Texture Streaming Overview

Basics of the system used to calculate which textures should be streamed in at what times and viewpoints.

The texture streaming system, or streamer, is the part of the engine responsible for increasing and decreasing the resolution of each texture. This enables you to have good visual quality while managing the available memory effectively. This is in part done with Mips, or Mipmaps, that are pre-calculated sequences of images for your texture resolutions. You can think of these like LODs for your textures. For more information on Mips you can read about it on our Texture Format Support and Settings page.

The streamer has its own view of the world, and its update cycle consists of:

1. Updating the world view
2. Computing which resolution would be ideal for each texture
3. Choosing which resolution is actually possible based on the streaming pool size
4. Choosing which textures to update
5. Generating load / unload requests

To achieve those tasks, the streamer uses an async worker thread to reduce the workload on the game thread, so that only the first and last tasks above need to be done in the game update loop. The world view consists of the list of all the textures used by each primitive component, and for each, world bounds and texel world sizes. From that information and a given viewpoint, it is possible to compute what resolution would be required to get a good texel per pixel ratio. Additional information is then taken into consideration like whether this component was actually visible onscreen or not. This ends up defining the ideal resolution for each texture. The streamer then computes if the streaming memory pool is big enough to hold those resolutions. If not, the streamer will reduce the planned quality for selected textures, dropping one mip at a time, until the planned solution gets under budget.

The order in which textures are processed when dropping mips is defined by the retention priority, and follows these rules in order:

1. Keep landscape textures, forced load textures, and textures already missing resolutions
2. Keep mips that are visible on screen
3. Keep character textures and textures that don't take much memory
4. Drop mips that are not visible, dropping the last recently seen first

Once the streamer has determined what resolution it will load for each texture, it then computes which textures should be updated first according to the load order priority. The priority is defined by several criteria evaluated in order:

1. Load visible mips first
2. Load forced load textures, landscape textures, and character textures first
3. Load textures which are far from their target resolutions first
4. For those that are not visible, load the most recently seen first

The final step is to generate a batch of update requests, each of those either increasing or decreasing the current texture resolution. The amount of memory updated for a single batch is limited by the temporary pool size, in order to keep the number of inflight requests low.

Improvements in 4.15

The Texture Streaming system has been optimized to reduce CPU usage, memory usage, and load times while elemintating low resolution artifacts and automatically handling limitations of varying memory budges of different platforms.

- Texure Memory Usage Improvements
- Texture Loading Time Improvements
- CPU Time Reduction:
    - Game thread update time is reduced by 50%

  - Texture processing stalls for streaming levels are reduced up to 98%
- Low Resolution Artifact Reduction
- Automatic Memory Budgeting
- Texture Streaming Debugging Visualization Viewmodes
  - Primitive Distance Accuracy
  - Mesh UV Density Accuracy
  - Material Texture Scales Accuracy
  - Required Texture Resolution

To learn more about these latest improvements and debugging methods for your games continue on to our Building Texture Streaming Data and Reporting Texture Streaming Metrics pages.