FluidNinja LIVE Manual

Document updated: 5 Oct 2021

Latest NinjaLive version: 1.5.23.3 (for UE 4.23 - 4.25)

1.5.26.3 (for UE 4.26 - 4.27)

NinjaLive is providing real me, responsive fluid simula on inside Unreal. For *baking* fluid simula on, see: [NinjaTools](https://www.unrealengine.com/marketplace/en-US/product/fluidninja-vfx-tools)

Support: andras.ketzer@gmail.com

Discord: h ps://discord.gg/VpcyBQa77w

Updates, news: h [ps://twi](https://twitter.com/FluidNinjaLIVE) [er.com/FluidNinjaLIVE](https://twitter.com/FluidNinjaLIVE)

Project home page: unrealengine.com/marketplace/fluidninja-live

Changelog, Issues, FAQ

h ps://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4\_Yx\_FM

(1) Showcase videos, (2) Tutorials, (3) Use cases:

h [ps://www.youtube.com/playlist?list=PLVCUepYV6TvOrOfQVLMCxl\_JoU\_cIkK8P](https://www.youtube.com/playlist?list=PLVCUepYV6TvOrOfQVLMCxl_JoU_cIkK8P) h ps://www.youtube.com/playlist?list=PLVCUepYV6TvPYbofpEf\_ghznfihM-yt-B

h ps://www.youtube.com/playlist?list=PLVCUepYV6TvO6QhpqaT1GRVGC96QMcUJs

Document share URL:

h ps://drive.google.com/file/d/1I4dglPjeXLcNkSGxGok8sQCy59qgYcF9

Contents

0. Step-by-step Guides / Tutorial Videos

1. Introduc on

1.1 Inputs

1.2 Level & Game Design

1.3 Rendering

2. Implementa on

11. U li es (PossessPawn, FPS limiter)

12. Performance Tests

13. Turning Ninjalive To A Plugin

14. Packaging / Compiling In General

15. Packaging To Android, Briefly

16. Planned Features

17. Op miza on

3. Loca on & Size

4. Versions - UE4, UE5 compa bility

5. Usage

6. Core Func ons & Terminology

18. Changelog & Updates

19. Known Issues & FAQ

20. Niagara

outsourced, see LINK outsourced, see LINK

6.1 NinjaLiveComponent

6.2 NinjaLive

6.3 Actor Details vs Actor Component Details

6.4 Preset Manager and Presets

6.5 Simple Painter Mode

7. Memory Demand

8. Performance (Excerp on)

9. High Level Se ngs: Merging NinjaLive to your Project

10. Low Level Se ngs

10.1 NinjaLive Actor Params *(briefly)*

10.2 NinjaLive Actor Component Params *(briefly)*

10.3 Adding *NinjaLiveComponent* to your own Actor classes

10.4 Other Ways To Include/embed/add Ninjalive To Actors

10.5 Interac on: se ng up Objects to trigger NinjaLive response

21. Complete List of NinjaLive parameters

21.1 GUI exposed params

21.2 NinjaLive Actor params

21.3 NinjaLive Actor Component params

22. Fluidsim pipeline: a technical descrip on

23. Sequencer & Movie Render Queue

24. Volumetrics

25. Detail Maps

26. Controlling Ninja In Real Time

27. Mapping sim on 3D mesh surfaces

28. Moving in WorldSpace

29. Addi onal Content

Step by Step guides in this Manual

*Topics*

Merging NinjaLive to your project

Adding NinjaLive Component To Your Own Actor Classes Linking Niagara and Ninja

Cinema cs Rendering via Movie Render Queue Linking Volumetrics And Ninja

(Re)configuring a sim container to use 3D sim mesh

*Chapter*

9 10.3 20.3 23.3 24.4 27.1

Tutorials on YouTube

*Topics*

Tutorial videos playlist Showcase videos playlist

Introduc on

Merging NinjaLive to your project

Changing default (planar) TraceMesh to Custom Mesh

Use TraceMesh to detect overlap (instead of Interac onVol.) User defined line-tracing source (instead of Camera)

*URL*

LINK

LINK

LINK LINK LINK LINK LINK

Render Buffers & Output Materials LINK

Disable Dynamic Asset Lookup (DAL), in the FAQ doc

i005

Sim ling / wrapping

Pain ng persistent Flow pa erns (2D, Fog, Clouds...etc)

LINK1, LINK2 LINK

Using Sequencer to animate Objects and Sim Params LINK

Character FX related videos:

Adding NinjaLiveComponent to Your Character

LINK

SimDetails, CameraFacing, SingleTargetMode LINK

Reposi oning TraceMesh, adding Custom Sockets LINK

Mul ple fluidsim components in a single character LINK

Hands: two dedicated sim components tracking all fingers LINK

Introduc on

FluidNinja LIVE is providing *real me, responsive fluid simula on* for PC and Mobile app developers, inside Unreal. Live is focusing on performance, backward compa bility and simple use. It could be u lized in any Unreal project: following a 5 minute setup, responsive fluid simula on Actors could be placed on level. Live provides an *ActorComponent* too, that could be added to user defined classes, pawns, vehicles. Live is easy to customize, features *preset-based fluid simula on management*, exposed variables and annotated blueprints, user editable output materials and *34 demo levels* providing usage examples. Live is *scalable*: could be op mized for average mobile devices - or run a 4k simula on container. Lowest supported UE version is 4.20. Live *does not* rely on Niagara - but could be used to drive Niagara.

1.1 Inputs

Live simula on could be driven by (a) Textures [sta c], (b) Materials [animated], (c) SceneCapture camera and Streaming video, (d) Scene geometry - including Sta c and Dynamic meshes, Skeletal Meshes (Bones, Sockets), PhysicsBodies. Plus user gestures via mouse, touchscreen or other device.

1.2 Level & game design

Live offers LOD, Proximity based Sleep/Wake and Memory Pooling - suppor ng a design approach with mul ple, local fluidsim containers for character and area effects. Live is not a robust "whole world system", but a system suppor ng a mul tude of dispersed, local interac ons.

1.3 Rendering

Live uses 2D fluid simula on, combined with technology that makes it behave like a volume.

A. Overlap Projec on is using a *volume* to track objects in the sim area, mapping 3D

posi onal and velocity data to the 2D sim using *Line Trace* from the camera (or from a user defined point). As a result, moving objects behind and in front of the simula on plane could also affect the sim - as if it would be spa ally extended.

B. Op onally, overlap projec on could be further improved by running the sim on a camera facing plane: viewers could walk around the interac on volume, while the pa erns generated by overlapping objects appear in the correct spa al posi on.

C. The moving direc on of overlapping objects is handled in 3D by the velocity solver: the perpendicular component of mo on (Z) causes expansion / shockwaves, while components aligning with the 2D simula on plane (X,Y) are dragging the fluid.

D. 3D iner al frame of reference is supported: in case the whole sim area is moving, the

fluid "lags behind" as Live channels world-space accelera on to the simula on.

E. Raymarching: light sources in 3D space (both sta c and moving) could generate self

shadows on the fluid with a proper direc on.

F. Parallax Mapping is further enhancing this effect by using fluid density as a height map

to offset sim data, making it look spa ally extended.

G. A well-balanced configura on of noises and a custom pressure solver helps to keep the

number of fluidsim pressure itera ons low, s ll producing detailed flow with vor ces.

In the end, we have dynamic fluid in 3D - with the GPU demand of a 2D sim.

2. Implementa on

NinjaLive is a compact Unreal Project - its feature set could be u lized in games by merging (see Chapter 9). It is based on standard Unreal assets (Blueprints, Materials), does not contain C++ code / pre-compiled elements / third party content. *None of the core func ons uses Niagara.* Cooking, compiling and packaging is tested for Windows PC, Apple PC, Android Mobile and iOS mobile.

Note1: Live blueprints are *annotated*, visually organized and cross referenced with links, to help developers understand and modify them. Example screenshots: [BP1](https://drive.google.com/file/d/11Pk-BK0sc4lEdPXjYcm2PvfZfGCqBi6M), BP2, BP3

Note2: while NinjaLive is currently implemented as a "Project", it could be turned into a "Plugin". See Chapter 13.

3. Loca on & Size

The project is located in a single folder: /Content/FluidNinjaLive

Func onal core is 35 Megabytes, Presets and Output Material Examples 15 MB, Tutorials 50

MB, the Shader and Asset Cache generated at first run is ~ 100 MB. The two most important

files, NinjaLive (Actor class) and *NinjaLiveComponent* (ActorComponent class) are 2 MB and

7 MB, respec vely.

4. Versions

NinjaLive is developed in separate branches to ensure compa bility with different UE versions. Check the version of your NinjaLive project at: /Edit /Project Se ngs /Project /Descrip on

UE 4.23 - 4.25 branch

Latest NinjaLive version: 1.5.23.3

UE 4.26 - 4.27 branch

Latest NinjaLive version: 1.5.26.3

Featuring: Volumetric Clouds (clouds are not supported below UE 4.26)

UE 5.00 branch: officially does not exist (as of 21 Aug 2021)

UE5 is in the early access phase - and EPIC Marketplace *does not allow* developers to offer UE5 dedicated product versions. For this reason, UE 4.27 ninja branch has been tested under UE5. Test results: (A) core func ons running fine - (B) a cri cal UE5 bug corrupts most setups

UE5 bug: *GetPhysicsLinearVelocity* and *GetComponentVelocity* blueprint nodes are failing. Nodes return ZERO for non-physics objects (eg. animated or blueprint driven objects).

The above two nodes work as expected under UE 4.20 - UE 4.27, returning correct velocity values for all kinds of moving objects. The bug has been reported to UE devs.

UE5 bug consequences: ninja massively u lizes object velocity to drive/control simula ons. In UE5 EA, objects leave density marks (as expected), but do not drag the fluid.

At stages where “kill brush at low velocity” feature is enabled: the stage is totally blacked out (nothing is visible or nothing happens - since UE5 constantly returns ZERO velocity).

Avoiding UE5 velocity bug: (A) “Single Target Mode” is immune, as it is using a different method to calculate velocity. (B) Texture and parameter defined velocity is also working. (C) By switching off the “kill brush at low velocity”feat, blackout stages could be fixed. See: *NinjaLiveComponent /LiveBrushSe ng /DampenBrushBelowThisVelocity* ---> set to 0.0

5. Usage

NinjaLive is a standalone Project: open it in Unreal Editor, check the Tutorial Levels! When star ng the game (pressing Play):

Use Selected Viewport Mode -- do NOT use Simulate Mode

For running ninja in Standalone Mode, see: chapter 21.3.6 (LiveCompa bility) In order to u lize fluidsim features in *your* project, NinjaLive should be *merged*:

6. Core Func ons & Terminology

FluidNinja LIVE is created from scratch to complement [FluidNinja](https://www.unrealengine.com/marketplace/en-US/product/fluidninja-vfx-tools) [VFX](https://www.unrealengine.com/marketplace/en-US/product/fluidninja-vfx-tools) Tools - a sibling project, focusing on non-responsive / baked fluid simula on. The two projects are shortly referred to as *NinjaLive* and *NinjaTools*. In this document, we are focusing exclusively on NinjaLive.

6.1 NinjaLiveComponent

FluidNinja Live is a library of func ons built around a single, compact unit: "*NinjaLiveComponent*", o en addressed as "simula on component", located at

1.

2.

Prepare the *target project* to host NinjaLive, by se ng the directly specified High Level Se ngs - see Chapter 9 !!!

copy */Content/FluidNinjaLive* to the */Content* folder of the target project

*/Content/FluidNinjaLive*. It is an "ActorComponent" class object that should be added to a host object (eg. an "Actor" class object) in order to use it. The host is always called "owner". The sim component could run autonomously (loading its default preset, crea ng RenderTargets for itself) - OR - it could connect to the interac ve *Preset Manager* and to the

Once merging is done, place/spawn NinjaLive fluidsim Actors at your levels - or add NinjaLive fluidsim Component to your own object classes. To make fluidsim Component work properly in your Actors, apply the directly specified Low Level Se ngs to the host actor class

( see Chapter 10.3 ).

Op onally you could u lize addi onal tools like Preset Manager with custom GUI, Memory Manager, U li es and Interface Controller.

Fluidsim related parameters could be stored / loaded as *presets* and handled by the Preset Manager (See *Chapters 6.4 and 21*).

Func onal parameters (eg. LOD se ngs, interac on filters) could be set at the *Actor* and *ActorComponent* Details panel - as exposed variables - or set by blueprints using *Live Interface*.

Besides NinjaLive Managers, UE Sequencer could be used to control SimContainers. See *Chapter 23*.

*Memory Manager*. An important restric on: sim component does \_*not\_* have built in overlap detec on func onality - and for this reason, interacts only with predefined owner components (eg. Bones, Sockets, Sta cMeshComponents... ).

6.2 NinjaLive

"*NinjaLive*", an "Actor" class object, referred as "sim container" or "sim area", located at */Content/FluidNinjaLive* ... is a dedicated owner, embedding "*NinjaLiveComponent*" and adding two important features to its core feature set:

(A) ac va on volume (usually much larger than the sim area): the proximity of any user defined agent could switch the sim component between wake/sleep states.

(B) interac on volume (usually same size as the sim area): detec ng overlapping / colliding objects and forwarding the spa al informa on + velocity to the sim component. Using the interac on volume, anything could interact with the simula on: *Sta c- and Dynamic Meshes, SkeletalMeshes, PhysicsBodies, Destruc bles*.

6.3 Actor Details VS ActorComponent Details

When selec ng an Actor on level, the *Details* panel is displaying all the exposed variables of that instance. In the case of "NinjaLive" Actor, the variables are collected into 4 groups, all star ng with "Live" prefix: *Ac va on, Interac on, Debug, ComponentOverrides*.

IMPORTANT: *NinjaLiveComponent* se ngs are exposed on the component level - accessible only at the Component Details panel - NOT visible at the Actor Details panel!

Accessing Component Details:

Select *NinjaLive Actor* - or any other sim component owner Actor on level. At the Details panel, there is a top sec on, where Actor Components are listed (to see all components, this "top list window" should be rescaled). Select "*NinjaLiveComponent*".

Here is a visual guide on “how to select NinjaLiveComponent” ---> [IMAGE](https://drive.google.com/file/d/10cHjZsM2alob8ZcfCYtDCNTHhYGHJLSK) LINK

Un l the Component is selected, the Details panel displays its (the component's) proper es - and NOT the owner proper es. *NinjaLiveComponent* variables are collected to 9 groups, all star ng with "Live" prefix:

*Ac va on, MemoryManagement, Performance, Compa bility, Debug, Generic, Interac on, BrushSe ngs, Raymarching*

In this document, we are referring to the two states of the Details panel as "Actor Details" and "Component Details".

6.4 Preset Manager and Presets

Ninja simula on parameters (eg. speed, forces, inputs) are stored in PRESET files. PRESET MANAGER is an *editor only tool* with a custom UI, made to…

A. develop / tweak fluidsim parameters in real- me

B. load / save se ngs as preset files

PresMan. is a *helper* func on, *not* needed to run ninja at all. In case you'd like to control sim via *game logic* like Preset Manager does, read *Chapter 26.: "Controlling Live in real me".*

6.4.1 A level placed NinjaLive container is associated with a *Default preset* at: *NinjaLive Actor Details /NinjaLiveComponent /LiveGeneric /DefaultPreset*

Fluidsim params stored in the *Default Preset* are loaded when a sim container ini alizes (No PresMan needed, containers are autonomous!)

6.4.2 Preset Manager could be used by placing */FluidNinjaLive /NinjaLive\_PresetManager* blueprint actor on a level. The most important UI elements are (see more at 21.1):

A. In the top row, you could enable *Tool ps* (recommended!)

B. In the second row (printed in red) you could SELECT a given NinjaLive Actor

C. In the third row (printed in blue) you could LOAD a preset for the selected actor.

The default preset of a given NinjaLive Actor is marked with \* symbol

D. Using the floppy disk icons, you could save and duplicate presets

6.4.3 More *Preset Manager related topics* in this Manual:

*10.2.2* *Preset Usage*

*14.1* *Remove Preset Manager From Levels Before Compile*

*21.1* *GUI exposed params*

*21.3.2* *LiveGeneric /DefaultPreset*

6.4.4 Preset Manager Error Handling

A. Level-placed PresMan could be de-ac vated, at PresMan Actor Details Panel.

Icon turns gray (when enabled, it is black).

B. PresMan tries to reach clients at startup. You could define the default client at the PresMan Actor Details panel. In case the default client field is empty / or client is not present / or client is asleep: PresMan tries to load someone else. Once started, you could manually select which client (which NinjaLive Actor) you would like to load for tweaking.

C. Mul ple Preset Managers per level: DO NOT

D. PresMan can't find clients - case1:

Preset Manager could go blank when it can not connect a valid ninja container. A possible cause: containers are usually set to be proximity ac vated. When the pawn or spectator camera (that triggers the proximity sensor) is away, the container is inac ve (asleep). PresMan can \_NOT\_ wake sleeping containers. This is communicated via an explicit on screen message.

E. PresMan can't find clients - case2:

PresMan recognizes clients via the Live INTERFACE -- in case you are not using a the standard NinjaLive BP - eg. you have embedded NinjaLiveComponent to your custom BP, like a pawn, make sure you have added "Live Interface" under BP/Class Defaults/Interfaces

F. PresMan can't find presets:

NinjaLive actors contain a non-dynamic reference for preset LOOKUP loca on. By default, it is: /Content/FluidNinjaLive/Presets

In case you have moved ninja to a new subfolder: update the “*PresetSearchPath*” variable in *NinjaLiveComponent* Blueprint.

G. Mul ple NinjaLiveComponents per Actor:

Preset Manager is currently *not* prepared to handle mul ple NinjaLiveComponents embedded in a Single Actor. You could try developing presets for each component separately (using duplicated actors) - and when done, simply referencing the final preset files for each Component under LiveGeneric/DefaultPreset.

6.5 Simple Painter Mode: experimental feature

NinjaLive has the poten al to track objects, detect collisions and write the data to a RenderTarget - this is what we call "Collision Painter". The func on could also accumulate and erode paint strokes by me.

As a (yet) experimental feature, we have built in a switch - SimplePainterMode - that (a) discouples Collision Painter from the rest of the system and (b) disables NinjaLive fluidsim func ons. The result is a minimal system using only 1 RenderTarget, keeping GPU load on a very low level. Combined with the regular "User Editable Output Materials" feature, SimplePainter could be used efficiently for certain VFX types.

The switch is located at *NinjaLiveComponent Details /Live General /SimplePainterMode* (at the very bo om)

Link: [See MODULE061 A-H in *NinjaLiveComponent* Blueprint]

Chart 1 Actors, Components: dependency and func ons. High.res bitmap: [LINK](https://drive.google.com/file/d/154V5Wbj0xIn-MAhkK16hicE_qob6-wkA)

7. Memory Demand

TYPICAL PERFORMANCE

LIVE 1.2 vs 1.3

GTX 1070, UE 4.23, average FPS

NinjaLive allocates memory mainly for RenderTargets. The rendering pipeline uses 8 RenderTargets for a single simula on container by default: 1 four-channel [ RGBA, for CollisionPainter ], 5 single channel [ R, for Density and ScalarFields ] and 2 bi-channel [ RG, for VectorFields ] - all set to 16 bit precision. The pipeline could be reconfigured many ways (resolu on, bit depth, channel usage).

A 256x256 fluidsim container with default se ngs allocates 1664 Kilobytes of memory - this could be crunched or expanded, depending on the needs. Similarly: 512x512 area = 6656

Level 11: SmokeChamber test scene FOUR 720 px fluidsim containers Live 1.2: 140 FPS vs Live 1.3: 230 FPS

Level 13: SmokeChamber test scene TWENTY 720 px fluidsim containers Live 1.2: 100 FPS vs Live 1.3: 190 FPS

Kbytes, 1024x1024 area = 26624 Kbytes, 2048x2048 area = 106496 Kbytes. Of course, the sim area could be non-square (eg. 128 x 512), mem consump on is changing accordingly.

Link1: See Help.uasset /Chart2 /RenderTargets for more details on possible RT configs. Link2: Select any actor containing the sim component, and switch on "/Component Details /LiveDebug /ShowMemoryManagement" for dynamic, on-screen report on actual memory consump on of a given container.

8. Performance (excerp on)

Important: Live 1.3 has been released with major performance improvements. The new results are NOT YET included in Chapter 12 / the currently available data is outdated.

The following excep ons (typical and peak) are here to represent the magnitude of changes.

PEAK PERFORMANCE

LIVE 1.3 (unlit scene, an alias and postprocessing off) GTX 1070, UE 4.23

9. High Level Se ngs: merging NinjaLive to your project Note: Live v1.1 merging process has changed - compared to v1.0. Access legacy guide: LINK

Before merging, the target project needs to be prepared! NinjaLive requires the following Project Se ngs to work properly:

STEP 1: *add custom Trace Channel*

*NinjaLiveComponent* uses a custom channel - *FluidTrace* - to perform line tracing while projec ng WorldSpace collision coordinates on the fluid simula on mesh (*TraceMesh*).

A. Go to */Edit /Project Se ngs /Collisions /Trace Channels*

B. Choose "*Set New trace channel*"

C. Name = *FluidTrace*, Default Response = *Ignore*

Note 1 : *Trace Channel Index* does *not* ma er - “FluidTrace” could be added to an exis ng set of Trace Channels, no need to re-configure the order of channels. Note 2: As a consequence of the default "ignore" response, all objects will be transparent for NinjaLive line tracers - except dedicated *TraceMeshes* in *NinjaLiveComponent* owners.

STEP 2: *enable "UV from Hit" engine feature*

Go to: */Edit /Project Se ngs /Engine /Physics /Op miza on* and enable the “*Support UV From Hit Results*” op on. Collision data is projected on the *TraceMesh* by linetrace as "Hit". Using this feat, 3D Hit coords could be converted to 2D sim UV-space instantly.

Please double check the above steps - compare your editor to this screenshot: LINK

STEP 3: *merging*

A. Quit Unreal. Copy "*/Content/FluidNinjaLive*" subfolder from the original (non-merged) Live project to the *target* project root (*/Content*) using Windows or iOS file manager, while UE is *not* running.

Emphasis: *do not* copy the *whole* NinjaLive project structure to the target project - copy *only* the “FluidNinjaLive” subfolder located in the “Contents”. *Do not* try to use Unreal *Migrate* or *Import* func ons - *copy* the above described subfolder.

B. Open *target* project in Unreal Editor. Open *NinjaLive Blueprint* with blueprint editor.

Loca on: */Content/FluidNinjaLive/NinjaLive.uasset* Press “*Compile*”, then “*Save*”.

C. Open *NinjaLiveComponent Blueprint* with blueprint editor.

Loca on: */Content/FluidNinjaLive/NinjaLiveComponent.uasset*

Set the default value of “*TraceChannel*” and “*CollisionChannel*” variables to “*FluidTrace*” See this visual guide on blueprint edi ng: LINK

Press “*Compile*”, then “*Save*”.

In case the blueprint compiles without errors, we are almost done.

D. In the level editor, select any *NinjaLive Actor* on a level. Select *NinjaLiveComponent*

(see HOW). Check the “*Live Compa bility*” group: the top 3 input fields should be set to “*FluidTrace*”.

Once done with these steps and checking: PROJECT MERGING IS FINISHED

10. Low Level Se

ngs

10.1.1 DISABLE/ENABLE

*NinjaLive actors* could be "force deac vated" using the "Disable" flag (a) Select one/more "NinjaLive" Actor on level

(b) find "LiveAc va on" param group on the Details panel

10.1 Ninjalive Actor Parameters (briefly)

Chapter 21.2 is providing a full descrip on on all *NinjaLive Actor* parameters. Right here, we are covering only a few *key* se ngs.

Once the original NinjaLive project is opened - OR - merging NinjaLive to another project is finished, *NinjaLive Actor* (sim container) is ready to be deployed to game levels.

(1) drag-n-drop NinjaLive from the Content Browser to a level - OR - (2) copy-paste already configured Actors from NinjaLive tutorial levels to your project levels - and modify them to get sa sfactory results. Start experimen ng with the se ngs, try to understand how Actors on tutorial levels are set up. You can find tutorial videos in this playlist.

When selec ng a "NinjaLive" Actor on level, the Details panel is displaying exposed variables, collected in FOUR groups

LiveAc va on: [9 params], disable switch + wake/sleep related params (ac vator, range...etc) LiveInterac on: [12 params], scale sim area + filter interac on types by class or bone name LiveDebug: enable on screen status messages

LiveComponentOverrides: helper func on to set variables when mul ple actors are selected.

See NinjaLive blueprint / GROUP1 / SUBGROUP003 for more informa on.

(c) set Disable flag state: on/off

The disable switch OVERRIDES ac va on se ngs: the Actor remains passive even if the ac vator is in range. Once disabled, the on-level Actor ICON turns gray (originally red).

10.1.2 SCALING / SIZE

Do \_not\_ try to scale the sim container actor via viewport transform gizmo - nor using the numeric *Scale Transform* input at the Actor's Details panel. Find the "Live Interac on" param group at the Actor's Details panel, and set "Trace Mesh Size" and "Interac on Volume Size" to influence the size of fluid simula on plane and the size of interac on volume, respec vely.

The default *TraceMesh*, found in *NinjaLive Actor,* is a planar mesh. For this reason, when you set "Trace Mesh Size" on the details panel, the Z-component of the input field does not have any effect on the visuals and workings. Later on, you will learn how to add a custom *TraceMesh* to your own actors. In case these meshes are not planar (eg. a hemisphere), the Z-scaling component starts to make sense - and that is the reason for that param being there.

The Interac on Volume is a true 3D object by default, so all dimensions of the "Interac on Volume Size" param field have effect on the workings.

Note: *TraceMesh* and Interac onVolume both have a separate visibility flag (at "Live

Interac on" group). Make sure that these flags are on - to visualise the men oned geometries. The two params are: "ShowTraceMeshInEditor" and "ShowInterac onVolumeInEditor"

10.1.3 NON-UNIFORM SCALING

Using the above men oned op ons, you could set up a rectangular (non-square) *TraceMesh*/ simula on plane. Since fluidsim XY resolu on is also manually provided (non-automa c), you should make sure that side-propor ons match - eg. a *TraceMesh*, sized X=4,Y=8 is matched with a Fluidsim resolu on set to 256x512 or 512x1024.

Note: you could reach fluidsim resolu on se ngs at the Details panel of *NinjaLiveComponent*. Select *NinjaLive Actor*. At the Details panel, there is a top sec on, where Actor Components are listed (some mes, to see all components, this "top list window" should be rescaled - by dragging the bo om part of the window-handle downwards). Select "*NinjaLiveComponent*". Un l the component is selected, the Details panel displays its (the component's) proper es - and NOT the owner proper es.

Find "Live Performance" submenu, set "Resolu on X/Y" respec vely.

10.1.4 POSITION & ROTATION

As opposed to Scaling: Posi on and Rota on of the sim area should be set using the viewport gizmo or the Transform input field at the Actor's Details panel.

10.1.5 LOCALISING INTERACTIONS: OVERLAP PROJECTION

Params related to this sec on are located at: Actor Details /Live Interac on

While our simula on is 2D, we would like to seamlessly integrate it to 3D space - so, we came up with the "Overlap Projec on" technique. Our fluid simula on container actor, placed on level has two related components:

(A) *Interac onVolume*: used to detect overlapping / collision - objects inside this volume could interact with the simula on - ini al and ongoing overlap events are con nuously monitored (both entering and leaving the volume). The type of interac ons could be defined / filtered by the user (see next chapter)

(B) *TraceMesh*: we are mapping the simula on output to this mesh (by default a plane, but could be defined by the user) plus: we are going to perform a line trace against this mesh.

The sim container is triggered by objects that overlap with the Interac on Volume. Once overlapping object(s) / components are targeted, a line-trace is performed in their direc on (A) from the camera or (B) from a user-defined point - against the *TraceMesh*. Line-trace hits the *TraceMesh*, the hit informa on is transformed to *TraceMesh* UV space (which is also simula on UV space), and the simula on density/velocity inputs are feeded with the related posi onal/velocity/size data.

Velocity could be derived from Frame/PreviousFrame posi onal delta or alterna vely by calling (reques ng from the engine) the world space velocity of the colliding object and transforming it to sim space. NinjaLive uses both methods. What if the mo on is perpendicular to the *TraceMesh*? NinjaLive contains a certain processor to handle this, by increasing pressure and velocity noise at the given loca on, giving an illusion of an object "blas ng through" a surface, producing shockwaves (if velo is large).

Non-perpendicular velocity vectors cause "drag-like" advec on. We always check composite velocity, and add "blast factor" propor onally to the perpendicular component.

Now, imagine that we are forcing *TraceMesh* to be camera facing. So we are mapping collisions from the camera view to a camera facing plane. And remember: we are tracking objects / bones that are inside Interac onVolume, but not aligned on *TraceMesh* surface (imagine the limbs of a character or the debris of a crumbling wall): since trace-targets are moving in 3D space and the advec on / density they generate is always posi onally following them, the fluid seems spa ally extended.

More on Camera Facing: 10.2.5

10.2 Ninjalive *Actor Component* Parameters (briefly)

10.1.6 FILTERING INTERACTIONS

Params related to this sec on are located at: Actor Details /Live Interac on - allowing you to specify / filter interac on types for the simula on area.

/Overlap Filter Inclusive Object Type: an array, where you could add/remove object type-classes: WorldSta c, WorldDynamic, Pawn, Vehicle, Destruc ble, PhysicsBody /Auto Exclude Large Overlapping Objects: eg. an extended floor object that overlaps the

Chapter 21.3 is providing a full descrip on on all *NinjaLive Component* parameters. Right here, we are covering only a few *key* se ngs.

Select any *NinjaLive-Component-Owner* Actor on level (eg. "NinjaLive" Actor or Orb, Pawn). Select *NinjaLiveComponent\** at the Actor Details panel (see *Actor Components* list at the top)

Interac on Volume, and we do not want it to interact with the fluid. You could do the same manually , under:

\*Here is a visual guide on “how to select NinjaLiveComponent” --->

IMAGE LINK

/Exclude Specific Actors From Overlap

Bone names: in case you have provided a skeletal mesh class (eg. Pawn) in the Filter array, the container is trying to detect all bones by default - most of the cases this is not what we want. "Par al name filter" allows you to type in "foot" ot "hand" - and all bones containing these strings will be included. Performance-wise, the most op mal solu on is providing "Exact Bone Names": eg. "feet\_r", "feet\_l" for a Mannequin, walking in a puddle sim.

10.1.7 WORLD SPACE VELOCITY

There is a single param for channelling World Space velocity to the simula on: eg. we have a ke le full of swirling liquid, and we start to push the ke le / or we have a pawn with flaming hands, and he starts to run, and we expect the smoke/flames to be le behind.

The param that influences this phenomena is a "preset" param, that could be changed using the preset manager,

see "Velocity Field influenced by sim area mo on" on the GUI - or modified by manually edi ng the preset file (defined at Component Details / LiveGeneric) at

No ce: as you select the actor component, the "Details" panel changes and you could see the exposed *Component Variables*, collected in 9 groups, all star ng with "Live" prefix.

Listed in order of importance:

Interac on [11 params] - con nuous (non-overlap based) interac on AND single target mode Generic [14 params] - PRESET and INPUT / OUTPUT MATERIAL defini ons + more Performance [19 params] - resolu on, LOD, render pipeline se ngs

MemoryManagement [3 params] - in case mem.manager is placed on level: set up connec on Raymarching [16 params] - picking lightsource for raymarching, se ng params

Compa bility [6 params] - system level switches, eg.: Flip RenderTargets for Mobile compile BrushSe ngs [11 params] - collision painter brush size overrides and noise se ngs

Ac va on [1 param] - component level disable switch (actor level is preferred, when available) Debug [10 params] - enable on-screen repor ng of LOD, memory...etc

10.2.1 DISABLE/ENABLE

When talking about *NinjaLive Actor*, we have TWO disable switches available:

"VeloFromSimAreaMo on" field. Actor Details /LiveAc va on /Disable (preferred)

Actor Component Details /LiveAc va on /Disable (not preferred)

Normally, we disable systems on the Actor Detail level. Once *NinjaLiveComponent* is added to a random owner (eg. pawn), the high level (actor level) disable is not there anymore. In this case, the component level disable could be used.

10.2.2 PRESET USAGE

*Component Details /Live Generic /Default Preset* - this is where you could associate a default preset for a *NinjaLiveComponent* - this preset will be automa cally loaded at area

ini aliza on. Note: in the Preset Manager, you could load / try arbitrary presets for a given area - but you have to provide DEFAULT here, at the Component Details.

10.2.3 OUTPUT MATERIAL USAGE

Output Materials: NinjaLive rendering pipeline is a chain of Blueprint controlled Materials,

wri ng to RenderTargets. Any RenderTarget could be used as output, depending on the needs (typically Density, Velocity, Pressure and PainterInput). So, a typical "Ninja Output" material contains at least one "Parametric Texture Object" with Specified name, eg. "DensityBuffer", "PressureBuffer"...etc. these materials are tapped into the rendering pipeline, and use the needed raw sim output to create arbitrary shaders.

Note: by default, the render pipeline is processing MONOCHROMATIC density data, and the output is COLORIZED - based on the available data (eg. Tonemapping monochrome density with two colors). The density could be sent to a raymarch shader or Pressure used to perform WorldPosi onOffset, Divergence for Refrac on... really up to you. Have a look at the provided output material examples at: */Content/FluidNinjaLive/OutputMaterials/*

Below the array of output materials, there is an INTEGER input field, labeled as:

*“Output Material Selected”* - provide here the array index for your default/favourite material. The array is also accessible in the Preset Manager - so you could check varia ons in run me.

Btw, run me: you could start/leave open the standard Unreal Material Instance Editor UI and tweak output material params while Ninja is running - and you see the real me fluidsim in the main viewport - very efficient way to fine-tune params!

10.2.4 RAYMARCHING

To set up *NinjaLiveComponent* for Raymarching, check *these* se ngs at *Component Details*: ● LiveRayMarching / Enable Raymarching switch

● LiveGeneric / OutputMaterials array: make sure there is a raymarch capable output

material on the list

● LiveGeneric / OutputMaterialSelected: set the array index for the above material

10.2.5 INTERACTIONS

Component Details /Live Interac on: these se ngs are partly described in the next chapter, see: STEP 9-10, Se ng up "non-overlap based" or "con nuous" interac on with the owner.

Note: Con nuous Interac on should be flagged ON every me you include Live Component to a new owner (the new host is NOT *NinjaLive Actor*, but some other class) - this flag ensures that LiveComponent could interact with the new owner. Other se ngs in this submenu:

Single Target Mode: focusing on a single superfast target (eg. pawn fist)

Character anima on could be erra c, extremely fast movements occur, the temporal sampling of the collision painter simply can not cope with this, we see density dots. This mode focuses on a single target, and connects the sampling points via interpola on, crea ng a con nuous mo on-trajectory, drawing density and velocity lines between the sparse sampled posi on data points - this way making "fluid ribbons" / "smoke or flame trails" behind the superfast moving object. The target is the first item in the "Bone names'' list - or the first element found based on type filtering.

Using the "Single Target Set Sim Speed" flag, a bullet- me like " me dila on effect" could be achieved: the moving speed of the single target influences fluidsim play speed.

Camera Facing: performed by a generic func on, stored at: */Content/FluidNinjaLive/Core/NinjaLiveFunc ons* - usually called by the sim component (see MODULE036 in the blueprint)

Important Note: when using NinjaLive as a component IN MOVING AGENTS (eg.: a Pawn),

keep Camera Facing in the sim component switched OFF and call the "Camera Facing" func on in the owner Blueprint's Tick flow. This results in smoother rota on. Reason: the cking of Pawn transform and *TraceMesh* transform should be synchronized (in case of non-moving owner sync does not ma er).

See included "mo le owner" type blueprints - referred at 10.3

10.2.6 USE CUSTOM LINE TRACE SOURCE (instead of camera)

New feature, added to NinjaLive 1.2 ---- UI access:

NinjaLiveComponent Details Panel /LiveInterac on /UseCustomTraceSource NinjaLiveComponent Details Panel /LiveDebug /VisualizeCustomTraceSource

Levels demonstra ng the feature: LEVEL 2C -- LEVEL 21 Stage 6B -- LEVEL 24A, 25, 26 Tutorial video on feature: [LINK](https://www.youtube.com/watch?v=2QlJ2f0aK5E&list=PLVCUepYV6TvOrOfQVLMCxl_JoU_cIkK8P&index=5&ab_channel=AndrasKetzer)

NinjaLive uses line tracing to project 3D collision data to the 2D simula on plane. By default, the trace line starts from the camera and targets the overlapping / colliding object. At the point where the trace line intersects the TraceMesh, the collision painter is drawing to the collision buffer. This is how overlapping objects generate density/velocity for the simula on.

Using the new feature, we could define a custom line tracing source point, by adding XYZ offset to the sim container posi on.

*Con nuing 10.2.6*

Advantage 1: no visual artefacts when looking at planar surfaces from low angle ("FPS view")

Advantage 2: detaching simula on container and simula on output - running the sim in the background while the user and the camera is somewhere else

Note1: before adding this new feature, NinjaLive was already capable to run "non-interac ve" simula ons in the absence of camera, eg. using a texture or material to generate density while ignoring overlapping objects --- see Level 24B setup!

Note2: to perform the "run a sim while we are away" feature, we need to switch off LOD and "Pause sim when not visible" --- see Levels 24A,24B,25,26 for details!

Advantage 3: no impact doubling when running the simula on on hollow meshes and viewing "from the other side". Imagine a sphere, used as TraceMesh - and an object colliding to the RIGHT hemisphere. In case we watch the impact "from the other side" the trace line (star ng from the camera) crosses the LEFT hemisphere as well, genera ng UNWANTED simula on input. In case we are NOT using the camera as trace source, this could be avoided --- see Level21, Stage 6B!

Advantage 4: we could avoid VR artefacts - when the user is rapidly changing his point of view and the overlapping object "hovers" above the TraceMesh (not exactly intersec ng the tracemesh), the projected collision point wanders (like a shadow of an "above the floor object" while you are moving the torch). By using a fixed line trace source point, this could be avoided.

10.3 Adding NinjaLive Component To Your Own Actor Classes

We are going to update an exis ng actor class to contain *NinjaLiveComponent*. Typical cases include Pawns, Vehicles and other mo le agents.

Before start, we'd like to men on: on the NinjaLive tutorial levels you'll find mul ple examples for this setup - including a floa ng orb and the classic UE-Mannequin pawn, both with *NinjaLiveComponent* added. Blueprints could be found at these loca ons (folders):

*/FluidNinjaLive/Tutorial/BP\_NinjaDemo\_MovingBall1\_NinjaComponentAdded*

*/FluidNinjaLive/Tutorial/UE\_Mannequin\_UsageExamples/ThirdPersonCharacter\_NinjaAsComponent*

Important: *NinjaLiveComponent* does \_not\_ have built in overlap detec on func onality, interacts only with predefined (user defined) owner components (eg. Bones, Sockets, Meshes). This type of interac on is labeled "non-overlap based" or "con nuous" interac on, handled by MODULE021 in the *NinjaLiveComponent* blueprint. Below, you'll find instruc ons on how to manage this (STEP 8-9, explained)

Let us start with a quick step-by-step guide, followed by explana on on the non-trivial steps:

1. open owner (the planned sim component host blueprint)

2. add component: *NinjaLiveComponent*

3. add component: *TraceMesh* (Sta cMeshComponent)

4. set the value of "TraceMeshComponent" variable of *NinjaLiveComponent* (value =

*TraceMesh*)

5. set *TraceMesh* proper es (note: scale X,Y is related to sim X,Y resolu on!)

6. add NinjaLiveInterface to the owner blueprint (at Class Se ngs /Interface)

7. set *NinjaLiveComponent* default proper es: while remaining in the blueprint editor, select *NinjaLiveComponent* in the "Components" list of owner to access *NinjaLiveComponent* Details panel. Find "Live Interac on" parameter group

8. set the "Con nuous Interac on with Owner Actor" switch to "ON"

9. set con nuous Interac on Inclusive Object Type / Bone names

10. set the default value of all other params here (in the blueprint) - eg. default preset,

material

11. enable *TraceMesh* CameraFacing (op onal)

12. save blueprint, and drag it on level from the Content Browser

STEP 3, ADD TRACEMESH COMPONENT TO OWNER

All *NinjaLiveComponent* owners must contain a Sta cMeshComponent labeled as "TraceMesh". *TraceMesh* has two func ons: linetrace is performed against it - and the fluidsim is mapped on it. Depending on the context, we could call this mesh "simula on plane" or "tracemesh" - it is the same.

Any mesh would do - NinjaLive contains a simple Sta cMesh type plane as default object, add that one for start: /FluidNinjaLive/NinjaLiveTraceMesh

STEP 4, TELL NINJALIVE ABOUT THE TRACEMESH

*NinjaLiveComponent* contains a "*TraceMeshComponent*" variable that MUST be set by the owner. Please have a look at this blueprint:

*/Content/FluidNinjaLive/Tutorial/BP\_NinjaDemo\_MovingBall1\_NinjaComponentAdded*

OnEventBeginplay, set *TraceMesh* as "value" to "*TraceMeshComponent'*' variable of

*NinjaLiveComponent*. Steps:

a. pull *NinjaLiveComponent* to the blueprint edi ng-surface

b. pull a data-flow thread from the component, and type: "*TraceMeshComponent*", place a "set" node

c. pull *TraceMesh* to the blueprint edi ng-surface - and wire it to the above "SET" node.

This way, we are explicitly telling *NinjaLiveComponent* about the *TraceMesh*, contained by the owner. From now on, the component does a lot of things with the mesh, all automa cally.

STEP 5, SET TRACEMESH PROPERTIES

a. Editor Visibility could be switched off: *TraceMesh* component details /Rendering / Visibility

b. Collision se ngs are extremely important - but *NinjaLiveComponent* handles this automa cally by force se ng all the necessary params: see MODULE004 in the sim component.

One thing that could \_not\_ be set from blueprint: *double sided*. This property should be enabled at the Mesh Editor. For our ready-to-use plane it is flagged "on''.

c. Scale: three methods to set *TraceMesh* scale

- At the level placed instance - by selec ng the component (this effects only that instance)

- In the owner blueprint, selec ng *TraceMesh* and set default transform (effects all instances) - Via an exposed variable (*NinjaLive Actor* does it this way, so you could customize instances)

Important: scale is related to simula on X,Y resolu on - see: 10.1.3 NON-UNIFORM SCALING

STEP 6, ADD INTERFACE

Add *NinjaLiveInterface* to the owner blueprint at: *Class Se ngs / Interface*

*NinjaLiveComponent*-Containing-Actors (owners) should have this interface implemented...

a. to be visible/accessible for GUI-controlled preset management

b. to be excluded from the line tracing of OTHER owners

c. to access / modify exposed variables of any level placed sim component owner

Both *NinjaLiveComponent* and *NinjaLive\_PresetManager* iden fy / collect "clients" based on INTERFACE (Get all Actors with Interface).

See: *NinjaLiveComponent*: MODULE020 (exclusion from linetrace) / NinjaLive\_PresetManager: SUBGROUP003 On Tutorial Level 5 you could find a setup where NinjaLive\_InterfaceController is communica ng with a *NinjaLive Actor* using the interface, ini a ng a "fade off / shutdown" sequence, have a look! The clients are defined in the Actor Details/ Interface Target Actors array of controller.

STEP 8-9, SETTING UP "NON-OVERLAP BASED" OR "CONTINUOUS" INTERACTION WITH THE OWNER

Next, we are going to set *NinjaLiveComponent* default proper es. While remaining in the blueprint editor, select *NinjaLiveComponent* in the "Components" list of owners - to access *NinjaLiveComponent* Details panel.

At the Component Details panel, find the "Live Interac on" parameter group. Set the "Con nuous Interac on with Owner Actor" flag to "ON".

When a sim component is embedded to its na ve host - *NinjaLive Actor* - the owner is doing overlap detec on and telling the component's line tracer "who to track". Sim component does not have built-in overlap detec on func onality. When embedded to new owner (eg. pawn), line tracer falls back to a user defined list on "who to track" - the list contains type filters

(Sta cMesh, Dynamesh, Pawn...etc) and bone/socket lists ("Inclusive Object Type" and "Bone names" ). This is all very similar to the Interac on se ngs of the Actor Detail menu - see

10.1.6 FILTERING INTERACTIONS for a general descrip on on interac on filtering and bone names.

STEP 10:

Set the default value of all other params - eg. default preset, material We have been talking about this in the previous chapter. See:

10.2.2 PRESET USAGE, 10.2.3 OUTPUT MATERIAL USAGE

While the brush size can be adjusted via Preset, you may want to tweak it in: Component Details /LiveBrushSe ngs

STEP 11: (OPTIONALLY) ENABLE TRACEMESH CAMERA FACING at: Component Details /LiveInterac on

Please note: with rota ng/mo le agents, it is be er to call the Camera Facing func on in the owner blueprint, see 10.2.5 INTERACTIONS / Camera Facing for case descrip on.

10.4 Other Ways To Include/embed/add Ninjalive To Actors

Adding *NinjaLiveComponent* to an object class is the preferred / advised way to equip any agent/object with fluidsim. However, there are other methods to do this. The updated version of this document will feature a chapter describing these methods. Briefly:

A. CHILD ACTOR COMPONENT

Unreal Engine implemented a structure not too long ago, called "ChildActorComponent". The structure enables us to add "Actor" class objects as components to another class (as opposed to adding "ActorComponent" class components). The UE implementa on looks somewhat unfinished and accessing child parameters is a bit confusing - but we can add *NinjaLive Actor* with all it's overlap and proxy ac va on management systems to any actor. We have managed to successfully embed *NinjaLive Actor* to the default Mannequin Pawn, see: *ThirdPersonCharacter\_NinjaAsChildActorComponent* in this folder:

*/Content/FluidNinjaLive /Tutorial /UE\_Mannequin\_UsageExamples*

B. SPAWNING

*NinjaLive Actor* could be spawned by code / blueprints - and a ached to the spawner, crea ng a similar situa on as if it was an "owner". The main problem is, how to (pre)define the params of the "to-be-spawned" Actor? Two ways to do this:

(a) spawn, set params instantly, re-ini alize the spawned Live actor

(b) by flagging "Expose on spawn" ON for variables in the "to be spawned" Blueprint (in this case NinjaLive), we could access them / set them before spawning - on the spawning node. We have tested this as well - see: *ThirdPersonCharacter\_NinjaSpawnedA ached* in this folder: */Content/FluidNinjaLive /Tutorial /UE\_Mannequin\_UsageExamples*

10.5 Interac on: se ng up Objects to trigger NinjaLive response

---> See Level 21 / Stage2

NinjaLive is pre-configured to handle a wide range of interac ons as "recipient" - but some steps should be taken on the "causer" side as well (causer = collider, overlapping object).

A. OVERLAP

In case you want an agent to trigger *NinjaLive Actor*'s OVERLAP sensor:

ck ON the "Generate Overlap Events" op on at Details /Collision (by default, it is OFF) Repea ng: by default, all build-in Unreal objects (eg. a sphere, a plane) are set "not-to" generate "overlap events", this se ng should be changed.

An interes ng fact [UE bug? :) ] - in case an object is INITIALLY (at BeginPlay) in the overlap zone (inside the interac on volume) of *NinjaLive Actor*, it will be detected as overlapping agent - the “generate overlap” se ng influences only "latecomers" and everybody who wants to "leave". In case you have an agent that (1) starts in the area and (2) later leaves and (3) "GenerateOverlapEvents" is NOT enabled - NinjaLive will not receive a no ce on the leaving and this might lead to anomalies.

B. SKELETAL MESHES

NinjaLive starts to track SkeletalMeshes if the "Pawn" interac on filter is ac ve/added. What if a SkeletalMesh is not a pawn? Eg.: you place a SkeletalMesh class object to level and set it to play an anima on, and it is not embedded in a blueprint? Not a problem: set the SkeletalMesh (or the object that contains it) to interact like a pawn:

Go *Details /Collision* and set "*Collision Preset*" to “*Pawn*”

In case you do not want to modify the whole collision preset, try to set only the “*Object Type*”.

11. U li es: Possess Nearest Pawn / Quality Se ngs

*NinjaLive\_U li es* Blueprint Actor could be op onally placed on game levels to help development. Symbol: a green le er N. Main features:

1. LIGHT WIDGET

Add main (direc onal) light rota on and intensity controller widget to viewport To use this feat, you need to manually add the widget and the target light.

2. EDITOR FPS

By default, UE is set to run at 120 FPS. As a simple kind of performance measurement, legacy ninja versions (un l 1.3) were UNLOCKING the UE limit, using the following console command: *t.MaxFPS 900 (900 FPS is an arbitrary, theore cal limit)*

As a result, tutorial levels were running with no FPS-limit - usually around 200-400 FPS on a GTX 1070. This was a simple, good way to have instant feedback on performance bo lenecks. With the release of RTX 3000 series GeForce cards, a problem emerged: these cards were able to run test levels on 900+ FPS --- and the card got exhausted. The “Override Editor 120 FPS limit” got switched off, by default (on most levels).

3. POSSESSING PAWN

*Possess Nearest Pawn* is a very useful func on: In case it is OFF, we have a free / unbound spectator camera while *in Play*. In case it is ON, it will find the NEAREST pawn (compared to our current posi on in the world) and possess that. Ninja tutorial levels are o en populated with mul ple pawns at different stages - it is handy to navigate in editor, press Play and instantly possess that given pawn at that given stage.

4. The u l (once placed on level) influences other SYSTEMIC things, like DOF, mo on blur,

an alias, camera and mouse-cursor smoothing.

12. Performance Tests

Important: on 2 June 2021 Live 1.3 was released with major performance improvements. The new results are NOT YET included in this chapter / the currently available data is outdated.

The excep ons in CHAPTER 8 are here to represent the magnitude of changes. You could read more about performance improvements in the ChangeLog (LINK)

12.1 Tes ng environment

Hardware: Nvidia GeForce GTX 1070, 1920x1080 HD resolu on

So ware: Win10 64bit, DirectX 12, UE 4.20, ac ve viewport switched to fullscreen, zoomed in - area of interest fills the viewport

Editor FPS cap raised (default=120) - using console command: "t.MaxFPS 999" Simula on FPS limited is to 120 (no more than 120 frames calculated, per second)

12.2 Peak performance

Test 01 mod (non-full-screen, excep on)

Single, 512x512 sim container with single collider, default sim quality Translucent, unlit material

As an excep on (compared to other tests) viewport is zoomed out, FX consumes only 25% of screen space. Average cost: 1.5 millisec per frame - viewport FPS fluctua ng around 670 FPS

Test 01

Single, 256x256 sim container with single collider, default sim quality Opaque, unlit material.

Average cost: 1.7 millisec per frame - editor viewport FPS fluctua ng around 590 FPS

Test 02

Single, 1024x1024 sim container with single collider, default sim quality Opaque, unlit material

Average cost: 1.9 millisec per frame - editor viewport FPS fluctua ng around 526 FPS

Test 03

Single, 4096x4096 sim container with single collider, default sim quality Opaque, unlit material

Average cost: 8.6 millisec per frame - editor viewport FPS fluctua ng around 116 FPS

12.3 Peak performance with Raymarch shader

Test 04

Single, 256x256 sim container with single collider, default sim quality

Translucent material with raymarch shader, processing data from 1 local lightsource Average cost: 2.1 millisec per frame - editor viewport FPS fluctua ng around 476 FPS

Test 05

Single, 1024x1024 sim container with single collider, default sim quality

Translucent material with raymarch shader, processing data from 1 local lightsource Average cost: 2.3 millisec per frame - editor viewport FPS fluctua ng around 434 FPS

Test 06

Single, 1024x1024 sim container with 10 collider Colliders: spawned, circular moving objects

Translucent material with raymarch shader, processing data from 1 local lightsource Average cost: 2.5 millisec per frame - editor viewport FPS fluctua ng around 400 FPS

12.4 Collider stress

Test 07

Single, 256x256 sim container with 10 collider Colliders: spawned, circular moving objects Default sim quality, Opaque, unlit material

Average cost: 1.9 millisec per frame - editor viewport FPS fluctua ng around 526 FPS

Test 08

Single, 256x256 sim container with 128 collider

Collider: Destruc ble mesh falling apart to chunks, each piece lands and remains ac ve in the sim area

Default sim quality, Opaque, unlit material

Average cost: 7.2 millisec per frame - editor viewport FPS fluctua ng around 138 FPS

12.5 Combined test: proximity ac va on / memory usage / medium stress

Test 09

Level configura on: 256 sim containers arranged in a 4 x 64 grid, ac vator agent moves along the array, sim containers gradually wake / go to sleep as the ac vator passes. On wake/sleep, they acquire/release RenderTargets. Ac va on range is set to guarantee: on average, we have

16 ac ve sim areas at a given me.

Sim container configura on: 256x256, one collider per container, default sim quality, opaque unlit material

Camera follows the ac vator agent, looking down / slightly towards the horizon, all containers visible. Average cost: 7.0 millisec per frame - editor viewport FPS fluctua ng around 140 FPS Peak memory consump on: 30.8 Mbytes

12.6 Maximum stress, test with LOD

Level configura on: 64 sim containers arranged in a 8 x 8 grid, all ac ve, running simultaneously

Container configura on: same setup as with peak performance test (256x256)

LOD se ngs: (1) pause off-screen sim areas, (2) distance based sim FPS reduc on, (3) distance based pressure itera on cycle number reduc on

Setup1: camera is close to the grid, 14 containers visible (distance based LOD inac ve) Average cost: 8.2 millisec per frame - editor viewport FPS fluctua ng around 120 FPS

Setup2: camera zoomed out, 64 containers visible

Average cost: 18 millisec per frame - editor viewport FPS fluctua ng around 55 FPS

12.7 Maximum stress, test without LOD

Level and sim configura on is similar to the above

Distance based LOD features are turned off. "Pause off-screen sim areas" is on.

Setup1: camera is close to the grid, 14 containers visible

Average cost: 8.2 millisec per frame - editor viewport FPS fluctua ng around 120 FPS

Setup2: camera zoomed out, 64 containers visible

Average cost: 74 millisec per frame - editor viewport FPS fluctua ng around 14 FPS

12.8 Performance measurements under UE 4.23 - 4.26

In general, higher UE versions tend to produce lower performance.

At the extremes, the difference is significant. In the typical / recommended usage cases: not too bad.

A. EXTREMES

Peak performance: an empty, fullscreen UE 4.20 editor viewport could run 700 FPS once FPS cap is removed - and it drops to 590 FPS during peak performance test. For UE 4.23 the maximum number for an empty viewport seems 400 FPS, drops to 300 during peak performance tests. Consequently, the measurable performance of an empty UE 4.23 viewport is worse than 4.20 with Ninja peak performance tests running.

Stress: running 16+ ac ve containers under 4.23+ got much worse - 30-50% loss, compared to UE 4.20 --- Performance drop cause: the handling/batching of RenderTargets has changed significantly in UE 4.23+

B. TYPICAL USAGE

Apart from the extremes, tes ng in the typical / recommended usage zone:

Hi profile single container (HD, raymarch.) performance is 5-10% worse, compared to UE 4.20 Running 4-16 containers parallel, performance is 10-25% worse, compared to UE 4.20

Conclusion: despite these losses, we are s ll good in the recommended usage zone.

13. Turning Ninjalive To A Plugin

NinjaLive is currently implemented as a "Project" - but it could be turned into a "Plugin". *Charles Cline* was kind enough to create a detailed guide for conver ng NinjaTools to a plugin (back in July 2020): the same method could be applied to NinjaLive.

Here, we are linking the original PDF:

h [ps://drive.google.com/file/d/1OugHbFj9mI3gwe8Mk1RbNQ3wsARBB8qS](https://drive.google.com/file/d/1OugHbFj9mI3gwe8Mk1RbNQ3wsARBB8qS) Thank you very much for the knowhow, *Charlie*!

14. Packaging / Compiling in general

14.1 Remove Preset Manager From Levels Before Compile

(A) Preset Manager is a developer feature: it is not needed for the final product to func on (B) PresetManager is using "Editor Only" func ons (for file management) that could block compiling - it is advised to remove it from Game Levels completely once development is finished / before compiling

14.2 Please carefully study chapter 21.3.6 *Live Compa bility* in this document

Compa bility se ngs include *Trace Channel management* and *Dynamic Texture Lookup management* - both being cri cal / having an impact in project compiling / packaging.

(A) trace channel “autofind” does NOT work in packaged projects - users need to define trace and collision channels manually before packaging - this is a global se ng (could be forced to all containers by defining trace channel variable values in the NinjaLiveComponent blueprint).

(B) dynamic texture lookup does NOT work in packaged projects

● METHOD1: define sim density and velocity input textures on a per container basis

at *NinjaLiveComponent /LiveCompa bility /Overwrite\*Input* (NOT a global se ng)

● METHOD2: go to *Project se ngs /Packaging /Addi onal Non-Asset Directories to Package* and link the folder where the velocity and density maps are, this forces the system to package the maps. (GLOBAL se ng)

14.3 NinjaLive 1.1 - 1.2 blueprints are branched to APEX and non-APEX versions

In case you are *not* using the APEX destruc on plugin, you could completely remove all APEX related assets from the project before compiling. APEX assets are located at:

*/Content /FluidNinjaLive /Versions*

You could read more about the APEX removal in the Changelog: LINK (see v1.1 changes) Important: in NinjaLive 1.3 and higher versions, all APEX related content is removed.

14.4 Mobile Rendering, Limita ons

(a) Android graphics drivers are flipping RenderTargets ver cally. Since Ninja is using a chain of RenderTargets, only ODD pieces (seem to) flip! To prepare the system for this, go to: *NinjaLiveComponent* Details /LiveCompa bility - and set "Flip RenderTargets for Mobile" to

ON (before compiling).

(b) Parallax Mapping is not working on Android Mobile with ES 3.1. Other features (like Raymarching) are tested: OK

(c) In NinjaLive 1.0 - 1.2 Tutorial levels contain Destruc ble Meshes to demonstrate the

interac on of fracturing meshes / fluid dynamics / LOD. During mobile packaging, UE returns a warning: "DM\_box01\_SphericalMapped has a LOD sec on with 150 bones and the maximum supported number for feature level ES3\_1 is 75." Since the referenced destruc ble is not used on mobile levels, you could ignore the warning - it does not block compiling / packaging.

14.3 RANDOM HINT: we could imitate mobile touchscreen in Unreal Editor, by: Edit/Editor Preferences/Level Editor/Play/Play in Editor/Use mouse for touch: ON

15. Packaging To Android, Briefly

Useful official guide from EPIC: Se ng up for development for the Android pla orm

h [ps://docs.unrealengine.com/en-US/SharingAndReleasing/Mobile/Android/Ge](https://docs.unrealengine.com/en-US/SharingAndReleasing/Mobile/Android/GettingStarted/index.html) ngStarted/i ndex.html

15.1 Prerequisites

(a) On the PC: install *CodeWorks* for Android

(b) On the Mobile: Go to /Se ngs /About device /So ware Info /Build Number: tap 7 mes to enable developer mode. In the se ngs root, search for "developer op ons", enable USB debugging.

15.2 se ng up the Unreal Editor for Android packaging

go /Edit /Project Se ngs /Packaging /Build Config: Developing OR Shipping go /Supported Pla orms: set to Android

go /Edit /Project Se ngs /Pla orms / Android: Enable APK packaging

go /Se ngs (cogwheel icon) /Preview Rendering Level: Android ES 3.1 (Shader (re)compiling!)

15.3 se ng up Ninja for Android packaging

Android graphics drivers are flipping RenderTargets ver cally. Since Ninja is using a chain of RenderTargets, only ODD pieces (seem to) flip! To prepare the system for this, go to: *NinjaLiveComponent* Details /LiveCompa bility - and set "Flip RenderTargets for Mobile" to ON (before compiling).

15.4 Packaging

go /File /PackageProject /Android /...(ASTC)

Select a Folder for Android package and auto-installer (Subfolder is automa cally generated)

(a) You could follow the packaging process in the Output window. (b) Once done, plug your phone, go to the above provided folder, and start "Install\*.bat" - (c) A console pops up with status, and in a minute process the package is copied / installed on Android. (d) The package appears at the "All Apps" list with typic UE logo.

Se ng up an Android Phone for developing / package transfer:

see UE documenta on (enabling developer mode, allowing USB access at file transfer, finding the executable) + USB usage mode: charging VS USB for file transfer

16. Planned Features

UPDATE (21 Aug 2021): with NinjaLive version 1.5.26.1, the first round of adding core features has ended. Star ng with *1 August 2021*, we are focusing on producing prac cal examples. The examples are developed in small packages: “UseCases” - see Chapter 29

LEGACY FEATURE PLANS: --- Short term

Enable memory management for SimplePainterMode blueprints

Status: blueprints in SimplePainterMode are currently autonomous (no memory manager connec on)

Update the current Memory Manager to cope with mul -resolu on RenderTargets

Status: currently Mem.Manager generates and manages RenderTGs in a single, pre-defined resolu on

Add PostProcess material output and support full 2D screen-space fluids Status: the current pipeline supports only geometry-mapped approach

--- Mid term

VR support

Further Niagara development Support for mul ple light sources

Support for Foliage: distort geometry with simula on velocity field Support for Virtual Textures

Adding more rendering-modes. Currently: (a) fluidsim, (b) simple painter. First add on candidate: (c) reac on diffusion - see LINK

(d) simple feedback mode: the current simple painter extended with a 2 or 4 rendertarget feedback loop - no fluidsim func onality, but able to deliver nice dynamic VFX

Implement "velocity based frame blending": calculate the sim on low FPS (5-25) and use the low frequency velocity data to smoothly interpolate between frames

Implement "capture unwrapped mesh" technique to improve collision painter Replace APEX destruc on support with Chaos

--- Long term

Add 3D sim capabili es

Add baking capabili es / Merge NinjaLive and NinjaTools

17. Op miza on

17.1 Use Ac va on Volume

By using the "Ac va on Volume feature", you could reduce the number of parallelly running sim containers.

Performance data suggests that we could use 1-2 high res area fx and 3-6 character fx simultaneously (on one screen) without taking over too much resources reserved for other game components. For an in-game cinema c the budget is more allowing: a single ultra HD (2k) - OR - 2-4 high res (1k) area fx - AND - 4-8 character effects.

Sim container sleep-wake state could be controlled at *NinjaLive Actor* Details panel /LiveAc va on.

When placing containers, please keep in mind: Live is not a robust "whole world system", but a system suppor ng a mul tude of dispersed, local interac ons. See chapter 8 and 12 on performance.

17.2 Use LOD

Ninja is capable of distance-based quality reduc on, regarding (a) sim sampling rate (FPS), (b) the number of pressure itera on cycles and (c) the number of tracked objects. See: *NinjaLiveComponent* Details /LivePerformance

17.3 Use Memory Pool Manager

Place a single RenderTarget Pool Manager actor on a (persistent) level and allow clients (*NinjaLiveComponent* owners) to acquire memory from the pool. The op on to enable this feature is located at:

*NinjaLiveComponent* Details /LiveMemoryManagement /AutoConnectToMemoryPool

Advantages: (a) Generate a pool of RenderTargets in advance, (b) Let waking / sleeping ninja components [acquire from] / [release to] pool, (c) Avoid lag caused by game me rendertarget crea on and destruc on, (d) Spare unreal garbage-collector load

17.4 Reduce the number of Trace Targets

NinjaLive is line tracing overlapping objects to project their posi on to the 2D sim plane.

The number of trace targets ma er - with 10-20 objects, we are fine. As soon as the number of targets exceeds 50, we could experience a 10-15% drop in performance. Op mize by reducing the number of tracked targets.

EXAMPLE1: DESTRUCTIBLES WITH MANY CHUNKS

Use the below op on to make Ninja ignore a certain percentage (%) of chunks: *NinjaLiveComponent* Details /LivePerformance /IngnoreDestruc bleMEshChunks%

EXAMPLE2: SKELETONS

Mannequin skeleton is made of 68 bones. Usually, we do not need them all - eg: for a full body collision we do not need all the fingers. By manually providing selected bone names at the below input field, we could cover the full body:

*NinjaLiveComponent Details /LiveInterac on /Con nuousInterac onBoneNamesExact*

Tip: 14 Mannequin bones covering the full body (paste these list items to “BoneNamesExact” array)

head, neck\_01, spine\_01, spine\_03, upperarm\_twist\_01\_l, upperarm\_twist\_01\_r, lowerarm\_twist\_01\_l, lowerarm\_twist\_01\_r, index\_01\_l, index\_01\_r, calf\_l, calf\_r, ball\_l, ball\_r

17.5 Disable Dynamic Asset Lookup (DAL)

Preset files could refer to bitmaps as density and velocity inputs. The preset stores only the bitmap's name and looks up the file dynamically, at sim ini aliza on. This gives flexibility to the system, we could easily swap images and experiment. On the other hand, we need to handle this, to run the game in Standalone mode and to Package the game:

(A) OPTIMIZATION

Once done with experimen ng and a preset file is "final", the dynamic parsing/lookup feature is not needed anymore - we could bypass it and "hardwire" a bitmap input to the blueprint, using the below op ons:

*NinjaLiveComponent Details /LiveCompa bility /Overwrite Preset Density Input and /Overwrite Preset Velocity Input*

The [Issues](https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM) [and](https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM) [FAQ](https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM) PDF contains a step-by-step guide on disabling DAL: *see i005*

(B) COMPATIBILITY

The velocity and density maps from the presets are dynamically loaded in real me but the package system of UE4 doesn't consider dynamically loaded assets as assets it should cook in the final package - so it doesn't include it.

In case users do not want to use the "overwrite method" (op on “A”), go to…

*Project se ngs /Packaging /Addi onal Non-Asset Directories to Package* ...and link the folder where the velocity and density maps were

*(/Content/FluidNinjaLive/Presets)*, this forces the system to package the maps.

17.6 Carefully plan Rendertarget resolu on

RenderTarget resolu on influences performance. RT resolu on could be scaled globally and selec vely at: *NinjaLiveComponent Details /LivePerformance*

17.6 Carefully choose the Pressure Solver (2 types) and solver se ngs

Simula on Pressure Field is calculated using an itera ve process. More itera ons result in more detail in turbulent structures / vor city. More itera ons also need more RenderTarget read/write opera ons per frame. Since RenderTarget write opera ons have a cri cal impact on performance, we are facing a TRADE OFF situa on: performance VS structural details

Please have a look at this demo level: NinjaLive\_Level02B\_Cri calSe ngs - demonstra ng the trade off. FIVE pressure itera ons per frame (per render cycle) seems like a good compromise -

5 is set as default for NinjaLive. You could adjust this value at: *NinjaLiveComponent Details /LivePerformance*

17.8 Reduce the number of "per Tick" refreshed Dynamic Material Params

To increase flexibility during VFX development, by default almost all fluidsim params are queried / refreshed / forwarded to sim materials ON A PER TICK BASIS.

Once your project is about to finish / you are done with param tweaking, you could eliminate dynamic (per ck) param refreshing by pu ng non-changing values on a "do once" branch, or ini alizing them on "EventBeginPlay".

Params like Noise ling and offset are typically targeted for this op miza on: probably nobody is going to animate noise dynamically in game.

17.9 Blueprint Na viza on (will be deprecated in UE5!)

Turn on *Blueprint Na viza on* to improve the performance of COMPILED (packaged) project at *Project Se ngs /Packaging* AND in the Ninja Core Blueprints, by opening the BP and visi ng the *Blueprint Class Se ngs: Na vize = TRUE*

18. Changelog / Updates

This chapter is outsourced to a separate document: Link [to](https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM) PDF

19. Known Issues & FAQ

This chapter is outsourced to a separate document: Link [to](https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM) PDF

20. Niagara

Update, 2 June 2021 - Live 1.3 has been released with more Niagara Features. You could read about these features in the ChangeLog (LINK)

20.1 Driving par cle systems using real me fluid data

As the ini al phase of planned Niagara feature development, a new Level (LV 20) has been added to NinjaLive at release v 1.0. The Niagara systems on this level serve as proof of concept: Niagara could be driven by externally generated, real me fluidsim data.

All systems are using a 2D grid setup: par cles are generated along an orthogonal grid and their ini al posi on is used as UV to map them with Fluidsim data, received from NinjaLive - by the Texture Sample Niagara Module.

*Setup 1*

Input: NinjaLive density and velocity buffers

Density is used to control par cle Color and Alpha via "Scale Color" modules Velocity is used to control par cle Accelera on

*Setup 2*

Same as Setup1, except VERTICAL par cle VELOCITY is modified by the Density buffer.

High density pushes par cles upward. As a result, par cles "burst" perpendicular to the 2D grid that spawns them and fall back as they "cool down".

*Setup 3*

Same as Setup1, but the NinjaLive preset that drives the simula on is different: a persistent brush is used, so collision strokes remain in the sim area for a long me, genera ng long smoke trails.

*Setup 4*

Only NinjaLive Density data is used (no velocity) to drive Mesh Par cle VERTICAL SCALE.

20.2 Warnings

(1) Niagara Plugin

Please note: under UE 4.23 / 4.24 the Niagara plugin should be ENABLED to view the systems on this level. In higher UE versions, the plugin is enabled by default.

(2) UE bug

Ninja-to-Niagara data flow is corrupted by a cri cal UE 4.26.0 bug - as a temporary solu on, we are using a workaround to send data to Niagara. You can read more about the UE bug at this LINK.

(3) Deprecated Niagara Systems

EPIC made many changes to Niagara since UE 4.23 came out. As we would like to maintain NinjaLive under UE

4.23 - 4.26 as a single, not-forked, compa ble dev branch... we are using Niagara Systems made with UE 4.23. In case you are opening the project under UE 4.25 / 4.26, Niagara Editor is displaying warnings that some data types are *deprecated*. Despite the warnings, the systems are up and running - and the deprecated fields / modules could be swapped with up-to-date func ons in a few minutes.

*See video demonstra ng Level 20:* LINK

20.3 STEP-BY-STEP GUIDE: LINKING NIAGARA AND NINJA

*NinjaLive* is performing real- me fluid simula on. The sim buffers could be used to drive Niagara GPU-par cle masses. For example: sim velocity field is ideal for accelera ng par cles, sim density could be used for opacity masking, height displacement and tone-mapping.

To do: we are wri ng ninja internal simula on buffer(s) to external RenderTargets and set a given NiagaraSystem to read these RenderTarget(s) as input

1. Create one or more RenderTargets in advance

These are going to serve as a data-bridge to Niagara. Right click in the Content Browser, go to "Materials and Textures", choose "RenderTarget"

2. Set ninja to write the RenderTarget(s)

At: NinjaLive Actor / NinjaLiveComponent /LiveGeneric

a. switch on "Draw Internal RenderTargets to External"

b. chose which buffers to export

c. pick an exis ng (previously created) empty RenderTarget to write

3. Once the RenderTargets are created and ninja is set up to write them:

tell Niagara to read the RTs

a. select a Niagara System on level that you would like to drive

b. locate the actual uasset that is used (Details Panel /Niagara System asset)

c. open the system with Niagara editor (double click on it in the Content Browser)

d. locate the "Sample Texture" module in the Stack

e. provide the pre-created RenderTarget as input in the "Texture" field

Ease of use: we could handle Niagara texture input as a parameter - so we can provide RenderTarget input on the *Actor Details* panel (no need to edit actual Niagara Systems when picking RenderTargets)

A. create a Texture-Sample type "User Parameter" in the Niagara System

B. provide this param as input to the "Sample Texture" module

C. set the value of this param by selec ng the on-level niagara system,

then going to the / Details Panel / Overrides

21. Complete List of NinjaLive parameters

21.1 GUI exposed params & func ons 47

21.2 NinjaLive Actor params 21

21.3 NinjaLive Actor Component params 94

21.1 GUI exposed params

Recalling chapter 6.4 on Preset Manager / Presets

By (1) placing */Content/FluidNinjaLive/NinjaLive\_PresetManager* on a level and

(2) pressing Play (“running the game”) you could acces a custom GUI side panel, layered over the level editor viewport, designed to manage

fluidsim related NinjaLive params:

A. Select the sim container that you would like to work with

B. Modify param values real me - while monitoring the changes in the viewport

C. Load, save, duplicate and modify parameter collec ons, stored as preset files

D. Set one the experimentally created preset files as *default preset* to a given NinjaLive instance on level - later on, the container is loading this default preset autonomously when ini alized, no Preset Manager is needed

Please note: *GUI exposed params* are in fact *NinjaLive Component params*, that

have *\_not\_* been exposed on the Blueprint Details panel - but made accessible for GUI

*Second, blue roll down menu at the sidebar top* PRESETS

NinjaLive is configured by 140 public variables

- Most variables (100) are exposed at blueprint details panel - Fluidsim specific params (40) are saved/loaded as a preset

2. Op onally, templates could be referenced by absolute paths, (eg.: /Game/Textures/T\_noise) by manually pas ng the path to a DataTable (a preset file)

Check LEVEL 2A / STAGE 14 to see how it works.

*Floppy disk icon*

SAVE PRESET

*Topmost, red roll down menu at sidebar top* NINJALIVE SIMULATION CONTAINER ACTORS LIST

Quick browse list via keyboard up/down arrows. The selected Actor is highlighted on level.

Actors are listed IF matching the following criteria:

1. placed on level

2. enabled (non-disabled)

3. containing NinjaLiveComponent

4. equipped with NinjaLive Interface

Each container is associated with a default preset. This is marked with a star [\*] besides the given preset.

Default preset could be changed (a) permanently at *NinjaLiveComponent Details /LiveGeneric* ... or

(b) temporarily by clicking the blue “star and arrow” picto on the right. This "temp default" condi on is for preview only, to link a given preset to a given container while the user is changing back-and-forth between containers. The "temp default" se ng is being erased as you quit NinjaLive GUI.

Presets are stored as DataTable uassets and could be modified (a) via Preset Manager or (b) by manually edi ng the DataTables

Presets for a given instance of NinjaLive are collected from a predefined folder (and its subfolders). Preset lookup loca on could be changed at

*NinjaLiveComponent Details /LiveGeneric*

Presets are recognized if… (a) Placed under pre-defined lookup loca on (+ all subfolders) --- (b) Properly named: [FileType]\_[ModuleName]\_[PresetName]

Eg.: DT\_NinjaLive\_Flame1

Presets could reference other assets (density and velocity textures) and pull them as input for simula on. Asset refs are called "Templates".

1. Normally, the preset contains only the name of the referenced asset and the preset loader looks for the template in the same folder where the given preset file is located - it is recommended to keep presets and their belongings together! (Preset Packages)

Instantly updates the original preset with current se ngs. Note: before closing Unreal Editor, updated preset files must be wri en to disk permanently via "File / Save All"

*Double floppy disk icon* DUPLICATE PRESET ("Save as...")

Writes current se ngs to a new file and asks for a file name. File naming: [FileType]\_[ModuleName]\_[PresetName]

Eg.: DT\_NinjaLive\_Flame1

Note: before closing Unreal Editor, newly created preset files must be wri en to disk permanently via "File / Save All"

*Floppy disk icon with line through* REMOVE PRESET

Instantly removes preset from list. Default preset (with \*) can't be removed

The removed asset is preserved as: DT\_RemovedPreset\_[original preset name]

*Blue star with upward poin ng arrow on top* SET CURRENT PRESET AS TEMP DEFAULT

Sets the currently loaded preset as TEMPORARY default to the selected NinjaLive Component.

IMPORTANT: default could be set PERMANENTLY on the details panel of a given level actor. The "temp default" condi on set via GUI control is for preview only - to link a given preset to a given container while the user is changing back-and-forth between containers. The "temp default"

se ng is being erased as you quit NinjaLive GUI.

Op onally, you could try pressing "K" on the keyboard - this calls the "Keep Simula on Changes" UE func on - tries to save the changes made to an actor to the actor's default state. The UE func on seems unstable.

Next we are going to describe GUI exposed numeric parameters.

NOTE ON PARAM OVERDRIVE

NinjaLive comes with parameter range-constraints that allow users to OVERDRIVE certain params. Reason: overdrive could look cool - in some cases. But it could also ruin the visuals.

*See LEVEL 2B in the project - demonstra ng overdrive.*

Rule: params behave op mal in the [-1]-[0]-[+1] range. GUI displays cri cal params in RED (out of recommended range). Be careful when boos ng them. This is especially true for velocity params, that behave addi vely - and produce rude glitches if set to high values.

*Parameters in the green “Brush” param group*

Size

Blueprint variable: BrushSize Range: 0.01 - 5

Note: apart from the this GUI param, brush size could be selec vely set for various object classes at: *NinjaLiveComponent Details /LiveBrushSe ngs*

Eg.: a pawn draws 0.5 stroke, while primi ves use 0.1 stroke

Density

Blueprint variable: BrushStrength Range: 0.001 - 10

Note1: a dense brush also draws stronger velocity signal Note2: when inver ng the brush, density below 0.1

makes it prac cally "invisible" while s ll dragging the fluid.

Enhance this by se ng "drag" above 3 Hardness

Blueprint variable: BrushHardness Range: 0 - 1

By increasing this, brush so edge gets thinner

BrushNoise

Blueprint variable: BrushNoise Range: 0 - 1

Brush density noise

PosRnd

Blueprint variable: BrushRnd Range: 0 - 1

Brush posi on noise

Persistence

Blueprint variable: InputFeedback Range: 0 - 1

Persistence sets brush stroke erosion, value = 1 means temporally stable, non-fading brush stroke.

Related param: simula on "Feedback"

Invert

Blueprint variable: EraserMode

Bool switch, subtracts brush density from the sim density field. Low density makes brush "invisible" while dragging the fluid. To invert velocity, use nega ve "drag" and "puncture"

Drag

Blueprint variable: VeloFromBrushMo on Normal range: [-1] to [+1]

Overdrive range: [-20] to [+20]

OVERDRIVE MIGHT CAUSE GLITCHES, BE CAREFUL! Adds the brush-masked velocity of colliding objects to the simula on. Nega ve values invert direc on.

Puncture

Blueprint variable: BrushPuncture Overdrive range: [-2] to [+2] Recommended range: [-0.1] to [+0.1]

Puncture adds radial velocity to the brush, modulated with strong noise. The result is a "boiling" effect - designed for sta c or slow moving objects, that otherwise would not produce visually interes ng turbulent advec on.

*Parameters in the yellow “Sim” param group*

Feedback

Blueprint variable: FlowFeedback Range: 0 - 1

Live render pipeline is a feedback loop. Feedback = 1 means no dampening, no fade-out, density informa on accumulates.

Related param: brush "Persistence"

Clear buffers

Clear all RenderTargets in the pipeline / blank Canvas

Speed

Blueprint variable: Speed Range: 0.01 - 10

Simula on state is stored in RenderTargets. The spa al propaga on of density/velocity informa on is the result of sampling RenderTargets with a given offset. "Speed" is a sampling-offset mul plier: larger values result in greater offset and faster propaga on. Note: sampling frequency

( mestep) also influences sim speed.

Turbulence

Blueprint variable: Divergence Range: 0.01 - 1.3

Velocity field divergence modifier

Larger values result more turbulent flow

*Parameters in the purple “Fields” param group*

Amplitude

Blueprint variable: VeloStrength Range: -/+

Velocity field strength mul plier Nega ve values invert direc on

Has \_no\_ effect on "drag" and "puncture"\* \*(stroke velo separated from field velo)

Offset X

Blueprint variable: VeloOffsetX Range: -/+

Adds horizontal force to velo field

Offset Y

Blueprint variable: VeloOffsetY Range: -/+

Adds ver cal force to velo field

Amp.Noise

Blueprint variable: VeloAmpNoise Range: 0 - 1.5

Adds noise to velocity field amplitude \_No\_ influence on velo direc on Intended use: introduce inhomogeneity, dampen strong direc onal forces

Dir.Noise

Blueprint variable: VeloDirNoise Range: -/+

Adds curl noise to velocity field direc on Intended use: introduce swirling

DirNoiseSize

Blueprint variable: VeloDirNoiseSize Range: 0.01 - 10

Curl noise scale

DirNoiseOffs

Blueprint variable: VeloDirNoiseSpeed Range: -/+

Curl noise dynamics offset

Any non-zero value animates the offset

Velocity Field from Bitmap

Texture based velocity field template, that could be transformed by "Rotate" / "Tile" and animated by "Offset"

Note1: By default, the preset loader looks for the template in the same folder where the given preset file is located.

Op onally, templates could be referenced by absolute paths (eg.: /Game/Textures/T\_noise) by manually pas ng the

path to the preset DataTable file in Content Browser. Check LEVEL 2 / STAGE 14 to see how it works.

Note2: template references could be "hardwired", transferred from the preset file directly to the blueprint. See Manual (Help) 17.5: DISABLE DYNAMIC ASSET LOOKUP

Warning, yellow triangle

Occasionally you could see a yellow triangle besides the Texture input field.

This is a warning: the referenced bitmap is NOT in the same folder as the preset file / is referenced with a manually inserted absolute path.

The path is not auto-upda ng: in case the referenced texture asset is moved/deleted the link is broken.

Rotate

Blueprint variable: VeloRotate Range: 0 - 1

Rotate velocity field vectors

0.25 means 90 degree, 1 is a full turn

Offset

Blueprint variable: VeloInputOffsetSpeed

Range: [-1] to [+1]

Velo field Dynamic offset

Any non-zero value animates the offset

Tile

Blueprint variable: VeloInputTile Range: -/+

Velo field ling: 0 = default, x > 0 means zooming out

x < 0 means zooming in

Offset From Sim Area Mo on

Blueprint variable: OffsetFromSimAreaMo on Range: -/+

Global iner al frame of reference: sim container WorldSpace posi on delta is used to offset simula on buffers.

Useful when moving objects leave trails or marks behind.

Velocity Field influenced by Sim area mo on Blueprint variable: VeloFromSimAreaMo on Range: 0 - 1

Global iner al frame of reference: sim container WorldSpace velocity is subtracted from the simula on velocity field.

Fluid behaves inertly when the area moves to a certain direc on. The iner a effect could be further enhanced by: *NinjaLiveComponent Details /LiveInterac on /LegacySimAreaMo onEffectsSimDensity*

Tile

Blueprint variable: DensityTxtScale, Range: -/+

Density field ling: 0 = default, x > 0 means zooming out, x < 0 means zooming in

Density Field from Bitmap

Texture based density field template, that could be transformed by "Scale" and animated by "Offset" (below)

Note1: By default, the preset loader looks for the template in the same folder where the given preset file is located.

Op onally, templates could be referenced by absolute paths (eg.: /Game/Textures/T\_noise) by manually pas ng the

path to the preset DataTable file in Content Browser. Check LEVEL 2 / STAGE 14 to see how it works.

Note2: template references could be "hardwired", transferred from the preset file directly to the blueprint. See 17.5: DISABLE DYNAMIC ASSET LOOKUP

Offset X

Blueprint variable: DensityTxtOffsetX, Range: -/+ Density field Dynamic horizontal offset

Any non-zero value animates the offset

Offset Y

Blueprint variable: DensityTxtOffsetY, Range: -/+ Density field Dynamic ver cal offset

Any non-zero value animates the offset

Density Field from Material [Preview]

Simula on density could be driven by (animated) Materials

Important: these materials are here to PREVIEW / DEBUG, and the actual selec on is \_NOT\_ stored in the preset. Input Materials list and defaults could be managed at: *NinjaLiveComponent Details /LiveGeneric*

See LEVEL3 /STAGE 4-5-6 for examples

See */Content/FluidNinjaLive/InputMaterials* folder for example materials

*Density Field Params*

Amplitude

Blueprint variable: DensityTxtMult Range: 0 - 10

Scales density field amplitude, both texture and Material input

Amp.Noise

Blueprint variable: DensityInputNoiseAmp Range: -/+

Adds a cloud-like mul scale noise to the density field

NoiseOffset

Blueprint variable: DensityInputNoiseOffset Range: -/+

Density field Dynamic offset

Any non-zero value animates the offset

NoiseTile

Blueprint variable: DensityInputNoiseTile Range: -/+

Density field noise ling: 0 = default, x > 0 means zooming out

x < 0 means zooming in

Sim area border is colliding with fluid Blueprint variable: SimEdgeBouncyness Range: 0 - 1

Makes sim area border behave like a wall, stopping / reflec ng fluid dri

Sim area border density fading (Pow) Blueprint variable: FadeDensityAtSimEdge Range: 0.01 - 1

Kills density near the sim edges

Sim area border density fading (Width) Blueprint variable: EdgeMaskWidth Range: 0.05 - 1

Defines the width of sim edge area, where density is being killed --- 0.5 means "from edge to center"

Output: filter sim output via Material [Preview] User defined Output Material, based on arbitrary combina on of simula on buffers.

Important: these materials are here to PREVIEW / DEBUG, and the actual selec on is \_NOT\_ stored in the preset. Output Materials list and defaults could be managed at: NinjaLiveComponent Details /LiveGeneric

Tip1: have a look at a base material to see how it works, eg.: /Content/FluidNinjaLive/OutputMaterials/BaseMaterials/M\_ NinjaOutput\_Basic

Tip2: OutMaterial Instances could be edited real me, while NinjaLive / Gameplay Scene is running, a quick way to fine-tune colors, alpha...etc.

SavePaintBuffer

Event Dispatcher: CallOnDensityMapSave Save density and velocity paint buffer

to the same folder where the preset is located

SaveSimBuffer

Event Dispatcher: CallOnDensityMapSave Save density and velocity simula on buffer

to the same folder where the preset is located

MetaInfo

Blueprint variable: MetaData

Add your notes, instruc ons, author data here

END OF GUI PARAM DESCRIPTION NEXT: NINJALIVE ACTOR PARAMS

21.2 *NinjaLive Actor* params

When selec ng a "NinjaLive" Actor on level, the *Details panel* is displaying exposed variables, collected in FOUR groups

LiveAc va on: [9 params]

disable switch + wake/sleep related params LiveInterac on: [12 params]

scale sim area + filter interac on types LiveDebug: enable on screen status messages

LiveComponentOverrides: helper func on to set variables when mul ple actors are selected.

21.2.1 LiveAc va on

DisableBlueprint

Disabled sim containers do not ini alize.

SimAc vatedByPawnProximity

Container "disabled" state is controlled by the proximity of a user defined, arbitrary object (called "Ac vator"). If NO

Ac vator is defined: using actually possessed pawn.

If container is disabled while running (eg. the Ac vator le the ac va on area), state is not preserved, container shuts down, memory returned to the manager. Re-ini alized by Ac vator proximity (by calling "RePlay").

ShowAc va onVolumeInEditor Show Ac va on Bounds

Ac va onVolumeSize Set Ac va on Bounds

Ac vatorProximityCheckFrequency Ac va on response me:

the frequency of Ac vator proximity tracking.

Default value is 0.1 second.

Ac vatorType

Ac vator Collision Object Type (eg. Pawn, Vehicle, Dynamic)

Ac vator

User defined agent, ac vates given container by proximity

TraceMeshInac veBehaviour Visual proper es of Disabled state

21.2.2 LiveInterac on

ShowTraceMeshInEditor

(1) Fluid sim Output Material is mapped on TraceMesh

(2) Line trace is performed against TraceMesh to project 3D collision data to the 2D sim

TraceMeshSize

Default TraceMesh is a planar mesh. For this reason, when you set size, the Z component of the input field does not have any effect on the visuals and workings.

You can replace TraceMesh with a non-planar object, and the Z-scaling factor could be used.

UserInputBasedInterac on

If disabled, mouse and touch gestures are ignored

OverlapBasedInterac on

If disabled, collision/overlapping is ignored

UseTraceMeshAsInterac onVolume (added to Live 1.1!) Choose between two objects for overlap detec on: *Interac onVolume* (default) and *TraceMesh*

Interac onVolume is prac cal when we would like to track moving objects in 3D. TraceMesh is advised when we need precise surface-hit posi ons. Note: TraceMesh "Collision Complexity" also ma ers - see FAQ doc for details.

ShowInterac onVolumeInEditor Show/hide volume

Interac onVolumeSize

Interac onVolume detects collision/overlapping. By default, its X,Y size should be the same as the TraceMesh. It is a true 3D volume, Z extension also influences behaviour. Z-scale enables the sim container to detect objects in front of and behind the TraceMesh 2D plane.

OverlapFilterInclusiveObjType

Inclusive filter for Collision Object Type: only listed classes could cause overlap events

AutoExcludeLargeOverlappingObjects

Ignores colliders that are larger than the sim area

ExcludeSpecificActorsFromOverlap

Name specific actors to exclude them from overlap detec on

OverlapFilterInclusiveBoneNamePar al

Type strings like "feet" or "hand" to include all bones containing the given string in their name

OverlapFilterInclusiveBoneNameExact

Name specific bones to inclusively track them by the overlap detec on mechanism.

This is the suggested way to handle skeletal mesh collision.

IgnoreDestruc blesBeforeFracture

Destruc bles need a hit event to start falling apart. NinjaLive could ignore destruc ble chunks,

before they start to fall apart.

21.2.3 LiveDebug

Ac va onEventsDebugPrint SaveDebugTextToLog DebugTextLifeTimeLength

21.2.4 LiveComponentOverrides

(14 params)

21.3 *NinjaLive Actor Component* params

Recalling Chapter 10.2

Select any *NinjaLive-Component-Owner* Actor on level Select *NinjaLiveComponent* at the Actor Details panel. Here is a visual guide on selec ng NinjaLiveComponent: IMAGE [LINK](https://drive.google.com/file/d/10cHjZsM2alob8ZcfCYtDCNTHhYGHJLSK)

No ce: as you select the actor component, the "Details" panel changes, revealing *Component Variables*, collected in 9 groups - listed in order of importance:

LiveInterac on [13 params] - con nuous (non-overlap based) interac on, single target mode LiveGeneric [14 params] - preset and input / output material defini ons

LivePerformance [19 params] - resolu on, LOD, render pipeline se ngs LiveMemoryManagement [3 params] - set up memory pool manager connec on LiveRaymarching [16 params] - picking lightsource for raymarching, se ng params LiveCompa bility [6 params] - system level switches, eg.: flip RenderTargets for mobile LiveBrushSe ngs [11 params] - collision painter brush size overrides and noise se ngs LiveAc va on [1 param] - component level disable switch

Debug [10 params] - enable on-screen repor ng of LOD, memory...etc

21.3.1 LiveInterac on

Con nuousInterac onWithOwnerActor THIS FLAG IS OBLIGATORY ON, IN CASE

NINJALIVECOMPONENT IS EMBEDDED TO A CUSTOM OWNER. (custom = NOT NinjaLive Actor) NinjaLiveComponent does \_not\_ have built in overlap

detec on func onality, interacts only with predefined (user defined) owner components (eg. Bones, Sockets, Meshes). This type of interac on is labeled "non-overlap based" or "con nuous" interac on, handled by MODULE021 in the NinjaLiveComponent blueprint.

Con nuousInterac onInclusiveObjType Inclusive filter for Collision Object Type

Con nuousInterac onBoneNamesExact Inclusive filter for specific bones/sockets

SingleTargetMode

STM is a special mode of Collision Painter, op mized to draw the mo on trajectory of very fast objects, by interpola ng between available posi on samples.

See LEVEL 2 / STAGE 3.

LIMITATION: only a single object / bone could be tracked

SingleTargetType

Primi ve OR Bone/Socket. Rule: the target is the first item in the "Con nuousInterac onBoneNamesExact" list or the first primi ve element found by type filtering.

SingleTargetMoveSetSimSpeed

A bullet- me like " me dila on effect" could be achieved: the moving speed of the single target influences fluidsim play speed

See LEVEL 2 / STAGE 5

SpeedInfluenceFactor

Blend between normal sim speed and " me dila on effect" defined speed.

0 = 100% normal speed, 1 = 100% dilated speed

CameraFacingTraceMesh

Forces the TraceMesh to con nuously look towards the Player Camera. TraceMesh transform is performed by a shared, public func on, stored at: */Content/FluidNinjaLive/Core/NinjaLiveFunc ons*

Important: when using NinjaLive as a component IN MOVING AGENTS (eg.: a Pawn), keep Camera Facing in the sim component switched OFF and call "Camera Facing" func on in the owner Blueprint's Tick flow. This results in smoother rota on. Reason: the cking of Pawn transform and the TraceMesh transform should be synchronized.

CameraFacing\_LockY-Axis

TraceMesh Ver cal axis locked (forced to remain ver cal). Result: a "billboard" like camera facing.

TraceMeshTranslucentSortPrio

A. In case you’d like to priori ze a translucent tracemesh against other translucent objects:

NinjaLiveComponent /LiveInterac on /TraceMeshTranslucentSortPrio - set prio here.

B. In case you’d like to force translucent TraceMesh in front of everything, including opaque objects: you need to modify an Output-Base-Material property. Go to */Content/FluidNinjaLive/OutputMaterials/BaseMaterials* , pick the related basemat, enter mat.editor, in the mat. proper es panel (on the right) find: "Disable Depth Test". Note: in the Material Instance (using the given base.mat), the "Depth Fade Switch"

must be OFF!

UseCustomTraceSource (added to NinjaLive 1.2)

Force NinjaLive to use a USER DEFINED Line Trace source POSITION - instead Of Camera posi on.

Explained in Chapter 10.2.6 + see Level 2C in the project.

CustomTraceSourcePosi on (added to NinjaLive 1.2)

The custom line trace posi on is RELATIVE to NinjaLiveComponent OWNER posi on. Add offset here.

LegacySimAreaMo onEffectsSimDensity

This feature is meant to support "Velocity Field influenced by Sim area mo on" FX (available on the Preset Manager GUI) - see GUI param descrip ons for more info!

21.3.2 LiveGeneric

DefaultPreset

This is where you could associate a default preset for a NinjaLiveComponent. This preset will be automa cally loaded at sim area ini aliza on. Note: in Preset Manager, you could easily load / swap presets as you develop visuals, but you have to provide DEFAULT here, at the Component Details.

PresetSearchPath

Presets for a given instance of NinjaLiveComponent are collected from a predefined folder (and its subfolders). Preset lookup loca on could be changed here. This could be used to separate presets.

PresetNameFilterCriteria

Presets in the lookup folder are recognized IF properly named.

The name consists of THREE parts: [FileType]\_[ModuleName]\_[PresetName]

Part 1 is fixed (DT, DataTable).

Part 2 could be changed here - this tag could be used to make presets INCLUSIVE for a given container - so they will be displayed in Preset Manager ONLY when this specific container is selected / they won't show up for other containers.

Part 3 is arbitrary, displayed as a preset name on the PresetManager GUI.

Sugges on: this method also could be used to separate hundreds of presets to groups.

OutputMaterialInvisible

Forces Null Material on TraceMesh: all func ons work (including collision) except the TraceMesh is not visible. Used when driving Volumetric containers with ninja - hiding 2D sim - so only the volume is visible

OutputMaterialSelected

The index of DEFAULT Material in the "OutputMaterials" array *(see below)*

OutputMaterials

An "Output Material" is (1) the final stage of NinjaLive Rendering pipeline, (2) mapped on the simula on TraceMesh, (3) used to COMPOSITE simula on buffers to a final visual, set colors, translucency, shading and such.

NinjaLive rendering pipeline is a chain of RenderTargets, all accessible for "Output Materials". Since Output Materials are user editable, many versions could co-exist --- this array func ons as a "my favourites" selec on: users could add elements to the list --- and all these elements are listed on the Preset Manager GUI, where they could be easily swapped, compared, used to check RenderBuffers...etc.

Pick output material examples from /Content/FluidninjaLive/OutputMaterials

InputMaterials

Simula on density could be driven by (animated/dynamic) Materials. Since Input Materials are user editable, many versions could co-exist, this array func ons as a "my favourites" selec on: users could add elements to the list --- and all these elements are listed on the

Preset Manager GUI, where they could be easily swapped for tes ng.

See LEVEL3 /STAGE 4-5-6 for examples

See /Content/FluidNinjaLive/InputMaterials folder for example materials

InputMaterialSelected

The index of DEFAULT Material in the "InputMaterials" array

RGBInputMaterial

Simula on could be driven by user defined materials. Default: using material RED channel to drive simula on density --- IF the RGB flag is TRUE, Ninja uses Material BLUE channel for density, and RG for velocity

InputSceneCaptureCamera

Simula on density could be driven by SceneCapture Camera. IF a valid camera is provided, its input OVERRIDES Material based input. Visit LEVEL 3 / Stage 7 to see how it works!

InputMediaPlayer

Simula on density could be driven by Local or Streaming Video, you could link your pre-configured media player here.

InputMediaSource

Simula on density could be driven by Local or Streaming Video, you could link your media file pointer here.

SimAreaClamp

Simula on RenderTarget handling method: Clamp or Wrap

RandomizeNoiseOffsets

In case you have mul ple containers placed nearby and would like to avoid visual similarity, you could randomize noises.

RandomizeDensityTextureOffset

In case you have mul ple containers placed nearby and would like to avoid visual similarity, you could randomize density texture offset. Pre-requisite: leable density texture

SimplePainterMode

NinjaLive has the poten al to track objects, detect collisions and write the data to a RenderTarget - this is what we call "Collision Painter".

SimplePainterMode (a) discouples Collision Painter from the rest of the system and (b) disables NinjaLive fluidsim

func ons.

The result is a minimal system using only 1 RenderTarget, keeping GPU load on a very low level. Combined with the regular "User Editable Output Materials" feat, SimplePainter could be used efficiently for certain VFX types.

See 6.4 in the Manual / Help.uasset See LEVEL 7

DrawInternalRenderTargetToExternal

Draws a selected simula on buffer to a user defined external, on disk RenderTarget uasset, per ck.

New feature (introduced in Live 1.5): not only simula on buffers, but the user defined *OutputMaterial* could be wri en to external RenderTarget as well.

InternalRenderTargetsToExport

Source buffer for drawing / saving

ExternalRenderTargets

Target file for drawing / saving

CollisionMask

Use textures to define sim area shape. Collision Mask is a Black-and-White texture with "Alpha'' type encoding. Black areas repel fluid-currents and behave like an obstacle. Works ONLY if "Sim-area-border-is-colliding-with-fluid" Preset Parameter is set to 1. Ideal for masking sta c environmental objects - eg: an altair in the middle of a fog-covered area.

AllowAbsoluteBlackDensity

Absolute ZERO black density input is causing visual artefacts when used as input for Parallax Occlusion Mapping. For this reason, Ninja uses "very dark gray" as density bo om.

By flagging this op on, 0-black could be forced.

PressureEdgeMasking

Range 0-1 --- When n>0, forces the pressure-buffer values to zero near the sim edges. Useful when pressure projected on a larger surface - eg. when making local ripples on a global water-surface

CoreSimMaterials

This array references ALL core simula on materials. You

could op onally replace them, in case you are experimen ng with your own core material versions.

21.3.3 LivePerformance

Resolu onX

Sim horizontal resolu on

Important: in case the container acquires RenderTargets from the MemoryManager, the local resolu on se ngs are OVERRIDDEN by mem.manager resolu on se ngs!

Resolu onY

Sim ver cal resolu on

UsePressureSolver1 (Default is 2)

Wri ng RenderTargets is a performance bo leneck in UE --- Pressure Solver1 is performing calcula ons using many RenderTarget writes --- Pressure Solver2 is op mized to keep the number of RT-writes low ------ Un l the sim.resolu on is below 2K pixel, Solver 2 is superior. Above 2K, Solver 1 starts to perform be er.

PressureSolver1\_MaxItera ons

Range: 1-32 --- Default is 5 --- Pressure Solver 1 visual quality could be improved by performing more itera ons (more RenderTarget writes)

PressureSolver2\_MaxItera ons

Range: 1-4 --- Default is 1 --- Pressure Solver 2 is op mized to keep RT-writes low. More itera ons result slightly more details - but (1) SUBOPTIMAL and (2) sensi ve to Divergence Overdriving --- use only for CINEMATIC rendering --- See LEVEL 2B for visual tests

PressureSolver2\_KernelReduc on

Pressure Solver 2 checks large neighborhood areas around a given texel to calculate average values. By reducing "Kernel size", the number of examined neighbours could be crunched --- increasing performance, but and altering visual quality

MaxSamplingFPS

The maximum number of fluidsim cycles per second. LOD scales fluidsim between min/max sample ranges. The value also constraints "CollisionPainter" temporal sampling: lower sampling causes fast moving objects

pain ng "discrete points" instead of con nuous lines - this could be avoided by switching on "Single Target Mode" (STM) at NinjaLiveComponent Details /Live Interac on. STM is drawing lines between sparse sample-points

MinSamplingFPS

The minimum number of fluidsim cycles per second. LOD scales fluidsim between min/max sample ranges.

HalfResPressureAndDivergenceBuffers

Calcula ng pressure on lower resolu on is op mal, but the method is sensi ve to simula on SPEED adjustment: in case we increase sim speed above 1 (while HalfRes is ON) visual glitches emerge --- Note: in case a given container is receiving sim buffers (RenderTargets) from the Memory Pool, the Memory Manager RESOLUTION se ngs overwrite local values (including the HalfRes flag)

LOD1-ReduceItera ons

Allow view-distance based reduc on of fluidsim pressure-subcycles

LOD2-ReduceSamplingFPS

Allow view-distance based reduc on of fluidsim cycles per second. Min-Max ranges are defined by "Min/Max SamplingFPS" params

LOD-FarBound

Set LOD max distance: outside this area the LOWEST sampling values are used. Note: this se ng is NOT related to ac va on volume size

LOD-NearBound

Set LOD min distance: within this area the HIGHEST sampling values are used

Note: this se ng is NOT related to ac va on volume size

LOD-CheckFrequency

LOD response me (frequency of view-distance change tracking) - default value is 0.1 second.

PauseSimWhenNotVisible

Pauses (freezes) sim when the player is not looking at it, and con nues when focus returns. State is preserved while paused (as opposed to "Disable", that erases state)

WaitBeforePause

Time before pausing the sim (when out of player focus)

SimSpeedAdjustmentLatency

As LOD reduces sampling FPS, the fluid would seemingly slow down.

To counter this, we are increasing simula on RenderTG SampleOffset, so density and velocity informa on propagates faster. At the end, sim remains normal as LOD kicks in - Default value is 0.1 sec

Important: when "Bullet Time FX" is on ("SingleTargetMoveSetSimSpeed"),

Sim speed adjustment is controlled by collider velocity!

StopUsingPainterCanvasWhenIdle

Halts collision painter RenderTarget overdraw when NO collision

Experimental\_PSolver2KernelIndexOffset

NinjaLive Pressure Solver v2 uses TWO type of kernels: (A) Gaussian for the feedback blurring of Pressure Buffer and (B) Custom for the blurring of primary, raw Divergence data --- Default Kernel size is 15 --- KernelIndexOffset adjusts Gauss Kernel size, rela ve to the Custom Kernel size --- Eg: a "KernelReduc on" mul plier of "0.5" reduces both Kernel to

7 (from 15) - and a "KernelIndexOffset" of "-2" sets the Gauss Kernel to "two level higher" to size "11" --- this param is experimental, allows us to seriously ABUSE the pressure solver - use carefully!

Experimental\_PressureFeedback

There is a feedback component in the pressure calcula on. "PressureFeedback" is a mul plier in this process, lower values dras cally kill the feedback component, resul ng short lived pressure waves

21.3.4 LiveMemoryManagement

AutoConnectToMemoryPool-IF-Found

Search for Memory Manager and connect when found

SelfService-IF-PoolEmpty

ON: container could manage its memory autonomously OFF: container relies on Memory Manager and skips

ini aliza on if pool is empty or no manager found

21.3.5 LiveRaymarching

EnableRayMarching

To set up NinjaLiveComponent for Raymarching: (1) enable this switch, (2) please make sure that you have a "Raymarching capable" material added to this list: LiveGeneric /OutputMaterials and (3) the material set as container DEFAULT via "OutputMaterialSelected" index

LightDirec onProvider

Pick any object (not necessarily a light source): its transform (posi on OR rota on) will be used to calculate raymarched self-shadows direc on.

LightDirec onSourceIsRota on\_NOT\_Pos

If TRUE, LightDirec onProvider's ROTATION is used to determine light direc on. If FALSE, LD.Provider's rela ve POSITION is used as light direc on

DistanceBasedLightA enua on

If TRUE LightDirec onProvider's posi on is used as center, to map light strength with a radial gradient

A enua onPower

Power func on applied on light a enua on gradient

TwoSidedShading

If TRUE, the light direc on angle is weighted by TraceMesh up vector

TwoSideBlendPow

Influences the transi on between "light" and "dark" sides of TraceMesh

OffsetLightVector

Offset the direc on of light vector with a constant

ForceManualSunPosi on

If TRUE, user provided light-source direc on is used

SunLa tude

User defined light source direc on, X component

SunLongitude

User defined light source direc on, Y component

SunHeight

User defined light source direc on, Z component

21.3.6 LiveCompa bility

Note: these params are extremely important, properly se ng them guarantees that compila on and cooking produces correctly working executables / apk files!

PreferredTraceChannelName

User defined *Line Trace Channel NAME* for a given container. The container automa cally tries to find the defined channel using the Project Se ngs. In case AUTOFIND fails, the container uses the values set in the *TraceChannel* and *CollisionChannel* ENUMS (below) as a "fallback" op on. Default value = “FluidTrace”

Experience shows that the na ve UE func on used to search / look up trace channels works fine in PIE and Standalone mode, and *DYSFUNCTIONAL* in compiled exe/apk files. For this reason, it is very important to set the two “fallback” variables before project compiling.

TraceChannel and CollisionChannel

These variables store a trace channel index: “1”

The index is NOT shown: the UI representa on displays the project defined *Trace Channel NAME* belonging to index 1. Once Ninja is merged to another project, it is important to (re)set this variable to an index that is associated with the “FluidTrace” Trace Channel.

To put it simple: set these variables to “FluidTrace”

SimPrecision

Fluid sim data precision bit depth: 8/10/16/32.

8/10 bit modes are experimental and need quite different sim params to produce nice results.

FlipRenderTargetsForMobile

Android graphics drivers are flipping RenderTargets

ver cally. To counter this, developers need to "pre-flip" RenderTGs BEFORE packing the project to mobile / VR

UseUnrealNa veEventTick

NinjaLive (by default) generates its own ck signal, but could be forced to use Unreal Engine cking. See NinjaLiveComponent Blueprint / BLOCK2 / MODULE027, 028

LimitUnrealNa veEventTick

Set max value for ck frequency ( ck per second). This affects NinjaLive when "UseUnrealNa veEventTick" is on.

OverwritePresetDensityInput and OverwritePresetVelocityInput

Important: Live uses dynamic asset lookup in PIE to access simula on input TEXTURES (density and velocity maps). Dynamic asset lookup DOES *NOT* WORK in STANDALONE mode and in COMPILED EXECUTABLES / APKs.

(Re-phrasing: the velocity and density maps from the presets are dynamically loaded in real me but the package system of UE4 doesn't consider dynamically loaded assets as assets it should cook in the final package - so it doesn't include it.)

To provide a given container with the needed input textures in Standalone mode / following compila on, there are two methods:

METHOD1: we need to properly fill the above input fields. METHOD2: we need to force UE to cook the texture folder

Read more about texture inputs, dynamic asset lookup and Standalone mode --->

In this document:

14. Packaging / Compiling

17.5 Disable Dynamic Asset Lookup (DAL)

In the [Issues](https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM) [and](https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM) [FAQ](https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM) PDF: *see i005 (issue 005)*

21.3.7 LiveBrushSe ngs

BrushScaledInverselyByTraceMeshSize

Brush size could be inversely scaled by canvas (TraceMesh) size. Result: a bigger canvas results in a similar brush as a small one.

BrushScaledByInterac ngObjSize

Brush size could be scaled by colliders: smaller object draws thinner stroke

GlobalBrushScale

Scales all other se ng

UserInputBrushScale

Brush size modifier for user gestures

Primi veObjBrushScale

Brush size modifier for Primi ves

SkeletalMeshBrushScale

Brush size modifier for SkeletalMeshes

Destruc veBrushScale

Brush size modifier for Destruc bles

BrushDensityNoiseScale

Sets CollisionPainter BRUSH default density noise scale

BrushDensityNoiseFreq

Sets CollisionPainter BRUSH default density noise anim-freq

BrushVelocityNoiseScale

Sets CollisionPainter BRUSH default velocity noise scale

BrushVelocityNoiseFreq

Sets CollisionPainter BRUSH default velocity noise anim-freq

DampenBrushBelowThisVelocity

Range: 0-1 --- By default, Brush is "Always ON" un l a target overlaps --- Using this THRESHOLD VALUE, we can FADE or KILL brush below a given velocity --- Example1: a pawn walks in a fluid, and we do NOT want to generate waves or turbulence when the pawn stands - set value to 0.01 --- Example2: we would like to scale/fade the brush with velocity con nuously - set value to 1 ------- ZERO means the brush is always on, and not scaled by velocity

DampenBrushFactor

Range: 0-9 --- This value influences how velocity scales brush density --- Near-zero values mean: even the smallest velocity causes MAX brush density --- Large values mean: only the highest velocity values scale up brush density

BrushVelocityPow

Range: 0.1 - 9 --- Control how raw brush velocity is being forwarded to the sim --- n=1 means no change, n>1 mean lower velocity values get killed, n<1 means lower velocity values get scaled up

SimAreaMo onEffectsBrushPuncture

The Z component of sim area mo on (perpendicular to TraceMesh) mul plies PUNCTURE

21.3.8 LiveAc va on

DisableComponent

If NinjaLiveComponent is embedded in NinjaLive actor, this switch is overridden by Actor level disable. In case you embed LiveComponent to your custom class, use this to disable the system.

21.3.9 LiveDebug

ShowDebugMessages-MemoryManagement

Enable on-screen debug messages for mem.management

ShowDebugMessages-CollisionAndTracing

Enable on-screen debug messages for collision and line tracing

ShowDebugMessages-LODIni al

Enable ini al on-screen debug messages for LOD

ShowDebugMessages-LODRun me

Enable con nuous on-screen debug messages for LOD

ShowDebugMessages-InterfaceControl

Enable on-screen debug messages for NinjaLive Interface events

ShowDebugMessages-RenderTargetExport

Enable on-screen debug messages for render buffer expor ng

SaveDebugMessagesToDefaultLog

Saves all debug messages to UE default log text file

DebugMessagesLife me

Define life me of on screen debug msgs

ShowMouseCursor

This se ng is overridden by NinjaLive U ls, Preset Manager and other containers.

(effects cursor only when a single container is placed on level, without managers)

ShowVelocityDebugCone

Displays a velocity direc on oriented debug wireframe arrow of the given collider

IgnoreInterfaceCommands

NinjaLive\_InterfaceController commands could be ignored.

Chart 2 Fluid simula on: data flow. High.res bitmap: LINK

22. Live fluidsim pipeline: technical descrip on

Update on 2 June 2021 - NinjaLive 1.3 released

To improve PC and Mobile performance, a new pressure solver has been introduced and simula on pipeline data flow has been modified.

22.1 About the new Pressure solver (briefly) 22.2 About the data flow changes, briefly

*Detailed descrip on in the ChangeLog PDF, Live 1.3:* [LINK](https://drive.google.com/file/d/17oVPVEoaW6Y6YKNISr4S0uUJY4_Yx_FM)

Wri ng RenderTargets is a performance bo leneck in UE. Pressure Solver v2 changes the balance in a tradeoff situa on - Compared to v1 solver - v2 performs more

opera ons per pixel and less RenderTarget read-write ops (see Level 02-B for compare)

All Pressure-solver related params could be accessed at:

*NinjaLiveComponent Details Panel /LivePerformance* (see Tool ps for details!)

Comparing the performance of v1 / v2 pressure solvers *(Live 1.2 vs 1.3)* GTX 1070, UE 4.23, average FPS

Level 11: SmokeChamber test scene FOUR 720px fluidsim containers

A. Density and Velocity pipelines have been merged (both BaseMaterials and RenderTargets): RGBA RenderTargets are being used, velocity mapped on RG channels, density on B channel

RGBA RenderTargets: *RT\_Composite, RT\_Advec on, RT\_Painter*

B. Dedicated Divergence RenderTarget is eliminated. The divergence Material is wri ng its

output to *RT\_PressureDivergence*, GREEN channel.

C. Classic Ninja Pressure solver replaced with a new version (v2) --- v1 solver is s ll

available: *NinjaLiveComponent /LivePerformance /UsePressureSolver1* FLAG

Solver v2 does *not* perform repeated itera on: calculates the pressure field in two steps - so pressure calcula on is not a "subcycle" anymore - but two regular steps in the

Live 1.2: 140 FPS

vs

Live 1.3: 230 FPS

execu on chain. The divergence and pressure materials are using bi-channel RG-RenderTargets, with pressure mapped on R, divergence on G.

Level 13: SmokeChamber test scene TWENTY 720px fluidsim containers

Live 1.2: 100 FPS vs Live 1.3: 190 FPS RG RenderTargets: *RT\_PressureDivergence, RT\_PressureDivergenceTemp*

22.3 Original fluidsim pipeline descrip on (valid for Live 1.0 - 1.2):

NinjaLive rendering pipeline is a chain of Blueprint controlled Materials >>> wri ng to RenderTargets >>> set up to form two cross-talking feedback loops: density & velocity. The numbers below are references to the included Chart (No.2).

1,2,3,4

Sim could be driven by textures, materials, user gestures, camera input and most importantly: objects overlapping / colliding with the simula on area. Collision data (posi on and velocity) is transformed from WorldSpace to SimSpace and fed to the CollisionPainter Material, wri en to a RenderTarget then injected to the simula on feedback loop.

5,6

Compositor Materials could mix / add / subtract various inputs ( eg. genera ng a cloud via Material Input (No.3) and subtrac ng Collision Input (No.2) as if colliders are "cu ng through" the cloud). Note: Material 6 is merged with Material 11 for prac cal reasons.

7,8

Following the merging of inputs by compositor materials, the data is ready for advec on: transport in velocity defined direc on. Technically, we do this by modifying the standard texture sampling UV coordinate grid (two linear gradients) with the velocity map - to offset sampling posi on. Subtrac ng the scaled velocity from UV causes a given grid-cell to read it's content from inverse flow defined direc on and distance - the sampled density and velocity informa on tends to "move over the grid" as the simula on advances.

Note1: why do we sample from the inverse direc on? Have a look at Reference No.2 - key insights on the sca er vs gather method, related to this ques on Note2: the velocity field drives its own advec on (besides density advec on).

9 - 10 - 11

This operator group plays an important part in construc ng and maintaining simula on velocity fields - by extrac ng divergence, using it to generate pressure field gradient and feeding it back to the simula on cycle.

9

Divergence is a key concept, that could be loosely defined as: "the rate at which 'density' exits a given region of space" (Ref.1) or "a measure of how much fluid enters or leaves each

(simula on) cell" (Ref.3), or the amount of local "compression and expansion" (Ref.2). Precisely: "the volume density of the outward flux of a vector field from an infinitesimal volume around a given point (Ref.4).

It is important to note that our fluid model describes the dynamics of incompressible fluids - and we expect constant pressure / homogeneous density over the field. Confusingly, divergence and the data generated from divergence is used in a context that makes it interchangeable with the concept of pressure - but it describes a local property of the velocity field (we could call it "local tension").

Technically, we calculate divergence by sampling the fluid velocity vectors in the four adjacent cells around a given texel, performing subtrac on of horizontal & ver cal neighbours separately (right-le , top-bo om) and adding the results. The divergence of a vector field is a scalar field with no direc onal informa on, only float data (like a height map).

Examples for possible null states: (1) neighbour vectors are null. (2) neighbour vectors point to the same direc on / same length, the result of both H/V subtrac on is zero. This could be the case when a cell is in a homogeneous unidirec onal flow. (3) the sum of horizontal and

ver cal components cancel each other, eg. horizontal vectors poin ng inwards, ver cal vectors poin ng outwards. In all cases (1-3) influx == ou lux, no divergence. Nega ve /

Posi ve divergence values indicate non-equilibrium state, a tension where influx =! ou lux, somewhat analogous to local (texel neighborhood) pressure building up or decreasing.

We have at least two reasons to reduce divergence of the velocity field. Theore cally: we need to keep pressure (near) constant. Prac cally: velocity field modulates our sampling offset in

the advec on module - and a highly divergent field with non-gradually / irregularly changing neighbour values causes visible "texture stretching" ar facts - or noise at the extreme.

10

Pressure cycle

One way of reducing local divergence is spa al dispersal / relaxa on. We are employing the Jacobi method to perform this, a recursive func on: (1) adding four adjacent neighbour texel values, (2) adding nega ve divergence, (3) dividing the total sum by the number of neighbours (in this case four) and finally using the func on output + raw divergence as input for the next cycle. Note: subtrac ng divergence and adding nega ve divergence results the same: compressed areas are represented by posi ve values and expanded areas with nega ve values during the itera on.

In common sense, the itera on process could be compared to "diffusion": peak divergence values (both nega ve and posi ve) "spread out", forming "ramps" or "gradients". A er N

itera on, we have a scalar field with "dispersed divergence"... s ll hesitant to say it is pressure (we could call it tension). Note: the dispersal turns (integrates?) local tension to a global field.

11

EXPERIMENTS

To validate the above statements, I have performed experiments using Ninja core - and found very similar experimental setups at the SideFX homepage (Ref.2) - published as part of a Pyro tutorial, Pressure projec on part - please have a look.

1. Forcefully injec ng outward poin ng velocity field:

posi ve divergence builds up mid field (like "pressure drop")

2. Forcefully injec ng inward poin ng velocity field:

nega ve divergence builds up mid field (like "pressure increase")

3. Forcefully injec ng a posi ve divergence spot mid-field:

inward poin ng velocity field builds up

4. Forcefully injec ng a nega ve divergence spot mid-field:

outward poin ng velocity field builds up

Gradient (note: this material is merged with No 6. for prac cal reasons) Observa ons:

Using the scalar field as an input, we calculate the gradient (the direc on of ascension on the "pressure landscape") by performing subtrac on on the horizontal / ver cal neighbours separately - and using these par al results as components of a direc onal vector. An example for horizontal direc on: if the pressure is increasing from le to right (right neighbour is larger than le ), value is posi ve, and if the pressure is increasing from right to le (le neighbour is larger than right), value is nega ve.

Using inverse gradient gives us the direc on of "descent" on the pressure landscape (instead of ascension) - and this seems matching the intui on: the direc on of advec on is the inverse of pressure gradient, flow goes from high pressure zones towards low pressure zones. By adding the inverse gradient to the exis ng velocity field each simula on step we are (1) facilita ng a divergence driven advec on and (2) correc ng (countering, elimina ng, dampening) field divergence at the same me.

● Divergence and velocity are connected in a feedback loop and have

induc ve effect on eachother

● If we remove the forceful maintenance of a certain extreme, the effects seem to

cancel out each other and the system returns to equilibrium state

But who comes first in a "realis c" situa on? By realis c we mean adding moving colliders to an otherwise neutral field (eg. throwing a pebble in a pond). Modifying the field directly is not a realis c situa on. We could decide this ques on using an experimental setup

(also demonstrated at sideFX page, Ref.2)

1. Imagine a 2D null velocity field, for direc ons we use up/down, le /right nota on

2. All field vectors zero and an object mid field, s ll / fixed posi on

3. The object suddenly starts to move upwards, with constant speed

4. The movement injects upward poin ng (posi ve, ver cal) velocity to the field.

5. At the ini al simula on step (when the object first moved) the texels immediately above the moving area show non-zero divergence: comparing their neighbour vectors, we have influx from below, and no ou lux at the top. Divergence is nega ve - we are compressing.

6. The area below the object shows posi ve divergence (decompression / expansion): the bo om row that has been "le behind" by the object is filled with upward poin ng velocity - and the row below that is zero, and this config gives back posi ve divergence values.

7. Intui vely, we would expect these states to propagate

8. Indeed: the Jacobi dispersal immediately (in the first round of larger simula on cycle) makes a gradient from the sharp (single row) high divergence values both in the upper and bo om areas. As we invert this gradient to see descent direc on, we find that both regions (above and below the moving object) contain upward poin ng direc onal vector components, but the direc on of horizontal components is inverted: the upper zone propels advec on "up and out" while the bo om zone propels advec on "up and inward" - just like in real fluids.

As SideFX folks put it: "Pressure projec on adds veloci es around its sides to channel the flow back to the bo om of the circular region." - see [image](https://www.sidefx.com/docs/houdini/images/pyro/pressure_projection/pp_test_up1.jpeg) link

CONCLUSION

In prac ce, it is the divergence-pressure-gradient operator group that constructs extended, proper velocity field during the process of pressure projec on.

*I would like to say thank you to Morten Vassvik* :)

REFERENCES

1. GPU Gems, Chapter 38

h ps://developer.nvidia.com/sites/all/modules/custom/gpugems/books/GPUGems/gpugems\_ch38.html

2. SideFX, Understanding how pyro works,

h [ps://www.sidefx.com/docs/houdini/pyro/background.html](https://www.sidefx.com/docs/houdini/pyro/background.html)

3. Content-Driven Mul pass Rendering

[h](https://www.youtube.com/watch?v=QGIKrD7uHu8) ps://www.youtube.com/watch?v=QGIKrD7uHu8

4. Wikipedia, Divergence

h [ps://en.wikipedia.org/wiki/Divergence](https://en.wikipedia.org/wiki/Divergence)

5. Divergence and Curl

h ps://www.youtube.com/watch?v=rB83DpBJQsE

6. Wikipedia, Flux

h [ps://en.wikipedia.org/wiki/Flux](https://en.wikipedia.org/wiki/Flux)

23. Using Sequencer to control NinjaLive Actors

*Related example content: Level 5, Stage 4 (added to Live 1.0.0.4) Related Tutorial Video:* [*LINK*](https://www.youtube.com/watch?v=dWJlKj446jQ&list=PLVCUepYV6TvOrOfQVLMCxl_JoU_cIkK8P&index=6&ab_channel=AndrasKetzer)

NinjaLive interac ons and behaviour could be keyframed by *UE Sequencer*.

In general, we could animate (1) *effectors* - any agent that triggers NinjaLive response and (2) *sim parameters* - internal variables that change fluidsim behaviour and

(3) *output material instance parameters*. Case “3” is not described in this document.

23.1 Effectors

An object / mesh could be effector if...

A. "*Generate Overlap Events*" op on at *Actor Details /Collision* is flagged ON

(by default, it is OFF) --- *See 10.5*

B. their "*Collision Object Type*" is set to one of the supported types

( *WorldSta c, WorldDynamic, Pawn, Vehicle, Destruc ble, PhysicsBody* )

C. the class filters of the triggered (overlapped) container (NinjaLive Actor) are set to

respond to the type of the given object. *See 10.1.6*

If an effector meets the above criteria and is keyframed to collide / overlap with a NinjaLive container it should trigger fluid response.

(Make sure you have properly set "Interac on Volume" size) --- *see 10.1.5, 21.2.2*

Note 1: Live generates fluidsim at run me. To preview effector-fluid interac on, you should run the game (press play in editor). In case you are running Live in a Viewport, you could fit Sequencer to a nearby panel and edit keyframes / move the me-slider while the sim is running.

Note 2: One method to run the already keyframed sequence automa cally at Play (and set the sequence to repeat) is to edit the *Level blueprint*. See Screenshot: LINK

23.2 Parameters

*NinjaLiveComponent* is using many params - most of them are set *only once*, at ini alisa on - these can not be animated/keyframed (typically system variables). A subset of params (typically fluidsim params) are being refreshed every ck - and could be efficiently animated. These params also appear on the Preset Manager UI. *See 21.1 for a complete list.*

Important: *NinjaLiveComponent* blueprint params are by default

NOT accessible for the sequencer.

A. To make a given fluidsim param accessible for the sequencer:

open */Content/FluidNinjaLive/NinjaLiveComponent* blueprint, look up the param in the "*Variables*" list, select it, go to the Details panel (in the blueprint editor) and

flag *"Expose to Cinema cs"* = ON. Compile and Save the blueprint.

B. To animate "Exposed" fluidsim params, add a given NinjaLive Actor to the sequencer

Track list (+), then add NinjaLiveComponent as subtrack (+), then add the (pre-emp vely) exposed param as subtrack (+)

23.3 Cinema cs Rendering via MOVIE RENDER QUEUE (UE 4.26+)

NinjaLive is working fine with the new UE 4.26 Movie Render Queue With the predecessor Render to Sequence it does not work!

A step-by-step guide

1. Edit /Plugins /MovieRenderQueue: ENABLE PLUGIN

2. Open the level that you would like to render

( *NinjaLive\_Level08\_Demo\_Roots* is ideal for tes ng: single container, no proximity ac va on )

3. Select "BP\_NinjaLive\_U li es" on level and on the Details Panel:

switch OFF the "Possess Nearest Pawn" flag

4. Content Browser: right click /CreateAdvancedAsset /Anima on /LevelSequence

5. Place Actor Panel: drag a camera on level ---> set the camera to the needed posi on

6. Content Browser: double-click on the level sequence

7. In Sequencer: right click, "Actor to Sequencer"

---> add the recently placed Camera from the Level

8. Viewport, Top-Le corner UI /Perspec ve roll down menu: select Camera Actor

9. Window /Cinema cs /MovieRenderQueue (MRQ)

10. In MRQ, click on the green "Render" roll down menu

--> add the previously created Level Sequence

11. Adjust Output Folder and Config - if needed

(by default: JPEG sequence, to "/Saved/MovieRenders/..")

12. Check 23.4: QUALITY NOTES

13. Press green "Render (Local)" in the Bo om-Right corner

14. A er a few seconds of ini aliza on, MRQ renders the image sequence to the provided

folder

23.4 IMPORTANT NOTES ON QUALITY

Note 1: on many levels, sim containers are Proximity Ac vated (work only when the spectator/pawn is close) - and the cinema c camera does not necessarily trigger the proximity sensor. In this case, your sim container is passive. Disable proximity sensor at:

*NinjaLive details panel /LiveAc va on*

Note 2: many mes, distance based LOD is enabled on sim containers (lowers quality) To make sure you render the sequence using the best available quality, disable LOD at: *NinjaLive details panel /NinjaLiveComponent /LivePerformance /LOD bool flags*

Note 3: ninja is performing calcula ons in the 60 FPS range. When the Movie Render Queue is set to 30 FPS, it hurts ninja visual quality. Set MRQ rendering FPS to 60 and (if needed) achieve lower frame rates by skipping odd/even frames when composi ng the output.

Related chapter in the Manual: Chapter 23, Sequencer (LINK)

Related tutorial videos:

FluidNinja LIVE - Using SEQUENCER to animate Objects and Simula on Containers ([LINK](https://www.youtube.com/watch?v=dWJlKj446jQ&list=PLVCUepYV6TvOrOfQVLMCxl_JoU_cIkK8P&index=22)) Improve Your Renders With Unreal Movie Render Queue (LINK)

24. Volumetrics

*Example content: Level 23-28 (added to Live 1.2) + Level 30-32 (added to Live 1.4) Tutorial Videos: LINK1, LINK2 + Live 1.4 Volume Smoke quick preview: LINK3*

NinjaLive could drive 3D volumetric systems with real me 2D fluid sim data: users could set up responsive fog/smoke containers on level and paint dynamic cloud structures. Simula on density is used as a height-field, combined with 3D noise that is advected by sim velocity. Output is a true 3D volume, shaded by scene ligh ng. Comparing the three available systems:

24.1 INTEGRATION

Driving Volumetric Systems is a new, experimental feature - for this reason setup components are purposely *kept external* (not integrated to Ninja blueprints) - so users could customize setups without blueprint modifica ons. Setup components:

1. On-disk (pre-generated) RenderTargets - to push fluid data towards Volumetric Systems

Added to NinjaLive version Lowest supported UE version UE na ve (built in) system Material Domain

Volume bounds defini on Spa al usage

Self shadows

FOG

1.2 4.23 True Volu Mesh Local

-

SMOKE

1.4

4.23

-

Volu Mesh Local True

CLOUD

1.2 4.26 True Volu UVW Global True

2. Manually defined volume bounds (Fog: Sta cMesh boxes, Smoke: Blueprints )

3. Volumetric domain BaseMaterials and Material Instances

4. Special UE Actors placed on level, needed to u lize UE Volumetrics (*Exponen alHeightFog* Actor, *VolumetricCloud* Actor)

Wri ng to *external* RenderTargets is managed on the following Ninja UI panels: *NinjaLive Actor Details Panel /NinjaLiveComponent /LiveGeneric /...*

Lit by direc onal light Lit by point light Receives Shadows Frustum aligned grid\*

- True True True

True

True

-

-

True

-

-

-

*DrawInternalRenderTargetToExternal InternalRenderTargetsToExport ExternalRenderTargets*

(enables the feature) (array of sources) (array of targets)

Volumetric system EXAMPLES are using the following premade RenderTargets:

Note1: UE na ve volume fog is implemented as \**froxels* - voxel array stretched by the cone-shaped camera frustum. This means the rela ve (per square unit) resolu on of the grid is ge ng lower by distance. For this reason, NinjaLive VolumeFog Base Material contains a “fade by distance” func on - highly advised to use it. Note 2: the spa al and temporal resolu on of fog could be adjusted by console commands, listed on level 23 r.volumetricfog.gridpixelsize is the most important (default = 8) - set to 5 by the *Level Blueprint* on LV23

FOG: CLOUDS: SMOKE:

*RT\_VolumeFog1-4 RT\_VolumeClouds1-8 RT\_VolumeSmoke1-2*

24.2 DEFINING VOLUME BOUNDS & POSITION

A. Volumetric Fog: a simple *Sta cMesh Box* is placed on level, equipped with Volumetric Domain material, reading NinjaLive output RenderTargets - see *"VoluCube"* meshes listed in the *WorldOutliner* and placed *on Level*

B. Volumetric Clouds: bounds and posi on are defined within the material (no mesh

needed), using UVW coordinate offset and scaling

- Select "VolumetricCloud" Actor on Level

- Go to Actor Details Panel, and double-click on "Cloud Material"

- In the Material, locate CloudU-Offset, \*V-offset and \*Al tude params

C. Volume Smoke: bounds are defined by a level-placed blueprint

Blueprint name: *VolumeSmokeContainer*

A debug box visualizes volume extension. To resize the volume, simply scale the level placed blueprint actor (use *uniform* scale).

Volume loca on is anchored to the blueprint actor

24.3 TILING SIM SPACE: FULL SKY / GROUND COVERAGE

Live 1.2.26.5 contains a new feature: sim space *ling* (by default OFF) Usage and setup demonstrated on a new level: 24C

The feature enables us to compute only a small part of the sky (or ground-fog) - and use this part as a “ le” - the sim space is wrapped - so, we could cover the whole area (sky or ground) using repeated pa erns.

Tiling could be set up / switched ON by tweaking the following params:

1. NinjaLiveComponent /LiveGeneric /SimAreaClamp bool flag - set to FALSE

2. PresetManager /SimAreaBorder se ngs - set all params to ZERO

3. VolumeFog and VolumeCloud material instances /Tiling bool flag - set to TRUE

More info: on se ng up sim-space ling: Level21 / Stage3 Tutorial videos on ling - specific: LINK1 / generic: [LINK2](https://youtu.be/aIfebrrR6CM?list=PLVCUepYV6TvOrOfQVLMCxl_JoU_cIkK8P&t=40)

24.4 STEP-BY-STEP GUIDE: LINKING VOLUMETRICS AND NINJA

*NinjaLive* is performing real- me fluid simula on. The sim buffers could be used to drive volumetrics. For example: sim density could be used to define volume density, height and ex nc on. Sim velocity field is ideal for driving the flow of detail-noise.

To do: we are wri ng ninja *internal* simula on buffer(s) to *external* RenderTargets and set a given volumetric material to read these RenderTarget(s) as input

1. Create one or more RGBA RenderTargets in advance

These are going to serve as a data-bridge to Volumetrics. Right click in the Content Browser, go to "Materials and Textures", choose "RenderTarget".

2. Set ninja to write the RenderTarget(s)

At: NinjaLive Actor / NinjaLiveComponent /LiveGeneric

a. switch on "Draw Internal RenderTargets to External"

b. chose which buffers to export

c. pick an exis ng (previously created) empty RenderTarget to write

3. Place special UE Actors on level

a. For *VolumeFog*: place a single Exponen al Height Fog Actor on Level.

Set the "Volumetric Fog" op on to ENABLED.

Important: VolumeFog materials ARE *NOT* DEFINED in the Fog Actor

4. Once special actors placed on level, RenderTargets created and ninja is set up to

write them... tell the volumetric materials to read the RenderTargets

a. In the case of *VolumeFog*, the volume-materials are applied on the level-placed Sta cMeshes (see Level 23, browse "VoluCubes" in the WorldOutliner). Important: mul ple meshes could be placed on level, and each mesh could be equipped with unique material. You could place your own mesh on level (any Sta cMesh will do). Cuboids are prac cal + could be scaled non-propor onally. It is suggested to duplicate one of the example material instances, and provide the duplicate with new RenderTarget Input - then, apply the material on the cuboid mesh.

Do not forget to switch on "VolumeFog visualisa on" in the editor (Ctrl+F), in UE 4.26.2 it is switched off by default.

b. In the case of *VolumeClouds*, the volume-material is directly provided in the VolumetricCloud Actor (no need to place a volume-mesh). You could set scale, offset and many other params in the material. Do not forget to set the input RenderTargets!

c. In the case of *VolumeSmoke*, the used Volume Material Instance and the input RenderTarget are defined on the NinjaLive\_VolumeSmoke blueprint details panel. Depending on the param flags in the VolumeSmoke Material Instance, we should dis nguish SIX material states (marked with No 1-6).

b. For *VolumeClouds*: place a single VolumetricCloud Actor on Level. Important: VolumeCloud material (one per level) IS DEFINED in the VolumeCloud Actor

VolumeNoise: FALSE VolumeNoise: TRUE

UNLIT

1

4

POINT-LIT

2

5

DIRECTIONAL-LIT

3

6

c. For *VolumeSmoke*: place NinjaLive\_VolumeSmoke blueprint on Level.

Cases 1-4 are fast, ready for real- me gaming usage. Cases 5-6 are somewhat slower, mainly developed for Cinema c / NextGen usage.

24.5 ON DEMO LEVEL CONTENTS

Level 23

Volume Fog

Levels 24 ABC, 25, 26, 27, 28 Volume Clouds

Levels 30, 31 Volume Smoke

Spa al arrangement

On Levels 23, 26, 30 and 31 the sim containers (detec ng collision) occupy the same loca on where the driven volume is. This setup enables actors to *seemingly* interact with the volumes. In fact, they interact with sim containers - and these drive volumes.

On levels 24A, 24B, 24C and 25 sim containers and volumes are *spa ally detached:* (containers are running "somewhere else" / "behind the stages'') - this setup style is used when we do not need direct interac on between volumes and scene actors / we would like to "playback" a predetermined sequence of events - eg. a cloud vortex forming when summoning the final boss.

For “background” containers, we also need “non-camera based” line tracing - a new feature, introduced in Live 1.2 ---> explained in Chapter 10.2.6 + see Level 2C

Sim output display

In a classic NinjaLive setup, TraceMesh has two separate func ons:

(1) It is capturing collision data (intersec ng with the line tracer)

Interac on

In a classic NinjaLive setup, TraceMesh captures collision input - and displays sim output at the same spa al loca on - we always see interac ng objects and the simula on together. This setup allows us to conveniently *use the CAMERA as line tracing source*. Volumes could be easily detached from the sim container and display sim output "somewhere else": in these cases, the camera can *not* be used as line tracing source. Reason: when we see the volume, we don't necessarily see the sim container - eg. watching a skydome with clouds, while the container is "behind the scene".

To resolve the problem, NinjaLive 1.2 introduces *User Defined Line Trace Source*.

(1) UI:*NinjaLiveComponent Details Panel /LiveInterac on /UseCustomTraceSource*

(2) Dedicated chapter in this manual: 10.2.6 / Usage examples: on *Level 2C*

(3) Level 24-26 help texts explaining usage in situ (on the level)

Custom LineTrace Source combined with *disabled LOD* results in containers that could work “behind the stages” / away from the volumes - while not visible.

No interac on

(2) Simula on is mapped on it

(it is visualizing simula on output)

There are cases when we don't need interac on at all: simula on input is a texture (sta c) or a material (animated), user interac on and overlap detec on is *completely*

In volumetric setups, simula on output is rendered using the volume - so TraceMesh *loses* its role as a display (like on Levels 24-25-26)... or, it could be combined with the volume (both TraceMesh and Volume are displaying sim output) - like on Level 23.

*switched off* (at NinjaLive Actor Details panel /LiveInterac on), simula on is running without any interac on with the scene ----> See *Level 24B, 24C*

25. Detail Maps

A feature to add dynamic, flowing details to low and medium resolu on simula ons at minimal cost. Introduced to NinjaLive 1.3

We are using sim velocity buffer as a flow map to advect a user defined map (typically procedural noise) - and mix the dynamically flowing details to the na ve simula on output

Included examples:

- Level 29 adding cell noise to flame or combus on type fluids

- Level 11 adding a cloud noise to a smoke-sim

- Level 10B, Stage1 adding grainy "sand like" noise to viscous fluids

Accessible at: Output Material Instance /FlowMap parameter group

Concept behind Detail Maps:

Ninja fluid-sim pipeline is a collec on of various data types like density, velocity, pressure, divergence. Density is generally handled like the "final product" while others considered as "by-products". How about u lizing sim "by-products" for something useful? Pressure is ideal for op cal and geometric distor on (see Level21 Stage6), velocity could be used to drive

par cles (see Level 20A,B) or: to drive texture advec on - tradi onally called "flow mapping". The flowmap concept: we are using two iden cal copies of a sta c texture and distort the image-pair using a shi ed period oscilla on. Distor on is driven by a velocity map. A velocity map is a field with direc onal vectors, ideal to tell par cles / texels "which direc on to go".

Ninja already u lizes flowmaps - this is how we make volumetric noise flow on clouds - see Levels 24-28. Now, an other flowmap feature has been implemented: detail mapping

26. Controlling Live in real me

To grasp the idea of *real me control,* load any ninja tutorial level, place a *Preset Manager*, start the Game *(Play)* and use Preset Manager to (A) interac vely set Fluidsim Parameters, (B) change Input Textures and (C) Output Materials.

See videos: changing Sim Params: [LINK](https://youtu.be/0O11SNavhM4?t=632) / changing Output Materials: LINK1, LINK2

Similar control could be achieved by Game Logic (Blueprints, Code) or Sequencer.

● *Prac cal example1:* using NinjaLive Actor for an area FX. We would like to control the FX

params using sequencer - to create a choreographed Cinema cs Sequence

See this video: LINK

● *Prac cal example2:* character ability FX is delivered by an embedded NinjaLiveComponent - we switch the fluidsim preset and the output material as the character uses various ability-FX

Control subjects in the NinjaLiveComponent blueprint could be grouped as:

1. Could be modified instantly

2. Could be modified by re-ini alizing NinjaLiveComponent blueprint

1. Simple Variables (eg. floats, integers) could be modified instantly by accessing NinjaLiveComponent. Variables have "telling names" - and you could read about them using the Tool ps and the Manual. Pls open up NinjaLiveComponent Blueprint: Preset Variables (for example) could be found under the "NONPUBLICLive Preset Variables" group. --- by code you can access them any me. In the Blueprint that embeds NinjaLiveComponent (eg. a character) params could be directly accessed. For other (external) blueprints, the LIVE INTERFACE could be used. To modify a variable by Sequencer, you need to set "expose to cinema cs" to TRUE. See Level 5, demonstra ng how to use the Interface and Sequencer.

2. Asset-type-variables (like a DataTable with preset-values, or an OutputMaterial Instance) could be modified by re-ini alizing NinjaLive. (Trivial example: the Preset Manager re-ini lizes ninja when changing materials or loading a new preset.)

---> See image below: Preset File and OutputMaterial index changing - combined with RePlay

Swapping Materials: only those output materials could be accessed (and used for swapping), that are previously added to the OutputMaterials array at LiveComponent /LiveGeneric Swapping process: (1) changing the OutputMaterialSelected INDEX, then (2) re-ini alize ninja

Please open NinjaLiveComponent Blueprint, and locate MODULE001. Check the *RePlay* node. As the label tells us: this event could be called remotely (eg. by an interface, or event dispatcher)

27. Mapping sim on 3D Mesh surfaces

See this video, demonstra ng the usage of 3D meshes: LINK The technique is UE5 compa ble.

NinjaLive is a 2D fluid sim. Besides the original (A) camera facing flat planes and (B) parallax mapping, new techniques are being added to enhance and spa alize sim output.

Live 1.2 could u lize (C) UE Volumetric Fog and (D) UE Volumetric Clouds (see Chapter 24). Live 1.4 introduces two more techniques:

1. raymarching based translucent volumetrics (en tled as "VolumeSmoke")

2. polygonal mesh based opaque 3D surfaces

This video demonstrates the two techniques: LINK + you could read about VolumeSmoke

in Chapter 24. This chapter describes the mesh based solu on.

Originally, Ninja used Parallax Occlusion Mapping (POM) to imitate depth. POM does not look good from a low (FPS) angle. Please have a look at LEVEL 10-B. The fluidsim containers on this Level are all using OPAQUE Output Material, which creates a discrete surface - the fluid surface is NOT at all fog like (it is non-translucent) - but opaque, shiny, reflec ve - like blood, mud, snow.

The opaque surface property enables us to use real 3D meshes to display sim output:

by defining a pre-tessellated (hi-poly) grid as TraceMesh, and using simula on density as a height-field to drive vertex World Posi on Offset (WPO), we could dynamically distort the TraceMesh: the result looks good from low (FPS) camera angles as well.

27.1 Reconfiguring a sim container from POM usage to 3D sim mesh usage A STEP-BY-STEP GUIDE

1. Select Container

2. Select TraceMesh Actor Component

3. Replace "NinjaLiveTraceMesh" (a single quad plane) with a tessellated plane

(eg. SM\_plane\_300x300)

4. Enable "Cast Shadow" at the mesh component Details Panel

5. Select NinjaLiveComponent

6. Go to LiveGeneric param group, Locate the currently used OutputMaterial in the array

(checking the OutputMaterialSelected INDEX below the array helps)

7. Locate "Parallax" param group, disable ParallaxMapping

(not needed when using a 3D sim mesh)

8. Locate "MeshDistort" param group, enable MeshDistor on, tweak params

(pay a en on to CLAMP!)

28. Moving in WorldSpace

Example content: /Tutorial/Levels/ Level 04

/Usecases/Levels/ Usecase\_001\_WorldSpaceWater

28.2 Addi onal se ngs to tweak:

Ninja sim containers could be used two ways:

1. Area FX: the sim is locked to a fixed WorldPos on - like a puddle, a bonfire, a portal ...

2. Character/Vehicle FX: the sim is a ached to a moving Actor - like a torch with a smoke

trail, turbulence around a moving ship, dust caused by footsteps ...

When the sim container is travelling in WorldSpace, we need to modify the local sim according to global movement direc on and speed.

28.1 Main methods:

A. Offset ALL simula on buffers inverse direc on

Feature introduced in *Live 1.5*

Enables the already emi ed fluid to "lag behind" the emi er/source

B. Modify sim VELOCITY buffer with the mo on vector

This feature could push fluid towards or against moving direc on

The key params are exposed at: Preset Manager UI /Fields sec on /... ... /All Fields Offset by Sim Area Mo on

... /Velocity Field Influenced by Sim Area Mo on

The combina on of A+B could result in interes ng FX. For example, forcing the sim to lag behind, but also se ng a nega ve velocity that pushes it along moving direc on results in a fluid that creeps / lingers behind the emi er :)

A. *Feedback* (Preset Manager GUI /Sim sec on)

Defines "Trail Life me" - the me needed to fade out the lingering fluid

B. *Drag* (PresetManager UI /Brush sec on)

S rring/dragging effect of moving objects on the simula on

*C. Puncture-modify*

NinjaLiveComponent Details /LiveBrushSe ngs /SimAreaMo onEffectsBrushPuncture The Z component of sim area mo on (perpendicular to TraceMesh) mul plies PUNCTURE --- in prac ce: when the sim container is moving towards/away from us, the "boiling" effect of puncture is more intense (by default) - and less intense when the param is set to ZERO

*D. Latency*

NinjaLiveComponent /LiveInterac on /LegacySimAreaMo onEffectsSimDensity

28.3 Warning:

FieldsOffset\* is a key param --- needs to be tweaked carefully

\*Preset Manager UI /Fields sec on /All Fields Offset by Sim Area Mo on

Different TraceMesh size / Different sim resolu ons need different FieldsOffset values, eg.: the larger the tracemesh, the lower the FieldsOffset. Turbulence values are also sensi ve: when working with HD sim, set Turb. < 1

29. Addi onal Content: UseCase Packages

With NinjaLive version 1.5.26.1, the first round of adding core features has ended. Star ng with *1 August 2021*, we are focusing on producing prac cal examples.

The examples are developed in small packages - and these packages are not automa cally merged to / added to ninia main branch. Plan:

1. we are developing the packages in union with the ninja-user community,

using the ninja discord as a communica on and exchange pla orm

2. once a specific use-case is done, it is…

- showcased on a dedicated [YouTube](https://www.youtube.com/playlist?list=PLVCUepYV6TvO6QhpqaT1GRVGC96QMcUJs) playlist - packed to a small zip file

- zip is distributed via ninja discord: on the #Announcements channel DISCORD INVITE LINK: [h](https://discord.com/invite/VpcyBQa77w) [ps://discord.com/invite/VpcyBQa77w](https://discord.com/invite/VpcyBQa77w)

3. the package is referencing core ninja assets

(eg. contains material instances - and base materials are in the core ninja project)

4. packages are not cross referencing each other

(so you will never “miss one” in order to open an other)

5. packages are numbered, and all located under /NinjaLive /Usecases

(to use a package, you only need to copy the zip content to /Content /NinjaLive)

When packages are becoming numerous, we are merging them in one step to the ninja main project - me by me (probably every month)

(so eventually all use case content becomes available via the marketplace project)

As an example, the *first two* UseCase packages has been included in the latest official NinjaLive version (1.5.23.2 / 1.5.26.2)

UseCase 001 is demonstra ng how a single, camera (player) a ached sim container with Sim.buffer WorldSpace Offset could be mapped on a large (infinite) water body to make the illusion that ninja is not a LOCAL, but a GLOBAL fluidsim. See preview *here.*

Package Assets: */FluidNinjaLive/UseCases/001\_WorldSpaceOffset\_FluidSurface/* Package demo level: */FluidNinjaLive/UseCases/Levels/Usecase\_001\_WorldSpaceWater.umap*

UseCase 002 uses similar technology as 001, demonstra ng heavy boat *trails*. The pack includes combo water materials that are both *translucent* and *reflec ve* + an example for UE na ve *SingleLayerWater* material. See preview *here*.

Package Assets: */FluidNinjaLive/UseCases/002\_BoatTrail/* Package demo level:

*/FluidNinjaLive/UseCases/Levels/Usecase\_002\_\*.umap (4 maps)*

UseCase 003 demonstrates how ninja could be used together with na ve UE 4.26 water bodies (sea, lakes, rivers). Package available ONLY at Ninja Community Discord / Announcements channel: h ps://discord.gg/VpcyBQa77w

UseCase 004 Tiling: running TWO ninja sim containers: (1) one to generate a led pa ern that could be used on infinite fields - and a second (2) that handles local interac on. Two sim outputs are merged by a Water BaseMaterial.

See preview: LINK

UseCase 005 Vehicle Trails: a WorldSpace ninja sim container generates area smoke/dust vfx for car tyres. Sim Container is a ached to the car (while rota on is locked) and hidden - output is used to feed a VolumetricSmoke Container with input data. Addi onally, four small containers are a ached to the wheels, to generate local smoke/flames. See preview: LINK

UseCase packages 4 & 5 have been included in a complete, new NinjaLive Release v1.5.26.3 - available via EPIC launcher and the Marketplace.

UseCase 006 Large Water Bodies: Rivers, Lakes, Whirlpools

This addon package demonstrates how mul ple ninja sim outputs are composited together by a single material - to create large water bodies. We are using a le generator (1) and a local solver (2) sim.

---> Youtube Demonstra ons: [SHORT](https://youtu.be/XfGXwYxKYIE), LONG

UseCase6 is released as a 10 Mbytes ZIP file - an ADDON for NinjaLive version 1.5.23.3 -- The ZIP file is available at the Ninja Community Discord *#Announcements* channel

UseCase 007 Small Water Bodies: Waterfalls, Pools, Fountains

This addon package demonstrates how local ninja simula ons could be turned to small water bodies by u lizing available sim buffers - pressure to distort geometry and calculate normals, velocity to drive detail maps (surface foam) and density to add large scale details. Buffers are integrated by a new BaseMaterial, capable to deliver reflec on, refrac on and translucency.

---> Youtube Demonstra on: LINK

UseCase7 is released as a 15 Mbytes ZIP file - an ADDON for NinjaLive version 1.5.23.3 -- The ZIP file is available at the Ninja Community Discord *#Announcements* channel

-------------------------------------------------------------------------------------------------------------

A new, 35 mins tut video is released, explaining all technical details behind the recent, WATER related use case packages 1-7: [LINK](https://youtu.be/l2qLnN3J3tI)