User Tools

Site Tools

Table of Contents

5.2 to 6.0 Porting Guide

SDK Basics

The *6.0 release notes* contain a comprehensive list of everything that has changed in the 6.0 SDK. Start by reading this list and then returning here for details on select topics.

Overview & Geometry

The fundemental organization of the 5.x SDK is the same. It still has:

- Four libraries: Core, Forest, Rendering Interface, and Platform (DirectX9, PS3, etc) Renderer
- Example reference application, detailing full use of the SDK
- SRT files
- Concept of base trees (CTree) and instances (CInstance).

However, there are some big improvements in 6.0 which led to some pretty signifcant source changes. Most importantly, there are no longer rigid geometry structures or shaders. Both are fluid and based on user direction during the art asset compilation process. In 5.x, there was a single bank of shaders that applied to all of the geometry. Users were permitted to change some features of these shaders (+/ specular, +/- normal mapping, etc), but those changes would affect the entire forest. In 6.0, every material and LOD of every tree may be configured differently. While this creates a great deal more shaders for a given forest, it dramatically increases performance via shader LOD and the use of more expensive shader effects on hero trees or only on those parts of a tree that need it.

A configurable shader system also means less memory usage. Each material has its own vertex declaration and for materials with fewer effects (e.g. distant LOD materials), fewer attributes are needed. Also, 6.0 uses half floats everywhere possible, further reducing the amount of memory needed. With non-constant vertex declarations, direct geometry access is not as straightforward as querying the geometry with a 5.x SBranchVertex struct pointer. You'll have to query the *vertex declaration* and ask for vertex properties by type. The documentation also details the basic *geometry structures* and accessing *3D* and *billboard* geometry.

Note: Those users who use the SpeedTree SDK in its entirety (e.g. rendering chores) probably will not need to access the SpeedTree geometry directly as the SDK will do this automatically. This section is mostly for those users that use the Core to retrieve the geometry for use in their own vegetation rendering system.

Note: The SDK no longer makes distinctions between geometry “types” (e.g. branches, fronds, leaf cards, etc) other than 3D geometry and billboards. Geometry types are merged during Compilation to reduce the number of draw calls as much as possible.

Edit

Population & Culling

Population and culling of tree instances has been greatly improved. In addition to being much simpler to manage, performance has been increased significantly. In 5.x, the user had to pass the entire world or level's tree population into the SDK up front so that it could determine which instances were visible from any given view. This caused some complexity issues for those integrating SpeedTree into a world-building environment as that population was always changing.

In 6.0, population/culling is organized much as it is in the 5.x grass system. That is, the tree instances are streamed from the client application to the SpeedTree SDK when they become visible. Hence, the SDK will never know the world's whole population, just those tree instances in the frustum.

The 6.0 SDK organizes the world as a series of cells, just like 5.x. As the camera moves, cells go in and out of visibility (in and out of the frustum). As cells become visible, the SDK will provide a list of these cells that need to have their populations streamed in. The client app will * Unordered List Item provide a list of instances at this time. World building applications can simply flush the visible cells and repopulate when new instances need to be defined.

The structure SForestCullResults has been replaced with a higher-functioning class called CVisibleInstances, where the bulk of the stream/cull code resides. The population and culling procedures are detailed *here* and liberally documented in the reference application code.

Edit

Instance Classes

Some minor changes were made to the CInstance class defined in Forest.h:

- CInstance is now a base class for the new derived classes CTreeInstance and CGrassInstance.
- Instances can now be arbitrarily oriented. That is, they are no longer restricted to being rotated only around their up axis. Their orientation is defined by providing “up” and “right” vectors.
- A trade off is available by #defining SPEEDTREE_COMPRESS_INSTANCE_VECTORS in Forest.h. This will trade space for speed. See [SDK]/Include/Forest/Forest.h for details.

Edit

Grass System

In 5.2, grass was rendered as a series of wind-blown camera-facing quads, lit to match the underlying terrain. With the release of 6.0, the grass system has been completely redone and dramatically enhanced. In 6.0, grass geometry is defined in SRT files, exactly as tree models are. This means that they are defined and edited in the SpeedTree Modeler as if they were any other tree model. They are subject to the same geometry, lighting, and wind definitions as the tree models. The only restriction is that they must have a single LOD and a single material.

While we call this the “grass system”, it may be used to populate the scene with any bit of ground cover like rocks, twigs, or leaves. If the Modeler app can load it, it'll go through the pipeline and can be used as a grass model.

Population of grass is very similar to how it was done in 5.2, which is closer to how tree instances are populated in 6.0. Please see the documentation on the *grass system* for details.

Edit

Heap Management

Console game developers are deeply concerned about heap fragmentation, as they should be. The 5.2 SDK went into a less-flexible/less-fragmenting mode when

SPEEDTREE_HEAP_FRIENDLY_MODE was #defined. In 6.0, there is no trade-off between flexibility and less heap fragmentation. Using the SDK's heap reserve system, it's possible to limit the number of heap allocations as well as prevent any from occurring during the render loops. More *here*.

Edit

Shaders

As stated earlier, the *shader system* has been overhauled. Shaders are generated by the Compiler application. Each material and LOD can had custom settings, from a minimal per-vertex diffuse-only lighting, all the way up to per-pixel normal mapped + specular + transmission + seam_blending + normal mapped detail + shadows, etc.

Shaders are created from a series of template files which make heavy use of #ifdef directives to activate or bypass features. The Compiler inserts custom code into the templates, namely to write the fluid vertex and pixel input declarations. All of the shaders share the same set of uniform shader constants.

Edit

File System Interface

In 6.0, the SDK's accesses to SRT files, textures, and shaders are now routed through a user-controlled interface, very much like the custom memory allocator. If not needed, the SDK will automatically handle the file loads.

More on the file system interface *here*.

Edit

Instancing

The SDK rendering library & shaders now have instanced rendering built-in where supported, which is most platforms. This helps dramatically reduce the number of draw calls needed to render a forest, especially when grass and billboards are heavily used.

Edit

Page Tools

- Edit this page
- Old revisions
- Backlinks
- Back to top

Read our new blog >>

- 5446 Sunset Blvd., Suite 201
- Lexington, SC 29072
- 803-356-1999