Tutorial for Asynchronous Asset-Loading

- Asynchronous Asset-Loading

Asynchronous Asset-Loading

Description

This tutorial describes Asynchronous Asset-Loading and provides short examples on how to load a bitmap.

Asynchronous Asset-Loading

Asset Loading, especially reading from the asset repositories, can be a time consuming task, which blocks the CPU from further processing. This can even slow down rendering in certain cases. To improve this, asset loading requests can now be submitted as asynchronous jobs. These jobs can be done either in parallel on a separate thread, or interleaved with the rendering in smaller chunks. Whenever an asset is available, the main thread can retrieve it and add it to the render loop.

How to activate this in the application

For a Courier based application, the AsyncLoadReqMsg message can be used to load Views. For a Candera based application, Async version of the asset loader methods can be used (see AsyncAssetProviderProxy and new ContentLoader methods).

Number of threads for asynchronous asset loading

To load the assets asynchronously in one single thread, the interface "AsyncAssetProviderProxy" can be used. It is available from any AssetProvider implementation via the AssetProvider::AsyncProxy() method. Using a second thread for async loading can be achieved by checking the CANDERA_ASSETLOADER_WORKER_THREAD_ENABLED in Cmake. AsyncRequests will be dispatched on the worker thread as soon as possible. For multithreaded async asset loading, the CMake flag CANDERA_ASSETLOADER_WORKER_THREAD_ENABLED specifies if the asynchronous asset load requests are handled on a dedicated thread or on the caller one.

Upload

Upload can be done asynchronously too, as ContentLoader::BuildSceneContextAsync can also upload to VRAM (depending on a parameter value). In this case, the single threaded variant must be used.

Examples

In the AsyncAssetProviderProxy, each Get* (GetScene) function schedules a Get* (GetSceneById) call on the AssetProvider that created the Proxy. In the case that CANDERA_ASSETLOADER_WROKER_THREAD_ENABLED flag is enabled in CMAKE, the AssetProvider::Get* call will be dispatched on a separate thread and the result will be available as soon as the worker thread completes the dispatch. In the case that the above mentioned CMake option is disabled, one or more calls to DispatchNext() need to be performed on the current thread

to assure that the request is dispatched. It is advised to call the DispatchNext() routine periodically during the Update part of the render loop of the application to ensure all asynchronous asset load requests are dispatched in a controlled and limited time. One option to call DispatchNext would be to introduce it in the LateUpdate part of the UpdateSystem provided by Candera.

```
        AssetProvider* assetProvider = ContentLoader::GetInstance().GetAssetProvider();
        if (assetProvider != 0) {
          UpdateSystem* updateSystem = EntityComponentSystem::EntitySystem::Get<UpdateSystem>();
          if (updateSystem != 0) {
            UpdateSystem::Handle updateHandle = updateSystem->CreateComponent();
            UpdateSystem::Delegate delegate = UpdateSystem::Delegate::Update<AsyncAssetProviderPr
            updateSystem->SetComponentLateUpdateDelegate(updateHandle, delegate);
            updateSystem->AttachComponent(updateHandle, &assetProvider->AsyncProxy());
          }
        }
```

Samples to retrieve a Bitmap Asynchronously:

- If CANDERA_ASSETLOADER_WORKER_THREAD_ENABLED is enabled (processing is done on a separate thread)

```
BitmapAsyncRequestSharedPointer request = assetProvider->AsyncProxy()->GetBitmap();
  Candera::AssetProvider* provider;
  Candera::AsyncAssetProviderProxy& proxy = provider->AsyncProxy();
  Candera::AsyncAssetProviderProxy::BitmapAsyncRequestSharedPointer
 asyncRequest = proxy.GetBitmap(0x123);
  //do other stuff
  Candera::Bitmap::SharedPointer bitmap;
  if (asyncRequest->IsCompleted()) {
      //bitmap was asynchronously loaded
      bitmap = asyncRequest->GetResult();
}
```

- If CANDERA_ASSETLOADER_WORKER_THREAD_ENABLED is disabled (processing is done later on the same thread, by calling DispatchNext)

```
BitmapAsyncRequestSharedPointer request = assetProvider->AsyncProxy()->GetBitmap();
  Candera::AssetProvider* provider;
  Candera::AsyncAssetProviderProxy& proxy = provider->AsyncProxy();
  Candera::AsyncAssetProviderProxy::BitmapAsyncRequestSharedPointer
 asyncRequest = proxy.GetBitmap(0x123);
  //do other stuff
  proxy.DispatchNext();
  //do other stuff
  Candera::Bitmap::SharedPointer bitmap;
  if (asyncRequest->IsCompleted()) {
      //bitmap was asynchronously loaded
      bitmap = asyncRequest->GetResult();
```

```
}
```