March 18, 2017

Stop using the if() node

I admit, that title is a bit misleading.
Too many time I see people using the "if()" node in UDK/UE4 for some specific use case while there are other way to avoid the if() node and therefore its cost.
(All my tests and comparisons were made with the UE4 4.12.3 and an NVIDIA GPU.)

The cost

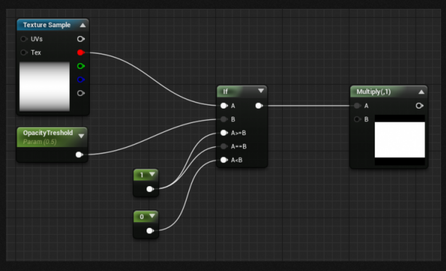
Branching on modern hardware has less impact that a few years ago, but if it can be avoided by some simple math it's often a better solution. The gain most of the time is that you get a fixed cost for your shader and its therefore much easier to predict, optimize and even profile performance wise. Now let's quote someone a bit more clever than me :

> Condition codes (predication) are used in older architectures to emulate true branching. If-then statements compiled to these architectures must evaluate both taken and not taken branch instructions on all fragments. The branch condition is evaluated and a condition code is set. The instructions in each part of the branch must check the value of the condition code before writing their results to registers. As a result, only instructions in taken branches write their output. Thus, in these architectures all branches cost as much as both parts of the branch, plus the cost of evaluating the branch condition. Branching should be used sparingly on such architectures. NVIDIA GeForce FX Series GPUs use condition-code branch emulation in their fragment processors.

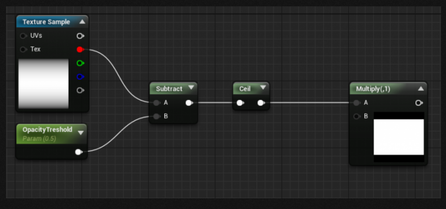
Source from GPUGems2

Binary mask comparison

One of the many usage of the if node, is to use it against a mask to decide if a mesh should be visible at a given pixel. This is often used to make meshes appear or disappear inside a game. Generally you use a single float parameter as a treshold to define if the pixel is visible or not. If the pixel is brighter than the floating value, then we return 1, otherwise we get 0 :



Now here is my better version :

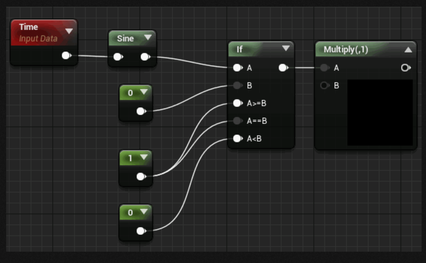


Instead of branching, here we use a simple subtraction and then call the ceil function. Subtracting here means we get negative values. The ceil function return 1 if the input value is greater than 0. So combining the subtract and the ceil we get the same result as the if node. 😉

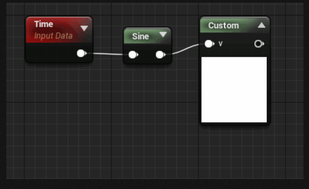
In the end we also gained some instructions. We went from 86 instructions in the base pass with the if() to 83 with the ceil.

Sign

Another use of the if() node could be to know if a sin() is over or below 0. This can be useful to creating flickering effects (instead of soft modulations). With the if() node we could get something like this :



There is however something much more simple to do :



Okay, okay, the image by itself doesn't help. Basically the node "custom" contains juste the following line : "return (v > 0)". This is still a comparison like the if() conditional, but much simpler and faster to perform. We could have also used the "sign()" function provided in UE4, but it just hide an if() node behind the hood.

As for the instruction count, we went from 86 instructions in the base pass with the if() to 84 with the custom node. Calling the native sign() function in a custom node seems to cost as much as the if() node itself.

You can still use the if() node

I won't bite. The if() node is still handy for many things. Modern GPU handle it without too much troubles anyway. Daniel Wright (senior rendering engineer at Epic Games) also mentioned to me that you can use additional HLSL commands to improve the performances of a condition rather than using alternative maths to get the same result. More information about flatten over here.

POSTED IN
Blog, Code, Shader, Technical, Unreal Development Kit, Unreal Engine 4