Mutable Shading Contexts

In the majority of circumstances, shaders are expected to treat the `RixShadingContext` as const, read-only memory. Nonetheless in other circumstances it may be useful for a shader to alter some geometric properties of the current shading context. For example, a shader may wish to perform super-sampling of an upstream pattern network in order to perform some brute force antialiasing that would be otherwise impossible. This would require the ability to move the current points of execution, leading to the concept of a *mutable shading context*: a shading context which allows a limited amount of update ability. Mutable shading contexts are also essential to the operation of volume shaders, which need to be able to set up new locations for light scattering during the volume line integration process.

Given a normal shading context bound to a shader, a mutable shading context can be created by calling the `CreateMutableContext()` on a *parent* `RixShadingContext`. The mutable context is created in the Bxdf memory of its parent context and does not need to be released, as it will be automatically reclaimed when its parent context is released.

The primary difference between a mutable shading context and a normal shading context is that the `SetBuiltinVar()` and `SetPrimVar()` methods are now available. (On the normal shading context, these methods are no-ops). These methods should be treated with care, especially in performance sensitive situations; every call can potentially trigger an expensive re-evaluation of the entire upstream pattern network. Moreover, any such reevaluation may lead to increase in the peak memory requirements of rendering, as memory buffers allocated by the render on behalf of the mutable shading context will not be released until the parent shading context is itself also released. This is particularly a concern in cases where a large amount of supersampling is taking place, or a volume integrator is taking a lot of samples in order to integrate a volume.

Builtin Variables

`SetBuiltinVar()` allows the values of any builtin variable to be overridden. The caller is responsible for providing a pointer to memory allocated with a full `RixShadingContext::numPts` worth of allocation. This memory will subsequently be owned by the `RixShadingContext`, and therefore should be allocated using the `RixShadingContext's` own memory allocation methods (i.e. `Allocate()`) using the appropriate hint for Bxdf memory duration.

Normally, a shader that alters any single builtin variable is responsible for ensuring that all other builtin variables are also updated appropriately. As a special exception to this, any alteration of the builtin variables `u`, `v`, or `w` by a surface shader will automatically trigger recomputation of `P`, `dPdu`, `dPdv`, and `Nn` to reflect the updated values of u, v, w, as well as any other builtins that are derived from those values. All other built-in variables will be flushed from the cache and recomputed as necessary. For volume shading, any alteration of P will automatically trigger recomputation of all built-in variables derived from P.

Primitive Variables

`SetPrimVar()` allows the value of any primitive variable and its associated radius to be overridden. The caller is responsible for providing a pointer to memory allocated with a full `RixShadingContext::numPts` worth of allocation. This memory will subsequently be owned by the `RixShadingContext`, and therefore should be allocated using the `RixShadingContext's` own memory allocation methods (i.e. `Allocate()`) using the appropriate hint for Bxdf memory duration.

Unlike `GetPrimVar()`, `SetPrimVar()` allows for the creation of primitive variables that are not specified in the original geometry description.