HIP-17-050

Parametric Representation and Refinement of Structured Meshes

Cameron Mackintosh[1,2,3,4], Konstantinos Vogiatzis[3,4]

[1]*HPC Internship Program, DoD HPC Modernization Program*

[2]*Carnegie Institute of Technology, Carnegie Mellon University, Pittsburgh, PA 15213*

[3]*PETTT/Engility Corp., DoD HPC Modernization Program*

[4]*DoD Supercomputing Research Center, U.S. Air Force Research Labs, Wright-Patterson Air Force Base, Ohio 45433*

Abstract

This paper introduces procedures to represent structured meshes with parametric surfaces. These representations have significantly smaller sizes than the represented mesh, reducing memory and disk space overhead of large simulations without a loss in fidelity. Meshes of any size and point distribution may then be generated in parallel from these representations, permitting arbitrary refinement.

Introduction

External aerodynamics simulations often require excessively large meshes. Commercial mesh generation software, the software responsible for generating meshes from CAD files, is inherently limited in maximum mesh sizes due to memory and disk space requirements. This software can't always generate sufficiently fine meshes for a simulation. This software, however, can usually generate coarse meshes that adequately represent the CAD geometry. We introduce procedures to represent these coarse meshes with parametric surfaces. These representations are even smaller than the coarse meshes, while also sufficiently representing the geometry. We then introduce procedures to generate arbitrarily fine meshes from these parametric representations. These procedures are computationally inexpensive and trivially parallelized, allowing the prohibitive memory and disk space requirements to be distributed across many nodes. This paper's procedures strictly apply to surfaces, however they can be trivially extended to volumes. These procedures are then implemented in Fortran 2003, with the purpose of being added to the Ablation Fluid Structure Interaction (AFSI) library.

Methods

A. Bézier Surfaces

Pierre Bézier developed Bézier curves in 1962 in order to represent automobile bodies using a precise mathematical definition, rather than the imprecise master molds used at the time. We use Bézier surfaces as our parametric surfaces.

A Bézier curve is defined in entirety by its vectors of control points; any point's real space coordinates $x, y, z \in \mathbb{R}$ can be calculated from these control points. Points are represented by their computational space parameter $u \in [0,1]$. Similarly, a Bézier surface is defined in entirety by its matrices of control points; any point's real space coordinates $x, y, z \in \mathbb{R}$ can be calculated from these control points. Points on the surface are represented by their computational space parameters $u, v \in [0,1]$.

The Bézier equations for three dimensional surfaces are as follows.

$$x(u,v) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} (B_{n_u,i}(u) \cdot B_{n_v,j}(v) \cdot \mathbf{X_{i,j}})$$

$$y(u,v) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} (B_{n_u,i}(u) \cdot B_{n_v,j}(v) \cdot \mathbf{Y_{i,j}})$$

$$z(u,v) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} (B_{n_u,i}(u) \cdot B_{n_v,j}(v) \cdot \mathbf{Z_{i,j}})$$

n_u is the order in the u dimension

n_v is the order in the v dimension

$B_{n,i} : [0,1] \to \mathbb{R}$ is the Bernstein coefficient function defined by $B_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}$

$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ are the control point matrices with dimensions $(n_u + 1) \times (n_v + 1)$

B. De Casteljau Algorithm

Real space coordinates may be calculated by applying the Bézier equations to desired $u, v \in [0,1]$ parameters. However, Bernstein coefficients introduce poor computational complexity and numerical instability at high orders. Thus, we introduce the de Casteljau algorithm, a fast and robust algorithm for evaluating the Bézier equations.

The de Casteljau algorithm for three-dimensional curves is as follows.

$$x_i^k(t) = \begin{cases} (1-t)x_i^{k-1} + t x_{i+1}^{k-1}, & \text{if } k > 0 \\ \mathbf{x_i}, & \text{if } k = 0 \end{cases}$$

$$y_i^k(t) = \begin{cases} (1-t)y_i^{k-1} + t y_{i+1}^{k-1}, & \text{if } k > 0 \\ \mathbf{y_i}, & \text{if } k = 0 \end{cases}$$

$$z_i^k(t) = \begin{cases} (1-t)z_i^{k-1} + t z_{i+1}^{k-1}, & \text{if } k > 0 \\ \mathbf{z_i}, & \text{if } k = 0 \end{cases}$$

$$x(t) = x_0^n(t)$$

$$y(t) = y_0^n(t)$$

$$z(t) = z_0^n(t)$$

n is the number of control points

$\mathbf{x}, \mathbf{y}, \mathbf{z}$ are the control point vectors with dimensions $1 \times (n+1)$

The de Casteljau algorithm may be extended to three-dimensional surfaces. We first apply the de Casteljau algorithm to each row of the control point matrices for a given computational-space parameter $u \in [0,1]$ to obtain $n_v + 1$ new points' real-space coordinates. We then apply the de Casteljau algorithm to these new points for a given computational-space parameter $v \in [0,1]$ to obtain the final point's real-space coordinates.

C. Representing Meshes with Bézier Surfaces

We can fit a Bézier surface to an initial structured mesh by solving the Bézier equations as an overdetermined system for the control points. The system of equations is as follows.

$$\mathbf{x}_{\text{init}} = \mathbf{B}\mathbf{x}_{\text{cp}} \qquad \mathbf{y}_{\text{init}} = \mathbf{B}\mathbf{y}_{\text{cp}} \qquad \mathbf{z}_{\text{init}} = \mathbf{B}\mathbf{z}_{\text{cp}}$$

$n_{\text{p}} = n_{\text{rows}} \times n_{\text{cols}}$ is the number of points

$n_{\text{cp}} = n_{\text{u}} \times n_{\text{v}}$ is the number of control points

$\mathbf{x}_{\text{init}}, \mathbf{y}_{\text{init}}, \mathbf{z}_{\text{init}}$ are the real-space coordinate vectors with dimensions $1 \times n_{\text{p}}$

$\mathbf{x}_{\text{cp}}, \mathbf{y}_{\text{cp}}, \mathbf{z}_{\text{cp}}$ are control point coordinate vectors with dimensions $1 \times n_{\text{cp}}$

$\mathbf{B}$ is the Bernstein coefficient matrix with dimensions $n_{\text{p}} \times n_{\text{cp}}$

We first parameterize the mesh (i.e. assign each point computational-space parameters) by approximated arc length to preserve, as much as possible, the real-space distribution in computational space. The arc length equations are as follows.

$$\mathbf{u}_{\mathbf{i},\mathbf{j}} = \frac{\sum_{k=1}^{i} \left(\mathbf{x}_{k,j} - \mathbf{x}_{k-1,j} \right)}{\sum_{k=1}^{n_u} \left(\mathbf{x}_{k,j} - \mathbf{x}_{k-1,j} \right)} \qquad \mathbf{v}_{\mathbf{i},\mathbf{j}} = \frac{\sum_{k=1}^{j} \left(\mathbf{x}_{i,k} - \mathbf{x}_{i,k-1} \right)}{\sum_{k=1}^{n_v} \left(\mathbf{x}_{i,k} - \mathbf{x}_{i,k-1} \right)}$$

We then calculate the Bernstein coefficient matrix, as follows.

$$B_{(i \times n_{\text{cols}} + j),(k \times n_v + l)} = \binom{n_u}{k} \mathbf{u}_{i,j}^k (1 - \mathbf{u}_{i,j})^{n_u - k} \times \binom{n_v}{l} \mathbf{v}_{i,j}^l (1 - \mathbf{v}_{i,j})^{n_v - l}$$

i is the real-space point's row index

j is the real-space point's column index

k is the control point's row index

l is the control point's column index

We finally approximate the solution to the overdetermined system for the control points using the least squares method. The initial structured mesh is thus represented by a matrix of control points.

D. Generating New Meshes

We can generate new meshes of any point distribution. We apply the de Casteljau algorithm with any computational-space distribution ($u, v \in [0,1]$) to generate a new mesh. However, the new mesh's real-space distribution ($x, y, z \in \mathbb{R}$) does not quite preserve the computational-space distribution used to generate it. The initial mesh was parameterized by *approximated* arc length, as the mesh is an approximation of a surface. The Bézier surface's computational-space distribution thus preserves the real-space distribution of the initial mesh's approximation of a surface, not the real-space distribution of the surface itself. Thus the new mesh's real-space distribution does not quite preserve the computational-space distribution used to generate it; we call this phenomenon arc length parameterization error.

To solve this problem, we introduce arc length tables. The larger the initial mesh is, the better it approximates the intended surface, and thus the smaller the arc length parameterization error is. We exploit this triviality by generating a large intermediate mesh with a uniform computational-space distribution; we maintain a table mapping the generating computational-space values (those used to generate a point's real-space value) to the actual computational-space values (those calculated from a point's real-space value) for each point. We calculate the actual values of each point by applying the definition of arc length, as follows.

$$\mathbf{u}_{\text{actual}_{i,j}} = \frac{L_u(\mathbf{u}_{\text{gen}_{i,j}}, \mathbf{v}_{\text{gen}_{i,j}})}{L_u(1, \mathbf{v}_{\text{gen}_{i,j}})} \qquad \mathbf{v}_{\text{actual}_{i,j}} = \frac{L_v(\mathbf{u}_{\text{gen}_{i,j}}, \mathbf{v}_{\text{gen}_{i,j}})}{L_v(\mathbf{u}_{\text{gen}_{i,j}}, 1)}$$

$$L_u(u,v) = \int_0^u \sqrt{\left(\frac{\partial x(u,v)}{\partial u}\right)^2 + \left(\frac{\partial y(u,v)}{\partial u}\right)^2 + \left(\frac{\partial z(u,v)}{\partial u}\right)^2}\, du$$

$$L_v(u,v) = \int_0^v \sqrt{\left(\frac{\partial x(u,v)}{\partial v}\right)^2 + \left(\frac{\partial y(u,v)}{\partial v}\right)^2 + \left(\frac{\partial z(u,v)}{\partial v}\right)^2}\, dv$$

The partial derivatives are calculated from either the de Casteljau algorithm or Bézier equations.

We can now generate new meshes such that the real-space distribution of the new mesh preserves the generating computational-space distribution. We binary search the arc length table's actual computational-space

values for our desired generating computational-space values, linearly interpolating as needed. We create a new computational-space distribution, replacing our desired distribution's values with their corresponding actual values. We now can apply the de Casteljau algorithm with our new computational-space distribution to generate a new mesh.

A large (and thus computationally intense) arc length table is necessary to sufficiently decrease arc length parametrization error as the table is queried via binary search and linear interpolation. We can instead query the table by fitting a Bézier curve to it and applying the de Casteljau algorithm. This approach permits us to drastically reduce the size of the table and thus the computational expense.

Results

We apply our procedures to a test case in order to evaluate their accuracy. We measure how the new meshes' real-space distributions correspond to their generating computational-space distributions by generating our new meshes with uniform generating distributions and calculating the averages of each row's and each column's standard deviation of the segment lengths. We measure how the new meshes approximate the CAD geometry by summing the squared distance between each point's real-space location and its corresponding CAD geometry real-space location. We measure a set of procedures' computational complexity by averaging the user-space runtime on a specific machine and compiler setting (in this case we will average 100 executions on a Thunder compute node with Intel -O2 -Ofast optimizations).

Our CAD geometry is a portion of a torus. We first generate a coarse mesh with one block and dimensions 20×20. We parameterize it by approximated arc length. From this, we generate a fine mesh with one block, mesh dimensions 100×100, fifth order, arc length table dimensions 50×50, and a uniform generating computational-space distribution. We will calculate the discussed statistics for several combinations of the discussed procedures.

	Average Segment Length Standard Deviation	Sum of Squared Distance to Geometry	Average User-space Runtime (seconds)
Bézier equations and no arc length table	3.015E-02	1.438E-02	2.966E-03
de Casteljau algorithm and no arc length table	3.042E-02	1.372E-02	2.856E-03
Bézier equations and linear arc length table	5.684E-07	1.248E-02	3.115E-02
de Casteljau algorithm and linear arc length table	5.44E-07	1.113E-02	3.107E-02
Bézier equations and fitted arc length table	2.446E-10	1.348E-02	6.820E-02
de Casteljau algorithm and fitted arc length table	2.239E-10	1.275E-02	5.304E-02

The de Casteljau algorithm is slightly faster and slightly more accurate than the Bézier equations in almost every case; the final code implements the de Casteljau algorithm. The average standard deviation of segment length dramatically decreases and the run-time slightly increases after implementing arc length tables; the final code implements arc length tables. Querying the arc length table by Bézier fit is slightly slower and slightly more accurate than querying the table by linear interpolation; the final code implements query by Bézier fit. The initial coarse mesh and new fine mesh, generated by the final code, are pictured below.

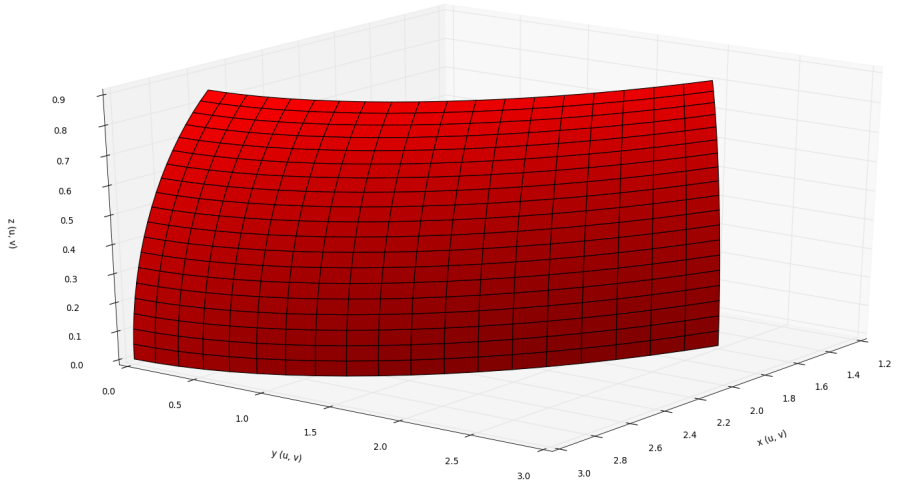


Fig. 1. Initial coarse mesh.

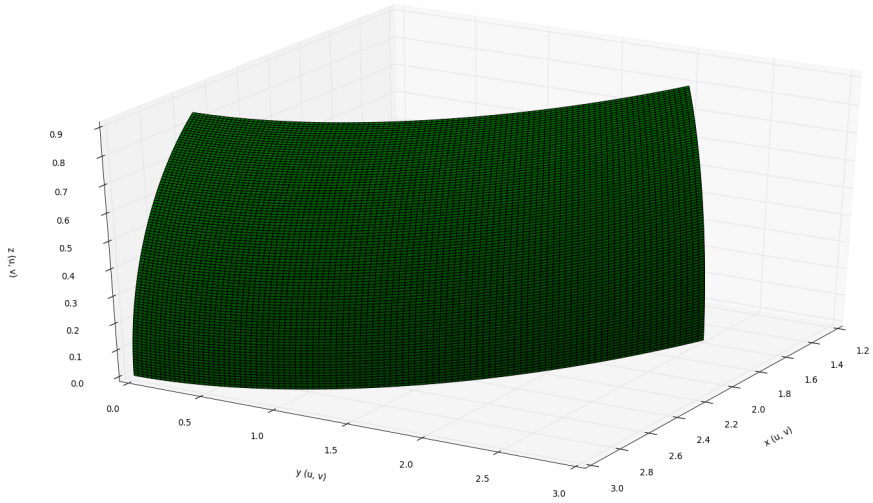
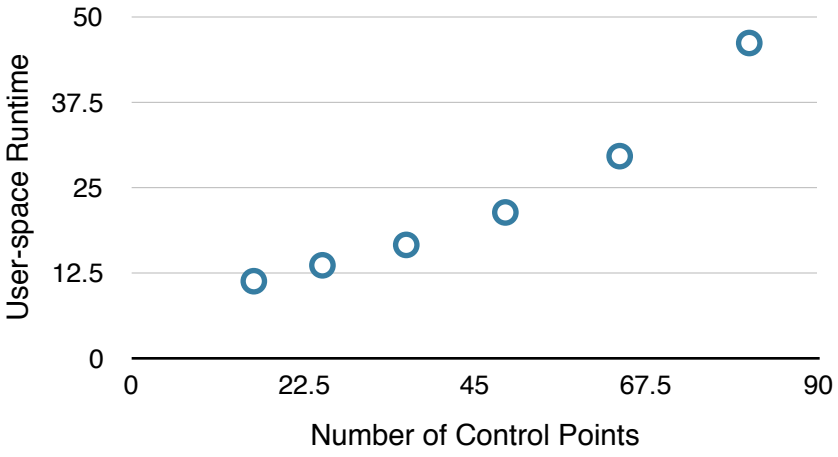
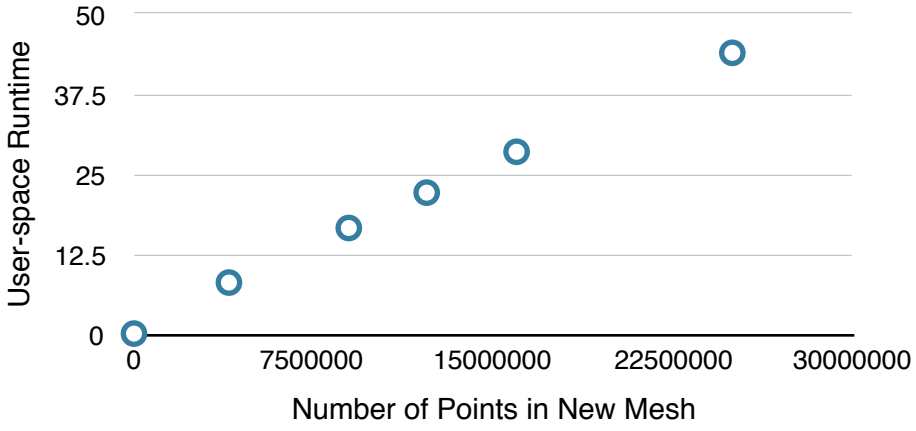


Fig. 2. New fine mesh.

We lastly verify our final code's computational time complexity, ensuring it matches the complexity of our procedures. In respect to the number of control points, the complexity is an element of $O\left(n^2\right)$, as pictured

below and as desired. In respect to the number of points in the new mesh, the complexity is an element of $O(n)$ as pictured below and as desired.





Conclusions

These procedures dramatically reduce the prohibitive memory and disk-space overhead of simulations running on multi-billion element meshes. Representing these meshes as Bézier surfaces reduces their size by orders of magnitude; meshes are stored as thousands or millions of control points rather than billions of elements. The procedures employed are trivially parallelized across the meshes' blocks, making mesh refinement and deformation computationally inexpensive. External aerodynamic simulations can use these

procedures to efficiently represent and deform excessively large meshes. Further, these procedures form the basis for other applications: element clustering, adaptive mesh refinement, multi-block interfacing, and so forth.

Impact of Summer Experience

I'm an upcoming junior studying electrical and computer engineering at Carnegie Mellon University. I participated in the Wright Scholars internship program during the summers of 2014 and 2015, interfacing an Arduino with an HPC compute node and integrating the ParaView Catalyst in-situ visualization toolkit with the US3D computational fluid dynamics solver, respectively.

My experiences this summer with the HPC community have been extraordinary. I was introduced to new topics, such as computational mathematics and computer geometry. I was also able to refine my existing knowledge of many topics, such as GDB, GNU Make, and Fortran 2003. These skills will find practical use throughout my career and likely influence future career decisions.

Acknowledgments

This research was sponsored by the High Performance Computing Modernization Program (HPCMP), the HPC Internship Program (HIP), and Engility Corporation. I'd like to thank my mentor, Konstantinos Vogiatzis for providing me with this opportunity and his guidance. I'd also like to thank Ryan Gosse for his guidance.