Abstract

Molecular Dynamics simulations are an essential tool for many applications. The simulation of large molecules—like proteins—over long trajectories is of high importance e.g. for pharmaceutical, biochemical and medical research. For analyzing these data sets interactive visualization plays a crucial role as details of the interactions of molecules are often affected by the spatial relations between these molecules. From the large range of visual representations for such data, molecule surface representations are of high importance as they clearly depict geometric interactions, such as docking or substrate channel accessibility. However, these surface visualizations are computationally demanding and thus pose a challenge for interactive visualization of time-dependent data sets.

We propose an optimization of the Contour-Buildup algorithm for the Solvent Excluded Surface (SES) to remedy this issue. An optimized subdivision of calculation tasks of the original algorithm allows for full utilization of massive parallel processing hardware. Our approach is especially well suited for modern graphics hardware employing the CUDA programming language. As we do not rely on any pre-computations our method is intrinsically applicable to time-dependent data with arbitrarily long trajectories. We are able to visualize the SES for molecules with up to ten thousand atoms interactively on standard consumer graphics cards.


1 Introduction

Visualization of large molecules, especially bio-molecules, is of high importance for many research fields, as the function of these molecules often not only depends on their chemical setup but also on their spatial structure. The most widely used model is the ball-and-stick representation, which shows all atoms and their covalent bonds explicitly as spheres and interconnecting cylinders. The cartoon representation [24] depicts the protein’s functional structure, abstracting from the detailed atomistic information to a more focused visualization.

When studying molecule-molecule or molecule-solvent interactions, such as in biomedical research and drug design, surface representations of the molecules that depict areas exposed to potential reaction partners gain importance. Phenomena appearing at the boundaries of molecules include, among others, docking and substrate channel accessibility. However, these surface visualizations are computationally demanding and thus pose a challenge for interactive visualization of time-dependent data sets. We propose an optimization of the Contour-Buildup algorithm for the Solvent Excluded Surface (SES) to remedy this issue. An optimized subdivision of calculation tasks of the original algorithm allows for full utilization of massive parallel processing hardware. Our approach is especially well suited for modern graphics hardware employing the CUDA programming language. As we do not rely on any pre-computations our method is intrinsically applicable to time-dependent data with arbitrarily long trajectories. We are able to visualize the SES for molecules with up to ten thousand atoms interactively on standard consumer graphics cards.


1 Introduction

Visualization of large molecules, especially bio-molecules, is of high importance for many research fields, as the function of these molecules often not only depends on their chemical setup but also on their spatial structure. The most widely used model is the ball-and-stick representation, which shows all atoms and their covalent bonds explicitly as spheres and interconnecting cylinders. The cartoon representation [24] depicts the protein’s functional structure, abstracting from the detailed atomistic information to a more focused visualization.

When studying molecule-molecule or molecule-solvent interactions, such as in biomedical research and drug design, surface representations of the molecules that depict areas exposed to potential reaction partners gain importance. Phenomena appearing at the boundaries of molecules include, among others, docking and substrate channel accessibility. However, these surface visualizations are computationally demanding and thus pose a challenge for interactive visualization of time-dependent data sets. We propose an optimization of the Contour-Buildup algorithm for the Solvent Excluded Surface (SES) to remedy this issue. An optimized subdivision of calculation tasks of the original algorithm allows for full utilization of massive parallel processing hardware. Our approach is especially well suited for modern graphics hardware employing the CUDA programming language. As we do not rely on any pre-computations our method is intrinsically applicable to time-dependent data with arbitrarily long trajectories. We are able to visualize the SES for molecules with up to ten thousand atoms interactively on standard consumer graphics cards.

image of this surface directly on the GPU. Using this technique to render the SES enabled interactive visualization of time-dependent data of medium-sized molecules for the first time [17]. Spherical triangles and toroidal patches can be transferred to the graphics card as one single graphical primitive each, massively reducing the data transfer load, and the analytic solution of the ray casting of these surface patches guarantees pixel-accurate images.

Even though the rendering of the SES is no longer a problem, calculating the graphical primitives based solely on the atom positions is still computationally intensive. These can be evaluated by different algorithms, such as evaluating the Reduced Surface [28], α-shapes [8], or following the Contour-Buildup algorithm [33] (cf. Fig. 1). The needed calculation work increases for large, time-dependent data sets, when the long trajectory and the resulting huge amount of data prohibits pre-calculation of the data, or, even worse, when performing in-situ visualization in parallel to a simulation, where the latency needs to be as low as possible. In the future the computing power of processors will likely scale with the number of cores rather than the clock rates. Thus, the algorithms need to be designed to scale well in such highly parallel environments. Modern graphics cards, with their freely programmable streamline processors and high level, general purpose programming APIs, such as CUDA or OpenCL, can already be utilized accordingly.

In this paper we present a variant of the Contour-Buildup algorithm, specially optimized to scale well with highly parallel processing hardware. Transferring the molecule’s atom data to the graphics card in every frame and performing all required calculations for the final image of the SES on the GPU enables interactive visualization of large molecules with up to ten thousand atoms for arbitrarily long trajectories at interactive frame rates on consumer graphics cards. The required graphical primitives are calculated using CUDA, and the final image is rendered by performing point-based ray casting of polynomial surface patches entirely on the GPU, without the need for any further data transfer. Our results section shows interactive rendering performance for large data sets as well as a good scaling behavior.

1.1 Molecular Surface Definition

There are several possible definitions of a molecular surface. The Van-der-Waals (VdW) surface is probably the most simple definition. Following the fundamental idea that each atom is modeled as a force field centered around the atom’s position, an effective radius can be specified—the VdW radius—and the atoms can be depicted as spheres with this radius. The VdW surface is also known as the space-filling model. When analyzing molecules interacting with each other or with a surrounding medium, the drawback of the VdW surface becomes apparent, as it is only defined on the molecule’s atoms and does not take any correlations with other molecules or atoms into account. That is, the VdW surface only shows the general shape of the molecule, but does not model the actual accessible region of the molecule with respect to the surrounding solvent. For example, it is not possible to reliably identify solvent channels using this representation.

Richards [23] thus defined the Solvent Accessible Surface (SAS) in 1977. A spherical probe, representing the possible interaction partners for the molecule to be visualized, rolls over the VdW surface, tracing out the SAS with its center (cf. Fig. 2). The SAS thus represents a surface which is accessible for solvent atoms of a certain type. Minor gaps and crevices not reachable for solvent atoms are closed in the SAS. This makes the SAS superior to the VdW surface in cases where solvent interactions are of interest, which is a common issue when analyzing Molecular Dynamics (MD) simulation trajectories. The apparent inflation of the SAS compared to the VdW surface, however, makes it difficult to perceive the positions reachable for solvent in relation of the internal structure of the molecule.

Considering this aspect, Richards [23] introduced the Smooth Molecular Surface that is constituted by a set of spherical patches lying on the surface of either an atom or a probe at a fixed position, as well as toroidal patches traced out by a probe rolling over a pair of atoms. Another definition of this surface was given by Greer and Bush [10], in which the surface is the topological boundary of the union of all possible probes that do not intersect with the atoms of the molecule. From this definition the name for the surface was derived: Solvent Excluded Surface (SES). A rendering of the SES for a sample protein is given in Figure 3.

To construct the SES, a spherical probe is rolled over the VdW surface, similar to the construction of the SAS. However, instead of the probe’s center, the surface of the probe defines the SES as illustrated in Figure 2. The resulting geometrical primitives are the same as defined by Richards:

- **Spherical patches** occur when the probe is rolling over the surface of a single atom and has no contact with any other sphere (two degrees of freedom). All areas of the atom surface which can be in contact with the probe surface will be part of the SES. These convex spherical patches are bounded by small circle arcs on the surface of the atom.

- **Toroidal patches** are formed when the probe has contact to two atoms and rotates around the axis connecting the atom centers (one degree of freedom). The probe traces out a torus whose internal surface area belongs to the SES. The contact points of the rolling probe and the two atoms form small circle arcs on the atom surfaces bounding the saddle-shaped toroidal patch.

- **Spherical triangles** are generated by the probe surface when the probe is simultaneously in contact with three or more atoms. The probe cannot roll any further without losing contact to at least one of the atoms (zero degrees of freedom). The negative of the probe’s surfaces form concave spherical triangles.

The SES can suffer from undesirable self-intersections. These so-called singularities arise either if the surface of a primitive intersects itself or if a primitive is intersected by another. The first case occurs at toroidal patches if the distance between the torus center and the center of the torus tube is smaller than the tube radius, resulting in a spindle torus. This case can be identified by a single comparison of the two radii and can be easily removed from the final image. The second case occurs if a spherical triangle is intersected by another spherical triangle as observable in Figure 4. Since these triangles are defined on the surface of probe spheres, this singularity can be removed by testing against stored, neighboring probe positions and removing all parts of the spherical triangle which are inside another probe sphere.
The SES closes small gaps and clefts not reachable for the solvent atoms, like the SAS does, but also retains the shape of the VdW surface, which makes this surface especially suitable for applications such as docking. The SES combines the advantages of the VdW surface and the SAS, since it shows the general shape of the molecule as well as the accessible surface with respect solvent molecules of a certain radius. The closed surface, which hides atoms that are not accessible, always creates a clear representation, while the furrowed VdW surface may be confusing for larger molecules. The SES is especially useful for investigating dynamic phenomena in MD trajectories such as substrate channel formation. Both the SAS and the SES of a very small molecule are shown in Figure 1.

2 Related Work

Richards defined the SES in 1977 [23] as already stated in section 1.1, which was later named by Greer and Bush [10]. Connolly [4] presented the first algorithm for computing the SES analytically, which is therefore also known as Connolly Surface. His work was later improved by Perrot et al. [20]. Varshney et al. [34] proposed a parallelized construction scheme based on the computation of an approximate Voronoi diagram. The calculation of the SES based on the reduced surface was presented by Sanner et al. [28], and shortly after its original publication it was extended to partial updates for dynamic data [27]. Krone et al. [17] combined the reduced surface algorithm with GPU ray casting to visualize the SES entirely on the GPU. Edelsbrunner and Mücke [8] evaluated the SES introducing the $\alpha$-shape which was recently expanded to the $\beta$-shape by Ryu et al. [26]. Totrov and Abagyan [33] proposed the Contour-Buildup algorithm, which was parallelized by Lindow et al. [18] for multi-core CPUs. Additionally, there are further publications on efficiently calculating triangulations of the SES, such as [5, 35, 25]. A variety of visualization tools based on these techniques exists, allowing for the exploration of simulation data (e.g. VMD [15], Pymol [7], and Chimera [21]).

Through the computing power and programming flexibility of modern graphics hardware, the geometric primitives of the SES, which are polynomial surface patches, can be directly rendered by utilizing GPU-based ray casting. This method was originally introduced by Gumhold [12] and Klein et al. [16] for quadratic surfaces. Various improvements have been published for different types of surfaces, e.g. composed surfaces [22, 11] or higher order polynomial surfaces [19, 30, 6]. Intersecting the viewing ray with higher order surfaces can be solved differently. Toledo and Levy [32] use the iterative Newton-Raphson algorithm. Singh and Narayanan [29] and Krone et al. [17] apply the analytic stabilized Ferrari-Lagrange method presented by Herbison-Evans [14]. Improving rendering performance and visual quality is the focus of further work, featuring specially tailored visualization programs, such as TexMol [2], BioBrowser [13], and QuteMol [31].

3 Algorithm

As stated in the introduction, we present a variant of the Contour-Buildup algorithm, originally presented by Totrov and Abagyan [33]. Our modification of the Contour-Buildup is specifically tailored to run on massive parallel computing architectures, such as modern GPUs. Therefore, we will first give a short summary of the original algorithm and then describe our changes.

3.1 Original Contour-Buildup

The general idea of the Contour-Buildup is to compute the contours of the SAS and derive the SES from them. Generally speaking, the intersection of two spheres is described by a small circle on the spheres’ surfaces. If more than two spheres intersect each other, the small circles may likewise intersect each other, thereby being partially cut into circle arcs. These remaining arcs of the small circles lie outside all other intersecting spheres. The contour of a set of intersecting spheres is defined as the union of all of these small circle arcs. The SES can be derived from this contour of the SAS as follows. At the two endpoints of a small circle arc, the rolling probe is in a fixed position, defining a spherical triangle. The arc itself describes the path of the probe while it traces out a toroidal patch. The third graphical primitives mentioned in section 1.1, spherical patches, are given by the spheres of the atoms of the VdW surfaces. Figure 1 shows the contour of a small molecule and the corresponding SES patches.

The Contour-Buildup algorithm allows the computation of the contour of the SAS efficiently. To obtain the SAS from a set of atoms, the radius $r_a$ of each atom $a_i$ is expanded by the VdW radius of the probe $r_p$. Each expanded atom $a'_i$ is tested for intersections with all neighboring atoms $a'_j$ and, in case of an intersection, the small circle is computed. Each newly created small circle is subsequently intersected with all previously computed small circles. If two small circles are not intersecting each other, two cases can occur, which must be dealt with accordingly:

1. One of the small circles completely cuts off the other, the cut circle must be removed.
2. The two small circles completely cover atom \(a'_i\), which thus has no surface and can be omitted.

If the two small circles are completely outside each other, no further treatment is required. If the small circles intersect each other, the two intersection points are computed and the resulting arc on the new small circle is tested against all existing arcs. An existing arc can either be cut off completely by the new arc, shortened by the new arc, or cut into two smaller arcs. The equations to evaluate which of these cases apply are given in [33].

### 3.2 Parallel Contour-Buildup

To fully utilize the enormous parallel computation power of modern GPUs, an algorithm needs to be divided into rather small tasks. Therefore, we modified the original algorithm as follows. Rather than executing the above described computations per atom in parallel for all atoms, as Lindow et al. [18] proposed, our changes enable parallelization over all neighbors of all atoms. Consequently, we do not compute the whole contour of one atom per thread, but only the remaining arcs of each small circle for each atom. In doing so, we not only increase the number of concurrent computations, but also reduce the computational load per thread, which is especially beneficial on the GPU, where each thread has much more limited resources than a CPU thread.

The first step of our algorithm is to find all neighbors for each atom and compute the corresponding small circles. Next, all unnecessary (i.e. cut off) small circles are removed based on the criteria given above. Atoms that are completely covered by two small circles are also located in this stage and excluded from further computations. Afterwards, the central step of the algorithm is executed: All small circles of all atoms are tested in parallel for intersections with the small circles of all other neighboring atoms. The result of this computation is a list of all arcs remaining from each small circle and contributing to the contour of the SAS. As with the original algorithm, the geometric primitives of the SES (spherical triangle and toroidal patches) can be derived from these contour arcs.

Note that in the original algorithm, each arc will be computed for both atoms that share the corresponding small circle. This is necessary to guarantee the correctness of the resulting contour for all atoms. However, in our approach, we can omit these redundant computations. As the atoms have an implicit order we can ignore small circles between an atom and its neighboring atoms with smaller indices (i.e. \(a'_i\) and \(a'_j\) with \(j < i\)) since any resulting arcs will be found on the identical small circle when the atom indices are reversed (i.e. \(a'_i\) and \(a'_j\), again with \(j < i\)). Each small circle is consequently processed only once. Every endpoint is shared by three arcs, which would also lead to multiple output of the same point (i.e. probe position). Therefore, a similar optimization was chosen for the arc endpoints to write each endpoint only once.

Just like the original Contour-Buildup [33], our modified parallel algorithm can also be used to compute only a certain part of the SES. A common example of where this is useful is the visualization treatment is required. If the small circles intersect each other, the two intersection points are computed and the resulting arc on the new small circle is tested against all existing arcs. An existing arc can either be cut off completely by the new arc, shortened by the new arc, or cut into two smaller arcs. The equations to evaluate which of these cases apply are given in [33].

4 Implementation

In this section, we detail our CUDA implementation of the parallelized Contour-Buildup algorithm given in Section 3. Figure 5 shows the processing steps of our method.

First, we compute the neighborhood for each atom \(a_i\). To this end, we use the spatial hashing approach given in the Particles Demo from the NVIDIA GPU Computing SDK [9]. This implementation is a fast, grid-based spatial subdivision which runs in parallel using CUDA (first three kernels, top row in Fig. 5). The atom positions and radii needed in this step are stored in an OpenGL vertex buffer object (VBO). By doing so, the atom data is uploaded to the GPU memory and can be accessed from a CUDA kernel as well as used for fast rendering, e.g. using GPU-accelerated ray casting. Please note that this is the only stage of our computational pipeline where data is transferred from the CPU to the GPU. Using the spatial subdivision, all neighboring atoms \(a_j\) of each atom \(a_i\) are extracted and stored in a lookup table in parallel for all atoms. The small circles are also computed and stored to a lookup table at this point.

A further kernel removes all small circles which are cut off by other small circles and marks atoms which are completely covered by two small circles for exclusion from further processing. This kernel is executed in parallel for all neighbors of all atoms.

After the input data has been prepared by these calculations, the main CUDA kernel to compute the contour arcs is started (Fig. 5, right kernel in middle row). Each small circle of each atom is intersected with all other small circles of the corresponding atom and the remaining arcs, if any, are extracted. The result of this stage is the arcs of the contour. However, for the rendering, we only need to know the endpoints of the arcs (i.e. the probe positions defining spherical triangles) and which small circles contribute to the SES and, therefore, define toroidal patches.

As mentioned above, each arc endpoint is shared by three arcs. That is, each arc endpoint lies on small circle \(sc_{ij}\) on the surface of an atom \(a_j\) arising from the intersection of atom \(a_i\) and \(a_j\), which is cut by a third atom \(a_k\). The endpoint is only written to the list of arc endpoints if \(i < j < k\). A small circle \(sc_{ij}\) is only marked visible if at least one of its arcs remains in the result set.

The elements required for the SES are now derived from these results. The required parameters for the toroidal patches and spherical triangles are stored in a set of VBOs (bottom row in Fig. 5). These are then used for point-based GPU ray casting, implemented in GLSL shaders, to achieve fast rendering and high visual quality. To determine the number of toroidal patches and spherical triangles, we use the parallel prefix sum provided by the CUDA Data Parallel Primitives library [1]. The results from the prefix sum are used to compact the probe and torus positions when writing them and the additional parameters required for the ray casting to the corresponding vertex buffer objects. This is done using a further CUDA kernel which is, again, executed in parallel for all neighbors of all atoms.
The last step in our computation pipeline is the singularity handling for the spherical triangles. As explained in Section 1.1, the SES can suffer from undesired self-intersections when a spherical triangle is intersected by another probe in a fixed position. To handle this issue correctly, we again use a neighborhood search based on spatial hashing. All probe positions are sorted into a grid that is then used to find all intersecting probes for each probe. The extracted probe positions are written to a pixel buffer object (PBO). The data stored in the PBO is then transferred to a 2D lookup texture to make the probe positions available in the GLSL shader performing the ray casting. The detour through the PBO is unavoidable, since CUDA kernels cannot write directly to textures. However, the data transfer can be executed rapidly since the PBO is already stored in the GPU, and no costly data transfer between CPU and GPU memory is needed.

5 Results

We tested our implementation on an Intel Core i7-920 CPU (4 × 2.66 GHz) with 6 GB RAM and an NVIDIA GeForce GTX580 GPU (1.5 GB VRAM) at a resolution of 1024 × 768 pixels. The performance measurements are given in Table 1. We used various data sets from real-world Molecular Dynamics simulations (denoted as MDSim 1 – 4 in Table 1) from our project partners as well as freely available data sets from the Protein Data Bank [3]. The PDB data sets are provided for better comparability with other methods. Figure 6 shows the four largest data sets used for testing in different coloring modes. The smallest simulation data set (MDSim 1) is depicted in Figure 3.

Table 1: Performance measurements for our CUDA implementation of the Contour-Buildup algorithm. CB denotes the execution time of the Contour-Buildup algorithm (including data transfer), while VBO+SH denotes the time for writing VBOs for rendering and the handling of singularities (in milliseconds). The column labeled FPS states the overall frame rates in frames per second (i.e. data transfer, Contour-Buildup computation and rendering of the SES). (+) marks data sets obtained from the PDB.

<table>
<thead>
<tr>
<th>Data set</th>
<th>#Atoms</th>
<th>CB (ms)</th>
<th>VBO+SH (ms)</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1OGZ*</td>
<td>∼1,000</td>
<td>9.5</td>
<td>2.1</td>
<td>20</td>
</tr>
<tr>
<td>1VIS*</td>
<td>∼2,500</td>
<td>14.1</td>
<td>2.6</td>
<td>50</td>
</tr>
<tr>
<td>MDSim 1</td>
<td>∼4,000</td>
<td>22.7</td>
<td>3.6</td>
<td>33</td>
</tr>
<tr>
<td>MDSim 2</td>
<td>∼4,800</td>
<td>19.6</td>
<td>5.4</td>
<td>36</td>
</tr>
<tr>
<td>MDSim 3</td>
<td>∼6,600</td>
<td>31.2</td>
<td>4.5</td>
<td>24</td>
</tr>
<tr>
<td>MDSim 4</td>
<td>∼8,000</td>
<td>35.7</td>
<td>6.0</td>
<td>21</td>
</tr>
<tr>
<td>1AF6*</td>
<td>∼10,000</td>
<td>36.1</td>
<td>7.3</td>
<td>20</td>
</tr>
</tbody>
</table>

Please note that the frame rates given in the last column of Table 1 include the data transfer and the complete re-computation of the contour for each rendered frame, regardless if the data is static or dynamic. That is, our implementation always behaves as if using fully dynamic data. The frame rates given in Table 1 show that our implementation is able to maintain the SES for fully dynamic data sets of up to ten thousand atoms at interactive frame rates. The GPU ray casting used for rendering, which was also applied by Krone et al. [17] and Lindow et al. [18] for the SES, produces high quality images. An additional benefit of the ray casting is that all vertex positions and attributes can be written to buffer objects directly on the GPU via CUDA. Please note that the time measurements for the Contour-Buildup in Table 1 (column CB) already include the data transfer. The computation times for the buffer objects used for rendering and the singularity handling are given separately (column VBO+SH).

As mentioned in Section 2, Lindow et al. [18] showed that the Contour-Buildup algorithm can be parallelized. By using OpenMP on the outer loop over all atoms, they managed to approximately gain a sixfold speedup on a CPU with eight physical cores, compared to the single-core execution. Their CPU-parallelized implementation scales linearly with the number of atoms. Our CUDA implementation of the Contour-Buildup in contrast exhibits sub-linear scaling, especially for the larger data sets. In our opinion, our modification of the Contour-Buildup algorithm is better suited to exploit massively parallel architectures due to the more fine-grain working tasks. For the maltoporin data set (PDB-ID: 1AF6) Lindow et al. measured an update time of 73 ms (2 Intel Xeon E5540 8 × 2.53 GHz) while our implementation takes only 43 ms. For very small data sets like the isomerase data set (PDB-ID: 1OGZ) our implementation reaches only slightly faster update times (16.7 ms versus 18 ms). This can be explained by the overhead introduced by using CUDA, which only provides a speed advantage for large problems.

Notably, MDSim 2 exhibits higher frame rates than MDSim 1, even though it is 20% larger. This can be explained by the three-dimensional structure of the data: While the other data sets are simulations of proteins, which have a relatively spherical shape and, therefore, a high number of neighbors per atom, MDSim 2 is a simulation of a elongated hydrogel, which has a filamentary structure and only a small number of neighbors per atom (cf. Figure 6, left). A similar effect explains the relatively high time for VBO+SH for this data set. As the ratio of the number of surface elements to the number of atoms is relatively high compared to the other data sets, the cases that need to be tested for the singularity handling are also quite high. In our tests, our method always reached interactive frame rates, despite processing the SES in every frame and performing GPU-based ray casting of the surface elements.
6 Conclusion
We have presented a version of the Contour-Buildup algorithm to evaluate the SES, originally presented by Totrov and Abagyan [33], which is optimized for massive parallel processing hardware, especially modern graphics cards. The required calculations were divided into fine-grain tasks, allowing for full utilization of the graphics hardware. The algorithm was implemented using the GPGPU programming language CUDA. We re-evaluate the SES in every frame; therefore, our algorithm is intrinsically suited for large time-dependent data sets with arbitrarily long trajectories and in-situ visualization. Our method achieves highly interactive rendering performance for data sets up to ten thousand atoms on standard consumer hardware and displays sub-linear scaling behavior with respect to the data set size. Thorough optimization of our CUDA kernels as well as our GPU ray casting shaders by algorithmic refinements and engineering efforts could further improve the performance of our implementation. However, we believe that our parallel algorithm is optimally suited for current and future massive parallel hardware architectures.

Acknowledgements
We thank our project partners from the Institute of Technical Biochemistry, University of Stuttgart for providing the simulation data sets. This work is partially funded by German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) as part of projects D.3 and D.4 of the Collaborative Research Centre (SFB) 716.

References