Courier Specific

MFR - Multi-Frequency Rendering in Courier

Multi-Frequency rendering in Courier can be used to create custom sort criteria for the render jobs.

By default, the Courier::RenderJobStrategy sorts them like this:

- offscreen render jobs before Display render jobs.
- 2d render jobs before 3d render jobs
- clear render jobs before normal view render jobs
- lower camera sequence number before higher camera sequence number

One can create a custom sort criteria, simply by deriving from Courier::RenderJobStrategy and then use the Courier::Renderer::SetRenderJobStrategy() method to use this custom one.IE:
SampleApp.cpp
mRenderer.SetRenderJobStrategy(CANDERA_NEW(PriorityRenderJobStrategy)(1000));

For example, PriorityRenderStrategy class uses such a custom sort criteria.

In this sample, the PriorityRenderStrategy demonstrates the ability to use the Renderer to sort the render targets depending on the layer they are on, and the sequence number of the associated camera.

```
bool PriorityRenderJobStrategy::IsPriorityRenderJob(const Courier::RenderJob& renderJob) const
{
    bool res;
    if (renderJob.IsClear()) {
        res = IsPriorityRenderTarget(renderJob.GetGdu());
        return res;
    }
    res = GetRenderJobCameraSequenceNumber(renderJob) <= m_lastPriorityCameraSequenceNumber;
    return res;
}

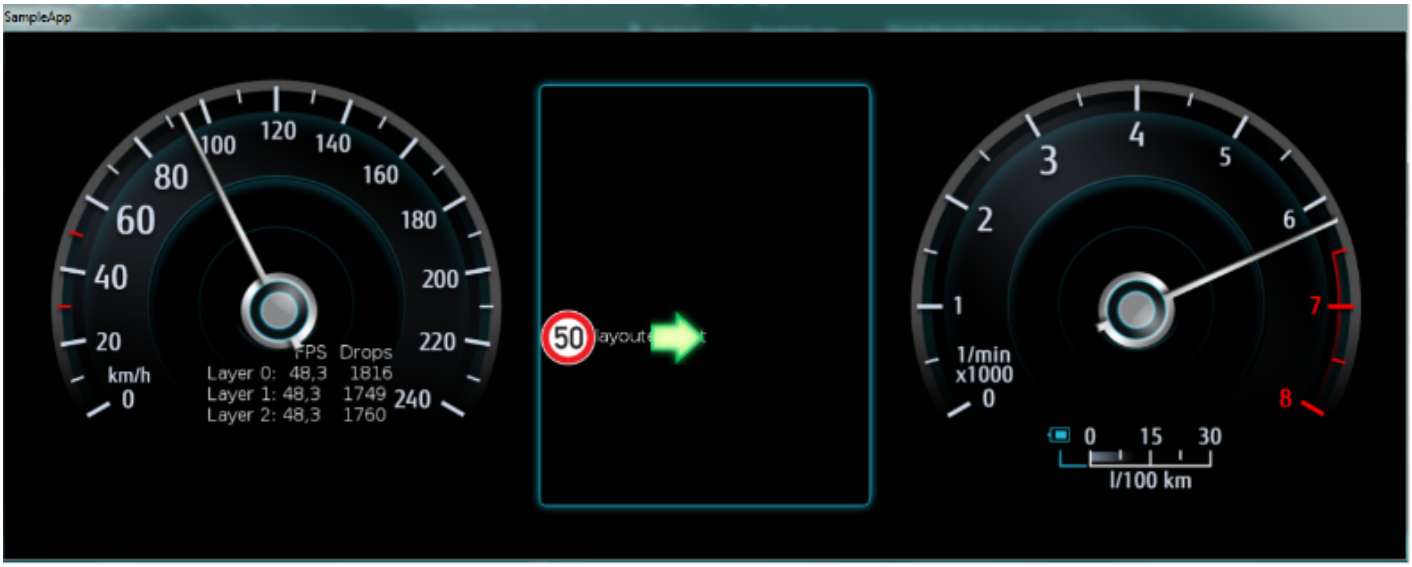
// ----------------------------------------------------------------------
bool PriorityRenderJobStrategy::IsPriorityRenderTarget(Courier::Gdu* gdu) const
{
    for (FeatStd::Int i = 0; i < m_priorityRenderTargets.Size(); ++i) {
        if (gdu == m_priorityRenderTargets[i]) {
            return true;
        }
```

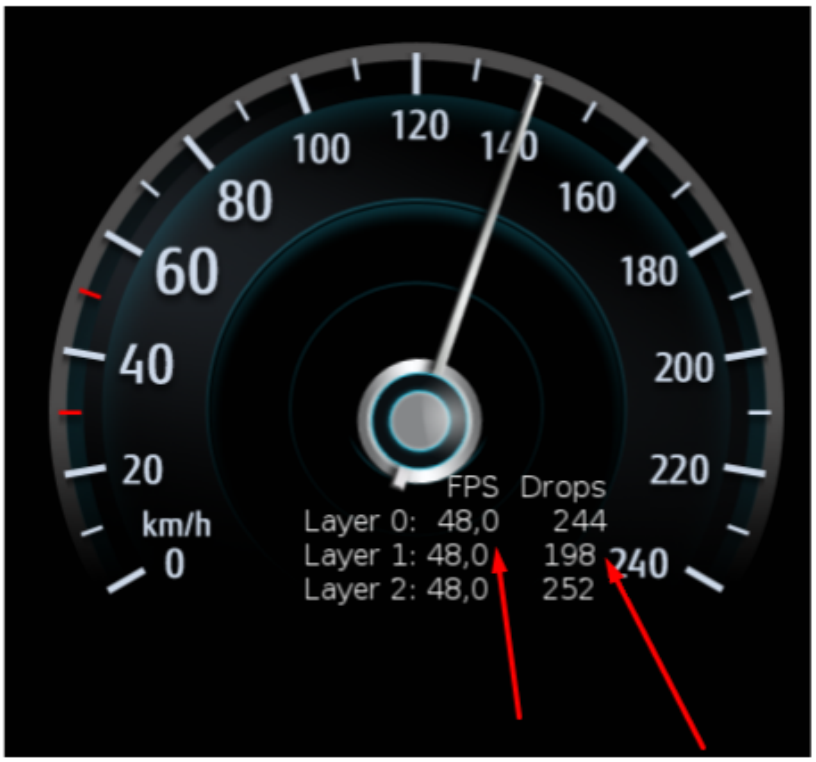
```
    }
    return false;
}
```

Also as you can see in RenderTargetPerformanceData class, we can compute the cost of rendering a frame and see how many frame drops we had.

The PriorityRenderStrategy uses two other classes, PriorityCamera2DRenderStrategy and PriorityCamera3DRenderStrategy which are derived from RendererListenerBase, and Candera::Camera2DRenderStrategy and these classes compute the render cost for each node.

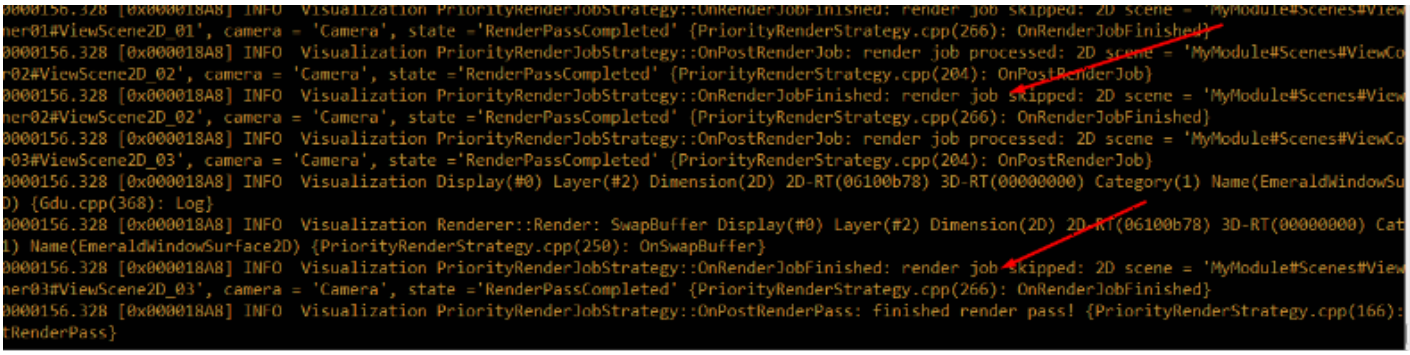
Example of the application running:





As you can see, the FPS is kept the same for all the layers, but the drops vary depending on the Layer/Nodes/Camera Sequence Number of each render target, some of the render jobs being skipped by the custom sort criteria in the PriorityRenderStrategy class.

In the sample output, red arrows mark skipped render jobs:



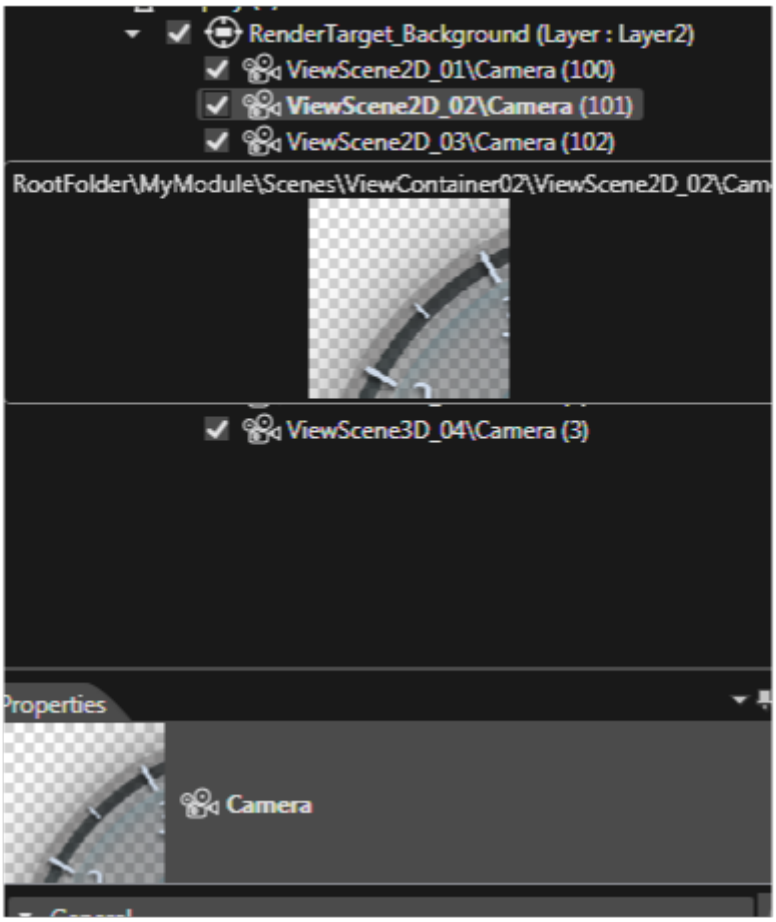
SceneComposer solution

In this demo application, there are three layers: Layer0, Layer1 and Layer2 displayed bellow:



For Layer 2 – which has the most content – the backgrounds, the respective cameras have the highest Sequence Number to have priority being rendered.


ViewScene2D_02
Animation1
500
Parent
Scene: Camera
Scene Editor
Shader Editor
Graph Dependencies
0
100
200
300
400
1
2
3
4
5
6
7
8
1/min
x1000
0
0
15
30
l/100 km


RenderTarget_Background (Layer : Layer2)
ViewScene2D_01\Camera (100)
ViewScene2D_02\Camera (101)
ViewScene2D_03\Camera (102)
RootFolder\MyModule\Scenes\ViewContainer02\ViewScene2D_02\Cam
ViewScene3D_04\Camera (3)
Properties
Camera

Invalidation

Description

Courier introduces an invalidation mechanism by providing Courier::FrameworkWidget::Invalidate and Courier::View::Invalidate methods, which shall be called if ViewScenes Candera::Camera(s) shall be rendered. Only enabled and invalidated cameras will be rendered. Cameras can be enabled/disabled by sending the appropriate Courier::ActivationReqMsg or by manually maintaining the state of the cameras via widget implementation.

The invalidation of the cameras causes appending a Courier::RenderJob to a Courier::RenderJobs array inside the Courier::Renderer. This array is then used for rendering the cameras when Courier::RenderComponent executes the Render method. If the render counter of a RenderJob reaches 0 then the RenderJob is removed from the array of RenderJobs. If a new RenderJob is appended with an already existing RenderJob for this camera, the older entry will be removed.

2D and 3D RenderJobs are processed in their own arrays. This is necessary because of above threading configuration scenarios.
Every Candera::Rendertarget used by a rendered camera will be swapped once after rendering the cameras.

A more specific method of invalidation is the usage of the methods:

- Courier::ViewScene2D::Invalidate(Candera::Camera2D * invalidatedCamera, UInt8 renderCounter = cDefaultRenderCounter)
- Courier::ViewScene3D::Invalidate(Candera::Camera * invalidatedCamera, UInt8 renderCounter = cDefaultRenderCounter)

Those methods allow invalidating the specified cameras (not necessarily all cameras of a view) and might also be used in the widget implementations. The widget has therefore use method Courier::FrameworkWidget::GetParentView, cast the returned pointer to ViewScene2D or ViewScene3D and then call the appropriate 2D or 3D Invalidation method.

Update & Invalidation Sample

For example the BindeableValueWidget in our GettingStarted App invalidates if the bound value has changed and it shall be redrawn.

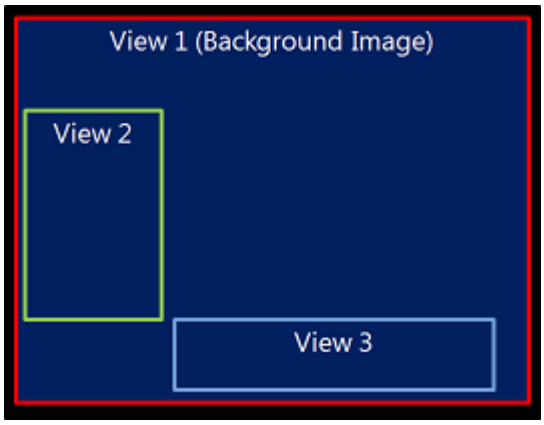
```
void BindableValueWidget::OnChanged(Candera::UInt32 propertyId)
{
    if(!mBuffered) {
        if(propertyId == ValuePropertyId) {
            UpdateValue(GetValue());
        }
        Invalidate();
    }
}
```

Invalidation in General

Most views are not overlapping, so there is no dependency between the displayed view. However, depending on the design, this may not always be the case.

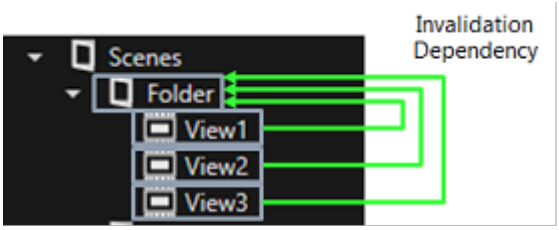
The following examples demonstrate various combinations of overlapping views and their dependencies regarding invalidation:

Example 1



Whenever View2 is invalidated, View1 must be invalidated too, because they are overlapping. View3 has to be invalidated as well, because it is overlapping with View2.

View2 is invalidated		
->	Overlap with View1	
->	View 1 must be invalidated	
	->	Overlap with View3
	->	View3 must be invalidated



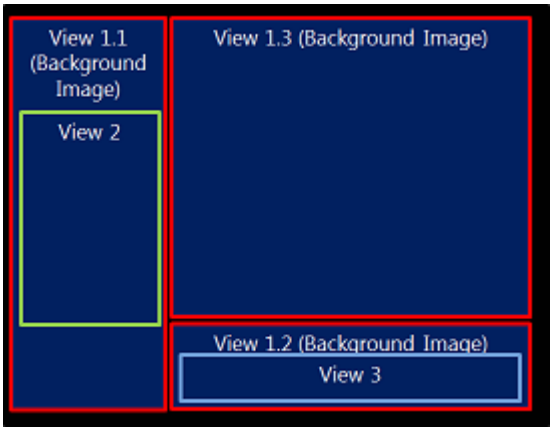
Implementing the Dependency

When a ViewContainer ("Folder" in SceneComposer) is invalidated, all its child views are invalidated.
View1-3 can set an invalidation dependency to the ViewGroup "Folder" to implement the mutual

invalidation.

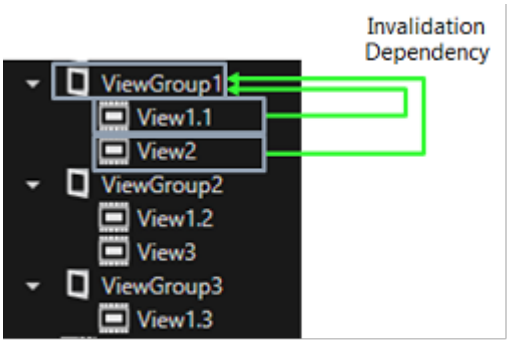
Example 2

This is a possible solution for a shared background: Split the background into multiple views.



View2 is invalidated
-> Overlap with View 1.1
-> View 1.1 must be invalidated

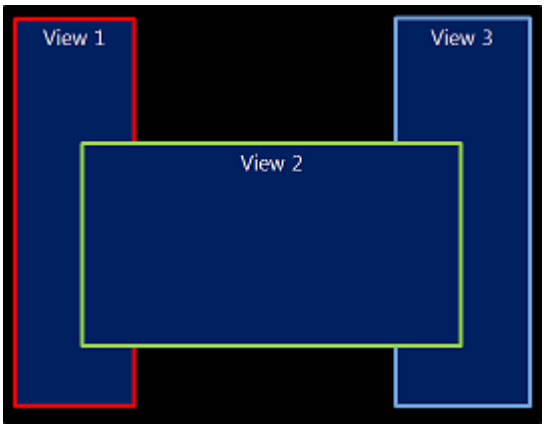
View 3 is invalidated
-> Overlap with View 1.2
-> View 1.2 must be invalidated



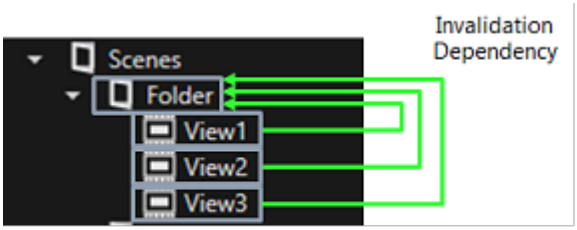
Implementing the Dependency

Example 3

Sometimes overlap can not be avoided because of the design.



An invalidation dependency between all 3 views exists.
Possible solution:
Use multiple hardware layers if available.
If only 1 hardware layer is available, invalidation dependencies must be set.



Implementing the Dependency

Wakeup Render Mechanism

Description

Courier::RenderConfiguration::UseWakeUpRenderMechanism: The default value for this is false. If this is set to true, each widget should call WakeUpAllRenderComponents() whenever a property setter is called. Otherwise the Update method will not be called if the RenderComponent is currently in sleep mode. If implemented correctly this setting will result in the best possible performance optimization, therefore it can be recommended.

Revision #6
Created 2023-02-17 08:10:04 UTC by Tsuyoshi.Kato
Updated 2025-01-24 08:28:32 UTC by Elisabeth Zauner