Particle-based Rendering for Porous Media

S. Grottel[1] and G. Reina[1] and T. Zauner[2] and R. Hilfer[2] and T. Ertl[1]

[1]Visualisation Research Center (VISUS), University of Stuttgart, Germany
[2]Institute for Computational Physics, University of Stuttgart, Germany

Abstract

Particle-based modeling and simulation of granular or porous media is a widely-used tool in physics and material science to study behavior like fracture and failure under external force. Classical models use spherical particles. However, up to 10^8 *polyhedral-shaped particles are required to achieve realistic results comparable to laboratory experiments. As contact points and exposed surfaces play important roles for the analysis, a meaningful visualization aiding the numeric analysis has to represent the exact particle shapes. For particle-based data sets with spherical particles, ray tracing has been established as the state-of-the-art approach yielding high rendering performance, optimal visual quality and good scalability. However, when rendering polyhedral-shaped particles, there is no issue with visual quality comparing polygon-based rendering approaches and ray casting, whereas the polygon-based approaches cause significantly lower fragment load. The paper at hand investigates the advantages and drawbacks of both approaches by analyzing the performance of state-of-the-art rendering methods employing vertex-buffer objects, hardware-supported instancing, geometry shader, and GPU-based ray casting.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations; I.3.8 [Computer Graphics]: Computational Geometry and Object Modeling—Applications.

1. Introduction and Related Work

In many fields of science, including physics and material science, particle-based simulation is a well-established tool for studying material properties and material behavior under external forces. Material samples are modeled by large amounts of particles representing discrete entities from the application domain. In molecular dynamics, these are usually atoms, or mass-center-based representations of molecules, e. g. Lennard-Jones mass centers [GRVE07]. These particles have no specific shape, but a simple effective radius, and are thus visualized as spheres. Rendering these is well understood [Gum03] and high-performance algorithms are available. Ray casting has been established as the state-of-the-art approach for rendering this type of data [GRDE10] allowing for interactive visualization of several millions of spheres on standard desktop computers. For quadratic surfaces, e. g. spheres or cylinders, this approach yields best performance, scalability and visual quality, compared to alternatives like texture-based point sprites or mesh-based approaches. For arbitrarily-shaped, curved surfaces the superior visual quality of ray casting is also favorable [KHK*09].

In materials science one is often confronted with materials exhibiting complex stochastic microstructures and textures. Important examples are porous materials [Hil96]. Some classes of porous materials, such as sandstones, exhibit a granular microstructure resulting from the physicochemical processes that generated the material. Simulations of such media often start from models with spherical grains, due to the simplicity of handling interactions. However, for realistic results non-spherical particles are required. These can be modeled by composing a grain out of several spherical sub-grain particles (e. g. employing the discrete element method [JBPE99]). This approach has scalability issues when trying to achieve simulation system sizes of 10^8 grains. This is the required size to be comparable to laboratory experiments. Therefore, current simulations try to employ polyhedral-shaped particles [LBFH10], which, however, complicates the process of evaluating contact points

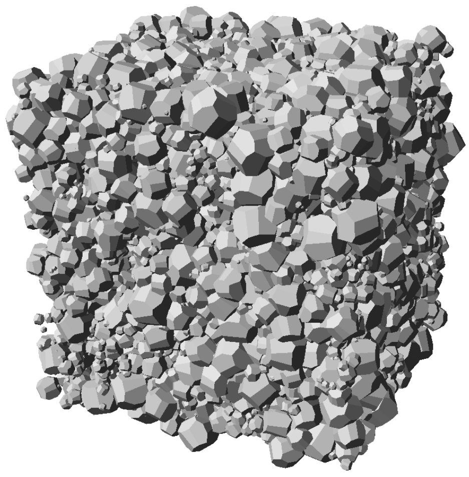

Figure 1: A small porous media sample modeled from 10 000 particles of 100 crystallite template types with 18 faces each.

and forces. To achieve consistency with numerical analysis, visualizing these particles correctly is of high importance.

The scenario of porous media, e. g. sandstone, which the work at hand specifically looks at, is closely related to visualizing granular media, as such media is also modeled from polyhedral-shaped particles, namely quartz crystallites [HZWH09]. An exact visualization of the particles is required as their shapes define the shape and amount of the surface of the media in cavities and tunnel networks exposed to surrounding media like gas or liquid (e. g. consider thick-film gas sensors [MC95]).

Rendering arbitrary polyhedral shapes is traditionally achieved by means of a polygon mesh. Although this is straightforward, it is not clear whether this is the best approach when visualizing very large data sets. On the one hand, a data set of 1 000 000 particles easily requires up to 200 000 000 triangles to be rendered (Tab. 1). On the other hand, usually there are not a million unique particle meshes, but particles are scaled and rotated instances of only several dozen particle templates. This fact can be exploited to optimize the rendering performance, allowing for interactive visualization of data sets with up to several millions of particles. These data set sizes are currently produced by simulations in the application domain. A similar approach was used by Lampe et al. [LVRH07] to visualize large proteins by rendering instances of amino acids instead of individual atoms.

The main contribution of this work is the presentation and comparison of state-of-the-art rendering techniques that yield polyhedra. The detailed performance analysis identifies the best approach based on data set size and particle complexity.

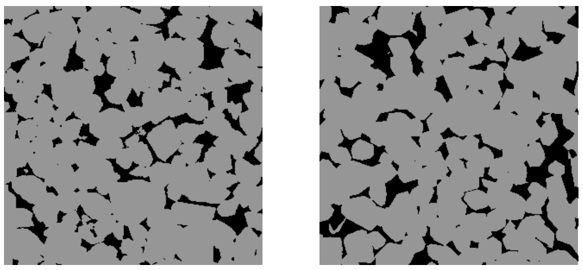

Figure 2: Comparison of 2D sections (2.25 mm $\times$ 2.25 mm) experimental data (μ-CT; **left**) with reconstructed model (**right**) of Fontainebleau sandstone. Grains are shown in gray, while space in-between is shown in black. The reconstructed model is stochastically very similar to the experimental data. The degree of stochastic similarity was measured and documented quantitatively using numerous geometric observables described in detail in [LBFH10].

2. Modeling Porous Media

Porous media may be loosely characterized as materials containing a complex system of internal surfaces and phase boundaries [Hil96]. The random appearance of the phase boundaries has lead to a variety of stochastic models [Hil02, Hil00]. Stochastic reconstruction models for porous media have been investigated extensively in [MH99, MTH00, BH99].

A fundamental drawback of stochastic reconstruction models as well as segmentation of 3D X-ray or synchrotron microtomograms obtained from scattering experiments is the representation of the microstructure on a regular (cubic) lattice [BHK*09]. Such a representation at a single, fixed resolution precludes multiscale modeling of porous media, because only a single scale can be represented with currently available data manipulation capacities. Recently this fundamental limitation of lattice-based models was overcome in stochastic continuum models [BHK*09,BØH*09,LBFH10].

Visualization and 3D-imaging of stochastic continuum models is important for identifying and modeling regions of interest in multiscale porous media, such a microporous regions with sub-micron-sized pores or microcrystalline regions with nanometer crystallites. It has never been carried out for multiscale sandstones due to the lack of suitable models on the pore scale. Here we report a first step in this direction. We present fast visualizations of continuum models for sandstones with polyhedral grains obtained from a novel molecular dynamics method of generating the stochastic point process underlying the stochastic continuum models. While the molecular dynamics method will be described elsewhere, we report here details of the rendering technique.

When visualizing media modeled by such crystallites we can exploit several factors. The rather small number of crystallite template types makes this scenario a perfect candidate

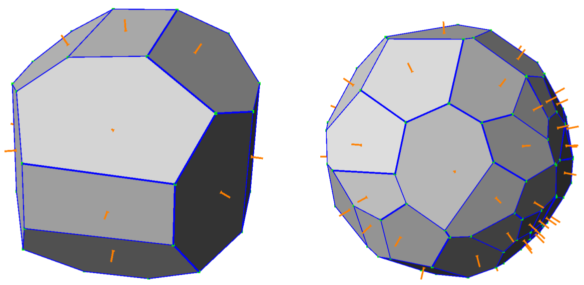

Figure 3: Two crystallites used to model the porous media; **Left:** rather uniform crystallite with 18 faces; **Right:** crystallite with 50 randomly placed faces used for performance measurements only. Orange lines show the face normals defining the tangent planes.

for instancing approaches. The flat faces of the crystallites seem to be well suited for classical mesh-based approaches, as there are no resolution concerns, neither in form of mesh tessellation nor as image-space resolution, e. g. for curved surfaces. The crystallite definition using tangent planes of a single sphere yields always completely convex crystallites as the structure can also be obtained by employing Voronoi Diagrams on spheres [NLC02]. Thus, each face of each particle also is a convex polygon within the tangent plane. This allows for the rather simple generation of a triangle-mesh representation for each particle.

3. Rendering Approaches

Particle-based rendering is originally based on graphical point primitives (i. e. `GL_POINT` in OpenGL). This approach scales very well for very large numbers of particles, but is usually restricted to simple particle types like point sprites. Ray casting with programmable graphics hardware allows for smooth particle shapes, like spheres or cylinders. For increased particle complexity or arrays of mesh instances there are several instancing techniques available. They usually employ vertex-buffer objects (VBOs), hardware-supported instancing, or programmable geometry shaders.

As has been published earlier [GRE09], modern graphics cards can easily process particle-based data sets with up to a million particles without the need of any optimized data structure. Since we are only interested in performance and scalability of the different rendering approaches described in this work, we do not apply any optimized data structure, as such would impair the scaling behavior with respect to data set sizes. For a system capable of visualizing multi-million particle data sets we can apply object-space subdivision and visibility prediction based on occlusion queries as has been previously presented [GRDE10].

All approaches share some common ideas: a particle-space coordinate system is introduced for each particle, similar to object-space, in which the geometry of the used crystallite is placed at the origin with a defined orientation. The transformation from this system to the object-space coordinate system is described by eight scalar values (3 for position, 1 for size, 4 holding an orientation quaternion). When using opaque representations, the order in which the particles are drawn can be optimized to minimize state changes of the rendering engine. This is the case when the particles are sorted based on the crystallite template type they instantiate.

3.1. VBO-based Rendering

The first approach is based on the idea to store all particle template types in graphics memory. For each particle the triangle mesh is computed in particle-space and stored within a VBO. The best-suited VBO access-mode is `GL_STATIC_DRAW` as the particle templates are not altered after creation.

For each particle type, the corresponding VBOs (vertices and normal vectors) are activated. To instantiate the particle the VBOs are drawn once (One call of `glDrawArrays` per particle). The transformation from particle-space into object-space can be implemented either by using the built-in model view matrix, or by using a simple shader program, which results in better performance due to fewer state changes. Because of the high number of OpenGL function calls this rendering approach has the highest CPU load.

3.2. Hardware-supported Instancing

The high number of function calls of the simple VBO-based rendering, which result in high CPU load, can be reduced using hardware-supported instancing. While this instancing rendering approach is very similar to the previous one—both using one VBO per crystallite template type—the placement, orientation, and scaling of the particles has to be transferred differently to the shader program when using instancing, since all particles are drawn with a single function call. The shader program must be able to determine the transformation data, which has to be uploaded to the graphics hardware as well, based on the instancing index.

We store the required transformation data in a single `RGBA` float texture (two texels per particle) for optimized texture upload, since the vertex data upload, which would originally be used for this data, is already used by the particle template VBOs. However, the maximum texture size ($8k \times 8k$) limits the number of particles of one crystallite type to 33.5 millions ($8k \times 8k/2$) to be drawn in a single call. To overcome this limit, we can simply use multiple draw calls.

3.3. Geometry Shader

Programable geometry shaders allow for using the vertex data upload for the particle data again, similar to the clas-

#Faces	#Triangles	#Vertices (unique/drawn)
4	4	4/12
10	~ 26	$\sim 15/\sim 46$
20	~ 68	$\sim 36/\sim 108$
50	~ 200	$\sim 98/\sim 300$

Table 1: Number of triangles and vertices per crystallite for a given number of faces; the numbers vary slightly for the different crystallite templates due to the different plane cutting conditions. The two numbers of vertices show the number of *unique* vertices, required when storing the mesh data in VBOs, and the number of vertices needed to be *drawn* based on the number of triangles (relevant for the geometry shader approach).

sical approach. The idea is to upload point data and perform the crystallite template instancing through the geometry shader. We generate one shader for each crystallite type, which is similar to the idea employed by [LVRH07]. The triangle-meshes of the crystallites are stored not in VBOs but in the corresponding geometry shader's code directly. The calculation of the mesh is done on the CPU. We then generate a geometry shader code which outputs this mesh directly in normalized device coordinates. The output type `GL_TRIANGLE_STRIP` allows to efficiently render each face of a crystallite, similar to the previously described approaches.

Since the output size of each geometry shader is known, load-balancing within the graphics hardware should be possible. However, the maximum number of vertices output from the geometry shader is limited. On current graphics cards, the limit[†] of scalar values output per geometry shader invocation is 1024, resulting in a maximum number of $1024/8 = 128$ vertices (8 components since we need to set `gl_Position` and `gl_FrontColor`). Table 1 shows that crystallites with 20 faces are already very close to that limit (23-24 faces exceed the limit). However, this limit is not crucial, since 18 faces are sufficient to achieve a realistic model, as stated in section 2. Nevertheless, if more complex crystallites are required in future, the geometry shader approach will need to be modified (e. g. splitting up the crystallites into several shaders), which will introduce more state changes and additional data to be uploaded.

3.4. Ray Casting

The original particle-based visualization works with point primitives and GPU-based ray casting of implicit surfaces resulting in perspective-correct object appearance of each

[†] `GL_MAX_GEOMETRY_TOTAL_OUTPUT_COMPONENTS`, see the geometry shader extension specification [Geo]

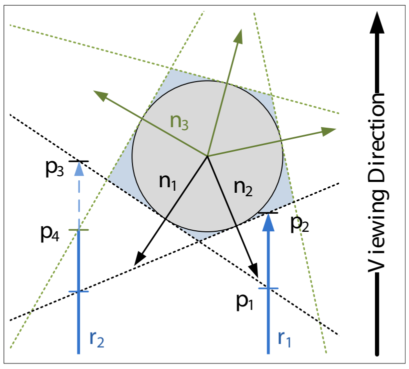

Figure 4: The principle of ray casting convex polyhedra (in 2D convex polygons). Viewing ray $\mathbf{r_1}$ hits tangent plane of normal $\mathbf{n_1}$ at point $\mathbf{p_1}$. Hit point $\mathbf{p_2}$ is the farthest front face hit point and thus the correct point. Viewing ray $\mathbf{r_2}$ would choose $\mathbf{p_3}$ this way. However the front-most back face hit point $\mathbf{p_4}$ with plane of normal $\mathbf{n_3}$ is closer to the viewer. Thus viewing ray $\mathbf{r_2}$ does not hit the polyhedron (polygon) at all.

particle. This approach is especially beneficial for smooth quadratic surfaces, like spheres, due to the always optimal image-space resolution. However the ray casting gets more and more computationally intense the more intersections between the viewing ray and particle geometry have to be calculated.

The central idea of GPU-based glyph ray casting follows the approach of point sprites, for which OpenGL `GL_POINTS` are extended to image-space window-aligned quads. While the classical point-sprite approach places a texture on each quad, e. g. the image of a sphere, GPU-based ray casting performs a single ray casting step, also known as local ray tracing, in the fragment shader for each fragment, calculating the perspective correct rendering of any implicit surface encoded in the ray-surface intersection equation.

For the polyhedral-shaped particles used in this paper, a viewing-ray plane intersection has to be calculated for each tangent-plane of the crystallite. For each intersection, a simple dot product of the viewing ray and the plane normal specifies whether the plane was hit front-side. The correct intersection is the farthest hit of a front face. However, if the nearest hit of a back face is in front of this one, the crystallite is not hit at all (See Fig. 4). Note that no sorting of intersections needs to be performed because simple `min` and `max` operations during the hit tests are sufficient. This approach is basically similar to the work on bounding objects for ray tracing presented by Kay and Kajiya [KK86].

Similar to the idea of the geometry shader-based approach, we can render all particles using a single draw call with one shader for each crystallite template type performing the ray casting and lighting operations.

4. Results

All performance measurements were conducted on a machine with Intel Core I7 980X 3.33 GHz CPU, 12 GB RAM, NVIDIA GeForce GTX 480 graphics card, running Windows 7 (x64). The viewport resolution was 1024×1024. Table 2 shows all rendering performance values.

In principle, all techniques scale linearly with the number of particles. The two exceptions to this are the performance values for 10 000 particles and the values of the ray casting rendering technique. For small particle numbers the overhead of the individual methods (e. g. even back buffer clearing) becomes the limiting factor. The ray casting approach scales better with the number of particles, because it is based on fragment processing—while all other approaches are based on vertex processing—and the number of fragments per particle decreases with increasing number of particles due to the limited screen-space resolution.

For the VBO-based rendering approach there is no significant difference between rendering of crystallites with 4 to 20 faces. We presume that this is due to the high CPU load of this method as discussed in Sec. 3.1. Thus, the difference between the VBO sizes is insignificant. This is not the case when comparing the values of the 20-faced crystallite with the values of the 50-faced crystallite. As the number of faces roughly double the frame-rates are roughly halved, as would have been expected.

The hardware instancing rendering method, which follows an idea similar to the VBO-based rendering, trades the high CPU load for the additional overhead of requiring the upload of the particle data to the graphics memory as textures. Quite surprisingly, despite of the upload this approach is favorable for almost all cases, except for complex particles (20 or 50 faces) in small data sets (10 000 particles). For the small data sets the overhead of the texture upload limits the overall rendering performance of this approach. However, even in these situations the performance is still comparable to the VBO-based method.

The geometry shader is known to perform quite poorly when the number of output primitives varies strongly or when the number of output primitives is much higher compared to the number of input primitives. Since we create one shader for each crystallite template type with a fixed number of output-primitives the first aspect of varying output is not an issue. However, the second aspect of a disadvantageous ratio of input primitives to output primitives can be seen with growing number of faces per crystallite. While the geometry shader clearly outperforms all other rendering techniques for tetrahedral-shaped crystallites ($\times 2$ compared to hardware instancing and up to one order of magnitude compared to VBO-based rendering; top-most curve in Fig. 6), it quickly becomes slower as the number of output triangles increases. For 10-faced crystallites it is roughly at the same level as VBO-based rendering, but it scales better with higher numbers of particles, possibly due the CPU limitation of the VBO-based approach. For 20-faced crystallites and at least 100 000 particles the ratio of input objects to output objects is disadvantageous to such an extent that the geometry shader approach results in the worst performance. The geometry shader programs fail to compile for 50-faced crystallites as discussed in Sec. 3.3. Although it would be possible to create the whole crystallite's geometry with two or three shaders, this would require increased particle upload ($\times 2$ or $\times 3$) and an increased number of state changes due to the additional shaders. Considering the huge performance drop from 10-faced crystallites to 20-faced crystallites and the additional overheads mentioned above, it is unlikely that this approach would yield better performance than the alternatives.

The ray casting approach is the only approach considered in the work which is based on fragment-processing instead of vertex-processing. It therefore adds a dependency to the screen-space sizes of the rendered particles. For large particles this method is much slower than the vertex-based methods, which is shown by the values of the small data set (10 000 particles). However, even for the 100 000 particles data sets, particles become small enough in screen space (still 20×20 pixels) that this method yields performance comparable to the other rendering methods. For data sets of 1 000 000 particles or more, the ray casting approach scales extremely well, as the particles keep getting smaller in screen space, and the method reaches frame rates $\times 2$ to $\times 5$ faster than the alternative methods (see blue lines in Fig. 6).

5. Conclusion and Future Work

In this paper we presented four different techniques for rendering polyhedral-shaped particles which are used to model porous media. The rather straightforward VBO-based rendering has a simple implementation but does not scale well with the number of particles due to the CPU-controlling scheme. Since the modeling of porous media has similar concepts as instancing, hardware-supported instancing yields better performance than pure VBO-based rendering for medium-sized and large data sets. The geometry shader shows the best rendering performance for simple particle sizes, i. e. tetrahedra. For large data sets and complex shaped particles GPU-based ray casting highly benefits from the small particle sizes in image-space and achieves the best performance of all methods. This may not be the case for large display installations.

As a rule of thumb, for data set sizes below 1 000 000 particles the *instancing* approach yields best performance results. Rendering data sets with several millions of particles the *ray casting* approach is fastest on workstation computers. The *geometry shader* works very well with tetrahedral-shaped particles.

An optimized hybrid implementation could be obtained when choosing the different rendering approaches for different situations, even within a single rendering cycle (e. g.

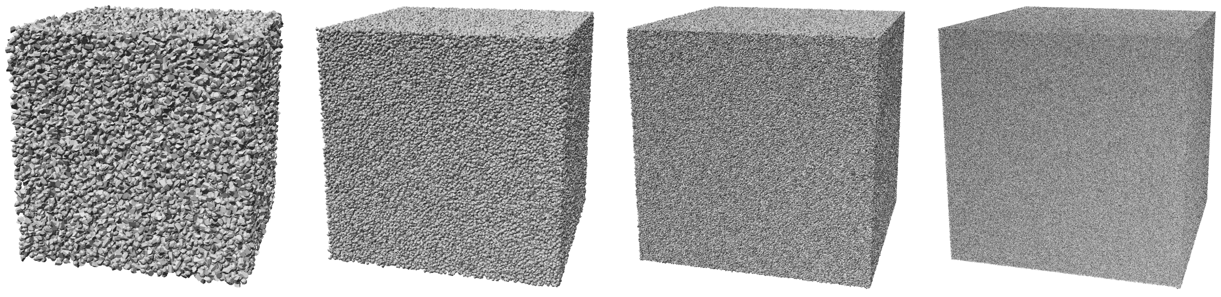

Figure 5: The data sets used for the performance measurements (Tab. 2) with (from **left** to **right**) 100 000, 1 000 000, 10 000 000, and 100 000 000 particles. The used crystallite types have always 20 faces. All rendering methods produce exactly the same images.

		#Particles				
#Faces	Technique	10 000	100 000	1 000 000	10 000 000	100 000 000
4	VBO	644.5	80.0	7.73	0.778	0.061
	Inst	1259.0	295.6	34.3	3.03	0.254
	GPUGeom	1430.1	524.8	69.0	7.51	0.747
	Raycast	181.3	81.7	25.9	4.31	0.566
10	VBO	635.5	80.1	7.7	0.758	0.064
	Inst	962.6	209.4	24.8	1.98	0.16
	GPUGeom	612.8	88.1	9.1	0.911	0.086
	Raycast	160.0	62.9	19.3	3.19	0.39
20	VBO	610.6	76.6	7.64	0.682	0.066
	Inst	592.3	105.8	11.5	1.12	0.094
	GPUGeom	174.1	19.2	1.86	0.183	0.018
	Raycast	133.8	61.0	15.2	2.11	0.25
50	VBO	315.2	38.0	3.87	0.37	0.038
	Inst	306.7	40.9	4.1	0.392	0.039
	GPUGeom	–	–	–	–	–
	Raycast	117.1	45.5	8.85	1.09	0.118

Table 2: The rendering performance values in FPS achieved by the different rendering techniques for different data sets. *#Faces* are the number of crystallite faces (not triangles). Details on the rendering techniques can be found in the corresponding subsections: *VBO* in Sec. 3.1, *Inst* in Sec. 3.2, *GPUGeom* in Sec. 3.3, and *Raycast* in Sec. 3.4. Note that the geometry shader was unable to compile for crystallites with 50 faces (see Sec. 3.3 for discussion).

using ray casting for many small particles in the background, while using instancing for the big particles in the foreground). However, such an implementation would be quite complex. Considering the fact that ray casting is always interactive and for large data sets it is even the fastest method, it is dubitable if this effort is justified.

There are some further improvements to the presented methods and additional methods we would like to investigate as future work. The geometry shader approach could be further optimized by only generating front faces. Similar to the test performed by the ray casting approach the geometry shader could perform a back-face culling which would roughly decrease the number of output primitives by a factor of two. However, assuming there were no overhead, the geometry shader would then e. g. reach the performance values of 10-faced crystallites for 20-faced crystallites, and thus, would still be slower than hardware-supported instancing for crystallites of relevant size. Nevertheless, the geometry shader approach could benefit from future hardware architectures.

Beyond the question of fast rendering the visual evaluation of material properties arises. When rendering additional information, e. g. onto the surface of each crystallite, the evaluation conditions on which rendering method to apply may change. Mesh-based approaches have the advantage of easily usable texture-coordinates, while ray casting is already performed on a per-fragment basis e. g. allowing for on-the-fly evaluation of a 3D scalar field, like a distance field to the particle surfaces. For future work we plan to analyze these aspects, not only for rendering but also for GPU-based

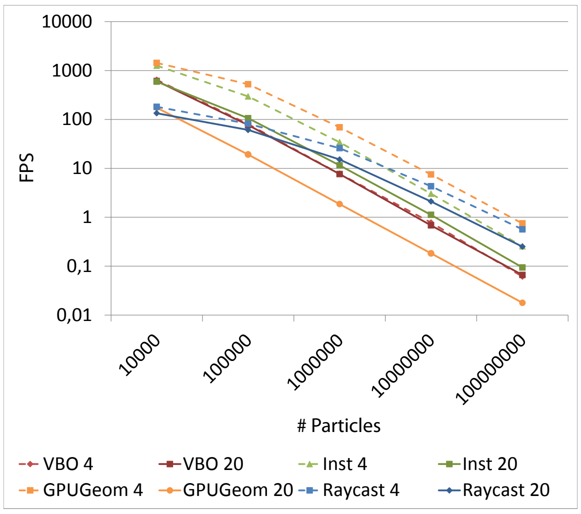

Figure 6: Rendering performance of all methods in FPS for tetrahedral-shaped crystallites (dashed lines) and 20-faced crystallites (solid lines). Geometry shader for 4-faced crystallite is fastest (orange line). Most methods decrease linearly with the number of particles, except for when rendering very few particles. Ray casting results (blue lines) decrease more slowly than linear and thus are beneficial for large data sets.

evaluation, especially when applied not to the particles themselves but to the empty space in between.

Acknowledgements

This work is partially funded by Deutsche Forschungsgemeinschaft (DFG) as part of SFB 716 projects B.3 and D.3.

References

[BH99] Biswal B., Hilfer R.: Microstructure analysis of reconstructed porous media. *Physica A 266* (1999), 307. 2

[BHK*09] Biswal B., Held R., Khanna V., Wang J., Hilfer R.: Towards precise prediction of transport properties from synthetic computer tomography of reconstructed porous media. *Physical Review E 80* (2009), 041301. 2

[BØH*09] Biswal B., Øren P., Held R., Bakke S., Hilfer R.: Modeling of multiscale porous media. *Image Analysis and Stereology 28* (2009), 23–34. 2

[Geo] GLSL geometry shader 4 extension specification. http://developer.download.nvidia.com/opengl/specs/GL_EXT_geometry_shader4.txt. 4

[GRDE10] Grottel S., Reina G., Dachsbacher C., Ertl T.: Coherent Culling and Shading for Large Molecular Dynamics Visualization. *Computer Graphics Forum 29*, 3 (2010), 953–962. http://www.visus.uni-stuttgart.de/megamol. 1, 3

[GRE09] Grottel S., Reina G., Ertl T.: Optimized Data Transfer for Time-dependent, GPU-based Glyphs. In *Proceedings of IEEE Pacific Visualization Symposium 2009* (2009), pp. 65–72. 3

[GRVE07] Grottel S., Reina G., Vrabec J., Ertl T.: Visual Verification and Analysis of Cluster Detection for Molecular Dynamics. vol. 13, pp. 1624–1631. 1

[Gum03] Gumhold S.: Splatting Illuminated Ellipsoids with Depth Correction. In *Workshop on Vision, Modelling, and Visualization VMV'03* (2003), pp. 245–252. 1

[Hil96] Hilfer R.: Transport and relaxation phenomena in porous media. *Adv. Chem. Phys. XCII* (1996), 299. 1, 2

[Hil00] Hilfer R.: Local porosity theory and stochastic reconstruction for porous media. In *Räumliche Statistik und Statistische Physik* (2000), Stoyan D., Mecke K., (Eds.), Lecture Notes in Physics, Vol. 254, Springer, p. 203. 2

[Hil02] Hilfer R.: Review on scale dependent characterization of the microstructure of porous media. *Transport in Porous Media 46* (2002), 373. 2

[HZWH09] Harting J., Zauner T., Weeber R., Hilfer R.: *Numerical Modeling of Fluid Flow in Porous Media and in Driven Colloidal Suspensions*. Springer, 2009, p. 349. 2

[JBPE99] Jensen R. P., Bosscher P. J., Plesha M. E., Edil T. B.: DEM simulation of granular media-structure interface: effects of surface roughness and particle shape. *International Journal for Numerical and Analytical Methods in Geomechanics 23*, 6 (1999), 531–547. 1

[KHK*09] Knoll A., Hijazi Y., Kensler A., Schott M., Hansen C., Hagen H.: Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic. *Computer Graphics Forum 28*, 1 (2009), 26–40. 1

[KK86] Kay T. L., Kajiya J. T.: Ray tracing complex scenes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1986), ACM, pp. 269–278. 4

[LBFH10] Latief F., Biswal B., Fauzi U., Hilfer R.: Continuum reconstruction of the pore scale microstructure for fontainebleau sandstone. *Physica A: Statistical Mechanics and its Applications 389*, 8 (2010), 1607 – 1618. 1, 2

[LVRH07] Lampe O. D., Viola I., Reuter N., Hauser H.: Two-Level Approach to Efficient Visualization of Protein Dynamics. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1616–1623. 2, 4

[MC95] Martinelli G., Carotta M. C.: Thick-film gas sensors. *Sensors and Actuators B: Chemical 23*, 2-3 (1995), 157 – 161. 2

[MH99] Manwart C., Hilfer R.: Reconstruction of random media using Monte Carlo methods. *Physical Review E 59* (1999), 5596. 2

[MTH00] Manwart C., Torquato S., Hilfer R.: Stochastic reconstruction of sandstones. *Phys.Rev.E 62* (2000), 893. 2

[NLC02] Na H.-S., Lee C.-N., Cheong O.: Voronoi diagrams on the sphere. *Computational Geometry 23*, 2 (2002), 183 – 194. 3