Eurographics Conference on Visualization (EuroVis) (2015)
E. Bertini, J. Kennedy and E. Puppo (Editors)

WebGL-Enabled Remote Visualization of Smoothed Particle Hydrodynamics Simulations

Jennifer Chandler[1], Harald Obermaier[1], and Kenneth I. Joy[1]

[1]University of California, Davis

Abstract

Large-scale simulations are often performed on machines without the necessary graphics hardware for visualization. Transferring full resolution data to a suitable machine for visualization is impractical and undesirable. We investigate solutions to the remote visualization problem for large-scale Smoothed Particle Hydrodynamics (SPH) simulations. Previous remote visualization strategies for SPH perform rendering on the server side and send rendered images to the client viewer. These approaches suffer from delays due to network latency in sending entire images every frame and adversely affect interactive visual data analysis. WebGL enables hardware acceleration for rendering in the browser. We combine WebGL volume rendering rendering with data compression and intelligent streaming to provide a fast and flexible remote visualization solution for SPH simulations, which enables easier access to simulations for analysis and sharing of data.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Remote systems

1. Introduction

Smoothed Particle Hydrodynamics (SPH) is a fluid simulation method that performs computations on a moving set of particles carrying physical properties of the fluid [GM77]. SPH simulations are often created on remote high-performance machines that may not have hardware for interactive rendering, necessitating remote visualization. A system that streams data from a server and renders it on the client is desirable to avoid indiscriminately downloading full resolution data. Users should be able to view simulations from many computers without specific client software.

Using WebGL for client-side rendering of SPH simulations meets these requirements. WebGL, based on OpenGL ES, allows web browsers to directly use the graphics card without plugins or extensions [Mar11]. WebGL is widely supported, making it suitable for use in a variety of portable remote visualization techniques such as in medical visualization [BMSP12], [JKD*12], [ML12b], [MJ12] and geospatial data visualization [XYL12], [PID12], [Slo11].

Client-side rendering of remote data suffers from delays in data retrieval due to network latency. Lavoué et al. [LCD13] address this concern for 3D meshes with a progressive compression scheme to load a low resolution mesh and fill it in with more detail as data becomes available. We use an octree as an efficient multi-resolution representation for SPH data so we can stream in low resolution versions for rendering while higher resolution versions are loading.

The proposed remote visualization system makes it easier to share results and is also suitable for in-situ visualization since data can be viewed remotely while the simulation is running. Our contributions are as follows:

- A compact streaming system for SPH visualization.
- A WebGL-based multi-resolution SPH volume renderer.

Our work adapts many existing SPH visualization techniques for a remote visualization system using WebGL.

2. Related Work

Visualization of SPH simulations commonly takes the following approaches: volume rendering of physical properties, rendering isosurfaces of fluid boundaries or physical variables of interest, and rendering column density, the integral of the density through the view direction. Goswami et al. [GSSP10], Linsen et al. [LMD*11], and Price [Pri07] implement a variety of these techniques for SPH visualization.

Very little work has been done in the area of SPH remote visualization. Feng et al. [FCDM*11] implement a web interface to view pre-computed snapshots of SPH data. Cui

DOI: 10.2312/eurovisshort.20151116

et al. [CMP13] use animated depth images to render a finite element analysis scene on the client from a variety of nearby view points and extend their approach to render SPH particles as spheres. Soumagne et al. [SBC10] do in-situ visualization of SPH simulations using a ParaView plugin that reads the streamed data from distributed shared memory.

SPH volume rendering techniques generally take two forms: splatting particles onto the image plane or into slices, and converting to a grid representation by evaluating the SPH kernel in each cell. Fraedrich et al. [FAW10] do the former, using the geometry shader to splat particles into slices of a 3D texture. In other work, Fraedrich et al. [FSW09] use an adaptive octree approach to stream large SPH data from disk. They combine nearby particles in coarser levels of the tree. We use an approach similar to Reichl et al. [RTW13] who create an octree by evaluating the SPH kernel for each grid cell at the highest resolution and averaging child cells up the tree. They render the octree using CUDA. For the WebGL volume rendering, we use a texture representation of the octree. Benson and Davis [BD02] first developed octree textures. Lefebvre et al. [LHN05] and Kniss et al. [KLS*05] adapt octree textures for texturing 3D models on the GPU. Boada et al. [BNS01] use an octree to reduce texture memory for rendering volume data. Campoalegre et al. [CNB13] create a gradient octree for streaming based on predefined transfer functions. Our approach is adapted from Gobbetti et al. [GMG08] who use an octree for ray casting volume data too large to fit in memory. Movania et al. [ML12a], Noguera and Jiménez [NJ12], and Congote et al. [CSK*11] provide algorithms for ray casting [Lev88] in WebGL using a tiled 2D texture of the 3D uniform grid data.

3. Background

SPH is a particle-based fluid simulation technique. Because of the dynamic nature of the particle set, it is easy to include moving obstacles or free boundaries in the flow field. Each particle carries data such as density and velocity as well as a smoothing length $h \in \mathbb{R}$ representing the particle's radius of influence, which can vary per particle. Near incompressible fluid simulations commonly use a single smoothing length.

To reconstruct values at arbitrary locations in the flow field, SPH uses a kernel function to weight the contribution of each particle based on the distance $\|\mathbf{r}\|$ from the sample location to the particle. Only particles within one smoothing length h are used for reconstruction. We use the same kernel function used in the simulation [MCG03]:

$$W(\mathbf{r},h) = \frac{15}{\pi h^6} \begin{cases} (h - \|\mathbf{r}\|)^3 & \text{if } 0 \leq \|\mathbf{r}\| \leq h \\ 0 & \text{if } \|\mathbf{r}\| > h \end{cases} \quad (1)$$

The value at a sample location of an arbitrary quantity is the weighted sum of the quantity's value per particle.

4. Server-Side Preprocessing

In preprocessing we construct an octree for each time step using a process similar to that of Reichl et al. [RTW13]. We use the smoothing length as the cell size of the lowest level of the octree. According to Reichl et al. this level of resolution can capture the fine details of the data. We compute the contribution of the particles to the cell centers using the SPH kernel weighting function (Equation 1) and store the gradients using the derivative of the kernel function. We compute the values for cells in higher levels by averaging the values of their children. Figure 1 illustrates this process.

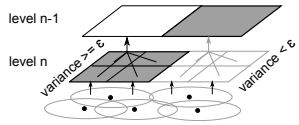

Figure 1: *We insert particles at level n by evaluating the SPH kernel for the desired scalar. The parent stores the average of its children. If the variance of the children is less than ε, we delete them and make the parent a leaf (gray).*

During the averaging we also perform data compaction. If the variance of the children's values and the magnitude of their gradients is less than a user-supplied threshold, we delete the children and make the parent a leaf. It is possible to rely only on variance; however, we found that in largely uniform regions with sharp transitions this can cause discontinuities when rendering because neighboring nodes of the octree may differ by more than one level, causing interpolation problems. Using the gradient magnitude as an additional threshold preserves continuity by disallowing compaction if the values change rapidly in neighboring nodes.

We store the octree in breadth first order with one file for each level. Each node has the scalar value and a flag that is zero for a leaf or contains the offset of the node's block of children in the array for their level. The level 0 file contains the root node and header information for the octree, including the bounding box size and number of levels.

5. Client-Side Viewing

We use WebGL to provide interactive volume rendering for remote SPH data. WebGL has some limitations: 3D textures are not supported and integer texture samplers are not available in shaders. We adapt our algorithms accordingly.

To load a time step, we send an asynchronous XML HTTP request for level 0 of the octree. Once the first request finishes, we request the next level and so on until we load the entire tree. We store an array for each octree level. Since we already stored the child offsets in preprocessing, we don't need to update the parent nodes when we read in a new level. Changing time steps before all data has loaded cancels pending requests for level files and starts loading the new time step. We do loading and processing in a separate thread with Web Workers [Hic12] to avoid stalling the UI. Communication with the worker occurs via message passing (Figure 2).

After loading each level we compute links to each node's neighbors in all 6 directions, 3 links to sibling nodes and 3 to nodes not contained within the parent (Figure 3). Linking

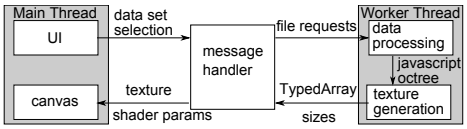

Figure 2: *We use Web Workers for loading and processing in the background to avoid disrupting the UI. Since JavaScript is single-threaded, without Web Workers, any processing will freeze the rendering. The threads run independently and communicate by sending messages through a handler.*

only uses the current level of the octree and its parent level. We assume neighbor links for the parent level were already computed. A neighbor link is always at the same or a higher level than the node. To compute a link pointing outside the parent, we find the parent's neighbor link in that direction. If it is a leaf we use it as the neighbor otherwise we use the appropriate child of the parent's neighbor.

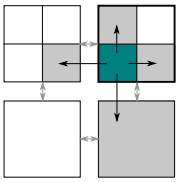

Figure 3: *Neighbor links for the teal node. We can easily find sibling links. For a link in a different parent, we first find the parent's neighbor in that direction (gray arrows). If it has children we link to the appropriate child (left link), otherwise we link to the parent's neighbor (bottom link).*

We present two methods for volume rendering: direct octree rendering and conversion to a uniform grid. Both methods use 8 bits per pixel textures.

Octree Texture Layout: We write the octree to a 2D texture with one texture for storing the nodes and a second for storing leaf data. For larger octrees we use multiple leaf textures. Using separate textures allows us to linearly interpolate in the data texture and use nearest neighbor in the node texture. Like Gobbetti et al. [GMG08] we store a grid in each leaf instead of a single value to use hardware interpolation (Figure 4). In the node texture, the alpha channel indicates leaf status. The RGB channels store an address: for internal nodes the texel containing the first of the node's children, and for leaf nodes, the location of the leaf grid. Since our file representation of the octree contains a single value in the leaves, we construct a grid by converting parent nodes of leaves into leaf nodes and using the child values in the grid. For each grid we also store ghost cells, computed using neighbor links to avoid backtracking, so that we can interpolate samples without additional lookups.

Uniform Grid Texture Layout: An alternative to the previous method is to write the octree to a uniform grid texture in a tiled 2D format by doing a depth first traversal and writing into the texture when we reach a leaf. Each slice has the same number of cells in the x and y direction as the finest resolution level of the octree. We can still display the data as

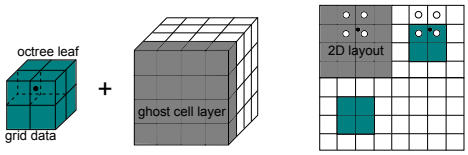

Figure 4: *Each leaf stores a grid composed of the child values plus a layer of ghost cells with neighbor values. To evaluate a sample (black point) we interpolate the cells with white dots. The hardware interpolates in x and y. We sample the two z slices and manually interpolate the values.*

it loads because we make the size of the uniform grid equal to the size of the currently loaded level of the octree.

Comparison: Rendering the uniform grid is simpler than rendering the octree texture since it does not have multiple dependent texture lookups and does not have to compute neighbor links to form leaf grids. The direct octree rendering has the advantage of being able to scale well with larger data sets since it does not need to allocate texture space for the entire grid, especially if high resolution levels are sparse.

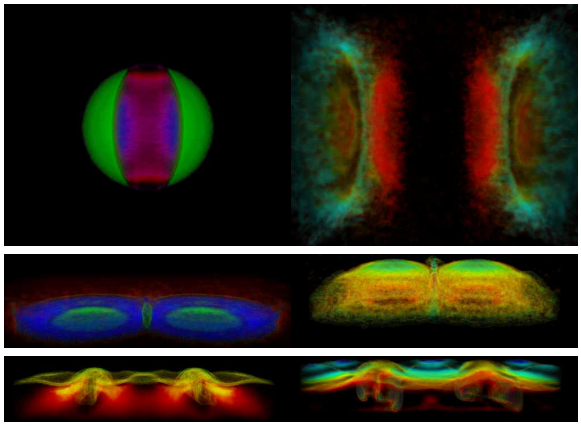

Figure 5: *Top: Sphere collision data time steps 80 and 300. Middle: Sphere drop data set time steps 70 and 120. Bottom: Mixing simulation time steps 5 and 50.*

6. Results

We evaluate our remote viewer with three data sets (Figure 5). The *mixing simulation* (161 time steps, 2,097,152 particles, 106,496 KB / time step) models a blender with two corotating blades located side-by-side in a box. The *sphere collision* (312 time steps, 1,048,576 particles, 53,248 KB / time step) models a gravity-free collision of two spherical fluid elements. The *droplet impact* (193 time steps, 2,097,152 particles, 106,496 KB / time step) models the impact of a pair of highly viscous fluid droplets onto a planar surface. We use the velocity magnitude for rendering.

We tested our application in Chrome and Firefox with an Nvidia Quadro 6000 graphics card. We use Trickle [Eri05] to simulate slower connections. Table 1 shows data transfer times and file sizes for selected time steps. With these transfer rates, we can view the data almost immediately up

time step	5	200	305	5	50	190	5	45	135
level 0	20ms (48B)	25ms (48B)	355ms (48B)	9ms (48B)	8ms (48B)	8ms (48B)	801ms (48B)	21ms (48B)	1.15s (48B)
level 1	6ms (160B)	5ms (160B)	9ms (160B)	6ms (160B)	6ms (160B)	5ms (160B)	6ms (160B)	5ms (160B)	6ms (160B)
level 2	6ms (1.3KB)	6ms (1.3KB)	7ms (1.3KB)	6ms (1.3KB)	5ms (1.3KB)	6ms (1.3KB)	7ms (1.3KB)	6ms (1.3KB)	6ms (1.3KB)
level 3	10ms (10KB)	16ms (10KB)	365ms (10KB)	10ms (10KB)	8ms (8.8KB)	9ms (10KB)	16ms (10KB)	8ms (10KB)	8ms (10KB)
level 4	56ms (70KB)	31ms (61.1KB)	38ms (59.4KB)	30ms (65KB)	25ms (57.5KB)	26ms (63.1KB)	419ms (80KB)	45ms (80KB)	474ms (80KB)
level 5	2.14s (447KB)	476ms (327KB)	812ms (310KB)	598ms (416KB)	416ms (339KB)	578ms (391KB)	1.92s (541KB)	1.14s (618KB)	1.2s (606KB)
level 6	5.27s (3.0MB)	4.09s (2.4MB)	3.98s (1.8MB)	4.72s (2.8MB)	3.36s (2.1MB)	4.51s (2.6MB)	6.51s (3.5MB)	7.17s (4.2MB)	7.39s (4.3MB)
level 7	38.79s (22.1MB)	27.93s (16MB)	18.42s (10.5MB)	34.61s (19.8MB)	24.23s (13.8MB)	26.97s (15.4MB)	35.38s (20.1MB)	47.53s (27.1MB)	46.94s (26.7MB)
level 8	–	–	1.8min (61.9MB)	3min (102MB)	2.9min (98.4MB)	2.2min (77MB)	–	–	–

Table 1: *Data transfer times for selected time steps of the sphere collision, sphere drop, and blender data sets respectively. We cap the download and upload speeds to approximately 5Mbps to simulate a slower connection.*

to level 5, which enables initial interactive viewing and analysis. Octrees for the data sets have different heights due to different levels of compactness. Level sizes of a data set also vary across time steps due to changes in particle distribution.

We measure the average frames per second (fps) across uniformly spaced time steps at level 7 for each simulation. Generally the fps for the uniform grid is higher than the direct octree method. For the collision simulation the uniform grid runs at 52.8fps, and the octree runs at 37.9fps. For the drop simulation the uniform grid runs at 60fps, and the octree runs at 46.8fps. For the mixing simulation the uniform grid runs at 60fps, and the octree runs at 29.5fps.

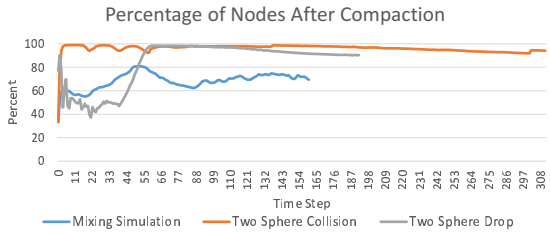

Figure 6: *This chart shows the percentage of nodes remaining in the octree after the compaction step. The more uniform the data set, the more nodes can be compacted.*

The raw SPH data exceeds the size of the largest octree level. The mixing simulation with its large uniform region especially benefits from the data compaction. Here, the total file size of all octree levels combined is only around a quarter of the size of the original SPH data. Figure 6 shows the percentage of nodes remaining after compaction. The level of compaction roughly corresponds to the uniformity of the scalar values in the data. In the drop data set the complexity increases after the impact, decreasing the compaction.

In Figure 7, we show images generated after loading different levels of the octree. Even with fewer levels loaded, the overall shape of the data is still clear. This is helpful for quickly exploring data because the user doesn't need to wait for all levels to load before going to a different time step.

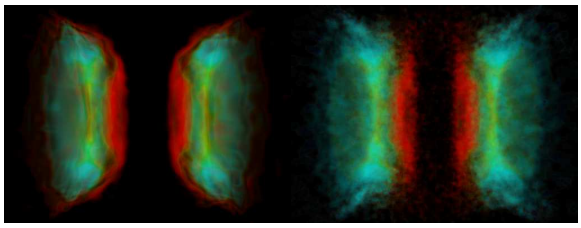

Figure 7: *Sphere collision time step 250: The first image has a maximum level of 5, and the second image has full resolution (level 7). Even at lower resolution the general shape is still apparent which gives the user an impression of the time step quickly before the full resolution data has loaded.*

7. Conclusion

We have presented a remote rendering system for SPH simulations. Our method constructs octrees from the SPH data on the server-side. On the client-side we use WebGL for volume rendering. Our implementation allows the user to view lower resolution representations of the data while the higher resolution levels load. Our experiments show that the emerging WebGL technology is a prime candidate for remote visualization of large SPH data sets. However, limitations present in the WebGL standard as well as bandwidth considerations may require careful adaptation of standard rendering and data representation techniques. In future work we plan to consider performance on mobile systems and additional methods for data compaction. We would also like to explore view-dependent methods for data loading.

8. Acknowledgments

This work was supported in part by the National Science Foundation under contracts IIS 0916289 and IIS 1018097, and NSF GRFP grant DGE-1148897, and the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-SC0007443 through the Scientific Discovery through Advanced Computing (SciDAC) programs Scalable Data Management, Analysis and Visualization Center (SDAV).

References

[BD02] BENSON D., DAVIS J.: Octree textures. In *SIGGRAPH '02 Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (2002), pp. 785–790. 2

[BMSP12] BIRR S., MÖNCH J., SOMMERFELD D., PREIM B.: A novel real-time Web3D surgical teaching tool based on WebGL. In *Bildverarbeitung für die Medizin 2012*. Springer, Mar. 2012, pp. 404–409. 1

[BNS01] BOADA I., NAVAZO I., SCOPIGNO R.: Multiresolution volume visualization with a texture-based octree. *The Visual Computer 17*, 3 (2001), 185–197. 2

[CMP13] CUI J., MA Z., POPESCU V.: Animated depth images for interactive remote visualization of time-varying datasets. *IEEE Transactions on Visualization and Computer Graphics 20*, 11 (Nov. 2013), 1474–1489. 2

[CNB13] CAMPOALEGRE L., NAVAZO I., BRUNET P.: Gradient octrees: A new scheme for remote interactive exploration of volume models. In *2013 International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)* (Nov. 2013), pp. 306–313. 2

[CSK*11] CONGOTE J., SEGURA A., KABONGO L., MORENO A., POSADA J., RUIZ O.: Interactive visualization of volumetric data with WebGL in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology* (2011), pp. 137–146. 2

[Eri05] ERIKSEN M. A.: Trickle: A userland bandwidth shaper for unix-like systems. In *USENIX Annual Technical Conference, FREENIX Track* (2005), pp. 61–70. 3

[FAW10] FRAEDRICH R., AUER S., WESTERMANN R.: Efficient high-quality volume rendering of SPH data. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (Nov./Dec. 2010), 1533–1540. 2

[FCDM*11] FENG Y., CROFT R. A., DI MATTEO T., KHANDAI N., SARGENT R., NOURBAKHSH I., DILLE P., BARTLEY C., SPRINGEL V., JANA A., ET AL.: Terapixel imaging of cosmological simulations. *The Astrophysical Journal Supplement Series 197*, 2 (2011), 18:1–18:8. 1

[FSW09] FRAEDRICH R., SCHNEIDER J., WESTERMANN R.: Exploring the millennium run-scalable rendering of large-scale cosmological datasets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (Nov./Dec. 2009), 1251–1258. 2

[GM77] GINGOLD R. A., MONAGHAN J. J.: Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society 181*, 3 (1977), 375–389. 1

[GMG08] GOBBETTI E., MARTON F., GUITIÁN J. A. I.: A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer 24*, 7-9 (July 2008), 797–806. 2, 3

[GSSP10] GOSWAMI P., SCHLEGEL P., SOLENTHALER B., PAJAROLA R.: Interactive SPH simulation and rendering on the GPU. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2010), Eurographics Association, pp. 55–64. 1

[Hic12] HICKSON I.: Web Workers. W3C Candidate Recommendation, May 2012. http://www.w3.org/TR/workers/. 2

[JKD*12] JACINTO H., KÉCHICHIAN R., DESVIGNES M., PROST R., VALETTE S.: A web interface for 3d visualization and interactive segmentation of medical images. In *Proceedings of the 17th International Conference on 3D Web Technology* (2012), pp. 51–58. 1

[KLS*05] KNISS J., LEFOHN A., STRZODKA R., SENGUPTA S., OWENS J. D.: Octree textures on graphics hardware. In *ACM SIGGRAPH 2005 Sketches* (2005), p. 16. 2

[LCD13] LAVOUÉ G., CHEVALIER L., DUPONT F.: Streaming compressed 3d data on the web using JavaScript and WebGL. In *Proceedings of the 18th International Conference on 3D Web Technology* (2013), pp. 19–27. 1

[Lev88] LEVOY M.: Display of surfaces from volume data. *Computer Graphics and Applications, IEEE 8*, 3 (May 1988), 29–37. 2

[LHN05] LEFEBVRE S., HORNUS S., NEYRET F.: Chapter 37: Octree textures on the GPU. In *GPU Gems 2*, Pharr M., (Ed.). Addison-Wesley Professional, Mar. 2005. 2

[LMD*11] LINSEN L., MOLCHANOV V., DOBREV P., ROSSWOG S., ROSENTHAL P., LONG T. V.: SmoothViz: Visualization of smoothed particles hydrodynamics data. In *Hydrodynamics - Optimizing Methods and Tools* (Oct. 2011), Schulz H. E., Simoñes A. L. A., Lobosco R. J., (Eds.), pp. 3–26. 1

[Mar11] MARRIN C.: WebGL specification. *Khronos WebGL Working Group* (2011). 1

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), Eurographics Association, pp. 154–159. 2

[MJ12] MARION C., JOMIER J.: Real-time collaborative scientific WebGL visualization with WebSocket. In *Proceedings of the 17th International Conference on 3D Web Technology* (2012), ACM, pp. 47–50. 1

[ML12a] MOVANIA M. M., LIN F.: High-performance volume rendering on the ubiquitous WebGL platform. In *IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems* (June 2012), pp. 381–388. 2

[ML12b] MOVANIA M. M., LIN F.: Mobile visualization of biomedical volume datasets. *Journal of Internet Technology and Secured Transactions (JITST) 1*, 2 (Mar./June 2012), 52–60. 1

[NJ12] NOGUERA J., JIMÉNEZ J.: Visualization of very large 3d volumes on mobile devices and WebGL. In *20th WSCG international conference on computer graphics, visualization and computer vision* (2012), pp. 105–112. 2

[PID12] PRIETO I., IZKARA J. L., DELGADO F.: From point cloud to Web 3D through CityGML. In *2012 18th International Conference on Virtual Systems and Multimedia (VSMM)* (Sept. 2012), pp. 405–412. 1

[Pri07] PRICE D. J.: SPLASH: An interactive visualization tool for smoothed particle hydrodynamics simulations. *Publications of the Astronomical Society of Australia 24*, 3 (2007), 159–173. 1

[RTW13] REICHL F., TREIB M., WESTERMANN R.: Visualization of big SPH simulations via compressed octree grids. In *IEEE International Conference on Big Data* (2013), pp. 71–78. 2

[SBC10] SOUMAGNE J., BIDDISCOMBE J., CLARKE J.: In-situ visualization and analysis of SPH data using a ParaView plugin and a distributed shared memory interface. In *5th International SPHERIC Workshop* (2010), pp. 186–193. 2

[Slo11] SLOUP P.: *WebGL Earth*. Bachelor's thesis, Faculty of Informatics, Masaryk University, 2011. 1

[XYL12] XUE L., YANGMING Q., LEI L.: Visualization of geomagnetic environment based on WebGL. In *2012 Fifth International Symposium on Computational Intelligence and Design* (Oct. 2012), vol. 2, IEEE, pp. 274–277. 1