Rendering techniques for molecular visualization on mobile devices

B. J. Larson

Sunset Lake Software LLC
3030 Laura Lane, Suite 120, Middleton, WI, USA, larson@sunsetlakesoftware.com

ABSTRACT

The processing power and graphics capabilities of mobile devices have grown at such a rate that tasks which once required a desktop workstation can now be performed on a handheld computer. One area enabled by this is mobile molecular visualization. An open source application has been written to display 3-D structures of molecules on Apple, Inc.'s iOS devices. These molecules can range in size from small compounds to larger theoretical structures for molecular nanotechnology. The rendering techniques used to enhance the display of these structures will be described, as well as the optimizations applied to make these techniques performant on handheld devices.

Keywords: molecular visualization, 3-D graphics, mobile devices, iPhone, iPad

1 INTRODUCTION

The computational power in handheld devices and tablet computers has advanced rapidly in recent years, with these devices becoming capable of tasks once reserved for desktop workstations. In particular, 3-D graphics hardware in modern mobile devices is making possible applications like molecular visualization.

The ability to render and manipulate realistic depictions of molecular structures on relatively low cost, highly mobile computing devices is attractive for use in education, enhancing research presentations, or as a convenient reference.

To enable this, a free open source molecular visualization application called Molecules has been written for Apple, Inc.'s iOS platform to run on their popular iPhone, iPad, and iPod touch devices. Its source code is available for download and reuse under the BSD license.[1]

The application can visualize 3-D structures of molecules drawn from the RCSB Protein Data Bank [2] or NCBI's PubChem. A glucose molecule from PubChem is shown in a ball-and-stick visualization mode in Figure 1. [3] Additionally, custom molecular structures in the PDB format can be directly loaded into the application for display. Figure 2 shows an example of a theoretical neon pump designed by K.E. Drexler and R.C. Merkle. [4]

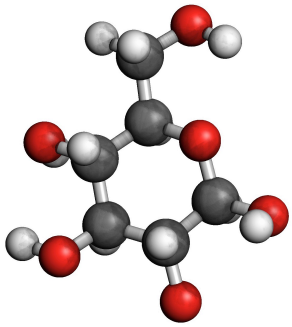

Figure 1: A ball-and-stick representation of glucose from Molecules, using the 3-D structure definition file computed by NCBI PubChem. [3]

Molecules produces realistic renderings of molecular structures using techniques first described by Tarini, *et al.*[5] This approach creates high-quality renderings through the use of procedurally generated sphere and cylinder impostors, combined with ambient occlusion lighting to simulate real-world illumination.

The processes described in that research were developed for high-performance desktop computers, with significantly different characteristics from modern mobile phones and tablet computers. Molecules was developed for the iOS operating system, which uses the industry standard OpenGL ES interface to present 3-D graphics. OpenGL ES is the mobile variant of the OpenGL standard present on the desktop, and it lacks some of the capabilities of its desktop-bound predecessor.

As a result of this, significant modifications needed to be made in order for the elements described by Tarini, *et al.* to be functional on handheld computers.

2 PROCEDURAL IMPOSTORS

One of the most challenging tasks for 3-D graphics on computers is to represent smooth, curved surfaces. 3-D objects are typically composed of a series of triangles that are placed in space and processed to create a 2-D representation of 3-D geometry. Curved surfaces can only be approximated through the use of many flat polygonal faces. The more faces used, the smoother the object, but this also places a greater burden on the

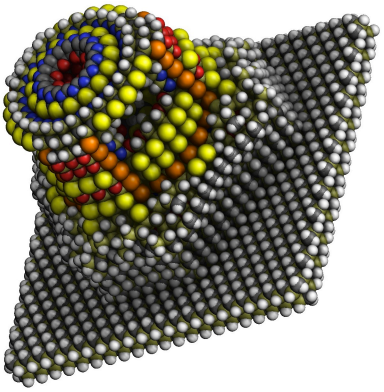

Figure 2: A spacefilling rendering from Molecules of a theoretical neon pump designed by K.E. Drexler and R.C. Merkle. [4]

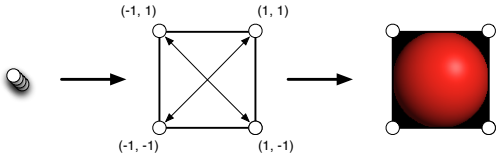

Figure 3: The drawing of a sphere impostor.

graphics processing unit (GPU). For structures containing many smooth objects, like the spheres in a molecular model, this can reduce the rendering speed on even powerful GPUs to noninteractive framerates.

Rather than use complex geometry to simulate smooth spheres or cylinders, procedural impostors can be used to provide better 2-D representation of these 3-D objects. Flat polygons are provided that always face the viewer, within which each pixel is colored as if a smooth solid object was behind that polygon, with the polygon merely being a window into that object. A representation of this process can be seen in Figure 3.

Modern mobile GPUs support the creation of custom programs, called shaders, that can perform calculations, manipulate geometry, and dictate the color of each pixel output to the screen. Using an appropriate shader, each pixel in an impostor can be filled in as if it were from a lit sphere or cylinder, producing objects as smooth as can be represented on the device screen using only a tiny fraction of the geometry that would otherwise be required for this.

In the simplest case, spheres are generated by providing four vertices to the GPU which will define two triangles. These vertices are each displaced by a vertex shader in such a way as to present a square facing right at the user. Within this square, a fragment shader is run for each pixel to calculate a color at that point corresponding to how a sphere would be lit. Both diffuse and specular lighting is calculated based on the dot product of a preset light direction and the normal of the virtual sphere surface at that point.

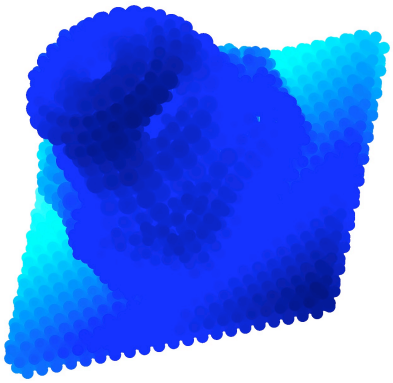

Figure 4: An example of a depth texture rendered for a structure.

Cylinder impostors are more complex, but the same basic principles apply. Four vertices are transformed into a rectangle that faces the user. This rectangle is rotated to match the orientation of the cylinder, and one of the ends is extended to account for the upwards curvature at the end facing away from the user. Similar raytracing calculations are performed for the cylinder as for the sphere, but the calculation of a lighting normal at each point now depends on the orientation of the cylinder, unlike the constant shape of the sphere.

3 PER-PIXEL DEPTH TESTING

While these impostors create smooth spheres and cylinders in a performant manner, they do introduce some complexity when two or more of these objects overlap. With traditional 3-D geometry, this would be handled automatically by the depth testing hardware in the GPU. However, these impostors don't actually exist, so custom depth tests need to be performed at each pixel.

For desktop GPUs, commands exist to write out custom depth values for each pixel within a shader, but these commands are missing on mobile GPUs. Therefore, a custom rendering pass was created where all objects are rendered once, with the color values that are output corresponding to the calculated depth of the impostor object at that pixel. The color-encoded depth values are used with a special blending mode so that only the depth for the frontmost object rendering to a pixel is used for that pixel. An example of the result of this depth image generation is shown in Figure 4.

These depth color values are then read in as part of the final impostor drawing pass, and they are used

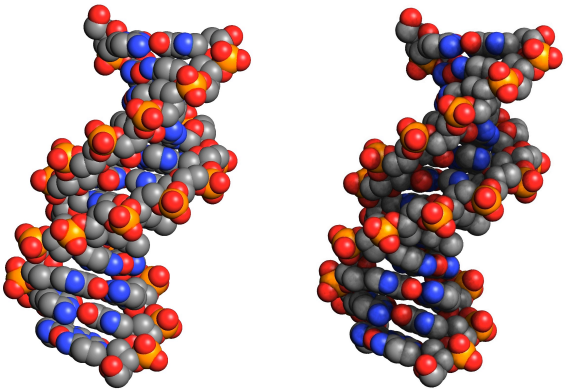

Figure 5: A spacefilling model of B-DNA [6] rendered without (left) and with (right) ambient occlusion lighting.

to determine whether or not a pixel is drawn to the screen. This simulates the custom per-pixel depth testing capabilities of desktop GPUs in mobile devices and makes possible molecular visualizations with intersecting spheres and cylinders.

4 AMBIENT OCCLUSION LIGHTING

Ambient occlusion lighting is a technique that can produce a more realistic depiction of an object by simulating the way that ambient light hits complex objects. Conceptually, a hemisphere is drawn out from each point on a surface, and the percentage of that hemisphere that is blocked by the presence of another part of the object is determined. That point on the surface is then darkened proportionally in response to the amount of the hemisphere that is blocked, because a physical object would have less ambient light hitting that point.

As shown in Figure 5, this doesn't just add realism to a rendering of a molecular model, it also helps to convey more information about the surface structure of that molecule. Folds and crevices can be seen clearly in one frame, where before they could only be determined by repeatedly rotating the model.

Performing these kinds of calculations for every rendered frame would be too computationally expensive, so this is determined once for a given molecular model and the resulting illumination pattern is stored in a texture for later reference. Figure 6 shows the internal structure of one of these textures, where separate rectangles contain ambient illumination intensities that are mapped to locations on an atom's surface. Figure 7 demonstrates the result of only applying the ambient occlusion lighting to the model from Figure 5.

Rather than perform ray tracing calculations at each surface point to generate such a lighting map, a hardware-accelerated process is used to approximate this lighting.

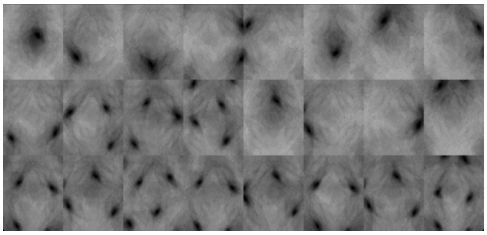

Figure 6: The internal structure of an ambient occlusion texture.

The molecular model is rotated into a series of orientations, points on the surface that are visible from that orientation are determined, and those points are brightened. This leads to recessed features being darkened, because they will be hidden behind other portions of the object more frequently.

Visibility is determined through the use of the previously described depth texture. For each orientation tested, a depth texture is rendered, and a separate shader is employed to render one pass of the ambient occlusion mapping texture. The 3-D location corresponding to each point on a portion of the ambient occlusion texture is calculated and the Z value of that point is compared to the depth texture value at the X, Y coordinate for that point. If the Z value matches the depth at that point, it is rendered with a luminance equal to the inverse of the number of orientations being tested. If it is lower than that depth, it is rendered as black. An additive blend mode is then used to accumulate these ambient occlusion brightnesses from the various orientations to produce the final lookup texture.

This ambient occlusion texture is constant for the entire duration that the user is viewing and manipulating a given structure, because these molecular models are static. During the color rendering of the procedural impostors, the light intensity at a pixel is scaled by the value read from the ambient occlusion texture for that point on the surface of the sphere or cylinder.

5 OPTIMIZATIONS FOR MOBILE GPUS

In addition to the previously described depth textures, other modifications were made to the technique described by Tarini, *et al.* in order for this rendering to take place at an interactive framerate on mobile devices.

The most significant change was the use of a rendering prepass that takes place just before the depth texture generation and final screen display steps. In this prepass, flat objects covering an area known to be opaque within the sphere impostors are rendered one sphere radius distance away from the viewer. This is done while the depth buffer (on the hardware, not the depth texture) is made writeable.

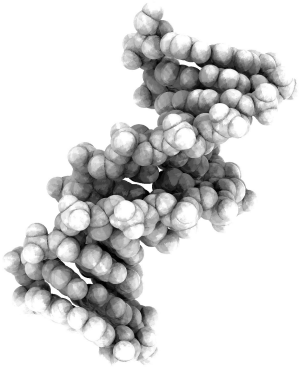

Figure 7: A rendering of just the ambient occlusion lighting component for a spacefilling model of a B-DNA structure. [6]

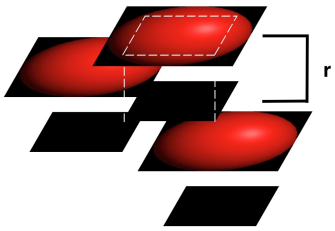

Figure 8: The use of opaque elements rendered below the impostors.

The depth buffer is then made read-only for the depth texture and final rendering passes. Most mobile GPUs are tile-based deferred renderers, and the placement of these opaque objects helps the GPU to ignore portions of objects that will never be visible under any circumstances. This process is illustrated in Figure 8. On several iOS devices, this led to an over sixfold improvement in rendering speed.

Although all diagrams to this point have illustrated the sphere impostors as coming from squares, octagons were used instead within the final version of this application. The corners within a sphere impostor never have any color within them, so using an octagon can reduce the number of pixels needed to be drawn to the screen.

The vertex and fragment shaders used for all rendering stages were profiled and optimized for the capabilities of mobile hardware. Floating point precision was reduced in specific points to speed up processing. Branching instructions were replaced with step functions, because the former can be costly on mobile GPUs. Calculations were moved from fragment shaders to vertex shaders where possible.

With these optimizations, the molecular structure depicted in Figure 5 renders at 43 ms / frame (23 FPS) on an iPhone 4, 4.9 ms / frame on an iPhone 4S (60+ FPS), and 25 ms / frame (39 FPS) on a 3^{rd} generation iPad.

6 DISCUSSION

The use of procedural impostors to generate smooth spheres and cylinders, combined with ambient occlusion lighting, produces renderings of molecular structures that both have visual appeal and convey additional depth information. With some modifications, the process described by Tarini, *et al.* can be applied to mobile devices.

While this particular application was developed for iOS, the general principles described here could be extended to other mobile platforms, due to the similarities in GPU design and programming interfaces. The source code for the application has been made available for others to modify and use under a BSD license.

7 ACKNOWLEDGMENTS

The author would like to thank Rachel Kramer Green, Wayne Townsend-Merino, and Peter Rose of the RCSB Protein Data Bank for enabling access to that repository. Similarly, the author would like to acknowledge Rana C. Morris and coworkers at NCBI User Services for their help in accessing NCBI's PubChem. Finally, the author would like to recognize the suggestions and other help provided by engineers at Apple, Inc., which were a significant help in the development of this application.

REFERENCES

[1] B.J. Larson. Molecules. http://www.sunsetlakesoftware.com/molecules.

[2] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The Protein Data Bank. Nucleic Acids Research 28, 235-242 (2000).

[3] National Center for Biotechnology Information. PubChem Compound Database; CID=206. http://pubchem.ncbi.nlm.nih.gov/summary/summary.cgi?cid=206.

[4] K.E. Drexler and R.C. Merkle. Simple Pump Selective for Neon. http://www.imm.org/research/parts/pump/.

[5] M. Tarini, P. Cignoni, and C. Montani. Ambient Occlusion and Edge Cueing to Enhance Real Time Molecular Visualization. IEEE Transactions on Visualization and Computer Graphics 12, 1237-1244 (2006).

[6] H.R. Drew, R.M. Wing, T. Takano, C. Broka, S. Tanaka, K. Itakura, R.E. Dickerson Structure of a B-DNA dodecamer: conformation and dynamics. Proc.Natl.Acad.Sci.USA 78: 2179-2183 (1981).