Unsolved Problems and Opportunities for High-quality, High-performance 3D Graphics on a PC Platform

David B. Kirk
NVIDIA
1226 Tiros Way
Sunnyvale, CA 94086, USA
dk@nvidia.com

ABSTRACT

In the late 1990's, graphics hardware is experiencing a dramatic board-to-chip integration reminiscent to the minicomputer-to-microprocessor revolution of the 1980's. Today, mass-market PCs are beginning to match the 3D polygon and pixel rendering of a 1992 Silicon Graphics Reality Engine™ system. The extreme pace of technology evolution in the PC market is such that within 1 or 2 years the performance of a mainstream PC will be very close to the highest performance 3D workstations. At that time, the quality and performance demands will dictate serious changes in PC architecture as well as changes in rendering pipeline and algorithms. This paper will discuss several potential areas of change.

A GENERAL PROBLEM STATEMENT

The biggest focus of 3D graphics applications on the PC is interactive entertainment, or games. This workload is extremely dynamic, with continuous updating of geometry, textures, animation, lighting, and shading. Although in other applications such as Computer-Aided-Design (CAD), models may be static and retained mode or display list APIs may be used, it is common in games that geometry and textures change regularly. A good operating assumption is that everything changes every frame. The assumption of pervasive change puts a large burden on both the bandwidth and calculation capabilities of the graphics pipeline.

GEOMETRY AND PIXEL THROUGHPUT

As a baseline, we'll start with some data and cycle counting of a reasonable workload for an interactive application. PC graphics hardware is capable of this throughput. As an example, this is a bandwidth analysis of a 400 MHz Intel Pentium *II™ PC* with an Nvidia *RIVA* TNT™ graphics processor. This analysis does not derive from a specific application, but is simply a counting exercise. Many applications push one or more of these limits, but few programs stress all axes.

For a typical application to achieve *1M triangles/second*, 100M 32bit pixels/second, 2 textures/pixel requires:

- 1M triangles * 3 vertices/triangle * 32 bytes/vertex = 100 MB; triangle data crosses the bus 3-5 times (read, transform and written by the CPU, and read by the graphics processor, so simply copying *triangle data* requires 300-500 MB/second on the PC buses.
- 100M pixels * 8 bytes/pixel (32bit RGBA, 32bit Z/stencil) = 800 MB; with 50% overhead for RMW requires 1.2 GB/second
- 2 textures/pixel * 4 texels/texture * 2 bytes/texel * 100M pixels = *1.6 GB; a texture cache can create up to 4X reuse* efficiency, so requires 400 MB/second

Assumptions here include:

32-byte vertices are Direct3D™ TLVertices
(X,Y,Z,R,G,B,A,F,SR,SG,SB,W)
triangle setup is done on the graphics processor
bilinear texture filtering
16bit texels are R5G6B5
50% of pixels written after Zbuffer read/compare

Transferring triangle vertex data to the graphics processor *from the CPU is commonly the bottleneck. This is different* from typical workstations or the PCs of just 1 year ago, when transform and lighting calculation, fill rate, or texture rate were limiting factors.

GEOMETRY REPRESENTATION

As pixel shading, texturing, and fill rates rise, the most constrained bottleneck in the system will increasingly become creation and transfer of geometry information. The data required to represent a triangle comprises the bulk of system bus traffic in an aggressive 3D application. As

1998 Workshop on Graphics Hardware Lisbon Portugal

triangles become smaller, there is increasing inefficiency in transferring information for 3 vertices, especially if fewer than 5 pixels are to be rendered.

There is an opportunity to re-think geometric representations to increase the number of screen pixels that are defined by the geometry information that is provided to the graphics processor. Potential solutions to this problem include geometry compression, as suggested by Deering [2]. One other alternative is the use of geometric level of detail (LOD), representing polygon models as families of similar models which may be substituted based on the viewing distance. Finally, geometry may be represented as curved surfaces and dynamically tesselated into triangles or pixels, but there are problems associated with the choice of surface formulation.

In addition to algorithmic issues, there are some additional constraints on the geometry representation problem. First, the source of the data is important. Model information is created through the use of authoring packages and it is important that the rendering pipeline directly consumes the representation that authoring packages create. Second, since the PC system is an open system standard, the communication mode between the application and the graphics accelerator is through a cross-vendor API. This API needs to match the representation as well. Microsoft's Fahrenheit™ initiative has the potential to unify competing APIs, but needs to address conflicts between software application needs and issues related to efficient hardware implementation. Finally, the more technical part of the problem is the algorithm and hardware for converting the geometry representation into pixels. Note that the technical issues are only a small part of the overall problem to be solved.

TEXTURE FILTERING

The increasing bandwidth cost of obtaining high quality texture samples motivates improved texture filtering quality beyond the standard set by square trilinear mip-mapping, described by Williams [5]. The primary shift in emphasis has been toward accurate representation of effects such as anisotropic projection of the texture area to be filtered, as described by Barkans [1]. An alternative method for accurate filtering of anisotropically projected textures is based on assembly of multiple trilinear samples, by Schilling, et al [4]. Heckbert [3] also did some fundamental work in this area. Note that trilinear filtering may be thought of as a composition of bilinear samples, and there may be an opportunity to improve the efficiency and quality of sampling through more generalized composition of bilinear samples.

Although the quality of the various anisotropic filtering solutions is improved over rectangular mip-mapping (particularly for the rendering of textures containing text), none of the algorithms described are practical for low-cost hardware implementation, due to the expenditure of sampling bandwidth on potentially large numbers of filtered samples, with limited advantage taken of coherence. Also, none of the techniques accurately sample and filter bump or environment maps.

The profligate use of many samples in obtaining a filtered texture estimate produces a high quality result, but fails to provide the predictable, regular memory access and precalculation benefits of the trilinear mip-mapping scheme. These factors make efficient hardware implementation problematic.

PER PIXEL SHADING AND LIGHTING

Both of the commonly used PC graphics APIs, Direct3D™ and OpenGL™, provide a lighting and shading model. Most interactive content developers do not use the provided lighting model, choosing instead to implement their own custom lighting. One reason for this is that both APIs present lighting models that are primarily vertex oriented. Vertex lighting results are interpolated across a triangle, but no per-pixel lighting occurs. The types of operations that may occur on a per-pixel basis include texturing and the composition of texture information with diffuse and specular lighting calculation results.

Consequently, the interactive entertainment content developers have gravitated to using textures for illumination, since that allows per-pixel shading operations to simulate per-pixel lighting through the use of texture blending and composition. Further, the use of multiple textures (for example, a surface material color texture multiplied by a diffuse lighting texture) has allowed content developers to create high quality lighting and shading. An illumination map texture can also include effects from multiple light sources as well.

This trend suggests that the graphics processor pipeline needs to evolve toward more aggressive shading at the per-pixel level, in order to provide more complex simulation of lighting. Possible directions include more extensive use of multiple textures (a full per pixel shader with any and all operands read from textures), and a greater level of programmability of the shading operations. Any lighting equation can be extended to consider any or all inputs sampled from texture maps.

CONCLUSIONS

In conclusion, there is a need to shift the focus of algorithm development for hardware implementation of 3D graphics.

The requirements are changing from "more polygons" and "more pixel fill rate" to more complex dynamic geometry and richer pixels. More complex dynamic geometry does not necessarily mean more triangles or structures for more efficient updates of triangle geometry, but rather better shapes and motion with less data, integrating authoring tools, APIs, and hardware accelerated rendering. Richer pixels does not necessarily mean more pixels rendered but rather that each pixel rendered is the result of far more effort spent lighting, shading, and texturing. The end result will be a higher degree of realism in the interactive experience.

BIBLIOGRAPHY

[1] Barkans, A., High Quality Rendering using the Talisman Architecture, Proceedings of the 1997 Siggraph/Eurographics Workshop on Graphics Hardware, pp. 79-88.

[2] Deering, M., Geometry Compression, Proceedings of Siggraph 1995, pp. 13-20.

[3] Heckbert, P., Fundamentals of Texture Mapping and Image Warping, Master's thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1989.

[4] Schilling, A., Knittel, G., Strasser, W., Texram: A Smart Memory for Texturing, Computer Graphics & Applications, May 1996, pp. 32-41.

[5] Williams, L., Pyramidal Parametrics, Proceedings of the 10th Siggraph Conference, pp. 1-11.