A collection of postprocess shader effects for Unreal Engine 5

Lewis Maskell – UP2206042

Introduction

While most shader effects are designed to be applied to objects within a scene, determining how they react to lighting within the environment, a postprocess effect in *Unreal Engine 5* (Epic Games, 2025b) is one that will apply to the screen-space render on the camera. This can be likened to how a photographer may colour grade their pictures to add the finishing touches needed to achieve the exact look that they desired.

Postprocess effects in *Unreal* can be used in the same way, the extensive library of the *Material Editor* (Epic Games, 2025a) means that there is potential to create a wide variety of unique and visually interesting effects.

One industry example of this is Semiwork Studio's *R.E.P.O* (Semiwork Studios, 2025). This game is a horror-styled multiplayer co-op game, with a very distinctive grunge/pixellated art style. While the use of well-designed assets, audio and textures are essential to developing this, the use of postprocess effects in fine, nearly imperceptible details are highly effective when used in combination to generate the needed finishing touches.

By analysing a single still frame from *R.E.P.O* in detail, several postprocess effects can be identified.

Postprocess materials are not only limited to subtle colour grading or stylistic effects. The *Chameleon* postprocess effect library for Unreal Engine 5 (SUMFX, 2026) demonstrates how these effects can be used for a much wider variety of applications, ranging from screen shake, dirt and blood effects to indicate damage to the player, to black bar masking and depth of field blur that are extremely useful for cinematic or cutscene applications.

(Screen dirt + damage - Chameleon postprocess effect library)

Postprocess effects are not a unique feature of *Unreal Engine 5*, as most modern game engines, including *Unity* and *Godot,* also support the creation of postprocess materials. Therefore, understanding the capabilities and limitations of postprocess effects is a highly valuable skill for game development.

Therefore, the aim of this project is to construct a variety of postprocess effects using the *Unreal Engine 5 material editor*. These effects should consist of one or more components with parameter controls to allow for easy customisation.

Appraisal of Chosen Technologies

The primary reason for choosing *Unreal Engine 5* in this project, is because of the functionality provided by the Material Editor. The Material Editor is a node-based shader construction tool, allowing for a simplified and easy-to-understand interface for node editing.

The alternative to using the Material Editor would have been to learn and write shader code using HLSL. While knowing this language is valuable for industry tech-art roles, opting to instead use a node-based workflow reduces the time required learning new syntax and allows for more time spent developing the shaders and exploring new concepts. Furthermore, the Material Editor enables a live view of the resultant shader, allowing for more of a “trial and error” approach to development and a clear representation of how each node influences the constructed shader as it is being built.

The node library provided by the Material Editor is also highly extensive, meaning that there is freedom to construct a wide variety of shaders. Unfortunately, official documentation of this library reserves the most in-depth explanations, tutorials and use cases for certain key functions, with only surface level description for most nodes. However, since the launch of *Unreal Engine 5* in 2022, the game engine has been used extensively both for indie development and in industry, resulting in vast amounts of community generated resources, tutorials and forums that supplement the lack of documentation.

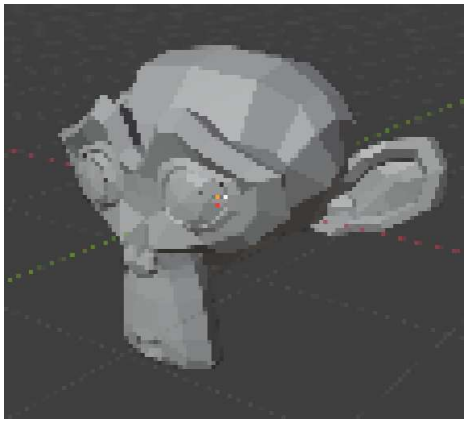
The way that postprocess effects in unreal engine are implemented in scene are through the use of Postprocess volumes. These volumes are configurable with an array of material instances, that allow for the easy combination and ordering of different postprocess effects. Being able to move between postprocess volumes also means that comparisons between effects are very easy to make. The use of material instances for postprocess volumes means that parameters can be changed and updated in real time, making workflow more efficient when calibrating ideal parameter values.

Project Definition & Planning

For this project, I will be building 3 distinct postprocess effects, with unique components that are controllable by variable parameters.

The first of these will be a “Pixel” shader, which will have a component to control the number of pixels available on the screen and a colour quantization component, which will control the number of available colours. These in combination will produce a shader that can be used to generate a “retro” effect.

Example (lacking colour quantization):

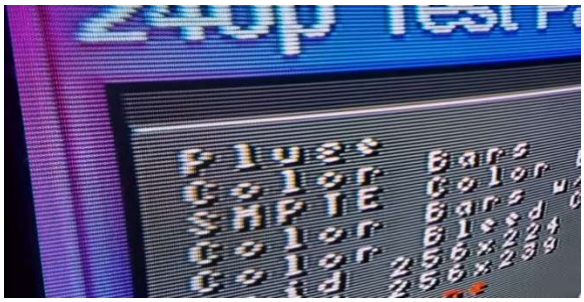


The pixel shader component is highly versatile and will also be used as a component of one of the later effects. Alone, this can be used as both a tool for a visually interesting transition, or simply as an important component of a unique 3D pixel art style. While colour quantization alone is less versatile, when used in combination with the pixellated scene, it will act as an important component of a “retro-style shader”, reflecting the limited number of colour channels available on older consoles and screens.

This effect has also deliberately been chosen as the first one to be developed, as its individual components are relatively simple to develop and will act as a good introduction to understanding and manipulating UVs as well as Aspect Ratio.

The second effect developed will be a Cathode-Ray-Tube (CRT) Screen effect. This will include a component for a “scanlines” mask, to make up the gaps in a CRT screen and a fisheye effect that will distort the screen to mimic the curved surface of an old CRT television screen. The previous “pixel shader” can also be used in combination to further sell the CRT effect.

Example:

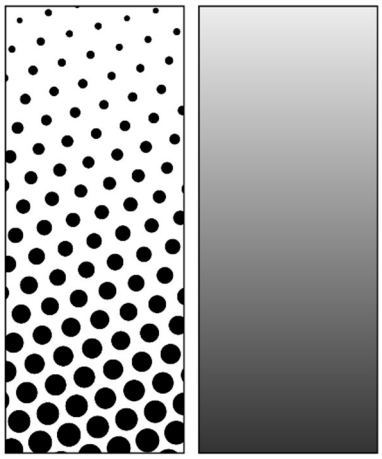


The scanlines mask, otherwise known as a “venetian blinds” effect also has applications in things such as scene transitions. This effect will build upon the understanding of UV manipulation, also introducing the application of masks on top of the scene colour and operating in a custom-defined repeating grid space.

The fisheye effect again furthers the understanding of how UV manipulation can be used. The fisheye effect is also commonly used to give the screen more of a three-dimensional feel, as demonstrated when applied to the user interface in *R.E.P.O*.

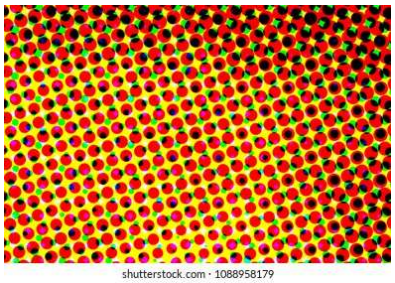
The third and most complex effect will be a black and white “halftone” effect, consisting of a grid of black points on a white background, that will change size to match the greyscale luminance of the scene. This effect will not include any separate components but will produce a highly artistic and interesting scene rendering effect.

Example:



The halftone effect originates from early print techniques used in the production of comic books, where the cyan, magenta, yellow and black dots were layered to create varying colour values. Abstracting this, to just use black dots of varying size to represent light values creates the desired halftone effect.

Comic book CMYK printing



This effect builds upon the UV manipulation and grid masks previously mentioned, while also introducing scene colour manipulation by desaturation.

The original plan for this project was to just develop the retro-style shader, with the focus being on implementing a unique way of sampling scene colours for the pixellated UVs. Preliminary research had shown that of the most common implementations for this solution, using a multiply-floor-divide method, often resulted in an unstable flickering effect as the pixel colour is effectively only sampled from one point, instead of acting as an average of all colours within the cell.

The implementation I had originally planned to work towards, would have included a custom sampling method to reduce this unstable flickering effect. Further research into perspective-stable pixel art then resulted in in-depth technical discussions surrounding texel splatting, wherein scene geometry is instead rendered as a series of cube maps (Ebert, 2026).

While exploring an approach like this would have been an appropriate subject topic for this assessment, it quickly leans away from working with postprocess effects. Furthermore, working on a single solution within the constraints of a postprocess effect would be less valuable than exploring a variety of techniques instead.

Therefore, the project plan was instead defined as building a collection of visually interesting effects, exploring a variety of techniques. The timeline for this project is defined as follows.

Justification & Implementation of Techniques

Pixelation Shader

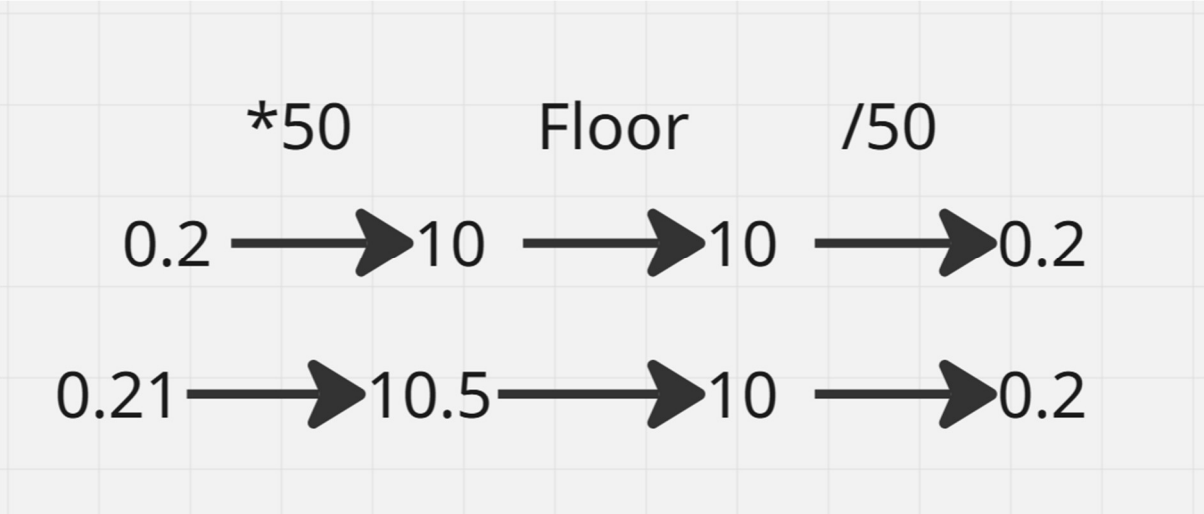
As mentioned previously, when researching how to implement this technique, I decided to implement the multiply-floor-divide method in the interest of adhering to scope and time requirements.

On a standard screen, each pixel will only ever display one specific and uniform colour per frame. These pixels are small enough on most modern resolutions such that a user could not easily identify individual pixels, allowing for the convincing rendering of smooth curves and edges.

When aiming to produce a pixellated screen effect, the size of the "pixels" must be increased such that they are individually identifiable. To implement this, it is important to first understand UVs in the context of screen effects.

"UV" corresponds to the horizontal (U) and vertical (V) coordinates of a two-dimensional texture. When considering postprocess effects, these instead relate to the coordinates of a pixel on the screen. These coordinates range from 0 to 1 on each axis.

To "increase the size of a pixel" on the screen, a group of pixels must represent the same colour each frame. The simplest way to do this is to have each pixel within the group sample the scene colour from the same point. The multiply-floor-divide method will first multiply the UV coordinate of each pixel by our parameter value. In this case our value can be set to 50. This value is then rounded down to the nearest integer using the floor node, then divided again by our parameter value of 50.

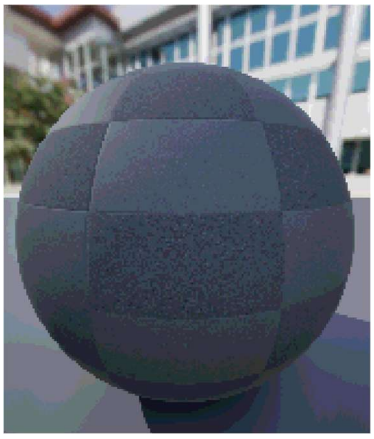


Effectively, the pixel at (0.2, 0.2) and (0.21, 0.21) are now both displaying the scene colour at screen coordinate (0.2, 0.2), resulting in the single point sampling.

This method is very computationally cheap and is frequently recommended as the go-to technique for any application requiring pixelization (Tezenari, 2023).

To ensure these pixels remain square, it is important to multiply the UVs by the aspect ratio.

To complete the retro-style effect, the next step is to reduce the number of available colour channels by a process of colour quantization. This effect can be achieved by simply applying the same method of multiply-floor-divide to the newly sampled scene colours.



Despite the computational efficiency of this solution, the drawbacks of single-point sampling result in flickering, as there are less pixels and colours available to blend between as lighting and perspective changes. This can be solved by implementing different sampling methods, such as Bicubic filtering (Finch & Worlds, 2004) but would drastically increase the complexity of the shader.

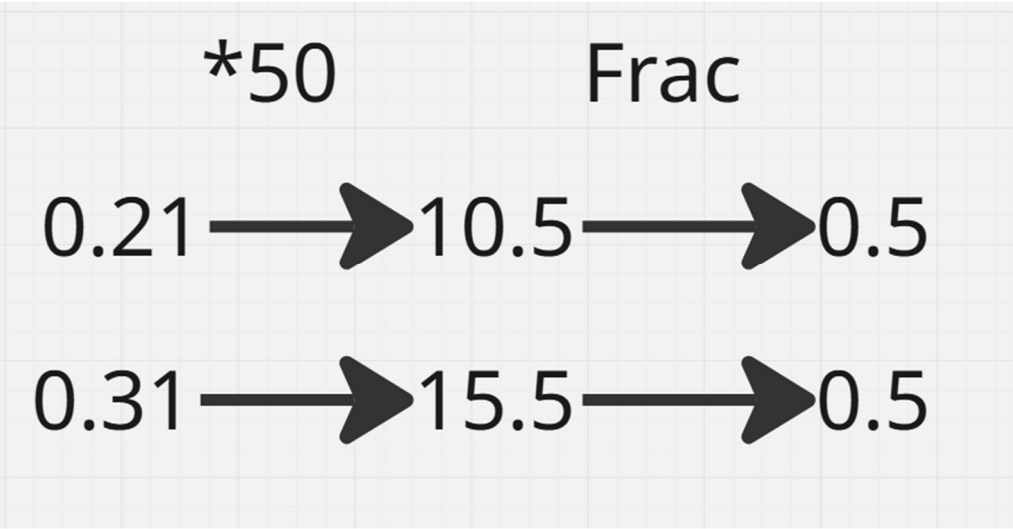
In an ideal solution, the retro-style effect would instead be achieved with the supplementation of appropriate assets and textures. However, the benefit of choosing this method introduces manipulating UVs and postprocess effects as whole, making it a good starting point for this project.

CRT Effect

The CRT effect consists of two essential features, constructed in separate postprocess effects.

The first component, to represent scanlines, introduces a venetian blinds effect. The essence of this component comes from masking out horizontal (or vertical, either works) in a repeating fashion. To target repeating groups of UVs, first multiplying by a density value, to increase the

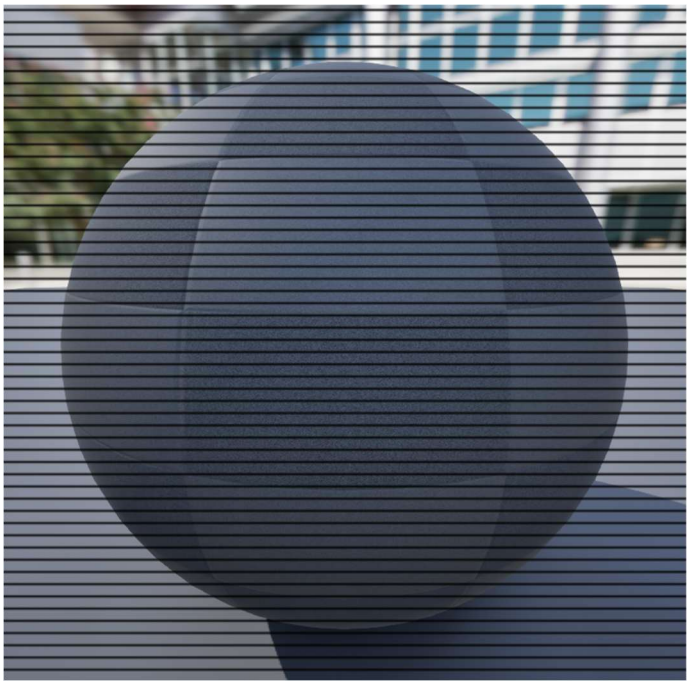
scale of each axis to include multiple integers across the screen. Then, by using a frac node and taking the fractional part of each integer, a repeating region can be defined.



Then, by using a smooth step node, edges of the defined regions can be blended between 0 and 1, and the repeating values can be converted to an alpha mask

This mask is then used with a lerp node, to set the scene colour of the defined regions to zero (black).

By taking the dot product of the screen UVs and a unit directional vector, the repeating mask can be rotated, allowing for a more versatile shader.



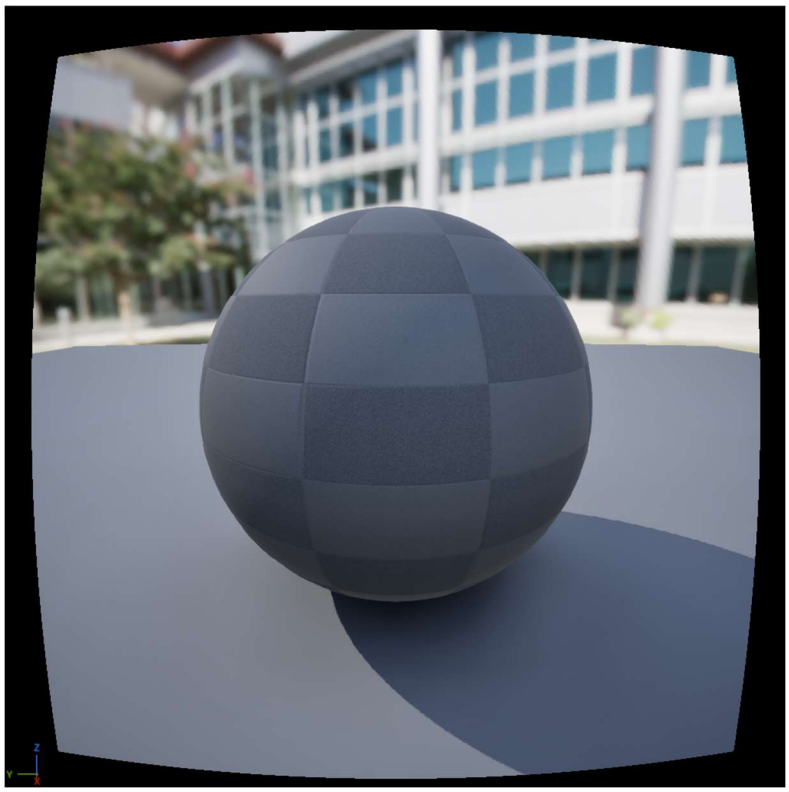
The only issues with this component are the fact that as the completion (bar width) value increases, the bars do not all increase in width simultaneously. However, by stepping values such that the next parameter value is after all widths have been increased, this can be avoided entirely.

The methods used in this effect, exploring solutions in repeating space, will also be used again in the implementation of the halftone effect.

To complete the CRT effect, the fisheye component is needed to represent the screen curvature. This can be achieved by reducing the scale of the UVs based on the distance from the centre of the screen. Fortunately, the “Scale UVs by Center” node (no official documentation could be found) greatly simplifies this but is insufficient when implemented alone.

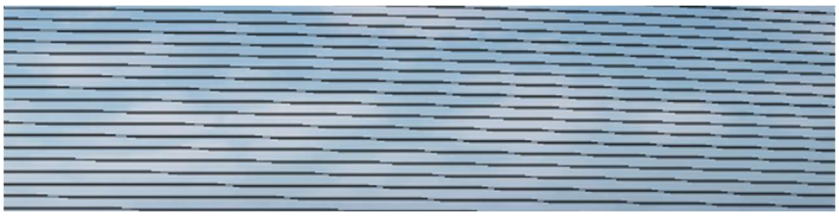
As the edges of the UVs are scaled down, they move closer to the centre of the screen, causing a smearing effect, where there is no adequate information for what should be rendered beyond those edges. To solve this, multiplying the scene colour by the “0-1 mask” returned by the scale UV node, allows the scene beyond the edges of the scale to be set to an alpha value of zero.

This mask can be leveraged to create custom sprite borders to the scene render, further selling the CRT screen effect. In this implementation, I simply decided to scale up the screen UVs such that the edges are not visible, but the curvature still is.



Implementing this effect with this method was much easier than expected, primarily due to the “Scale UVs by centre” node. If I were to implement this in a different engine, it is likely that I would need to create my own implementation for this node. To achieve the same result, the UVs would still need to be reduced in scale based on the distance from (0.5, 0.5) then multiplied by aspect ratio to maintain a consistent size.

One of the main issues with this effect is the tearing on scanlines due to the fisheye effect acting as a separate component



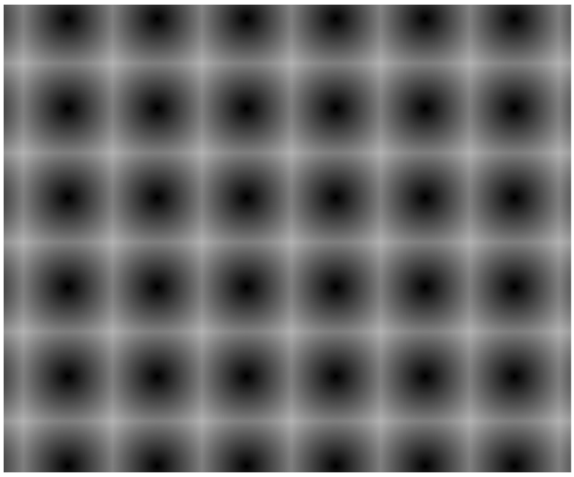
This can be avoided in a more polished solution by including the scanlines and fisheye in the same postprocess effect, avoiding one being baked into the scene colour before the other, allowing a more precise mathematical application of the distortion methods. However, I opted into implementing these effects separately as a way of demonstrating their individual use cases.

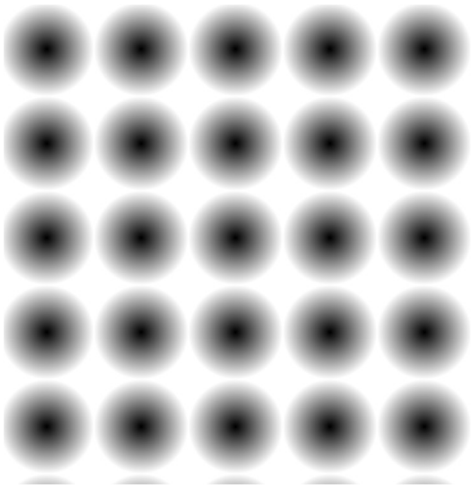
Halftone

The final and most complex postprocess technique I decided to implement was a halftone shading effect.

To begin work on implementing this effect, it is important to break it down into its essential components.

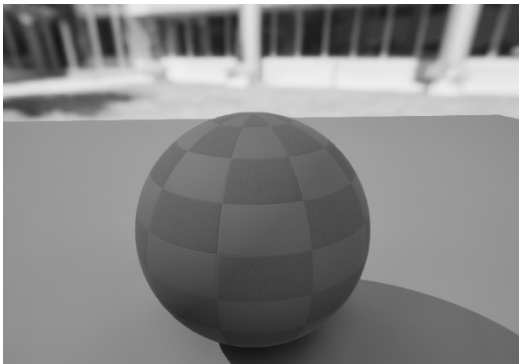
The first part to implement is a repeating grid of dots. By using the pixelation method, to group together pixels using multiply-floor then comparing the distance between the sampled point and the edge of the generated cell, a circular mask can be created.



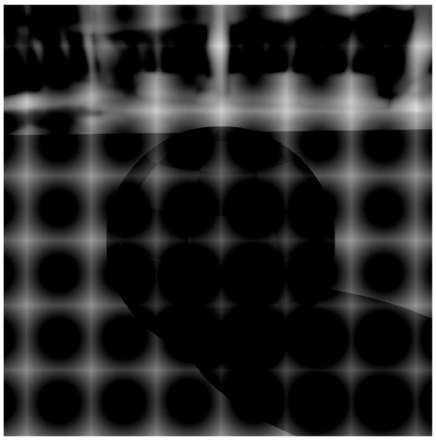


In order to make these dots scale based on the scene luminance, several steps need to be taken beforehand.

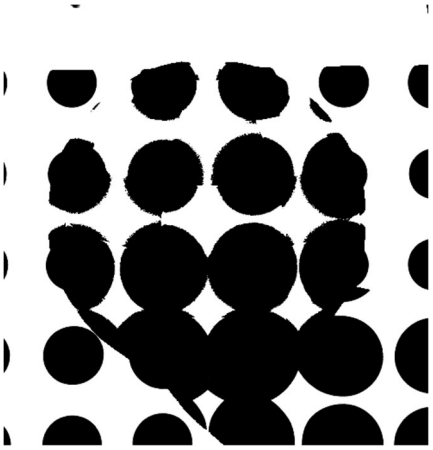
Desaturating the scene colours produces a colour gradient that can be used directly with the previous grid of cells. Unreal Engine provides a node that does this automatically, to achieve this effect otherwise, it is important to consider the BT.709-6 standard of Parameter values for HDTV standards. This standard defines a “Derivation of luminance and colourdifference signals via quantized RGB signals” through a ratio of 0.2126 (R), 0.7152 (G), 0.0722(B). (ITU-R, 2015)



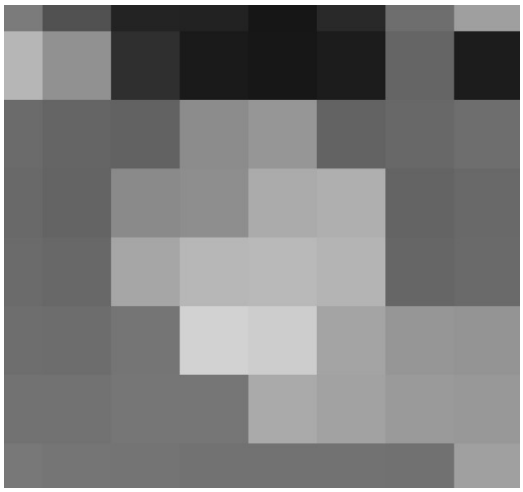
By inverting this greyscale map then subtracting the value from our grid of dots, the dot scale will change to represent the scene luminance beneath it.



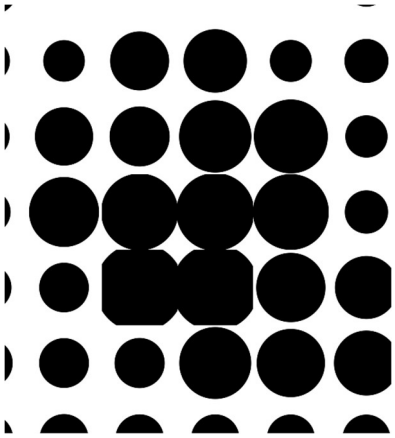
By increasing the contrast (multiplying by around 100) the next issue to solve becomes apparent



Because our postprocess effect is being applied per pixel, instead of per-cell, our dots do not remain round. To solve this, we must apply the same UVs used to generate our cell grid to our scene colour reference.



This then ensures that all pixels used to render our dot, are referencing the same scene colour.



By increasing the number on the screen, the desired halftone shading effect becomes clear.

This method is based on an implementation by *Visual Tech Art* in 2023, who continues to build upon this towards a full comic-book style CMYK print halftone effect. (Visual Tech Art, 2023). To solve the issue of dots being cut off at the edges, the effect must be repeated with an offset, for each adjacent cell. Alternatively, the radius could be limited to half the width of the cell, but this would mean a fully-black value would be unattainable. This unfortunately had to be left incomplete due to time constraints.

I settled on this method due to its relative simplicity, but it came at a trade off in versatility. Experimenting with alternative methods, involving lerp nodes, I was able to specify background and dot colours, as well as being able to rotate the cell UVs to create more interesting styles. However, these methods continued to suffer from the non-round dot issue, making them unsuitable.

It may be possible to implement these alternative effects outside of shader language, where a two-dimensional array of dots can be defined and manipulated with greater freedom, at the cost of performance, due to being unable to leverage GPU processing.

Regardless, the final shader effect was still convincing and gave the originally desired outcome, despite a few minor flaws.

Reflection

From approaching this project with minimal prior experience of developing screen-space shaders, the implementation of these postprocess effects has allowed me to build new skills through exploring a variety of core techniques used in rendering, from UV and colour manipulation to understanding and operating within cell spaces and sampling.

I was able to stick to my original timeline, with the only concession resulting in an incomplete halftone shader. While developing skills by writing shaders using HLSL may have also been valuable, I believe I was justified in my decision to use the node-based material editor as it allowed me to focus on and better understand the concepts that I was exploring. The preview feature was especially helpful in developing and understanding each individual node that was added in real-time.

If I were to continue this project, I would start by completing and polishing each shader to include more parameter control options, reflecting how the *Chameleon VFX library* effects are constructed. I would also like to attempt to recreate some of these shaders using HLSL, developing new skills based on existing understanding and concepts.

Overall, I am pleased with the resulting quality of the developed post-process effects, meaning with only a minor bit of polish it would be appropriate for an interesting portfolio piece.

Bibliography

Ebert, D. (2026). *Texel Splatting: Perspective-Stable 3D Pixel Art* (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2603.14587

Epic Games. (2025a). *Material Editor* [Computer software]. https://dev.epicgames.com/documentation/unreal-engine/unreal-engine-material-editor-user-guide

Epic Games. (2025b). *Unreal Engine 5.6.1* [Computer software]. Epic Games. https://www.unrealengine.com

Finch, M., & Worlds, C. (2004). High-Quality Filtering. In *GPU Gems*. Addison-Wesley Professional. developer.nvidia.com/gpugems/gpugems/part-iv-image-processing/chapter-24-high-quality-filtering

ITU-R. (2015). *BT.709: Parameter values for the HDTV standards for production and international programme exchange*. https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.709-6-201506-I!!PDF-E.pdf

Semiwork Studios. (2025). *R.E.P.O* [Computer software]. Semiwork Studios. https://semiwork.se

SUMFX. (2026). *Chameleon Post Process* [Computer software]. SUMFX.

Tezenari. (2023). Post Process Pixelization. *Unrealdirective*. https://unrealdirective.com/tips/post-process-pixelization/

Visual Tech Art (Director). (2023). *Advanced HalfToning PostProcess [UE5, valid for UE4]* [Video recording]. https://www.youtube.com/watch?v=ldv94GUzTOA