Drawcall Batching/Geometry Instancing

Description

This chapter describes how Candera's Drawcall Batching utilizes Geometry Instancing and how to set it up.

Introduction

OpenGL ES3.0 introduced a feature called Geometry Instancing. With this feature, a vertex buffer can be drawn several times using a single draw call. This single draw call uses exactly one shader and render state for all instances of that vertex buffer. The instances can have different shader parameters to individually customize their appearance on screen. Basically, this feature renders multiple copies of the same mesh at once, with the ability to parameterize each copy.

Use Case

Whenever objects share a mesh, render state, shader and textures, these are potential candidates for geometry instancing. E.g. trees that only have a different size and color, houses which share the same shader using a uniform as an offset for a shared texture atlas, street lamps, or any other recurring or repeating geometry that only differs in shader parameters is a use case for geometry instancing.

Motivation

Submitting drawcalls to the GPU is a relatively slow operation due to the necessary overhead caused by OpenGL and the driver. When drawing thousands of small objects individually, chances are that this overhead is leaving the GPU underutilized, and thus becoming a bottleneck. In such a scenario, batching several objects together to be submitted in a single drawcall decreases the total CPU overhead to be performed, which in turn increases overall performance.

> The GPU still has to perform the same amount of work, so if the bottleneck of your application is the GPU (e.g. pixel or vertex operations are the limiting factor), don't expect significant performance gains by using geometry instancing.

Geometry Instancing and Shaders

Dedicated shaders are required to utilize geometry instancing. In a shader the reserved keyword *gl_InstanceID* represents the index of the instance currently being rendered. This index can be used to fetch instance specific parameters from arrays of uniforms.

Example of a geometry instancing vertex shader:

```
/* Uniforms */
#define MAX_INSTANCE_COUNT 100                 // Used for the size of the uniform arrays
uniform mat4 u_MVPMatrix[MAX_INSTANCE_COUNT];  // Model-View-Projection matrices for each instar
uniform vec4 u_Color[MAX_INSTANCE_COUNT];      // Color for each instance

/* Attributes */
in vec4 a_Position;

/* Varyings */
out mediump vec4 v_Color;

void main(void)
{
    /* Transform vertex into world space using the MVP matrix of the instance indexed by gl_Inst
    gl_Position = u_MVPMatrix[gl_InstanceID] * a_Position;

    /* Pass the color of the instance indexed by gl_InstanceID to the pixel shader */
    v_Color = u_Color[gl_InstanceID];
}
```

The pixel shader for the vertex shader above:

```
in mediump vec4 v_Color;
out mediump vec4 FragColor;

void main(void)
{
    FragColor = v_Color;
}
```

> OpenGL ES3.0 pixel shaders can only index uniform arrays using a constant index. Therefore the vertex shader has to index and pass any uniforms required by the pixel shader.

This shader uses a different color and model view projection matrix for each instance of the mesh. It supports drawing of up to 100 (MAX_INSTANCE_COUNT) instances. Therefore the model view

projection matrix array and the color array have a size of 100 to hold the individual values for each instance. The maximum array size that can be used depends on the total number of uniforms in the shader as well as the target platform.

> Array sizes that are too big for the platform will not throw an error during shader compilation, but only during linking the shader. Therefore you have to export shader binaries to verify if the shader can be used on your target platform.

Geometry Instancing in Candera

Candera does not expose geometry instancing explicitly, instead it is used automatically (if possible). This approach is called **Drawcall Batching**.
After renderable objects have been culled and sorted (ideally by the BatchOrderCriterion to produce as big sequences of batchable objects as possible), the Renderer identifies sequences of objects that can be batched, and submits those objects in a single drawcall using geometry instancing. Each such sequence is a batch. Each batch corresponds to one draw call.

Prerequisites for Drawcall Batching

The following list of prerequisites must be met to facilitate draw call batching in Candera:

- The target platform must support geometry instancing (i.e. OpenGL ES3.0 or better).
- The object must be a Mesh or PointSprite (Billboard, LineList, MorphingMesh are currently not supported).
- The object must have an Appearance.
- The Appearance must not be a multi-pass appearance.
- The Appearance must have a Shader with a maximum instance count bigger than 1. I.e. the shader supports instancing.
- The Appearance must have a ShaderParamSetter.
- The ShaderParamSetter must support instancing. (Candera's ShaderParamSetter and GenericShaderParamSetter support instancing. Existing custom AbstractShaderParamSetter implementations will need to be adapted to support instancing.)
  - If a GenericShaderParamSetter is used with lights activation enabled, the light coordinate space must be World space.
- Objects to be batched must share:
  - the VertexBuffer
  - the RenderMode
  - the Shader
  - the Textures

> If objects share an Appearance, they automatically share the RenderMode, Shader, and Textures, too.

How to increase the Drawcall Batching rate.

Share as much vertex buffers and appearances between objects as possible. If sharing the appearance is not possible (e.g. because different materials, and/or shader parameter setters have to be used, which results in different appearances), share as much shaders, render modes and textures as possible.

The Renderer does not alter the order of contents of RenderOrderBins. In order to maximize the drawcall batching rate (i.e. the amount of objects that are batched versus the total amount of objects), objects have to be sorted by the criteria matching the prerequisites listed above. Candera'a BatchOrderCriterion offers two sorting modes to facilitate batching:

- Sort by Appearance
- Sort by Shader and RenderMode

Both modes sort by the Batchable flag (used by Mesh and PointSprite), the VertexBuffex, and either Appearance or Shader plus RenderMode to produce batches.

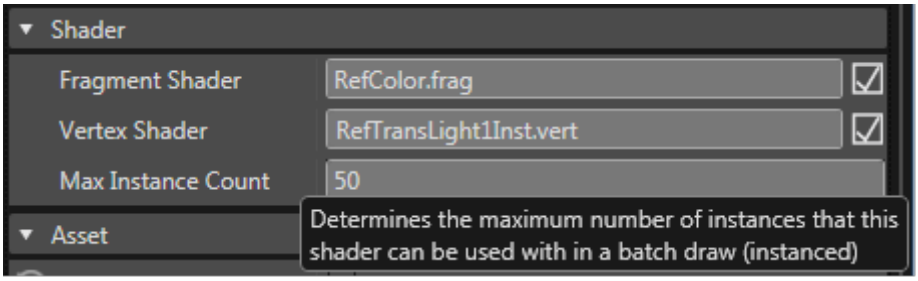
Drawcall Batching in SceneComposer

To configure SceneComposer solutions for drawcall batching, the following steps have to be performed:

Use an instanceable shader

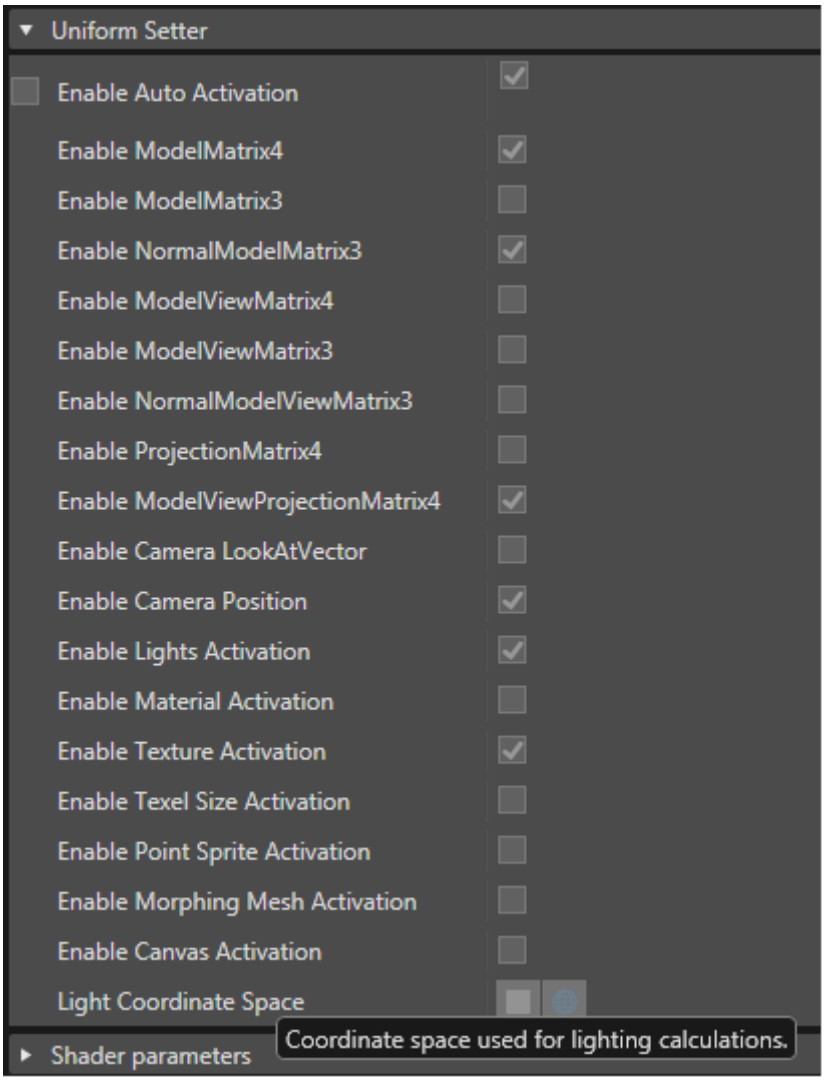
The shader of an object must support instancing. See this section for a detailed explanation.

If a shader was recognized as instanceable by SceneComposer, "Max Instance Count" listed in the Properties of the Shader is bigger than one. The shader in the following example is able to draw batches of up to 50 instances.



Configure the Uniform Setter

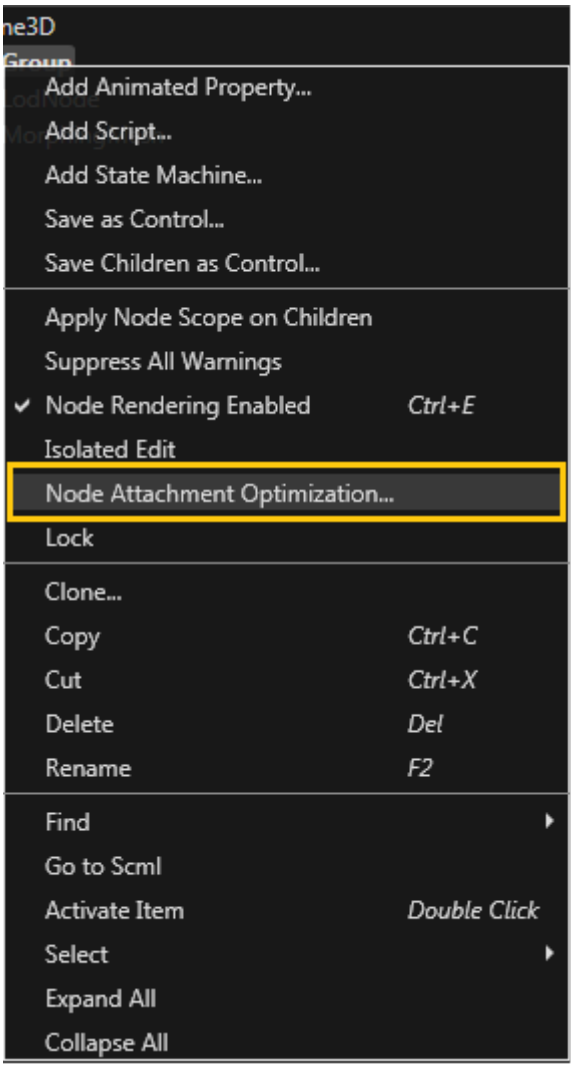
If the shader uses lighting, the Light Coordinate Space must be set to World.



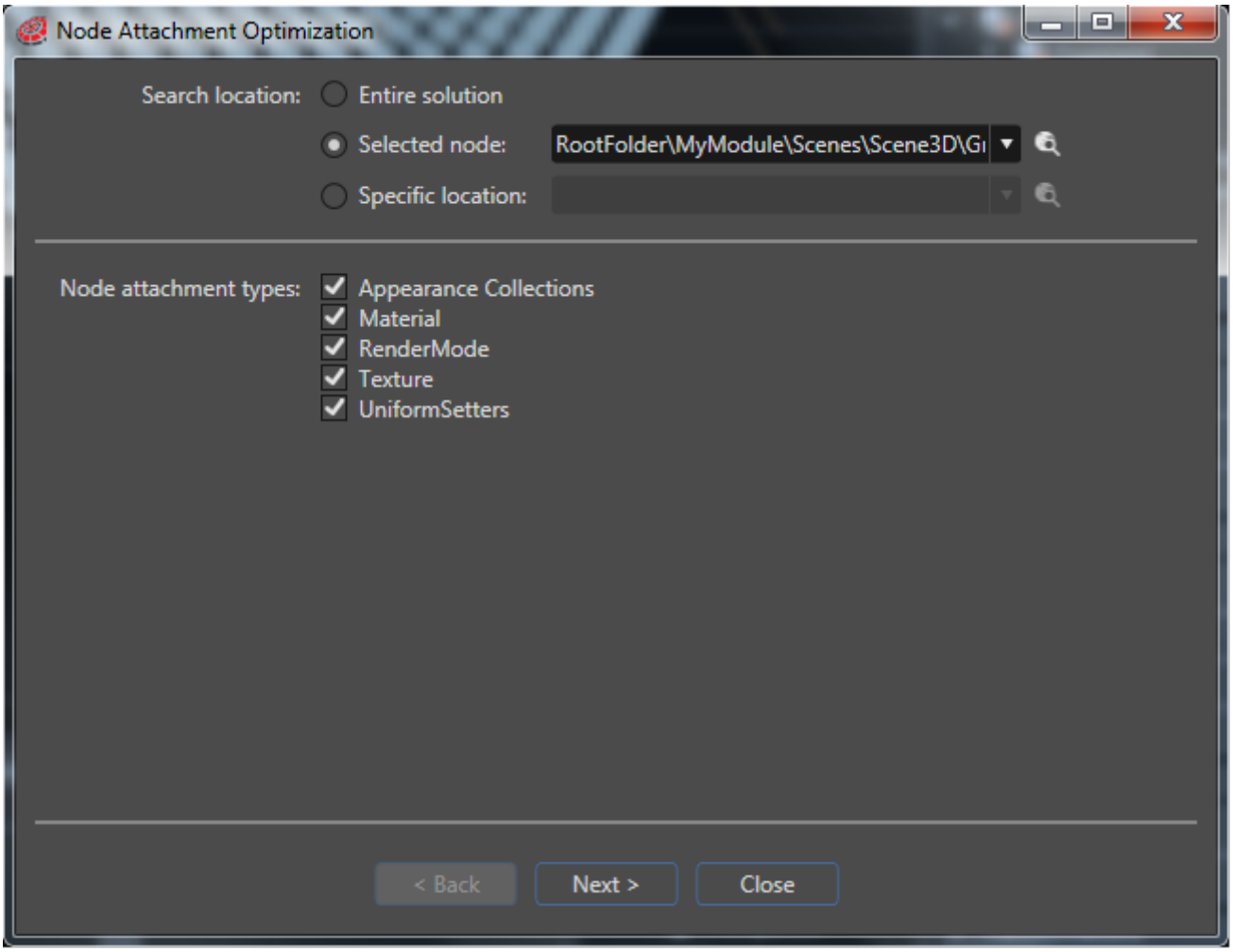
Share Assets

To utilize drawcall batching, vertex buffers, shaders, render modes and textures have to be shared as detailed in this section.

SceneComposer offers the option to optimize asset usage by identifying assets that are identical and substituting these assets by a single shared instance. After selecting a node or a group, and pressing the right mouse button, the following context menu will appear:



Selecting "Node Attachment Optimization" will launch the following window, where you can select which assets should be optimized, and initiate the optimization:



Sort Render Order Bins using the Batch Order Sort Criterion

To maximize the batching rate, use "Batch Order" as the Render Order Bin's sort criterion.



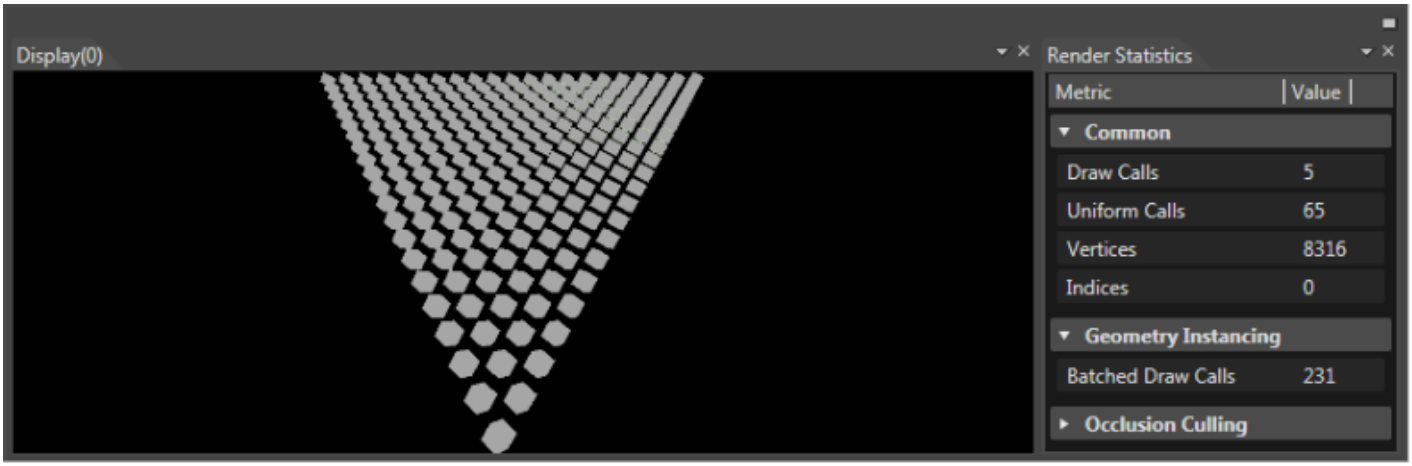
If the scene uses a lot of different appearances (e.g. with different Materials), that share a shader, render mode, and textures, the batch order criterion "SortByShaderAndRenderMode" might achieve a better batching rate then "SortByAppearance". Use the Renderer Statistics to verify which criterion works best with the given content.

> A Render Strategy set on a camera can influence batching. E.g. the Occlusion Culling, and the Benchmark Render Strategy do not utilize drawcall batching due to their

specific needs.

Renderer Statistics

The Renderer Statistics window (which can be opened via View->Renderer Statistics), provides per frame information about what is being drawn by the cameras in the current scene. The values are updated in realtime, and directly correspond to the rendering performed in the "Display" window.



In this example 231 cubes are rendered using 5 draw calls. All 231 cubes are batched (i.e. instanced), therefore they are counted as "batched draw calls".

Revision #8
Created 2023-02-17 08:09:52 UTC by Tsuyoshi.Kato
Updated 2025-01-24 08:30:12 UTC by Elisabeth Zauner