An Exposition on POV Ray

In the pursuit of beauty and bringing forth artistic works, one might chase along many paths. But for those who eschew the traditional artistic media, there is the digital computer, and various pieces of software that have been written. Most software, in order to smooth the way, allows the artist to pursue artistic creation visually. But in my case, I prefer logical systems; that is, if I cannot write a code to generate the work of visual art, then there is nothing left for me, as I have little patience with visual systems.

Fortunately there is POV Ray, which is a ray-tracing engine that provides a scene description language (SDL), and in the expressiveness of this SDL I can write code which, when executed by a computer in the context of the ray tracer, brings forth the visualizations whose hints and glimmers I gather, but the final structure belongs to the computer as it, in honoring my codes, calls various pseudorandom number generators.

I do not specify, e.g., the location in space of tori centers, or their angles of inclination. I do not specify the lengths of the minor and major radii. I do not know of what form the textures that grace the surfaces will take. These aspects depend on various random number generators, which avail themselves of various distributions, both discrete and continuous, including Beta, Cauchy, Chi Square, Exponential, F, Gamma, Lognormal, Normal, Pareto, Student's-t, Weibull, and so forth. And, of course, there are random number generators based on the very useful uniform distribution.

With a knowledge of CS 170, e.g., the artist will then have a sufficient foundation to begin learning the SDL of POV Ray, or the C-like language RenderMan provides to write shaders. With ray tracing, the behavior of iterative control structures (aka loops) in the context of invoking macros to specify mathematical surfaces is visually apparent. Sometimes the loop control variable's value, for a given entity, is useful to see rendered near the entity for the clarification and refinement of the code. Also POV Ray, while able to render surfaces that are triangulated, similar to those generated in computer games, is not constrained by such a rendering requirement and the rasterization of triangles, but utilizes the actual mathematical equations for entities, such as spheres or tori, to calculate *light ray-entity* intersection points. Unlike the learning curves for OpenGL, Vulkan, or DirectX 12, which are the main graphics APIs, the learning curve for the SDL of POV Ray is not so steep. With POV Ray, the artist can begin to produce visual output within 25 minutes of instruction. And POV Ray also supports photon mapping.

Then, of course, there is also the call of various gaming engines that have been released over the last few years. To borrow a Newtonian phrase, one might be able to see further standing on the shoulders of these giants, but the climb to their shoulders is steep. As for me, I will stay with POV Ray and its strange SDL, at least for a bit longer. And I also need to study the Metropolis Light Transport algorithm and global illumination. Or for a given scene, one could simply have the computer solve the Maxwell equations, but that is a trail that few dare to blaze.

Donald D. Derkacht
3 March 2018