Cyberpunk City Pack

Product Link: https://www.unrealengine.com/marketplace/cyberpunk-city-pack

Demo Build: https://1drv.ms/u/s!ApUq5n3P_zoflvUL485B6cWF9ltpBQ?e=16sQhc

Demonstration Video: https://youtu.be/WIo7hbXLqQM

Showcase Video: https://youtu.be/K6jUu6GAgVc

Systems showcase: https://www.artstation.com/artwork/nY6VkE

Release Notes

1.1

- Refactored building construct system to have more coherence results
- Random scattering is now using custom logic with seed numbers for consistency
- Wetness material function is now taking metallic channel into account
- Added driveable car asset, blueprint, customization and basic logic
- Added child spline class for adding parked cars
- Added detailed car master materials
- Added manhole cover assets and city ground actor is procedurally placing them by default
- Added possession logic for BP_TestPlayer and simple UI to showcase interaction information
- Fixed parent spline actor HISM error
- Product documentation is now examted to cover vehicle systems

1.2

- Truck added
- Sportscar added
- Compact car overhaul
- Shipping containers added
- Shipping containers blueprint added
- Wetness layer static water drops option added
- Static vehicle blueprint supports all car types
- Wetness layer metallic issue solved
- Spline HISM now includes collision option
- Spline HISM now includes constraint debris mode
- Vehicle materials overhauls
- Vehicle camera is now supporting collision
- Vehicle sounds tweaked
- Vehicle handling tweaked
- Example map re-organized

1.2.2

- BP_P_Building is now handling mesh bounds precision better to avoid gaps
- Seed numbers are now taking actor positions into account but seed numbers can still be changed and results will be consistent
- Vehicles are now supporting doors that can be opened/closed

- Vehicle animation logic is now using an interface for transferring steering and door values between animBP and vehicle actor
- Vehicle pawns are now using separate wheel meshes for future addons in mind
- More details to truck model

1.2.3

- Building system now supports fully static mesh options (UseStaticMeshVersion)
- Other blueprint systems now supports fully static mesh options (UseStaticMeshVersion)
- Distant buildings are now using pseudo random functions, lock option removed
- Random decal material sorting issue fixed
- Window material now contains a switch to disable environment controller effect
- Null warnings are now solved that were caused by some blueprints

1.3 (Unreal Engine 5+)

- Meshes that can work with Nanite system are converted to Nanite meshes
- Blueprints are converted and tested to work well in UE5
- "Add Instances" nodes changed to UE5 versions
- PhysX vehicles are turned into Chaos vehicles
- Example map level lighting tweaked to work better with Lumen system
- Building system V2 added that supports interior generation (coming soon)

1.3 (Unreal Engine 4.26 - 4.27)

- BP_ParentBuilding system is now fully updated to V2
- Added holo blueprint that randomly generate holo ads
- Added more advertisement logos
- Fixed PhysX vehicle random shaking when colliding with the ground

1.4 (Unreal Engine 5.1+)

- Building system refactored/optimized
- Building system bug fixes
- Spline system refactored/optimized
- Spline system bug fixes
- Example player overhaul
- Drivable vehicle system overhaul
- All of the inputs are now using Enhanced Inputs
- Some mesh caps fixed
- Example level is now using World Partition
- Example level lighting overhaul
- Added rock cliffs
- Added better cloud material
- Added virtual texturing support
- All of the textures compression settings retweaked

Feature requests & Upcoming features

-

Make sure you are not manually copying files with file explorer but instead use the migrate workflow. More information here. This way you will avoid possible issues that might occur if you move files outside the Unreal Engine.

If the editor asks to import files, choose “Don´t Import”. This is because the pack contains source files for those who like to edit them and importing them will reset some settings and cause issues.

This pack will not work with static lighting because blueprint assets are creating instanced mesh components that are not supporting baked lightmaps.

F.A.Q

Q. Is this pack going to work with static lighting?

Short answer is no. Because this pack is very dynamic and optimized it's not possible to bake lighting for instanced meshes. Also it's not optimal to use static lighting with large environments.

Q. How big is the demo/example level?

Example level is around 3 km x 3 km in size.

Q. Why are rocks covered with sand?

Rock material is using distance fields by default for blending rock meshes better with landscape. In order to use this and avoid fully sand covered rocks it is important to turn off “Affect Distance Field Lighting” from the mesh “Lighting” tab. If you don't want to use this effect at all then you can turn it off from material instances.

Q. Level is missing shadows and what I need to enable in order to get all of the features working?

This package requires that you have the “Generate Distance Fields” setting enabled in your project settings.

Q. I'm seeing greenish cards on top of surfaces. What are those?

You need to enable the DBuffer Decals option to use d buffered decal materials. You can find this setting in *Project Settings -> Engine -> Rendering -> Lighting*.

Q. I can’t move when I press play?

This can happen if you don’t have correct input settings specified in your project settings. Look for the “*Example Player*” section to see what action and axis mappings you need to set up

Q. Vehicle is shaking when I'm driving?

Vehicle is using the PhysX vehicle movement class. If your performance drops below a certain threshold it will start to affect physics and therefore the vehicle handling. To fix this you might need to optimize your project performance more. If you need to optimize this pack then read the "*Optimization*" section.

Q. I'm getting errors in Unreal Engine 5 and can't drive vehicles. How to fix that?

Drivable vehicles are using the Unreal Engine Chaos vehicle system. You need to enable **ChaosVehiclesPlugin** (Edit -> Plugins) and then restart the editor. After that you should be able to use these vehicles. Make sure to set up your input settings (Edit -> Project Settings) so you can also drive.

Q. How can I trigger the train blueprint via Sequencer?

By default, BP_Train_Track blueprint is using *Event BeginPlay* to trigger the *TrainMovement* timeline node. You want to disable that and instead create a custom event that you can trigger with *Sequencer*. For example, an event called **SequencerMoveTrain**.

After that you can open *Sequencer* and press the green "+*Track*" button and select "*Actor To Sequencer*" and choose BP_Train_Track. Then select that BP_Train_Track and press the plus icon, *Event* and choose *Trigger*. You can then create a keyframe and then right click over that keyframe, choose *Properties*, click "*Unbound*" and then "*Quick Bound*" and find that custom event you created inside BP_Train_Track.

You can read more about this in the Unreal documentation.

Q. Opening the demo level takes a long time. What is causing that?

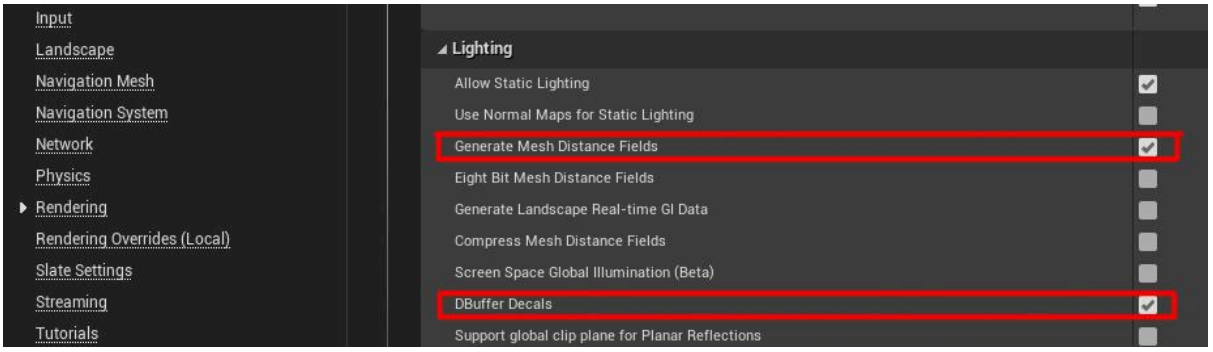
At the moment it's still unknown why UE5 versions of marketplace packs take so long to open but they will open eventually. I will update things when I'm sure what is causing this or if Epic Games change something on their end.

Project Setup

Required Settings

In order to avoid issues and use all of the features this pack can offer you should enable the following settings.

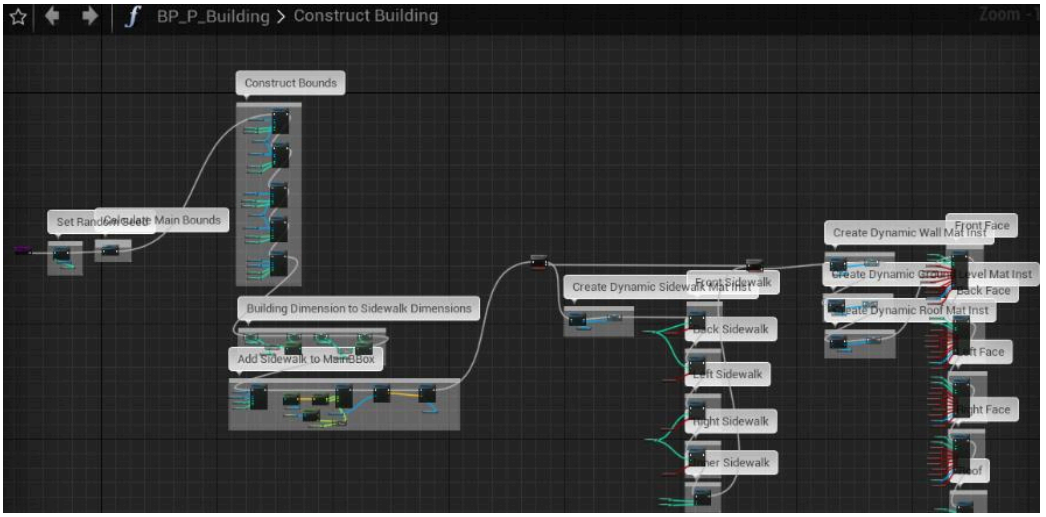
Example level is using distance field shadows and ao. In order to use these lighting features you need to enable the "Generate Distance Fields" setting. Some materials need the "DBuffer Decals" setting to be true too. Both of these settings can be found in Project Settings -> Rendering.



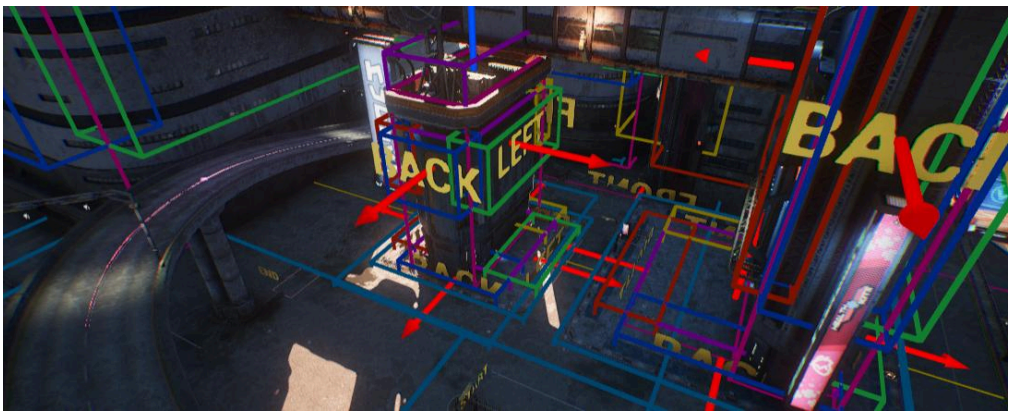
Building Blueprint System

Overview

This system is constructing buildings using different building sets. These sets contain building meshes that are modeled in a modular fashion to work well together. Currently there are 6 different styled building blueprints with over 80+ variables to change. This way you can control building size, colors, damage, windows/roofs/doors etc… Blueprint will then use HISM (*Hierarchical Instanced Static Mesh*) components to construct the actual building in an optimized way.



Main logic is in the BP_P_Building parent class. Every building is then inheriting that logic from this class. Some common variables are public so you can tweak them in the level and create more variations between buildings. Others are private that you need to change inside that child blueprint. Most of the time you don't need to do that.



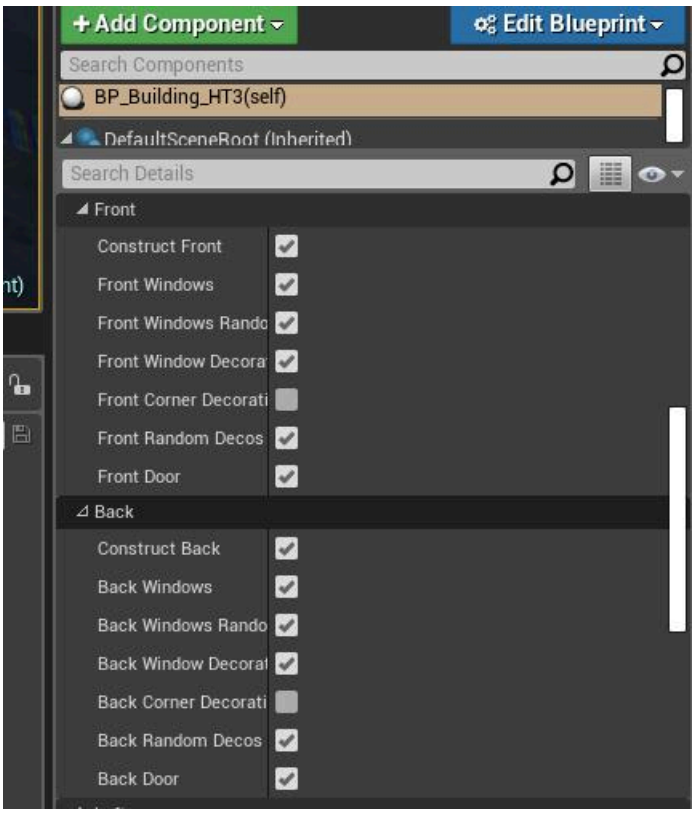
Main Control

System is divided into different parts. First stage is the planning stage. In this part the system will calculate building dimensions and generate bounds that are later used for the actual building. You can find a variable called “Use Planning Mode” under “Control”. Turning that on will make it faster to work and edit buildings but you want to change that to false when you are done.

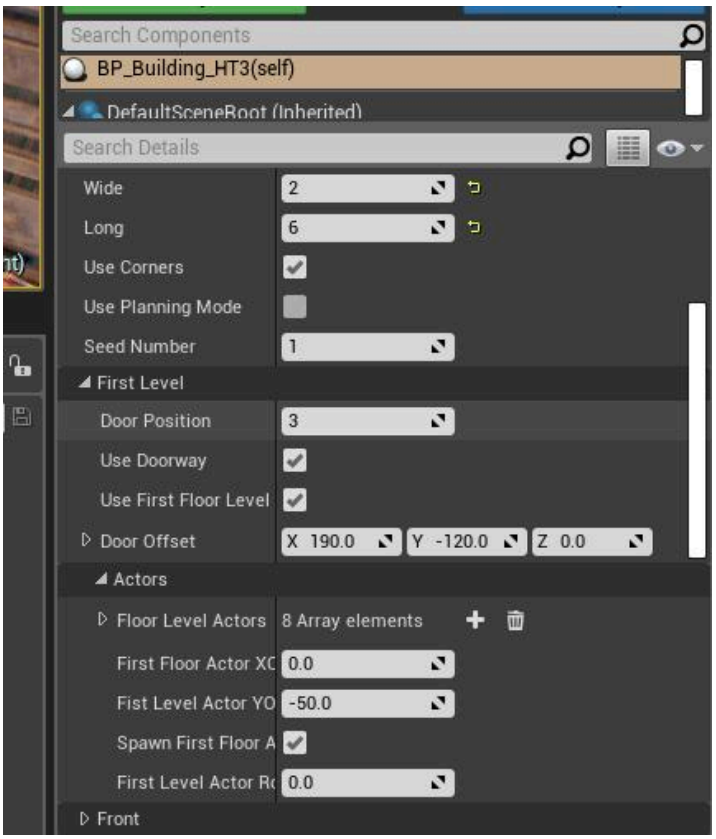
In the same place you find variables called “Floor Amount”, “Wide” and “Long”. These are used to control the building dimensions and are the most important variables in this system. “Seed Number” is a handy way to generate different small variations and keep those changes. It will affect every random function that the system uses.

Building Walls

When bounds are calculated, the next stage is wall building. This is done for left, right, front and back sides and you can find individual control for every side under “Control”. You can choose to disable constructing specific sides totally, enable/disable windows and so on. These settings are the most used ones and can have a huge impact on how the building looks.



There is also a separate control for the “First Level”. You can enable/disable this part and control where the door would appear or just disable doors totally. “Door Offset” controls the door blueprint position that is different for every building due to different building dimensions. You can also spawn actors and controls for them can be found under “Actors”. Most buildings are spawning actors like vending machines, trash containers etc… If you don't want to spawn any actors you can turn “Spawn First Floor Actors” to false. Otherwise the system will try to find actors from the actors array.

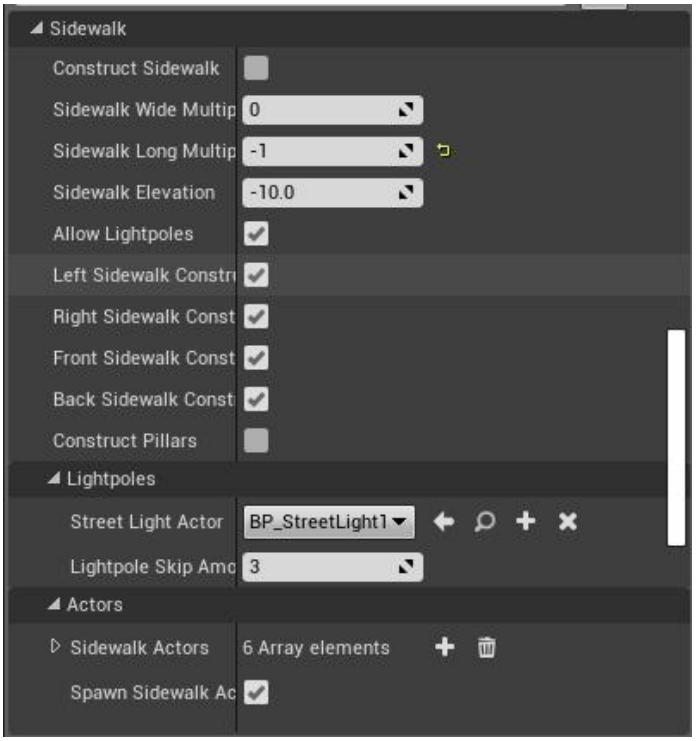


Building Roofs and Sidewalks

Then there are controls for roofs. You can disable roof construction or choose to scatter roof decorations that are specified under “Meshes”.

If you wish you can also construct sidewalks. This will construct sidewalks using the building dimensions and then you can specify multipliers that will shrink or grow that system. On top of this you can also choose to spawn light poles and control how often they are spawned.

To speed up level creation there is also an option to spawn actors on sidewalks. You can specify those actors in the “Sidewalk Actors” array and the system will then choose those actors randomly. Channing the “Seed Number” under “Control” will choose what actors are placed and where. This is a good way to generate desired outcomes.

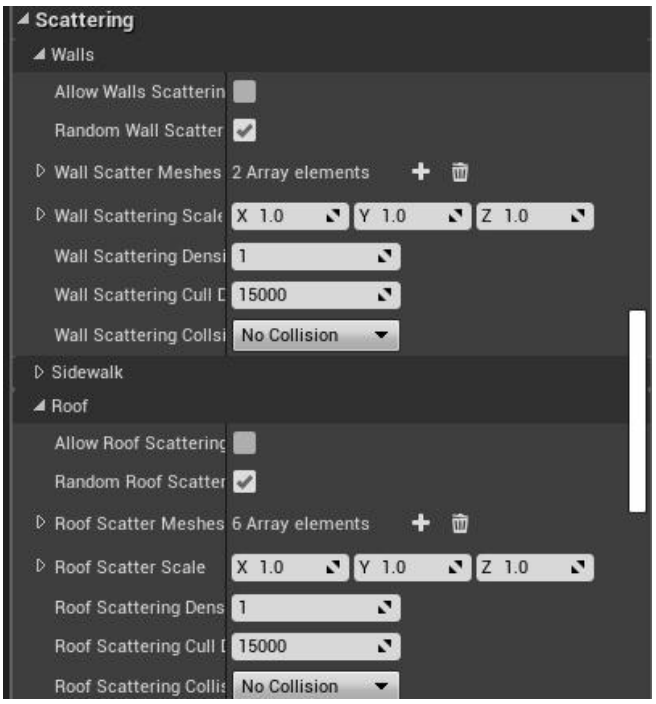


Scattering

Scattering settings are controlling different layers of meshes that are scattered on top of buildings. You can choose to enable this system for walls, sidewalks and roofs. Every layer is using the same structure variable so settings are identical for all of them.

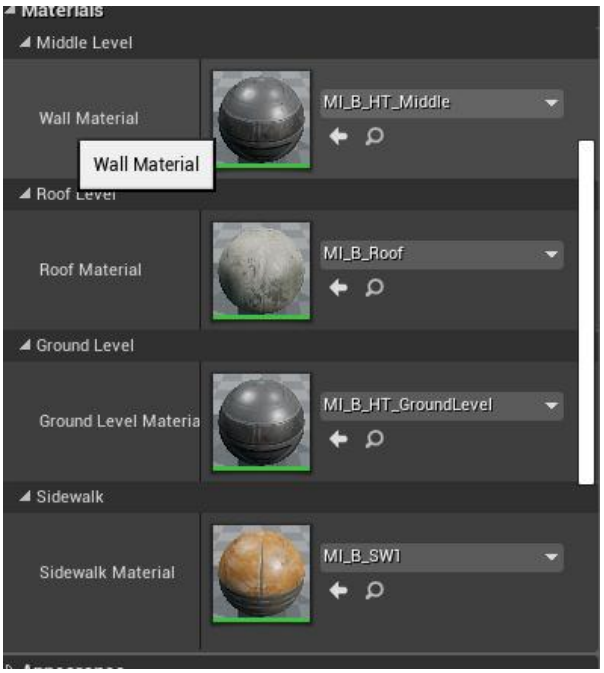
You can choose to use random rotations, apply different scales for scattered meshes, change culling distances, scatter density and even collision settings. System will look for meshes from the “Scattered Meshes” array and randomly choose different ones and put them into correct HISM components to save draw calls. You can simply drag & drop meshes and don´t have to worry about mesh instancing

For example you can spawn some air pumps and antennas on walls with a specified scale and density, allow collisions and correct culling distance and then spawn smaller debris on sidewalk with smaller scale, without any collision, random rotations enabled and smaller culling distance and maybe larger scatter density. This way you can add lots of details procedurally with minimal performance impact and still have enough control.



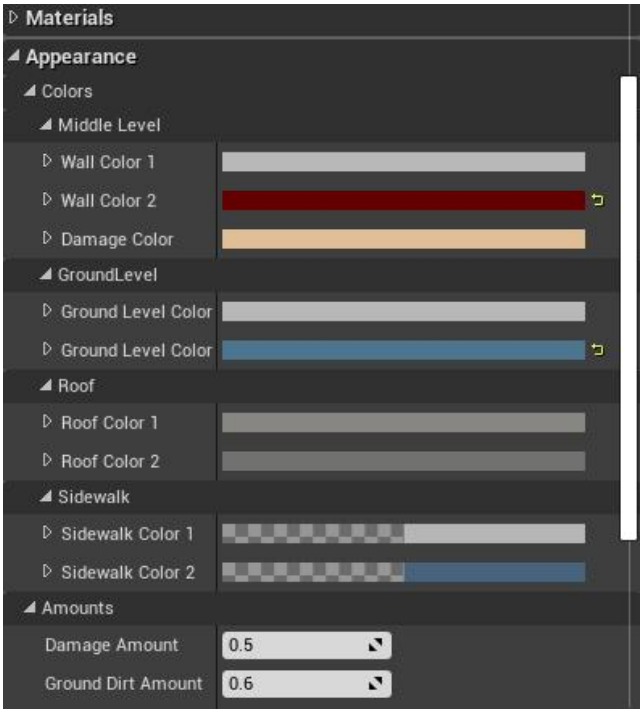
Building Appearance

Buildings are using materials that are specific for that building type. You can however change those if needed under “Materials”.



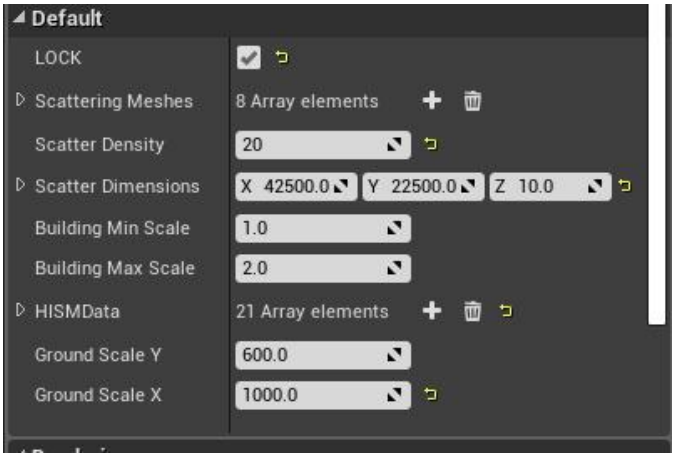
Remember if you are changing building materials to make sure those materials contain the right parameters in order to change colors and damage values.

Changing building colors is very easy. You can do that in the “Appearance” section and specify colors for walls (Middle Level), first level (Ground Level), roof and sidewalks. Under “Amounts” you can find settings to control the amount of damage and dirt buildings have.



Distant City System

Even though this building system is optimized it's not optimal to use this for far away city creation. If you need to add distant cities then you can use BP_DistantBuildings blueprint. It will scatter merged building meshes and randomly choose different building colors using material functions.



These merged buildings are usually costing only a few draw calls and the system is also instancing these meshes so it´s possible to create very large cities for background use with

minimal performance footprint. Merged buildings are low poly meshes and are using low resolution textures so they are not optimal for situations where the player is able to get close to them.

Instanced meshes need to be initialized every time the level loads and this system also uses random functions so we need to store data in order to load those instances exactly the same way every time the level is loaded. “LOCK” boolean variable is built for this use. When it´s true the system will try to read stored information so when you are happy with the results you can turn this true, save level and the same results should appear every time the level is loaded. You can also turn this variable on/off to generate different outcomes.

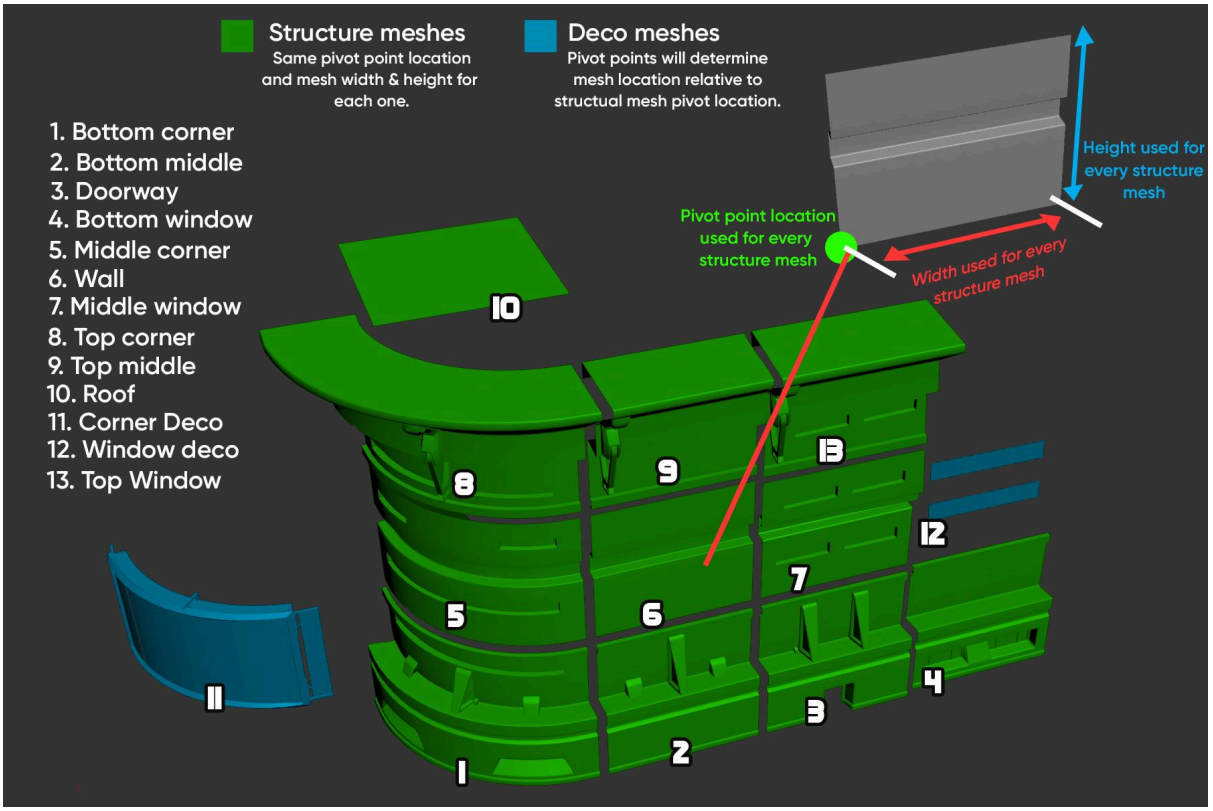
Custom Building Meshes

If you want, you can create custom building meshes to be used with this building system. It would be a good idea to create a new child class from BP_P_Building class and use it as a base for this new building type.

Buildings are using few structure meshes (green) and few decoration meshes (blue). This system is using wall mesh bounds (6) to figure out width and height for a single building slot. Building dimension variables are then controlling how many building slots wide and tall the building is going to be. Wider wall mesh means the building will be larger compared to narrower wall mesh using the same blueprint dimension values.

Every structure mesh needs to be modeled in the same width and height and pivot point location needs to be at the bottom left corner of the mesh. Make sure these meshes are also tiling correctly both vertically and horizontally. I advise you to use a grid when you are modeling so width, height and pivot point locations would match perfectly with each other. This way buildings can scale in every direction without issues.

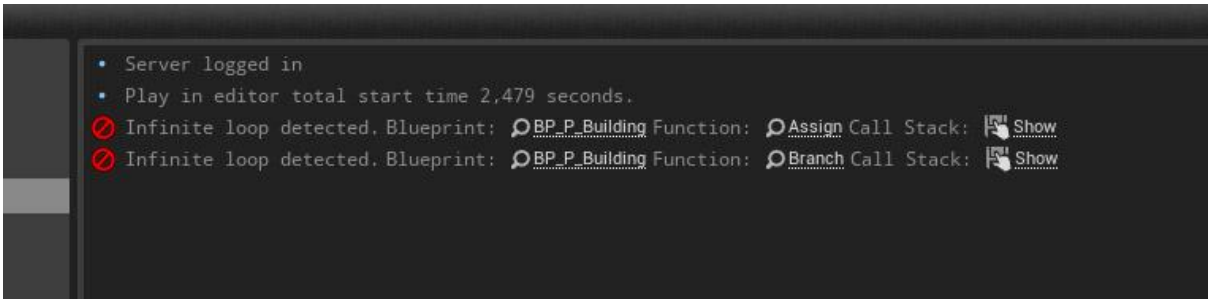
Roof and corner meshes should have the same X and Y scales. For example if the wall mesh is 4 units wide the X and Y values should also be 4 for corner and roof meshes. This way there would be no overlapping or gaps between meshes. Roof mesh pivot point Z location will determine where the roof is placed and can be used to raise or lower the roof.



Decoration meshes are then placed on top of these structure meshes. That means decoration mesh pivot locations will determine the actual location where it will be placed relative to structure mesh pivot location. Best practice would be to model these deco meshes on top of structure meshes and then use the same pivot points with both of these. This way the results in Unreal Engine would match with results in our DCC application.

Known limitations

Level Load Error

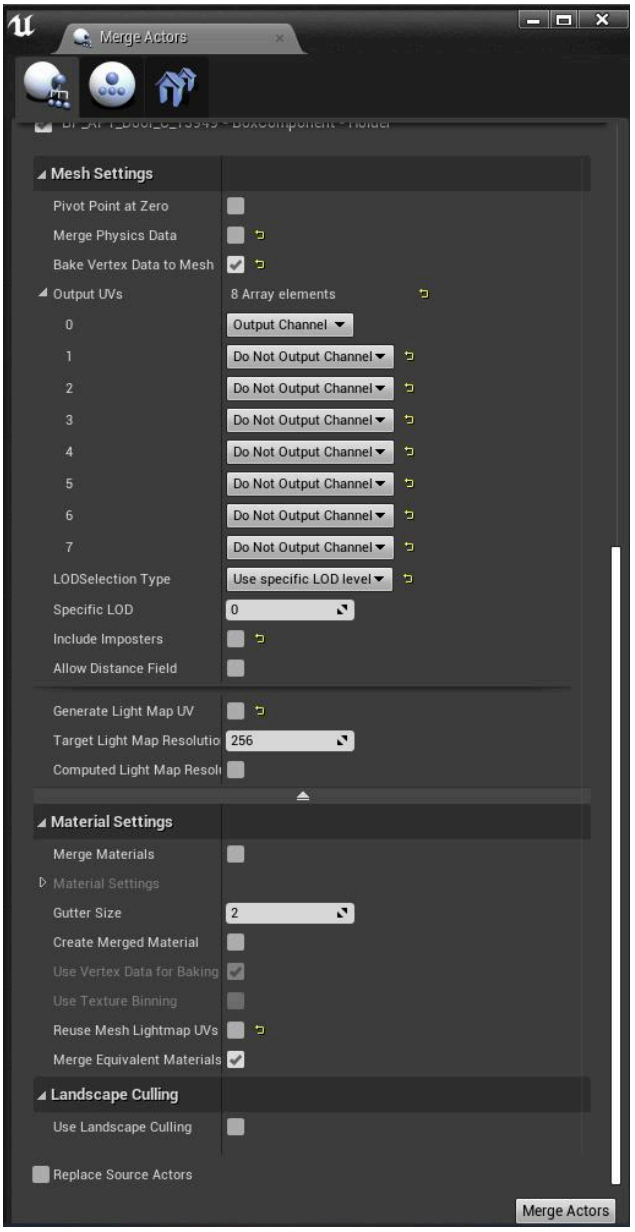


If your level contains lots of building actors then you might encounter this issue. This is because Unreal needs to load buildings each time when you open levels and there is a certain loop limit that prevents Unreal from going over it. This issue is more related to how Unreal Engine works but here are few workarounds to solve this.

1. *Reduce the amount of buildings you have in your level to reduce construction script loops.*

2. *Disable building features that you don't really need. Disabling wall, roof, floor and sidewalk scattering systems will reduce loops drastically. If this doesn't help then disable floor and socket based scattering.*

3. *Merge building actors down to simple static meshes. This way Unreal Engine doesn't have to load building actors or run any loops. You should first enable "StaticMeshVersion" because Unreal's merge tools works better that way.*



4. *For background/distant buildings you should use merged buildings or **BP_DistantBuildings** system that will generate and scatter simple meshes rather than more complex and heavier building actors.*

Lighting

Because there are lots of different light sources in this environment, many of them are put into actors. This way it's easier and faster to edit them. For example you can change street light colors and intensities just by changing those values in Blueprints -> Lights -> BP_StreetLight1.

Many of the light components are using specific draw distance when the light is culled to save performance. If you want to change that you can simply select the light inside a blueprint and then specify desired values for “Max Draw Distance”.

Static Mesh Versions

Latest update will bring an option to switch using static meshes over hierarchical instanced static meshes. Simply enable the “Use Static Mesh Version” boolean variable and it will make systems run different functions that will generate simple static mesh components. This works with **BP_P_Building**, **BP_P_Spline**, **BP_P_Scatterer** and **BP_DistantBuildings** blueprint classes.

This way it’s possible to avoid certain limitations that instanced meshes will create but it will have some disadvantages too. Because each static mesh is going to be its own component, it will result in lots of draw calls that will increase CPU and GPU cost. Another issue is related to some material functions that would randomly change colors and values based on per random instance node. These will not work with simple static mesh components.

You can change this variable for each child blueprint separately or you can change it in the parent class to enable/disable this feature for every child actor that is using this class.

Spline Blueprint System

Overview

Splines blueprints are inheriting main functions from BP_P_Spline class. This system is using a spline curve to spawn different types of meshes. This system can be divided into different parts and it's possible to use all of them at the same time to have a very advanced system.

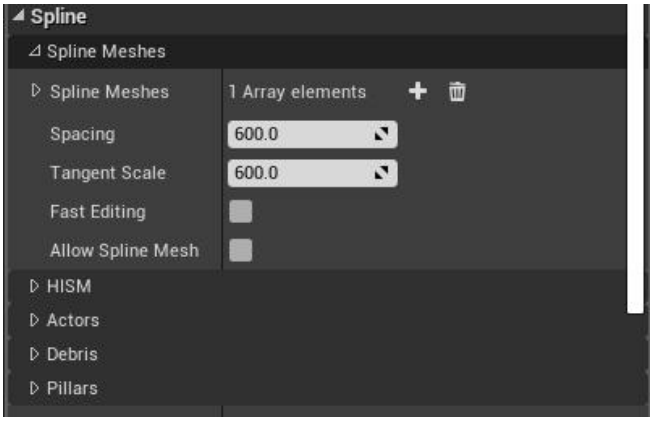
Spline Meshes

Basic use for this blueprint is to spawn spline meshes along the spline. This way meshes will deform and follow the spline curve accurately. You can read more about this system here.

"Spline Meshes" array will store meshes that are used for this system and you can specify them as much as you need. "Spacing" will control how many meshes the system will place along the spline. Using too low value will stretch meshes and too high values will flatten them. "Tangent Scale" will control the overall smoothness of the curve that we use for those meshes. Usually you want to keep this pretty close with the "Spacing" value.

"Fast Editing" is a way to make it faster to edit splines in the editor but you want to change that to false when you are done with editing. "Allow Spline Mesh" boolean will tell the system whether we use spline meshes or not.

Keep in mind that spline meshes can increase your draw calls dramatically. The more spline meshes you use the more draw calls it will generate. Using small meshes with high spacing values is the worst case scenario.



HISM Meshes

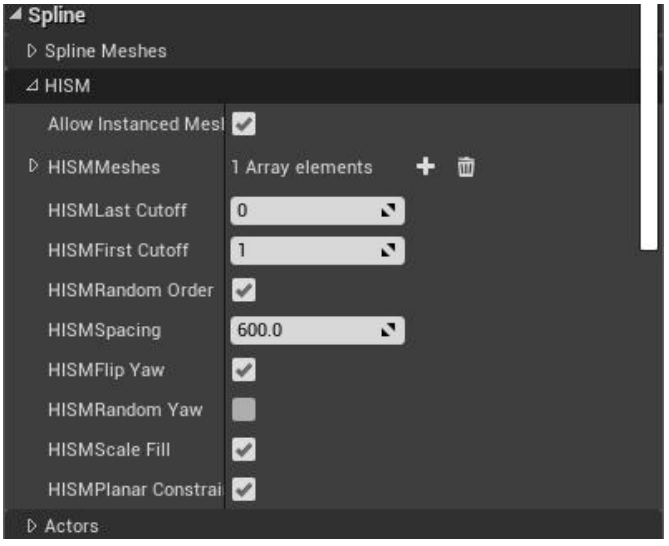
Using instanced meshes is a more optimized way because then we can batch meshes better. Unfortunately HISMs are not supporting mesh bending in the same way that spline meshes do. However there are lots of cases where we don´t really need to bend meshes so HISMs are a more optimal choice then.

You can find similar settings here like those that are used for Spline Meshes. You can enable/disable this function, specify what meshes to use, random order to use these. Then there are some settings to control rotations.

Cutoff settings allow you to cut mesh instances from start or end spline points. This is useful when there are unique meshes in start and end points and we don´t want to add HISM instances on top of those.

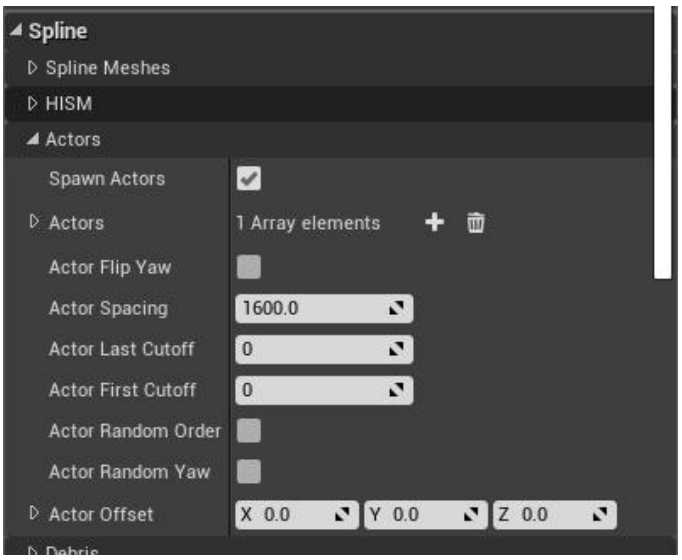
“HISMScale Fill” will help to fill gaps in cases where the mesh length will not match with the spline length. When this setting is true the system will scale the first mesh to fill this gap.

“HISMPlanar Constraint” setting will constraint HISM instances onto a plane. This way it will keep z location the same for every instance. Useful for situations where meshes need to be on top of surfaces.



Actor Spawning

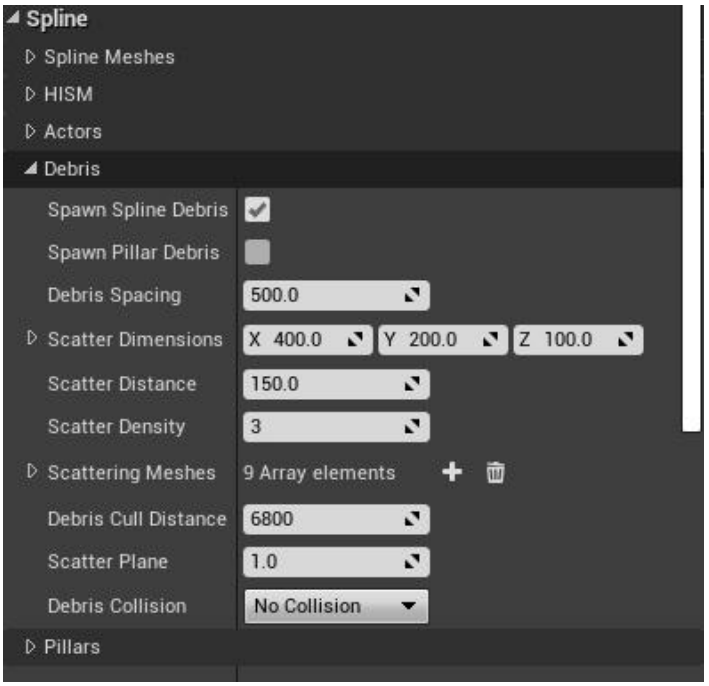
You can also spawn actors along the spline. These settings are almost identical with HISM settings. There is an array where to specify what actors to spawn and actor spacing value will control the density how often these actors are spawned based on the spline length.



Debris Spawning

Spawning debris is also supported with this system. “Scattering Meshes” array will contain meshes that are going to be scattered. “Debris Spacing” value is controlling how often the scattering will happen along the spline and “Scatter Density” will control the amount of meshes that are spawned when that happens. “Scatter Dimensions” will specify how large the scatter area will be and “Scatter Distance” is controlling how far traces are going to go.

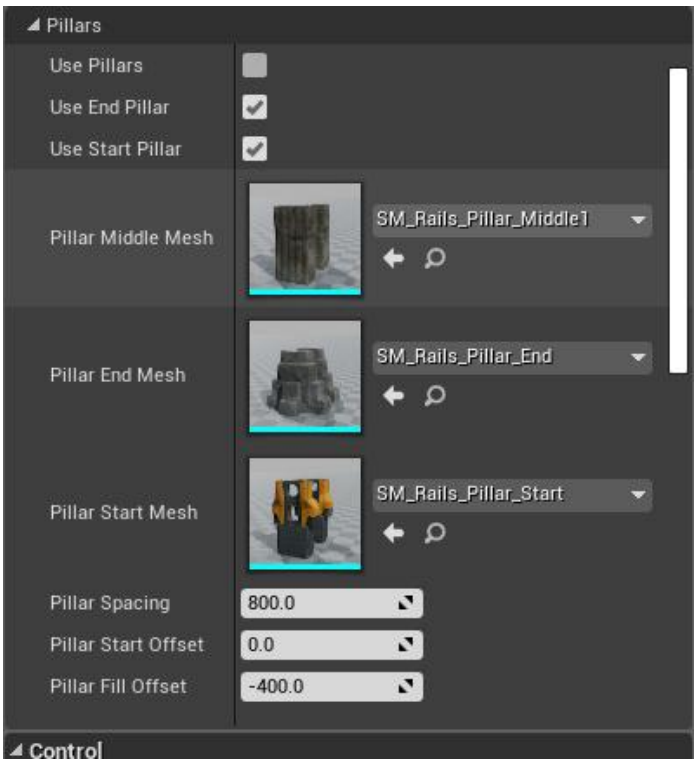
This system is creating HISM components under the hood to optimize performance footprint as much as possible.



Pillar Spawning

If you need to add supporting pillars you can turn the “Use Pillars” option on. This system will trace lines from locations that are taken from the spline based on the “Pillar Spacing” value. You also have extra control to specify whether or not to use end and start pillar meshes.

Start offset will allow to move pillars up/down from the spline and “Pillar Fill Offset” is used to control mesh scaling to avoid gaps or mesh overlapping.



Scattering Blueprint System

Scattering blueprint (BP_P_Scattering) is basically using two different systems to find transforms where to add instanced meshes. These systems are line traced based and plane based.

Line traced based is more accurate but will be a bit slower to update when changing values in the editor. Plane based will just take random points from a plane and will ignore the actual environment. This is faster and works well when you need to scatter something on a flat surface. You can change between these two modes with the “ConstraitPlaneMode” variable.

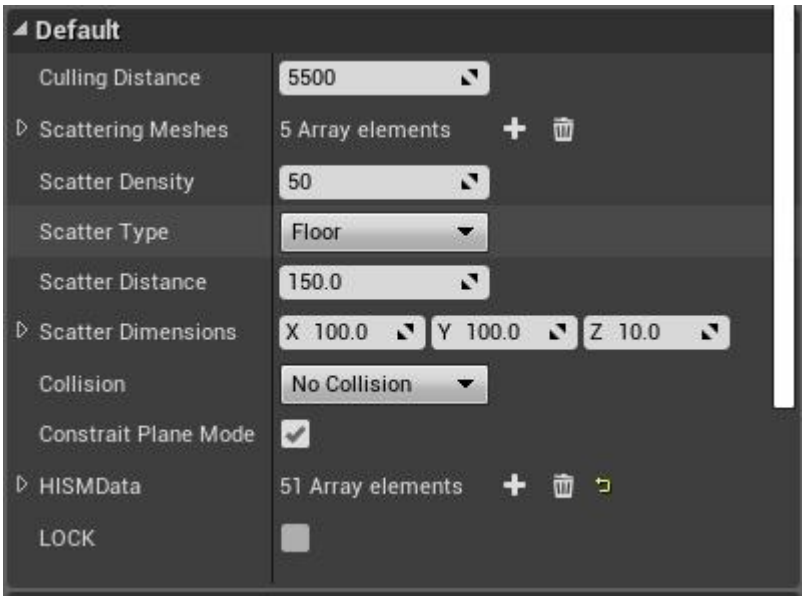
“Culling Distance” is controlling when to cull mesh instances. Smaller value means that instances will be culled faster and a value 0 means instances will not be culled unless culling volumes or systems similar to that cull it.

“Scattering Meshes” array will be used to figure out what meshes the system will scatter. It will automatically create HISM components for each unique mesh type and then store the same type of meshes there. This will keep draw calls at minimum.

“Scatter Density” will control how many times the system will run its loops and scatter meshes. “Scatter Type” enum controls how to align meshes onto surfaces.

“Scatter Dimensions” vector variable is specifying the actual scatter area where the system will take points for tracing. You can also change collision settings whether or not to use collision.

Instanced meshes need to be initialized every time the level loads and this system also uses random functions so we need to store data in order to load those instances exactly the same way every time the level is loaded. “LOCK” boolean variable is built for this use. When it´s true the system will try to read stored information so when you are happy with the results you can turn this true, save level and same results should appear every time the level is loaded. You can also turn this variable on/off to get different outcomes. If you want to get random results every time the level is loaded, leave this boolean value to false.



You can also use this blueprint as a child actor. This way you can for example scatter cans from vending machine blueprints etc...

Procedural Foliage

Overview

Rocks and larger foliage are scattered using the procedural foliage spawning system that comes with Unreal Engine. You can find more information about that system here. Example level uses two *ProceduralFoliageVolumes*, one for foliage like cactus and another one for rocks. If you want to resimulate instances you can select these volumes and then find the "Resimulate" button and press it. If you need to block this scattering for some areas you can use *ProceduralFoliageBlockingVolumes* for that.

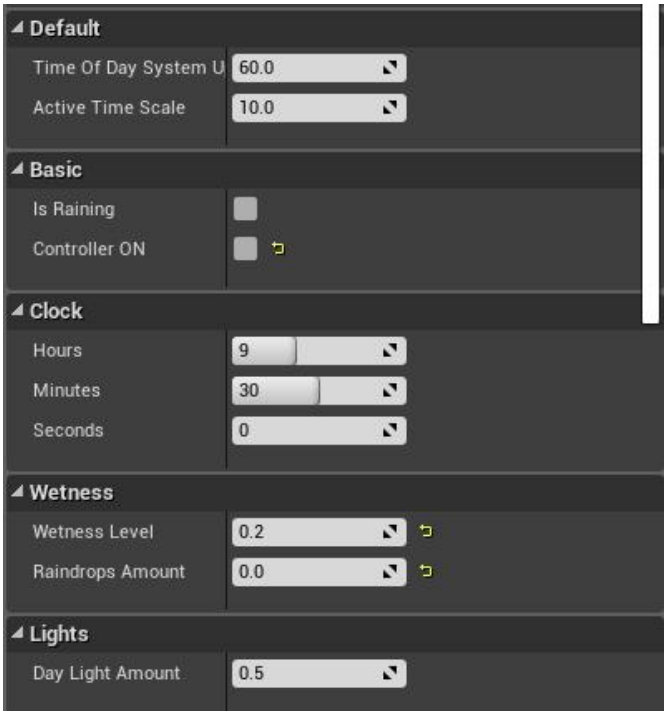
Grass and smaller rocks are using the landscape grass system. You can tweak individual LandscapeGrassTypes (*Materials -> Nature -> Landscape*) to have better control for scale, density and lighting settings. You can find more info here.

Environment Controller

Overview

This pack also comes with a basic environment controller. It supports a fully dynamic day and night cycle and an option to use rainy weather or control different wetness values (Wetness Level, Raindrops Amount).

You can specify hours, minutes and seconds and the system will update values based on that. Main logic is using curves that you can tweak for your needs and the system also works in the editor. If you wish to use it at runtime you can turn the "Controller ON" variable to true.



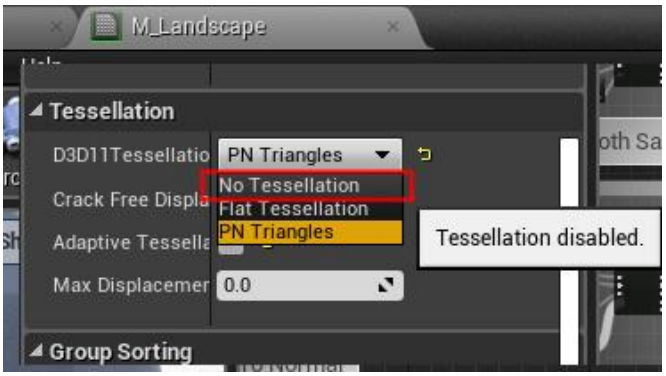
“Is Raining” will toggle runtime rain on/off. This will change wetness values and will also add raining water particles to the active camera. “Day Light Amount” will control building window interior lighting intensity. 0 would mean that there is no interior lighting at daytime.

Optimizations

If you need to optimize performance even more, here are few tips for that.

Remember that in this case optimizing will disable some features that can have a huge visual impact.

Landscape material is using tessellation for better visuals but you can turn this feature.



Another thing to consider is grass density. Grass is using landscape layers and is a part of the landscape grass system. Foliage setting in Engine Scalability is also affecting how dense

the grass is and this can have a huge impact on overdraw. You can also tweak individual LandscapeGrassTypes to have better control. You can find more info here.

Master materials are already using quality switches to disable some heavy features when material quality is changed to low. You can also turn various layers off from the material instances to save instructions.

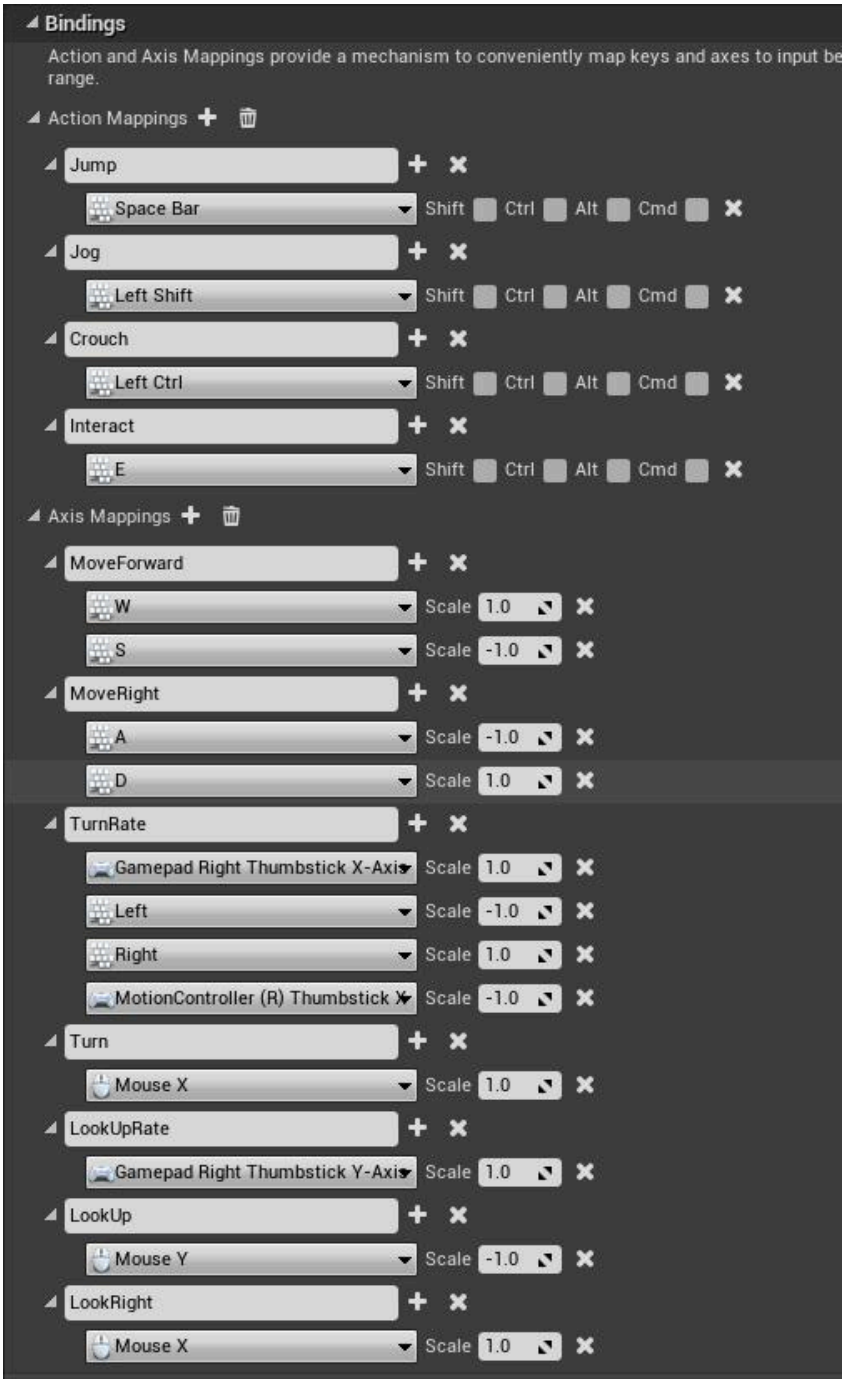
Post process effects can eat performance and you can turn off features that you don't need. Screen space reflections can cost a lot so you can decrease the “Quality” setting or disable this feature completely off by changing “Intensity” to 0. You can do the same thing with ambient occlusion. Both of these settings can be found in the post process volume (PostProcessVolume2).

Example Player

This pack also comes with an example player blueprint. It is based on the basic Unreal Engine first person pawn with some small changes.

It will support basic WASD movement, sprinting (Left Shift), crouching (C), zooming (Right Mouse Button).

Because this product is an asset pack, you need to manually specify these key mapping settings in the “Project Settings” in order to control the player. Alternatively, you can also download input settings and simply import them into your project (Choose “**Import**” on the top right in the Project Settings > Input tab).



Drivable Vehicles

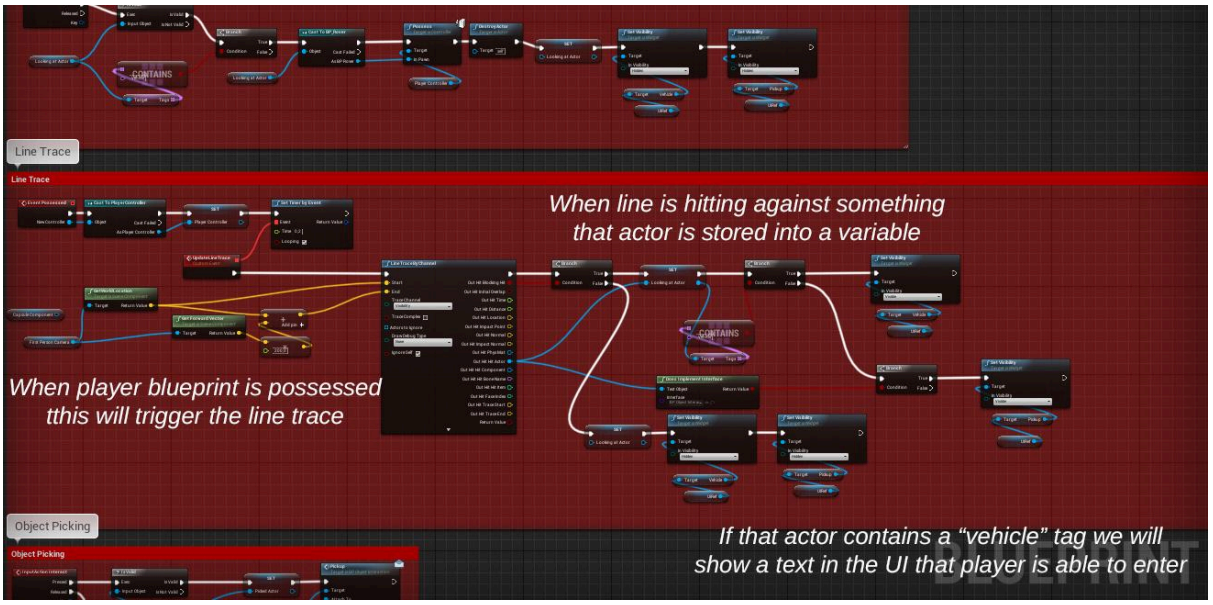
Overview

Latest update will now include a car that the player can possess and drive. It's based on the UE4 PhysX vehicle class. There are lots of values to tweak and you can find more

information about that system here:
https://docs.unrealengine.com/en-US/Engine/Physics/Vehicles/VehicleUserGuide/index.html

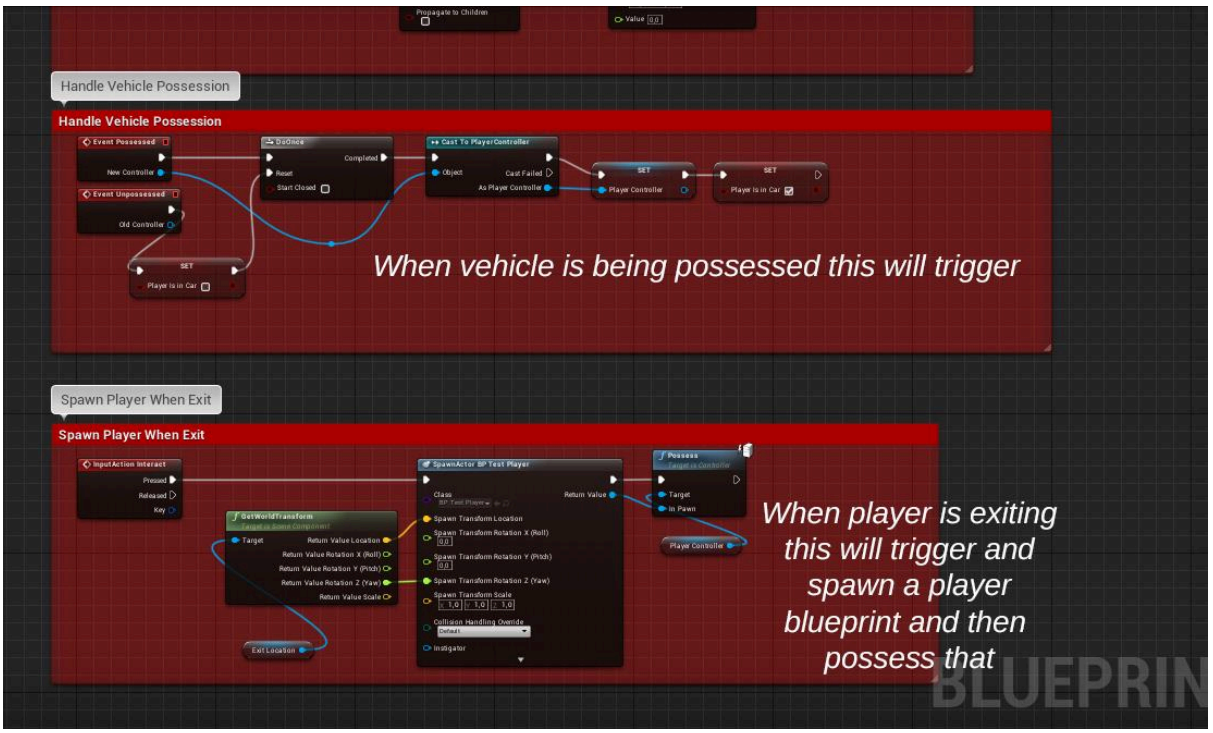
Driving

Main concept here is that the player is using a line trace to find the actor. This actor is stored in the **LookingAtActor** variable so it can be used as a reference when casting to *BP_P_VehiclePawn*. Car blueprint has a tag "Vehicle" and the player blueprint is trying to see if the line trace is hitting on anything that has that tag. If it's true then the player is able to enter into the car when pressing interact (input is added in project settings). Player Controller is then handling possession so in this case "In Pawn" will be the car.



In the car blueprint "Event Possessed" will now trigger because that is a pawn that we are controlling now. There we are setting a few variables that the player is in the car and set the *PlayerController* variable. When the player wants to get out and is pressing the interact key we will just spawn a *BP_TestPlayer* or your custom character and then possess that using again that possess node.

That will then trigger the "Event Possessed" node in the player blueprint and we then set the *PlayerController* variable and then set a timer that will trigger the line trace that was mentioned at the beginning.

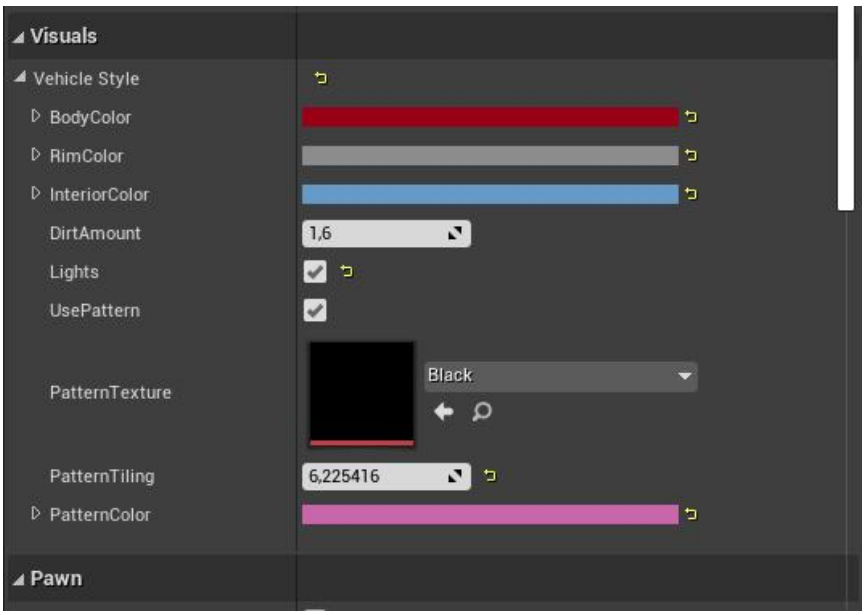


You can also change camera view from third person to first person. By default the button “*V*” is used but you can change this based on your needs.

Visuals

On top of the driving logic this system will also include some variables that change car visuals like colors, patterns and dirt.

There is a struct variable called *Vehicle_Style_Struct.* This holds different colors and variables that you can change. If you select the actor *BP_Car_Compact_Pawn* and go to the Details panel, you can see this Visuals tab. There you can change body, rim and interior colors, dirt amounts and even use patterns. You can also turn lights on/off or you can do this at runtime by pressing a “*Q*” button.



Demo

In the demo you can test the whole example map. It will contain everything that comes with the product. Inputs are:

Movement: **WASD**
Sprinting (**Left Shift**)
Crouching (**C**)
Zooming (**Right Mouse Button**)
Toggle rain (**T**)
Enter vehicle (**E**)
Change vehicle camera (**V**)
Toggle vehicle lights (**Q**)
Restart level (**R**)
Exit game (**ESC**)
Toggle developer console (**Tab**)

Any feature requests or questions? Please feel free to ask them
(kimmo.koo@hotmail.com)

Join to the Discord server
KK Design support policy

More Unreal Engine assets be found here