Volume xx (200y), Number z, pp. 1–6

Hardware-Accelerated Glyphs for Mono- and Dipoles in Molecular Dynamics Visualization

G. Reina and T.Ertl

{reina, ertl}@vis.uni-stuttgart.de
Visualization and Interactive Systems Institute, University of Stuttgart

Abstract

We present a novel visualization method for mono- and dipolar molecular simulations from thermodynamics that takes advantage of modern graphics hardware to interactively render specifically tailored glyphs. Our approach allows domain experts to visualize the results of molecular dynamics simulations with a higher number of particles than before and furthermore offers much better visual quality. We achieve this by transferring only visualization parameters to the GPU and by generating implicit surfaces directly in the fragment program. As a result, we can render up to 500.000 glyphs with about 10 fps displaying all the simulation results as geometrical properties that resemble the classical abstract representation used in this research area. Thus we enable researchers to visually assess the results of simulations of greater scale than before. We believe that the proposed method can be generalized to create other kinds of parametrized surfaces directly on graphics hardware to overcome the bandwidth bottleneck that exists between CPU and GPU.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism,I.3.8 [Computer Graphics]: Applications

1. Introduction

Nowadays, simulations on a molecular level are gaining more and more importance in research areas like chemistry and thermodynamics, materials and several other areas where nanoscalar particles are of importance. Such simulations are very expensive in terms of computational power, but with commodity-off-the-shelf clusters becoming more common and filling this gap at relatively low cost, the simulations are becoming larger in terms of particle numbers as well as simulated time. The data generated by such simulations is only limited by the available processing power, even including hundreds of thousands of molecules and thus posing a challenge as well for those striving to visualize the results.

Simulations bridge the gap between theory and experimental practice, making it possible to verify the theoretical models on the one hand and on the other hand replacing experiments under difficult conditions, like extremely small scales [AT87] or metastable state. Simulations on a molecular level have been successfully employed since the 1950s, however, especially on the nanoscalar level, many effects are not yet completely understood. Since scientists need to continuously compare the results from experiments and simulations, the need to visualize these results arises.

The conventional approach would be to generate polygon models for each of the molecular constituents, composed of spheres and other geometrical primitives. However, this has several drawbacks. The tessellation must be adapted depending on the rendered size to obtain smooth surfaces and not degrade performance below 5-10 frames per second. This can be achieved taking advantage of level-of-detail methods [FS93],[Lak04], to adapt the number of polygons as to guarantee smooth surfaces for near objects and coarsely render distant ones to save performance. However a large number of polygons has to be sent to the graphics card in any case. Furthermore the orientation has to be converted into matrices for each molecule or be kept in memory, which needs four times as much space. To overcome these problems, we propose to make use of several different techniques to allow for high-quality visualization of molecules in a way that is tailored for experts from the problem domain. In our approach we take advantage of modern graphics hardware to ray trace an implicit surface representation of a customized glyph for

each dipole in the fragment units. The data to be sent to the graphics card is reduced to a single point and some adjustable glyph dimensions per dipole. This allows us both to save bandwidth between CPU and GPU and enhance the visual quality at the same time, since every pixel is ray traced distinctly. We demonstrate the performance of this method with a few example data sets from the molecular dynamics domain. This basic idea of generating geometry on the graphics card can be applied to a wider range of applications, because the problem of having to transmit large amounts of polygons to the graphics hardware is generic and the system bus cannot be expected to be accelerated at a pace as quick as applications demand it. Consumer graphics cards, however, are always being improved in order to deliver an increasingly realistic experience to computer gamers. The concept of mapping a small number of parameters to a 3D surface on graphics hardware can be exploited to overcome these bandwidth limitations. This method can be used to replace tessellated primitives as well as other kinds of glyphs, like Haber glyphs [Hab90].

The rest of this document is structured as follows: in section 2 we summarize related work; in section 3 we detail our own method. The results are shown in section 4 and the respective performance measurements can be found in section 5; section 6 concludes this work.

2. Related Work

In recent years, there has been diverse work in the area of visualization of molecular dynamics simulations. There have been several approaches which visualize simulation results in a VR environment and optionally allow simulation steering if the number of simulated particles is not very high [AF98], [SGS01]. However also massive data sets of more than a million molecules have been visualized by using multiresolution rendering to ensure interactivity [NKV99]. Our work is directly based on the highly optimized visualization algorithm for volumetric point clouds presented by [HE03] (referred to as *pointcloud renderer* from now on). This code base allows us to start from an approach that can render extremely high numbers of points and to adaptively coarsen the visualization in areas of low interest should the hardware not be powerful enough to maintain interactive frame rates. Besides the possibility of actually visualizing large data sets, a feature we also require for our approach is the optimized memory footprint as well as the possibility to visualize time-dependent data. Points can thus be visualized even in very high numbers, however, they are not adequate primitives for visualizing dipoles in molecular dynamics since the internal configuration, which also defines the spatial extents of such an element, is lost. There also is the need to represent the orientation of the molecules. The algorithms presented in [KE04], [Gum03] were a starting point for us, since in both cases ellipsoids are generated in graphics hardware thus providing a primitive that conveys a detectable orientation. An ellipsoid still has the drawback of having undistinguishable ends, and we need to customize the visual representation of the molecules further to map the relevant parameters onto the final representation in order to suit the experts' needs. Different methods to avoid sending complex geometry to the GPU have been proposed in [LH04] and [MP03], where, in contrast to our approach, the concept is to render into a so-called *vertex buffer* and thus explicitly create the geometry, but on the graphics card itself. Another method for accelerating the rendering of complex geometry is to render it once and use the result as a texture for a bounding billboard (*imposter*) [Sch95], updating it only when the viewing angle change exceeds a preset threshold. A conventional visualization currently used for thermodynamics simulations can be seen in figure 1. In our discussions with users from the problem domain we found that even this kind of dipole representation is suboptimal. They really want a visual representation resembling figure 2 and they would also appreciate higher visual quality. The two spheres of the glyph should represent the molecular radii, while the cylinder, as a symbolic representation of a bar magnet, should depict polarity and charge with the conventional red/green coloring as known even from school experiments.

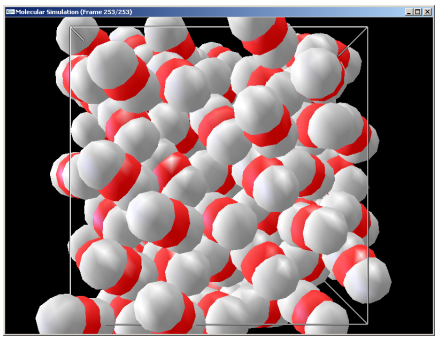

Figure 1: *Conventional visualization of molecular dynamics simulation with polygonal primitives and simple glyphs lacking polarity*

3. Our Algorithm

The data that is available to us from simulations of molecular dynamics consists of discrete time steps, each of which contains a list of the available molecules along with their spatial position, orientation and type. To alleviate the bandwidth problems that come with a polygon-based rendering method, we decided to extend the pointcloud renderer available in our department, since it offers high performance and already creates a hierarchical data structure to give the option to fit quantized relative positions into one byte instead of four-byte-long floats. The program is also able to adaptively render a scene to guarantee visual quality in a region of interest as well as sustaining interactive frame rates by

replacing elements outside the ROI with summarizing primitives. Support for time-dependent data is also already available [HLE04].

Having circumvented the bandwidth problem, we want to render glyphs with only points as input. The general idea behind our approach is to send 'fat points', with all the attributes that result from the molecular simulation, to the graphics card. We use the single point per glyph as a bounding representative - or *point sprite*. However we cannot simply use textures to obtain the correct representation, since each molecule type has different attributes which show up in the proportions of the glyph. This could be resolved by generating textures for each molecule type, but we also have to consider spatial orientation, which is different for each dipole and for each time step. That is why we dynamically generate an implicit surface representation for each glyph and ray trace this surface with per-pixel precision on the point sprite using the fragment units of the graphics card. The parametrization of a molecule type can be resolved into

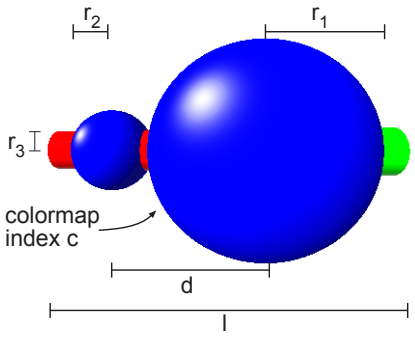

Figure 2: *Dipole glyph and the adjustable visualization parameters*

a tuple containing the spatial configuration of a molecule. For dipoles the tuple can be broken down to the distance between the two molecules d, the radii of the two molecules r_1, r_2, and a color map index c for visual identification. Furthermore there is the need to customize the radius of the magnet r_3 and the length of it to fit the dipole and proportionally show the respective charge (see figure 2).

If we assume that the glyph is always at the origin with the cylinder lying along the x axis, the implicit surface we can generate from these parameters is defined as

$$\left(\begin{pmatrix} x-\frac{d}{2} \\ y \\ z \end{pmatrix}^2 - r_1^2\right) \cdot \left(\begin{pmatrix} x+\frac{d}{2} \\ y \\ z \end{pmatrix}^2 - r_2^2\right) \cdot \left(\begin{pmatrix} 0 \\ y \\ z \end{pmatrix}^2 - r_3^2\right) = 0 \tag{1}$$

Additionally, the infinite cylinder must be capped:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot \vec{x} = r_1 + \frac{l}{2}, \quad \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} \cdot \vec{x} = -r_2 - \frac{l}{2} \tag{2}$$

Intersecting these surfaces with a generic ray cast from the eye position

$$\vec{r} = \lambda \cdot \vec{s} + \overrightarrow{pos_{eye}} \tag{3}$$

yields 3 quadratic equations. Solving those equations leads to the problem that this glyph is already computationally expensive as such, even if we do not consider the orientation at all. To keep the calculations as simple as possible, we emulate the local rotation of the glyph by orbiting the eye position around it. To minimize the data that has to be sent to the GPU, we decided to calculate the matrix for the orientation of a molecule on the GPU from a quaternion.

3.1. The Vertex Program

The data we send to the graphics card consists of the relative vertex positions, as contained in the hierarchical data structure, the four components of the orientation quaternion $q_{1..4}$ and d, r_1, r_2, r_3, c, l, which are passed as texture coordinates. We can also completely avoid to transmit color information if we look it up from a color table by using c (see figure 2). What the vertex program actually does is the calculation of the absolute coordinates from the relative ones. Then the camera orbit quaternion and the local orientation quaternion are multiplied together and transformed into a rotation matrix (which is also passed to the fragment shader, see below). This operation makes most sense in the vertex program since the result is constant for each glyph and thus for each vertex. Subsequently the camera is rotated around the molecule. The point size is adjusted to make sure that the glyph fits inside.

3.2. The Fragment Program

The main part of the calculations for generating the glyph surface is performed in the fragment program. First we have to find the vector which connects the eye to the current pixel starting from the x and y component of the fragment's *window position WPOS*. To that end, we convert x and y to the *View Coordinate System* (top t, bottom b, left l and right r are the parameters of the viewport, w and h its sizes in pixels and z_N is the position of the near clipping plane)

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{VCS} = \begin{pmatrix} \frac{x_{WPOS}}{w} \cdot (r-l) \cdot z_N + l \cdot z_N \\ \frac{y_{WPOS}}{h} \cdot (t-b) \cdot z_N + b \cdot z_N \\ -z_N \end{pmatrix} \tag{4}$$

and cast a ray through it. Since the eye position orbits the glyph, we have to apply the rotation passed from the vertex program to this ray as to keep the center of the glyph in the center of our view. The next step consists of solving the three quadratic equations given by intersecting the ray with the two spheres and the cylinder. We first calculate the expression under the three square roots for an 'early' discarding of the fragment if all three of them are negative (and we hit nothing). We also keep the combined sphere hit result for later use. After calculating the roots, illegal results are discarded by assigning a ray parameter λ which positions

the fragment behind the far clipping plane. The cylinder is a bit tricky to calculate. We need the near and far intersections with the cylinder (CN,CF). Then we must calculate the planes forming the caps cutting off the cylinder according to the length l and decide which intersections is on the near plane (PN) and which is on the far plane (PF). Now we can distinguish the five different cases describing which part of the geometry we are going to hit (see figure 3). The

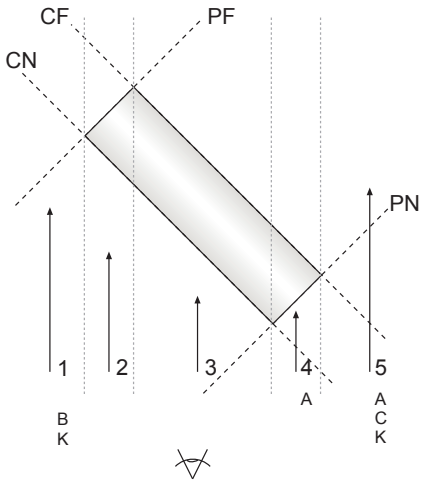

Figure 3: *The different cases when raycasting the cylinder alongside indications where the distinguishing conditions are true*

conditions of the five cases are listed along with the action we need to perform:

$$\begin{array}{ll} 1: \lambda_{CF} > \lambda_{CN} > \lambda_{PF} > \lambda_{PN} & Sphere/Kill \\ 2: \lambda_{CF} > \lambda_{PF} > \lambda_{CN} > \lambda_{PN} & Cylinder \\ 3: \lambda_{PF} > \lambda_{CF} > \lambda_{CN} > \lambda_{PN} & Cylinder \\ 4: \lambda_{PF} > \lambda_{CF} > \lambda_{PN} > \lambda_{CN} & Cap \\ 5: \lambda_{PF} > \lambda_{PN} > \lambda_{CF} > \lambda_{CN} & Sphere/Kill \end{array} \quad (5)$$

If we search for distinguishing conditions, we find that three relations are sufficient for deciding the rendering result:

$$A: \lambda_{PN} > \lambda_{CN}\,, \quad B: \lambda_{CN} > \lambda_{PF}\,, \quad C: \lambda_{PN} > \lambda_{CF} \quad (6)$$

The condition $K = B$ or C discards all cylinder information by again setting an impossibly high ray parameter λ, so that either the spheres are rendered (because the corresponding λ is smaller) or the fragment is discarded if there was no sphere hit in the first place. Based on the results of these calculations we can decide which of the available normals is the correct one:

$$\overrightarrow{N_{cap}} = A \cdot \overline{C} \cdot \overrightarrow{N_{cap}}\,, \quad \overrightarrow{N_{cyl}} = \overline{A} \cdot \overline{B} \cdot \overrightarrow{N_{cyl}} \quad (7)$$

The discarding condition K conveniently also ensures $\overline{B}$ and $\overline{C}$, so the only remaining condition is A (the cap is hit) and $\overline{A}$ (not the cap, but the cylinder is hit). We then apply Phong shading to the surface. The color for the poles of the cylinder is decided as a by-product using the sign of the cylinder intersection x coordinate.

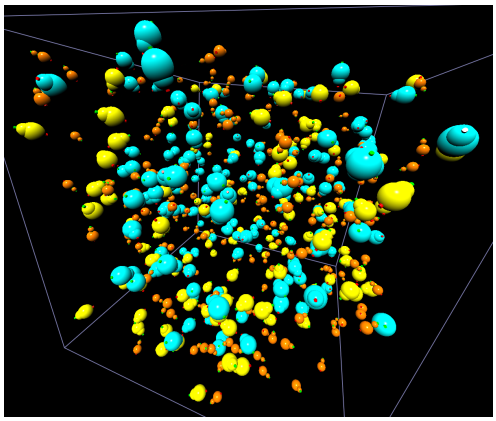

Figure 4: *A mixture with three different dipoles, colored by dipole type*

4. Results

Figure 4 shows a mixture of three different molecule types in a fluid. Two of them are dipoles, which can clearly be recognized by the magnet, and the last one is a monopole. The type is also color-coded. We can see that orientation and polarity is easily discernible in this data set. An example for a simulation in the gas state can be seen in figure 5. The goal of the simulation was to understand the clustering of particles, which can be interpreted as germ formation. The germs can be quite easily made out because of the spatial closeness. The data set is relatively small (1372 monopoles) because before now, the domain experts only used conventional visualization methods, which lacked the performance to display large systems. The supersaturation in this example is higher than under realistic conditions to ensure that the probability of clusters forming is high enough to give results in the 2500 simulated time steps (which correspond to few femtoseconds). Future simulations will more closely resemble reality by having a far lower saturation but much higher scale, thus resulting in a much higher number of particles and more load for the GPU (but our approach can also handle a lot more glyphs as can be seen in table 1). The overall distribution would also be thinner than in this example, making clusters even better to spot. The 'massive' data set is the setup time step of a very large simulation of which unfortunately no further time steps are available to us.

We also used one very recent data set that comes closer to the aforementioned future conditions (100,000 more thinly spread molecules) to compare the performance of the legacy tool and our method. The legacy tool needed about 3 seconds per frame, while our method reached well beyond 25

fps even on a plain GeForce 6800. Using only 10,000 molecules, the results were about 3 fps versus about 100 fps.
The domain experts were quite pleased with the visual results as far as the performance is concerned, since the new method allows them to study data sets of several hundred thousand particles, which was not possible for them before. Up to now the researchers had to put up with bad visual quality and still had to wait several seconds for an interaction to become effective. The increase in quality was also a very important factor to them, while the glyph itself simply was as specified.
While experimenting with some of the bigger data sets, we noticed that the perception of the visualization depth was lacking. The addition of depth cues could help to better grasp the spatial distribution of the molecules [WE02]. We implemented a very simple scheme which attenuates the lighting with the distance from the viewer. The result of rendering a synthetic data set with 200,000 molecules can be seen in figure 6.
It should also be mentioned that our approach can be used with stereo projection and/or tiled displays to allow immersive interaction and better interpret spatial information, since the pointcloud renderer can be synchronized among many machines to generate such output.

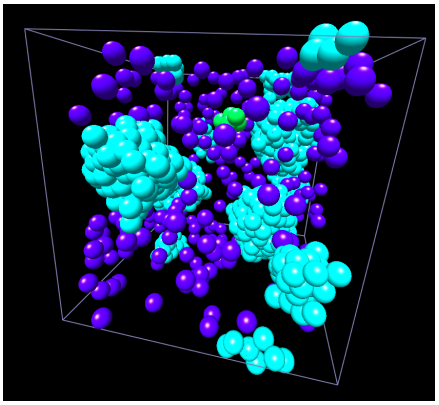

Figure 5: *Supersaturated gas; detected germs(clusters) are cyan*

5. Performance

Our method makes it possible to visualize massive amounts of our glyphs on current hardware. The measurements were taken on 3Ghz-class Pentium 4 machines with at least 2GB of RAM using Nvidia GPUs and ForceWare 61.77 drivers. Our algorithm is implemented as *ARB_vertex_program* and *ARB_fragment_program*, but since ATI does not support the *window position* fragment attribute, which we need for casting the eye ray, it will not run on current ATI hardware. There are different aspects we need to consider when evaluating the performance of our approach. Table 1 shows the

	particles	FX5700	Quadro3K	6800Ultra
mixture	500	30.0	60.1	490.0
gas	1372	10.5	18.4	187.0
fluid	27984	3.4	6.3	52.5
massive	500,000	0.6	1.2	10.0

Table 1: *Performance in fps on different Nvidia graphics cards. All measurements are performed with adaptive rendering switched off as to obtain maximum quality and put maximum load on the GPU.*

performance not decreasing linearly with the number of glyphs contained in the simulation. This is caused by the fact that the limiting factor for our approach is the processing power of the fragment units on the GPU. Since the screen space for each glyph decreases as the number of molecules in the simulation increase, one factor compensates for the other, so we are not slowed down that much by a higher number of particles. When we zoom in, the major part of the molecules is off-screen, which in turn compensates for the fact that the visible ones take up more screen space. When we compare these results with those obtained on the previous GPU generation, we can see that the approach of sending less data over the AGP bus has its strong points. With each new GPU generation we gain performance as more fragment units are added, the clock speeds increase and so on. It is true that AGP is being replaced by PCIe today and high performance graphics cards are becoming available. This transition will double the theoretical bandwidth from CPU to GPU, but as can be seen in table 1, the speedup from one generation of GPUs to the next is far beyond double and from our experience these updates happen more frequently.

6. Conclusions and Future Work

We have shown that it is feasible to generate a dedicated glyph for dipoles in graphics hardware in order to improve the visual quality compared to polygonal primitives. This approach can be generalized for generating suitable surfaces, like geometric primitives or different types of glyphs, in hardware instead of having to tessellate them and sending the resulting polygons over the AGP or PCIe bus. We have also shown that performance and quality by far exceed those of the straightforward approach with polygons. Using LOD techniques this difference can be reduced, however without resorting to fragment programs as well the same shading quality cannot be obtained. Another benefit is the fact that our technique significantly gains performance when using newer GPUs, which justifies the decision to move the load from CPU and bus to the GPU. We were also able to take advantage of the highly optimized pointcloud framework, which in turn brings up a problem to be solved for the future: we do not know yet which kind of primitive is optimally suited to act as placeholder for a cluster of dipoles when we

either have to simplify the scene to improve performance or to implement a focus + context or region of interest visualization to help the user navigate extremely large simulation results. It also remains to be seen if we can further improve performance using an approach similar to[Gum03], namely sending quads instead of points to better approximate the silhouette of our elongated glyphs, thus saving the fragment units as much work as possible. We are also planning to fuse this approach with our efforts to improve interaction with pointclouds in a VR environment, allowing the users to take advantage of depth perception for better understanding their simulation results.

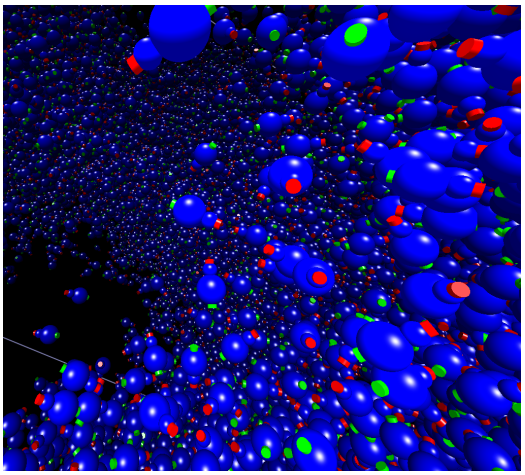

Figure 6: *Synthetic data set consisting of 200,000 dipoles, visualized with depth cues turned on*

7. Acknowledgements

We want to thank Jadran Vrabec and Bernhard Eckl from the Institute of Thermodynamics and Thermal Process Engineering Stuttgart for their valuable input and help with interpreting the results of their simulations. We also want to thank Matthias Hopf for the pointclouds source code and help with finding our way through the code.

References

[AF98] AI Z., FRÖHLICH T.: Molecular dynamics simulation in virtual environments. *Computer Graphics Forum 17* (1998), 267–273. 2

[AT87] ALLEN M. P., TILDESLEY D. J.: *Computer Simulation of Liquids*. Oxford University Press, 1987. 1

[FS93] FUNKHOUSER T. A., SÉQUIN C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics 27*, Annual Conference Series (1993), 247–254. 1

[Gum03] GUMHOLD S.: Splatting illuminated ellipsoids with depth correction. In *VMV* (2003), pp. 245–252. 2, 6

[Hab90] HABER R. B.: Visualization techniques for engineering mechanics. *Comput. Syst. Educ. 1*, 1 (1990), 37–50. 2

[HE03] HOPF M., ERTL T.: Hierarchical Splatting of Scattered Data. In *Proceedings of IEEE Visualization '03* (2003), IEEE. 2

[HLE04] HOPF M., LUTTENBERGER M., ERTL T.: Hierarchical Splatting of Scattered 4D Data. *IEEE Computer Graphics and Applications 24*, 4 (2004), 64–72. 3

[KE04] KLEIN T., ERTL T.: Illustrating Magnetic Field Lines using a Discrete Particle Model. In *Workshop on Vision, Modelling, and Visualization VMV '04* (2004). 2

[Lak04] LAKHIA A.: Efficient interactive rendering of detailed models with hierarchical levels of detail. In *2nd International Symposium on 3D Data Processing, Visualization, and Transmission* (2004), pp. 275–282. 1

[LH04] LACZ P., HART J. C.: Procedural Geometric Synthesis on the GPU. In *Proceedings of the GP^2 Workshop* (2004). 2

[MP03] MĚCH R., PRUSINKIEWICZ P.: Generating subdivision curves with L-systems on a GPU. In *GRAPH '03: Proceedings of the SIGGRAPH 2003 conference on Sketches & applications* (2003), ACM Press. 2

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

[Sch95] SCHAUFLER G.: Dynamically Generated Imposters. In *MVD '95 Workshop "Modeling Virtual Worlds – Distributed Graphics"* (1995), pp. 129–136. 2

[SGS01] STONE J. E., GULLINGSRUD J., SCHULTEN K.: A system for interactive molecular dynamics simulation. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), ACM Press, pp. 191–194. 2

[WE02] WEISKOPF D., ERTL T.: *A Depth-Cueing Scheme Based on Linear Transformations in Tristimulus Space*. Tech. Rep. TR-2002/08, Universität Stuttgart, Fakultät Informatik, September 2002. 5