Computers & Graphics 54 (2016) 113–120

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Issue on CAD/Graphics 2015

High quality illustrative effects for molecular rendering

P. Hermosilla [a,b,*], V. Guallar [b], A. Vinacua [a], P.P. Vázquez [a]

[a] Universitat Politècnica de Catalunya, Spain
[b] Barcelona Supercomputing Center, Spain

ARTICLE INFO

Article history:
Received 12 April 2015
Received in revised form
12 July 2015
Accepted 13 July 2015
Available online 26 July 2015

Keywords:
Molecular visualization
Ambient occlusion
Halos

ABSTRACT

All-atom simulations are crucial in biotechnology. In Pharmacology, for example, molecular knowledge of protein-drug interactions is essential in the understanding of certain pathologies and in the development of improved drugs. To achieve this detailed information, fast and enhanced molecular visualization is critical. Moreover, hardware and software developments quickly deliver extensive data, providing intermediate results that can be analyzed by scientists in order to interact with the simulation process and direct it to a more promising configuration. In this paper we present a GPU-friendly data structure for real-time illustrative visualization of all-atom simulations. Our system generates both ambient occlusion and halos using an occupancy pyramid that needs no precalculation and that is updated on the fly during simulation, allowing the real time rendering of simulation results at sustained high framerates.

1. Introduction

Molecular visualization is of key importance in elucidating the atomic mechanism for protein-ligand recognition, a crucial process in pharmacology drug design and enzymatic catalysis. The specificity and potency of a drug, for example, will depend on its molecular interactions at the bound complex. Moreover, drug efficacy might also depend on its delivery to the target active site—i.e. fluctuations in the entrance pathway/channel (for instance from mutations, etc.) can decrease or abolish its action. These aspects can be clearly understood when visualizing the continuous dynamical protein-ligand recognition, typically a microsecond timescale process. Recent developments in software and hardware, such as the special purpose Anton machine [1], allow today biophysical simulations at such timescales if large dedicated computational time is available. Moreover, breakthrough technological advances have reduced this computation time from weeks/days to hours/minutes [2], introducing processes that can be interactively inspected and directed. In this interactive new scenario, enhanced molecular visualization is essential. Inspection may be used both to understand the recognition process and, more importantly, to change parameters of the simulation process in an interactive and iterative manner. These changes might improve the exploration (sampling) process but also, for example, introduce modifications *on the fly* to drug candidates to better design more effective inhibitors. These substantial advances in the speed of simulation will continue to evolve. We are working thus, on the next generation of simulation software that will bring a tight control of the process to the desktops of the researchers (see Fig. 1). In this paper we concentrate on the visualization cues that facilitate the comprehension of the simulation.

Illustrative visualization techniques help researchers gain insights about the structure of large proteins [3]. Some of these techniques include abstraction [4], salient region enhancement [5], halos or ambient occlusion [6,7]. Popular molecular visualization packages such as Avogadro [8] or VMD [9] include some of these techniques in different ways. However, most previous approaches do not support illustrative effects in real-time for molecules in constant motion. This makes our scenario different from other approaches in that we need to recompute the illustrative visualization motifs for each frame. Thus, our requirements are:

- *Facilitate molecule understanding*: We achieve this with high quality ambient occlusion both for far views and closeups.
- *Illustration of the ligand position and trajectory*: This provides better comprehension of the current simulation state. This is solved by rendering halos around the drug, and along its covered path.
- *Realtime rendering with no pre-process*: Since changing atom positions are generated on the fly by the simulation software, we need to render them directly from the output of the simulation. This disallows the use of precomputed information (as in Crassin et al. [10]).
- *Different visualization modes*: The space filling mode and the ball and sticks mode are the most used to inspect simulations, so we set out to support them both, but will not deal here with other techniques such as surfaces [11–14].

Our main contributions are:

- A new data structure to store an occupancy pyramid suitable for high quality ambient occlusion and halos.

* Corresponding author.
E-mail address: pedro.hermosilla@cgstarad.com (P. Hermosilla).

http://dx.doi.org/10.1016/j.cag.2015.07.017
0097-8493/

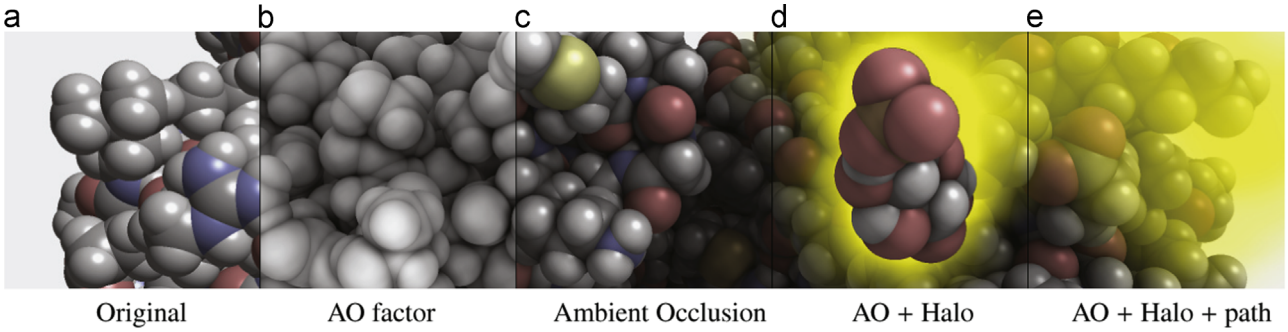

Fig. 1. On the fly illustrative visualization of protein-ligand simulations. The different stages of our algorithm are shown in the figure. The initial visualization uses impostors (left). To this rendering mode, we add object space ambient occlusion (factors shown in (b)), that is computed using an occupancy pyramid that is generated on the fly in the compute shader. Besides ambient occlusion, (result shown in (c)), we also use an occupancy pyramid to generate halos on a ligand (d) and to generate the halo path that illustrates the movement of the ligand along the simulation (e).

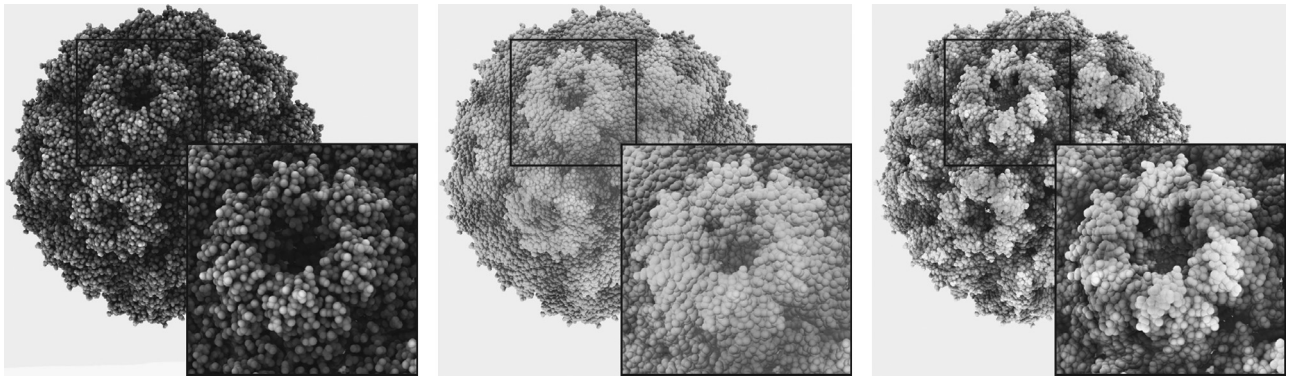

Fig. 2. Comparison of our ambient occlusion algorithm—left—with the algorithm proposed by Grottel et al. [7,15] with a resolution of 8^3—middle—and 16^3—right. Note how the shape is better perceived using our approach thanks to our high quality ambient occlusion.

- An efficient algorithm to fill the occupancy data at each frame using compute shaders.
- A fast method to render object space temporal halos.

Since the positions of atoms change continuously, our algorithm needs to update the occupancy hierarchy at each frame. We do so with an efficient voxel-atom overlapping test that approximates the intersection using atomic operations. Moreover, our rendering algorithm produces higher visual quality than previous approaches.

The main advantages of our approach are: (i) *Multiresolution*: The hierarchical nature of our occupancy structure allows for low and high frequency ambient occlusion shadows. (ii) *No precomputed data structures* are used, so we can support dynamic simulations. (iii) With a small modification of the initial algorithm, we are able to use a *unified data structure* for both ambient occlusion and halos, which facilitates its incorporation to molecular rendering systems.

The rest of the paper is organized as follows: In Section 2 we analyze the related work. We outline our technique in Section 3. We introduce our data structure and its construction process in Section 4. Sections 5 and 6 deal with the generation of AO and halos. We conclude the paper in Section 7 with a discussion of the results.

2. Previous work

Molecular graphics has received a lot of attention lately in visualization [16,12–14,17]. Being a broad field, it has been addressed under many different perspectives. New visualization motifs have been created or adapted to molecules in order to provide better insights on the different elements presented [11,4,17–19]. Other researchers focus on solving the horsepower problem that massive models imply when real-time is required [20,21]. A common problem is also the implementation of illustrative visualization techniques such as edge cueing, hatching, ridge and valley emphasizing [6,22,23,5]. Other systems adapt well-known shading methods such as ambient occlusion [6,7] for the concrete case of molecules or particle systems.

We propose a state of the art report [24] for the interested reader, that introduces several visualization techniques.

Well established molecular rendering packages include VMD [9], Avogadro [8], and QuteMol [6]. However, while some of these packages provide high quality rendering, none of them is able to cope simultaneously with dynamic trajectories with illustrative visualizations and support for the interactive exploration of the simulation. Since the trajectories of the molecules are computed in real-time, this averts the precomputation of data structures that might accelerate ambient occlusion or halo rendering.

Illustrative visualization and especially the two motifs we have chosen (ambient occlusion and halos) are known to facilitate understanding of shape [25]. Tarini et al. [6] proposed a system that simulates ambient occlusion in real time. They precompute AO for a molecule by calculating shadow maps in many directions on the GPU. Krone et al. [19] perform depth darkening [26] in a real-time ray casting of molecules using a surface-based model. Later, they improve their method by computing AO in object space for particle models [7]. Our approach is similar to the later in that we compute AO in object space, but differs in how we build our data structure. Since we build the occupancy pyramid in the compute shader, we can afford larger grids and compute a better approximation of the occupancy of each cell. Moreover, our approach can handle high frequency shadows due to the hierarchical nature of our data structure— Fig. 2. While this paper was under review, Staib et al. [27] presented an extension of their algorithm [7] to calculate

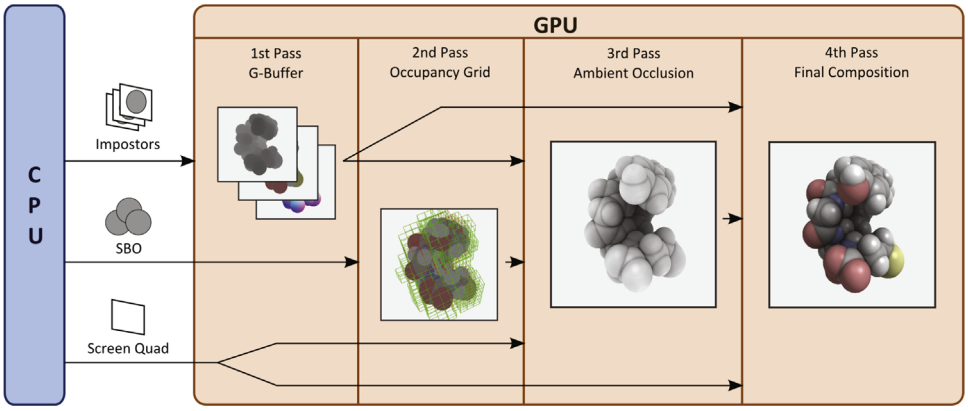

Fig. 3. The steps of our basic algorithm: the depths, colors and normals are computed in the first step; independently, the voxelization used to compute the occupancies is computed by the compute shader (only the non-empty voxels are shown); next, from the depths and occupancies, we compute darkening coefficients and finally we composite these intermediate results to obtain the rendered molecule.

the ambient occlusion factor and transparencies in real-time for molecular dynamics simulations. Their approach is similar to the work presented in this paper; both use an occupancy pyramid stored as a 3D texture with different mipmap levels, but the implementations differ. Our implementation computes the occupancy value for the voxels of all levels in a single pass using the compute shader and the occupancy approximation increases continuously with the penetration of an atom into a voxel, allowing a continuous animation without popping artifacts. Besides, our algorithm can also work with another shapes—as cylinders in the ball-and-stick and licore rendering modes—and generate both local and temporal halos of the ligand position and trajectory in real-time, which is not supported by other methods.

3. Overview

The basic version of our algorithm proceeds in four steps (see Fig. 3):

1. Render the molecule for the current frame. We render a ball-and-stick or a space filling representation of the molecule using impostors. The program renders a quad for each atom and for each bond, and the shaders generate respectively the geometry and update textures that store the depth, the color and the normal at each fragment.
2. The occupancy data structure is filled-in by compute shaders that process each atom and bond.
3. To compute the ambient occlusion dimming coefficient at each position, we now send a quad covering the whole viewport, and evaluate the AO in a fragment shader.
4. Finally, we composite the result image, using the previous results and incorporating lighting computations using the normal stored for each fragment in step 1.

These steps are discussed in more detail in later sections. Specifically, Section 4 details step 2, and Section 5 details step 3, the two main building blocks. Later, in Section 6, we discuss how modifications of these steps yield other illustrative effects, using the same data structure and approach.

4. Occupancy pyramid

The occupancy pyramid is computed by first uniformly subdividing a bounding box of the molecule(s) to render. Our implementation sets this voxelization at 128^3, which we have found to be sufficient even for the most complicated molecules. To enhance the computations, we also build coarser resolutions of the same grid, of 64^3, 32^3, 16^3 and 8^3 voxels. These regular grids are stored in 3D textures in a straightforward fashion.

In order to fill-in this occupancy pyramid, we resort to two compute shaders, one for atoms and one for bonds. For each level of the voxel grid, the compute shader calculates –using the bounding box of the element (atom or bond)– the minimum and maximum voxel positions that the element possibly intersects. Then, for each voxel, an intersection test is computed to determine whether the element really intersects the voxel or not. Where the intersection test succeeds, the compute shader increments the occupancy value with an approximation of the incident volume. To compute the overlap of an atom—given by a center and a radius—with a voxel, we find the point in the voxel that is closest to the center, and compute the distance r between them. If that distance r is less than the radius r_a of the atom, we increase the occupancy of the voxel by $(r_a - r)/D$, where D is the length of the diagonal of a voxel (see Fig. 4, right). Bonds between atoms are modeled as cylinders, given by a base center, axis vector, height and radius. In this case, we compute the point on the cylinder that is closest to the center of the voxel, and compare the signed distance between them with $D/2$ (see Fig. 4, left). If the distance is smaller than $D/2$, we increment the occupancy of the voxel by $(D/2 - r)/D$. These ratios of the *penetration depth* with the size of the circumscribing sphere produce a very rough estimation of the volume of the voxel that is occupied, and are chosen as a compromise, as they are very quick to compute, and produce visually acceptable results (see Fig. 6 for an example comparison with a computation-intensive AO computed using path tracing).

The textures that store the occupancy pyramid are floating-point textures to afford interpolation of the occupancy values during rendering. However, in order to be able to use the mechanisms in GLSL for avoiding collisions in their update, they are accessed as integer textures using `imageAtomicCompSwap()`—as in [28]—, converting values from one format to the other during read and write operations via `floatBitsToUint()` and `uintBitsToFloat()`. Since the occupancies computed are only approximate, this precaution may be dismissed in some settings, but on fast hardware the number of collisions will tend to increase, and not using an exclusion mechanism will result in slight but detectable flicker in the darkening factors computed for ambient occlusion, when the positions of atoms or the camera change.

In order to reduce the number of collisions in the updating process, we shuffle the elements in the vertex buffer, ensuring that elements that are close in the scene are far away in the buffer. This simple modification reduces the number of collisions for

molecules with a high number of atoms, increasing the rendering speed. For small molecules, however, the performance may be reduced, because all the elements can be processed in parallel (obviously, this depends on the number of execution threads of the GPU). Apart from that, this solution has another potential drawback: the coherence in the texture access for each compute shader workgroup is reduced, thus increasing the texture cache misses. In our implementation shuffling is only used when the molecule contains 300K atoms or more (for the space-filling representation) and always for the balls-and-sticks representation. Table 1 shows framerates for different test molecules with and without the shuffling step.

With this algorithm, occupancy pyramids can be refreshed at each frame, with excellent performance, as detailed in Section 6, allowing for the computation of dynamic AO and other effects in real-time, as the results of the simulation are received by the rendering client.

5. Object-space ambient occlusion

Ambient occlusion is an algorithm that approximates the global illumination by measuring the amount of light that may reach each surface point. We introduce it briefly, for completeness (the interested reader may find more complete information in [29]). The *irradiance* E that reaches a point on a surface with normal n_p can be defined as [30]

$$E(p) = \int_{\Omega} n_p \cdot \omega L(\omega)\, d\omega$$

where $L(\omega)$ is the irradiance arriving from direction ω and Ω is the set of directions above the surface. In order to compute $E(p)$, the domain Ω is discretized into k sectors $\overline{\omega}_i$ subtending a solid angle $|\overline{\omega}_i|$. Sampling one direction ω_i per sector, and considering only direct lighting, yields the well known ambient occlusion equation

$$E(p) = \frac{1}{4\pi} \sum_{i=1}^{k} n_p \cdot \omega_i O(\omega_i)$$

where $O(\omega)$ represents the occlusion function that takes value 0 if the point is visible from the environment, and value 1 if it cannot be reached. This equation can be considered as an approximation of the whole rendering equation [30].

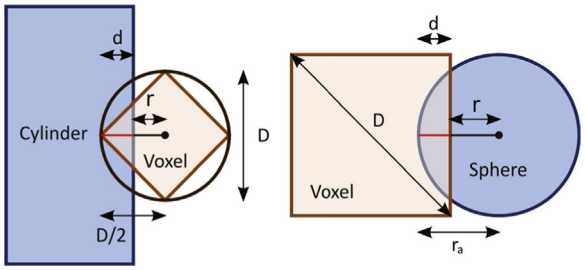

Fig. 4. Evaluation of the occupancy of a voxel. The portion of a voxel in the occupancy pyramid that is occupied by a bond (left) or atom (right) is approximated as a ratio of a penetration distance d and the diagonal of the voxel D.

There are several approximations of ambient occlusion in the literature for scenes composed of particles [31], but we are more concerned with object-space approximations [32,10,7] since they commonly produce results that are coherent with the real 3D arrangement of geometry.

Like Crassin et al. [10], we analyze the occlusion by sampling along a set of cones that cover the hemisphere above the point to shade. In contrast to their approach, we use a regular occupancy pyramid computed on the fly instead of a pre-computed sparse octree that is updated with the dynamic data. We have also developed a GPU-accelerated overlap approximation for the shapes we use (spheres and cylinders). Our data structure better fits the architecture of the GPU and thus permits high framerates. In contrast to the approach by Grottel et al. [7,15], we do not sample a single time at the surface of the molecule, but instead take advantage of the hierarchical structure of our grid to perform queries along a larger distance. This generates both low and high frequency shadows leading to high quality effects that easily illustrate the geometry details such as pockets or cavities. A second advantage of our method is that we can also visualize shadows in other representations such as licorice and ball-and-stick.

In order to generate the ambient occlusion values, we render a quad that covers the whole screen. For each fragment, we recover the world coordinates by querying the depth buffer. Then, we generate a number of cones (from 3 to 9, depending on the render quality desired) to cover the hemisphere centered at the point to shade. For each cone, we compute the center and radius of a set of tangent spheres that fit inside the cone as illustrated in Fig. 5. We have tested different configurations and four spheres are enough to have a high shadowing quality. We use the center of each sphere to perform a sampling in the occupancy pyramid at the level determined by its radius—using trilinear interpolation in order to obtain a smooth result. We use this value as an approximation of the occupied volume of the sphere. Then, we accumulate the occupancy

Table 1
Performance measured in frames per second for different molecules (N)ear the camera (occupying the whole viewport), at (M)id-distance, and (F)ar from the molecule. Ambient occlusion has been computed at the highest quality (nine cones per point). "S-F" indicates usage of the space filling render mode (as, for example, in Fig. 12), while "B&S" indicates usage of the ball-and-stick render mode (see Fig. 9). Framerates have been measured without shuffling the buffer of elements (NS) and with the shuffle applied (S).

Molecule name #atoms [-Figure]			A 3.9K [- Fig. 10]		3J3A 46K [- Fig. 12]		1CWP 227K [- Fig. 13]		1K4R 545K		3IYJ 1356K [- Fig. 8]	
Vertex sorting			S	NS	S	NS	S	NS	S	NS	S	NS
S-F	N	No AO	816.52		547.88		169.74		136.66		64.72	
		AO	188.19	**201.22**	171.53	**265.45**	76.98	**89.16**	**70.83**	53.58	**34.09**	25.44
	M	No AO	1953.11		1233.95		382.59		192.29		88.68	
		AO	223.36	**241.46**	216.79	**393.55**	120.51	**130.62**	**85.38**	61.57	**38.95**	28.03
	F	No AO	3444.91		1491.38		499.96		456.49		96.05	
		AO	236.02	**257.51**	226.86	**425.65**	115.98	**143.48**	**116.42**	76.01	**42.51**	29.46
B & S	N	No AO	2255.4		1156.75		351.47		209.63		88.23	
		AO	671.81	**706.51**	**345.12**	342.05	**123.88**	92.53	**64.6**	34.36	**30.63**	14.83
	M	No AO	3104.45		1504.23		527.38		275.68		107.19	
		AO	779.95	**840.02**	**402.41**	398.52	**148.24**	105.86	**71.63**	36.21	**33.08**	15.35
	F	No AO	3705.44		1858.25		600.48		276.19		108.08	
		AO	852.78	**907.55**	**434.37**	431.88	**157.04**	109.98	**72.14**	36.41	**33.27**	15.42

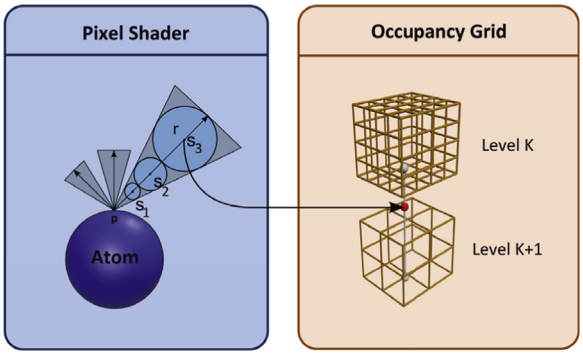

Fig. 5. In order to evaluate AO, we trace rays along the direction of a set of cones that cover the space above the shading point. For each ray, we sample a set of points $s_1, s_2, \ldots s_n$ and the ambient occlusion factor is obtained by querying the occupancy pyramid at these points. The radius of the inscribed spheres determines the LOD to use when querying the occupancy pyramid.

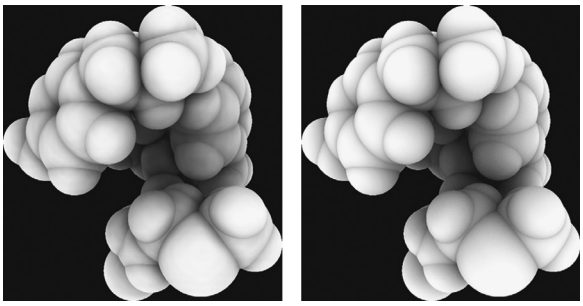

Fig. 6. Comparison of our ambient occlusion algorithm (left) with the ground truth, as computed using path tracing [30] (right). The differences are hardly visible. While our method yields slightly higher occlusions in marked creases, the shadings are smooth and coherent with the shape, offering correct depth cues and adequate high frequency detail.

along the ray using the following equation to determine the percentage of indirect light rays that arrive to the point from this set of directions:

$$ray\_occlusion += current\_sample * (1.0 - ray\_occlusion)$$

The final occlusion factor of the point is the average occlusion factor of all the cones.

Although a relatively large number of samples are made, the framerates stay above 30 fps even for large viewports (1280×720) and the largest molecules in closeup views that occupy the whole viewport.

The final composition step combines the color with the ambient occlusion and calculates the lighting using the G-buffer generated in the first step. Note the high quality of the ambient occlusion obtained, as compared to a ground truth image generated using path tracing in Fig. 6.

6. Halos

In this section we describe the algorithm we use to generate object-space halos and temporal halos.

6.1. Object-space halos

Halos can easily be rendered for the whole molecule using the same data structure. However, their utility becomes prominent when rendered around the ligand, since this helps researchers to clearly and quickly identify it and distinguish it from the protein. Therefore, we add a second copy of our data structure that stores the occupancy pyramid of the drug only. Since the drug is smaller, this data structure can be of lesser resolution. However, for simplicity, we keep it with the same dimensions as the original occupancy pyramid used in AO computations for the whole molecule.

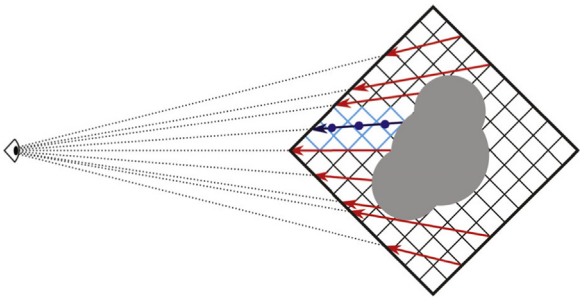

Fig. 7. Ray casting of the occupancy pyramid. The rays are generated at the back-faces of the bounding box or at the current pixel depth, and they finish at the front. The blue ray illustrates how a ray accumulates the occupancy of the voxels by sampling the grid along its path. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

In this variant of the algorithm, at the pyramid construction step, the new occupancy pyramid is also computed, but, because the number of image bindings to a shader is limited, we can only update the first three levels of this new occupancy pyramid—128^3, 64^3 and 32^3. The algorithm adds the following steps at the end of the rendering process:

- Create a mask in the stencil buffer, corresponding to the molecule we want to highlight.
- Render the bounding box of the drug in order to generate the halos.

The mask is created by rendering the atoms and bonds of the drug with a small depth offset to ensure that the proper fragments are rasterized. The objective is to make sure that the halo does not overlap the drug elements. Once the stencil is initialized, we render the bounding box of the molecule and clip away front faces. For back faces, each rasterized fragment generates a ray towards the observer. If the back face is further than the current depth, the ray starting point is at the current depth. Rays stop at the front faces of the bounding box (see Fig. 7). At each sampling position, the halo occupancy pyramid is queried using the normal 3D texture fetch with trilinear interpolation and the accumulated value is used to compute the halo factor. Notice that with this algorithm the halo is calculated using 3D information and it can be rendered even if the molecule to highlight is hidden in the scene.

6.2. Temporal halos

While halos help quickly identifying the ligand against the protein, it is often desirable to capture the evolution of the interaction between the two molecules. This is useful not only for simulations that take long or to highlight the entrance/exit drug pathways, but also when the researcher is not able to devote undivided attention to the simulation. In such cases, the emphasis on the drug may not be enough since we may not be aware of the regions around the protein that have been explored by the drug. To facilitate the easy inspection of the recent history of the simulation, we introduce temporal halos. Temporal halos show the path the drug has covered as a slowly vanishing glowing wake of the ligand, going back a chosen amount of time T.

Instead of clearing the halo occupancy pyramid at the beginning of each frame, our implementation launches a compute shader that reduces the values in the occupancy pyramid—at the

three levels we are using—according to the time elapsed between the last frame and the current one. The new occupancy is computed as follows:

$$O_f = O_i - \max\left(\frac{t - t_o}{T}, 1\right)$$

where O_i is the current occupancy of the voxel, t is the current time, t_o is the time of the last frame and T is the desired duration of the halo persistency.

7. Results and discussion

In this section we present results of the proposed algorithm, both in terms of the quality of the resulting images and the performance achieved. All these tests have been run on a Intel Core i7 PC, running at 3.5 GHz, with 16 Gb of RAM, and a GeForce 770GTX, and rendering in a 1280×720 viewport.

The algorithm we have presented here works in real-time, and for regular renderings achieves framerates of several hundred frames per second (including the computation of the occupancy pyramid at each and every frame). For close-ups where the molecule completely fills the screen, the rendering slows-down somewhat, but it is still realtime even for very large models of up to 1.3M atoms. Notice that the molecules used in the tests are among the largest encountered in pharmacological simulations, and some are much larger than the average size encountered. In Table 1 we list the framerates for five example molecules of different sizes. For each molecule we list the number of atoms it consists of, its name, when available, and the rendering speed for the whole molecule, using nine cones at each point in the calculation of the ambient occlusion. The framerates are listed at three different distances, because the algorithm is fragment-bound, and the performance depends, therefore, on the portion of the viewport occupied by the molecule. The column labeled "Near" corresponds to a camera close enough that the molecule occupies the whole viewport. The table gives the performances in two different render modes, space filling —where atoms are represented as spheres with their van der Walls radius—and ball-and-stick, where atoms are represented by smaller spheres, joined by sticks representing the bonds between atoms.

Except for the molecule 1K4R, the other molecules in Table 1 appear as Figs. 10, 12, 13 and 8, respectively. The algorithm also works for models in ball-and-stick format (see Table 1 and Fig. 9), and licorice.

Fig. 10 depicts molecule A from Table 1, with a small molecule trying to find a docking spot, highlighted with a blue halo. Fig. 11 shows a different view of this same pair of molecules, but with a temporal halo that indicates the recent positions occupied by the ligand as a gradually vanishing green wake.

Fig. 8. A render of molecule 3IYJ (last column in Table 1), consisting of 1356K atoms.

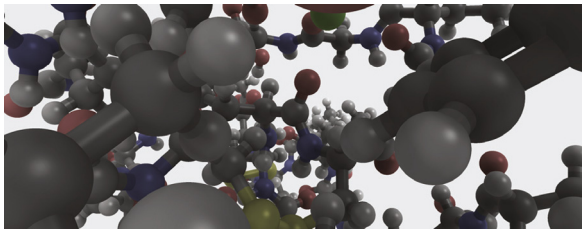

Fig. 9. Ball-and-stick rendering with our ambient occlusion.

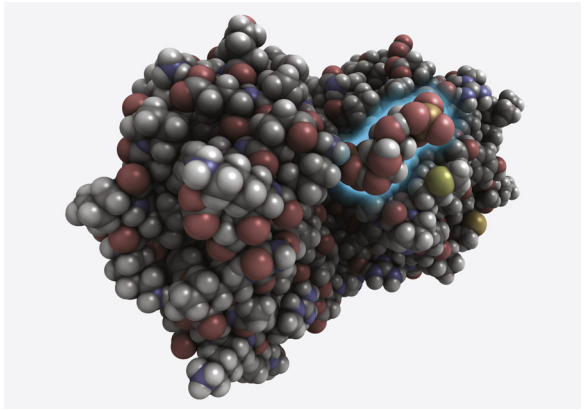

Fig. 10. Molecule A from Table 1, interacting with a small ligand, highlighted by a blue halo. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

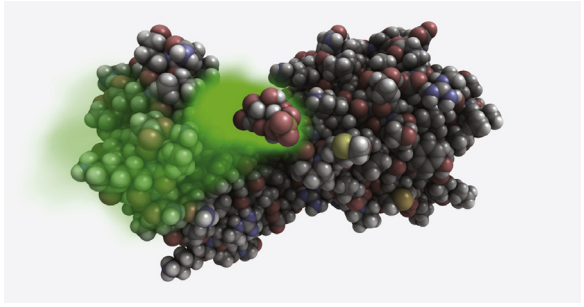

Fig. 11. The same pair of molecules as in Fig. 10, from a different viewpoint, and with temporal halos activated to show the path traversed by the ligand in reaching its present position. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Naturally, the rendering of molecules with AO and halos is more expensive than with AO alone. Table 2 shows the framerates obtained when rendering the molecules of Fig. 10 without halos, with a halo around the ligand, and with a halo around the larger molecule, both for the space filling and for the ball-and-stick render modes. In all cases, we give the performance at near, mid-distance or far positions of the camera, as we did in Table 1. The molecules commonly used in our pharmacology simulations rarely exceed 60–80K atoms. However, our method presented here works in realtime even for much larger molecules.

Finally, we offer in Table 3 a measure of the effect of changing the number of cones used for computing the ambient occlusion attenuation factor at each point. Naturally, a larger number of cones give a better estimation of accessibility, and therefore higher quality images. All molecule renders in this paper have been obtained with nine such cones, but the table shows the increases of speed that can be achieved by reducing this number; three cones give low quality, but 5 cones may be quite acceptable in many applications.

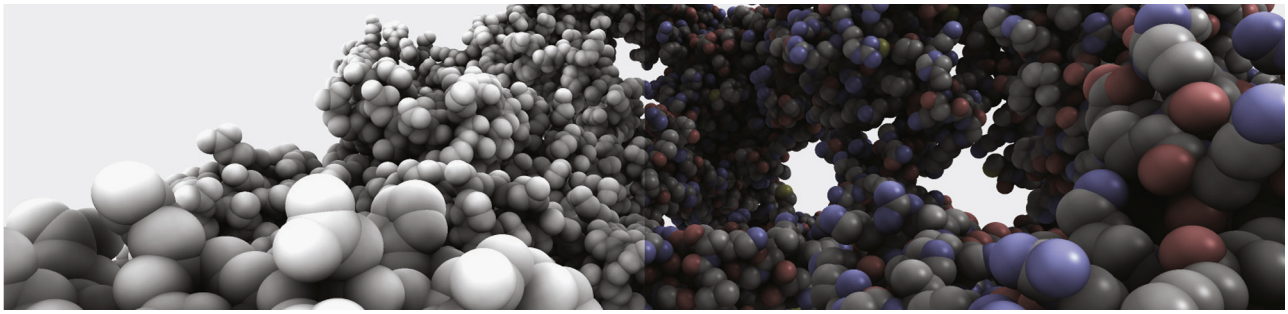

Fig. 12. A render of (a portion of) molecule 3J3A in Table 1, consisting of 46 276 atoms. Note the quality of the AO shading.

Fig. 13. A render of molecule 1CWP in Table 1, consisting of 227 040 atoms.

In order to assess the utility of the illustrative motifs for the problem we are tackling, we prepared a small questionnaire that was answered anonymously by 11 domain experts, all of them working with drug simulation software. We asked them to rank in a 1…7 Likert scale (where 1 is either totally agree or totally preferred and 7 is totally disagree) their preference with respect to five questions, summarized in the following listing. The brackets indicate the contents of the pictures we showed to the users when asking the questions (in most cases, they consisted of images that are shown across this paper).

1. [*AO image*] is better to understand the rendering [than an image without AO].
2. [Image with halo around a drug] helps better understand the drug position than [the image without halo].
3. [Halos] are helpful to visualize simulations.
4. [Image with halo path] helps in understanding the path covered by the molecule.
5. The resulting image [image with halo path] is useful.

The average answers were 1.9, 1.27, 2.1, 2.9 and 2.0, respectively. This shows that the experts appreciate the quality of the rendering results and feel it is useful to better understand the simulation process.

In summary, we have presented an algorithm capable of rendering complex, dynamic molecules at interactive framerates, with high and low frequency ambient occlusion shading, halos and temporal halos computed on the fly. While other algorithms have been published to produce quality renders of complex molecules, to our knowledge, none is able to produce all these effects, at interactive rates on molecules (in space filling or ball-and-stick modes) that are changing in shape and interacting with each other.

We also carried out a small user study to determine whether the illustrative effects were suitable for molecular inspection with 11 domain experts. The users were enthusiastic with the results, and would use such kinds of effects in their common daily work.

This work represents a step in improving the perception of the molecule's shape by the user. Further improvements may include also shadows and better anti-aliasing than that provided by a simple oversampling.

Table 2
Impact of halos on rendering speed. The same molecule (A, see Fig. 10) is rendered with no halos, a halo around the ligand, and a halo around the larger molecule. Framerates are reported both for the space filling and the ball-and-stick render modes. Again ambient occlusion has been computed with nine cones per point.

		No halos	Ligand	Mol.
Space filling	Near	201.22	113.49	101.75
	Mid.	241.46	126.68	122.91
	Far	257.51	131.28	129.52
Ball-and-stick	Near	706.51	484.77	328.78
	Mid.	840.02	575.9	489.98
	Far	907.55	621.83	604.08

Table 3
Impact of the number of cones used to compute the ambient occlusion on the rendering speed in fps. The same molecule A from Fig. 10 was rendered in space filling and ball-and-stick modes using 3, 5 and 9 cones per fragment.

		9 cones	5 cones	3 cones
Space filling	Near	201.22	204.45	206.28
	Mid.	241.46	243.16	243.65
	Far	257.51	258.19	258.28
Ball-and-stick	Near	706.51	772.49	783.12
	Mid.	840.02	882.29	886.01
	Far	907.55	939.53	941.86

Acknowledgments

This work has been supported by the Projects TIN2013-47137-C2-1-P and TIN2014-52211-C2-1-R of the Spanish Ministerio de Economía y Competitividad, and the project 2014-SGR 146 from the Catalan Government.

Appendix A. Supplementary material

Supplementary data associated with this paper can be found in the online version at http://dx.doi.org/10.1016/j.cag.2015.07.017.

References

[1] Dror RO, Dirks RM, Grossman J, Xu H, Shaw DE. Biomolecular simulation: a computational microscope for molecular biology. Annu Rev Biophys 2012;41:429–52.

[2] Madadkar-Sobhani A, Guallar V. Pele web server: atomistic study of biomolecular systems at your fingertips. Nucleic Acids Res 2013;41(W1):W322–8.

[3] Goodsell D. Illustrating molecules. In: Hodges ERS, editor. The guild handbook of scientific illustration, vol. 2, 2003. p. 267–70.

[4] Van Der Zwan M., Telea A., Isenberg T. Continuous navigation of nested abstraction levels. In: Short paper proceedings of the EG/IEEE VGTC conference on visualization (EuroVis), Songdo, 2012. p. 13–17.

[5] Lawonn K, Krone M, Ertl T, Preim B. Line integral convolution for real-time illustration of molecular surface shape and salient regions. Comput Graph Forum 2014;33(3):181–90.

[6] Tarini M, Cignoni P, Montani C. Ambient occlusion and edge cueing for enhancing real time molecular visualization. IEEE Trans Vis Comput Graph 2006;12(5):1237–44.

[7] Grottel S, Krone M, Scharnowski K, Ertl T. Object-space ambient occlusion for molecular dynamics. In: 2012 IEEE Pacific visualization symposium. IEEE; 2012. p. 209–16.

[8] Hanwell MD, Curtis DE, Lonie DC, Vandermeersch T, Zurek E, Hutchison GR. Avogadro: an advanced semantic chemical editor, visualization, and analysis platform. J Cheminformatics 2012;4:17.

[9] Humphrey W, Dalke A, Schulten K. VMD: visual molecular dynamics. J Mol Graph 1996;14(1):33–8.

[10] Crassin C, Neyret F, Sainz M, Green S, Eisemann E. Interactive indirect illumination using voxel cone tracing. Comput Graph Forum 2011;30 (7):1921–30.

[11] Parulek J, Turkay C, Reuter N, Viola I. Implicit surfaces for interactive graph based cavity analysis of molecular simulations. In: IEEE symposium on biological data visualization (BioVis). IEEE, Seattle, WA, 2012. p. 115–22.

[12] Sarikaya A, Albers D, Mitchell J, Gleicher M. Visualizing validation of protein surface classifiers. Comput Graph Forum 2014;33(3):171–80.

[13] Scharnowski K, Krone M, Reina G, Kulschewski T, Pleiss J, Ertl T. Comparative visualization of molecular surfaces using deformable models. Comput Graph Forum 2014;33(3):191–200.

[14] Lindow N, Baum D, Hege H-C. Ligand excluded surface: a new type of molecular surface. IEEE Trans Vis Comput Graph 2014;20(12):2486–95.

[15] Grottel S, Krone M, Muller C, Reina G, Ertl T. Megamol—a prototyping framework for particle-based visualization. IEEE Trans Vis Comput Graph 2015;21(2):201–14.

[16] Le Muzic M, Parulek J, Stavrum AK, Viola I. Illustrative visualization of molecular reactions using omniscient intelligence and passive agents. Comput Graph Forum 2014;33(3):141–50.

[17] Parulek J, Jönsson D, Ropinski T, Bruckner S, Ynnerman A, Viola I. Continuous levels-of-detail and visual abstraction for seamless molecular visualization. Comput Graph Forum 2014;33:276–87. http://dx.doi.org/10.1111/cgf.12349.

[18] Parulek J, Ropinski T, Viola I. Seamless visual abstraction of molecular surfaces. In: Spring conference on computer graphics. ACM; Smolenice, Slovakia, 2013. p. 107–14.

[19] Krone M, Bidmon K, Ertl T. Interactive visualization of molecular surface dynamics. IEEE Trans Vis Comput Graph 2009;15(6):1391–8.

[20] Sigg C, Weyrich T, Botsch M, Gross M. Gpu-based ray-casting of quadratic surfaces. In: Proceedings of Eurographics/IEEE VGTC conference on point-based graphics. Eurographics Association, Boston, Massachusetts, USA, 2006. p. 59–65.

[21] Lampe OD, Viola I, Reuter N, Hauser H. Two-level approach to efficient visualization of protein dynamics. IEEE Trans Vis Comput Graph 2007;13 (6):1616–23.

[22] Goodsell DS, Olson AJ. Molecular illustration in black and white. J Mol Graph 1992;10(4):235–40.

[23] Weber JR. Proteinshader: illustrative rendering of macromolecules. BMC Struct Biol 2009;9(1):0–19.

[24] Kozlíková B, Kroney M, Lindow N, Falk M, Baaden M, Parulek J, et al. Visualization of molecular structure: the state of the art. In: EuroVis 2015 state of the art reports, 2015.

[25] Lindemann F, Ropinski T. About the influence of illumination models on image comprehension in direct volume rendering. IEEE Trans Vis Comput Graph 2011;17(12):1922–31.

[26] Luft T, Colditz C, Deussen O. Image enhancement by unsharp masking the depth buffer. ACM Trans Graph 2006;25(3):1206–13.

[27] Staib J, Grottel S, Gumhold S. Visualization of particle-based data with transparency and ambient occlusion. Comput Graph Forum 2015;34:151–60. http://dx.doi.org/10.1111/cgf.12627.

[28] Crassin C, Green S, Octree-based sparse voxelization using the gpu hardware rasterizer. In: Cozzi P, Riccio C, editors. OpenGL insights. CRC Press, 2012. p. 303–18.

[29] Ritschel T, Dachsbacher C, Grosch T, Kautz J. The state of the art in interactive global illumination. Comput Graph Forum 2012;31(1):160–88.

[30] Kajiya JT. The rendering equation. SIGGRAPH Comput Graph 1986;20 (4):143–50.

[31] Eichelbaum S, Scheuermann G, Hlawitschka M. PointAO—improved ambient occlusion for point-based visualization. In: Hlawitschka M, Weinkauf T, editors. EuroVis—short papers. The Eurographics Association; Leipzig, Germany, 2013. p. 013–7.

[32] Reinbothe C, Boubekeur T, Alexa M. Hybrid ambient occlusion. In: EUROGRAPHICS 2009 areas papers, 2009.