Viewport 2.0 API Porting Guide for Locators

Introduction

This document analyzes the choices for porting plug-in locators (MPxLocatorNode) to Viewport 2.0 mostly based on the following factors.

- Portability: For example, how easy is it to port the plug-in locators, how much workload is required, is it a long-term solution?
- Scalability: For example, how does it perform as the number of locators increases?
- Compatibility/Flexibility: For example, which drawing APIs/platforms is it compatible with, does it interact with other features, and which scenarios are suitable?

There are also a family of “footPrint” plug-ins in the SDK examples which demonstrate how to port a custom locator to Viewport 2.0 using different solutions. A pointer of the related SDK examples will be provided for reference at the beginning of each porting choice.

Use MPxLocatorNode legacy fixed draw code and selection in Viewport 2.0

Example: footPrintNode.

It is possible to reuse legacy viewport drawing and picking when using Viewport 2.0 for plug-in locators. However, this is available since Maya 2017 and should only be considered a temporary solution to allow for an incremental migration to Viewport 2.0.

Enabling

To reuse legacy viewport drawing code implementation when using Viewport 2.0, set the MAYA_ENABLE_VP2_PLUGIN_LOCATOR_LEGACY_DRAW environment variable to any value.

In Maya versions prior to 2017 Update 3, the legacy viewport picking implementation isn’t enabled by the above environment variable and the selection picking is only performed on the locator's bounding box center. To re-enable the legacy viewport picking implemention, set the MAYA_VP2_USE_VP1_SELECTION environment variable to any value.

Scalability

- Optimizations from Viewport 2.0 are not supported (for example, consolidation).
- There is no guarantee of equivalent performance when compared to:

    - Legacy viewport drawing
    - Other porting choices for Viewport 2.0

Compatibility

- You must run Viewport 2.0 in OpenGL Core Profile (Compatibility) or OpenGL Legacy mode.
    - OpenGL Core Profile (Strict) mode is not supported, as it does not allow a fixed function pipeline to be run.
    - DirectX 11 mode is not supported.
- OpenGL state may not be set up in the same way as the legacy viewport. Legacy draw code should avoid making assumptions about the initial state.
    - It is the responsibility of the legacy draw code to set up fixed function state such as lighting and so forth.
- It is the responsibility of the legacy draw code to ensure that it does not corrupt any OpenGL state.

Use MUIDrawManager

Examples: footPrintNode, footPrintNode_SubSceneOverride.

The MUIDrawManager class provides a straight forward way for API users to draw basic UI elements when porting plug-in locators to Viewport 2.0. The decision as to using this class is mostly based on a choice for simplicity while flexibility and scalability is not required.

MUIDrawManager is not designed for accessing arbitrary times or arbitrary places. Plug-ins must associate a custom locator with an implementation of MPxDrawOverride/ MPxGeometryOverride/MPxSubSceneOverride and override the addUIDrawables() method on these classes to access MUIDrawManager. For discussion purpose, it's better to treat MUIDrawManager as an independent porting choice, while the three override classes can be analyzed based on cases without using MUIDrawManager.

Portability

MUIDrawManager avoids the complexity of using MRenderItem. Each UI drawable uses the appropriate geometry, shader and matrix transformation automatically based on the type. For example, text has an appropriate "text shader" and 2D drawing has an appropriate "2D" matrix transformation. This knowledge is hidden from the user for simplicity. The pros of this approach include:

- Suitable for simple UI, where simple means a small amount of UI, or a when a small number of objects are drawing the UI.
- Supports text, icons, lines, circles, basic 2D and 3D primitives, and arbitrary meshes.
- Easier to port legacy draw code as it looks more like fixed function drawing.

- Requires only one set of code which can be reused across all supported drawing APIs.

Scalability

- UI drawables are transient render items in nature, so they cannot take advantage of consolidation or hardware instancing, and they may not scale well due to the potential cost of memory reallocation.
- It is possible for plug-ins to avoid any unnecessary overhead due to the frequency of calling addUIDrawables() and recreating UI drawables.
    - A draw override should be constructed with the MPxDrawOverride constructor argument isAlwaysDirty set to false whenever possible.
    - A subscene override should override the areUIDrawablesDirty() method to return false when UI drawables doesn't need to be updated.
- As noted in the API documentation for MUIDrawManager, Viewport 2.0 may batch the same type of UI drawables created by the same call to the addUIDrawables() method.
    - A subscene override is responsible for drawing all instances if the associated DAG object is instanced, so UI drawables for all instances will need to be created by the same call to the addUIDrawables() method. This will allow the UI drawables to use the batching optimization, however, the trade-off is to transform each UI drawable from its belonging instance's object space to the orginal instance's. For cases where the trade-off can outweigh the batching optimization, e.g. complex meshes, the MRenderItem interface should be used.
    - To inspect whether UI drawables are batched or not, execute the following MEL command and observe the number of render items being drawn.
        - ogs -traceRenderPipeline true
- Plug-ins can consolidate the geometries and reduce the number of UI drawables in their code in order to avoid the internal repetitive batching for the same set of UI drawables by Viewport 2.0.

Compatibility/Flexibility

Pros:

- Drawing API agnostic.
- Locators can be selected by default. It is possible to make a custom locator non-selectable by using the *Selectability* argument in the beginDrawable() method.

Cons:

- Only a fixed set of shading options and single-textured drawing are provided.
- The geometry attributes that can be specified are fixed.
- No inherent concept of participating based on viewport display modes. Requires the implementation to track these modes manually.
- No override options for participating in post effects and advanced transparency algorithms (as MRenderItem can) in case they are required by plug-in locators.

Use MPxGeometryOverride

Examples: footPrintNode_GeometryOverride.

The MPxGeometryOverride class is similar to the internal classes used to support native DAG objects, like polygonal meshes and NURBS surfaces, in Viewport 2.0. By using this class, plug-ins take advantage of all internal optimizations such as consolidation and hardware instancing. Since the MUIDrawManager class has been discussed separately, this section analyzes the porting choice of using a geometry override which has no UI drawables.

Portability

- The implementation operates using the MRenderItem, and requires only one set of code which can be reused across all supported drawing APIs.
- Relatively simple to work with. When porting custom locators that require a stock shader, the workload can even be minimized by using a footPrintNode_GeometryOverride implementation as a template requiring only minimal changes, such as updates for custom geometry data or a different stock shader.
- This porting choice requires some knowledge about the Viewport 2.0 API and its underlying design philosophy, especially when porting locators that require a custom shader, or trying to take advantage of Viewport 2.0 performance schemes for optimal performance. Refer to Viewport 2.0 API Porting Guide (http://www.autodesk.com/developmaya).

Scalability

- A well optimized implementation of MPxGeometryOverride provides predictably better performance over legacy viewport or other porting options. Here are some general optimization guidelines:
  - Plug-ins should allow render items to use consolidation or hardware instancing whenever possible. For example, maintain shader instance caches to allow render items for various shapes to share the same shader instance. Refer to section 3.6 in Viewport 2.0 API Porting Guide (http://www.autodesk.com/developmaya) for details about how render items can be considered for consolidation.
  - Plug-ins can merge the same type of render items regarding categorization, display properties and shader etc. in their code and reduce the draw overhead in case consolidation or hardware instancing cannot be used.
  - Render items can share vertex buffers, as well use different index buffers if appropriate, to reduce memory usage and transfer overhead.
- By default, the Viewport 2.0 consolidation system uses a hybrid mode composed of traditional static consolidation and multi-draw consolidation.
  - Traditional static consolidation improves drawing performance for static shapes.
  - Multi-draw consolidation can improve drawing performance for both matrix-animated (non-deforming) shapes and static shapes, as long as it is supported.

        - Requires OpenGL Core Profile.
        - Supported on most recent GPU architectures and drivers for Windows and Linux. Warning messages display in output window when not supported.
        - Not yet available on Mac OS X due to missing graphics driver support.
- If a geometry override has UI drawables, it is incompatible with multi-draw consolidation (but still compatible with traditional static consolidation). This can cause a decrease in draw performance when the associated locators are being animated.
- Hardware instancing is the alternative solution to optimize the drawing performance for both matrix-animated and static objects when multi-draw consolidation is not supported on the desired platforms or incompatible with a geometry override.
    - Requires the GPU Instancing option to be turned on from Hardware Renderer 2.0 Settings (i.e. the "hardwareRenderingGlobals" node).
    - Requires the associated locator to be instanced.
- To determine whether consolidation or hardware instancing kicks in, execute the following MEL command and observe the number of render items being drawn.
    - ogs -traceRenderPipeline true

Compatibility/Flexibility

- Drawing API agnostic.
- Picking is handled automatically, although it is possible to override selection via the refineSelectionPath() method.
- Participation in viewport display modes, post effects, and advanced transparency algorithms can be specified on render items.
- Render items can be assigned with a stock shader, a shader translated from a Maya shading group (via the setShaderFromNode() method), or a custom shader.

Use MPxSubSceneOverride

Examples: footPrintNode_SubSceneOverride.

Although the MPxSubSceneOverride class is primarily designed for "scene-cache" style nodes that manage a large set of objects, it can be used for any type of DAG object including locators. You can use this class for a custom locator that is instanced with multiple attributes having per-instance data.

Portability

- The implementation operates using the MRenderItem interface and requires only one set of code which can be reused across all supported drawing APIs.
- Required to manage all render items to draw all instances of the associated DAG object, but gains direct access and flexible control to hardware instancing.
- Requires explicit update logic. It is totally up to the implementation to determine whether updates are needed.

- Relatively more coding work due to the amount of the control given to the implementation when compared to a geometry override. Requires knowledge about Viewport 2.0 API and its underlying design philosophy. This is especially the case if you want flexible hardware instancing support. Refer to Viewport 2.0 API Porting Guide (http://www.autodesk.com/developmaya).

Scalability

- A subscene override can take advantage of flexible hardware instancing support when the associated DAG shape is instanced with multiple attributes having per-instance data.
  - It is possible for hardware instancing to be used due to the ability to add extra streams of per-instance data using the setExtraInstanceData() method.
  - In this case, subscene override outperforms geometry override since consolidation and hardware instancing cannot be used by a geometry override.
- It is possible for render items of a subscene override to take advantage of "subscene consolidation" by using the setWantSubSceneConsolidation() method.
  - It is a simplified version of traditional static consolidation, which doesn't take into account the spatial proximity of render items. As a result, performance scalability might be unstable.
- Shader instances should be reused whenever possible to allow hardware instancing or consolidation to be used. Plug-ins may maintain shader instance caches to allow different shapes to use the same shader instance.
- To inspect whether hardware instancing or consolidation kicks in, execute the following MEL command and observe the number of render items being drawn.
  - ogs -traceRenderPipeline true

Compatibility/Flexibility

- Drawing API agnostic.
- Picking is handled automatically, while it is possible to override selection via getSelectionPath() or getInstancedSelectionPath().
- Participation in viewport display modes, post effects and advanced transparency algorithms can be specified on render items.
- Render items can be assigned with a stock shader, a shader translated from a Maya shading group (via the setShaderFromNode() method), or a custom shader.

Use MPxDrawOverride

Examples: rawfootPrintNode.

The MPxDrawOverride class is designed for accessing low-level drawing APIs which gives complete and total control (and responsibility) for drawing the associated DAG object.

Portability

- Requires multiple sets of code if multiple drawing APIs need to be supported.
  - OpenGL Legacy mode and OpenGL Core Profile with compatibility mode may reuse the legacy fixed draw code.
  - OpenGL Core Profile strict mode and DirectX 11 need new sets of code if required.

Scalability

- Due to the wide-open nature of the interface, it is up to the implementation to use its own performance schemes as Viewport 2.0 performance schemes will not be used, otherwise equivalent performance cannot be expected when compared to a geometry or subscene override.
- A draw override should be constructed with the MPxDrawOverride constructor argument isAlwaysDirty set to false whenever possible, to avoid unnecessary update. When well-tuned, it should outperform MUIDrawManager and the temporary solution reusing MPxLocatorNode legacy fixed draw code.

Compatibility/Flexibility

- A draw override is free to act as necessary in the draw callback to draw the DAG object (apart from triggering evaluation of the Maya dependency graph).
  - If the override needs to modify the state (in any manner), it must be sure to restore that state before completing execution to avoid state corruption.
- A draw override can participate in post effects by overriding excludedFromPostEffects(). It is up to the implementation to draw correctly for each effect based on the context information.
- Transparency is supported; however a draw override cannot participate in advanced transparency algorithms when isTransparent() is set to true.
- The selection picking can work only when camera-based selection is active. If not active, the selection picking can be only performed on the locator's bounding box center.