Unreal Engine Reusable Stylised Framework

Haohan Zhang

MSc Computer Animation and Visual Effects

Bournemouth University

September 2025

Abstract

Stylized rendering has grown increasingly important in contemporary games and animation, particularly within anime-inspired visual styles. While traditional cartoon shaders offer distinct lighting bands and outlines, their implementation in production environments often yields mediocre results. This project focuses on exploring a reusable real-time cartoon shader framework implemented in Unreal Engine 5.5. It aims to establish a relatively efficient workflow for achieving stylized rendering across different styles, integrating physically based rendering with non-photorealistic techniques for character rendering. This approach provides a stable and fast process for cartoon rendering while offering solutions with greater artistic flexibility.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Jon Macey, and Jian Chang for their continuous guidance, encouragement, and invaluable feedback throughout the course of this project. Their professional advice and patient guidance helped me overcome many challenges.

Additionally, I am grateful to my family and friends for their unwavering encouragement and understanding during my studies. Their support has been a constant source of motivation.

Contents

List of Figures

1 Introduction

In recent years, the demand for stylized rendering in computer animation and games has grown significantly, particularly in titles adopting anime-inspired or “cartoon” aesthetics. The popularity of games like The Legend of Zelda series and Genshin Impact, alongside films such as Spider-Man: Across the Spider-Verse, demonstrates increasing acceptance of non-photorealistic rendering styles.

Traditional NPR and cartoon shading methods have established effective techniques for stylized lighting and contour rendering (Gooch et al., 1998), but these methods are often designed for specific use cases and lack the modularity required for contemporary production environments (Barla et al., 2006). In contrast, modern game engines like Unreal Engine and Unity have introduced Shader Graph tools and official documentation for stylized rendering (Epic Games Developer, 2025; Unity Technologies, 2019), making these techniques more accessible to non-programmers. However, built-in tools remain limited: shadow boundaries often flicker during animation, gradient textures require manual adjustments, and reusing shader components across projects remains cumbersome. Emerging stylized games also reveal studios exploring combinations of non-photorealistic rendering (NPR) with physically based rendering (PBR). This approach balances artistic style with technical realism, though its implementation grows more complex.

For stylized artists, these limitations create cumbersome workflows. On one hand, stable and efficient cartoon shading is crucial for maintaining consistent visual styles across game characters and environments. On the other hand, the lack of reusable frameworks forces teams to repeatedly build similar solutions, leading to inefficient production. Therefore, this project focuses on implementing a shader framework that is not only visually convincing but also parametric, reusable, and relatively friendly to the art workflow.

This project addresses the issue by developing a reusable real-time cartoon shader framework within Unreal Engine 5.5. The framework will implement several stylised effects rapidly and reliably, leveraging the engine's updated post-process materials. It offers artists seeking to experiment with rendering stylised characters using a blend of non-photorealistic rendering (NPR) and physically based rendering (PBR) techniques a relatively straightforward approach, granting them greater creative scope and artistic freedom.

2 Related Work

Non-photorealistic rendering (NPR) has been an active research field in computer graphics for more than two decades, focusing on techniques that go beyond realism to achieve stylised, illustrative, or cartoon-like appearances. One of the earliest influential works in this area is the lighting model proposed by Gooch et al. (1998), which replaced traditional Phong shading with cool-to-warm colour interpolation, enhancing shape perception while maintaining clarity. Decaudin (1996) introduced one of the earliest practical algorithms for cartoon-style contour rendering, laying the foundation for many subsequent edge-detection methods. Gooch et al. (2001) further explored real-time stylised rendering, establishing the feasibility of interactive cartoon shading in animation and games.

Building on these foundations, more advanced cartoon shading techniques have been developed to provide greater flexibility. Barla et al. (2006) proposed X-Toon, a framework that allows the use of lookup tables to control diffuse and specular responses, giving artists finer stylistic control. These methods reflect a broader trend in NPR research: balancing artistic flexibility with computational efficiency. As technology has advanced, stylised rendering has also become increasingly diverse. Line rendering has remained a recurring research theme; Kim et al. (2015) provided a comprehensive tutorial on drawing lines from 3D models, covering both geometry-based and image-space edge detection methods.

The application of cartoon shading in the game industry demonstrates its practical significance. Stylised rendering has been a key factor in the success of games such as *The Legend of Zelda* series and *Borderlands*, which employ carefully designed shading and contour techniques to reinforce their art direction. More recently, miHoYo's *Genshin Impact* has popularised an anime-inspired NPR pipeline that integrates quantised lighting bands, stable shadows, and custom shaders for hair and eyes (Gdcvault.com, 2021). Such large-scale productions indicate that NPR is not only of academic interest but also a cornerstone of contemporary game art.

Game engines have also incorporated support for stylised rendering into their workflows. Unity provides Shader Graph nodes for toon shading and rim lighting (Unity Technologies, 2019), while Unreal Engine's documentation offers guidance on stylised materials and post-processing (Epic Games Developer, 2025). These official resources highlight the growing demand for accessible and reusable NPR techniques in standard pipelines. However, despite these advances, existing engine-level tools often lack stable shadow boundaries and flexible gradient control, requiring technical artists to extend or customise shaders.

In summary, this project aims to bridge this gap by developing a reusable, parameterised toon shading framework in Unreal Engine, designed to provide a relatively stable and streamlined workflow for artists.

3 Technical Background

3.1 Non-Photorealistic Rendering (NPR)

Traditionally, computer graphics has focused on photorealism, which involves simulating the physical properties of light to generate lifelike images. In contrast, non-photorealistic rendering (NPR) aims to create stylised, illustrative, or artistic images rather than strict realism (Gooch et al., 2001). NPR encompasses techniques such as painterly rendering, shadow lines, cartoon rendering, and line art. Within this broad category, cartoon rendering (also termed ‘cartoon-style rendering’ or ‘anime-style rendering’) stands as one of the most widely employed methods due to its relevance in animation and gaming.

Cartoon shading typically simplifies the shading model by quantising continuous light values into discrete banded areas, creating sharp distinctions between light and shadow. This produces the distinctly ‘flat’ yet expressive appearance characteristic of animation and comic art (Todo et al., 2009). Beyond shading, NPR techniques often incorporate contour detection to draw outlines around objects, further enhancing the image's illustrative quality.

3.2 Toon Shading Principles

At the core of toon shading is the modification of the standard Lambertian lighting equation:

$$I = k_d \cdot \max\,(N \cdot L)$$

where N is the surface normal, L is the light direction, and k_d is the diffuse color. Instead of using the raw dot product, toon shading maps this value into discrete levels using a **ramp function** or thresholding. For example, a two-band toon shader may render pixels either fully lit or fully shadowed, whereas multi-band ramps provide smoother stylized gradations.

Key extensions of this model include:

- **Specular highlights**: Quantized or threshold-based highlights to simulate the sharp, stylized shine often seen in hair or polished surfaces.
- **Rim lighting**: An additional lighting term emphasizing object silhouettes relative to the camera, frequently used in anime to enhance readability.
- **Ramp textures**: Custom 1D textures that map lighting intensity to color, enabling a wide variety of stylistic effects beyond simple hard bands.

3.3 Outline Rendering Techniques

Outlines are another crucial element in cartoon shading, enhancing the hand-drawn effect. There are primarily two approaches:

- **Geometry-based outlines (mesh extrusion):** The object's mesh is rendered again using inverted normals and extended vertices, producing stable and efficient outlines.
- **Image-space outlines (post-process edge detection):** Utilising depth and normal buffers, edges are detected through filters such as the Sobel operator. This method captures both external outlines and internal feature lines.

Modern NPR systems typically combine both approaches: mesh extrusion for stable external outlines and screen-space methods for internal detail lines (Liao, 2023).

3.4 Physically Based Rendering (PBR)

By contrast, Physically Based Rendering (PBR) has become the industry standard for photorealistic rendering in modern engines. PBR materials rely on physically based models to simulate the interaction between diffuse reflection, specular reflection, and roughness, enabling consistent image generation under varying lighting conditions (Kumar, 2020).

Key components of PBR include:

- **Energy conservation:** Ensuring a material reflects no more light than it receives.
- **Fresnel effects:** Angle-dependent reflections.
- **Micro-surface models:** Employing statistical models of surface roughness to calculate realistic highlights**.**

Whilst PBR excels at photorealism, it frequently conflicts with the stylised objectives of NPR. Consequently, PBR+ cartoon-shading hybrid rendering for stylised characters seeks to retain PBR's physical realism whilst overlaying NPR techniques to achieve stylistic control.

-

3.5 Unreal Engine 5.5 Rendering Pipeline

Unreal Engine 5.5 introduces several features related to stylised rendering:

- **• Deferred rendering pipeline:** Efficiently handles multiple lights while still permitting customised post-processing.
- **• Material Parameter Collection (MPC):** Provides a mechanism for passing global values (e.g., light direction) to shaders.
- **• Post-Processing Materials:** Enable screen-space effects such as outline detection and colour quantisation.

These features make Unreal Engine particularly well-suited for implementing reusable stylised rendering frameworks, combining advanced rendering infrastructure with the scalability of stylised workflows.

4 Solution

4.1 Post Process Materials

To address the limitations of standard Unreal Engine materials in achieving stable and expressive non-photorealistic rendering, this section employs Unreal Engine's post-process materials to implement a modular shader framework for stylized rendering in real-time environments. This solution centers on six core components—Hatching, Ink, Mixer, Outline, Toon Shader, and Vignette—collectively forming a versatile toolkit. Each module exposes key parameters via Material Instance, enabling users to rapidly preview rendering effects and adjust styles at the asset/level layer without modifying textures.

4.1.1 Hatching

The Hatching module introduces line-based shading reminiscent of traditional pen-and-ink illustrations. Two layers of rotated texture coordinates are sampled and blended to generate primary and secondary hatch directions. By adjusting UV scale, rotation, and blend thresholds, artists can simulate variable shading density corresponding to light intensity. This achieves expressive shadow rendering, complementing the cartoonish flat colors with additional texture and detail (Figure 4-1 & Figure 4-2).

Key Parameters (MI_Hatching):

- **Overall Hatching UV Scale** – density of hatching lines. Lower → denser shading.
- **Hatching Thickness** – adjusts line width. Negative values produce very thin lines.
- **Overall Hatching Rotation** – rotates the entire hatch pattern.
- **Primary Hatching Area / Secondary Hatching Area** – threshold values controlling where each hatching layer appears. Higher → only darkest regions are affected.
- **Secondary Hatching Rotation / Offset** – avoids moiré patterns and increases textural richness.

Impact:

- Primary ≈ 8–12, Secondary ≈ 14–18 produces subtle tonal layering.
- Angling secondary rotation at 25°–45° mimics cross-hatching in traditional illustration.

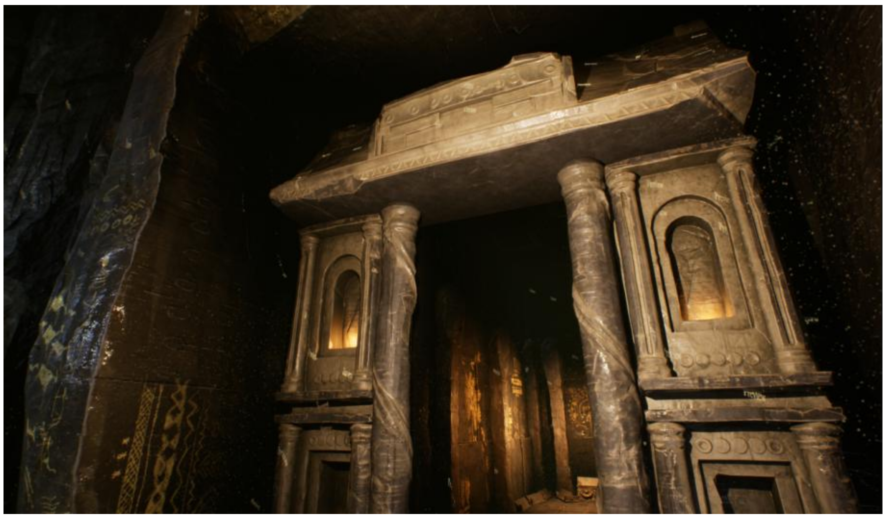


Figure 4-1: Original Scene Render

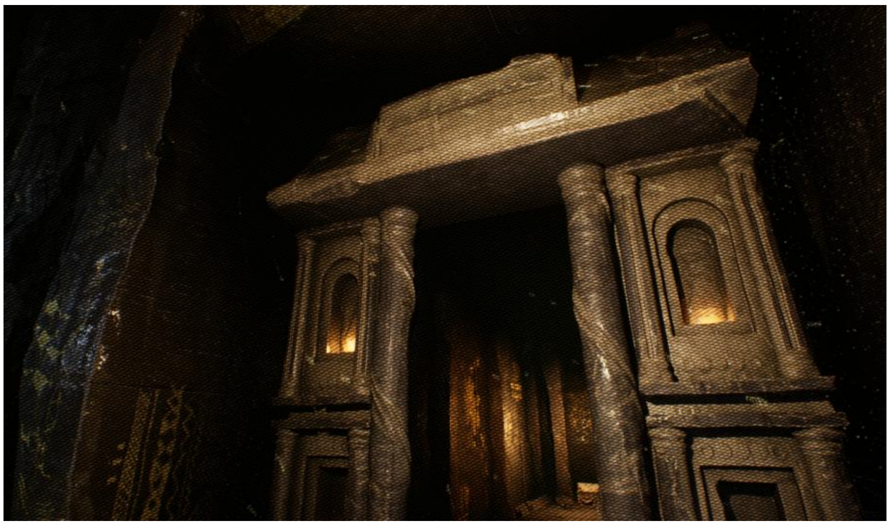


Figure 4-2: Rendering effect with MI_Hatching material added during post-processing

4.1.2 Ink

The Ink module is designed to replicate the bold contouring typical of hand-drawn comic art. Building on screen-space normal and depth information, outlines are extracted via thresholding and composited back into the scene. The Ink implementation includes toggles for black-on-white or white-on-black presentation, expanding its usability for different artistic contexts such as manga or storyboard visualization. Unlike a simple Sobel filter, this module supports parameterized width and intensity, providing control over stylistic expressiveness (Figure 4-3 & Figure 4-4).

Key Parameters (MI_Ink):

- **Two Tone** – enables dual thresholds for shadows (dark + light).
- **Darker Shadows Cutoff / Color** – darkest areas, usually pure black. Higher cutoff → fewer areas covered.
- **Lighter Shadows Cutoff / Color** – lighter shadow band.
- **Denoise** – smooths transition edges, stabilizing shading in motion.

Impact:

- Close-up characters: Darker cutoff ≈ 0.08–0.12, Lighter cutoff ≈ 0.18–0.25.
- For distant backgrounds, increase cutoff values to avoid overly large black areas.

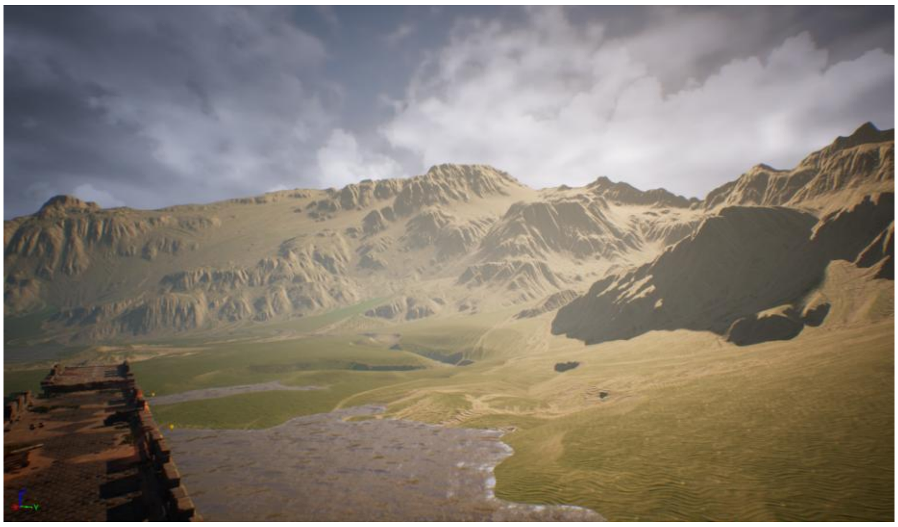


Figure 4-3: Original Scene Render

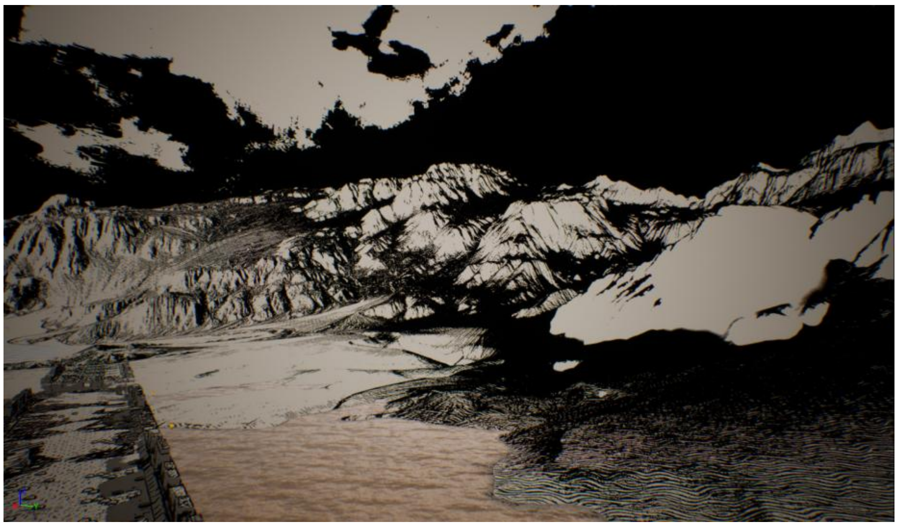


Figure 4-4: Rendering effect with MI_Ink material added during post-processing

4.1.3 Outline

Complementing the Ink channel, the Outline module employs geometry-based extrusion techniques. Mesh normals are inverted and offset to generate stable outlines around the model, proving particularly effective for character rendering. When combined with Ink's screen-space outlines, this dual approach strikes a balance between stability and internal detail capture. The two outline sources within this design—Depth-Based Outlines and Diffuse-Based Outlines—are dedicated to achieving stable outer contours under varying conditions, complemented by optional 'diffuse-contrast' inner lines. This resolves issues of scale and detail fidelity at varying distances within UE5 (Figure 4-5 & Figure 4-6).

Key Parameters (MI_Outlines):

- **Line Width (near / far)** – outline thickness for near vs. distant objects.
- **Threshold (near / far)** – sensitivity to depth or color changes. Higher → captures finer details.
- **Blend Distance** – defines near-to-far transition range.
- **Exponential Blend Distance / Exponent** – non-linear scaling for large outdoor scenes. Default exponent = 2.
- **Diffuse Based Outlines** – supplements depth-based lines, useful for flat but patterned surfaces.
- **White Background** – outputs outlines on a pure white canvas for concept art export.
- **Debug Plane** – visualizes where blending transitions occur.

Impact:

- Recommended values: Near Width ≈ 1.0–1.5, Far Width ≈ 0.1–0.3.
- Outdoor shots: enable Exponential Blend with Distance ≈ 15–25, Exponent ≈ 2.
- Use Diffuse outlines for low-contrast materials (fabric, ceramic tiles).

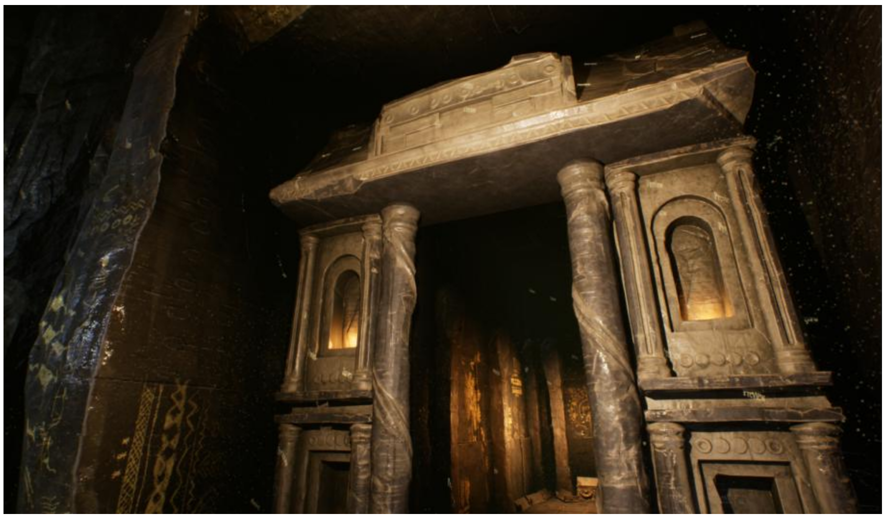


Figure 4-5: Original Scene Render

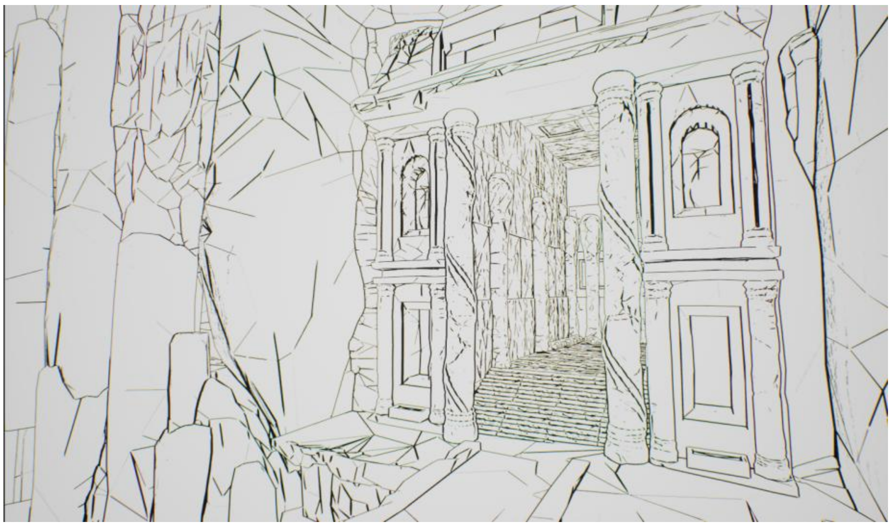


Figure 4-6: Rendering effect with MI_Outines material added during post-processing

4.1.4 Toon Shader

The Toon Shader module forms the core of this framework. It achieves quantized diffuse illumination through gradient textures, stable shadow boundaries, and an optional specular reflection threshold. Parameters such as band count, softness, and edge intensity are presented via material instances. This ensures consistency across multiple assets whilst still affording artists flexibility to experiment with stylistic variations. The module guarantees stable cartoon shading under camera and light animation, thereby overcoming common limitations of simpler implementations (Figure 4-7 & Figure 4-8).

Key Parameters (MI_ToonShader):

- **Diffuse Color Brightness** – boosts albedo contribution after toon quantization.
- **Mix in Original Lighting** – Combined with the original lighting
- **Shadows 1–4 (Cutoff & Color)** – up to four discrete shadow bands. Each cutoff = threshold; color defines tone.
- **Denoise** – stabilizes shadow bands.
- **Outline Distance Blend** – consistent outline behavior with the Outline module.

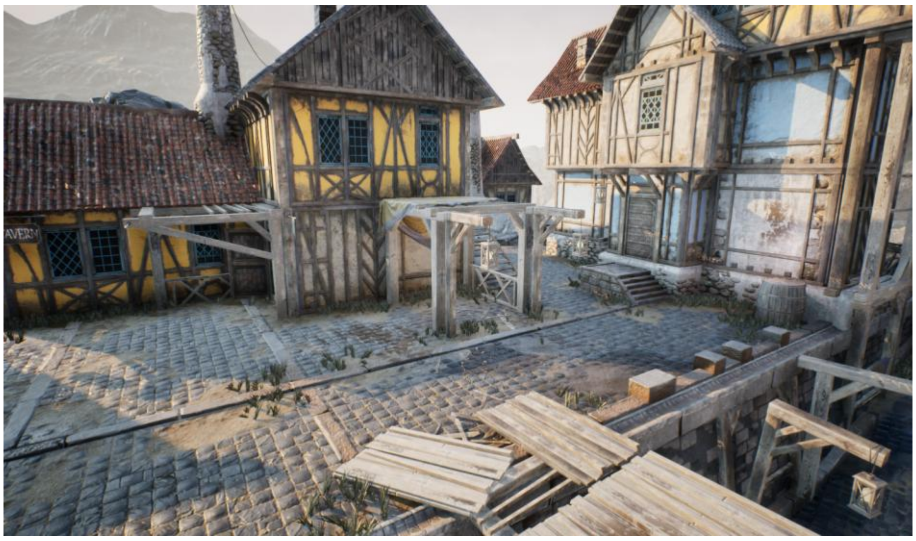


Figure 4-7: Original Scene Render

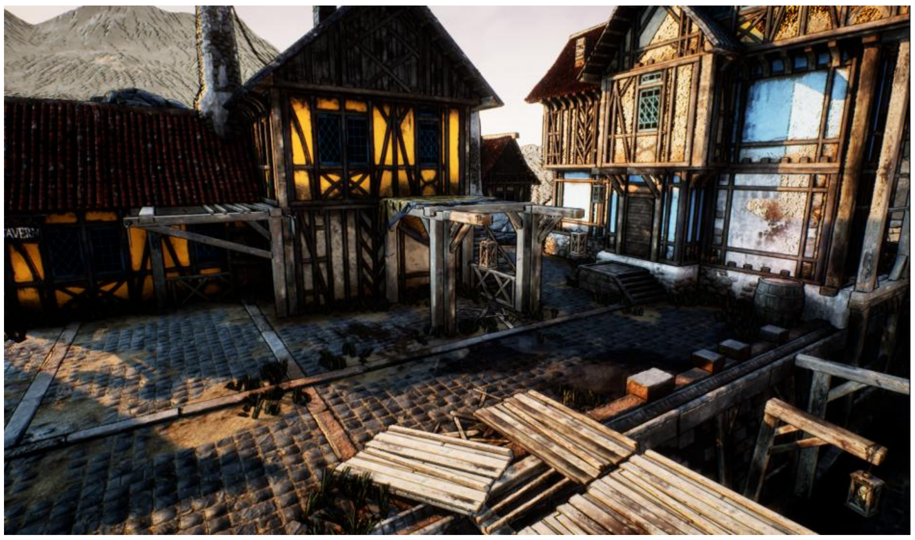


Figure 4-8: Rendering effect with MI_ToonShader material added during post-processing

4.1.5 Vignette

The vignette module introduces an additional post-processing layer that simulates the soft, fading effects characteristic of traditional illustration and photography. Implemented as a screen-space radial mask, it darkens or fades the edges of the image, thereby directing visual focus towards the subject. Whilst the vignette effect is somewhat more subtle compared to other modules, it contributes to elevating the overall cinematic quality of the rendering pipeline and enhances visual readability.

Key Parameters (MI_Vignette):

- **Intensity / Hardness / UV Scale** – per-vignette control of strength, softness, and radius.
- **Color** – tone of vignette (neutral, warm, or cool).
- **Hatched Vignette** – applies hatching lines within vignette areas for illustration-like style.
- **Vignette Texture Slots (T_Vignette1–4)** – customizable feathering and shape masks.

4.1.6 Mixer

The Mixer module functions as a synthesizer, blending multiple NPR layers—such as Hatching and Ink—with the Toon Shader base layer. By assigning adjustable weights to each input, the Mixer facilitates the blending of visual styles, encompassing everything from flat cartoon rendering to textured illustration. As it enables rapid exploration of blended aesthetics without requiring new shader code, it significantly reduces the time needed to produce hybrid artistic styles (Figure 4-9).

Key Parameters (from MI_Mixer):

- **Features Toggles** – switch on/off Hatching, Ink, Outlines, Vignette, White BG.
- **Outlines Distance Blend** – ensures global consistency of line scaling.
- **Lighting-only Source Option** – ensures modules derive from lighting data only, aligning with Greyshade behavior.

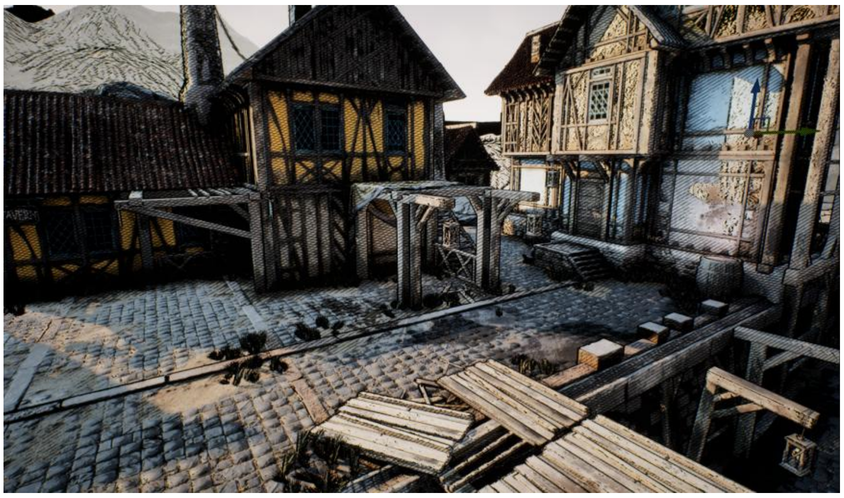


Figure 4-9: Rendering effect with MI_Mixer (Hatching, Outlines and ToonShader) material added during post-processing

4.1.7 Integration and Workflow

All modules are implemented as material functions within Unreal Engine 5.5 and combined via the master material. This solution exposes parameters through material instances, enabling artists to customise shading at the resource or project level without modifying the underlying shader graph. The workflow is also highly convenient, requiring artists only to create a PostProcessVolume within the current level and add Post Process Materials.

The solution's modular design permits selective activation of different effects. For instance, in scenarios demanding high performance, only the Toon Shader and Outline effects may be enabled, whilst Hatching and Vignette effects can be disabled. Conversely, for concept art rendering or pre-production visualisation, Mixer can combine multiple stylisation modules to achieve experimental effects.

In summary, this solution extends Unreal Engine's standard stylised rendering capabilities through a modular NPR framework. By integrating Hatching, Ink, Outline, Toon Shader, Vignette, and Mixer, the framework ensures both stability and stylistic diversity.

4.2 NPR+PBR Character Rendering Material

This shader primarily offers artists wishing to experiment with PBR+NPR character rendering a straightforward approach. Some HLSL code for the effects has already been written in the custom parameters. It requires only creating material instances for all base textures and assigning them to a master material (M_PBR_NPR). By adjusting parameters, the entire character's rendering can be achieved.

4.2.1 Shader Structure

The master material (M_PBR_NPR) is composed of several key functional blocks:

1. **Texture Calibration (Base Color)**

   Inputs a standard BaseTex (albedo) and applies optional RGB calibration and tonemapping before mixing with toon ramp colors.

   The calibration ensures consistency across scanned PBR textures and hand-painted stylized textures.

2. **Lighting Model**

   Uses Dot (Normal, LightDir) with SkyAtmosphereLightDirection(0) to compute diffuse shading.

   Supports Signed Distance Field (SDF) based shadow masking for sharper stylized edges, selectable with a UseSDF? parameter.

   A Ramp gradient is applied to quantize shading into discrete toon bands.

3. **Ramp Color Mixing**

   A Ramp texture (1D LUT or gradient) is sampled using the shadow mask.

   Provides control over shadow thresholds and highlight falloff, allowing multiple bands (e.g., light / midtone / shadow).

   The artist can override with Final Color or additional MatCap inputs.

4. **MatCap Integration**

   A camera-facing MatCap lookup is calculated using world normals and camera vectors (see Matcap block screenshot).

   This allows painterly rim-lighting, anisotropic sheen, or eye reflections without complex PBR setup.

   Separate MatCap Main (base shading) and MatCap Mask (selective blending) are provided.

5. **RMO Map Support**

   Standard Roughness / Metallic / AO packed maps (RMO_Tex) are supported.

Artists may override with color sliders (Metallic_Color, Roughness_Color) or MatCap-based replacements (Metallic_Matcap, Roughness_Matcap).

This enables mixing between physically-based surface response and stylized overrides.

6. **Highlight / Specular Control**

   Custom nodes introduce NoseHighlight (anime-style facial highlights) and optional extra rim light.

   Allows exaggeration of specific anatomical features beyond standard BRDF.

4.2.2 Usage for Character Assets

- **For clothing**

  Uses BaseTex with RMO map.

  Ramp factor ≈ 2.0 for moderate cel-band separation.

  Metallic and Roughness overridden with MatCaps for a more “illustrated fabric” look.

  Produces believable folds with stylized shading transitions.

- **For the eyes**

  Ramp_NoShadow for consistent flat tones (to avoid noisy shadow flicker on eyes).

  MatCap provides anime-style circular highlight.

  Shadows are disabled to maintain clean readability.

- **For the skin**

  Ramp banding with 2–3 tones.

  NoseHighlight and MatCap rim light accentuate form without realistic subsurface scattering.

  Allows dynamic expression while keeping “anime look.”

- **For hair**

  Customise hair highlights using Matcap to enhance dimensionality.

  Ramp_Hair adjusts hair colour variation and gradients.

4.2.3 Technical Impact

This material provides characters with greater detail compared to conventional cartoon rendering (e.g., shadows, highlights, cloth/metal textures) while achieving more harmonious overall coloration (Figure 4-10 & Figure 4-11). Through blending controls, artists can also choose to use either PBR-based or NPR-based rendering exclusively by adjusting blend values, offering greater flexibility in rendering expression. Additionally, when creating characters of the same type, artists can opt to replace only the textures to quickly render another character, enhancing flexibility.

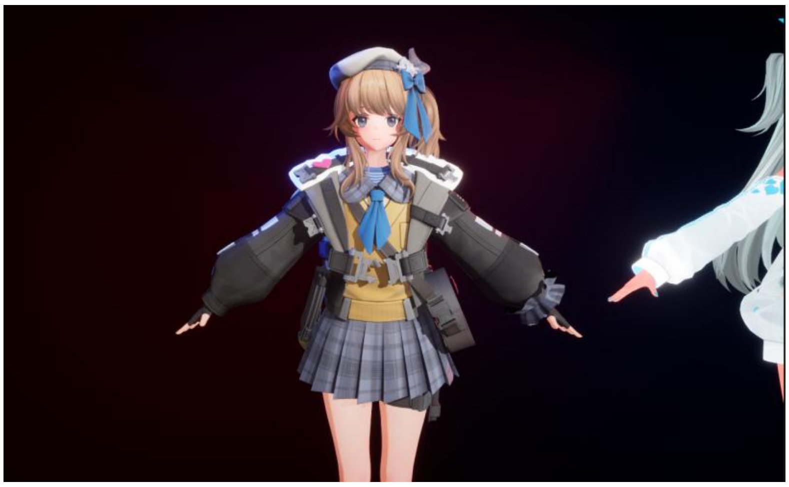


Figure 4-10: Characters rendered using the M_PBR_NPR material.

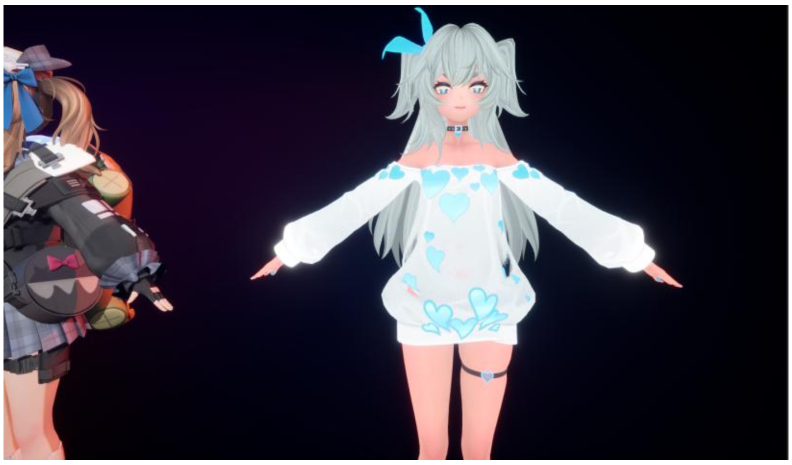


Figure 4-11: Standard cartoon-style rendered characters.

5 Conclusions

This project successfully achieved its primary objectives by developing a reusable real-time stylised rendering framework within Unreal Engine 5.5. The shader system was constructed using a modular and reusable approach, incorporating master materials, material functions, and adjustable material instances. This design facilitates easy adaptation across different projects and assets. The BR+NPR shader strikes a balance between physically based realism and stylisation. Artists can smoothly interpolate between fully photorealistic PBR shading and flat cartoon shading by adjusting exposed parameters, thereby simulating the hybrid workflows employed in some game productions. The framework performs well on Windows platforms, significantly accelerating workflows. For simple post-processing operations, the instruction count is approximately 150, while complex character shaders require around 300–350 instructions.

Despite these achievements, the project retains certain limitations:

Shader Complexity: Compared to traditional PBR-only materials, PBR+NPR hybrid shaders are relatively expensive, limiting scalability for large scenes or background assets.

Environment Integration: The PBR+NPR framework is primarily optimised for character rendering. Applying the same techniques to environments, props, or atmospheric effects requires additional adjustments.

Workflow Streamlining: PBR+NPR character rendering requires artists to manually create material instances for each character texture, presenting a slight inconvenience.

Future work can address these limitations through targeted improvements:

Unified Lighting Model: Develop a single, consistent lighting and shadow model to inform post-process NPR effects and per-material shaders, enhancing visual coherence.

Creative Tools: Implementing editor utilities—such as custom gradient editors, MatCap preview tools, or node-based stylisation controls—would streamline artistic workflows.

Model Import: Create streamlined import tools that automatically generate corresponding material instances and parameters during character import.

Procedural Stylisation: Integrating procedural noise functions or machine learning techniques to dynamically generate gradients, shadows, or shadow patterns could expand stylistic diversity.

Overall, this project demonstrates considerable technical artistry and offers further insights for advancing stylised rendering workflows and their industry applications.

6 References

West, R. and Mukherjee, S. (2024) “Stylized Rendering as a Function of Expectation,” *ACM Transactions on Graphics (TOG)*, 43(4), pp. 1–19. Available at: https://doi.org/10.1145/3658161.

Tang, C. et al. (2010) “Stable stylized wireframe rendering,” *Computer Animation and Virtual Worlds*, 21(3-4), pp. 411–421. Available at: https://doi.org/10.1002/cav.370.

Petikam, L., Anjyo, K. and Rhee, T. (2021) “Shading Rig: Dynamic Art-directable Stylised Shading for 3D Characters,” *ACM Transactions on Graphics (TOG)*, 40(5), pp. 1–14. Available at: https://doi.org/10.1145/3461696.

Todo, H., Anjyo, K. and Igarashi, T. (2009) “Stylized lighting for cartoon shader,” *Computer Animation and Virtual Worlds*, 20(2-3), pp. 143–152. Available at: https://doi.org/10.1002/cav.301.

Epic Games Developer. (2025). *Stylized Rendering | Unreal Engine 4.27 Documentation | Epic Developer Community*. [online] Available at: https://dev.epicgames.com/documentation/en-us/unreal-engine/stylized-rendering?application_version=4.27 [Accessed 15 Sep. 2025].

Gdcvault.com. (2021). *‘Genshin Impact’: Crafting an Anime Style Open World*. [online] Available at: https://gdcvault.com/play/1027538/-Genshin-Impact-Crafting-an [Accessed 14 Sep. 2025].

Liao, J. and 2023 13th International Conference on Information Technology in Medicine and Education (ITME) Wuyishan, China 2023 Nov. 24 - 2023 Nov. 26 (2023) “The Research of Cel-Shading in Non-photorealistic Rendering,” in *2023 13th International Conference on*

Information Technology in Medicine and Education (ITME), pp. 569–572. Available at: https://doi.org/10.1109/ITME60234.2023.00119.

Polycount. (2025). *Documentation: Concept Art Shader 2 - for Unreal Engine*. [online] Available at: https://polycount.com/discussion/236761/documentation-concept-art-shader-2-for-unreal-engine [Accessed 15 Sep. 2025].

Gooch, B. and Gooch, A. (2001) *Non-Photorealistic Rendering*. Natick: CRC Press LLC. Available at: https://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=5939381 (Accessed: September 13, 2025).

Gooch, A., Gooch, B., Shirley, P. and Cohen, E. (1998) 'A non-photorealistic lighting model for automatic technical illustration', *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*, New York, NY, USA: Association for Computing Machinery, pp. 447–452. Available at: https://doi.org/10.1145/280814.280950.

Barla, P., Thollot, J. and Markosian, L. (2006) 'X-toon: an extended toon shader', *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering (NPAR '06)*, New York, NY, USA: Association for Computing Machinery, pp. 127–132. Available at: https://doi.org/10.1145/1124728.1124749.

エピック ゲームズ ジャパン (2023). *Stylized Rendering Insights from Japan (Unreal Fest Gold Coast 2023).* [online] Docswell. Available at: https://www.docswell.com/s/EpicGamesJapan/5DEVPV-2023-12-01-082936 [Accessed 15 Sep. 2025].

Unity Technologies (2019). *Unity - Manual: Unity User Manual (2019.2)*. [online] Unity3d.com. Available at: https://docs.unity3d.com/Manual/index.html.

Philippe Decaudin (1996). *Cartoon Rendering of 3D Scenes, Cel Shading (Philippe Decaudin)*. [online] Sourceforge.net. Available at: https://phildec.users.sourceforge.net/Research/Cartoon.php [Accessed 15 Sep. 2025].

Bui, Kim, J. and Lee, Y. (2015) "3D-look shading from contours and hatching strokes," *Computers & Graphics*, 51, pp. 167–176. Available at: https://doi.org/10.1016/j.cag.2015.05.026.

Kumar, A. (2020) *Beginning PBR texturing : learn physically based rendering with allegorithmic's substance painter*. [United States]: Apress. Available at: https://doi.org/10.1007/978-1-4842-5899-6.

7 Appendices

7.1 Appendix A: Blueprints

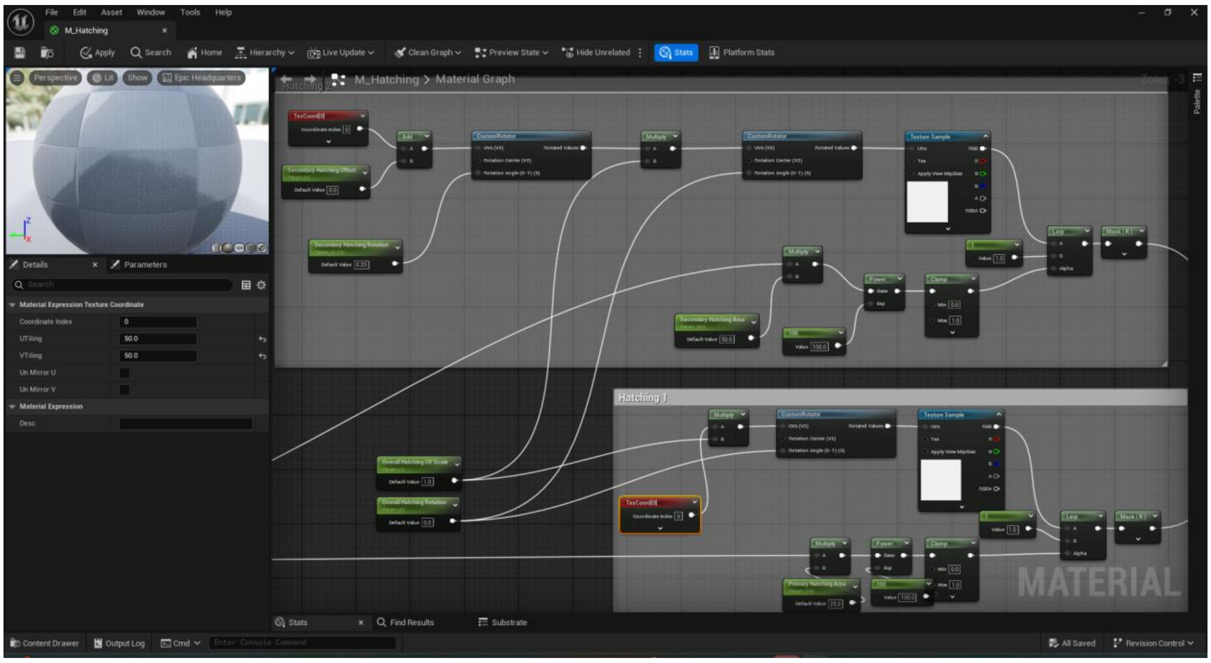


Figure A.1 Hatching Blueprint

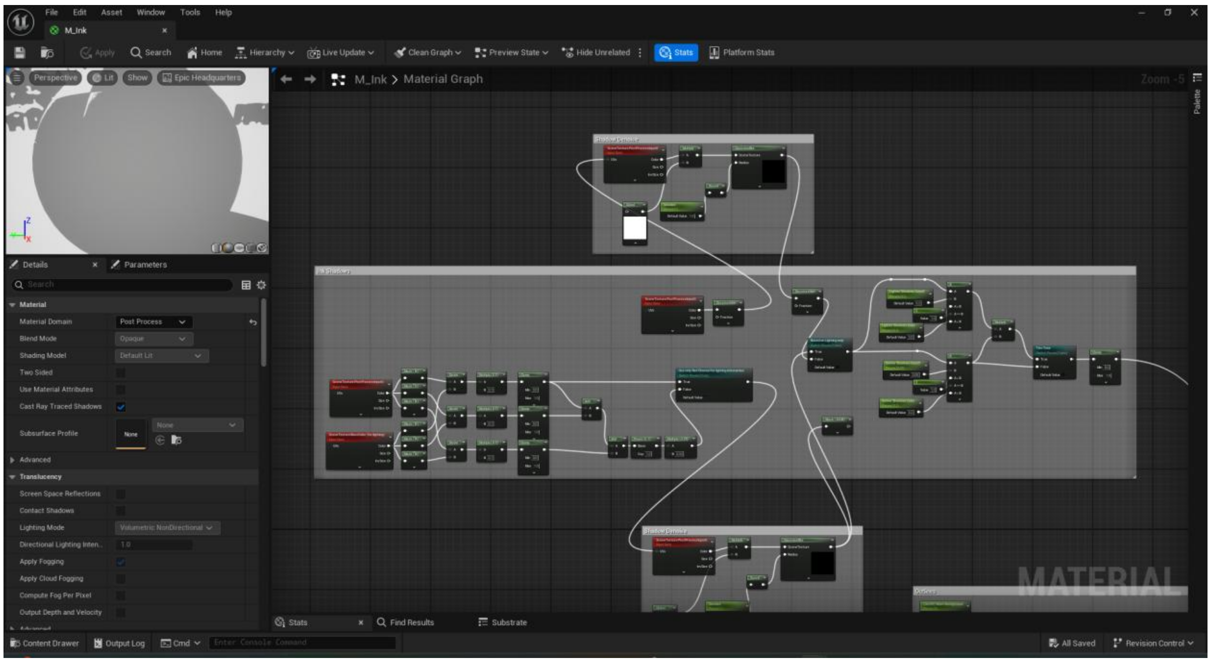


Figure A.2 Ink Blueprint

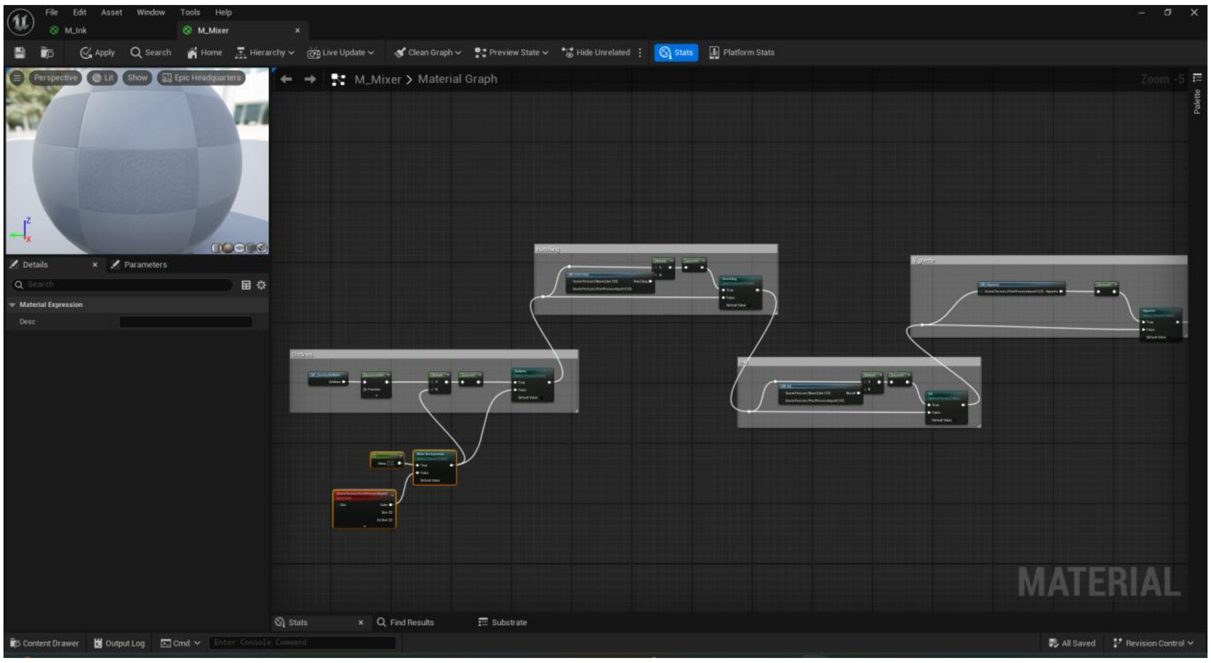
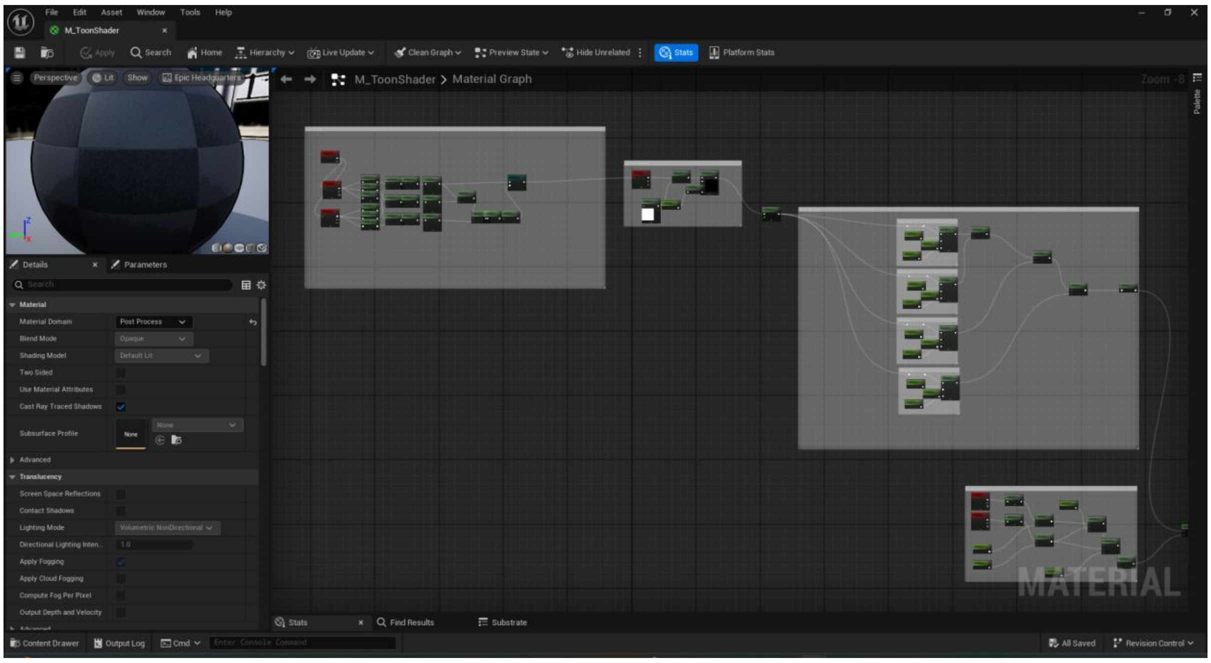
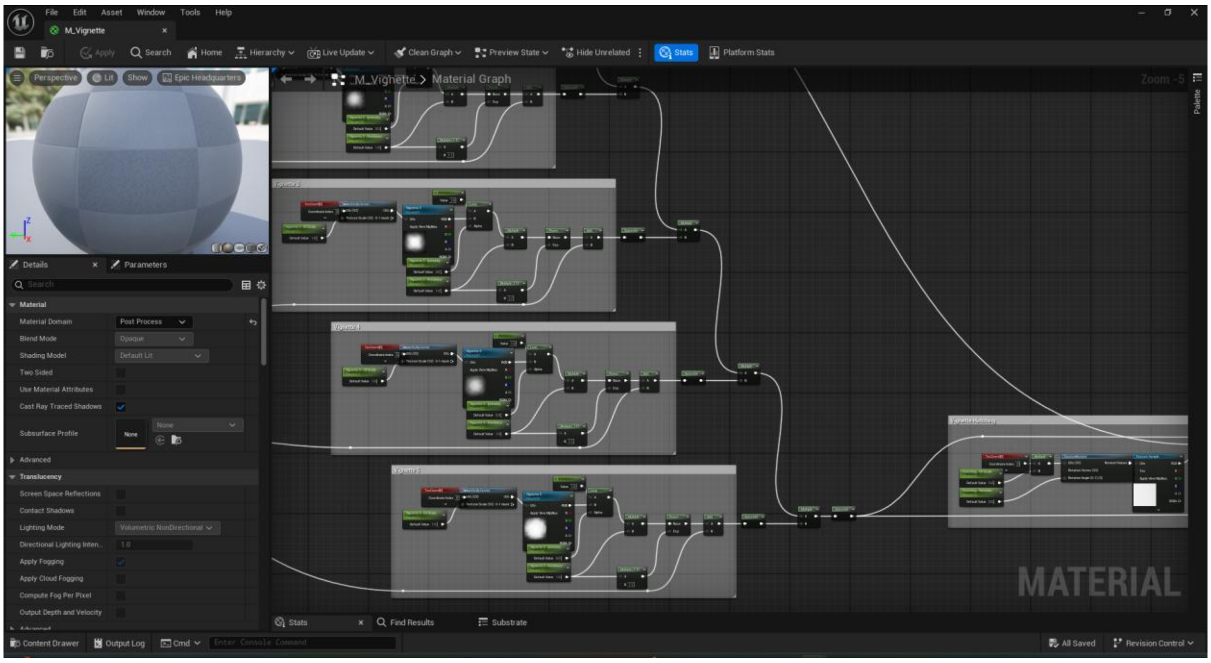


Figure A.3 Mixer Blueprint



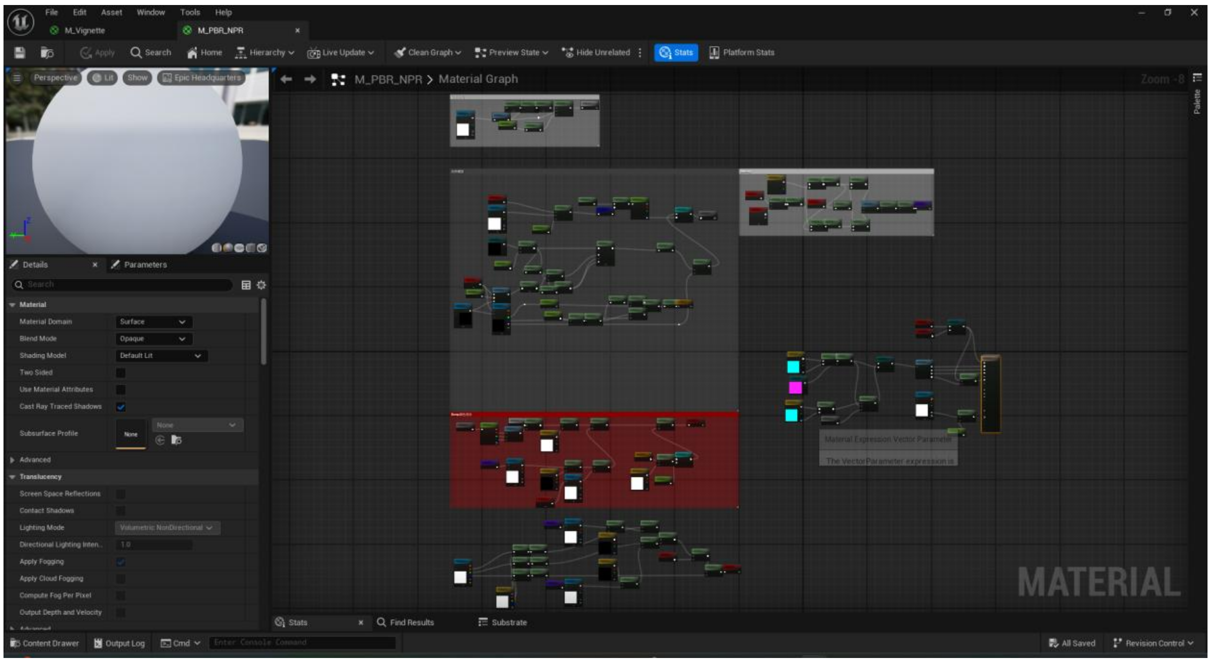
Material
Material Domain: Post Process
Blend Mode: Opaque
Shading Model: Default Lit
Two Sided
Use Material Attributes
Cast Ray Traced Shadows ✓
Subsurface Profile: None
Advanced
Translucency
Screen Space Reflections
Contact Shadows
Lighting Mode: Volumetric NonDirectional
Directional Lighting Inten.. 1.0
Apply Fogging ✓
Apply Cloud Fogging
Compute Fog Per Pixel
Output Depth and Velocity

Figure A.4 ToonShader Blueprint


File Edit Asset Window Tools Help
M_Vignette
Apply Search Home Hierarchy Live Update Clean Graph Preview State Hide Unrelated Stats Platform Stats
Perspective Lit Show Epic Headquarters
M_Vignette > Material Graph
Zoom -5
Palette
Details Parameters
Stats Find Results Substrate
Content Drawer Output Log Cmd Enter Console Command
All Saved Revision Control
MATERIAL

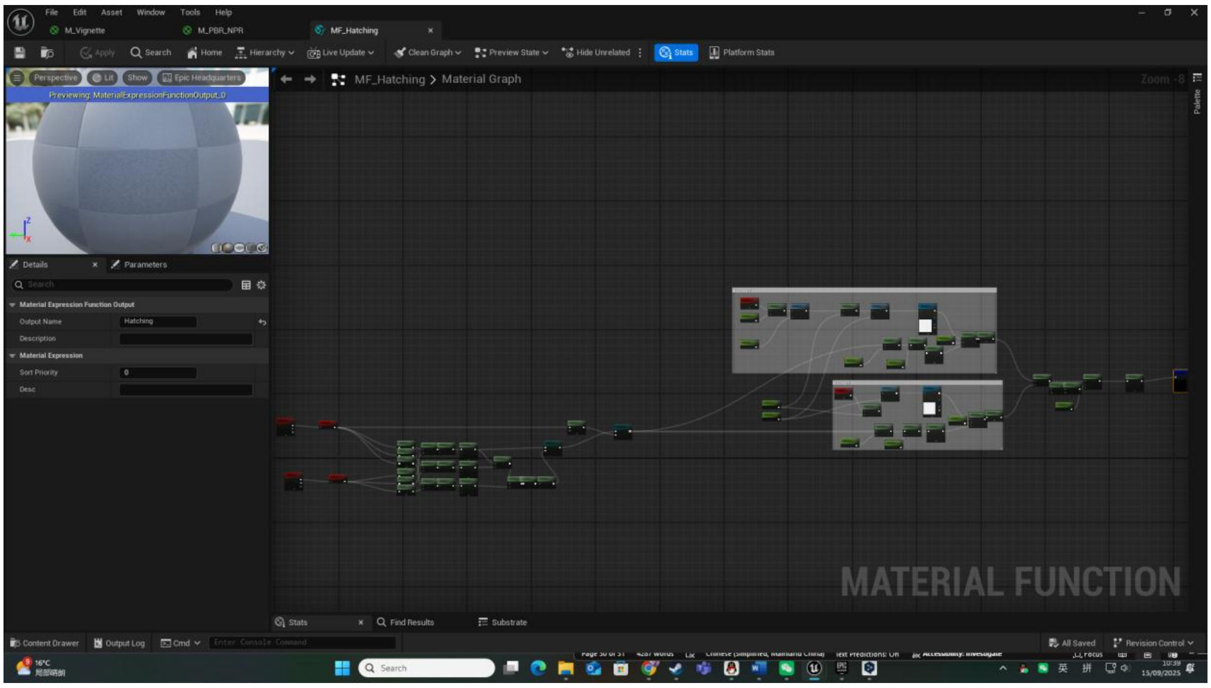
Material
Material Domain: Post Process
Blend Mode: Opaque
Shading Model: Default Lit
Two Sided
Use Material Attributes
Cast Ray Traced Shadows ✓
Subsurface Profile: None
Advanced
Translucency
Screen Space Reflections
Contact Shadows
Lighting Mode: Volumetric NonDirectional
Directional Lighting Inten...: 1.0
Apply Fogging ✓
Apply Cloud Fogging
Compute Fog Per Pixel
Output Depth and Velocity

Figure A.5 Vignette Blueprint


File Edit Asset Window Tools Help
M_Vignette
M_PBR_NPR
Apply Search Home Hierarchy Live Update Clean Graph Preview State Hide Unrelated Stats Platform Stats
Perspective Lit Show Epic Headquarters
M_PBR_NPR > Material Graph
Zoom -8
Palette
Details Parameters
Material Expression Vector Parameter
The VectorParameter expression is
Stats Find Results Substrate
Content Drawer Output Log Cmd Enter Console Command
All Saved Revision Control
MATERIAL

Material
Material Domain: Surface
Blend Mode: Opaque
Shading Model: Default Lit
Two Sided
Use Material Attributes
Cast Ray Traced Shadows ✓
Subsurface Profile: None
Advanced
Translucency
Screen Space Reflections
Contact Shadows
Lighting Mode: Volumetric NonDirectional
Directional Lighting Inten...: 1.0
Apply Fogging ✓
Apply Cloud Fogging
Compute Fog Per Pixel
Output Depth and Velocity

Figure A.6 M_PBR_NPR Blueprint



Previewing: MaterialExpressionFunctionOutput_0

Material Expression Function Output

Output Name: Hatching

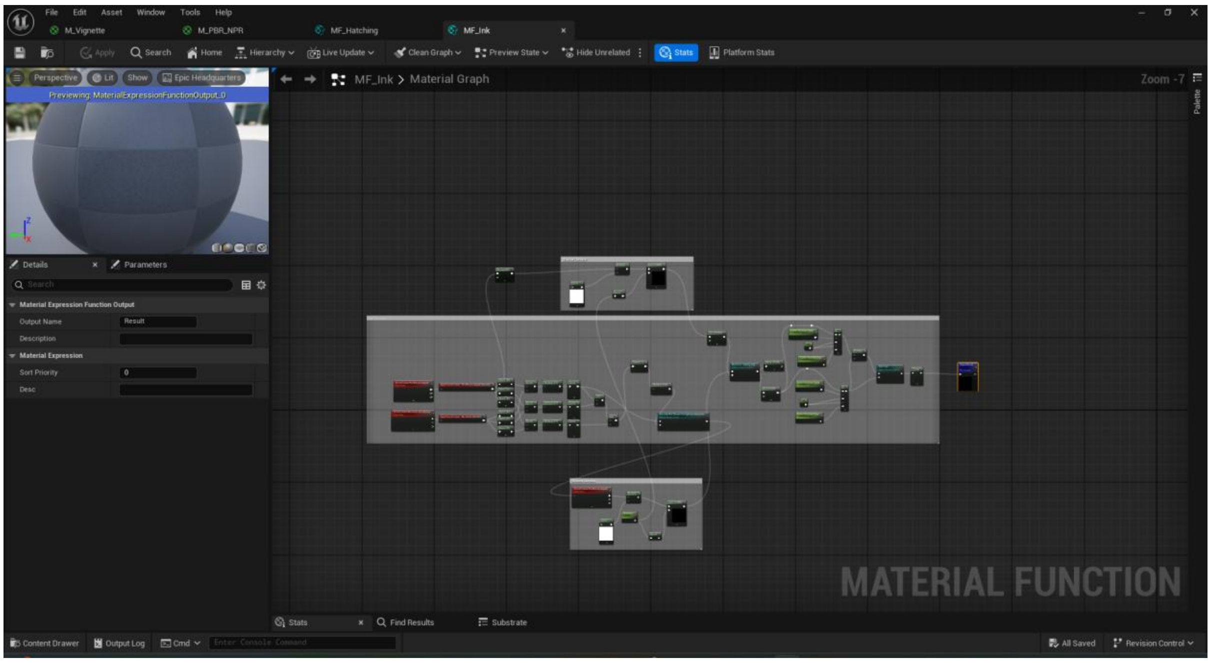
Description:

Material Expression

Sort Priority: 0

Desc:

Figure A.7 Hatching Function Blueprint



Previewing: MaterialExpressionFunctionOutput_0

Material Expression Function Output

Output Name: Result

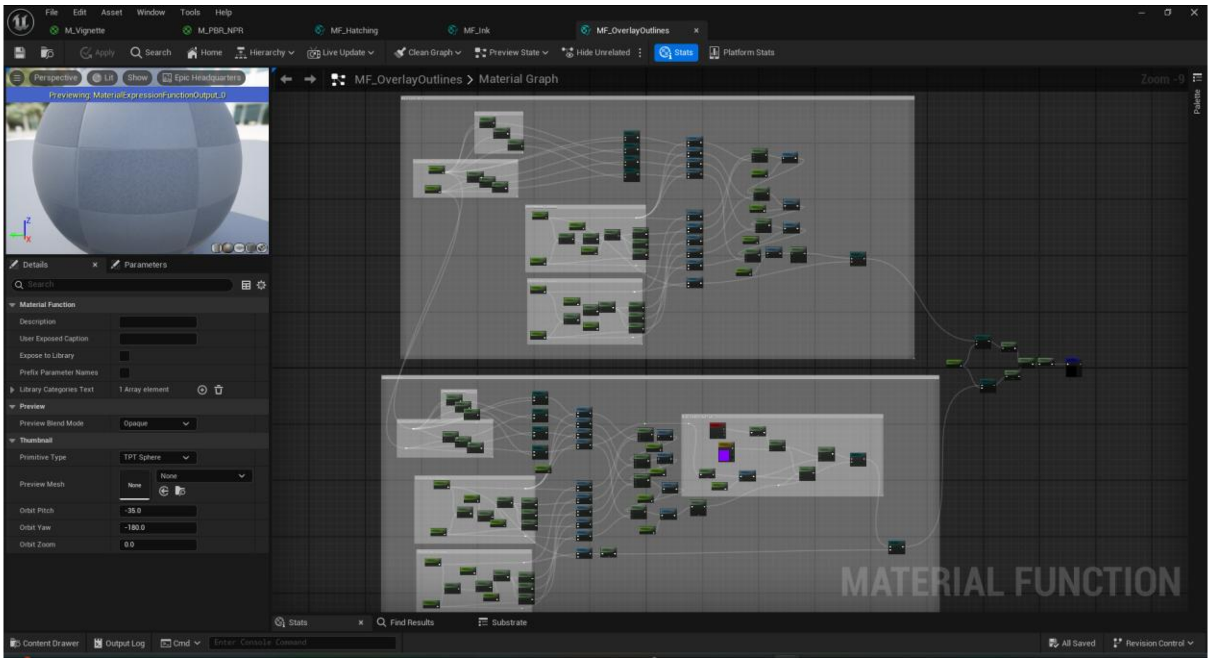
Description:

Material Expression

Sort Priority: 0

Desc:

Figure A.7 Ink Function Blueprint



Material Function

Description	
User Exposed Caption	
Expose to Library	
Prefix Parameter Names	
Library Categories Text	1 Array element

Preview

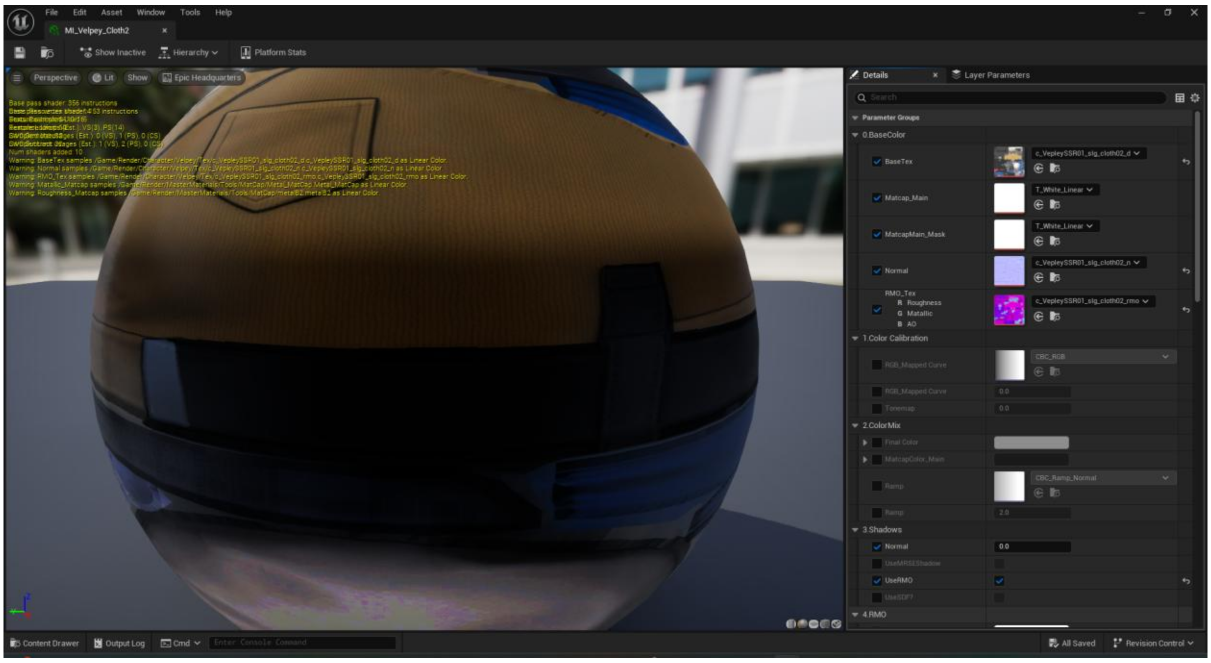
Preview Blend Mode	Opaque

Thumbnail

Primitive Type	TPT Sphere
Preview Mesh	None
Orbit Pitch	-35.0
Orbit Yaw	-180.0
Orbit Zoom	0.0

Figure A.8 Outlines Function Blueprint

7.2 Appendix B: Material Instance Parameters



```
Base pass shader: 356 instructions
Base pass vertex shader: 453 instructions
Texture samplers: 4/16
Texture lookups (Est.): VS(3), PS(14)
Shader stages (Est.): 0 (VS), 1 (PS), 0 (CS)
Constant stages (Est.): 1 (VS), 2 (PS), 0 (CS)
Num shaders added: 10
Warning: BaseTex samples /Game/Render/Character/Velpey/Tex/c_VepleySSR01_slg_cloth02_d.c_VepleySSR01_slg_cloth02_d as Linear Color.
Warning: Normal samples /Game/Render/Character/Velpey/Tex/c_VepleySSR01_slg_cloth02_n.c_VepleySSR01_slg_cloth02_n as Linear Color.
Warning: RMO_Tex samples /Game/Render/Character/Velpey/Tex/c_VepleySSR01_slg_cloth02_rmo.c_VepleySSR01_slg_cloth02_rmo as Linear Color.
Warning: Matalic_Matcap samples /Game/Render/MasterMaterials/Tools/MatCap/Metal_MatCap.Metal_MatCap as Linear Color.
Warning: Roughness_Matcap samples /Game/Render/MasterMaterials/Tools/MatCap/metalB2.metalB2 as Linear Color.
```

Parameter Groups

0.BaseColor

✓ BaseTex	c_VepleySSR01_slg_cloth02_d
✓ Matcap_Main	T_White_Linear
✓ MatcapMain_Mask	T_White_Linear
✓ Normal	c_VepleySSR01_slg_cloth02_n
✓ RMO_Tex (R Roughness, G Matallic, B AO)	c_VepleySSR01_slg_cloth02_rmo

1.Color Calibration

RGB_Mapped Curve	CBC_RGB
RGB_Mapped Curve	0.0
Tonemap	0.0

2.ColorMix

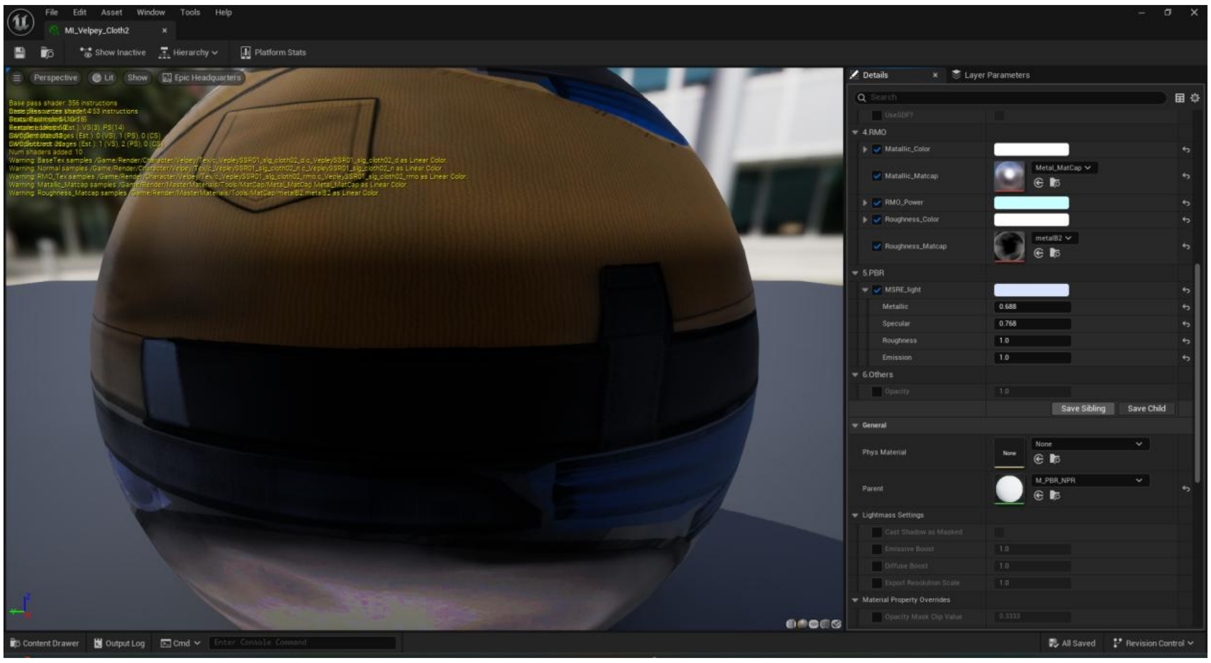
Final Color	
MatcapColor_Main	
Ramp	CBC_Ramp_Normal
Ramp	2.0

3.Shadows

✓ Normal	0.0
UseMRSEShadow	
✓ UseRMO	✓
UseSDF?	

4.RMO

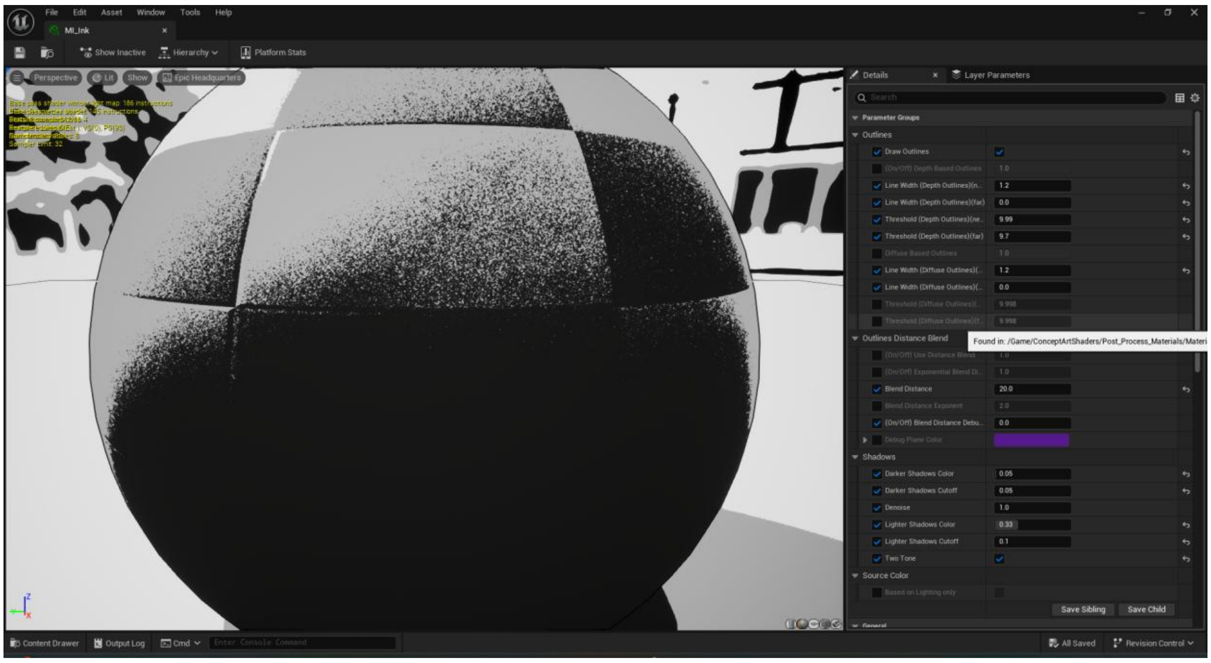
Figure B.1 M_PBR_NPR Material Instance Parameters part1



```
Base pass shader: 356 instructions
[illegible] 453 instructions
[illegible]
[illegible] (Est.): VS(3), PS(14)
[illegible] (Est.): 0 (VS), 1 (PS), 0 (CS)
[illegible] (Est.): 1 (VS), 2 (PS), 0 (CS)
Num shaders added: 10
Warning: BaseTex samples /Game/Render/Character/Velpey/Tex/c_VepleySSR01_slg_cloth02_d.c_VepleySSR01_slg_cloth02_d as Linear Color.
Warning: Normal samples /Game/Render/Character/Velpey/Tex/c_VepleySSR01_slg_cloth02_n.c_VepleySSR01_slg_cloth02_n as Linear Color.
Warning: RMO_Tex samples /Game/Render/Character/Velpey/Tex/c_VepleySSR01_slg_cloth02_rmo.c_VepleySSR01_slg_cloth02_rmo as Linear Color.
Warning: Matallic_Matcap samples /Game/Render/MasterMaterials/Tools/MatCap/Metal_MatCap.Metal_MatCap as Linear Color.
Warning: Roughness_Matcap samples /Game/Render/MasterMaterials/Tools/MatCap/metalB2.metalB2 as Linear Color.
```

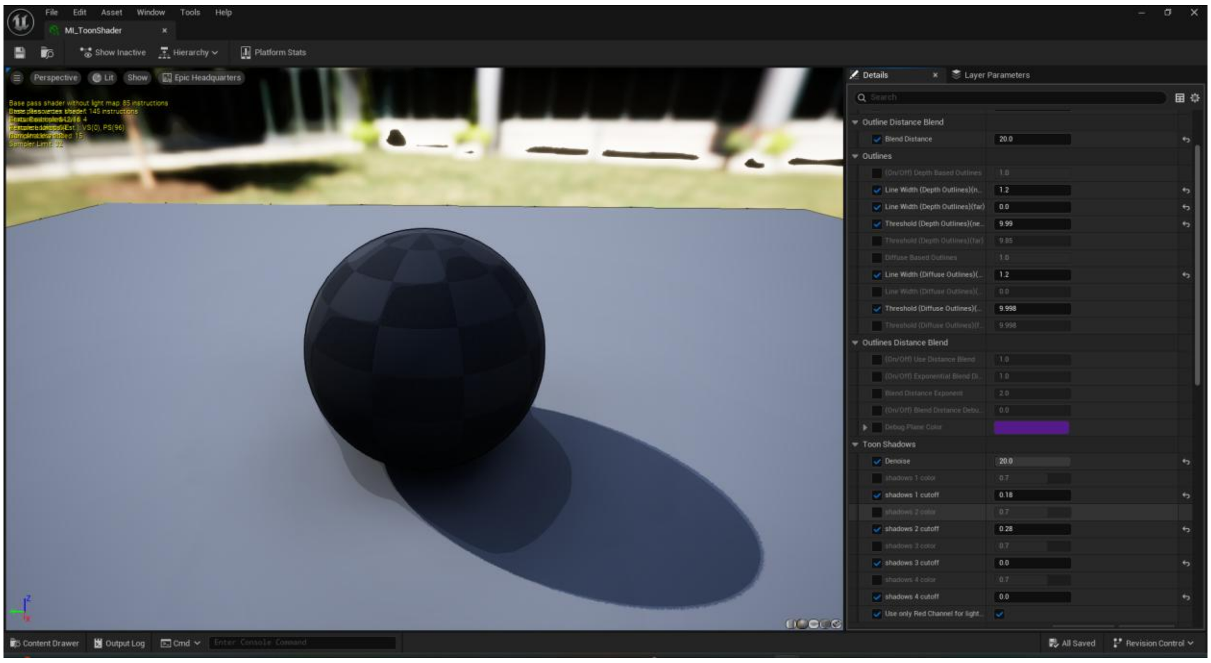
UseSDF?	
4.RMO	
✓ Matallic_Color	⬜
✓ Matallic_Matcap	Metal_MatCap
✓ RMO_Power	🟦
✓ Roughness_Color	⬜
✓ Roughness_Matcap	metalB2
5.PBR	
✓ MSRE_light	🟦
Metallic	0.688
Specular	0.768
Roughness	1.0
Emission	1.0
6.Others	
Opacity	1.0
General	
Phys Material	None
Parent	M_PBR_NPR
Lightmass Settings	
Cast Shadow as Masked	
Emissive Boost	1.0
Diffuse Boost	1.0
Export Resolution Scale	1.0
Material Property Overrides	
Opacity Mask Clip Value	0.3333

Figure B.2 M_PBR_NPR Material Instance Parameters part2



Parameter Groups	
Outlines	
✓ Draw Outlines	✓
(On/Off) Depth Based Outlines	1.0
✓ Line Width (Depth Outlines)(n...	1.2
✓ Line Width (Depth Outlines)(far)	0.0
✓ Threshold (Depth Outlines)(ne...	9.99
✓ Threshold (Depth Outlines)(far)	9.7
Diffuse Based Outlines	1.0
✓ Line Width (Diffuse Outlines)(...	1.2
✓ Line Width (Diffuse Outlines)(...	0.0
Threshold (Diffuse Outlines)(...	9.998
Threshold (Diffuse Outlines)(f...	9.998
Outlines Distance Blend	Found in: /Game/ConceptArtShaders/Post_Process_Materials/Materi
(On/Off) Use Distance Blend	1.0
(On/Off) Exponential Blend Di...	1.0
✓ Blend Distance	20.0
Blend Distance Exponent	2.0
✓ (On/Off) Blend Distance Debu...	0.0
Debug Plane Color	🟪
Shadows	
✓ Darker Shadows Color	0.05
✓ Darker Shadows Cutoff	0.05
✓ Denoise	1.0
✓ Lighter Shadows Color	0.33
✓ Lighter Shadows Cutoff	0.1
✓ Two Tone	✓
Source Color	
Based on Lighting only	
General	

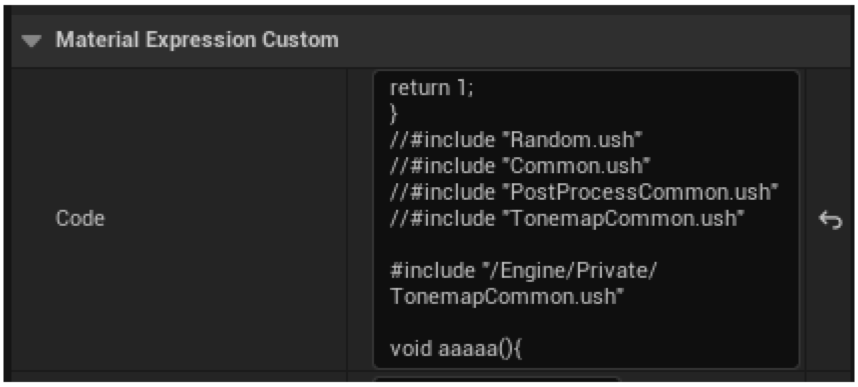
Figure B.3 M_Ink Material Instance Parameters



Outline Distance Blend		
✓	Blend Distance	20.0
Outlines		
	(On/Off) Depth Based Outlines	1.0
✓	Line Width (Depth Outlines)(n...	1.2
✓	Line Width (Depth Outlines)(far)	0.0
✓	Threshold (Depth Outlines)(ne...	9.99
	Threshold (Depth Outlines)(far)	9.85
	Diffuse Based Outlines	1.0
✓	Line Width (Diffuse Outlines)(...	1.2
	Line Width (Diffuse Outlines)(...	0.0
✓	Threshold (Diffuse Outlines)(...	9.998
	Threshold (Diffuse Outlines)(f...	9.998
Outlines Distance Blend		
	(On/Off) Use Distance Blend	1.0
	(On/Off) Exponential Blend Di...	1.0
	Blend Distance Exponent	2.0
	(On/Off) Blend Distance Debu...	0.0
	Debug Plane Color	🟪
Toon Shadows		
✓	Denoise	20.0
	shadows 1 color	0.7
✓	shadows 1 cutoff	0.18
	shadows 2 color	0.7
✓	shadows 2 cutoff	0.28
	shadows 3 color	0.7
✓	shadows 3 cutoff	0.0
	shadows 4 color	0.7
✓	shadows 4 cutoff	0.0
✓	Use only Red Channel for light...	✓

Figure B.4 ToonShader Material Instance Parameters

7.3 Appendix C: HLSL code



```
return 1;
}
//#include "Random.ush"
//#include "Common.ush"
//#include "PostProcessCommon.ush"
//#include "TonemapCommon.ush"

#include "/Engine/Private/
TonemapCommon.ush"

void aaaaa(){
```

Figure C.1 Texture Colour Correction HLSL Code