Occlusion Queries and Occlusion Culling

Description

In scenes containing nodes with lots of vertices or expensive fragment shaders, the rendering speed can be drastically improved by avoiding to render occluded objects. For this purpose Candera offers the Occlusion Queries and Occlusion Culling, which are described on this page.

Occlusion Queries

Overview

Occlusion queries are the collective name for a particular type of queries, which are useful to detect whether an object is visible or not. They interrogate asynchronously the Render Device to determine whether the scoped rendering commands pass the depth test and if so, how many samples pass. Occlusion queries are typically used to enable features like occlusion culling and varying lens flare or bloom intensity.

> Queries are related to one specific rendering context and must not be uploaded to multiple contexts.

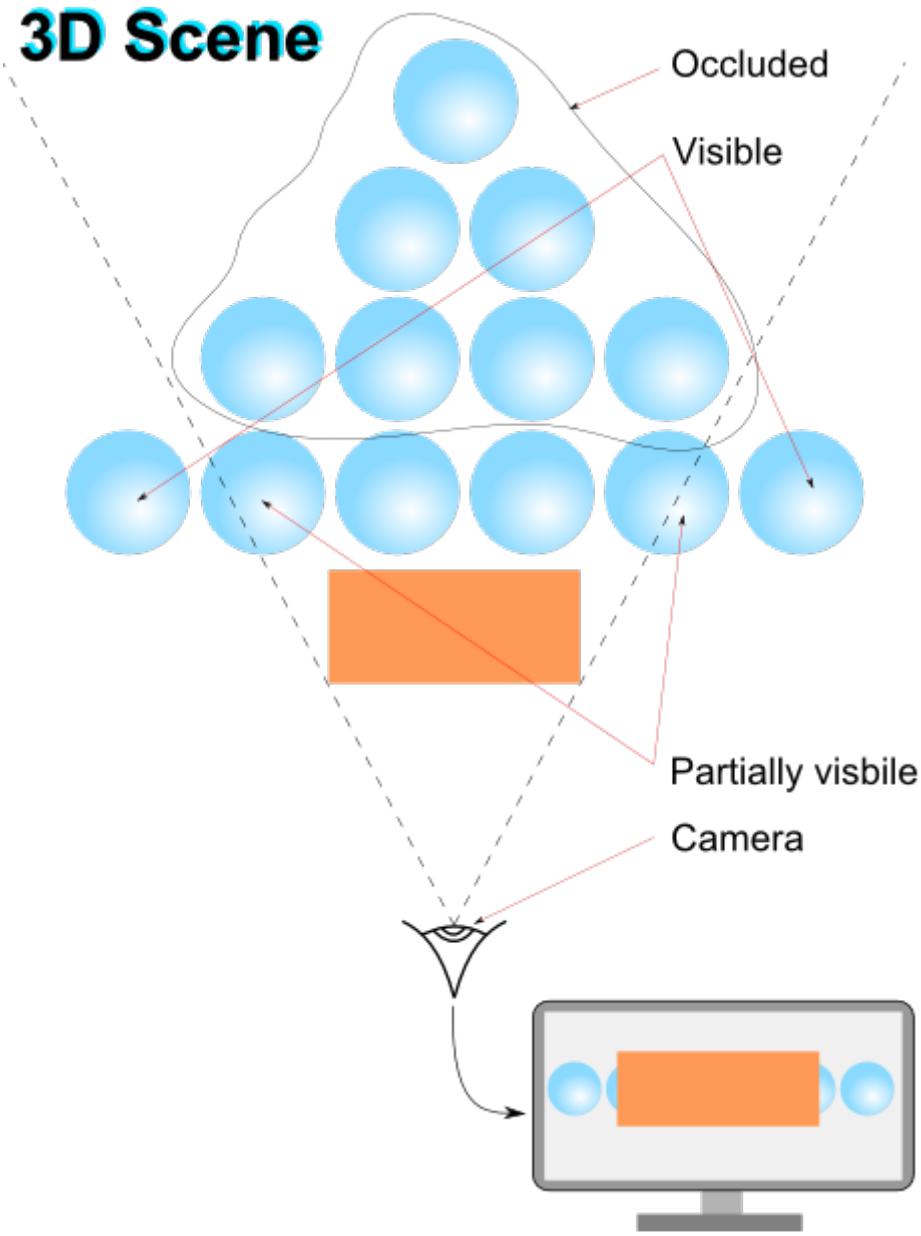
Occlusion Query Types

There are three types of occlusion queries:

- Query::QueryAnySamplesPassed: query determines, if any sample passes the depth test.
- Query::QueryAnySamplesPassedConservative: similar to QueryAnySamplesPassed type except that the implementation may use a less accurate algorithm, which may be faster, but at the cost of more false positives.
- Query::QueryTransformFeedbackPrimitivesWritten: query determines how many vertices have been written into bound transform feedback buffer.

Conditional Rendering Example

The figure below depicts the demand for occlusion queries to determine and subsequently discard occluded objects in a highly populated scene:



Find below an abstract code sample for occlusion queries to obtain information whether any pixels have been rendered or not:

```
SharedPointer<Query> query;
query = Query::Create(Query::QueryAnySamplesPassed);
query.Upload();
query.Begin();

// Do some rendering here.
```

```
query.End();

// Process other operations to provide GPU with sufficient time to complete query or skip query

if (query.IsResultAvailable()) {
    if (query.GetResult() > 0)) {
        // Samples have been written to framebuffer while query was active, this is between quer
        // In case low resolution model was rendered, render next frame with high resolution mod
    }
    else {
        // No sample has been written to framebuffer while query was active.
        // In case high resolution model was rendered, render next frame with low resolution mod
    }
}
```

Occlusion Culling

Overview

Occlusion Culling is a feature that disables rendering of objects when they are not currently seen by the camera because they are obscured by other objects. The feature can be enabled by setting the render strategy for camera to OcclusionCullingCameraRenderStrategy. During the rendering, all nodes initialized by the strategy will issue occlusion queries. If query results are available from the previous frame, they are evaluated and all nodes deemed occluded will have their bounding box rendered invisibly instead of the node itself. Visible nodes, or nodes not having a QueryProperty attached, will be rendered as usual. A new query to determine visibility for the next frame will be issued regardless the result of the previous query.

> A performance increase by using occlusion culling can only be expected when objects occlude each other, and occluded objects consist of lots of vertices, have expensive fragment shaders, or both.

Workflow

- Attach an OcclusionCullingCameraRenderStrategy to the camera.

```
    m_occlusionCullingStrategy = CANDERA_NEW(OcclusionCullingCameraRenderStrategy)();
    m_camera->SetCameraRenderStrategy(m_occlusionCullingStrategy);
```

- Initialize the rootNode's subtree for use by the render strategy.

```
m_occlusionCullingStrategy->Initialize(m_group, Query::QueryAnySamplesPassedConservative);
```

> This camera render strategy requires nodes to have a valid bounding box. This camera render strategy reaches its full potential when nodes were sorted front to back first.

- Render camera

```
m_gdu->ToRenderTarget3D()->Activate();
RenderDevice::ClearDepthBuffer(1.0f);
Renderer::RenderAllCameras();
```

- OcclusionCullingCameraRenderStrategy interface provides also methods to get the number of occluded and non-occluded nodes since last render pass by using the methods OcclusionCullingCameraRenderStrategy::GetOccludedNodesCount() and OcclusionCullingCameraRenderStrategy::GetVisibleNodesCount().
- Before destroying the render strategy remove the association with nodes by calling the OcclusionCullingCameraRenderStrategy::Reset() method

```
m_occlusionCullingStrategy->Reset(m_group);
CANDERA_DELETE(m_occlusionCullingStrategy);
```

> The nodes removed from the subtree are automatically detached from the render strategy

Occlusion Culling in SceneComposer

Refer to chapter Occlusion Culling to see how to configure occlusion culling in SceneComposer.

Revision #8
Created 2023-02-17 08:09:30 UTC by Tsuyoshi.Kato
Updated 2025-01-24 10:01:28 UTC by Elisabeth Zauner