3D Scene Graph Dynamics

Description

Usually all required nodes are already specified and preconfigured via SceneComposer, so in most cases it won't be necessary to make any adaptations within the scene graph loaded from an asset at runtime.

However, some use cases might require such adaptations. This tutorial covers the most common means to manipulate the scene graph structure dynamically.

Example Solution

- SceneGraphDynamicsSolution_3D in folder *cgi_studio_player/content/Tutorials/03_SceneGraphAndNodes*

Adding and Removing 3D Nodes

3D Scene Graph Dynamics Example

For creating and adding a new node, the following can be used:

- **SceneGraphDynamicsSolution_3D** from folder *cgi_studio_player/content/Tutorials/03_SceneGraphAndNodes*
- **SceneGraphDynamicsWidget_3D** (this is used in the example solution - refer to SceneGraphDynamicsWidget_3D)

As an example use case, a Candera::Billboard can be created, added and removed. Use the SceneGraphDynamicsWidget_3D properties described below:

- **AddBillboard**: If AddBillboard is enabled a new Billboard is created and added to the widget node. If disabled, the Billboard will be removed.

```
        // Definition of WidgetProperty "AddBillboard"
        CdaProperty(AddBillboard, bool, GetAddBillboard, SetAddBillboard)
            CdaDescription("If AddBillboard is enabled a new Billboard is created and added to t
        CdaPropertyEnd()
```

```
        // End Definition of WidgetProperty "AddBillboard"
```

Creating a new Billboard

The following code snippet generates a basic Candera::Billboard (size 30 x 30) with a Candera::Material and a matching Candera::Shader. The Shader has to be provided in the asset and is delivered through the AssetProvider by its name. Further, some translation is applied to the new Billboard.

```
        // Create and init a new Billboard

        SharedPointer<Appearance> myAppearance = Appearance::Create();
        SharedPointer<Material> myMaterial = Material::Create();
        myMaterial->SetEmissive(Color(0.5F, 0.0F, 0.0F, 1.0F));
        myAppearance->SetMaterial(myMaterial);
        myAppearance->SetShader(Base::GetAssetProvider()->GetShader(CgiAssetNames::RefTransLight
        myAppearance->SetName("Generated Appearance");

        m_billboard = Billboard::Create(30.0F, 30.0F);
        m_billboard->SetAppearance(myAppearance);
        m_billboard->Translate(Vector3(30.0F, 0.0F, 0.0F));

        // end Create and init a new Billboard
```

Adding the new Billboard to the scene graph

Next the Billboard is uploaded to VRAM and appended to the node associated with the widget.

```
        // Add Billboard

        static_cast<void>(m_billboard->Upload());
        static_cast<void>(node->AddChild(m_billboard));

        // end Add Billboard
```

Removing and destructing the Billboard

To remove the Billboard simply from the scene graph, use Candera::Node::RemoveChild. If the Billboard will be used for further operations e.g. attaching it to an other node removing would be enough. Since in this example the Billboard isn't used further it should not only be removed but also be destructed completely (freeing the resources).

The following code snippet removes the previously generated billboard from its parents, unloads the Billboard from VRAM and frees with Candera::Node::Dispose the resources.

```
        // Remove previously generated billboard and destruct it

        Node* parent = m_billboard->GetParent();
        if (parent != 0){
            static_cast<void>(parent->RemoveChild(m_billboard));
            FEATSTD_LOG_ERROR("- Node %s removed from parent %s ...\n", m_billboard->GetName(),
        }
        // Destruct node

        static_cast<void>(m_billboard->Unload());
        m_billboard->Dispose();
        m_billboard = 0;

        // end Destruct node

        // end Remove previously generated billboard and destruct it
```

As Candera::Node::Dispose internally calls the Candera::Node::RemoveChild on parent node following code would be enough for removing and destructing a node.

```
        // Destruct node

        static_cast<void>(m_billboard->Unload());
        m_billboard->Dispose();
        m_billboard = 0;

        // end Destruct node
```

> In SceneComposer the newly added Node does not appear in the Scene Tree view because it is only added dynamically by the widget.
> - The used Shader for the generated Billboard in the widget has to be included when generating an asset from the solution - ensure that "Always include in asset" flag is marked at the Shaders properties.
> - The widget must control memory and VRAM management of its internal dynamic subtree autonomously.

Widget Interaction in the Player

First, the asset must be exported and loaded in the Player. Then, one of the two existing *SceneGraphDynamicsWidget_3D* widgets must be selected for the 3D scene. After, **ChangeAppearance**, **CloneOperation** and **AddBillboard** properties can be enabled/disabled (1/0) to obtain different results.

Changing Appearance and other 3D Node Attachments

For changing the appearance of a node, the following property of the SceneGraphDynamicsWidget_3D can be used:

- **ChangeAppearance**: If ChangeAppearance is enabled, the appearance of the widgets associated node will be changed to a new generated appearance. If disabled, the previous appearance will be restored.

```
        // Definition of WidgetProperty "ChangeAppearance"
        CdaProperty(ChangeAppearance, bool, GetChangeAppearance, SetChangeAppearance)
            CdaDescription("If ChangeAppearance is enabled, the appearance will be exchanged to
        CdaPropertyEnd()
        // End Definition of WidgetProperty "ChangeAppearance"
```

Try the ChangeAppearance property at the SceneGraphDynamics_3D_Billboard widget in the example solution to change the Billboards appearance from a texture appearance to a material appearance.

Creating and changing an Appearance

The following code snippet generates a basic Candera::Appearance. A Candera::Material is created and its emissive color is set to a light green. The material and a proper shader is set to the Appearance. The shader has to be provided in the asset and is delivered through the AssetProvider by its name.

```
    // Create and set a new Appearance

    SharedPointer<Appearance> myAppearance = Appearance::Create();
    myAppearance->SetMaterial(Material::Create());
    myAppearance->GetMaterial()->SetEmissive(Color(0.0F, 0.5F, 0.0F, 1.0F));
    myAppearance->SetShader(Base::GetAssetProvider()->GetShader(CgiAssetNames::RefTransLight1_Re
    myAppearance->SetName("Generated Appearance");

    // end Create and set new Appearance
```

In the next step the node's previous Appearance is saved (for restoring purpose) and the new generated appearance is set to the node associated with the widget.

```
    // Change Appearance

    m_previousAppearance = node->GetAppearance();
    node->SetAppearance(myAppearance);
    static_cast<void>(myAppearance->Upload());
```

```
    // end Change Appearance
```

Restoring / releasing an Appearance

Restore the previous Appearance and destruct the dynamically generated appearance.

```
        // Restoring appearance and destructing

        SharedPointer<Appearance> currentAppearance = node->GetAppearance();
        static_cast<void>(currentAppearance->Unload());
        currentAppearance.Release();
        node->SetAppearance(m_previousAppearance);

        // end Restoring appearance and destructing
```

> In SceneComposer the changed Appearance does not appear in the Appearance panel because it is only changed dynamically by the widget.
> - The used shader for the generated Billboard in the widget has to be included when generating an asset from the solution - ensure that "Always include in asset" flag is marked at the Shaders properties.
> - The widget must control memory and VRAM management of dynamically created node attachments autonomously.

Cloning 3D Nodes

For cloning a node, the following property of the SceneGraphDynamicsWidget_3D can be used:

- **NodeToClone**: Select an arbitrary node in the scene graph, which shall be cloned and added to the widget node

```
        // Definition of WidgetProperty "NodeToClone"
        CdaProperty(NodeToClone, Candera::Node*, GetNodeToClone, SetNodeToClone)
            CdaDescription("Select a Candera::Node which shall be cloned and added to the widget
        CdaPropertyEnd()
        // End Definition of WidgetProperty "NodeToClone"
```

- **CloneOperation**: If enabled, the clone operation will be executed and the cloned node will be added to the widget node.

```
        // Definition of WidgetProperty "CloneOperation"
        CdaProperty(CloneOperation, bool, GetCloneOperation, SetCloneOperation)
```

```
            CdaDescription("If CloneOperation is enabled, the Candera::Node to clone will be clo
        CdaPropertyEnd()
        // End Definition of WidgetProperty "CloneOperation"
```

For the cloning operation, refer to Candera::Node::Clone. The following code snippets illustrate, what the SceneGraphDynamicsWidget_3D is actually doing for cloning a node and destructing it again.

Validating Node to Clone

This example should allow to clone nodes, which are part of the scene graph the widget is linked to. The scene graph can be traversed from a given node up to its actual top level scene node to check whether it is part of the scene graph.

```
    // Search the parent scene of a given node

    Node* currentNode = node;

    while (currentNode != 0) {
        if (currentNode->IsTypeOf(Scene::GetTypeId())) {
            return static_cast<Scene*>(currentNode);
        }
        currentNode = currentNode->GetParent();
    }
    return 0;

    // end Search the parent scene of a given node
```

Creating and Adding Clone

If the widget node and the node to clone have same top level scene, the node can be cloned and added to the widget node according to the following code snippet.

```
        // Clone the node and add it to the widgets associated node

        m_clonedNode = TreeCloner().CreateClone(*m_nodeToClone);
        static_cast<void>(node->AddChild(m_clonedNode));
        static_cast<void>(m_clonedNode->UploadAll());
        m_clonedNode->SetName("Clone");

        // end Clone the node and add it to the widgets associated node
```

> Nodes can be attached to any 3D node.

TreeCloner and Cloning Strategies

For more complex cloning see Candera::TreeCloner and Candera::TreeCloneStrategy.

Candera::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 Candera::TreeCloneStrategy. This TreeCloneStrategy is typically aware of different types of nodes, and delegates to other strategies.

Removing and Destructing Clone

Analogous to Removing and destructing the Billboard (removing/destructing a 3D node) the cloned node is destructed.

```
        // Remove a previously cloned node

        static_cast<void>(m_clonedNode->UnloadAll());
        m_clonedNode->Dispose();
        m_clonedNode = 0;

        // end Remove a previously cloned node
```

> In SceneComposer the newly added node clone does not appear in the Scene Tree view because it is only added dynamically by the widget.
> - It is not allowed to modify the scene graph maintained by SceneComposer outside of the subtree owned by the widget!
> - The widget must control memory and VRAM management of its internal dynamic subtree autonomously.

Revision #7
Created 2023-02-21 06:20:27 UTC by Tsuyoshi.Kato
Updated 2025-01-10 11:28:56 UTC by Elisabeth Zauner