

CS354P

DR SARAH ABRAHAM

OVERVIEW: PHYSICS

ASPECTS OF GAME PHYSICS

- ▶ Forces applied to objects
 - ▶ World systems and rules
 - ▶ Object interactions
- ▶ Physical representation of objects
 - ▶ Point masses
 - ▶ Rigid bodies
 - ▶ Soft bodies
- ▶ Collision detection of objects

FORCES APPLIED TO OBJECTS

- ▶ Many types of forces:
 - ▶ Gravity
 - ▶ Impulses
 - ▶ Drag
 - ▶ Restitution
 - ▶ Springs
 - ▶ etc...

FORCES IN ACTION

Kerbal Space Program

Portal 2



Just Cause 3

CLASSICAL MECHANICS

- ▶ Area of physics that explores motion of objects
 - ▶ Relationship between force, trajectories, acceleration, and mass
 - ▶ Newton's second law: $F = ma$
- ▶ Forces in game engines relate to object velocities and accelerations (mathematical vectors) and object masses
- ▶ What else do we need to know to calculate forces?

OBJECT REPRESENTATION

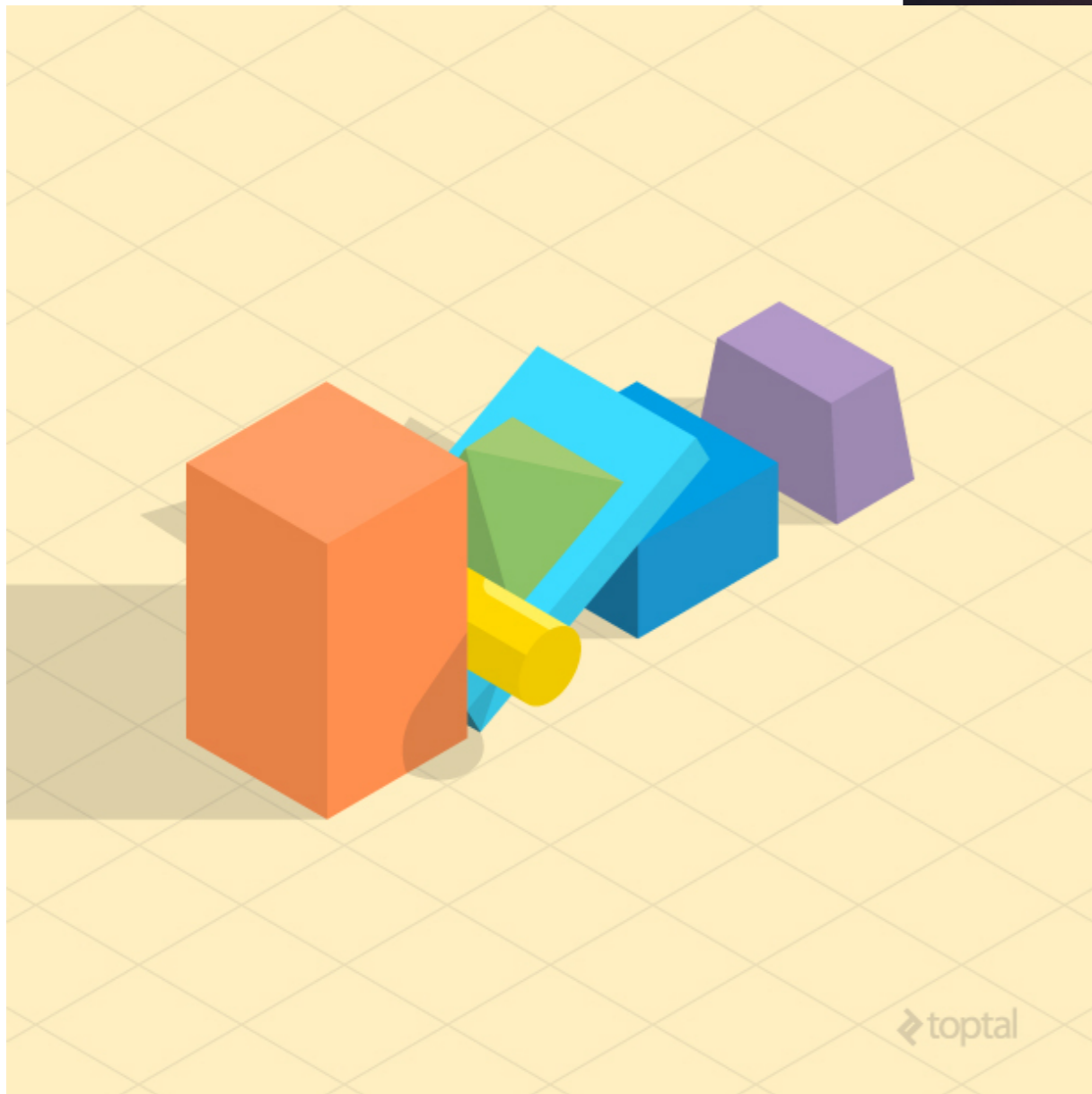
- ▶ Simplest representation of an object is a **point mass**
 - ▶ Position and mass with no volume (infinitesimally small)
 - ▶ Simplifies physical calculations
- ▶ Better representation is a **rigid body**
 - ▶ Object has volume but no deformation
 - ▶ More complex calculations to account for angular position and velocity
- ▶ Most accurate representation is a **soft body**
 - ▶ Object has volume and deformation
 - ▶ Much more complex calculations due to no fixed distance between objects
 - ▶ Can be pretty well approximated with a rigid body systems of springs

OBJECT REPRESENTATIONS

Point mass particle systems

<https://realtimevfx.com/t/unreal-engine-effects-in-marketplace/10088>

Rigid bodies



Soft bodies

<https://github.com/chrismarch/SoftBodySimulation>

COLLISION DETECTION

- ▶ Detection of collisions is a separate concern from application of forces
 - ▶ e.g. Collisions can result in an event trigger rather than a physical interaction
 - ▶ e.g. Forces can be applied to objects that are not collidable
- ▶ Detecting collisions can be as expensive (or more expensive!) than applying forces
 - ▶ Why?

WHEN TO DETECT?

- ▶ How do we know when two objects are colliding/about to collide/have collided?



WHEN TO DETECT?

- ▶ We detect collisions (and current forces) **per time step**
 - ▶ May be **based** on frame rate but should not be tied directly to frame rate
- ▶ Detect object collisions before they occur (**a priori**)
 - ▶ Will the two objects hit based on their current trajectories in the next time step?
- ▶ Detect object collisions after they occur (**a posteriori**)
 - ▶ Did the two objects hit between the previous time step and the current time step?
- ▶ Why can't we try to detect when a collision happens?

UE5 AND PHYSICS

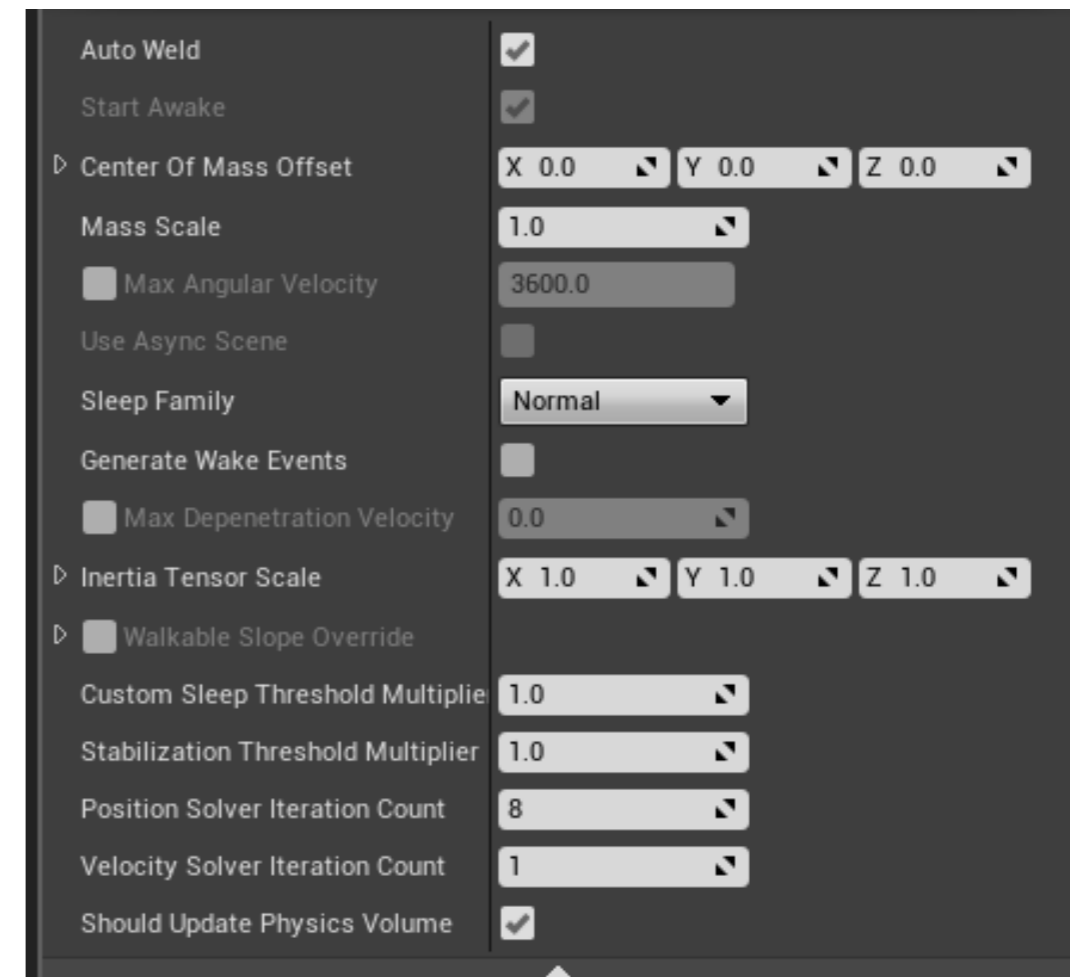
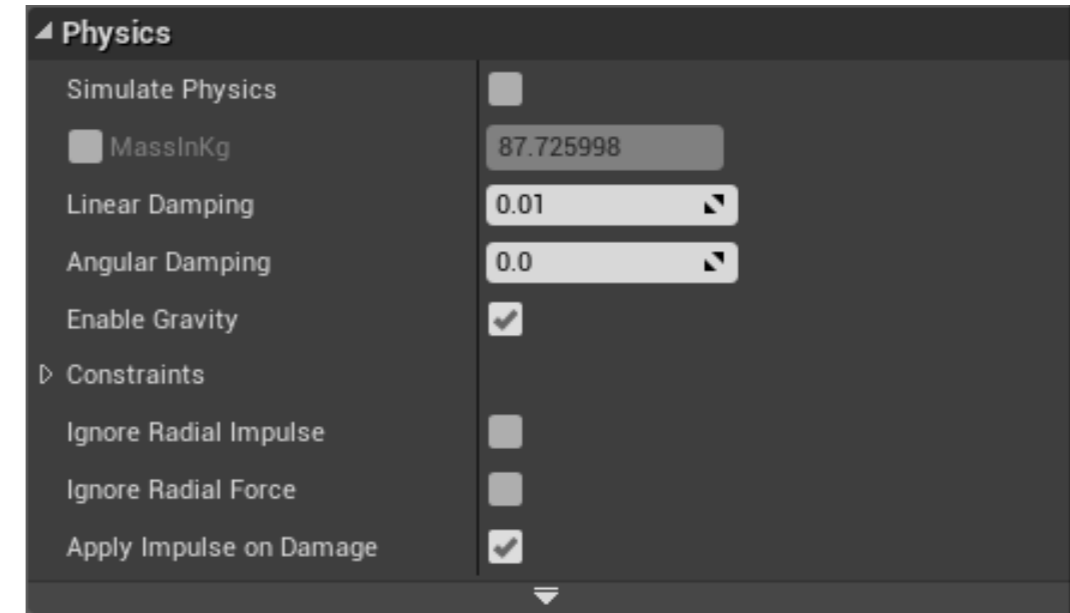
- ▶ UE5 uses Epic's Chaos physics engine
 - ▶ NVidia PhysX no longer supported
- ▶ Many advanced physical features supported in UE5
 - ▶ Cloth
 - ▶ Fluid
 - ▶ Destruction
- ▶ We will mostly focus on the basics...

PHYSICS BODIES

- ▶ Simplified 3D meshes that UE4 uses to represent rigid bodies
 - ▶ Contains related physical and collision information
- ▶ Uses the `FBodyInstance` struct to store information

PHYSICS PROPERTIES

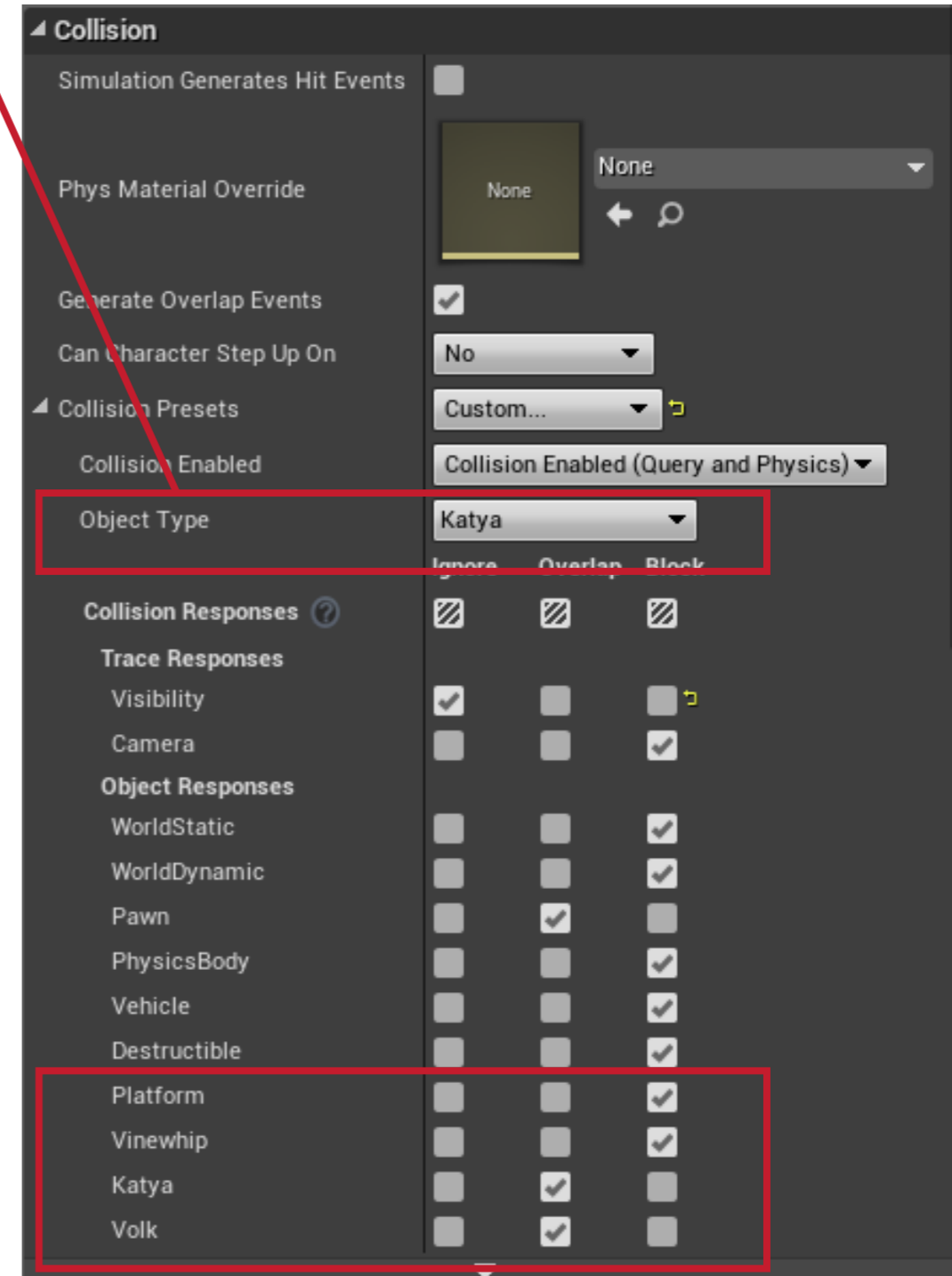
- ▶ Simulate Physics determines if body is **simulated** or **kinematic** (i.e controlled outside of simulation)
- ▶ Linear and angular damping are drag forces
- ▶ Constraints lock rotations to an axis
- ▶ And more...



COLLISION PROPERTIES

- ▶ Can generate "Hit" and "Overlap" events to perform actions after a collision
- ▶ Type of collision responses based on object type, collision type, and other object type
 - ▶ Physics allows for physical simulation
 - ▶ Queries allow for spatial checks (overlaps, raycasts, sweeps)
- ▶ Can define additional Object/Trace Channels for collision response

Custom channels for custom functionality and handling



COLLISION SETTINGS

- ▶ Collision settings (like most engine settings) under Edit -> Project Settings
- ▶ Settings stored in the .ini files found in the Config folder
 - ▶ Can look through and edit this in plain text as well

Engine - Collision
Set up and modify collision settings. Set as Default Export... Import... Reset to Defaults

These settings are saved in DefaultEngine.ini, which is currently checked out.

Object Channels

You can have up to 18 custom channels including object and trace channels. This is list of object type for your project. If you delete the object type that has been used by game, it will go back to WorldStatic. New Object Channel... Edit... Delete...

Name	Default Response
Platform	Overlap
Vinewhip	Overlap
Katya	Overlap
Volk	Overlap

Trace Channels

You can have up to 18 custom channels including object and trace channels. This is list of trace channel for your project. If you delete the trace channel that has been used by game, the behavior of trace is undefined. New Trace Channel... Edit... Delete...

Name	Default Response
------	------------------

Preset

You can modify any of your project profiles. Please note that if you modify profile, it can change collision behavior. Please be careful when you change currently existing (used) collision profiles. New... Edit... Delete...

Name	Collision	Object Type	Description
NoCollision	No Collision	WorldStatic	No collision
BlockAll	Collision Enabled (Query and PhysicWorldStatic		WorldStatic object that blocks all actors by default. All new custom cha
OverlapAll	Query Only (No Physics Collision)	WorldStatic	WorldStatic object that overlaps all actors by default. All new custom c
BlockAllDynamic	Collision Enabled (Query and PhysicWorldDynamic		WorldDynamic object that blocks all actors by default. All new custom c
OverlapAllDynamic	Query Only (No Physics Collision)	WorldDynamic	WorldDynamic object that overlaps all actors by default. All new custom c
IgnoreOnlyPawn	Query Only (No Physics Collision)	WorldDynamic	WorldDynamic object that ignores Pawn and Vehicle. All other channels
OverlapOnlyPawn	Query Only (No Physics Collision)	WorldDynamic	WorldDynamic object that overlaps Pawn, Camera, and Vehicle. All othe
Pawn	Collision Enabled (Query and PhysicPawn		Pawn object. Can be used for capsule of any playable character or AI.
Spectator	Query Only (No Physics Collision)	Pawn	Pawn object that ignores all other actors except WorldStatic.
CharacterMesh	Query Only (No Physics Collision)	Pawn	Pawn object that is used for Character Mesh. All other channels will be
PhysicsActor	Collision Enabled (Query and PhysicPhysicsBody		Simulating actors
Destructible	Collision Enabled (Query and PhysicDestructible		Destructible actors
InvisibleWall	Collision Enabled (Query and PhysicWorldStatic		WorldStatic object that is invisible.
InvisibleWallDynamic	Collision Enabled (Query and PhysicWorldDynamic		WorldDynamic object that is invisible.
Trigger	Query Only (No Physics Collision)	WorldDynamic	WorldDynamic object that is used for trigger. All other channels will be
Ragdoll	Collision Enabled (Query and PhysicPhysicsBody		Simulating Skeletal Mesh Component. All other channels will be set to
Vehicle	Collision Enabled (Query and PhysicVehicle		Vehicle object that blocks Vehicle, WorldStatic, and WorldDynamic. All c
UI	Query Only (No Physics Collision)	WorldDynamic	WorldStatic object that overlaps all actors by default. All new custom c

PHYSICS SETTINGS

- ▶ Physics settings also under Edit -> Project Settings
- ▶ Determines parameterizations for the physics simulation in Chaos as well as memory usage/accuracy

What's this?

The screenshot shows the 'Physics' settings in Unreal Engine, divided into several sections:

- Constants:**
 - Default Gravity Z: -980.0
 - Default Terminal Velocity: 4000.0
 - Default Fluid Friction: 0.3
 - Simulate Scratch Memory Size: 262144
 - Ragdoll Aggregate Threshold: 4
- Simulation:**
 - Enable 2DPhysics:
 - Default Degrees Of Freedom: Full 3D
 - Bounce Threshold Velocity: 200.0
 - Friction Combine Mode: Average
 - Restitution Combine Mode: Average
 - Max Angular Velocity: 3600.0
 - Max Depenetration Velocity: 0.0
 - Contact Offset Multiplier: 0.02
 - Min Contact Offset: 2.0
 - Max Contact Offset: 8.0
 - Simulate Skeletal Mesh on Dedicated Server:
 - Default Shape Complexity: Simple And Complex
 - Disable CCD:
 - Enable Enhanced Determinism:
- Optimization:**
 - Suppress Face Remap Table:
 - Support UV From Hit Results:
 - Disable Active Actors:
- Framerate (highlighted with a red box):**
 - Max Physics Delta Time: 0.033333
 - Substepping:
 - Substepping Async:
 - Max Substep Delta Time: 0.016667
 - Max Substeps: 6

PHYSICS TIME STEP

- ▶ Physics is continuous but our simulations are not
- ▶ Must approximate physical interactions within a time step
 - ▶ Larger time steps are generally faster but less accurate
 - ▶ Fixed time steps are generally better for stability
- ▶ How does this relate to frame rate?

FRAME RATE AND SUB-STEPPING

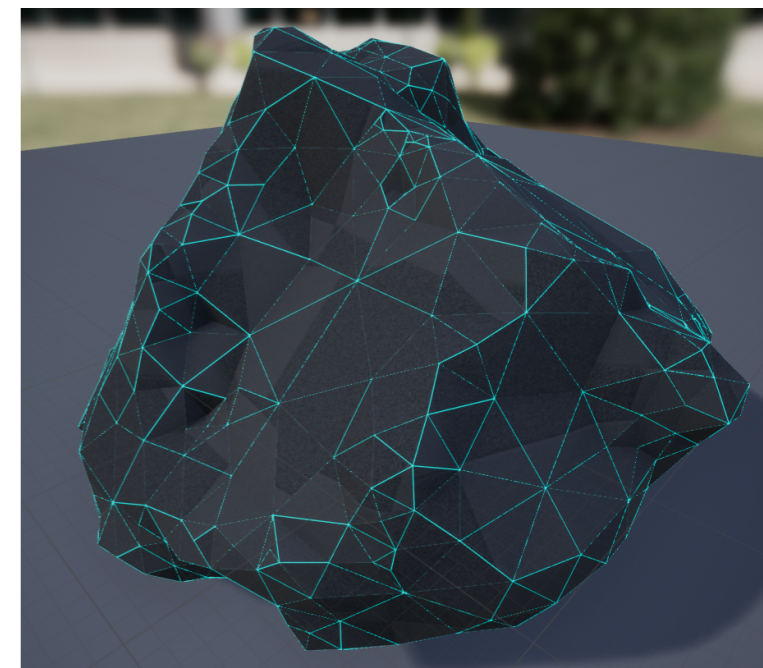
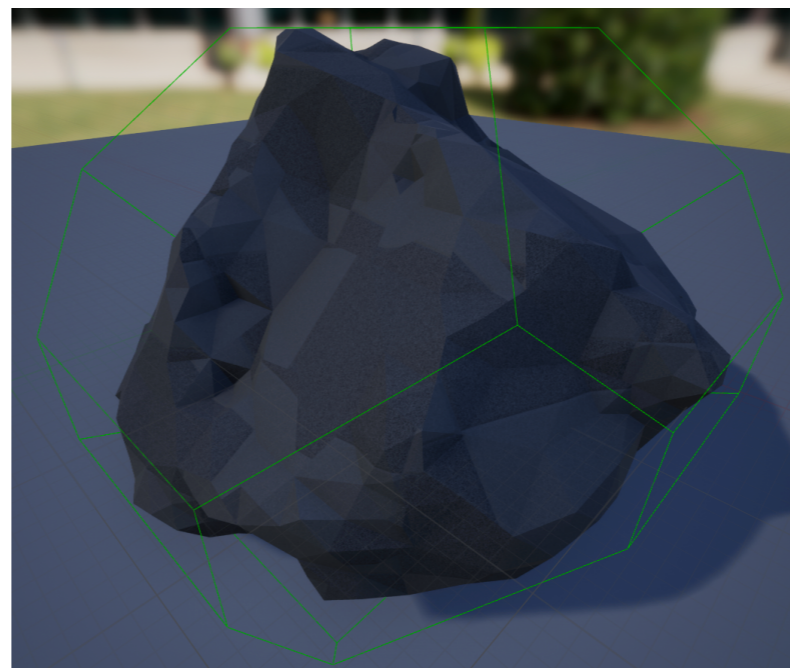
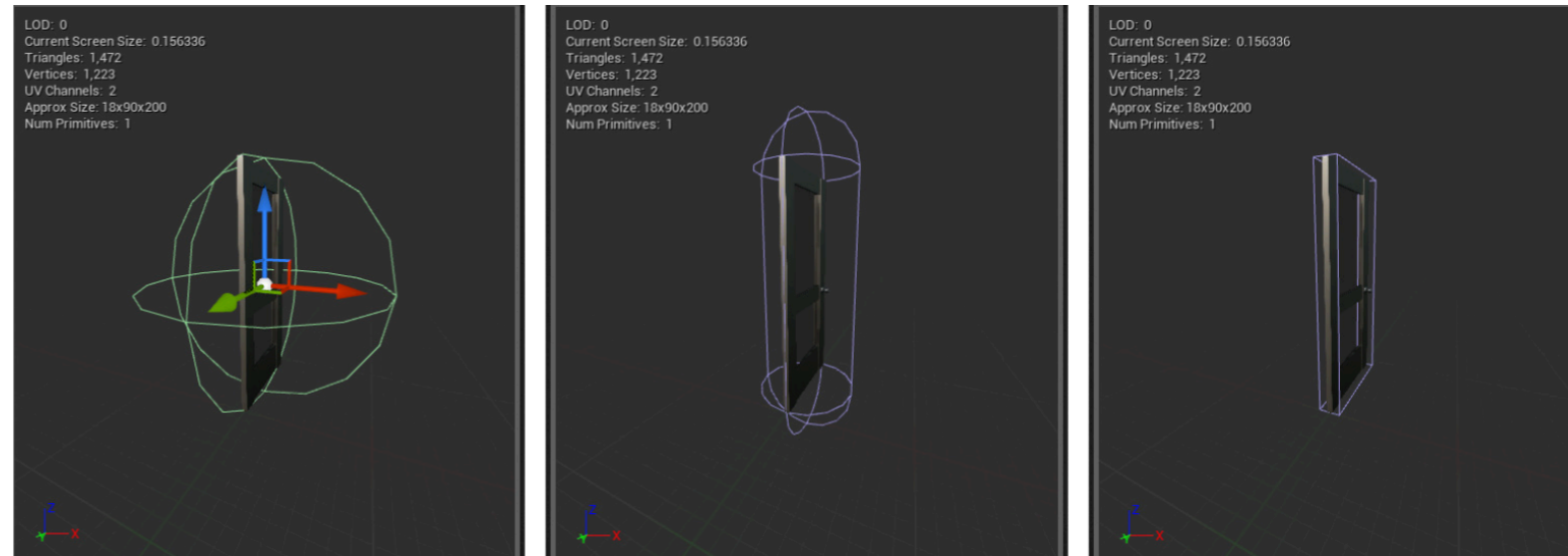
- ▶ We often want physics tied to frame rate to ensure responsiveness but frame rate is highly variable
 - ▶ Naively connecting time steps to frames may result in physics bugs/inaccuracies
- ▶ Solution: sub-stepping divides a frame into smaller physics time steps which execute each frame
 - ▶ Extra time can roll over to the next frame
- ▶ Enabling sub-stepping incurs execution overhead but results in better accuracy
- ▶ Side note: collision callbacks are delayed until the final sub step is finished for threading efficiency
 - ▶ Thus you can have multiple callbacks for an object executed within a single frame in FIFO order

COLLISION VOLUMES

- ▶ Collision checking is based on the mesh faces of an object
 - ▶ Must consider how the interactions per-face of an object's mesh will impact the collision
 - ▶ Similar problems/solutions in graphics: spatial data structures, fast intersection tests etc
- ▶ High level idea: simpler collision volumes means faster collision checks

UE4 COLLISION VOLUMES

- ▶ Can compose collision volumes out of simple shapes: boxes, spheres, capsules
- ▶ Or generate collision volumes from a mesh (simple vs complex)
- ▶ How should you decide?



STATIC VS SKELETAL MESHES

- ▶ Static meshes are the standard meshes used to create world geometry
 - ▶ Set of polygons that can be cached in video memory for efficient rendering
 - ▶ Can apply **affine** transformations (scale, rotate, translate) but not other vertex manipulations
- ▶ Skeletal meshes are meshes that have hierarchical controls used to create characters and other animating objects
 - ▶ Set of polygons manipulated via a skeleton
 - ▶ Vertices move relative to they underlying skeleton based on **skinning** algorithm
 - ▶ Can convert skeletal meshes to static meshes to save poses but will not generally work for dynamic scenes

PHYSICS CONSTRAINTS AND DAMPING

- ▶ Constraints can be used to connect actors in a physically-based way
- ▶ Constraints are types of joints (ball-and-socket, hinge, etc) but can also be customized
- ▶ **Physics Constraints** can be actors or components
 - ▶ Actors placed into a scene
 - ▶ Components placed into an actor
- ▶ Can apply a wide range of parameters to emulate different physical behaviors
- ▶ Can test using "Play" in editor or "Simulate"
- ▶ Read here for more tutorial information: <https://docs.unrealengine.com/en-US/Engine/Physics/Constraints/index.html>

PHYSICS CONSTRAINTS IN C++

- ▶ All Blueprint constraints can be done in C++
 - ▶ I'd recommend quick prototyping in Blueprint, building the foundation in C++, then building the in-game instance in Blueprint based on the C++
 - ▶ ...this may seem round-about, but it will generally result in pretty fast development cycle, good looking code, and a designer-friendly final product
- ▶ Must create and attach static mesh components then create an `FConstraintInstance` to set properties in code
- ▶ Any Blueprint class created from this C++ class will have values set in the C++ constructor
 - ▶ Remember to make the `UPROPERTY` BlueprintReadWrite if you want values accessible within the Blueprint

PHYSICS CONSTRAINTS C++ EXAMPLE

stable component is fixed; bounce component moves relative to it

```
RootComponent = CreateDefaultSubobject<USceneComponent>(TEXT("RootComponent"));
stableComponent = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("stableComponent"));
bounceComponent = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("bounceComponent"));
stableComponent->AttachToComponent(RootComponent, FAttachmentTransformRules::KeepRelativeTransform);
bounceComponent->AttachToComponent(RootComponent, FAttachmentTransformRules::KeepRelativeTransform);
```

Set properties of constraint interactions
(in this case, a bouncy platform)

```
FConstraintInstance platformConstraintInstance;
FConstraintProfileProperties platformConstraintProperties =
platformConstraintInstance.ProfileInstance;
platformConstraintInstance.SetLinearXMotion(ELinearConstraintMotion::LCM_Limited);
platformConstraintInstance.SetLinearYMotion(ELinearConstraintMotion::LCM_Locked);
platformConstraintInstance.SetLinearZMotion(ELinearConstraintMotion::LCM_Limited);
platformConstraintInstance.ProfileInstance.LinearLimit.Limit = 5.0;
platformConstraintInstance.ProfileInstance.LinearLimit.bSoftConstraint = true;
platformConstraintInstance.ProfileInstance.LinearLimit.Stiffness = 3000.0;
platformConstraintInstance.ProfileInstance.LinearLimit.Restitution = 1.0;
platformConstraintInstance.ProfileInstance.LinearLimit.ContactDistance = 1.0;
platformConstraintInstance.SetAngularSwing1Limit(EAngularConstraintMotion::ACM_Locked, 0);
platformConstraintInstance.SetAngularSwing2Limit(EAngularConstraintMotion::ACM_Locked, 0);
platformConstraintInstance.SetAngularTwistLimit(EAngularConstraintMotion::ACM_Locked, 0);
platformConstraintInstance.ProfileInstance.ConeLimit.Stiffness = 1000.0;
platformConstraintInstance.ProfileInstance.ConeLimit.Restitution = 1.0;
```

constraint component connects
the two "physical" pieces

```
constraintComponent =
CreateDefaultSubobject<UPhysicsConstraintComponent>(TEXT("platformConstraintComponent"));
constraintComponent->AttachToComponent(stableComponent,
FAttachmentTransformRules::KeepRelativeTransform);
constraintComponent->ConstraintInstance = platformConstraintInstance;
constraintComponent->SetConstrainedComponents(stableComponent, "Stable Component", bounceComponent,
"Bounce Component");
```


PHYSICAL MATERIALS

- ▶ Unreal uses physical materials to define an object's interactions with the world
 - ▶ Can adjust parameterization to be applied to any object using that material
 - ▶ Can be use in conjunction with regular materials (i.e. the shaders and lighting models used on objects for rendering)

CLOTH SIMULATION

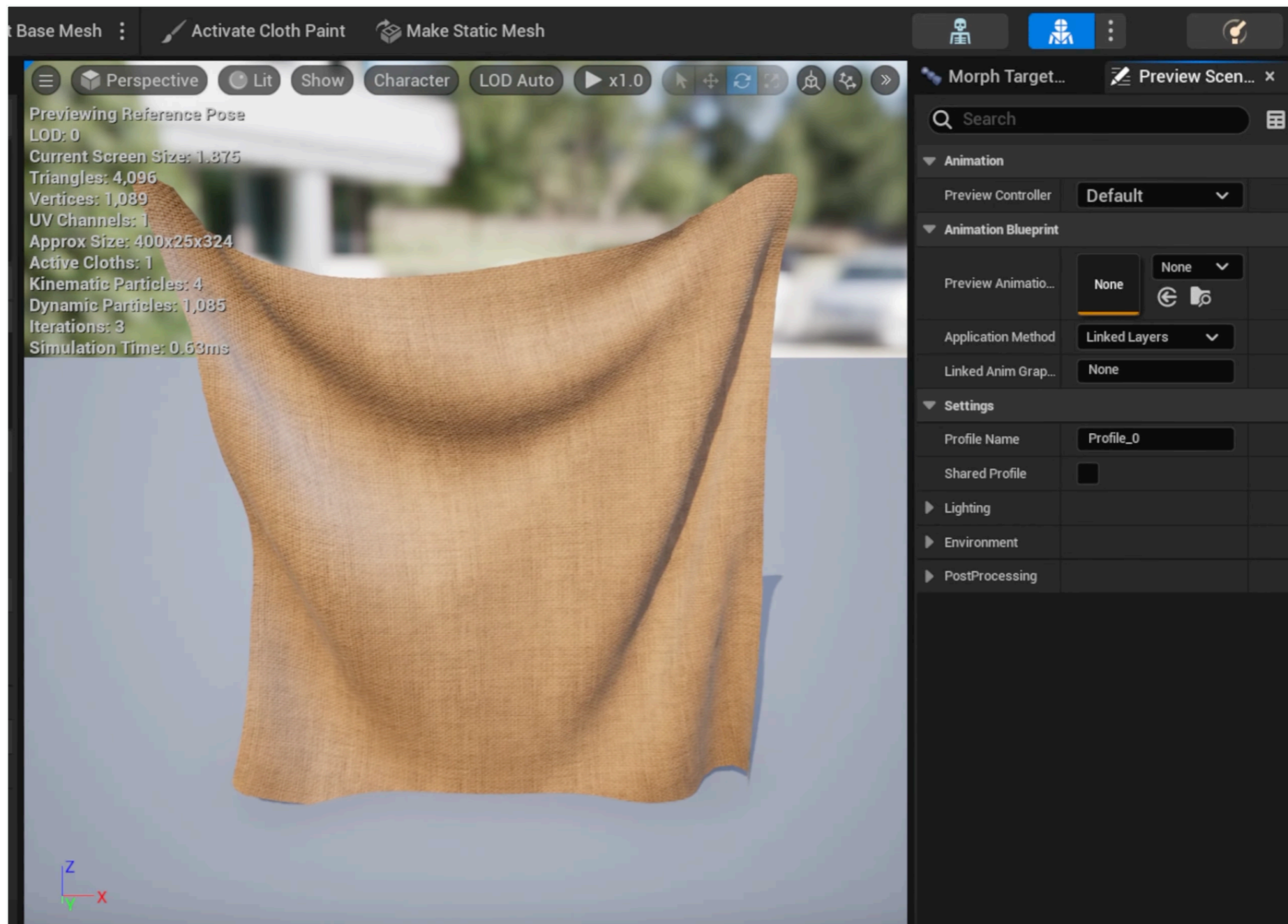
- ▶ UE5 uses Chaos Cloth solver to create cloth effects
 - ▶ Uses a particle system with constraints to create cloth-like movements and collision responses
- ▶ UE5 allows artists to import cloth asset then paint “clothiness” onto mesh
 - ▶ Determines how much the individual parts react like cloth

DESTRUCTIBLE ACTORS

- ▶ UE4 uses Chaos Destruction to create destructible meshes
 - ▶ Allows static meshes to be broken into dynamic pieces in a parametrizable way
 - ▶ Works in real time
- ▶ Can be integrated with Niagara particle system and Audio Mixer to incorporate VFX and SFX

CHAOS CLOTH DEMO

▶ <https://youtu.be/un6ZNdcxQIk?si=fEHwhQ0WitotpgD4&t=552>



CHAOS DESTRUCTION DEMO

- ▶ <https://www.youtube.com/watch?v=XaPECMaKbSI>

