ABSTRACT

While molecular visualization software has advanced over the years, today, most tools still operate on individual molecular structures with limited facility to manipulate large multi-component complexes. We approach this problem by extending 3D image-based rendering via programmable graphics units, resulting in an order of magnitude speedup over traditional triangle-based rendering. By incorporating a biochemically sensitive level-of-detail hierarchy into our molecular representation, we communicate appropriate volume occupancy and shape while dramatically reducing the visual clutter that normally inhibits higher-level spatial comprehension. Our hierarchical, image-based rendering also allows dynamically computed physical properties data (e.g. electrostatic potential) to be mapped onto the molecular surface, tying molecular structure to molecular function. Finally, we present another approach to interactive molecular exploration using volumetric and structural rendering in tandem to discover molecular properties that neither rendering mode alone could reveal. These visualization techniques are realized in a high-performance, interactive molecular exploration tool we call TexMol, short for Texture Molecular viewer.


Keywords: molecular visualization, image-based rendering, texture-based rendering, imposter rendering, volume rendering, programmable graphics hardware, level-of-detail, hierarchy, multiresolution, synchronous view, computer graphics

1 INTRODUCTION

While molecular visualization software has developed over the years, today, most tools still operate on individual molecular (protein or RNA) structures and small electron charge and electrostatic potential fields, with little facility to manipulate larger multi-component complexes, integrate geometric and volumetric visual representations, or effectively depict molecular flexibility and dynamics. Few, if any, currently used programs allow for or enable interaction with multi-component macromolecules and their associated properties at the atomic level, reconstructed volumetric maps from tomographic and cryo imaging, that will become common in the next five to ten years.

We present TexMol (short for Texture Molecular Viewer), an interactive molecular exploration application created in response to the increasingly demanding visualization needs of the biology community. To efficiently visualize dynamic and flexible structures, TexMol uses a molecular specification file to construct the Flexible Chain Complex (FCC), a robust, dynamic data structure that serves as TexMol’s internal representation for molecular structures. The FCC models the flexible joints of a molecule and contains a biochemically-based hierarchy for level-of-detail (LOD) optimizations.

Besides high visualizer functionality, TexMol delivers rapid, accurate rendering via various novel applications of texture-based rendering techniques for structural and for volumetric representations. For the field of structural representation (e.g. CPK, (of union of van der Waals balls, where each atom is represented by a sphere with the radius equal to the van der Waals radius.), ball-and-stick model), recent advances in programmable graphics hardware have opened the door for texture-based rendering – also known as imposter rendering – that greatly reduces geometric complexity while preserving, and in some cases improving the visual fidelity of the final image. Structural rendering is aided by TexMol’s LOD hierarchy, allowing for static and dynamic multiresolution which reduce the visual clutter that often accompanies atom-level visualization while still maintaining biochemical structural information, such as residue-level grouping. Combined with the hierarchy, texture-based rendering allows TexMol to render large and previously intractable molecules.

TexMol also supports efficient volumetric visualization via texture-based techniques. By combining rendering modes, the visualizer can either map volumetric data onto the structural model of the molecule or it can juxtapose multiple volume sets and structure models concurrently. In both cases, the resulting visualization ties molecular structure to molecular function in an elucidating manner.

In Figure 1, we show some example snapshots from TexMol of the various visualizations it can produce from proteins and RNA.

Section 2 presents contemporary molecular visualization techniques. Section 3 explains the FCC representation of the molecule and its properties, including LOD hierarchies. Section 4 provides an overview of rapid imposter-based rendering and describes our specific imposter algorithms including dynamic LOD rendering. Section 5 analyzes TexMol’s size scalability and rendering performance measurements. Finally, Section 6 presents our conclusions and ideas for future work.

---

*This research was performed in the Center for Computational Visualization, http://www.ices.utexas.edu/CCV, and supported in part by NSF grants ACI-0220037, CCR-9988357, EIA-0325550 and a subcontract from UCSD 1018140 as part of the NPACI project, Interaction Environments Thrust.

1Ribo-nucleic acid

---

1Ribo-nucleic acid

1Ribo-nucleic acid
A vast amount of previous work has covered the representation and visualization of molecular surfaces. Originally proposed by Lee and Richards for use in biochemistry [21], the solvent accessible surface was first extended to computer graphics by Connolly [7, 6] and Max [24]. Much of this preliminary work, along with later extensions [29, 25, 1, 26, 3] focused on finding fast methods of triangulating the solvent-accessible surface. Two prominent obstacles in surface visualization are the correct handling of surface self-intersections to avoid visual artifacts and the high communication bandwidth needed when sending tessellated surfaces to the graphics hardware.

**Image Based Rendering Techniques** Large environments have traditionally been rendered using image based rendering techniques. These techniques usually suffer from popping artifacts and are rigid to changes in lighting. Hence, a renderer must strike the right balance between visual fidelity and speed of rendering. View dependent texture mapping, which we employ in TexMol, is commonly used for image-based rendering. In [10], the authors describe an efficient way of implementing such texture mapping through the traditional graphics pipeline. A survey of image based techniques for improving the rendering quality of traditional techniques is given in [5]. Large scale environment visualization through images was shown in [2, 28, 22, 15]. Better depth maintenance was shown in [8, 19]. In [12], an image based rendering algorithm is given which depends on lighting conditions. Using artistic techniques and multiperspective images, [16] present a method of improving the occlusions of objects during motion.

**Programmable Graphics Hardware** The recent interest in image-based rendering has been due in large part to increasingly powerful and increasingly programmable graphics hardware. Cg is a high level Graphics Processing Unit (GPU) programming language developed by NVIDIA that allows users to write custom programs for the vertex and fragment processors in commodity GPU's. Previous work for practical applications of GPU's [13] describes the texture-based sphere renderer which was adapted for use in TexMol.

Our main contribution for structural rendering is extending previous work on sphere rendering by proving new impostor-based algorithms for cylinders and helices. Our fast, general visualization technique overcomes the bandwidth and artifact problems associated with triangle-based structural rendering. Under impostor-based rendering, each complex geometric primitive is represented by a set of simpler geometric objects, such as a single quadrilateral, in contrast to the numerous triangles needed to tessellate complex curved primitives. In addition, primitive-primitive intersections under impostor-based rendering are correct on a pixel level with no extra computation, which differs from the expensive clipping algorithms needed to remove artifacts from intersecting sets of triangulations.

Figure 1: Different structures and functions at various LODs can be visualized interactively using TexMol. Figure (a) shows the chains in a rhinovirus colored by the residue they belong to. In Figure (b), we show a small ribosomal subunit at the residue level. Each residue is approximated by a primitive to provide more overall information to the user than that given by a set of atoms. We show the secondary structures of hemoglobin in Figure (c). The four hemes containing the iron atom is shown enclosed between the α helices. A combined volume and isosurface rendering of a virus capsid protein’s hydrophobicity is shown in Figure (d), with the hydrophobic regions shaded yellow and the hydrophilic regions shaded red.
3 Internal Representation

The Flexible Chain Complex (FCC) is our main hierarchical representation for molecular structure. For each chain in the molecule, the chain backbone is stored along with all the residues (amino acids or nucleotides), which are placed at the connection point to the backbone. A helix surrounded by the electron density around the backbone is shown in Figure 2. FCC allows the application to interactively update flexible dihedral angles along the backbone. TexMol also has support for manipulating the rotamer angles between two neighboring monomers in a protein backbone.

Physical properties of a molecule (e.g. the electron density, electrostatic potential, hydrophobicity, and surface curvatures) can be represented as functions of the atom centers along with the atoms’ properties. Hence the FCC provides a unique skeletal and an implicit volumetric format to efficiently represent structural and functional properties of a molecule. In this paper, we concern ourselves only to visualization and exploration of large FCCs.

The FCC can also contain pointers to volume files which explicitly represent properties of the molecule. The volume files are in two formats. Apart from the conventional color index format, we also use the older RGBA format to efficiently render pre-computed structures of the molecule. For example, we could have an RGBA dataset where the molecule is colored per atom according to the color of the residue to which it belongs. The A channel is used to show the dropoff in electron density as distance from the atom center increases.

3.1 Static level-of-detail

The FCC sorts the atoms and superstructures of a molecule into a four-tiered hierarchy which is derived from the biochemical hierarchy used in Protein Data Bank (PDB) files [31]. In this paper, all filenames associated with a molecule refer to the commonly used PDB filename. The following is a top-down view of the hierarchy.

- Chain - chain backbone information is stored
- Secondary structure - a Helix, a Turn, or a Sheet
- Residue - either an amino acid or a nucleotide
- Atom - lowest tier, each member is a single atom

The hierarchy is constructed from the bottom up, so that every member of the hierarchy, except chains, contains all relevant members from the level immediately below it. For example, each atom only contains itself, since atom is the lowest level of the hierarchy.

A residue contains all atoms that are a part of it, while a secondary structure contains all residues that are a part of it. Chains are a unique in that they only contain some of their component members, which will be explained in detail later in this section.

TexMol associates each level of the hierarchy with a geometric representation. An atom is represented by a single sphere with the atom’s van der Waal radius. A residue is represented by a minimal single bounding sphere that encloses all of its component atoms. A volume rendering of the atomic and residue level is shown in Figure 3 for a large ribosomal subunit (1JJ2.pdb). In Figure 9, we show the bacteriophage virus’ capsid proteins (1GW7.pdb) at two different levels using different visualization techniques.

For secondary structures, sheets and helices are represented by an appropriately-sized geometric sets of cylinders and helices. The orientation, length and radius of these are determined by a least mean square fitting of the central axis to all of the atom centers in either the helix or the sheet’s strand. The third type of secondary structure, turns, are simply represented by their component atoms and bonds along the chain backbone, since turns are usually short atom sequences that feature sharp angles. Not all residues are part of secondary structures, so we simply cluster these otherwise parent-less residues under an imaginary SS_NULL group.

Chains are represented by including only the atoms in the backbone of the chain. Hence, the chain level does not include any of the functional groups attached to the molecular backbone.

Once the hierarchy is created, the user may interactively change the LOD in the hierarchy by picking one of its tiers to visualize. Under static LOD, this selection is applied uniformly across the entire molecular complex (contrast this with dynamic LOD explained in Section 4.5).

4 Imposter Rendering

The models we encounter in molecular visualization, including the union of balls model, the ball and stick model, the secondary structure model, are composed of regular curved surfaces. The traditional method of rendering these surfaces is to triangulate the surface and render the triangle strips. Due to the large sizes of the data sets we are trying to render, triangulated curved surface rendering is either too computationally expensive or is plagued by visual artifacts.

Hence we introduce some new image-based rendering techniques that extend earlier work in sphere rendering. Our main idea comes from the fact that in the programmable graphics hardware, we are allowed to perform a depth replacement at the fragment level. This allows us to render a curved surface with some axis of symmetry without the need for triangulation. NVIDIA’s Cg Tu-
quadrilateral, with correct depth and normals under an orthographic
projection. In the perspective view, the depth turns out to be an ap-
proximation, but still retains good smoothness properties. We ex-
tend this work to include other primitives (Figure 4) and apply these
new primitives in our molecule visualization.

One key advantage of imposter-rendering is per-pixel accurate
shading within the resolution of the normal map. Although the
resolution of the normal map imposes some limits, the normal
map’s resolution may be increased arbitrarily based on the applica-
tion requirements. In contrast to imposter-based renderers, typical
molecular visualizers, such as RasMol and its derivatives, tessellate
their geometric primitives, which produces approximate lighting ef-
facts and tessellation-related artifacts. Another benefit of imposer-
rendering is its occurrence near the end of the rendering pipeline,
meaning it can be combined with software-based acceleration tech-
niques, such as occlusion culling and adaptive tessellation, to fur-
ter increase visualizer performance.

4.1 CPK model
The CPK model of a molecule is the geometric union of all of Van
der Waal radius spheres that correspond to an atom in the molecule.

4.1.1 Sphere rendering
Texture-based sphere rendering is shown in one of NVIDIA’s Cg
Tutorials [13]. We use their method to render spheres in TexMol.
The cylinder and helix renderers that will be described in Sections
4.2.1 and 4.3.1, respectively, are extensions of the sphere renderer.

4.2 Ball and Stick model
The ball and stick model is the geometric union of all the atoms
and bonds in a molecule, where each atom is represented by a Van
der Waal radius sphere and each bond is represented by a cylinder
that extends between the two atoms that form the bond. Atom radii
must be intentionally reduced from the values given in the PDB
file in order to make the bonds visible. While the ball and stick
model is otherwise very similar to the CPK model, the understated
atomic radii places more emphasis on the internal connectivity of
the molecule.

4.2.1 Cylinder rendering
The cylinder uses a single quadrilateral as the underlying geometric
primitive, along with three texture maps. Unlike a sphere, a cylin-
der is not rotationally invariant, so its corresponding quadrilateral
in clip space reflects the orientation of the actual cylinder in clip
space. In our algorithm, we need only one quad to obtain a per-
fected cylinder, but due to a compiler bug in Cg, we are forced to
use two quads. The algorithm for the vertex program and the frag-
ment program is given in Algorithm 1 and Algorithm 2. Our vertex
and fragment programs work in NVIDIA’s vp20 profiles, which is
supported in GeForce FX cards and beyond.

In each of the algorithm descriptions, we have tried to be
as brief as possible and also use common abbreviations due
to space constraints, without losing any information. ES, OS refer to
EyeSpace and ObjectSpace respectively. MV, MV1, MVP, Proj, MVPI are the OpenGL’s ModelView,
ModelViewInverse, ModelViewProjection, Projection and the
ModelViewProjectionInverse matrices respectively.

By using the vertex and fragment programs in Algorithm 1 and
Algorithm 2, we create a single front sided cylinder where the two
cylinder bases are not well defined. This design increases perfor-
mance with no loss in visual quality, since the cylinder bases are
always hidden inside spheres in the stick and ball model.

In Algorithm 3, we create cylinders with defined bases by en-
suring that we have the correct opacity at the ends of the cylinder.
This algorithm requires the coordinates of one end point, the axis
of the cylinder, and the MVPI matrix. Due to space constraints, we
present only the subset of the program that needs to be inserted into
the Stick Fragment Program given in Algorithm 2. Figure 4 (b) and (c) depict how quads are transformed in the programmable graphics card into cylinders with proper depth and lighting. Algorithm 1 is still used as the vertex program.

Algorithm 1 Stick vertex program

1: Inputs are: [P1, P2, MV, radius, offset, MVI, MVP, Proj]
2: \[ OS[\text{axis}] \leftarrow P2 - P1 \]
3: \[ ES[\text{axis}] \leftarrow \text{normalize}(MV \ast OS[\text{axis}]) \]
4: \[ h \leftarrow |ES[\text{axis}]|/\text{radius} = \sqrt{1 - |ES[\text{axis}]|^2} \]
5: \[ ES.xy[\text{unitAxis}] \leftarrow \text{normalize}([ES.x, y, -ES[\text{axis}].x, 0, 0]) \]
6: \[ ES[\text{axis}] = [ES.x, y, \text{unitAxis} + offset \ast radius] \]
7: if \( P \) associated with \( P1 \) then
8: if \( ES[\text{axis}].z < 0 \) then
9: \[ \text{offset} \leftarrow [ES.xy].offset \ast h \ast radius \ast ES[\text{axis}].z \]
10: else
11: \[ \text{offset} \leftarrow [ES.xy].offset \ast h \ast radius \ast ES[\text{axis}].z \ast -1 \]
12: end if
13: else
14: if \( ES[\text{axis}].z < 0 \) then
15: \[ \text{offset} \leftarrow [ES.xy].offset \ast h \ast radius \ast ES[\text{axis}].z \ast -1 \]
16: else
17: \[ \text{offset} \leftarrow [ES.xy].offset \ast h \ast radius \ast ES[\text{axis}].z \]
18: end if
19: end if
20: \[ ES[\text{offset}] \leftarrow MV1 \ast ES[\text{offset}] \]
21: if \( P \) associated with \( P1 \) then
22: \( \text{outP} \leftarrow P1 \)
23: else
24: \( \text{outP} \leftarrow P2 \)
25: end if
26: \( HPOS \leftarrow MVP \ast (\text{outP} + \text{offset}) \)
27: \( \text{radius} \leftarrow \text{radius} \ast h \)
28: \( [z, w] \leftarrow [\text{MVP[2]}, \text{MVP[3]}, \text{outP}] \)
29: \( [z, w].offset \leftarrow \text{radius} \ast [\text{Proj[2]}, \text{proj[3]}].z \)
30: Output: \([z, w][\text{center}, \text{offset}], HPOS, \text{LightVector}, \text{Normal}, \text{EndPoints}, \text{normalized xy position}] \)

Algorithm 2 Stick fragment program

1: Inputs are: [normalized xy position (xy), LightVector, Color, Ambient light]
2: \( \text{Normal} \leftarrow \text{Lookup}([\text{normalMap}, \text{xy}]) \)
3: \( \text{Depth} \leftarrow \text{Lookup}([\text{depthMap, xy}]) \)
4: \( \text{OutColor} = \text{CalculateOutputColor()} \)
5: \( [z, w] \leftarrow \text{Depth} \ast [z, w].offset + 1 \ast [z, w].\text{center} \)
6: \( \text{OutDepth} \leftarrow z/w \ast 0.5 + 0.5 \)
7: Output: [color, opacity, depth]

4.3 Secondary structure representation

We visually represent helix and sheet secondary structures using imposter-based helices and cylinders. Turns are represented by their component atoms and bonds along the backbone since they usually consist of only a handful of atoms.

4.3.1 Helix rendering

Helices are composed of two quadrilateral primitives and are an extension to cylinder rendering. We present the fragment program in Algorithm 4, which is the extension which needs to be added to the simpler cylinder fragment program in Algorithm 3. For this fragment program, we need the additional input parameter of a crossRefVector, which is a vector from one end point of the helix on the axis to any particular point on the helix in object space. Another input needed is an opacity map (a bitmap containing a wrapped diagonal band of the required thickness), which generates the shape of the helix via alpha values. Figure 4(d), shows the two cylinders from Figure 4(c) under the new helix opacity mapping. A slightly different viewpoint is used for the helices to show the correct per-pixel intersections and shading. Figure 5 shows the myoglobin molecule with its helices represented using imposters.

Algorithm 3 Cylinder fragment program

1: Inputs are: [P1, axisDCs, length, MVP1, IN.xy]
2: \( ndcPos \leftarrow (IN.x, IN.y, z/w, 1) \)
3: \[ OS[P] = MVP1 \ast ndcPos \]
4: \( y \leftarrow \text{axisDCs}((|OS[P] - P1).xyz) \)
5: if \( y < 0 \) then
6: \( \text{opacity} \leftarrow 0 \)
7: else if \( y > \text{length} \) then
8: \( \text{opacity} \leftarrow 0 \)
9: else
10: \( \text{opacity} \leftarrow 1 \)
11: end if

Algorithm 4 Helix fragment program

1: Inputs are: [refVector, crossRefVector, opacityMap, pitch]
2: if \( y \in \text{Range()} \) then
3: \( V \leftarrow \text{normalize}([\text{IN}.x], \text{IN}.y, \text{IN}.z, \text{axisDCs}) \)
4: \( x \leftarrow \cos^{-1}(V.(\text{refVector})) \)
5: \( \text{side} \leftarrow V.\text{crossRefVector} \)
6: if \( \text{side} < 0 \) then
7: \( x \leftarrow 2 \pi - x \)
8: end if
9: \( y \leftarrow \text{mod}(y, \text{pitch}) \)
10: \( \text{opacity} \leftarrow \text{lookup}([\text{opacityMap}, \text{z}, \text{y}]) \)
11: end if

4.3.2 Sheet rendering

Sheets are composed of multiple strands. One method of rendering a sheet is to render the set of strands as cylinders. Hence we use our cylinder rendering when visualizing sheets. The two \( \beta \) sheets of a snake toxin are represented by sets of differently colored cylinders in Figure 6. Multiple cylinders per strand and a color gradient could be used to enhance the folding of the sheet and the strand directions.

4.4 Function-on-surface rendering

We implement embedded surface rendering by extending the previous GPU programs. It is often useful to study the variation of functions on molecular surfaces. These properties can be visualized by accessing two additional custom texture maps in the fragment program. The user can set one texture to the volumetric function, while assigning to the other texture a transfer function map which allows the user to interactively modulate the function being mapped onto the impostor model. One could also implement implicit functions using the fragment program instead of an explicit volumetric grid. In Figure 8, we show the hydrophobicity function being mapped onto the union of balls model of the hemoglobin molecule.
4.5 Dynamic level-of-detail

Dynamic LOD uses the static LOD hierarchy to change regions of the molecule based on the spatial occupancy of these regions from the current viewpoint. Dynamic LOD is currently implemented with bounding spheres on all levels of the hierarchy to limit visually distracting popping when one region switches its LOD.

But the preprocessing step needed for this algorithm, even for thousands of centers was very high.

When dynamic LOD is active, molecular regions that occupy fewer pixels on screen are represented with a coarser level in the static hierarchy. This allows the user to see atom-level detail by zooming in, while also being able to capture the overall molecular structure by zooming out. Additionally, realtime scenes with many molecules interacting with one another benefit from dynamic LOD by automatically reducing the geometric complexity of a scene from far away viewpoints, while maintaining the power to explore the scene in detail from other viewpoints.

4.6 Synchronized multi-view visualization

We define a molecular information view (MI view) to be either a structural representation, a functions-on-surface representation, or a volumetric representation of a molecule. TexMol provides the ability for split-screen visualization that juxtaposes multiple MI views.
<table>
<thead>
<tr>
<th>Mol</th>
<th>NumAtoms</th>
<th>NumRes</th>
<th>Mem (M)</th>
<th>LOD</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hem</td>
<td>4770</td>
<td>574</td>
<td>27</td>
<td>Atom</td>
<td>28.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Residue</td>
<td>21.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Chain</td>
<td>85.2</td>
</tr>
<tr>
<td>Rib</td>
<td>98543</td>
<td>6577</td>
<td>79</td>
<td>Atom</td>
<td>7.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Residue</td>
<td>9.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Chain</td>
<td>42.6</td>
</tr>
<tr>
<td>Mic</td>
<td>1210410</td>
<td>78210</td>
<td>643</td>
<td>Atom</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Residue</td>
<td>3.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Chain</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Table 1: Performance results for rendering in 800x600 resolution, full screen mode for Hemoglobin (Hem), Ribosome (Rib) and the Microtubule (Mic).

simultaneously. We also allow both synchronous and asynchronous viewing operations to be performed on each view, in effect separating the data model from the viewing model.

5 Results and Discussion

We will first discuss the performance results for running TexMol on a high-end workstation with commodity graphics hardware. Next, we will briefly summarize the various features of the software.

5.1 Performance

TexMol’s performance was measured on a dual Pentium III Xeon 800MHz machine with one gigabyte of memory and an NVIDIA GeForce4 Ti 4200 AGP8X for its GPU. Only one of the dual CPU processors was used during testing. Table 1 shows performance for the hemoglobin, ribosome, and microtubule datasets. The hemoglobin (1A00.pdb) and the large ribosomal subunit (1JJ2.pdb) are taken from the Protein Data Bank [31]. The microtubule data set was donated by Dave Sept and Nathan Baker. The second and third columns list the number of atoms and residues, respectively. The fourth column lists the peak CPU-side memory usage for each experiment. Since a given dataset is always processed the same way to generate the static LOD hierarchy, from which all static LODs can be derived, the CPU-side memory usage is the same regardless of which LOD is selected for rendering. The time taken to parse the files to a hierarchy is considered as a preprocessing step and is not considered here. When we are using dynamic LOD rendering, the time taken to determine the cut in the tree to render is negligible compared to the rendering times.

For each trial, the given molecule was rendered across three LODs (fifth column): atom level, residue level, and chain level. The final column lists the frames-per-second performance of the trial. All trials were performed at 800x600 resolution using TexMol’s full-screen mode.

The atom LOD sends the most number of geometric primitives to the GPU, so cases where the atom LOD performs worst are vertex limited. We expect this case to occur for extremely dense molecules with high atom counts. The residue LOD provides fewer geometric primitives to the GPU, but residue representations (a conservative bounding ball) take up more screen space than their component atoms. Thus, more fragments are generated in the GPU fragment processor, so cases where the residue LOD perform worst are fragment limited. Finally, the chain LOD reduces both vertex and fragment load from the atom LOD by 1) rendering only atoms and bonds along the backbone and 2) using primitives that are the size of atoms rather than larger bounding balls. Note, however, that the chain LOD renders more than one geometric primitive per residue, so it still has greater vertex complexity than the residue LOD, which renders exactly one geometric primitive per residue.

For the atom LOD, the hemoglobin visualization performs fastest. Since hemoglobin is a relatively small molecule, it stands to reason that it is not vertex limited. On the other hand, the medium-sized ribosome shows signs of being vertex limited, since its performance improves slightly under the residue LOD. Finally, the microtubule, which contains the most vertex information, shows a speedup of about 90% when switching from atom LOD to residue LOD. The slowdown of the microtubule chain LOD is a result of the vertex-limited nature of the rendering, since three atoms are still being drawn per residue. Memory usage for the two larger molecules scales linearly with the size of the molecule, while the hemoglobin memory usage is almost entirely due to the operating overhead of the program (about 20 M). The bacteriophage shown in Figure 9 renders at a peak rate of around 1.2 fps.

5.2 Visualizer features

This section summarizes TexMol’s various features. First, TexMol used fast imposter-based rendering to render the structural molecular shape at various LODs (Figures 1(a-c), 5, 6). TexMol also renders volumes (using hardware assisted 3D texture maps) independently (Figures 1(d), 3) or in tandem with structure. Structure-volume rendering modes include embedded rendering (Figure 2), function-mapping, and synchronized multi-view (Figure 8). Curvatures and isosurfaces are rendered with traditional geometry. Finally, TexMol can display and animate large molecules at high resolution that were previously intractable. Currently, the massive microtubule complex (Figure 7) animates at a peak rate of 4 fps, and we hope future optimizations lead to interactive rates.
6 CONCLUSIONS AND FUTURE WORK

TexMol’s ability to perform a wide range of structural and volumetric rendering concurrently, combined with its aggressive use of programmable graphics hardware, distinguish it from existing molecular visualizers. We hope TexMol will effectively serve the biology community in its ongoing research.

In our continuing work, we intend to incorporate visibility and culling techniques to further increase rendering performance. Our ultimate goal is to visualize the 1.2 million atom microtubule dataset at an interactive rate. We are also working on a GPU technique that increases the visual fidelity of volumetric rendering by exploiting symmetry, which is inherent in many large biomolecules, like icosahedral viruses. Future work will also include experimenting with the synchronous multi-view framework in order to discover new ways to visualize the inherent links between molecular structure and molecular function.

Acknowledgements The microtubule dataset was provided by Dave Sept and Nathan Baker of Washington University in St. Louis. Several of the Virus datasets were provided by Charlie Brooks, Jack Johnson, and Vijay Reddy of the Scripps Research Institute.

REFERENCES