Candera Base

Camera Groups

CameraGroup is a container that groups cameras from multiple scenes. CameraGroup has an arbitrary number of 2D and/or 3D cameras from different scenes and it is used by applications to easily access and change camera settings uniformly, e.g. in transitions.

See also:

Texture Render Targets

Unified Cloning Interface

Base classes implement a virtual "Clone() const" method, which is overridden in all children. "Clone() const" builds a new instance of the class using the copy constructor. Copy constructors are protected for use only in derived copy constructors. Copying via assignment operator is prohibited, as it could cause problems, like orphaned nodes, memory leaks or dangling pointers. Assignment operators are thus private.

Cloning is shallow, with only shared pointers and constant character pointers being copied. All other coupling via pointers is broken for the clones.

For more complex cloning see Tree Cloner and Tree Cloning Strategies.

Tree Cloner

TreeClonerBase is provided for cloning trees. The default clone strategy is to call the Clone interface on each node, and build the clone tree with the same node pattern as the original tree.

For deeper cloning, TreeClonerBase supports attaching a TreeCloneStrategy. This TreeCloneStrategy is typically aware of different types of nodes, and delegates to other strategies.

Tree Cloning Strategies

The deep cloning strategies respect the following rules:

- all shared resources are deep cloned as long as they are not device objects; shared resources in the destination are shared in the same pattern as in the source tree.

- node references are rebuilt as long as source references are found as descendants of the source root; descendants are mapped to the destination tree.
- strategies are designed to support the decorator pattern.
- general strategies delegate to more specific strategies.

Bitmap as SharedPointer

Instances of class Bitmap can only be created by calling static method Bitmap::Create() which returns a SharedPointer.

All interfaces have been changed to pass the shared pointer and the corresponding disposer function has been removed from the interfaces.

Old interface (example):

```
void BitmapImage2D::SetBitmap(Bitmap* bitmap, BitmapDisposerFn disposerFn);
```

New interface (example):

```
void BitmapImage2D::SetBitmap(Bitmap::SharedPointer bitmap);
```

Changes in Bitmap Enumerations

Until now, Bitmap class described the organization of the pixels with the enumerators Bitmap::Format and Bitmap::Type.

With introducing OpenGL ES 3.0 ETC2/EAC Texture Compression, it came up that the bitmap description was not suitable for describing compressed images (which have a format descriptor, but no type descriptor). Another problem was that the Bitmap class allowed invalid combinations of Format and Type, e.g. Bitmap::Format::RgbFormat (3 components) and UnsignedShort4444Type (4 components).

In order to enable compressed bitmaps and to avoid invalid combinations the enumerator Bitmap::PixelFormat has been introduced, here Format and Type have been merged to valid enumeration items, while the ETC2/EAC items have been added.

For this reason Bitmap::Format and Bitmap::Type have become deprecated. Of course it is still possible to use custom formats.

Shared Pointer Concept in Animation Framework

Objects of the following classes can only be created by calling static method Create() which returns a SharedPointer:

- AnimationController
- AnimationTimeDispatcher
- AbstractEasingFunction (and all derived classes)
- InterpolationStrategy (and all derived classes)
- AnimationPropertySetter (and all derived classes)

All interfaces have been changed to pass the shared pointer and the corresponding disposer function has been removed from the interfaces.

Old interface (example):

```
void AnimationPlayer::SetController(AnimationController* controller, DisposerFn disposerFn = 0);
```

New interface (example):

```
void AnimationPlayer::SetController(AnimationController::SharedPointer controller);
```

TreeTraverserBase for 'const' Traversing

Interface of template class TreeTraverserBase has been changed to use only Traverse() and ProcessNode(). TraverseConst() and ProcessConstNode() have been removed. This ensures type safe const correct usage but enforce all const-traverser implementations to be changed. Const traversers have to be defined the following way:

```
class MyTraverser : public TreeTraverserBase<const Node> {
    protected:
        virtual TraverserAction ProcessNode(const Node& node)
        {
        ...
        }
};
```

Candera::Animation Namespace extension

Classes and Types located in `<cgi_studio_candera>/src/Candera/EngineBase/Animation` which formerly belonged to the *Candera* namespace were reassigned to the Candera::Animation namespace.

Affected classes/types:

- AbstractEasingFunction
- AnimationBlendedProperty
- AnimationController
- AnimationGroupPlayer
- AnimationKeyframeSequence
- AnimationPlayer
- AnimationPlayerBase
- AnimationPlayerListener
- AnimationPropertySetter
- AnimationTimeDispatcher
- WorldTimeType
- SequenceTimeType
- BackEaseFunction
- BezierInterpolationStrategy
- BounceEaseFunction
- EaseInterpolationStrategy
- ElasticEaseFunction
- ExponentialEaseFunction
- InterpolationStrategy
- KeyframeSequence
- LinearInterpolationStrategy
- PowerEaseFunction
- SplineInterpolationStrategy
- StepInterpolationStrategy

> Moved classes and types are still accessible via the *Candera* namespace in CGI-Studio 3.0.0 but these deprecated interfaces will be removed in the next release.

CanderaObject becomes DynamicPropertyHost

CanderaObject inherits from DynamicPropertyHost. Objects previously inheriting from DynamicPropertyHost now inherit from CanderaObject. ApplicationData is deprecated. The DynamicPropertyHost allows to attach multiple application data objects.

Revision #4
Created 2023-03-02 08:28:52 UTC by Kanai Tomoaki
Updated 2023-06-19 04:56:04 UTC by Tsuyoshi.Kato