UE4 Chaos/Destruction R&D

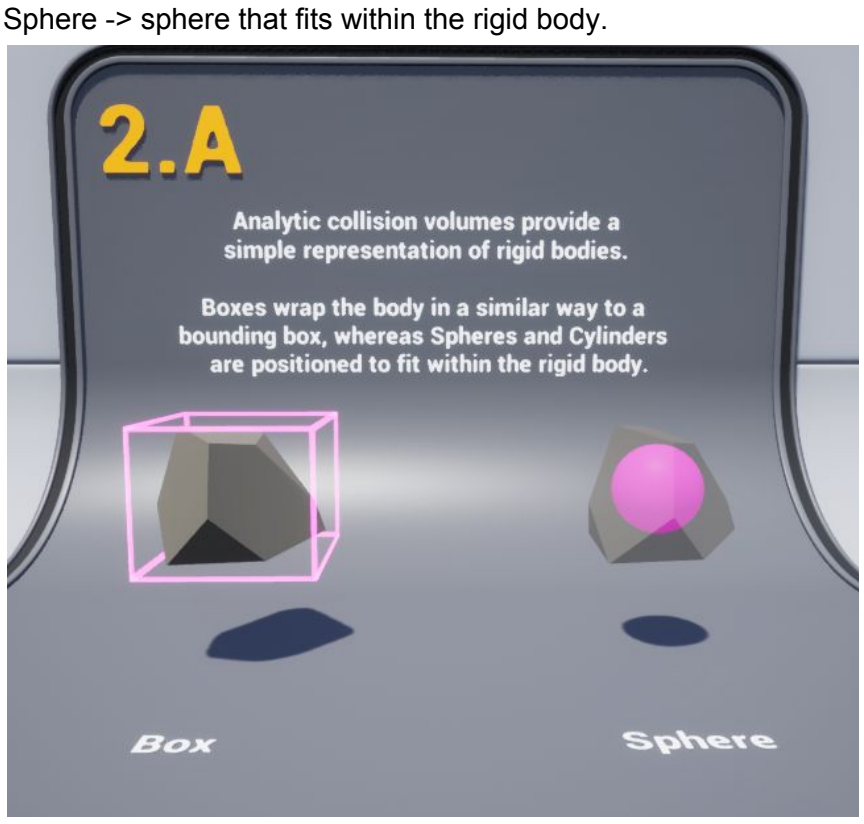
Overview

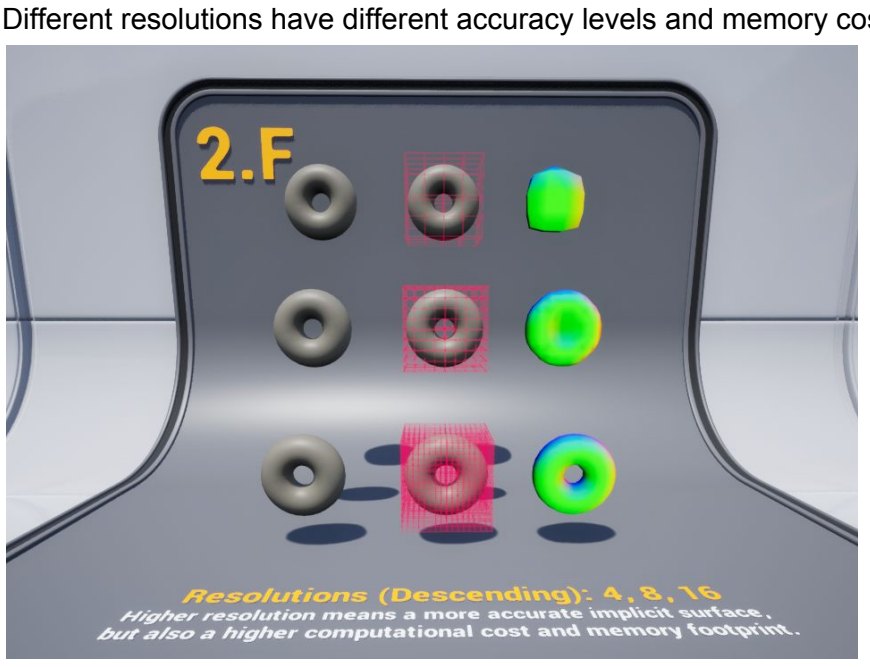
https://docs.unrealengine.com/en-US/Engine/Chaos/ChaosDestruction/ChaosDestructionOverview/index.html

UE4 Chaos is an in-editor system for creating fractured meshes and setting up rules for destruction. There are very similar pipelines using RayFire Plugin for Max and with RBD/Voronoi fracturing in Houdini, however this one is built into Unreal and can create complex/film-quality destruction in-editor. It uses the PhysX system to move/animate destruction and offers similar simulation options like constraints, keeping chunks together until X damage or time and other similar options used in pre-simulated workflows.

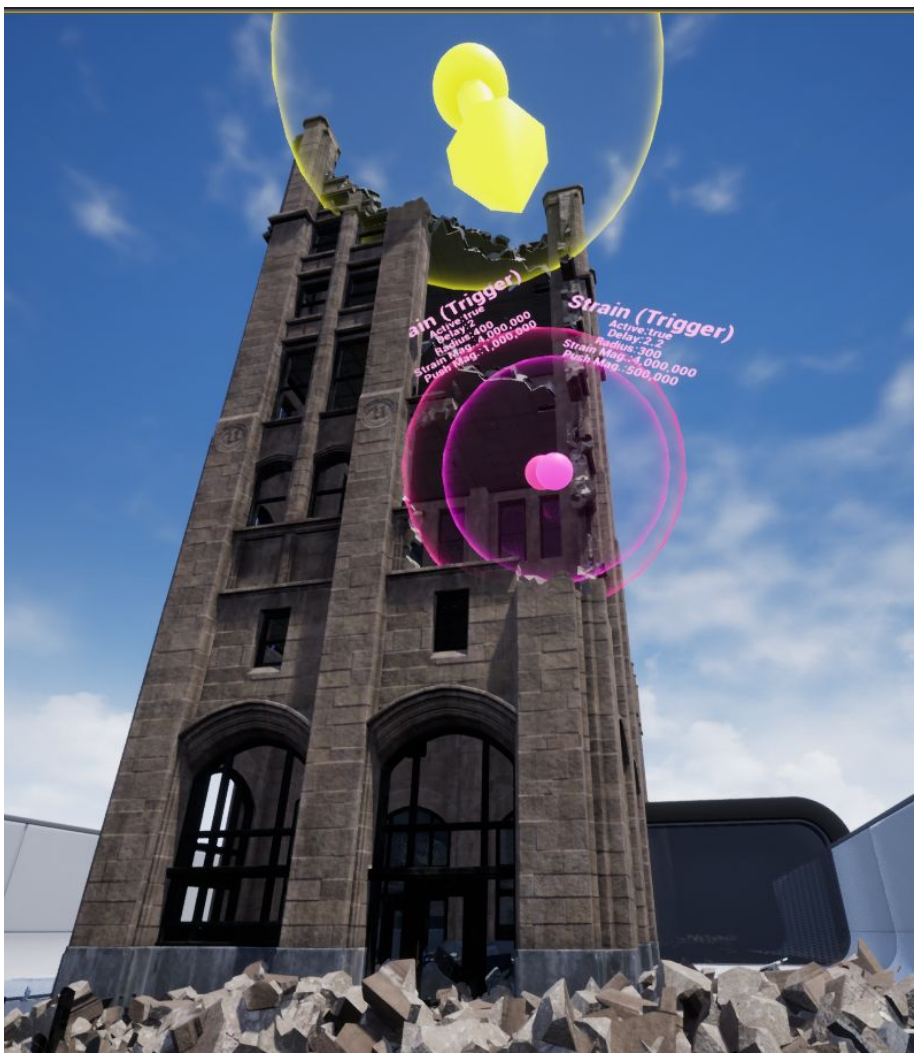
Testing/experimenting with Chaos is quite difficult as you have to do a custom source engine build, pull it from github and make code changes. Once we add this to our game branch it won't be an issue for artists to experiment.

General Chaos Notes

- The workflow/pipeline allows for real-time sim or there's methods to bake a sim and play it using LevelSequencer, this is particularly powerful because the workflow seems VERY fast (even faster than Houdini) and has a lot of performance options.
- There are a variety of "helper" BP's like anchor points, forces, etc. to get exact behavior for a sim. This is similar to using animated primitives in max to "push" chunks a certain direction or force nodes in Houdini.
- It has a bunch of collision solutions:
    - Analytic Collision
        - Box -> bounding box around the chunk.
        - Sphere -> sphere that fits within the rigid body.
        - 

    - Implicit Collision
        - Uses voxel grid to generate a signed distance field for simulation.
        - Different resolutions have different accuracy levels and memory cost.
        - 

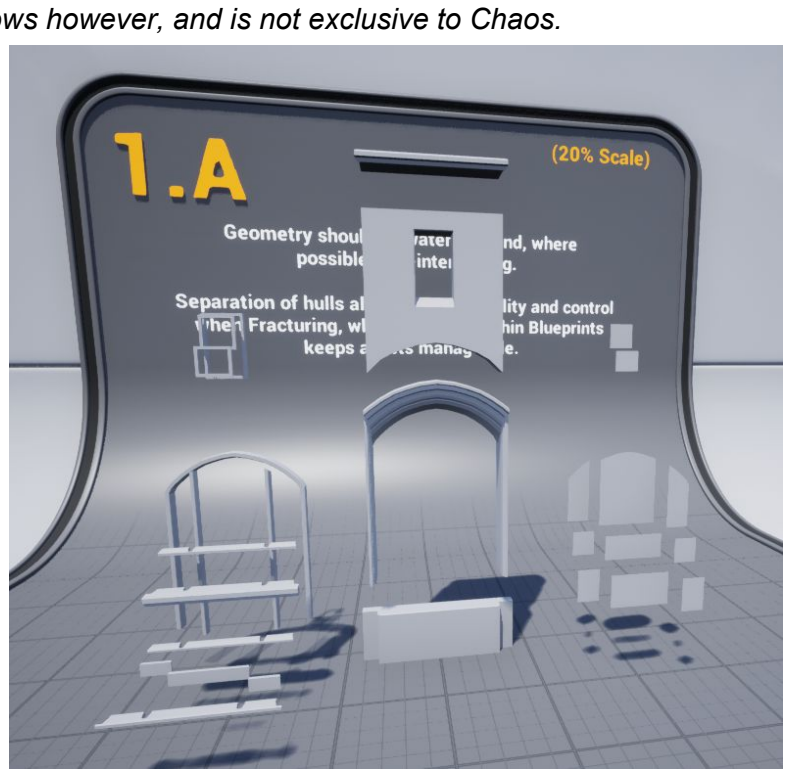
- Level designers or world builders can place ***StaticMeshSimulationComponents*** on a placed SM to allow it to interact with Chaos simulations.
- There are trigger BP's that can define where a simulation/initial force starts from, as well as other forces.



- These are also called “fields” and there are a bunch of different ones:
  - Anchor fields: holds chunks in place (like the base of structures).
  - Strain field: Applies “strain” to chunks, once strain surpasses the damage threshold it’ll break.
  - Force field: Applies force to the location, can include angular torque and other force types.
  - Culling field: restrains operations within the volume to help with performance.
  - Sleep/disable field: Brings clusters to rest when not moving/below a velocity threshold, this helps take clusters out of sims for performance when they’re done moving.
  - Internal strain field: Applies an over time, slow strain application that falls off to “weaken” the geometry collection over time until it falls apart.
- Each chunk can generate OnBreak, Trailing or OnCollision events that can tie into Niagara or Blueprint to trigger FX spawning and/or any desired logic.
- You can spawn a field OnHit (like from a projectile hitting something) to trigger destruction from gameplay/shooting/etc.
- Caching simulations is very involved, requires really strong organization and is fairly cumbersome. The houdini mesh export process is faster; but since its fully skeletal animation would likely be a far higher runtime performance cost than playing a cached sim in Chaos/Sequencer because it appears to be fairly optimized.
- The entire process involved a lot of configuration and has a lot of settings overall, doing simple breaks like walls would be fairly straightforward but large building/structure networks could require a dedicated developer.

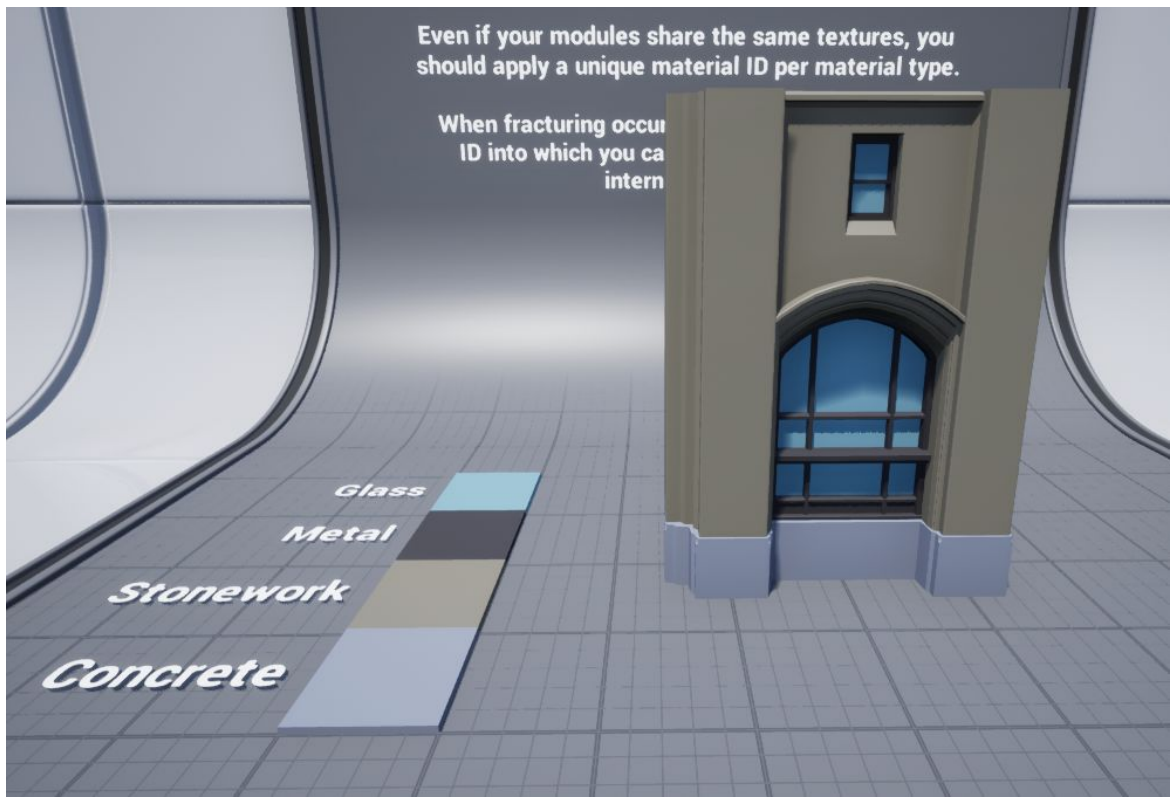
Best Practices

1. Meshes/pieces of meshes must be watertight for Chaos to work it’s best. *This is kind of a general rule for all destruction workflows however, and is not exclusive to Chaos.*



a.

b. All pieces of this facade are two-sided and have a small amount of thickness and do not interpenetrate. Basically how it’s actually constructed. This is important for not getting chunks launched into space and other typical physics bugs.

2. Even if your modules share the same textures, you should apply a unique material ID per material type. This is because the fracturing process assigns an ID for the internal geometry based on material ID, so glass gets inside glass; wood gets inside wood.
    a. 

3. Meshes can be converted into Geometry Collections in any combination.
    a. Small modules are more flexible in terms of placement, but risk high draw calls, visible seams and repetition when duplicated.
    b. Larger, combined modules are less flexible, but can be fractured and clustered to hide seams.
4. Collision types have different costs, consider them carefully:
    a. Analytic (box/sphere) collision: Faster, less memory cost; lower accuracy.
    b. Implicit (distance field) collision: Highly accurate, scalable for perf; higher memory cost
5. Organization, especially for caching simulation is of utmost importance. A large building structure may be made up of dozens of geometry collections and each one has its own cache assets, level sequence layer and is fairly involved.

Performance

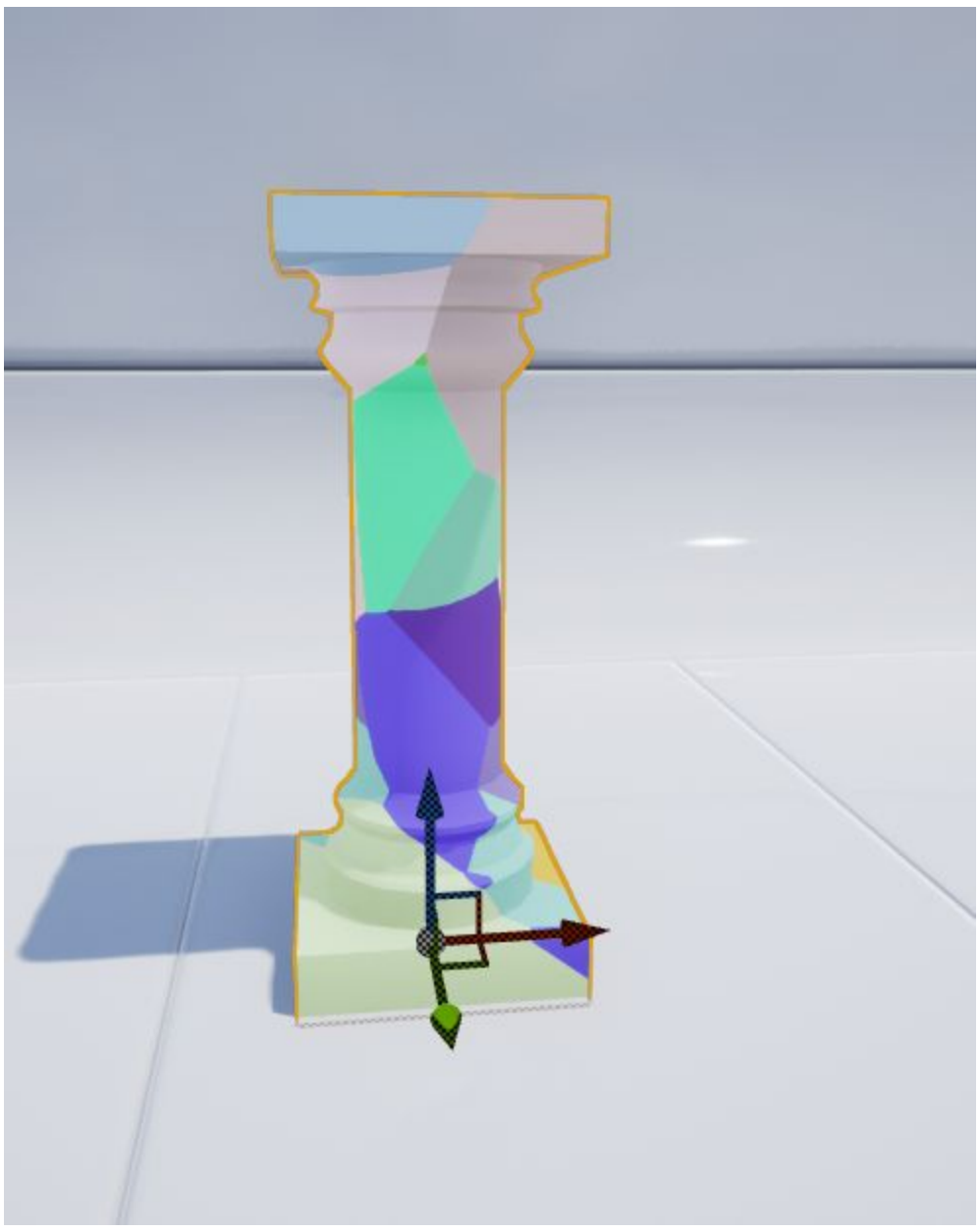
- 
- Playing a large scale modular building destruction real-time sim was about a 30fps drop on a 1070, this would likely be a 40-50fps drop on a lower end PC or last gen (Xbox One/PS4) console and 10-20fps drop on a next gen console/2000 series card).
- Playing the same simulation cached/played as an animation was only a 5-10fps drop on a 1070.

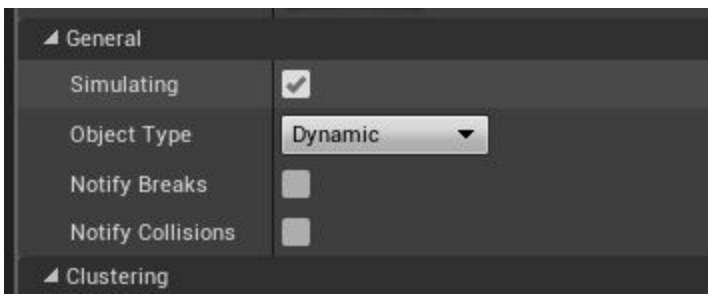
Use Case #1: Single Asset Implementation

The following sample asset was taken and put through the fracture tool to measure workflow efficiency and to see how easy it is to use the tools:

1. From the Fracture mode click Create Geometry Collection from Static Mesh Selection
    a. 

2. From that point, once a GeometryCollection is created you can configure your fracture to your desired result and hit ***Fracture*** button once satisfied:

a. 

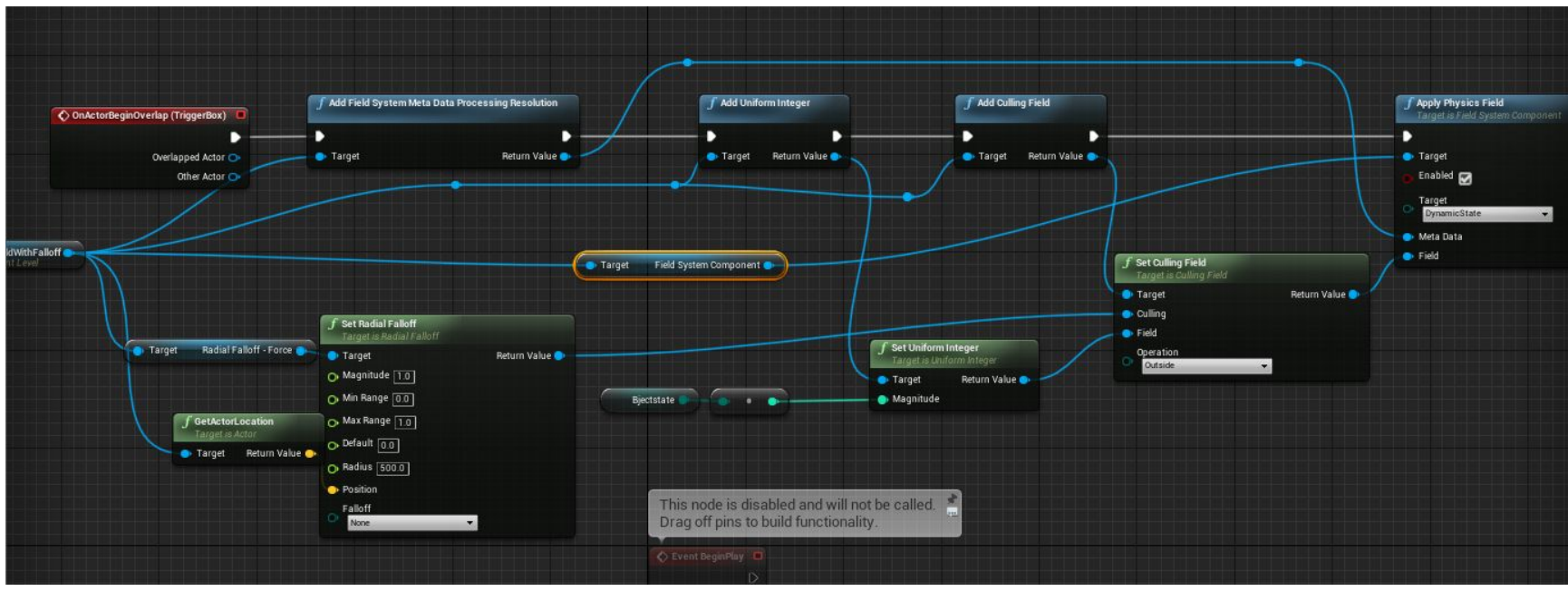
3. The ***Object Type*** in the General settings determines if it falls apart right away, or requires some kind of collision:
   a. 

   b. These default settings will have the object destruct on BeginPlay
   c. 

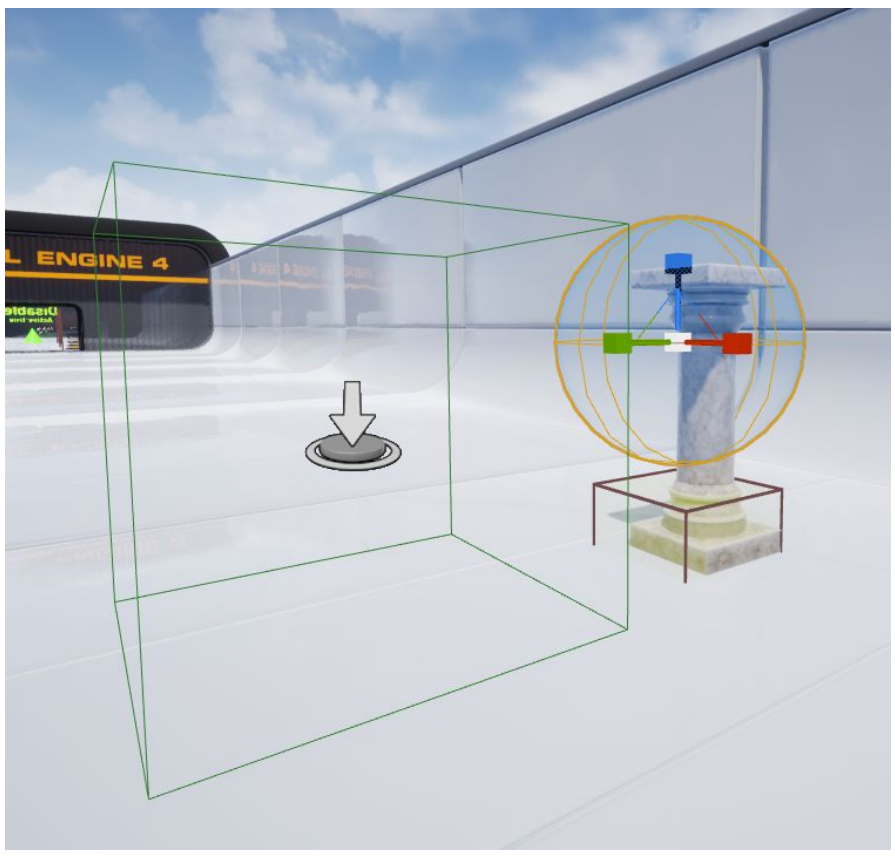
Static -> Dynamic

1. By default, chaos geometry collections will immediately break except for chunks surrounded by an Anchor Field. In most cases, devs will want some trigger.
2. Setting the ObjectType to Static will keep it from breaking as soon as the game starts.
   a. 

3. Dynamically changing it to dynamic to receive forces/impulses from a field or something else is fairly involved and somewhat challenging. The setup below is a minimal setup, using the generic forcefield to trigger a destruction sim when you walk into a simple trigger volume.

a. 

b. This setup appeared to be the minimum required configuration to trigger destruction from a spherical field:

i. 

4. It appears that managing component costs on many forcefields in a level may introduce performance issues if there are dozens or hundreds of chaos actors in a single scene.

Use Case #2: Modular Building Implementation

1. The following building facade was created out of multiple air-tight modular meshes provided in the Chaos sample project:

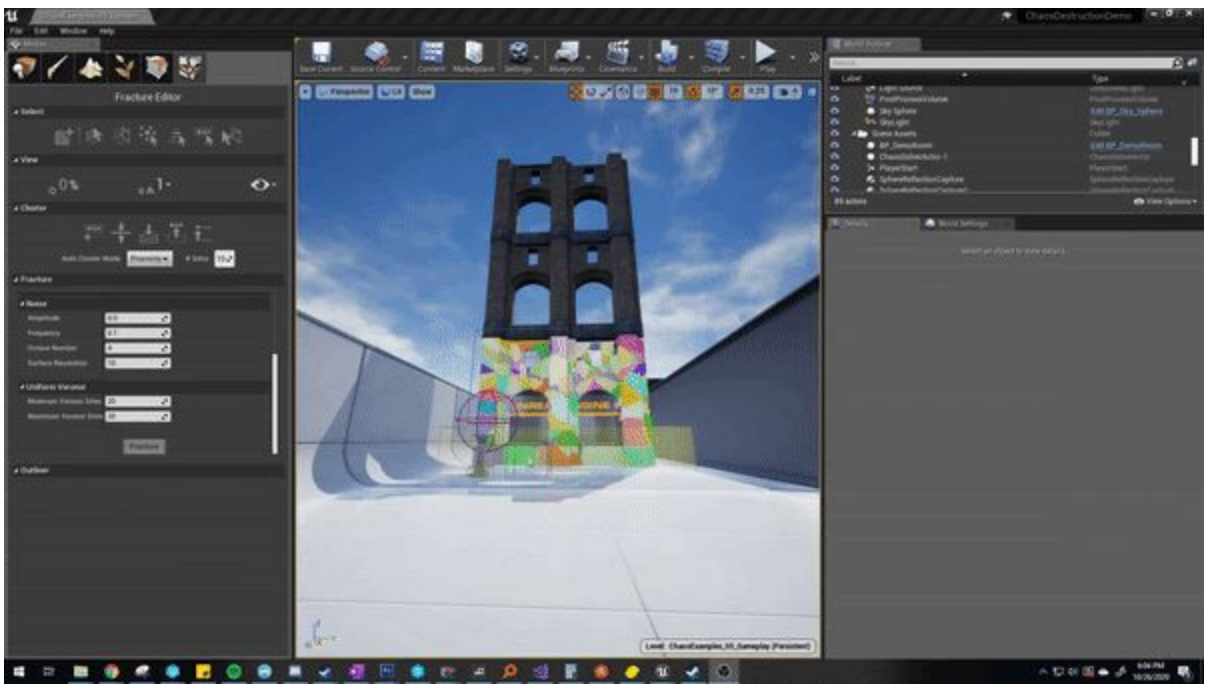
a. 

2. Combination of meshes were added to one GeometryCollection used twice via the Fracture tool, then fracture settings were set up:
    a. Each individual mesh in the collection can be fractured separately.

b. 

   c. It appears that new geometry collections need a pivot point to be near the ground. In a few attempts, the building fell through the world by a “story” because the pivot point was between the first and second levels.

3. This can be stacked into as many levels as you want, in the final example below, the third level had no clusters (so everything fell to pieces) while the 1st and second story had clusters held together until “damage” thresholds were met:

a. 

4. This is fairly complicated physics to barely affect framerate, but setup is fairly complicated. This system is fantastic but fairly time intensive.

Conclusion

The UE4 Chaos system is extremely versatile, powerful and expansive. It has the flexibility and performance to quickly fracture and have a strong simulation in-engine without external software.

For simple, single-object destructions, such as a wall panel or a column Chaos would have an easy, simple setup to work efficiently and not take a ton of time. However, simple destruction sims may be cheaper/easier with a quick Houdini vertex animation export.

For complex, full-building destructions or a cinematic destruction of a full structure setup would be very time intensive and require a lot of detailed configuration and testing. In some cases, it may be better to do a baked sim in houdini however, the chunk count/complexity to performance ratio seems to favor Chaos over playing skeletal or vertex animation. The above simulation has hundreds of chunks and barely hits frame rate, while a comparable skeletal animation would be far more impactful on performance.

This also appears to perform and be far superior to UE4's classic Physics engine where you would check ***Simulate*** on an object and have physics simulated at real-time. The caching/saving a sim option within Chaos is very easy to use to save a sim and play it back more efficiently for performance.