An Introduction To In-Situ Visualization Concepts

Raffael P. Theiler*
Student University of Zurich, 5. Semester Bsc Neuroinformatics

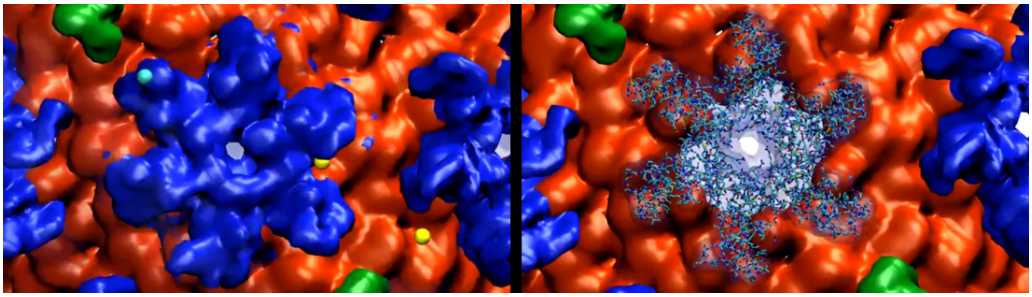

Figure 1: *Supported by the rapidly evolving technology, scientists are able to capture and simulate structures to the very last detail. The magnitude of the challenge to visualize such data is demonstrated by a recent simulation of the HIV capsid where insight is gained by bringing the parts together down to the atomic structure.[TCBG 2013]*

Abstract

In-situ visualization is becoming more and more relevant with simulations producing large amounts of data (exa-scale), effectively exceeding the capabilities of current I/O systems, allowing only partial data persistence. To gain insight, visualization information must be decoupled from a running simulation cluster or is lost. To achieve this, modern in-situ rendering pipelines extend traditional visualization techniques such as ray casting and depth maps, known for the past 20 years. This survey presents two state-of-the-art concepts using these methods in detail. These pipelines are hard to understand for researchers from different fields who are addressed in this paper. To provide insight, key elements of each in-situ pipeline are addressed individually and explained from scratch to give the scientist a better understanding of how they interact.

Keywords: in situ rendering, large scale visualization, visualization technologies, marching cubes algorithm, layered depth images, volumetric depth images, depth maps, feature extraction and tracking

Concepts: •**Computing methodologies** → *Massively parallel algorithms;*

1 Introduction

It is a scientists desire to gain as much insight as possible from available data. This can be challenging, especially when researchers approach a new project with no knowledge about what is important. An inspection of the traditional approach of data collection, which is then proceeded by data analysis, reveals several issues that can be summarized:

- Simulations reach exa-scale (10^{18} floating point operations per second) [Ye et al. 2015, p. 1]. Thus it is no longer possible to collect all the produced data due to technical and financial limitations, considering the price of storage, resulting in a potential loss of information.
- The visualization of collected data is computationally expensive [Parker et al. 1998, p. 1]. Because of that, it is challenging to provide an interactive visualization to the user.
- Sophisticated algorithms are developed to allow researchers to view the collected data. This requires deep knowledge of how to apply these concepts and knowing the potential drawbacks of each method.
- The researcher should have a priori knowledge of the simulation to decide what to simulate. This is not possible in many cases, hence the only way to gain insight is by repeating the simulation over and over.
- Large scale parallelization is necessary to produce results in a time efficient manner.

To find a solution to many of these issues, it becomes necessary to separate from the traditional iterative approach and find new ways to process data. Ongoing research can be grouped roughly in tree categories: First, a promising technique is to combine simulation and visualization (in-situ), accomplished first in the 1990s [Ma 2009], which addresses the issue of lost simulation data. To speed up the visualization, secondly, the process of rendering is parallelized to be processed on supercomputers, which requires dedicated algorithms such as the 2-3 Swap Composing Algorithm to reduce I/O [Yu et al. 2008]. Third, the visualized data is addressed directly and optimization is done by reducing the data (selection), optimization of the rendering process, or tracking of features of interest. This paper tries to make the following contributions:

- Break down the different aspects of large scale visualization in a way that it is understandable to researchers who are in need of large scale visualization but operate in a different field.
- Introduce a depth map based in situ-pipeline with feature extraction and tracking. Then show the key differences to a different approach: in-situ space time volumetric depth images.
- Introduce the technologies that were engineered into these pipelines.

*e-mail:raffael.theiler@uzh.ch
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org. © 2016 ACM.

2 History and Prerequisites

The need for visualization is historically related to the medical field, where computer tomography and magnetic resonance imaging provided three dimensional surface data that had to be visualized. In 1987, the same Year as VGA was introduced, displays could draw RGB graphics with resolutions up to 640x460 [Polsson 2015], making it technically possible to display data as we know it today. In the same year, the marching cubes algorithm (MCA) was designed to triangulate models of constant density [Lorensen and Cline 1987] which is used, mostly optimized for graphical processing units (GPUs), until today. However MCA is computationally limited by the number of polygons [Parker et al. 1998, p. 1] defining the isosurface and the time it takes to render.

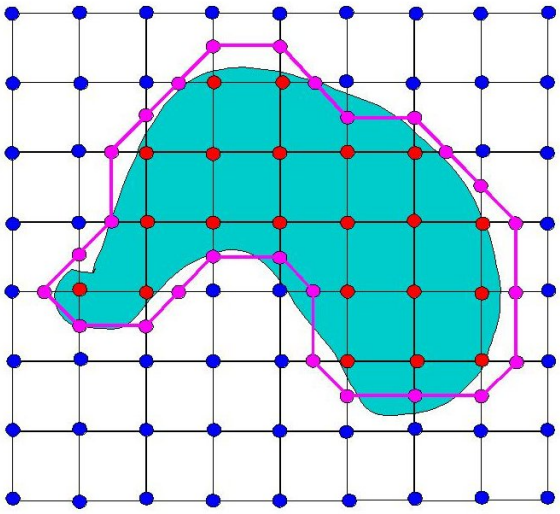

Figure 2: *Marching cubes algorithm surface approximation in 2D. Each red dot is a data point representing the green object in a constant density data grid. Edges connecting red and blue vertices are used by MCA to form the purple approximation[Anderson 2016].*

2.1 Ray Casting

At the same time, isosurface rendering was accomplished using ray-casting, enhancing the scaling towards bigger volumetric datasets because the computation is view-port oriented. This technique calculates direct intersections between view-port rays and the isosurface, allowing to skip the explicit extraction of the dataset geometry. Figure 3 shows the principle for a single pixel. The main advantage for the scientist using this technique is the noticeable speed up for large data sets, which allowed real-time interaction with the simulation [Parker et al. 1998, p. 4]. Raycasting, enhanced by modern GPUs, is a state-of-the-art technique today for interactive volumetric dataset rendering[Frey et al. 2013, p. 2].

2.2 Extension of Ray Casting

The basic principle of ray-casting, where for each ray, having its origin at the configured camera, the color value and the depth was stored, was extended in further research. The new approach does not only store a single pair of information for each ray but expands to a supersegment of data containing tuples of color and depth values. First, this was accomplished using multiple camera views. A simple setup is explained in Figure 4 This technique is called Layered Depth Image (LDI). This technique was originally used to render complex materials such as synthetic objects or skin. [Fernandes et al. 2014].

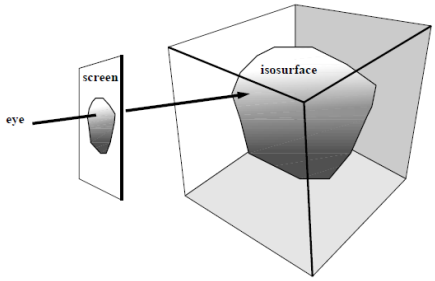

Figure 3: *Viewport centric ray-casting. The intersection between the eye and the isosurface is calculated. In contrast to the MCA, no explicit surface is computed. [Parker et al. 1998].*

Layered Depth Imaging was extended to Volumetric Depth Imaging (VDI) as a generalization for LDI for volumetric data sets, where each ray represents a horizontally segmented pyramid. Each frustum in the cut pyramid contains pairs with information about its depth and color. The main advantage for data set visualization with this technique is that VDI image data is not bound to its original data set [Fernandes et al. 2014], allowing simulation-agnostic viewport changes.

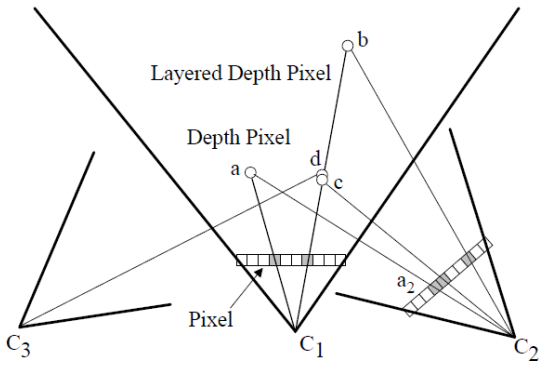

Figure 4: *Layered Depth Ray Casting (LDRC) is an extension of the classic ray-casting method where, very similar to the original ray-casting, information was collected from different cameras (C2, C3) and then aggregated to the LDI camera (C1). Images can then later be reconstructed close to the LDI camera frame. [Shade et al. 1998, p. 6].*

2.3 Depth Maps

A depth map, also called z-buffer, is a 2d array of pixels, the image space, where a 3d space is mapped into. A mapping is created by measuring the distance from a 3d space point to a predefined reference pane, usually near the camera. Depth maps are view dependent because only one scalar value per pixel is maintained of the object in 3d space. Despite its limitations, depth maps have several advantages:

- Depth maps allow local surface normal estimation by calculating the cross product of two pixel intensity gradients. From that, a surface can be approximated and reconstructed in 3 dimensions. [Ye et al. 2015]

- Normal vector fields can then be used to estimate lightning [Ye et al. 2015]
- Normal vector field can be used for feature extraction and tracking. [Ye et al. 2015]
- Depth maps reduce the amount of data required to render a surface by making it needless to calculate an intermediate representation because they can be calculated trough ray tracing, which, despite conventional wisdom, becomes feasible for very large data sets. [Parker et al. 1998, p. 1] This reduces the amount of data to generate a visual representation drastically.

2.4 Feature Extraction and Tracking

Feature extraction and tracking are different techniques but closely related and therefore often mentioned together. Feature extraction algorithms try to find elements with distinct attributes and aspects of a scene or simulation and try to isolate them as a subset of the given the simulation space. Feature tracking is required in time related research where a feature is given as a function of time. Therefore it has to be tracked trough all the frames, even when attributes, such as the shape change.

The need for feature extraction and tracking has increased drastically during the last years because the increase in simulation scale leads to several issues: Due to the complex nature of the data, it is increasingly harder to follow features by eye-tracking. Because of that, without proper visualization, important elements of a simulation can be overseen. Not only is it harder for humans to understand the data, it is technically more challenging and expensive to process data, one strategy is to reduce the processed data right away by tracking features of interest. [Wang et al. 2013]

Feature tracking itself is a non trivial task because simulations are processed on clusters with thousands of calculation units where each unit operates on an assigned part of the dataset. Thus, it is highly possible that features extend to several, or move in between parts of datasets assigned to different calculation units. To overcome this issue, an algorithm that operates on the final, composed image was proposed by Wang et al. 2013 and is explained in chapter 5.1.7 on page 6.

3 Large Scale Simulations

A basic visualization pipeline is usually physically separated by concerns. This is necessary because the modularity is required to guarantee horizontal scaling to process datasets with different sizes within an acceptable duration, therefore the computational power has to be adjustable. Remember that overscaling has to be avoided due to the significant cost of simulation power.

Because of these requirements, basic large scale visualization system tend to consist of the following components:

- 1 visualization node
- $1...n$ simulation nodes
- usually a scientist's desktop

Figure 5 shows the loose coupling in between the scientist's desktop and the visualization machine as well as the increased (red) bandwith required in between the simulation storage and the visualization machine.

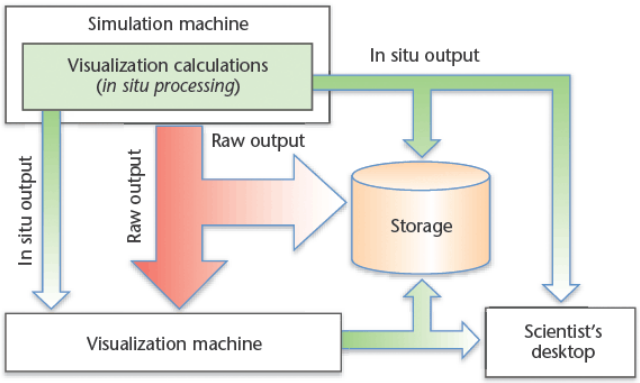

Figure 5: *In situ data is processed on the simulation machine and then decoupled and sent to the scientists desktop. (right).[Ma et al. 2010].*

4 In-Situ Visualization

The need for in-situ visualization came up because the amount of data produced by the simulation nodes exceeds todays storage capability. Further, as the simulation scale increases, the amount of parallelization increases as well. Basic algorithms such as the Marching Cubes Algorithm and ray-casting can still be applied to the dataset, however due to the distribution of the simulation logic, a visualization can not immediately be produced.
However, in many cases, scientists are in need of direct feedback to work efficiently, otherwise the time to gather meaningful results is increasing significantly.

In-situ visualization is a direct approach to improve the feedback and interaction from and to the simulation. The basic concept of in-situ visualization is that the simulation cluster does not only perform the simulation, but contains a part of the visualization algorithm as well, producing in-situ visualizable data.

The data produced by the decoupled simulation is transfered to the scientists machine as in the architecture described in Figure 5. This brings different issues by concept:

- Most of the time, the researcher desires to explore the simulation time and space domain. This requires a camera pose or similar parameter change. Such parameters have to be distributed to all the nodes.
- There is a computational cost for the visualization on each simulation node.
- There is a transfer cost, including compression, transmission and decompression of the visualization data. Moving data is expensive, therefore different concepts for compression are applied independent on the visualization method.[Ye et al. 2015]

To overcome these issues, two main concepts are considered. One is to use a small fraction of the simulation node to process visualisation data. This cost must be reasonable small to not interfere with the simulation. As a rule of thumb, $<5\%$ of the computational capacity should be used for visualization [Ye et al. 2015]. It is important that this cost is, in regard to execution time on each node, as homogeneous as possible. If not, the simulation is going to desynchronize or will be bound to the slowest node. This strategy is called tightly-cupled synchronous processing.
Processing can be losely-coupled (also denoted as in-transit), which means that data is moved first to another node and then processed

there. The drawback is that the information cannot be reduced beforehand, resulting in bigger transfer costs.

Currently, in-situ visualization is already integrated into toolkits such as ParaView Catalyst, VisIt [Ye et al. 2015]. However this functionality is limited to static images.

5 Pipelines

In the following chapters, two pipelines implementing different concepts, depth maps and volumetric depth images, are examined in detail. The main concerns are the general pipeline engineering and the advantages and disadvantages of each layout. The different building blocks are separated and described individually to make it simpler for newcomers to understand the impact of each. Table 1 gives a brief overview.

	depth maps	**volumetric depth images**
capturing	ray casting	ray casting
network layer	2-3 swap composition	delta encoding hufmann / LZ4 compression
viewport	fixed	movable in proximity to original camera
simulation type	isosurface	volumetric data
in-situ simulation adaption	isovalue range camera position	-

Table 1: *Concept summary of two in-situ visualization approaches.*

5.1 First Approach: Depth Maps

One strategy for in-situ visualization is based on the usage of depth maps. On each of the simulation nodes, depth maps are calculated using ray-casting. Ray-casting is favored because of the computational performance advantages over marching-cubes when applied on large data sets, but marching-cubes could be used as well.

On each cluster, for a given moment in time space, several isosurface maps are created for different isovalues. The isosurface range is part of the user interaction with the simulation. The researcher is able to update $[v_{min}, ..., v_{max}]$ as part of the configuration file. This is called adaptive isovalue selection. This procedure is directly applied during ray-casting, where each ray is tested against the isovalues. If the ray intersects a isosurface $I(v_{iso})$, the depth is saved to $DM(v_{iso})$.

The simulation space is split up into $k \times k \times k$ cubes. Each cube is assigned to a different calculation node.

After a simulation step is calculated, the 2-3 swap algorithm is used to combine the partial depth maps from each node n at isovalue v. This is accomplished by assigning the new pixel depth $p_{x,y}$ for the combined image $p_{x,y} = min(DM_{n_1,v}, DM_{n_2,v})$ for two example nodes n_1, n_2 [Ye et al. 2015].
The main disadvantage of this technique is a performance loss whenever many layers are stacked on top of each other due to the nature of the simulation or the camera position. In this scenario, all calculation except the foremost one are discarded.

5.1.1 Processing the Combined Depth Layers

These combined depth maps, containing the different depth layers for each isovalue within range, are then transmitted to the visualization machine. The computationally less demanding post processing is applied on a single calculation unit, decoupled from the simulation cluster. The main data flow is described in Figure 7

5.1.2 Tuning: Adaptive Isovalues

Complex structures extend to more than one iso surface layer. Due to the discrete isovalues, the simulated results does not allow to surf continuously in between the depth layers. Therefore, it is in the users interest to estimate a guess of the isosurface in between given depth layers.
A good estimation of the isosurface in between $L_1 = DM_{n,v}$ and the next layer $L_2 = DM_{n,v+1}$ is to intersect the surface normals of layer v with layer $v + 1$ and store these intersections. Using surface normals for this task produces more accurate results than a plain pixel mapping because normals derive the direction flow of a structure, which is usually not directly at the same location (and therefore the same pixel). Intermediate isosurfaces are calculated on the fly by scaling the intersecting vector with a constant factor $(0, 1)$.
The interpolated points build a point cloud in between the two depth layers. To render the point cloud, coordinates of the point cloud are projected onto the image space and then for all 2d coordinates in a specific (user defined) radius around a pixel the weighted average depth is calculated and set as the new depth of that pixel. [Ye et al. 2015].

The algorithm used to generate the interpolation segment is proposed and explained in detail in the paper of Ye et al. 2015.

5.1.3 Post Processing

Direct rendering of isosurfaces does not yield to human readable results because shadowing and therefore depth perception is missing. Figure 6 shows the advantages of postprocessing. Post processing of isosurfaces is done in two steps. The first step is to calculate the surface normals of the given depth layer.
When using adaptive isovalues, the surface layers are already there for further processing. Surface normals can be color encoded (ex. Figure 6), however a sense of depth is still not given.

The second step to achieve depth perception is to linear interpolate the surface normal vectors per pixel. and then apply a reflection model to each pixel. This algorithm was named after its inventor Phong shading.

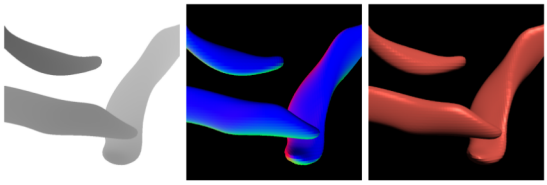

Figure 6: *The importance of preprocessing is obvious when looking at the intermediate output of the preprocessing pipeline. First, three features of a depth layer collected from a simulation (left). A colorized normal map generated from the depth map (direction is encoded with color, middle). And the same features with applied phong shading (right). [Ye et al. 2015].*

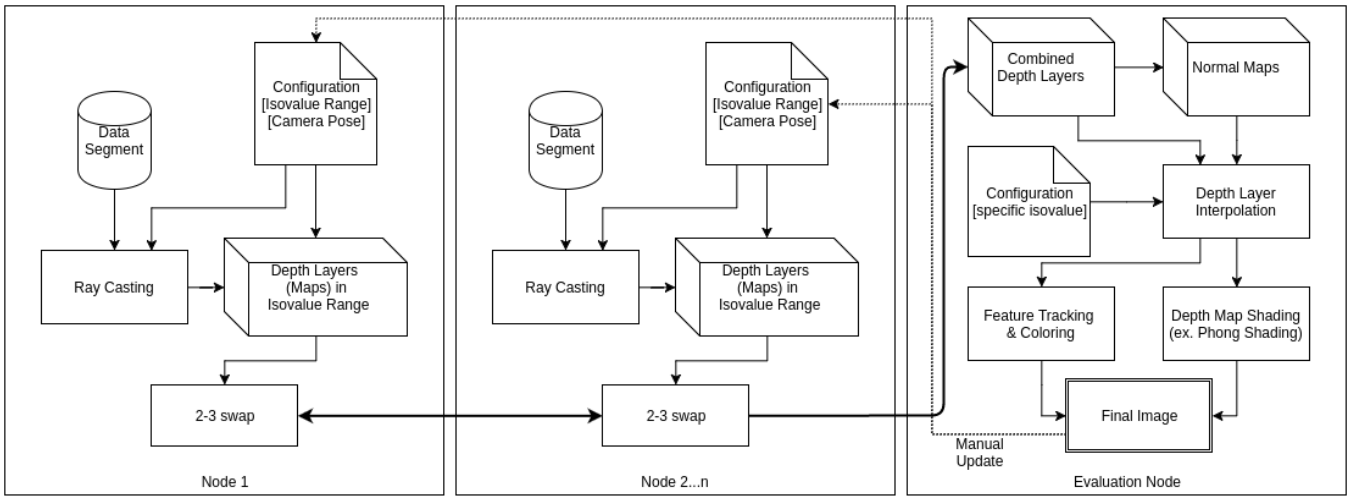

Figure 7: *Draft of a depth map based pipeline for in-situ rendering. The partial depth layers are simulated on node $1..n$ and then later combined according to the 2-3 swap protocol. The combined depth layers are sent to the evaluation node where the post processing is applied. Configuration files can be updated for the next simulation steps if required.*

5.1.4 Feature Extraction

Visualization of data fragments is only partially helpful if the resulting image is very crowded. Therefore, there is need for a selection mechanism filtering elements from the view if they do not belong to the same feature.

Ye et al. 2015 provide an algorithm to extract features by using the fact that isosurfaces are usually continuous in time, image and depth space. The algorithm does connect depth layer fragments in a greedy manner, by selecting surrounding elements when they are closely related to the original point, meaning they are in a user defined boundary. The fragments are then assigned by an id. This is done simply by iterating over the pixel array, including all pixels that are smaller than a specific depth threshold t. The other values are ignored (set to 0) and therefore considered to be background. The array iteration is visualized with figure 8.

Due to the dynamic nature of these features, two cases must be considered:

- If two segments with different ids merge, one of the ids is taken over to the new segment, the other id is discarded.
- If a segment does split up, the two new segments keep the same feature id.

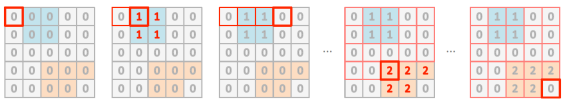

Figure 8: *The figure shows the execution of the feature detection algorithm on a 5×5 pixel array of a depth map. All pixels that are smaller than a user defined threshold t are excluded from the feature tracking and considered to be back ground. The red square is the current execution pointer. Whenever the pointer is inside of an untagged feature, it expands to all connected pixels before going on with the next iteration step. Already assigned pixels are ignored during the iteration [Ye et al. 2015].*

5.1.5 Concrete Implementation

Depth map in-situ visualization was implemented by Ye et al. in 2015. Tests were performed in two different scenarios:

- FFV-C: A $3D$ thermal flow simulator (FrontFlow / violet Cartesian)). A jet engine simulation was used.
- JHTD: A turbulence data set from the John Hopkins Turbulence database. 1024 time steps, $1024 \times 1024 \times 1024$ velocity field.

For the FFV-C simulation, different modes were simulated according to table 2 to test the scalability regarding total simulation volume, subvolume size and image size. The main results are that:

- The simulation time, local render time and image composition time correlate slightly with the subvolume count, due to network traffic.
- The simulation time does not depend on the view-port image size. This is because all volumes are simulated for every time step.
- The image composition time is robust to subvolume voxel count changes. This is related to ray-casting.
- The node render time and image composition time increases linearly with the image size. This is related to ray-casting.

The JHTD data set was used to test the feature tracking. It was possible to track a large vortex trough 1024 time steps.

MPIP	IS (n^2)	NVS (n^3)	TVS (n^3)
8	512	$4-64$	–
512	$64-1024$	32	256
512	512	32	$64-256$

Table 2: *FFV-C simulation runs: MPIP = MPI Processes, IS = image size, NVS = node volume size: the subvolume assigned to a computation node, TVS = total simulation volume size*

5.1.6 Rendering Limitations

Limitations of the depth map in-situ rendering are mostly connected to the fact that the visualization is strictly view dependent. That means the user is not able to explore the simulation space by altering the camera pose. Because of that, it is still necessary to save some simulation data for post-simulation exploration. [Ye et al. 2015, p. 6]

Depth maps only capture the foremost layer of the isosurface, therefore losing information about all the subsequent ray intersections with the surface behind its first occurrence. This drawback is most unfortunate when the structure is heavily folded, having key features in all 3 dimensions.

The proposed interpolation algorithm is prone to artifacts when the point cloud density is low.

5.1.7 Feature Tracking Limitations

The feature tracking algorithm is calculated on the image space, therefore, whenever features overlap in a camera configuration in a way that feature F_a splits feature F_b in two pieces in the image space (for example F_a is a long small segment on top of F_b, separating the view port in a top and bottom half, F_b is behind F_a), F_b is not correctly recognized as $[F_{b,1}, F_{b,2}]$ but as F_b and F_c.
Features that are not defined on exactly one isosurface are not tracked by design.
Aggregated quantities that depend on the 3d structure of the feature cannot be calculated (for example the isosurface volume). [Ye et al. 2015]

5.2 Second Approach: In-Situ Volumetric Depth Images (ISVDI)

A different approach to depth maps are ISVDI. ISVDI were used by Fernandes et al. 2014 to render volumetric data. Ray-casting is adapted to gather VDI data of a specific time step as described in chapter 2.2 on page 2. This was done because the volumetric output data (opposed to isosurfaces in the first approach) produced by each node required the architecture of the cluster to have higher transfer bandwidth than what was currently available, effectively binding the simulation scale to the cluster network bandwidth. The main focus of Fernandes et al. 2014 was to reduce network traffic while allowing the user to explore the simulations space and time domain without a round trip to the cluster.

The overall pipeline is derived from the traditional in-situ setup as described in figure 5 with N simulation nodes and 1 visualization portal. Fernandes et al. 2014 introduced delta encoding of ISVDI images to reduce the network bandwith required. The extended layout is displayed in figure 9.

Visualization viewport changes are calculated on the visualization node. The new image is created by interpolating pixel frustums of the new camera pose with the frustums of the LSVDI camera as in figure 10.

5.2.1 Limitations

Limitations of the LSVDI techniques can be separated in concerns regarding the technique and general disadvantages. The technique itself is limited in space by a specific angle, which is usually $< 50°$ but, depending on the data can be extended to up to $90°$. Details can be less trusted depending on the angle. The loss of detail is caused by the interpolation algorithm and therefore the segment size [Frey et al. 2013]. In general, the most noticeable disadvantage is that all data is trivially limited to parts covered by the camera

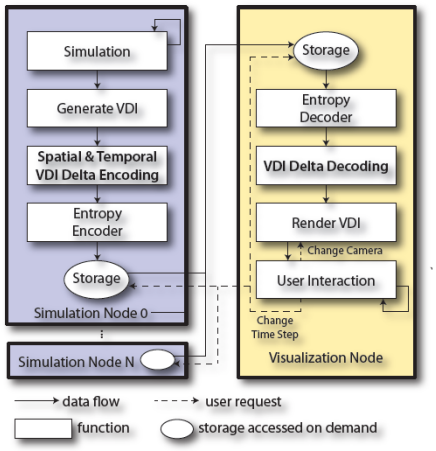

Figure 9: *VDI Pipeline for in situ rendering. (replace image)[Fernandes et al. 2014].*

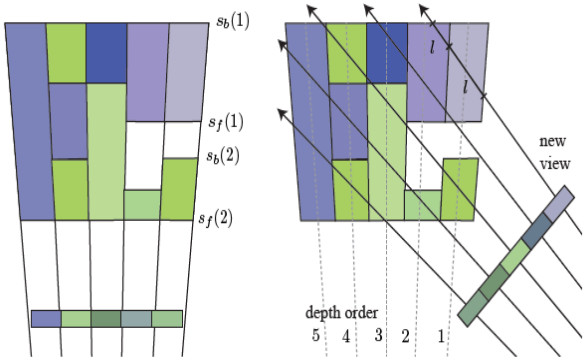

Figure 10: *VDI frustums (left) and an altered viewport image reconstruction (right). The graphic represents a cut trough the frustums for a pixel row (image space x-axis). The different colors are depth segments (simulation space x-and y-axis) recorded from the simulation. [Frey et al. 2013].*

pose. This is similar to the depth maps approach but more noticeable because the user is able to alter the visualization perspective. This leads to perception issues where an object seems to be isolated, but is not because it extends the simulation camera pose.

6 Distributed Composition & Transmission

Network traffic is, looking at current state of the art technology, one of the parallel computing bottlenecks. Therefore it must be the users concern to correctly distribute and collect all parts of the simulation. Even though is out of scope of this work to introduce these network algorithms in detail, some key elements are summarized in the look up table below and should point to further information. This list focuses exclusively on how physically separated computation units interact in both of the explained pipelines above.

While images must be composed in the first place, bandwidth can be reduced even further by compressing the data and by calculating delta information between time steps.

Image composing concepts:

- Direct Send: Direct send is the simplest image composing method. Each simulation node communicates directly with the image composing node. This results in a worst case of $N \cdot (N-1)$ transfers. [Yu et al. 2008]
- Binary Swap: To prevent the worst case of direct send, binary swap organizes the image composition more strictly. Two nodes are coupled and one of them is responsible for the composition. The biggest drawback of this method is that the simulation cluster is required to have a power of two nodes [Yu et al. 2008].
- 2-3 Swap: Is an image composing algorithm that extends the binary swap algorithm. The 2-3 swap algorithm allows any number of calculation nodes to work together [Yu et al. 2008].

Data compression and encoding:

- VDI Delta encoding: VDI rays are grouped together and stored in a tree structure. In this tree, changes are detected and the delta is transmitted to connected nodes [Fernandes et al. 2014].
- Hufmann Encoding: It was shown that, with help of a tree data structure, data can be mapped to code words with variable length. Symbols that are used more frequently are mapped to shorter words, increasing the efficiency and therefore compressing the message. Huffmann Encoding was first published by its inventor David Huffmann in 1952.
- LZ77 Encoding: The main concept of LZ77 encoding is to cut the message whenever a sequence is repeated within a window. This technique was introduced by Jacob Ziv and Abraham Lempel in 1977.

7 Summary and Discussion

7.1 General Concerns

In-situ visualization is still a very young field of research. Most concepts are not older than 5 years and there is not much explanatory, theoretical literature available. To find the best in-situ visualization technique for a new project, one relies heavily on papers. Besides depth maps and VDI, more concepts were developed simultaneously but are not covered in this paper. One approach is to simply extend the concept of a single simulation camera and use multi-viewport data to reconstruct the simulation.

Scientists optimizing systems for exa-scale simulations face, beside its intrinsic complexity, yet another barrier which is the hardware cost of these systems. While the depth map approach was tested on a larger super computer (up to 1024 compute nodes [Ye et al. 2015, p. 6]), other approaches that are similar (ex. Biedert et al. Contour Tree Depth Images For Large Data Visualization or ISVSDI) have yet to be tested on large scale systems. Therefore its hard to say if these methods do scale without complications.

In this paper, despite the recognition of its importance, the network layer is not the main focus, however, due to the slow network architecture compared to the speed of GPUs and CPUs, it can easily become the bottleneck of a simulation.

7.2 Concerns about the Pipelines

Both pipelines introduced in this paper provide unique advantages. While the LSVDI pipeline allows the user to alter the camera perspective for closer examination, the depth maps pipeline introduces many extended features such as isosurface interpolation, dynamic isovalue range updates and feature tracking that can be included with minimal changes to the design. This brings the benefit that, with the knowledge of the underlying system, issues with new features can be resolved quickly.

As an extension I suggest the combination of both pipelines since they extend basic ray-casting. This can be done by not just recording one intersection for a depth map when sweeping a specific isovalue. Similar to the strategy in chapter 2.2 on page 2 an intersection point could be stored whenever the ray crosses isosurface. For each pixel, not only one depth value v would be stored but an array of values $[v_1, ..., v_n]$. From that data, a new viewport could be computed on the visualization node. On that viewport, all benefits of the depth map pipeline, such as feature coloring and layer interpolation can be computed.

References

ANDERSON, B., 2016. An implementation of the marching cubes[1] algorithm. http://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html. [Online; accessed 11-12-2016].

BIEDERT, T., AND GARTH, C. 2015. Contour tree depth images for large data visualization. In *Proceedings of the 15th Eurographics Symposium on Parallel Graphics and Visualization*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, PGV '15, 77–86.

FERNANDES, O., FREY, S., SADLO, F., AND ERTL, T. 2014. Space-time volumetric depth images for in-situ visualization. In *2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV)*, 59–65.

FREY, S., SADLO, F., AND ERTL, T. 2013. Explorable volumetric depth images from raycasting. In *Proceedings of the 2013 XXVI Conference on Graphics, Patterns and Images*, IEEE Computer Society, Washington, DC, USA, SIBGRAPI '13, 123–130.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (Aug.), 163–169.

MA, K.-L., YU, H., GROUT, R. W., WANG, C., AND CHEN, J. H. 2010. In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications 30*, undefined, 45–57.

MA, K.-L. 2009. In situ visualization at extreme scale: Challenges and opportunities. *IEEE Comput. Graph. Appl. 29*, 6 (Nov.), 14–19.

PARKER, S., SHIRLEY, P., LIVNAT, Y., HANSEN, C., AND SLOAN, P.-P. 1998. Interactive ray tracing for isosurface rendering. In *Proceedings of the Conference on Visualization '98*, IEEE Computer Society Press, Los Alamitos, CA, USA, VIS '98, 233–238.

POLSSON, K., 2015. Chronology of ibm personal computers: 1987. http://pctimeline.info/ibmpc/ibm1987.htm. [Online; accessed 11-12-2016].

SHADE, J., GORTLER, S., HE, L.-W., AND SZELISKI, R. 1998. Layered depth images. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 231–242.

TCBG, 2013. Theoretical and Computational Biophysics Group (TCBG): Press Release Highlights TCBG's HIV Capsid Re-

search. http://www.ks.uiuc.edu/Publications/Stories/hiv/. [Online; accessed 11-11-2016].

WANG, Y., YU, H., AND MA, K.-L. 2013. Scalable parallel feature extraction and tracking for large time-varying 3d volume data. In *Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, EGPGV '13, 17–24.

YE, Y. C., WANG, Y., MILLER, R., MA, K.-L., AND ONO, K. 2015. In situ depth maps based feature extraction and tracking. In *Proceedings of the 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*, IEEE Computer Society, Washington, DC, USA, LDAV '15, 1–8.

YU, H., WANG, C., AND MA, K.-L. 2008. Massively parallel volume rendering using 2-3 swap image compositing. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, IEEE Press, Piscataway, NJ, USA, SC '08, 48:1–48:11.