MobiRT: An Implementation of OpenGL ES-based CPU-GPU Hybrid Ray Tracer for Mobile Devices

Jae-Ho Nah[*] Yoon-Sig Kang, Kwang-Jo Lee, Shin-Jun Lee, Tack-Don Han, Sung-Bong Yang
Yonsei University

Abstract

Three-dimensional user interfaces on mobile devices are increasingly important. For more realistic three-dimensional visualization on mobile devices, we present the implementation of an OpenGL ES-based CPU-GPU hybrid ray tracer. This ray tracer exploits the availability of CPU and GPU architectures to fully support reflection, refraction, hard shadows, and dynamic scenes. To the best of our knowledge, our ray tracer is the first to demonstrate full Whitted ray tracing of dynamic scenes using OpenGL ES.

CR Categories: Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Ray tracing

Keywords: ray tracing, global illumination, OpenGL ES

1 Introduction

Currently, three-dimensional (3D) user interfaces are a key application of visualization on mobile devices [Capin et al. 2008]. In particular, the increasing popularity of touch-screen phones has accelerated the necessity of 3D user interfaces. However, 3D user interface design in mobile environments is difficult due to complex shader programming and low rendering performance. We believe that ray tracing [Whitted 1980] can be an excellent solution for 3D user interfaces for several reasons. First, ray tracing naturally supports global illumination effects such as reflection, refraction, and shadows. Thus, ray tracing provides not only accurate physical effects but also greatly simplified shader programming for these effects. Second, ray tracing performance is inversely proportional to pixel size, so it has advantages for small screen mobile devices. Third, ray tracing supports flexible primitive types such as triangles, boxes, spheres, and splines. Therefore, more elaborate 3D objects can be created by using fewer primitives. In this paper, we propose an OpenGL ES [Ope b]-based mobile ray tracer called MobiRT. The MobiRT utilizes the CPU and GPU architectures to support both full Whitted effects (reflection, refraction, and hard shadows) and dynamic scene rendering. First of all, the CPU constructs kd-trees and generates primary rays. Then, the GPU executes kd-tree traversal, intersection tests, shadow ray generation, Phong shading, and texture mapping using the data transferred from the CPU. After that, the CPU recursively generates secondary rays using the hit point data transferred from the GPU. Figure 1 illustrates the architecture of the MobiRT.

[*]e-mail: jhnah@msl.yonsei.ac.kr

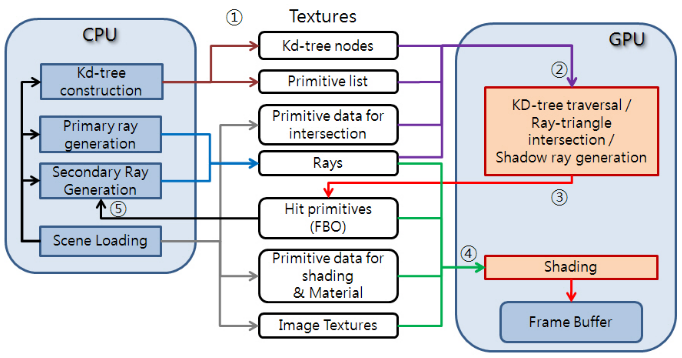

Figure 1: *The MobiRT architecture*

2 Problems to Solve

There are three problems in implementing an OpenGL ES-based ray tracer. First, mobile GPUs have much poorer performance than desktop GPUs, so the implementation of real-time ray tracing is difficult. Second, writing to video RAM (VRAM) is limited to 8-32 bits for each pass because the minimum number of render buffers (GL_MAX_RENDERBUFFER_SIZE) in the OpenGL ES 2.0 specification is only one [Munshi et al. 2008]. Hence, management of the ray tree is limited on the GPU. Third, the number of textures in the entire scene is constrained by the number of texture units in the GPU (GL_MAX_TEXUTRE_IMAGE_UNITS). This is because ray tracing requires access of entire scene data. In addition, the use of data textures for acceleration structures (AS), geometry data, and material data decrease available image textures. Thus, effective texture management is required.

3 Solutions

We propose solutions for these three problems. First, we utilize both the CPU and GPU to overcome poor performance of mobile GPUs. In animated ray racing, performance for both ray tracing and AS updating is important [Wald et al. 2009]. Thus, we assign AS updating to CPU and ray traversal to GPU. Of the various ASs, we use the kd-tree because of its fast traversal performance. This kd-tree is rebuilt from scratch every frame. Thus, we chose a binned-SAH (surface area heuristic) build [Shevtsov et al. 2007] for fast construction. When the construction is complete, the kd-tree data is converted into texture data and transferred to the GPU for ray traversal. After that, kd-tree traversal is performed on the GPU. We use a short-stack algorithm [Horn et al. 2007] to keep the traversal stack in register, not in RAM.

Second, we propose using the compact 32-bit output format to overcome the limitation of memory writing on the GPU. This output format consists of two kinds of data. When the shader kernel executes ray traversal, this kernel outputs hit primitive index (24 bit) and shadow results (8 bit). In contrast, when the shader kernel executes shading, the output is RGBA color (32 bit). The first output format for hit and shadow results supports up to 16.7 M primitives and eight light sources. If the bit width of shadow results is set to 16 bits, the maximum number of light sources increases to 16, and the maximum number of primitives decreases to 65 K. If a ray is not intersected with any primitives, the hit primitive index is 0. After the GPU finishes a ray traversal pass, these hit and shadow results are transferred to the CPU using a frame buffer object (FBO). The CPU then calculates hit points, normals, and texture coordinates. This calculated information is used for both secondary ray generation on the CPU and shading on the GPU. In the MobiRT architecture, due to ray tree management on the CPU and the compact 32-bit output format on the GPU, there are only 32 bits of writing data to VRAM for each pass.

Third, we propose a novel image texture integration method to increase the number of available textures. In this method, 16 local textures not exceeding 512x512 resolution are integrated into one global texture with 2048x2048 resolution. To support variable size textures including non-square textures, the widths and heights of each texture are stored in arrays and transferred to the GPU using glUniformMatrix. The texture mapping kernel then recalculates the actual texture coordinate using the local texture number in a global texture.

4 Results

We used AMD OpenGL ES Emulator 1.4 to evaluate our approach. All tests were performed on a 2.9 GHz AMD Athlon-X2 with 2 GB DDR2 RAM and NVIDIA Geforce 9800GT. We used a single thread for the kd-tree construction.

We used three scenes as the benchmark. The first scene is the toaster with 11 K triangles and six image textures. We set the material of the floor to reflective. The second scene is the marbles with 8 K triangles and 10 textures. This scene consists of moving texture-mapped spherical objects. The last scene is the fish with 1.4 K triangles. We set the material to refractive. We used a single light source in all scenes.

Figure 2 shows that the MobiRT fully supports Whitted effects and texture mapping. First, the toaster scene shows reflection and shadows. Second, the marbles scene shows that the MobiRT can manage image textures more than eight (=GL_MAX_TEXUTRE_IMAGE_UNITS). Third, the fish scene shows refraction. According to table 1, rendering performance was 12 - 48 frames per second (FPS), and it was inversely proportional to the number of rays. We expect that the MobiRT will show 1 - 5 FPS on real mobile devices.

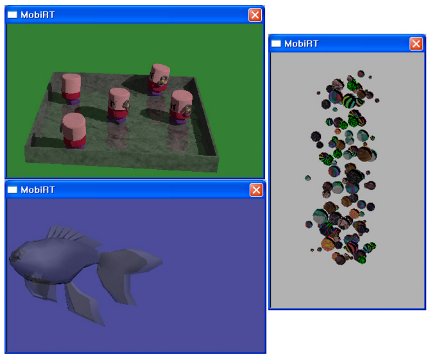

Figure 2: *Benchmark scenes : (top left) Toaster, (right) Marbles, (down left) Fish*

Table 1: *Benchmark results on the Open GL ES emulator (frames per second)*

Scene (resolution)	Toaster (400x240)	Marbles (240x400)	Fish (400x240)
ray casting	24	26	48
+shadow	20	N/A	N/A
+1-bounce reflection/refraction	15	N/A	26
+2-bounce reflection/refraction	12	N/A	20

5 Conclusions

We proposed an OpenGL ES-based CPU-GPU hybrid ray tracer for mobile devices. It supports animated ray tracing with full Whitted effects, and we demonstrated these features using the OpenGL ES emulator. In future studies, we think the use of OpenCL [Ope a] will provide faster and more efficient ray tracing.

Acknowledgements

This work was supported by Samsung Electronics Co., Ltd. The Toaster and Marbles animations are courtesy of Utah 3D Animation Repository. The Fish scene comes from the Arauna Ray Tracing.

References

Capin, T., Pulli, K., and Akenine-Moller, T. 2008. The state of the art in mobile graphics research. *IEEE Computer Graphics and Applications 28*, 4, 74–84.

Horn, D. R., Sugerman, J., Houston, M., and Hanrahan, P. 2007. Interactive k-d tree GPU raytracing. In *Symposium on Interactive 3D graphics and games (I3D '07)*, ACM, 167–174.

Munshi, A., Ginsburg, D., and Shreiner, D. 2008. *OpenGL ES 2.0 Programming Guide*. Addison-Wesley Professional.

OpenCL. `http://www.khronos.org/openglcl/`.

OpenGL ES. `http://www.khronos.org/opengles/`.

Shevtsov, M., Soupikov, A., and Kapustin, A. 2007. Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2007) 26*, 3 (Sept.), 395–404.

Wald, I., Mark, W. R., Gunther, J., Boulos, S., Ize, T., Hunt, W., Parker, S. G., and Shirley, P. 2009. State of the art in ray tracing animated scenes. *Computer Graphics Forum 28*, 6, 1691–1722.

Whitted, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6 (June), 343–349.