Anime Toon Shading

Documentation v 1.2



Anime Toon Shading is advanced powerful material that will easily make your game looks like toon. [Showcase/Tutorial video]

Main references to create Anime Toon Shading was **"Gravity Rush"** and **"Dragon Ball FighterZ"** so if you want achieve this kind of feeling witch dynamic toon lighting in your game then this package will be a perfect starter for you.

Anime Toon Shading uses geometry based shading model that is customizable separately for every object on the scene to specific lighting conditions and works perfectly in the forward rendering mode. Package contains full lighting system that supports single dynamic directional light and multiple dynamic point lights. Perfect choice for VR games as well as mobile.

Material is divided on three configurable groups of effects (**shadow**, **specular** and **outline**):

Toon specular parameters:

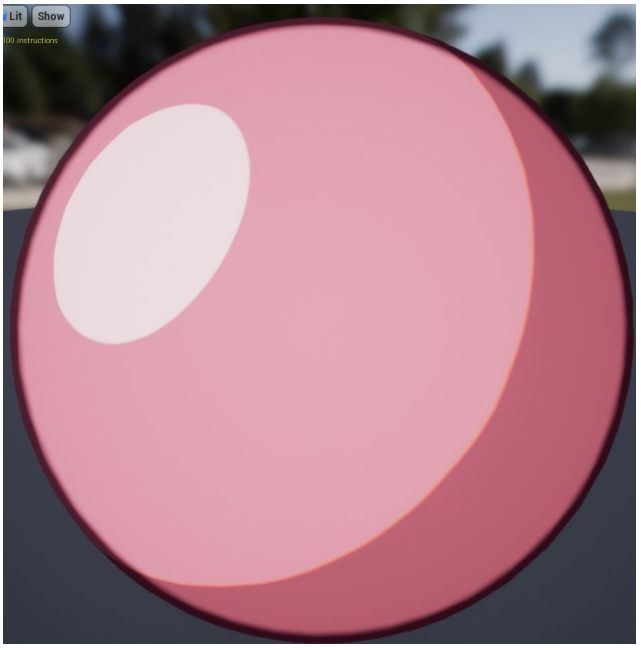
- Specular size
- Color, and translucent
- Softness

Toon shadow parameters:

- Glossy effect
- Aberration effect
- Softness
- Width
- Two shading methods (texture and smoothstep function)
- Multiple shading textures

Toon outline parameters:

- Width
- Softness
- Color and translucency



Features

- Advanced material configuration instanced per object **specular/shading/outline**
- Dynamic **directional light** support
- Multiple dynamic **point lights** that can affect on actors (you can add some flashes and explosions in your game).
- Geometry based shading so can be used without post processing.
- Very fast and GPU friendly.
- Implemented as material function can by used as unlit/lit material.
- Tested on meshes from paragon (great for stylizing assets)

Introduction

There are two types of packages that support toon/cel shading in UE4, **Post-process based (other packages)** and **Geometry based (this package)** Both has cons and pros and It is important to understand the difference.

	Post-process based toon shading (Other packages*)	**Geometry based toon shading (This package)**
Description	uses final image generated by ue4 to translate scene colors on cartoon stylized image.	uses material system and meshes and is applied per object.
Platforms	PC (full post-process in forward rendered scenes is problematic)	All platforms VR/PC/Mobile Just works this is oldschool method.
Scene	Full scene effect in one click.	Need to be applied per object/per material.
Tweaking	Hard to customize. Tweaking parameters for one mesh can destroy effect on other.	Easy to customize. Just open material and modify shading/ outline/colors/specuar.
Usability	Easy to use just plug and play.	A lot of work to configure materials. Little complicated, need setup for meshes.
Characters	Hard to customize single character.	Very functional and useful in setting up character shading.
Environment	Easy to change visual effect of environment on full scene.	Global tweaking can be done by using shared material instance by multiple materials.
Receiving shadows	Supported by default.	Supported only in forward rendering as experimental functionality.
Dynamic lighting	Supported UE4 lights. Looks strange when you use more lights than 1 light.	Not supported.
Specular	Not supported	Supported by default.
Outline	Full scene	Applied per model as shell mesh.
Translucents	Not supported	Supported by default.
Reflections	Not supported	Looks properly in reflections.
Fog	Not supported	Supported by default.

*Probably there will be support for post-process shading model in future versions of this package but in current state I'm focused on geometry based shading.

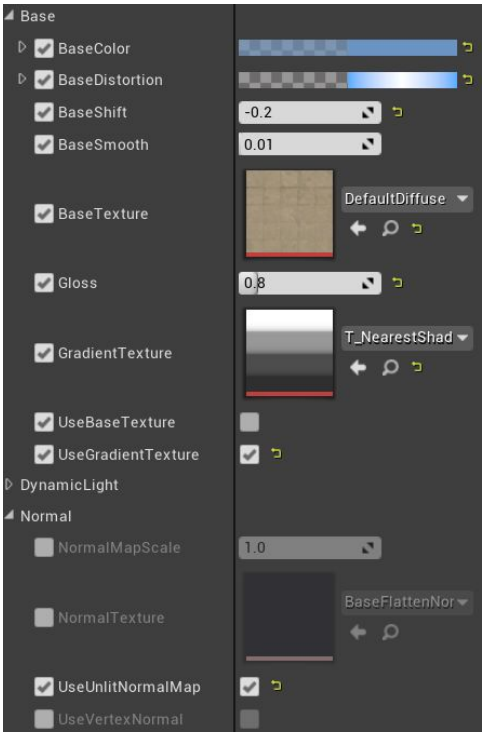
1. Getting Started [Creating base material]

Supper toon shading is divided on two types of material that can be used on geometry, Lit (**M_ToonShadingLit**) and Unlit (**M_ToonShadingUnlit**) . Both types of material supports similar functionalities and parameters. Basic difference:

Unlit: Very fast but can't cast shadows
Lit: Little slower but allow to cast shadows.

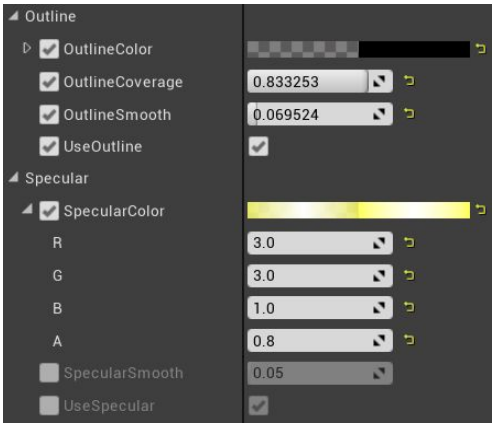
To start working witch *Anime Toon Shading* you need to create material for your mesh:

a. Chose material type that will be used on mesh for example **M_ToonShadingLit**.
b. Click right and select "create new material instance".
c. Open material and setup parameters of new material as you like to see object on scene.
d. Set created material to your mesh.



Material instance parameters:

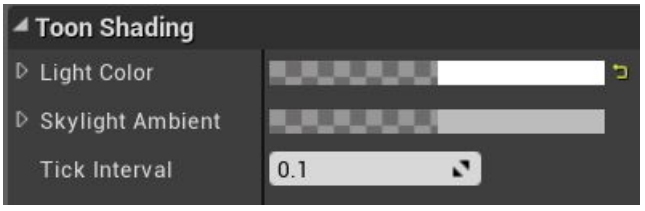
- **Base Color** - color of material.
- **Base Distortion** - distortion of shadow on all RGB channel.
- **Base Shift** - shift of base shadow.
- **Base Texture** (UseBaseTexture)- texture color multiplied by base color.
- **Gloss -** Gloss effect. Changes the direction of shadow to camera.
- **Gradient Texture** (UseGradientTexture)- gradient texture used for shadow. Allow to change stepping mode of shading.
- **Normal Map Scale** (UseUlitNormalmap) - scale normal map vector.
- **Normal Map Texture** - texture used as normal map.
- **Outline Color** - color of outline. Alpha channel contains width.
- **Outline Coverage** - setting translucent outline



- **Outline Smooth** - smooth edge of outline can be used as fresnel.
- **Use Outline** - enable/disable outline effect.
- **Specular Color** - color of specular. Alpha channel used for specular size change.
- **Specular Smooth** - smooth specular effect.
- **Use Specular** - enable/disable specular effect.

2. Using directional light

Directional light is used for setting up global lighting that will be affecting on all objects on scene. Just drag and drop **BP_ToonDirectionalLight** on scene and setup parameters:



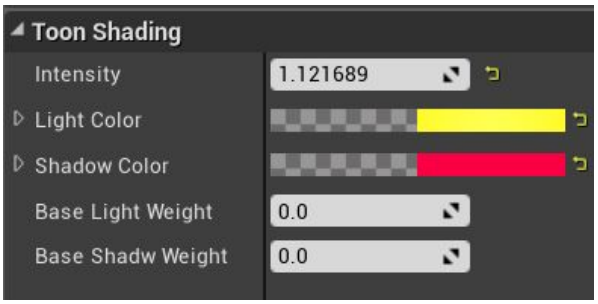
a. **Rotation** - direction of light
b. **Light Color** - direct light color
c. **Skylight ambient** - skylight shadow color.
d. **Tick Interval** - frequency of update the values. If light will be static then set to -1.

3. Using multiple point lights

Anime toon shading supports multiple point lights that are combined into second directional light data and applied onto base shading model. That functionality can be used for light flashes like explosions, some additional lighting on scene and shadowing in darker places where directional light should not affect on object.

System designed for this requires little more effort from user:

1. Drag and drop **BP_ToonPointLight** to your level. Set up parameters:



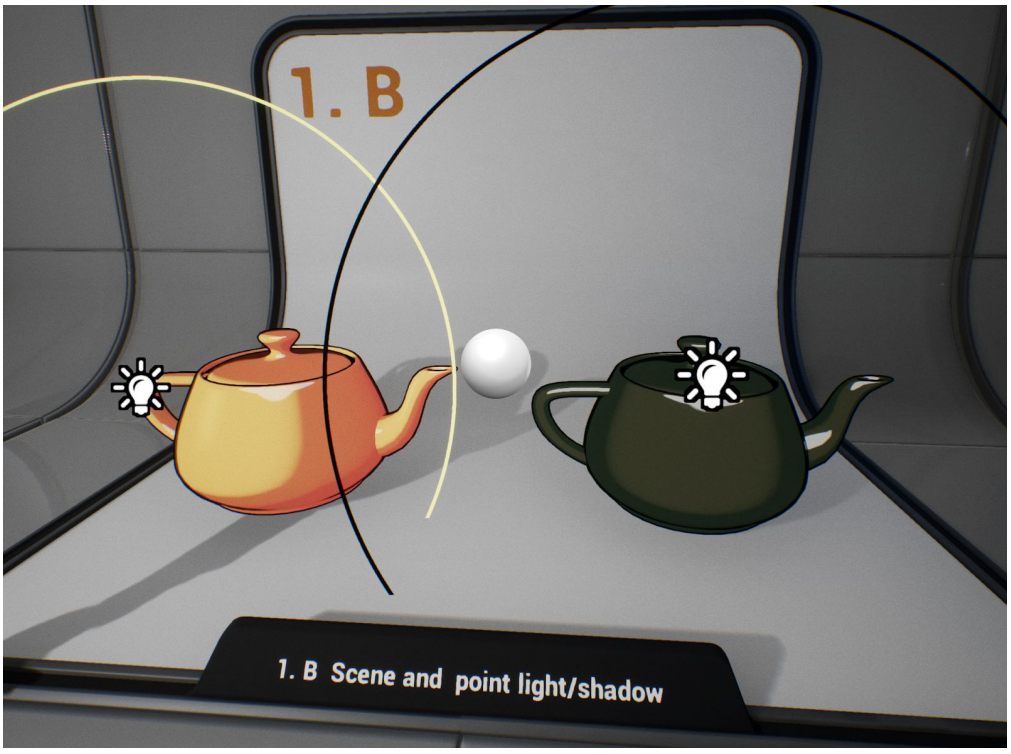
- **Intensity** - light color and shadow multiplier.
- **Light Color** - color of direct lighting.
- **Shadow Color** - color of indirect lighting.
- **Scale (radius)** - radius of lighting.
- **Position** - position of light on scene
- **Base Light Weight** - How base directional light color affects on meshes in this light. 0 means additive, 1 means override the base light.
- **Base Shadow Weight -** How base directional light shadow effects on meshes in this light. 0 means override the base light direction 1 for no change.
- **Scene - DO NOT CHANGE** Will be updated automatically after adding light to the scene.

2. Create new actor that will be affected by light. Dynamic light will only affects on objects that implements properly BI_ToonShading interface. You can use example implementation and inherit base class of static mesh (**BP_ToonStaticMesh**) and skeletal mesh (**BP_ToonSkeletalMesh**)

    - Content Browser -> Crete Actor
    - Select BP_ToonStaticMesh or BP_ToonSkeletalMesh
    - Setup default mesh in blueprint.
    - Set material (from 1 step of documentation) on mesh.
    - Drag and drop created actor to your level.

3. Drag and drop **BP_ToonScene** to your level. Scene will contains references to all point lights and actors that will be affected by this lights.

4. Select **BP_ToonScene** and add **Lights** and **Actors** that should be managed by lighting system.



4. Implementing toon shading interface in custom character (ADVANCED)

Sometimes there is no possibility to inherit example actors (**BP_ToonStaticMesh, BP_ToonSkeletalMesh, ThirdPersonCharacter**) that contains the implementation of dynamic lighting. Fortunately "Anime Toon Shading" is very extendable, you can modify your own character to be affected by dynamic Toon Shading lighting as well. It's a bit complicated but believe me, I did everything I could to make it user-friendly.

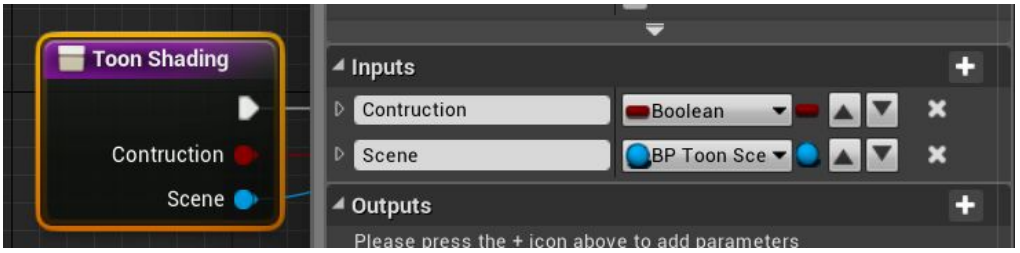
4.1 Open Character blueprint. Add **BP_ToonScene** reference attribute to your actor. Variables->Add:

Variable Name **Scene**

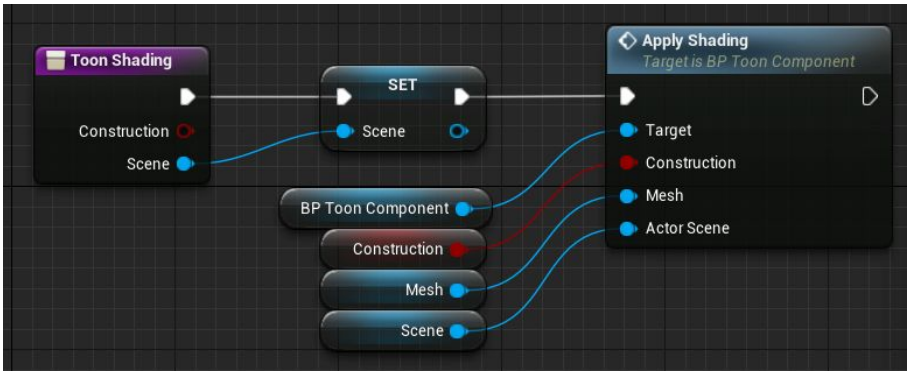
Variable Type **BP_ToonScene**

4.2 Add new component **BP_ToonShadingComponent** that will be used to communicate with scene.

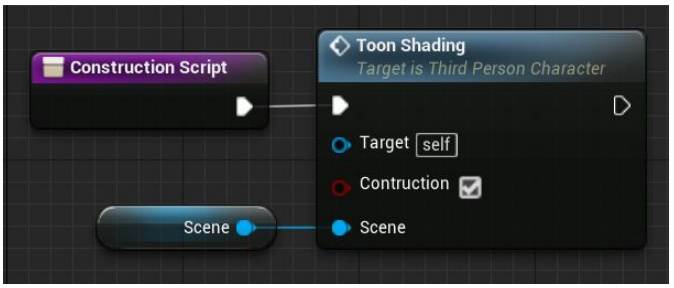
4.3 Add new function Toon Shading that will be evaluated to update lighting on object. Functions->Add.



Add **ApplyShading** evaluation for every mesh in pawn that should be affected by lighting. In this case Character contains one **Mesh**.

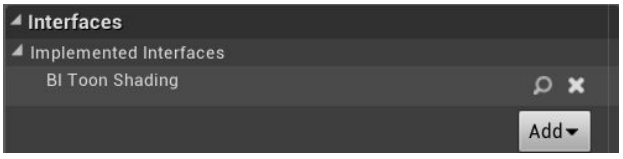


4.4 Call function in constructor to activate lighting on start.

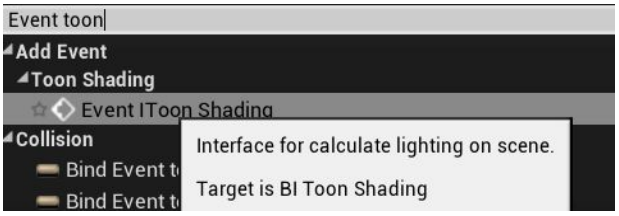


4.5 Implement ToonShading interface. Open Class Settings And add interface BI Toon Shading to list.





4.6 Call toon shading update on events **BeginPlay** and **IToonShading**





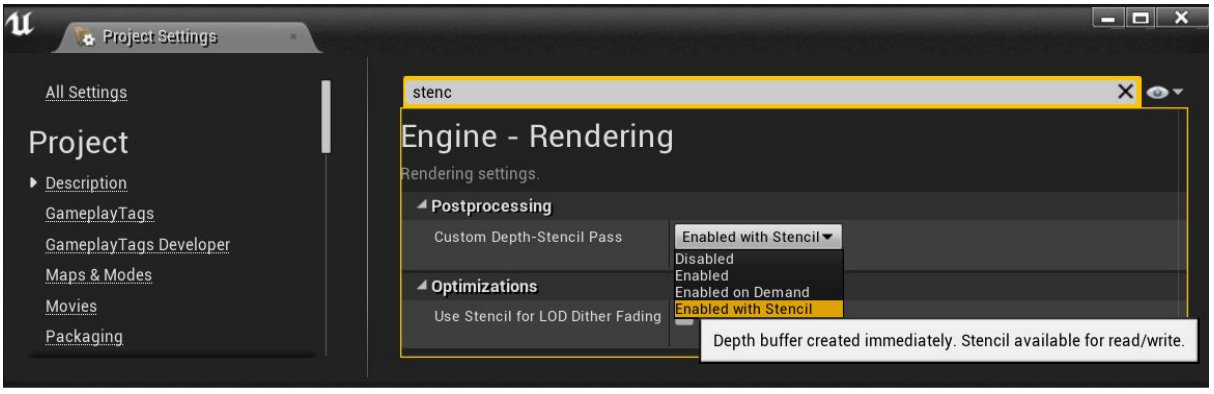
Remember! This modification creates dynamic material instances to communicate with scene manager, you should use them instead of creating new one.

5. Anime Toon Shading and external post-process effects

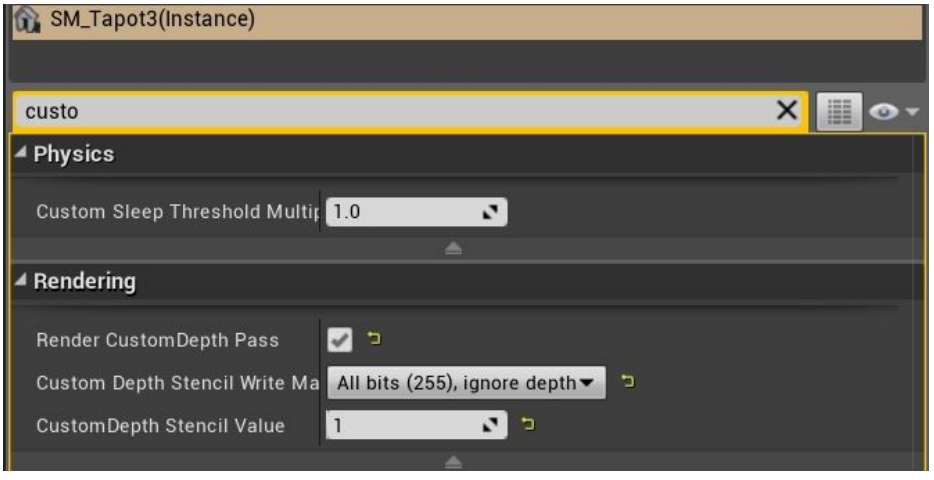
It's good practice to combine two types of toon shading system for example by using “Anime Toon Shading” for characters and post-process material for the environment.

Post-process based toon shading should support excluding the shading materials that using "Anime Toon Shading". Generally in this kind of problems full screen mask that tells to post-process shader where to live pixel not changed. It can be implemented easily by using custom depth stencil mask. This is simple example how to integrate this package with post-process material.

5.1 Set up your project to use stencil buffer. This functionality is required because we need additional buffer to mark pixels rendered by AnimToonShader.

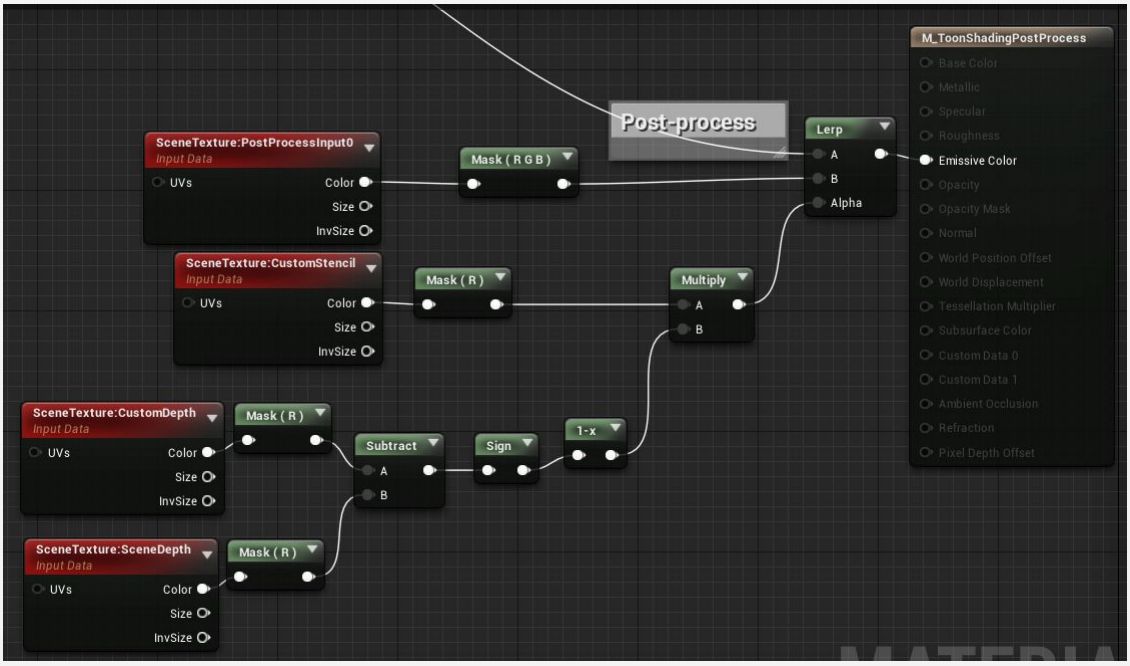


5.2 Select mesh that is using “Anime Toon Shading” material and set stencil mask value:



Remember: It is just an example configuration of mask. If you already using custom depth stencil for outlines then you should reserve one bit for “Anime Toon Shading” masking.

5.3 Use MF_PostProcessMask material function (available since version 1.3) or modify your post process material to mask the post-process effect:

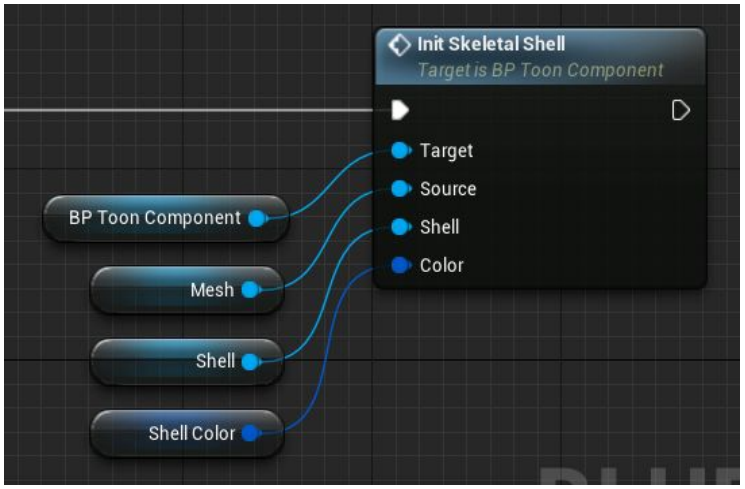


6. Outline shells

Outline effect for forward rendering can be realised by creating second expanded mesh with front face culling (rendered from inside). “Anime Toon Shading” contains tricky material that can do it for you. You can find example for this part of documentation in **Demo/Characters/ThirdPersonCharacter** blueprint.

a. Duplicate mesh component in your character and rename to Shell.

b. Initialize Shell in construction script



c. You can add attribute **ShellColor** to your blueprint or use MakeVector. Alpha value in this vector is responsible for outline width.
d. Additional shells supports vertex color alpha multiplier. You can locally control the width of shell by painting (alpha channel) on vertex color of mesh. It is cool feature used in the current generation games.



7. Frequently Asked Questions

This shader works with post processing?
This implementation is geometry based. Material function can be injected in any material you want to add functionality of this shader. For full scene toon shading you should use another package.

Do the materials self-shadow?
Nope self shadowing is not available because this method uses material emissive color. However you can add dynamic light that will absorb directional lighting to simulate walking into shadow. Receiving shadows is not supported.

Does it work on mobile and VR? Forward rendering is supported?
Yes it should work fine on this platforms.

Dynamic lights? how it works in forward and deferred?
It is based on observation that toon shading don't need to realistic lighting model.
"Anime Toon Shading" material supports two directional lights. First one is global directional light shared by all objects on scene. Second one is dynamic light calculated per mesh as sum of closest point lights. The tricky part is to combine this two lights in material to not destroy the shading on mesh :)

8. References

GuiltyGearXrd's Art Style : The X Factor Between 2D and 3D