Programmatic and Dynamic Post Process

William Tallarico

Objective:

I wanted to work with post process materials, specifically how they integrate with the Blueprint systems to turn on, off and change other effects dynamically. MNy objective was to make a few post process materials, and have a way to integrate them dynamically into the demo.

Type of Material: Outline Shader

To fulfill this effect, I decided to create an Outline Effect. This effect is common to many games to highlight certain objects. I have been exposed to them before in Unity, and wanted to implement one into Unreal Engine.

Part 1: Outlines [https://ameye.dev/notes/rendering-outlines/]

I am not a technical artist, and have never implemented an outline shader, so I started with some research into approaches. The one I ultimately went with was the *Robert's Cross* approach. My reasoning is ease of implementation and optimization. This requires 5 samples for each pixel, which is a relatively small number of operations for a shader. The results look very good, and only have some issues which depend on the data taken into account for the disparity.

Edges are detected through disparity in scene data:

- Depth buffer
- Normals
- Colors

I went with utilization of the depth buffer for this. Some approaches utilize all 3 to determine outlines. I used the depth for speed of implementation and optimization, as it is less operations.

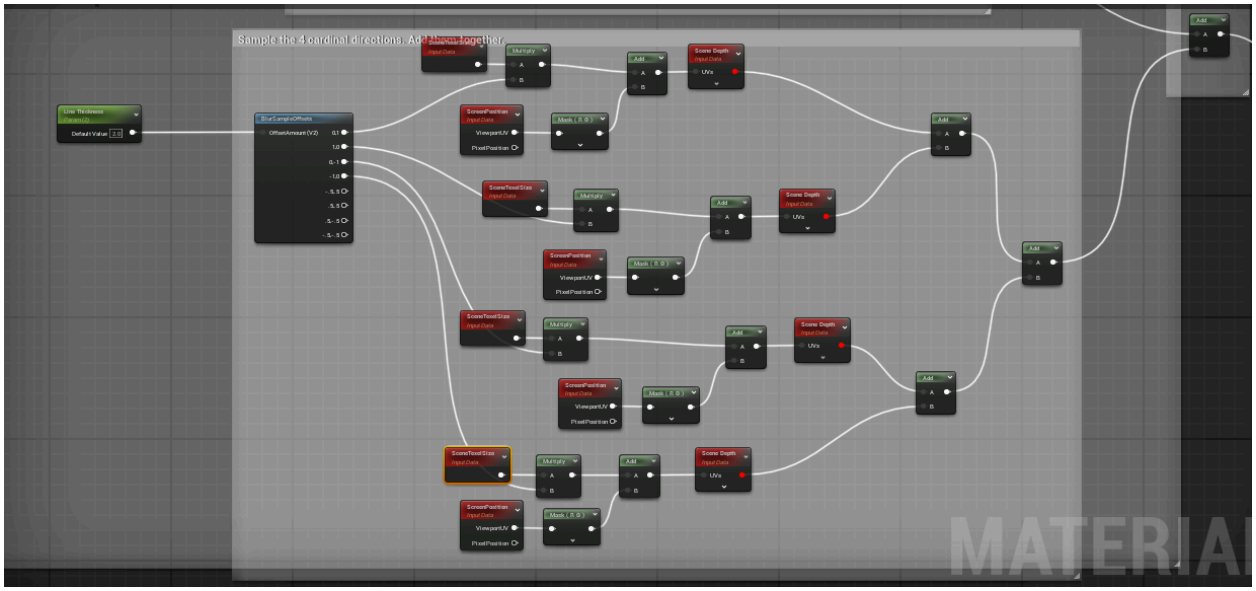
How it works:

I sample the depth from 4 directions. These are used to determine the difference between depths. Then I sample the depth from the front (camera look at). This is used to offset the 4 other samples, to create the outline.

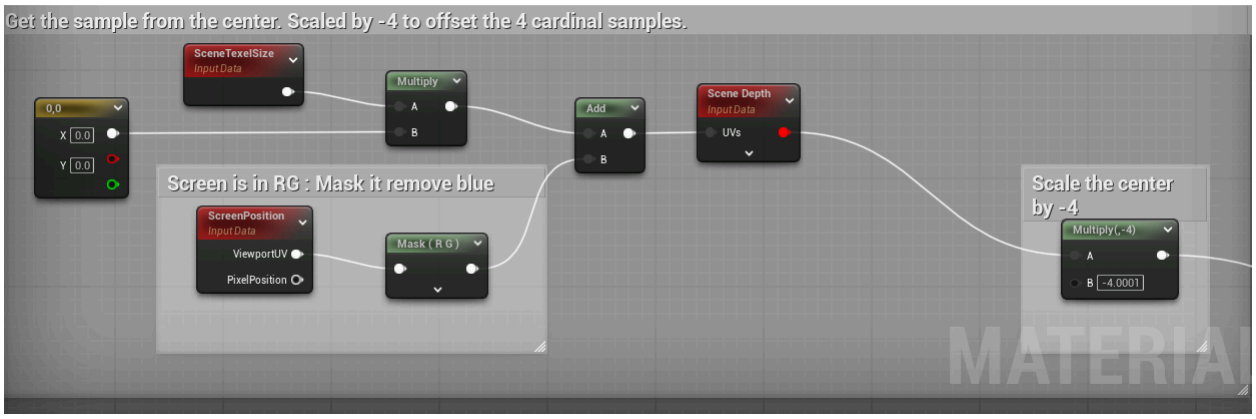
Issues: Depth at a long distance is the same, so at certain angles objects and the floor can outline unintentionally. Scaling up the disparity between the centers slightly helps with this. A combination with Normal/Color sampling could assist with this issue.

Center sample - scaled by -4.0001. Normally this would be scaled by 4, but I get slightly better results from scaling by a bit more.

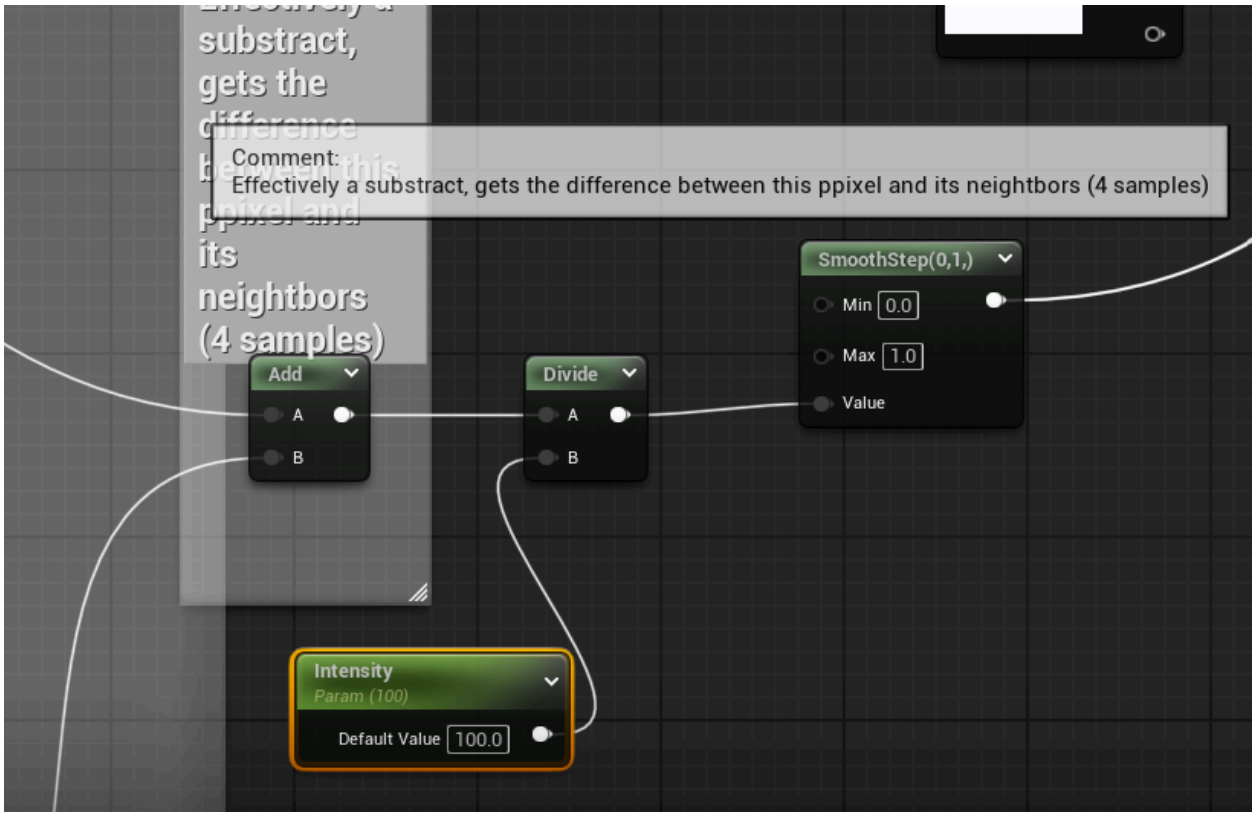
4 Direction Sample



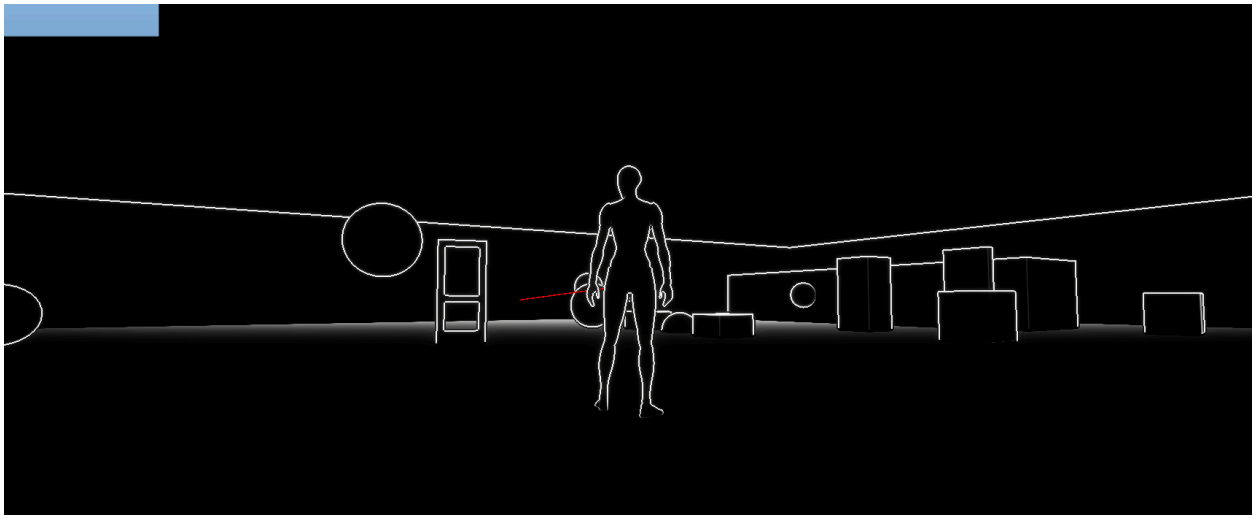
Center Sample.



Add together, then divide by an intensity and smoothstep. This is then lerped with the scene color to apply the outline.



In my approach, I made the scene color black, this is optional and scene color can still be used in the outline.



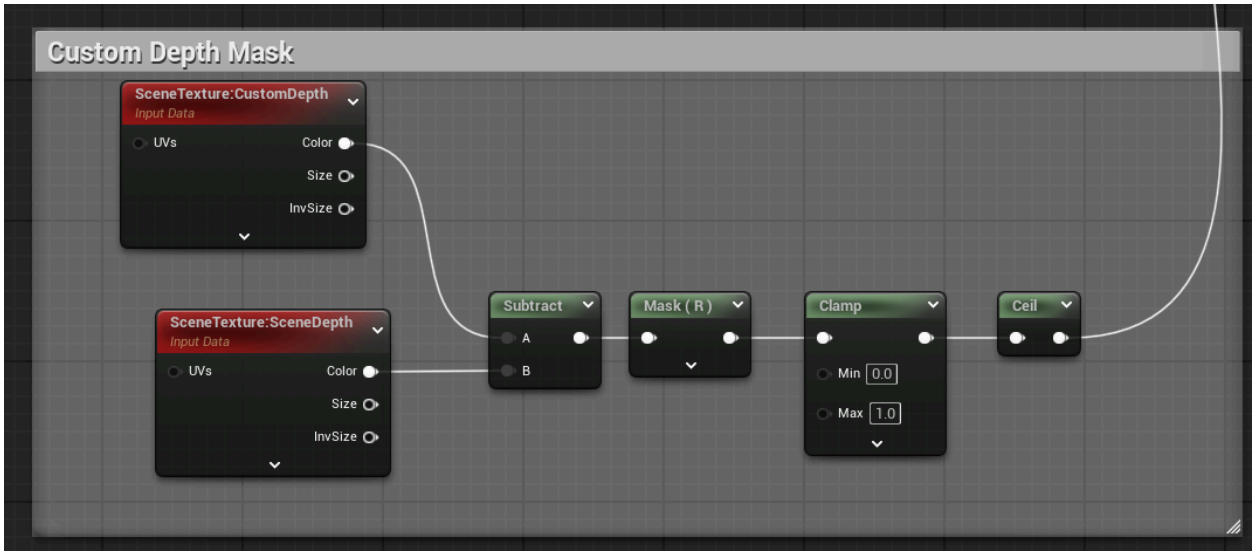
Refactor:

After I wrote this section of the devlog I further refined this system. I created a material function that took in parameters for intensity and line width. It then performed the same calculations. This was due to multiple materials I was working on needing the calculation. Furthermore, I utilized named reroute nodes to clean up the blueprint.

Part 2: Depth Masking

The next challenge was getting the outline to appear on certain objects and not others. The simplest approach I encountered was to utilize the custom depth feature. This assigns a special custom depth to objects, which can be utilized in materials.

This ability to utilize custom depth can be turned on and off in inspector or programmatically. To start, I outlined only the objects with the custom depth enabled. This led to the outline being shown on certain objects.



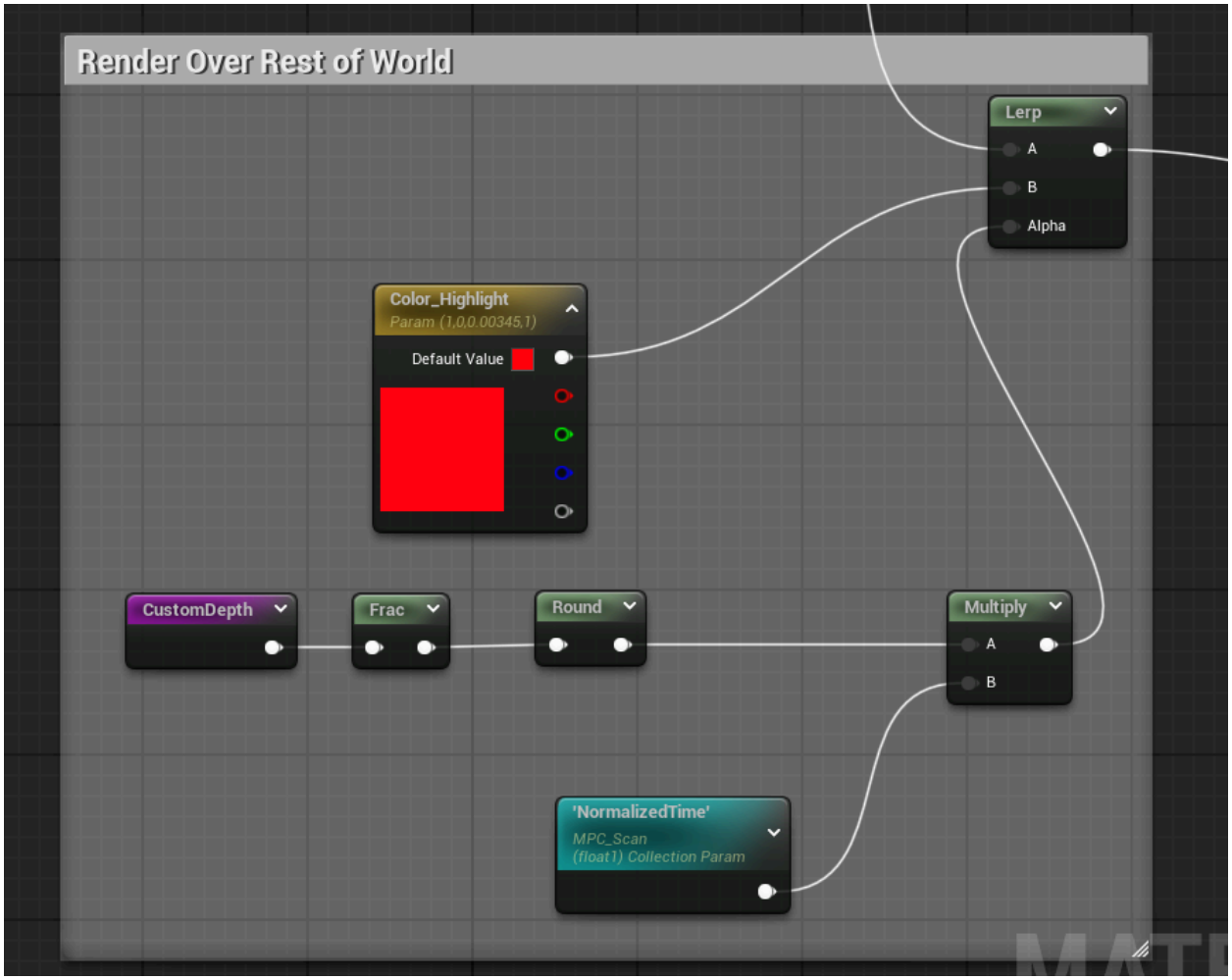
This returns 0 for objects with no custom depth, and 1 for objects with custom depth.

Part 3: Wallhack

Now, what if I wanted that same masked object to appear over the rest of the scene?

This approach is very simple: if a pixel has a custom depth, set its color. In this I scale by a normalized time, which I will get to later. But an object with a custom depth will render over everything.

This works in conjunction with the masking, I mask out the object from the outline, then use it to render over the rest of the scene. I do not want the outline appearing on the object that is wallhacked.



Part 4: Blueprint Control - Simple on and off [Deprecated]

What if I want the player to be able to set the custom depth enabled on an object, scanning and outlining individual objects?

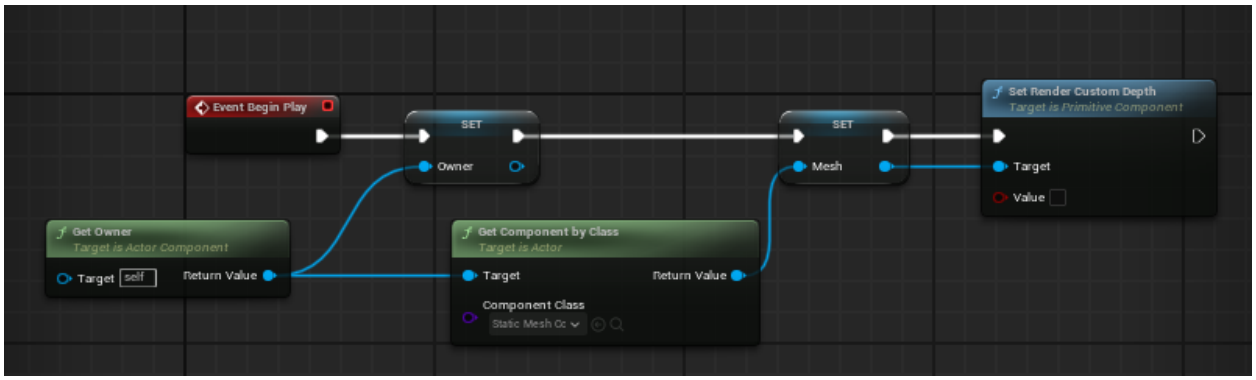
This would involve not outlining anything without a custom depth, and outlining those with. A setup in the material is simple, lerping the outline with the mask, but there needs to be blueprint control to make it work.

To do this I made the following:

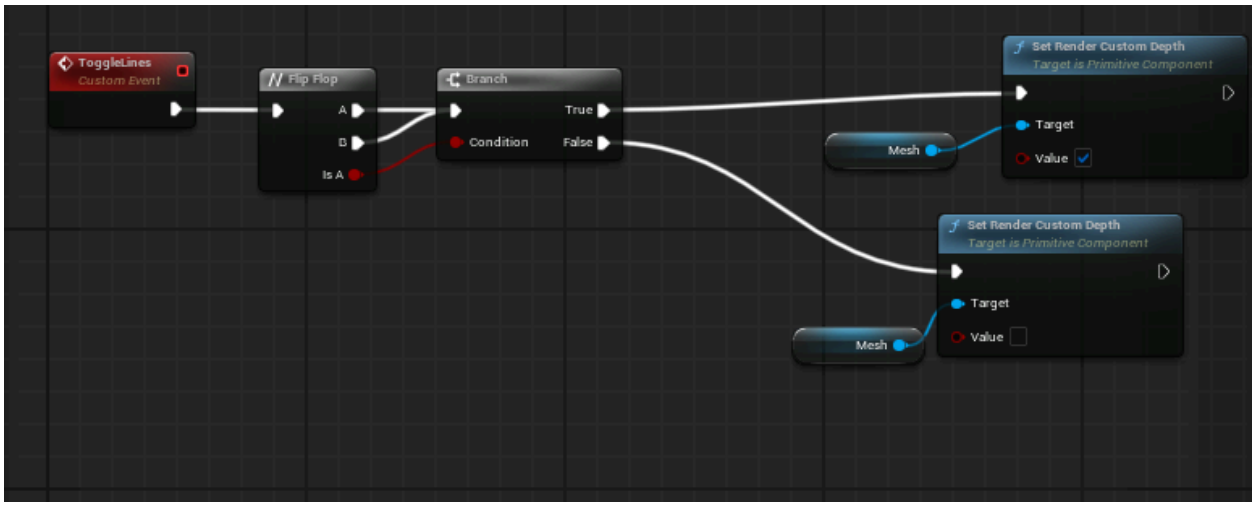
1. Interaction system utilizing an interface. Players can interact with objects through the `e` key.
2. An abstract actor class, defining a blueprint that can be inherited from.
3. A component class, containing the code for swapping depth.

My interaction system is a holdover from an earlier challenge, I will not provide an in depth explanation here.

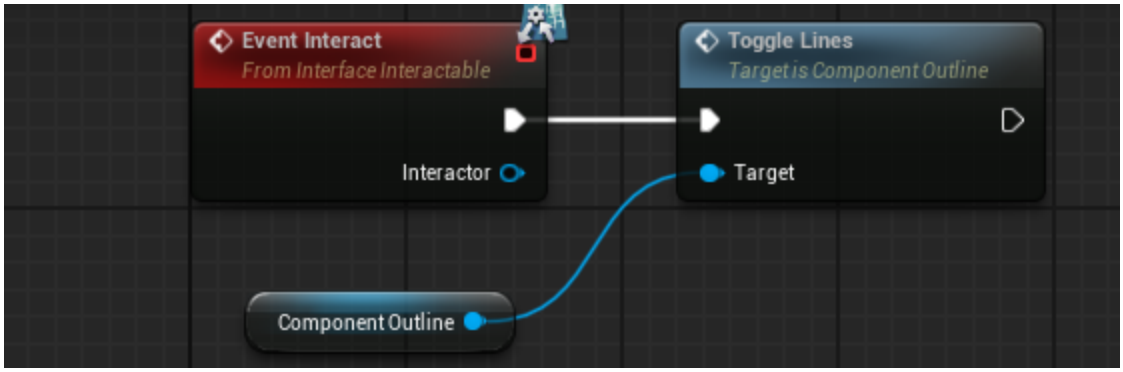
On `BeginPlay` of the component, I grab a static mesh of the owner.



`ToggleLines` is a function that swaps the custom depth bool of the mesh.



The actor implements the interact interface, and calls toggle lines on interact.



Problems:

- Need anything that can have lines enabled have to inherit from the base class that implements the interface.
    - Setup is not hard, but tedious
- No checks for valid on the mesh, could fail to grab

Plans: I went in a different direction, but could revisit

- Actually add checks to begin play to silently fail if missing components.
- Rather than utilize a class, dynamically setup the component to interact if not present, leading to a less hardcoded and easier to design scan loop. The player could scan an object, it would attach the component if not there, then outline the object.

Part 5: Blueprint Control - Complicated Fading [Cool stuff]

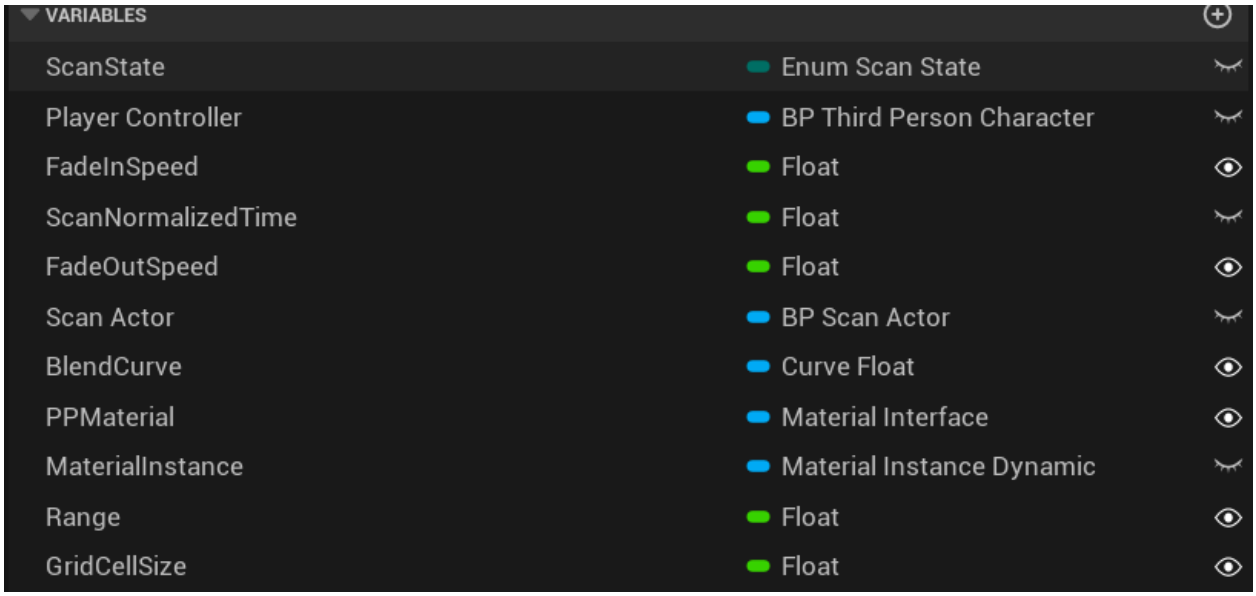
[Inspiration]

I decided instead of letting the user scan and outline objects, I would (thanks for the suggestion) implement a scan effect.

This involved another much more complicated component. This component would need to perform multiple tasks.

1. Activate a scan
2. Deactivate the scan
3. Do so over time, fade the scan in and out
4. Setup a post process volume
5. Setup a post process material

Variables

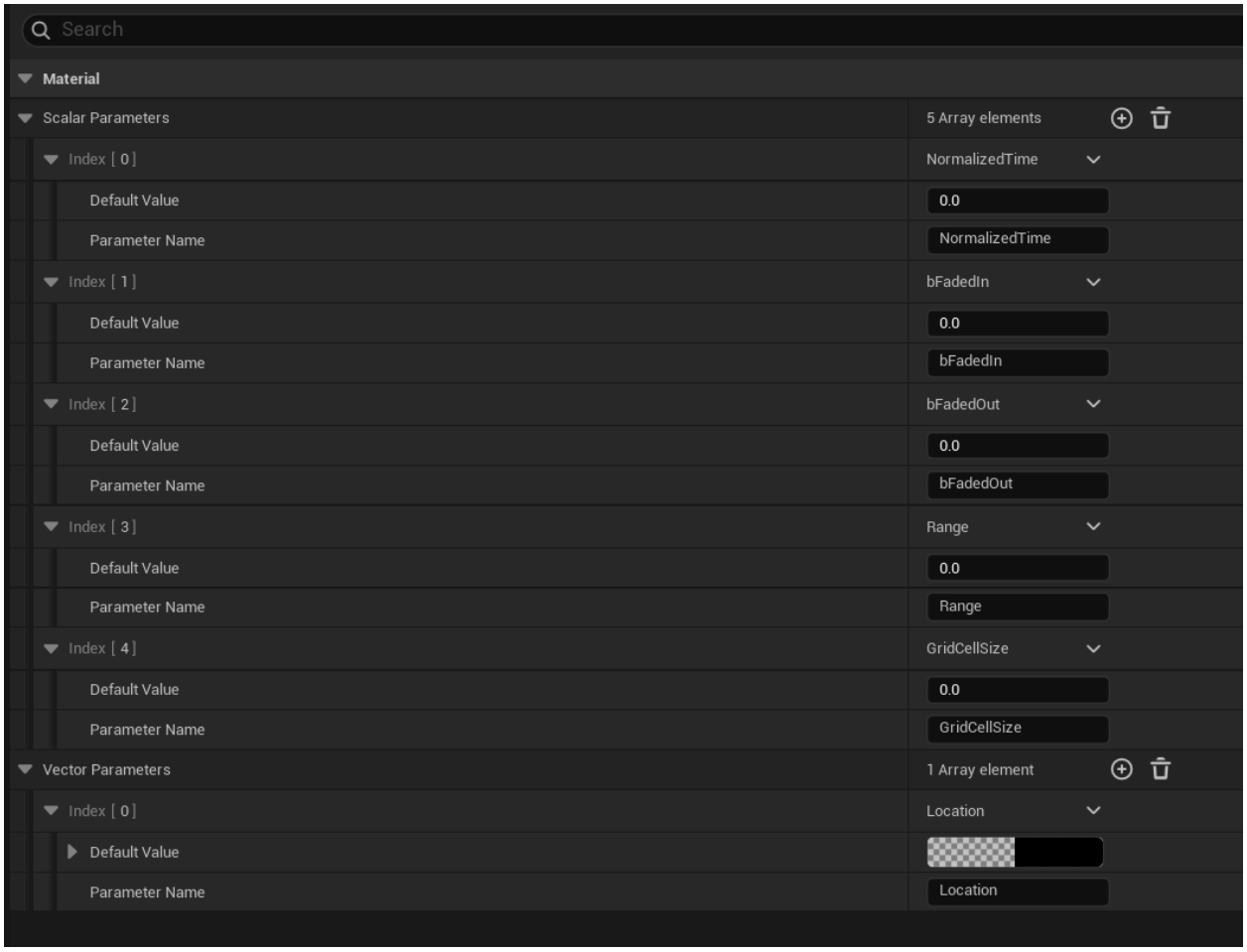


- `Player Controller` Is the player.
- `FadeInSpeed` Controls how fast the scan fades in
- `FadeOutSpeed` Controls how fast the scan fades out
- `ScanNormalizedTime` is how far into the scan the player is
- `Scan Actor` is an actor that holds the post process volume that will be created
- `BlendCurve` is the curve to blend the scan post process effect in and out. Linear would feel to slow, the curve feels snappier.
- `PPMaterial` is a material interface. Used as a base for all materials
- `MaterialInstance` is the dynamic material instance created from the `PPMaterial`, without this, the scan wouldn't be able to properly change the material at runtime.
- `Range` is how far the scan impulse/distortion goes
- `GridCellSize` designates how fluid the distortion UV's will appear
- `ScanState` is an enumeration. It's defined by FadeIn, FadedIn,FfadeOut, FadedOut. This is utilized to track the scan.

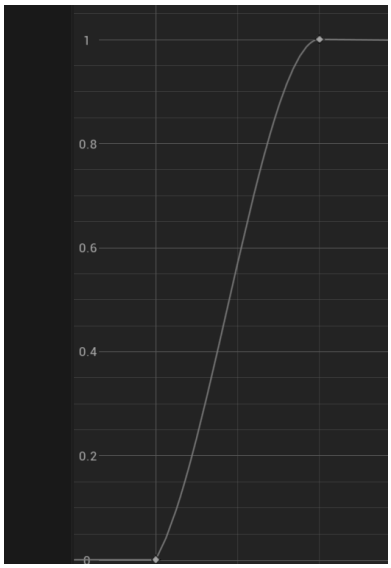
Other

Material Parameter Collection.

A material parameter collection holds lots of information utilized by the various materials and their functions. Lots of this data is passed/updated in the component to dynamically change the material.

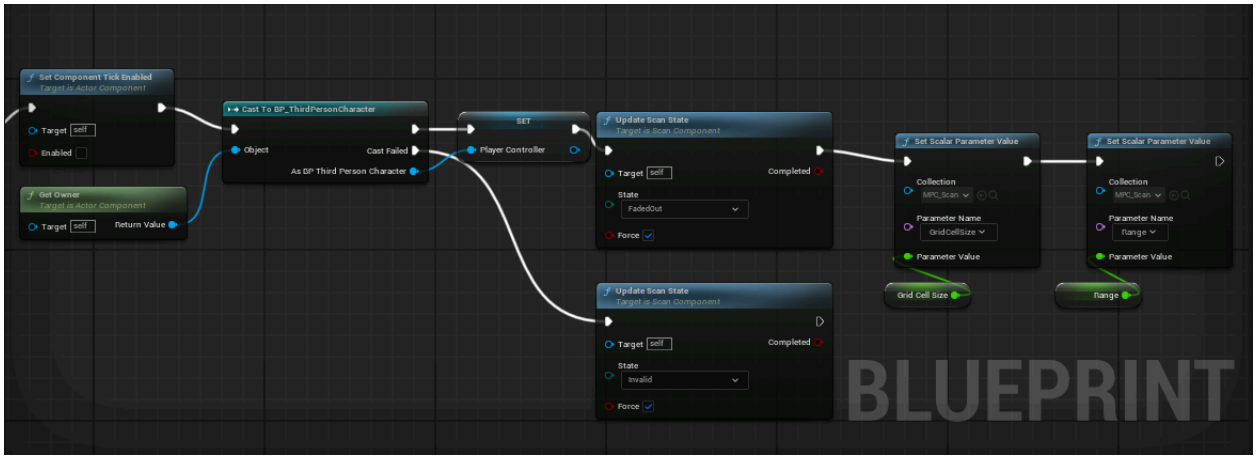


Blend Curve for the blend in and out.



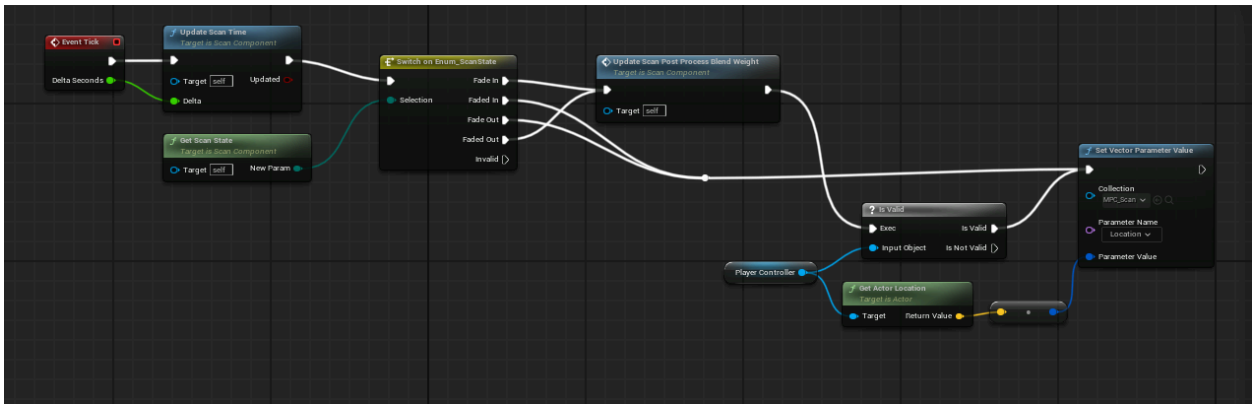
Begin

1. The timer on this object will be controlled by the tick. Therefore start with the tick disabled. (it has been disabled in the BP details, but this ensures it is).
2. Then get the player controlling the component.
3. If invalid, set the state to invalid and fail
4. Otherwise, update the scan state and set values to a MaterialParameter Collection.



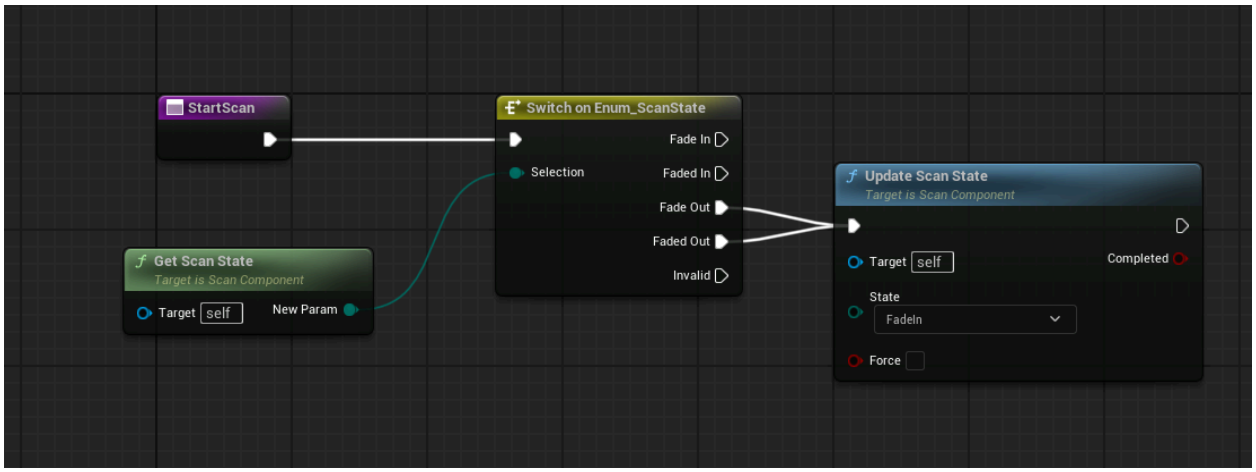
Tick

1. Tick is enabled and disabled on this component, so it can always function.
2. If a tick is enabled, update the scan time. Then, based on the state either, update the blend weight of the post process AND update the location, OR just update the location.
3. The scan weight is only updated during face in and fade out, as those are the states where the scan is actively transitioning. The other states only need the location as they are either fully or not blended.



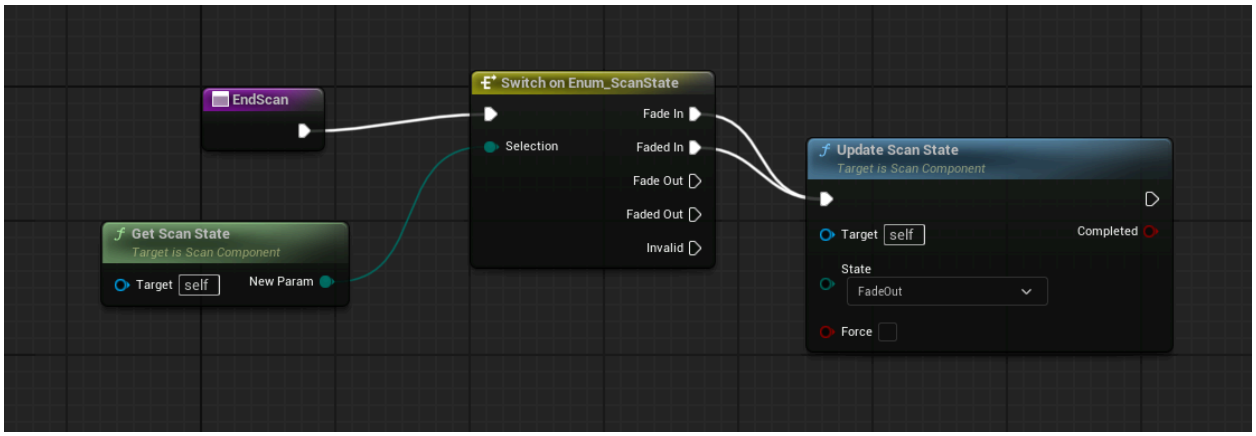
StartScan

If fading out or faded out, start fading in.



EndScan

If fading in or faded in, start fading out.



GetScanState

Returns the scan state.

UpdateScanState

Parameters:

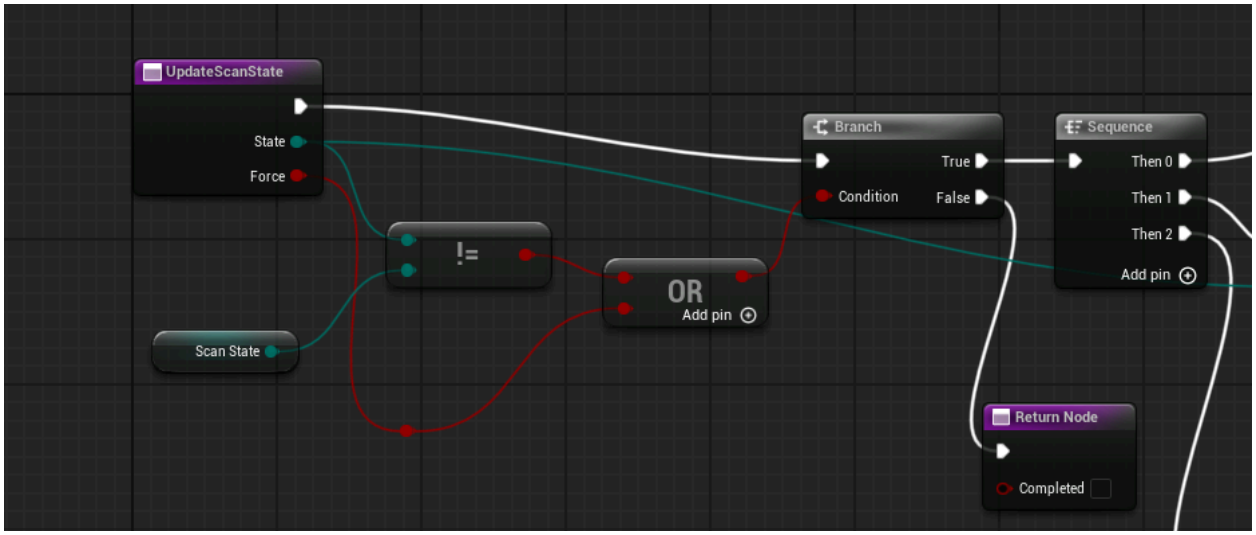
- ScanState State
- Bool Force

Output

- Bool Completed

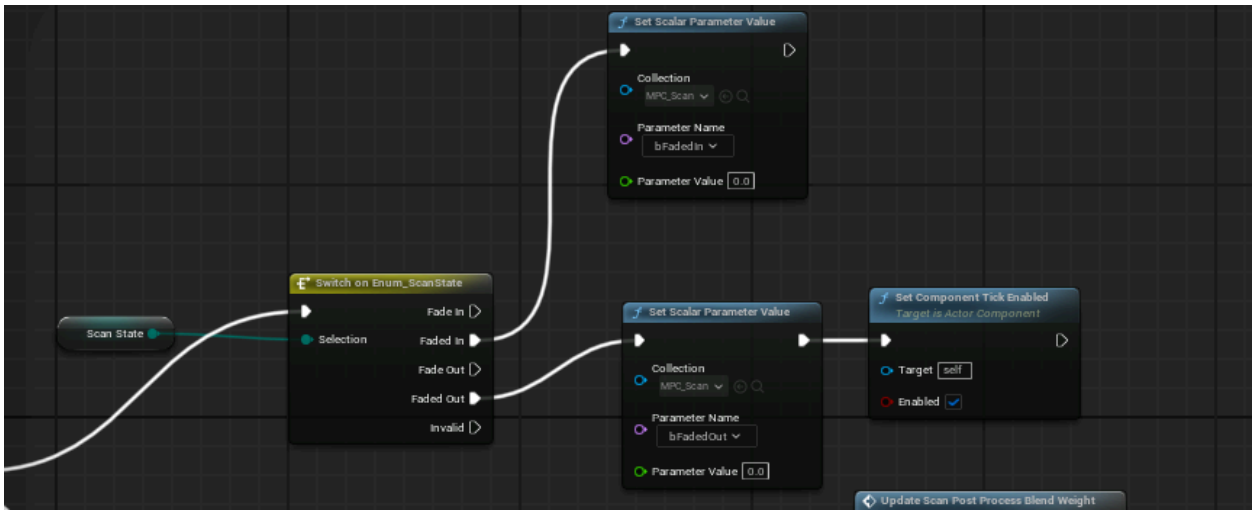
This function is responsible for updating the state of the scan. Depending on the new state, it sets tick enabled or disabled, and also updates Material Parameters.

It first checks the scan state, if it is equal to the current state it returns, unless force it true. Force forces the update, and is needed to start up a scan for the first time to set everything up.



Then there is a sequence:
First Check if the current state is faded in or faded out.

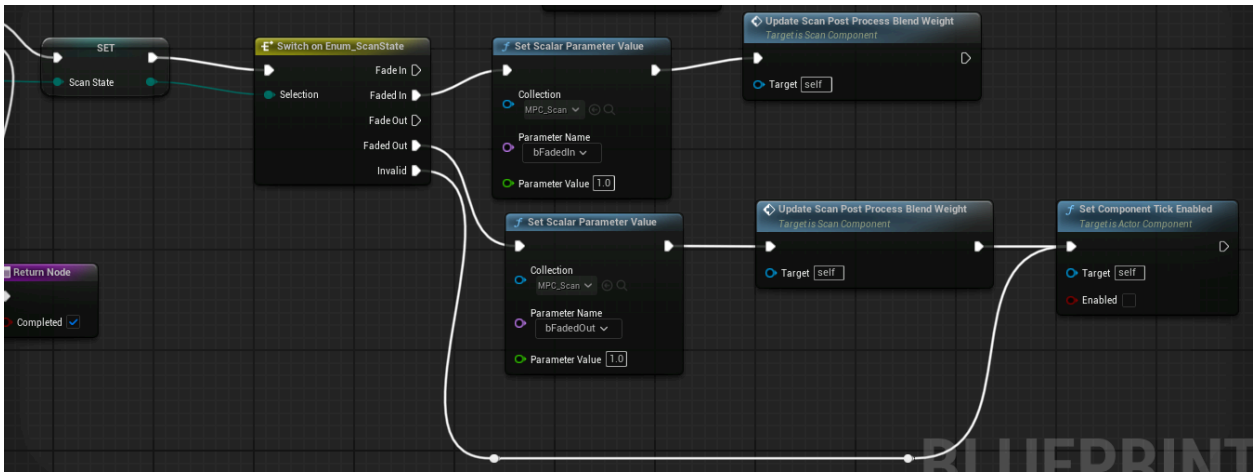
Faded In => reset the faded in bool to false.
Faded Out => set the faded out bool to false, and set tick to enabled, as the next state from faded out would be fade in, which needs the tick.



The next step in the sequence assigns the new scan state (parameter), and does various things based on the new state.

Faded in => set the faded in bool to true, and update the blend weight of the post process.

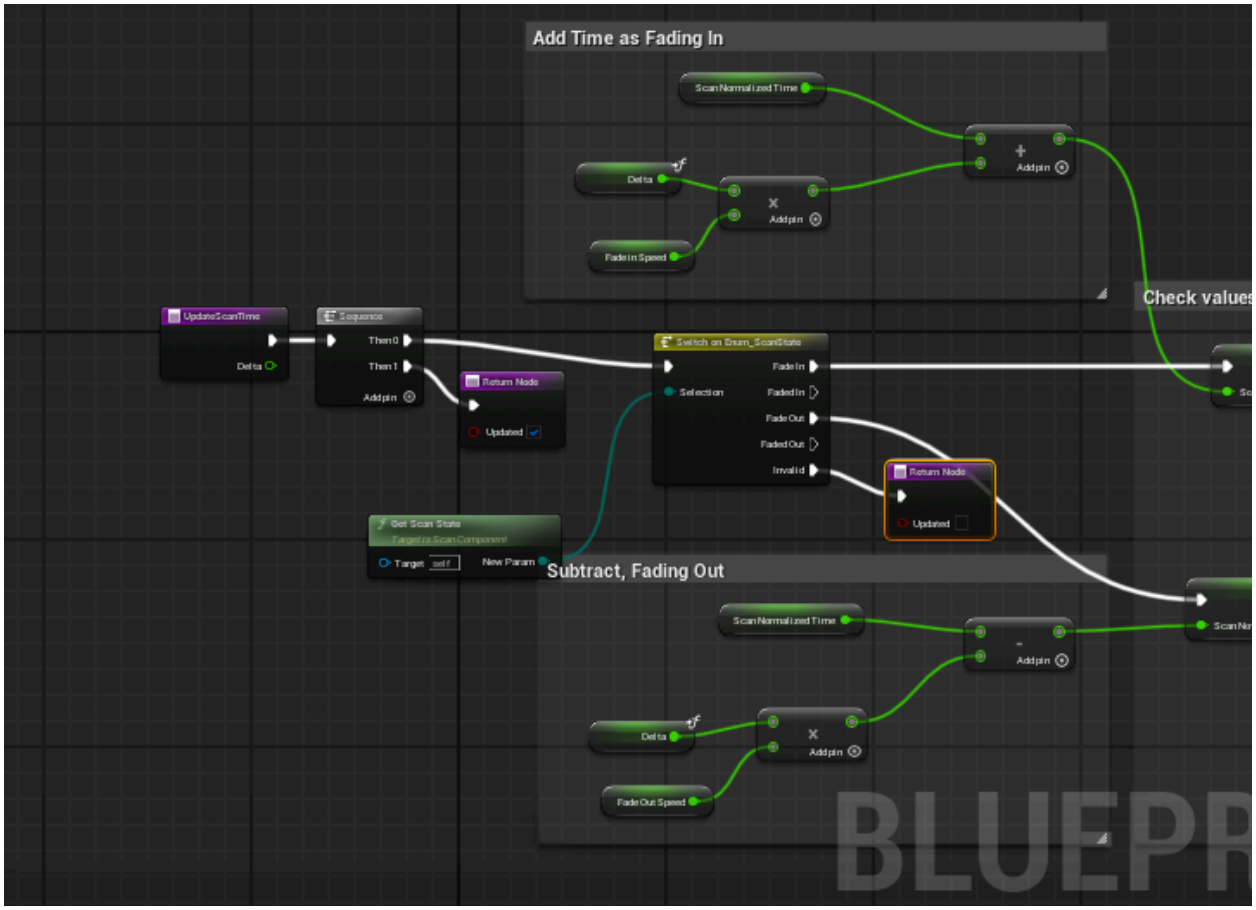
Faded out => Set the faded out bool to true, then set tick to false as updates are no longer needed as the scan is over.

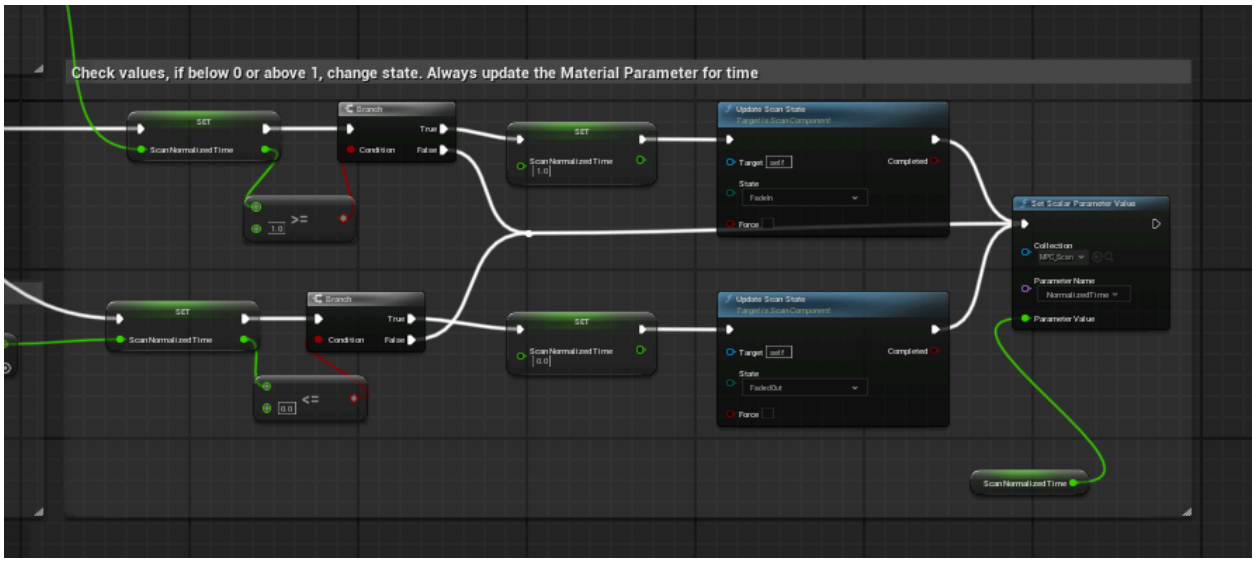


Finally, return true.

UpdateScanTime

This function updates the scan normalized time with a given delta parameter. It either adds or subtracts from the time given the state is fading in or fading out. When the scan time reaches 0 or 1, the function `UpdateScanState` is called to change the state of the scan.


Add Time as Fading In
ScanNormalizedTime
Delta
Addpin
Addpin
FadeInSpeed
Check values
UpdateScanTime
Delta
Sequence
Then 0
Then 1
Addpin
Return Node
Updated
Switch on Enum_ScanState
Selection
FadeIn
FadedIn
FadeOut
FadedOut
Invalid
Return Node
Updated
Get Scan State
Target
self
New Param
Subtract, Fading Out
ScanNormalizedTime
Addpin
Delta
Addpin
FadeOutSpeed
BLUEPR

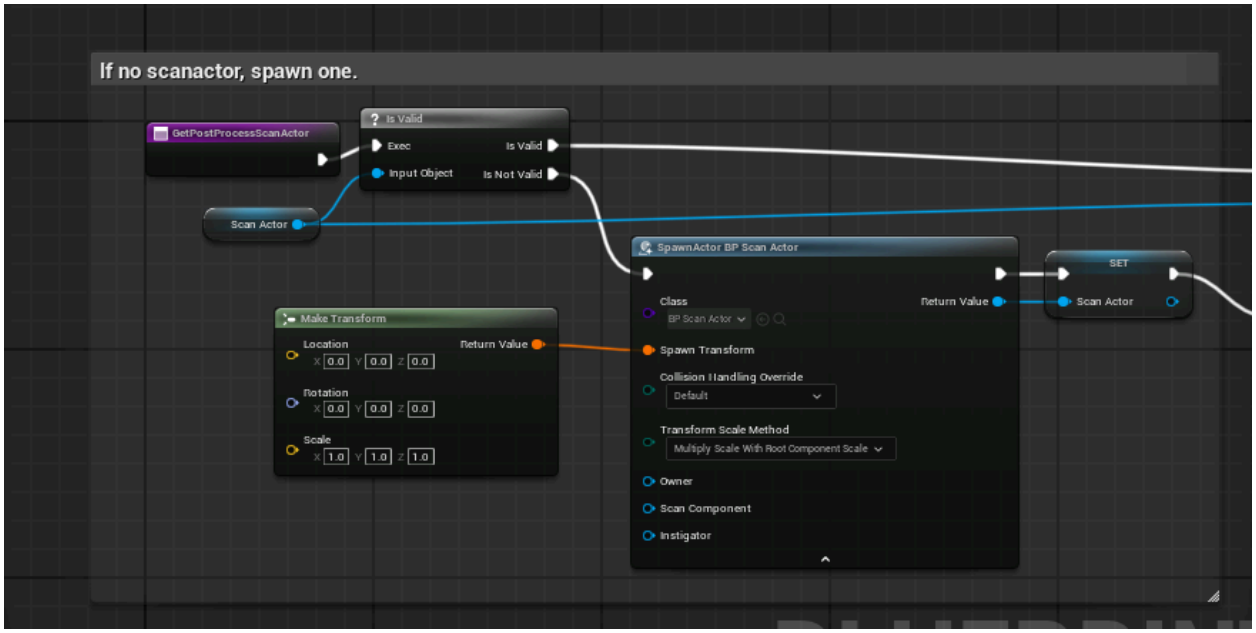

Check values, if below 0 or above 1, change state. Always update the Material Parameter for time
SET
ScanNormalizedTime
Branch
Condition
True
False
SET
ScanNormalizedTime
1.0
Update Scan State
Target
self
State
FadeIn
Force
Completed
Set Scalar Parameter Value
Collection
Parameter Name
NormalizedTime
Parameter Value
SET
ScanNormalizedTime
Branch
Condition
True
False
SET
ScanNormalizedTime
0.0
Update Scan State
Target
self
State
FadedOut
Force
Completed
ScanNormalizedTime

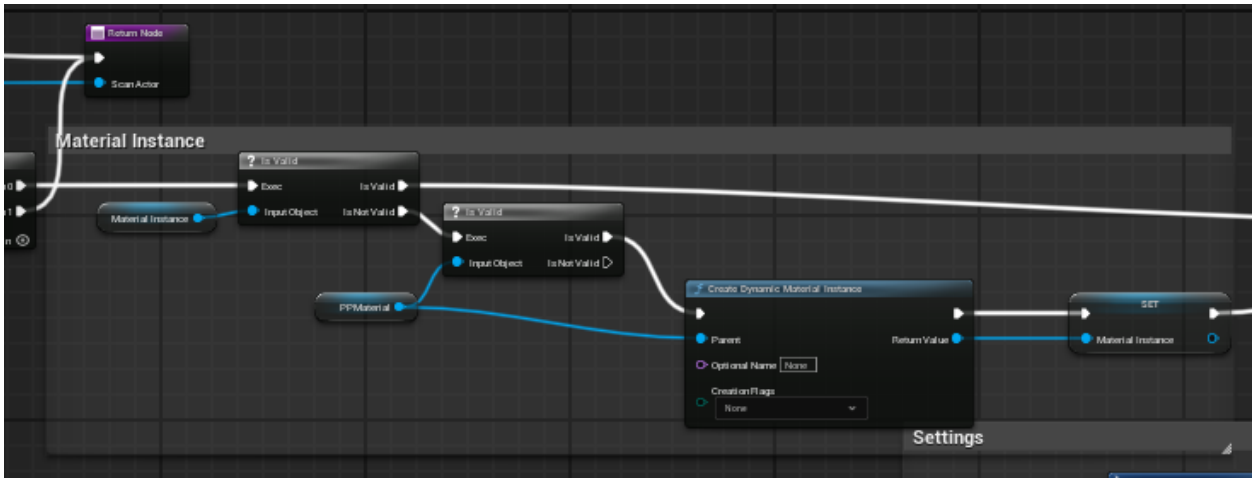
GetPostProcessScanActor

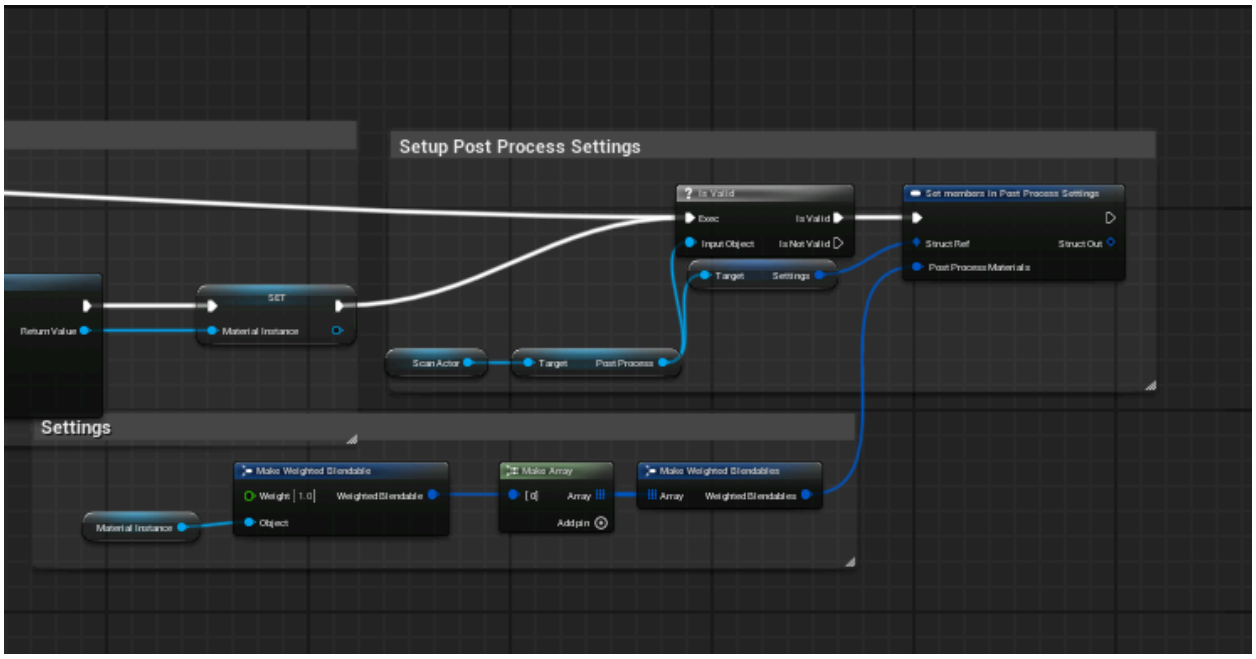
This function gets the post process component on an actor. If the actor does not exist in the scene, it spawns one in. The `BP Scan Actor` has a Post Process Unbound Component.

Note: Could potentially make it bound to the player, and have the player have a limited scan range in the future.

1. Check for a scan actor, if invalid, spawn a new one. If valid, return the actor.
2. Check material instance, if invalid make a new one.
3. Then get post process volume from the actor, if valid, set its settings to include the instance material. Also make sure the instance is blendable.
4. Return the actor

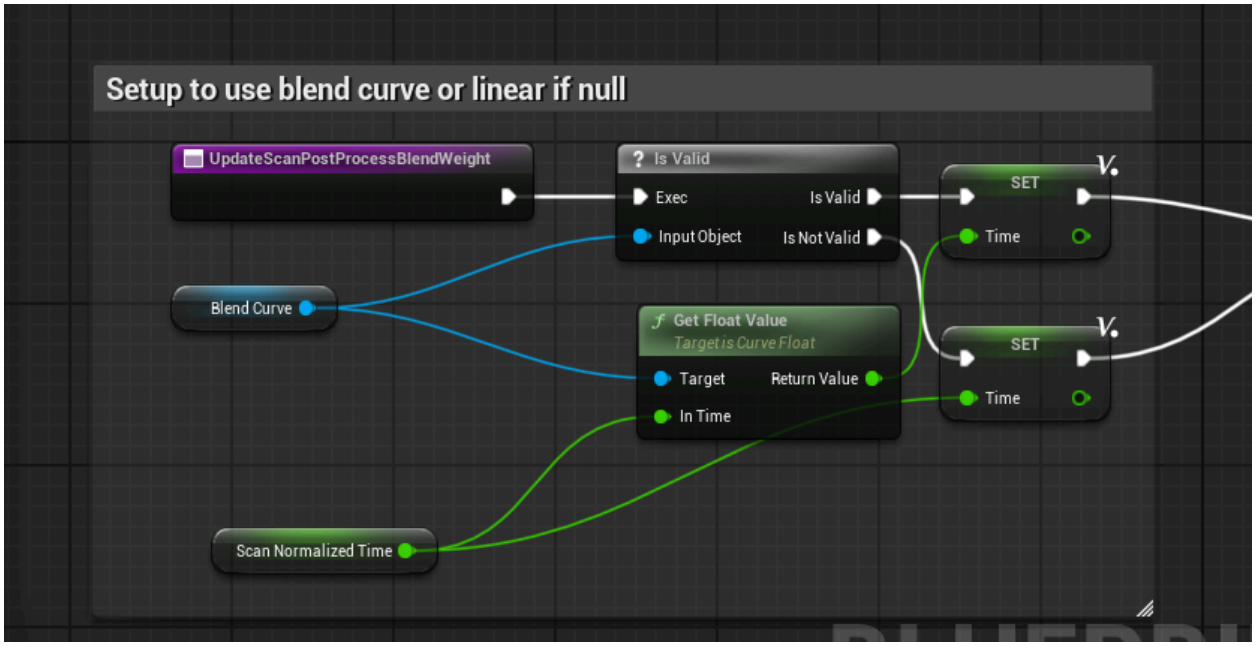


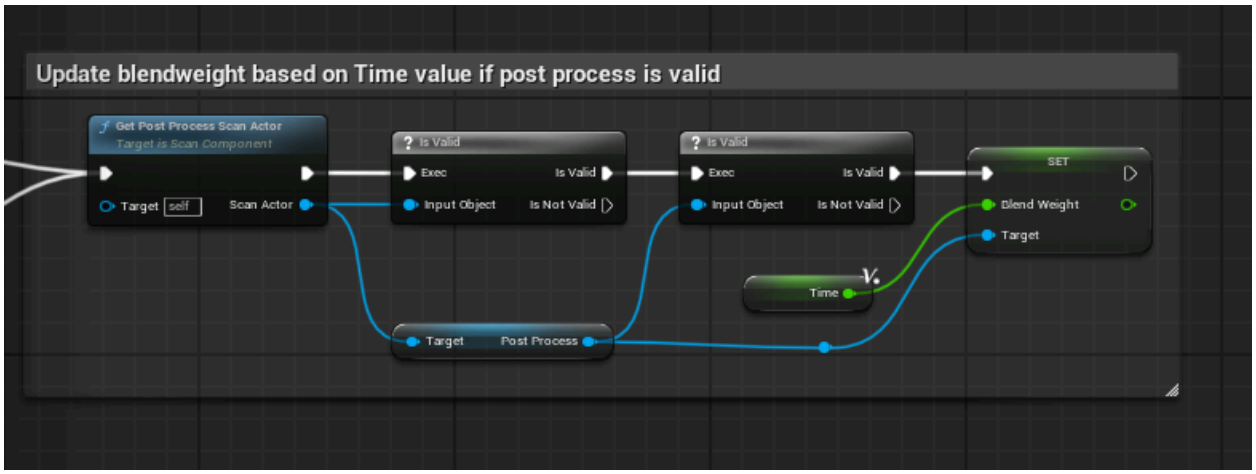




UpdateScanPostProcessBlendWeight

This function updates the blend weight of the dynamic material instance. It utilizes the curve to update the blend weight based on the normalized scan timer.





Part 6: Material Function DistortionRing.

To have a scan emulate from the player like a wave, I decided to follow the video (referenced above) to distort UV's around the player. The first step was to create a wave through a ring. This is achieved by utilizing an outer and an inner mask to create a ring.

1. Thickness Control

Controls how thick the effect is.

- `MPC_Scan` provides a base thickness value.
- `Input Thickness` and `Input Thickness Multiplier` allow extra control.
- The result is saved into a local variable `Thickness`

2. Location

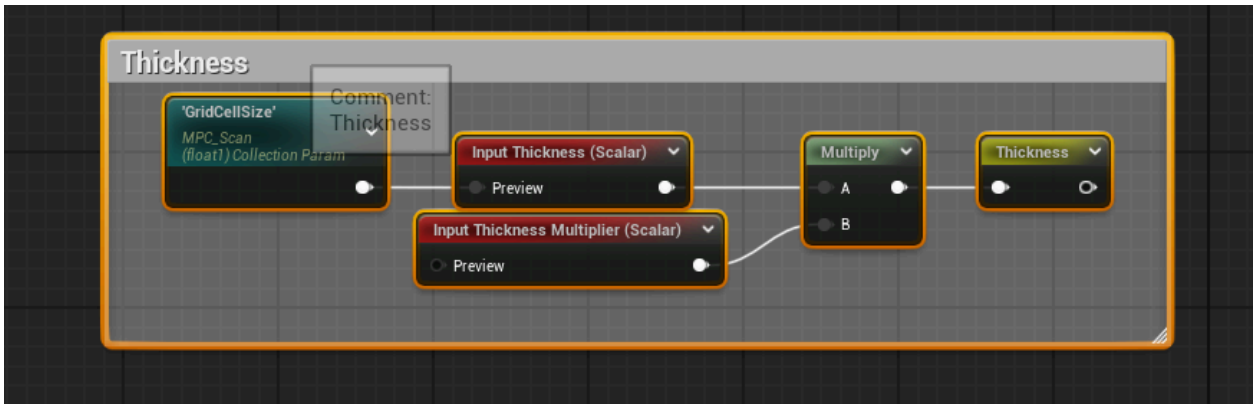
Defines where the scan effect originates from in world space.

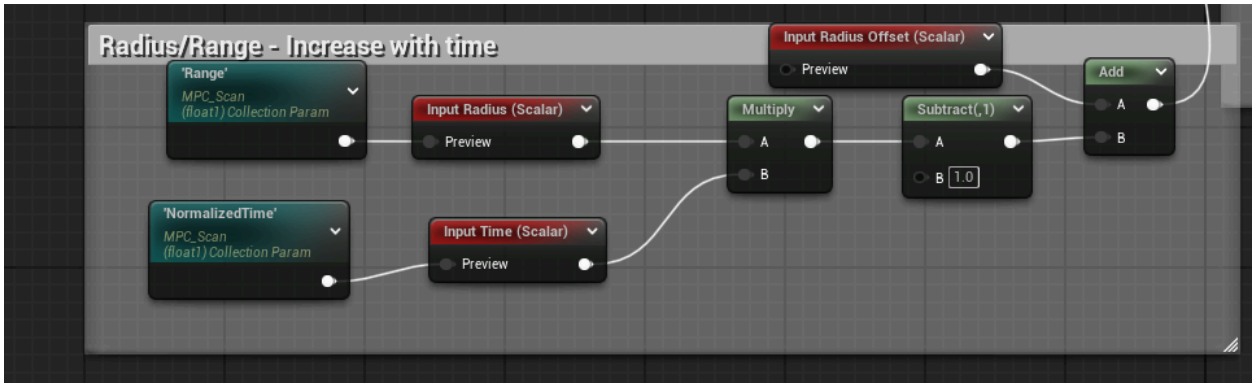
- `Absolute World Position` gets the current world position of each pixel.
- `MPC_Scan` again provides a `Location`
- The system calculates **distance** from this origin to each pixel, giving a gradient.

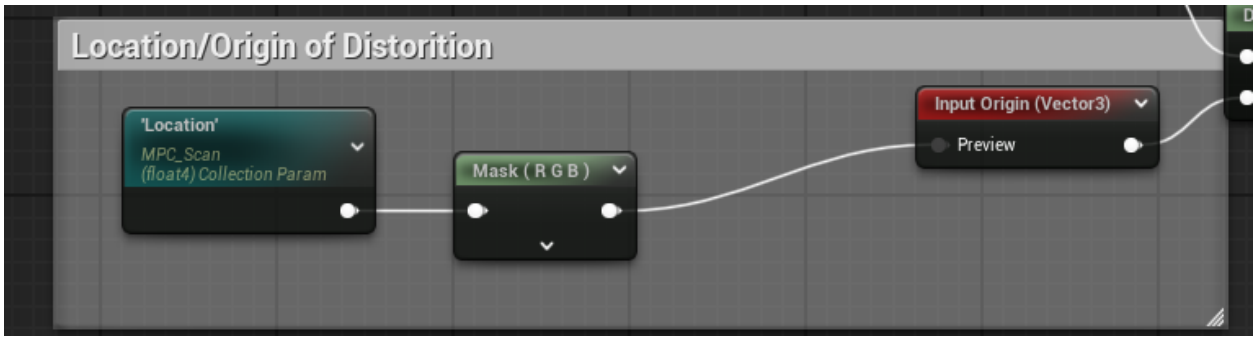
3. Radius

Expands the scan radius over time for a growth/pulse

- `Range` is the maximum scan radius.
- `NormalizedTime` (0 to 1) is multiplied by Range, making the radius expand as time increases.
- `Input Radius Offset` added control.
- NOTE: The subtract node should subtract the thickness, caught the issue after adding photo to doc. Happened when I swapped to a named node.


Thickness
Comment:
Thickness
'GridCellSize'
MPC_Scan
(float1) Collection Param
Input Thickness (Scalar)
Preview
Input Thickness Multiplier (Scalar)
Preview
Multiply
A
B
Thickness


Radius/Range - Increase with time
'Range'
MPC_Scan
(float1) Collection Param
Input Radius (Scalar)
Preview
Input Radius Offset (Scalar)
Preview
Multiply
A
B
Subtract(,1)
A
B 1.0
Add
A
B
'NormalizedTime'
MPC_Scan
(float1) Collection Param
Input Time (Scalar)
Preview


Location/Origin of Distorition
'Location'
MPC_Scan
(float4) Collection Param
Mask (R G B)
Input Origin (Vector3)
Preview

4. Inner Mask

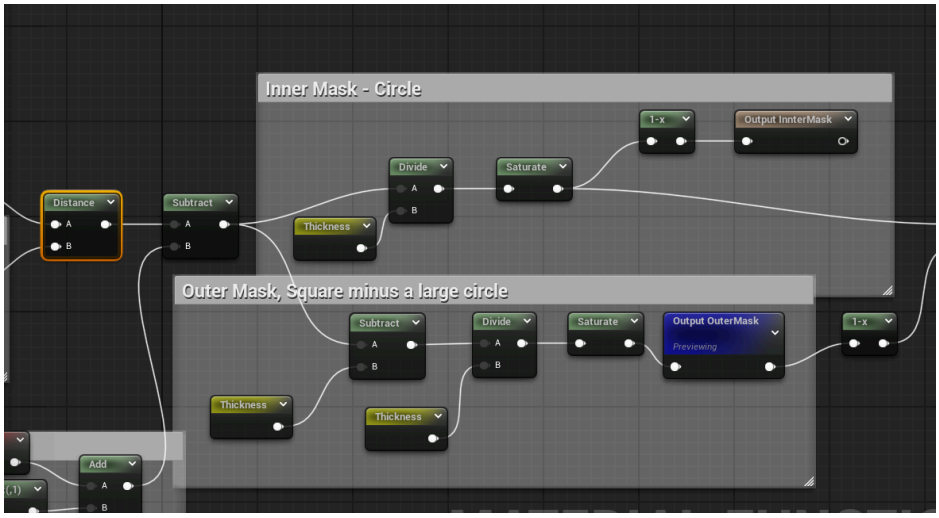
Defines the inner part of the scan circle

- Distance from the center is **subtracted** by the current scan thickness.
- Divided by Thickness to create a transition
- `Saturate` clamps the values between 0 and 1.

5. Outer Mask

Controls an **outer falloff** beyond the main circle — like a shockwave fading outward.

- First, **divide Thickness** instead of subtracting it.
- Then saturate.

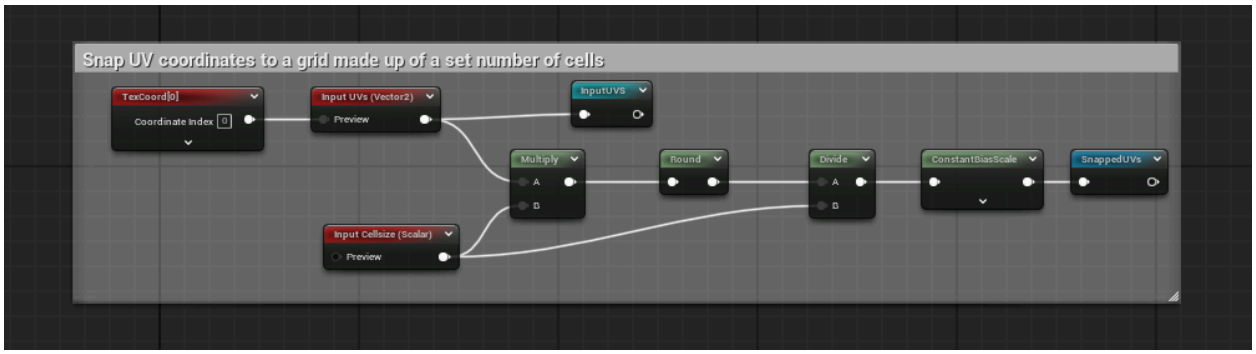


Finally, the inner and outer masks are multiplied to combine their effects.

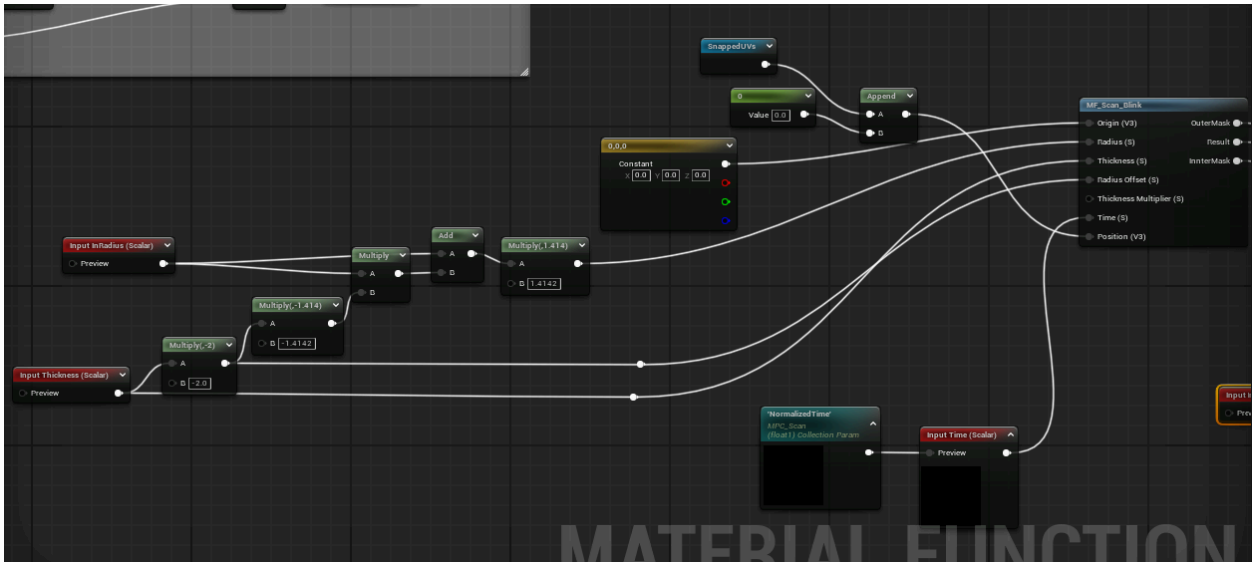
Part 7: Radial Distortion.

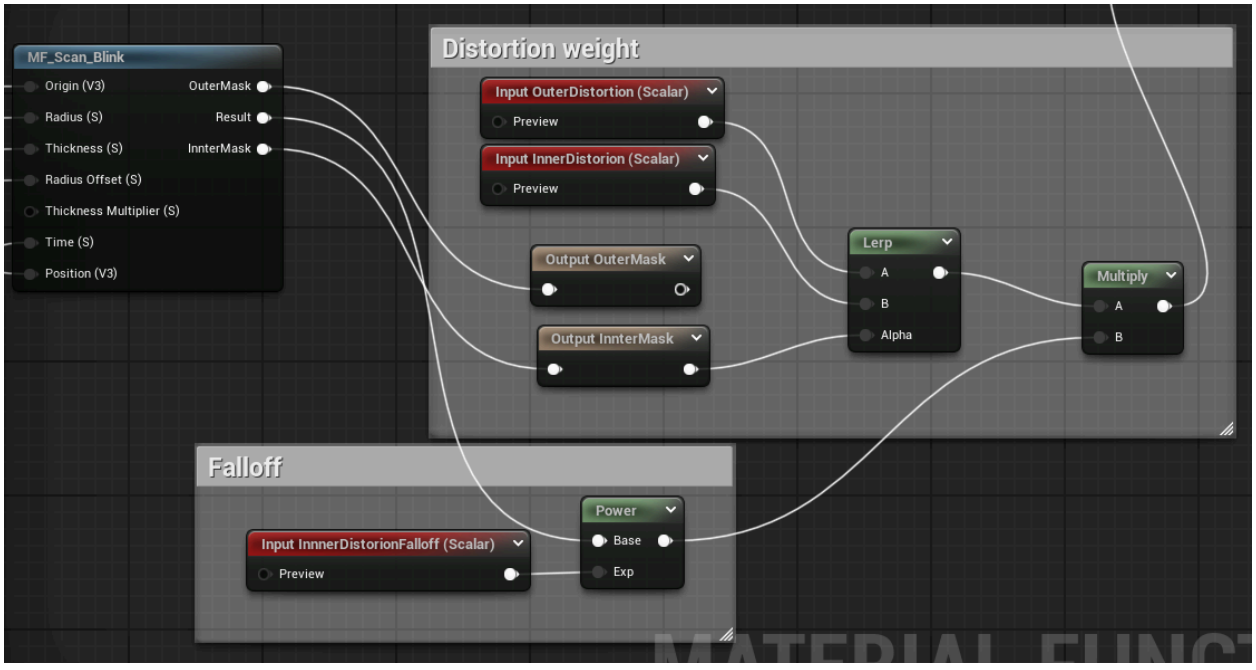
This effect utilizes the moving masks to create the radial distortion effect.

1. Snap UV's to a grid. This can have varying resolutions, but there are some issues with lower resolutions. A current bug is the middle pixels get averaged to the same value, creating a "block" of one color in the middle.

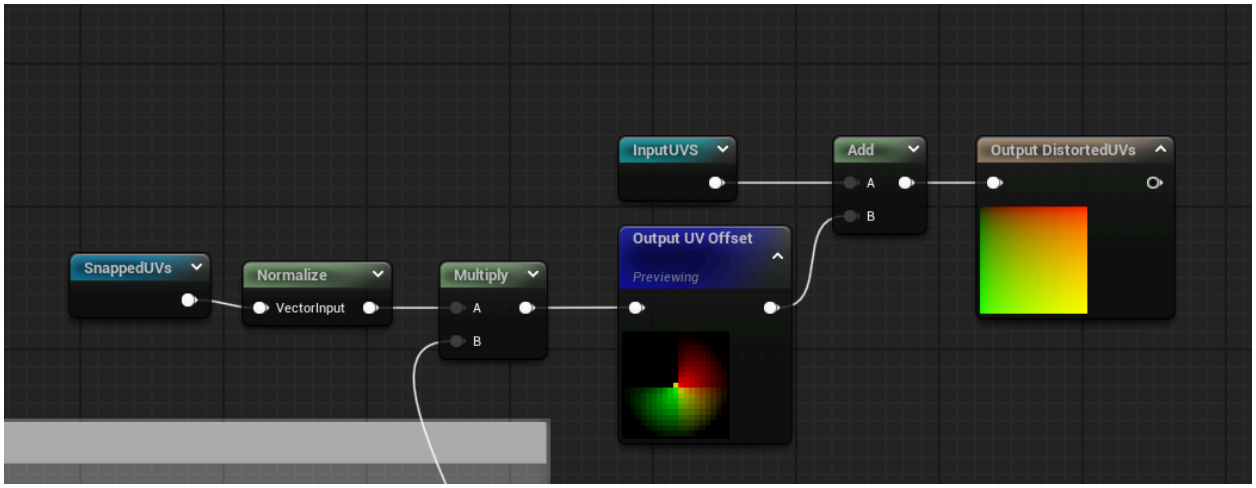


2. Setup function parameters.
    a. 0 is appended to UVs then passed to position. (simulate a 3d vector)
    b. Origin is (0,0,0)
    c. Time is normalized time
    d. Thickness is used for thickness
    e. Radius and thickness are changed to account for a better looking effect.
        i. Thickness is multiplied by -2 and added to the radius offset. This creates the black hole compression.
    f. The multiplication by root 2 accounts for the diagonals (pythagorean). The radius is multiplied by the root 2 scaled thickness then reverted after an addition. This creates a diamond-like effect.



3. Apply Blink, use the input to apply a weighted distortion and falloff to the effect for more control.
    a. Output the inner and outer masks so they could be used in the master material.
    b. Lerp the distortion inputs together using the inner mask as the alpha control.
    c. Power the result mask by the falloff to reduce the mask's distortion around the center.
    d. Multiply the inner falloff by the lerped distortion.
4. 

5. Apply the distortion mask to the Snapped UV's, then add them to the normal UVs


InputUVS
Add
A
B
Output DistortedUVs
Output UV Offset
Previewing
SnappedUVs
Normalize
VectorInput
Multiply
A
B

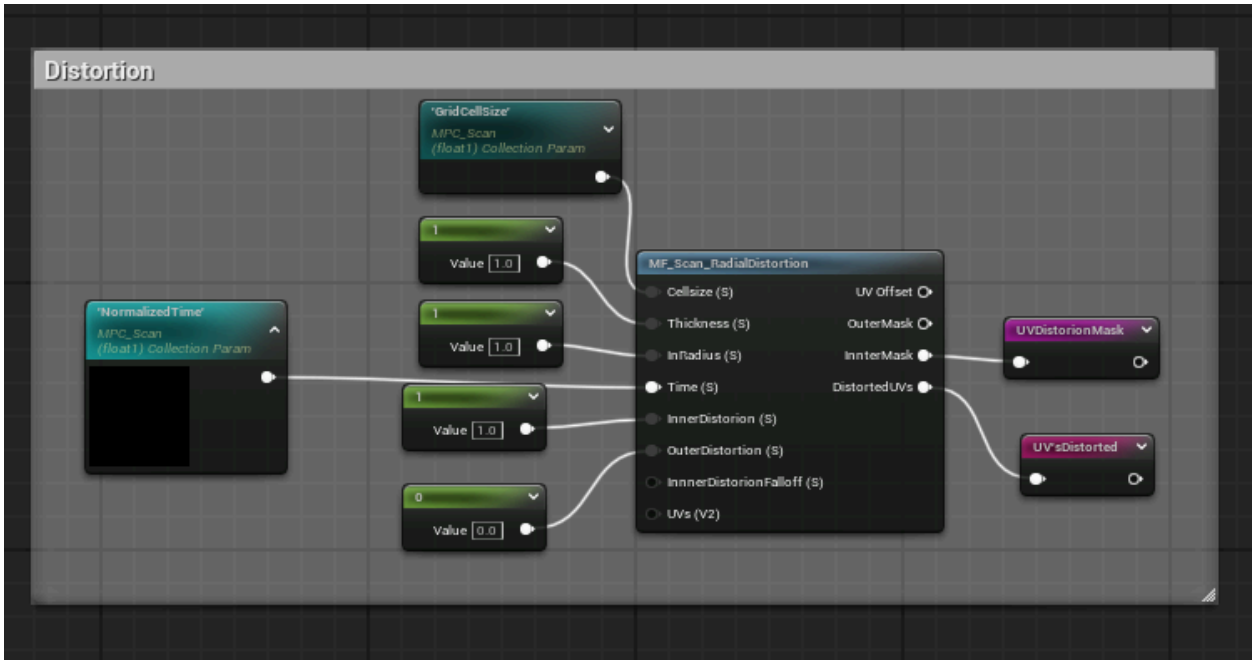
Part 8: Putting it all together

All the functions and blueprints are done, it all must be setup

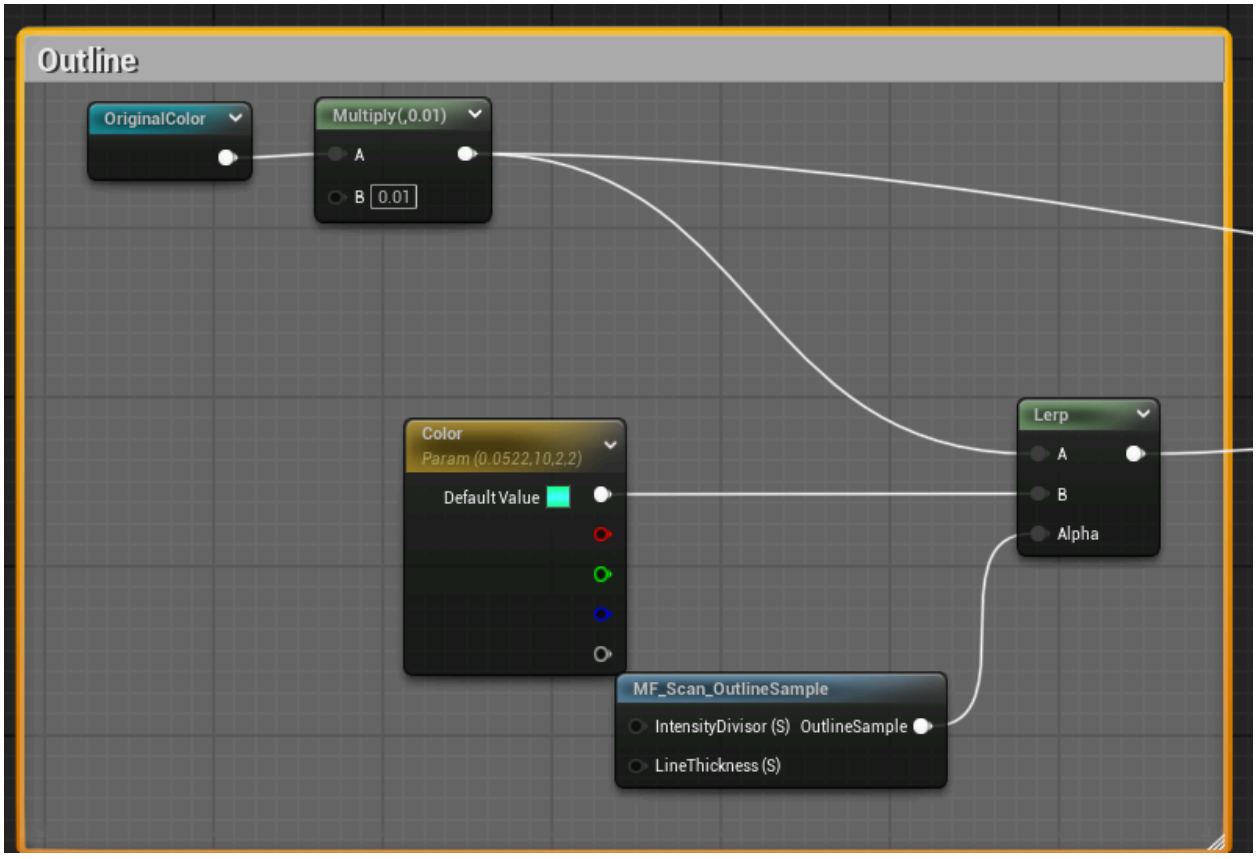
Some Initial Samples

Distortion

Distortion function created, and outputs the masks and distorted UV's . Lots of default values used here. A future iteration could make them parameters and more dynamic.

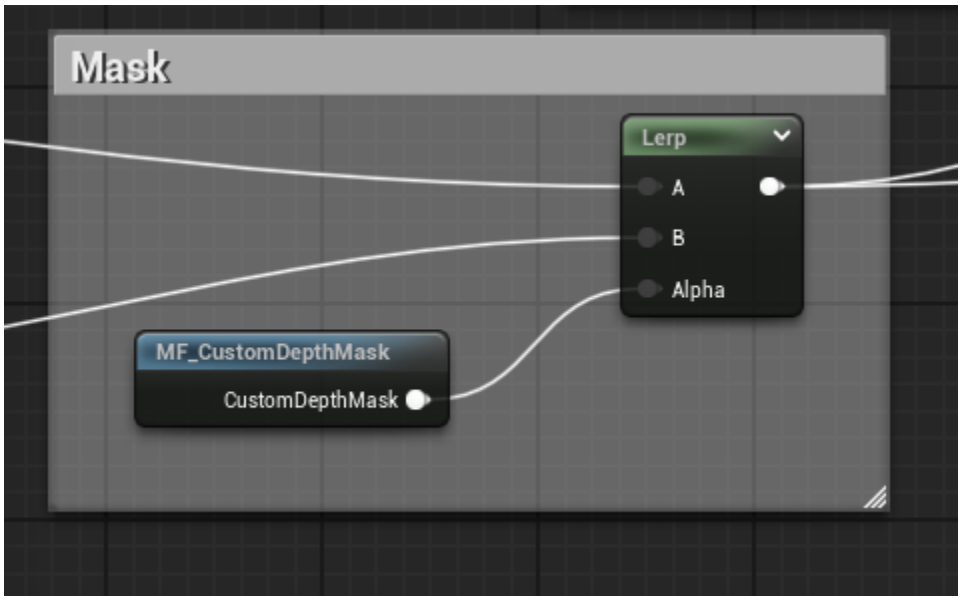

Distortion
'GridCellSize'
MPC_Scan
(float1) Collection Param
1
Value 1.0
1
Value 1.0
1
Value 1.0
0
Value 0.0
'NormalizedTime'
MPC_Scan
(float1) Collection Param
MF_Scan_RadialDistortion
Cellsize (S)
Thickness (S)
InRadius (S)
Time (S)
InnerDistorion (S)
OuterDistortion (S)
InnnerDistorionFalloff (S)
UVs (V2)
UV Offset
OuterMask
InnterMask
DistortedUVs
UVDistorionMask
UV'sDistorted

Outline

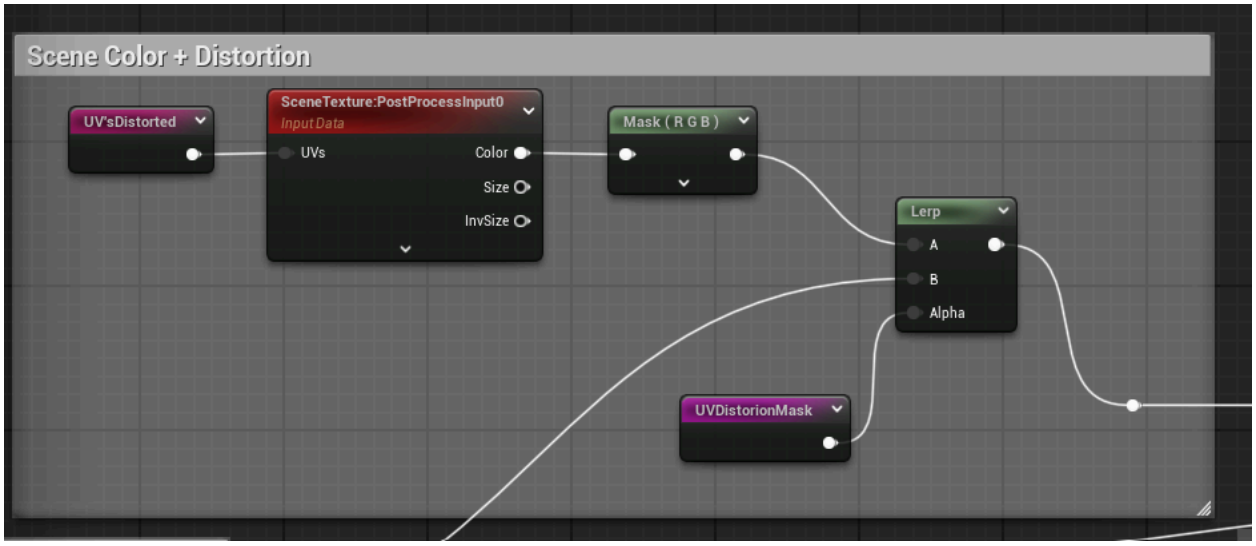

Outline
OriginalColor
Multiply(,0.01)
A
B 0.01
Color
Param (0.0522,10,2.2)
Default Value
Lerp
A
B
Alpha
MF_Scan_OutlineSample
IntensityDivisor (S)
LineThickness (S)
OutlineSample

Mask

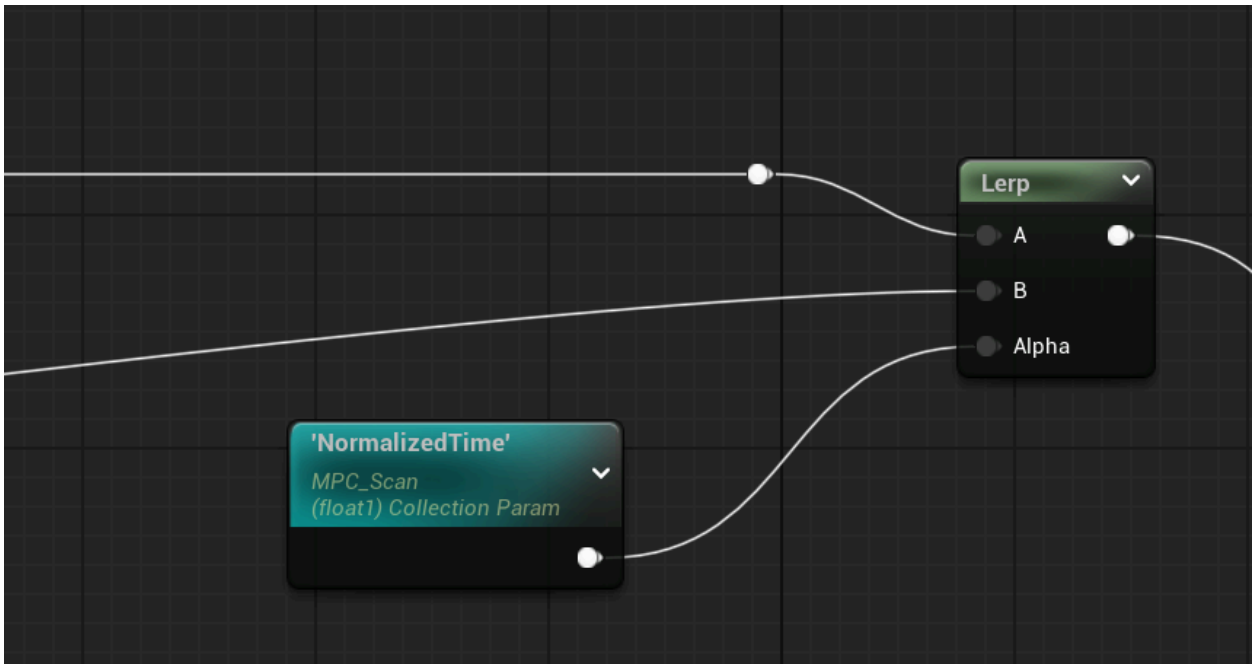
Remove the outline from all objects with a custom depth.



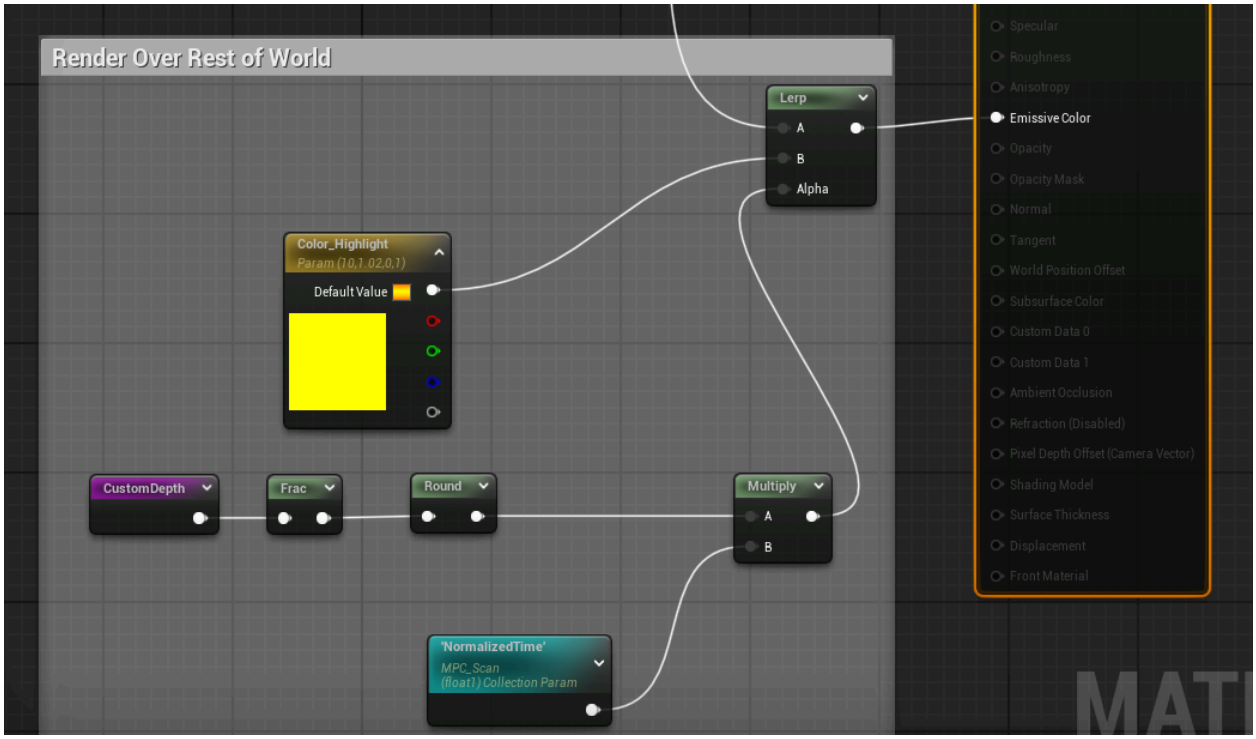
Combine Distortion and Outline



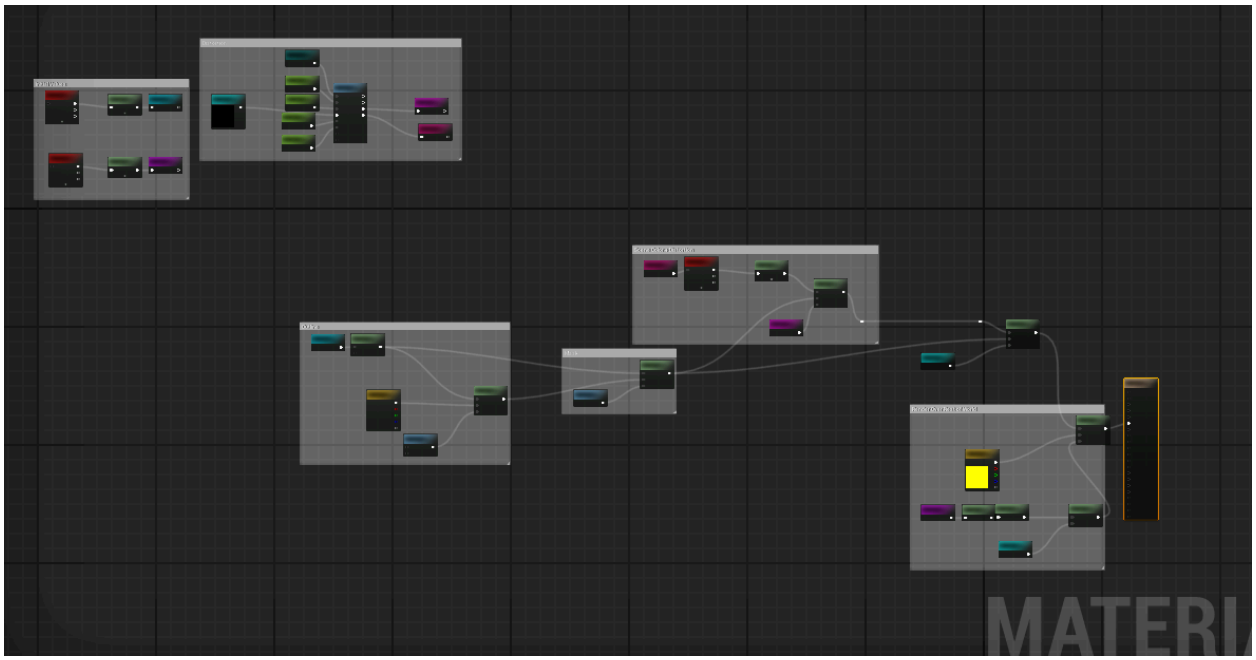
Fade in or out with Time



Render elements with a custom depth over the rest with a set color



Overall



Conclusion

I learned a ton from this research.

1. Standards for blueprints - privacy and tick disabling
2. Components - creation and standards
3. Dynamic material instance creation + setup
4. Material blueprints and related math.
5. Use of material functions for reusability.
6. Process behind material creation and testing
7. Material Collection Parameters
8. Material Blending

This was a valuable use of last week's time. I learned a ton regarding Unreal overall and the application of both materials and post process materials.