Towards more flexible shading architectures

Uwe Behrens

GMD – German National Research Center for Information Technology
Schloß Birlinghoven; D–53754 Sankt Augustin; Email: uwe.behrens@gmd.de

Abstract

We describe a flexible programmable shading architecture based on the concept of poxels. *A poxel is some kind of rich description of a point in space. Poxels are passed between shaders and encapsulate part of the state of the rendering system. The concept corresponds to a less rigid definition of shaders than the traditional one, but at the same time promotes flexibility and reuse. By introducing special shaders that represent control structures, we are able to define a programming language for the dataflow between shaders, giving us the full power of computable functions. A system that uses this approach has been developed and a number of shaders have been implemented. We present and describe concept and implementation.*

1. Motivation

RenderMan[5] and similar systems opened a door to more flexible rendering by separating shape from shading. Whereas a core renderer is used to perform visibility calculations and hidden surface determination, illumination and shading are expressed using programs written by the user in a specialized *shading language*.[3] These programs, called *shaders* can be used to calculate a variety of phenomena, from light sources over geometric transformations to surface color.

Unfortunately, most of these systems are less flexible than necessary: For example, *RenderMan* does not allow to call shaders as subroutines from other shaders.[4] This makes the reuse of code practically impossible, unless pasteable shader sources are available. This is due to the concept of *typed shaders*.[a] The renderer decides when to call each type of shader in the rendering pipeline, and there can be only one active shader per type[b] at the same time. A shader's type specifies which part of the rendering state it may access and when in the rendering process it will be called. To preserve this order, shaders are not allowed to call other shaders to perform subtasks, not even shaders of the same type. This means shaders are given only limited control over the rendering process. A surface shader that implements a "multiple levels of detail" scheme by dynamically choosing one of a number of subshaders to perform the shading can not be implemented under these constraints, unless the shader code of all subshaders is pasted into the new shader. This is unfortunate, as it hampers reuse and flexibility.

To overcome these problems, we developed and implemented a rendering architecture that uses a less rigid definition of shaders. Here, shaders can (and shall) be layered arbitrarily and calculate and return a wide variety of data.

2. Poxels and Shaders

Formally, a surface shader s[c] is any function that maps geometry (G), material properties (M) and lighting (L) into a color (C)[1]. Thus, the following signature represents the shader's type:

[a] The following shader types are available in *RenderMan*: *surface* (calculates light reflected from a surface), *light* (models light sources), *displacement* (displacement mapping), *transformation* (geometric transformations), *volume* (volumetric absorption and scattering) and *imager* (target image representation).

[b] With the exception of *light* shaders of which more than one may be in current use.

[c] A similar argument holds for other shader types. Due to space constraints we limit the discussion to surface shaders.

$$s : G \times M \times L \rightarrow C$$

Other shader types have slightly different signatures. As the renderer decides about the order in which to call shaders, all light source shaders may have already been evaluated when the surface shader is called. This strongly limits the user, as he can not call any other shader to perform a subtask or forward the job to a shader of even the same type. But a simple extension of the shader's signature gives us the chance to incorporate the desired effect by making shaders typeless and be evaluated in any order. If we extend the definition of a shader to any function with the following signature:

$$s : (G \times M \times L \times C) \rightarrow (G \times M \times L \times C)$$

then a shader is any function that takes (practically) the entire state of the shading system and returns a new state. We call this combined state a "poxel,"[1] because it is a mixture of spatial and appearance information, and thus lies somewhere between voxels and pixels. Using the new signature, every component of the poxel can be modified by the shader at any time in the rendering pipeline. This includes texture coordinates, point, normal, material, color, etc. The main advantage of the approach comes from the fact that these shaders can easily be *composed.* Under the new signature, any composition (sequence) of shaders is itself a shader. A shader modifying texture coordinates alone is not very useful, but in composition with a shader that uses the warped coordinates to apply a texture file, plus a third one to illuminate the now textured surface we build a complex shader. This approach has a number of advantages:

- Complex algorithms (shader sequences) can easily be modified and experimented with by changing components. The result can be stored in a shader library.
- Similar to the sequence, other control structures like "if-then-else" or "while-do" can themselves be formulated as shaders. The if-then-else construct returns the result of applying one of its two associated subshaders, based on some boolean predicate. This releases the full power of a programming language built from shaders. One writes simple components as shaders and ties them together with control structures (other shaders). Due to the recursive nature of the approach one gets yet another shader as the final result which may be used accordingly.
- Composition of shaders is left to the user, not to the renderer exclusively.

3. Implementation and Results

To investigate the power of the concept, a rendering system was developed whose architecture is based entirely on fine-grained shaders and poxels. The system currently consists of about 70 classes, written in C++, most of them being shaders plus geometric primitives and various support classes. Available shaders include local illumination models, raytracing, a radiosity subsystem, bump- and texturemapping, and custom shaders like one that renders primitives invisible, assisting the renderer in visibility calculations.

Our experiences show that the concept of fine-grained shaders and poxels works pretty well. Once basic functional units have been developed, the control-structures-as-shaders approach allows to quickly tie them together and construct complex algorithms. One example are local illumination shaders that can be extended to incorporate shadows, using an if-then-else shader whose predicate tests whether the shaded point is visible from the light source before shading is performed.

Currently, we are investigating the question how to optimize the shader programs in order to speed up the rendering. There are two major paths to follow: static optimization, which tries to remove redundant shaders from the program, and dynamic optimization, using techniques like parallel processing or shader specialization[2] to achieve the desired effects. First experiments show that a substantial performance gain is possible, especially for the dynamic phase.

References

1. U. Behrens: *Rendering with Poxels – Increasing Flexibility in Programmable Shading Systems*, Proceedings 3D Image Analysis and Synthesis '97, November, 17-18 1997, Erlangen, Germany

2. B. Guenter, T. Knoblock, E. Ruf: *Specializing Shaders*, Computer Graphics, Proceedings SIGGRAPH 1995, p. 343–350

3. P. Hanrahan, J. Lawson: *A Language for Shading and Lighting Calculations*, Computer Graphics 24(4), Proceedings SIGGRAPH 1990, p. 289–298

4. P. Slusallek: *Vision – An Architecture for Physically-Based Rendering*, Ph.D. Thesis, Universität Erlangen-Nürnberg, 1995

5. S. Upstill: *The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics*, 2nd edition. Addison-Wesley, 1992