Web-based vascular flow simulation visualization with lossy data compression for fast transmission

Rok Oblak, Ciril Bohak, and Matija Marolt

Faculty of Computer and Information Science, University of Ljubljana,
Večna pot 113, 1000 Ljubljana, Slovenia,
2448.rok@gmail.com,
{ciril.bohak, matija.marolt}@fri.uni-lj.si

Abstract. In this paper, we present a web-based system for visualization of flow simulation results in the vascular system for use with consumer-level hardware. The presented tool allows users to design, execute and visualize a flow simulation with a simple workflow on a desktop computer or a mobile device. The web interface allows users to select a vascular model, define the flow simulation parameters, execute the simulation, and interactively visualize the simulation results in real time using multiple visualization techniques. The server-side prepares the model for simulation and performs the simulation using SimVascular. To provide a more efficient transfer of the large amounts of simulation results to the web client, as well as reduce storage requirements on the server, we introduce a novel hybrid lossy compression method. The method uses an octree data subdivision approach combined with an iterative approach that regresses the data points to a B-Spline volume. The evaluation results show that our method achieves compression ratios of up to 5.7 for the tested examples at a given error rate, comparable to other approaches while specifically intended for visualization purposes.

Keywords: visualization toolkit, blood flow simulation, data visualization

1 Introduction

Cardiovascular diseases are the primary cause of deaths in modern developed world and consist of conditions with any kind of cardiovascular system malfunction caused by the heart or the vascular system itself [1]. Many of these can be predicted, explained or identified in an early stage by examining the blood flow dynamics causing various loads and stresses to the cardiovascular system [2].

There were many attempts to describe the cardiovascular system using mathematical models, but in recent years the most efficient ones directly simulate the blood flow in the veins and arteries. They include the generation of patient-specific cardiovascular structures models [3], construction of hypothetical vascular structures to test their flow dynamics, and the combination of both to examine the implications of surgeries changing the patient's vasculature. Such example applications are a stent placement or a bypass surgery, procedures that change the blood vessel geometry and pose significant risks because of various unknowns in hemodynamics. The simulation results can be analyzed to provide insights that could not otherwise be obtained. These simulations are

non-invasive and do not require the presence of a patient. They can be performed many times with many different parameters and thus provide more detailed data about flow dynamics. Their applications have been expanded in recent years as modern advances in imaging technologies enabled scans of live patients to be used for patient-specific model generation and medical result analysis.

The goal of the presented work was to develop a web application focused on accessibility, ease of use and fast workflow iterations where the user can quickly and effortlessly advance from a basic input model to the final visualization of simulation results. Nevertheless, the practical use-cases of the application cover the most frequent cases and offer the same quality of results as other proprietary systems without the need of installing any additional user-side tools. The application integrates the most common simulation result visualization options, relieving the user from need of exporting the results and importing them into a third-party tool, thus shortening the workflow. The simulation result files are compressed in an efficient manner with visualization purposes in mind, enabling large simulation results to be saved on the server without taking up too much of disk space and speeding up the data transmission to the user. The advantages of using the presented client-server approach to simulation are: (1) accessibility of simulation and visualization on low-end devices (such as ultra books, tablets and/or mobile phones), (2) the accessibility of the system from any location with fast enough Internet connection and (3) a possibility of queuing multiple simulation calculations on the server and their overview after they are finished at later time.

2 Related work

The complete process from taking patient observations to analyzing the simulated blood flow results typically involves four major stages: (1) data acquisition where patient-specific data is gathered, (2) model generation where meshes are created from the gathered data and prepared for simulation, (3) the simulation itself and lastly, (4) results gathering and post-processing [2]. Our application workflow follows the same steps, excluding the step (1) of data acquisition.

Stanford Virtual Vascular Laboratory (SVVL) was one of the first attempts to model blood flow in blood vessels by developing a software framework integrating model construction, mesh generation, flow simulation and visualization [4]. It introduced a knowledge-based engineering approach to build a more complex vascular model from a hierarchy of simpler primitives defined parametrically. The results were collected at pre-defined nodal points and visualized in different ways, including flow vector fields and streamlines. [5] was one of the first attempts to reduce the time needed for patient-specific model generation by performing automatic level-set segmentation of 2D slices to create a model and a simulation-suitable mesh. In recent years, various more sophisticated approaches have been developed for user-guided modeling and meshing from medical image data, such as the Vascular Modeling Toolkit [6], Vascular Editor [7], sweep surfaces based approach presented [8] or SimVascular [9], frameworks which provide powerful functionality for vascular modeling and mesh generation. SimVascular was used for meshing the vascular models in our application.

The simulation itself is a task of solving Navier-Stokes equations describing the flow. The solution to these equations is a flow velocity field, which serves as a basis for calculating other physical quantities such as pressures or temperatures. Software CFD solvers have initially simulated blood flow in two dimensions and later in three as technological progress enabled more detailed and faster simulations. Today, flow simulations of the entire vasculature can be performed, as demonstrated by [10]. Many CFD solvers are also available as open source packages, such as: Palabos [11], HemeLB [12] and MUPHY [13]. Among them is also the solver in SimVascular, which was used to perform the simulation in our case. Also relevant is a recent survey on cardiac 4D PC-MRI data processing is presented in [14].

Visual representation of flow simulation results is an important aspect of post-simulation analysis. Scalar values such as pressures or velocity magnitudes are usually displayed by sprite points or 3D spheres colored and scaled proportionally to their values. Volumetric rendering is common for displaying medical images from Magnetic Resonance Imaging [15] or Computed Tomography scans. Velocity field can also be visualized with streamlines and derived methods [16] which display flow movement through space (streamlines) or time (streampaths).

Commercial visualization software is usually limited to use of unconventional file formats and often requires additional preprocessing for visualization purposes. Visualization Toolkit (VTK) [17] is a very general visualization toolkit supported by many popular simulation (e.g. Palabos [11]) and visualization software packages (e.g. ParaView and ParaViewWeb [18]). The VTK file format is also used internally in our application but we also use our own file format for storing and transferring the results due to our use of a custom compression method.

Flow simulations produce results in form of scalar or vector values for different quantities (e.g. velocity, pressure, wall shear stress etc.). The results can consist of millions of values depending of the number of measured quantities and number of time steps in the simulation yielding gigabytes of data for a single simulation run. To optimize the storage and shorten the transfer times it is meaningful to use data compression.

A general approach to floating point data compression is presented in works [19,20]. To address medical domain, researchers have proposed lossless compression methods, such as lossless stationary wavelet transform on 2D slices of 4D medical images (volumetric data through time) [21]. Wavelets have also been used for flow simulation data, such as [22] where a discrete wavelet transform was used on an octree subdivision of airflow simulation data. Other types of regressions have also been described, such as using polynomials [23] on rectangular blocks of 2D medical images and encoding the polynomial coefficients with Huffman coding, and also encoding the residual error with run-length encoding to achieve lossless compression. Another approach for rapid high-quality compression of volumetric data is presented in [24].

While the above methods are suitable for storing the original simulation data for further analysis, we can exploit the fact that the data will be used for visualization purposes only. In this case, the required precision is substantially lower than the IEEE 754 double-precision binary floating-point format [25].

For purposes of real-time rendering of time-varying volumetric data researchers have extended and adapted an MPEG compression method for isosurface and volumet-

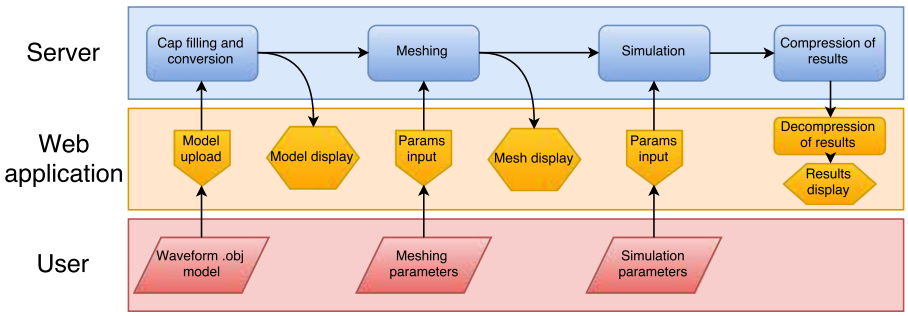


Fig. 1: The pipeline of the proposed system which also presents the user work-flow.

ric data [26]. Another approach [27] aims for high-accuracy compression by linearizing the data points, sorting them and fitting them to a precise one-dimensional B-Spline. In [28], authors present a lossy compression method for structured and unstructured grid simulation data modeled as a graph decomposition problem, where sets of vertices are replaced with a constant value bounded by a user defined error. Because the above presented approaches are targeted to specific domains, mostly for non-porous media and higher compression accuracy, they could not be directly applied to our case.

Our approach combines the meshing, simulation and visualization steps in a single application able to process crude 3D models without the need of additional manual annotations. No additional software is thus needed in the workflow and since the application is web-based, no installation is required. Ease of use is also of note, as the intuitive user interface doesn't need additional documentation and can be used by anybody not previously acquainted with CFD software.

In the following section we present the developed application and the novel compression methods used for compressing the simulation results data. In Section 4 we present the qualitative application evaluation and quantitative evaluation of compression method. In Section 5 we present the conclusions and give the guidelines for possible future work.

3 Methods

In this section we present the application workflow, how the system functionality is split between front-end and back-end part, and a novel lossy compression method for flow simulation data compression used in our system for optimizing the storage space and for faster transmission of simulation data from back-end to front-end part of the system for visualization purposes displayed in the diagram in Figure 1.

3.1 Application Workflow

The application consists of two parts. The back-end is a Node.js server application and performs the meshing operations and executes the flow simulation. The front-end is

a web application developed using the Angular.js JavaScript framework intended for flow simulation results visualization using Three.js, a WebGL graphics library, used for interactive display of the simulation models and simulation results.

Preprocessing The first step in the workflow is to upload a 3D model of a vascular system in Wavefront .obj file format. The uploaded model must have holes for inlets/outlets (inflow and outflow caps) and is converted into VTK polydata format for further processing. All the faces of the model are annotated as either blood vessel walls or input/output cap surfaces. This preprocessing is done with a custom back-end Python script, which centers the mesh geometry, identifies the cap holes, constructs the appropriate cap surfaces and annotates the individual faces. The script also estimates each cap radius from its surface area to provide the user with a suggested initial edge length value in the following tetrahedral meshing step. The resulting model, along with cap surface annotations, is displayed in the front-end application for user inspection.

Meshing After the user confirms the preprocessing step results the second step is executed on the back-end – a tetrahedral mesh generation. The generated mesh is suitable for use with numerical simulation methods such as the subsequent flow simulation. The user has to provide the desired tetrahedron edge length value for defining the simulation resolution. While the user is already given a suggested edge length from the previous step, he can define a different value in the dialog window before starting the meshing process. In our system, TetGen is used for mesh generation, exposed through the SimVascular TCL (Tool Command Language) shell.

Simulation The most processing-intensive back-end step of the system workflow is the simulation step. To start a simulation, the user has to define set of simulation parameters containing the inlet/outlet configuration, inlet flow rates, and outlet flow resistances. The user is already provided with the default parameter values for an individual inlet/outlet but should adjust them according to the desired simulation scenario. All inlets/outlets are displayed in a list (see Figure 2 left). By selecting an individual item from the list, the selected inlet/outlet cap surface is highlighted and centered on the screen making it easy for the user to identify it and adjust its parameters. After the user defines all the simulation parameters, the simulation process can be started.

The simulation is performed using SimVascular-provided command-line utilities: (1) pre-solver for boundary condition generation, (2) solver for performing the finite-element simulation and (3) post-solver for collecting the simulation results data. The solver supports a high-degree of parallelization to speed up the simulation process through the use of MPICH, a high-performance portable implementation of the Message Passing Interface standard.

Compression In the following step, the obtained simulation results in VTK format are parsed and compressed by the compression module using the compression method presented in Section 3.2, before they are stored to the disk or sent to the front-end for visualization. They contain time step data for flow velocities, flow pressures, in-plane tractions, wall shear stresses and time-derivatives of flow velocities.

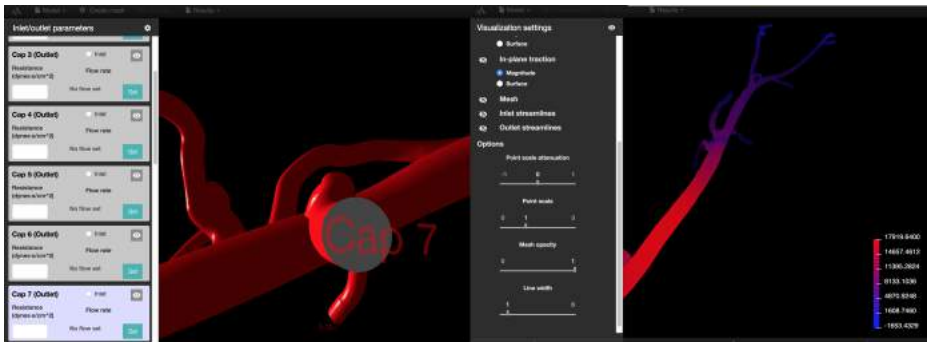


Fig. 2: Left: the application interface when previewing cap faces and setting inlet/outlet parameters. Right: the interface when visualizing the simulation results.

Visualization In the final workflow step, the flow simulation data is transmitted to the front-end application where it is visualized. The visualization is implemented using the Three.js library and allows the user to move and rotate the view as well as change the zoom. The visualization parameters can be set in the side panel of the web application. The user can also move between the individual time steps of simulation using time slider at the bottom and see what values are represented by individual colors in the legend on the right.

The visualization component supports different visualization types:

Mesh display with adjustable mesh opacity, enabling quantities inside the mesh to be visible while the overall structure of the vascular system is displayed.

Scalar point display shows individual simulation values as 2D sprites colored and scaled relative to their values, with adjustable scale and attenuation factors (Figure 3a).

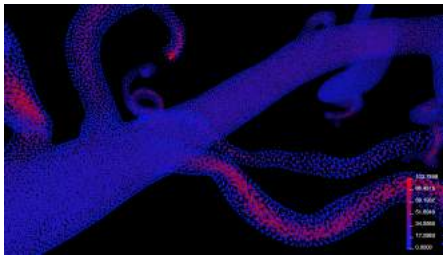
Vector display shows individual lines, scaled and colored relative to the vector data (Figure 3b).

Surface display shows a mesh texture of the vascular system generated by interpolating between vertex values, intended for quantities along the vessel surfaces such as wall shear stresses (Figure 3c).

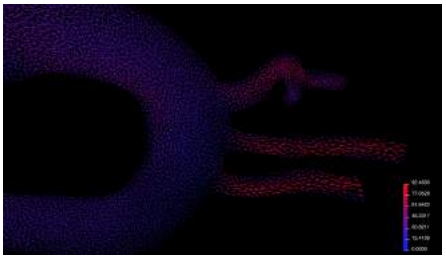
Streamlines generated by sampling starting points at mesh inlets or outlets and following the flow in a forward or backward direction (Figure 3d).

3.2 Compression of flow simulation data

Simulations on meshes with hundreds of thousands to millions of data points can produce large result files measuring several gigabytes in size if no compression is used and the data is stored at its original (usually 64-bit) precision. This presents challenges when transferring these files over the network or storing them on a medium.

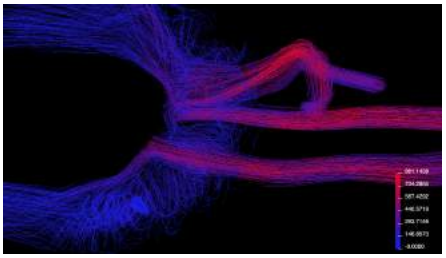


(a) Flow velocity displayed as scalar points.



(b) Flow velocity displayed as a vector field.

(c) Wall shear stresses displayed on the mesh's surface.



(d) Streamlines and resulting vortices of four inlets and two outlets.

Fig. 3: The four implemented visualization methods.

Our problem domain is specific enough to justify the development of a new compression method. Specifically, (1) the points are not distributed evenly and only take values on sparse positions of blood vessels, requiring the use of a space-decomposition method, (2) the targeted accuracy is lower than is usual for others since the data is intended for visualization purposes only - for example, [27] targets a higher compression accuracy but requires an index to be stored for each value, requiring an additional amount of bits per each point.

In our approach we use an octree to subdivide the data points into blocks (N), where each block is treated independently, to take advantage of local data coherence. The subdivision level depends on the point density so that the majority of blocks contain a predefined number of points n (in our experiments, $n = 1000$ turned out to be the most suitable for most meshes).

The process of compressing each block consists of a iterative approach that attempts to fit the data points to a B-Spline volume S:

$$S(u,v,w) = \sum_{g=0}^{n} \sum_{h=0}^{n} \sum_{i=0}^{n} N_{g,k}(u)\, N_{h,k}(v)\, N_{i,k}(w)\, p_{g,h,i},$$

where u, v and w are our point coordinates and $p_{g,h,i}$ are B-Spline control points in a cubic formation. We can rearrange the basis function outer products in a single three-dimensional matrix:

$$x = [N_{0,k}(u), \ldots N_{n,k}(u)] \otimes [N_{0,k}(v), \ldots N_{n,k}(v)] \otimes [N_{0,k}(w), \ldots N_{n,k}(w)],$$

where $\otimes$ is the outer product of two vectors. Having M values we can express this as a linear system $X * P = B$ where X is a matrix of size $M \times n^3$ filled with rows of matrices x flattened into vectors of size $1 \times n^3$, P is a vector of control points of size $1 \times n^3$ and B is a vector of quantity values of size $1 \times M$. We can find a best fit solution $P = (X^T X)^{-1} X^T B$ using linear least squares approach to minimize the sum of squared differences.

The method attempts to perform iterative B-Spline fitting to encode the data by going through a list of "presets" defining the number of B-Spline control points N_c in one dimension, starting with 2 (total 8 points) and going through to 9 (total 729 points). After every iteration, the average error level is measured and the process is repeated with the next control point preset until the measured error level is sufficiently low; this way a suitable B-Spline solution is found. The control points are converted to IEEE 754 half-precision format (16 bits) [25] in order to save additional space. The process for a single octree block is described in Algorithm 1.

Algorithm 1 Iteration process for an octree block

$errThr \leftarrow getErrorThreshold(blockVals)$
$bitsPerPoint \leftarrow getBitsPerPoint(blockVals)$
$solution \leftarrow null$
for $numCtrlPts \in \{2, ..., 9\}$ **do**
  if $16 \cdot numCtrlPts^3 \geq bitsPerPoint \cdot len(blockVals)$ **then**
    break
  end if
  $avgErr, encData \leftarrow encode(blockVals, numCtrlPts)$
  if $avgErr \leq errThr$ **then**
    $solution \leftarrow encdData$
    break
  end if
end for
if $solution \equiv null$ **then**
  $solution \leftarrow quantize(blockVals)$
end if

The iteration stops when the number of bits required to encode the control points ($16N_c^3$) is larger than the number of bits required to do local quantization (bn where b is the number of bits per point and n is the number of points in the block), or if the error level was not sufficiently low. If the local value range $|V_{max} - V_{min}|$ is smaller than the desired error level, the values are encoded as a constant, implicitly set to be the midpoint between the minimum and the maximum values ($\frac{V_{min}+V_{max}}{2}$).

Each region is encoded in a binary format with the following information: compression type used for this region, minimum local value V_{min}, maximum local value V_{max}, followed by data depending on the type of compression used, as displayed in Table 1. The constant encoding does not need any further data, while the other two types need the number of control points along one dimension N_c and the control points c_i for

bit block num.	1	2	3	4	5	6	...
B-Spline	type (2)	V_{min} (32)	V_{max} (32)	N_c (5)	c_1 (16)	c_2 (16)	...
quantization	type (2)	V_{min} (32)	V_{max} (32)	b (4)	v_1 (b)	v_2 (b)	...
const. enc.	type (2)	V_{min} (32)	V_{max} (32)	-	-	-	-

Table 1: The binary file format of a single octree block, depending on the type of encoding used for this block (B-Spline regression, quantization or constant encoding).

B-Spline compression, or the number of bits per point b and quantized values v_i for quantization.

Decompression is performed on the front-end to decode the data. Since point positions are sent separately, the octree can be reconstructed implicitly.

4 Results

The application was deployed on a Ubuntu server with an 8-core, 16-thread Intel Xeon CPU enabling fast simulations (the tested scenarios all completed within a few minutes). To evaluate our application, we tested the simulations on seven blood-vessel models, most of which were modeled after real volumetric scans in SimVascular or other modeling and segmentation tools. We needed to validate the proper input parameter parsing - the inlet-outlet mappings, flow rates and resistances - as well as how the visualized results (after a lossy compression) compared with an external tool, ParaView (Figure 4). The main differences between left (our system) and right (ParaView) image in Figure 4 is the used color map and the orientation of the vascular model. One can clearly see that the emerging parts of the vessel tree are same in both cases.

Another thing we tested was rendering performance since real-time 3D interaction is crucial for informative visualizations. While the web interface was intended for larger displays, the visualizations were rendered with smooth framerates even on mobile phones, including the largest of scenarios tested in Table 2, which is an important advantage of our web-based approach with server-side computational offloading. Table 2 contains the properties of simulation models used during system testing. While the smallest model has approx. 43 thousand mesh points and 685 thousand total simulation data points in the 16-step simulation scenario the most complex model consists of more than 200 thousand mesh points and over 3.4 million total simulation data points in 16-step simulation scenario.

The streamlines present the largest rendering bottleneck because of the high number of individual line segments they consist of. A mobile phone (Samsung Galaxy S6) could render a combined display of 26,695 scalar points and 752,949 streamline line segments with an average framerate of 18 frames per second. A larger model with 181,411 points rendered at 27 frames per second in case of scalar points and 34 frames per second in case of velocity vectors. A larger number of streamline segments becomes unpractical for the mobile phone, but can still be rendered on a laptop computer. A scenario with

	DoubleCylinder	ForkFlat	AortaSmall	ForkSmall	Coronary	AortaBig	ForkBig
caps	3	3	6	3	11	18	20
mesh pts.	42864	50674	185447	107652	212519	132049	181411
6 st. pts.	257184	304044	1112682	645912	1275114	792294	1088466
16 st. pts.	685824	810784	2967152	1722432	3400304	2112784	2902576

Table 2: Model set used along with the number of caps, mesh points and total data points for 6 and 16 simulation steps

4,356,391 streamline segments renders at 48 frames per second, on average on a 2015 MacBook Pro.

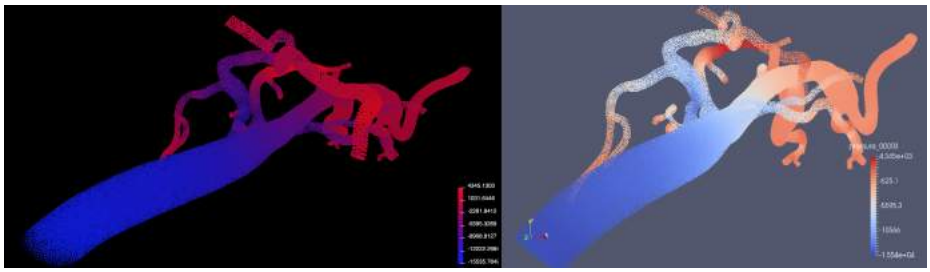


Fig. 4: Comparison of pressure visualizations in our application (left) and ParaView (right).

The first part of our evaluation was ensuring the input parameters affect the simulation as intended. For this task, we tested several models using different inlet-outlet mappings and different flow rates and outlet resistances, and visually evaluated the simulation results. An example of two quantities being displayed in a model of aorta can be seen on Figure 5.

4.1 Compression results

The developed compression method was compared with (1) the baseline method where 6 or 8 bits of precision is used to quantize the values (yielding an average target error rate of 0.09 % and 0.39 %, respectively), (2) a method where the values are quantized in a time dimension over consecutive time steps, and (3) the hybrid method without using B-Spline regression (instead quantizing all blocks). We evaluated the performance on the set of models presented in Table 2 for two main quantities – pressures and velocities – over four scenarios, dividing them in terms of precision (6-bit or 8-bit error target or 0.09 % / 0.39 % maximum average error rate, respectively) and simulation parameters. The simulation was performed over 16 consecutive time steps with varying sinusoid-shaped pulse flow.

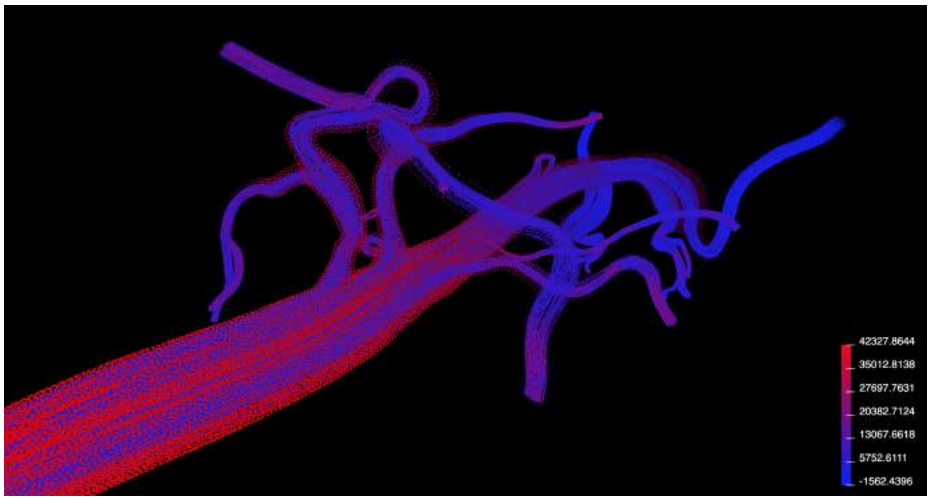
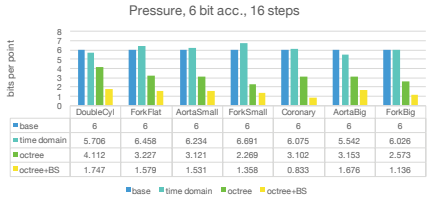
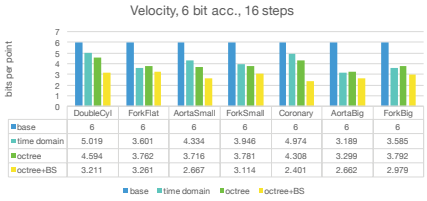


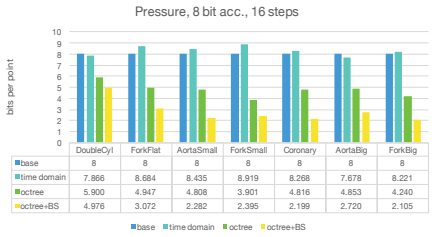
Fig. 5: Simulation results in a model of aorta, with pressures displayed as scalar points alongside streamlines.



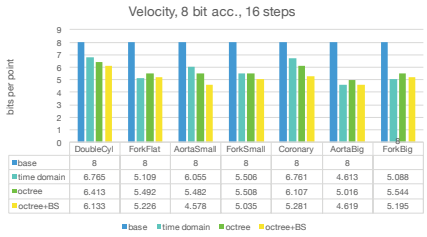
(a)



(b)



(c)



(d)

Fig. 6: Comparison ratios of compression methods at 6 bit accuracy target (max. avg. error of 0.39 %) 6a and 6b and comparison ratios at 8 bit accuracy target (max. avg. error of 0.09 %) 6c and 6d.

The charts displayed in Figure 6 present the comparison of the four compression methods in terms of the average number of bits needed to encode a single data point (each 1D component in case of 3D velocity vectors).

The results show that our hybrid method using a mix of B-Spline regression and quantization in octree subdivisions is always more effective than the method only using octree quantization, and only in a single instance minimally less efficient than time-domain quantization. The amount of improvement over the method using just octree quantization is larger in case of pressure values than it is in case of velocity vectors, ranging from 40 % to 55 % and 6 % to 25 % fewer bits per point on average for pressures and velocity vectors, respectively, translating to compression ratios of up to 5.7 over the baseline in case of pressures encoded with 6-bit accuracy target, on average. The method is very sensitive to the initial error rate target, being much more efficient on larger maximum average error rates (e.g. 6 bit target or maximum average error of 0.39 %), especially in case of velocity vectors.

Velocity vectors encoded under a stricter accuracy constraint (8 bit target or maximum average error of 0.09 %) are the least efficient scenario, providing 6 % to 9 % fewer bits per point compared to just using octree quantization. This is due to larger numbers of “outliers" skewing the smoothness of the values and making it harder for a regressed B-Spline volume to fit the data, making the error rate unsuitable and thus requiring more control points to meet the error rate criterion. If the error criterion is relaxed, the iteration process can find a suitable B-Spline model in a significantly larger portion of regions and thus the method’s effectiveness is larger. This is also the reason why this method is more efficient when encoding pressure values, as these values usually have a much lower number of outliers and values’ transitions are much “softer”.

The average number of bits per point for compressed data is thus sufficiently low to enable network transfers and convenient disk storage. For example, in case of pressures in a *Coronary* model simulation with 6 time steps, the original data size of 10.2 MB with 64-bit accuracy was compressed to 0.083 MB with 6-bit accuracy, a compression ratio of 122.9. In case of velocities and 16 time steps, the original data size of 81.6 MB at 64-bit accuracy MB was compressed to 3.06 MB with 6-bit accuracy, a compression ratio of 26.6.

5 Conclusions

Simulation of cardiovascular systems and informative display of results is an interesting field with great past record and even greater future promises in providing alternative means for cardiovascular disease study and surgery planning. In this work we developed an application for accepting cardiovascular models, simulating the blood flow inside them and visualizing the results in different ways. We exposed the functionality through a web application with a simple and intuitive GUI to enable fast workflow iterations and the ability to load previously saved results and models. We connected the web application to a back-end application performing the computationally intensive parts of the process. We implemented different ways of visualizing the results through an interactive 3D canvas. We also implemented a method for result data compression to minimize the amount of storage needed and maximize network file transfer speeds.

Our evaluations showed successful simulations and visualizations in line with provided parameters and expectations. The model conversion, meshing and simulation processes performed as expected and the simulation results mirrored the desired changes to the input parameters. The visualizations displayed interesting features formed by blood flow and were each able to provide an informative and complete display of data. Their performance was good enough to enable smooth real-time interaction even on mobile devices. The developed compression method was able to achieve good results, producing relatively small file sizes and retaining the accuracy needed for the visualization without the accuracy loss being noticeable to the user, even though the improvement of the hybrid method compared to the others was not always as big in the best performing cases.

This work could serve as a base for several further improvements. The user interface could accept more parameters for more advanced usage and more advanced users wanting more fine-tuning in their simulations. More visualization options could be developed, such as volumetric rendering which requires the development of an efficient method to enable real-time rendering, and other 3D structures derived from streamlines, such as streamribbons.

The hybrid compression method could also see improvements in tweaking the B-Spline parameters and regression to enable a larger ratio of optimal B-Spline regressions in octree blocks since its performance is limited by the number of blocks suitable for efficient B-Spline regression. Different volume subdivision methods could also be attempted to achieve better data distribution in individual subdivisions and limit the number of subdivisions unfit for B-Spline regression. Finally, the compression algorithm could be rewritten in a more efficient language and executed on a GPU to improve its execution speed.

References

1. Mendis, S., Puska, P., Norrving, B., et al.: Global Atlas on Cardiovascular Disease Prevention and Control. World Health Organization (2011)
2. Doost, S.N., Ghista, D., Su, B., Zhong, L., Morsi, Y.S.: Heart blood flow simulation: a perspective review. BioMedical Engineering OnLine **15**(1) (Aug 2016) 101
3. Arts, T., Lumens, J., Kroon, W., Donker, D., Prinzen, F., Delhaas, T.: Patient-specific modeling of cardiovascular dynamics with a major role for adaptation. In: Patient-Specific Modeling of the Cardiovascular System. Springer (2010) 21–41
4. Taylor, C.A., Hughes, T.J., Zarins, C.K.: Finite element modeling of blood flow in arteries. Computer Methods in Applied Mechanics and Engineering **158**(1-2) (1998) 155–196
5. Wilson, N., Wang, K., Dutton, R.W., Taylor, C.: A software framework for creating patient specific geometric models from medical imaging data for simulation based medical planning of vascular surgery. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer (2001) 449–456
6. Antiga, L., Piccinelli, M., Botti, L., Ene-Iordache, B., Remuzzi, A., Steinman, D.A.: An image-based modeling framework for patient-specific computational hemodynamics. Medical & Biological Engineering & Computing **46**(11) (2008) 1097
7. Marchenko, Y., Volkau, I., Nowinski, W.L.: Vascular editor: From angiographic images to 3d vascular models. Journal of Digital Imaging **23**(4) (Aug 2010) 386–398

8. Kretschmer, J., Godenschwager, C., Preim, B., Stamminger, M.: Interactive patient-specific vascular modeling with sweep surfaces. IEEE Transactions on Visualization and Computer Graphics **19**(12) (Dec 2013) 2828–2837
9. Updegrove, A., Wilson, N.M., Merkow, J., Lan, H., Marsden, A.L., Shadden, S.C.: Simvascular: An open source pipeline for cardiovascular simulation. Annals of Biomedical Engineering **45**(3) (Mar 2017) 525–541
10. Zhou, M., Sahni, O., Kim, H.J., Figueroa, C.A., Taylor, C.A., Shephard, M.S., Jansen, K.E.: Cardiovascular flow simulation at extreme scale. Computational Mechanics **46**(1) (2010) 71–82
11. Meier, S., Hennemuth, A., Tchipev, N., Harloff, A., Markl, M., Preusser, T.: Towards patient-individual blood flow simulations based on pc-mri measurements. Informatik Journal **41** (2011) 4–7
12. Mazzeo, M., Coveney, P.: Hemelb: A high performance parallel lattice-boltzmann code for large scale fluid flow in complex geometries. Computer Physics Communications **178**(12) (2008) 894 – 914
13. Bernaschi, M., Melchionna, S., Succi, S., Fyta, M., Kaxiras, E., Sircar, J.: Muphy: A parallel multi physics/scale code for high performance bio-fluidic simulations. Computer Physics Communications **180**(9) (2009) 1495 – 1502
14. Köhler, B., Born, S., van Pelt, R.F.P., Hennemuth, A., Preim, U., Preim, B.: A survey of cardiac 4d pc-mri data processing. Computer Graphics Forum **36**(6) 5–35
15. Anastasi, G., Bramanti, P., Di Bella, P., Favaloro, A., Trimarchi, F., Magaudda, L., Gaeta, M., Scribano, E., Bruschetta, D., Milardi, D.: Volume rendering based on magnetic resonance imaging: Advances in understanding the three-dimensional anatomy of the human knee. Journal of anatomy **211**(3) (2007) 399–406
16. Ueng, S.K., Sikorski, K., Ma, K.L.: Fast algorithms for visualizing fluid motion in steady flow on unstructured grids. In: IEEE Conference on Visualization, 1995. Proceedings. (10 1995) 313–320
17. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit (4th ed.). Kitware (2006)
18. Jourdain, S., Ayachit, U., Geveci, B.: ParaViewWeb: A Web Framework For 3D Visualization And Data Processing. International Journal of Computer Information Systems and Industrial Management Applications **3** (2011) 870–877
19. Lindstrom, P., Isenburg, M.: Fast and Efficient Compression of Floating-Point Data. IEEE Transactions on Visualization and Computer Graphics **12**(5) (2006) 1245–1250
20. Lindstrom, P.: Fixed-rate compressed floating-point arrays. IEEE Transactions on Visualization and Computer Graphics **20**(12) (2014) 2674–2683
21. Belhadef, L., Maaza, Z.M.: Lossless 4d medical images compression with motion compensation and lifting wavelet transform. International Journal of Signal Processing Systems **4**(2) (2016) 168–171
22. Sakai, R., Sasaki, D., Obayashi, S., Nakahashi, K.: Wavelet-based data compression for flow simulation on block-structured cartesian mesh. International Journal for Numerical Methods in Fluids **73**(5) (2013) 462–476
23. Al-Khafaji, G., George, L.E.: Fast lossless compression of medical images based on polynomial. International Journal of Computer Applications **70**(15) (2013)
24. Nguyen, K.G., Saupe, D.: Rapid high quality compression of volume data for visualization. Computer Graphics Forum **20**(3) (2001) 49–57
25. : Ieee standard for floating-point arithmetic. Standard, IEEE (Aug 2008)
26. Sohn, B.S., Bajaj, C., Siddavanahalli, V.: Feature based volumetric video compression for interactive playback. In: Proceedings of the 2002 IEEE Symposium on Volume Visualization and Graphics. VVS '02, Piscataway, NJ, USA, IEEE Press (2002) 89–96

27. Lehmann, H., Werzner, E., Mendes, M.A.A., Trimis, D., Jung, B., Ray, S.: In situ data compression algorithm for detailed numerical simulation of liquid metal filtration through regularly structured porous media. Advanced Engineering Materials **15**(12) (2013) 1260–1269
28. Iverson, J., Kamath, C., Karypis, G.: Fast and Effective Lossy Compression Algorithms for Scientific Datasets. Euro-Par (2012) 843–856