Eurographics/ IEEE-VGTC Symposium on Visualization 2010
G. Melançon, T. Munzner, and D. Weiskopf
(Guest Editors)

Volume 29 (2010), Number 3

Coherent Culling and Shading for Large Molecular Dynamics Visualization

Sebastian Grottel[†], Guido Reina[†], Carsten Dachsbacher[†], and Thomas Ertl[†]

Visualization Research Center, University of Stuttgart, Germany (VISUS)

Abstract

Molecular dynamics simulations are a principal tool for studying molecular systems. Such simulations are used to investigate molecular structure, dynamics, and thermodynamical properties, as well as a replacement for, or complement to, costly and dangerous experiments. With the increasing availability of computational power the resulting data sets are becoming increasingly larger, and benchmarks indicate that the interactive visualization on desktop computers poses a challenge when rendering substantially more than millions of glyphs. Trading visual quality for rendering performance is a common approach when interactivity has to be guaranteed. In this paper we address both problems and present a method for high-quality visualization of massive molecular dynamics data sets. We employ several optimization strategies on different levels of granularity, such as data quantization, data caching in video memory, and a two-level occlusion culling strategy: coarse culling via hardware occlusion queries and a vertex-level culling using maximum depth mipmaps. To ensure optimal image quality we employ GPU raycasting and deferred shading with smooth normal vector generation. We demonstrate that our method allows us to interactively render data sets containing tens of millions of high-quality glyphs.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Computer Graphics—Three-Dimensional Graphics and Realism

1. Introduction

Sciences have come to rely more and more on molecular dynamics simulations to complement costly and dangerous real-world experiments. As the cost of computational power decreases, the size of the simulations challenges interactive visualization systems on workstation computers. It is common practice nowadays that glyph-based visualization of molecular dynamics simulation data applies raycasting techniques implemented on programmable graphics hardware yielding high rendering quality without costly tessellation. Older techniques are based on normal and depth maps to render glyphs [BDST04], but yield lower image quality due to visible texture filtering artifacts and do not provide better performance on contemporary GPUs. Independent of the underlying rendering technique, such visualization systems have to deal with several challenges: simulations of interesting scenarios typically require at least hundreds of thousands of particles or molecules, which exceeds the memory of the graphics hardware. This is relevant especially for mid-range hardware, which would be fast enough to render several millions of glyphs, but lacks the required graphics memory to store them. Time-dependent data sets aggravate this problem because the amount of data increases by orders of magnitude, and hence data transfer easily becomes a severe bottleneck.

This bandwidth problem has already been analyzed by us [GRE09]. Our contributions in this context are a frame-to-frame coherent occlusion culling and caching strategy reducing the overall data transfer to the graphics hardware for large data sets. Our culling technique makes intensive use of the occlusion query mechanism of GPUs to determine the visible parts of the data sets. The key is to carefully budget these queries as they introduce latencies. Our method exploits the waiting time and computes a hierarchical depth buffer for fine-granular culling, effectively hiding these latencies. Actually all current GPUs already provide an implementation of the hierarchical depth buffer [GKM93] that

[†] {grottel | reina | dachsbacher | ertl}@visus.uni-stuttgart.de

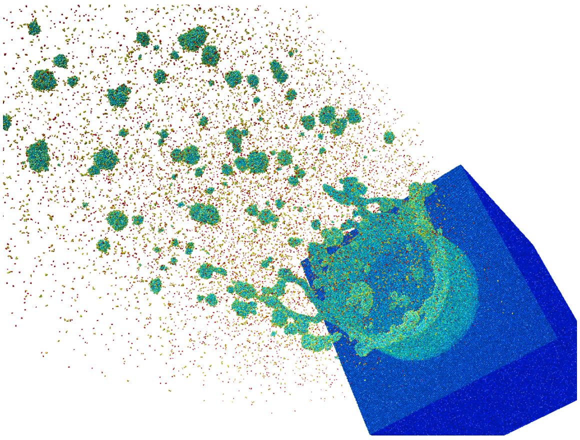


Figure 1: *Data set D4 showing a laser ablation simulation with 48 million atoms. Using our two-level occlusion culling method, a common workstation can interactively render this data set with up to 12 FPS (given the whole data set is visible and no frustum culling is applied).*

culls blocks of pixels prior to rasterization, but unfortunately this mechanism is suspended as soon as any fragment shader outputs a computed depth value, which is necessary for raycasting glyphs. As an alternative, we integrated custom fine-granular occlusion culling into the vertex processing stage using a conservative maximum depth mipmap to cull individual glyphs prior to raycasting.

When displaying large data sets the image-space footprints of individual glyphs are typically very small and raycasting them is prone to aliasing artifacts due to strongly varying surface normals. Furthermore, raycasting small glyphs introduces unnecessary overhead and thus they are often rendered using splats (without normal information). We propose to tackle these problems by introducing a deferred shading pass that estimates normal vectors in image space and significantly improves the perception of meso-level structures.

2. Previous Work

Molecular Dynamics Visualization The visualization of molecular dynamics data has been approached with various substantially different visualization techniques over the years. The most wide-spread tools are Chimera [Chi], PyMOL [Pym], and VMD [VMD]. Generic visualization packages, such as AVS [AVS] or Amira [Ami] also provide special modules for molecular visualization. These tools offer many analysis features for the specific application domains, however, their rendering performance does not allow for interactivity when rendering very large data sets, e.g. consisting of more than 50,000 atoms, since most of them use polygon-based representations without level-of-detail techniques. Recent research addresses these performance problems: Max [Max04] proposed a splat-based approach for large molecules, however, the rendering is more similar to volume splatting than to what researchers in the field of molecular dynamics are used to. A solution based on depth-correct textured billboards for sphere, cylinder, and helix primitives has been presented by Bajaj et al. [BDST04]. These approaches typically achieve lower visual quality as the limited resolution of the normal and depth textures causes artifacts. Halm et al. [HOF05] support texture-based, glyph-based, as well as polygon-based rendering including level-of-detail techniques for whole molecule surfaces. Multi-resolution rendering, see e.g. Nakano et al. [NKV99], allows the interactive visualization of massive data sets with more than a million molecules.

Occlusion Culling The data transfer to the GPU easily becomes the bottleneck when visualizing large molecular dynamics data sets. We [GRE09] analyzed this issue in detail and concluded that the data transfer is one of the most significant bottlenecks. We also found that the performance increase due to quantization beyond single precision is highly situational, but the memory saving inherent to it is beneficial for large data sets and GPUs with limited memory anyway. The determination of visible and occluded parts of 3D scenes in computer graphics, and data sets in visualization, is of central interest for interactive applications and large input data. There is an enormous amount of research that has been conducted in this field. Since covering the entire literature is beyond the scope of this paper we refer the reader to the comprehensive overviews by Bittner and Wonka [BW03], and Cohen-Or et al. [COCSD03].

Our culling method is based on an output-sensitive way to determine the visibility of parts of the data set. The most popular mechanism therefor is the hardware occlusion query supported by current GPUs, which allows the user to determine the number of visible pixels of rasterized geometry against the depth buffer. The query result can be read back to the application after a certain latency which is due to the processing in the graphics pipeline. Further methods for determining visibility are widely used, e.g. the hierarchical depth buffer [GKM93] and variants such as [Déc05]. Functionality akin to the hierarchical depth buffer is implemented in essentially every GPU, however, it is often deactivated, e.g. if fragment shaders output depth values. The item buffer method [KS00], extended by Engelhardt and Dachsbacher [ED09], assigns a unique color to each object, rasterizes the geometry, and then determines its visibility by counting the pixels with the respective color. These methods are very efficient for determining the visibility of many objects at once, however, in our algorithm the associated overhead does not amortize as the number of queries is comparatively small and we explicitly utilize the waiting time for the hardware occlusion query results.

Recent GPUs also provide support for predicated or conditional rendering [Ope08]: similar to the hardware occlusion query, a query is issued by the application, and after-

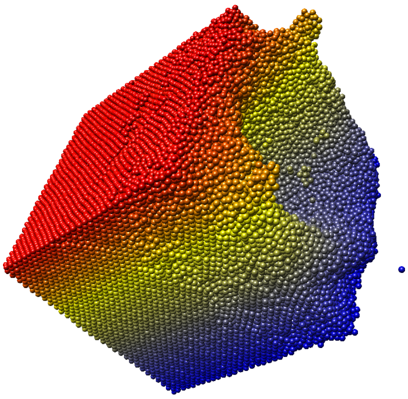


Figure 2: *Data set D1 obtained from a small laser ablation simulation (107,391 atoms). Although the number of atoms is small, a brute-force raycasting of the atoms as spheres results in low performance (27 fps) due to the large number of depth replacements caused by the highly overlapping glyphs.*

wards the GPU is instructed to render the geometry. The primitives are then automatically excluded from rendering if no pixel of the query passed the depth test. Note that this mechanism does not solve the problem of high bandwidth consumption: since the application is not aware of the result, the data has to reside on or be transferred onto the GPU in any case.

Coherent Occlusion Culling Visibility determination in complex scenes typically involves a large number of occlusion queries and latency often becomes an issue. Different strategies to reduce this overhead have been investigated. Guthe et al. [GBK06] model occlusion probability of queries using statistical means and by this aim to reduce the number of unnecessary queries. Bittner et al. [BWPP04] and Mattausch et al. [MBW08] use occlusion queries to perform culling in complex scenes where a bounding volume hierarchy is available. In contrast, we can issue all queries for determining visibility (on the coarse level) at the beginning of each frame before computing the depth mipmap. By this the latencies have practically no impact on the rendering times. Coherent occlusion culling is also important for costly point-based rendering techniques.

Deferred Splatting The deferred splatting method [GBP04] is closely related to our method and exploits temporal coherence in a point selection algorithm for high quality surface splatting. Deferred splatting aims at executing the costly steps of point *splatting* for visible primitives only. For rendering a new frame, the previously visible point primitives are used to create a possibly incomplete depth buffer of the current frame. Next, the points that are newly visible in this frame are determined using a cheap point primitive rendering combined with an item buffer visibility determination. Finally, the depth buffer is completed using this set of primitives. Analogously the primitives constituting the visible surfaces, whose color information is to be splatted into the frame buffer, are determined.

Deferred Shading Molecular dynamics data sets often consist of hundreds of thousands of glyphs and the rendering is prone to aliasing, which typically stems from strongly varying normals. Similar problems are known from point-based rendering and are typically addressed using prefiltering [ZPvBG01] or by accumulating prefiltered input to the lighting computation [BSK04,HSRG07]. We do not prefilter normal information, but instead estimate normals on-the-fly from data in image-space. By this we effectively remove high-frequency noise and generate a coherent impression of the large-scale structure of the data sets.

3. Coherent Two-Level Occlusion Culling

In this section we detail the individual steps of our two-level occlusion culling technique. We use coarse granularity culling to omit large chunks of data from rendering to reduce geometry processing load and bandwidth requirements, followed by fine-granular culling to omit individual glyphs from rendering and save fragment processing power.

We first perform coarse frustum and occlusion culling exploiting temporal coherence. This requires a spatial index structure to partition the input data into cells of glyphs. We use a regular grid (for a discussion see Subsection 3.1) and denote the set of cells containing the glyphs visible in frame t as O_t (for $t = 0$ the set O_t contains all cells, since no visibility determination has taken place before). As in previous work [GRE09] we assume that all glyph data resides in main memory and is uploaded to the GPU for rendering. In contrast to that work, we do not upload all data in a brute-force manner every frame, but reduce the upload to a minimum. After these per-cell operations, we cull individual glyphs in the geometry processing stage prior to rasterization using a conservative estimate and a maximum depth mipmap. By this we reduce the costly computation for raycasting in fragment programs.

Our culling technique builds on the following stages referenced in the following sections and also depicted in Fig. 3:

1. Initialize the depth buffer for occlusion culling rendering the particles stored in cells O_{t-1}.
2. Issue occlusion queries for all cells' bounding boxes of the spatial data structure.
3. Compute a maximum mipmap from the depth buffer of step 1 for fine-granular culling.

4.1. Read back the results of the occlusion queries, update O_t, and render all visible glyphs with per-glyph culling directly on the GPU.

4.2. Estimate normals and perform deferred shading in image-space.

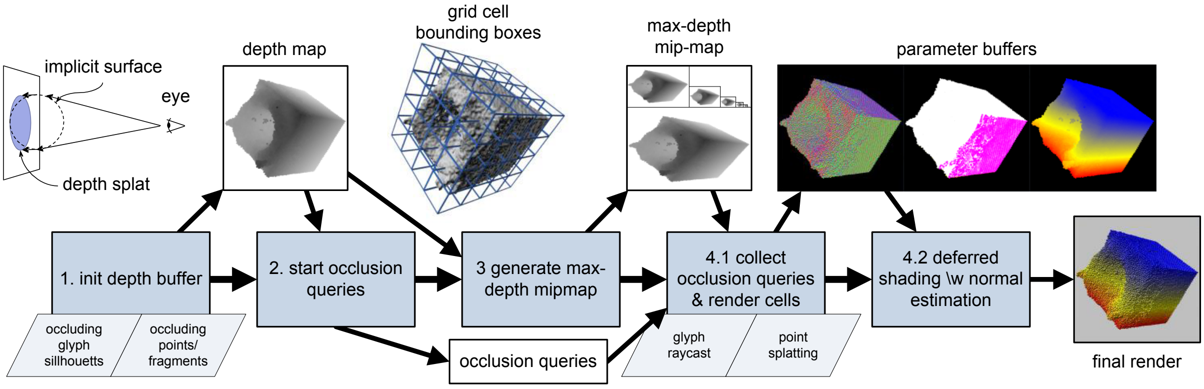

implicit surface
eye
depth splat
depth map
grid cell bounding boxes
max-depth mip-map
parameter buffers
1. init depth buffer
2. start occlusion queries
3 generate max-depth mipmap
4.1 collect occlusion queries & render cells
4.2 deferred shading \w normal estimation
occluding glyph sillhouetts
occluding points/ fragments
occlusion queries
glyph raycast
point splatting
final render

Figure 3: *The stages of our method: 1. initialization of the depth buffer with known occluders in O_{t-1}, 2. start of occlusion queries for all grid cells by testing against the bounding boxes, 3. generation of maximum-depth mipmap, 4.1. collection of occlusion queries, updating of list O_t of visible cells, and rendering of remaining visible glyphs. Stages 1 and 4.1 can output raycast glyphs, or points if the glyphs become to small in image-space, 4.2. deferred shading: image-space calculation of normals and phong lighting. Note that the rendering in stage 1 initializes the depth buffer with a conservative depth splat for the maximum-depth mipmap, as well as for subsequent render passes.*

A pseudo-code description can be found on the web site with supplemental material[†].

We further reduce the overall memory footprint by storing quantized coordinates in the local coordinate systems spanned by each cell of the spatial index structure. In addition to standard short (16-bit integer) representations offered by the GPU, we also analyze the use of geometry shaders with respect to arbitrary quantizations for coordinates and glyph attributes.

3.1. Coarse-Granular Cell-Level Culling

In our implementation we chose a regular grid as spatial data structure which has the same extents as the bounding box of the data set and each cell stores all the glyphs it contains. We opted for a non-hierarchical data structure for various reasons: first, a hierarchical decomposition of the data would make a stop-and-wait algorithm necessary [MBW08], which in turn would require occlusion queries to be interleaved with the rendering. Second, despite the simplicity and hence possibly higher number of occlusion queries with a regular grid, we did not experience detrimental impact on the rendering performance. Self-evidently, we also perform trivial per-cell frustum culling on the CPU, but omit it from the following description to focus on the occlusion culling. Our method exploits frame-to-frame coherency because cells visible in the previous frame (O_{t-1}) have a high probability to be visible in the current frame t as well.

Our methods begins by rendering the particles from the set O_{t-1} for the new frame. Note that grid cells in O_{t-1} are not only potentially visible in frame t, but also contain a significant fraction of occluders for other cells and glyphs. The depth image obtained from this rendering will be used to perform the culling on both granularity levels. In this stage, the raycasting of the glyphs in O_{t-1} is simplified by only outputting the maximum depth of a glyph instead of computing exact depths, but with correct glyph silhouettes. This maximum depth splat is a conservative estimate of the occluding potential of a glyph. It can be computed once per glyph and thus a per-pixel depth computation is not necessary and the early-z culling of the GPU remains active. In order to benefit from early-z culling as much as possible we sort of the cells from front to back. We retain the order of the cells between frames. This reduces the sorting cost since the cells are already almost in order when the camera position does not move abruptly.

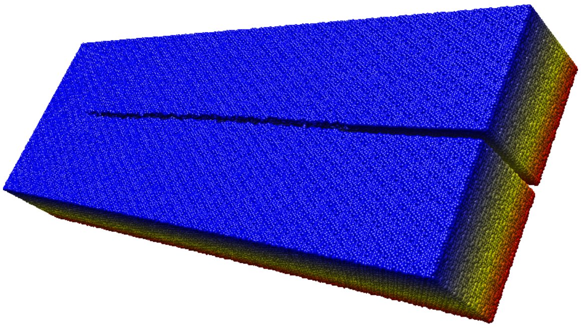


Figure 4: *Data set D2 is a molecular dynamics simulation of a crack propagation in solid media. Due to the crack, inner cells of the grid can become visible depending on the view point.*

[†] Web site with supplemental material: http://www.vis.uni-stuttgart.de/~grottel/eurovis10/

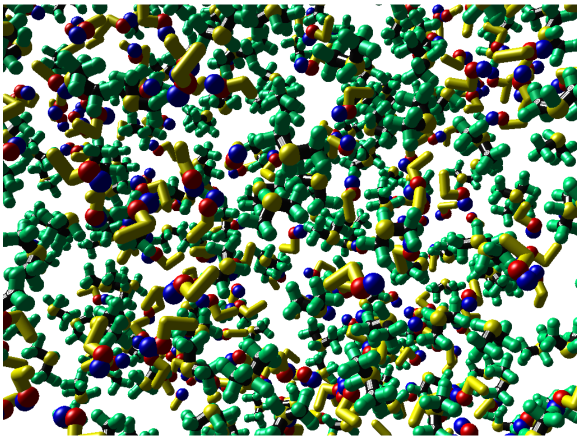


Figure 5: *A sparse data set obtained from a thermodynamics molecular dynamics simulation of a mixture of ethanol (C_2H_6O) and heptafluoropropane (C_3HF_7). In such data sets there is almost no occlusion, hence our approach can only reduce small glyphs to points and provide frustum culling based on the grid structure.*

In stage 2 of our method we issue the occlusion queries against the depth buffer generated in stage 1 for all cells of the grid to obtain O_t. The results of the queries for previously invisible cells provide information about cells that become visible and need to be rendered (this adds new grid cells to O_t). Occlusion queries for cells contained in O_{t-1}, i. e. previously visible cells, determine if a grid cell becomes occluded (and is thus removed from O_t).

The resolution of the spatial grid is a user parameter in our implementation. On the one hand, more cells yield more accurate results for cell-level culling. On the other hand, the more cells we use, the more expensive the cell sorting becomes and the more occlusion queries must be issued introducing longer latencies which have to be bridged. For all data sets we used a 15^3 grid, which yielded satisfactory results for all our test data sets. This indicates – for realistic scenarios with a reasonably large viewport resolution – that stage 3 of our method takes long enough such that the latency of $15^3 = 3375$ occlusion queries can be effectively hidden. For the rather elongated data sets (D2, D3, and D4) there was no difference between using 15^3 cells compared to a partitioning that takes the aspect ratios of the data sets' bounding boxes into account.

In the final stage 4 we read back the results from each cell's occlusion query individually. According to the results we update O_t and render all glyphs (for final display) only if the cell is visible. Since we use the same cell order for starting the occlusion queries as for collecting their results and for rendering the glyphs, the rendering itself adds to the time budget available for the query results to return.

3.2. Fine-Granular Vertex-Level Culling

As already mentioned, GPUs provide an intrinsic fine-granular culling through the hierarchical depth buffer [GKM93], however, this functionality becomes deactivated in the case of per-pixel raycasting of the glyphs. In stage 3 we compute a mipmap resolution pyramid of the depth image from stage 1 to remedy this issue. When down-sampling, instead of averaging the depth values, we compute the maximum of a 2×2 pixel block, for conservative occlusion testing.

There is a one-to-one correspondence of glyphs and vertices in our visualization, as one point primitive is rendered at stage 4.1 for every glyph which is then raycast in the fragment shader. For every glyph we compute its bounding box in image space and determine the mipmap level where the area of a texel is just larger than that of the bounding box size. The glyph's bounding box then covers four texels at most (since the vertices cannot be expected to be aligned with the mipmap) and we take the maximum of their depth values as a conservative estimate. We can then cull the glyph if its minimum depth (that can be easily computed) is larger than this estimate. Since we use a vertex program for these operations, we "cull" by moving the point primitive to infinity. Obviously this does not reduce the geometry processing load, but saves costly raycasting operations.

Our approach is primarily suited for dense data sets, as in the molecular dynamics simulations in the field of material science (Fig. 1 shows an example). Obviously, for sparse data sets (Fig. 5) the room for performance improvement by culling, other than frustum culling, is small, although the large number of particles and the limited display resolution still cause optimizable overdraw.

3.3. Bandwidth Reduction

Determing invisible parts of the data set and omitting them from rendering at an early stage of the pipeline is one option to reduce bandwith requirements. Apart from culling, the two most promising approaches are bandwidth reduction by quantization and upload avoidance by using GPU-side caching.

Caching We implemented such a caching mechanism based on static vertex buffer objects. With the data set sizes our approach aims at, storing the whole data on a consumer graphics card is impossible. Since the upload of a vertex buffer object (VBO) is known to be more costly than direct rendering from vertex arrays [GRE09], we need to ensure that once uploaded VBOs are used multiple times. After the cell-level culling pass we already know which vertices will be used at least twice per frame, as all potentially visible glyphs will be processed to initialize the depth buffer and are rendered as glyphs. Thus the respective data blocks for these cells are obvious candidates for caching, but the set of cells is view-dependent, so a complete pre-caching at

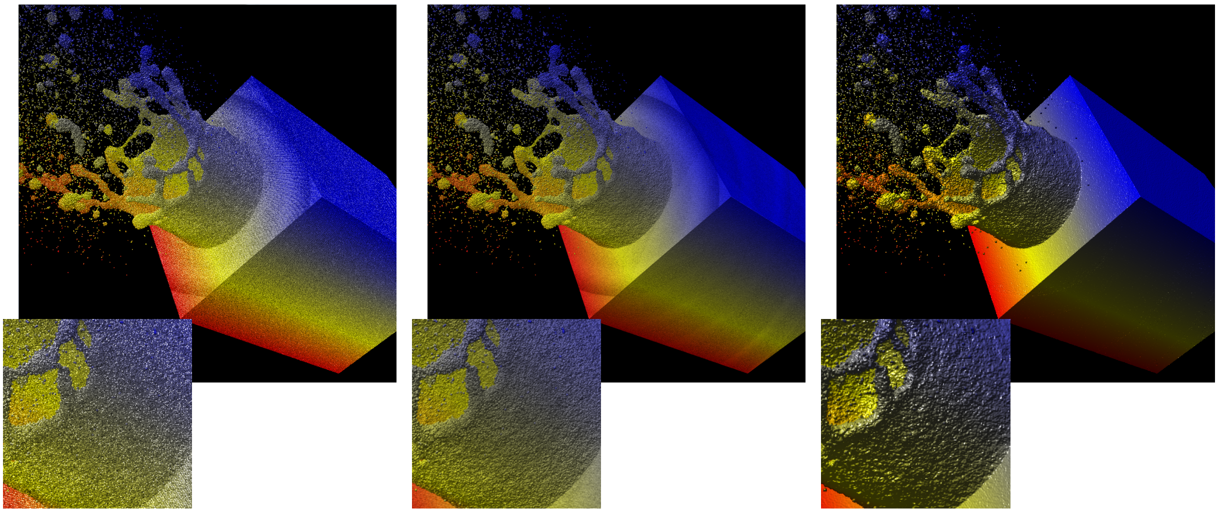


Figure 6: *Comparison between standard and deferred shading (please zoom the electronic version). Left: standard, naïve raycasting of spheres. Middle:* 8×8 *super-sampling of raycast spheres. Right: deferred shading with artificial normals.*

startup is not expedient. To compensate the upload overhead of VBOs, we limit the number of VBO uploads to just a few per frame (currently three has proven to be acceptable). However, this strategy cannot be used when streaming data from time-dependent data sets or performing in-situ visualization. In this case, the data is never reusable and we resort to quantization to compensate the higher upload bandwidth.

Quantization Several quantization strategies for geometry and attributes have been presented to date, e.g. hierarchical positional quantization [HE03], uniformly distributed normals [Paj03], or color, geometry, and normal quantization for point rendering [RL00]. Since most of the data sets do not provide glyph attributes, we focused on the quantization of glyph coordinates. We showed that a quantization to shorts results in the optimal performance-precision tradeoff [GRE09]. In our approach the quantization is carried out after a glyph position has been transformed into the local coordinate frame of a grid cell. For reconstruction of the original coordinate we pass the origin and extent of a grid cell to the GPU prior to rendering the glyphs contained therein.

As an alternative, and as a basis for more flexible quantization, we also implemented a de-quantization using both geometry shaders and instancing with vertex shaders where a shader program extracts two or more glyphs from a single compressed representation. As a test case, we chose a 2×12-bit encoding of two values into the mantissa of one IEEE 754 float to pack the data for two glyphs into one vertex element in the data stream . This data is either unpacked in the geometry shader to generate two primitives, or the instances are assigned the respective portion of the data in the vertex shader. In case of the geometry shader the required calculations after unpacking the data, such as the approximation of the glyph silhouette to estimate the point sizes, are done in parallel for both glyphs. These unpacking shaders were employed in the stages 1 and 4.1 of our approach, for glyphs as well as for points. The performance impact of arbitrary quantization is discussed in Sect. 4.

3.4. Image-Space Normal Estimation

Since our glyphs are rendered using raycasting, the resulting normals are always mathematically accurate. However, if the image-space size of a glyph becomes small, i.e. if it covers few fragments only, undersampling of the glyph surface yields aliasing effects, appearing as noise or flickering (see Fig. 6). Glyphs of the image-space size of a single fragment or below are rendered using a flat color for the same reason. While the aliasing effect of the former can be lessened using expensive super-sampling, the complete lack of normal vectors for the latter splats requires a different approach.

To this end, we store the glyph surface information, i.e. surface normal, color, and position, into an off-screen buffer (similar to standard deferred shading approaches). To extract a smoothed surface normal for a pixel we use the nine coordinates – stored at the pixel and at its eight neighbor pixels – as control points for a quadratic bézier patch. Quadratic patches are suitable to capture the curvature of the implicit surfaces of the glyphs. Background fragments and information from outside the viewport are replaced by replicating the data from the currently considered fragment. In Section 4 we discuss the quality of the different approaches and provide links to a comparison video.

However, since we only want to use the estimated normal for points and small glyphs, we store a fourth parameter for every fragment: a confidence rating based on the image-space size of a glyph that determines how strong the influence of the raycasting normal is. This value ranges from zero for single-fragment-sized glyphs, to 1 for larger glyphs (4 × 4 fragments or larger in our examples), and is directly used for obtaining a smooth blending between the artificial and raycast normal.

4. Results and Discussion

Table 1 shows the data sets used for benchmarking the proposed approaches. D1 (see also Fig. 2) is the smallest among the data sets and shows one time frame of a molecular dynamics simulation of a laser ablation scenario. The data sets D2 (Fig. 4) and D3 have been obtained from molecular dynamics simulations of crack propagation in a solid medium, whereas D2 just comprises exactly ten percent (in width) of the whole data set. D4 (Fig. 1) has also been obtained from a laser ablation simulation, but is considerably more complex than the first example. It is also the largest real-world data set that was available for benchmarking our system. We generated an artificial data set, D5, by randomly distributing 100 million atoms. Significantly larger data sets would require out-of-core rendering on our test platform which we wanted to avoid since this would introduce additional performance issues which are not the focus of the work at hand. Consequently, we loaded the data sets entirely into the main memory of our test machine which is running Windows 7 with an Intel Xeon 5530 2.4 GHz with a GeForce GTX 285 with 1 GB graphics memory. The rendering is performed using OpenGL and a viewport size of 1024^2 pixels.

The rendering performance and culling results of our method are shown in Table 2 and Figure 7. For each data set we measured the sphere-only (rows denoted with S-*) and the point-only (P-*) performance. In real-world scenarios the rendering performance lies in between both values, because the glyphs are rendered as one of these primitives depending on their image-space size. For all test scenarios we measured an increase in rendering performance, the only exception to this are the two D1 P-* series of benchmarks. In this case, single fragment points and small data sets, the performance is reduced due to the overhead introduced by our method. In addition, this data set is not dense enough to generate significant occlusion which is underlined by the percentage of visible cells which is 2 to 5 times higher than it is when rendering spheres (D1 S-*). Obviously, small data sets such as D1 defy all optimizations targeted at rendering large molecular dynamics simulation data. Furthermore, D1 and D2 show unexpectedly low frame rates when rendering sphere glyphs (S-*) which is supposedly due to the large overlap of the primitives resulting in many depth buffer replacements. This, in turn, also provides significant occlusion, causing large performance improvements when culling is activated.

data set	number of glyphs	description
D1	107,391	small laser ablation
D2	4,456,963	small crack propagation
D3	44,569,630	large crack propagation
D4	48,000,000	large laser ablation
D5	100,000,000	synthetic test data set

Table 1: *Sizes and descriptions of the example data sets: D1 - D4 have been created by molecular dynamics simulations, D5 is an artificial data set and has been created using a statistical distribution.*

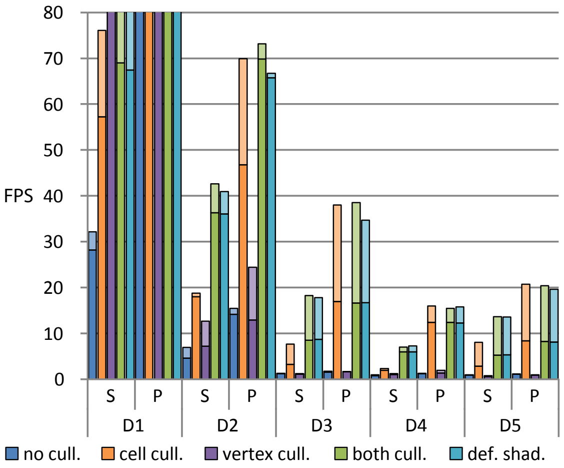


Figure 7: *The rendering performance results from Table 2 without any culling (no cull.), cell-level culling only (cell cull.), vertex-level culling only (vertex cull.), both culling techniques together (both cull.), and both culling techniques and the deferred shading pass (def. shad.). Both culling techniques together result in the best performance, while for pure point representations the share of the vertex-level culling is negligible. This is due to a change of the limiting bottleneck from rendering to data transfer. The overhead of the deferred shading pass is very small. Note that the bars of data set D1 are truncated (see Table 2 for the values) to keep the focus on the values of the larger and more interesting data sets.*

Two-level culling We also measured the performance of the cell-level and vertex-level culling separately. In general, the cell-level occlusion culling provides higher performance improvements which is due to the reduction of uploaded data. The vertex-level culling can only reduce the workload of the fragment shaders. For small data sets, such as D1, the data transfer is not the bottleneck and reducing the work load of the fragment processing through vertex-level culling is highly beneficial. For large data sets (D3-D5) the data transfer is the bottleneck and thus the cell-level culling has a significant impact on the performance, while the effect of vertex-level culling alone is negligible. The possible granularity of the cell-level culling is however limited since we want the number of grid cells to be low, because of the front-to-back sorting on the CPU and the number of occlusion queries. On the other hand, the vertex-level culling is

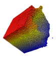
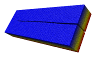
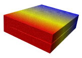
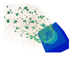
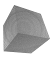
data set	view con-figuration	culling					quantization (with two-level culling)				deferred shading		visible data	
		none	cell	vertex	both	both	shorts	shorts	GS	inst.	shorts	floats	cells (%)	# glyphs
caching		no	yes	yes	yes	no	no	yes	no	no	yes	yes		
D1 	S-Best	28.21	76.06	172.67	90.23	89.61	80.58	91.62	86.62	84.79	84.39	82.30	8.09	3600
	S-Worst	32.12	57.25	137.37	68.98	69.88	63.65	68.93	69.67	70.03	66.95	67.45	23.38	3136
	P-Best	621.20	99.42	195.34	97.22	104.70	81.02	70.31	75.14	74.62	69.68	104.02	39.38	27713
	P-Worst	593.23	99.27	200.96	98.33	101.75	88.05	72.75	74.20	73.76	72.35	97.39	45.75	46338
D2 	S-Best	4.60	17.96	7.22	42.60	39.62	37.94	40.52	37.64	39.54	40.53	40.90	6.67	12594
	S-Worst	6.92	18.77	12.64	36.30	33.42	30.97	35.87	28.58	29.47	35.47	36.04	20.53	115003
	P-Best	14.17	69.90	24.44	69.81	42.84	48.04	72.83	45.24	32.53	67.28	65.75	6.67	44804
	P-Worst	15.46	46.78	12.91	73.15	65.27	54.74	68.21	65.56	59.24	66.45	66.70	22.93	261221
D3 	S-Best	1.29	7.62	1.22	18.27	15.23	14.51	17.89	15.64	13.65	17.99	17.78	6.67	153786
	S-Worst	1.18	3.21	1.06	8.48	6.19	5.81	8.20	6.53	5.62	8.36	8.67	18.70	623188
	P-Best	1.70	16.91	1.62	16.61	9.71	9.37	16.79	11.65	7.31	16.79	16.66	6.70	460425
	P-Worst	1.56	38.02	1.65	38.52	16.53	15.99	37.15	32.54	17.29	35.70	34.69	18.70	989268
D4 	S-Best	0.77	1.94	1.23	5.96	4.28	3.80	5.85	4.92	3.77	5.89	5.94	6.67	160066
	S-Worst	0.95	2.33	1.03	7.02	4.44	4.41	6.92	5.72	4.61	7.14	7.23	59.35	654718
	P-Best	1.26	12.41	1.93	12.36	6.56	6.77	12.15	10.03	5.19	12.26	12.29	6.67	214358
	P-Worst	1.19	15.98	1.36	15.43	9.32	9.05	15.77	11.85	5.89	16.61	15.76	58.90	1144420
D5 	S-Best	0.88	8.04	0.73	13.65	9.38	9.18	13.45	10.00	7.92	13.43	13.59	6.67	740977
	S-Worst	0.92	2.85	0.55	5.22	3.32	3.42	5.19	3.91	2.80	5.24	5.29	18.70	1313734
	P-Best	1.08	20.71	0.90	20.40	11.70	10.93	20.60	16.92	9.04	19.89	19.63	6.67	799347
	P-Worst	1.10	8.34	0.88	8.26	4.25	3.46	8.27	6.30	3.31	8.19	8.13	18.70	1521383

Table 2: *Performance measurements of our method. All numbers denote frames per second if not otherwise noted. We selected different views such that glyphs are large enough in screen space and rendered as raycast spheres (S-*), or simple points (P-*). We also chose viewing directions and distances such that we obtain a best (*-Best) and worst (*-Worst) case for the culling algorithms (i. e. maximum/minimum number of grid cells occluded). The last two columns show statistics for the case that both culling levels are active: the percentage of the grid cells that are visible under the respective view configuration (column 14), and the number of glyphs that are actually raycast after the vertex culling stage (column 15). Column 3 (culling: none) shows the baseline performance for the unoptimized approach. Columns 6 and 7 demonstrate the impact of caching. The performance impact of deferred shading and normal estimation can be seen comparing columns 7 and 13 as well as 9 and 12, respectively.*

well-suited to further reduce the number of actually rendered glyphs after coarse culling, especially for massive data sets, which consequently still contain a high number of densely packed glyphs in a single cell. The S-* series of measurements for D3 and larger data sets shows that the combination of both methods is up to one order of magnitude faster (compared to no culling) in these cases.

Bandwidth reduction Reducing the data upload through GPU-side caching or quantization further improves the rendering performance. When employing our caching strategy a speedup of about 40-50% can be observed on average (see columns 6 and 7 in Table 2). Obviously, caching is most effective when the view parameters change smoothly, as the GPU-side cache is updated incrementally. Moreover, the impact of the caching is prominent when the workload of the fragment processing stage is not the bottleneck of the entire pipeline. In our tests quantization in general does not show any positive effect since our cell-level culling and caching methods reduce the data transfer load beyond relevance for optimization. As presented in previous work, quantization to short has only minor precision issues [GRE09] and did not introduce any visual artifacts in our tests.

If no caching is used, the geometry shader approach improves performance compared to unquantized data, but the instancing mechanism in OpenGL failed to convince, probably because of the too small number of instances (only two). When used together with caching, the rendering speed is bounded by the rasterization or raycasting stage. However, for very large data sets a combination of caching and quantization is still useful as quantized data consumes less memory and thus larger fractions of the data set can reside in local GPU memory making the system less sensitive to interactive changes of the viewing direction. In addition, quantization provides speed-up for sparse data sets where the cell-level culling is not effective. Our results indicate that our implementation can use both techniques simultaneously without any negative mutual implication.

Image-space normals Typically, deferred shading is used to increase the rendering performance by moving the rather costly lighting calculations from the raycasting stage, which is subject to heavy overdraw, into a single image-space pass. Since our culling strategies almost completely resolve the overdraw problem, this optimization is actually no longer achievable, and in principle, the introduction of the additional image-space rendering pass results in a slight performance drop. However, our intention in using deferred shading was not to increase rendering performance, but image quality. When raycasting small spheres their normals and thus the results of per-pixel lighting calculation are subject to aliasing artifacts (see Section 3.4 and Figure. 6). This problem even retains when using super-sampling. We generated this image using a non-interactive rendering at 8 × 8 higher resolution, i.e. 8192^2, followed by a down-sampling to obtain the resulting image. Although the aliasing noise, previously appearing as bright speckle highlights all over the image, is largely removed, there still remain artifacts (i. e. dark concentric circles on the surface of the block in Fig. 6, center). Obivously, the rendering performance drops significantly due to the increased image resolution. We measured a drop to 16% for 16× super-sampling, and a drop to 4.5% for the previously mentioned 64× super-sampling for the data sets D2 and D4. In contrast, our deferred shading stage has only negligible impact on the rendering performance (Table 2, columns 12 and 13) and yields consistent, smooth results. This is possible as we use normals for the lighting computation obtained from raycasting whenever a glyph is large enough, and use normals generated from the fragment positions stored in the deferred shading buffers otherwise when glyphs are replaced by point primitives. A smooth transition between both is ensured by blending the generated normals and the computed normals for glyphs within a certain size interval. This results in a consistent lighting for the whole dataset. The comparison between the naïve raycasting, super-sampling, and our image space method can be seen in Figure 6 and a supplementary video[‡].

Comparison to visualization tools These results demonstrate that our approach achieves interactive frame rates even for data sets containing several tens of millions of glyphs. We tried to compare our approach with freely available visualization tools. Sadly, none of them was able to load any of the data sets D3-D5. TexMol [BDST04] failed to display D2 after successfully loading it but achieved a very good frame rate of about 110 fps for D1. BallView [MHLK05] was able to load D1 and render it at less than 1 fps. AtomEye [Li03] performs quite well for D1, but even for D2 the performance drops to about 0.5 fps. Even when using GLSL for raycasting the glyphs VMD [VMD] renders D1 at 4 fps and requires more than 20 seconds for a single frame for D2. This is despite the VMD GLSL shader being almost identical to ours except for several branches, which should be remedied by on-the-fly shader generation, but to our knowledge this does not explain the huge difference in performance.

5. Conclusion and Future Work

Our two-level occlusion culling approach allows the interactive visualization of molecular dynamics data sets of the order of 10^8 glyphs on commodity workstations. The biggest share of the performance increase is due to the data transfer reduction through coarse occlusion culling on the grid cell level. Besides the lower bandwidth consumption, this also removes workload mainly from the geometry processing. The vertex-level culling operates on a finer granularity and considerably reduces the number of raycasting operations, and thus fragment processing load. Obviously, occlusion culling does not improve the rendering performance for sparse data sets. Large molecular dynamics data sets are often dense and especially the combination of both culling levels is beneficial for such cases. Additionally even sparse data sets with several million particles are subject to overdraw, since normal workstation displays do not offer enough pixels. Caching and quantization yield almost similar rendering performance when used alone, but allow for more data to be cached in GPU memory when used together. However, caching is only suitable for static data sets, whereas quantization can be used with dynamic data. Our deferred shading technique ensures improved shading quality for small glyphs and points with only negligible impact on the rendering performance.

Anticipating that the data set sizes increase even further in the future, the capacities of a single workstation are exceeded and parallel rendering is the only option to provide interactive visualization. Our occlusion culling method to reduce the overdraw is naturally suited for image-space subdivision. However, object-space subdivision might be required to handle the large amount of data. As future work we therefore want to explore hybrid approaches evaluating the visibility queries in parallel in object-space, while finally rendering the image – after our method has removed occluded parts of the data – in a distributed manner by an image-space subdivision. With larger data sets there will be another, possibly even more severe, bottleneck than the rendering itself: the data transfer from secondary storage. An object-space method would allow for a distributed loading, or out-of-core rendering, of the data sets to remedy this issue.

Acknowledgment

This work is partially funded by Deutsche Forschungsgemeinschaft (DFG) as part of Collaborative Research Centre SFB 716.

References

[Ami] Amira. http://www.amiravis.com/. 2

[AVS] AVS. http://www.avs.com. 2

[BDST04] BAJAJ C., DJEU P., SIDDAVANAHALLI V., THANE A.: Texmol: Interactive visual exploration of large flexible multi-component molecular complexes. In *Proceedings of the conference on Visualization '04* (2004), pp. 243–250. 1, 2, 9

[BSK04] BOTSCH M., SPERNAT M., KOBBELT L.: Phong splatting. In *Proceedings of Symposium on Point-Based Graphics 2004* (2004), pp. 25–32. 3

[BW03] BITTNER J., WONKA P.: Visibility in Computer Graphics. *Environment and Planning B: Planning and Design 30*, 5 (2003), 729–756. 2

[BWPP04] BITTNER J., WIMMER M., PIRINGER H., PURGATHOFER W.: Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum 23*, 3 (2004), 615–624. 3

[‡] A video comparing the image quality of super-sampling vs. image-space normal estimation (Xvid-compressed AVI; 1280 × 720, or scaled-down 640 × 360) can be found on the web site with supplemental material: http://www.vis.uni-stuttgart.de/~grottel/eurovis10/eurovis10-grottel.xvid.avi
http://www.vis.uni-stuttgart.de/~grottel/eurovis10/eurovis10-grottel.xvid_sml.avi

[Chi] UCSF Chimera. http://www.cgl.ucsf.edu/chimera/. 2

[COCSD03] Cohen-Or D., Chrysanthou Y. L., Silva C. T., Durand F.: A Survey of Visibility for Walkthrough Applications. *IEEE Transactions on Visualization and Computer Graphics 09*, 3 (2003), 412–431. 2

[Déc05] Décoret X.: N-Buffers for Efficient Depth Map Query. *Computer Graphics Forum 24*, 3 (2005). 2

[ED09] Engelhardt T., Dachsbacher C.: Granular Visibility Queries. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2009), pp. 161–167. 2

[GBK06] Guthe M., Balázs A., Klein R.: Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries. In *Eurographics Symposium on Rendering 2006* (June 2006), Akenine-Möller T., Heidrich W., (Eds.). 3

[GBP04] Guennebaud G., Barthe L., Paulin M.: Deferred Splatting . *Computer Graphics Forum 23*, 3 (septembre 2004), 653–660. (EG2004 Proceedings). 3

[GKM93] Greene N., Kass M., Miller G.: Hierarchical Z-Buffer Visibility. In *SIGGRAPH '93* (1993), pp. 231–238. 1, 2, 5

[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. 1, 2, 3, 5, 6, 8

[HE03] Hopf M., Ertl T.: Hierarchical Splatting of Scattered Data. In *Proceedings of IEEE Visualization '03* (2003), IEEE. 6

[HOF05] Halm A., Offen L., Fellner D.: BioBrowser: A Framework for Fast Protein Visualization. In *Proceedings of EUROGRAPHICS - IEEE VGTC Symposium on Visualization 2005* (2005). 2

[HSRG07] Han C., Sun B., Ramamoorthi R., Grinspun E.: Frequency domain normal map filtering. *ACM Transactions on Graphics 26*, 3 (2007), 28. 3

[KS00] Klosowski J. T., Silva C. T.: The Prioritized-Layered Projection Algorithm for Visible Set Estimation. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (2000), 108–123. 2

[Li03] Li J.: Atomeye: an efficient atomistic configuration viewer. *Modelling and Simulation in Materials Science and Engineering 11*, 2 (2003), 173–177. 9

[Max04] Max N.: Hierarchical molecular modelling with ellipsoids. *Journal of Molecular Graphics and Modelling 23* (2004). 2

[MBW08] Mattausch O., Bittner J., Wimmer M.: CHC++: Coherent Hierarchical Culling Revisited. *Computer Graphics Forum (Proceedings Eurographics 2008) 27*, 2 (Apr. 2008), 221–230. 3, 4

[MHLK05] Moll A., Hildebrandt A., Lenhof H.-P., Kohlbacher O.: Ballview: An object-oriented molecular visualization and modeling framework. *Journal of Computer-Aided Molecular Design 19*, 11 (2005), 791–800. 9

[NKV99] Nakano A., Kalia R. K., Vashishta P.: Scalable Molecular-Dynamics, Visualization, and Data-Management Algorithms for Materials Simulations. *Computing in Science and Engineering 1*, 5 (1999), 39–47. 2

[Ope08] OpenGL Extension Registry: NV_conditional_render. http://www.opengl.org/registry/, 2008. 2

[Paj03] Pajarola R.: Efficient level-of-details for point based rendering. In *Proceedings IASTED International Conference on Computer Graphics and Imaging (CGIM)* (2003). 6

[Pym] PyMOL. http://pymol.sourceforge.net/. 2

[RL00] Rusinkiewicz S., Levoy M.: QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 343–352. 6

[VMD] Visual Molecular Dynamics. http://www.ks.uiuc.edu/Research/vmd/. 2, 9

[ZPvBG01] Zwicker M., Pfister H., van Baar J., Gross M.: Surface splatting. In *Proceedings of SIGGRAPH 2001* (2001), pp. 371–378. 3