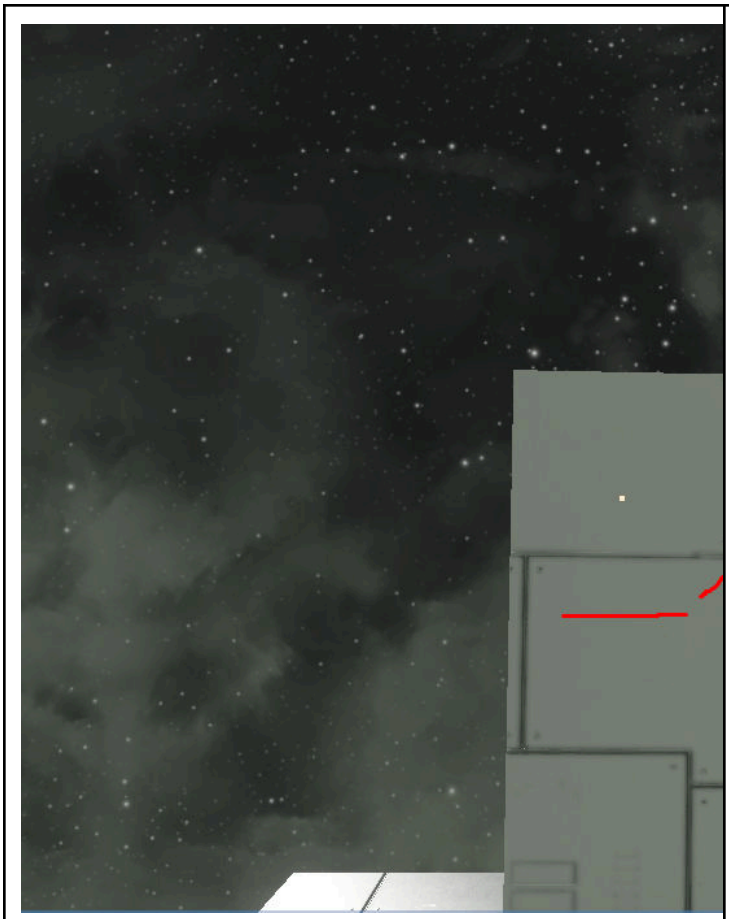
Destruction Manual

Deformation Patterns

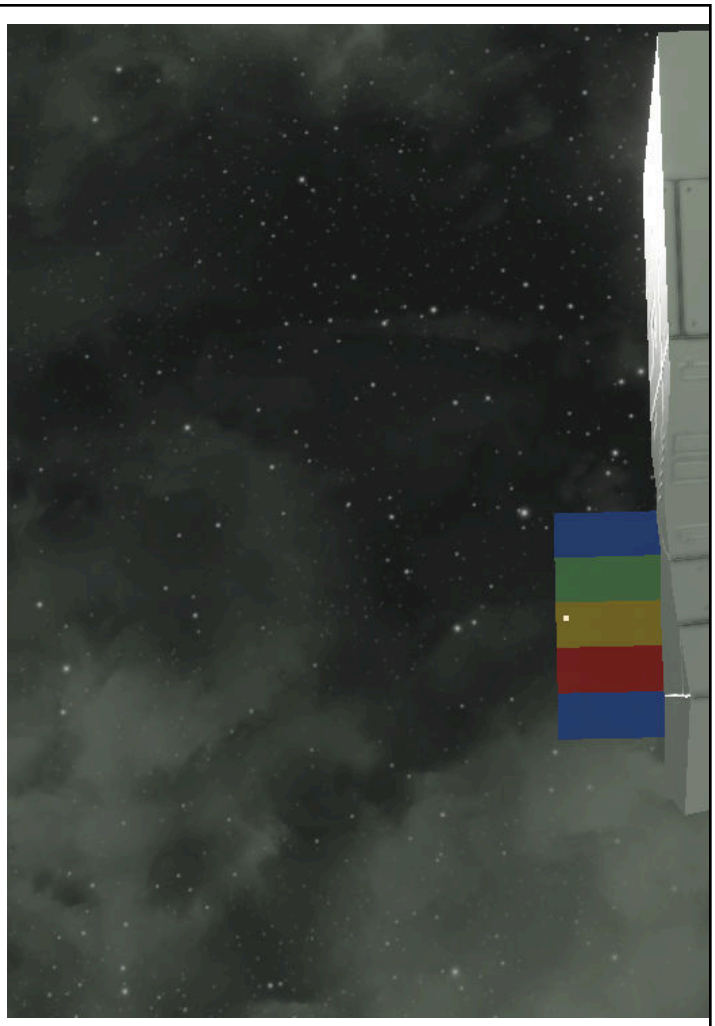
Corner/Edges Deformation

This pattern was created to be as close to the deformations in SE as possible within the new bone storage structure.

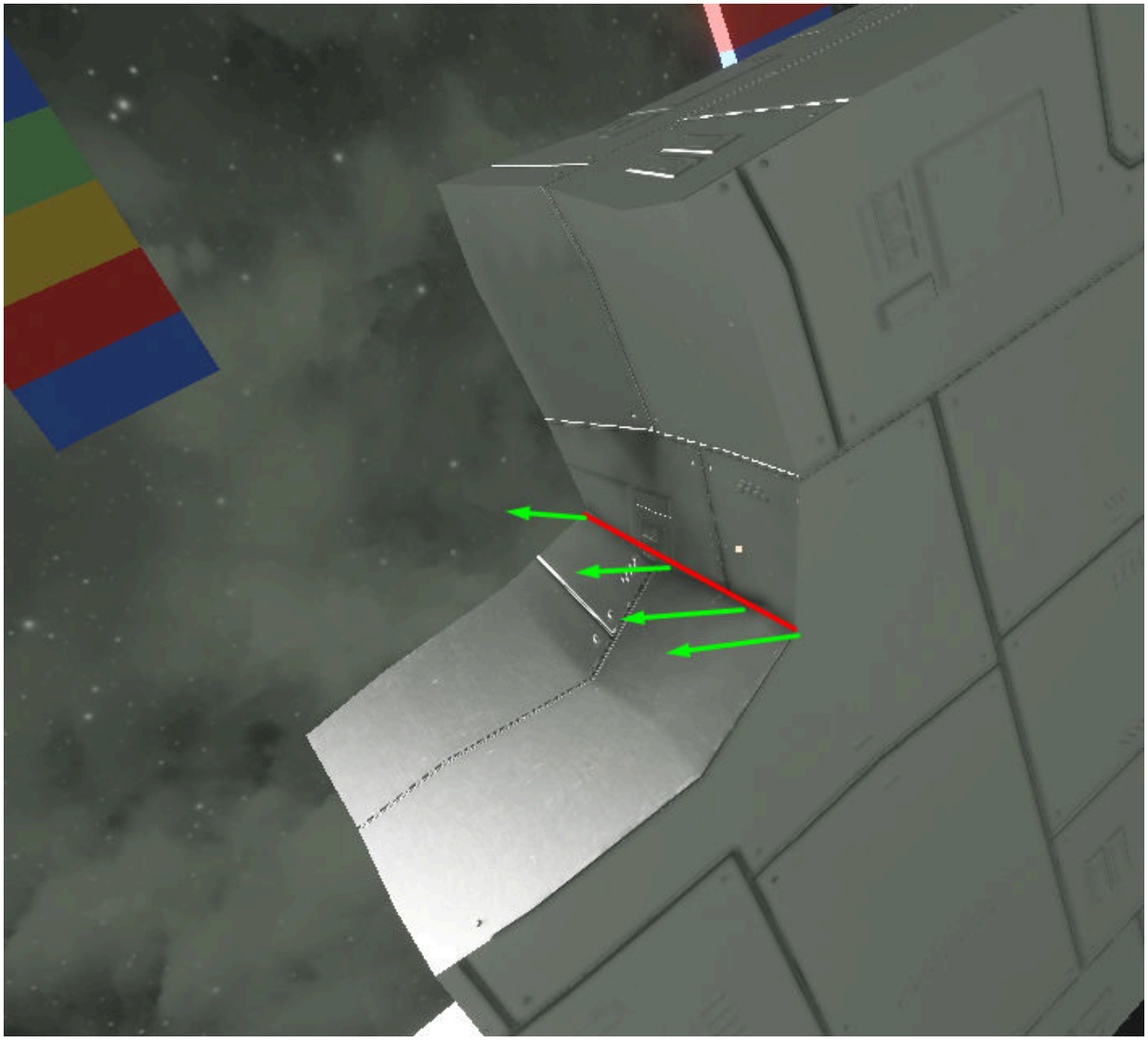
The general idea is that upon block destruction, each one of the 12 edges of the cube is checked for whether there are 3 blocks around the edge that form a corner. If there are, the whole edge gets pulled inwards in the direction from the edge towards the center of the block.



When there are 3 blocks around the edge that form a corner, the corner bone gets pulled inwards.



When there aren't 3 blocks around the edge that form a corner, the corner/edge deformation algorithm is not applied.

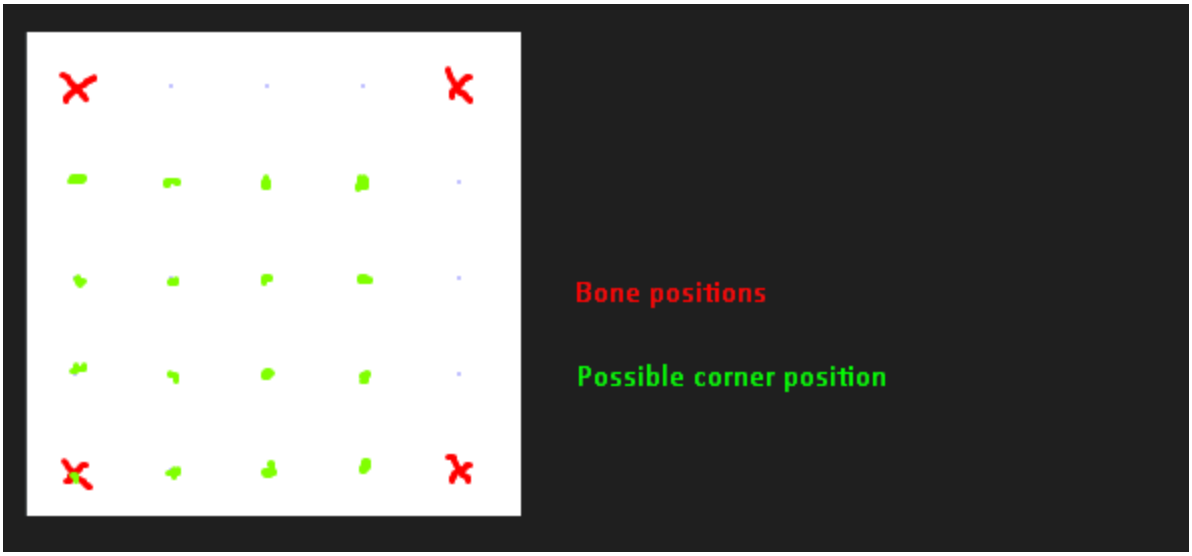


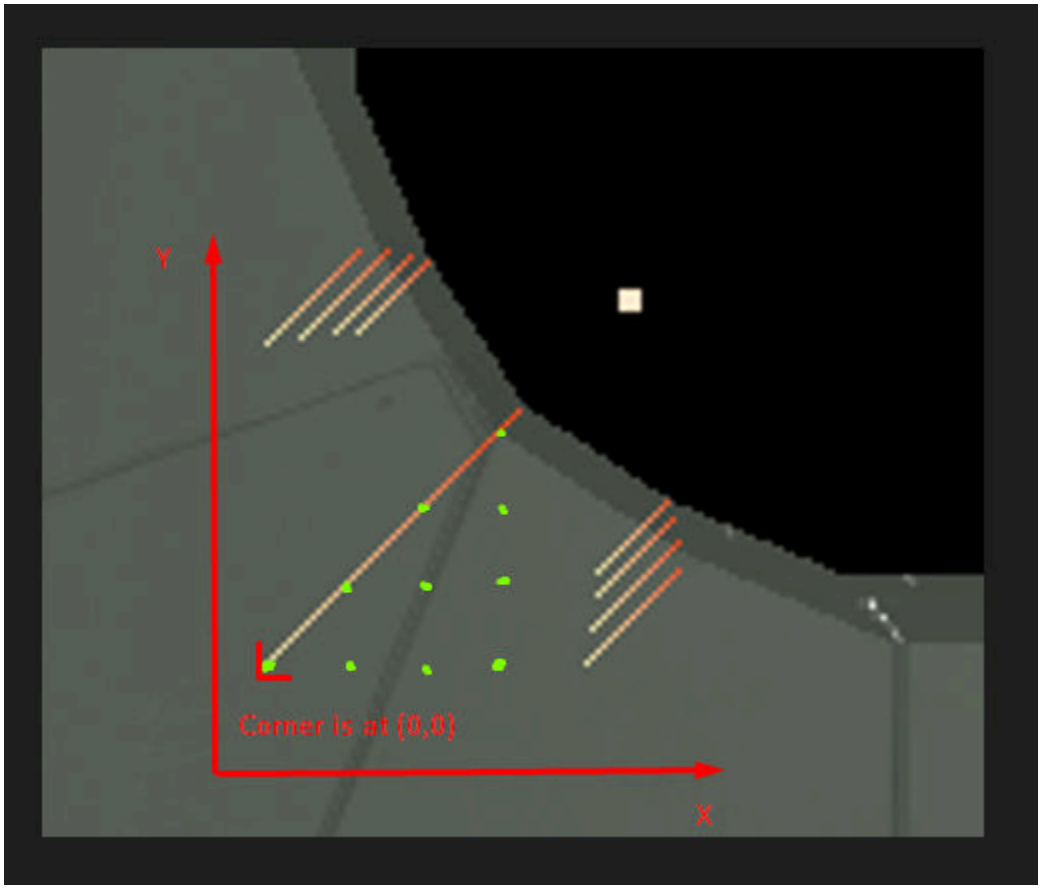
The bones get pulled inwards along the whole edge of the bone storage

This check happens around each of the 12 edges that the cube has.

Block Corner vs Bone Storage

In TnT we have bones on a grid with 1 meter spacing. The smallest cell of the grid is 0.25 cm. If you imagine a block corner in a 2D space, it leaves us with 16 points in which the corner can be situated in:

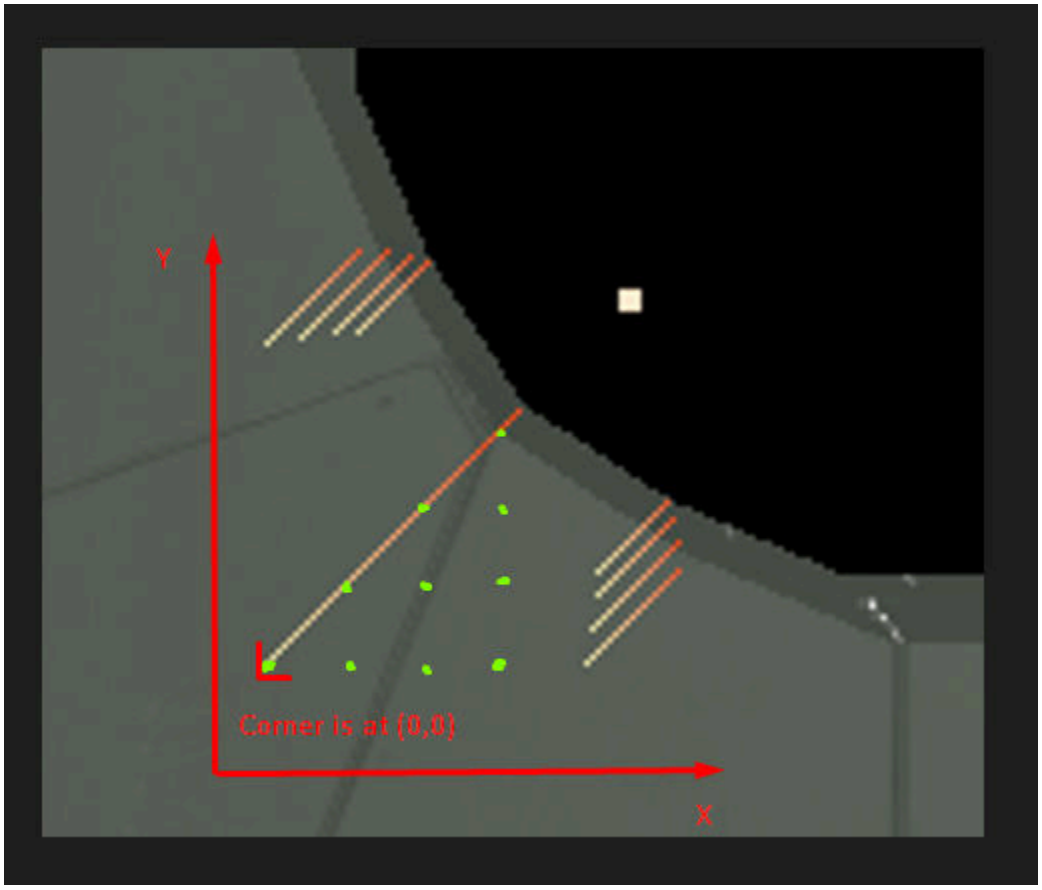




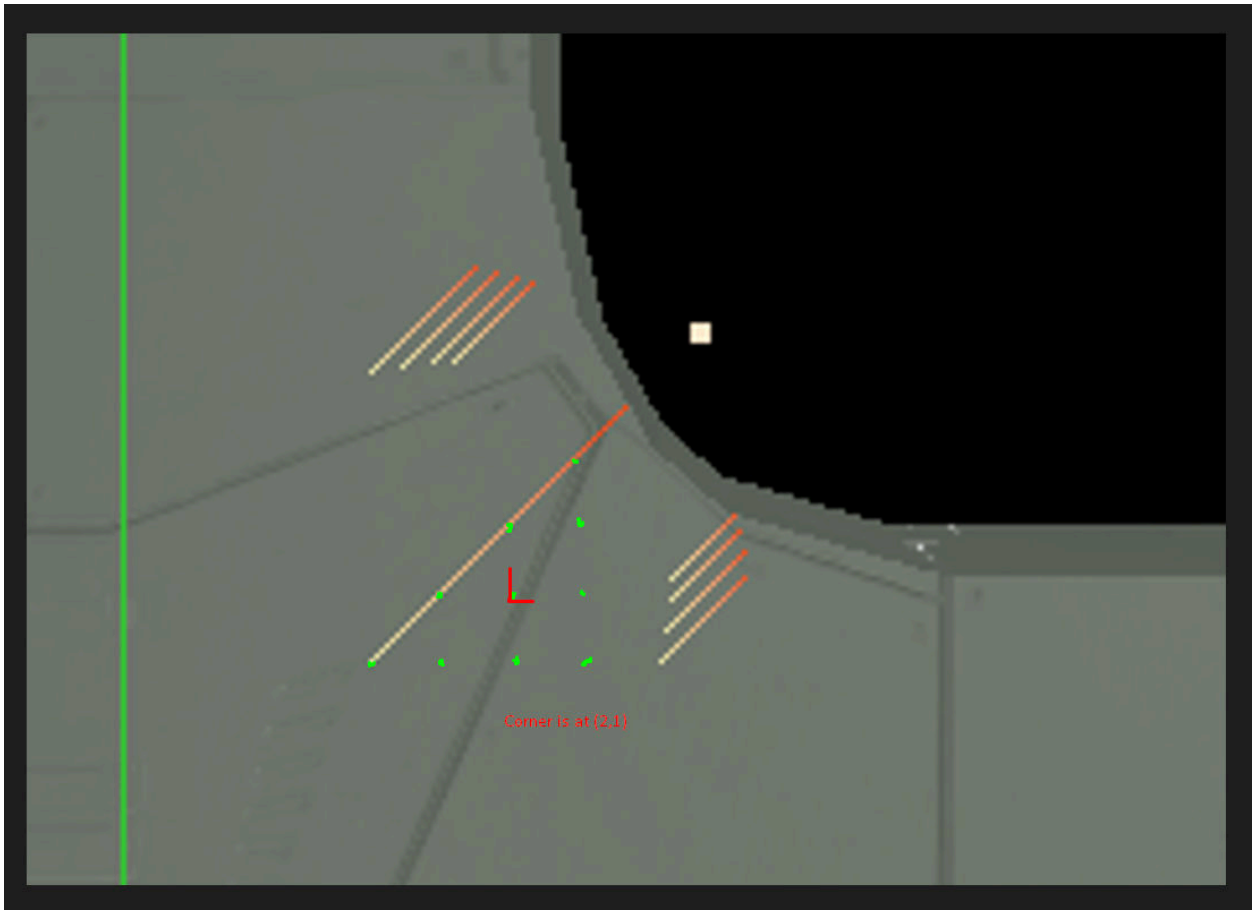
Where the corner will be depends on the size of blocks around it. E.g. if you first place a 0.25cm block followed by a 2.5m block, the bone storage will align differently then if you placed just the 2.5m block.

Game Example

In the following example, the left-bottom block corner was perfectly aligned with the bone storage, which means its corner is in position (0, 0):



If we add a “padding” to the grid, i.e. we first build a grid with three 0.25cm blocks in the x axis and two in the y axis, the bone storage gets shifted and the corner of the removed block will land in position (2, 1):



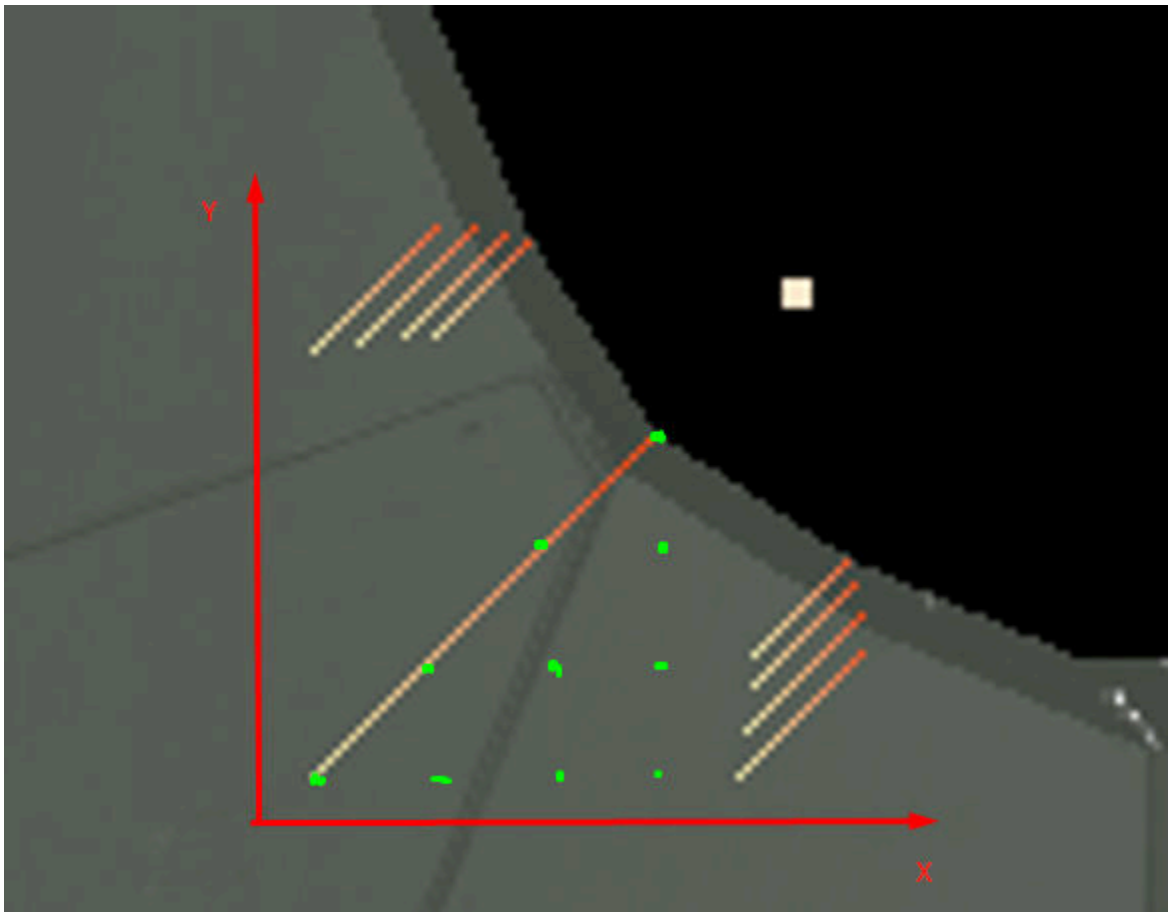
Configuration

To adjust how much a corner gets pulled inwards in different scenarios, the values in *CubeGridDeformationConfiguration* can be adjusted. Currently, a deformation table is exposed that contains ten 2D offsets that represent the position of the corner within the bone storage. Depending on the position of the corner it is necessary to deform the bones with a different intensity to achieve similar-looking results for each position.

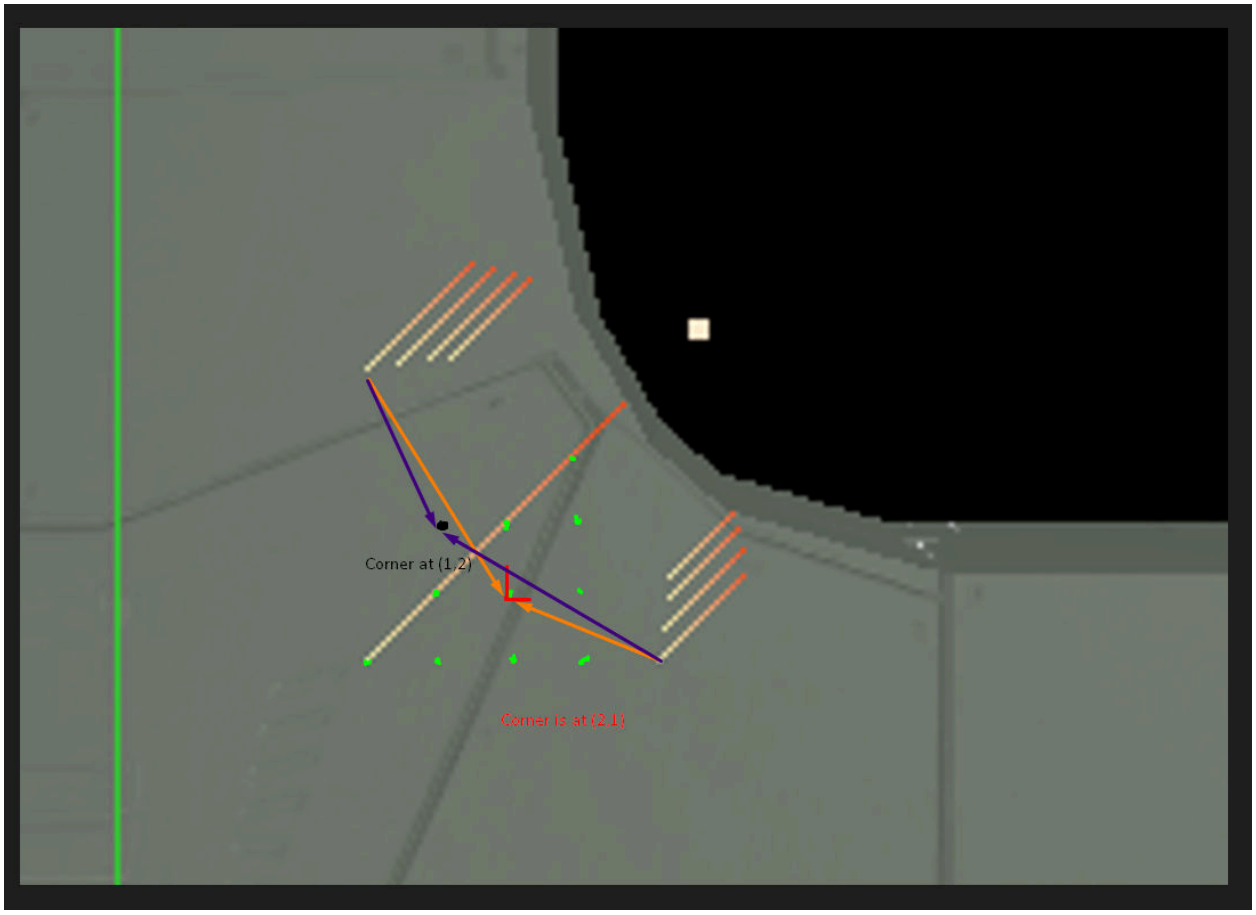
Deformation Table

This section describes some concrete examples of how the values in the table influence the look of the deformed corner. For the rationale behind this approach please refer to the previous chapter - Block Corner vs Bone Storage.

The table should always contain 10 offsets that represent these points:



While there technically are 16 points in which the corner can be, 6 of those will just be a mirrored version, e.g. (1,2) should contain the same values as (2,1) so that the corners look consistent.

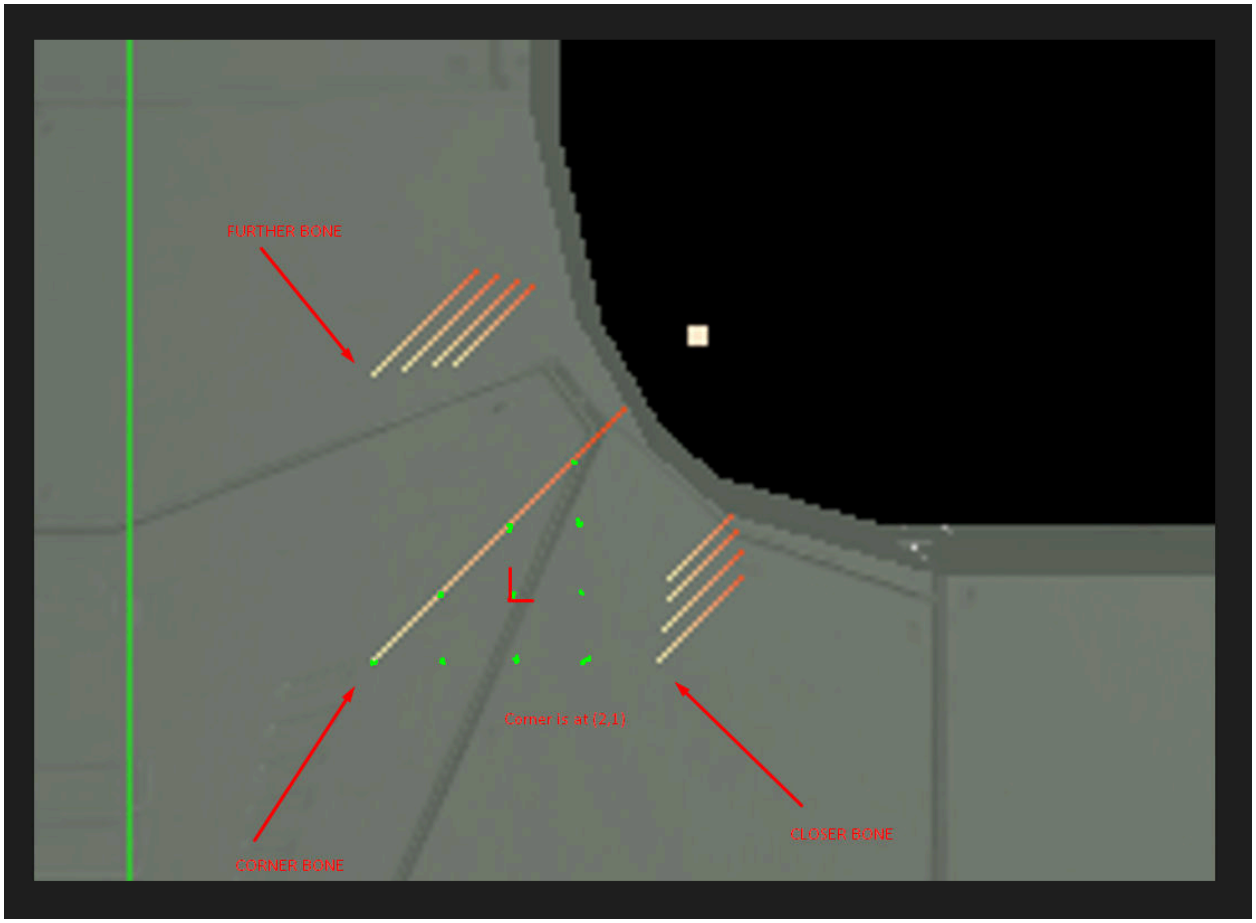


Mirroring

For each offset a *CornerDeformationIntensities* struct is defined. This struct describes how intensely the specific bones get deformed (0-1, 0 = no deformation, 1 = max deformation):

- CornerBoneIntensity - the bone that represents the corner of the block transformed to the block storage.
- CloserBoneIntensity, FurtherBoneIntensity - the bone that is closer/further away from the block corner position. For positions that have the same value for X and Y the values should be the same.

Game Example



The picture above shows the following values for offset (2,1):

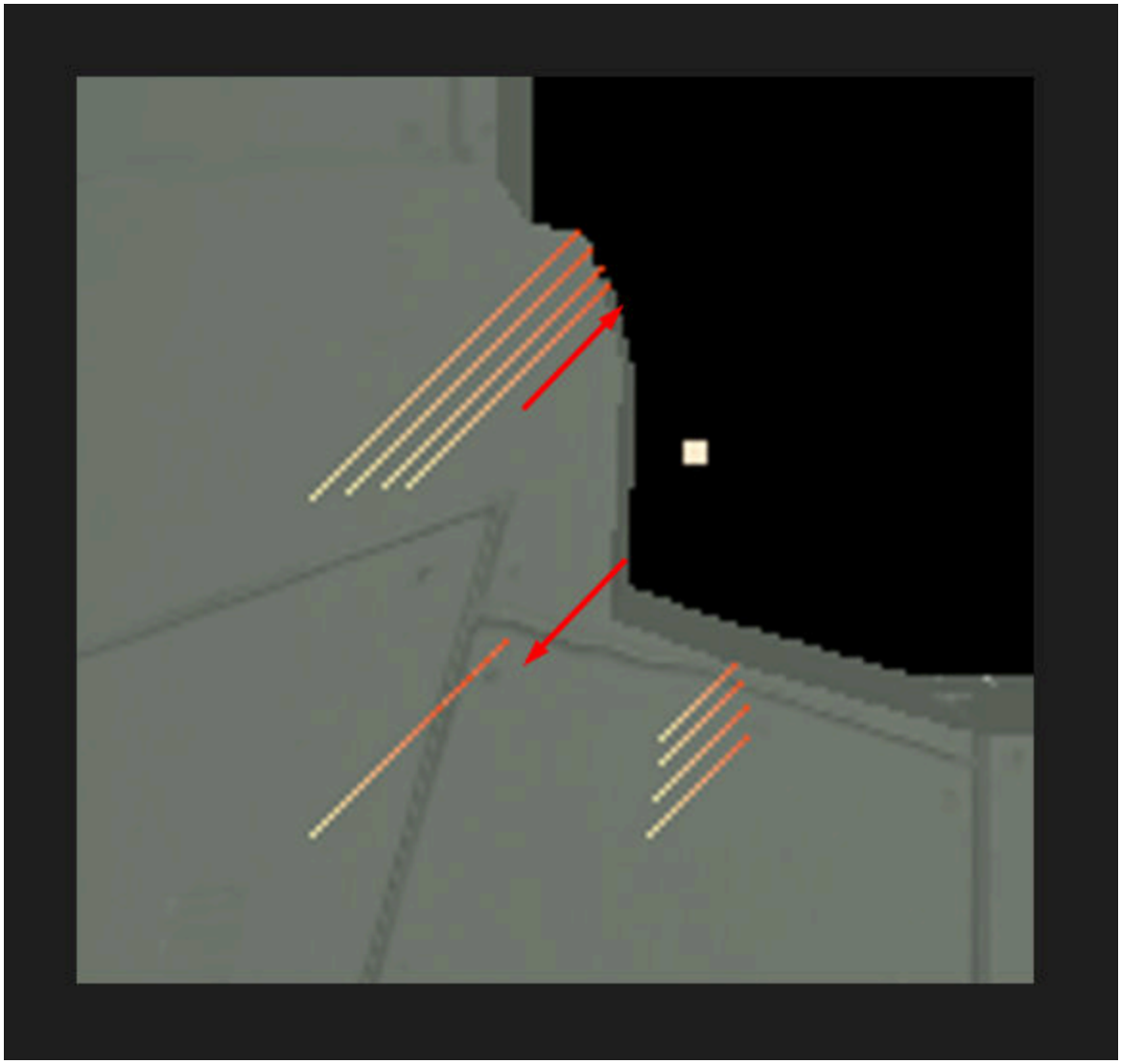
```
{ new Vector2(x: 1, y: 2), new CornerDeformationIntensities(0.8f, closerBoneIntensity: 0.3f, furtherBoneIntensity: 0.375f) },
```

Tuning

Let's say we decide that we want lower corner intensity and higher further bone intensity. We could change the values like this:

```
{ new Vector2(x: 1, y: 2), new CornerDeformationIntensities(0.4f, closerBoneIntensity: 0.3f, furtherBoneIntensity: 0.8f) },
```

Which would result in the following changes:

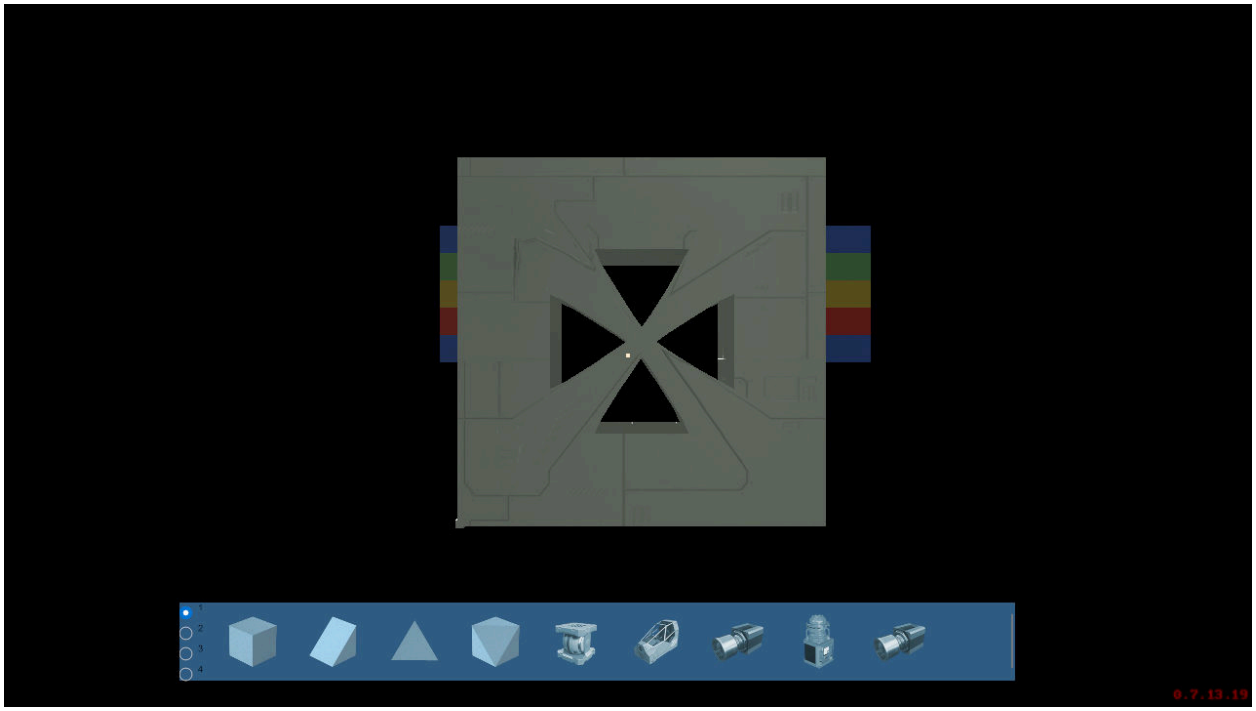


Calibration

The system was calibrated for 2.5 meter blocks, smaller blocks will get proportionally lower intensities based on their size. For example, the value 1 in the deformation table means that a 2.5m destroyed block will do maximal deformation possible. A 1m block will instead do 1 / 2.5 = 0.4 of the maximum deformation, 0.5m block will do 0.2, and 0.25m block will do 0.1. If this ever becomes problematic, we could expose some sort of multiplier that would specify how much should each block get deformed.

The maximal deformation possible is then translated based on values in *BoneDeformationConfiguration.MaxBoneOffset*. E.g. if *MaxBoneOffset* is set to 1 meter (current value at the time of writing this document), the maximal deformation will translate into 1 meter. This value could technically be adjusted too, but keep in mind that it will

probably require changes to all offsets in the deformation table as max intensity might suddenly be too much in some cases:

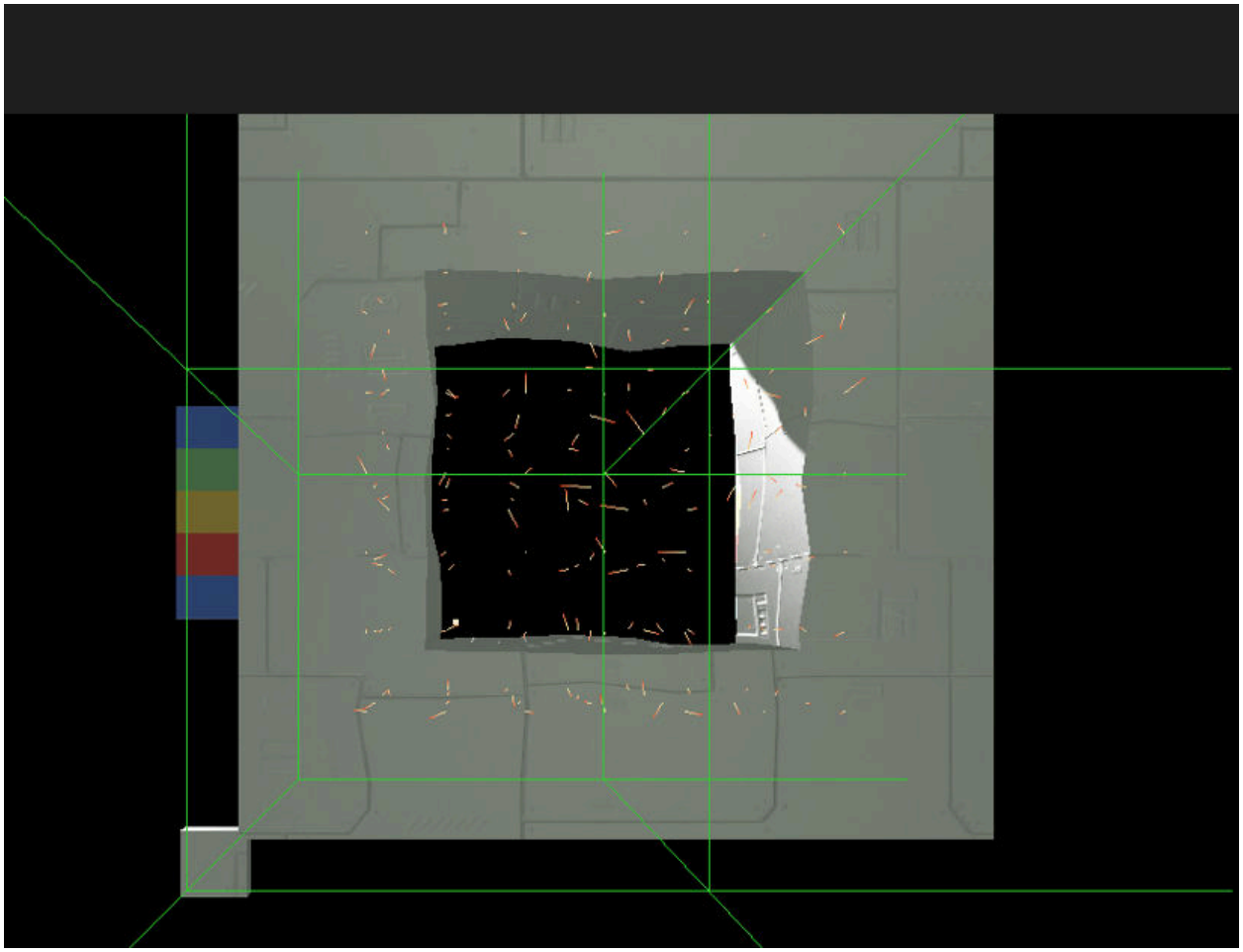


Deformation table (0,0) with MaxBoneOffset of 5 meters - not a result we want

So in case we find out that we cannot deform a corner far enough at the max intensity, we can increase the *MaxBoneOffset*, but it would mean that deformation intensities of corners in other offsets would need to be tuned down proportionally to compensate.

Randomization

In addition to the corner deformation pattern there is also seeded random pass that deforms bones in random direction with random intensity clamped within a predetermined range.



Currently, these values are not exposed in any configuration, but if necessary we can discuss how to expose them so that they can be tuned too.

Deformation Triggers

Here's a list of the possible triggers that can currently lead to deformations in TnT:

1. When a block is destroyed by a source of damage (not when it is removed or ground down), it deforms all bones around it based on a predetermined pattern with additional random deformations based on a seeded random.
2. When an exploding projectile explodes, it deforms all blocks within the damage radius. The deformation intensity decays with distance from the center of the explosion.
3. When a block is shot, it gets deformed in the direction of the projectile based on the projectile's deformation coefficient and the armor block's deformation resistance

4. When a block receives a light impact that is not strong enough to cause damage, it can also slightly deform based on the impact force, as a way to show physical wear & tear. The deformation amount is based on the block's deformation, the impact force and the MinCosmeticDeformationImpulse and MaxCosmeticDeformationImpulse properties of the GridImpactDamageSettingsConfiguration. The deformation will happen in the opposite direction of the normal at the point of impact (pushing into the block).