Chapter 1

Introduction

Marc Olano

1 About This Course

Or, “why do we want to do real-time shading, and why offer a course on it?”

Over the years of graphics hardware development, there have been obvious strides in the geometric complexity of objects that can be rendered in real-time. The first statistic quoted for any new piece of graphics hardware is the number of polygons it can render per second. However, there has also been a steady pace of improvement in appearance for objects rendered in real-time (Figure 1). These improvements are harder to benchmark and tend to come in jumps across the industry. Nonetheless, no one today would seriously consider buy a new graphics system that did only flat shading only.

Compare this to software rendering, where techniques like procedural shading have been in use for 15-20 years [3, 5, 8, 10]. Procedural shading is popular in a large part because of the power it provides to customize the appearance of everything you render by changing the procedures that control that appearance.

In the past few years, we’ve begun to see graphics hardware that can do some form of procedural shading in real-time. This new freedom in expressing the appearance of rendered objects has excited the imagination of people across the spectrum of interactive graphics users, including everyone from game developers to car designers.

However, the capabilities and ease of use of new real-time shading hardware varies widely. This course is designed to provide a solid comparison of many of the latest offerings. Even as we offer this course, hardware capabilities and software interfaces for them are improving. Any attempt to show the “state of the art”, is at best a snapshot. As such, these notes will be but one snapshot, and the course presentations another. While the notes may serve as a starting point, we encourage you to check the web sites of the various course presenters for the latest developments at and after SIGGRAPH.

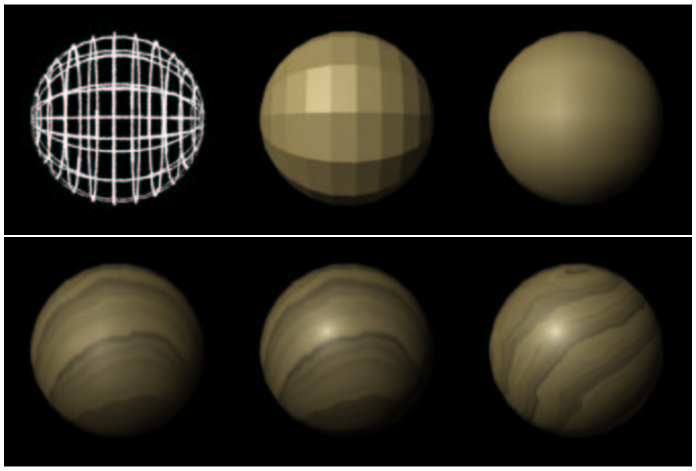

Figure 1: Progression of hardware-accelerated appearance: vector, flat shading, Gouraud shading, 2D Texture + Gouraud shading, 2D Texture + per-vertex Phong, 3D Procedural Shader

The course itself is divided into two major sections. The morning presenters (Bill Mark from NVIDIA, Jason Mitchell from ATI and Marc Olano from SGI) will focus on the how the *shaders* that determine surface appearance are described. The afternoon presenters (Chas Boyd from Microsoft on DirectX, Randi Rost from 3DLabs on OpenGL 2.0 and Michael McCool from the University of Waterloo on API Design and SMASH) will focus on API issues. That is, on the interface for using shaders and shaded objects within an application.

2 The Examples

To provide a common ground for comparison, each presenter in both sections will show three common examples on their latest and greatest system. These will be supplemented by their additional examples for each presenter to show off other important features of their system.

The common examples are not so much a benchmark of performance as a benchmark of ease of use and understanding. By using the same set of examples, course participants will be able to compare the different hardware and software interfaces. On the other hand, we haven't attempted to define the examples too precisely since each system has its strengths and weaknesses. If we'd defined every detail of the examples, we'd run the danger of giving a false comparison by the chance overlap with strengths for some systems and weaknesses for others. Instead, the examples are more roughly defined, giving each presenter the option to target their strengths — they way you'd do it if you were writing the shaders.

Note that not all chapters in these notes show implementation of the examples — in some cases major changes are expected between the course notes deadline and SIGGRAPH. In those cases, these notes include reference material that may continue to prove useful for those platforms.

The following sections describe each of the three common examples, including the problem statement given to the presenters at the outset of the course.

2.1 Shiny Bump Map

Environment mapped bump mapping ... dependent texturing, everybody seems to like it

The first example combines environment mapping, a common technique for simulating reflection, with bump mapping, a common technique for simulating fine-scale surface features through shading without changing the surface geometry. These two are interesting when put together since both bump map and environment map are results of texturing operations. Put together, they require the results of one texture lookup to influence the texture coordinates used in a second lookup.

It's also included because bumpy-shiny things have become a trite examples on recent graphics hardware, being applied to practically every object in some cases.

The problem statement intentionally avoids specifying exactly how the bump map is computed. The traditional formulation originally proposed by Blinn uses a bump

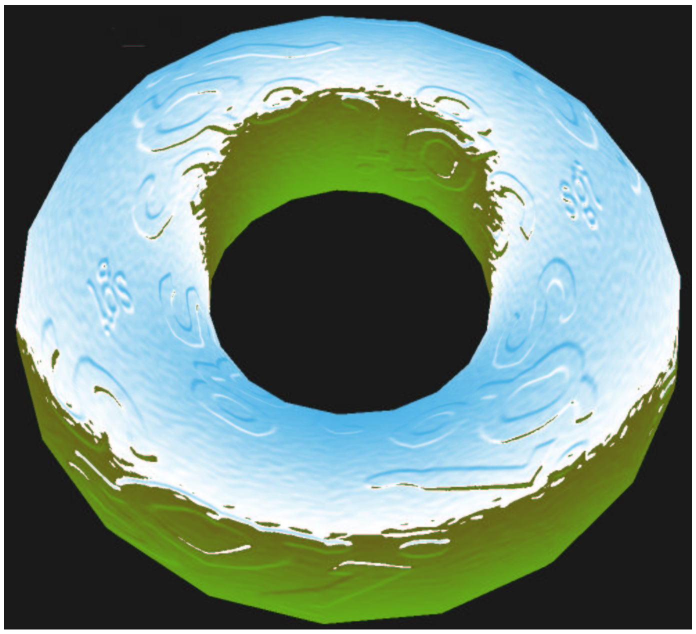

Figure 2: Shiny Bump Map on a low-tessellation torus

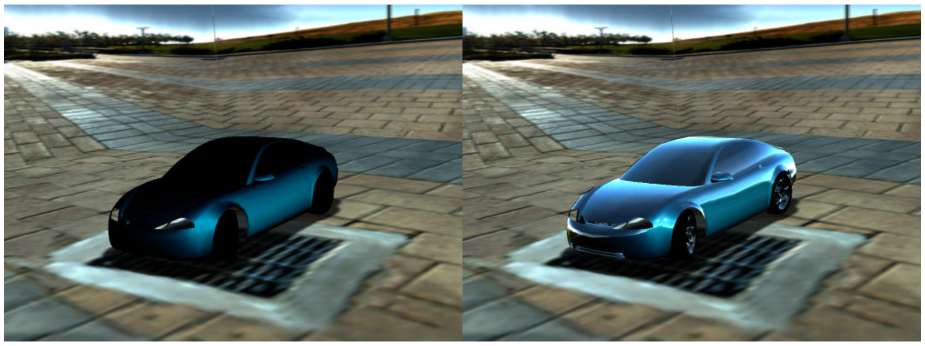

Figure 3: Car rendered with Homomorphic BRDF factorization for paint from [7], and again with environment-map based Fresnel reflectance layer on top of BRDF-based paint

texture representing a grey-scale height map of the surface [1]. Changes to the shading normal are determined from the gradients of this bump map texture. Another formulations by Cabral, subtracts shifted versions of the texture in a technique similar to 3D image embossing to compute the bump gradients [2]. Yet another formulation by Fournier uses a texture map containing surface normals (a *normal map*) instead of computing perturbations to the original shading normals [4]. Any of these or other method of computing the bumped surface normals could be used. Also, the shader could compute some other related quantity rather than the bumped normal if it seems more efficient.

The problem statement also avoids specifying how the environment map is stored. Once again, there are many options that may make more or less sense for certain implementations. All systems in this course can represent environment maps in *sphere map* form, as an image of a reflective sphere. Some can also use *cube map* form, mapping reflection vectors onto the faces of a cube, or *parabolic map* form, as images of two reflective paraboloids [6].

2.2 Homomorphic BRDF Factorization

non-standard texgen, realistic surfaces

```
texture("p",V) * texture("q",H) * texture("p",L)
* diffuse * color
```

This is the run-time aspect of McCool, Ang and Ahmad's 2001 SIGGRAPH paper [7]. The bulk of this paper dealt with numerical factorization of arbitrary bidirectional reflectance distribution functions (BRDFs) into combinations of 2D textures.

A BRDF is a 4D function that encodes the reflectance of a surface based on both the direction of view (`V` = 2 dimensions) and the incoming light direction (`L` = 2 dimensions). Equipment exists to measure the BRDF of a real surfaces, typically at a large number of discrete locations for both light and view directions. Given this BRDF, we

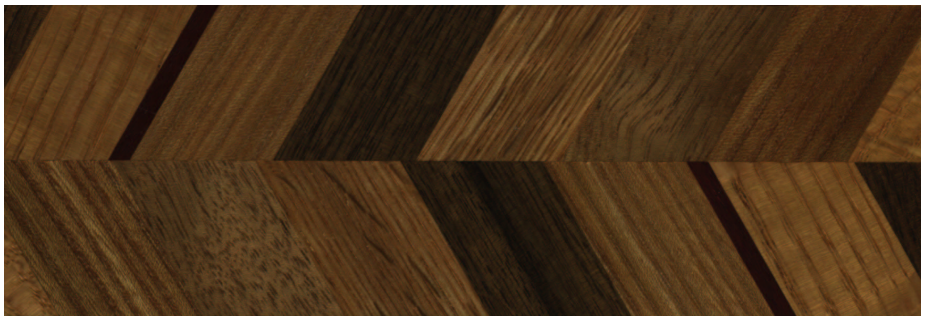

Figure 4: Scan of wood

can create realistic renderings of many surfaces. However, the nature of the BRDF as a 4D function prevent its direct use for real-time rendering.

The homomorphic factorization method computes a least-squares fit to a full 4D BRDF by a product of 2D textures, each with a unique set of texture coordinates dependent on both V and L. The method doesn't constrain the choice of texture coordinates for each 2D texture, but good results were obtained in the original paper using one texture lookup indexed by V, one indexed by L (actually the same texture used over again) and one indexed by $H = V + L$, all expressed in the local tangent coordinates. This set of textures makes some physical sense relative to "microfacet" BRDF models that model the surface as a distribution of microscopic perfectly reflective facets. The H texture can be interpreted as the probability any microfacet will have the given orientation, H. The V and L textures can be interpreted as shadowing and masking of some microfacets by other facets.

This is a good choice for an example since it requires non-standard texture coordinate generation (and hence application or vertex-level computation). It also gives more realistic appearance than is typically seen in real-time rendering, even on systems with full shading support.

2.3 Procedural Wood

Should be able to morph between 3D versions of the different wood samples in [Figure 4]. ...a good basis is the wood shader in The RenderMan Companion [9]. It should be parameterized for dark and light bands (color, width and transition) and also different fine grain in the dark and light bands (color, frequency and specularity). I'll try to get a better scan to show the last effect – there is a variation in the specular highlight intensity that correlates well with the fine grain of some of the wood.
Since it should be parameterized for all of these things, a simple 3D wood texture won't cut it, but feel free to use 1D, 2D or 3D textures for other things if it makes it easier.

This example is intended to be more complex than the typical real-time shader (though not as complex as some of the 1000-line shaders used in software rendering). A single wood shader, with a great degree of parameterization, should give all of us a reasonable challenge in comparison to the relative simplicity of the first example.

References

[1] BLINN, J. F., AND NEWELL, M. E. Texture and reflection in computer generated images. *Communications of the ACM 19* (1976), 542–546.

[2] CABRAL, B. K., PEERCY, M. S., AND AIREY, J. M. Method, system, and computer program product for bump mapping in tangent space. US Patent 5,949,424, 1999.

[3] COOK, R. L. Shade trees. In *Proc. ACM SIGGRAPH* (July 1984), pp. 223–231.

[4] FOURNIER, A. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination* (May 1992), pp. 45–52.

[5] HANRAHAN, P., AND LAWSON, J. A language for shading and lighting calculations. In *Computer Graphics (SIGGRAPH '90 Proceedings)* (Aug. 1990), pp. 289–298.

[6] HEIDRICH, W., AND SEIDEL, H.-P. View-independent environment maps. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware* (1998), pp. 39–45.

[7] MCCOOL, M. D., ANG, J., AND AHMAD, A. Homomorphic factorization of brdfs for high-performance rendering. In *Proc. ACM SIGGRAPH* (Aug. 2001).

[8] PERLIN, K. An image synthesizer. vol. 19, pp. 287–296.

[9] UPSTILL, S. *The RenderMan companion: A Programmer's Guide to Realistic Computer Graphics*. Addison-Wesley, 1990.

[10] WHITTED, T., AND WEIMER, D. M. A software testbed for the development of 3D raster graphics systems. *ACM Transactions on Graphics 1*, 1 (January 1982), 43–57.