NIAGARA - Particle System

Contents

1 What is Niagara?

The Niagara system is the modern particle and VFX system built into Unreal Engine 5. Niagara gives us a modular, programmable way to create visual effects like fire, smoke, sparks, trails, and magical effects, all of them without writing code. It allows you to create effects by combining several building blocks — systems, emitters, and modules — to build complex simulations.

- A **Niagara System** is a container that holds one or more emitters together into a full effect.
- A **Emitter** is where particles are generated and controlled over time.
- **Modules** are small, reusable pieces of logic that define how particles behave.
- **Parameters** hold data used by modules (e.g., color, size, velocity).

Think of a system as the final special effect, emitters as the sources of particles, and modules as the rules that govern how particles behave.

1.1 Creating a Niagara System and Emitter

Normally, to start Niagara VFX:

1. Right-click in the Content Browser.
2. Choose "Niagara" → "Niagara System" or "Niagara Emitter".
3. Choose a template or a blank effect.

A Niagara System will contain at least one emitter that defines how particles are spawned and updated over time.

1.2 Niagara Editor Interface

Inside the Niagara editor, you'll see something like:

- **System Overview** – shows all emitters inside the Niagara system.
- **Emitter Stack** – modules arranged into groups that define particle behavior.
- **Timeline / Preview Viewport** – simulates and previews the particle effect.
- **Details Panel** – shows settings for selected emitters or modules.

1.3 Terminology Explained

Below are the key Niagara terms every student should understand:

1.3.1 Niagara System

A Niagara System is the top-level asset that contains one or more emitters. It represents the complete effect that you can place in your scene. For example, a fire effect might have smoke, sparks, and glow emitters inside the same system.

1.3.2 Emitter

An Emitter is responsible for creating particles. Each emitter handles:

- When new particles are spawned.
- How each particle is initialized.
- How particles update over time.
- How particles are rendered.

A system can have multiple emitters, for example one for tiny sparks and another for smoke plumes.

1.3.3 Particle

A particle is a single point of visual data in a simulation. When many particles are spawned together, they form an effect like smoke or sparks. Particles carry attributes like position, color, size, velocity, and life.

1.3.4 Modules

Modules are the building blocks that define behavior inside Niagara. They can:

- Set the number of particles spawned.
- Define initial particle attributes like position, color, or size.
- Apply forces or logic as particles age.
- Adjust rendering behavior.

Modules are grouped based on when they run:

Emitter Spawn Modules These run when the emitter is first created and define initial setup logic.

Emitter Update Modules These run every frame at the emitter level (affects all particles generally).

Particle Spawn Modules These apply to each particle right when it appears (initial conditions).

Particle Update Modules These run continuously on each particle and are often used to apply physics like velocity, color change, or lifetime updates.

Render Modules Render modules define how particles are drawn — whether as sprites, meshes, ribbons, or lights.

1.3.5 Parameters

Parameters in Niagara are named values that store data. They can be:

- **Primitive types** (float, int, vector).
- **Enums** (a fixed set of named values).
- **Structs** (grouped values).
- **Data Interfaces** (connect Niagara to external data like meshes).

Parameters can be global (affect the entire system), emitter-level, or particle-level.

1.4 Module Stack and Execution Order

Niagara processes modules like a stack:

- First the emitter spawn group runs.
- Then emitter update repeatedly each frame.
- When particles are created, the particle spawn group runs.
- Every frame for each particle, the update group runs.
- Finally, render modules draw the particles.

The order is essential because it determines when settings like velocity, size, or lifetime take effect.

1.5 Niagara Workflow Summary

1. Create a Niagara System.
2. Add one or more emitters.
3. In each emitter, configure spawn settings and modules.
4. Set initial particle behavior in the Spawn sections.
5. Add forces, changes, and logic in Update modules.
6. Choose a renderer (sprite, mesh, etc.).
7. Test and tweak the effect in the viewport.

1.6 Why Niagara Matters

Niagara gives artists and designers powerful control over visual effects without needing to write C++ or shader code. Effects are built from reusable modules and data-driven parameters, and the system scales from simple spark effects to complex fluid simulations.

2 Creating a Fire Particle from Scratch using Niagara

In this section, we will create a basic fire particle effect entirely from scratch using Niagara in Unreal Engine. We will not use templates. Instead, we will understand each step and build the fire logically.

2.1 Step 1: Create a New Niagara System

1. Right-click inside the Content Browser.
2. Select `FX → Niagara System`.
3. Choose **Create Empty System**.
4. Name it `NS_Fire`.
5. Double-click to open the Niagara Editor.

When the system opens, it will be empty. We must now add an emitter.

2.2 Step 2: Add a New Emitter

1. Click the **Add Emitter** button.
2. Choose **Create Empty Emitter**.
3. Name it `NE_FireEmitter`.

The emitter contains several sections:

- Emitter Spawn
- Emitter Update
- Particle Spawn
- Particle Update
- Render

These sections control when and how particles behave.

2.3 Step 3: Configure the Emitter Spawn

In the Emitter Spawn section:

- Add a **Spawn Rate** module.
- Set Spawn Rate to around 50 - 100.

This means 50 – 100 particles will be generated every second.

2.4 Step 4: Set Particle Lifetime

Go to **Particle Spawn**:

- Add **Initialize Particle**.
- Set Lifetime to approximately 1.0 – 2.0 seconds.

Fire particles should live briefly before fading.

2.5 Step 5: Set Initial Size

Still inside **Initialize Particle**:

- Set Sprite Size to something like `(20, 40)`.

Fire usually stretches vertically, so Y value should be slightly larger.

2.6 Step 6: Add Upward Motion (Velocity)

Inside **Particle Spawn**:

- Add **Add Velocity**.
- Set Z velocity to a value like `(`100–200`)`.

This makes the fire rise upward.

2.7 Step 7: Add Randomness

Fire should not move uniformly.

- Add a **Random Vector** to velocity.
- Add small X and Y variation (e.g., -20 to 20).

This gives the flame a natural flicker motion.

2.8 Step 8: Change Color Over Life

Go to **Particle Update**:

- Add **Color Over Life**.

Set gradient:

- Start: Bright Yellow / White
- Middle: Orange
- End: Dark Red / Transparent

Fire typically transitions from bright to darker tones as it dies.

2.9 Step 9: Add Size Over Life

Still in Particle Update:

- Add **Scale Sprite Size**.

Make the particle slightly grow and then shrink. Example:

- Start small
- Grow slightly
- Fade out

2.10 Step 10: Add Drag (Optional but Recommended)

To slow particles naturally:

- Add **Drag**.
- Set Drag value around 1–3.

This prevents fire from moving too fast upward.

2.11 Step 11: Add a Renderer

Go to the **Render** section:

- Click **Add Renderer**.
- Choose **Sprite Renderer**.

The sprite renderer displays each particle as a 2D image facing the camera.
You should now see a basic flame simulation in the preview window.

2.12 Optional Improvements

To improve realism:

- Increase spawn rate slightly.
- Add a second emitter for smoke.
- Add subtle noise force for flickering motion.
- Adjust lifetime and color curve carefully.

2.13 Understanding What We Built

Let us summarize what is happening:

- The emitter spawns particles every second.
- Each particle lives for a short time.
- Particles start bright and fade darker.
- Velocity pushes them upward.
- Randomness makes them flicker.
- Drag slows them down.
- The sprite renderer makes them visible.

This combination produces a believable fire effect.

2.14 Testing the Fire Effect

1. Save the Niagara System.
2. Drag `NS_Fire` into your level.
3. Press Play to preview it in the environment.

If the fire looks too strong:

- Reduce spawn rate.
- Reduce lifetime.
- Reduce velocity.

If it looks too weak:

- Increase spawn rate.
- Increase sprite size.
- Increase brightness.

PS: I'm sorry for not being able to add photos for this lecture.