Fresh Audio System

Technical Design

Jeff Wofford

# Use cases and edge cases

Fire and forget

AudioSystem::instance().playSound( “bird\_chirp” );

Sound customization and maintenance

sound = AudioSystem::instance().createSound( “engine” );

sound->looping( true );

// … repeatedly:

sound->pitchScalar( engineSpeed );

// …eventually:

sound->stop();

sound = nullptr;

Automatic retention and deletion of used and unused buffers

AudioSystem::instance().deleteUnusedBuffers();

Voice management: allowing and preventing multiple sounds from the same “voice”; cutting off old sounds or suppressing new ones

~~Limitation of sources on a single buffer.~~ Identified as a non-goal, and significantly so.

# Memory management structure

Audio Buffers are assets for all the usual reasons: they need to be retained when possible to avoid repeated expensive reloads, they need to be loadable simply by name, and they need to have metadata such as filenames set in a central data file (e.g. assets.xml).

Like other assets, the retention of buffers is ambiguous. Within a level, or when otherwise unchallenged, unused buffers can remain available to avoid expensive reloads. On the other hand, between levels or when a memory warning is triggered, unused buffers should be deleted. Per-level music is an example of a large asset that may need to be deleted after a level.

The AssetManager takes care of retaining assets and removing unused ones. But how is a buffer marked as used?

From a client perspective, buffers don’t exist. Rather, the client references and retains Sounds. Therefore, it makes sense that a Sound should reference (by SmartPtr) the Buffer that it uses. When all sounds referencing a buffer are deleted, the buffer is available for deletion.

There are some complications, however, centered around “fire and forget” semantics.

A “fire and forget” sound is retained by the system rather than by any specific user. This simply implies that the AudioSystem itself maintain a container of the sounds that it has returned from createSound(). Other users can reference and retain these sounds, but the AudioSystem periodically culls this list based on whether the sounds have sufficient references and whether they are stilling playing. A sound that isn’t playing and that has exactly one reference will be released (and thus deleted) immediately.

This implies that I should move Sound retention from AudioCue to AudioSystem. This, in turn, removes the concept of “retriggering” at the buffer/sound level.

# Voices

A voice is a single sound source for which only a single sound may play at a time. If a voice is already playing a sound and a second play (of the same sound or a different one) is requested, then either the new play is suppressed or the old sound is replaced, depending on the voice’s old-new priority policy.

# Queuing, Streaming, and Synchronization

More advanced features for queuing of multiple buffers on a single OpenAL source, streaming buffers from disk, and synchronizing sounds by beat (etc.) are relegated for a future version.