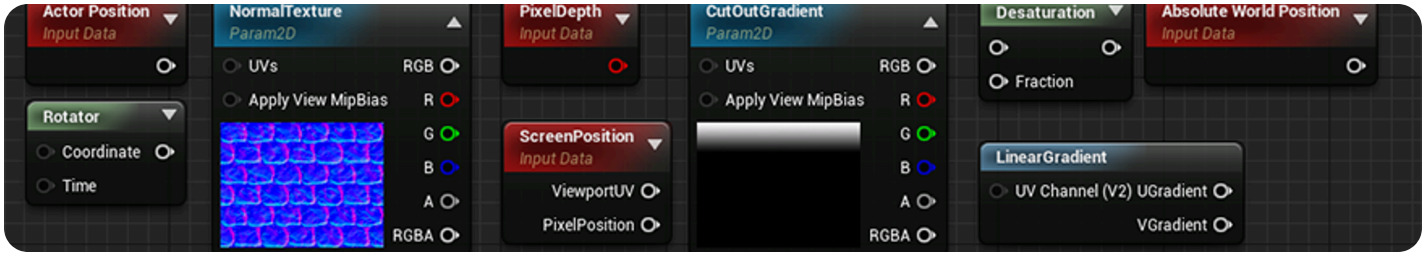
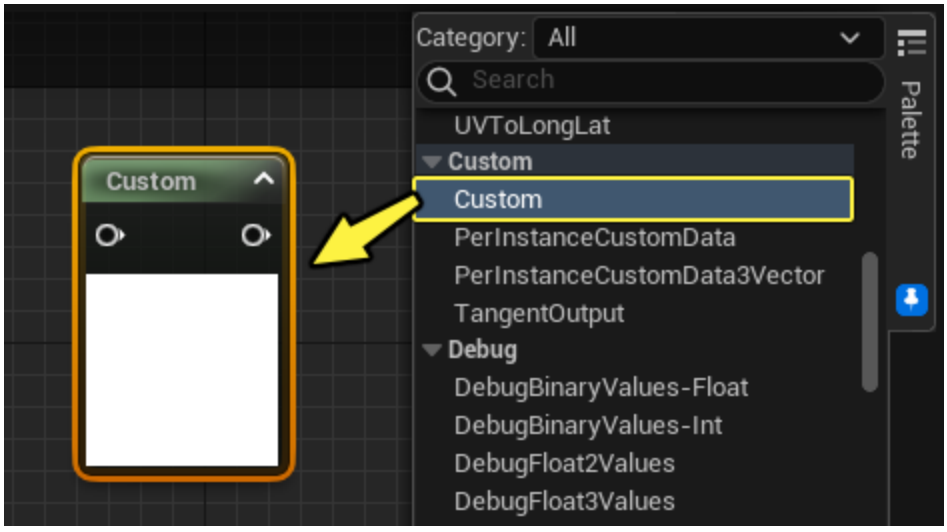
Custom Material Expressions

Material expressions that allow the use of custom, plain shader code.

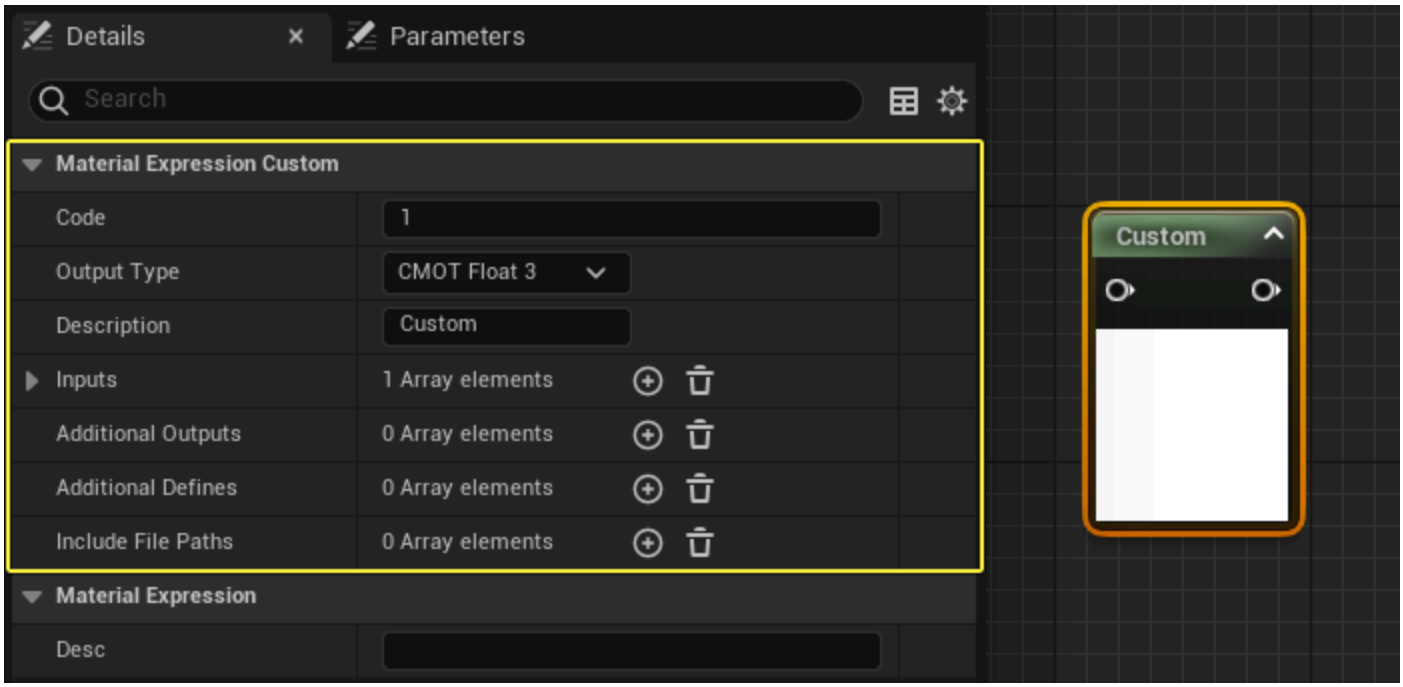


The **Custom** Material Expression enables you to write custom HLSL shader code operating on an arbitrary amount of inputs and outputting the result of the operation.

You can insert the Custom expression from the **Custom** category in the Material palette or from the right-click search menu in the Material Graph.



When the Custom node is selected, the following properties display in the **Details Panel**.



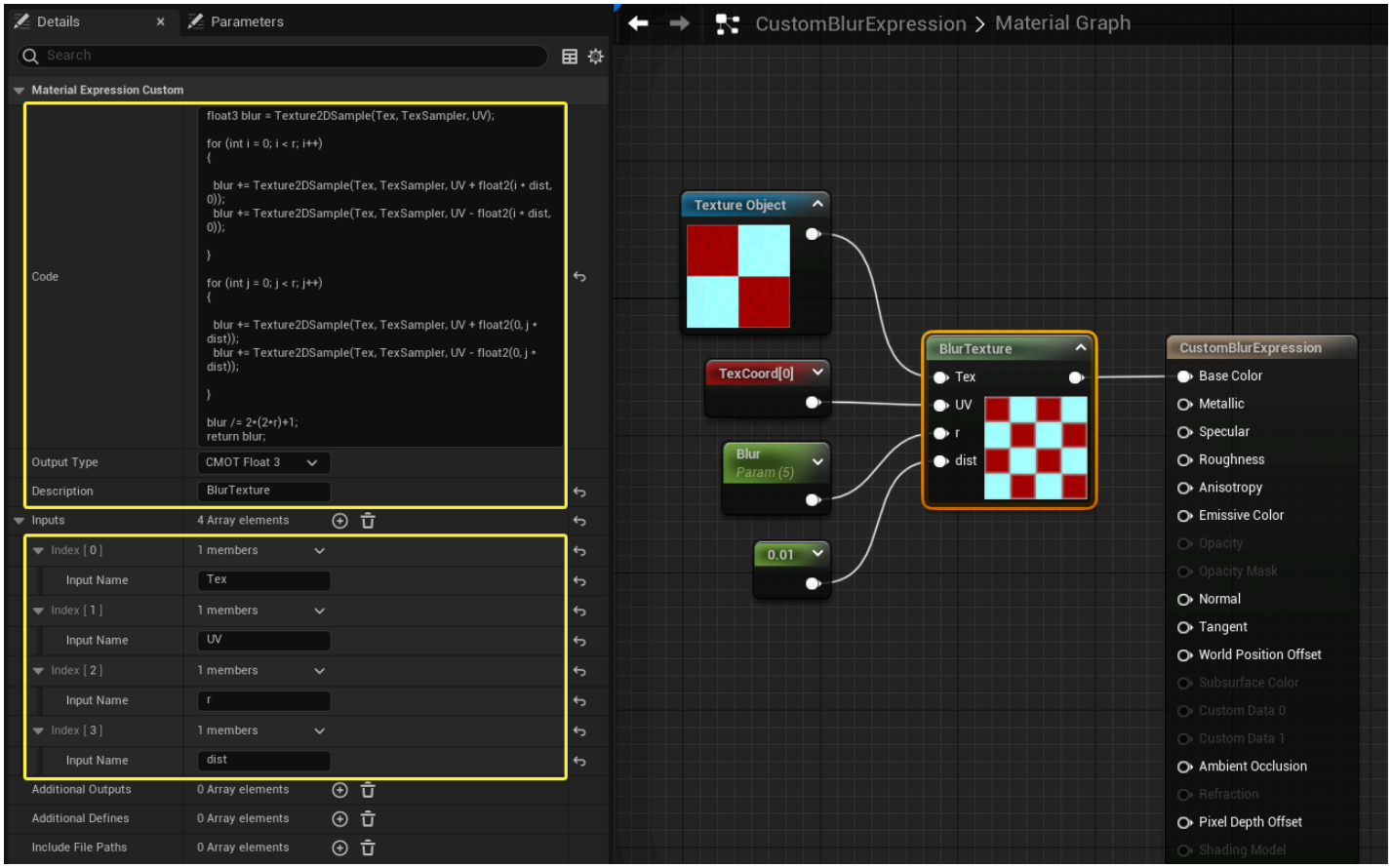
Property	Description
Code	Contains the shader code the expression will execute. (See warnings below)
Output Type	Specifies the type of the value output by the expression.
Description	Specifies the text to display in the title bar of the expression in the Material Editor.
Inputs	The array of inputs used by the expression.
Input Name	Specifies the name of the input. This is the name displayed on the expression in the Material Editor as well as the name used within the HLSL code to reference the input's value.
Additional Outputs	Enables you to define additional output pins to appear on the custom expression. When you add an output you must expand the array element and specify its **Output Name** and **Output Type** properties.
Additional Defines	Enables you to add additional Defines required by your custom HLSL code. When you add an element to the array you must specify the **Define Name** and **Define Value** properties.

Property	Description
Include File Paths	Specifies the file path to shader code you would like to include from source files outside the common shader paths Unreal Engine already provides.

Using the Custom Material Expression

Add as many inputs as you need to the **Inputs** array, and name them. Then type or paste your HLSL code in the **Code** property field. You can type either a full function body with return statements as shown in the example, or a simple expression such as **Input.bgr**. You must also specify the output data type in OutputType.

The example shown below is a Custom expression that can blur a Texture Object based on the value in a Scalar Parameter.



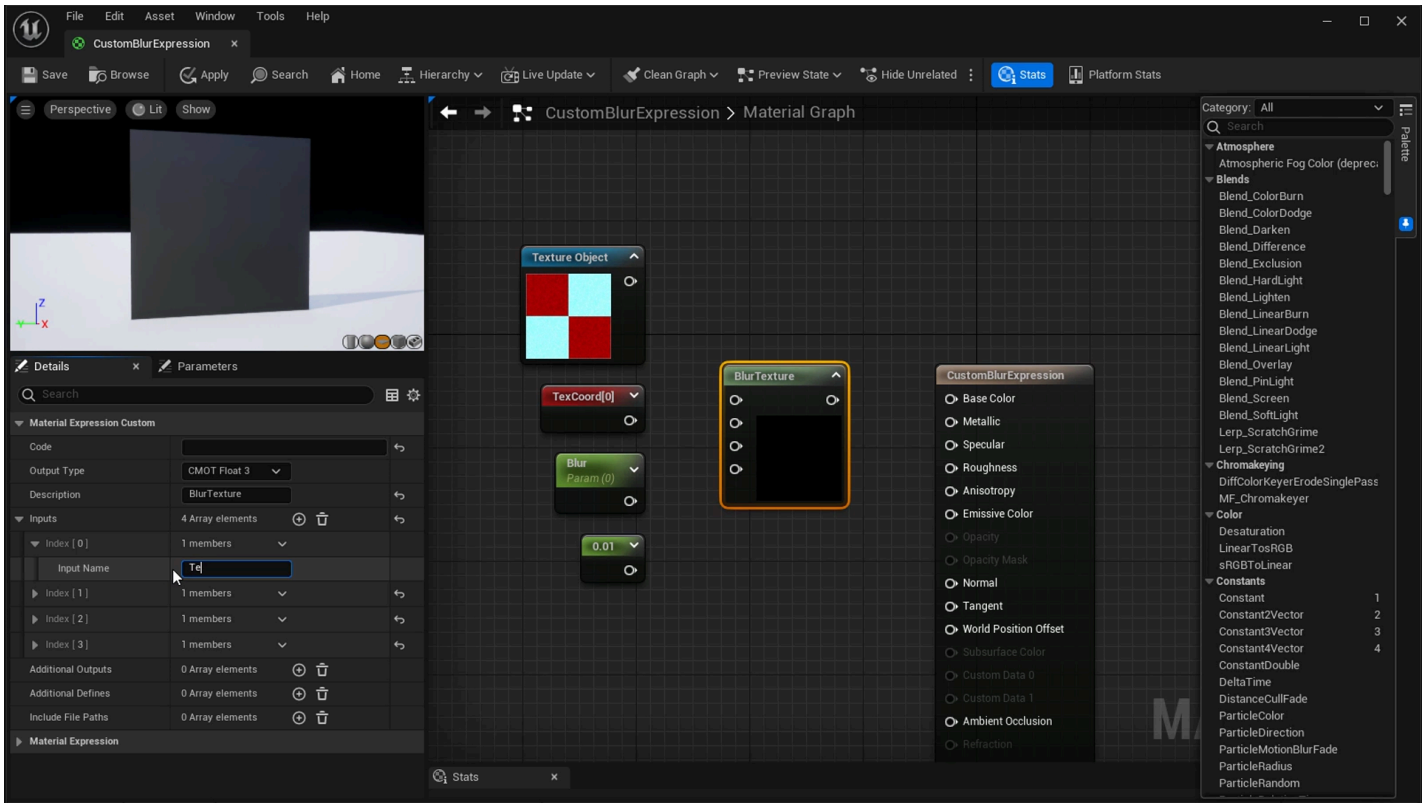
Click for full image.

Here is the code that was used above so that you can try the Custom node for yourself. Copy and paste the following text into the **Code** field in the Details Panel properties.

```
float3 blur = Texture2DSample(Tex, TexSampler, UV);

for (int i = 0; i < r; i++)
{

blur += Texture2DSample(Tex, TexSampler, UV + float2(i * dist, 0));
blur += Texture2DSample(Tex, TexSampler, UV - float2(i * dist, 0));

}

for (int j = 0; j < r; j++)
{

blur += Texture2DSample(Tex, TexSampler, UV + float2(0, j * dist));
blur += Texture2DSample(Tex, TexSampler, UV - float2(0, j * dist));

}

blur /= 2*(2*r)+1;
return blur;
```

Copy full snippet

The following video shows the creation and result of a Custom expression using the above HLSL code.



Known Issues

The following sections detail some common pitfalls to be aware of when writing custom Material expressions.

Input Arguments Are Function-Local

Note that Custom expressions always have a return value because they are wrapped inside another function. All input arguments (like Tex, UV, r, and dist in the above example) are declared as parameters of that outer function, so their scope of visibility is function-local.

Sometimes developers write structures with member functions and expect these parameters to be accessible inside that structure, but they cannot access function-local parameters. Therefore, you must copy those parameters by hand when member functions are used inside the Custom expression.

The example below would fail to compile (assuming the same example as above).

```
1 struct InnerStruct
2 {
```

```
float4 Run()
{
// ERROR: Tex, TexSampler, and UV is not accessible within InnerStruct
return Texture2DSample(Tex, TexSampler, UV);
}
};
InnerStruct S;
return S.Run();
```

Copy full snippet

Syntax Errors When Using HLSLcc Compiler

Some shader backends that use the older HLSLcc compiler report syntax errors on HLSL types such as vector and matrix. Try to be explicit with your data types and use **float4** or **float4×4** respectively. We keep fixing those bugs but at the same time we are moving our efforts to the new DirectXShaderCompiler (DXC).

Warnings

- **Using the custom node prevents constant folding and may use significantly more instructions than an equivalent version done with built in nodes!** Constant folding is an optimization that Unreal Engine employs under the hood to reduce shader instruction count when necessary. For example, an expression chain of `Sin >Mul by parameter > Add to something` can and will be collapsed by Unreal Engine into a single instruction, the final add. This is possible because all of the inputs of that expression (parameter) are constant for the whole draw call, they do not change per-pixel. Unreal Engine cannot collapse anything in a custom node, which can produce less efficient shaders than an equivalent version made out of existing nodes. As a result, it is best to only use the custom node when it gives you access to functionality not possible with the existing nodes.
- **Shader code written in a custom node must be valid HLSL.**