# Rendering Only Updated Content

## Description

This chapter briefly describes how to achieve performance optimization by rendering only updated content.

## Introduction

### Invalidation Introduction

Basic principles for composing graphical content:

- Place content which is never changed in an own scene and render it to an own layer.
- Partition content which is frequently changed by widgets or animations. Use several cameras for this purpose.
- If rarely modified graphical content is part of a dynamic scenery then think about using Texture Render Targets.

### How is it done?

In the render loop of the application all Widgets are updated by calling Candera::WidgetBase::Update(). The basic render loop looks like this:

```
// 3D Render Loop

  // handle input events
  // ...

  // update animations by calling AnimationDispatcher::DispatchTime()
  if (m_animationDispatcher3D != 0) {
    m_animationDispatcher3D->SetWorldTimeMs(nowMs);
    m_animationDispatcher3D->DispatchTime();
  }

  // update all widgets
  CgiApp::GetInstance().Update();
```

```
    Renderer::RenderAllCameras();

  // end 3D Render Loop
```

The widgets must now implement a mechanism to tell the application whether associated graphical content needs to be rendered. The following pages explain two ways to achieve this:

- Use Candera::Camera::SetRenderingEnabled() / Candera::Camera2D::SetRenderingEnabled()
- Explicitely Pick Camera to Render

# Use SetRenderingEnabled()

## Use Candera::Camera::SetRenderingEnabled() / Candera::Camera2D::SetRenderingEnabled()

The example below shows how in CGIApplication all 3D cameras are turned off in first step of render loop.

```
void CgiApp3dGc::Update() {
    Base::Update();

  // Turn off all cameras
    UInt32 cameraCount = Renderer::GetCameraCount();
    for (UInt32 i = 0; i < cameraCount; i++)
    {
        Camera* cam = Renderer::GetCamera(i);
        cam->SetRenderingEnabled(false);
    }

    // Update widgets in all visible scenes
    WidgetBase* widget = 0;
    Vector<SceneContext*>::ConstIterator it = m_sceneContexts.ConstBegin();
    for (; it != m_sceneContexts.ConstEnd(); it++) {
        if ((*it) != 0 && (*it)->GetScene() != 0 && (*it)->GetScene()->IsRenderingEnabled()) {
            widget = (*it)->GetFirstWidget();
            while (widget != 0) {
                widget->Update();
                widget = (*it)->GetNextWidget();
            }
        }
    }
}
```

The Candera::Widget::Update() method needs to be implemented such that the recently turned off camera is turned on again whenever rendering is required. The Candera::Camera can be either set as a widget property or retrieved inside the scene by masking the widget and the associated camera with the same Candera::ScopeMask value.

```
void SampleBaseWidget3D::InvalidateCamera(bool value)
{
  m_camera->SetRenderingEnabled(true);
}
```

```
void MyWidget::Update()
{
  // update graphical content from cached property value(s)
  //...

  Base::InvalidateCamera(true);
}
```

# Pick Camera to Render

## Explicitly Pick Camera to Render

Another method is to maintain a list of cameras by the application.

The render loop has to be changed

- to call Candera::Renderer::RenderCamera( Camera * camera )
- instead of Candera::Renderer::RenderAllCameras().

> The application needs to take care on Render Buffer Swapping.

Revision #10
Created 2023-02-17 08:07:45 UTC by Tsuyoshi.Kato
Updated 2026-08-19 02:17:37 UTC by Tsuyoshi.Kato