Candera Engine 3D

Shader: Support for precompiled shaders on all enabled platforms.

Uploading shader programs using Shader::Upload can now be done in a more flexible way. Until now it was only possible to upload either shader source or pre-compiled binaries, through a target-specific implementation of RenderDevice::CompileAndLoadShaderPair.

Now it is possible to upload (and compile if necessary) either shader source or shader binary objects. It is even possible to upload complete shader programs in order to avoid shader linking. How shader upload is done depends on the kind of data that is available in the asset and the capabilities of the target platform, however at runtime this is done completely transparent to the user.

Advantage is that the more flexible way of runtime created shader source code (as used in Candera2D implementations over OpenGL ES) is supported as well as using pre-compiled shaders to save execution time for compiling them. While using pre-compiled shaders on enabled platform is recommended due to performance reasons, still backward compatibility to existing solutions using uncompiled shaders is given.

OpenGL ES 2 Reference Shaders relocated

The Candera OpenGL ES 2.0 Shader language reference Shaders have been moved from `<cgi_studio_candera>/src/Candera/Engine3D/ReferenceShaders` to `<cgi_studio_candera>/src/Candera/Engine3D/RefShaders/ESSL1`

OpenGL ES 3 Reference Shaders

Candera now contains a set of reference Shaders for the OpenGL ES 3.0 Shader language. `<cgi_studio_candera>/src/Candera/Engine3D/RefShaders/ESSL3`

Shader Resource Objects.

Fragment and vertex shader data or access information is stored in the new ResourceDataHandle structure. See migration guide for an example on how it can be used.

VertexGeometry Resource Objects.

The vertex, index and element format arrays are stored in the new ResourceDataHandle structure. See migration guide for an example on how it can be used.

VertexBuffer primitive iteration.

Iteration over triangles is replaced by primitive iteration using a forward iterator, which is more performant, allows iteration over point and line primitives and supports multiple parallel iterations. See migration guide for an example on how it can be used.

VertexBuffer vertex access.

Vertex access interface is moved from the VertexBuffer to a separate class for better performance in case of vertex data accessed from assets. See migration guide for an example on how it can be used.

8/16/32 bit indexed array buffers.

VertexGeometry accepts index buffers that consist of 8, 16 or 32 bit indexes. The configuration of index size can be set through the new values of BufferType enumeration: UInt8IndexedArrayBuffer, UInt16IndexedArrayBuffer and UInt32IndexedArrayBuffer.

VertexElementFormat access.

VertexElementFormat access interface is moved from the VertexGeometry to a separate class for better performance in case of vertex data accessed from assets. See migration guide for an example on how it can be used.

VertexGeometryBuilder support for 8/16/32 bit index buffers.

VertexGeometryBuilder can create index buffers with values on 8, 16 or 32 bit, depending on the number of vetices, or on the configured format if explicitly set.

BlendEquation Min/Max support.

Min/Max options were added to the RenderMode BlendEquation, which specifies how a new pixel is combined with previously rendered pixels in the frame buffer.

Revision #2
Created 2023-03-02 07:37:43 UTC by Kanai Tomoaki
Updated 2023-06-14 01:33:46 UTC by Kanai Tomoaki