Ambient Occlusion and Edge Cueing to Enhance Real Time Molecular Visualization

Marco Tarini, Paolo Cignoni, and Claudio Montani

Abstract— The paper presents a set of combined techniques to enhance the real-time visualization of simple or complex molecules (up to order of 10^6 atoms) space fill mode. The proposed approach includes an innovative technique for efficient computation and storage of ambient occlusion terms, a small set of GPU accelerated procedural impostors for space-fill and ball-and-stick rendering, and novel edge-cueing techniques. As a result, the user's understanding of the three-dimensional structure under inspection is strongly increased (even for still images), while the rendering still occurs in real time.

1 Introduction

Interactive protein visualization is an important application with harder requirements every year: public databases like RCBS Protein Data Bank [1] are storing a large number of molecular structures of ever-increasing complexity. It is essential to be able to visualize the shape of these proteins in an interactive, meaningful and insightful way so that users can correctly understand the three-dimensional structure of these shapes.

Many software systems, such as for example RasMol[28], CN3D [6] or Chimera [5], are available to help scientists in this task. These systems offer various visualization modalities that map the inner structure of the molecules into 3D shapes according to an almost standard set of paradigms as *Balls-and-Sticks*, *Space-Fill*, *Licorice*, *Ribbons* and various kinds of accessibility surfaces. These approaches are able to describe local geometrical and chemical properties of the inspected structure and to provide insights on chemical traits of the molecule. An emerging problem is that, due to the growing size and complexity of the analyzed proteins, all of these visualization modes map proteins into three dimensional structures that, when rendered with standard local shading techniques, fail to produce a high-level comprehensible picture to the user. In other words, for complex molecules it is very difficult to perceive the overall 3D structure of the protein from a single image. This problem is only diminished, but not solved, when other common visualization techniques, as ribbons, provide a higher level description of the structure.

From a rendering oriented point of view, the problem is closely related to the use of local lighting models, since global illumination effects are usually not available in a handy interactive way. Sometimes more sophisticated approaches are used by means of off-line tools like raytracers for generating high quality images for important presentation cases (like journal or web covers); but these tools never aid the research during the interactive visualization process.

The most common (non-local) effects that are used for enhancing the rendering are cast shadows and depth cueing. Both of them are available in most of the well-known molecular visualization systems, but these approaches often fail to create an easily comprehensible image. For example, consider the molecule shown in Fig. 7 lower left: while it is clearly better than the plain rendering (top left), the shape of some portions of the protein is still very ambiguous. Other common techniques adopted to overcome this problem are interactivity and the use of stereoscopic displays.

In recent years much of the research efforts on large protein visualization has targeted efficient rendering of these large 3D structures, to achieve real interactivity. Various systems were presented with this purpose, both relying on existing 3D API [4] or trying to exploit features of recent graphics hardware or multiresolution techniques [11, 10]. Importantly, while the focus of this paper is to raise the quality of the perception of the shapes through sophisticated shading and edge cueing effects (summarized in Fig. 1), the interactivity of the whole system is a fundamental feature of our approach: all presented techniques work in realtime for very large proteins like the 1AON depicted in Fig. 7 (around 60K atoms) and do not need any preprocessing phase of significant length.

The main contributions of this paper can be summarized as follows:

- the use of advanced shading techniques for enhancing the perception of the 3D shape of large proteins;
- a novel technique for parameterizing the surface of molecules represented as SpaceFill or Ball and Sticks models (sec. 3.3);
- a novel technique for efficiently computing ambient occlusion information for molecules (sec. 4);
- an efficient approach for the rendering of molecular models using GPU based procedural textured impostors. (sec. 3.1);
- two interactive techniques for enhancing edges in molecular renderings (sec. 5).

- *Marco Tarini is with Università dell'Insubria, Varese, Italy, E-mail: m.tarini@isti.cnr.it.*
- *Paolo Cignoni is with I.S.T.I. - C.N.R, Pisa, Italy, E-mail: p.cignoni@isti.cnr.it*
- *Claudio Montani is with I.S.T.I. - C.N.R, Pisa, Italy, E-mail: c.montani@isti.cnr.it*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

2 Previous work.

We aim to enhance the shape perception process during molecular visualization by means of stylistic rendering techniques combined with a more sophisticated shading model. In other words, we are defining an *illustrative visualization* approach [31, 8] for enhancing the information-effectiveness of the rendered images of molecules.

We adopt more a sophisticated shading model that approximates the light coming from an uniformly diffusing lighting environment. This was shown to be useful in [17], where the authors report the results of perceptual experiments showing that depth discrimination under diffuse lighting is superior to that predicted by a classical sunny day/direct lighting model, and by a model in which perceived luminance varies with depth. The inadequacy of local lighting models was already noted in [30], where the use of a *vicinity shading*, a variant of the obscurance term proposed in [32], was proposed to enhance the visualization of volumetric datasets. Similarly in [21] the accessibility shading approach was introduced, where the geometric local (and global) accessibility of a point is used to modify the Lambertian shading of the surface in order to darken deep, difficultly accessible areas.

Enhancing edges and silhouette As was shown in [26, 9, 20, 14] finding and displaying silhouette edges is an important task that is used by illustrative rendering approaches to improve the readability of a 3D scene. Many different techniques have been proposed and a good survey on this matter can be found in [13]. Some of the proposed

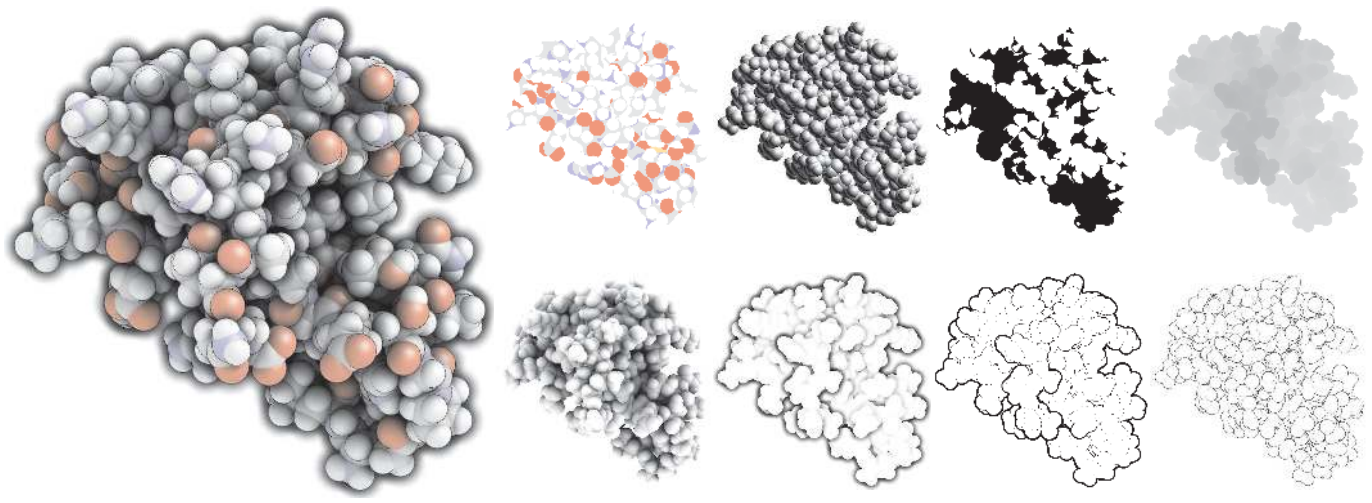


Fig. 1. Good molecular rendering can be more informative, clearer, and more capable of communicating a shape when it is the combined result of several effects and techniques. Top row: base atom colors, direct Lambertian and Phong illumination, self-shadowing, depth cueing. Bottom row (the topics of this paper): ambient occlusion, depth-aware halos, depth-revealing contour lines, and intersection-revealing contour lines.

approaches, like [26], work directly in image space, by processing the rendered depth buffer of the scene; others works on the mesh representation finding edges of the mesh that are part of the silhouette for the current viewpoint. Other authors exploit graphics hardware rendering features to perform this task, such as the z-offsetting technique presented in [25] that produces borders by rendering front and back faces of a scene with a different z-offset. Recent approaches, like ours, exploit the programmability of the graphics hardware to implement directly on the GPU most of these approaches. A more detailed comparison between our approaches for enhancing the edges of a molecule and the existing approaches is reported in section 5.

Enhancing the ambient term In local shading models the effect light which does not come directly from the primary light source has to be approximated. Otherwise, the portion of the scene which is not directly lit will come out entirely dark. Even without resorting to more correct (and complex) global illumination solutions, shortcuts are possible. The commonest and cheapest solution [23] is to use a simple per-scene constant term, but this approach leads to a notable flatness in the portions of the scene which are not directly lit.

The approach has been improved by explicitly computing for each point of the surface its accessibility value, which is the percentage of the hemisphere above each surface point not occluded by geometry [16]. This useful technique is commonly known as *ambient occlusion* and it is used in many production environments to add an approximation of the shadowing of diffuse objects lit with environment lighting. For example, ambient occlusion is precomputed in the interactive visualization system described in [2].

Variants of the ambient occlusion term called *obscurance* have been introduced in [32, 12], where the authors propose to exponentially weight the occlusion factor according to the distance of the occluders in order to enhance the shadowing effects of near occluding surfaces. In all of the above proposals, the computation of this extended ambient term is performed using a traditional ray-traced approach. In [27] graphics hardware is exploited to efficiently compute a per vertex ambient occlusion term. They render all geometry as seen from a light source direction into the depth buffer. Then, all vertices are rendered again as a point set. For each vertex an individual hardware based occlusion query is used to find vertices that passed the depth test and which are therefore visible for the considered light source; finally a visibility matrix M is stored per vertex. A similar approach, based on the use of the GPU for the computation of the ambient occlusion term, has been proposed in [22], and extended to the computation of a first bounce of the diffuse interreflection of light in [3]. Our approach for computing the ambient occlusion term is somewhat similar to the above techniques, but it exploits a different parametrization and access strategy for storing the computed results relying on the procedural nature of the molecular datasets; moreover, we add two significant optimizations: one exploiting the structure of molecular shapes, the other extendible to the general case.

As a final note, 'accessibility' itself is quite an important concept in chemistry: introduced by Lee and Richards [18] it is used to determine which parts of a given molecule are accessible to (the spherical atoms of) another molecule. Note that this paper does not discuss the direct visualization of this chemical property but focuses on the improvements on 3D shape perception triggered by (approximate) non-local shading models and other illustrative rendering techniques.

2.1 Ambient Occlusion Definitions

Let us consider a point p on the surface with surface normal n_p. According to [15] we can define the *irradiance*, E, arriving at p as:

$$E(p) = \int_{\Omega} n_p \cdot \omega L(\omega) d\omega \tag{1}$$

where $L(\omega)$ is a scalar with magnitude equal to the radiance arriving from direction ω, and Ω is the set of directions above the surface, i.e. the direction for which $n_p \cdot \omega > 0$. This can be approximately evaluated by discretizing the domain Ω into k sectors $\overline{\omega_i}$ with a possibly uniform solid angle measure $|\overline{\omega_i}|$, and, for each sector, evaluating the radiance L only for a sample direction ω_i:

$$E(p) = \sum_{i=1}^{k} n_p \cdot \omega_i L(\omega_i) |\overline{\omega_i}| \tag{2}$$

The above equation becomes simpler if we consider a uniform lighting environment (where light comes uniformly from every direction, as under a cloudy sky). In this case, if we discard diffuse interreflection effects and therefore we take into account only direct lighting, $L(\omega)$ can be substituted by a simple binary function $O(\omega)$ valued 0 if the ray shoot from p along ω intersects our surface (and therefore the light coming from the sky is obscured) and 1 otherwise. The result can be considered a simple first order approximation of the whole rendering equation.

With the assumption of a uniform sampling of the ω directions,

$$E(p) = \frac{1}{4\pi} \sum_{i=1}^{k} n_p \cdot \omega_i O(\omega_i) \tag{3}$$

3 Efficient Rendering of Ball and stick and Space Fill

In our scenario the scene is composed by two simple types of primitives: cylinders (side area only) and spheres. We use 2D impostors for both primitives. One impostor is dedicated to each occurrence of cylinder or sphere in the scene, and it is rendered as a 2D rectangular quad in the viewing plane which encapsulates the projection of the primitive it stands for (see Fig. 5).

The rationale is that impostors can be made much more rendering-efficient than a triangular tessellations, both during pre-computation of global illumination and in the final rendering. This is critical because we need to target large ($> 50k$ atoms) organic molecules as well. This is also memory-efficient, as the surface of the molecule is succinctly defined in an implicit way. The result is visually better, as the triangle tessellations are just linear approximations bound to produce shading and intersection artifacts.

Another advantage of the impostor approach is that, working on image space, a number of rendering effects (in particular border-oriented ones) become straightforward and computation-friendly (see Sec. 5).

However, this approach poses the problem of how to store a signal, defined over the implicit surface, to record the precomputed ambient occlusion terms. This problem is addressed in Sec. 3.3.

3.1 Procedural impostors

We need our impostors to be z-, normal-, α- and u,v- mapped. For a given view, each 2D point q inside an impostor represents a 3D point p on the primitive P (the point visible through it), and has the following attributes:

- the Boolean membership of q inside the projection of P — in order to discard the corresponding fragment otherwise;
- the normal p_n of p — in order to apply direct illumination;
- the depth z of the fragment — in order to correctly compute the intersections between primitives, and also to compute shadow-maps;
- the texture position u,v for the corresponding visible 3D point on the primitive — in order to access any attribute previously stored for p.

The last point is crucial, because we need to store per-position information for our primitives (in particular, a value for ambient occlusion, see Sec. 4). In practice we are resorting to a global 2D parameterization of the entire surface of the molecule, intended as the set of the surfaces of all its primitives (see Sec. 3.3).

A solution could be to store the listed attributes in a set of fixed textures accessed for each fragment. This could impact performance because of the additional texture bandwidth consumption, and would cause aliasing problems (especially around borders, as presence of semitransparent interpolated texels makes the rendering sort-dependent). On the contrary, our impostors are procedural, meaning that all attributes are synthesized on the fly. This greatly helps aliasing problems, and also improves flexibility and adaptability.

3.2 General schema

For each impostor, we send four vertices with appropriate values.

A **vertex program** is dedicated to expand the impostor around the processed primitive, aiming at producing the least number of fragments outside the impostor (but producing all the fragments relative to the front facing part of the primitive). The initial vertex position is projected and the impostor is then expanded in image space.This happens differently for the two primitives (see Sec. 3.3).

Another objective of the vertex shader is to perform as many pre-computations as possible in order to minimize residual per-fragment workload (for this reason we prefer not to use the *Point Sprites* extension). This general optimization technique is, in our case, particularly fruitful because our impostors represent large, high-level primitives, and the pixel-to-vertex ratio is accordingly larger than usual.

Constant per-primitive parameters (e.g. base atom color) are passed down unchanged by the vertex program to the fragment processor.

The **fragment program** computes and then processes the required fields, including membership, $u-v$ texture position, depth, or lighting (according to need of the current rendering mode and rendering pass). Depending on the rendering technique, the final values are written either in the current screen buffers or in intermediate textures for subsequent passes.

3.3 Parameterizing the surface of a molecule

Our visualization algorithm requires a data structure to store the computed ambient occlusion terms. This can be assumed to be a low frequency signal, i.e. to vary smoothly over the surface of the molecule. For the case of atoms, this signal is defined over spheres, so it would be possible to store it, in a conveniently compact way, as a small set of spherical harmonic coefficients [29] per atom. However compact this representation would be, we choose a direct sampling representation because it is more local in nature and therefore more efficient during rendering: only few samples need to be read and interpolated to reconstruct the signal in a specific location.

Now we need a way to store a sampling over spheres and cylinders. Since the surfaces are implicit, we resort to a specially formatted texture that is to be coherently accessed during the rendering of the impostors.

We assign to each instance of sphere or cylinder a unique rectangular (non necessarily squared) *patch* of texture space. All patches have the same height so that they can be trivially packed in a single global texture. During any rendering that requires texture accesses, the 2D offset of the patch, relative to the origin of the the global texture, is sent as an additional attribute.

Depending on the size of the molecule (number of atoms) the sizes of texture patches vary from as few as 4 to hundreds of texels per side; for example, for a molecule with around 1K atoms 32×32 patches can be packed in a 1024×1024 texture, while the surface of up to 64K atoms can be sampled into 4×4 large patches of a 1024×1024 texture (when many atoms are present, the radius of the molecule is probably very large as well, reducing atom average screen size and which means that fewer texels per atom will suffice).

For both kinds of primitive we define a mapping M between every point on its area and a position inside the corresponding texture patch. The function M needs to be simple, as it will be computed both ways within the fragment shader: during scene rendering, given a point inside the impostor q, we need to compute $M(P(q)) = (u,v)$ to find texture coordinates for current fragment; during ambient occlusion computation (see later in sec 4), we will need to compute M^{-1} for each fragment. Naturally we seek functions M that exhibit low distortions, so that texture mapping artifacts are minimized.

We use one of two alternative parametrization schemas M_{s1} and M_{s2} for spheres, and one schema M_c for cylinders.

Spheres: parameterization The mapping M_{s1} is a gnomonic projection over a cube, followed by packing of its 6 faces in a rectangular patch with a 2×3 aspect ratio. Following [24], M_{s2} is a gnomonic projection over an octahedron, which is unfolded into a square. The mapping M_{s2} and its inverse are easier to compute, taking fewer operations in the fragment shader (in our implementation, 9 ARB low level fragment operation instead of 16), whereas M_{s1} presents a minor stretch energy ([24]). Another advantage of M_{s2} over M_{s1} is that it requires less duplicated texels (see later in Sec 3.5), so no choice fully dominates the other. This choice is orthogonal with the rest of any of the algorithm discussed here.

M_{s1} and its inverse are well known. We report the exact version that we use for M_{s2}, that goes from the surface the unit, origin centered sphere to the patch parameterized as the square $[-1..+1]^2$ (we will use a GPU friendly formulation, with the fewest possible cases):

$$M_{s2}(x,y,z) = \begin{cases} (\frac{x}{d}, \frac{y}{d}) & \text{if } z \leq 0 \,, \\ (sign(x)(1-\frac{|y|}{d}), sign(y)(1-\frac{|x|}{d})) & \text{if } z > 0 \end{cases} \quad (4)$$

where $d = |x| + |y| + |z|$. The inverse, up to a normalization, is:

$$M_{s2}^{-1}(u,v) = \begin{cases} (u,v,h) & \text{if } h \geq 0 \,, \\ (sign(u)(1-|v|), sign(u)(1-|v|), h) & \text{if } h < 0 \end{cases} \quad (5)$$

where $h = 1 - |u| - |v|$

Note that the mapping M_{s1}, being gnomonic, does not need its argument to be normalized prior to use.

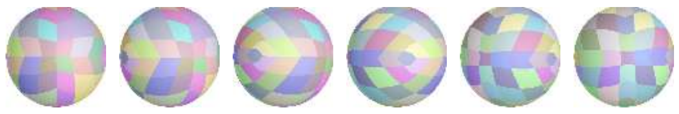


Fig. 2. Applying a small 2D texture map over a sphere impostor, using a 10×10 octahedron mapping M_{s2}. In order to show the behavior of M_{s2} the virtual sphere (impostor) is rotated from left to right by 180^o. For illustration purposes bilinear interpolation is disabled and random values are assigned to texels, so that they are clearly visible. Duplicated texels are assigned to the same color (see later). Refer to fig. 6 to see the used 2D texture.

Spheres: impostors (see Fig. 2). Four vertices are sent at the position of the sphere center, distinguished by the value of a special attribute (s,t) assigned respectively to $(\pm 1, \pm 1)$ to designate each vertex to one different corner of the impostor quad.

Each vertex is displaced in screen coordinates after projection to produce a screen aligned quad. The displacement is given by $(s \cdot r \cdot S_g, t \cdot r \cdot S_g,)$, where r is the radius of the sphere (passed as an attribute) and S_g is the global scale factor, extracted once per frame from the current view matrix (as the cubic root of its determinant) and stored in a environment parameter.

Interpolated values for s and t are passed down to the rasterizer, so each fragment inside the screen quad is produced with an assigned relative position $(s,t) \in [-1..+1]^2$. We call this 2D space *impostor space*. Fragments with $|(s,t)| > 1$ are immediately discarded. For every surviving fragment, the original z value is decreased by $(1 - |(s,t)|) * r$ and its normal in screen space is assigned to $n = (s, t, (1 - |(s,t)|))$. The normal is then transformed in object space with $n' = A_{MV}^{-1}(n)$ (i.e. by a multiplication with the inverse — i.e. the transpose — of the current model-view matrix), and the result is fed to the chosen mapping $(u,v) = M_s(n')$. Texture is fetched at the position (u,v) plus the offset for the current patch.

Cylinders: parametrization Let us consider the normalized, z-axis aligned cylinder centered in the origin defined as the set of points $\{(x,y,z) | x^2 + y^2 = 1, x \in [-1..+1]\}$.

The side area of cylinders is developable, so it would be naturally parameterizedwith the zero-stretch parametrization $M(x,y,z) = (atan2(x,y)/\pi, z)$.

However computing it in the fragment shader (either direction) would be very demanding, since it requires trigonometric functions (direct and inverse) which are not supported in the typical GPU. These functions could be either approximated (e.g. with Taylor formulas), which is time consuming, or sampled from a 1D texture, which requires additional texture accesses. We prefer a cheaper alternative, consisting in the adoption of a simplified mapping M_c, which is a projection over a square-based prism and is defined by:

$$M_c(x,y,z) = \begin{cases} \left(\frac{y}{2(|x|+|y|)} - 0.5, z\right) & \text{if } x \geq 0\,, \\ -\left(\frac{y}{2(|x|+|y|)} + 0.5, z\right) & \text{if } x < 0 \end{cases} \qquad (6)$$

Note: the prism sides are defined on the $|x| \pm |y| = \pm 1$ planes rather than the $|x| = \pm 1$ and $|y| = \pm 1$ to simplify the number of cases; this, of course, does not affect the quality of the parametrization. The inverse, up to a re-normalization of first two components of the result, is given by:

$$M_c^{-1}(u,v) = \begin{cases} (1 - |2u-1|, 2u-1, v) & \text{if } u \geq 0\,, \\ (|2u-1| - 1, 2u-1, v) & \text{if } u < 0 \end{cases} \qquad (7)$$

Cylinders: impostors (see fig.3) To draw a cylinder impostor we send two pairs of vertices, each located at either end of the axis of the cylinder. After projection the points are displaced in a direction parallel to the image plane and orthogonal to the cylinder axis, to form a rectangular quad.As before, each vertex is assigned a coordinate $(s,t) \in (\pm 1, \pm 1)$, which will be interpolated for the fragments.

To lift the burden off the fragment program, the vertex program computes a set of (signed) intermediate values which are constant over all the impostors and will be reused by all the fragments. Some of these values are found in *cylinder plane*, which is defined as the plane embedding the cylinder axis and orthogonal to the viewing plane (see fig.4).

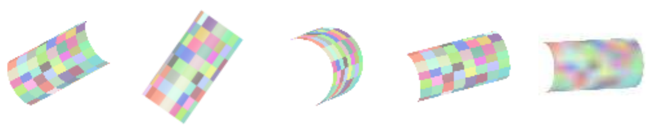


Fig. 3. Applying a 8×16 texture over a cylinder impostor, using a prism mapping M_c. For illustration purposes, bilinear interpolation is disabled, except in the last image, and random color values are assigned to texels. Duplicated texels are assigned to the same color.

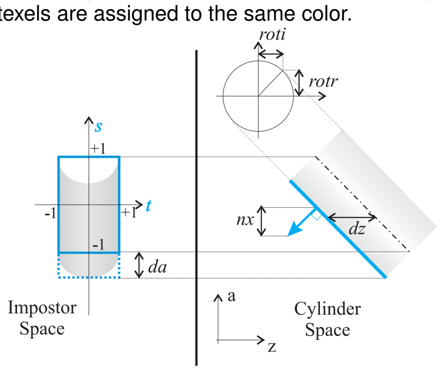


Fig. 4. Processing for the corners of cylinder impostors. See text.

In particular , we compute the values of dz, da and (nx, ny) which are, respectively, the offset in the z direction, the offset in v texture position, and the normal, for a fragment in impostor space at $s = 0$, i.e. on the projection of the cylinder axis. For all other fragments, these values must be multiplied by $(1 - \sqrt{s})$. The normal, in impostor space, will simply be given by (s, nx, nz).

After adding da, fragments with v lying outside $[-1..1]$ are discarded.

Another task performed by the vertex shader is to extend the position of the projected vertices also along the axis direction to accommodate for the part of the cylinder extruding from the original impostor space (dotted blue line in fig. 4), only for vertices for which $da \cdot s$ is positive. Finally, the vertex shader also computes the offset of the texture position u (the one varying along the diameter of the cylinder). This data consists of an offset angle and is stored and sent to the fragment shader as the unit-length complex number $(roti, rotr)$, because that is a space- and computation-efficient way to store a 2D rotation (we cannot send a single scalar value because of the distortions introduced by the mapping M_c).

3.4 Patch packing and determination of patch size

Thanks to the low-frequency nature of the stored signal, we can use a sparse sampling: every visible texel will cover in most cases multiple screen pixels. This means that minification filters are not needed: MIP-mapping is disabled, and so texture patches can be of arbitrary (non power of 2) sizes without causing any artifact. We are free to choose the best fitting patches size, according to the number of patches and available texture size. Consequently unused space (at right and bottom borders of the global texture) is usually small.

For simplicity we choose to ignore differences in the sizes of atoms, devoting squared patches the same size s_p for each one. If sticks are present, we pack two (optionally three) rectangular stick patches into a $s_p \times s_p$ meta-patch. If k is the total number of patches and meta-patches, and we plan to use a $s_t \times s_t$ texture, we choose s_p simply as $\lfloor s_t / \lceil \sqrt{k} \rceil \rfloor$.

3.5 Accessing texture

Because of the same reason, it is mandatory for us to interpolate between samples. While it would be possible to adopt an *ad-hoc* texel interpolation schema that accesses several non necessarily adjacent texels, in order to increase performance we prefer to perform a single, standard bilinearly interpolated texture access per fragment (an

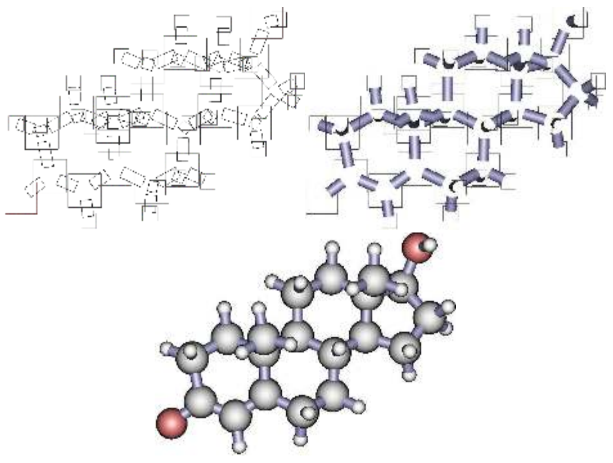


Fig. 5. Rendering a molecule with impostors. Top left: the actual impostors used are shown as wireframed quads. Cylinders impostors are parallel to the projection of their axis, and ball impostors are screen aligned. Top right: cylindrical impostors have been processed. Bottom: ball impostors have been projected. Intersections between primitives are correctly computed via the zeta-buffer.

heavily optimized operation in graphic cards). If this is done without care, discontinuity of M would became visible as discontinuity artifacts in the corresponding parts of the primitive (see Fig. 6). We solve the problem by offsetting all texture coordinates by half a texel and by replicating some of the texel in each patch: fig. 6) shows how the problem is fixed for M_{s2}; similar solutions are adopted for M_{s1} and M_c.

The amount of texel replication needed by the two sampling is different. With M_{s1}, a texture patch of size $3n \times 2n$ texels contains $(n^3 - (n-2)^3)$ unique texels out of $6 * n^2$. For M_{s2}, a $n \times n$ texture has $n^2 - 2n + 1$ unique texels out of n^2. From this point of view M_{s2} is therefore advantageous.

4 GPU Computated Ambient Occlusion

In order to compute ambient occlusion, we first build a set of directions sampling ω. To make it well distributed, we start with a tetrahedron or octahedron and we subdivide it until the desired number of directions is reached, then we apply a Laplacian smoothing to the set of found directions.

Similarly to [27], we perform a pair of off-screen rendering passes for each direction d_i in the set. In the first pass a "shadow-map" is produced rendering the z-buffer of molecule using d_i as view direction.

The next pass is a rendering that writes over the texture for the molecule, and accesses the shadow-map produced in the previous pass. For each primitive we send a quad covering the corresponding texture patch, complete with attributes encoding the position and shape of that primitive. For each produced fragment at position (u,v), we compute $p = M^{-1}(u,v)$, then we transform p with the same viewing conditions used during the first pass, and comparing the resulting depth with the one extracted from the shadow-map to determine whether p is lit by light coming from d_i. If so, the light contribution for d_i, according to equation 2, is accumulated at the corresponding pixel/texel through alpha blending.

Note that replicated texels (see Sec. 6) are dealt with correctly because both copies will be mapped by M^{-1} in the same 3D position and therefore will produce the same result.

Shadow-map computation is particularly undemanding in our case. It is known that shadow-map, in the case of closed blockers, is more robust if the mid-surfaces (half-way between front-facing and back-facing ones) are drawn. In our case (sphere and cylinders), the mid-surfaces are flat and can be rendered by simply disabling per-fragment depth displacement during impostor rendering. The fragment programs stops being depth-replacing, and, since shading is disabled as well, reduces to a simple membership test (for spheres) or to a single output operation (for cylinder).

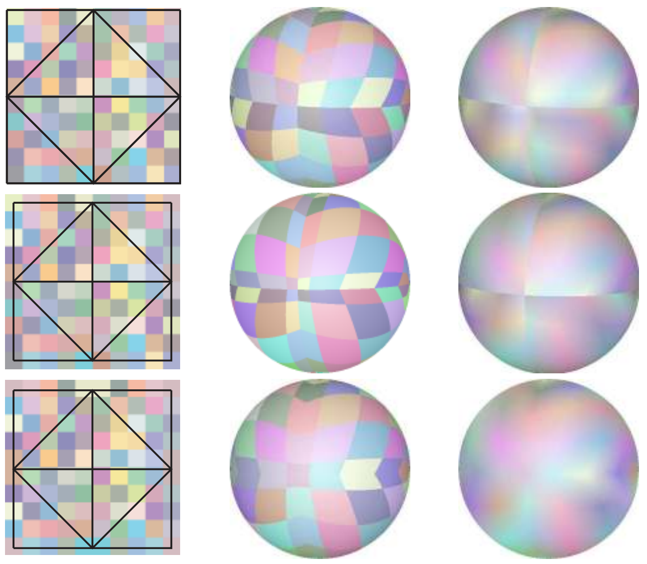


Fig. 6. The continuity problem when using bilinear interpolation to access texture. In the left column the texture patch is shown in $u-v$ parameter space. The second and third column show a rendering of the impostor, with the sphere rotated to show the most problematic region, which is its "back" part, where the four corners of the texture patch meet. For illustration purposes, in the middle column we use a closest-sample filter; in the right column a standard bilinear-interpolation filter is used. Top row: when the entire surface of the texture patch is used, bilinear interpolation shows discontinuity lines across cuts. To solve this problem, the texture patch is shrunk in texture space by half a texel in every direction (middle row), and in all four sides of the patch texels values are mirrored around the middle of the edge (bottom row). Texture is never accessed at positions outside the outlined square. The resulting impostor is smooth (bottom right). In this case, a total of 81 out of 100 texels are unique.

We add another optimization, which works for general shapes, consisting in the computation of the contributions of two opposed light directions d_i and $-d_i$ at a time. The two corresponding shadow-maps are drawn side to side into a single texture. In the second pass, each 3D point p can be lit by only one of d_i and $-d_i$, according to normal of p (see equation 2), so a single texture access is performed in any case from the appropriate half of the shadow-map. This effectively halves the processing time for the second passes.

4.1 Applications

We found that ambient occlusion adds dramatically to the clarity of the molecular rendering, in all visualization modes. This is true for small molecules (few dozens or few hundreds atoms, fig. 8), for medium molecules (few thousands atoms, fig. 8), and even more so for larger ones (several tenths of thousands of atoms, fig. 7). In the last case, standard direct shading alone, even when enhanced with depth cueing and shadows, can fail to produce an intelligible still image, while the 3D structure becomes evident when ambient occlusion is used, even without any other contribution.

5 Further Enhancing Visual Quality

There are a number of visual effects that are easy to add in our impostor-based framework.

5.1 Depth aware contour lines

Since we work in image space, contour lines can be easily added.

The first effect consists in drawing a solid line around each primitive. Since these lines do not affect the depth value of fragments, they naturally disappear at the intersections of atoms, ad occur only to separate actually detached atoms, boosting the clarity of rendered images (see fig. 9). Less importantly, the border can help to tell the difference between larger vs. just nearer atoms when perspective views are used.

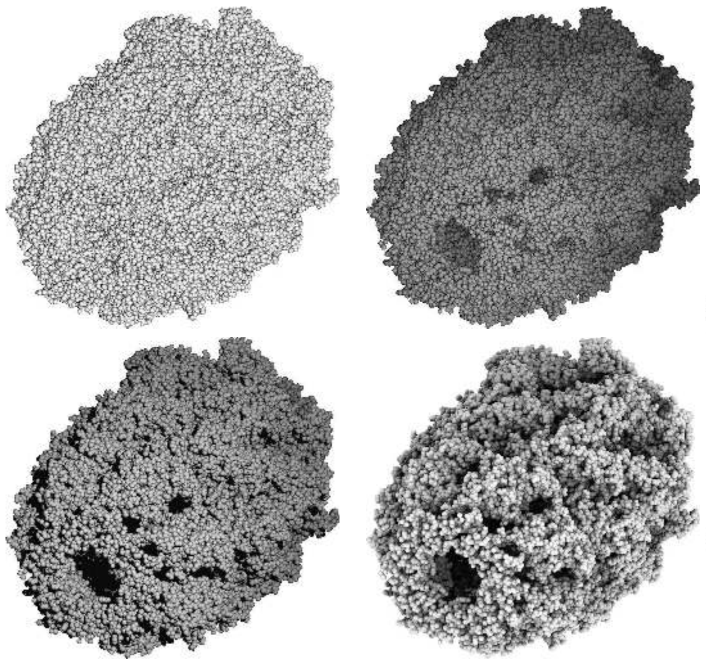


Fig. 7. Example of the efficacy of Ambient Occlusion to deliver impression of 3D shape in still images for large molecules. A molecule of *1AON* (model taken from [1]), consisting of 58688 atoms is shown with (from top left, in Z order): standard direct lighting, direct lighting with depth cueing, direct lighting with cast shadows and depth cueing, and Ambient Occlusion alone. The last image uses a 1024×1024 texture with 4×4 sized octahedron based patches.

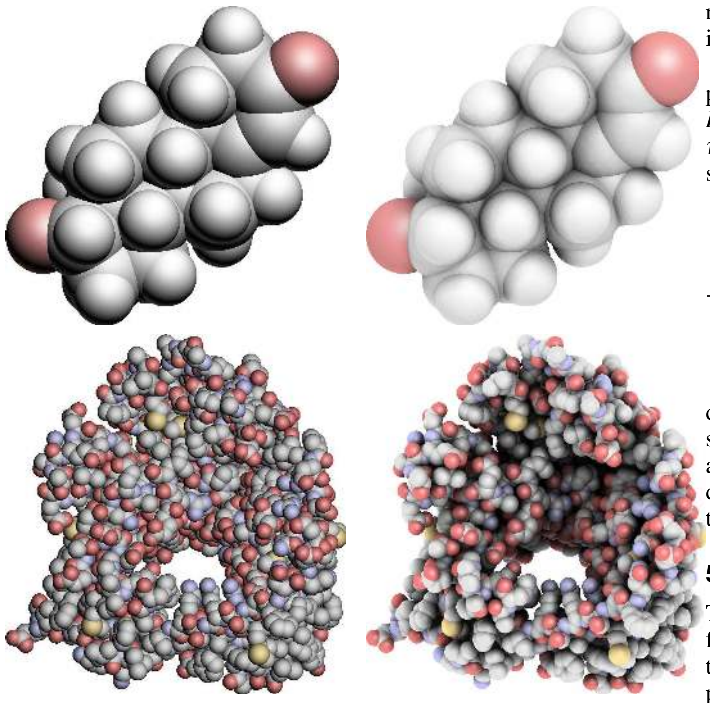


Fig. 8. Other comparisons between direct illumination (left) and ambient occlusion (right) for small (above) and medium (below) molecules. Above: *testosterone* (49 atoms) and, below, *porin* (2219 atoms). Models taken from [1].

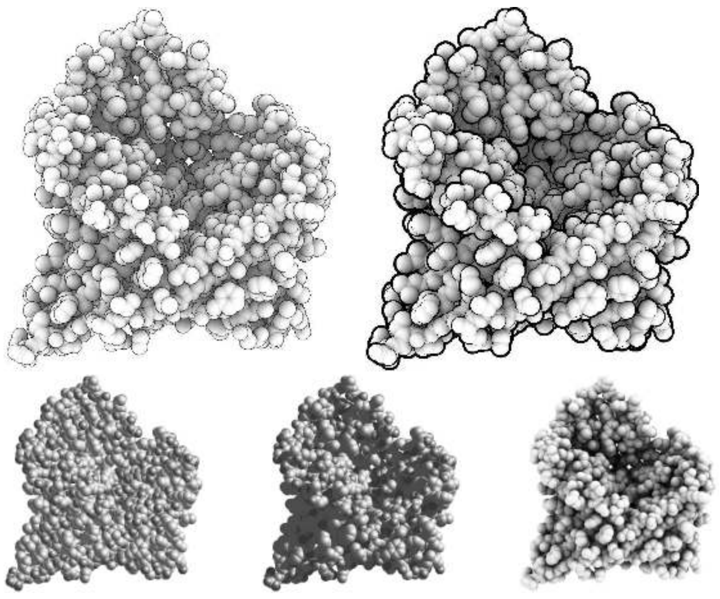
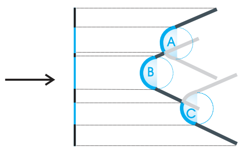


Fig. 9. Above, left: a rendering with constant width, anti-aliased lines. Above, right: thinker, depth aware lines. Bottom (for comparisons): direct illumination, cast shadows, Ambient occlusion.

To achieve solid lines, fragments detected to be just outside the range R of the impostor (namely with a distance d from the center with d between R and $R+\xi$, for a given parameter ξ) are overwritten with black color rather then discarded. The internal border of the lines can be interpolated between line color and color of the primitive (using $(d-R)/\xi$ as weight) to eliminate pixel aliasing on the internal part of the contour.

For a more informative rendering, the thickness of the lines can be made dependent on the jump in depth between the primitives separated by the contour line (see fig. 9). This effect proved useful in illustrative rendering (in the different context of pen-and-ink illustrations of trees) in [7].

In our approach this can be cheaply done in a single-pass rendering: fragments at a 2D distance d between R and $R+\xi$ are pushed back, increasing their depth by the value $\eta(\xi - R)$. This produces a truncated cone topped with a semi-sphere, and the depth buffer ensures the desired effect. To explain why, here we see a 2D "side view" of the depth buffer after that atoms A B C are drawn. Depth-culled fragments are shown in green. In the final screen buffer (represented by a line), the black segment separating A and B appears shorter than the one separating B and C, signalling to the viewer that the depth jump between A and B is smaller. The parameter η dictates how strongly depth jumps affect thickness (ξ^2 is the maximal depth jump after which the line thickness stops increasing). Relying solely on the depth buffer, this algorithm is independent of the order of rendering of the primitives.



5.2 Halo Effect

The very common way to suggest depth for 2D images (especially useful when they represent unfamiliar objects like molecules) is to resort to depth-cueing, where fragments are darkened — or more generally pushed towards a given background color — according to their depth. This, however, has drawbacks: the vision of the furthest parts of an object is hindered by the artificial "fog", which is bound to reduce contrast. Even worst, this effect is distributed over all the images, also far from difficult to visualize depth steps.

We propose here to achieve a similar effect, but without unnecessary losing contrast over the entire image: we draw transparent "halos"

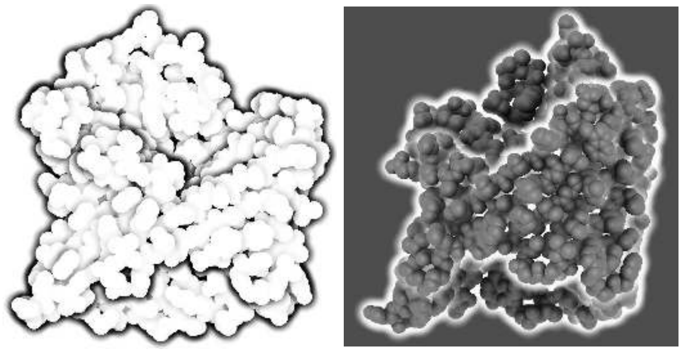


Fig. 10. Halos drawn around molecules to communicate a sense of depth. Left: for illustrative purposes, black halos alone are drawn. Right: a dimly lit rendering combined with white halos.

around each atom. Each point in the halo is more opaque the bigger the distance between it and its background. The halo fades to zero also with the distance from the atom border (see fig. 10). Both darkening and lightening halos can be used.

Not only this helps identifying depth discontinuities, but it also helps when it comes to immediately recognize the general slope of "walls" of atoms (wall slopes cannot be easily identified by direct shading alone because they are composed by primitives). When a wall composed by several atoms is seen from grazing angles, the cumulated effect of their halos communicates so to the viewer.

A similar use of brightening/darkening halos to improve the perception of depth discontinuities for general scenes was proposed, in a totally independent way, by Luft et al. [19].

In our approach halos are rendered in a second pass, after the depth buffer is set, and stored in a separate texture. Larger circles are drawn around each atom, as flat rendering impostors passing through the atom. In this pass we do not affect the zeta of the produced fragments. Only a color is produced according to z-difference and distance from the center. The depth test is enabled but we do not write on the depth buffer. Every rendered fragment darkens (or brightens when white halos are used) the color of the corresponding screen pixel, in an order independent way.

5.3 Z-clipping of impostors

When the near clipping plane cuts through an atom, the top part of the atom is clipped out of view, leaving visible whatever is behind the atom (background, or other intersecting atoms). The effect harms the clarity of the scene because most people intuitively imagine the atom modeled as "full" solid spheres rather than as thin empty shells. To solve this, when a fragment of an impostor (after z computation) is nearer to the eye than the near clipping plane, we test the depth of the other intersection of the current view ray with the sphere (which is trivially found inverting the z displacement). If also this point falls on the viewer's side of the clipping plane than the fragment must be discarded. Otherwise, the fragment is moved onto the clipping plane, its normal is overwritten to $(0,0,-1)$, to suggest a flat surface of a "cut" atom, and the ambient occlusion term is also set to a full lit value.

If the new depths of clipped fragments were all set to the same value, the second of two clipped, intersecting primitives would be drawn over the first. To show a plausible intersection inside the atoms, we set the new depth value of capped atoms to to a small positive value dependent on their original computed depth z. We use $\varepsilon(K+z)$ for a small ε and a constant K. This way, depth-test correctly computes the intersection (see Fig. 11).

We also lighten the ambient occlusion term for fragments close to the clipping plane, to roughly simulate the temporary culling of light-blocker in front of them.

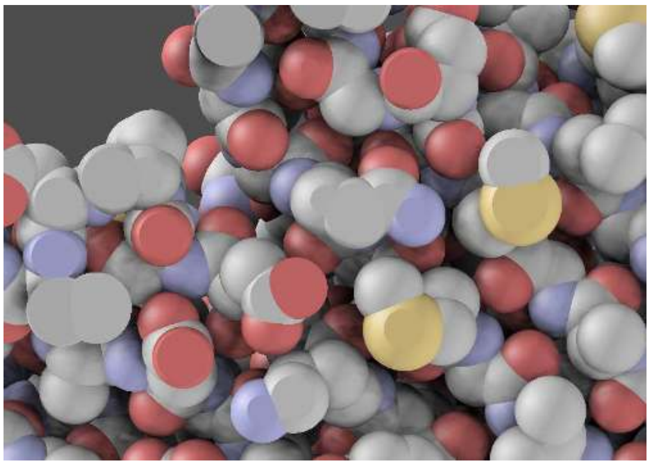


Fig. 11. A detail of a rendering showing a molecule cut by the near clipping plane.

6 Results

We presented a set of rendering techniques that, in our opinion, effectively extends the ability to produce real time molecular renderings which clearly communicate 3D shapes. Some of these techniques, like edge-cueing ones, have not been presented before (to our knowledge), whereas the occlusion culling consisted in the adaptation and optimization for the case of impostor-based rendering of a class of methodologies developed for polygonal meshes. These visual effects are not mutually exclusive and are designed to be combined in the same rendering.

The ambient occlusion computation time is always very short, averaging 233 light directions per second for a medium model of 2219 atoms and around 900K effectively used texels, and 15 views per sec for the largest we tried of around 60K atoms (on a Athlon - 2.6 GHz, with an ATI X1600). This means that ambient occlusion can be computed interactively if the application requires it.

The proposed impostor based rendering approach proved very effective. In our tests the frame-rate kept in sync with the monitor refresh rate in all but the most demanding scenarios (full screen, very large molecules, all effects combined), and it never dropped below 20 frames per sec. The main weakness of our approach lies in the depth complexity. Use of hierarchical depth buffer structures for quick culling of entire primitives could further accelerate the rendering.

One key advancement proposed here is the ability to combine the flexibility of texture with the efficiency of 2D impostors, admittedly only for the special case of spheres and cylinders. This can probably be exploited in other ways as well.

A simple, molecule visualization tool that opens PDB files and delivers what is described in this paper is publicly available at the project home-page: `http://qutemol.sourceforge.net`.

Acknowledgements

We wish to thank Enrico Gobbetti and Mario Valle for insightful discussions, on importance of high quality shading in the three dimensional perception of complex shapes in space and Monica Zoppè for having introduced us to the field of Molecular Visualization.

References

[1] H.M. Berman, J. Westbrook, Z. Feng, T.N. Gilliland, G.and Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Res.*, 28:235242, 2000. http://www.pdb.org.

[2] R. Borgo, P. Cignoni, and R. Scopigno. An easy to use visualization system for huge cultural heritage meshes. In D. Arnold, A. Chalmers, and D. Fellner, editors, *VAST 2001 Conference Proc.*, pages 121–130, Athens, Greece, Nov. 28-30 2001. ACM Siggraph.

[3] Michael Bunnell. *GPU Gems 2*, chapter Dynamic Ambient Occlusion and Indirect Lighting, pages 223–233. Addison-Wesley, 2005.

[4] Tolga Can, Yujun Wang, Yuan-Fang Wang, and Jianwen Su. Fpv: fast protein visualization using java 3d. *Bioinformatics*, 19(8):913–922, 2003. http://www.pdb.org.

[5] USCF Chimera. http://www.cgl.ucsf.edu/chimera/.

[6] Cn3D. http://ncbi.nih.gov/structure/cn3d/cn3d.shtml.

[7] Oliver Deussen and Thomas Strothotte. Computer-generated pen-and-ink illustration of trees. In *SIGGRAPH*, pages 13–18, 2000.

[8] David S. Ebert and Penny Rheingans. Volume illustration: non-photorealistic rendering of volume models. In *IEEE Visualization*, pages 195–202, 2000.

[9] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard F. Riesenfeld. Interactive technical illustration. In *SI3D*, pages 31–38, 1999.

[10] Andreas Halm, Lars Offen, and Dieter Fellner. Biobrowser a framework for fast protein visualization. In *EUROVIS 2005: Eurographics / IEEE VGTC Symposium on Visualization 2005*, pages . 287–294, 2005.

[11] Andreas Halm, Lars Offen, and Dieter W. Fellner. Visualization of complex molecular ribbon structures at interactive rates. In *IV*, pages 737–744. IEEE Computer Society, 2004.

[12] Andrey Iones, Anton Krupkin, Mateu Sbert, and Sergey Zhukov. Fast, realistic lighting for video games. *IEEE Computer Graphics and Applications*, 23(3):54–64, May/June 2003.

[13] Tobias Isenberg, Bert Freudenberg, Nick Halper, Stefan Schlechtweg, and Thomas Strothotte. A developer's guide to silhouette algorithms for polygonal models. *IEEE Computer Graphics and Applications*, 23(4):28–37, 2003.

[14] Tobias Isenberg, Maic Masuch, and Thomas Strothotte. 3D Illustrative Effects for Animating Line Drawings. In *Proceedings of the IEEE InfoVis, July 19–21, 2000, London, England*, pages 413–418, Los Alamitos, California, 2000. IEEE Computer Society.

[15] James T. Kajiya. The rendering equation. *Computer Graphics (SIGGRAPH)*, 20(4):143–150, 1986.

[16] Hayden Landis. Production ready global illumination. In *Siggraph 2002 Course Notes:*, pages 331–338, 2002.

[17] Michael S. Langer and Heinrich H. Bulthoff. Perception of shape from shading on a cloudy day. Technical Report Technical Report No. 73, Max-Planck-Institut fur biologische Kybernetik, October 1999.

[18] B. K. Lee and Fred M. Richards. The interpretation of protein structures: estimation of static accessibility. *J Mol Biol.*, 55(3):379–400, Feb 1971.

[19] Thomas Luft, Carsten Colditz, and Oliver Deussen. Image enhancement by unsharp masking the depth buffer. *ACM Transactions on Graphics (Siggraph Proc.)*, 25(3), jul 2006. to appear.

[20] Lee Markosian, Michael A. Kowalski, Daniel Goldstein, Samuel J. Trychin, John F. Hughes, and Lubomir D. Bourdev. Real-time nonphotorealistic rendering. In *SIGGRAPH*, pages 415–420, 1997.

[21] Gavin Miller. Efficient algorithms for local and global accessibility shading. In *ACM SIGGRAPH '94*, pages 319–326, New York, NY, USA, 1994. ACM Press.

[22] Matt Pharr. *GPU Gems*, chapter Ambient occlusion, page 667692. Addison-Wesley, 2004.

[23] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975.

[24] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. *ACM Trans. Graph.*, 22(3):340–349, 2003.

[25] Ramesh Raskar and Michael F. Cohen. Image precision silhouette edges. In *SI3D*, pages 135–140, 1999.

[26] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *SIGGRAPH*, pages 197–206, 1990.

[27] Mirko Sattler, Ralf Sarlette, Gabriel Zachmann, and Reinhard Klein. Hardware-accelerated ambient occlusion computation. In *VMV*, pages 331–338, 2004.

[28] R.A. Sayle and E.J. Milner-White. Rasmol: biomolecular graphics for all. *Trends Biochem. Sci.*, 20:374376, 1995. available at: http://www.umass.edu/microbio/rasmol/.

[29] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH 2002:*, pages 527–536, New York, NY, USA, 2002. ACM Press.

[30] A. James Stewart. Vicinity shading for enhanced perception of volumetric data. In *IEEE Visualization 2003 (VIS'03)*, page 47, Washington, DC, USA, 2003. IEEE Computer Society.

[31] Ivan Viola, M. Eduard Gröller, Markus Hadwiger, Katja Bühler, Bernhard Preim, Mario Costa Sousa, David S. Ebert, and Don Stredney. Illustrative visualization. In *IEEE Visualization*, page 124, 2005.

[32] Sergej Zhukov, Andrej Inoes, and Grigorij Kronin. An ambient light illumination model. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98*, Eurographics, pages 45–56. Springer-Verlag Wien New York, 1998.