automatic proximity detection
3. Real-time cardinal spline interpolation of a user-defined polygon
4. Automatic sampling of points from a cardinal spline for model construction

Algorithm 1

In Insert mode, when the user clicks at an arbitrary point *M* in space, we insert the point *M* into the control polygon between the closest control point *Q* to *M*, and the neighbor of *Q* which is closest to *M*.

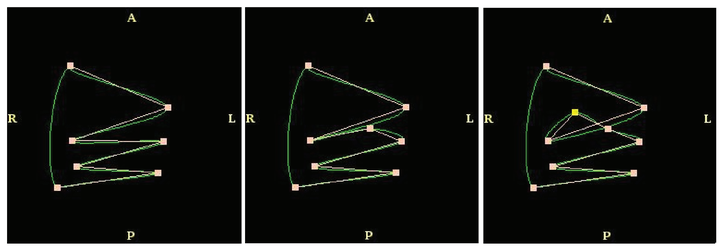

Figure 2. Two points are inserted into a control polygon with high curvature. A self-intersection occurs after the second point is added.

Algorithm 2

To mimic standard drawing programs, after each mouse motion, we compute the distance from the mouse location to each edge of the control polygon. If the mouse is close to any edge, we set the mode to Move. Otherwise we set the mode to Select.

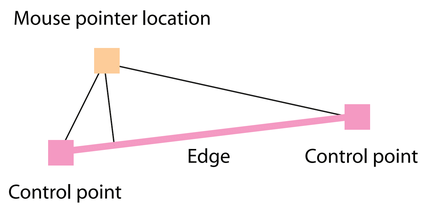

Figure 3. The perpendicular distance from the mouse location to each line segment of the control polygon is computed using only one non-integer variable.

Algorithm 3

As the user plots control points, we interpolate a Catmull-Rom spline through those points [3]. Each end derivative is computed by reflecting the neighboring control point's derivative vector about the end edge's perpendicular bisector.

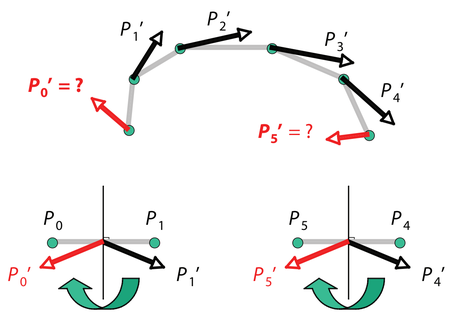

Figure 4. End derivatives are computed by reflection.

Algorithm 4

Once the user has finished plotting a control polygon, we sample *d* points from each cubic curve segment of the cardinal spline in addition to the original control points. Here *d* is the sampling density chosen by the user. These points are used for constructing a 3D model. Each curve has a parametric representation *B*(*t*), where *t* ranges from 0 to 1. For a particular cubic curve, we compute the points $B(1/(d+1)), B(2/(d+1)), \ldots, B(d/(d+1))$.

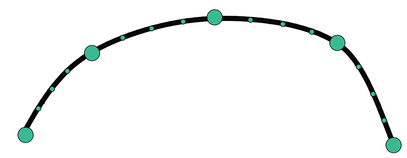

Figure 5. A cardinal spline sampled with density $d = 3$. The large circles indicate the original control points, while the small circles indicate the sampled points.

Summary

Algorithms 1 and 4 both work under normal conditions, where contours are drawn with only gentle variations in curvature between adjacent control points, as is the case for most structures segmented in biomedical images. Algorithm 2 works well for all practical purposes. Algorithm 3 performs well overall but does have curvature problems at the endpoints. This can be improved by checking whether the computed end derivative satisfies some curvature constraint and negating the vector if it does not. All of the presented algorithms are efficient enough to avoid any experience of computational delay for the user.

References

[1] A. S. Arnold, S. Salinas, D. J. Asakawa, S. L. Delp. *Computer Aided Surgery*, 5: 108-119, 2000.
[2] M. Bern, J. Chen, H. C. Wong. *RECOMB*: 118-132, 2005.
[3] G. Farin. *Curves and Surfaces for CAGD*, 5th ed., Academic Press, 2002.
[4] http://slicer.org

Acknowledgements

This work was supported by an NIH Predoctoral Fellowship. Thanks to Allison Arnold, Thor Besier, Silvia Blemker, Christie Draper, Kate Holzbaur, Luis Ibáñez, Ron Kikinis, Bill Lorensen, Steve Pieper, Will Schroeder, and Nathan Wilson.