EUROGRAPHICS 2019 / P. Alliez and F. Pellacini
(Guest Editors)

Volume 38 (2019), Number 2

Dynamic Visibility-Driven Molecular Surfaces

Stefan Bruckner

University of Bergen, Norway

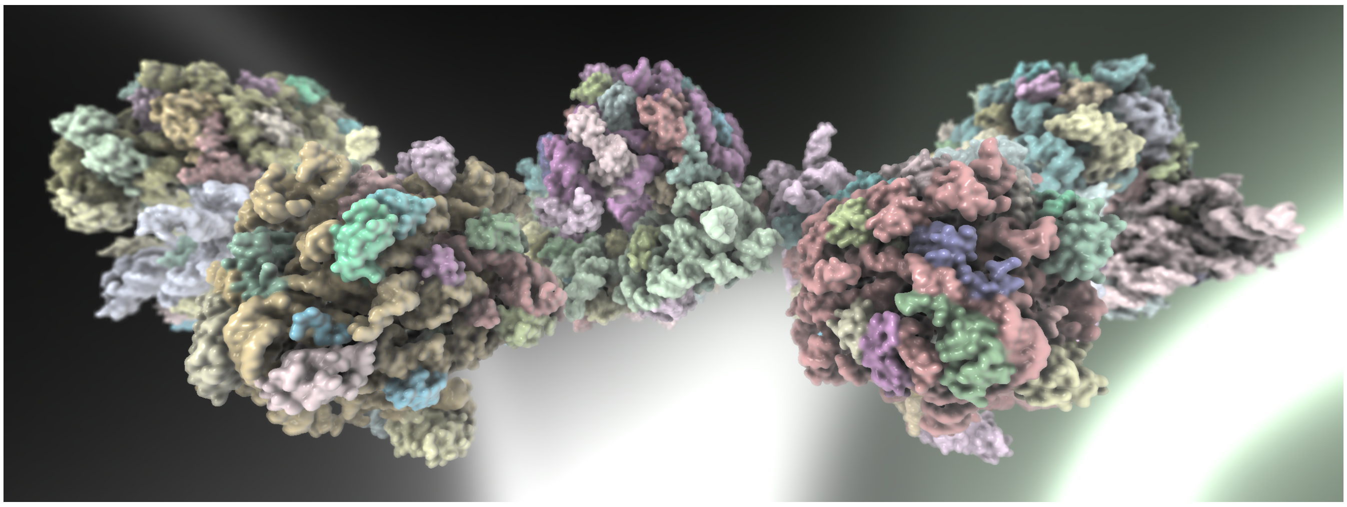

Figure 1: *Molecular surface visualization of 717K atoms rendered at real-time frame rates (PDB ID: 4V4G). Our method is able to render this and even larger structures fully interactively without any preprocessing.*

Abstract

Molecular surface representations are an important tool for the visual analysis of molecular structure and function. In this paper, we present a novel method for the visualization of dynamic molecular surfaces based on the Gaussian model. In contrast to previous approaches, our technique does not rely on the construction of intermediate representations such as grids or triangulated surfaces. Instead, it operates entirely in image space, which enables us to exploit visibility information to efficiently skip unnecessary computations. With this visibility-driven approach, we can visualize dynamic high-quality surfaces for molecules consisting of millions of atoms. Our approach requires no preprocessing, allows for the interactive adjustment of all properties and parameters, and is significantly faster than previous approaches, while providing superior quality.

CCS Concepts

• ***Human-centered computing*** → ***Scientific visualization;*** • ***Computing methodologies*** → *Visibility; Point-based models;*

1. Introduction

Particle-based simulations are a widely used tool in many fields. Molecular dynamics (MD) simulations, for instance, are employed to simulate biomolecules such as proteins. They are commonly used for the investigation of protein-ligand or protein-protein interactions, with important applications in fields such as drug design.

An MD simulation can, for example, calculate the behavior of a molecule and a ligand, and output a set of consecutive spatial atom configurations known as trajectories. These simulations can generate very large time-dependent data sets composed of hundreds of thousands of time steps and potentially millions of atoms per frame. MD simulations are now at a point where the achieved scales begin to be compatible with biological processes [HGO*15]. Visualization is a fundamental tool for interpreting these results, but it is essential that methods are able to handle the ever-increasing size of the data. A common way to visualize such dynamic data sets is by depicting their van der Waals surface, i.e., the union of spheres defined by the van der Waals radii of the respective atoms. The main advantage of this approach is that each atom can be treated independently – the union operation is performed using the graphics hardware's z-buffer, enabling fast high-quality rendering on current GPUs. While this simple approach is attractive due to its efficiency, more advanced surface representations offer several advantages in conveying relevant molecular properties and structure. Such surface representations can be particularly helpful in the identification of cavities or channels accessible by a solvent or other specific molecules, as well as the analysis of binding sites or hydrophobic and hydrophilic regions, which may impact the protein characteristics.

However, these types of representations are more difficult to handle, as the surface definition is typically based on the neighborhood of an atom. In contrast to the van der Waals representation, where each atom simply corresponds to a single sphere, the presence and shape of a surface between two atoms is based on their distance. Common approaches for visualizing such surfaces typically construct an intermediate data structure in order to accelerate rendering, which is challenging when dealing with time-dependent data, as this costly processing step needs to be executed for every frame. For large molecules, this is often prohibitive in terms of computational load and memory consumption.

In this paper, we present a novel approach for the visualization of molecular surfaces based on the Gaussian model. In contrast to previous methods, which typically rely on an intermediate volumetric grid which is then either triangulated or rendered using ray casting, our approach aims to defer surface construction to the rendering stage, where visibility information can be exploited. As molecular data is typically dense, only a fraction of the actual surface will be visible. This enables us to avoid a majority of the computational load of object space methods, enables the handling of dynamic data without any overhead, and lets us change the surface parameters on-the-fly without additional costs. To the best of our knowledge, our method is the first to enable the interactive rendering of dynamic molecular surfaces for structures consisting of millions of atoms, without any prior data reduction.

2. Related Work

Our research aims to enable the real-time visualization of high-quality molecular surfaces for dynamic data. Kozlíková et al. [KKF*17] provide a comprehensive overview of approaches for the visualization of molecular data. One of the most common molecular surface representations is the Solvent Excluded Surface (SES) [Ric77], which is the surface traced out by a spherical probe rolling over all atom spheres. The first practical approach for computing the SES was presented by Connolly [Con83], who provided an analytical description based on three types of patches as well as an algorithm for their extraction.

Due to the costly nature of this initial algorithm, a significant amount of research has been devoted to finding alternative solutions. Totrov and Abagyan [TA96] presented the contour-buildup method, which extracts the patches of the SES from the Solvent Accessible Surface (SAS). Varshney et al. [VBWW97] developed a parallel approach based on power diagrams. The reduced surface, proposed by Sanner et al. [SOS96], is an intermediate representation to accelerate the computation of the analytical SES. Later developments made increasing use of multi-core CPUs and GPUs to further improve the performance of SES generation. Lindow et al. [LBPH10] presented a parallel version of the contour-buildup method for multi-core CPUs, and Krone et al. [KGE11] described a GPU adaptation of the algorithm, achieving interactive computation times for smaller molecules of up to 10K atoms. These approaches use GPU-based ray casting for rendering the individual patches [KBE09]. In an alternative approach, Parulek and Viola [PV12] proposed an implicit formulation of the SES which can be rendered directly, but its performance is limited by the costly evaluation. Parulek and Brambilla [PB13] subsequently presented a faster implicit approximation of the SES.

Due to the limited scalability of analytical approaches, several approximate methods which rely on an intermediate grid representation have been presented. These methods are generally faster, but require more memory. Examples include the method of Can et al. [CCW06] which employs a level-set-based approach for SES computation, and recent work by Hermosilla et al. [HKG*17] who use an incremental scheme to mask the latency until the final surface has been computed. As an alternative to the SES, Gaussian surfaces offer many practical advantages, while still conveying important molecular properties [BC10]. Originally introduced as a general modeling approach in computer graphics termed Metaballs by Blinn [Bli82], the Gaussian model has also been used for visualizing particle-based data outside the context of molecular visualization. Müller et al. [MGE07] presented a multi-pass image-space approach, but did not achieve interactive performance beyond a few thousand particles. The method by Kanamori et al. [KSN08] used a GPU-based approach based on a Bezier approximation of the Gaussian density surface, achieving interactive frame rates for up to several thousand spheres with a five-fold speedup compared to a CPU implementation. Their approach has conceptual similarities to our method, but suffers from the use of depth peeling which requires the rendering of a large number of spheres, which they partially address by using occlusion queries and CPU-based intersection testing based on image-space tiles. Fraedrich et al. [FAW10] sampled particles from Smoothed Particle Hydrodynamics (SPH) data on a perspective grid which was then rendered using ray casting. Knoll et al. [KWN*14] presented a CPU-based framework for the rendering of large particle data sets. Their approach focuses on direct volume rendering with radial basis function kernels and uses a bounding volume hierarchy constructed in a preprocessing step. The approach by Hochstetter et al. [HOK16] uses an adaptive sampling scheme to perform volume rendering of dynamic particle sets driven by a user-controlled screen space error tolerance. Dos Santos Brito et al. [dSBVeSTT18] presented a ray tracing method for particle data that achieves several frames per second for millions of particles. Zirr and Dachsbacher [ZD18] proposed an on-the-fly voxelization method for particle data as well as an accelerated ray casting approach for rendering that achieves a considerable speedup compared to the method by Fraedrich et al. [FAW10]. While these approaches focus on more general particle data, several techniques specifically targeted at molecular visualization have been presented. The method by Dias et al. [DG11] uses a CUDA-based approach for the generation of a density grid and subsequent surface extraction using Marching Cubes [LC87]. QuickSurf was presented by Krone et al. [KSES12] and is currently known to be one of the fastest approaches for molecular surface extraction [KKF*17]. The method is conceptually similar to the work of Dias et al. [DG11], but due to their use of a gathering instead of a scattering approach for grid generation, it is able to achieve over a ten-fold speedup, which enables its application to dynamic data. However, the technique is still heavily constrained by the resolution of the intermediate volume. Coarse grid resolutions can result in a significant loss of detail for larger molecules, while for higher resolutions, both, the cost of the volume generation and the resulting vast number of triangles, prevent interactive performance.

The approach presented in this paper aims to address these drawbacks of object space methods and enable the interactive visualization of Gaussian surfaces for large dynamic molecular data. A concept closely related to our method is the notion of an output-sensitive algorithm, i.e., an algorithm whose running time is mainly determined by the size of its output – in our context the complexity of the output on the image plane, as opposed to the number of atoms in the molecule. Falk and Weiskopf [FW08], for instance, developed a largely output-sensitive technique for 3D line integral convolution, and Šoltészová et al. [ŠBS*17] presented an output-sensitive approach for the filtering of streaming volume data.

3. Visibility-Driven Molecular Surface Rendering

A Gaussian molecular surface is an implicit surface based on a density function $\rho(\mathbf{x})$ defined as

$$\rho(\mathbf{x}) = \sum_i e^{\frac{-s\|\mathbf{x}-\mathbf{c_i}\|^2}{r_i^2}} \tag{1}$$

where $\mathbf{c_i}$ are the atom positions, r_i are their van der Waals radii, and s is a user-defined scaling factor > 0 that controls the appearance of the surface. The surface is the locus of all points $\mathbf{x}$ such that $\rho(\mathbf{x}) - t = 0$, where t is a suitably-chosen threshold. By decreasing the parameter s, the individual Gaussian curves become wider, causing nearby atoms to smoothly merge into a single structure. In the context of molecular visualization, this emulates the behavior of the SES which closes gaps between atoms that cannot by entered by the spherical probe, and it has been shown that an appropriate parametrization leads to a good approximation of the SES [LCL15].

The typical approach for the visualization of such a surface is to first sample $\rho(\mathbf{x})$ into a volumetric grid, which can subsequently be rendered directly using isosurface ray casting or triangulated to create a surface mesh for a specific threshold. In both cases, there are several disadvantages. First, the choice of the grid resolution represents a difficult trade-off between performance and surface quality. This is especially problematic since a significant proportion of the computational resources is potentially spent on regions of the volume that may not be visible from the current viewpoint. In particular when we are considering time-dependent data, the grid needs to be recomputed for every new timestep. Methods based on volume ray casting, which typically rely on spatial acceleration data structures to skip empty space, need to regenerate these data structures for every frame or forego such optimizations. Triangulation-based approaches, on the other hand, also need to recompute the surface mesh which may contain a large number of triangles if a sufficient surface quality is desired.

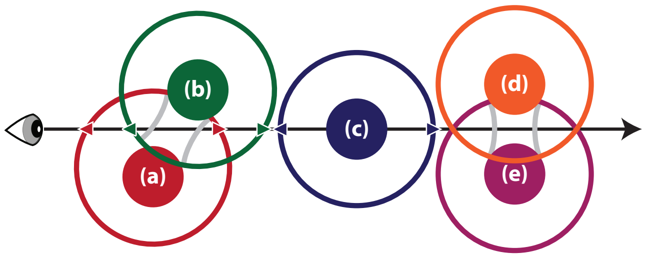

Figure 2: *Illustration of our intersection list generation pass. The Gaussian surface can only occur where two or more spheres of influence (indicated by the outer circles) overlap, as is the case for the red (a) and green (b) spheres. Overlaps located behind a van der Waals sphere (indicated by the filled inner circles) do not need to be considered, so the orange (d) and purple (e) spheres are not stored in the list for the depicted ray, as they are occluded by the blue sphere (c).*

In order to avoid these issues, our approach aims to evaluate the density function directly, without an intermediate discretization. Instead, we construct a compact on-the-fly data structure for every rendered frame that allows for fast visibility-based traversal. This view-dependent list-based data structure is then used in order to compute the intersection with the visible surface. By avoiding a majority of the computations for occluded parts of the molecule, our method provides real-time performance even for large and dynamic data. The remainder of this section is subdivided according to the three main steps of our method. First, in Section 3.1, we present our approach for quickly identifying the atoms that potentially contribute to the visible surface. Next, in Section 3.2, we present a simple algorithm for visibility-based ray traversal that allows us to further minimize the amount of computation devoted to occluded regions of the data. Finally, we detail how to efficiently evaluate intersections between the viewing ray and the Gaussian surface in Section 3.3.

3.1. List Generation

A main challenge in the efficient rendering of Gaussian surfaces is the identification of contributing atoms. For every point in space, the surface is modeled as a sum of Gaussian functions. As we are interested in visualizing a representation that smoothly envelopes the van der Waals spheres, we can assume that the final surface will always be a union of the Gaussian surface and the van der Waals surface. While, in principle, the Gaussian footprint of an atom has infinite support, as is common, we specify a cutoff radius after which the contribution of a particular atom is considered zero. Hence, each atom is defined by its center $\mathbf{c_i}$, its van der Waals radius r_i, and the radius of its sphere of influence $r_i{}'$. Since an atom will not contribute to the density function outside its sphere of influence, the contributions of the density function to the final image can only occur in regions where two or more spheres of influence overlap. Furthermore, for each viewing ray, only those overlaps that are not occluded by a van der Waals sphere can potentially contribute to the surface, as illustrated in Figure 2.

Therefore, as an important first step, we can identify the first intersection of every viewing ray with any van der Waals sphere, which simply amounts to rendering the entire molecule as spheres. We use the standard approach of rendering only one vertex per atom, constructing a quad according to the screen space bounding box of the corresponding sphere, and computing the ray-sphere intersection analytically [SWBG06]. Next, for each viewing ray, we need information about which of the spheres of influence that it intersects actually overlap. For each atom, we render its sphere of influence in the same manner as the van der Waals spheres. However, instead of only storing the first intersection, all intersection points that are not occluded by a van der Waals sphere are recorded. To achieve this, we use an approach similar to the generation of an A-buffer for order-independent transparency [KKP*13, JPSK16], where a linked list of intersections is stored for each pixel. Each entry in the intersection list for a pixel stores the near and far intersection depths with the atom's sphere of influence, the center position of the atom, an attribute field, as well as the index of the previous list entry. The attribute field contains the element ID of the atom which is used to retrieve its radius, as well as other attribute information such as residue ID and chain ID, which are used for mapping additional quantities to the surface.

```
Input: an array of $count$ intersection point indices $ii$,
       referring to the $near$ and $far$ intersection depths of
       the respective atom spheres of influence
1  $ss = 0$
2  for $i = 0$ to $count - 1$ do
3      k = i
4      for $j = i + 1$ to $count - 1$ do
5          if $near_{ii[j]} < near_{ii[k]}$ then $k = j$
6      end
7      swap($ii[i]$,$ii[k]$)
8      if ss < i then
9          if $i \leq count - 1$ or $far_{ii[ss]} < near_{ii[i]}$ then
10             if intersect($ss$,$i$) then break
11             $ss = ss + 1$
12         end
13     end
14 end
```

Algorithm 1: Pseudocode of our algorithm for visibility-driven ray traversal.

3.2. Ray Traversal

After generating the list of intersections with the unoccluded spheres of influence for all atoms, our approach now proceeds by traversing viewing rays for each pixel in a front-to-back manner. In order to do so, the obvious next step would be to sort the list of intersections for each ray accordingly. However, considering the fact that the data is quite dense, sorting the entire list before proceeding further is rather inefficient, as we are only interested in the first intersection with the Gaussian surface. Instead, if we interleave surface intersection and sorting, we can terminate as soon as an intersection with the Gaussian surface has been detected, avoiding the need to sort the remainder of the list.

For this purpose, we use an adaptation of selection sort, which has the advantageous property that after the i-th iteration, the first i elements are always correctly sorted. This means that after iteration $i = 1$ (assuming zero-based indices), we can proceed to check whether the first two spheres of influence intersected by our ray overlap. If this is not the case, the sorting process continues. Otherwise, we are at the beginning of a span, i.e., two or more atoms whose spheres of influence overlap and which therefore need to be considered when testing for intersection with the Gaussian surface. As all overlapping atoms constituting the span need to be known, the intersection test can only commence once the end of a span has been reached, i.e., once we have identified that the current atom's sphere of influence does not overlap with the first sphere of the span, or if we have reached the end of the list. In both cases, we then initiate the intersection test, which is detailed in the next section. As soon as an intersection has been detected, the sorting loop can be terminated. While more efficient sorting algorithms could be used when the entire list has to be sorted (e.g., in the case of transparency), for opaque rendering this approach provides superior performance.

In practice, at the beginning of the ray traversal, we scan the intersection list once and store the indices into the global memory pool in a local array for more efficient access. In this way, during sorting, we only need to swap the locally stored indices instead of the larger list entries. The pseudocode for our simple algorithm is given in Algorithm 1. Its inputs are an array of unsorted intersection indices ii corresponding to the list entries for the current ray, as well as the list of near and far intersection depths for the spheres of influence. In each iteration, the algorithm finds the smallest near intersection depth in the remainder of the array and swaps it with position i, which means that the entries of ii from 0 to i are then correctly sorted. The span start index ss tracks the beginning of the current span. As soon as the end of a span has been detected, or all list entries have been traversed, intersection testing is performed for the corresponding interval of, now sorted, indices from ss to i.

3.3. Surface Intersection

Obviously, the algorithm described in the previous section requires an efficient procedure to determine the intersection of a ray with the Gaussian surface generated by a set of atoms. In general, the roots for a sum of an arbitrary number of exponential functions cannot be found analytically. In his original work on Metaballs [Bli82], Blinn therefore suggests the use of standard iterative methods such as Newton iteration and regula falsi. While Newton iteration converges relatively rapidly, it may diverge if the initial guess is not sufficiently close to the real solution. Regula falsi is guaranteed to converge if the initial interval contains a root, but typically considerably more slowly, and may not converge to the first root which would be desirable for ray casting [Har93]. Hence, most approaches begin with a root isolation phase, which divides the search interval into segments containing only a single root. Then, a root refinement phase uses a faster method to approach the root until the desired precision has been reached. As our intersection test will have considerable impact on the overall performance and quality, it is thus important to identify the best possible choice.

Hart [Har96] introduced sphere tracing as an efficient approach for rendering implicit surfaces based on signed distance functions. If a distance bound for an implicit function is available, the approach is guaranteed to converge linearly to the first root in the interval without

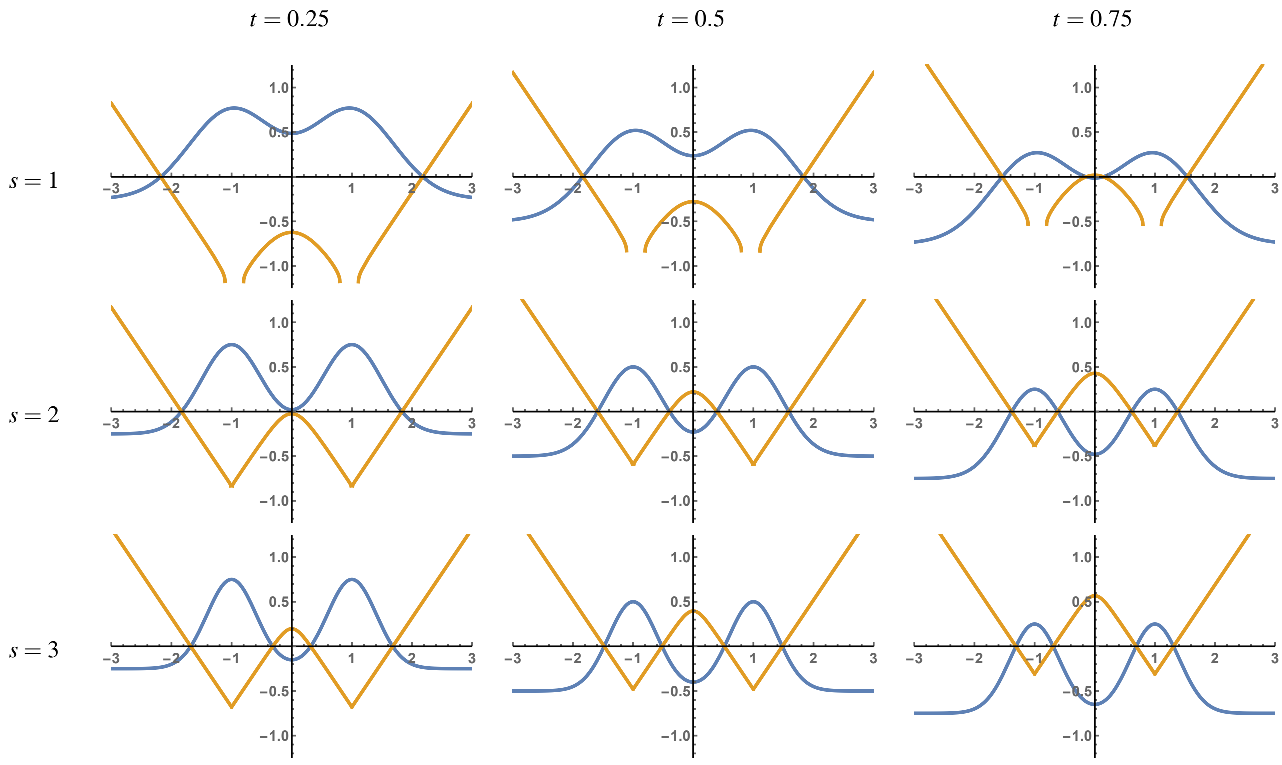

Figure 3: *Plots of the Gaussian density function $\rho(\mathbf{x}) - t$ (in blue) and the corresponding distance function $\hat{\rho}(\mathbf{x}) - \hat{t}$ (in orange) for different values of s and t for two atoms positioned at -1 and 1. As can be seen, the behavior of the distance estimate is close to linear, and the roots of both functions are the same.*

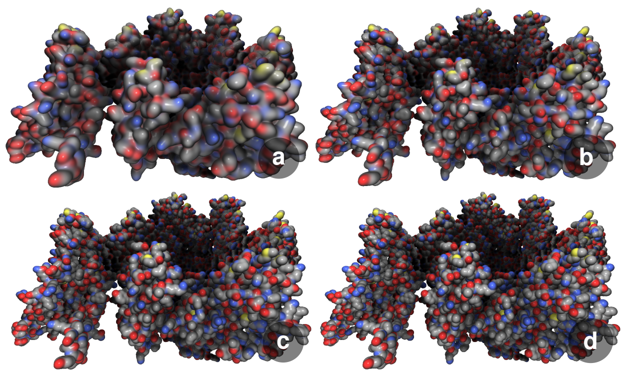

Figure 4: *Capsid protein (PDB ID: 6B0X) with CPK element coloring using (a) $s = 1$, (b) $s = 2$, (c) $s = 4$, and (d) $s = 8$ – as the value of s is increased, the surface approaches the van der Waals spheres.*

ever penetrating the surface. It has therefore been widely used for distance-function based modeling, and has shown to be a fast and robust approach for rendering as compared to more general methods. For Gaussian surfaces, Hart suggested the use of a polynomial approximation [WT90], for which a distance bound can be easily found. However, as we target molecular visualization, we want to preserve the appearance and properties of Gaussian surfaces.

Following Hart, a function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ is a signed distance bound of its implicit surface $f^{-1}(0)$, if and only if

$$|f(\mathbf{x})| \leq d(\mathbf{x}, f^{-1}(0)) \tag{2}$$

where d is the point-to-set distance between point $\mathbf{x} \in \mathbb{R}^3$ and a set $A \subset \mathbb{R}^3$ defined as

$$d(\mathbf{x}, A) = \min_{\mathbf{y} \in A} \|\mathbf{x} - \mathbf{y}\|. \tag{3}$$

Examining Equation 1, we see that for a single atom an ideal distance function can be easily obtained by applying the transformation $\sqrt{-\ln(\cdot)s^{-1}}$ to both sides of the equation $\rho(\mathbf{x}) = t$, as this will give us the normalized Euclidean distance to the center of the atom with an appropriately adjusted threshold. However, the same approach also works for multiple atoms, as the real roots of the equation will be unaffected by this transformation.

Given a density function $\rho(\mathbf{x})$ and corresponding threshold t as defined in Equation 1, our signed distance estimate is then

$$d_\rho(\mathbf{x}) = \hat{\rho}(\mathbf{x}) - \hat{t} \tag{4}$$

with

$$\hat{\rho}(\mathbf{x}) = \sqrt{-\frac{\ln(\rho(\mathbf{x}))}{s}} \tag{5}$$

and

$$\hat{t} = \sqrt{-\frac{\ln(t)}{s}}. \tag{6}$$

The function $\hat{\rho}(\mathbf{x}) - \hat{t}$ has the same roots as $\rho(\mathbf{x}) - t$, but progresses almost linearly for positions farther away from the surface, as can be seen in Figure 3 which shows the original function $\rho(\mathbf{x}) - t$ in blue and our distance estimate $\hat{\rho}(\mathbf{x}) - \hat{t}$ in orange for different combinations of s and t. While the infinities visible for $s = 1$ may seem problematic, we note that they only occur for negative values, i.e., inside the object, and therefore do not affect the outcome.

While Equation 4 allows us to reproduce the results of existing methods which use different values for s and t, not all combinations of these values produce meaningful results. Hence, in practice it has proven to be more convenient to fix $\hat{t}$ and let the user only specify s directly, which then simultaneously also adjusts t. In our implementation, we use $\hat{t} = 1$, which then means that due to Equation 6, t is always set to e^{-s}. Intuitively then, when s is increased the resulting surface will smoothly approach the van der Waals surface, as demonstrated in Figure 4.

This also leads us to a way to estimate an atom's sphere of influence, which we have so far omitted. The simplest approach, used by most common grid-based methods, is to use a fixed cutoff distance beyond which the contribution of an individual Gaussian in Equation 1 is assumed to be zero. As an alternative, we instead propose to set this distance according to an estimate of how many atoms will on average influence each position in space. The sphere of influence for each individual atom can then be chosen such that, if less than N atoms contribute to the density at a position, the sum of their contributions will be below the threshold. Assuming a base atom radius r_i, we thus want to find the sphere of influence radius r_i' such that

$$Ne^{\frac{-s r_i'^2}{r_i^2}} - t = 0. \tag{7}$$

Solving for r_i', we obtain

$$r_i' = r_i \sqrt{\frac{\ln\left(\frac{N}{t}\right)}{s}}. \tag{8}$$

Hence, using this approach we specify the radius of influence of an atom r_i' as its van der Waals radius r_i multiplied by the second term on the righthand side of Equation 8. As to be expected, this multiplier approaches infinity as s approaches zero (where the Gaussian density is constant), and smoothly approaches one as s increases. Figure 5 plots of the behavior of this function with respect to s for different values of N. For all the results in this paper, we use a value of $N = 32$, which results in $r_i' \approx 2.11 r_i$ for $s = 1$.

Using this approach, we can now use Hart's sphere tracing approach for intersection computation, and, due to the close-to-linear behavior of our function will do so at a rapid pace. The intersection function takes as input an interval of overlapping atom indices $[i, j]$ which are already sorted according to the closest intersection with the sphere of influence by Algorithm 1. Since we have information on the overlapping spheres, we can additionally set relatively tight bounds on the search range. As the Gaussian surface can only

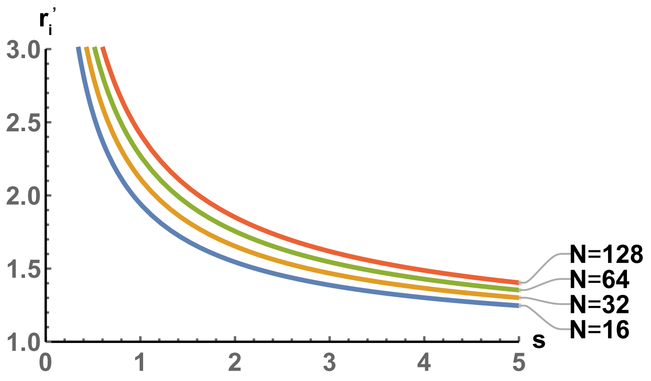

Figure 5: *Sphere of influence radius r_i' using $N \in \{16, 32, 64, 128\}$ for different values of s according to Equation 8 with $r = 1$ and $t = e^{-s}$.*

be intersected where the corresponding spheres overlap, only the range from $near_{i+1}$ to far_{j-1} needs to be considered. As an iterative method, sphere tracing requires the specification of a tolerance value. We use a value of 0.0125 Å – roughly 1% of the van der Waals radius of an hydrogen atom – which we determined experimentally to have no detectable impact on the quality, and all results in this paper were generated with this value.

We use a standard implementation of sphere tracing. During each iteration, the value of $\rho(\mathbf{x})$ is computed by summing up the respective contributions of each atom, and then $\hat{\rho}(\mathbf{x}) - \hat{t}$ is compared against the tolerance value, and, if larger, the ray is advanced in viewing direction by $\hat{\rho}(\mathbf{x}) - \hat{t}$. Since the exponential function is its own derivative, the normal at a point $\mathbf{x}$ can be computed analytically as

$$\frac{d\rho}{d\mathbf{x}} = \sum_i -\frac{2s(\mathbf{x} - \mathbf{c_i})}{r_i^2} e^{\frac{-s\|\mathbf{x}-\mathbf{c_i}\|^2}{r_i^2}}, \tag{9}$$

which only consists of quantities that are needed during the evaluation of the density function anyway, i.e., the normal can be computed during the summation process by simply weighting the vector from the center of the current atom to the point $\mathbf{x}$ accordingly.

Likewise, any additional quantities to be mapped onto the surface, such as colors or charges, can be easily interpolated in the same manner using the Gaussian weights for the respective atom. Using the actual Gaussian weights at the intersection points can have a considerable beneficial impact on the visual quality of the result images compared to triangulation-based methods, where there barycentric interpolation over the triangles can result in the diamond-shaped artifact well-known from Gouraud shading, as can be seen in Figure 11.

4. Implementation

Our method was implemented using C++ and OpenGL and runs completely on the GPU. Apart from the necessary boilerplate C++ code, the main steps of our technique are written in GLSL. The only input required are the atom positions and radii. To allow for a

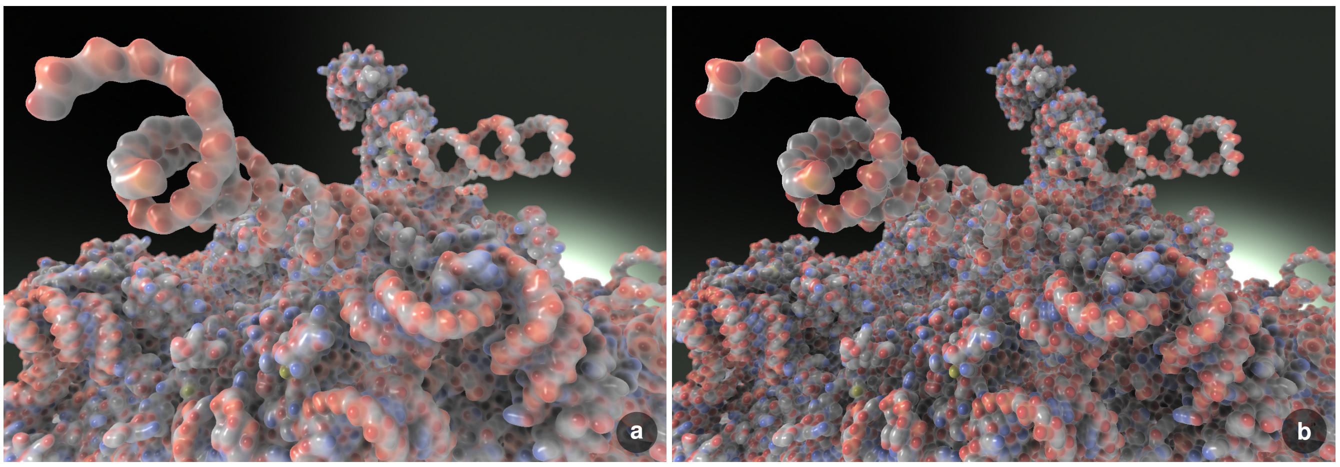

Figure 6: *Structure of the human 80S ribosome (PDB ID: 4V6X) consisting of 237K atoms with CPK element coloring rendered for (a) $s = 1$ and (b) $s = 2$ using distance-based blending between the molecular surface and the van der Waals spheres.*

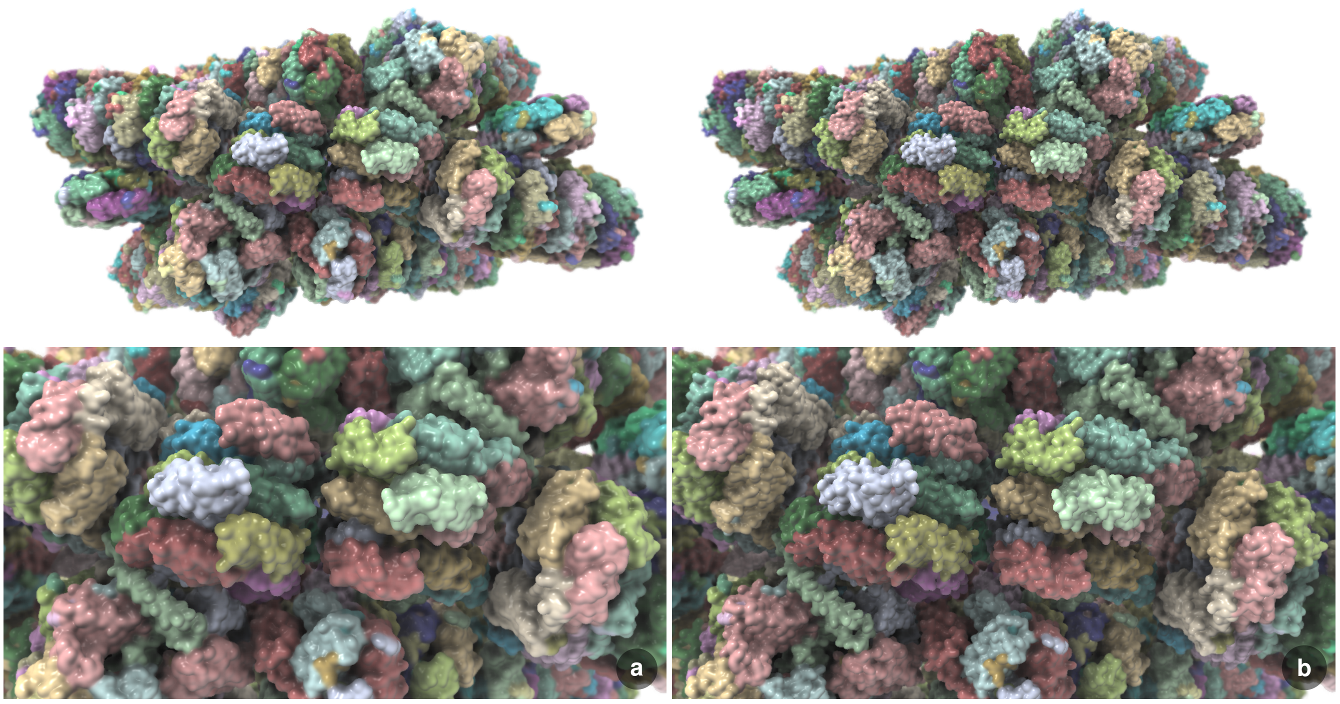

Figure 7: *Structure of the phycobilisome from the red alga Griffithsia pacifica (PDB ID: 5Y6P) consisting of 1.2M atoms rendered with chain coloring for (a) $s = 0.5$ and (b) $s = 1$.*

more flexible mapping of molecular attributes to optical properties, we store the atom positions as well as an attribute field in a single 4-component vertex buffer object. The attribute field is then used to look up the atom radii and any optionally mapped quantities from a set of uniform buffer objects. Our current implementation supports color mapping of element ID (see Figure 6), chain ID (see Figure 7), and residue ID, but other attributes can be added easily.

The vertex and geometry shaders for van der Waals sphere rendering and list generation are identical, but the fragment shader differs. For the van der Waals spheres, the depth, normal, and attribute field are stored in a framebuffer object. The list generation fragment shader does not write to the framebuffer, but instead uses an image object (initialized to zero) to store the index of the last list entry for each pixel, as well as a shader storage buffer object to store the intersection data. This buffer contains a single counter for the total number of allocated entries (initialized to zero), as well as an array

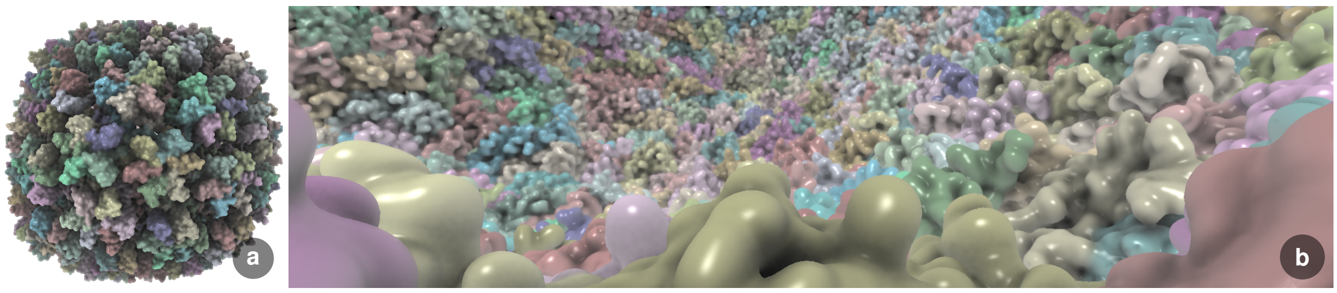

Figure 8: *Atomic-level structure of the entire HIV-1 capsid (PDB ID: 3J3Q) consisting of 2.4M atoms using chain coloring from (a) outside and (b) inside.*

of the actual list entry structures. Each entry stores the near and far intersection depths with a sphere of influence (one floating-point value each), the atom's center (3 floating-point values), the attribute field (one integer), and the index of the previous list entry (one integer), amounting to a total of 28 bytes per list entry. The memory for this buffer is allocated as a fixed pool on startup and resized if necessary.

At the beginning of the list generation fragment shader, the intersection with the sphere of influence is computed. If the near intersection depth is larger than the depth from the van der Waals pass, the fragment is immediately discarded and no output is generated. Otherwise, a new list entry is allocated and the tail of the list for the current pixel is updated. This requires the use of two atomic operations to avoid race conditions: first, the total entry count is atomically incremented to allocate a new list entry, and then the returned value is atomically exchanged with the last entry index for the corresponding pixel to update the tail of the list. The value returned by the atomic exchange operation is then used to set the previous index field of the new list entry.

The ray traversal is initiated by rendering a screen-filling quad. At the very beginning, the tail buffer is read and if it is zero (i.e., the list for the pixel is empty) the fragment is immediately discarded. The normal and material value (using the attribute field) for the current ray are initialized with the values from the van der Waals pass to ensure that the van der Waals spheres are always rendered when no other intersection occurs. The shader then proceeds as described Section 3.2 and 3.3. After the ray traversal has terminated, the surface normal and depth as well as the unshaded color are written to two 4-component textures which are subsequently used for deferred shading. Our prototype also supports screen space ambient occlusion using the method of McGuire et al. [MML12] as well as the depth-of-field technique by Bukowski et al. [BHOM13] which are applied at the usual stages of the deferred shading pipeline. This demonstrates that our approach can be easily integrated into modern rendering pipelines. Furthermore, as these effects – in particular screen space ambient occlusion – can be sensitive to normal and depth discontinuities such as those due to triangulation, the result images in this paper show that our method does not suffer from these problems.

Since our method only needs atom positions, radii, and optional attributes as input, handling time-dependent data such as molecular dynamics simulations is straight-forward. The individual time steps are stored in vertex buffer objects, and interpolation between two time steps is easily possible by passing the positions of two consecutive time steps to the vertex shader for van der Waals sphere rendering and list generation. Furthermore, if not all time steps can be resident on the GPU, a double (or multiple) buffering strategy can be used where the next time step is transferred asynchronously while the current one is rendered.

While our current implementation exploits some features only available in more recent OpenGL versions (e.g., uniform buffers and shader storage blocks) as they simplify the code, our method does not rely on any hardware capabilities that have not been available for at least two prior generations of GPUs. We did not perform any low-level optimizations, and the entire implementation comprises less than 1000 lines of shader code. Our method was implemented as a stand-alone prototype which we plan to make available to the community under a permissive license to facilitate integration into existing applications.

5. Results

In order to evaluate our approach, we performed extensive experiments using data from the Protein Data Bank (PDB) ranging from thousands to millions of atoms. Our test system was an Intel Core i7-4939K 3.40 GHz CPU equipped with an NVIDIA GeForce 1080 GTX GPU. Figures 6, 7, and 8 show data sets consisting of 237K, 1.2M, and 2.4M atoms, respectively, all of which our method is able to easily render at interactive frame rates. In Figures 6 and 7, we show the visual effect of increasing s, which results in more surface details, eventually smoothly approaching the van der Waals spheres of the individual atoms. Figure 8 also demonstrates that there are no restrictions on the viewpoint by showing an outside and an inside view of the depicted structure.

In Table 1, we give the frame rates for a representative selection of molecular data sets, all shown in this paper, ranging from relatively small to large for different viewport sizes and values of s. The bounding boxes of the data sets were scaled to fully fit within the viewport dimensions to avoid any clipping of the model, and the measurements represent minimum, maximum, and average frame rates for 360 degree rotations of the model around its center along the principal coordinate axes over 1080 frames (360 frames per axis). As can be seen, our approach remains fully interactive even for millions of atoms and high resolutions. Even for a viewport size of

Table 1: *Rendering performance of our method in frames per second ($^{min}_{max}$avg) as measured on an Intel Core i7-4939K 3.40 GHz CPU equipped with an NVIDIA GeForce 1080 GTX GPU. The bounding boxes of the data sets were scaled to fully fit within the viewport, and the frame rates were computed for three full rotations of the model around its center along the principal coordinate axes over 1080 frames (360 frames per axis).*

PDB ID	Atoms	768^2 Viewport				1024^2 Viewport				1280^2 Viewport			
		$s=1$	$s=2$	$s=3$	$s=4$	$s=1$	$s=2$	$s=3$	$s=4$	$s=1$	$s=2$	$s=3$	$s=4$
6B0X	16 791	$^{98}_{221}148$	$^{175}_{504}350$	$^{228}_{777}538$	$^{297}_{990}708$	$^{63}_{161}94$	$^{121}_{335}228$	$^{169}_{507}366$	$^{184}_{649}472$	$^{53}_{127}77$	$^{111}_{276}182$	$^{154}_{405}289$	$^{179}_{506}377$
5VOX	34 724	$^{97}_{201}156$	$^{201}_{511}402$	$^{277}_{842}642$	$^{323}_{1089}821$	$^{70}_{129}100$	$^{137}_{337}265$	$^{191}_{524}420$	$^{227}_{676}551$	$^{62}_{105}82$	$^{125}_{272}214$	$^{166}_{420}337$	$^{205}_{538}435$
1AON	58 870	$^{91}_{231}150$	$^{182}_{510}357$	$^{241}_{773}538$	$^{254}_{933}678$	$^{55}_{137}93$	$^{110}_{323}225$	$^{165}_{476}347$	$^{183}_{587}435$	$^{51}_{113}76$	$^{105}_{266}183$	$^{146}_{385}279$	$^{173}_{471}349$
5OT7	154 665	$^{80}_{129}105$	$^{152}_{305}250$	$^{203}_{450}373$	$^{231}_{569}476$	$^{51}_{82}65$	$^{101}_{199}154$	$^{129}_{301}241$	$^{165}_{376}311$	$^{43}_{64}53$	$^{92}_{159}131$	$^{126}_{232}193$	$^{148}_{294}247$
4V6X	237 685	$^{80}_{135}111$	$^{148}_{317}262$	$^{197}_{461}381$	$^{229}_{566}474$	$^{51}_{87}67$	$^{101}_{210}161$	$^{137}_{312}245$	$^{165}_{386}313$	$^{42}_{68}56$	$^{89}_{165}137$	$^{118}_{245}199$	$^{144}_{304}250$
4V4G	717 805	$^{106}_{261}198$	$^{163}_{394}324$	$^{202}_{451}382$	$^{213}_{483}416$	$^{82}_{214}154$	$^{136}_{347}271$	$^{160}_{406}333$	$^{180}_{436}365$	$^{75}_{178}123$	$^{127}_{310}234$	$^{160}_{368}300$	$^{172}_{402}335$
5Y6P	1 234 811	$^{74}_{125}100$	$^{117}_{210}172$	$^{138}_{248}211$	$^{150}_{270}232$	$^{49}_{83}68$	$^{89}_{165}138$	$^{115}_{210}179$	$^{125}_{236}205$	$^{44}_{70}57$	$^{11}_{143}116$	$^{76}_{187}156$	$^{118}_{209}180$
3J3Q	2 440 800	$^{57}_{88}71$	$^{87}_{128}110$	$^{98}_{144}127$	$^{106}_{155}139$	$^{40}_{68}51$	$^{69}_{109}89$	$^{79}_{128}108$	$^{88}_{136}119$	$^{36}_{58}43$	$^{66}_{100}82$	$^{79}_{120}102$	$^{85}_{131}113$

1280^2 and a dataset consisting of 2.4M atoms, we still achieve over 40 frames per second for $s = 1$. As can be seen, since our method is visibility-driven, it scales sublinearly with the atom count. In some cases, larger data sets may even achieve higher frame rates than smaller ones. This is not unexpected, as it means that the specific geometric arrangement has a heavier impact on the performance than the number of atoms. For instance, 4V4G has a highly asymmetric and elongated shape (see Figure 1), so its projection on the image plane will cover less pixels than more spherical data sets. Of course our approach is not completely independent of the atom count, as the list generation phase still requires us to compute the intersection with the van der Waals spheres and spheres of influence for all atoms, but the much costlier intersection with the Gaussian surface significantly benefits from our visibility-based method.

To analyze this further, we performed measurements for the individual phases of our technique. In Figure 9, we show how the total render time is divided among the rendering of the van der Waals spheres ("Spheres"), list generation ("List"), ray traversal and surface intersection ("Surface"), and final shading ("Shading") for the extremal values of $s = 1$ and $s = 4$ and a viewport size of 1280^2. For smaller data sets, sphere rendering and list generation only account for a relatively small fraction of the total render time, but as the number of atoms increases, their relative impact rises, in particular for higher values of s as in these cases the ray traversal will terminate more quickly. For $s = 1$, surface intersection still dominates the render times even for the largest tested data sets. Figure 10 additionally depicts how the total render time scales with increasing sphere of influence radius (corresponding to decreasing values of s) for a viewport size of 1280^2. This figure additionally includes measurements for $s = 0.5$ where our method approaches its limitations with respect to interactivity, achieving 16 frames per second for the largest data set.

The presented measurements include the costs for regular Phong shading, but ambient occlusion and depth of field were disabled. The overhead for these optional effects only depends on the image size, and is around 1.5 ms each at a viewport size of 1280^2 in our implementation. This means that for the largest dataset 3J3Q with $s = 1$ and a viewport size of 1280^2, the performance drops from 43 to 40 frames per second when both effects are enabled.

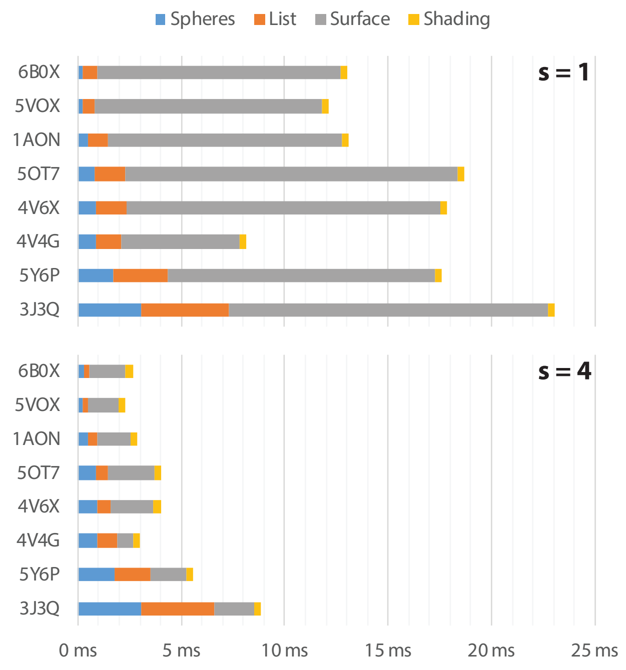

Figure 9: *Distribution of the total render time among the different phases of our method for a viewport size of* 1280^2 *and values of* $s = 1$ *and* $s = 4$.

The data sets listed in Table 1 were static, since we did not have access to simulation data spanning a representative range of atom counts. However, the handling of time-dependent data comes at no additional costs with our method as everything is computed on-the-fly for every frame, which we also verified experimentally.

We compare our method with Krone et al.'s QuickSurf approach [KSES12], as available in the popular software package Visual Molecular Dynamics (VMD) [HDS96]. This heavily optimized CUDA-based implementation uses a parallel gathering ap-

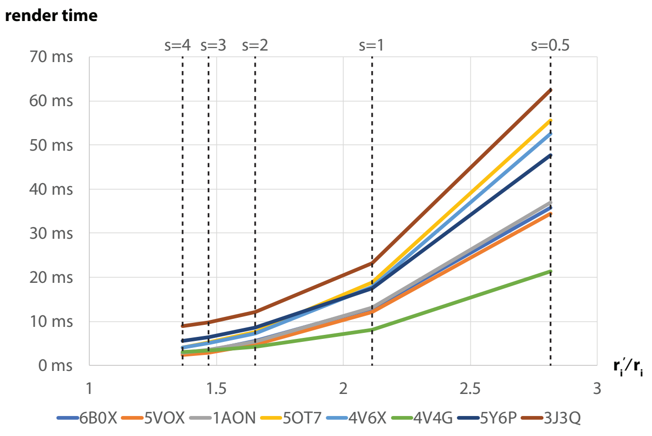

Figure 10: *Scaling of the total render time with increasing sphere of influence radius according to Equation 8 for a viewport size of* 1280^2.

Table 2: *Performance of the QuickSurf method in frames per second as implemented in VMD 1.9.3 using a viewport size of* 1280^2. *The column "Full" gives the total performance including computation of the surface, while the column "Render" only gives the rendering times once the surface has been computed.*

PDB ID	Grid res. 1.0		Grid res. 0.7		Grid res. 0.5	
	Full	Render	Full	Render	Full	Render
6B0X	74	120	29	96	12	44
5VOX	22	48	10	22	5	14
1AON	27	62	10	21	4	9
5OT7	12	29	4	5	3	8
4V6X	8	31	1	3	n/a	n/a
4V4G	2	6	n/a	n/a	n/a	n/a
5Y6P	2	5	n/a	n/a	n/a	n/a
3J3Q	0.8	4	n/a	n/a	n/a	n/a

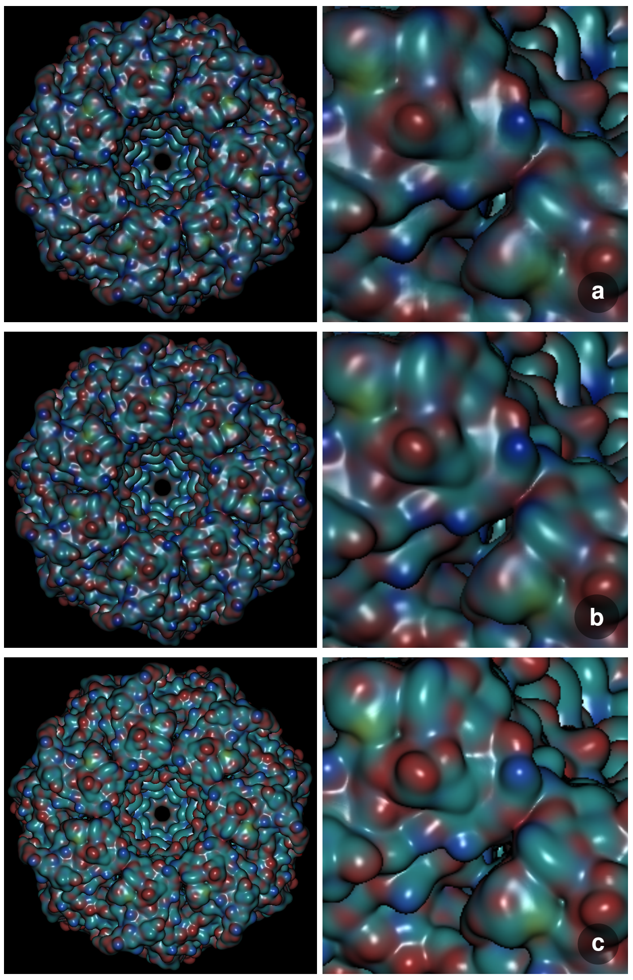

Figure 11: *Comparison of QuickSurf's Marching Cubes surface with a grid spacing of (a) 1 and (b) 0.5 to (c) our method (PDB ID: 1AON).*

proach for computing the Gaussian density grid, as well as a parallel version of the Marching Cubes [LC87] algorithm. Accounting for the differences in the used graphics hardware, the results we obtained are consistent with the original experiments presented by Krone et al. Per default, QuickSurf uses a rather coarse grid spacing of 1 Å, which needs to be reduced if more surface details are desired, so we measured the performance for grid spacings of 1, 0.7, and 0.5. The results are shown in Table 2. As this method is an object space approach, we give the frame rates for the total computation time including grid computation and Marching Cubes (column "Full"), which need to be performed for every time step in the case of dynamic data, as well as the rendering time alone (column "Render"). We used the same data sets and viewing setup as in Table 1. As the main computational load of the QuickSurf approach (grid generation and Marching Cubes) is independent of the viewport size, we only performed measurements for the largest size of 1280^2 for a fair comparison. As can be seen, QuickSurf only performs on par with our technique for the smallest of the data sets and the coarsest grid resolution, and as atom counts exceed 100K the interactivity of the method becomes quite limited. It is interesting to note that for larger molecules even the rendering frame rates without recomputation are significantly lower than ours. This is likely due to the high geometry load caused by the excessive number of triangles produces by Marching Cubes, whereas our method only requires one vertex per atom. While we were able to measure the performance for all the data sets using the default grid resolution, we were not able to obtain results for fine resolutions of the larger data sets. When comparing our approach to the triangulated surfaces in terms of quality, we also see that the default grid spacing of 1 exhibits the typical artifacts caused by barycentric interpolation of surface properties across triangles, as shown in Figure 11 (a), even if per-pixel lighting is enabled. Finer grid resolutions remedy this to some degree, as depicted in Figure 11 (b), but the appearance is still more blurry than for our method shown in Figure 11 (c).

Table 3 lists the mean values for maximum and average number of entries per pixel in the intersection list (a longer intersection

Table 3: *This table lists the mean values for maximum and average number of intersection list entries per pixel, as well as pixel coverage and memory consumption measured for a viewport size of* 1280^2 *and* $s = 1$.

PDB ID	Max. entries	Avg. entries	Coverage	Memory
6B0X	60.30	1.17	13.63 %	47.20 MB
5VOX	76.52	0.85	9.37 %	34.24 MB
1AON	57.56	1.04	12.01 %	42.18 MB
5OT7	67.59	1.37	15.30 %	55.54 MB
4V6X	68.69	1.22	13.76 %	49.38 MB
4V4G	67.79	0.31	3.26 %	12.43 MB
5Y6P	66.89	0.89	10.02 %	35.88 MB
3J3Q	71.43	1.09	11.99 %	43.92 MB

list will result in lower performance, as it bounds the number of iterations that need to be performed in Algorithm 1) as well as the memory required to store all list entries for the largest tested viewport size of 1280^2 and $s = 1$, averaged in the same manner as for Table 1. The table also gives the average percentage of viewport pixels covered by the model. In Figure 12 (b), we also show an example of how the number of entries in the intersection list is distributed over the image. As can be seen in Table 3, the size of the intersection lists is independent of the atom count, and results in a very modest memory consumption of only around 50 MB. Together with the memory requirements for the framebuffers, which vary depending on whether any additional postprocessing steps such as screen space ambient occlusion are used and are similar to those of any other deferred shading pipeline, the total memory footprint of our technique is only about 150 MB for a viewport size of 1280^2. This means that our method is also suitable for lower-end GPUs such as those available on laptop computers which typically have less memory.

We would like to note that QuickSurf is a highly capable solution for the generation of triangulated surfaces from molecular data, that outperforms other implementations by orders of magnitude, and is among the best methods for extracting surfaces for analysis purposes. However, for the visualization of dynamic surfaces our method represents an attractive alternative in terms of performance, quality, flexibility, and memory consumption.

As our method is fully dynamic, it is also straight-forwardly possible to vary surface parameters in a view-dependent manner. As an example, we implemented a magic lens tool which varies the s parameter continuously based on the image space distance from the mouse cursor. While very simple, the result is quite compelling as the surface dynamically and gradually shrinks towards the van der Waals spheres of nearby atoms, as demonstrated in Figure 13. Similarly, we could employ a seamless abstraction approach to adjust surface details based on the viewing distance [PJR*14] or other properties.

6. Discussion and Limitations

As we have shown, our approach is capable of rendering even large molecules with high quality and performance, and only a modest memory footprint. As it is fully dynamic, we are able to handle time-dependent and even procedurally generated data sets without any overhead. One limitation of our method is that its performance is dependent on the scaling factor s – as we showed, lower values of s, where a larger neighborhood of atoms is taken into account, will reduce the achievable frame rates. As demonstrated in Figure 10, reasonably interactive performance for the largest tested data sets is still achievable at $s = 0.5$, but when the value is further reduced our approach hits its limitations. For very large scale structures consisting of tens of millions of atoms, this will limit the performance of our method if a high degree of surface abstraction is desired. However, in such cases a prior generation of a level-of-detail hierarchy by merging nearby spheres is a meaningful solution, since the projection of a single sphere is likely to only cover very few pixels. Our method is fully compatible with such approaches and its dynamic nature enables smooth blending between hierarchy levels. Furthermore, for approximating the SES with a standard probe radius of 1.4 Å, Bernstein et al. [BC10] give values of s close to 1 (note that in their formulation, s is defined in a reciprocal manner). In addition, it is relevant to mention that our method's behavior is inverse to grid-based methods with respect to this parameter, which generally achieve better performance when a higher degree of smoothing is performed, as coarser grid resolutions can be used.

While Gaussian surfaces are useful and have been shown to provide a meaningful and relevant representation for molecular visualization [LCL15], it would nonetheless be desirably to have methods of similar performance for other types of molecular surfaces. There are cases when a Gaussian approximation is insufficient, and current approaches for SES visualization are limited in their performance

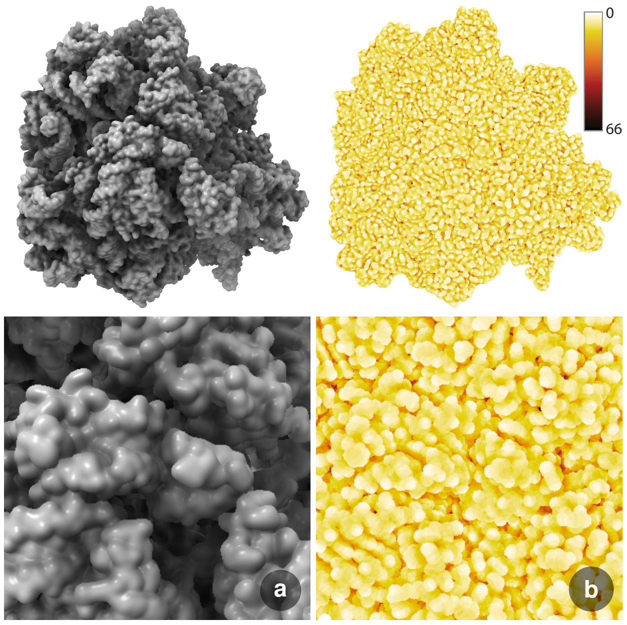

Figure 12: *Structure of an elongation factor G-ribosome complex (PDB ID: 5OT7). (a) Rendering for* $s = 1$. *(b) Number of list entries per pixel mapped to color.*

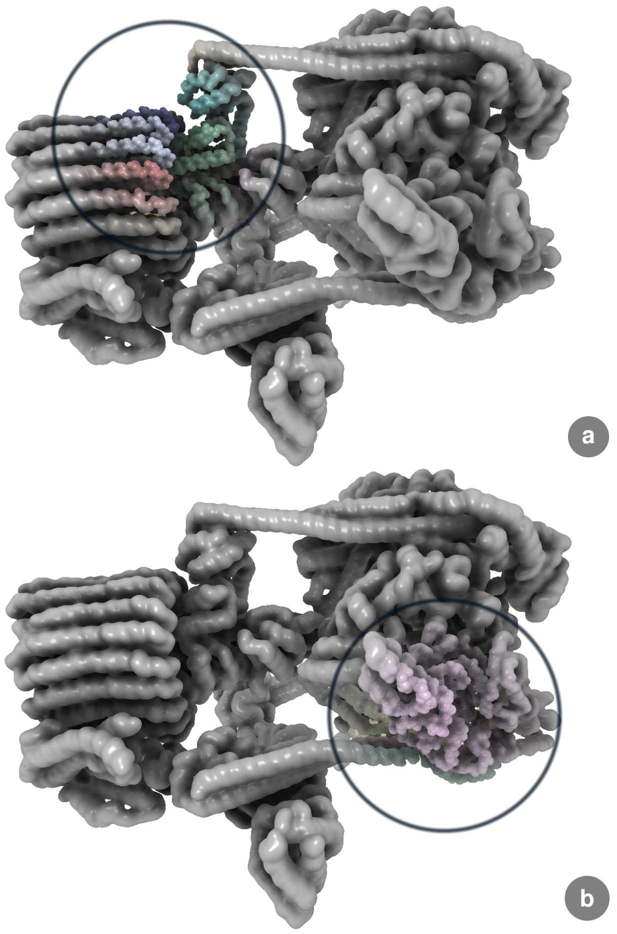

Figure 13: *Dynamic lensing effect adjusting the value of s to reveal detail structures for the lens positioned at (a) the top left, and (b) the lower right of the image – the focus region additionally uses coloring by chain (PDB ID: 5VOX).*

and ability to handle large data. In future work, we therefore plan to investigate the applicability of our approach to other types of molecular surfaces, in particular the SES and the Molecular Skin Surface (MSS) [Ede99]. While the list generation and ray traversal phases of our method should be directly applicable, intersection testing may be more complex as different types of patches need to be identified and clipped. The SES, in particular, can exhibit singularities [KBE09] that can only be resolved using non-local information, which may prove challenging to do efficiently.

Finally, while our method was specifically developed in the context of molecular visualization, we note that the technique itself is applicable to other types of particle-based data. For instance, we believe that our approach could be valuable in the context of real-time Smoothed Particle Hydrodynamics (SPH) fluid simulation, enabling a fully GPU-based simulation and visualization loop.

7. Conclusion

We presented a novel approach for the visualization of Gaussian molecular surfaces. By exploiting visibility information, we were able to develop an efficient method which enables the rendering of large dynamic data sets without the need for an intermediate grid representation. Our technique uses an on-the-fly generated list of intersections with the atoms' spheres of influence, and interleaves visibility sorting and intersection testing to avoid unnecessary computations. Using real-world data sets including large structures consisting of millions of atoms, we demonstrated that our method achieves high performance and quality, while having only a small memory footprint.

Acknowledgments

The research presented in this paper was supported by the MetaVis project (#250133) funded by the Research Council of Norway.

References

[BC10] Bernstein H. J., Craig P. A.: Efficient molecular surface rendering by linear-time pseudo-Gaussian approximation to Lee–Richards surfaces (PGALRS). *Journal of Applied Crystallography 43*, 2 (2010), 356–361. doi:10.1107/S0021889809054326. 2, 11

[BHOM13] Bukowski M., Hennessy P., Osman B., McGuire M.: The Skylanders SWAP Force depth-of-field shader. In *GPU Pro 4: Advanced Rendering Techniques*. 2013, pp. 175–184. 8

[Bli82] Blinn J. F.: A generalization of algebraic surface drawing. *ACM Transactions on Graphics 1*, 3 (1982), 235–256. doi:10.1145/357306.357310. 2, 4

[CCW06] Can T., Chen C.-I., Wang Y.-F.: Efficient molecular surface generation using level-set methods. *Journal of Molecular Graphics and Modelling 25*, 4 (2006), 442–454. doi:10.1016/j.jmgm.2006.02.012. 2

[Con83] Connolly M.: Analytical molecular surface calculation. *Journal of Applied Crystallography 16*, 5 (1983), 548–558. doi:10.1107/S0021889883010985. 2

[DG11] Dias S. E., Gomes A. J.: Graphics processing unit-based triangulations of blinn molecular surfaces. *Concurrency and Computation: Practice and Experience 23*, 17 (2011), 2280–2291. doi:10.1002/cpe.1783. 2, 3

[dSBVeSTT18] dos Santos Brito C. D., Vieira e Silva A. L. B., Teixeira J. M., Teichrieb V.: Ray tracer based rendering solution for large scale fluid rendering. *Computers & Graphics 77* (2018), 65–79. doi:10.1016/j.cag.2018.09.019. 2

[Ede99] Edelsbrunner H.: Deformable smooth surface design. *Discrete & Computational Geometry 21*, 1 (1999), 87–115. doi:10.1007/PL00009412. 12

[FAW10] Fraedrich R., Auer S., Westermann R.: Efficient high-quality volume rendering of sph data. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1533–1540. doi:10.1109/TVCG.2010.148. 2

[FW08] Falk M., Weiskopf D.: Output-sensitive 3D line integral convolution. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (2008), 820–834. doi:10.1109/TVCG.2008.25. 3

[Har93] Hart J. C.: Ray tracing implicit surfaces. SIGGRAPH 1993 Course Notes: Design, Visualization and Animation of Implicit Surfaces, 1993. 4

[Har96] Hart J. C.: Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10 (1996), 527–545. doi:10.1007/s003710050084. 4

[HDS96] HUMPHREY W., DALKE A., SCHULTEN K.: VMD: Visual molecular dynamics. *Journal of Molecular Graphics 14*, 1 (1996), 33 – 38. doi:10.1016/0263-7855(96)00018-5. 9

[HGO*15] HOSPITAL A., GOÑI J. R., OROZCO M., , GELPÍ J. L.: Molecular dynamics simulations: advances and applications. *Advances and Applications in Bioinformatics and Chemistry 8* (2015), 37–47. doi:10.2147/AABC.S70333. 1

[HKG*17] HERMOSILLA P., KRONE M., GUALLAR V., VÁZQUEZ P.-P., VINACUA À., ROPINSKI T.: Interactive GPU-based generation of solvent-excluded surfaces. *The Visual Computer 33*, 6 (2017), 869–881. doi:10.1007/s00371-017-1397-2. 2

[HOK16] HOCHSTETTER H., ORTHMANN J., KOLB A.: Adaptive sampling for on-the-fly ray casting of particle-based fluids. In *Proc. High-Performance Graphics* (2016), pp. 129–138. doi:10.2312/hpg.20161199. 2

[JPSK16] JURČÍK A., PARULEK J., SOCHOR J., KOZLÍKOVÁ B.: Accelerated visualization of transparent molecular surfaces in molecular dynamics. In *Proc. IEEE PacificVis* (2016), pp. 112–119. doi:10.1109/PACIFICVIS.2016.7465258. 4

[KBE09] KRONE M., BIDMON K., ERTL T.: Interactive visualization of molecular surface dynamics. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1391–1398. doi:10.1109/TVCG.2009.157. 2, 12

[KGE11] KRONE M., GROTTEL S., ERTL T.: Parallel contour-buildup algorithm for the molecular surface. In *Proc. IEEE BioVis* (2011), pp. 17–22. doi:10.1109/BioVis.2011.6094043. 2

[KKF*17] KOZLÍKOVÁ B., KRONE M., FALK M., LINDOW N., BAADEN M., BAUM D., VIOLA I., PARULEK J., HEGE H.: Visualization of biomolecular structures: State of the art revisited. *Computer Graphics Forum 36*, 8 (2017), 178–204. doi:10.1111/cgf.13072. 2, 3

[KKP*13] KAUKER D., KRONE M., PANAGIOTIDIS A., REINA G., ERTL T.: Rendering molecular surfaces using order-independent transparency. In *Proc. Eurographics Symposium on Parallel Graphics and Visualization* (2013), pp. 33–40. doi:10.2312/EGPGV/EGPGV13/033-040. 4

[KSES12] KRONE M., STONE J., ERTL T., SCHULTEN K.: Fast visualization of Gaussian density surfaces for molecular dynamics and particle system trajectories. In *Proc. EuroVis Short Papers* (2012), pp. 67–71. doi:10.2312/PE/EuroVisShort/EuroVisShort2012/067-071. 2, 9

[KSN08] KANAMORI Y., SZEGO Z., NISHITA T.: GPU-based fast ray casting for a large number of metaballs. *Computer Graphics Forum 27*, 2 (2008), 351–360. doi:10.1111/j.1467-8659.2008.01132.x. 2

[KWN*14] KNOLL A., WALD I., NAVRATIL P., BOWEN A., REDA K., PAPKA M. E., GAITHER K.: RBF volume ray casting on multicore and manycore CPUs. *Computer Graphics Forum 33*, 3 (2014), 71–80. doi:10.1111/cgf.12363. 2

[LBPH10] LINDOW N., BAUM D., PROHASKA S., HEGE H.: Accelerated visualization of dynamic molecular surfaces. *Computer Graphics Forum 29*, 3 (2010), 943–952. doi:10.1111/j.1467-8659.2009.01693.x. 2

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Grapics 21*, 4 (1987), 163–169. doi:10.1145/37402.37422. 2, 10

[LCL15] LIU T., CHEN M., LU B.: Parameterization for molecular Gaussian surface and a comparison study of surface mesh generation. *Journal of Molecular Modeling 21*, 5 (2015), 113. doi:10.1007/s00894-015-2654-9. 3, 11

[MGE07] MÜLLER C., GROTTEL S., ERTL T.: Image-space GPU metaballs for time-dependent particle data sets. In *Proc. VMV* (2007), pp. 31–40. 2

[MML12] MCGUIRE M., MARA M., LUEBKE D.: Scalable ambient obscurance. *Proc. High-Performance Graphics* (2012), 97–103. doi:10.2312/EGGH/HPG12/097-103. 8

[PB13] PARULEK J., BRAMBILLA A.: Fast blending scheme for molecular surface representation. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (2013), 2653–2662. doi:10.1109/TVCG.2013.158. 2

[PJR*14] PARULEK J., JÖNSSON D., ROPINSKI T., BRUCKNER S., YNNERMAN A., VIOLA I.: Continuous levels-of-detail and visual abstraction for seamless molecular visualization. *Computer Graphics Forum 33*, 6 (2014), 276–287. doi:10.1111/cgf.12349. 11

[PV12] PARULEK J., VIOLA I.: Implicit representation of molecular surfaces. In *Proc. IEEE PacificVis* (2012), pp. 217–224. doi:10.1109/PacificVis.2012.6183594. 2

[Ric77] RICHARDS F. M.: Areas, volumes, packing, and protein structure. *Annual Review of Biophysics and Bioengineering 6*, 1 (1977), 151–176. doi:10.1146/annurev.bb.06.060177.001055. 2

[ŠBS*17] ŠOLTÉSZOVÁ V., BIRKELAND Å., STOPPEL S., VIOLA I., BRUCKNER S.: Output-sensitive filtering of streaming volume data. *Computer Graphics Forum 36*, 1 (2017), 249–262. doi:10.1111/cgf.12799. 3

[SOS96] SANNER M. F., OLSON A. J., SPEHNER J.: Reduced surface: An efficient way to compute molecular surfaces. *Biopolymers 38*, 3 (1996), 305–320. doi:10.1002/(SICI)1097-0282(199603)38:3<305::AID-BIP4>3.0.CO;2-Y. 2

[SWBG06] SIGG C., WEYRICH T., BOTSCH M., GROSS M.: Gpu-based ray-casting of quadratic surfaces. In *Proc. Point-Based Graphics* (2006), pp. 59–65. doi:10.2312/SPBG/SPBG06/059-065. 4

[TA96] TOTROV M., ABAGYAN R.: The contour-buildup algorithm to calculate the analytical molecular surface. *Journal of Structural Biology 116*, 1 (1996), 138–143. doi:10.1006/jsbi.1996.0022. 2

[VBWW97] VARSHNEY A., BROOKS JR. F., WILLIAM J., WRIGHT W.: Linearly scalable computation of smooth molecular surfaces. *IEEE Computer Graphics and Applications 14*, 5 (1997), 19–25. 2

[WT90] WYVILL G., TROTMAN A.: Ray-tracing soft objects. In *Proc. Computer Graphics International* (1990), pp. 469–476. 5

[ZD18] ZIRR T., DACHSBACHER C.: Memory-efficient on-the-fly voxelization and rendering of particle data. *IEEE Transactions on Visualization and Computer Graphics 24*, 2 (2018), 1155–1166. doi:10.1109/TVCG.2017.2656897. 2