**TITLE: Modularity for Next Gen Games Designers**

# AUTHOR: (confidential)

**ADDRESS: (confidential)**

# Abstract

Games require more visuals as virtual worlds become larger and gamers’ expectations soar. Unreal EngineTM (version 4, or UE4) technology was used to demonstrate the merging of visual scripting, textures and materials to create a solution that satisfies modern graphics creation. The product demonstrated in this paper is a Modular Planet System complete with Blueprint, which enables designers and artists to develop fully customisable worlds. The method shows the utilisation of linear interpolation, material parameters, panners, material parameters collection and blueprint to provide a real time in game solution.

Keywords: Blueprint, scripting, designer, UE4, games design, parameters

# 1 Introduction

Modularity, as described by Epic Games (2001), is ‘concerned with making lots of high quality chunks of levels, and reusing those chunks intelligently’. These chunks are often used as part of an overall level design kit such as those mentioned by Burgess and Nate (2013) in their Modular Skyrim demonstration. Level design kits have been an important part of virtual world building for many generations of games, from the work of Perry (2002) to that of Norris (2014).

The techniques that create level design kits can be separated into two main areas: (a) modelling (Perry (2002); Burgess and Nate (2013)); and, (b) textures and materials (Norris (2014) and Klevestav (2010)). Researchers such as Brugess and Nate (2013) have highlighted the requirement for a large number of models. One issue with kit models is repetition; artists and designers have to create ways to avoid games from becoming boring due to the same assets being used frequently, as shown by Epic Games (2001). The problem of repetition is not exclusive to modelling - texturing can have similar difficulties, as discussed by Hajba (2001).

Modern workflows such as layered materials (Paschall, 2014), decals (Epic Games, 2014), vertex painting (Kaminer, 2014), and offsetting (Albeluhn, 2010) all work together to combat repetition within kits. Most, if not all, of these workflows apply to material and texturing techniques. This is not to say that modelling does not play a role in creating variety in level design kits. Perry (2002), and Burgess and Nate (2013) showed that modelling has its place, but in terms of new methodologies, it has not evolved quite so dramatically.

Regardless of technique, the level design kit is set up eventually in a game engine such as UnrealTM or UnityTM. A programmer will often seek to implement the kits through an algorithm to set up the geometry and textures (Gamasutra, 2012). The implementations of these algorithms are becoming more available to designers through visual scripting, particularly with Epic Games’ Unreal Engine Blueprint Technology (2014), and Hutonggames’s Playmaker for Unity (2011).

Visual scripting is a great aid to modularity, as it allows for iterative gameplay and visual changes without requiring any traditional game code (Epic Games, 2014). Advances like this are particularly advantageous to modularity as a developer can preview model alignment and alter in-game materials quickly. Integrating visual scripting into a modular kit may become common-place, as demonstrated in Choo’s (2014) Procedural Apartment.

The method set out in this paper uses visual scripting, and texture and material modification to create a modular planet system.

## 2 Method

The Modular Planet System was created with the following design goals in mind:

1. a system to control landmass;
2. a system to control vegetation;
3. a system to control oceans;
4. a system to control clouds; and,
5. a system to control poles.

Figure 1 shows the textures that were created at the beginning. The system uses a colour map for the different parts of terrain, and then applies several masks to mix textures (Ahearn, 2009).

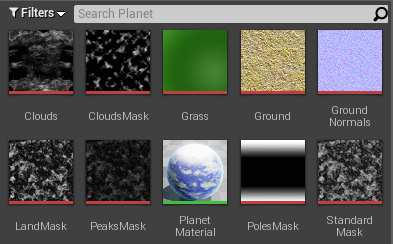


Figure 1. UE4 starting textures

To blend the surfaces and give each surface its colouration, a linear interpolate node was used inside Unreal’s material editor. The Linear Interpolate node blends colour inputs based on the texture masks. To help add additional modularity, the masks can be strengthened and inverted inside the material (Figure 2). Linear Interpolation blends two inputs / textures using values between 0 (black) and 1 (white) values. This technique works in a similar to displacement mapping which alters vertex positions rather than colour information (Wilson, 2013), an advantage of linear interpolation however is that it is possible to generate both colour and height information from the same set of mask / textures. Other interpolation possibilities in Unreal Engine include the Image Adjustment material nodes such as a 3 Colour Blend. This blending method allows for an additional input colour and works the same as linear interpolate however these extra colours can fill up the Blueprint UI.

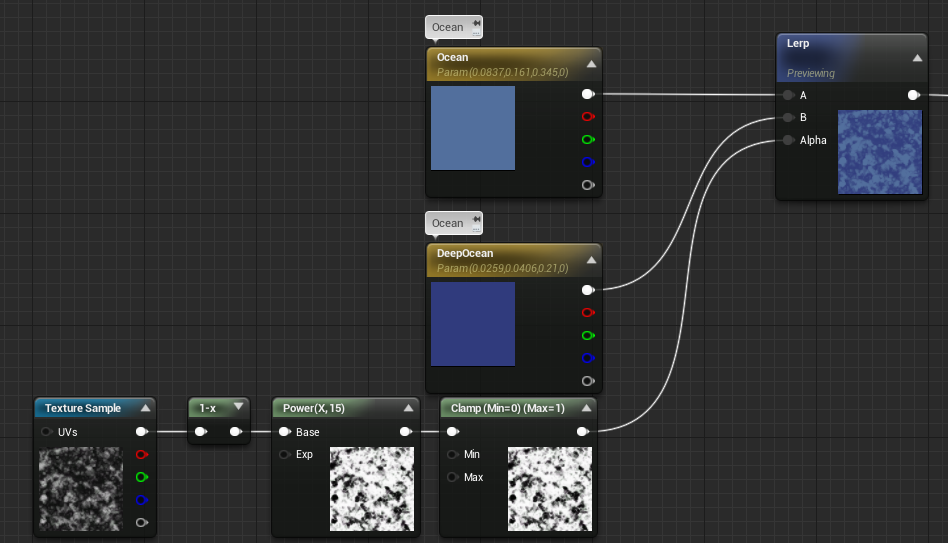


Figure 2. Linear interpolate

Materials inside the UE4 can feature editable parameters, as shown by (Kaminer,2014) (Figure 3). In the Modular Planet System, material parameters are used to control colour and, in some cases, to alter strengths of various blends. It is this feature that will be passed onto designers later in the system.

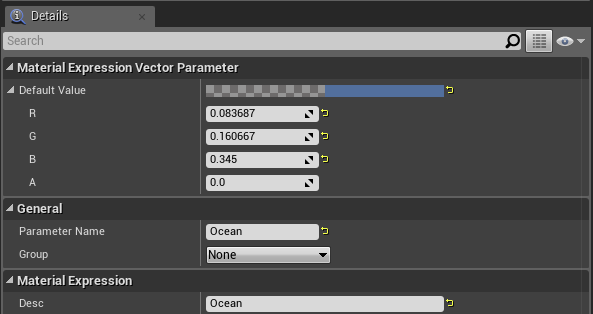


Figure 3. Material parameters

Adding colour into the system can be done in two ways: via the texture method (explored above), or by interpolating between a texture and a parameter. The second approach allows the artist or designer to tint a colour with a texture (Figure 4). This was used in the Modular Planet System to add in more customisation, though it can be tricky to avoid the pitfall of the aesthetics looking too monochromatic. Alternatives to colour parameterisation do exist such as gradient mapping demonstrated by (Grimes, 2010) where by a greyscale image is colourised by a series of gradients, this method whilst produces great results requires an additional gradient ramp / texture.

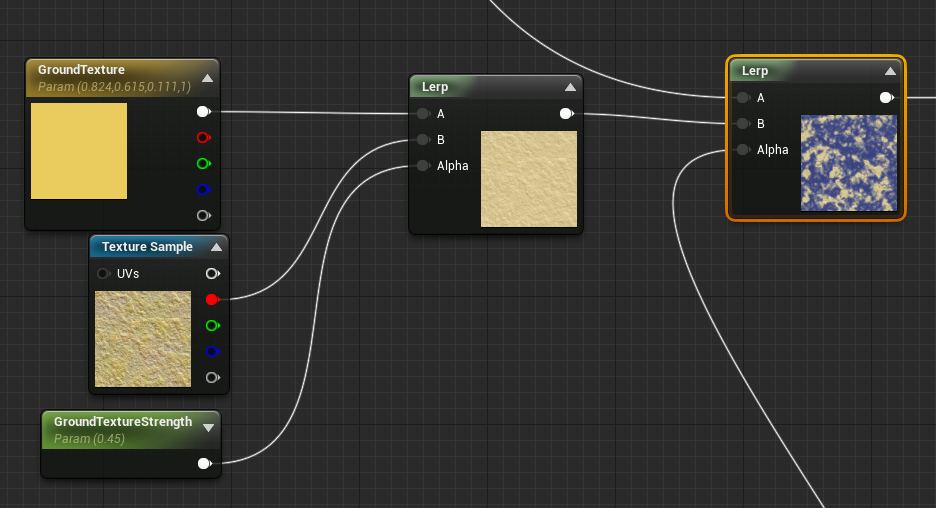


Figure 4. Tint textures

Switches were also added into the material (Figure 5); by using a Boolean true / false system, they allow for the toggling of various surfaces such as grass and oceans. The advantage of using switches is that they prevent the creation of multiple materials and can make the UI less cluttered / more user friendly UI Epic Games (2015).

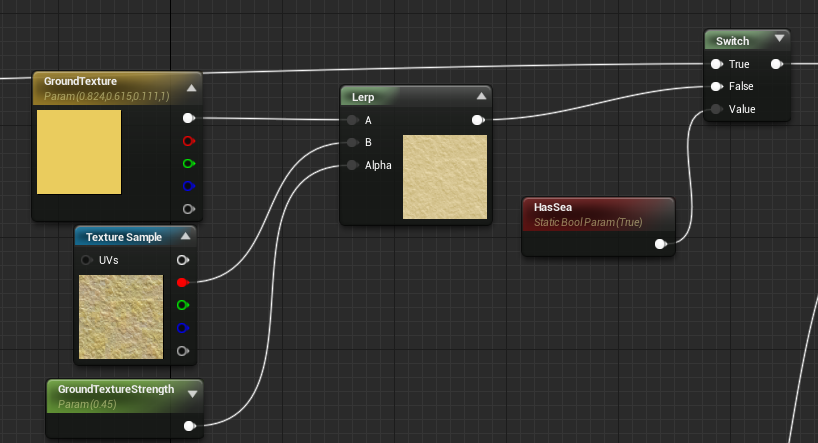


Figure 5. Material switches

To add in polar ice caps, a texture mask was interpolated with the colour white. To allow for further variation, the ground masks were subtracted away (Figure 6) from the polar mask. This approach allowed some of the landmass to break up and affect the polar areas (Figure 7).

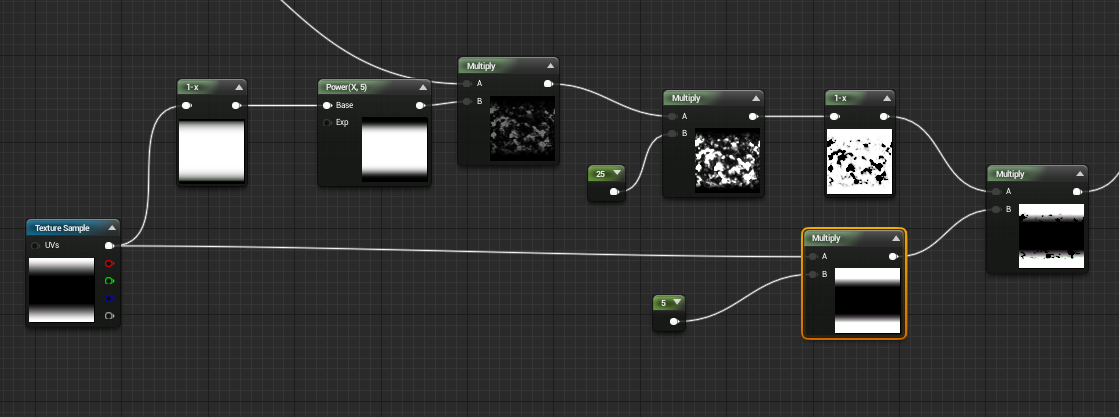


Figure 6. Masking masks

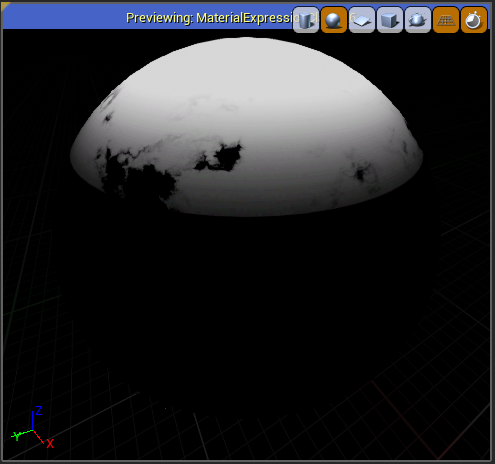
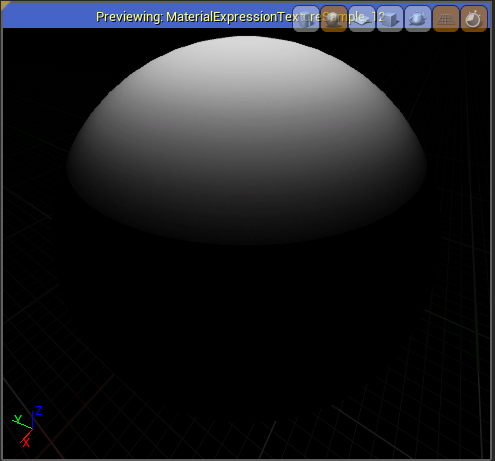


Figure 7. Polar mask before and after modification

After the general appearance was created, additional layers were added for moving elements using panners (Albeluhn, 2010). The first of these were cloud layers, which were created by panning several textures over one another (the results can be seen in Figure 9). This type of effect could also be created by using Sprite Sheets shown by (Antoine & Parrish, 2013) or by plugins such as TrueskyTM. These alternatives produce greater visual fidelity such a particle self-shadowing, complex cloud animation and interaction with lighting systems. However these methods require more packages to produce their assets and in the case of plugins often impact negatively on engine performance. The panner solution is advantageous on two fronts; firstly, it matches Epic Games (2014) own implementation in their default map that ships with Unreal EngineTM and advanced graphical benefits of other systems are not noticeable as the camera position is far from the planets.

Figure 8. Material panning

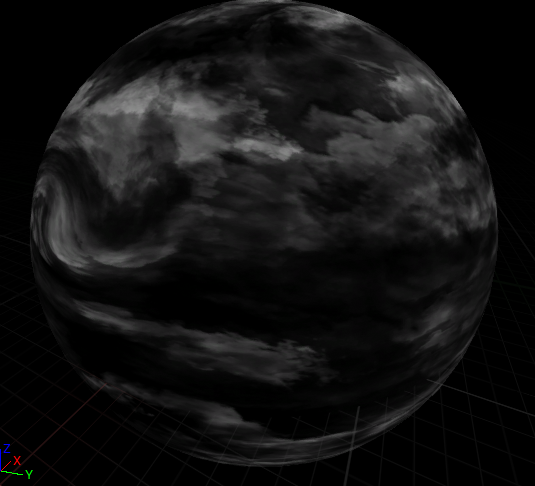


Figure 9. Cloud panning preview

A coloured Fresnel was parameterised to complete the materials atmosphere (Figures 10, 11). Fresnel creates an outline which can be colourised colour and altered in strength. (Baxter, 2015) takes this further by adding additional material mathematics to create the effect of the atmosphere fading into space. This method was discarded in the Modular Planet System for reasons of complexity, as many planets were rendered instead of Baxter’s single example.

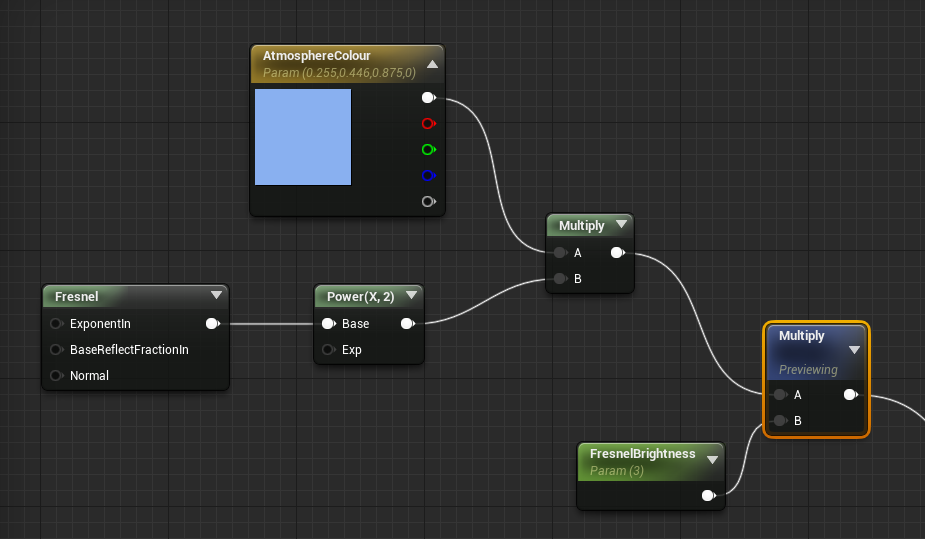


Figure 10. Fresnel node

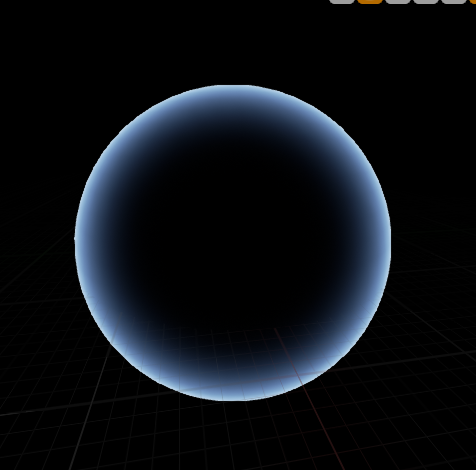


Figure 11. Fresnel preview

In UE4, a material can instance a parent / master material (Figure 12). By using the complete material, the parameters setup can be exposed (Kaminer, 2014). Figures 13 and 14 show the customisation of the planet using colour and vector parameters.

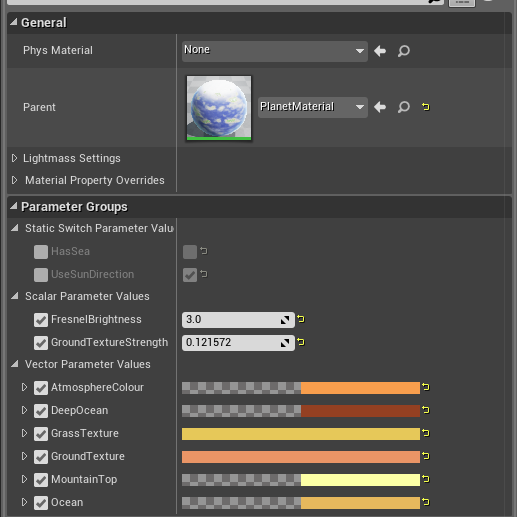


Figure 12. Material instances

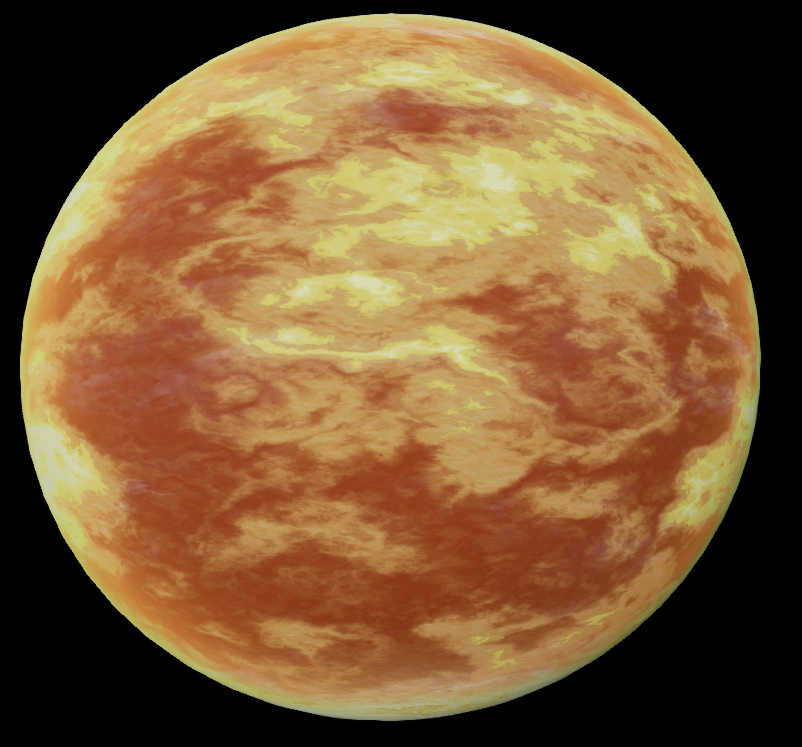


Figure 13. Material instance preview

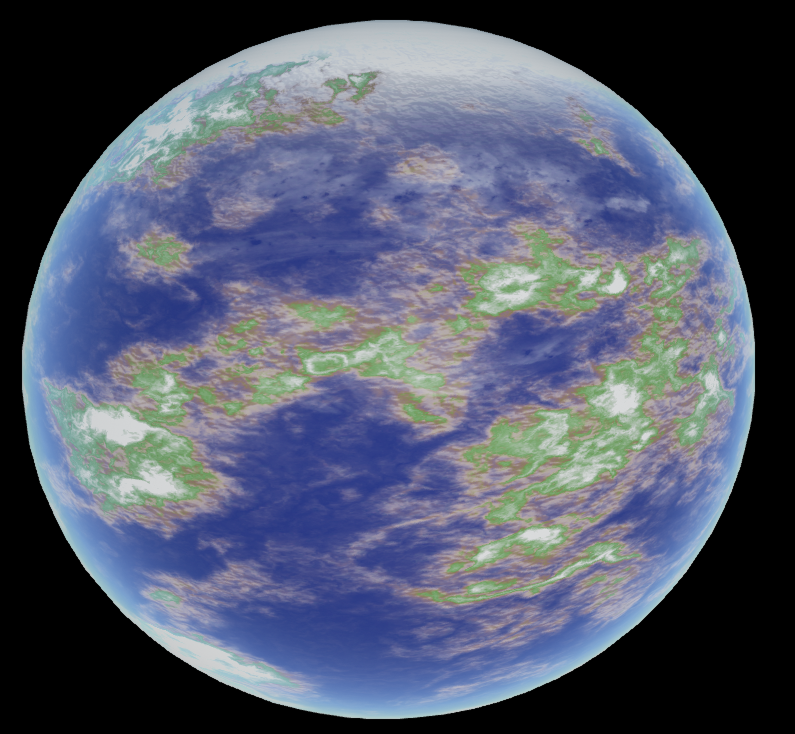


Figure 14. Master material

(Baxter, 2015) and (Caudle, 2014) use Material Parameter Collections (MPCs) as a way of altering materials through visual scripting. In the Modular Planet System, a material and MPC rollout are set up so that the parameter names and colours match the material with the MPC (Figure 15).

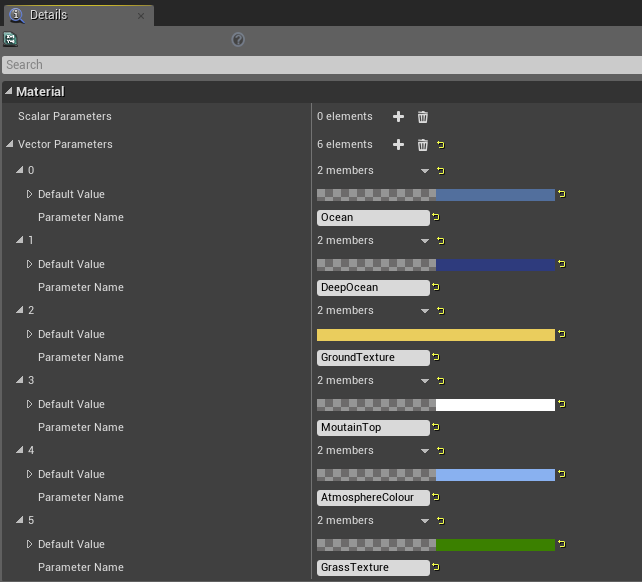


Figure 15. Material parameter collection

To create the placeable planet, a Blueprint actor was created with a sphere component that visually represents the planet (Epic Games, 2014). This actor could also be created with C++ however Blueprint was used instead in order to allow for rapid graphics generation and manipulation by designers.

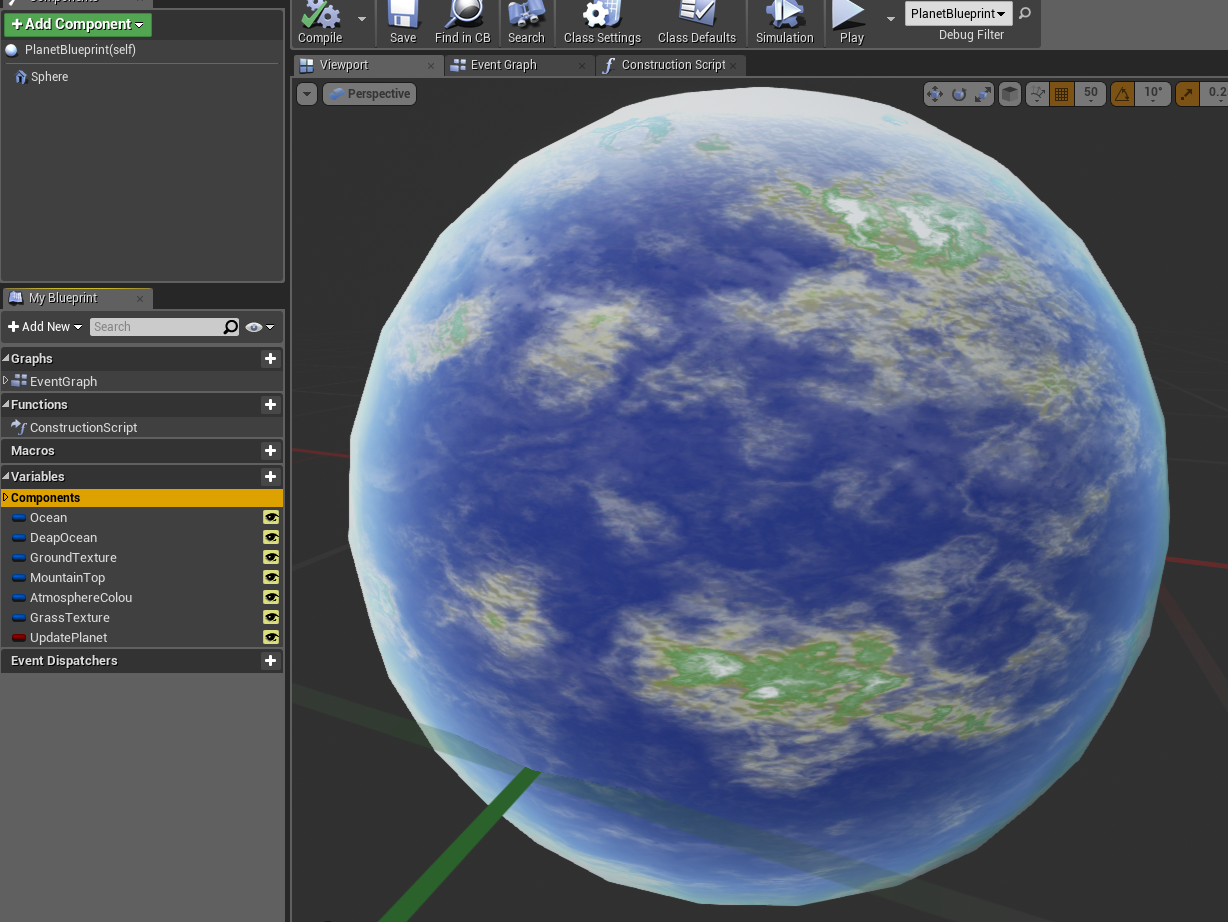


Figure 16. Blueprint component

Inside the Blueprint construction script, nodes were added to update MPC with parameters that were set up inside the blueprint. The parameters set up inside the blueprint matched the names and default colours of the MPC, as shown in Figures 17 and 18. Construction script modification allows a designer / artist to see instant visual feedback when the planets are placed into a level, making it very quick as you block out planet colours and styles.

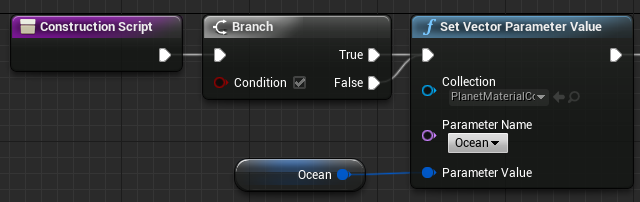


Figure 17. Updating a material parameter collection

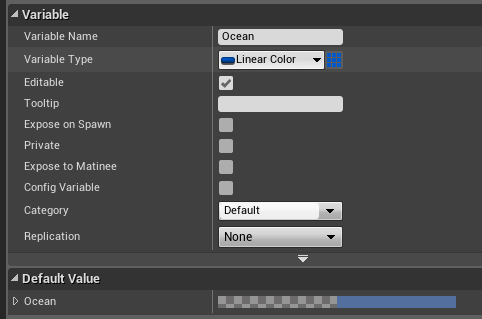


Figure 18. Creating a colour parameter in Blueprint

The construction script was completed so that all the parameters were updated and compiled. To test the blueprint, it was placed into an environment. Figure 19 shows the customisable colours, and Figure 20 shows the customised planet. One of the current lacks in Blueprint is a styled UI system; a designer can separate the parameters by group but this is the current limit. A more advantageous UI to designers would indicate where colours and textures were being used to aid secondary designers not originally involved in the setup.

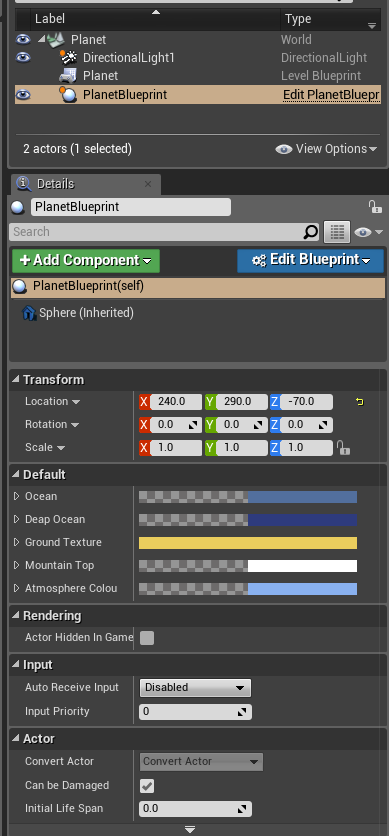


Figure 19 Blueprint actors in a level

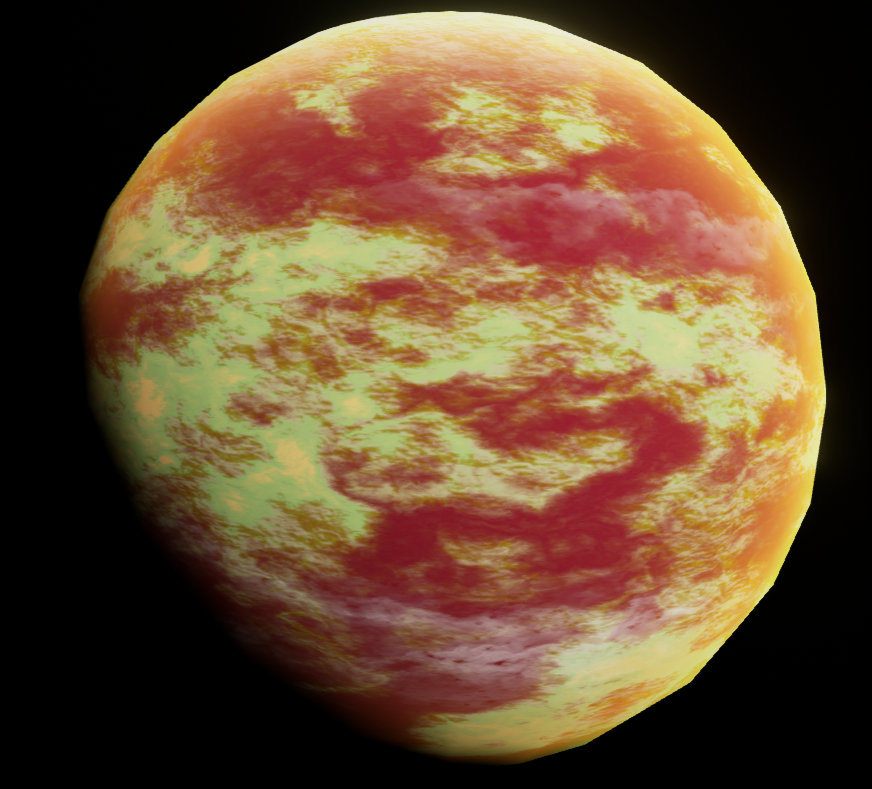


Figure 20. Planet after customisation

The implementation is completed at this stage, the planet can be altered in-game using the completed blueprint that communicates with the completed material.

## 4. Conclusion

The Modular Planet System demonstrates that it is possible to combine visual scripting, textures and materials. The result enables designers to place planets into an environment, and customise their appearance easily. Besides being operational, the system also employs many techniques including parameterisation, panning, and blending. With these modern techniques it is possible to create a variety of unique planets and avoid the repetitious nature of modularity.

The material parameter collection used in the system allows the designer to customise the planet without having to tinker with Blueprint or with materials. One limitation is that if the MPC is shared between objects it will override other planets, making them share the same appearance. A suggested remedy for this is to alter the construction script and make use of material instances (Figure 21), allowing for variety amongst many planets by assigning different parameter values.

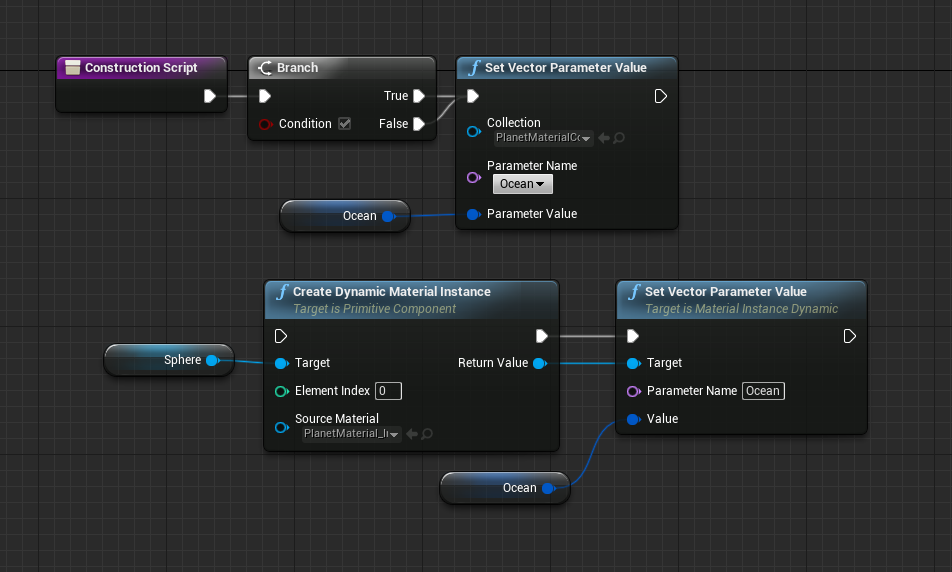


Figure 21 Material instances in construction script

An issue here is that the Blueprint vector parameter editing does not have a drop-down parameter name option, so the blueprint scripter would need to incorporate all of the parameter names in the material. Because of this, the material parameter collection works well for a singular planet system, but larger systems might benefit from a material instances approach instead. A designer could easily swap between the two systems using a Boolean function, as shown in Figures 22 and 23.

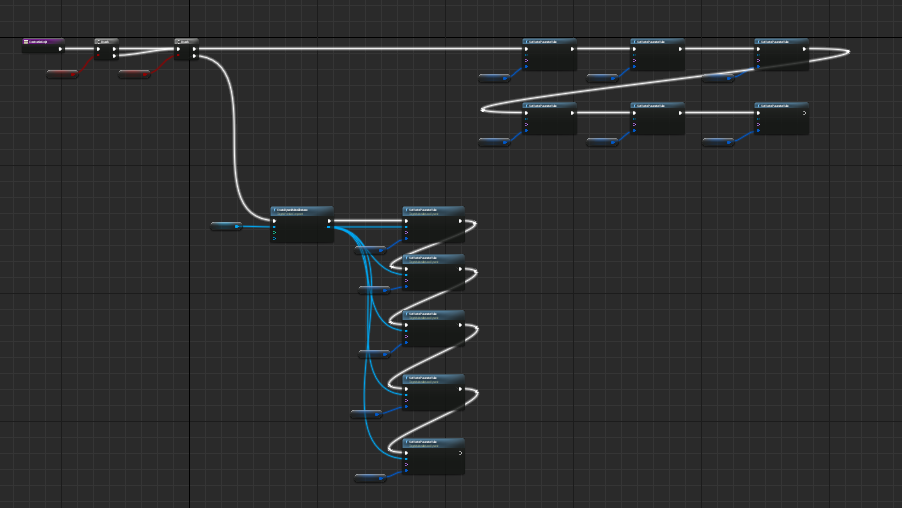


Figure 22 Toggling MPCs and material instances

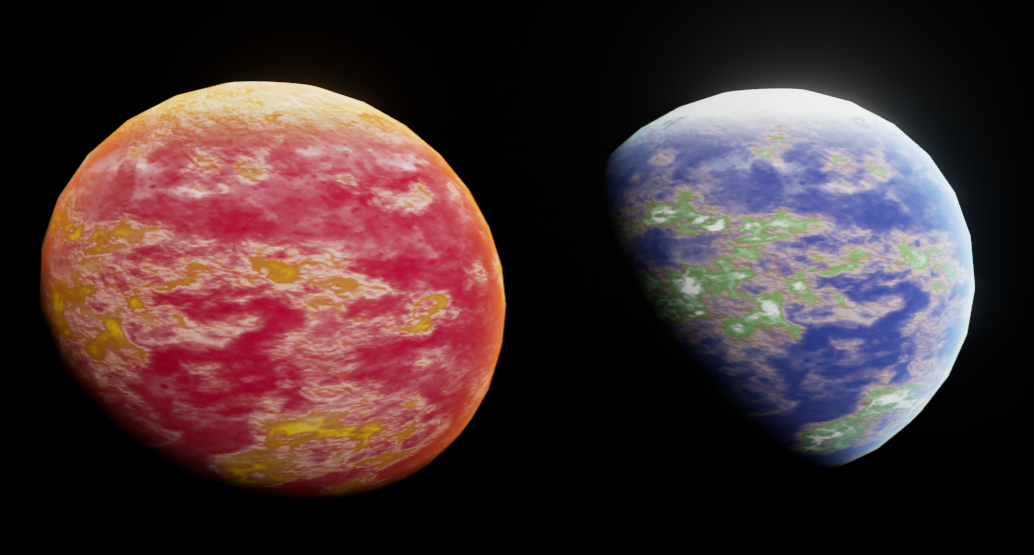


Figure 23 System results

Using procedural texture generation would help in creating further varieties of planets as the system currently relies on textures painted in 2D packages (see Figure 1). Substance Designer is an example of a package that can be integrated into game engines to produce the masks that blend the various terrains. A final benefit of this approach is that disk space would be saved due to the procedural textures (Allegorithmic, 2014).

There are numerous graphical effects that could be added to the system to increase its complexity. (Baxter , 2015) shows that atmospheres can be better replicated with material nodes other than Fresnel. His work could be integrated with switches so that the atmosphere could be toggled between an advanced setup and the current implementation allowing for speed and quality toggles.

The Modular Planet System generates its colour and landmasses from initial textures / masks. This is useful as it gives the artist a starting point however it does require texture creation. (Baldwin, 2015) shows some textures can be done without through TextureCoordinate manipulation and maths. A future addition to this system could be to utilise some material node maths to create swirls / gas type planets.

Another visual area of development could be an animated ocean, the current ocean is static and created by greyscale masks. An animated system could make use of flow maps which would allow texture distortion over time (Linquist, 2014) to create the illusion of moving water. Flow Maps distortion would negatively impact FPS when rendered on a large scale, these systems would need to be dependent on adaptive camera position to create the best visual result without harming a games performance.

On a final note, since the Modular Planet System is exposed through Blueprint, it is also possible to alter the planets’ physics and mesh. Designers could alter their gravity, scale, rotation rates, and other such parameters. Thus there is the possibility of combining graphical and game mechanic designs through one system.

Full Unreal Download Package Available at

YouTube Demonstration Video Available at

Unreal Engine forum Thread Available at

## References

Allegorithmic. 2014. Substance in UE4 [ONLINE] Available at: <https://www.allegorithmic.com/substance-ue4>.[Accessed 16 April 15].

Ahearn, L, 2009. Create Professional Game Art Using Photoshop. 3rd ed. UK: Focal Press.

Albeluhn, C. 2010. Mesh Position Offset the Materials Texture [ONLINE] Available at: <http://www.chrisalbeluhn.com/UDK_Asset_Position_Offsets_Texturet_Tutorial.html>. [Accessed 16 April 15].

Baldwin, J. 2015*.FX-Gas-Planet-Starter-Kit* [ONLINE] Available at: https://forums.unrealengine.com/showthread.php?58809-FX-Gas-Planet-Starter-Kit-vA [Accessed 2 Junel 15].

Baxter, J. 2015. Unreal Engine Demystifying The Dot [ONLINE] Available at: <https://www.youtube.com/watch?v=API4FbyJtDw>. [Accessed 16 April 15].

Bugress, J & Purkeypile, N (2013). Skyrim’s Modular Level Design . In GDC. San Francisco, 2013.

Caudle, S. 2014. Material Parameter Collections. [ONLINE] Available at: <https://www.unrealengine.com/blog/material-parameter-collections>. [Accessed 16 April 15].

Epic Games. 2001 *Workflow and Modularity.* [Online] Available at: <http://udn.epicgames.com/Two/WorkflowAndModularity.html>. [Accessed 16 April 15].

Epic Games. 2014 *Introduction to Blueprint.* [Online] Available at: <https://docs.unrealengine.com/latest/INT/Engine/Blueprints/GettingStarted/index.html>. [Accessed 16 April 15]

Epic Games. 2014 *Basic Decals.* [Online] Available at: <https://docs.unrealengine.com/latest/INT/Resources/ContentExamples/Decals/1_1/index.html>. [Accessed 16 April 15].

Epic Games. 2014. *How to use Texture Masking.* [Online] Available at: <https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/HowTo/Masking/index.html>. [Accessed 16 April 15].

Epic Games. 2015. *Material Function Expressions.* [Online] Available at: https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/ExpressionReference/Functions/index.html [Accessed 16 April 15].

Epic Games. 2015. *How to use Fresnel.* [Online] Available at: https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/HowTo/Fresnel/index.html [Accessed 2 June 15].

Gamasutra. 2012. Procedural Content Generation: Thinking with Modules Available at: <http://www.gamasutra.com/view/feature/174311/procedural_content_generation_.php>. [Accessed 16 April 15].

Grimes. B 2010. *Shading a Bigger, Better Sequel.* [ONLINE] Available at : <http://www.valvesoftware.com/publications/2010/GDC10_ShaderTechniquesL4D2.pdf> [Accessed 2 June 15].

Hajba, P. 2001. The Power of the High Pass Filter. [ONLINE] Available at: <http://www.gamasutra.com/view/feature/131482/the_power_of_the_high_pass_filter.php?print=1>. [Accessed 16 April 15].

HuttonGames. 2011. PlayMaker[DISC].

Jeremy Choo. 2014. Procedural Apartments 2[ONLINE] Available at: <https://www.youtube.com/watch?v=C7oaZ7brp4Q>. [Accessed 16 April 15].

Jeremy Choo. 2014. Procedural Apartments Unreal Marketplace [DISC].

Jeroen Maton.2008. Top Ten Tips of Texturing[ONLINE] Available at: <http://www.cgsociety.org/index.php/CGSFeatures/CGSFeatureSpecial/the_top_ten_tips_of_texturing>. [Accessed 16 April 15].

Kaminer, A. 2014. UE4 Materials for Beginners: Vertex Painting and Water Puddles [ONLINE] Available at: <https://gumroad.com/l/Ecphc>. [Accessed 16 April 15].

Klfake, T. 2011. Creating Modular Environments in UDK. [ONLINE] Available at: <http://www.thiagoklafke.com/modularenvironments.html>. [Accessed 16 April 15].

Klevestav, P. 2010. Working with Modular Sets [ONLINE] Available at: <http://www.philipk.net/tutorials/modular_sets/modular_sets.html>. [Accessed 16 April 15].

Lindquist, J. 2014. Photoshop Generated Flow Maps [ONLINE] https://www.unrealengine.com/blog/photoshop-generated-flow-maps [Accessed 2 June 15].

Paschall, A. 2014. Creating Layered Mateirals[ONLINE] Available at: <https://wiki.unrealengine.com/Creating_Layered_Materials_%28Tutorial%29>. [Accessed 16 April 15].

Perry, L. 2002. Modular Level Design [ONLINE] Available at: <https://udn.epicgames.com/Three/rsrc/Three/ModularLevelDesign/ModularLevelDesign.pdf>. [Accessed 16 April 15].

Norris, J. 2014. Modular Building Set [ONLINE] Available at: <http://www.purepolygons.com//temp_images/building_tut/Building_Breakdown.pdf>. [Accessed 16 April 15].

Shadden, I. 2015. Procedural Room Generation [ONLINE] Available at: <https://www.youtube.com/watch?v=mI7eYXMJ5eI>. [Accessed 16 April 15].

Wanlass, T. 2011. Modular building with UDK [ONLINE] Available at: <http://3dmotive.com/series/modular-building-with-udk.html>. [Accessed 16 April 15].

Wilson. J. 2013. *Direct X 11 Displacement Mapping* [ONLINE] Available at: http://www.marmoset.co/toolbag/learn/displacement 2013 [Accessed 2 June 15]