Scattering Guide

This is a short guide for setting up the relevant scattering assets and components controlling the scattering.

Scattering Point Clouds

PointCloudFromHoudini (Asset)

This asset contains the actual scattering data extracted from point clouds. The extraction tries to automatically find prefabs used for scattering by inferring it from the pointcloud filename. The format used is scatter_pcache_<name>_pc_prefab or scatter_pcache_vegetation_<name>_pc_prefab. The filename suffixes in red are removed from the name and the green postfixes are added to the rest of the filename. So for example, for a pointcloud named scatter_pcache_grassfield3, the logic tries to find a prefab with a name grassfield3_pc_prefab. If this logic fails, or a different prefab is desired, one can override the prefabs used in the asset after extraction. Ticking “Show Prefab List” shows the prefabs used and one can override the resolved prefabs. Click “Apply prefab overrides” and “bake” to save the overridden prefabs and force a rebake for scattering.

The Basic UI for the asset is:

- **Point Caches**: This is a list of point cloud data that is to be extracted from
- **ExtractAndBake:** Extracts the data from the point caches and forces a bake for DOTS entities

Under Advanced tab:

- **ExtractPointCloudData:** Extract data from point cloud but doesn’t force a rebake for DOTS entities
- **Bake:** bake whatever data is present in the asset without re-extracting from the source point caches.
- **Show Prefab List:** List prefabs used for scattering and allow overriding them
- **Ignore Max Scatter Distance:** This is relevant when using tile impostors, if this is ticked, the scatter won’t be faded out when the tile impostors appear

ScatterPointCloudAuthoring (Game Object)

The actual authoring component for the DOTS entities. It needs to exist in a DOTS subscene and will issue the baking of the DOTS runtime data from the PointCloudFromHoudini asset.

- **PointCloudAsset:** reference to the asset containing the pointcloud data
- **Scatter Active Distance Min:** the minimum distance from the viewer where the scatter scene sections should be loaded. This is usually 0 but if you want to unload scatter scene sections when going closer, this can be something higher

- **Scatter Active Distance Max:** The maximum distance for the scattering scene sections. After this distance, the scene loading logic will unload the scatter. Not that this is not the same thing as how far the scatter can be seen, rather, its the distance at which point the while scene section is removed and the scattering with it. Usually the range that you actually see the scatter is lower.
- **Scatter Scene Section Size:** The size of a scene section. The baking will bake the scattering data to tiles and this defines the size of the tile that is loaded/unloaded from disk. Note that there is a hard limit in the scene header file and having too many scene sections will cause an error in baking. If the point cloud is too big spatially, the scene section size needs to be big enough to accommodate. If you see an error saying the the scene header is too big, increase this size.
- **Scatter Tile Size:** This is the granularity that the scene section data is further subdivided to tiles. When a scene section is loaded, the data is fed to point cloud system for instantiation, which then takes batches of scatter data to instantiate. This defines the size of that batch. Note that there is a separate control at runtime of how many instances (and thus batches) are actually instantiated at a time.

Point Cloud System Config (Game Object)

This is a control for the different system participating in the scattering logic. There should only be one of these active at any given time.

Streaming settings:

- **sceneSectionsMaxInstancesToLoadConcurrently:** This throttles the maximum amount of instances to load concurrently from disk.
- **maxNumberOfInstancesToSpawnPerBatch:** The instantiation of scattering is done on worker threads and copied to the mainworld when done. This controls the number of instances to instantiate concurrently. Small values increase the instantiation latency, big values can cause stutter when the instances are copied to main world.
- **disableScatterWhileLoadingPrefabs**: Forces scattering to wait for the prefabs to load. Usually should be kept on.
- **immediateBlockingScatterOnStartup:** Do a blocking instantiate pass at startup (when entering play mode)
- **targetSizeOnScreen, scatterInstanceSizeMinMaxInterpolation & unloadSizeMarginRatio**: When selecting what batches to instantiate next, there is a heuristic that tries to find the most important ones. This is affected by distance and if the tile is visible in camera, but also by size. **ScatterInstanceSizeMinMaxInterpolation** interpolates between the minimum and maximum size of an instance in a given batch that is then checked against the camera parameters to deduce the approximate size on screen. **TargetSizeOnScreen** is used to decide if the batch should be instantiated or unloaded. **UnloadSizeMarginRatio** is added to the size when checking for

unload so that the tile isn't immediately unloaded if the camera goes back and forth in small area.

Tile impostors and culling:

- **tileImpostorLOD0Distance**: a distance where tile impostors start to become visible and the scattering starts to fade out (unless Ignore Max Scatter Distance is set in the scatter asset).
- **tileImpostorLOD1Distance:** distance to switch to sparser tile impostor
- **tileImpostorCullDistance:** distance where the tile impostors are culled too
- **tileImpostorDistanceFade:** the distance over which the tileimpostors fade in and scattering fades out.
- **shadowCullScreenSizePercentage, shadowCullScreenSizeVariance, cullShadowsOutsideCamera, cullShadowsCameraPlaneMargin & maxShadowCasterChunkChangesPerFrame:** Controls when to cull an instance from being a shadow caster. If the instance is smaller than shadowCullScreenSizePercentage +- shadowCullScreenSizeVariance, it will have shadow casting toggled off. If cullShadowsOutsideCamera, then instances outside of camera will also have shadow casting toggled off. Use cullShadowsCameraPlaneMargin to increase the camera frustrum to mitigate problems when the camera rotates and maxShadowCasterChunkChangesPerFrame to control how many instances to change per frame as there is a cost associated with it.
- **lodTransitionConstantOffset, lodTransitionMultiplier & lodTransitionMultiplierAffectsCullingDistance:** bias and multiplier for the LOD transitions (inferred from LODGroups of the scattered prefabs). lodTransitionMultiplierAffectsCullingDistance controls whether the bias and multiplier will also trigger completely culling the instance when too small on screen.
- **alwaysVisibleScreenSize, neverVisibleScreenSize & screenSizeCullingEasing:** On top of culling instances based on LODGroups, the scattered instances are culled based on their size and the relative density of the area they are in. AlwaysVisibleScreenSize defines what is the size of the instance on the screen when an instance is always visible and neverVisibleScreenSize defines when an instance is never seen. Between these sizes, the object might be culled if it occupies a high density area. screenSizeCullingEasing controls the interpolation between the alwaysVisibleScreenSize and neverVisibleScreenSize.

Miscellaneous:

- **forceConstantEditorUpdate:** Unity doesn't update the ECS systems in editor view constantly. Settings this will force the update and have the scattering logic constantly running.
- **drawLODDebugColors:** Fade in coloring for the scattered instances that show the LOD they are using

- **debugColors:** Colors used for the LODs debug.

Tile Impostors

BakedTileImpostorDataSet (Asset)

Tile impostors are used to approximate a large area of grass on a given tile. A tile impostor consists of scattered instances rendered to a texture from above and an atlas of most used scattered instances rendered from side. These are combined at runtime to give somewhat correct coloring and silhouette of the actual grass residing in the tile. However, these textures tend to be fairly low res and the heuristic nature makes them break down when seen from too close. Tile impostors also generate a mesh with approximate height information and materials and prefab to be used to bake the impostors into ECS scene.

The Basic UI for the asset is:

- **Bake Sources**: This is a list of PointCloudFromHoudini assets along with transform information to bake the instances to correct locations.
- **ImpostorType:** Type of the impostors. Only TileCards are actually used, the TileMesh is an unfinished prototype.
- **Tile Texture Resolution:** Output resolution of the tile impostor textures
- **DownSampling Count**: The internal rendering resolution of the tile impostor textures can be higher (this is a multiplier for the output resolution) and then downsampled to produce the final output resolution
- **Size Of Tile impostor**: the size of a single tile impostor. Bigger means fewer tile impostors but also more approximate results.
- **Impostor Tile Discard Ratio:** if a given area contains too few instances, the tile impostor can be omitted. This is the threshold to omit a tile impostor if the texture rendered from above has less than discard ratio of valid pixels.
- **ExtraBoundsMargin:** Extra bounds added to the tile bounds when generating the impostor
- **Impostor Height Bias:** Height Bias added to tile impostor.
- **Diffusion Profile Override:** Diffusion profile used by tile impostors
- **Use Area Limits & Area Limits:** This is mostly for debug to constrain the area that tileimpostors are generated for

TileImpostorAuthoring (Game Object)

Authoring object for the tile impostors. It just takes a reference to the BakedTileImpostorDataSet and tile impostors are always in scene section 0, which means that they are loaded first before any scattering.