AIAA Aviation
22-26 June 2015, Dallas, TX
22nd AIAA Computational Fluid Dynamics Conference

meshCurve: An Automated Low-Order to High-Order Mesh Generator

Jeremy Ims* and Zhaowen Duan†, Z.J. Wang‡
The University of Kansas, Lawrence, KS, 66044, US

In this paper, we provide an overview of the soon-to-be released program meshCurve. The software aims to be a user-friendly tool to convert linear unstructured meshes to curved high-order meshes, based on a geometry reconstruction procedure. Usability and accuracy are its goals. Using state-of-the-art algorithms, the software reconstructs a high-order curved surface from the linear surface mesh, while retaining sharp feature edges from the original low-order surface. The upgrade process is automatic and does not require a CAD file. meshCurve may be used by CFD practitioners and researchers to easily produce quality high-order meshes from existing low-order meshes. The CGNS standard is used as the file format for input and output.

I. Introduction

OVER the last two decades, the CFD research community has invested significant effort into the development of high-order methods. The potential advantage of high-order methods is that they offer greater accuracy than current simulation tools, making it possible to reach engineering error tolerance level with less effort.[1] However, for these methods to be effective, they must be paired with high-order meshes that include high-order representations of curved boundaries.[2,3]

Software to generate high-order meshes is not readily available because the algorithms to produce high-order meshes are still under development. Furthermore, since high-order methods aren't yet available, software companies are not yet motivated to provide the level of support for high-order meshing that they provide for low-order meshing. As of now, commercial meshing software has only partial high-order functionality. Free meshing software, with the exception of Gmsh, has no high-order functionality. Gmsh is a useful tool, but it cannot easily accommodate complex models due to its limited CAD features. The lack of adequate software has prompted researchers at the 1st International Workshop of High-Order CFD to prioritize high-order mesh generation as a pacing item for future research.[4]

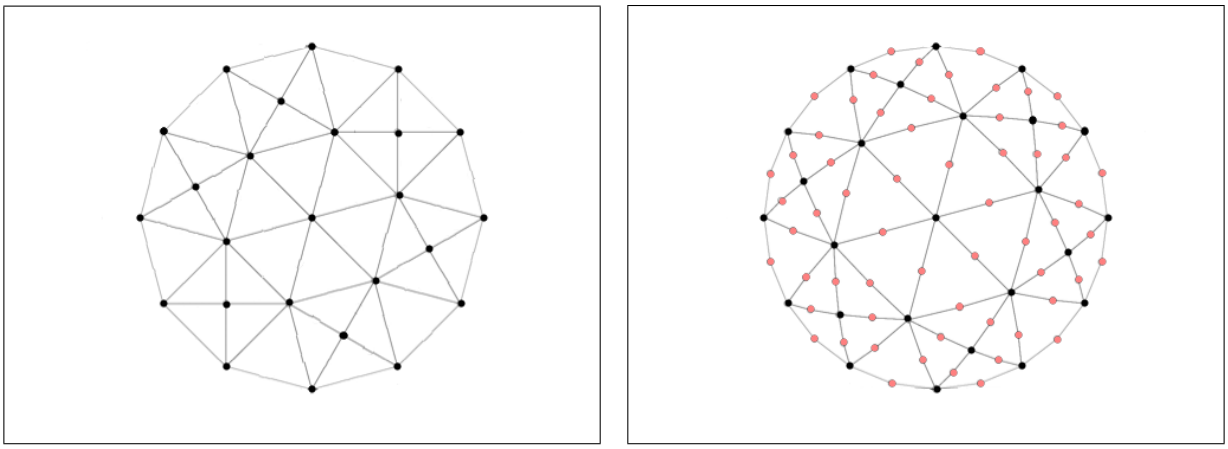
At the University of Kansas CFD lab, our work requires unstructured, high-order meshes, and like others, we could not find a robust, general purpose, high-order mesh generator. A second problem was the unavailability of original CAD geometries. No existing software had the capability to convert a mesh from low to high-order. So we decided to create the software. Our software utility, called meshCurve, is designed to be a user-friendly system to upgrade unstructured meshes from low-order to high-order, without reliance on a CAD model. Although it is designed for CFD work, it is not CFD specific. Therefore, it may be used generally as a low to high-order unstructured mesh converter.

This paper provides an overview of meshCurve in terms of its usability and capability, with explanations of the overall design goals, structure, and algorithms.

II. Low versus High-Order Meshes: An Illustrative Example

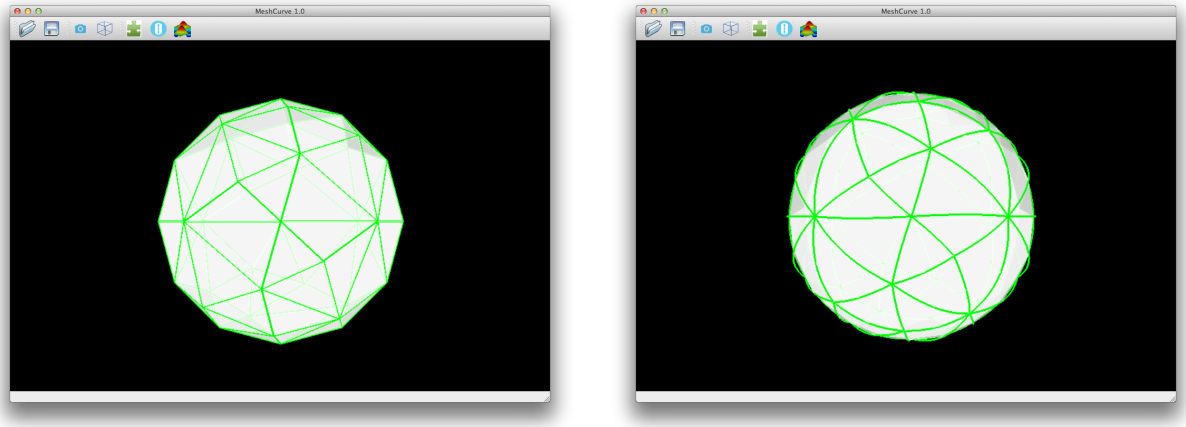
Figure 1 illustrates the difference between a low-order mesh and a high-order mesh with a side-by-side comparison of two circular meshes. The key difference is the number of nodes, with the low-order mesh, shown on the left, having fewer than the high-order mesh, shown on the right. The extra nodes enhance

*PhD Student, Department of Aerospace Engineering, AIAA Member, email:ibjeremy@ku.edu
†PhD Student, Department of Aerospace Engineering, AIAA Member, email:zhaowen@ku.edu
‡Spahr Professor and Chair, Department of Aerospace Engineering, AIAA Associate Fellow, email:zjw@ku.edu



(a) a low-order mesh (b) a high-order mesh

Figure 1. A low-order mesh contrasted with a high-order mesh. The key difference is that the high-order mesh has more nodes than the low-order mesh. The extra nodes enhance the geometric accuracy by tracing the path of curved edges.



(a) a low-order mesh (b) a high-order mesh

Figure 2. Before-and-after comparison of a mesh upgraded with meshCurve. The additional nodes in the high-order mesh are not visible but the surface curvature is visible.

simulation accuracy by serving as secondary interpolation points. Also, they enhance geometric accuracy by tracing the arc of curved edges.

The goal of meshCurve is to transform a mesh similar to the one on the left of figure 1 into a mesh similar to the one on the right, but in 3D. Figure 2 is a before-and-after image from meshCurve. The additional nodes are not visible in the high-order mesh, but the surface curvature is clearly discernible.

III. The Design of meshCurve

III.A. Feature Criteria

From our requirements, we assembled the following list of features for meshCurve.

- Full support for 3D unstructured CGNS meshes—any cell type in any combination.
- Support for multi-zone, multi-patch CGNS meshes, including the ability to selectively reconstruct interior patches without affecting far-field boundaries.
- Mesh processing without CAD geometry files.
- Mesh processing in the face of complex edge arrangements and sharp corners.

- Easy-to-use graphical interface, requiring minimal effort to learn and navigate.
- Interactive 3D graphics to show the mesh and to provide visual feedback after the upgrade.
- Cross-platform support—Linux, Windows, and Mac.
- Solid code base with minimal bugs, designed for maintainability and future upgrades.
- Reasonably low memory footprint and fast operation on a desktop computer.
- Minimal reliance on outside software libraries to increase control of the code and simplify design.

III.B. Design Decisions

To satisfy requirements, we made the following decisions.

- Programming would be in C++ due to the speed and flexibility of the language. We opted for C++11, the most recent version of the language, because of newly introduced optimizations and platform-independent multithreading. With multithreading, it would be possible to utilize the multiple cores of modern CPUs.
- To support our desired cross-platform graphical-user-interface (GUI), we selected the popular Qt application framework.[a] The Qt framework is a platform-independent windowing library which includes multiple user interface widgets and support libraries. Applications built with Qt adapt to the target operating system. Consequently, meshCurve appears to be an X11 application when run on Linux, a Windows application when run on Microsoft's OS, and a Macintosh application when run on OS X. Figure 3 shows an annotated view of the meshCurve user interface as it appears on Microsoft Windows. The figures elsewhere show the interface as it appears on Linux (Ubuntu) and Mac OS X.
- For interactive 3D visualization, we chose the Visualization Toolkit (VTK), an open source, cross-platform 3D visualization system created by Kitware.[5] VTK wraps the underlying graphics APIs, simplifying cross-platform development while providing a capable visualization environment.
- For fast linear algebra, we decided to use the high-speed Armadillo C++ library.[6] Armadillo interfaces platform-specific LAPACK and BLAS libraries to provide vectorized mathematics and fast matrix routines. Its syntax is based on Matlab, making mathematical programming both simple and efficient.
- We constructed our own mesh database. Building our own database provided freedom to optimize for specific needs, while also granting flexibility to design for future extensibility.

IV. User Workflow

Figure 4 diagrams the mesh-processing workflow from the user's perspective. The first step of the workflow is the loading of a mesh file, either CGNS or Gmsh format. The loaded mesh displays as a 3D view—rotatable, zoomable, and translatable via the mouse. The view can optionally be set to wireframe or opaque solid, and separate patches within the mesh can be drawn individually. Also, surface nodes can be displayed on the mesh. Viewing the nodes is especially useful as a final verification after the upgrade has completed.

The next step is to select patches to process. During the upgrade, the selected patches will be given curvature. For efficiency, unselected patches will not be given curvature although they will be given high-order nodes. Figure 5 illustrates patch selection, showing a fighter jet mesh with wing and tail patches selected. Green highlight marks the selection.

Next comes the identification of feature curves, such as the rim of an airplane canopy or the sharp edge of a box. Feature curves define the geometry, so it is important that meshCurve know the location of the curves within the mesh. Every feature curve may be manually flagged edge-by-edge, via the mouse. Alternatively, the curves can be flagged automatically by way of meshCurve's own feature-curve detection engine. Employing discontinuity detection algorithms developed by Jiao and Bayyana,[7] the feature-curve detection engine can identify curves even when they cross nearly flat portions of the mesh. After auto-detection, the mouse can be used to fine tune the result.

[a] http://www.qt.io

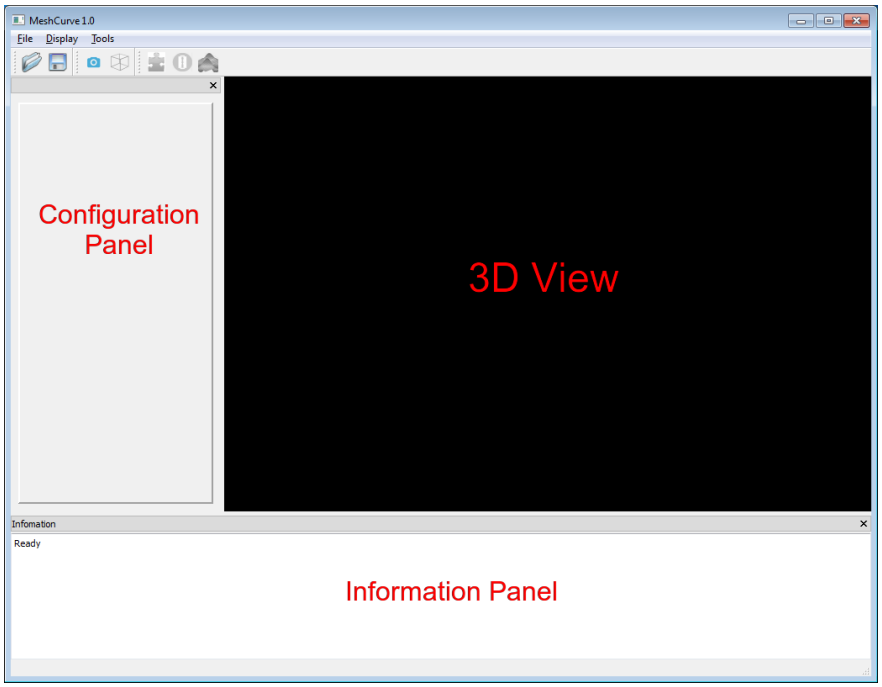


Figure 3. The meshCurve graphical interface, as it appears on Microsoft Windows 7. By default, three panels are visible: 1) 3D viewing, 2) configuration setting, and 3) information display. The information panel and the configuration panel can be hidden to increase the size of the 3D viewport. A row of toolbar buttons and menus along the top provide options to the user.

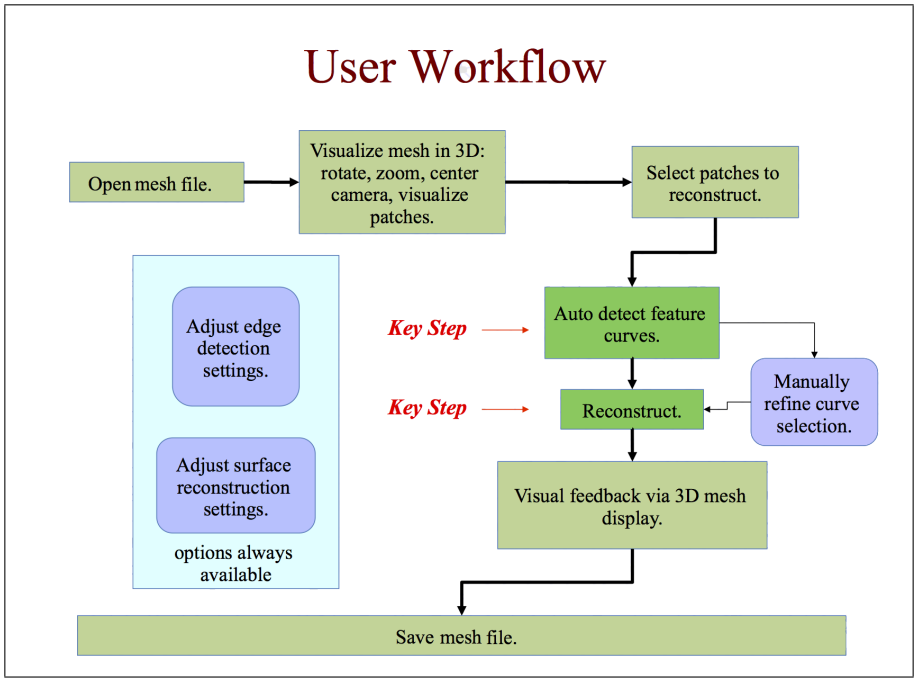


Figure 4. The mesh upgrade workflow from the user's perspective.

Once feature curves have been flagged, upgrading can commence. During the upgrade, surface geometry is approximated by a continuous 2D polynomial, which is used to position high-order nodes on the surface, giving the surface curvature. If the curvature is too great, the cells abutting the boundary will intersect. To prevent intersections, meshCurve can bend the edges of interior cells to coincide with the boundary. This bending action is applied by default but may be turned off by a checkbox in the configuration panel for surface reconstruction, as a way to speed calculation. Not all meshes need to have their interior deformed in this way.

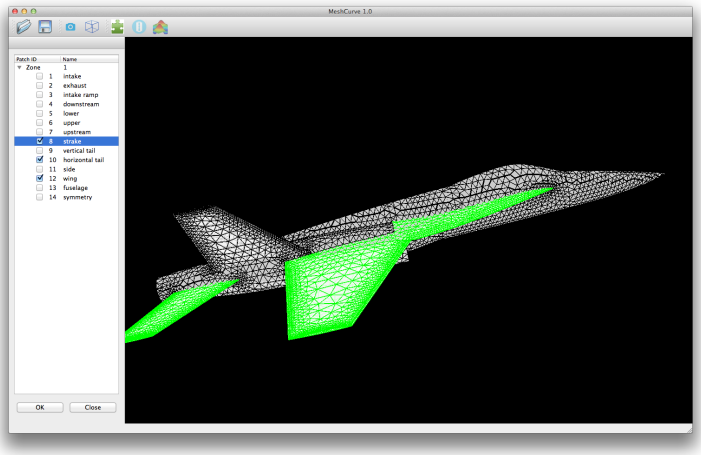


Figure 5. Screenshot of the program meshCurve, illustrating patch selection. The green portions of the mesh are the patches that have been selected for processing.

At the completion of the upgrade, the 3D view updates to show the new high-order mesh. The user can then save the mesh as a new CGNS file.

V. Case Studies

Three case studies are presented to illustrate the capability of meshCurve. For each case study, a mesh is upgraded from linear to 2nd order (i.e. low to high-order). A picture of the mesh is presented, rendered with meshCurve. A table follows, summarizing key information to demonstrate performance. The table lists:

1. the total number of cells
2. the total number of nodes
3. the fraction of patches given curvature
4. the time it took to process the mesh
5. whether or not interior deformation was activated
6. minimum and maximum Jacobians of the original low-order mesh
7. minimum and maximum Jacobians of the new high-order mesh

Processing was done on a Macintosh computer with a 2.7 GHz Intel i5 processor (quad core), running Mac OS X.

The Jacobians are a check for the presence of intersecting and/or inverted cells, both of which could be problematic for simulations. Negative Jacobians indicate trouble. Only one of the case studies encountered a negative Jacobian, and this occurrence was remedied by interior deformation. For the other case studies, interior deformation had little affect. Since the timing calculations show that interior deformation can be the most time consuming step, it is best to apply interior deformation only when it is truly needed. meshCurve applies it by default, assuming the worst, but interior deformation can be easily unselected. It should be noted that the interior deformation algorithm isn't guaranteed to prevent negative Jacobians; it merely reduces their likelihood. For example, negative Jacobians can occur if boundary cells are borderline degenerate.

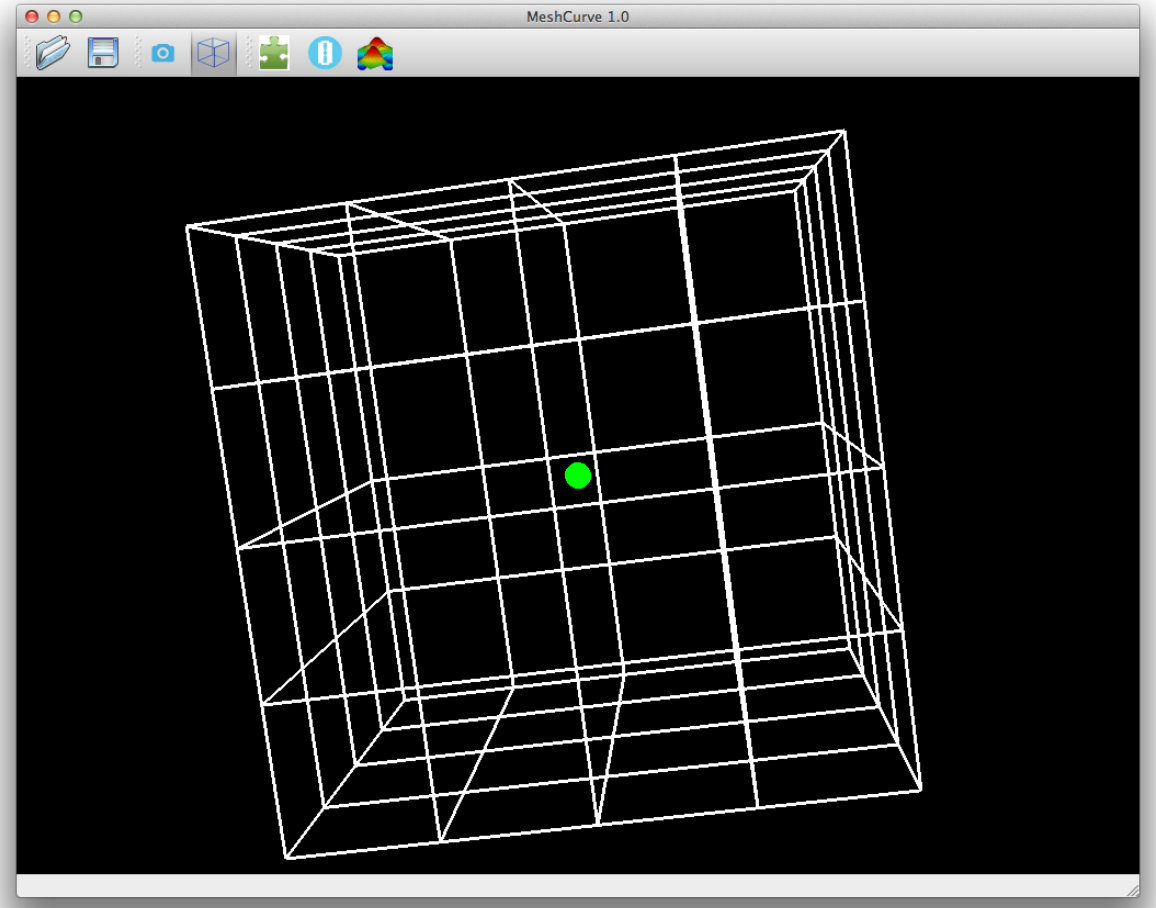


Figure 6. Case study: meshCurve was used to process the above mesh, upgrading it from linear to 2nd order. The mesh represents a sphere inside a square shaped domain. There are two patches in the mesh, one representing the surface of the sphere (highlighted green) and the second representing the outer surface of the domain (shown but not highlighted). The green areas of the mesh were given curvature during the upgrade. Table 1 presents statistics for the before and after mesh.

Table 1. Statistics for figure 6 mesh: meshCurve was used to upgrade the originally linear mesh to 2nd order. The program was executed on a Macintosh computer with a 2.7 GHz Intel Core i5 processor running Mac OS X.

# cells	# nodes	patches: recon./total	interior defor.	time	original Jacobian [low — high]	new Jacobian [low — high]
480	588	1 / 2	no	0.08 sec	[0.0131343 – 192.609]	[0.0129203 – 192.613]
480	588	1 / 2	yes	0.11 sec	[0.0131343 – 192.609]	[0.0129203 – 192.613]

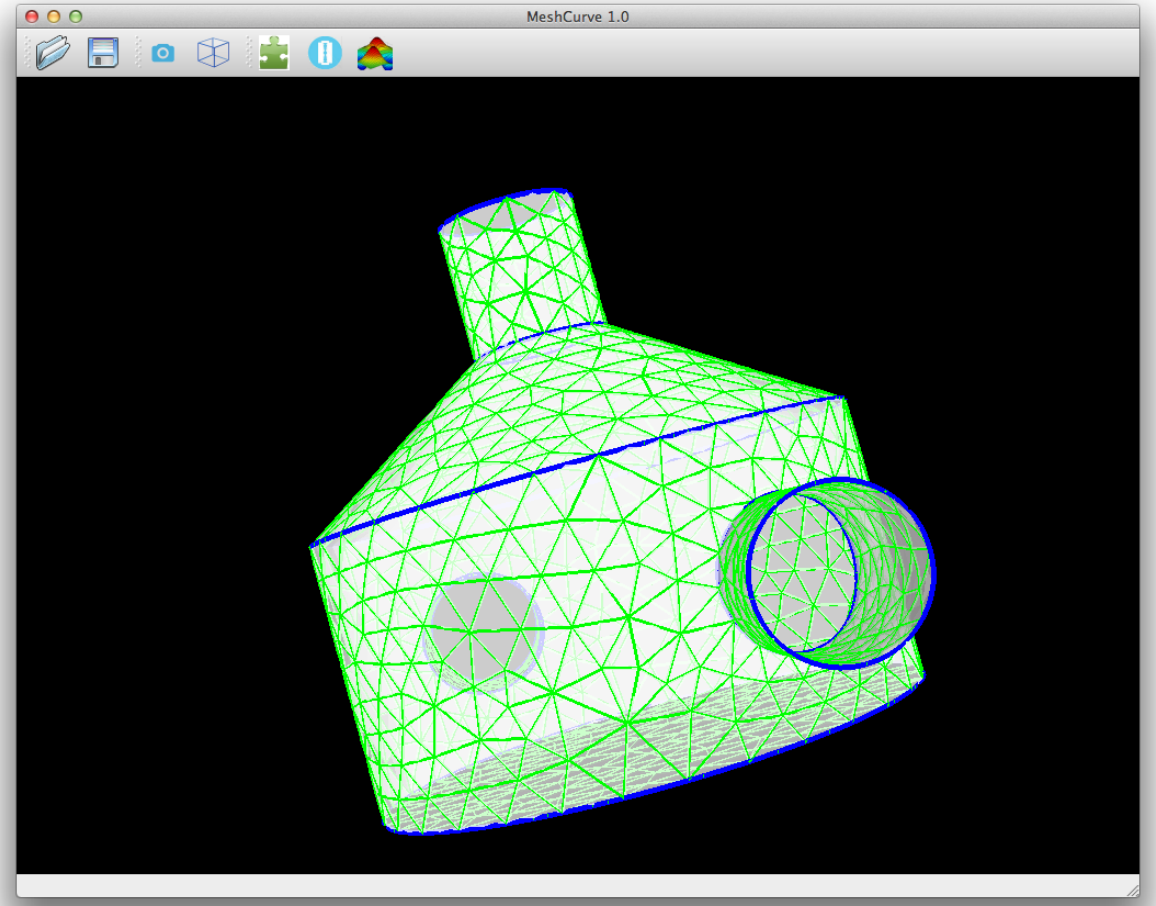


Figure 7. Case study: meshCurve was used to process the above mesh, upgrading it from linear to 2nd order. The mesh represents a static mixer. There are four patches in the mesh, one of which is shown (highlighted green). The other three patches have been hidden from view to make the visualization easier to interpret. Curvature imparted to the high-order mesh is clearly discernible around the rim of the inlet spout. Table 2 presents statistics for the before and after mesh.

Table 2. Statistics for figure 7 mesh: meshCurve was used to upgrade the originally linear mesh to 2nd order. The program was executed on a Macintosh computer with a 2.7 GHz Intel Core i5 processor running Mac OS X.

# cells	# nodes	patches: recon./total	interior defor.	time	original Jacobian [low — high]	new Jacobian [low — high]
13,761	2,786	4 / 4	no	15.3 sec	[0.00164224 – 0.0418735]	[0.000572914 – 0.043354]
13,761	2,786	4 / 4	yes	16.1 sec	[0.00164224 – 0.0418735]	[0.000572914 – 0.0429421]

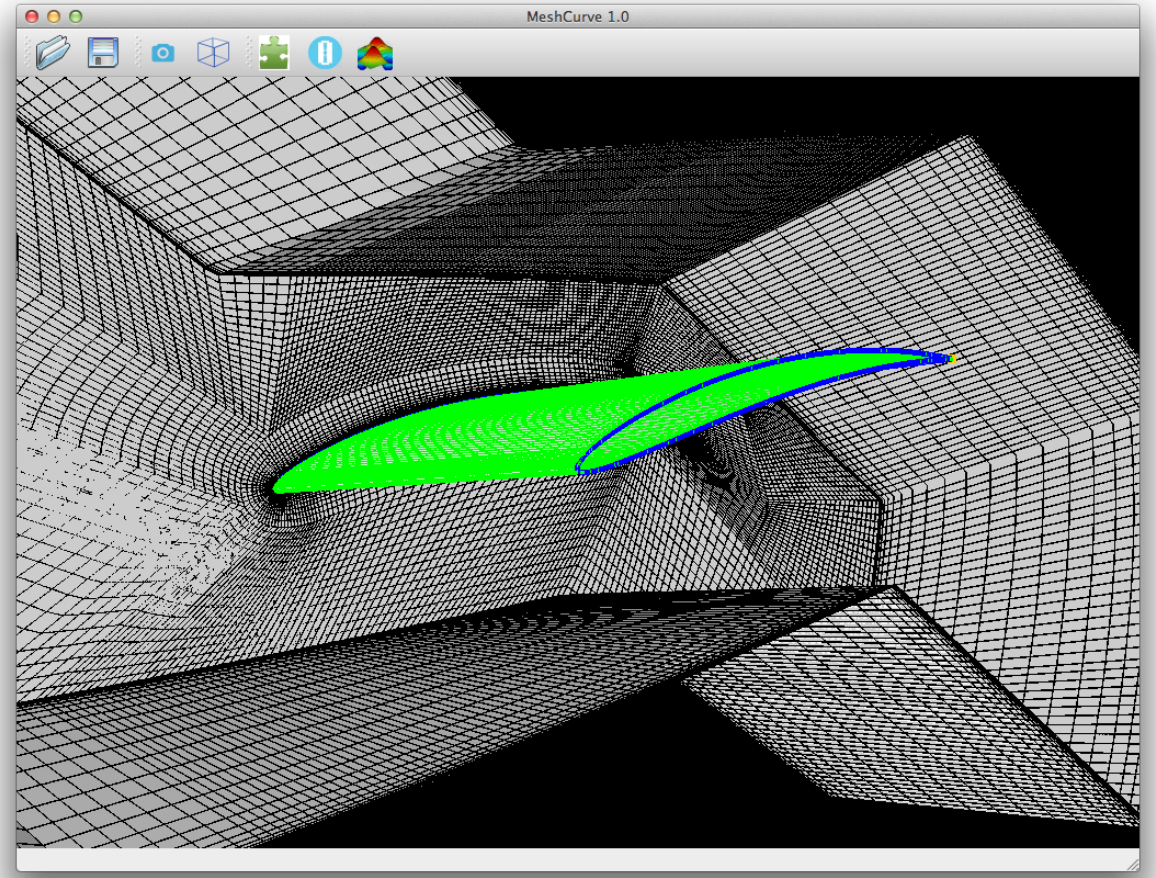


Figure 8. Case study: meshCurve was used to process the above mesh, upgrading it from linear to 2nd order. The mesh represents an airfoil. There are seven patches in the mesh, one representing the surface of the airfoil (highlighted green) and six representing the outer surfaces of the simulation domain. In the above view, the front patch has been hidden to reveal the interior airfoil. Table 3 presents statistics for the before and after mesh.

Table 3. Statistics for figure 8 mesh, representing an airfoil: meshCurve was used to upgrade the originally linear mesh to 2nd order. The program was executed on a Macintosh computer with a 2.7 GHz Intel Core i5 processor running Mac OS X.

# cells	# nodes	patches: recon./total	interior defor.	time	original Jacobian [low — high]	new Jacobian [low — high]
455,988	478,840	1 / 7	no	14.2 sec	[3.00E-016 – 1.31E-008]	[-4.54E-018 – 1.31E-008]
455,988	478,840	1 / 7	yes	6.6 min	[3.00E-016 – 1.31E-008]	[3.00E-016 – 1.31E-008]

VI. Internal Architecture Overview

The mesh database is the core of meshCurve. It manages all information associated with the mesh, serving as both the information storage facility and the conduit through which the various components of meshCurve communicate. There are four external classes which interface the database to provide additional functionality. These classes perform the following tasks: 1) file input/output, 2) feature curve detection, 3) surface reconstruction/interior deformation, and 4) 3D visualization. The code for the user interface wraps around the database and accompanying classes. It exposes the internal switches for the user to manipulate. The modular structure of the code base and its object-oriented design make the program easy to edit. Figure 9 illustrates the flow of data.

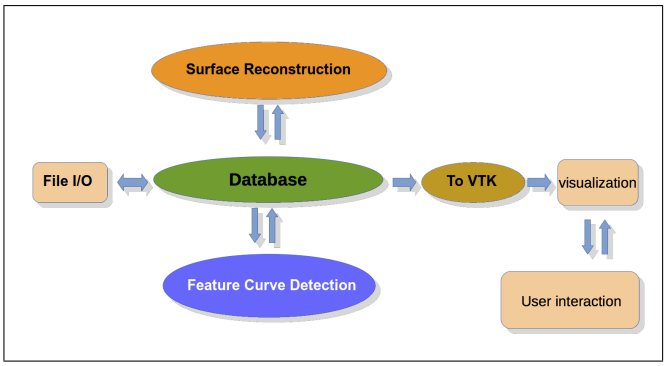


Figure 9. The internal data flow for the program meshCurve. The database is the core of the program, serving as storage facility and communication hub. Four external classes interface the database to accomplish the four tasks 1) File input/output, 2) 3D visualization, 3) feature curve detection, and 4) surface reconstruction.

The mesh database is built for maximum flexibility. It is designed with no assumption about the cell types or node orderings stored within. These definitions are placed elsewhere, as a configuration table, which makes updating the definitions extremely easy. A second example of flexibility is the storage scheme for element connectivities, which uses both forward and backward references to not only answer questions such as "Which nodes occupy this cell?" but also the inverse, "Which cells contain this node?" Furthermore, the database provides a mechanism to cycle through elements as a list, as if the data were stored as an array. The database also provides a mechanism to jump through elements neighbor-to-neighbor, as if the mesh were stored as a connected graph.

VI.A. Key Algorithms: feature curve detection

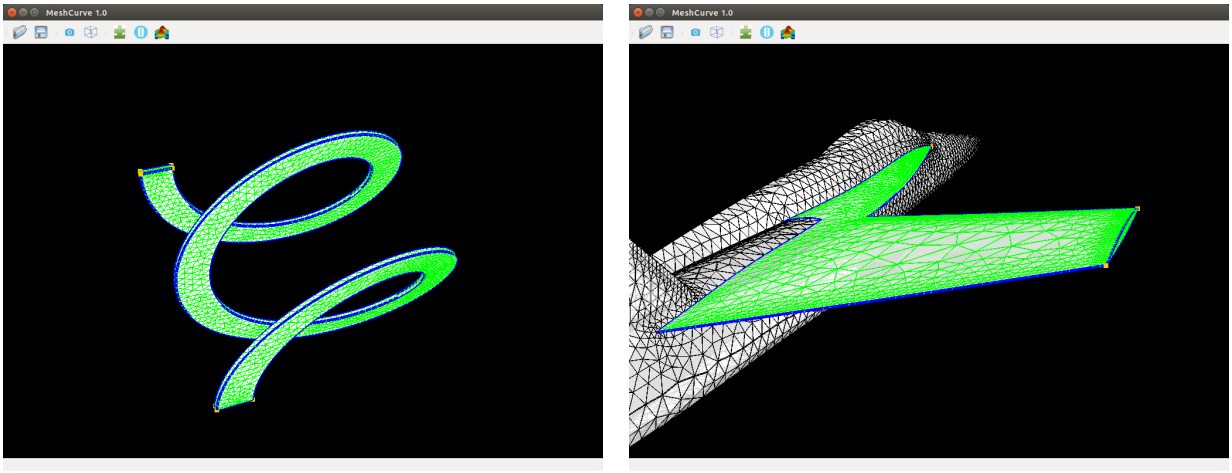
To find feature curves, meshCurve employs an algorithm proposed by Jiao and Bayyana, which identifies the curves probabilistically, using a combination of several topological metrics.[7] The algorithm proceeds through several steps. Surface nodes and edges are flagged as candidate discontinuities if the local dihedral angle, ridge direction, and angle defect fall within a predefined range. Then the candidate discontinuities are linked into chains. The list of chains is filtered, and each chain receives a classification based on the properties of its starting and ending nodes. As the algorithm progresses, the list is narrowed until only the most likely candidates remain. In the end, the remaining chains are classified as feature curves. Figure 10 illustrates the capabilities of the method, implemented in meshCurve. Blue lines represent detected curves; orange dots represent detected corners.

VI.B. Key Algorithms: surface reconstruction

For high-order surface reconstruction, meshCurve employs a variation of Weighted Averaging of Local Fittings (WALF), developed by Jiao and Wang.[8] WALF reconstructs a mesh surface by averaging a set of locally fitted 2D Taylor polynomials. In the original algorithm, the polynomials are fit to every node on the surface. In our version, the polynomials are fit to every *face*. The use of faces as the location-of-fit reduces the required number of averages. Also the use of face-centered-fits facilitates the treatment of feature curves as boundaries, thereby preventing the feature curves from being smoothed away. An iterative solver, initialized to zero, helps reduce oscillations.

VI.C. Key Algorithms: interior deformation

To deform the interior elements, meshCurve employs an interpolation scheme developed by Luke, Collins, and Blades originally designed for the simulation of fluid structure interaction.[9] According to this method, each interior node moves in accordance with a weighted average of the boundary node motion. Every boundary node is weighted inverse to its distance and in proportion to the area it commands.



(a) (b)

Figure 10. Screenshots of the program meshCurve, illustrating automatic feature curve detection. Blue lines mark feature curves identified by the program. In the left subfigure, the edges of a spring have been identified as feature curves. In the right subfigure, feature detection has been limited to the portion of the mesh isolated for processing. The mesh represents a fighter jet with a single wing selected. Curve detection has marked the wing's boundary as a feature curve. Interior curves and corners have also been identified.

VII. Conclusions

This paper has provided an overview of meshCurve, a new low-to-high order mesh conversion utility produced by the University of Kansas CFD Lab. This software aims to be a cross-platform, simple-to-use system for upgrading linear meshes to high-order without the need for a CAD geometry file. Since existing software does not provide such capability, we hope that meshCurve will be useful to researchers and high-order CFD practitioners who need high-order meshes for their work.

Acknowledgments

This material is based upon work supported by NASA and also by the National Science Foundation Graduate Research Fellowship Program under Grant No. NSF0064451.

References

[1]Wang, Z., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H., et al., "High-order CFD methods: current status and perspective," *International Journal for Numerical Methods in Fluids*, Vol. 72, No. 8, 2013, pp. 811–845.

[2]Luo, X., Shephard, M. S., and Remacle, J.-F., "The influence of geometric approximation on the accuracy of high order methods," *Rensselaer SCOREC report*, Vol. 1, 2001.

[3]Xie, Z. Q., Sevilla, R., Hassan, O., and Morgan, K., "The generation of arbitrary order curved meshes for 3D finite element analysis," *Computational Mechanics*, Vol. 51, No. 3, 2013, pp. 361–374.

[4]Wang, Z., "High-order methods for the Euler and Navier–Stokes equations on unstructured grids," *Progress in Aerospace Sciences*, Vol. 43, No. 1, 2007, pp. 1–41.

[5]Schroeder, W., Martin, K., and Lorensen, B., *The Visualization Toolkit*, Kitware, Inc., United States of America, 4th ed., 2006.

[6]Sanderson, C., "Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments," 2010.

[7]Jiao, X. and Bayyana, N. R., "Identification of C1 and C2 discontinuities for surface meshes in CAD," *Computer-Aided Design*, Vol. 40, No. 2, 2008, pp. 160–175.

[8]Jiao, X. and Wang, D., "Reconstructing high-order surfaces for meshing," *Engineering with Computers*, Vol. 28, No. 4, 2012, pp. 361–373.

[9]Luke, E., Collins, E., and Blades, E., "A fast mesh deformation method using explicit interpolation," *Journal of Computational Physics*, Vol. 231, No. 2, 2012, pp. 586–601.