Candera V3.1.1

Release Date: November 3, 2015

To keep backwards compatibility for any interface change in the public API the old interface has been kept and marked as deprecated. The deprecated interfaces will be removed in future versions. Compiler warnings will be generated when using a deprecated interface through the usage of Candera Macro CANDERA_DEPRECATED_3_1_1.

- Candera Engine 3D
- Candera Engine 2D
- Candera Base
- Candera Device
- Migration Guide

Candera Engine 3D

Shader Uniform Cache Handles

If a shader uses more than a single uniform, a Shader::UniformCacheHandle is the fastest way to access uniform locations. After a uniform cache accessor was registered in a shader with an array of uniforms to be accessed, the resulting uniform cache handle can be used to retrieve uniform locations in constant time. It does not matter if the uniform is an auto or a custom uniform.

Candera uses this mechanism to get rid of uniform location related (string) compares.

Render Order Criterion

The RenderOrderCriterion class extends the OrderCriterion class to enable the user to control preparation and sorting of render order bins. By default it is now used to skip sorting if the order of the bin has not changed since the last pass.

If an OrderCriterion was assigned to the RenderOrderBin, everything is processed as usual. If a RenderOrderCriterion was assigned to the RenderOrderBin, RenderOrderBin::Sort calls RenderOrderCriterion::PrepareRenderOrderCriterionValues, RenderOrderCriterion::HasOrderChanged, and depending on the result of the previous function, RenderOrderCriterion::Sort.
Using the new interface enables the user to have more control over how sorting is performed, and if sorting needs to be performed at all.

Candera's DistanceToCameraOrderCriterion, ReverseDistanceToCameraOrderCriterion, RankOrderCriterion, as well as RenderStateOrderCriterion have all been changed to use the new RenderOrderCriterion interface.

Batch Order Criterion

The BatchOrderCriterion is a RenderOrderCriterion implementation that sorts bins by either VertexBuffer+Shader+RenderMode+Distance to camera, or by VertexBuffer+Appearance+Distance to camera to increase drawcall batching (i.e. implicit

geometry instancing) rates. It also takes advantage of the new RenderOrderCriterion feature to only sort bins when the order has changed.

Extended Geometry Instancing Support

Classes derived from CameraRenderStrategy can now utilize implicit geometry instancing (i.e. drawcall batching) by passing a flag to its base class. The BreakNodeCameraRenderStrategy has been updated to use it by default.

The PointSprite class is now recognized as being batchable, i.e. point sprites will automatically be picked up by drawcall batching (geometry instancing) if possible.

Render Order Usage Change

The render order was previously associated with the Scene. With this version it has been moved to the Camera. Every camera requires its own render order now. This was done to be able to skip sorting of previously sorted render order bins.

The following functions were declared deprecated:

```
    void Scene::SetRenderOrder(AbstractRenderOrder* renderOrder);
    AbstractRenderOrder* Scene::GetRenderOrder();
    const AbstractRenderOrder* Scene::GetRenderOrder() const;
```

PointSprite Interface and Behavior Changes

Point sprites share a single vertex buffer now, unless a custom vertex buffer is set for an instance of a point sprite by the user. Therefore PointSprite::GetVertexBuffer() returns a const pointer in order to avoid alteration of the shared vertex buffer. The color of a point sprite is now determined by the diffuse color of the material to establish consistent usage of the auto uniform that was used by the point sprite shader. GenericShaderParamSetter::SetPointSpriteActivationEnabled() must be set to true, when using it with a PointSprite.

The following functions were deprecated:

```
    void PointSprite::SetColor(const Color& color);
    const Color& PointSprite::GetColor() const;
```

MorphingMesh Changes

Just like the PointSprite, the MorphingMesh was changed to establish consistent usage of its auto uniform. Therefore GenericShaderParamSetter::SetMorphingMeshActivationEnabled() must be set to true, when using it with a MorphingMesh.

GenericShaderParamSetter Changes

The GenericShaderParamSetter now handles the auto uniforms of the PointSprite and the MorphingMesh.

The following functions were added, and must be used in conjunction with point sprites and morphing meshes:

```
    bool GenericShaderParamSetter::IsMorphingMeshActivationEnabled() const;
    void GenericShaderParamSetter::SetMorphingMeshActivationEnabled
(bool enableMorphingMeshActivation);
    bool GenericShaderParamSetter::IsPointSpriteActivationEnabled() const;
    void GenericShaderParamSetter::SetPointSpriteActivationEnabled
(bool enablePointSpriteActivation);
```

Renderer Statistics Improvements

Renderer::Statistics now also exposes the total number of queries and occlusion query results for the last render pass.

Generation of index buffers update.

Math3D::CreateVertexBufferWithIndexGenerated is now extended with an argument specifying the minimum size of one index (8, 16 or 32 bit).

Candera Engine 2D

TextNode2D and TextNode2DLayouter

TextNode2D is a RenderNode specialized for text rendering. A TextNode2DLayouter needs to be attached to the node for text layout. The DefaultTextNode2DLayouter implements horizontal or vertical truncation for correct text layout inside given or computed area. TextNode2D requires a BitmapBrush effect for text rendering and one TextNodeRenderer object for generating correct images to be rendered. Four TextNodeRenderer implementations are available in Candera, each corresponding to one CacheType option in the existing TextBrush rendering concept.

SkipNode functionality to Camera2DRenderStrategy

SkipNode action was added to Camera2DRenderStrategy to tell the Renderer2D to proceed with rendering, but skip current node.

Candera Base

TextureImage to Image2D Adaptor

An adaptor class TextureImageToImage2DAdaptor was added to enable using a TextureImage (3D only) as an Image2D. The TextureImageToImage2DAdaptor class implements Image2D interfaces and can therefore be used directly with a corresponding effect.

This can for instance be used to make the DirectTextureImage functionality available in 2D scenes.

Example Usage:

```
TextureImageToImage2DAdaptor::SharedPointer image = TextureImageToImage2DAdaptor::Create();
image->SetTextureImage(texture, pixelFormat);
image->Upload();
```

Default layout alignment change

The default value for horizontal and vertical layout alignment was modified from HCenter and VCenter to HStretch and VStretch.

Candera Device

Shared Sampler Objects

On OpenGL ES 3.0 Platforms texture sampling parameters is decoupled from texture information using sampler objects. The benefit is that multiple textures may share the same sampling parameters and therefore a sampler object. A sampler object can be bound to a texture unit in a single draw call, eliminating the need to set all sampling parameters such as filter mode, wrap mode or anisotropy separately per frame.

In Candera the usage of sampler objects is implicit and hidden from the user. The RenderDevice will determine which textures can share a sampler object and will bind them accordingly. Sampler objects can be shared if the MinLod and MaxLod parameters are left at their default values, to utilize the full potential of sampler objects it is beneficial if the other sampler parameters MaxAnisotropy, WrapMode, MipMapFilter, MagnificationFilter and MinificationFilter are the same for as many textures as possible. The LodBias parameter has no effect on sampler objects as it is a texture parameter only.

A new value recorder id was added for CGI Studio Analyzer to track the number of sampler objects that are created: ValueRecId::SamplerObjectCount. The value represents the number of sampler objects created across all contexts.

DirectTextureImage Alpha Format

The newest iMX6 drivers support setting an 8 bit alpha format on DirectTextureImage. The format FormatAlpha has been added.

Added check for custom Bitmap Format

A check for custom bitmap format has been added to Bitmap class. The new function is used by Candera RenderDevice to check the bitmap format before Upload is attempted.

See also:

Bitmap::IsCustomPixelFormat

Bitmap size computation

As different platforms support different bitmap formats, the size of the pixel array for a bitmap, in VRAM, is platform dependent. To provide a more accurate value for this size the following modifications were made:

- new RenderDevice::GetSize and RenderDevice2D::GetSize methods were added. This method receives a Bitmap parameter and returns the size of its pixel array in VRAM.
- new BitmapImage2D::GetSize method was added. This method returns the estimated value of the video memory used by the bitmap.
- the possibility to explicitly specify a size for a Bitmap within the constructor was preserved. However, the automatic computation of the size, if default value 0 is used with the constructor, was removed. Now, if default value 0 is used for the *size* parameter in constructor, Bitmap::GetSize method returns 0. If a value is set for *size* parameter, the Bitmap::GetSize method returns this value.

New draw methods for RenderTarget class

New RenderTarget::SafeBeginDraw, RenderTarget::SafeEndDraw and RenderTarget::GetActiveRenderTarget methods have been added.

SafeBeginDraw and SafeEndDraw are used to store the active render target during RenderTarget::BeginDraw / RenderTarget::EndDraw sequence. A pointer to the active RenderTarget is set on a SafeBeginDraw call and reset to 0 on a SafeEndDraw call.

The stored render target can be further retrieve using RenderTarget::GetActiveRenderTarget method.

As the current active render target is further used in the internal implementation of Candera, it is recommended to change calls of BeginDraw and EndDraw to calls of SafeBeginDraw and SafeEndDraw.

Migration Guide

Following an overview about interface changes between Candera V3.1.0 and Candera V3.1.1.

Description	Candera V3.1.0	Candera V3.1.1
Render Device deprecated texture parameter functions are replaced with a single function.	RenderDevice::SetTextureLodMode RenderDevice::SetTextureFilter RenderDevice::SetTextureWrapMode	RenderDevice::ActivateTexture
Scene deprecated funtions are replaced with Camera functions (Note: Every Camera requires its own RenderOrder!)	Scene::SetRenderOrder Scene::GetRenderOrder	Camera::SetRenderOrder Camera::GetRenderOrder
PointSprite deprecated functions are replaced by using the corresponding functions of the PointSprite's Material	PointSprite::SetColor PointSprite::GetColor	Material::SetDiffuse Material::GetDiffuse
PointSprites share a single VertexBuffer by default to facilitate drawcall batching. Custom VertexBuffers can still be set, but getting the VertexBuffer returns a const pointer now.	VertexBuffer* PointSprite::GetVertexBuffer	const VertexBuffer* PointSprite::GetVertexBuffer