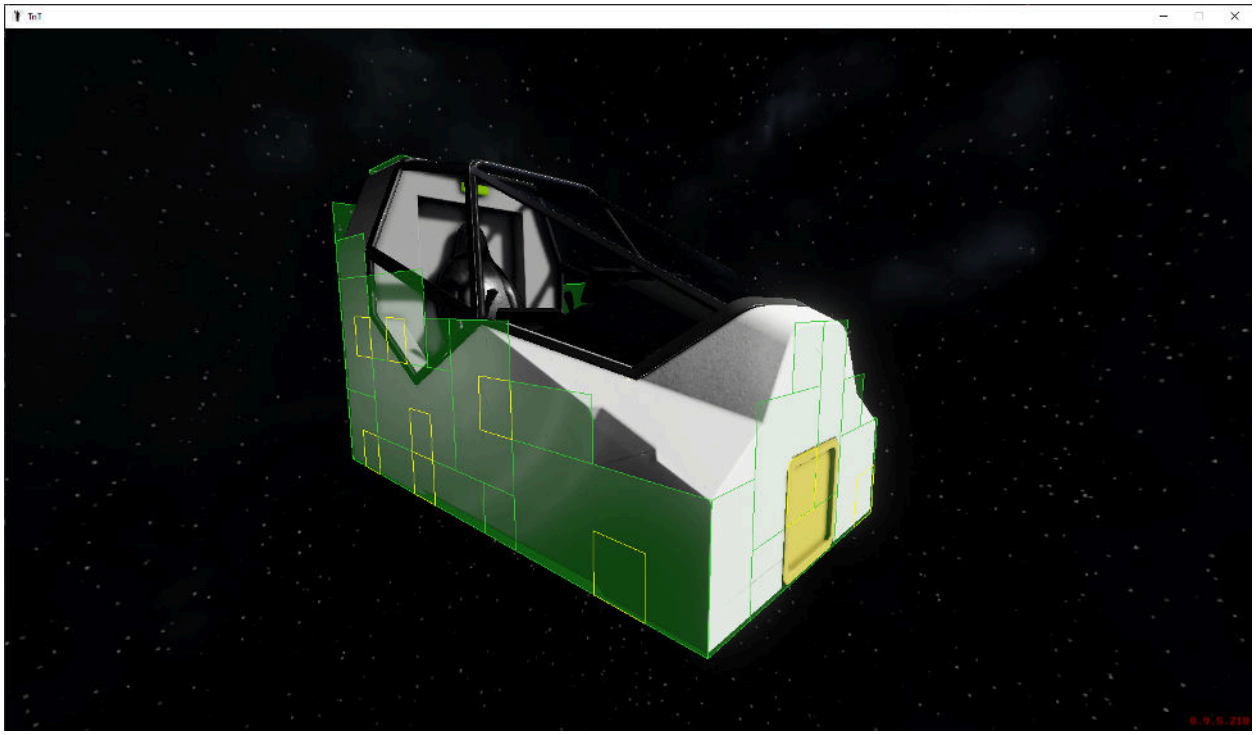
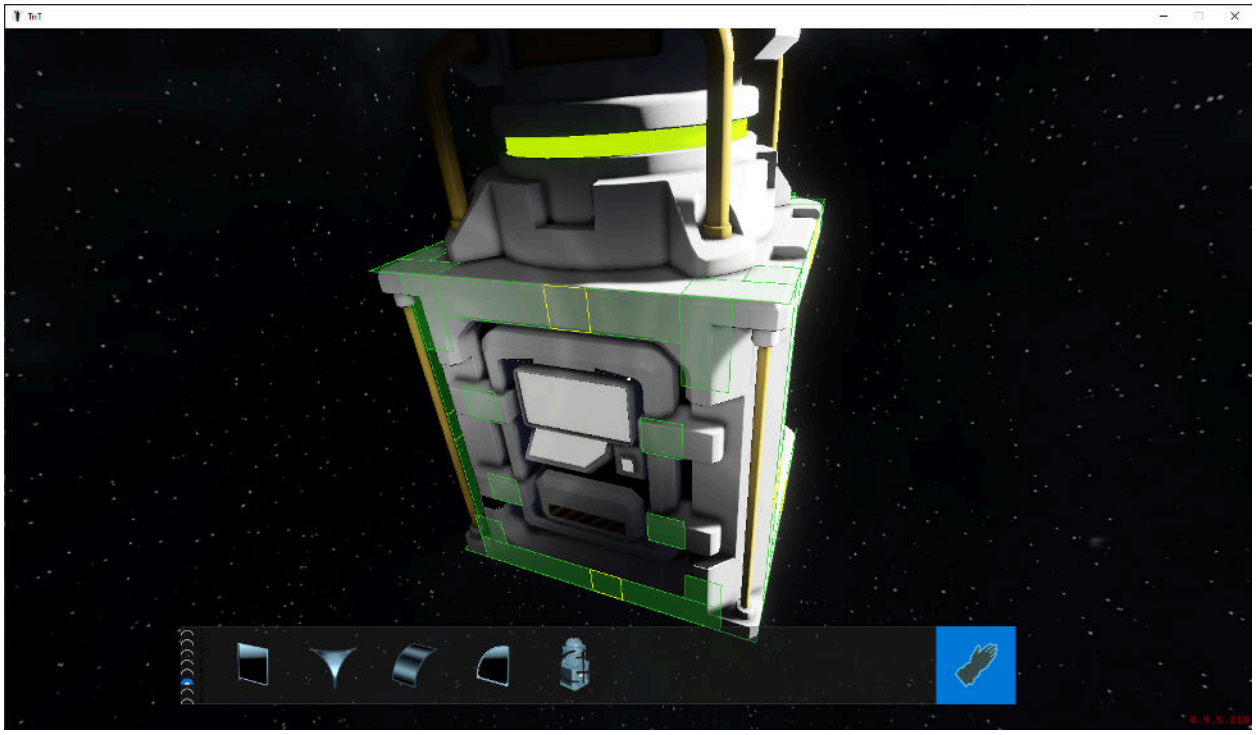
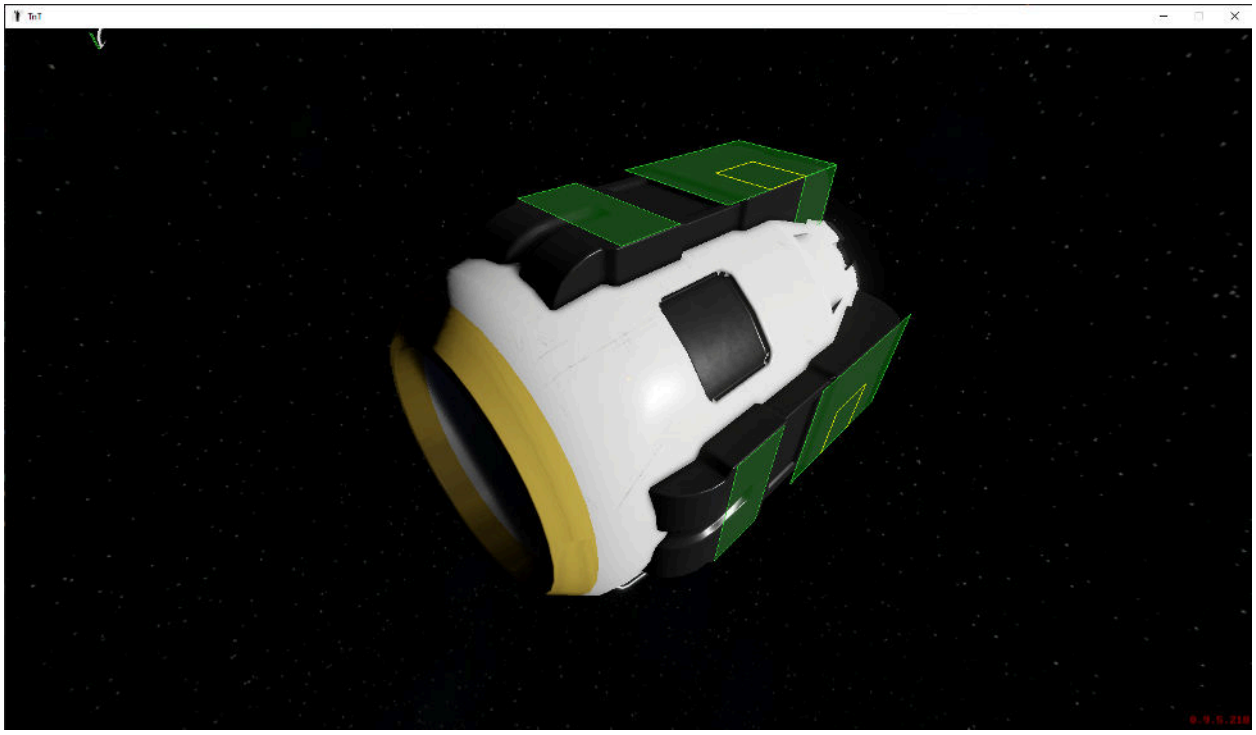
Mount Points Generator

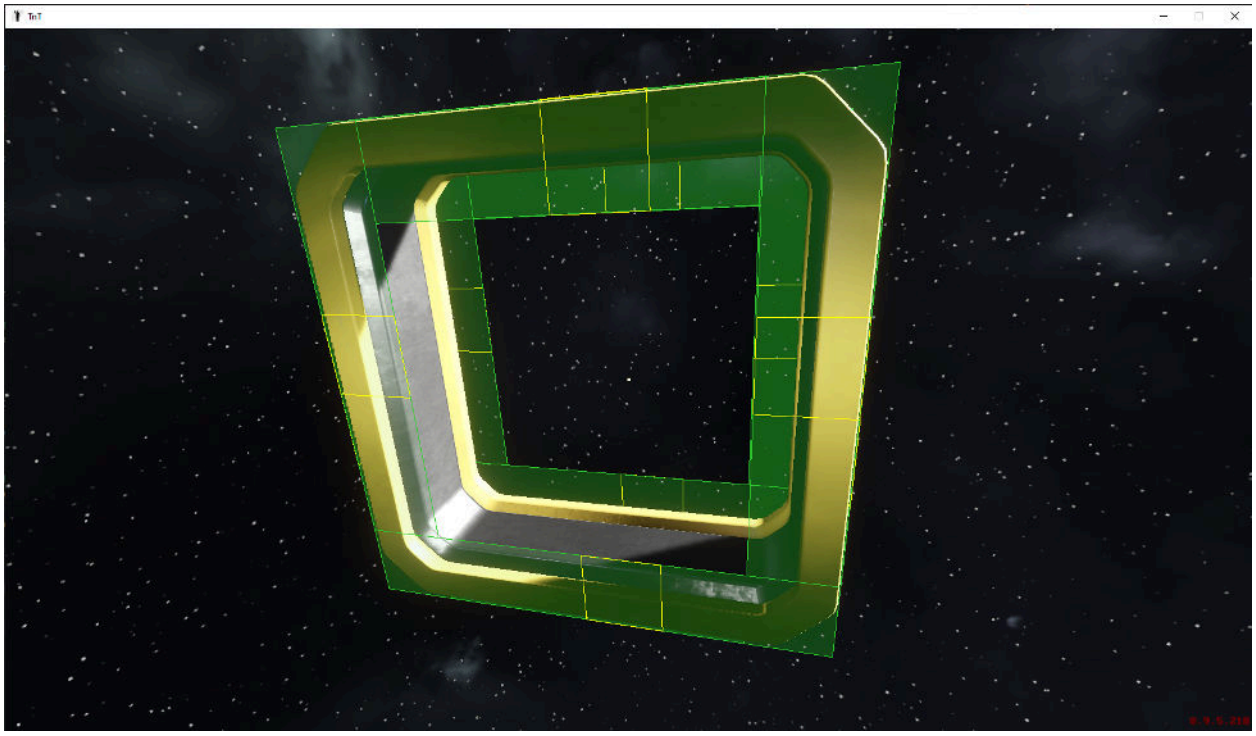
Overview

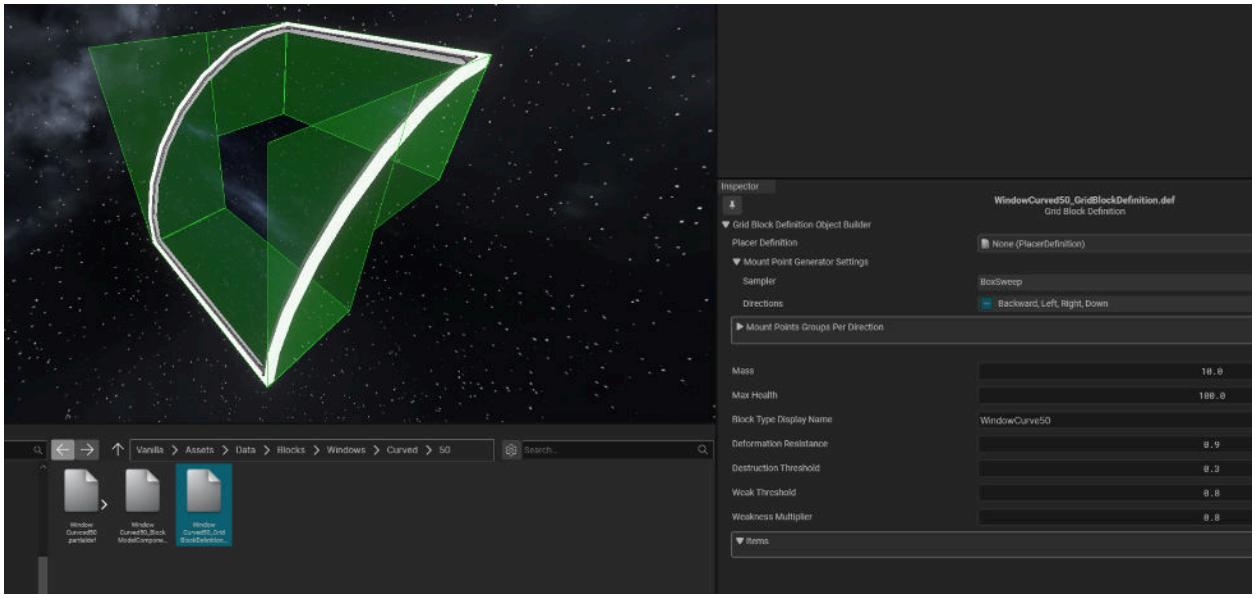
The mount points generator aims to automate the creation of mount points on every block in game. Mount points are areas of the block that the player can place other blocks onto. Valid mount points are found on flat areas of the block, orthogonal to one of the 6 viewing directions.






TnT


TnT



Implemented Features

- Automatically generate mount points of all blocks in the game (armor & regular)
- Associate mount points to closest fracture on breakable blocks
- Grouping of individual mount points (by face & fractures)
- Basic primary mount point selection
- Detailed debug draw with multiple toggles to control debug output
- ContentCache support for armor blocks
- Support loading extra artist-provided colliders from block model used as additional hint by the MP generator

Not Implemented Yet

- Better primary mount point selection using dummies as hints
- ContentCache support
- Content Validation
- More thorough unit tests coverage
- Material-based rules
- Regenerate mount points when model changes in editor

Known Issues & Limitations

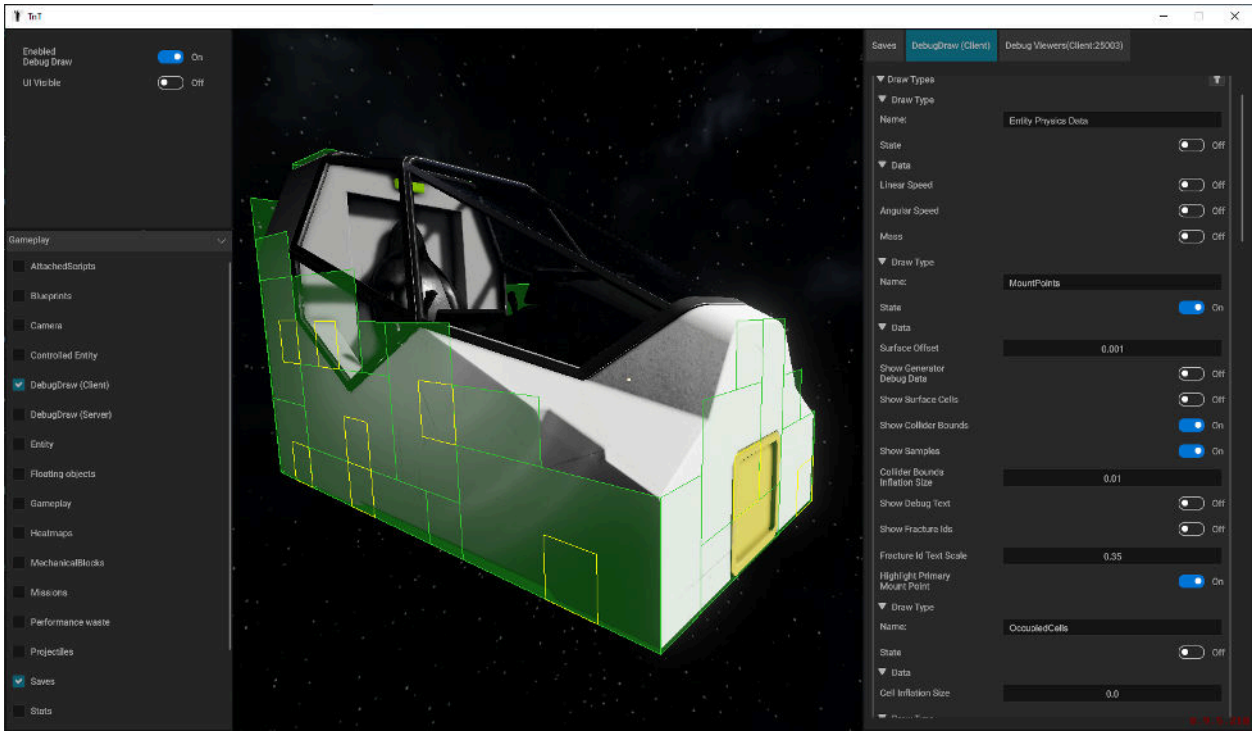
- The generated MP data will not refresh in editor if the associated model changes (you will need to restart the editor at the moment)
- The detailed debug draw information is not togglable in editor yet (debug data settings not exposed to the editor UI), so they can only be controlled in-game at the moment

How It Works TLDR

- Upon game start, a DefinitionPostProcessor will run on all block prefabs in the game
- The MP generator will be run for each eligible block prefab and takes the following input:
    - Block occupancy data (output of block occupancy generator)
    - Precise collider of the block's LOD0 model (generated at runtime)
    - Optional: fractured model collider (to lookup fractures)
- Block occupancy is used to determine the "surface cells" that are on the outermost regions of the block
- The generator will use the block's selected "mount point sampler" to scan the geometric details of each surface cell and identify valid cells that can receive a mount point. For instance, the Circular8 sampler checks the sample distance and normal to ensure the target surface is flat and each valid sample hit is at similar depth.
- If model has fractures, the generator will also perform an additional raycast on the fractured model collider for each sample to find the fracture it corresponds to
- Individual mount points are grouped by side and fracture
- The CubeGridDefinition of the block is updated with the generate mount points data (CubeGridDefinition.MountPointsGroupsPerDirection)

Debug Draw

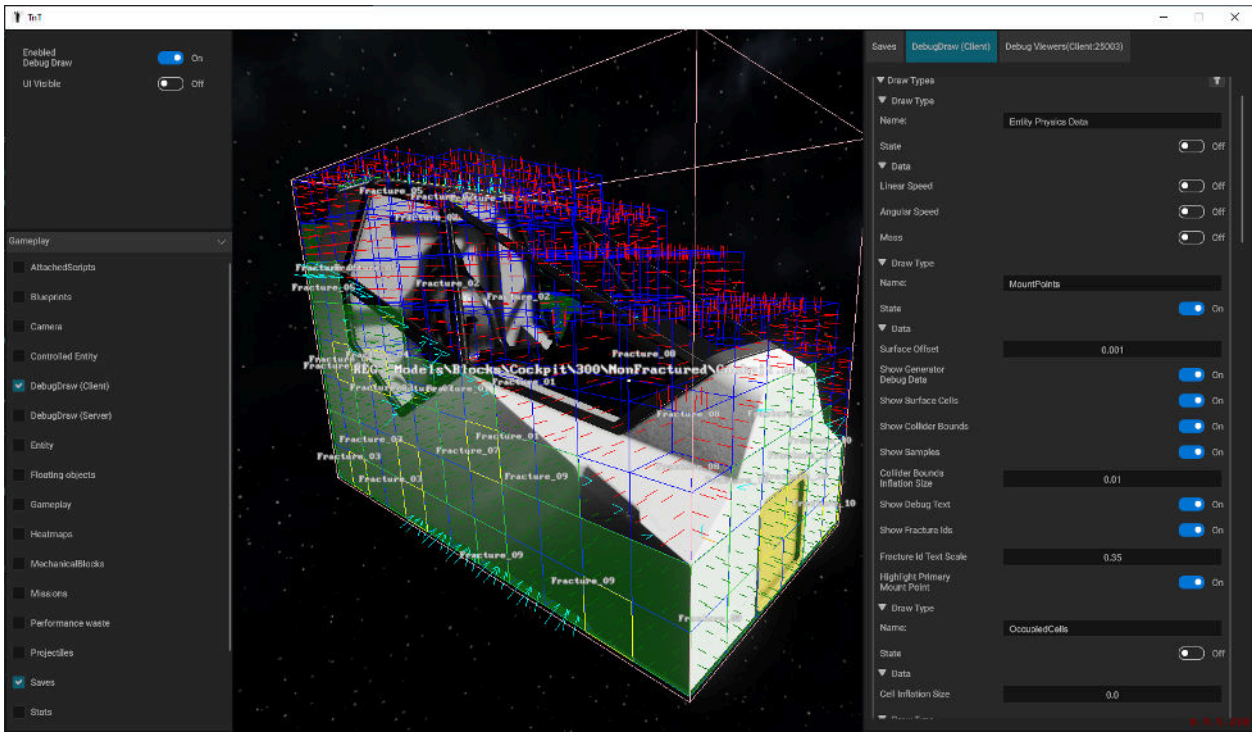
In ***F12 > Gameplay > DebugDraw (Client) > MountPoints*** you can enable various debug draw options for mount points:



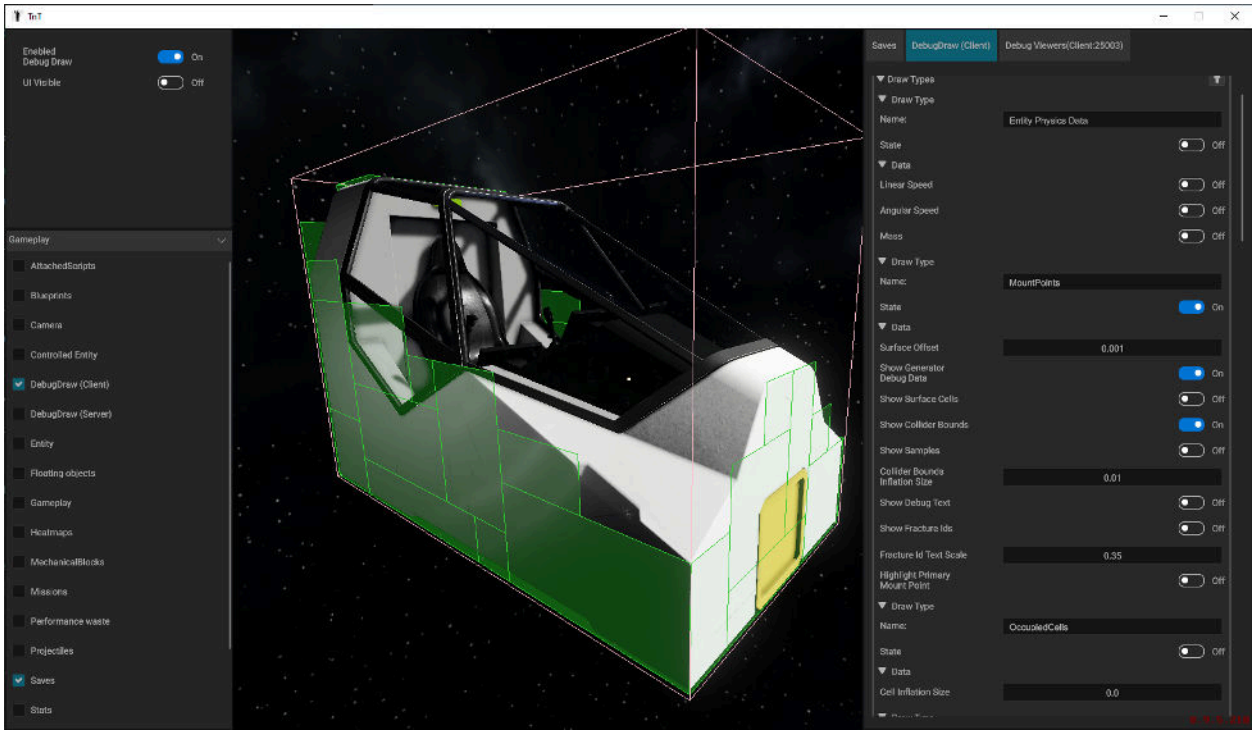
By default, enabling MountPoint debug display will show the following data:

- Green cells represent individual mount point groups
- Yellow squares highlight the primary cell assigned to that mount point group
- Surface Offset lets you control how far from the mesh surface to render the debug geometry

When running the client with the ***-debugMountPoints*** command line, additional debug information is generated and can be displayed after enabling **Show Generator Debug Data**.

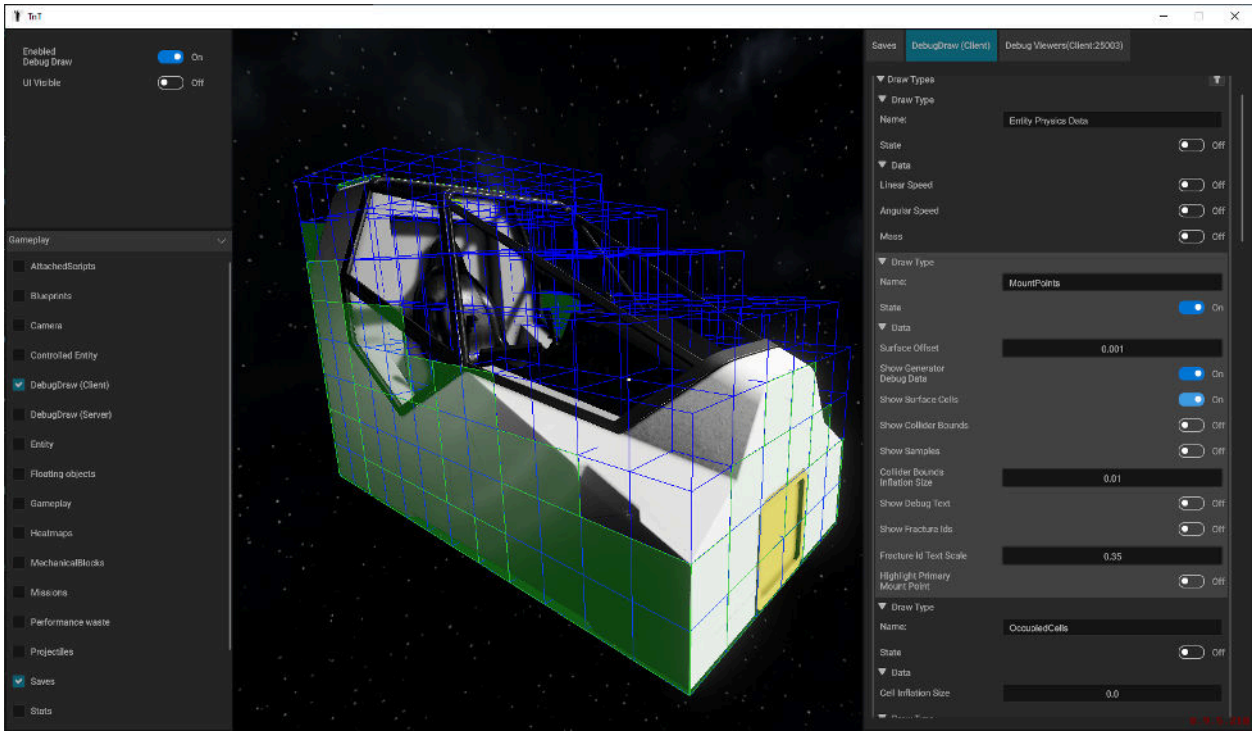


Show Collider Bounds will display a pink bounding box of the precise collider used as input by the mount points generator

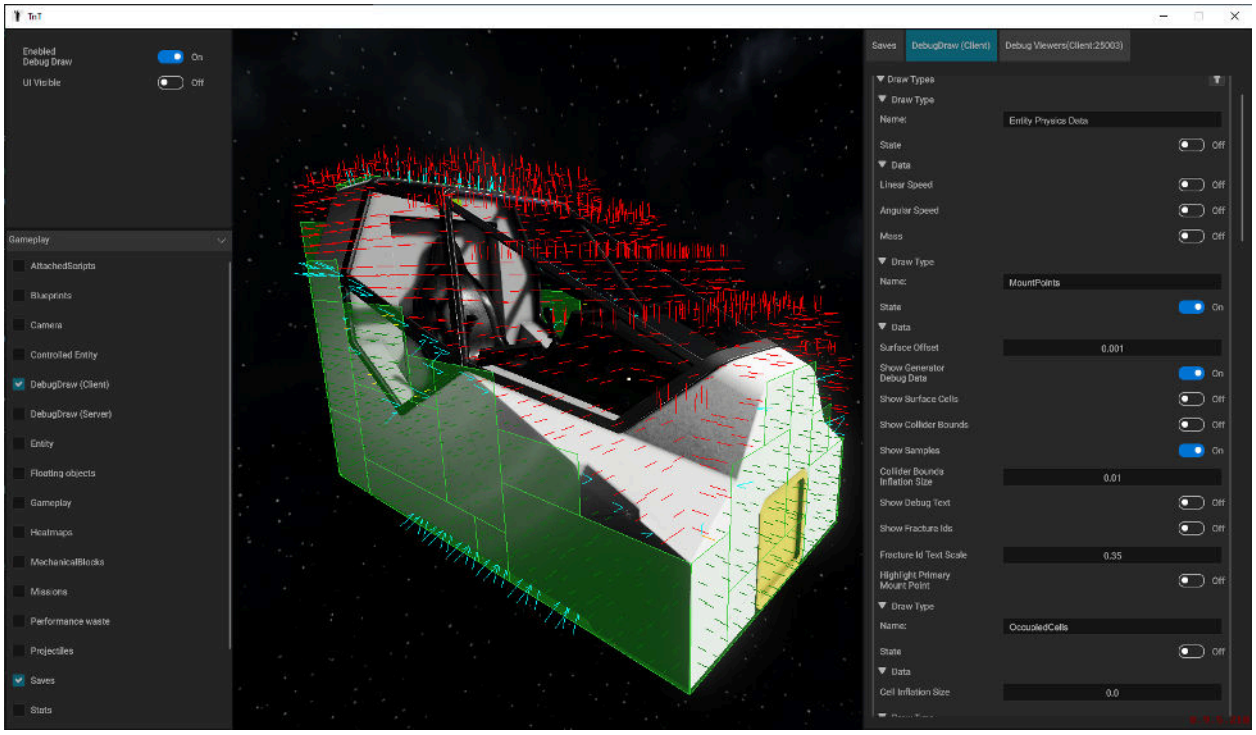


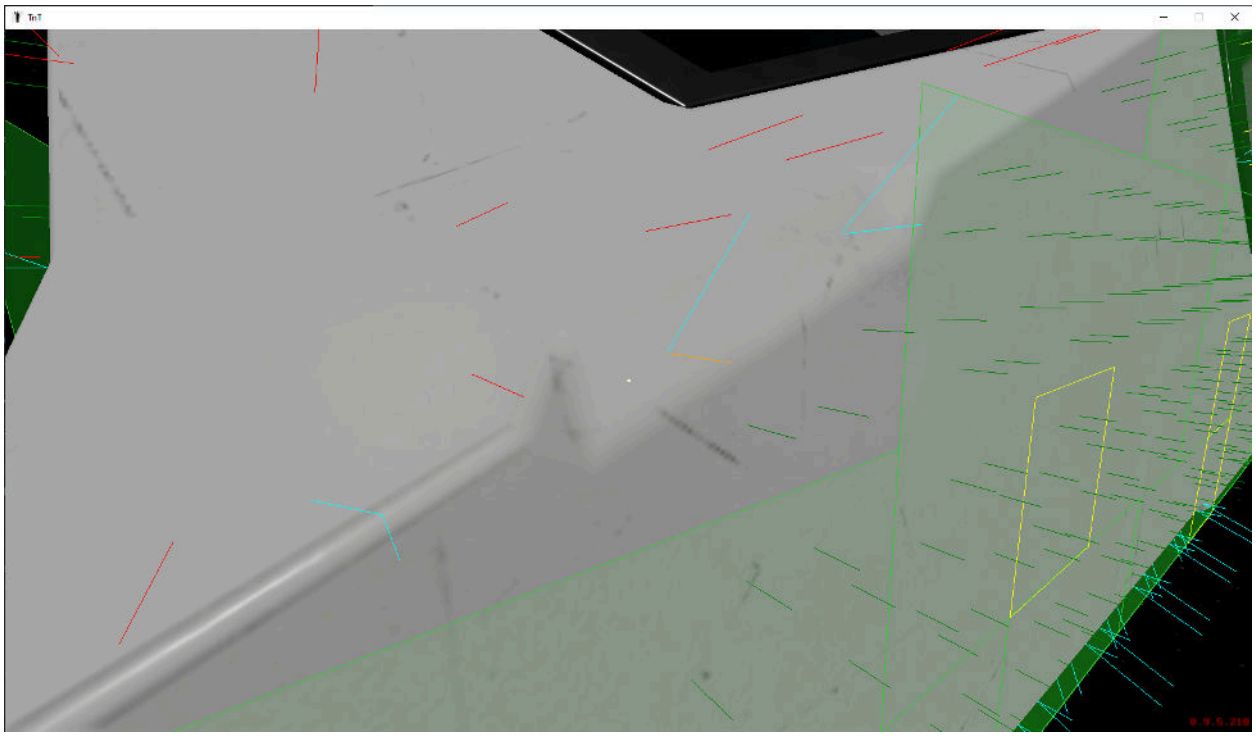
T

Show Surface Cells will show the selected surface cells in blue



Show Samples will display each individual raycast sample performed by the mount point generator to identify eligible mount point areas on the block model

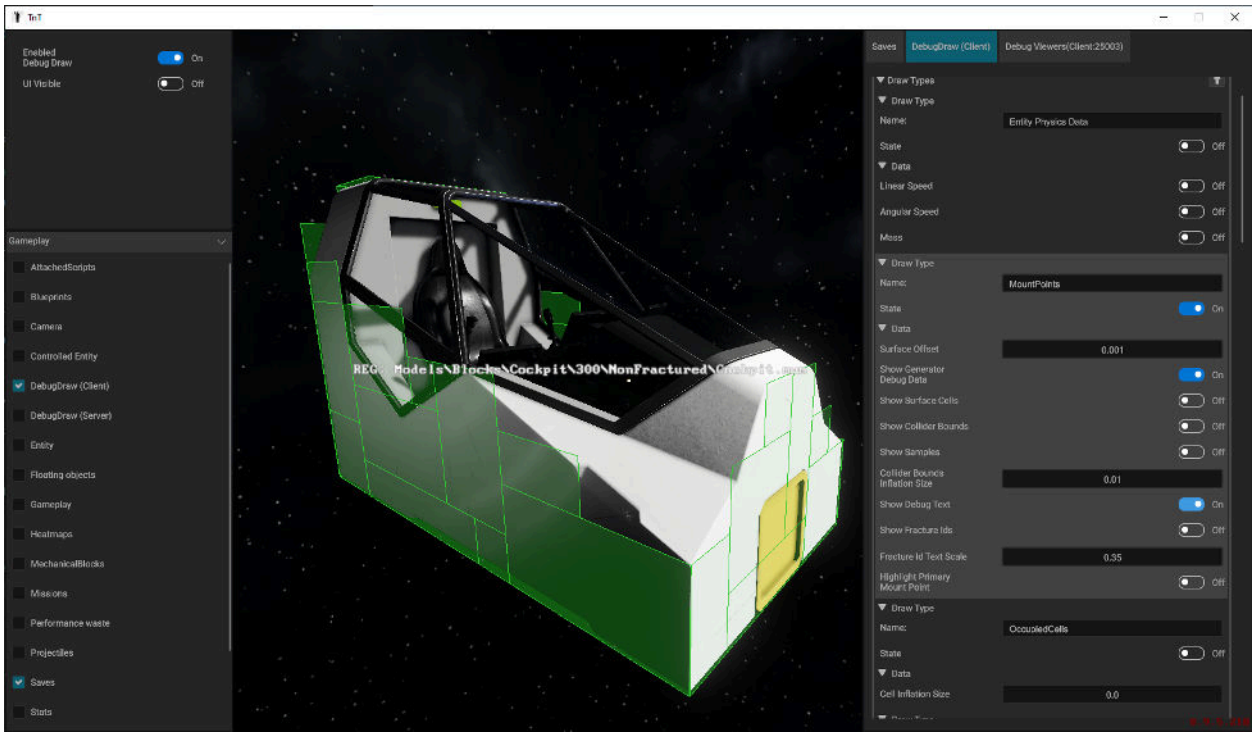




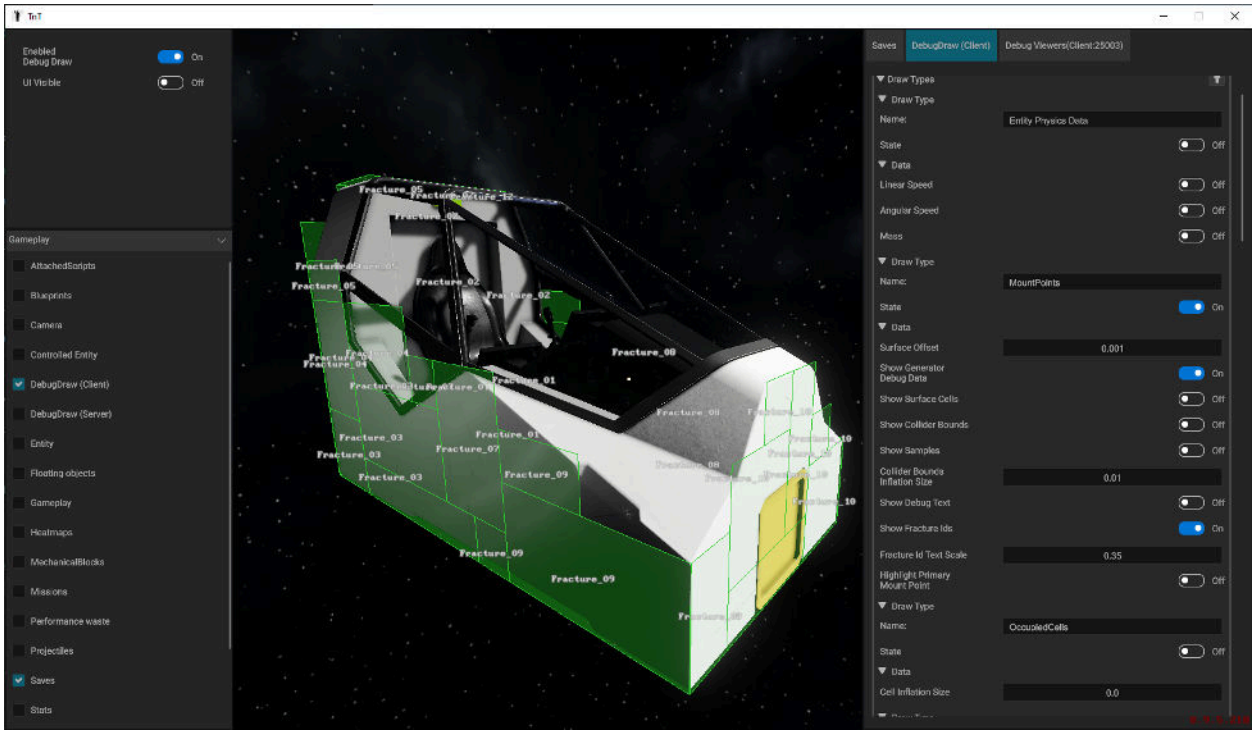
- **Red** segments show a sample that missed (no geometry was hit)
- **Green** segments show a valid sample hit
- **Cyan** segments show sample hits with invalid normal (surface not flat). The longer segment shows the actual normal found at sample hit location.

- **Orange** segments shows invalid hits with different depth/distance than the other samples

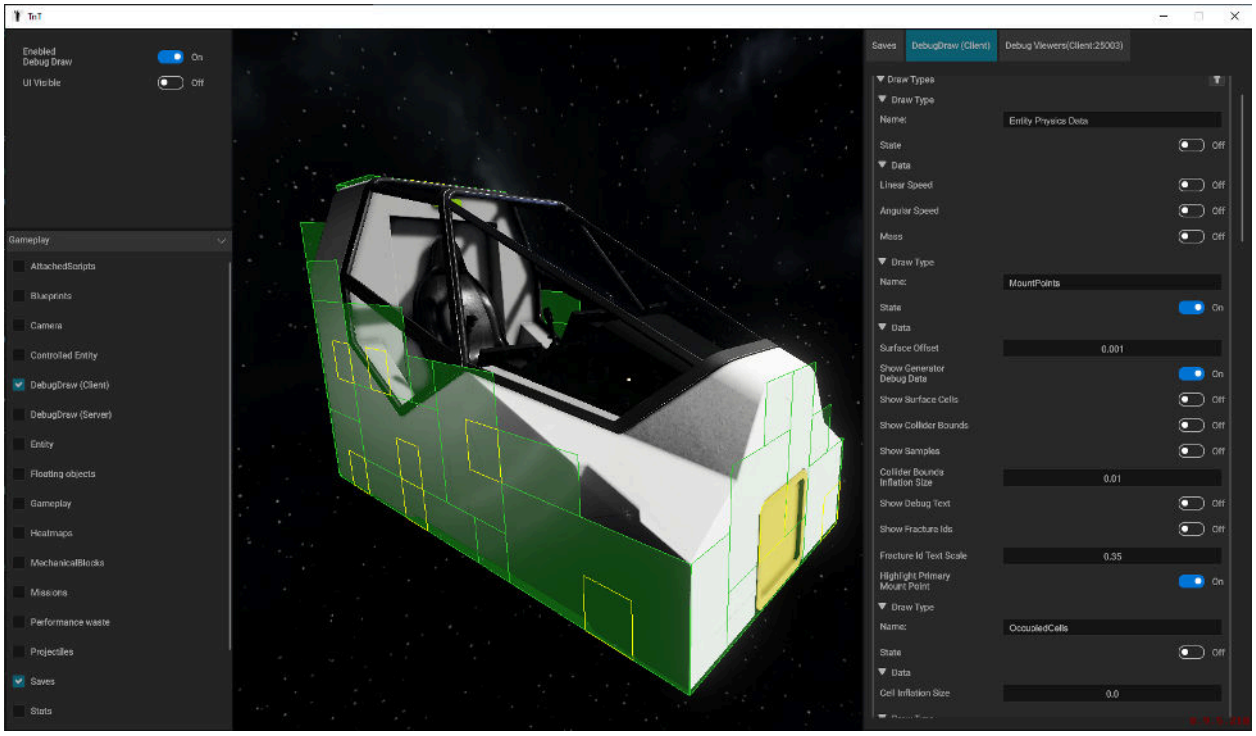
Show Debug Text will display the debug text associated with this MP generator output. This will currently show the name of the model used as input for the MP generator.



Show Fracture Ids will display the name of the closest fracture associated to each individual mount points group.



Highlight Primary Mount Point will show a yellow outline around the cell selected as the primary mount point for each mount point group.



Reference Prefab Selection

Several block prefabs can share the same CubeGridDefinition. An important step of the system is to determine which of these prefabs to pick as the “reference” one, which will then be used to determine the model resource to use as input for the generator. All other prefabs will also share that generated data.

The current selection logic will favor a prefabs in this order of priority:

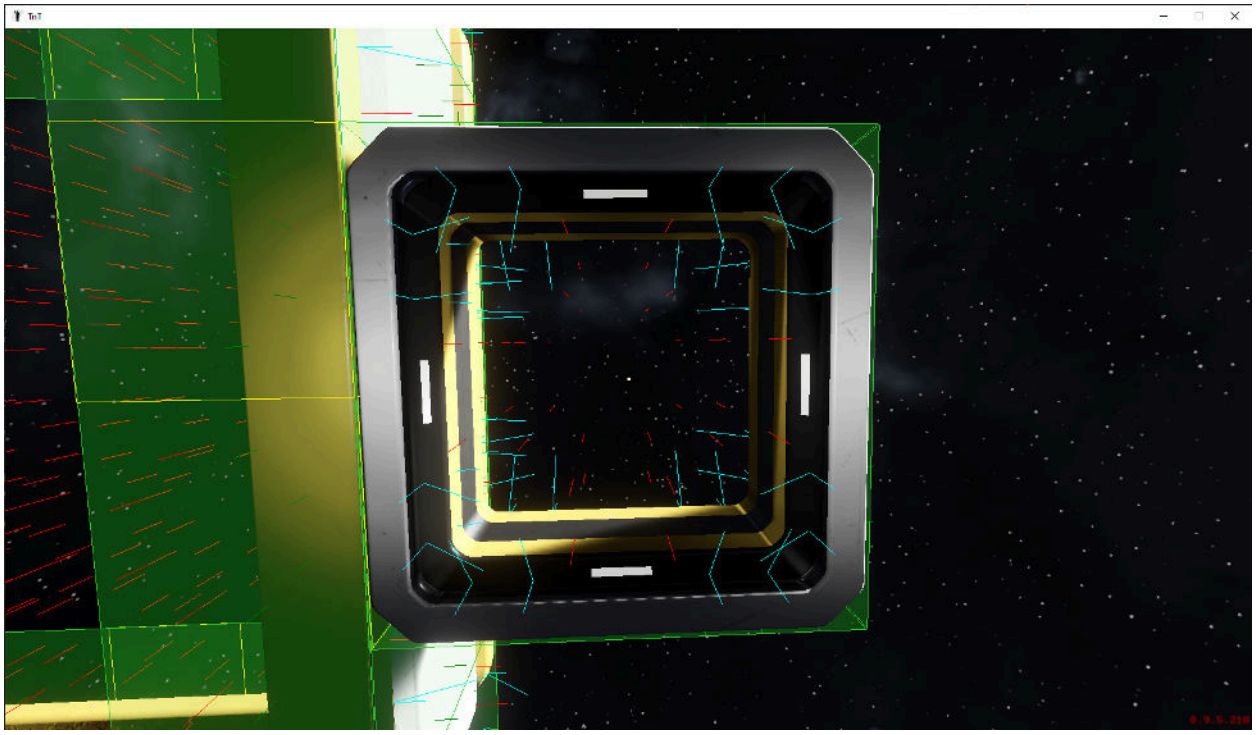
- prefab with armor block component
- prefab with non-breakable component (e.g. pick regular block prefab over its scaffolding version)
- any other block prefab

Details of that selection process can be found in the logs when running with -logOccupancy command line toggle:

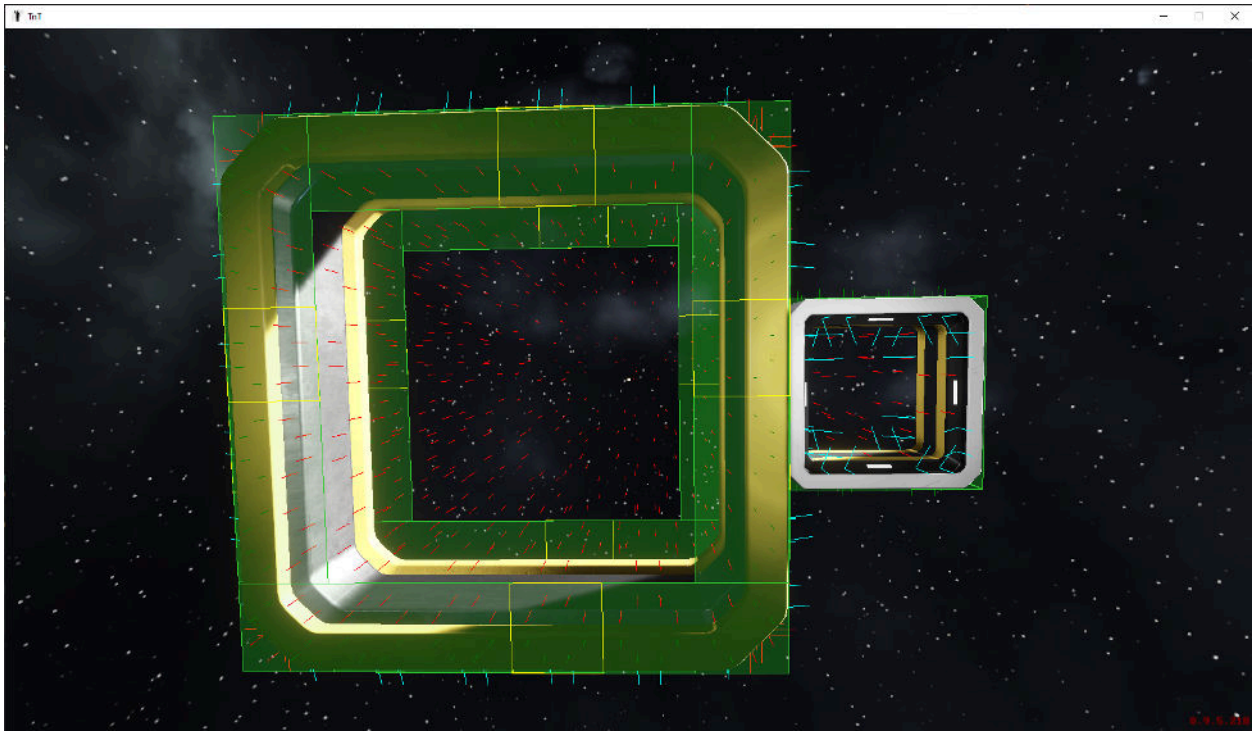
```
2023-11-15 16:26:33.560 - Thread:   1 ->  Info: GeneratedBlockDataLoader >
def=64ce180b-5fd7-40f0-a8fc-50ae139394bb > prefab=89f76714-c17d-44d5-ba06-60a71991d5e4
> Def = Data\Blocks\Windows\CurvedInwardTriangle\50\WindowCurvedInwardTriangle50_CubeGridDefinition.def
> Prefab =
Data\Blocks\Windows\CurvedInwardTriangle\50\LegacyPrefab\WindowCurvedInwardTriangle50_Client.def
> Model =
Models\Blocks\Windows\WindowCurvedInwardTriangle\50\NonFractured\WindowCurvedInwardTriangle.mwm
> Bounds = {Min:[X:0, Y:0, Z:0] Max:[X:1, Y:1, Z:1]}
> Occupancy = 3 > {Min:[X:0, Y:1, Z:0] Max:[X:0, Y:1, Z:1]}, {Min:[X:0, Y:0, Z:1] Max:[X:0, Y:0, Z:1]},
{Min:[X:1, Y:1, Z:1] Max:[X:1, Y:1, Z:1]}
> Prefab Candidates = 4
   #0 = 89f76714-c17d-44d5-ba06-60a71991d5e4 :
Data\Blocks\Windows\CurvedInwardTriangle\50\LegacyPrefab\WindowCurvedInwardTriangle50_Client.def
   #1 = 6e369441-2636-48ac-9f0f-fc99701c2c0b : DefinitionFlags:Abstract|SkipValidation
   #2 = 1079dd22-04d0-442a-940e-2e818d4f74d8 : DefinitionFlags:Abstract|SkipValidation
   #3 = 7c4a1cbb-a919-410c-8cb4-fbfb0cf9f81b :
Data\Blocks\Windows\CurvedInwardTriangle\50\LegacyPrefab\WindowCurvedInwardTriangle50_Server.def
```

Providing Hints To The Generator

While the generator should output correct results for most blocks, there are certain cases where additional manual hints need to be provided on the model by artists to help the generator produce the desired output.



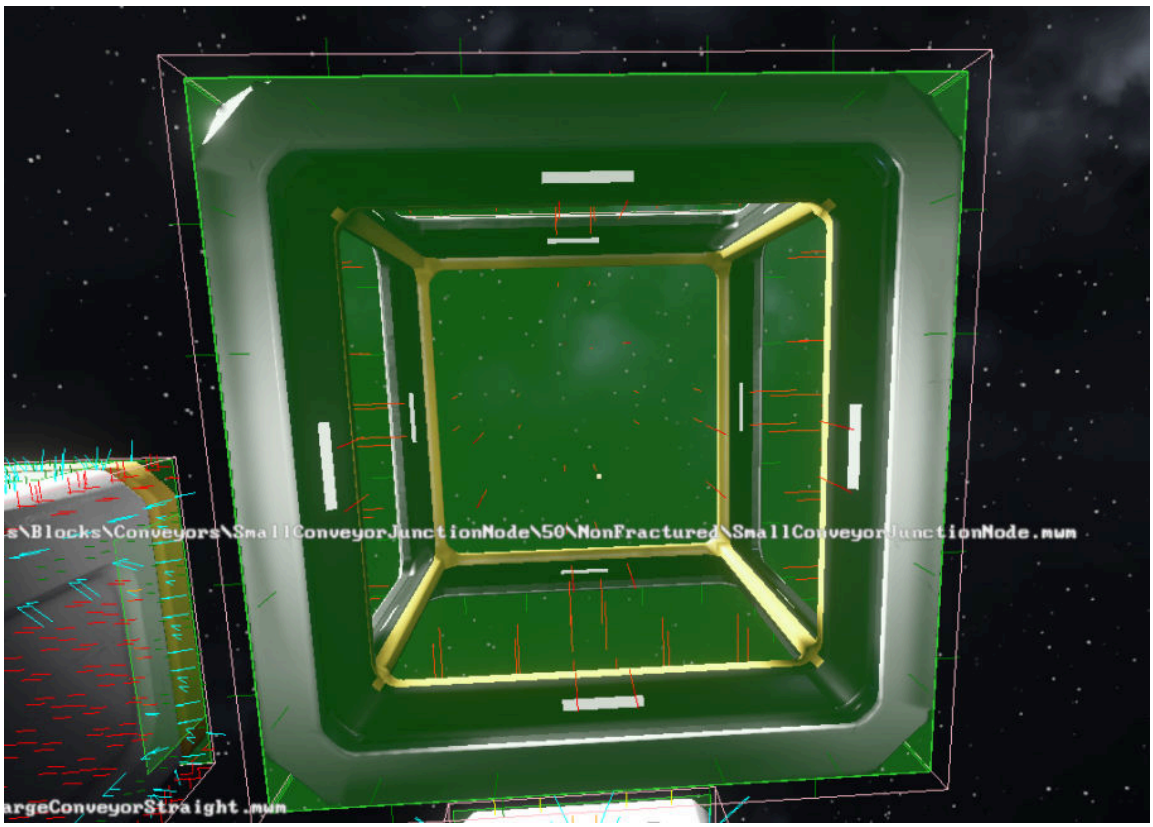
As you can see on the screenshot above, the small conveyor block on the right has a thin rim that is missed by the sampling pattern of the generator. By default, this results in this face having no mount points, making this block unusable for its intended purpose.

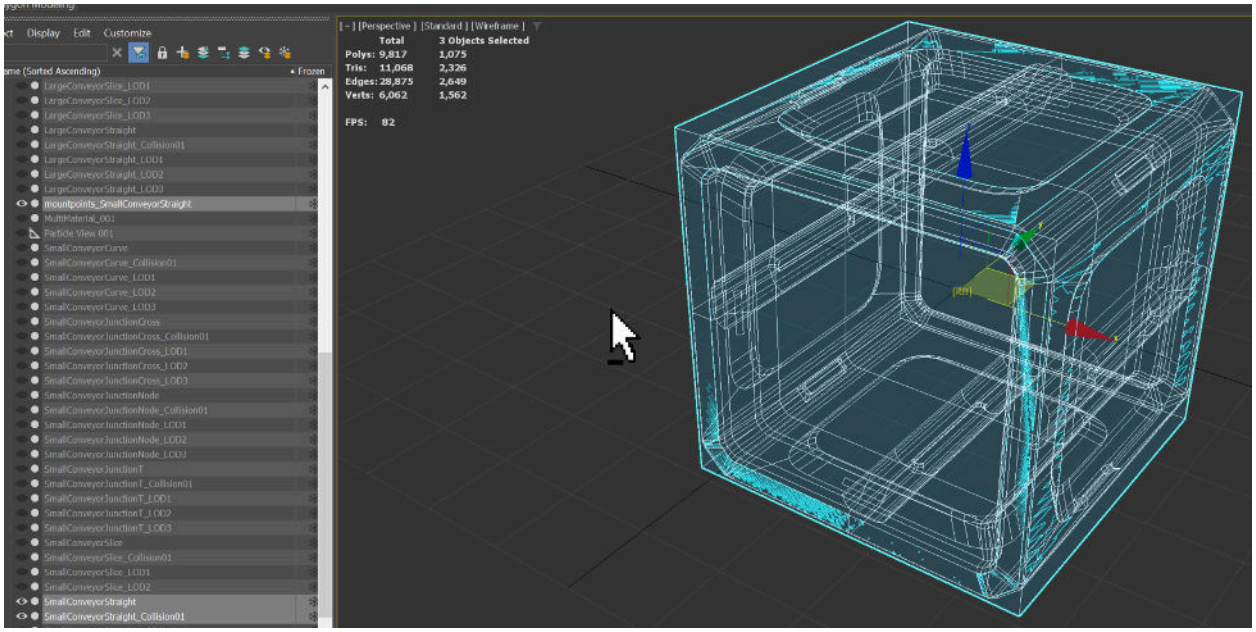


On the other hand, the bigger version of the block has no such problem because the rim is large enough to be hit by multiple valid samples, and causing the MP generator to correctly place mount points along the rim at these locations.

To help with this, artists can place special colliders used exclusively by the MP generator to affect the output. These special collider names should be prefixed “***mountpoints***” so they are correctly extracted by the system and do not affect the block physics.

Adding these special colliders can be used to make certain areas of the block model seen as flat/valid by the mount points generator. For instance, if we were to add a mountpoint quad collider on the front face of the small block, we would get the following output (single 2x2 mount group on this face)




Display
Edit
Customize
(Sorted Ascending)
Frozen
[-] [Perspective] [Standard] [Wireframe]
Total
3 Objects Selected
Polys: 9,817
1,075
Tris: 11,068
2,326
Edges: 28,875
2,649
Verts: 6,062
1,562
FPS: 82