The Rule of Three

Successfully uploading geometry data

Overview

The rule of three is a general guideline to help you successfully transmit a set of geometric data you wish to rasterize via a hardware accelerated 3D graphics API. This document will outline the three rules and highlight some of the surrounding issues that must be considered when adjusting them to match up.

The Rule of Three

The following three things must match:

1. The C/C++ Vertex Structure
2. The API Vertex Description
3. The HLSL Input Vertex Structure

You can have multiple sets of three, but each set must match up.

Common Pitfalls

When adjusting the rule of three for a new geometric format, watch out for these common oversights.

1. **The C/C++ Vertex Structure**

* Supplying the wrong number of vertices when creating your vertex buffer.
* Supplying the incorrect amount of bytes when creating a vertex buffer.
* Accessing the vertex data of an std::vector<> improperly when populating a vertex buffer.
* Ignoring any index data that comes with(and is required to draw) your vertex data.
* The first three bullet points again but replace the word vertex with index.
* Not setting or binding the correct Vertex and/or Index buffer before calling a draw function.
* Not using the proper Index variable format when setting/binding. (Ex: 16bit vs. 32bit index)
* Not using the correct Draw function. (Ex: Draw vs. DrawIndexed)

1. **The API Vertex Description**

* Not having the correct byte offsets for each element from the start of the structure.
* Not specifying the correct byte FORMAT for each individual structure variable.
* Not specifying the correct location(the order) of each structure variable.
* Not specifying the correct SEMANTIC name for each structure variable. (D3D Only)
* Not informing the API there are now more(or less) attributes in use than before.
* Not adjusting the vertex stride setting to reflect the new size of your vertex.
* Not setting/binding the correct vertex description before drawing. (D3D11/OpenGL)
* OpenGL: Forgetting to enable each of the new vertex attributes.
* OpenGL: Not casting your attribute offsets directly to void pointers. (Ex: (void\*)12)

1. **The HLSL Input Vertex Structure or GLSL Vertex Shader Input Variables**

* The shaders input does not have the same arrangement or size of variables as C/C++ does.
* The outgoing Vertex’s position is not a 4-float vector type with a proper W.
* HLSL: The new input structure has no SEMANTICS linked to each of its variables.
* HLSL: No outgoing Vertex structure was created to utilize the new attributes.
* HLSL: The outgoing Vertex has no semantics and/or does not make use of SV\_POSITION.
* HLSL: The vertex shader’s main() entry point still returns the older vertex format.
* HLSL: The vertex shader’s main() entry point duplicates the : SV\_POSITION semantic.
* GLSL: Not using **layout(location = X)** on **in**put variables to clearly define their source data.
* GLSL: Not providing an **out** variable to pass extra interpolated attributes. (Ex: UVs)

Verification of Correct Geometry

Following the rule of three correctly will hopefully allow you to draw your geometry. However, this is not a guarantee; other things(like matrix uniforms) could still be wrong causing nothing to show up on screen(or be distorted).

If you have poured over the rule of three pitfalls above and are pretty sure everything is correct, but nothing is showing up; then it is a good time to use a more direct method to inspect the geometry.

Geometry Inspection with RenderDoc

If you have not already, take a moment to install the RenderDoc. You can find a link to the software in the FAQ section of any lab doc.

Once installed open the software and use it to capture a frame from your latest executable:

<https://renderdoc.org/docs/how/how_capture_frame.html>

After you capture a frame, double click it to load it for inspection. Go to the left side panel and expand it until you find the draw call that should be working but currently is not. Select it, and then on the right side find the mesh viewer tab to get a direct preview of your geometry.

If it shows up correctly then your rule of three is correct and the issue lies elsewhere (probably a bad uniform matrix). If it does not look correct, then the rule of three is still wrong and you can continue using RenderDoc to look at the various pitfalls discussed above from the GPU’s perspective.