GPU-Accelerated Direct Volume Rendering of Finite Element Data Sets

Bingchen Liu
Dept. of Computer Science
University of Auckland
Auckland, New Zealand
bliu035@aucklanduni.ac.nz

Alexander Bock
Scientific Visualization Group
Linköping University
Norrköping, Sweden
alexander.bock@liu.se

Timo Ropinski
Scientific Visualization Group
Linköping University
Norrköping, Sweden
timo.ropinski@liu.se

Martyn Nash
Dept. of Engineering Science
and Auckland Bioengineering Institute
University of Auckland
Auckland, New Zealand
martyn.nash@auckland.ac.nz

Poul Nielsen
Dept. of Engineering Science
and Auckland Bioeng. Institute
University of Auckland
Auckland, New Zealand
p.nielsen@auckland.ac.nz

Burkhard C. Wünsche
Dept. of Computer Science
University of Auckland
Auckland, New Zealand
burkhard@cs.auckland.ac.nz

ABSTRACT

Direct Volume Rendering of Finite Element models is challenging since the visualisation process is performed in world coordinates, whereas data fields are usually defined over the elements' material coordinate system. In this paper we present a framework for Direct Volume Rendering of Finite Element models. We present several novel implementations visualising Finite Element data directly without requiring resampling into world coordinates. We evaluate the methods using several biomedical Finite Element models. Our GPU implementation of ray-casting in material coordinates using depth peeling is several orders of magnitude faster than the corresponding CPU approach, and our new ray interpolation approach achieves near interactive frame rates for high-order finite element models at high resolutions.

Categories and Subject Descriptors

I.3.3 [**Picture/Image Generation**]: Display algorithms; I.3.7 [**Three-Dimensional Graphics and Realism**]: Ray-tracing; I.3.8 [**Computer Graphics**]: Applications

Keywords

visualisation, direct volume rendering, GPU computing, finite elements

1. INTRODUCTION

Finite Element (FE) models are popular in science, engineering, and biomedicine for simulations, shape analysis, data fitting, and as reference frames for model and data comparison. Finite element models represent complex model geometries by connecting sample points (nodes) to (potentially curvilinear) elements. Both the geometry and data fields over the domain are defined by interpolating nodal values over each element using interpolation functions, whose parameter space is referred to as the element's *material coordinates*.

In order to analyse and understand finite element data sets it is beneficial to visualise them. Direct Volume Rendering (DVR) is a popular visualisation technique for interactively exploring complex multi-dimensional data sets. Applying the technique to FE data sets is difficult since the ray-casting process is performed in world coordinates, but data fields are defined over the elements' material coordinates. Hence the visualisation either requires resampling the data into world coordinates, which introduces numerical errors, or computing along each ray sample points in material coordinates, which is slow and numerically complex. For example, a tricubic interpolation $\mathbf{x}(\xi)$ over a cuboidal element contains 64 terms (8 nodes with 8 nodal values and derivatives each), which are cubic functions in the three material coordinate directions. Computing the inverse mapping

$$\xi(\mathbf{x}) = (\mathbf{x}(\xi))^{-1} \tag{1}$$

with a multi-dimensional Newton method requires multiple iterations involving the Jacobian of that mapping (matrix of all first-order partial derivatives) [9].

In this paper we present and compare several techniques for visualizing FE data sets using DVR. For comparison purposes we use a straightforward CPU implementation (as gold standard for evaluating numerical precision), and a traditional resampling process using GPU-accelerated ray tracing of the resulting regular sample grid (as gold standard for evaluating efficiency). We then introduce a more efficient GPU implementation using depth peeling and hardware accelerated FE interpolations and coordinate transformations. We also summarise a new algorithm developed within this framework [3], that decouples the expensive world-to-material space transformation from the rendering stage, thereby allowing it to be performed within a pre-computation stage.

Section 2 reviews previous work on direct volume rendering of finite element models. Section 3 and section 4 present

IVCNZ '12, November 26 - 28 2012, Dunedin, New Zealand

the design and implementation of different DVR techniques for visualising FE models. Section 5 summarises the results of evaluating these implementations. Section 6 concludes this paper and gives an outlook on future work.

2. LITERATURE REVIEW

In our research we use a *ray-casting* method, which traces for each image pixel a ray through the volume and accumulates colour and transparency information representing field values along the ray [12]. This approach is more flexible than alternative approaches [7] and can be efficiently accelerated using graphics hardware.

Traditionally ray-casting is performed in world coordinates using a regular grid of samples, which are interpolated using a reconstruction kernel [13]. Wihelms et al. explored the visualisation of a curvilinear volume by directly ray-casting the volume and by resampling it first into a rectilinear grid [17]. The authors use a greedy algorithm to find the cell that contains the ray sample. The field data for the sample point is then computed by interpolating the cell's vertices. The ray-casting process can be sped up by identifying ray entry, re-entry, and exit cell faces, projecting them onto the image plane, and sorting them by their depth value [6]. In order to alleviate the visualisation error introduced by resampling, Mao et al. apply a stochastic sampling technique called *Poisson disk sampling* to a low-order curvilinear volume and render the samples using a splatting algorithm [10].

FE models can be visualised directly (without resampling) by developing fast approximations for finding the material coordinates of a world coordinate point and/or by accelerating the computation on the GPU. Marmitt et al. subdivide curvilinear volumes into tetrahedral meshes, find tetrahedral cells using a k-d tree structure, and then use Plücker coordinates to speed up ray intersection tests with the triangular element faces [11]. Moreland and Angel visualise linear tetrahedral meshes with interactive frame rates by performing a partial preintegration of the volume rendering integral [14].

Üffinger et al. employ GPU-based raycasting for direct volume rendering of high-order FE models. By parallelising the visualisation process onto a cluster of GPUs the authors achieve interactive visualisation for high-order FE models [16]. To reduce the overhead of the world-to-material space transformation, the FE simulation solution is represented in a reference space using barycentric coordinates. This enables the authors to represent the solution in a cell using a compact monomial representation, which can be sampled in physical space without requiring the expensive world-to-material space mapping from equation 1. The technique assumes a Galerkin FE method solution and is not suitable for large highly curved elements with high field variations.

In summary methods based on resampling suffer from a limited resolution when zooming into a data set, sampling can result in important features being missed, sampling makes it difficult to accurately represent element boundaries, and it results in a loss of the relationship between field values and element geometry (e.g., strain direction relative to an object's surface). Direct visualisation methods can overcome these constraints, but current methods are unable to interactively render complex high-order curvilinear FE models.

The key challenge of direct volume rendering high-order FE models is to avoid or speed up the expensive world-to-material mapping from equation 1. Several of the presented methods employ GPU-acceleration for speeding up computations. In this paper we present two GPU accelerated techniques and compare them with a CPU direct visualisation approach and a resampling approach.

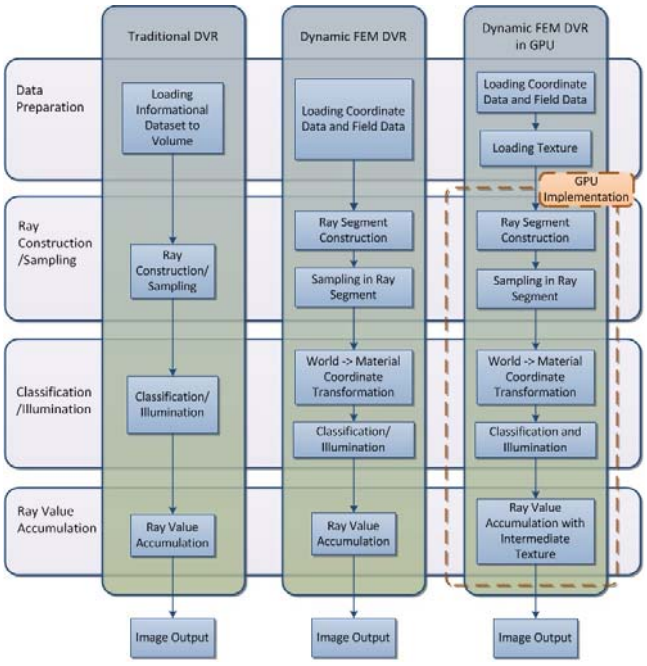


Figure 1: The visualisation pipeline for the traditional DVR approach using resampling (left), and the direct visualisation of FE data without resampling implemented on the CPU (middle) and the GPU (right).

3. DESIGN

3.1 GPU Implementation using Resampling

We can visualise FE data with a traditional GPU-accelerated DVR algorithm [12] by using the following steps illustrated by the pipeline on the left hand side of figure 1:

1. Represent the field data as a regular grid.

2. Compute for each point of the image plane a viewing ray, compute entry and exit points with the volume, and sample the ray.

3. Compute field values at the sample points and determine colour and opacity values at the sample points using user defined classification functions.

4. Accumulate colour and opacity values along each ray.

The regular grid of sample points is computed as follows: we first determine the bounding box of the FE model and define an equidistant sample grid in world coordinates. The sample points' material coordinates are computed using equation 1, which is solved using a multi-dimensional Newton method [15].

3.2 DVR in Material Space - CPU Implementation

In order to visualise a FE data set directly (i.e., without resampling) for each ray traversing the world coordinate space its entry and exit points with a finite element must be found, and the material coordinates of sample points along a ray must be computed. The latter can be achieved using a multi-dimensional Newton method as explained above. The element ray entry and exit points for each ray are efficiently determined by rendering the FE geometry (element surfaces) and colour coding it. The idea is adapted from a similar scheme for GPU-based volume ray-casting [8], however, in our case we associate the three material coordinates ξ_1, ξ_2, and ξ_3 with the RGB channels and encode an element's unique ID in the opacity channel. The resulting visualisation is shown in figure 2. Each pixel encodes for the corresponding viewing ray the intersected element (if any) and the material coordinates of the intersection point. This information can then be used during ray sampling as initial guess for the multi-dimensional Newton method in order to compute the material coordinates of the next ray sample point.

Since a viewing ray can intersect multiple elements we use depth peeling, which has been originally proposed for ray tracing of non-refractive transparent surfaces [5]. In our application this approach yields for each pixel a sequence of element entry and exit points and hence the ray segments for all intersected elements. Shared faces are rendered twice (once for each element), which results in z-fighting problems. We identify the correct sequence of intersected faces by taking into account element IDs and face normals [9].

The process is illustrated by the visualisation pipeline in the middle of figure 1.

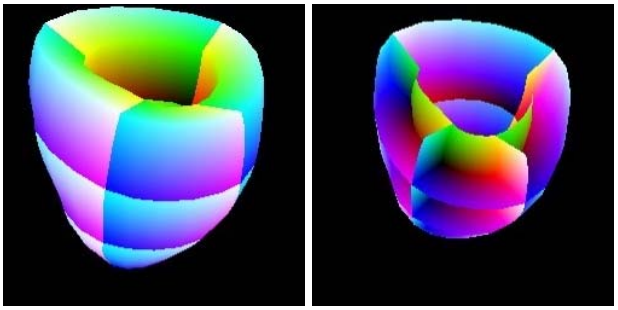


Figure 2: Colour coding of elements' material coordinates for the first entry-points layer (left) and exit-points layer (right).

3.3 DVR in Material Space - GPU Implementation

Implementing the above described direct visualisation approach on the GPU requires several changes: (1) Ray-casting is performed with fragment shaders in parallel using the segments obtained with the depth peeling process. (2) The FE data is stored on the GPU taking into account the limited number of uniform variables on the GPU. The FE geometry is encoded into a 3D texture, where the three texture dimensions correspond to element ID, node ID, and nodal and derivative values. For example, a tricubic interpolation requires a nodal value, derivatives in all three coordinate directions, and four mixed derivatives for a total of 8 values. For data fields we use different representations dependent on whether they are defined by interpolating nodal values, as subsampled field over each element, or as image acquisition raw data (for details see [9] and section 5). (3) The ray sampling and colour and opacity calculation must be implemented on the GPU. This required us to implement FE element interpolation functions and the multi-dimensional Newton method on the GPU [9]. Since the size of a FE model can easily exceed GPU memory, ray segments are processed one depth layer at a time, with one fragment processor for each pixel. Accumulation of colour and opacity values is performed using a separate intermediate texture.

Figure 3 displays the differences between the CPU and GPU-implementation of the direct visualisation approach.

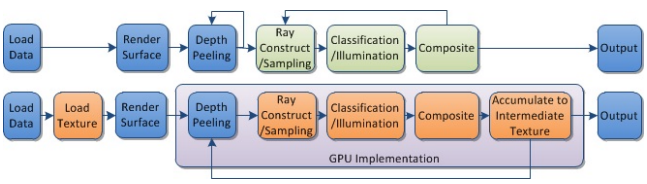


Figure 3: The differences in the pipeline for the CPU (top) and the GPU (bottom) direct visualisation approach.

3.4 Proxy-ray Interpolation Approach

The DVR process can be be sped up by performing the world-to-coordinate mapping in a precomputation step. The resampling approach in subsection 3.1 enables us to utilise the full speed of traditional GPU-accelerated DVR methods. However, the approach introduces numerical errors when reconstructing data from sample points, the resampled data set has a fixed resolution (image resolution decreases when zooming into the data), the approach requires precalculation of all data fields which might be used during interactive exploration, and it does not represent the relationship between the finite element geometry and field values.

In order to overcome these shortcomings we have developed a novel DVR technique for FE models. The method avoids computation of the inverse transformation at run time by pre-computing view-independent proxy rays in material coordinates, and interpolating them during run time in order to approximate rays and ray sample points for a given view point during interactive rendering. This decouples the expensive coordinate inverse transformation from the ray-casting, but we still perform the field data interpolation in the sampling stage.

The number of precomputed proxy rays is defined by sampling an element's surface using n entry points and m exit points and defining rays for each combination of entry and exit points. The values for n and m depend on the element size and field complexity. In our examples we found that $n = m = 5$ yielded good results. The proxy ray segments are straight in world coordinates, but curved in material coordinates as illustrated in figure 4. We represent the proxy rays in material space using Catmull-Rom splines [4], and store them using the spline's control points. In order to reduce storage space we cluster the rays using an approach similar to [1]. During interactive rendering for a given ray, the entry and exit points of the closest proxy rays are determined. The

material coordinates of the ray's sample points can hence be computed efficiently by interpolating the known material coordinates of the surrounding precomputed proxy rays. More details are found in [3].

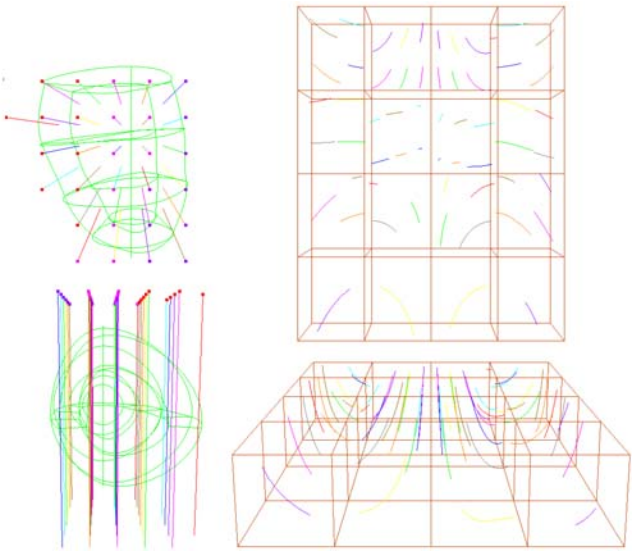


Figure 4: A FE model of a left ventricle of the heart in world coordinates (left) and the elements' material space (right) shown using an axial view (top) and a sagittal view (bottom). Straight parallel rays in world coordinates (left) are curved in the material space of the elements and neighbouring rays have a similar geometry (right).

4. IMPLEMENTATION

All visualisation techniques presented above were integrated into the Voreen volume rendering framework [12]. This enabled us to make use of existing functionalities such as reconstruction kernels, classification functions, and camera and interaction widgets. All newly developed functionalities were defined as processors, in order to use them inside the visual programming interface of Voreen (figure 5).

5. RESULTS

We tested our DVR algorithms using two different Finite Element models:

Heart Model: A model of the left ventricle of a healthy human heart consisting of 16 bicubic-linear elements. A strain field is defined using $6 \times 11 \times 11$ equidistant sample points over the material space of each element.

Tongue Model: A model of a bovine tongue using 64 tricubic elements. Multiple vector fields representing muscle fiber groups are defined over the elements by tricubicly interpolating nodal values.

All visualisations were generated on a PC with Intel i7 3.40GHz CPU, 8 GB RAM and a GeForce GTX 580 graphic card with 512 CUDA cores and 1.5GB GDDR5 memory.

5.1 Comparison with Visualisation Tools

In order to test the correctness of our DVR techniques we compared them with equivalent visualisations obtained

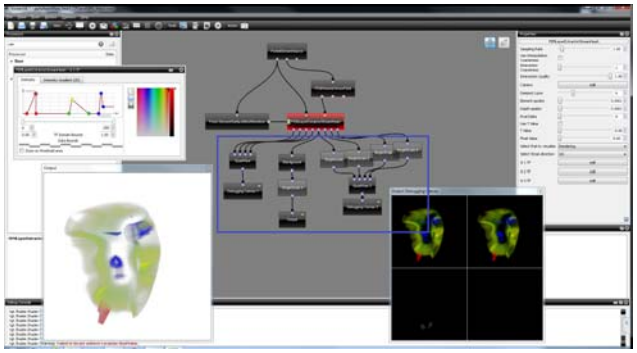


Figure 5: A network for visualising the heart model using the GPU direct visualisation approach. The reused processors of Voreen are highlighted by a blue box. The transfer function widget on the left shows the employed transfer function.

using existing tools. Figure 6 shows on the left a visualisation of the circumferential strain on the endocardial surface of the left ventricle obtained using colour mapping. The 0-isosurface of the circumferential strain is indicated by the blue surface segments at the top of the model. The visualisation was obtained using a polygonal rendering tool for biomedical finite element models [18, 19]. The image on the right of figure 6 shows the equivalent visualisation using our DVR framework.

Figure 7 shows on the left a visualisation of one muscle fiber group within the tongue model. The visualisation was obtained using CMGUI, a 3D visualisation software which is part of CMISS, an open source modelling environment developed by the Auckland Bioengineering Institute at the University of Auckland [2]. The image on the right shows the equivalent visualisation using our DVR framework, with line segments indicating vector directions at ray sample points.

In both cases the visualisations show similar structures and values. The purpose of these tests was to verify that the data sets are correctly loaded and the depth peeling, ray construction, and ray sampling is correctly performed.

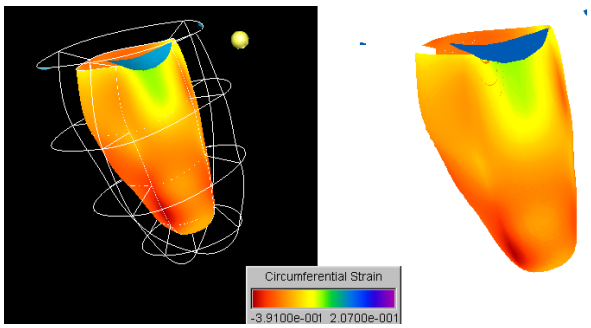


Figure 6: A visualisation of the circumferential strain in the human left ventricle using a polygon-based visualisation tool [18, 19] (left) and our CPU direct visualisation approach (right).

5.2 Precision

Figure 8 illustrates the differences between the four presented visualisation techniques. We use the CPU direct vi-

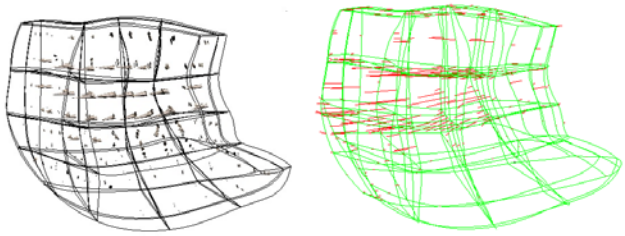


Figure 7: A visualisation of the direction of a muscle fiber group in a bovine tongue performed using CMGUI [2] (left) and our GPU direct visualisation approach (right).

sualisation approach as a gold standard and compare results with: The GPU direct visualisation approach (top); the proxy-ray interpolation visualisation using 5×5 ray entry and exit points for each element face, which results in 360,000 pre-computed rays (middle); and the traditional DVR approach using a regular sample grid of 500^3 voxels (bottom).

The top row of the figure shows that the CPU and GPU implementation of the direct visualisation approach result in very similar, but not identical images. The visualisation obtained with the GPU implementation is less smooth and features have slightly different locations. There are two reasons for this: (1) the multi-dimensional Newton method is running under 64bit (double) precision on the CPU but only 32bit (float) precision on the GPU. This reduces the accuracy of the material coordinate computation. (2) We use GPU hardware interpolation for the strain field, which is less accurate than the corresponding CPU implementations and requires adding a padding layer in all three coordinate directions [9].

The middle row of the figure shows that the proxy-ray visualisation results in a very good visualisation containing all relevant details. However, there are some slight artifacts, as indicated by the red box in the centre of the image. The discontinuity in that artifact is most likely caused by neighbouring rays using different proxy rays in the interpolation process. This error could be reduced using super-sampling at the expense of an increased computation time.

The bottom row of the figure shows that resampling results in a smooth and visually very similar image to the CPU direct visualisation approach. However, the difference image shows larger variations than for the other two visualisation techniques due to the limited resolution of the sample grid.

5.3 Efficiency

We investigated the running time of our DVR algorithms using different data sets and different visualisation parameters. Figure 9 illustrates that the GPU direct visualisation approach is considerably faster than the CPU visualisation. The performance advantage increases with the complexity of the visualisation. For example, when using a complex transfer function more sample points along a ray need to be computed and used for the accumulation step, and when zooming into the image the required precision of the world-to-material coordinate mapping increases. The image on the right hand side of the figure demonstrates that the GPU implementation can be more than four orders of magnitude faster than the corresponding CPU implementation.

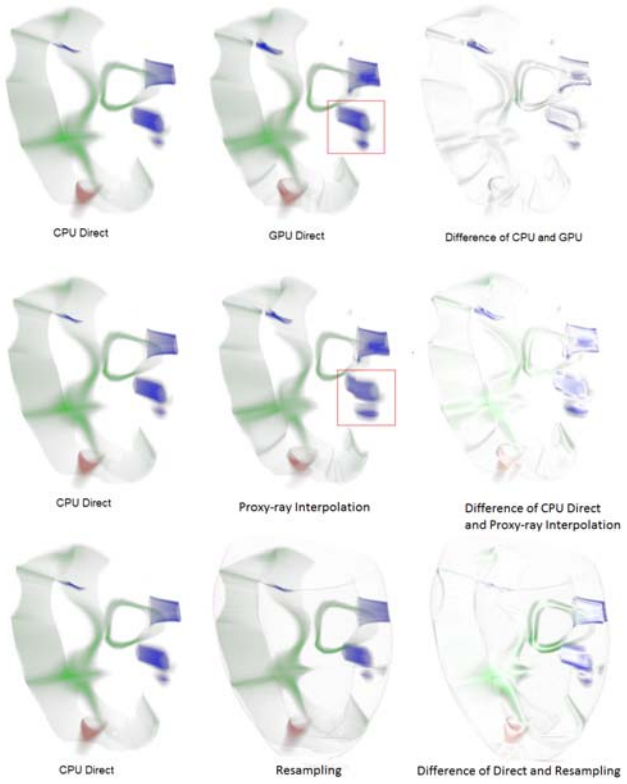
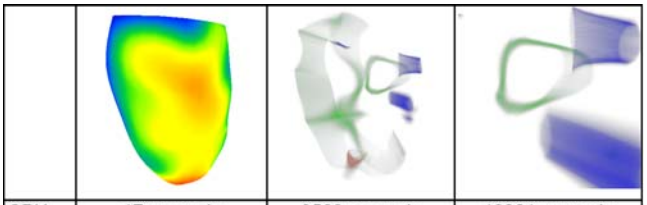


Figure 8: Direct volume rendering of the normal strains of the left ventricle. The opacity and colour transfer functions were designed to display the regions with strain values [-0.184, -0.130] (red), [0.017, 0.044] (green), and [0.236, 0.263] (blue). The column on the left shows the results obtained with the CPU direct visualisation approach. The column in the middle shows the GPU direct visualisation approach (top), the proxy ray interpolation (middle) and the resampling approach (bottom). The column on the right shows the corresponding difference images.



CPU Direct	17 seconds	3583 seconds	19301 seconds
GPU Direct	0.482 seconds	0.752 seconds	1.322 seconds

Figure 9: Three different visualisations of the radial strain in the left ventricle obtained using different opacity and colour transfer functions and zoom factors (top), and the corresponding rendering times using the CPU (middle) and GPU (bottom) direct visualisation approach.

Figure 10 shows that the proxy-ray interpolation approach is another order of magnitude faster than the GPU direct

visualisation approach and it approaches interactive frame rates. However, none of the presented algorithms using the FE material space can achieve the performance of the re-sampling approach, that solely operates in world space.

	Resolution	CPU Direct	GPU Direct	Proxy-ray	Re-sampling
Radial Strain	128^2px	473	0.248	0.061	< 0.01
	256^2px	1197	0.388	0.070	< 0.01
	512^2px	3583	0.752	0.085	< 0.01
	1024^2px	8356	2.439	0.142	< 0.01

Figure 10: Performance comparison of the four algorithms for visualising FE data (rendering time per frame in seconds).

6. CONCLUSIONS AND FUTURE WORK

We have presented a framework for visualizing FE data in material space and presented two new GPU implementations for this task. We compared the presented algorithms using different FE data sets and showed that the GPU direct visualisation approach is several orders of magnitude faster than the corresponding CPU implementation. The novel proxy-ray interpolation visualisation is another order of magnitude faster and approaches interactive frame rates. None of the presented algorithms using the FE material space can achieve the performance of the traditional resampling approach. However, considering the limitations of the resampling approach, the new GPU implementations are viable alternatives, especially for the exploration of multi-field FE data.

In future work we want further to improve the GPU implementations and use them for comparative visualisations, where the FE material space provides a common reference frame for, e.g., models of healthy and diseased left ventricles.

7. REFERENCES

[1] C. Abraham, P. A. Cornillon, E. Matzner-Løber, and N. Molinari. Unsupervised curve clustering using b-splines. *Scandinavian Journal of Statistics*, 30(3):581–595, 2003.

[2] Auckland Bioengineering Institute. CMISS - Introduction to CMGUI, 2012. `http://www.cmiss.org/cmgui`, Last retrieved 16th September 2012.

[3] A. Bock, E. Sundén, B. Liu, B. Wünsche, and T. Ropinski. Coherency-based curve compression for high-order finite element model visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2315–2324, Dec. 2012.

[4] E. Catmull and R. Rom. A class of local interpolating splines. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 317–326. Academic Press, 1974.

[5] C. Everitt. Interactive order-independent transparency, 2001. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.9286`, Last retrieved 10th September 2012.

[6] L. Hong and A. E. Kaufman. Fast projection-based ray-casting algorithm for rendering curvilinear volumes. *IEEE Transactions on Visualization and Computer Graphics*, 5:322–332, October 1999.

[7] K. Kim, C. M. Wittenbrink, and A. Pang. Extended specifications and test data sets for data level comparison of direct volume rendering algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):299–317, Oct. 2001.

[8] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 287 – 292, Washington, DC, USA, 2003. IEEE Computer Society.

[9] B. Liu. GPU-accelerated direct volume rendering of curvilinear finite elements. Master's thesis, Dept. of Computer Science, University of Auckland, Apr. 2012.

[10] X. Mao, L. Hong, and A. Kaufman. Splatting of curvilinear volumes. In G. M. Nielson and D. Silver, editors, *Proceedings of Visualization '95*, pages 61–68, Los Alamitos, California, 1995. IEEE.

[11] G. Marmitt, H. Friedrich, and P. Slusallek. Recent Advancements in Ray-Tracing based Volume Rendering Techniques. In G. Greiner, J. Hornegger, H. Niemann, and M. Stamminger, editors, *Proceedings of 10th International Fall Workshop - Vision, Modeling, and Visualization (VMV '05)*, pages 131–138, Erlangen, Germany, November 2005.

[12] J. Meyer-Spradow, T. Ropinski, J. Mensmann, and K. Hinrichs. Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations. *IEEE Comput. Graph. Appl.*, 29(6):6–13, Nov. 2009.

[13] T. Möller, K. Mueller, Y. Kurzion, R. Machiraju, and R. Yagel. Design of accurate and smooth filters for function and derivative reconstruction. In *Proceedings of the 1998 IEEE symposium on Volume visualization*, VVS '98, pages 143–151. ACM, 1998.

[14] K. Moreland and E. Angel. A fast high accuracy volume renderer for unstructured data. In *Proceedings of the 2004 IEEE Symposium on Volume Visualization and Graphics*, pages 9–16, Washington, DC, USA, 2004. IEEE Computer Society.

[15] W. H. Press, W. T. Vetterling, S. A. Teukolsky, and B. P. Flannery. *Numerical Recipes in C - The Art of Scientific Computing*. Cambridge University Press, 2^{nd} edition, 1992.

[16] M. Üffinger, S. Frey, and T. Ertl. Interactive high-quality visualization of higher-order finite elements. *Computer Graphics Forum*, 29(2):115–136, 2010.

[17] J. Wihelms, J. Challinger, N. Alper, S. Ramamoorthy, and A. Vaziri. Direct volume rendering of curvilinear volumes. *SIGGRAPH Comput. Graph.*, 24:41–47, November 1990.

[18] B. C. Wünsche. A toolkit for visualizing biomedical data sets. In *Proceedings of GRAPHITE '03*, pages 167–174, New York, NY, USA, 2003. ACM.

[19] B. C. Wünsche, R. Lobb, and A. A. Young. The visualization of myocardial strain for the improved analysis of cardiac mechanics. In *Proceedings of GRAPHITE '04*, pages 90–99, New York, NY, USA, 2004. ACM.