Title:
GPU-Accelerated Post-Processing and Animated Volume Rendering of Isogeometric Analysis Results

Authors:
Harshil Shah, harshil@iastate.edu, Iowa State University
Xin Huang, ryan125@iastate.edu, Iowa State University
Onur Rauf Bingol, orbingol@iastate.edu, Iowa State University
Manoj R. Rajanna, manoj95@iastate.edu, Iowa State University
Adarsh Krishnamurthy, adarsh@iastate.edu, Iowa State University

Keywords:
Volumetric spline, Ray intersection, Isogeometric analysis post-processing, GPU-accelerated geometric algorithm, Animated volume rendering, Ray-casting, FSI, Biomechanics

DOI: 10.14733/cadconfP.2021.177-181

Introduction:

Isogeometric analysis (IGA) [1] has enabled better CAD integration by using the same spline representations (Non-Uniform Rational B-Splines, NURBS) for modeling and analysis. Traditionally, the finite element analysis results are visualized by creating a texture map of the property of interest and superimposing them over the boundary representation (B-rep) model or the mesh. This technique cannot be directly used to render internal quantities of interest without computationally intensive sectioning and remapping of the textures, which does not allow for interactive interrogation of the analysis results.

Ray-casting is usually used to render volume data and is computationally more intensive than rasterization. Performing ray casting with volumetric splines used in IGA is still computationally intensive to perform interactively. In this work, we first voxelize the isogeometric mesh using a GPU-accelerated ray intersection algorithm for cubic-Bézier volumes to convert volumetric splines to time-varying voxelized data structures. We then use GPU ray casting to volume-render the time frames of the simulation. This approach leads to interactive volume rendering of the results of dynamic IGA simulations, allowing for real-time manipulation in the 3D environment.

One of the early algorithms to compute ray intersection with surfaces can be found in [2], where they reduce the ray-surface intersection to computing roots of a polynomial. [4] presented the Newton iteration technique that uses subdivisions of a surface to compute all the roots for the ray-surface combinations. [3] developed a method to decompose the NURBS surfaces into Bézier patches and then perform the triangulation of the surface for rendering. However, most of these previous approaches were not fast enough for animated volume rendering or mainly rendered the analysis results on surfaces as textures.

To voxelize the IGA models, we first decompose the NURBS elements into Bézier elements by performing Bézier extraction. This is required to deal with the non-uniformity of the knot vector in a general NURBS element. We then perform a modified ray intersection test with the six Bézier surfaces of the element using a grid of rays. We then generate a variable density voxel model representing the analysis results using the intersection data, which is repeated for the different time frames of the analysis. The

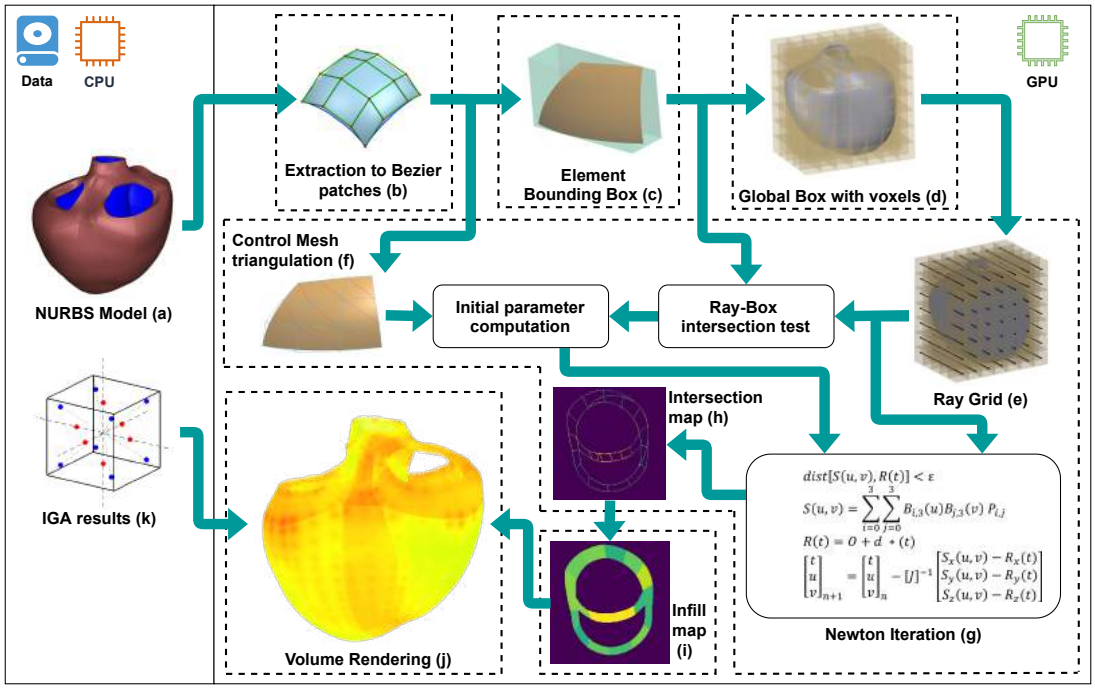

Fig. 1: Different steps of our approach for animated volume rendering of isogeometric analysis results.

complete time-series data is then rendered using GPU-accelerated ray casting. This direct voxelization technique enables a detailed analysis of the simulation using interactive slicing techniques without computationally intensive post-processing. We demonstrate this approach for two biomechanics simulations—a cardiac solid mechanics model and an aorta fluid structure interaction (FSI) model.

Methodology Overview:

Our method to render the results of dynamic IGA simulations involves first voxelizing the deforming NURBS volumes. We perform GPU-accelerated ray intersection with the element surfaces using a modified Newton method (Fig. 1(g)). We compute the initial parameters of the Newton method by performing ray intersections with the surface control mesh. We generate a boundary voxel model from the intersection data(Fig. 1(h)). Once all the boundary voxels have been identified, we perform an infill operation along the ray (Fig. 1(i)). Finally, the value corresponding to the IGA results is assigned to the voxel, which is repeated for all time frames(Fig. 1(j)). After voxelizing all the frames, we use a GPU-accelerated volume rendering method to animate the results of the simulations interactively.

Ray Intersection with Bézier Elements:

Performing a ray intersection test directly on the NURBS element is computationally intensive due to the non-uniformity of the knot vector and the rational nature of the surfaces. To improve the efficiency of the process, we first extract the six NURBS surfaces of each NURBS volume element, each represented using the (u, v) parametric space. We then decompose each knot span of the NURBS surface into Bézier patches. (Fig. 1(b)). We compute a bounding box for each Bézier patch and a global bounding box for the model (Fig. 1(d)). We create a voxel grid around the model using the global bounding box based on the desired voxelization resolution (Fig. 1(e)), with the individual voxel centers as the ray origin.

We use a modified Newton method to find the intersection of the ray with the parametric surface. Finding an initial parameter closer to the solution reduces the number of iterations of the Newton method. We compute this initial parameter by finding the intersection with the control mesh of each Bézier element. We triangulate the control mesh (Fig. 1(f)) and perform a line-plane intersection. A ray can have multiple intersections with a single surface patch, which we account for by obtaining multiple unique initial parameters using the line plane intersection between a single ray and different triangles of the control mesh. We then calculate the surface parameters (u, v) and the ray parameter t of the intersection point as the initial parameters.

$$\begin{gathered} dist\ [S(u,v),\ R(t)] < \varepsilon \\ S(u,v) = \sum_{i=0}^{p_u} \sum_{j=0}^{p_v} B_{i,p_u}(u) B_{j,p_v}(v) P_{i,j} \\ R(t) = o + d * (t) \end{gathered} \tag{2.1}$$

The objective function is the distance between surface points on a Bézier element corresponding to the parameters (u, v) and point on the ray corresponding to the parameter t (Eqn. (2.1)). Here Bs are the Basis functions, and Ps are the control points of the Bézier surface. O and d are the origin and direction of the ray, respectively. The parameters are updated using the Newton method, which has a second-order convergence rate. We modified the update equation to incorporate the parametric ray (Eqn. (2.2)). To avoid missing any potential intersection point, we also perform the intersection test on the triangles formed by control points of the individual surface edges. There is a possibility that we may compute an intersection point multiple times. We make use of the ray parameter t to identify and remove these repeated intersections.

$$\begin{gathered} \begin{bmatrix} t \\ u \\ v \end{bmatrix}_{n+1} = \begin{bmatrix} t \\ u \\ v \end{bmatrix}_{n} - [J]^{-1} \begin{bmatrix} S_x(u,v) - R_x(t) \\ S_y(u,v) - R_y(t) \\ S_z(u,v) - R_z(t) \end{bmatrix} \\ where\ J = \begin{bmatrix} -\frac{dR_x(t)}{dt} & \frac{dS_x(u,v)}{du} & \frac{dS_x(u,v)}{dv} \\ -\frac{dR_y(t)}{dt} & \frac{dS_y(u,v)}{du} & \frac{dS_y(u,v)}{dv} \\ -\frac{dR_z(t)}{dt} & \frac{dS_z(u,v)}{du} & \frac{dS_z(u,v)}{dv} \end{bmatrix} \end{gathered} \tag{2.2}$$

Voxelization and Infill Operation:

Based on the intersection data, we generate a voxelized model with boundary element(Fig. 1(h)). These voxels are assigned the value of the element to which the intersection point belongs. A single voxel may contain the intersection that occurs at the boundary or the common surface between two elements(Fig. 2(left). Parallelizing the intersection test among the elements might result in a race condition, where multiple threads may write different element index values at the same voxel memory location. To account for multiple elements in a voxel and avoid the race condition, we serialize the voxelization and infill operation using an element `for` loop inside the GPU kernel. We perform a marching operation along the ray to identify the entry and exit voxel for the same element. We then fill the intermediate voxels with the same element value to get a filled voxelized model (Fig. 2)(right).

Interpolation of IGA Results:

The IGA results are usually computed at specific feature points such as the control points or the Gauss points. These values are interpolated to the voxel center using Shepard's inverse distance weighting. This method is inefficient in skewed elements or elements with a large number of feature points. In such cases, we compute the closest feature point to the voxel center and assign its value. This approach is faster

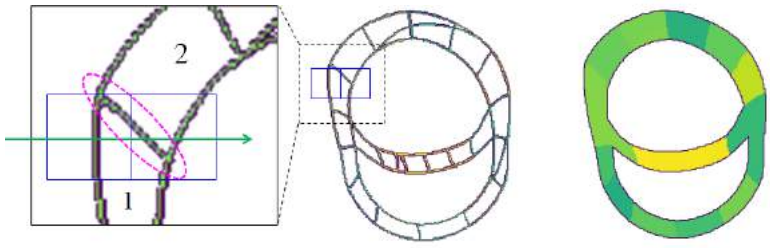

Fig. 2: Left: Boundary voxelization with a detailed view of the interface; Right: A slice of filled voxelized model with respective elements.

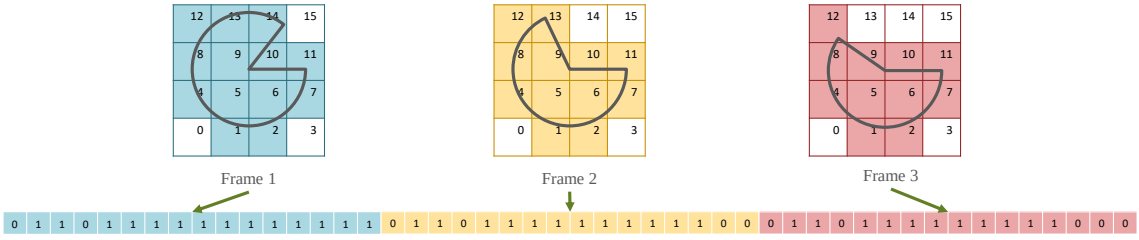

Fig. 3: Voxelized time frame data represented using a flat data structure in the GPU memory.

and gives visually accurate results for high-resolution models. The values assigned to the voxel center are normalized to create a variable density model.

Animated Volume Rendering:

Finally, to animate the volume rendering, the 3D voxelized structure for each frame is converted into a flattened array (Fig. 3) for storing in the GPU memory. This method facilitates using a linear memory location for storing all the time frames, and the voxelization size is used to set the stride. The volume rendering is performed using a GPU voxel ray-casting algorithm, where each pixel on the screen corresponds to a single ray (pixel resolution rendering). The voxelization is then sampled along the ray using a user-defined pitch. The average density along the ray is then used to calculate the pixel color.

Results:

We show the performance of our method on two different types of models for multiple voxelization resolutions (Tab. 1). The cardiac model consists of cubic Bézier volume elements and includes the strain tensor values for 200 time frames. The aorta FSI model consists of quadratic NURBS and the IGA results defined on the control points for 161 time frames. As we can see in Tab. 1, our approach can generate a voxelized model for the entire simulation within a few seconds. In addition, while volume rendering the results, we can maintain a consistent frame rate of over 30fps throughout the animation. Fig. 4 shows a couple of frames from the animated volume rendering output for both models. The animation can also be dynamically sectioned without affecting the overall frame rate.

Table 1: Timing data for voxelization and rendering of the Cardiac and the Aorta model

Resolution	Cardiac Model				Aorta Model			
	Processing time (s)	Ray Intersection Time (s)	Total time (s)	Frame Rate (FPS)	Processing time (s)	Ray Intersection Time (s)	Total time (s)	Frame Rate (FPS)
64R	0.686	0.786	1.472	39.9	1.884	2.660	4.544	35.5
128R	2.870	3.558	6.440	36.7	5.093	18.172	23.266	35.3
256R	24.13	20.873	45.046	49.9	32.336	133.673	166.009	36.1

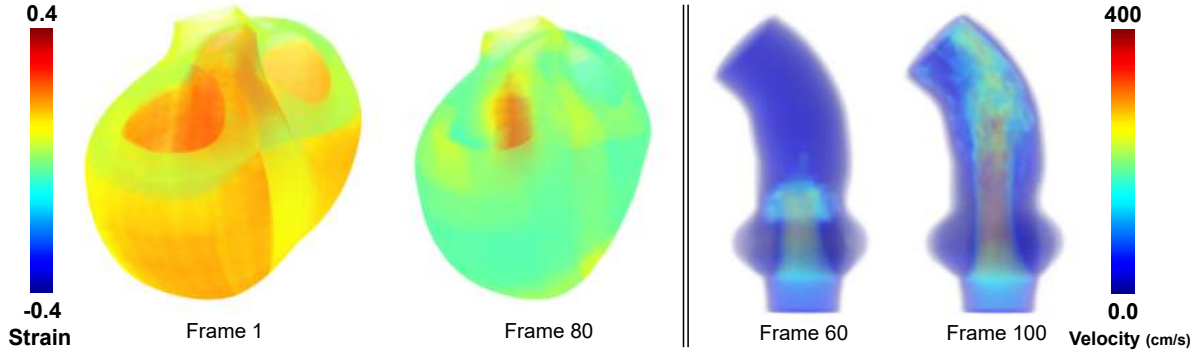

Fig. 4: Left: Rendering of von-mises strain for Heart model, Right: Rendering of velocity of Aorta mode.

Conclusions:

We have developed a framework to animate the results of dynamic isogeometric analysis. This approach uses ray intersections to convert volumetric splines to a time-varying voxel representation. The time-varying voxel models are then volume rendered using GPU-accelerated ray casting to animate any property of interest from the IGA simulations. The animation can be interactively sectioned to study the quantities of interest in greater detail without compromising the frame rates. Our approach allows interactive interrogation of the IGA results that can yield valuable insights into the physical simulation.

Acknowledgements:

We would like to thank Dr. Hsu at Iowa State University for providing the Aorta model. This work was partially supported by NIH R01HL131753, NSF 1750865, and NVIDIA.

References:

[1] Hughes, T.J.R.; Cottrell, J.A.; Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exactgeometry, and mesh refinement, Computer Methods in Applied Mechanics and Engineering, 194(39-41), 2005, 4135-4195. https://doi.org/10.1016/j.cma.2004.10.008

[2] Kajiya, J.T: Ray tracing parametric patches, SIGGRAPH, 116(3), 982, 245-254. https://doi.org/10.1145/800064.801287

[3] Kumar, S.; Manocha, D.: Efficient rendering of trimmed nurbs surfaces, Computer-Aided Design, 27(7), 1995, 509-521. https://doi.org/10.1016/0010-4485(94)00003-V

[4] Toth, D.: On Ray Tracing Parametric Surfaces, ACM SIGGRAPH Computer Graphics, 19(3), 1985, 171-179. https://doi.org/10.1145/325334.325233