



집중 탐구 가이드: Unity에서 고급 시각 효과 만들기



Unity에서 제작된 씬, Sakura Rabbit 제작

목차

들어가는 말.....	7
도움을 주신 분들.....	8
주요 저자.....	8
도움을 주신 유니티 직원 분들	8
도움을 주신 외부 인사 분들	8
실시간 VFX 시작하기.....	9
시각 워크플로	11
코어 그래픽스 패키지	12
추가 샘플 콘텐츠.....	12
빌트인 파티클 시스템	13
VFX Graph 소개	16
VFX Graph Asset 및 컴포넌트.....	17
VFX Graph 창.....	19
그래프 로직.....	21
시스템, 컨텍스트, 블록	22
프로퍼티 및 연산자.....	27
Blackboard	30
그룹 노드 및 스티커 메모	32
하위 그래프.....	33
속성.....	35
Blackboard 속성.....	39
이벤트	42
이벤트 속성.....	44
Unity 6의 UI 개선 사항	45
노드 검색.....	45

활성화 포트	46
새로운 VFX 툴바	47
키보드 단축키	47
VFX 샘플 콘텐츠 살펴보기	48
더 많은 자료	55
추가 참조 자료	55
예시로 보는 시각 효과	56
VFX Graph Learning Templates	56
기즈모 시각화	57
그래프 기초	58
Contexts & Data Flow	58
Spawn Context	59
Capacity Count	59
Multiple Outputs	60
Bounds	61
파티클 방향 및 회전	62
텍스처링과 플립북	63
파티클 피벗	65
메시 및 텍스처 샘플링	66
충돌 및 상호 작용	69
데칼 파티클	72
파티클 스트립	73
VFX Graph 샘플(HDRP)	79
연기 포털	80
실시간 연기를 위한 6방향 조명	84
구볼(GooBall)	87
물리 기반 효과	92

Ribbon Pack	95
Meteorite 샘플	98
메시 샘플링 효과	102
스킨드 메시 샘플링	105
더 많은 예시	107
렌더 파이프라인 및 VFX Graph	108
URP용 VFX Graph	109
릿 출력	109
2D 렌더러 및 Shader Graph 지원	110
데칼	111
6방향 연기 조명	111
모션 벡터	111
카메라 버퍼	111
HDRP용 VFX Graph	112
레이트레이싱 지원	112
볼류메트릭 안개 출력	113
Shader Graph 연동	114
빌트인 출력	114
셰이더 효과	115
Shader Graph와 VFX Graph	122
Shader Graph 기능 예시	123

상호 작용	124
이벤트 바인더	124
OnPlay 및 OnStop 이벤트	125
Mouse Event Binder	126
Rigidbody Collision Event Binders	127
Trigger Event Binder	128
Visibility Event Binder	129
타임라인	130
이벤트 속성	132
프로퍼티 바인더	133
출력 이벤트	136
파이프라인 툴	139
포인트 캐시	139
포인트 캐시 베이킹 툴	140
포인트 캐시 사용	141
Signed Distance Fields	142
SDF 사용	142
SDF 베이킹 툴	143
벡터 필드	144
TFlow(에셋 스토어)	145
VFXToolbox	145
Image Sequencer	145
디지털 콘텐츠 제작 툴	146
SideFX Houdini	146
Autodesk Maya	146
Blender	147
Adobe Photoshop	147

최적화.....	148
VFX Graph의 Profiling 및 Debug 패널	149
VFX Graph의 Profiling 및 Debug 패널	149
Particle System Info 패널	150
Context Debug 패널	150
Unity 프로파일러 및 프레임 디버거	151
VFX Control의 디버그 모드.....	153
Bounds	155
메시 LOD.....	156
Mesh Count.....	158
고급 기능	160
그래픽스 버퍼 지원.....	160
Custom HLSL	162
추가 리소스.....	164
동영상 튜토리얼	165
GitHub의 VFX 프로젝트.....	166

들어가는 말

게임에서 VFX(시각 효과)를 사용하면 손끝에서 화염구가 나가거나 원홀을 통과하는 등 현실에서는 불가능한 현상을 실제로 구현할 수 있습니다. 시각 효과는 주변 환경의 완성도를 높이고 게임의 스토리를 더 효과적으로 전달할 수 있을 뿐만 아니라, 플레이어의 마음을 사로잡는 디테일로 상상의 세계에 생동감을 불어넣습니다.

Unity는 VFX Graph 같은 툴로 실시간 그래픽스의 한계를 넓히고 있습니다. 테크니컬 아티스트와 VFX 아티스트는 노드 기반 에디터인 VFX Graph를 사용하여 단순하고 평범한 파티클 동작부터 파티클, 선, 리본, 트레일, 메시 등이 포함된 복잡한 시뮬레이션까지 역동적인 시각 효과를 디자인할 수 있습니다.

이 종합 가이드는 VFX Graph를 게임 애플리케이션에 통합하려는 아티스트와 크리에이터를 위한 가이드로, VFX Graph와 관련 툴을 사용하여 Unity에서 실시간 시각 효과를 제작하는 방법에 대한 구체적인 지침을 제공합니다.

본 가이드는 1인 개발자와 대규모 팀 소속 개발자들의 경험을 모두 고려하여, AAA급 품질의 Unity 툴을 더 쉽게 사용할 수 있는 다양한 예시를 제공합니다. 경력에 관계없이 게임 디자인의 재미를 느껴 보세요.

중요 참고 사항: 이번 개정판에는 Unity 6 이상에서 사용할 수 있는 새로운 기능과 편의성 개선이 포함되어 있습니다. 이 가이드를 따라 하려면 Unity Hub에서 Unity 6를 설치해야 합니다. 또한 Unity 6 및 향후 릴리스부터 Unity 릴리스의 명명 규칙이 변경되었습니다. [새 명명 규칙을 이 Discussions 게시물](#)에서 자세히 알아보세요.



VFX Graph 샘플의 Smoke Portal

도움을 주신 분들

주요 저자

윌머 린은 영화 및 TV 업계에서 15년 이상 경력을 쌓았으며, 지금은 독립 게임 개발자 겸 교육자로 활동 중인 3D 및 시각 효과 아티스트입니다. 윌머는 엑스맨: 데이즈 오브 퓨처 패스트, 인크레더블 헐크, 나니아 연대기: 사자, 마녀, 그리고 옷장 등의 장편 영화 제작에 참여했습니다.

도움을 주신 유니티 직원 분들

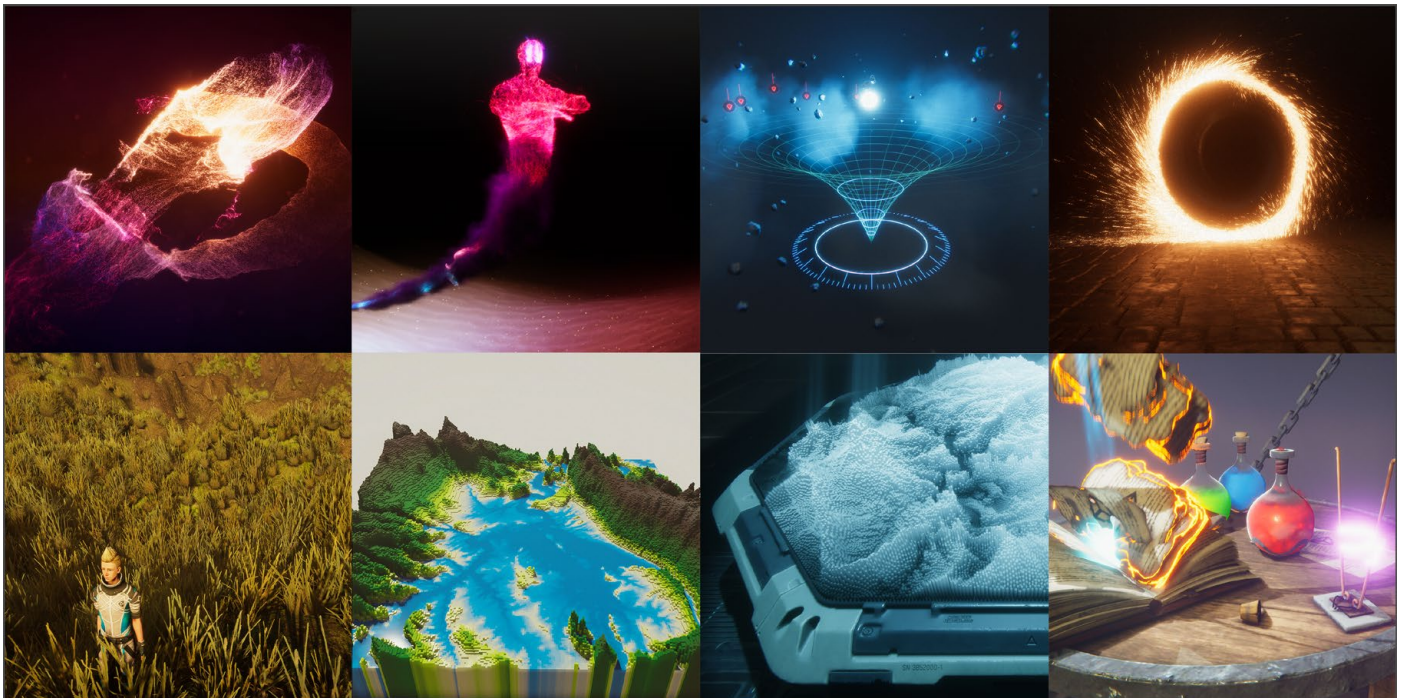
마티외 뮐러, 그래픽스 부문 리드 프로덕트 매니저
블라드 네이코프, 소프트웨어 엔지니어링 및 품질 부문 디렉터
울슨 파브렐, 테크니컬 아티스트(이 전자책에 사용된 여러 신규 샘플 제작)
줄리앙 프라이어, 그래픽스 엔지니어
프레드 모로, 테크니컬 프로덕트 매니저

도움을 주신 외부 인사 분들

마리 구프루아, 테크니컬 아티스트
토마스 이쎬, VFX 아티스트 겸 스페셜리스트

실시간 VFX 시작하기

최근 게이머들은 매우 몰입도 높은 경험을 원합니다. 하드웨어의 발전으로 모바일과 콘솔 플랫폼의 한계가 확장되면서, 전에는 할리우드 블록버스터 제작에만 사용되던 시각 효과를 이제 실시간으로 구현할 수 있습니다.



VFX Graph 예시

고급 그래픽스에 대한 관심과 투자가 늘어나면서 게임에 사용되는 시각 효과도 발전을 거듭하고 있습니다. 게임플레이 효과는 플레이어가 액션에 몰입하도록 만듭니다.

마법을 시전하는 캐릭터가 없는 판타지 RPG(롤플레이잉 게임)나 번쩍이는 검기를 볼 수 없는 핵 앤 슬래시 (hack-and-slash) 게임은 상상하기 어렵죠. 경주용 자동차가 아스팔트 도로를 질주하면 누구나 땅에서 피어나는 먼지구름을 예상합니다.

환경조차도 시각 효과 없이는 지금처럼 구현할 수 없습니다. 필름 느와르 스타일의 탐정 스토리를 제작한다면 비와 안개에 휩싸인 도시의 풍경을 디자인하고 싶을 것입니다. 하지만 캐릭터가 황야로 퀘스트를 수행하러 간다면, 나뭇잎과 초목이 바람에 흔들리고 캐릭터의 모든 움직임에 반응하도록 구현할 것입니다.



Stunlock Studios의 [V Rising](#)은 고향상도 렌더 파이프라인과 VFX Graph로 제작된 게임입니다.

시각 효과는 게이밍 경험의 품질을 높이는 특별한 방법입니다. 하지만 시각 효과를 제작하려면, 모양, 컬러, 타이밍을 조작할 수 있는 종합 아티스트로서의 면모를 갖춰야 합니다.

VFX Graph는 바로 이러한 상황에서 진가를 발휘합니다. 이 정교한 툴은 영화 VFX의 워크플로가 반영된 워크플로를 사용하지만, 영화와 달리 초당 30프레임, 60프레임 또는 그 이상으로 작업할 수 있습니다.



Sakura Rabbit이 Unity VFX 툴을 사용해 개발 중인 프로젝트의 이미지

시각 워크플로

하이엔드 플랫폼에서 복잡한 AAA급 시각 효과를 구현하려면 VFX Graph를 사용하여 직관적인 노드 기반 인터페이스에서 GPU 가속 파티클을 생성하면 됩니다.

다음과 같은 작업에 [VFX Graph](#)를 활용할 수 있습니다.

- 하나 이상의 파티클 시스템 제작
- 정적 메시 추가 및 셰이더 프로퍼티 제어
- 효과의 일부를 켜고 끌 수 있도록 C# 또는 [타임라인](#)을 통해 이벤트 생성
- 일반적으로 사용되는 노드 조합의 하위 그래프를 생성하여 기능 라이브러리 확장
- 다양한 속도로 변경 사항 미리 보기 또는 단계별 시뮬레이션 수행

VFX Graph는 URP([유니버설 렌더 파이프라인](#))* 및 HDRP([고해상도 렌더 파이프라인](#))와 함께 사용할 수 있습니다. 또한 URP에서 사용할 수 있는 [릿\(Lit\)](#) 출력 및 [2D 렌더러](#)도 지원합니다. 모든 렌더 파이프라인에 VFX Graph 기능을 비교한 결과를 [여기](#)에서 확인하고, [기술 자료](#)에서 VFX Graph의 호환성에 대해 자세히 알아보세요.

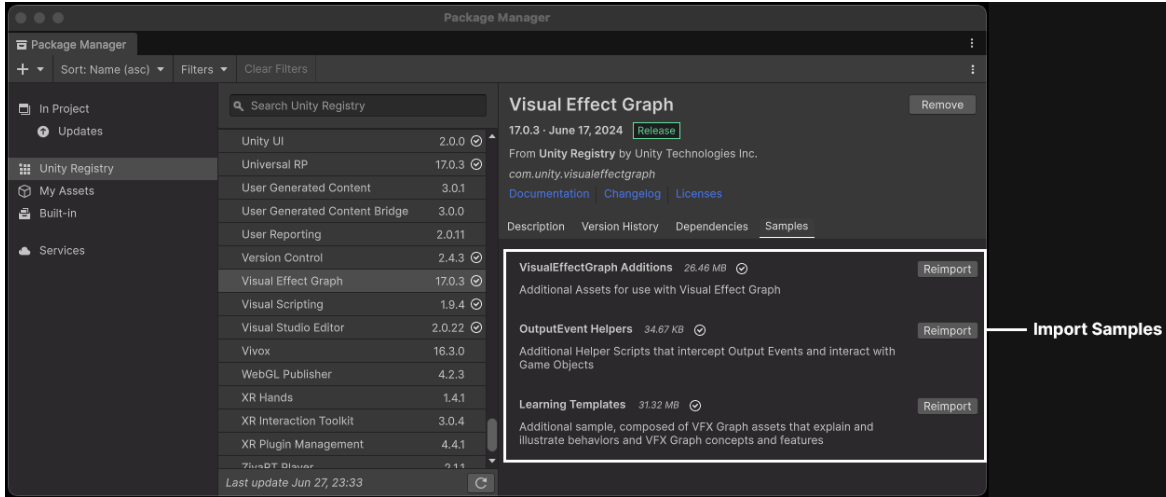
VFX Graph는 디바이스와 호환성을 유지할 수 있도록 컴퓨터 셰이더 지원이 필요합니다. 지원되는 디바이스는 다음과 같습니다.

- [Metal 그래픽스 API](#)를 사용하는 macOS 및 iOS 플랫폼
- [Vulkan](#) 또는 [GL ES3](#) API를 사용하는 Linux 및 Windows 플랫폼
- 일부 하이엔드 컴퓨터 지원 Android 디바이스(URP 사용 시에만)

코어 그래픽스 패키지

최신 Unity 릴리스를 설치하면 URP, HDRP, Shader Graph, VFX Graph 등을 위한 최신 패키지가 함께 설치됩니다. 이제 코어 그래픽스 패키지가 기본 Unity 설치 프로그램에 포함되므로, 항상 검증된 최신 그래픽스 코드로 프로젝트를 실행할 수 있습니다.

자세한 내용은 [VFX Graph 시작하기](#)를 참조하세요.



패키지 관리자에서 샘플 콘텐츠 설치

추가 샘플 콘텐츠

VFX Graph를 처음 사용하는 경우 **패키지 관리자**에서 **샘플**을 설치해 보세요. 다음과 같이 살펴볼 수 있는 몇 가지 기본 예제가 포함되어 있습니다.

- 새로운 **Learning Templates**: [Learning Templates](#) 샘플은 VFX Graph 관련 개념과 기능을 학습할 수 있게 제작한 VFX 에셋 모음입니다. URP 및 HDRP 렌더 파이프라인과 모두 호환됩니다.



새로운 Learning Templates 샘플에서는 다양한 VFX Graph 기능을 소개합니다.

- **Visual Effect Graph Additions:** 불, 연기, 불꽃, 전기의 예제 프리팹이 포함되어 있습니다. 각 샘플은 기본적인 그래프 로직과 구성을 설명하는 가장 기본적인 효과를 보여 줍니다. 샘플 프리팹 중 하나를 계층 구조(Hierarchy)에 드래그 앤 드롭하면 실제로 작동하는 모습을 확인할 수 있습니다.



Visual Effect Graph Additions 패키지의 샘플

빌트인 파티클 시스템

타겟 플랫폼이 VFX Graph의 **최소 시스템 요구 사항**을 충족하지 않는 경우, **빌트인 파티클 시스템**으로 Unity에서 실시간 효과를 제작할 수 있습니다. 빌트인 파티클 시스템을 사용하려면 **GameObject > Effects > Particle System**을 선택합니다.

VFX Graph와 마찬가지로 빌트인 파티클 시스템을 사용하면 불, 폭발, 연기, 마법 주문 등 **다양한 효과**를 만들 수 있습니다. VFX Graph보다 적은 수의 파티클을 렌더링하지만, 실시간 효과를 제작할 수 있는 유용한 툴입니다.

VFX Graph와 빌트인 파티클 시스템의 주요 차이점은 하드웨어에 있습니다. 빌트인 파티클 시스템은 CPU에서 시뮬레이션되지만, VFX Graph는 GPU에서 실행되는 컴퓨트 셰이더를 통해 대부분의 계산을 수행합니다.

VFX Graph는 수백만 개의 파티클을 시뮬레이션할 수 있다는 장점이 있지만, GPU에서 시뮬레이션되므로 CPU로 데이터를 다시 읽고 CPU에 있는 다른 시스템과 상호 작용하는 것이 계산 측면에서 쉽지 않습니다.

모바일 플랫폼을 사용하는 경우, VFX Graph를 사용하려면 해당 플랫폼이 컴퓨트 셰이더를 지원하는지 확인해야 합니다. 지원되지 않는 경우, 빌트인 파티클 시스템으로 CPU 기반 효과를 사용해야 할 수 있습니다.

두 시스템을 일대일로 비교하면 다음과 같습니다.

	CPU Particle System	GPU VFX Graph
Particle count	Thousands	Millions
Simulation	Simple	Complex
Physics	Underlying physics system	Scene representation • Primitives (spheres, boxes, etc.) • Signed Distance Fields
Gameplay readback	Yes	No*
Other	-	Depth buffer Can read frame buffers

빌트인 파티클 시스템과 VFX Graph 비교

빌트인 파티클 시스템은 기본 물리 시스템을 사용하고 게임플레이와 더 직접적으로 상호 작용할 수 있지만, 파티클의 수가 제한되고, 시뮬레이션이 상대적으로 단순해야만 합니다.

Unity 에셋 스토어에서 [Particle Pack](#)을 다운로드해 빌트인 파티클 시스템을 사용한 예제 세트를 확인해 보세요. 이 샘플 에셋에서 다양한 게임 내 효과(불, 폭발, 얼음, 용해 등)를 만나 볼 수 있습니다. 이 Dev Takeover 시리즈에서 Shader Graph와 빌트인 파티클 시스템 사용 방식에 대해 자세히 알아볼 수도 있습니다.

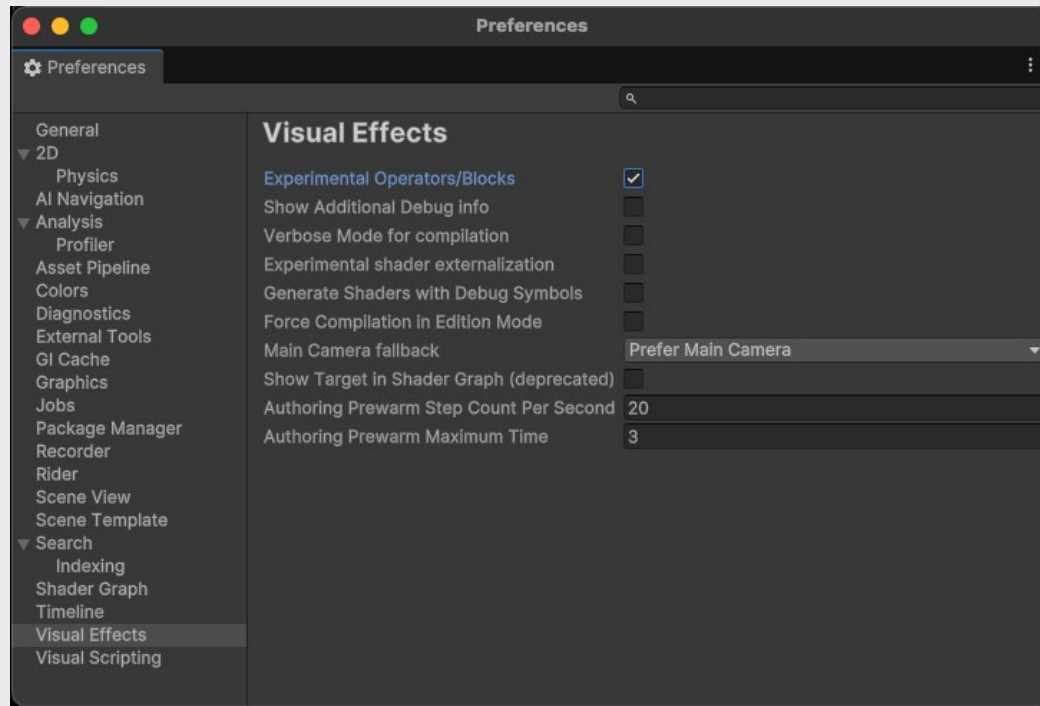
URP는 빌트인 파티클 시스템을 위한 스탠다드 셰이더(릿, 언릿, 심플 릿)를 제공하는 반면, HDRP는 [HDRP 패키지 샘플](#)에서 Shader Graph 기반 셰이더 샘플을 제공합니다. 렌더 파이프라인별 파티클 시스템 기능 비교는 [여기](#)에서 확인할 수 있습니다.



Unity 에셋 스토어에서 다운로드할 수 있는 Particle Pack

참고: 실험 기능은 완전히 검증되지 않은 기능이므로 변경될 수 있습니다. 정식 제작에는 Unity 정식 릴리스 버전을 사용하는 것이 좋습니다. 이 [Discussions 게시물](#)에서 Unity 6의 출시에 따라 명명 규칙이 어떻게 변경되는지 확인하세요.

가이드에서 소개하는 실험 기능은 **Preferences > Visual Effects > Experimental Operators/Blocks**에서 다음과 같이 활성화할 수 있습니다.

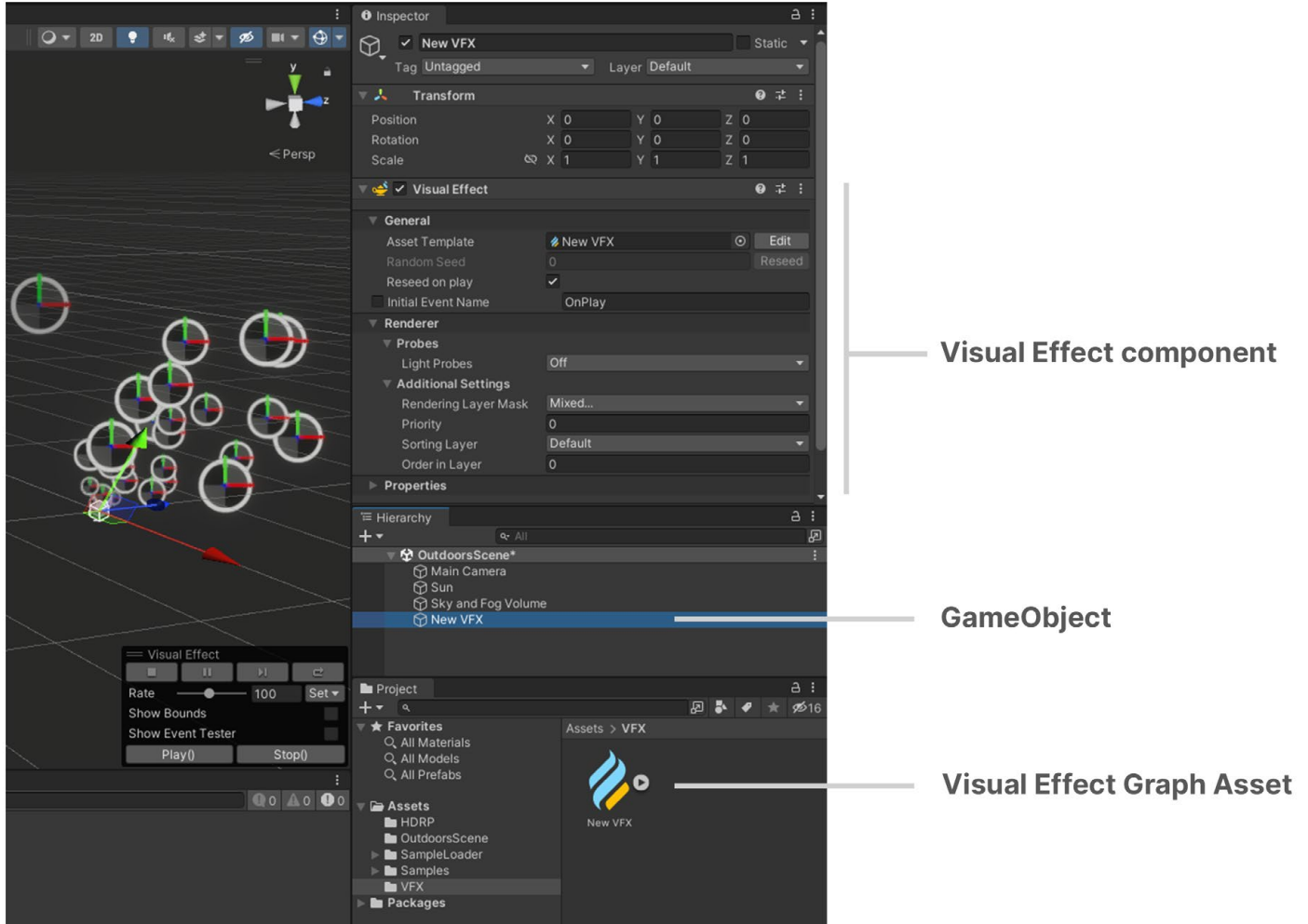


VFX Graph 소개

VFX Graph의 모든 시각 효과는 다음 두 부분으로 구성됩니다.

- 씬의 게임 오브젝트에 연결된 [VFX \(VisualEffect\) 컴포넌트](#)
- 프로젝트 수준에 존재하는 [VFX \(VisualEffect\) Graph 에셋](#)

Unity는 각 VFX Graph를 Assets 폴더에 저장하므로 각 에셋을 씬의 Visual Effect 컴포넌트에 연결해야 합니다. 여러 게임 오브젝트가 런타임에서 같은 그래프 하나를 참조할 수 있다는 점을 기억하세요.



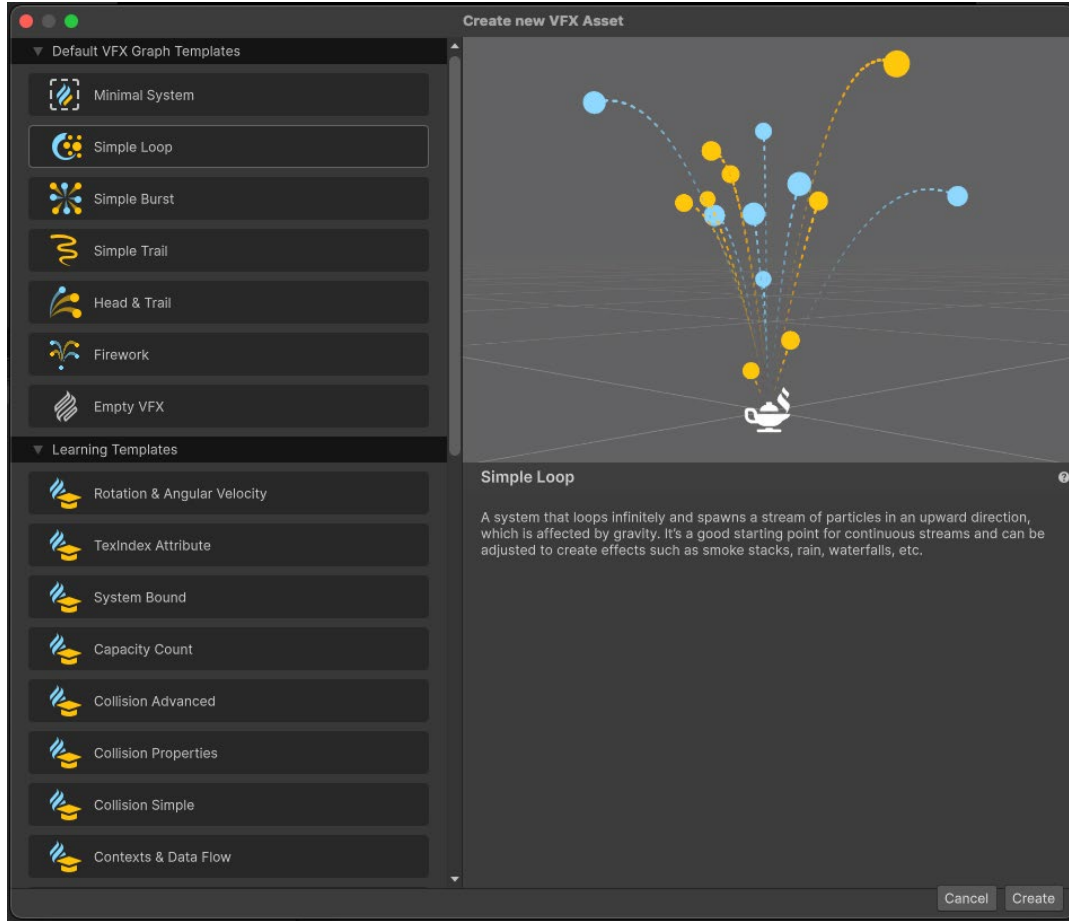
VFX Graph Asset과 Visual Effect 컴포넌트

VFX Graph Asset 및 컴포넌트

새로운 시각 효과를 생성하려면 **프로젝트(Project)** 창을 오른쪽 클릭하고 **Create > Visual Effects > Visual Effects Graph**로 이동합니다.

그러면 시작 템플릿을 선택할 수 있는 생성 마법사가 열립니다. 기본 VFX Graph 중 하나로 시작하거나 Samples에서 Learning Templates 중 하나를 선택할 수 있습니다.

패키지 관리자에서 추가 패키지를 설치하지 않으면 Learning Templates가 창에 표시되지 않습니다.



템플릿을 선택하여 새 VFX Graph를 생성합니다.

Create를 선택하여 프로젝트에 **VFX Graph Asset**을 생성합니다.

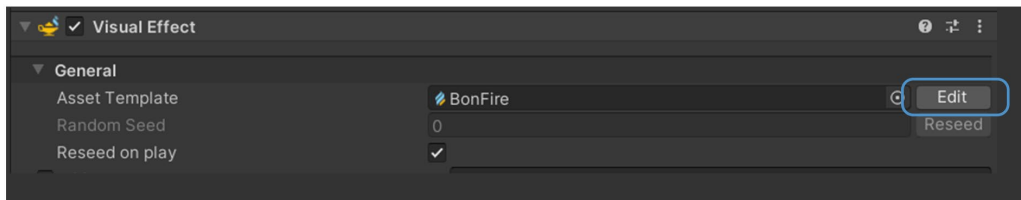
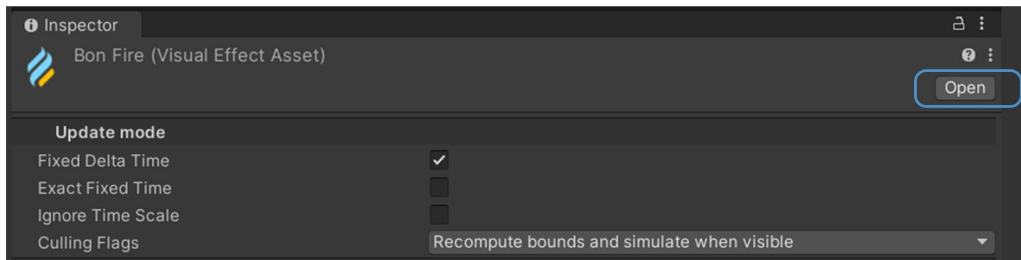
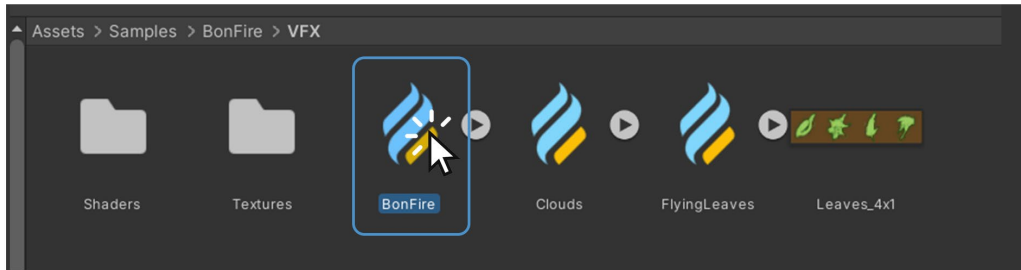
씬에 효과를 추가하려면 **Visual Effect 컴포넌트**를 게임 오브젝트에 연결한 다음 VFX Graph Asset을 연결합니다. 이를 수행하는 방법은 몇 가지가 있습니다.

- 생성된 에셋을 씬(Scene) 뷰나 계층 구조에 드래그합니다. 계층 창에 새 기본 게임 오브젝트가 생성됩니다.
- 인스펙터(Inspector)에서 기존 **Visual Effect 컴포넌트**에 에셋을 할당합니다. 계층 구조를 오른쪽 클릭하여 빈 게임 오브젝트를 생성(**Visual Effects > Visual Effect**)하거나, 게임 오브젝트를 생성한 다음 Visual Effect 컴포넌트를 수동으로 추가할 수 있습니다.
- 게임 오브젝트를 선택한 상태에서 에셋을 인스펙터 창으로 드래그 앤 드롭합니다. 그러면 Visual Effect 컴포넌트가 생성되고 에셋이 신속하게 할당됩니다.

VFX Graph Asset은 모든 로직을 포함합니다. 다음 중 한 가지 방법으로 에셋의 동작을 편집할 수 있습니다.

- 프로젝트(Project) 창에서 VFX Graph Asset을 더블 클릭합니다.
- 프로젝트 창에서 VFX Graph Asset을 선택하고 헤더의 **Open** 버튼을 클릭합니다.

- Visual Effect 컴포넌트에서 **Asset Template** 프로퍼티 옆에 있는 **Edit** 버튼을 클릭합니다.



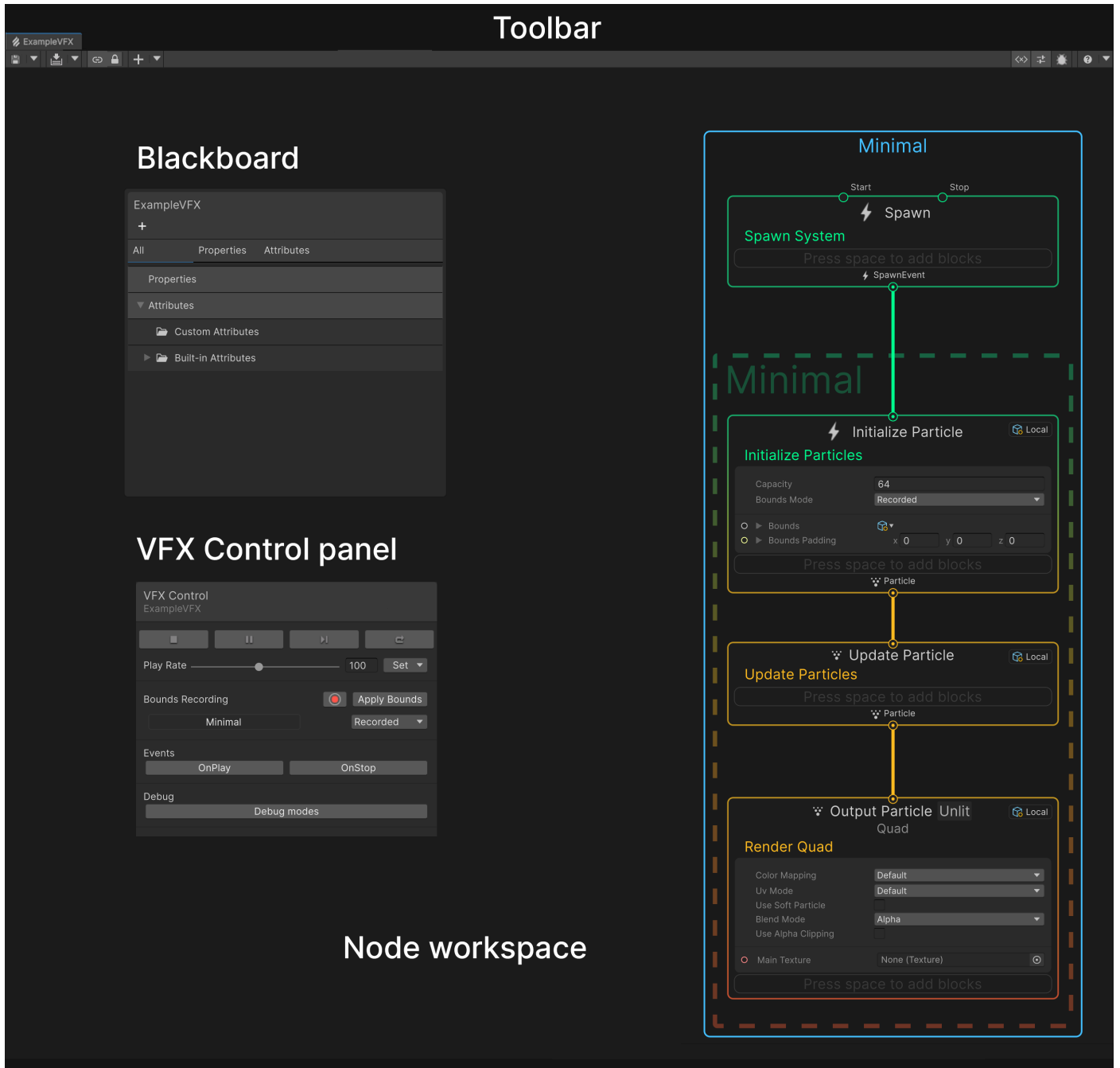
VFX Graph를 여는 세 가지 방법

예셋은 [VFX Graph 창](#)에서 열리며, 이 창은 **Window > Visual Effects > Visual Effect Graph**에 있습니다.

VFX Graph 창

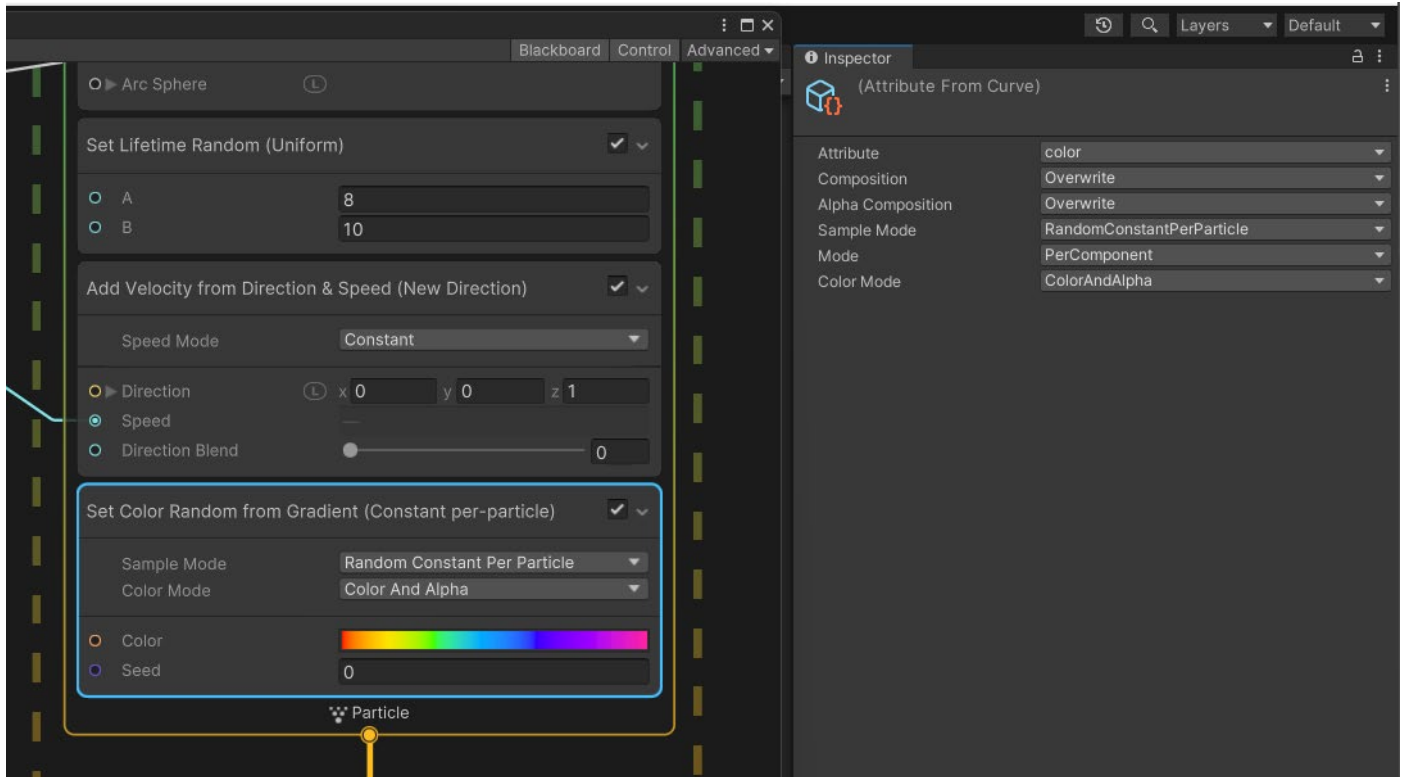
다음과 같이 [VFX Graph 창의 레이아웃](#)을 알아 두면 좋습니다.

- **툴바**: 전역 설정 액세스 및 여러 패널에 대한 토글 제공
- **노드 작업 공간**: VFX Graph 구성 및 편집
- **Blackboard**: 그래프 전체에서 재사용할 수 있는 속성 및 프로퍼티 관리
- **VFX Control 패널**: 연결된 게임 오브젝트에서 재생 내용 수정



VFX Graph 창

에디터 레이아웃에 인스펙터를 위한 공간을 남겨 두어야 합니다. 그래프의 일부를 선택하면 파티션 옵션, 렌더 상태 같은 특정 파라미터가 나타날 수 있기 때문입니다.

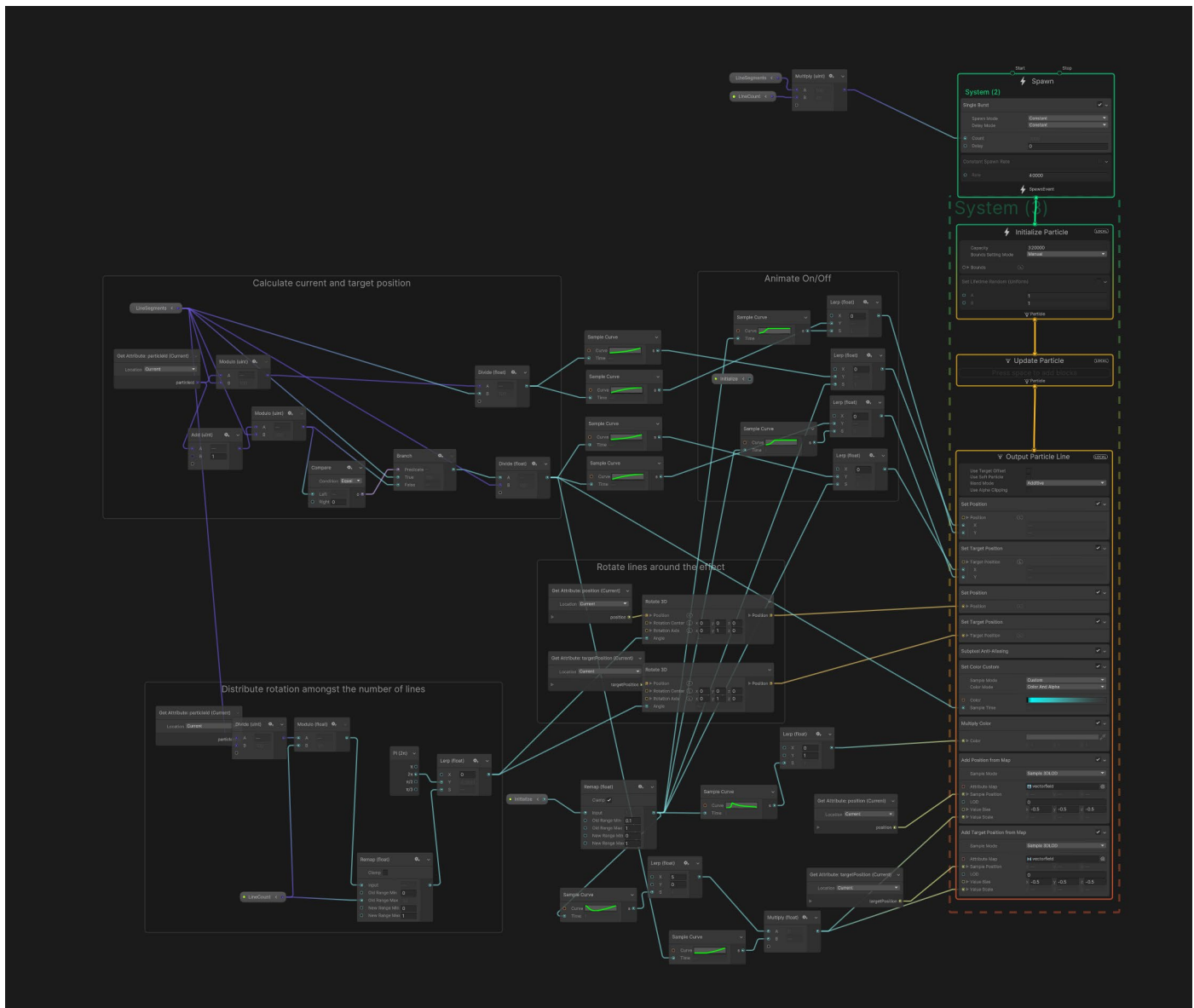


인스펙터를 사용하여 특정 파라미터를 변경할 수 있습니다.

그래프 로직

시각 효과는 창의 작업 공간에 포함된 노드를 연결해 만들어야 합니다. VFX Graph는 **Shader Graph** 등의 다른 **노드 기반 툴**과 유사한 인터페이스를 사용합니다.

스페이스바를 누르거나 오른쪽 클릭으로 새 그래프 요소를 만들 수 있습니다. 빈 작업 공간 위에 마우스를 놓고 **Create Node**를 선택하여 그래프의 **컨텍스트**, **연산자** 또는 **프로퍼티**를 만들 수 있습니다. 기존 컨텍스트 위에 마우스 커서를 올리면 **Create Block**을 사용할 수 있습니다.



복잡한 연결망으로 구성된 VFX Graph

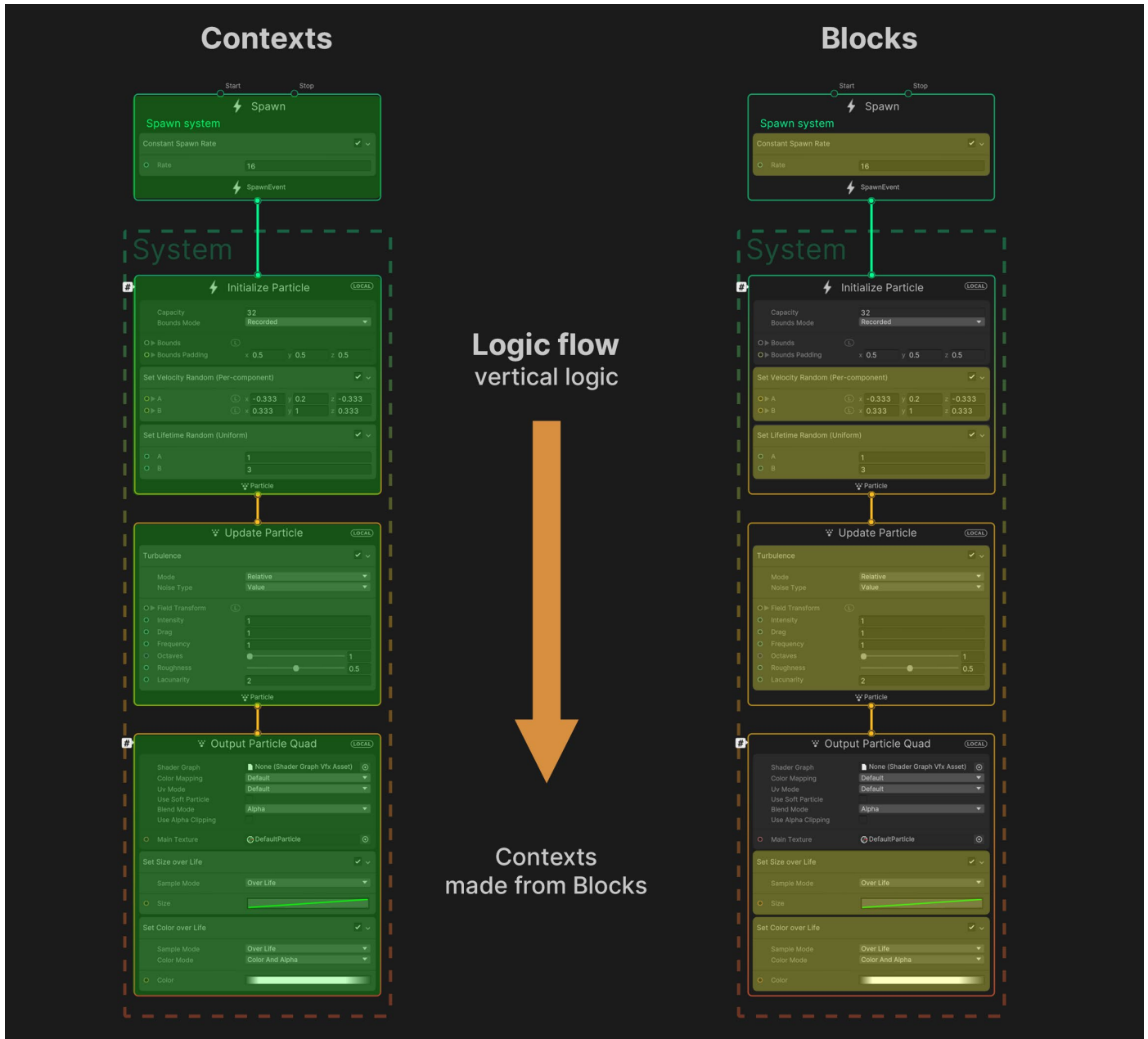
복잡한 VFX Graph를 열면 처음에는 부담스럽게 보일 수 있습니다. 정식 제작 수준의 그래프에는 수백 개의 노드가 포함되기도 하지만, 다행히도 크기와 관계없이 모든 그래프는 동일한 규칙을 따릅니다.

VFX Graph의 각 부분을 살펴보고 이들이 어떻게 함께 작동하는지 알아보겠습니다.

시스템, 컨텍스트, 블록

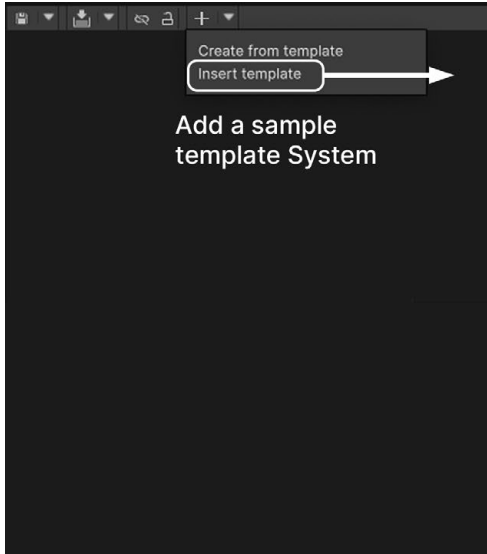
VFX Graph에는 **시스템**이라는 세로형 스택이 하나 이상 들어갑니다. 시스템은 그래프의 독립된 부분을 정의하고 여러 **컨텍스트**를 포함합니다. 시스템은 해당 시스템을 구성하는 컨텍스트를 묶는 점선으로 표시되어 있습니다.

각 컨텍스트는 개별 **블록**으로 구성되며, 블록에서 파티클과 메시의 **속성**(크기, 컬러, 속도 등)을 설정할 수 있습니다. 하나의 그래프 내에서 여러 시스템이 연동되며 최종적인 시각 효과를 만들어 냅니다.

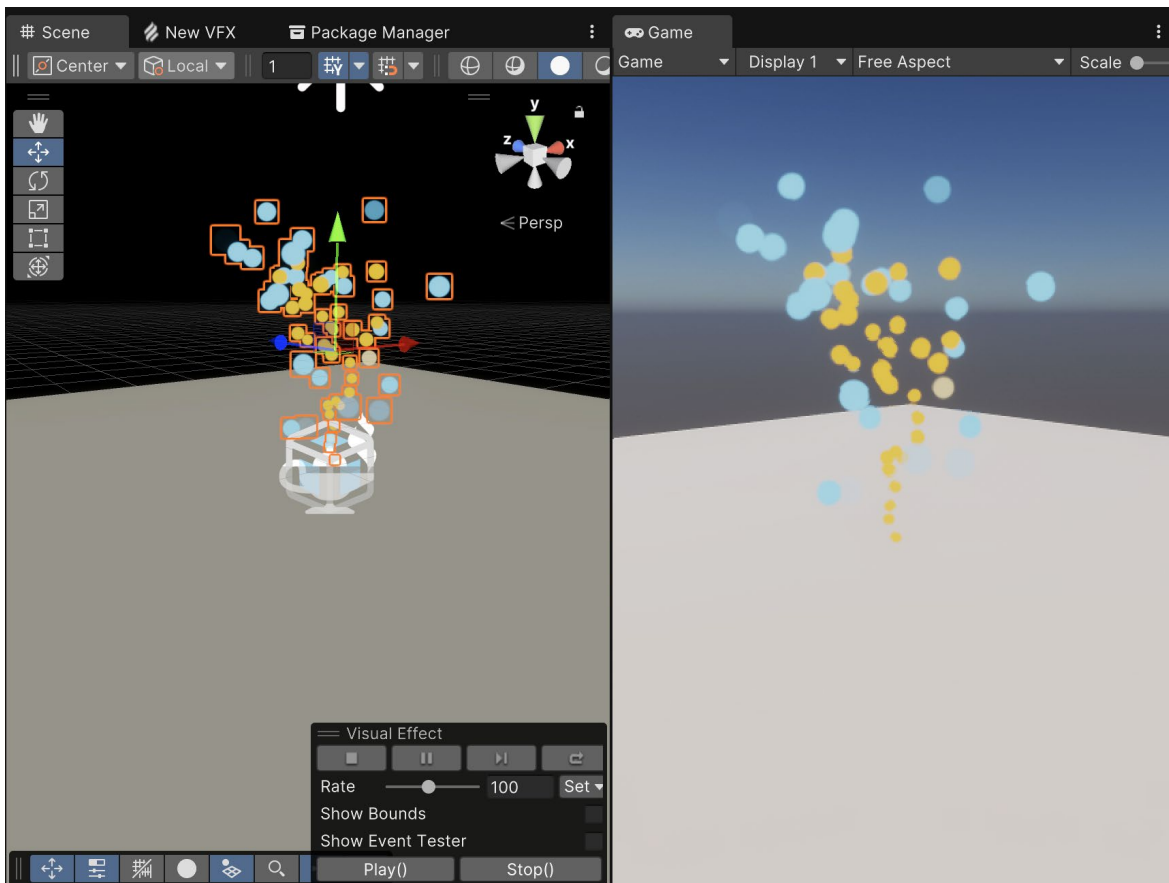
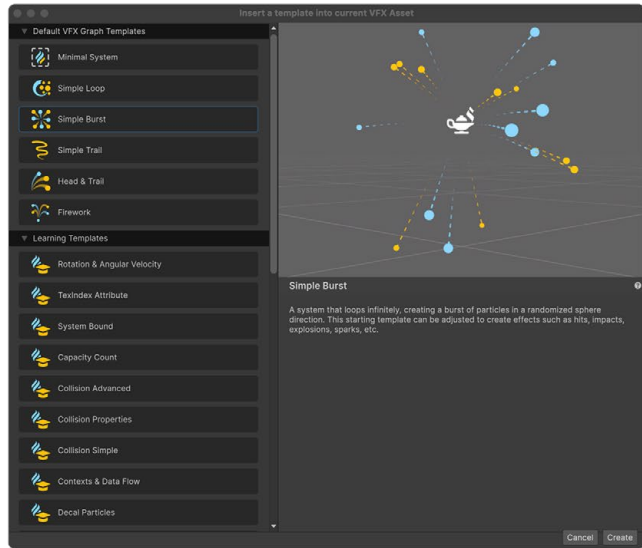


그래프의 세로형 로직은 위에서 아래로 진행됩니다.

메뉴 드롭다운에서 **Insert template**을 선택하여 기존 템플릿의 샘플 시스템을 현재 VFX Graph에 추가합니다. 이렇게 하면 사전에 빌드된 그래프 로직으로 시작할 수 있습니다. 간단한 시스템이라면 기본 VFX Graph 템플릿 중 하나를 선택하거나, 의도한 효과와 유사한 것이 있다면 Learning Templates 중 하나를 선택합니다(아래 [VFX 샘플 콘텐츠 살펴보기](#) 참조).

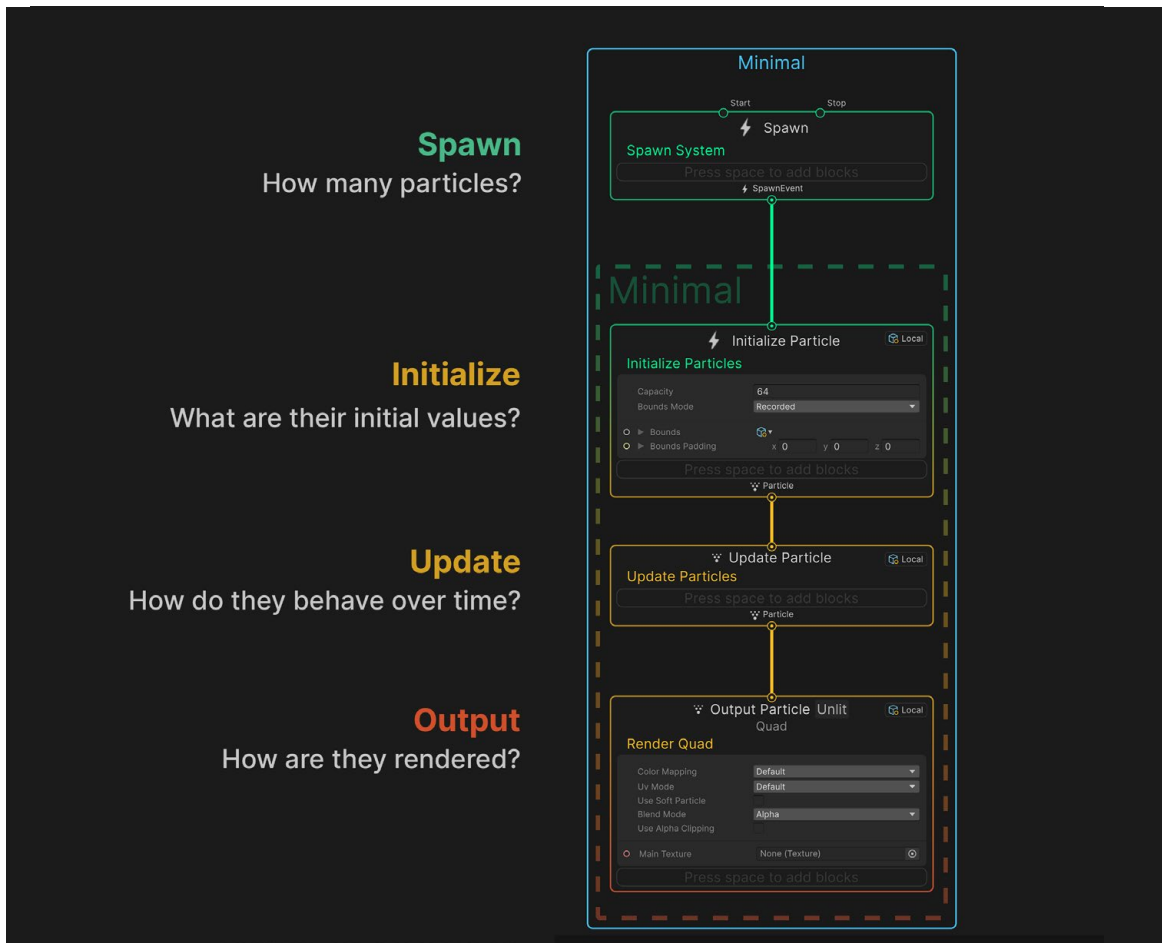


메뉴에서 템플릿을 삽입합니다.



씬 및 게임(Game) 뷰의 시각 효과

기본 VFX Graph 템플릿에서 **Minimal System** 템플릿을 선택하면 다음과 같이 네 부분으로 구성된 기본 시스템을 볼 수 있습니다.



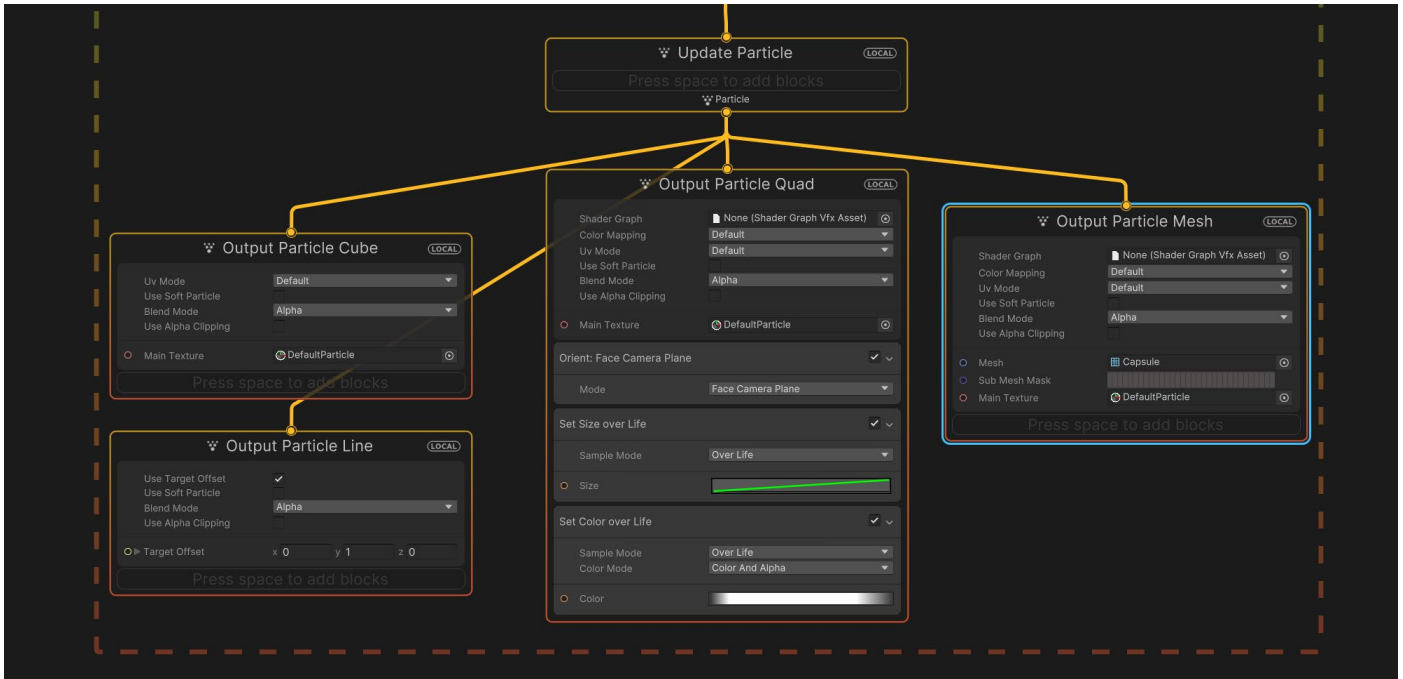
기본 템플릿의 Minimal System

컨텍스트 간의 플로우에 따라 파티클 생성 방식과 시뮬레이션 방식이 결정됩니다. 각 컨텍스트가 한 단계의 계산을 정의합니다.

- **Spawn:** 생성할 파티클의 수와 생성 시기(예: 한 번에, 반복, 딜레이 적용 등)를 결정합니다.
- **Initialize:** 파티클의 시작 속성, 용량(최대 파티클 수), 경계(효과가 렌더링되는 볼륨)를 결정합니다.
- **Update:** 각 프레임별 파티클 프로퍼티를 변경합니다. 힘을 적용하거나, 애니메이션을 추가하거나, 충돌을 생성하거나, SDF(Signed Distance Fields) 같은 여러 상호 작용을 설정할 수 있습니다.
- **Output:** 파티클을 렌더링하고 최종 모습(컬러, 텍스처, 방향)을 결정합니다. 시스템마다 여러 출력을 적용해 보다 유연하게 활용할 수 있습니다.

시스템 및 컨텍스트는 그래프의 세로형 로직, 즉 **처리 워크플로**의 기반이 됩니다. 시스템의 데이터는 위에서 아래로 처리되며, 진행 과정에서 발생하는 각 컨텍스트가 시뮬레이션에 따라 데이터를 수정합니다.

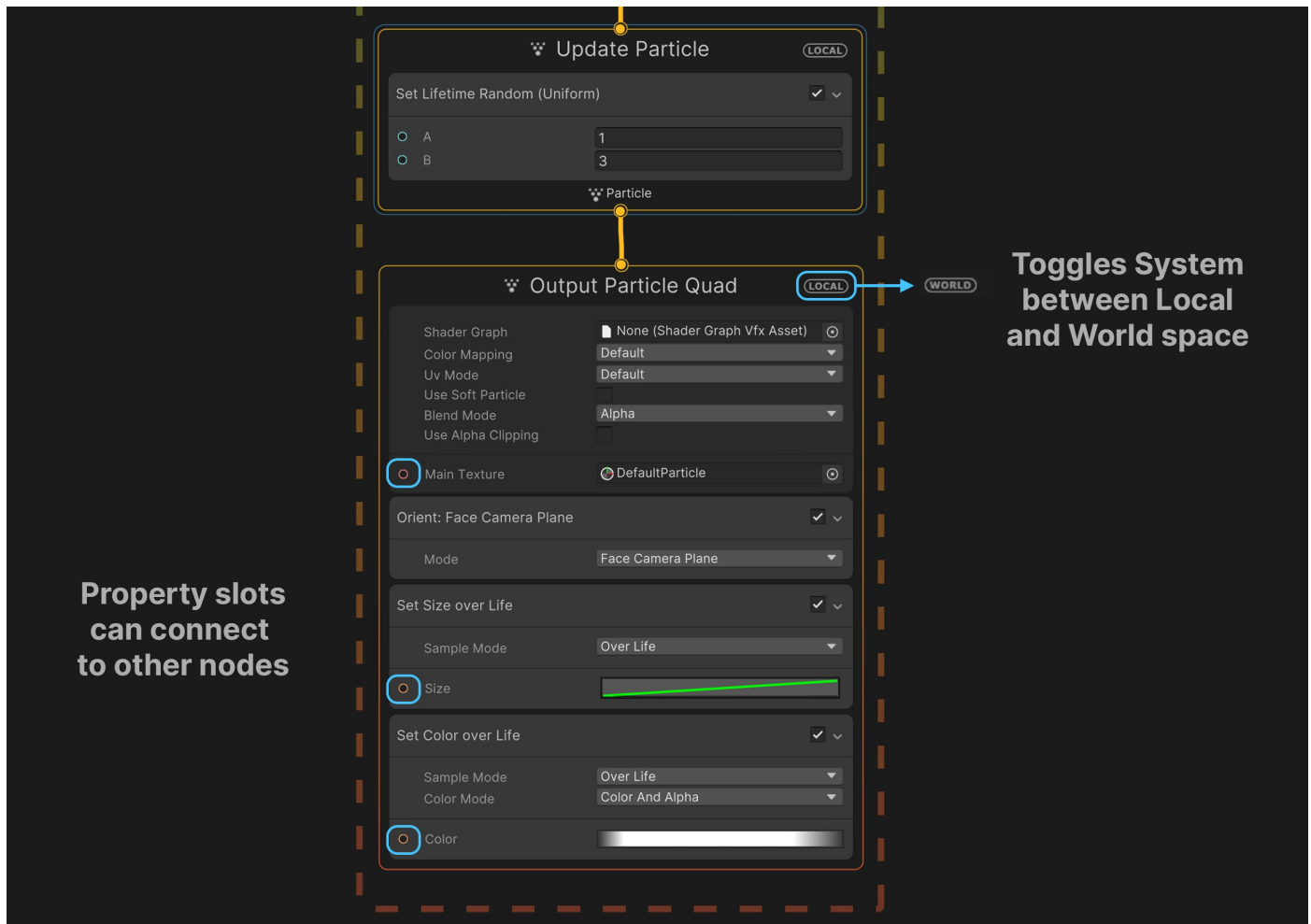
시스템은 유연하므로 필요에 따라 컨텍스트를 생략하거나 여러 출력을 서로 연결할 수 있습니다. 이 예제에서는 동일한 시스템 내에서 두 개 이상의 **Output 컨텍스트**를 렌더링하는 모습을 살펴볼 수 있습니다.



같은 시스템 안에 둘 이상의 Output 컨텍스트가 있는 모습

컨텍스트 자체는 개별 블록에 따라 다르게 작동하며, 시스템과 유사하게 위에서 아래로 데이터를 계산합니다. 더 많은 블록을 추가하고 조작해서 해당 데이터를 처리할 수 있습니다.

컨텍스트 오른쪽 상단에 있는 버튼을 클릭하면 시스템의 시뮬레이션 공간을 **Local**과 **World**로 토글할 수 있습니다.



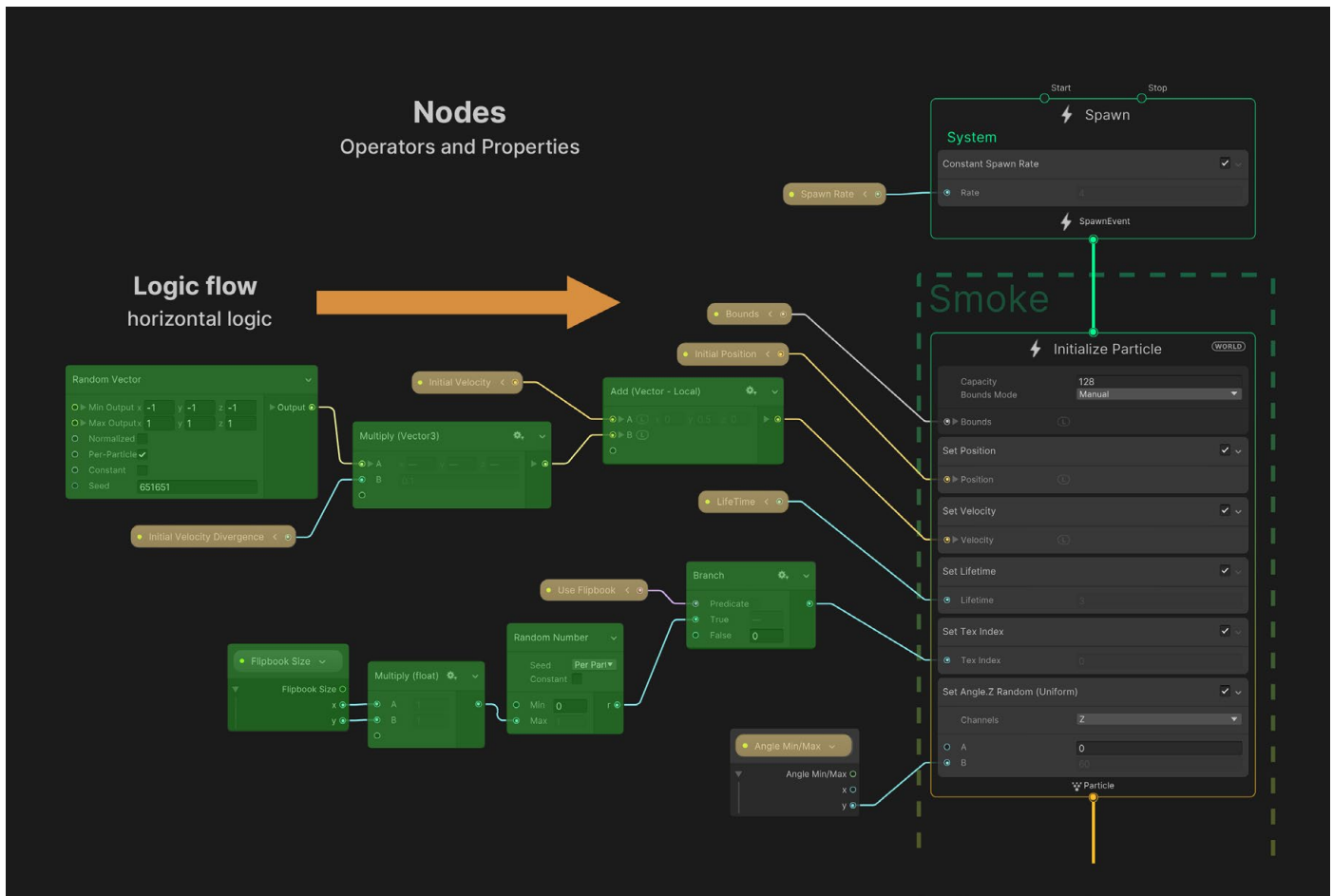
다양한 블록의 예시

블록은 컬러에 대한 간단한 값 저장부터 **노이즈**, **힘**, **충돌** 등의 복잡한 연산까지 거의 모든 작업을 수행할 수 있습니다. 왼쪽에 연산자와 프로퍼티의 입력을 수신할 수 있는 슬롯이 있는 경우가 많습니다.

전체 컨텍스트 및 블록 목록은 [노드 라이브러리](#)를 참조하세요.

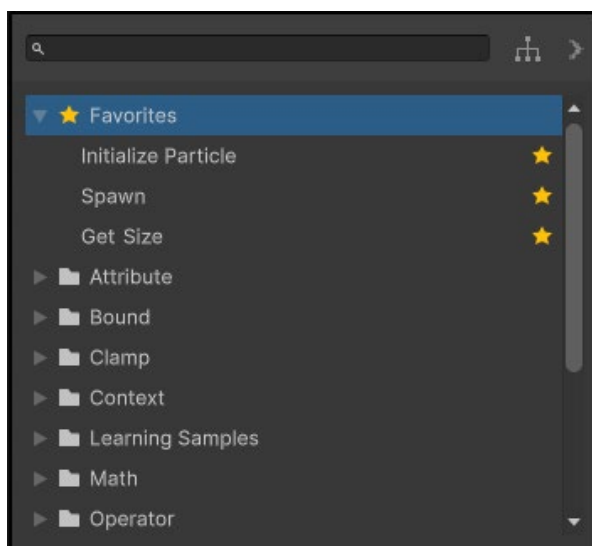
프로퍼티 및 연산자

시스템이 그래프에서 세로형 로직의 대부분을 구성하듯이 연산자는 [프로퍼티 워크플로](#)의 '가로형 로직'을 구성합니다. 연산자를 사용하면 커스텀 표현식이나 값을 블록에 전달할 수 있습니다.



가로형 로직

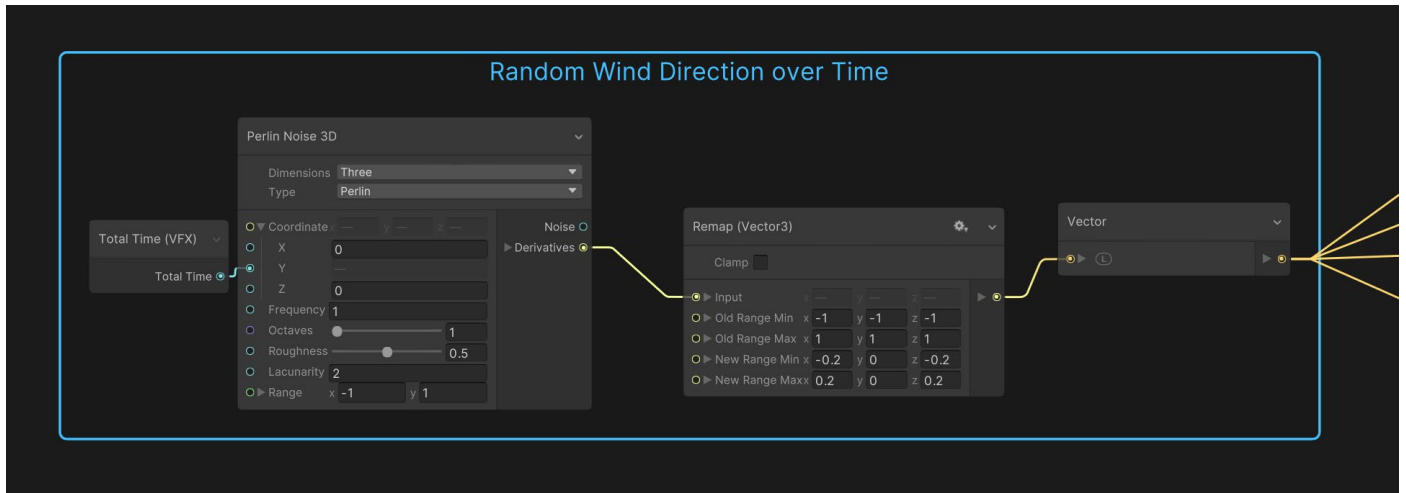
연산자는 Shader Graph 노드처럼 왼쪽에서 오른쪽으로 진행됩니다. 연산자를 사용해 값을 처리하거나 다양한 계산을 수행할 수 있습니다.



메뉴에서 연산자 노드를 생성합니다.

Create Node 메뉴(오른쪽 클릭 또는 스페이스바 사용)에서 **연산자 노드**를 만들 수 있습니다. 프로퍼티 슬롯에서 Edge Connection을 드래그하여 빈 공간에 놓으면 호환되는 연산자만 표시된 동일한 메뉴가 열립니다.

이러면 아래 Bonfire 샘플의 연산자는 무작위 바람 방향을 계산합니다.

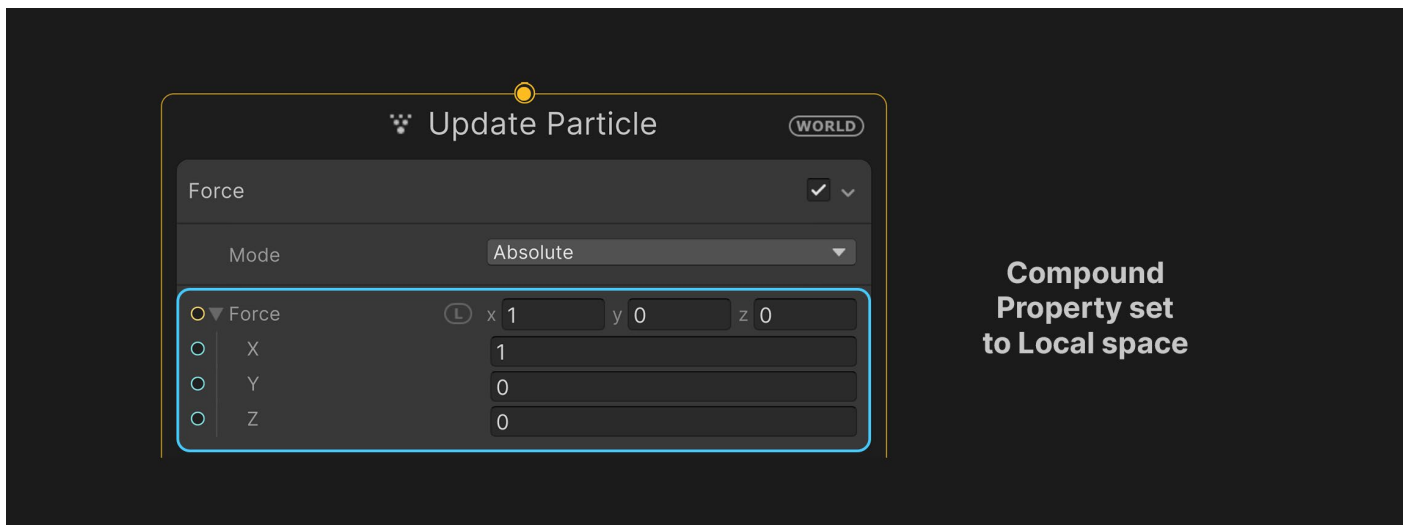


Bonfire 샘플에서 바람의 방향이 결정되는 방식

프로퍼티는 편집 가능한 필드로, **프로퍼티 워크플로**를 사용하여 그래프 요소에 연결됩니다. 프로퍼티는 다음과 같습니다.

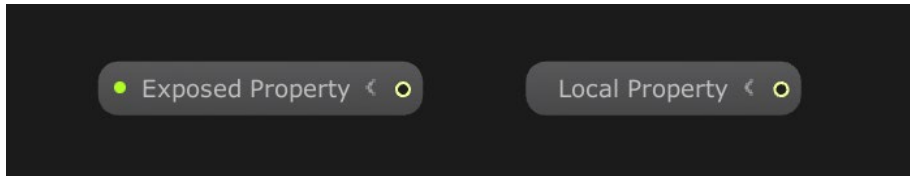
- 정수, 플로트, 부울을 비롯한 모든 **유형**
- **Compound** 컴포넌트로 구성되는 **벡터 및 컬러** 등
- **캐스트 및 전환**(예: 정수에서 플로트로)
- **로컬 또는 월드 공간**(**L** 또는 **W** 키를 클릭해 전환)

프로퍼티는 그래프에서 실제 값에 따라 값이 변경됩니다. 프로퍼티 왼쪽에 있는 입력 포트를 다른 그래프 노드에 연결할 수 있습니다.



한 블록 내의 Force 프로퍼티

프로퍼티 노드는 그래프의 여러 위치에서 **같은 값을 재사용** 하는 데 사용할 수 있는 연산자입니다. 프로퍼티 노드는 상응하는 전역 프로퍼티를 가지며, 이 프로퍼티는 Blackboard에 표시됩니다.



프로퍼티 노드

Blackboard

Blackboard 유틸리티 패널은 **프로퍼티** 및 **속성**을 관리합니다. 이 패널을 열려면 Toolbar 창에서 **Blackboard** 버튼을 클릭하거나 기본 Shift-1 단축키를 사용합니다.

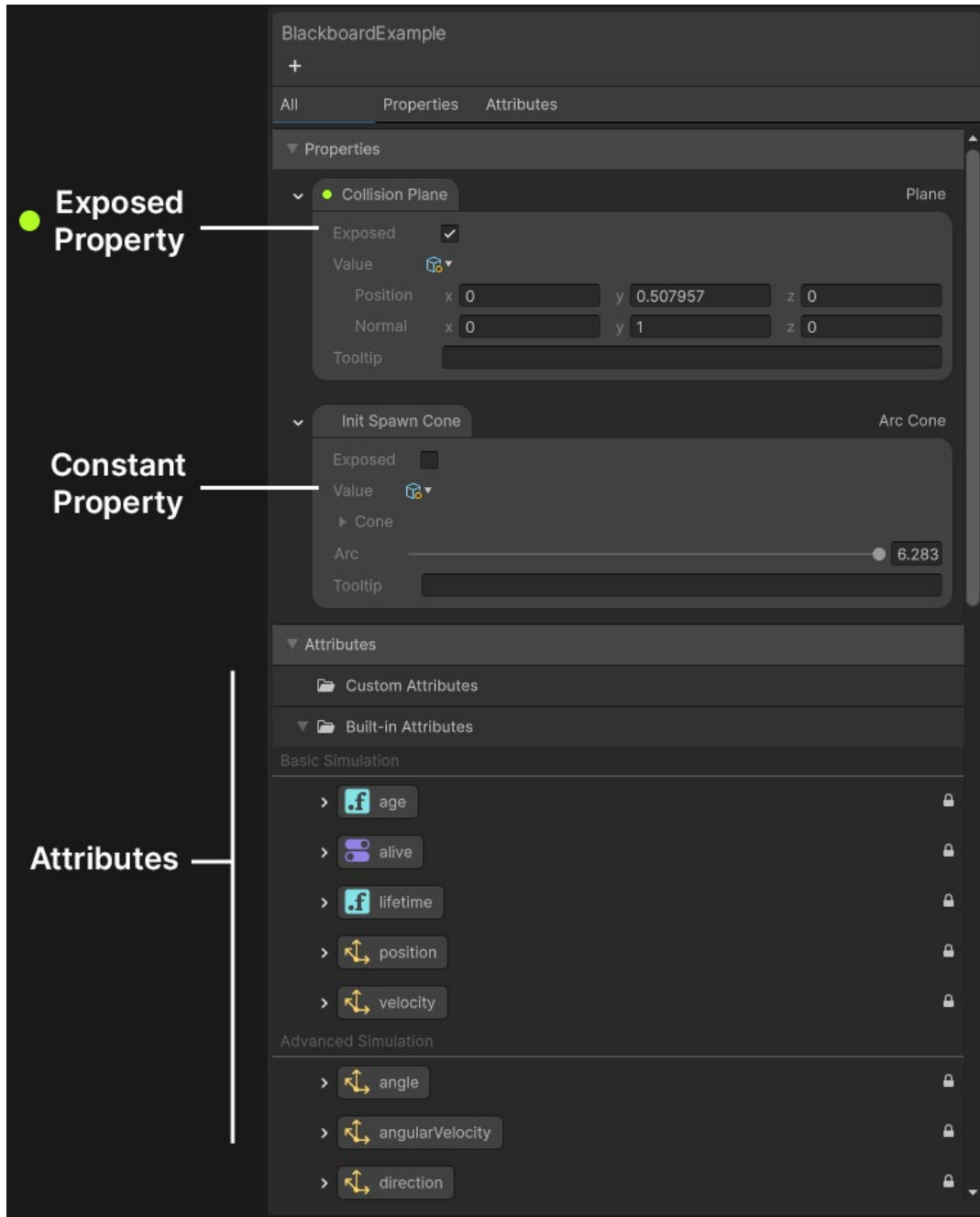
프로퍼티 및 속성을 함께 보려면 Blackboard 상단의 **All** 탭을 선택합니다. 유형별로 필터링하려면 해당 Properties 또는 Attributes 탭을 선택합니다.

Blackboard에서 정의한 프로퍼티는 그래프 전체에서 **프로퍼티 노드**로 재사용할 수 있는 전역 변수로 작동합니다. 예를 들어 bounding box 프로퍼티를 한 번 정의한 다음 동일한 그래프 내의 여러 파티클 시스템에 이를 적용할 수 있습니다.

Blackboard의 프로퍼티는 다음 중 하나에 속합니다.

- **Exposed:** Exposed 프로퍼티 왼쪽에 초록색 점이 보이면 그래프 외부에서 해당 프로퍼티를 보고 편집할 수 있습니다. 스크립트에서 **Exposed Property 클래스**를 사용하여 인스펙터에서 Exposed 프로퍼티에 액세스합니다.
- **Constant:** 초록색 점이 없는 Blackboard 프로퍼티는 Constant 프로퍼티입니다. 그래프 내에서 재사용할 수 있지만, 인스펙터에는 표시되지 않습니다.

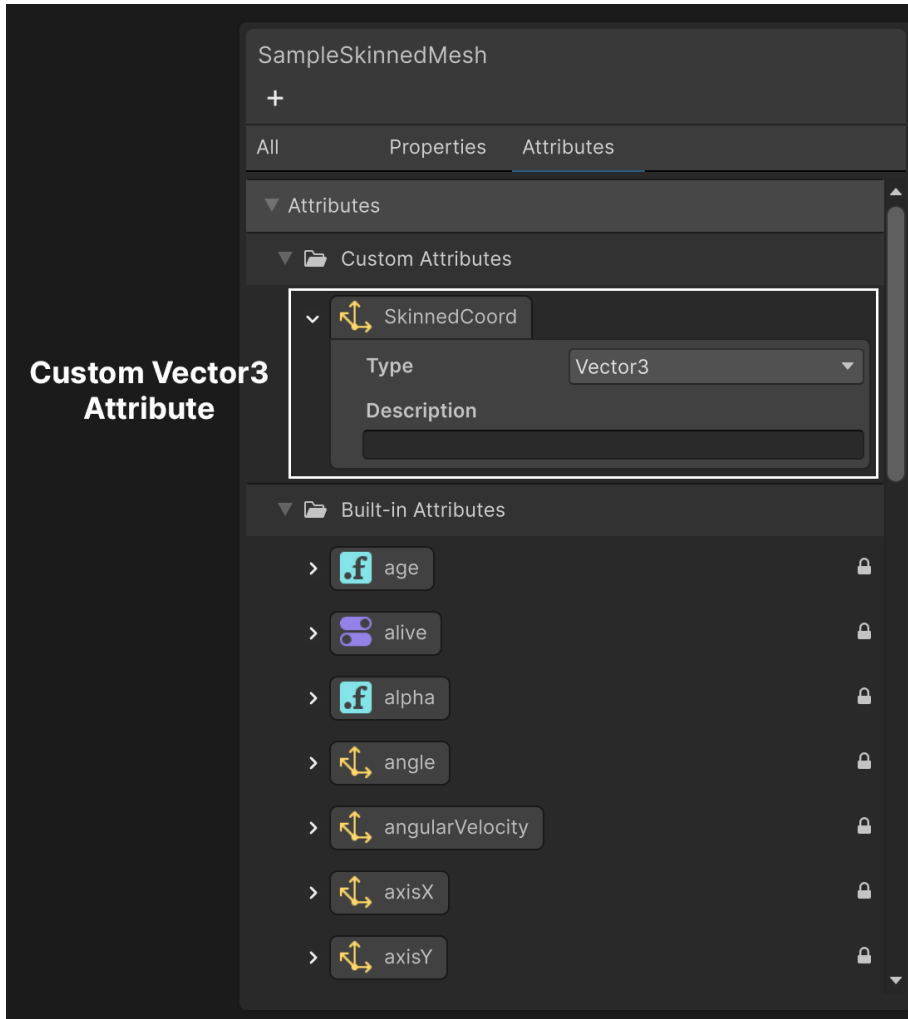
새 프로퍼티는 기본적으로 Exposed로 설정되며, 따라서 인스펙터에 표시됩니다. 그래프 외부에서 프로퍼티를 숨기려면 **Exposed** 옵션을 선택 해제하고 **카테고리**를 생성하여 프로퍼티를 정리해야 합니다.



Blackboard 및 사용할 수 있는 프로퍼티

Blackboard에서는 그래프에 드래그 앤 드롭하거나 인터페이스에서 직접 생성할 수 있는 빌트인 및 커스텀 속성을 모두 관리합니다. 각 속성에는 간단한 설명이 포함되어 있습니다. 속성에 마우스 커서를 올리면 그래프에서 해당 속성이 나타나는 위치를 강조 표시할 수 있습니다.

자세한 내용은 아래의 [Blackboard 속성](#)을 참조하세요.



Sample Skinned Mesh 템플릿에는 커스텀 속성이 포함되어 있습니다.

그룹 노드 및 스티커 메모

그래프 로직의 규모가 커짐에 따라 그룹 노드 및 [스티커 메모](#)를 사용하여 노드를 정리할 수 있습니다. 그룹 노드를 사용하면 노드 그룹에 레이블을 지정하여 한꺼번에 이동시킬 수 있습니다. 스티커 메모는 코드 주석 같은 역할을 합니다.

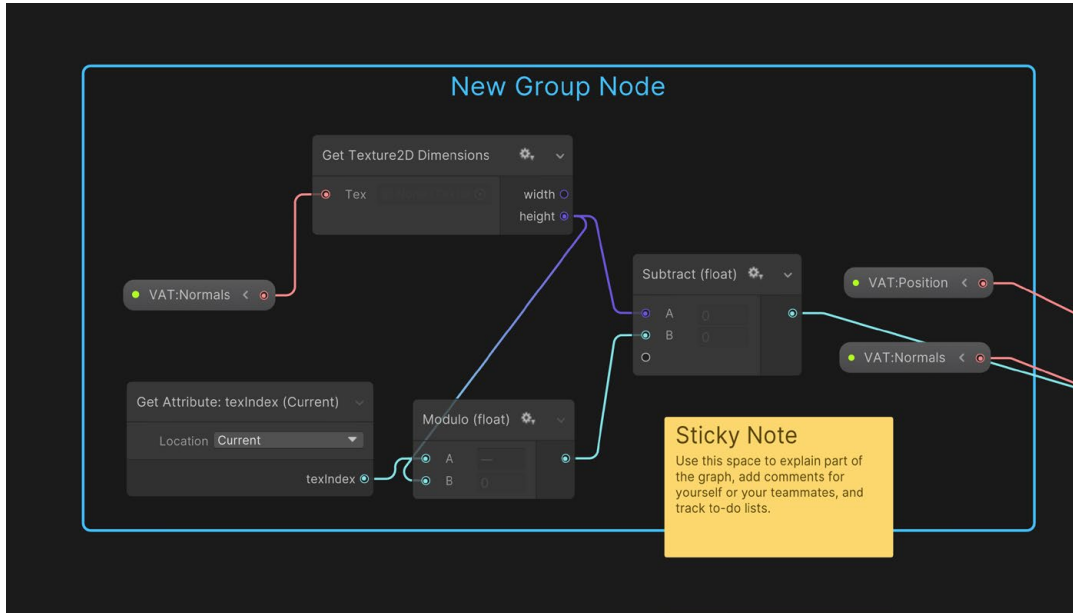
그룹 노드를 만들려면 여러 노드를 선택하고 오른쪽 클릭한 후 **Group Selection**을 선택합니다. 새로운 기본 단축키인 Shift+G를 사용할 수도 있습니다.

노드를 기존 그룹 노드로 드래그 앤 드롭할 수도 있습니다. 그룹 위로 노드를 가져갔다가 그룹이 강조 표시될 때 놓으면 됩니다. 그룹에서 노드를 제거하려면 Shift 키를 누른 상태에서 노드를 밖으로 드래그합니다.

Delete 키를 사용하거나 오른쪽 클릭 메뉴를 통해 그룹 노드를 삭제해도 그룹에 포함된 노드는 삭제되지 않습니다.

스티커 메모를 사용하면 그래프 섹션의 작동 방식을 설명하고 자신 또는 팀원에게 주석을 남길 수 있습니다. 스티커 메모는 원하는 만큼 추가할 수 있으며, 옮기거나 크기를 조절하는 데에도 제한이 없습니다.

각 스티커 메모에는 제목과 본문이 있습니다. 그래프 뷰에서 오른쪽 클릭하여 스티커 메모를 만듭니다. 텍스트 영역을 더블 클릭하면 내용을 편집할 수 있습니다. 오른쪽 클릭 메뉴에서 테마 컬러(어둡게/밝게)와 텍스트 크기를 설정하여 노트를 구성하세요.



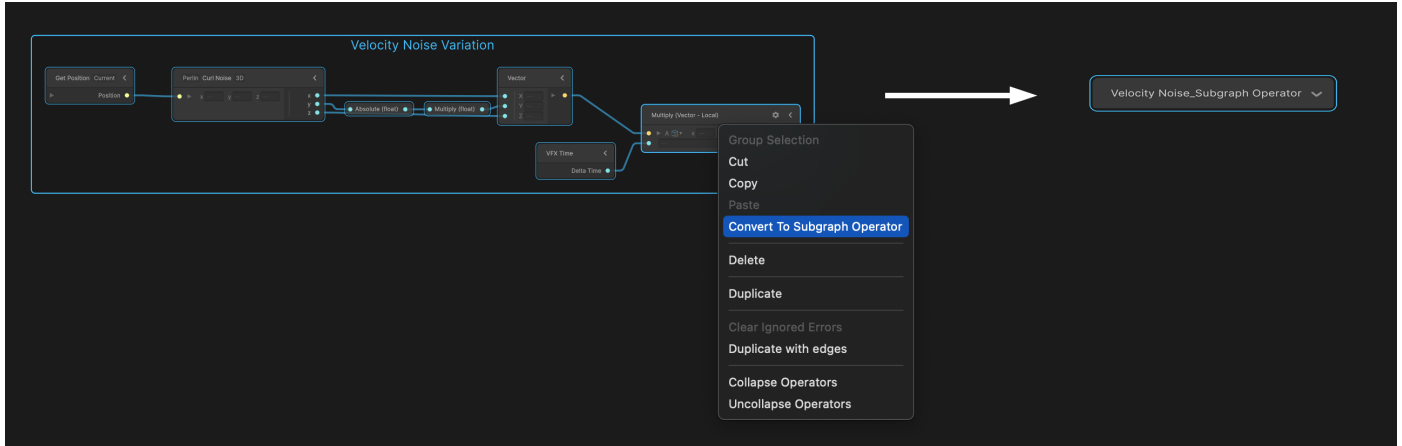
그룹 노드와 스티커 메모를 활용하세요.

하위 그래프

하위 그래프는 단일 노드로 표시되며 그래프 로직을 깔끔하게 정리하는 데 도움이 됩니다. 하위 그래프를 사용하여 VFX Graph의 일부를 별도의 에셋으로 저장하면 다른 VFX Graph로 가져가서 재구성하고 재활용할 수 있습니다. 시스템, 블록, 연산자를 다양한 유형의 하위 그래프로 묶을 수 있습니다.

하위 그래프는 프로젝트 창에서 직접 만들 수 있습니다. **Create > Visual Effect > Subgraph Operator** 또는 **Subgraph Block**으로 이동하여 새 하위 그래프를 처음부터 시작합니다. 이 방법을 사용하면 기존 VFX Graph에 하위 그래프를 먼저 만들지 않고도 하위 그래프를 디자인할 수 있습니다.

또는 VFX Graph를 편집할 때 노드 세트를 선택한 다음 오른쪽 클릭 메뉴에서 적절한 하위 그래프 유형(블록 또는 연산자)을 선택하여 하위 그래프를 생성할 수 있습니다. 예를 들어 연산자 세트를 전환하려면 **Convert To Subgraph Operator**를 선택합니다. 에셋을 디스크에 저장하면 선택한 노드가 단일 하위 그래프 노드로 대체됩니다.



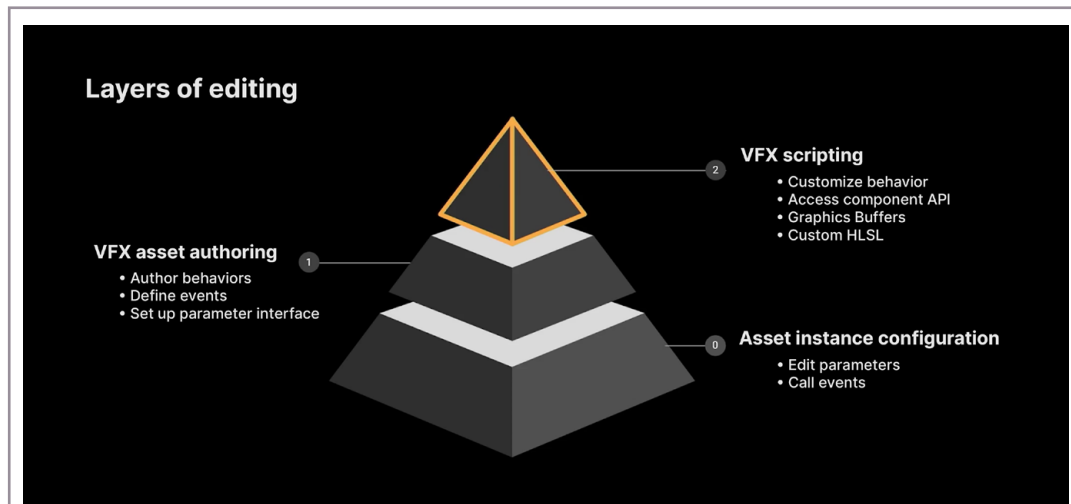
하위 그래프 생성 방법

하위 그래프에 Input 프로퍼티를 생성하려면 Blackboard에 새 프로퍼티를 추가하고 해당 Exposed 플래그를 활성화합니다.

하위 그래프에 Output 프로퍼티를 생성하려면 새 프로퍼티를 추가한 다음 Blackboard의 Output 카테고리로 옮깁니다.

또한 Blackboard에서는 하위 그래프 블록이 나타나는 메뉴 카테고리를 정의할 수 있습니다. 이를 사용하여 하위 그래프를 정렬하거나 검색할 수 있습니다.

하위 그래프를 만드는 과정은 코드 리팩터링과 비슷합니다. 로직을 재사용 가능한 메서드나 함수로 구성하듯이, 하위 그래프는 VFX Graph의 요소를 모듈화합니다.



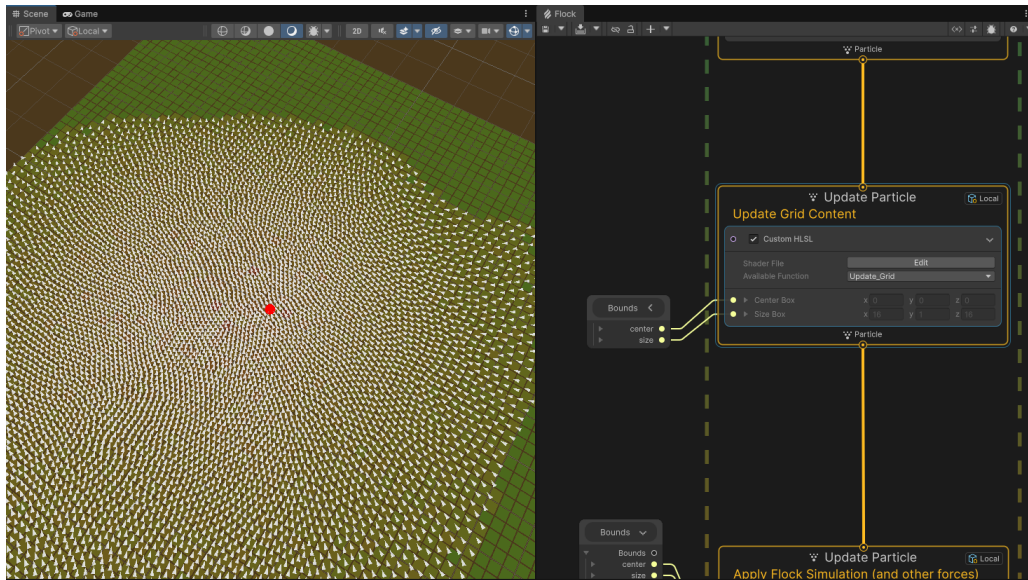
VFX Graph의 편집 레이어

VFX Graph는 다음 세 가지 수준의 편집을 지원합니다.

- **에셋 인스턴스 설정:** 이 설정을 사용해 기존 VFX Graph를 수정할 수 있습니다. 디자이너와 프로그래머 모두 인스펙터에서 노출 파라미터를 조정하여 효과의 모양, 타이밍, 설정을 조정할 수 있습니다. 아티스트가 외부 스크립팅이나 이벤트를 사용하여 미리 저작된 콘텐츠를 변경할 수도 있습니다. 이 수준에서는 각 그래프를 블랙박스로 취급합니다.

- **VFX 에셋 저작(authoring):** 창의력을 마음껏 발휘할 수 있는 방법입니다. 연산자 노드로 이뤄진 연결망을 만들어 자신만의 VFX Graph를 만들고, 커스텀 동작과 파라미터를 설정하여 커스텀 시뮬레이션을 만들 수 있습니다. 기존 샘플을 활용하거나 처음부터 새로 만드는 등 특정 효과를 원하는 대로 만들 수 있습니다.
- **VFX 스크립팅:** **컴포넌트 API**를 사용해 VFX Graph의 동작을 커스터마이징할 수 있습니다. 보다 경험이 많은 테크니컬 아티스트나 그래픽스 프로그래머를 위한 기능입니다. VFX 스크립팅을 사용하면 특정 효과를 더 효율적으로 관리하는 파이프라인을 사용하고, 그래픽스 버퍼 같은 고급 기능도 사용할 수 있습니다.

Unity 6의 **Custom HLSL**을 사용하면 표준 VFX Graph 노드로 쉽게 구현할 수 없는 복잡하거나 독특한 파티클 동작을 구현할 수 있습니다. 예를 들어 커스텀 물리 시뮬레이션, 파티클 상호 작용, 군집 동작을 만들 수 있습니다.



Custom HLSL은 커스텀 동작이나 상호 작용을 제작할 수 있습니다.

속연도에 관계없이 누구나 VFX Graph로 효과를 만들어 볼 수 있습니다. 미리 만들어진 효과를 사용하여 워크플로에 익숙해진 다음, 조금씩 자신만의 그래프를 만들어 보세요.

속성

속성은 파티클의 컬러, 위치, 크기, 생성할 파티클 수 등 시스템 내에서 사용할 수 있는 데이터 조각입니다. 속성은 시뮬레이션 중에 읽거나 수정하여 동적 효과를 만드는 데 사용할 수 있습니다.

속성은 `typexq float`(단정밀도 부동 소수점), `Vector2`, `Vector3`, `Vector4`(2D, 3D, 4D 벡터), `bool`(`true`/`false`), `int`/`uint`(정수/부호 없는 정수)일 수 있습니다.

VFX Graph에서 자주 사용하는 몇 가지 속성은 다음과 같습니다.

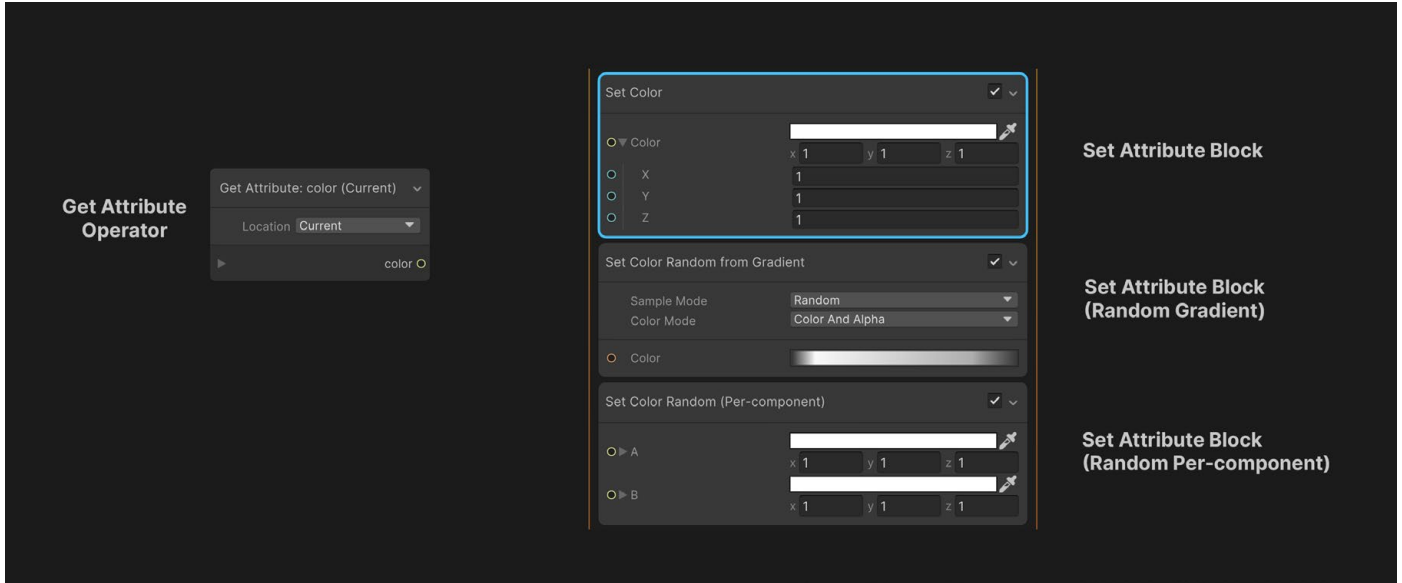
속성	유형	설명
position	Vector3	시스템 공간에서 파티클의 위치를 정의합니다.
velocity	Vector3	파티클의 이동 속도 및 방향을 결정합니다.
age	float	시물레이션된 요소가 생성된 후 얼마나 지났는지를 초 단위로 표시합니다.
lifetime	float	시물레이션된 요소의 기대 수명을 초 단위로 표시합니다.
color	Vector3	파티클의 RGB 컬러를 설정합니다.
alpha	float	파티클의 투명도를 조정합니다.
size	float	파티클의 균일한 크기를 제어합니다.
scale	Vector3	파티클에 균일하지 않은 스케일링을 적용합니다.
mass	float	파티클의 질량을 뜻하며 중력과 같은 힘에 반응하는 방식에 영향을 미칩니다.
direction	Vector3	임의의 방향을 저장하거나 파티클 이동을 유도하는 데 사용할 수 있습니다.
angularVelocity	Vector3	파티클의 회전 속도를 초당 각도(°/s) 단위로 표시합니다.
targetPosition	Vector3	파티클이 나아가야 하는 목표 위치를 뜻하며 라인 렌더러 같은 특정 효과에 유용합니다.
texIndex	float	렌더링된 요소의 플립북 UV 샘플링에 사용되는 애니메이션 프레임입니다.

속성은 VFX Graph 파티클의 기본적인 측면을 관리하는 데 필수입니다. 전체 목록은 표준 속성 [기술 자료 페이지](#)를 참조하세요.

Blackboard의 Attributes 탭([아래 참조](#))을 사용하여 여러 빌트인 속성을 탐색하거나 커스텀 속성을 정의할 수도 있습니다.

노드를 사용하여 속성을 읽거나 쓸 수 있습니다. 사용해 볼 만한 노드는 다음과 같습니다.

- **Get Attribute Operator: Particle** 또는 **ParticleStrip System**의 속성에서 읽기
- **Experimental Spawner Callbacks: Spawn systems**의 속성에서 읽기
- **Set Attribute Block**: 속성에 값 쓰기, 속성의 값을 직접 설정하거나 랜덤 모드 사용(예: **Color Attribute**에 **Random Gradient** 또는 **Random Per-component Block** 설정)



연산자로 속성을 가져오고 블록으로 속성을 설정합니다.

대부분의 속성은 파티클별로 저장되므로 파티클과 속성의 수가 증가하면 메모리 사용량이 늘어날 수 있습니다. 예를 들어 파티클이 10,000개고 각 파티클이 위치, 속도, 컬러, 크기 같은 여러 속성을 저장하는 경우 이 데이터를 유지하는 데 필요한 메모리가 상당히 커질 수 있습니다.

컨텍스트를 선택하면 인스펙터에 표시되는 System Attribute Summary 및 Current Attribute Layout을 사용하여 속성을 모니터링하고 최적화할 수 있습니다.

- **System Attribute Summary:** 이 섹션에서는 현재 시스템에서 사용 중인 모든 시스템 수준의 속성에 대한 개요를 제공합니다.
- **Current Attribute Layout:** 선택한 컨텍스트에서 사용된 속성을 표시합니다.
- **Source Attribute Layout:** 소스 컨텍스트(초기 데이터 또는 입력을 제공하는 컨텍스트)에서 사용된 속성(예: Initialize 컨텍스트에서 초기화된 다음 Update 컨텍스트에서 사용된 속성)을 표시합니다.

예를 들어 **Trigger Event on Collide** 템플릿에서 Dart_Spawn 시스템의 컨텍스트를 선택하면 인스펙터에 다음과 같이 표시됩니다.

Name	Value		
Capacity	32		
System Attribute Summary			
alive	Boolean	4 bytes	ReadWrite
color	Float3	12 bytes	ReadWrite,
angleZ	Float	4 bytes	ReadWrite,
direction	Float3	12 bytes	ReadWrite
lifetime	Float	4 bytes	ReadWrite
seed	UInt32	4 bytes	ReadWrite
position	Float3	12 bytes	ReadWrite
velocity	Float3	12 bytes	ReadSourc
collisionEventPosition	Float3	12 bytes	ReadSourc
particleId	UInt32	4 bytes	Read
pivotX	Float	4 bytes	Read
pivotY	Float	4 bytes	Read
pivotZ	Float	4 bytes	Read
size	Float	4 bytes	ReadWrite
scaleX	Float	4 bytes	Read
scaleY	Float	4 bytes	Read
scaleZ	Float	4 bytes	Read
axisX	Float3	12 bytes	ReadWrite
axisY	Float3	12 bytes	ReadWrite
axisZ	Float3	12 bytes	ReadWrite
angleX	Float	4 bytes	ReadWrite
angleY	Float	4 bytes	ReadWrite
age	Float	4 bytes	ReadWrite
Current Attribute Layout			
0	alive	age	
1	color	color	color
2	directio	directio	directio
3	lifetime	angleZ	positior
		positior	positior
		positior	particle
Source Attribute Layout			
0	positior	positior	positior
1	angleZ		
2	lifetime		
3	color	color	color
4	collisior	age	
5	collisior	collisior	collisior

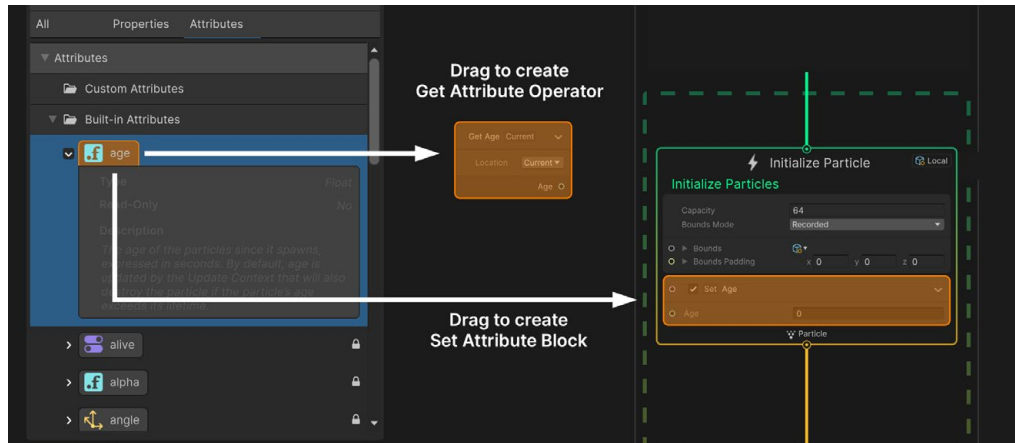
인스펙터를 사용하여 컨텍스트의 속성 사용량을 확인합니다.

시스템은 메모리 사용량을 최적화하기 위해 실제로 필요한 속성만 저장합니다. 속성의 시뮬레이션 데이터가 저장되지 않은 경우, VFX Graph는 불필요한 데이터를 저장하지 않고 기본 상수 값을 사용합니다. VFX Graph의 프로파일링 툴에 대한 자세한 내용은 [최적화](#) 섹션을 참조하세요.

Blackboard 속성

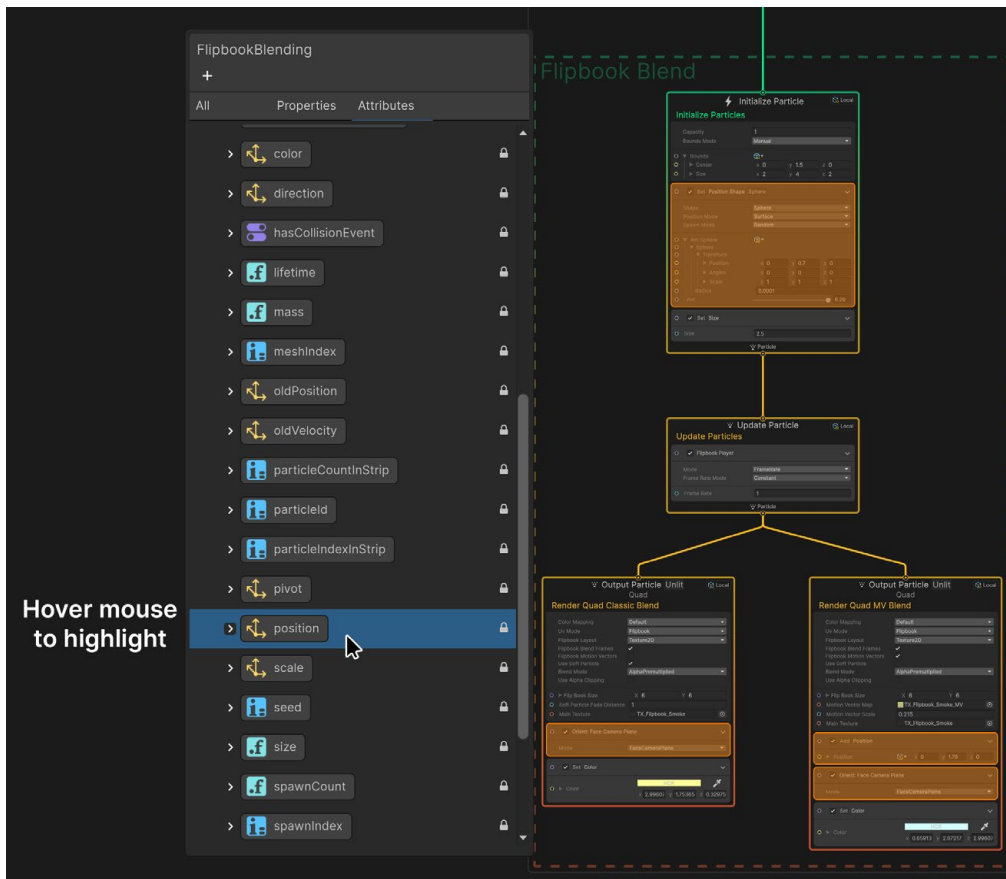
이제 Blackboard 패널에 속성 전용 섹션이 새로 추가되어 컨텍스트에 따른 드래그 앤 드롭으로 해당 연산자 또는 블록을 쉽게 생성할 수 있습니다. 이 새로운 기능은 Unity 6에서 사용할 수 있습니다.

- Blackboard에서 노드 작업 공간으로 드래그 앤 드롭하여 Get 속성 연산자를 생성합니다.
- Blackboard에서 System 컨텍스트로 드래그 앤 드롭하여 **Set 속성 블록**을 생성합니다.



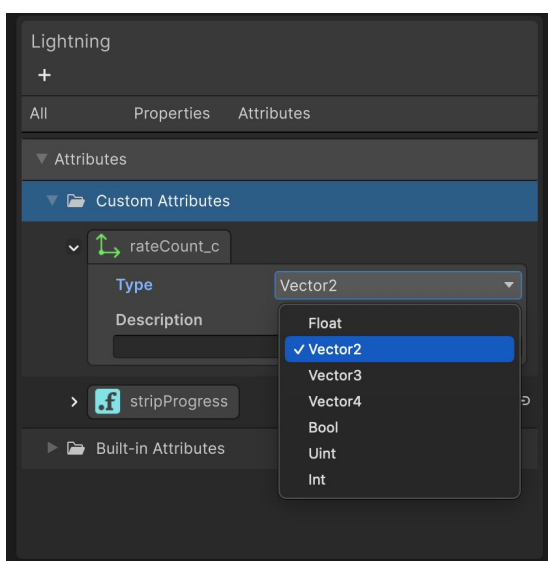
Blackboard에서 속성을 드래그합니다.

- 그래프 요소 위로 마우스 커서를 올려 속성 사용량을 강조 표시합니다.



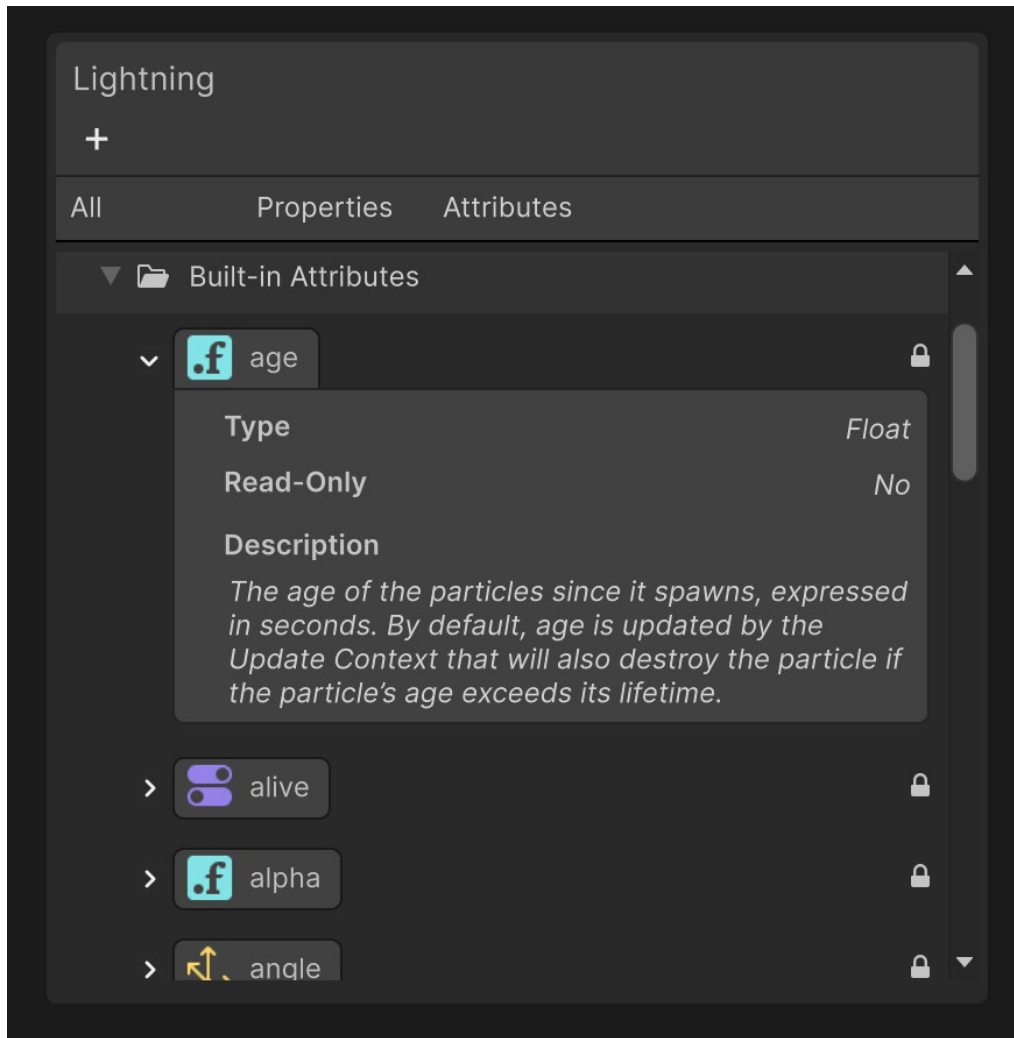
Blackboard에 마우스 커서를 올려 속성 사용량을 강조 표시합니다.

- Blackboard에서 커스텀 속성 연산자를 생성하고 유형을 변경합니다.



커스텀 속성의 유형을 선택합니다.

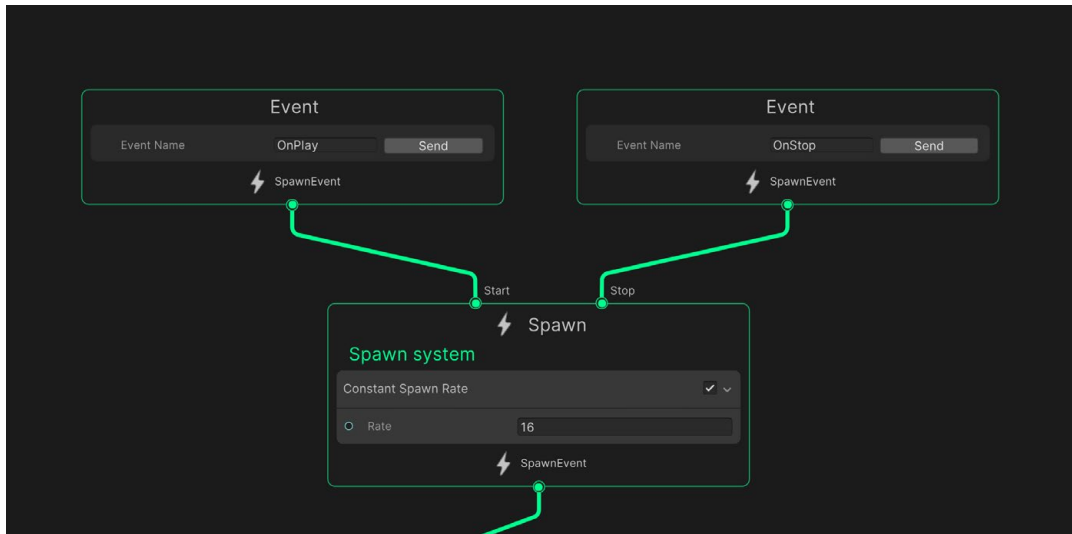
— 빌트인 속성의 설명 및 유형을 확인합니다.



빌트인 속성을 검토합니다.

이벤트

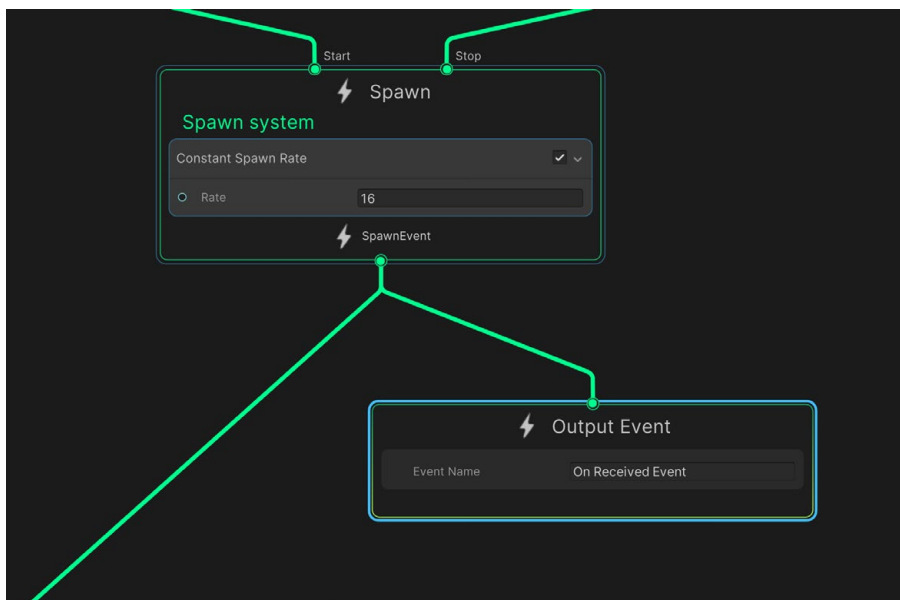
VFX Graph를 구성하는 다양한 요소는 **이벤트**를 통해 서로 또는 씬의 나머지 요소들과 커뮤니케이션합니다. 예를 들어 각 **Spawn 컨텍스트**에는 파티클 생성을 제어하는 이벤트를 수신하는 **Start** 및 **Stop** 플로 포트가 있습니다.



파티클 생성을 제어하는 이벤트

특정 동작이 발생해야 하는 경우, 외부 게임 오브젝트는 **C# API**의 **SendEvent** 메서드를 사용하여 그래프의 일부에 이를 알릴 수 있습니다. 그러면 Visual Effect 컴포넌트는 문자열 이름이나 프로퍼티 ID를 사용해 이벤트를 전달합니다.

Event 컨텍스트는 **그래프 내의 ID 또는 이벤트 문자열 이름**으로 이벤트를 식별합니다. 위 예제에서는 씬에 있는 외부 오브젝트가 **OnPlay 이벤트**를 발생시켜 생성 시스템을 시작하거나 **OnStop 이벤트**를 발생시켜 생성 시스템을 중지할 수 있습니다.

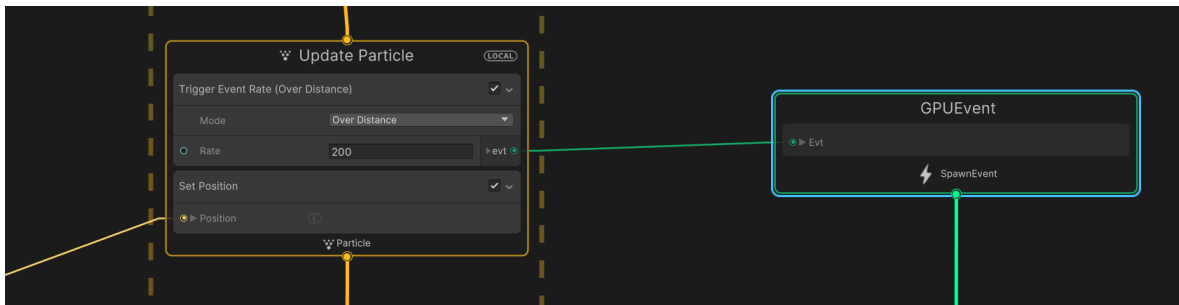


출력 이벤트는 씬에 메시지를 보낼 수 있습니다.

출력 이벤트를 출력 이벤트 핸들러와 결합할 수 있습니다. 출력 이벤트는 파티클이 처음 생성되며 씬에서 다른 무언가를 구동해야 하는 경우에 유용합니다. 조명이나 게임플레이를 시각 효과와 동기화할 때 주로 사용합니다.

위 예제에서는 **OnReceivedEvent**를 그래프 외부의 GameObject 컴포넌트로 전송합니다. 그러면 C# 스크립트가 이에 따라 반응하여 더 강한 빛이나 불길을 뿜어내거나 불꽃이 터지는 등의 작업을 수행합니다. 출력 이벤트에 대한 자세한 내용은 [상호 작용 섹션](#)을 참조하세요.

동시에 GPU 이벤트를 사용하여 다른 파티클 동작을 기준으로 파티클을 생성할 수 있습니다. 이렇게 할 경우, 한 시스템에서 파티클이 사라지면 다른 시스템에 알림을 보낼 수 있으므로, 어떤 발사체 파티클이 사라지면 먼지 효과가 생성되게 하는 등 유용한 연쇄 효과를 구현할 수 있습니다.



GPU Event 컨텍스트는 Trigger Event Rate 블록에서 이벤트를 수신합니다.

이러한 **업데이트 블록**은 다음과 같은 방법으로 **GPU 이벤트** 데이터를 전송할 수 있습니다.

- **Trigger Event On Die:** 파티클이 사라지면 다른 시스템에 파티클 생성
- **Trigger Event Rate:** 매초마다(또는 파티클의 속도에 따라) 파티클 생성
- **Trigger Event Always:** 프레임마다 파티클 생성

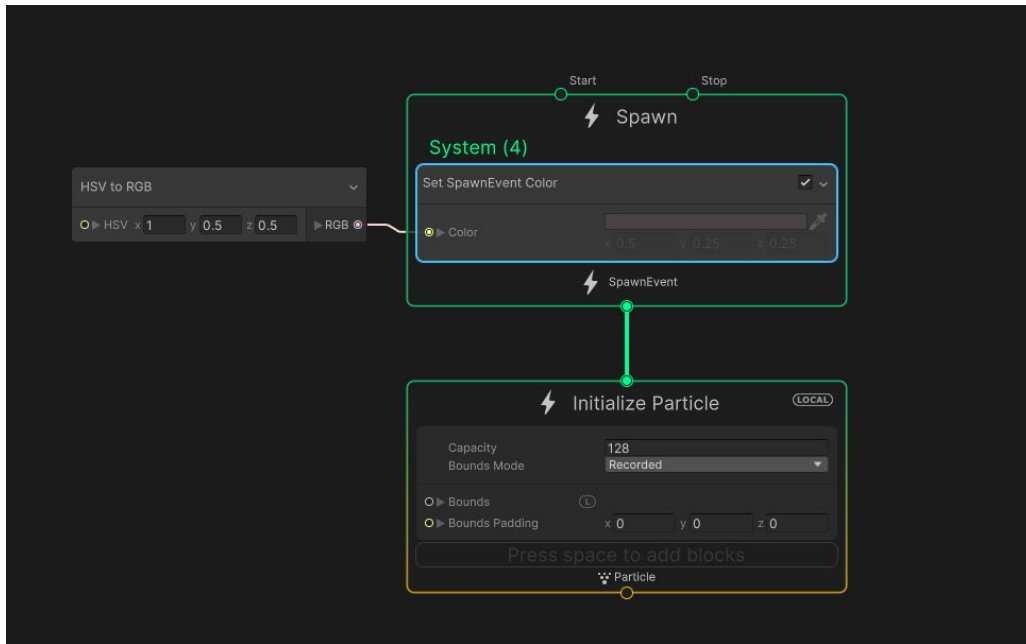
블록의 출력은 **GPU Event 컨텍스트**에 연결되고 종속 시스템의 **Initialize 컨텍스트**에 이를 알릴 수 있습니다. 이러한 방식으로 여러 시스템을 연결하면 풍부하고 복잡한 파티클 효과를 만들 수 있습니다.

GPU 이벤트 시스템의 Initialize 컨텍스트는 트리거 이벤트 이전에 부모 시스템에서 사용할 수 있는 속성을 상속할 수도 있습니다. 예를 들어 위치를 상속하면 새 파티클이 해당 파티클을 생성한 원래 파티클과 같은 위치에 생성됩니다.

이벤트 속성

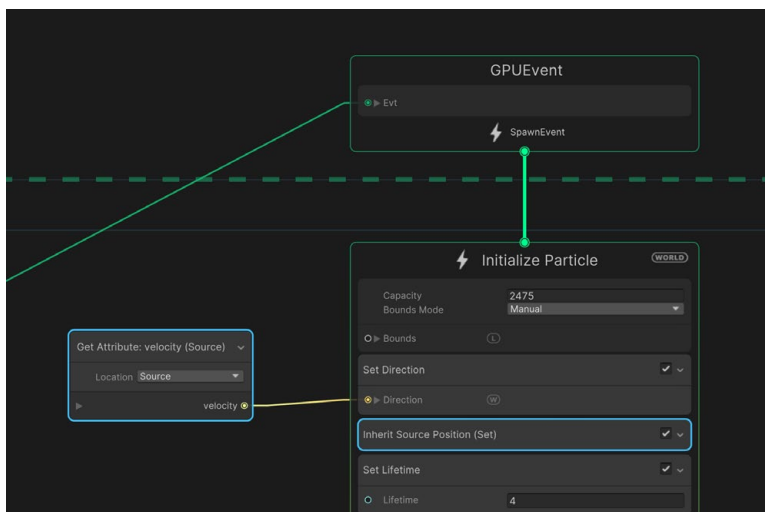
이벤트 속성 페이로드를 사용하면 3D 위치 또는 컬러 등의 데이터를 이벤트와 함께 전달할 수 있습니다. 페이로드는 연산자 또는 블록에서 데이터를 포착할 수 있는 그래프를 따라 암묵적으로 이동하는 속성을 지닙니다.

생성 이벤트 또는 **타임라인 이벤트**로 전달된 속성을 읽을 수도 있습니다. **Set SpawnEvent Attribute** 블록은 Spawn 컨텍스트의 이벤트 속성을 수정합니다.



Spawn 컨텍스트에서 이벤트 속성 읽기

Initialize 컨텍스트에서 페이로드를 포착하려면 **Get Source Attribute** 연산자 또는 **Inherit Attribute** 블록을 사용해야 합니다.

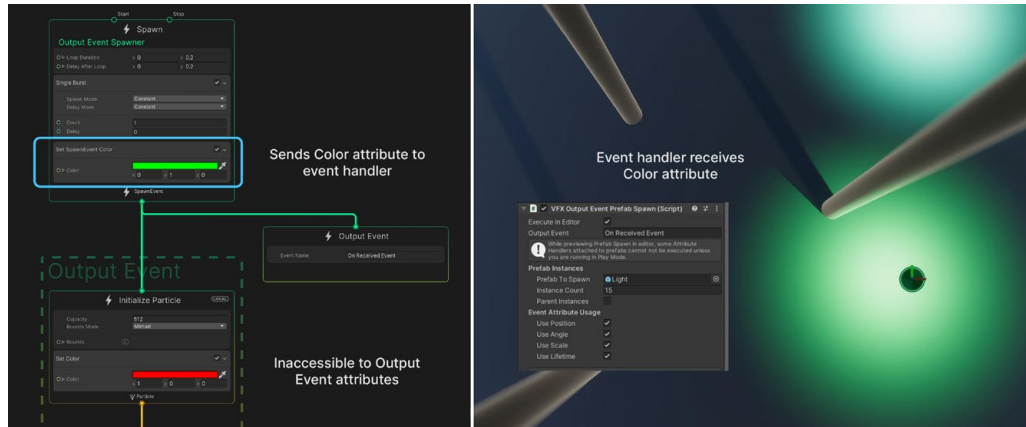


Initialize 컨텍스트에서 이벤트 페이로드 포착

그러나 이벤트 속성을 사용할 때는 다음과 같은 부분에 주의해야 합니다.

- **일반 이벤트 속성**은 Initialize 컨텍스트에서만 읽을 수 있습니다. Update 또는 Output 컨텍스트에서 상속할 수 없습니다. 후반 컨텍스트에서 속성을 사용하려면 Initialize에서 상속하고 설정해야 합니다.
- **출력 이벤트 속성**은 Spawn 컨텍스트에서 설정된 초기 값만 전달합니다. 그래프의 후반부에서 발생하는 변경 사항은 포착하지 않습니다.

Visual Effect 컴포넌트 API의 [이벤트 전송](#)에서 자세한 내용을 확인할 수 있습니다.



출력 이벤트 속성은 Spawn 컨텍스트의 값을 전달합니다.

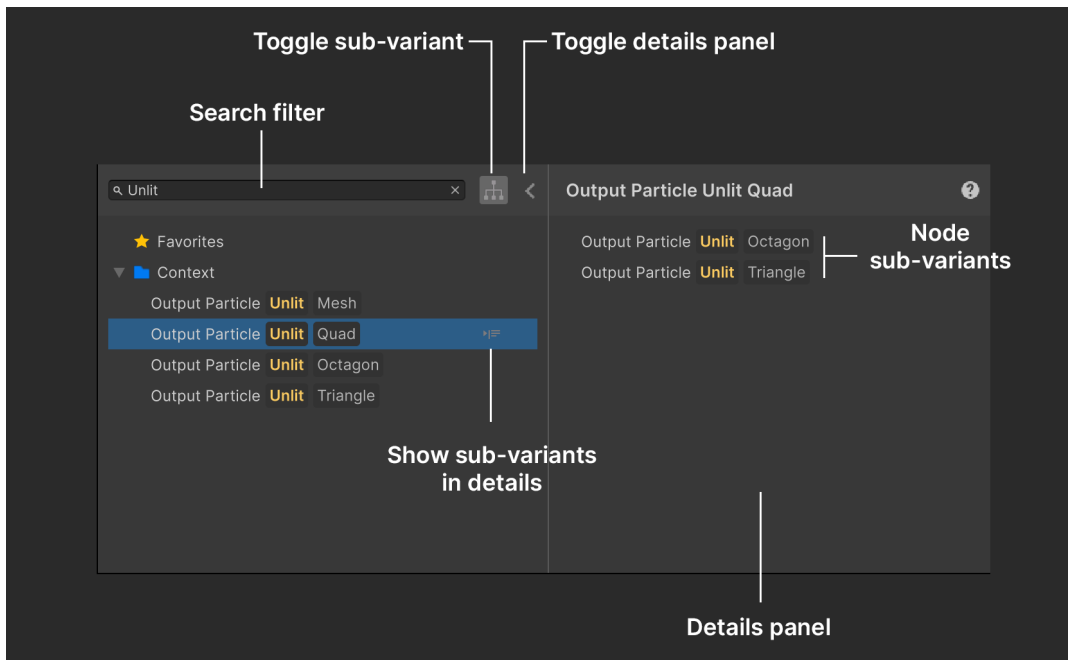
Unity 6의 UI 개선 사항

Unity 6에는 여러 품질 개선 사항과 VFX Graph UI 업데이트가 포함되어 있습니다.

노드 검색

이제 노드 또는 블록을 생성할 때 [계층 트리 뷰](#)를 사용하여 노드 라이브러리를 더 쉽게 탐색할 수 있습니다. 더욱 효율적이고 개인화된 검색 경험을 위해 커스텀 컬러 및 즐겨찾기 폴더가 개선되었습니다. 고급 검색 필터링으로 사용 가능한 노드를 선택할 수 있습니다.

새로운 세부 정보 사이드 패널에는 노드의 하위 배리언트도 표시됩니다(예: Output Particle Unlit Octagon 및 Output Particle Unlit Triangle은 Output Particle Unlit Quad의 하위 배리언트). 버튼을 토글하면 하위 배리언트를 표시하는 방식으로 표시 여부를 제어할 수 있습니다. 가장 일반적인 노드만 표시하려면 비활성화하고 사용 가능한 모든 배리언트에 액세스하려면 활성화합니다.



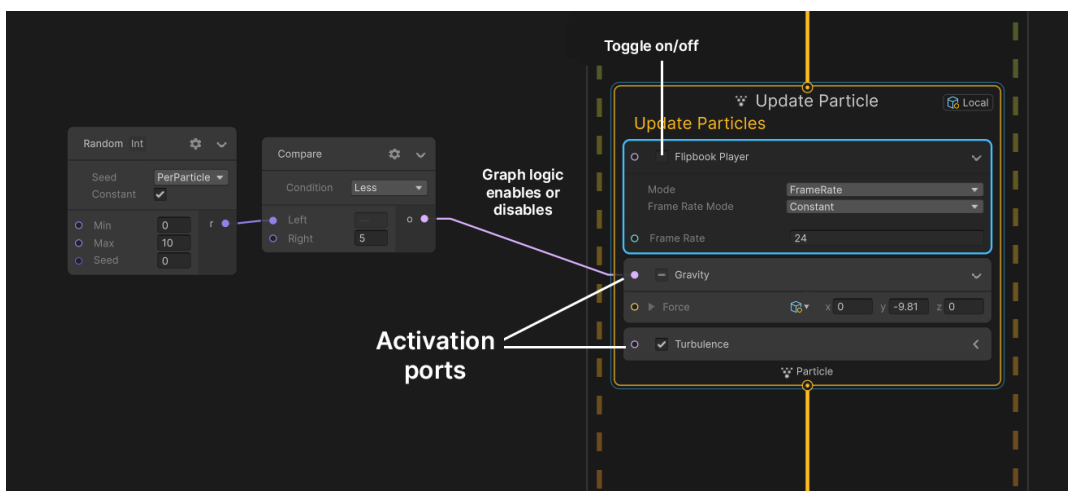
노드 또는 블록을 만들면 사이드 패널과 검색 필터링이 추가됩니다.

활성화 포트

블록에는 이름 옆 왼쪽 상단에 특수 **활성화 포트**가 있으며, 이 포트는 부울 프로퍼티와 연결되어 있습니다. 이 포트를 통해 블록의 활성화 여부를 제어할 수 있습니다.

블록을 수동으로 토글하거나 그래프 로직을 포트에 연결하여 블록이 활성화되는 시점을 제어할 수 있습니다. 이를 통해 동일한 시스템 내에서 파티클마다 다른 동작이나 상태를 구현할 수 있게 됩니다.

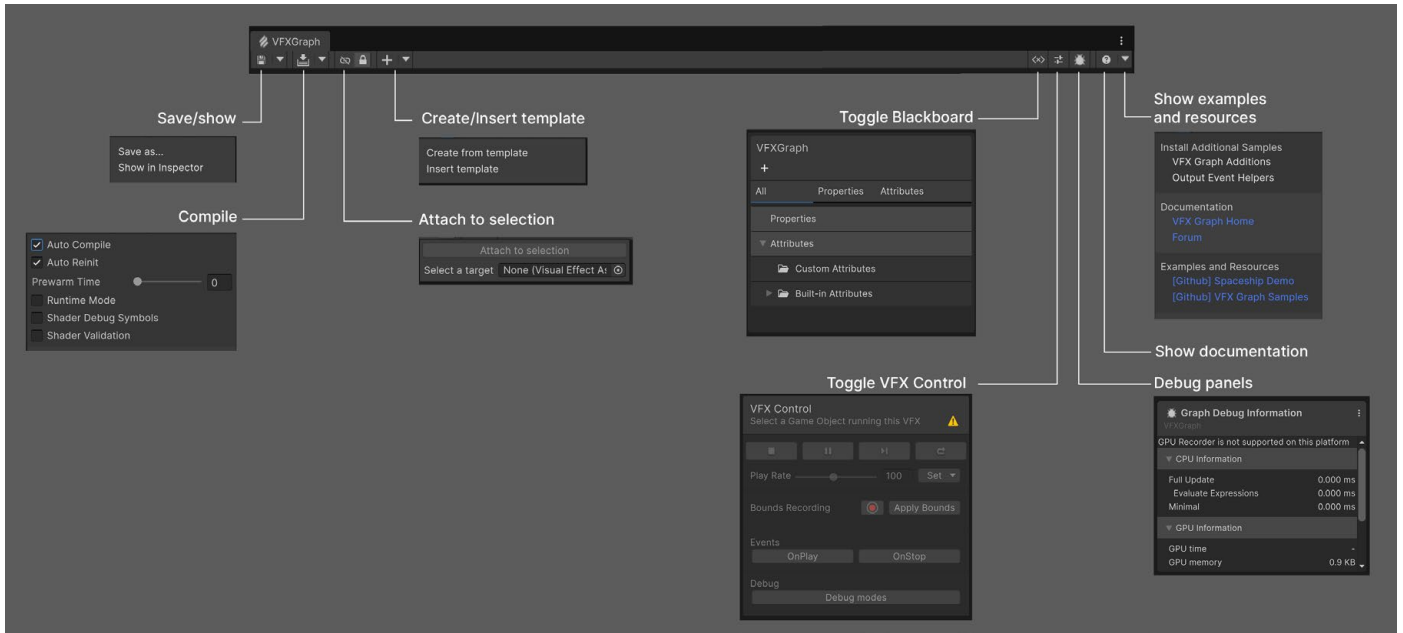
정적으로 비활성화된 블록은 컴파일 중에 회색으로 표시되고 자동으로 제거되므로 런타임 비용이 들지 않습니다.



각 블록에는 활성화 포트가 있습니다.

새로운 VFX 툴바

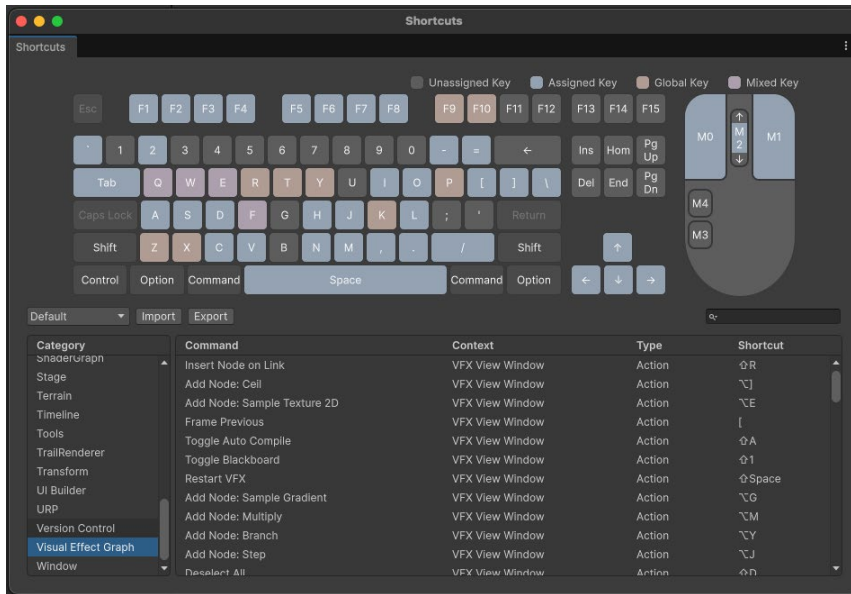
VFX 툴바가 간소화되었으며 이제 기술 자료와 샘플에 빠르게 액세스할 수 있는 새 옵션이 추가되었습니다.



VFX 툴바가 간소화되었습니다.

키보드 단축키

단축키 관리자에 VFX Graph 카테고리가 생겼으므로 Visual Effect Graph 창에서 사용할 수 있는 단축키 커맨드를 수정할 수 있습니다. VFX 아티스트의 워크플로 속도를 높이기 위해 새로운 단축키 커맨드가 추가되었습니다.

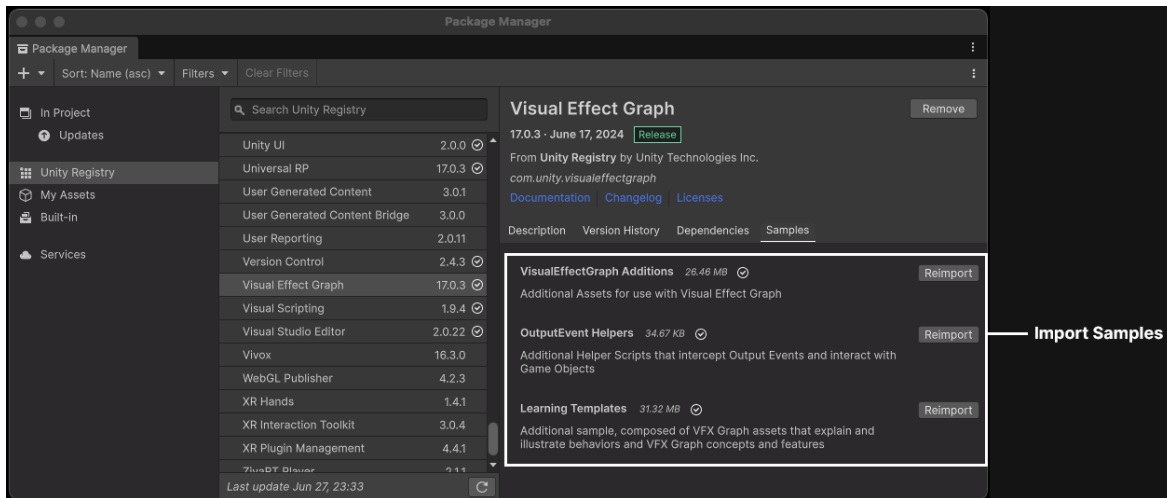


이제 단축키 관리자에 VFX Graph 카테고리가 생겼습니다.

VFX 샘플 콘텐츠 살펴보기

VFX Graph는 그저 여러 요소를 결합하는 툴이 아닙니다. 각 노드와 연산자의 적용 방법을 파악하고 이들이 어떻게 연동하는지 확실하게 이해해야만 제대로 활용할 수 있습니다.

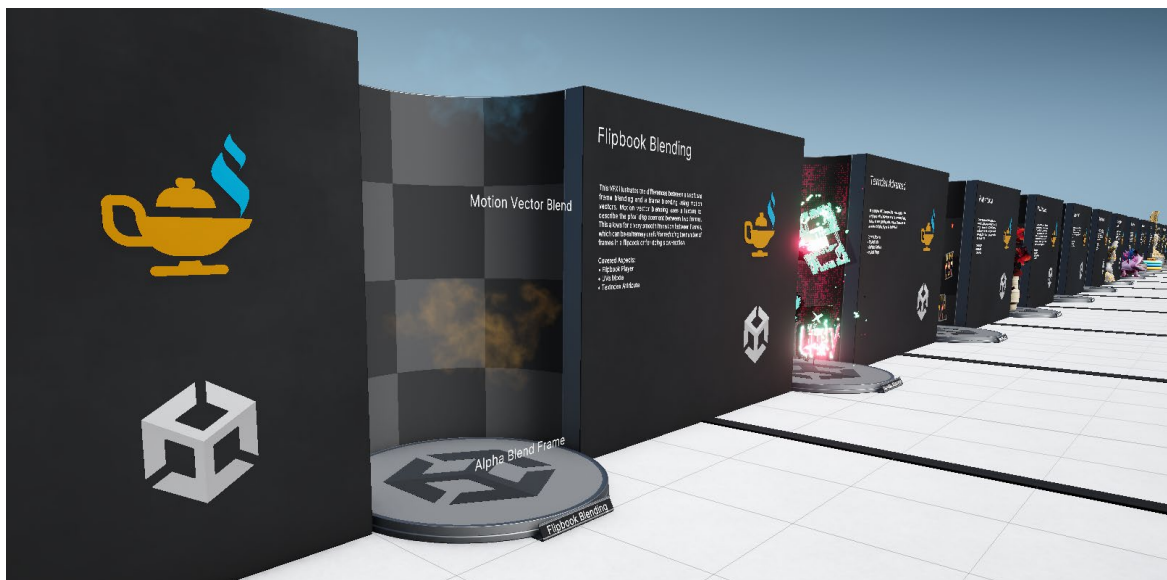
패키지 관리자에서 사용 가능한 두 가지 샘플을 통해 이러한 기능을 컨텍스트에 맞게 확인할 수 있습니다. [VFX Graph Learning Templates](#) 및 [VFX Graph Additions](#)를 살펴보세요.



패키지 관리자에서 샘플 콘텐츠를 설치합니다.

[VFX Graph Learning Templates](#)는 다양한 기술을 선보입니다. 이 교육용 샘플 컬렉션은 VFX Graph의 특정 측면이나 기능 세트를 살펴보는 데 도움이 될 수 있습니다.

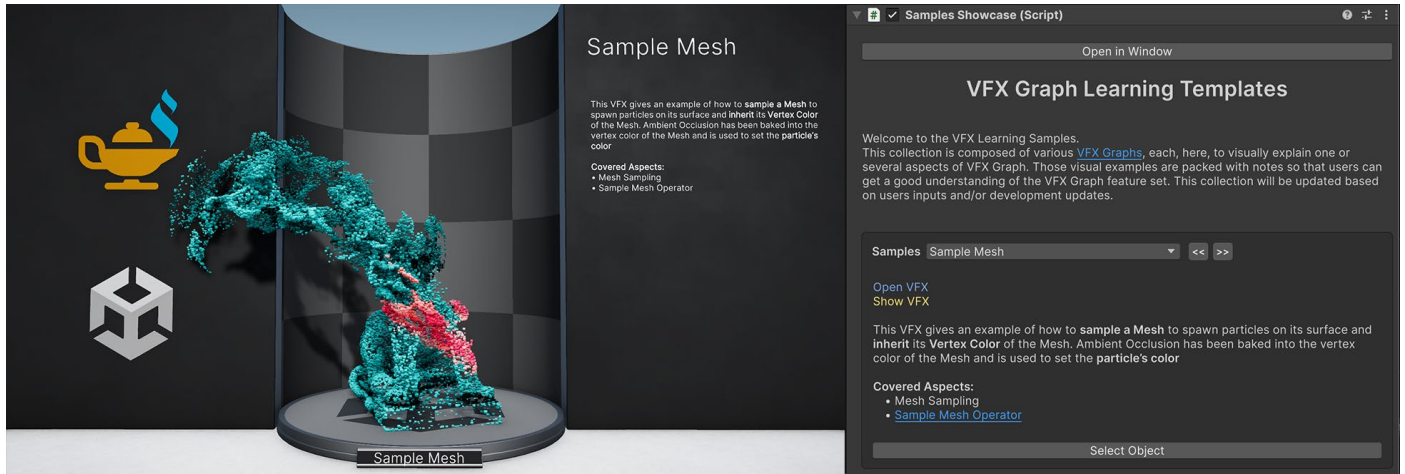
샘플 콘텐츠는 VFX Graph 버전 17.0(Unity 6) 이상에서 URP 및 HDRP 프로젝트와 모두 호환 가능합니다.



VFX Graph Learning Templates를 사용해 보세요.

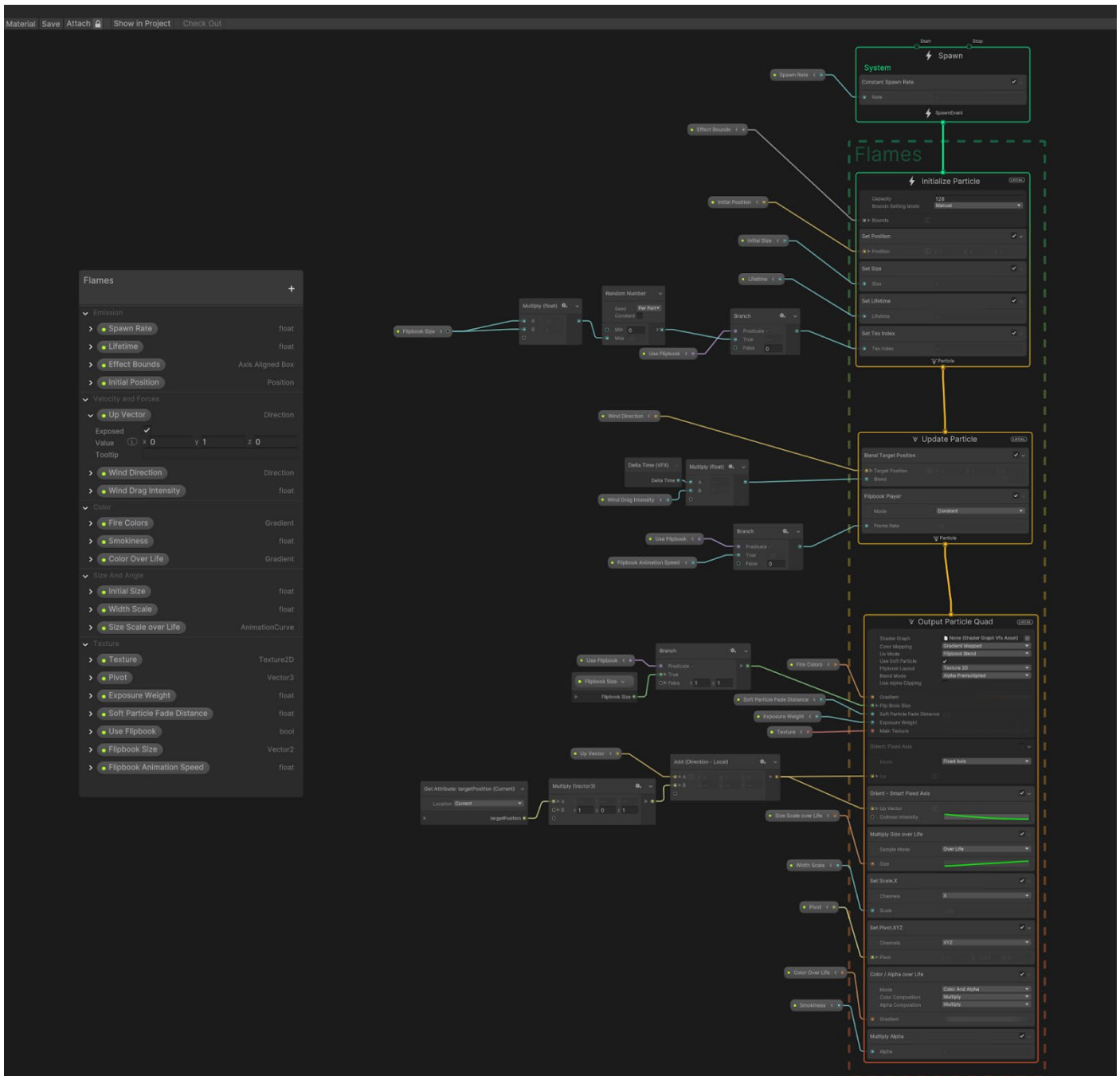
씬 뷰를 사용하여 자유롭게 이동하거나 게임 뷰를 사용하여 각 효과를 집중적으로 확인해 보세요. 인스펙터의 **Sample Showcase** 창에 해당 정보가 표시되며, 링크를 통해 기술 자료에 빠르게 액세스하거나 효과를 전환할 수 있습니다. 각 VFX 에셋에는 참고 사항과 설명이 포함되어 있습니다.

이러한 샘플은 [예시로 보는 시각 효과](#)에서 더 자세히 살펴볼 예정입니다.

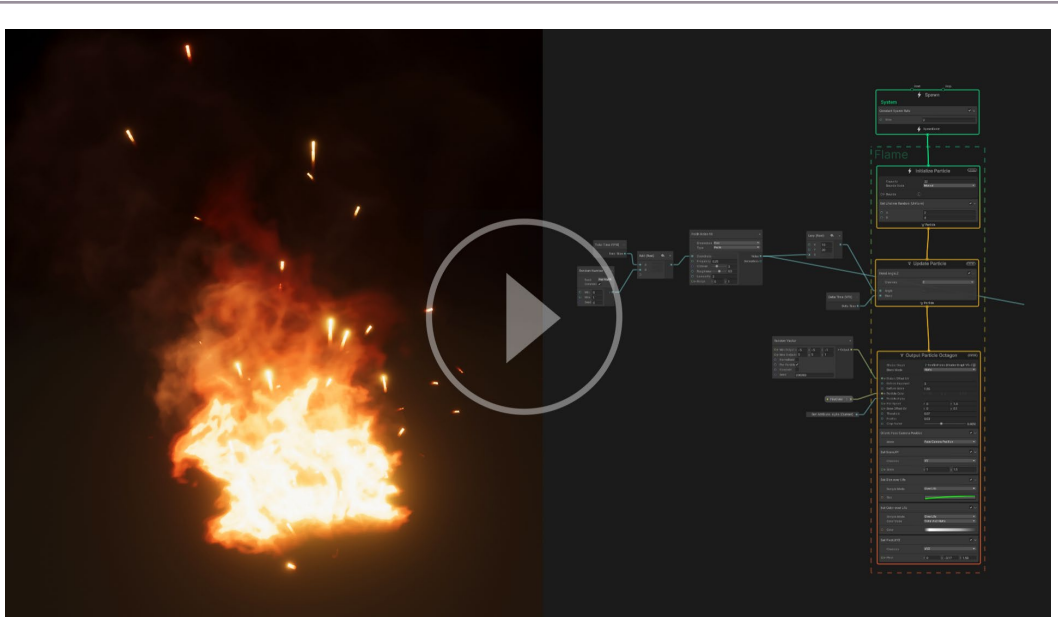


Sample Showcase 창을 통해 각 VFX Graph를 탐색합니다.

패키지 관리자의 [VFX Graph Additions](#)에서 소개하는 몇 개의 간단한 그래프부터 살펴보면 파티클을 어떻게 관리할지 배우는 데 큰 도움이 될 것입니다. 아래 예시에서는 Smoke, Flames, Sparks 그래프를 조합해 Bonfire 효과를 만든 내용을 살펴볼 수 있습니다.



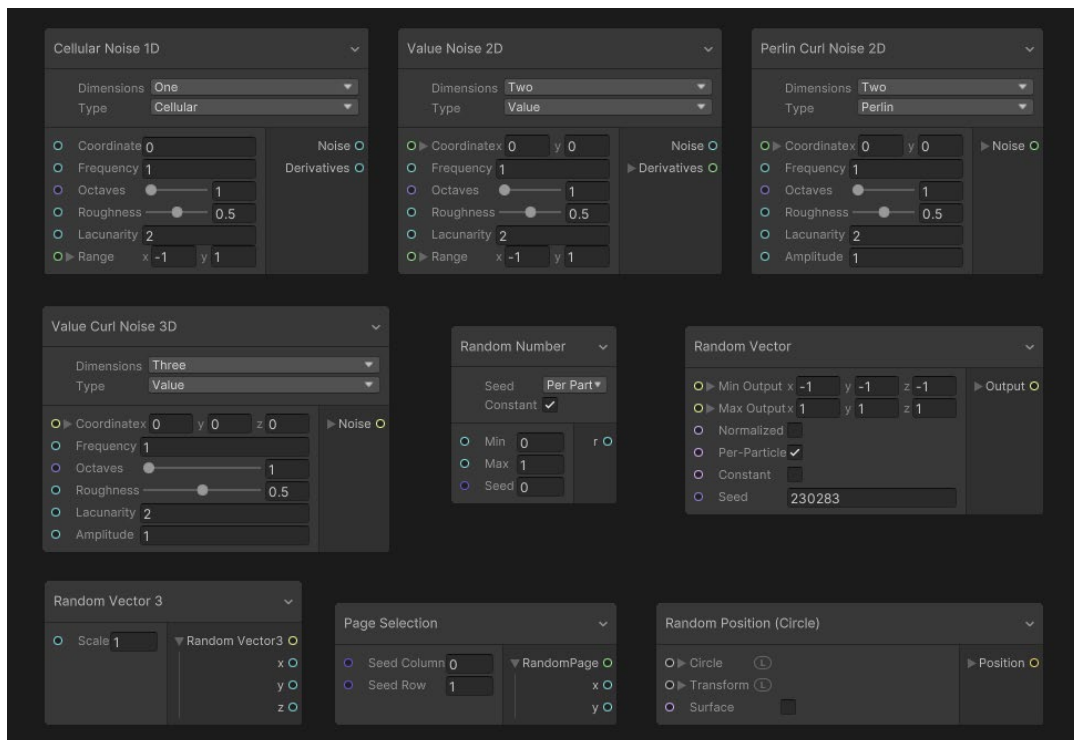
Flames 효과는 기본 그래프를 보여 줍니다.



Bonfire 하위 그래프 동영상

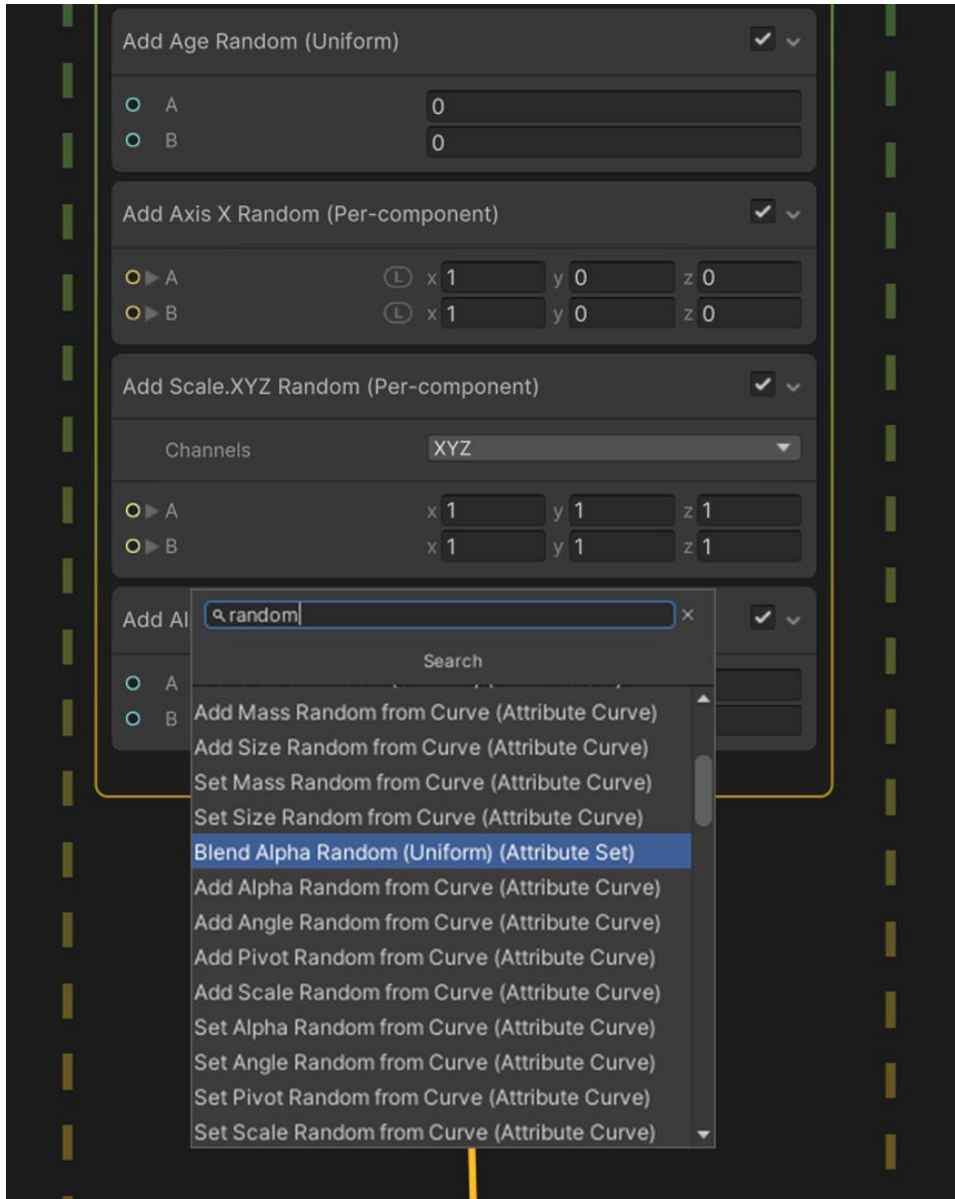
제공된 샘플을 살펴보면 다음과 같은 블록과 연산자가 공통적으로 쓰이는 것을 볼 수 있습니다.

- **노이즈 및 무작위 연산자:** 절차적 노이즈를 사용하면 렌더링된 이미지의 기계 같은 느낌을 줄일 수 있습니다. VFX Graph는 1차원, 2차원, 3차원 **노이즈** 및 **무작위화**에 사용할 수 있는 여러 연산자를 제공합니다.



노이즈 및 무작위 연산자

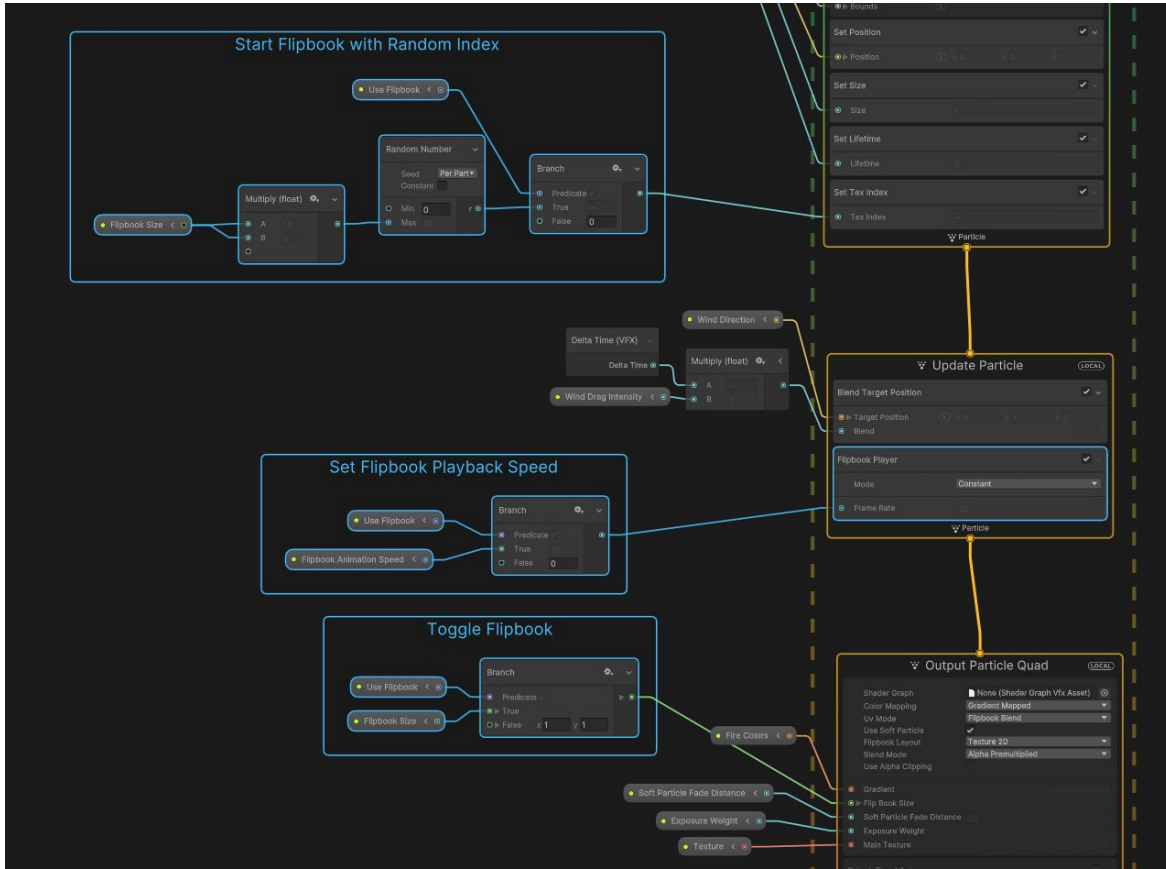
- **속성 블록:** 마찬가지로 다양한 모드에서 무작위화를 적용하는 옵션이 있습니다. 속성마다 조금씩 다를 수 있으므로 여러 가지를 실험해 보며 각 속성의 동작을 숙지하세요.



무작위화 블록

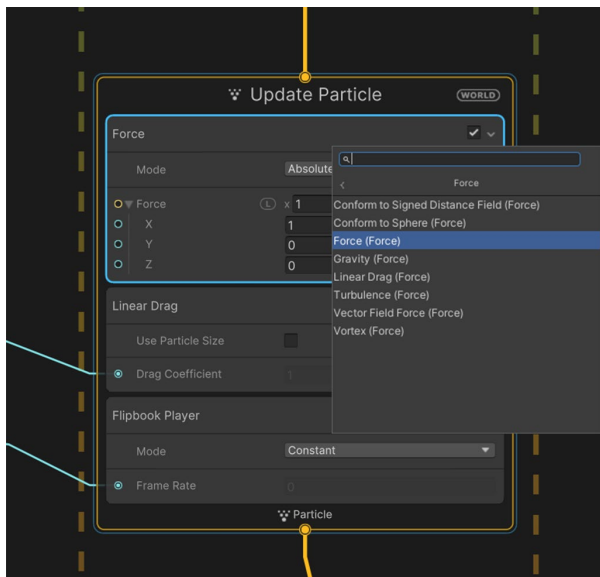
- **플립북:** Smoke 샘플과 Flames 샘플에서 알 수 있듯이 애니메이션 텍스처를 사용하면 사실적인 효과를 더 쉽게 구현할 수 있습니다. 외부 **DCC**(디지털 콘텐츠 제작) 툴이나 Unity 자체 기능으로 애니메이션 텍스처를 생성하세요. 연산자를 사용하여 **플립북 블록**을 관리할 수 있습니다.

Unity에서 플립북을 직접 제작하는 자세한 방법은 [VFXToolbox](#) 섹션의 **Image Sequencer**를 참조하세요.



플립북 노드

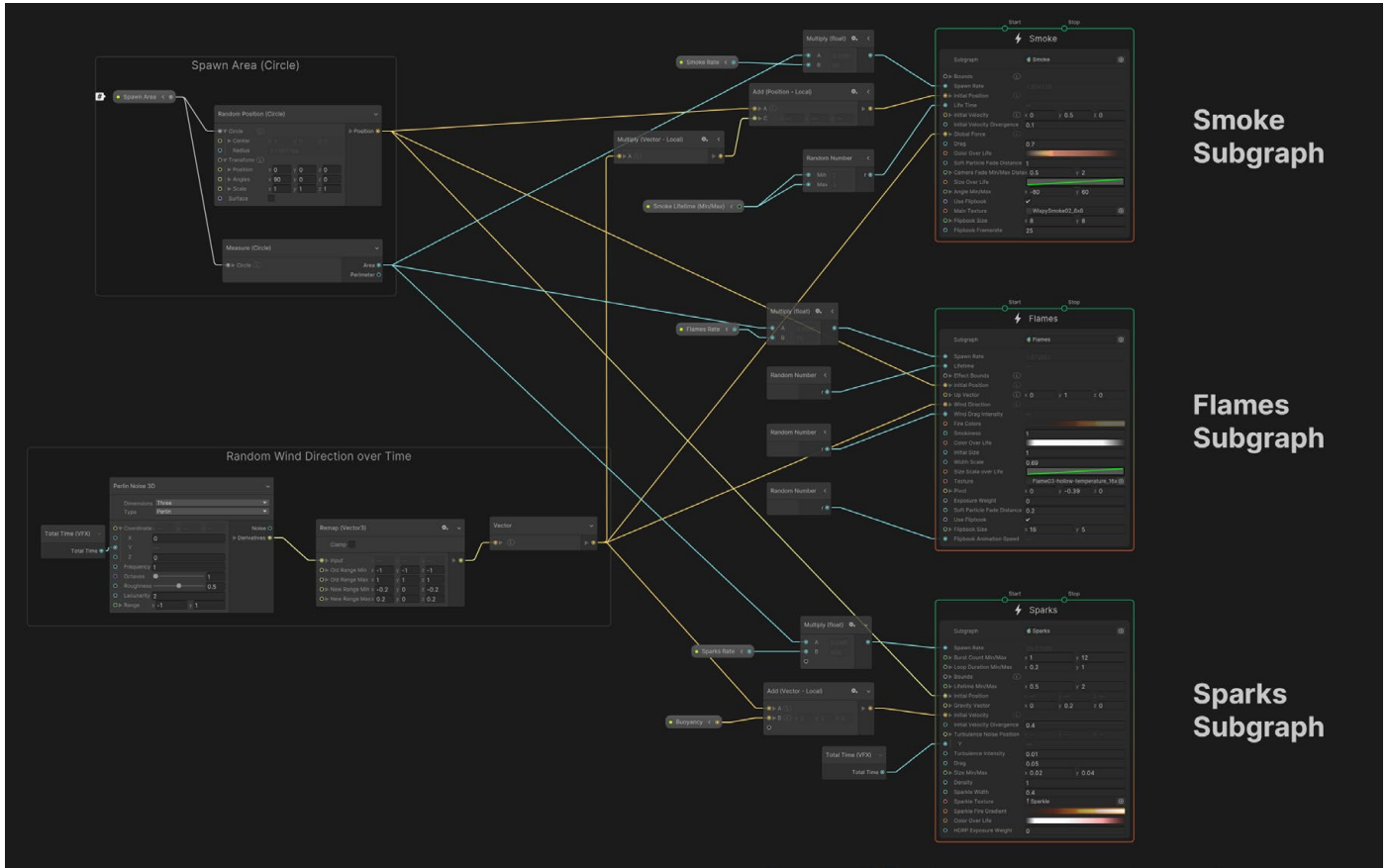
- 물리: **힘**, **충돌**, **항력**은 파티클로 자연 현상을 시뮬레이션할 때 반드시 필요한 요소입니다. 하지만 현실성의 한계를 너무 엄격하게 규정하지는 마세요. 아티스트로서 본인의 판단에 따라 표현하면 됩니다.



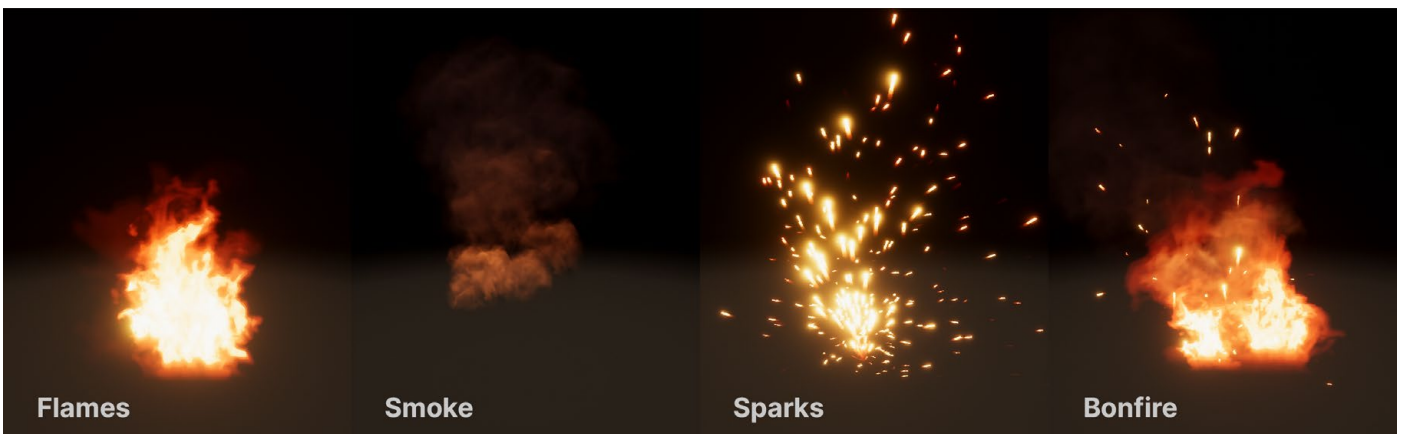
물리 블록

— **하위 그래프:** Smoke, Flames, Sparks는 **하위 그래프**입니다. 나중에 재사용하기 위해 에셋으로 저장된 VFX Graph의 일부입니다.

주요 요소를 작은 부분으로 나누면 Bonfire 그래프를 더 쉽게 파악할 수 있습니다. 예를 들어 애플리케이션의 다른 위치에서 폭발 효과를 새로 만들어야 할 경우, 하위 그래프를 다른 그래프에 드래그 앤 드롭하여 배포하면 됩니다. 하위 그래프가 에셋이라서 가능한 작업입니다.



세 개의 하위 그래프를 사용하는 Bonfire 그래프



개별 하위 그래프 요소가 결합하여 Bonfire 효과를 만듭니다.

다른 효과 중에서도 Bonfire 그래프를 구성한 방법에 대한 요약 설명을 보고 싶다면, 샘플과 Unity 스페이스쉽(Spaceship) 데모 제작에 참여한 시니어 VFX 및 테크니컬 아티스트 토마스 이췌의 [이 커뮤니티 동영상](#)을 참조하세요.

더 많은 자료

VFX Graph가 기본적으로 어떻게 작동하는지 파악했다면 몇 가지 효과를 처음부터 만들어 보세요. [흘날리는 눈](#)처럼 간단한 시스템부터 만들어 본 다음, [불](#), [연기](#), [안개](#) 같은 다양한 효과를 만들어 보세요.

다음 동영상은 몇 가지 효과를 소개합니다. 다만 이전 버전의 VFX Graph를 사용하고 있다는 점에 유의하세요.

- [VFX Graph로 눈 만들기](#)
- [Unity에서 VFX Graph로 불, 연기, 안개 효과 만들기](#)
- [Unity에서 VFX Graph로 파티클 렌더링하기](#)
- [Unity의 VFX Graph로 만드는 멀티레이어 효과](#)
- [Hardspace: Shipbreaker 기술 대담: VFX Graph를 사용한 폭발 효과](#)
- [더 헤러틱\(The Heretic\)의 실시간 VFX 워크플로](#)

추가 참조 자료

VFX Graph에 더 익숙해지면 그래프 간의 미묘한 차이를 더 구체적으로 살펴볼 수 있습니다. 특정 노드나 연산자 기능을 참조해야 할 때 항상 다음과 같은 페이지를 활용하세요.

- VFX Graph의 모든 컨텍스트, 블록, 연산자를 설명하는 [노드 라이브러리](#)
- 모든 공통 속성에 대한 포괄적인 목록을 제공하는 [표준 속성 참조 자료](#)
- VFX Graph에서 사용되는 데이터 유형을 나열하는 [VFX 유형 참조 자료](#)

예시로 보는 시각 효과

VFX Graph의 기본을 이해했다면 이제 더 복잡한 그래프를 만들어 볼 차례입니다. 제작을 진행하며 직면할 수 있는 문제에 더 효과적으로 대응할 수 있도록 다양한 예시 프로젝트를 준비했습니다.

예시 샘플을 통해 시각 요소에서 구현하게 될 전반적인 사항을 실행해 볼 수 있습니다. 은은한 연기와 불꽃 효과부터, 완전히 스크립팅된 AAA급 시네마틱 게임플레이까지 하나하나 살펴보시기 바랍니다.

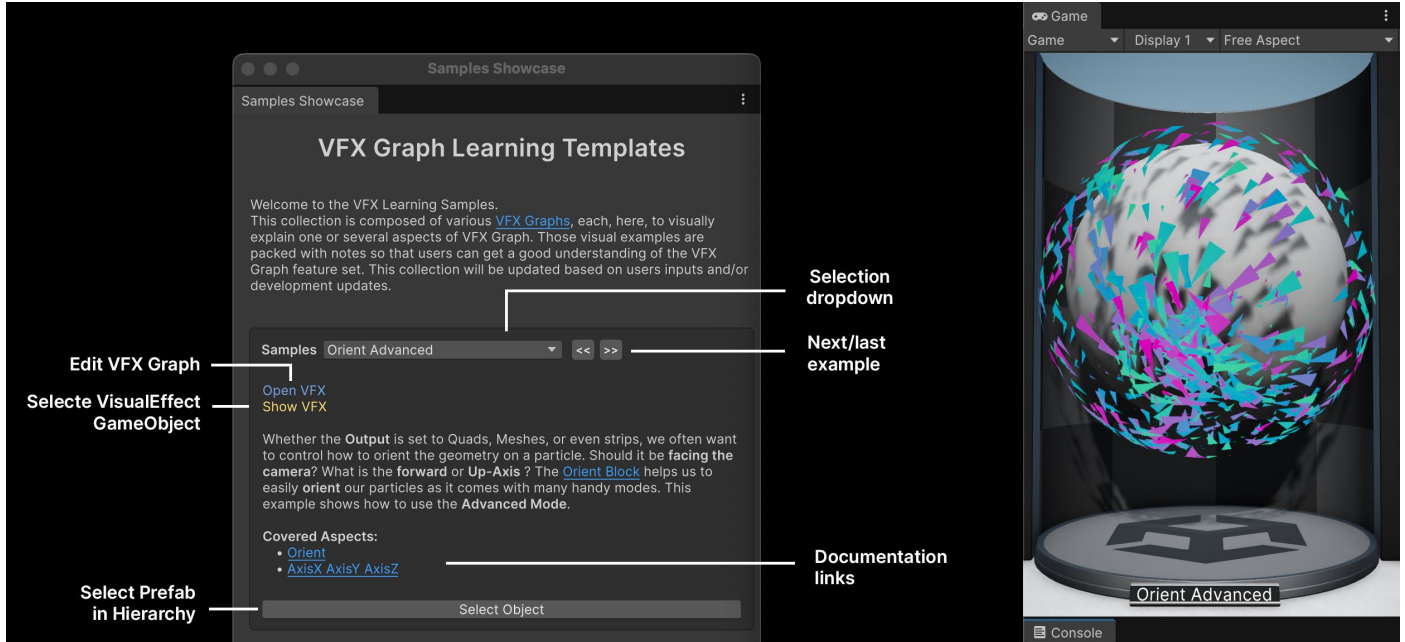
VFX Graph Learning Templates

VFX Graph에 아직 익숙하지 않다면 이 VFX Graph 컬렉션을 먼저 살펴보세요. Learning Templates는 다양한 실시간 효과로 구성되어 있으며, 각 효과를 통해 VFX Graph의 여러 기능을 알아볼 수 있습니다.

그래프는 간결하면서도 중요한 내용을 집중적으로 보여 주므로 학습하기에 좋습니다. 각 템플릿을 자세히 살펴보고 새로운 기술을 익히거나 자체 효과를 제작하기 위한 시작점으로 활용해 보세요. 각 그래프에는 그래프 구성을 이해하는 데 도움이 되는 자세한 참고 내용이 함께 제공됩니다.

Learning Templates는 새 VFX Graph를 만들 때 마법사를 통해 사용하거나, 패키지 관리자를 통해 URP 또는 HDRP의 전체 샘플 씬으로 импорт할 수 있습니다.

VFX Samples Showcase 창을 통해 샘플을 둘러볼 수 있습니다. Learning Templates를 간단히 살펴보겠습니다.

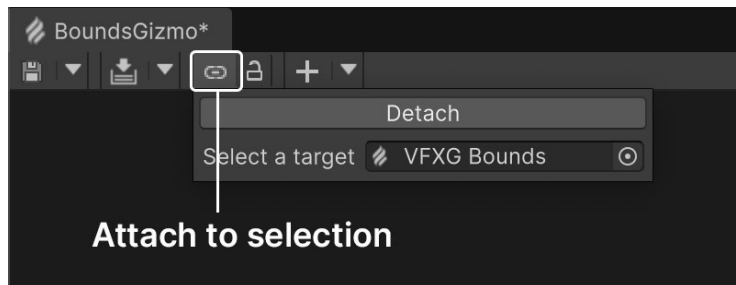


Samples Showcase 창

기즈모 시각화

많은 블록에는 씬 뷰의 시각적 기즈모에 해당하는 조정 가능한 파라미터가 있습니다. 블록 내에서 직접 값을 변경하거나 씬 뷰에서 기즈모를 조작할 수 있습니다.

기즈모를 조작하려면 편집 중인 VFX Graph를 씬의 해당 VFX 인스턴스에 연결해야 합니다.

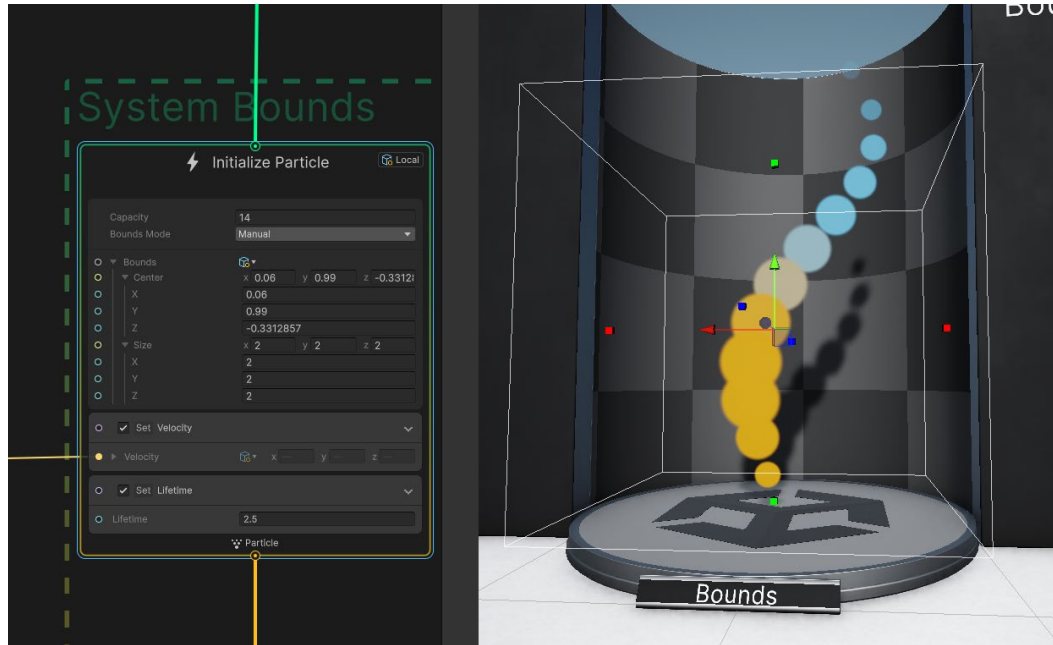


VisualEffect 컴포넌트가 있는 게임 오브젝트에 VFX Graph를 연결합니다.

창 상단의 작은 '링크' 아이콘을 사용하여 Visual Effect 컴포넌트가 있는 선택된 게임 오브젝트에 VFX Graph를 연결합니다.

연결이 완료되면 VFX Graph 에디터에서 해당 블록을 선택할 경우 동기화되어 표시됩니다. 기즈모 핸들을 통해 상호 작용하며 설정을 변경할 수 있습니다.

예를 들어 아래에는 Initialize Particle 컨텍스트의 Bounds 설정과 씬 뷰의 해당 기즈모가 나타나 있습니다.



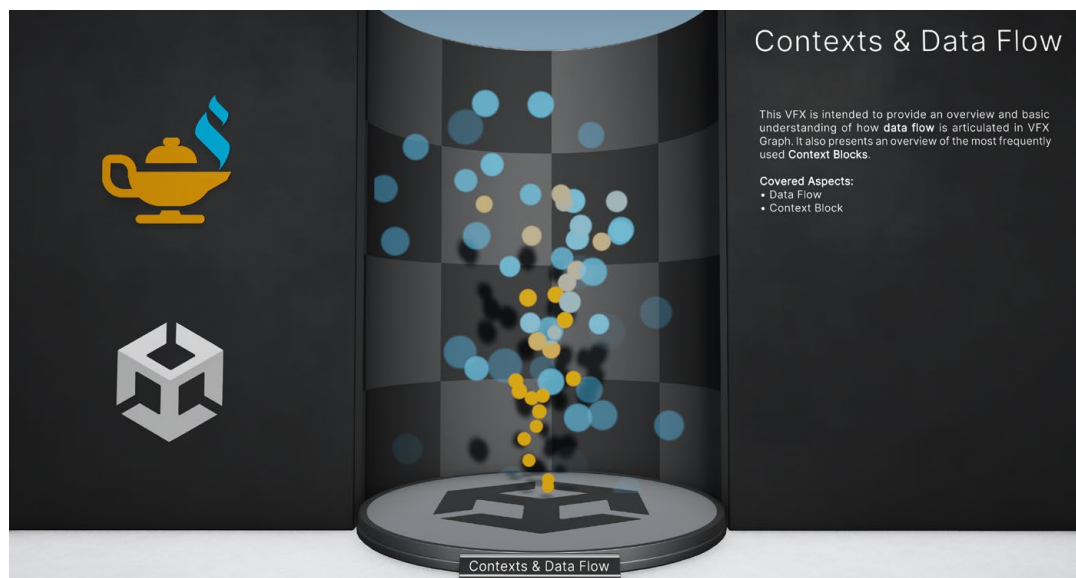
Bounds 기즈모는 Initialize Particle 컨텍스트의 Bounds 설정을 나타냅니다.

그래프 기초

이 샘플은 그래프 로직의 기본 사항과 VFX Graph를 만들고 최적화하는 방법을 보여 줍니다.

Contexts & Data Flow

이 샘플은 VFX Graph에서 데이터 플로우가 처리되는 방식에 대한 개요를 제공하며 기본적인 이해를 돕습니다. 가장 자주 사용되는 컨텍스트와 블록에 대한 개요도 제공합니다.



Contexts & Data Flow 샘플

Spawn Context

이 VFX Graph에서는 Spawn 컨텍스트의 옵션을 사용하여 파티클이 처음 나타나는 방식을 제어하는 방법을 확인할 수 있습니다. Spawn State를 활용하면 루프 인덱스, 생성 횟수, 루프 상태, 루프 지속 시간 같은 중요한 정보를 익힐 수 있습니다.

예시 효과는 텍스처 인덱스를 조작하여 숫자 스택이 하나씩 올라가는 것처럼 보입니다. 이는 각 버스트 사이에 약간의 지연이 있는 단일 파티클 버스트를 생성하여 구현됩니다.



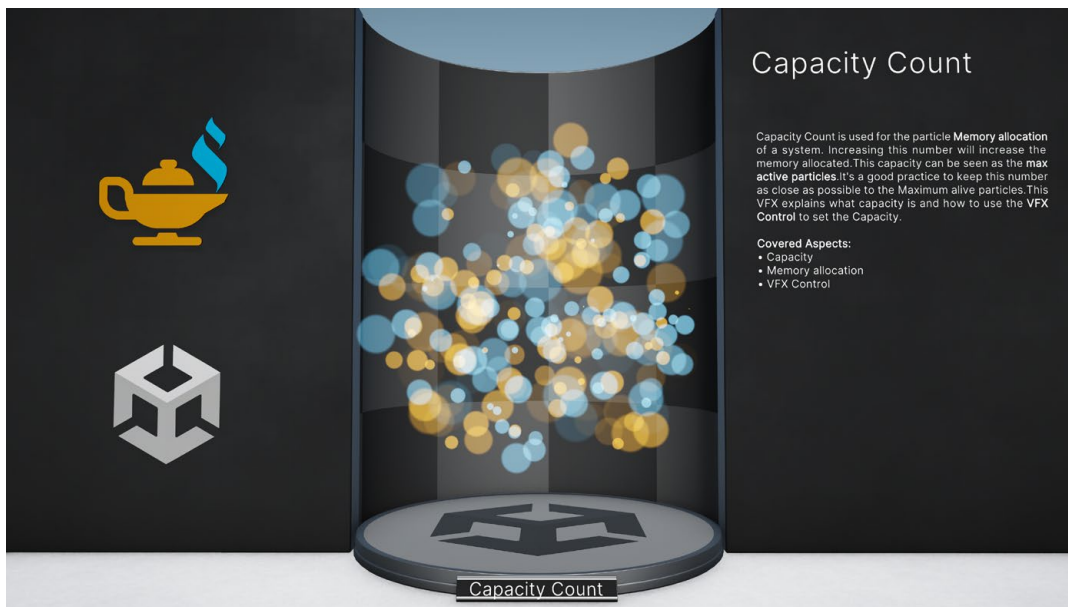
Spawn Context를 사용합니다.

또한 이 예제에서는 Spawn 컨텍스트에서 수명, 컬러, 알파 등의 속성을 설정하여 나중에 Initialize Particle 컨텍스트에 상속되는 방법을 보여 줍니다.

Capacity Count

이 VFX Graph에서는 볼륨 내에서 임의의 크기의 파티클을 생성하여 Capacity 속성을 사용하는 방법을 살펴볼 수 있습니다.

Capacity Count는 시스템의 파티클 메모리 할당에 사용됩니다. 이 수치를 높이면 할당되는 메모리가 증가합니다. Capacity는 현재 활성 파티클 수에 대한 최댓값으로 사용됩니다.

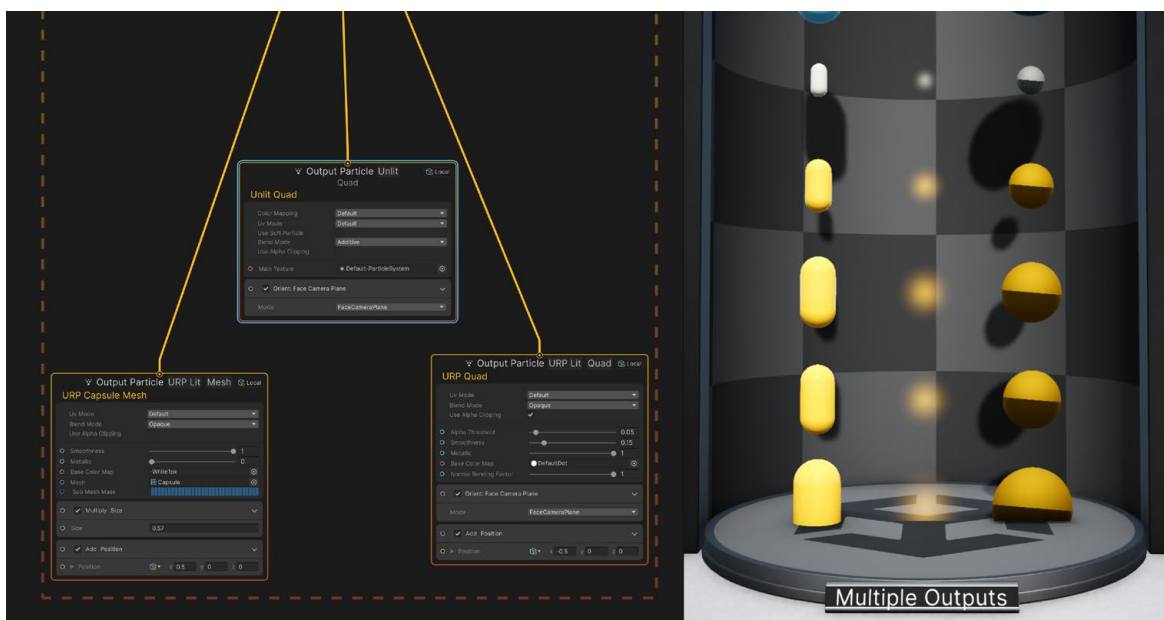


Capacity Count를 늘리면 할당된 메모리가 증가합니다.

Multiple Outputs

이 VFX Graph에서는 Initialize 또는 Update 컨텍스트를 하나 또는 여러 출력 컨텍스트에 연결하는 방법을 살펴볼 수 있습니다. 이를 통해 하나의 파티클 시뮬레이션을 생성하고 각 파티클에 여러 유형의 렌더러를 생성할 수 있습니다.

이 데모에서는 각 파티클이 언릿 파티클, 메시, 사각형으로 렌더링됩니다. Mesh 출력을 사용하여 미사일 코어를 만든 다음 두 번째 Quad 출력을 사용하여 그 위에 이미시브 글로우를 추가한다고 생각해 보세요.



다양한 효과를 위해 여러 개의 출력을 사용합니다.

일부 출력은 특정 SRP만 호환됩니다(예: HDRP 볼류메트릭 안개 출력은 HDRP만 호환).

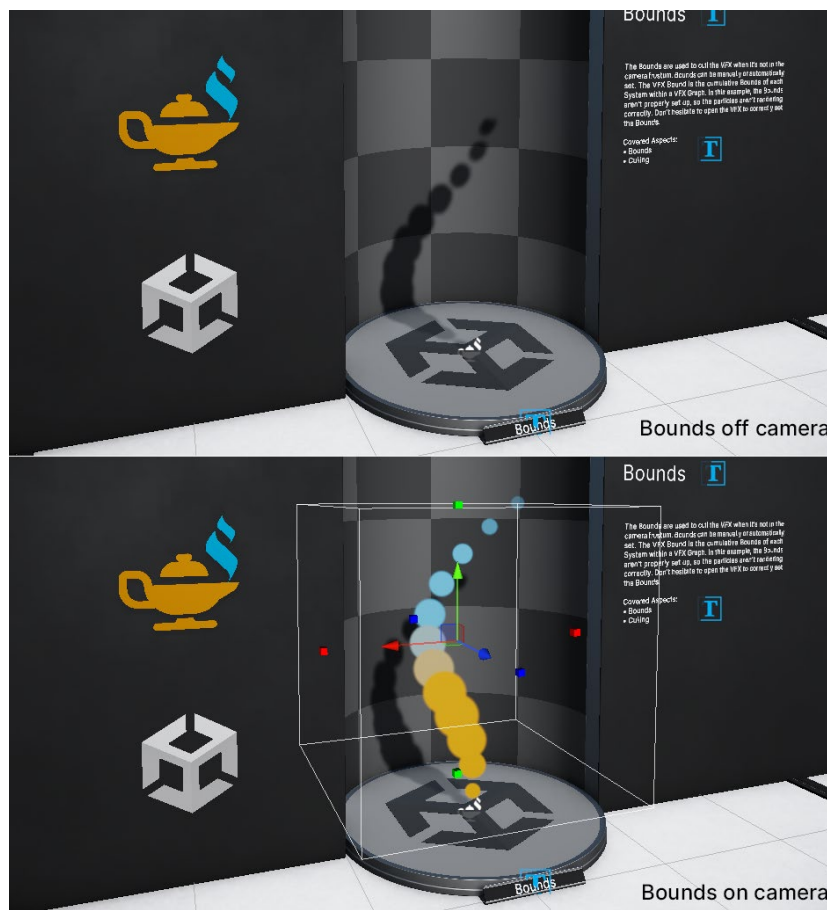
Bounds

Bounds 설정으로 효과를 컬링하여 성능을 향상시키는 방법을 보여 줍니다. 이 예시에서는 Bounds가 기본적으로 카메라 절두체 외부에 의도적으로 설정되어 있습니다. 그 결과 파티클이 컬링됩니다(이 예시에서는 Output 컨텍스트에 활성화된 경우 여전히 그림자를 드리울 수 있음).

Bounds 기즈모가 카메라 절두체 안에 있고 파티클이 다시 나타나도록 Bounds 설정을 조정합니다.

Bounds Mode를 선택합니다.

- **Manual:** Bound Size 및 Center를 수동으로 설정합니다.
- **Recorded:** 이를 통해 사용자는 VFX 컨트롤을 사용하여 경계를 녹화한 다음 해당 설정을 시스템에 적용할 수 있습니다.
- **Automatic:** 경계가 프레임마다 계산됩니다. 이 옵션은 동적 VFX 또는 VFX 반복 작업(iteration)에서 필요할 수 있지만 리소스를 많이 사용합니다. 가능하면 Manual 또는 Recorded를 사용하세요.



Bounds 설정을 조정하여 파티클이 다시 나타나도록 합니다.

파티클 방향 및 회전

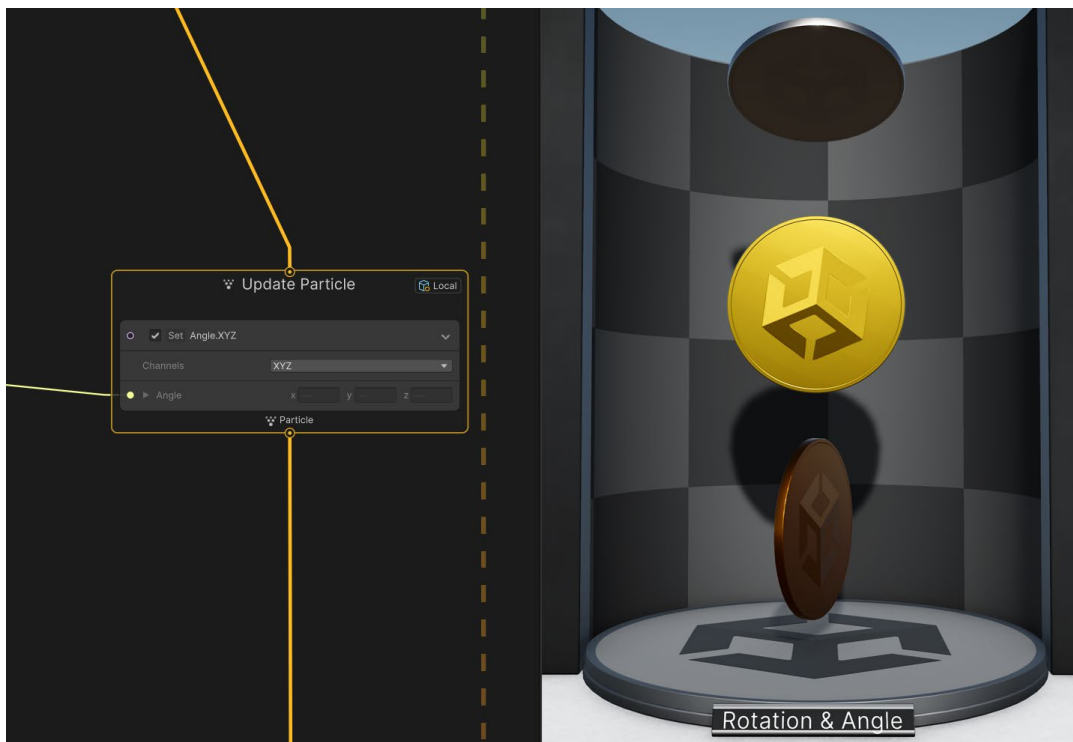
출력이 사각형, 메시, 스트립 중 무엇으로 설정되어 있는 파티클의 지오메트리 방향을 제어해야 하는 경우가 많이 있습니다. 카메라를 향해야 할까요? 앞쪽 또는 위쪽 측은 무엇인가요? Orient 블록을 활용하면 다양한 모드를 통해 파티클의 방향을 쉽게 설정할 수 있습니다.

Orient Face Camera, Orient Fixed Axis, Orient Advanced VFX Graph는 파티클 방향을 사용하는 다양한 방법을 보여 줍니다.



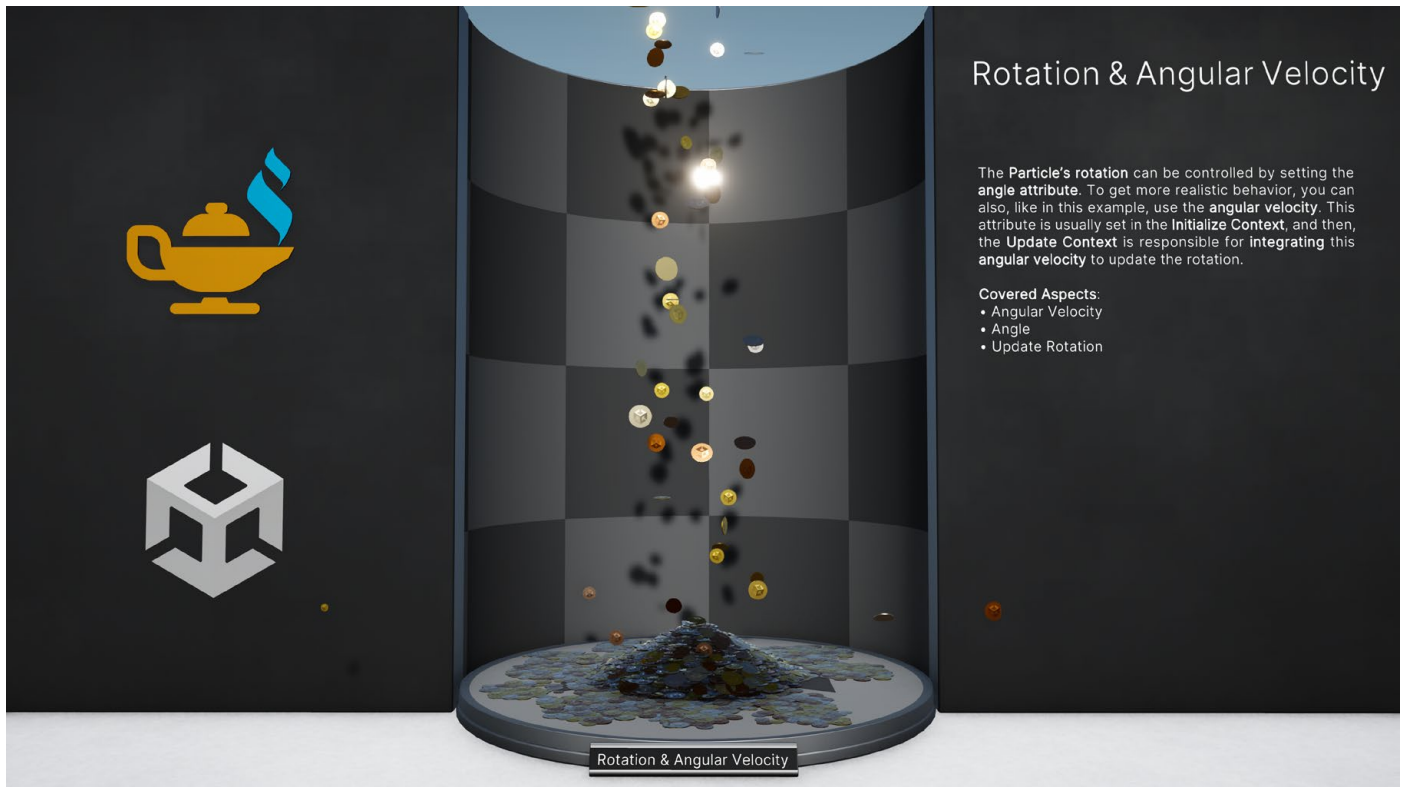
이 샘플에서는 파티클의 방향을 지정하는 다양한 방법을 확인할 수 있습니다.

Rotation & Angle VFX Graph에서 볼 수 있듯이 Angle 속성을 사용하여 파티클을 회전할 수도 있습니다.



Angle 속성을 사용하여 파티클을 회전합니다.

더 사실적인 결과를 얻으려면 각속도를 사용하세요. **Rotation & Angular Velocity** 샘플은 Initialize 컨텍스트에서 이를 설정한 다음 Update 컨텍스트를 사용하여 회전을 업데이트하는 방법을 보여 줍니다.



보다 사실적인 모션을 얻으려면 각속도를 사용하세요.

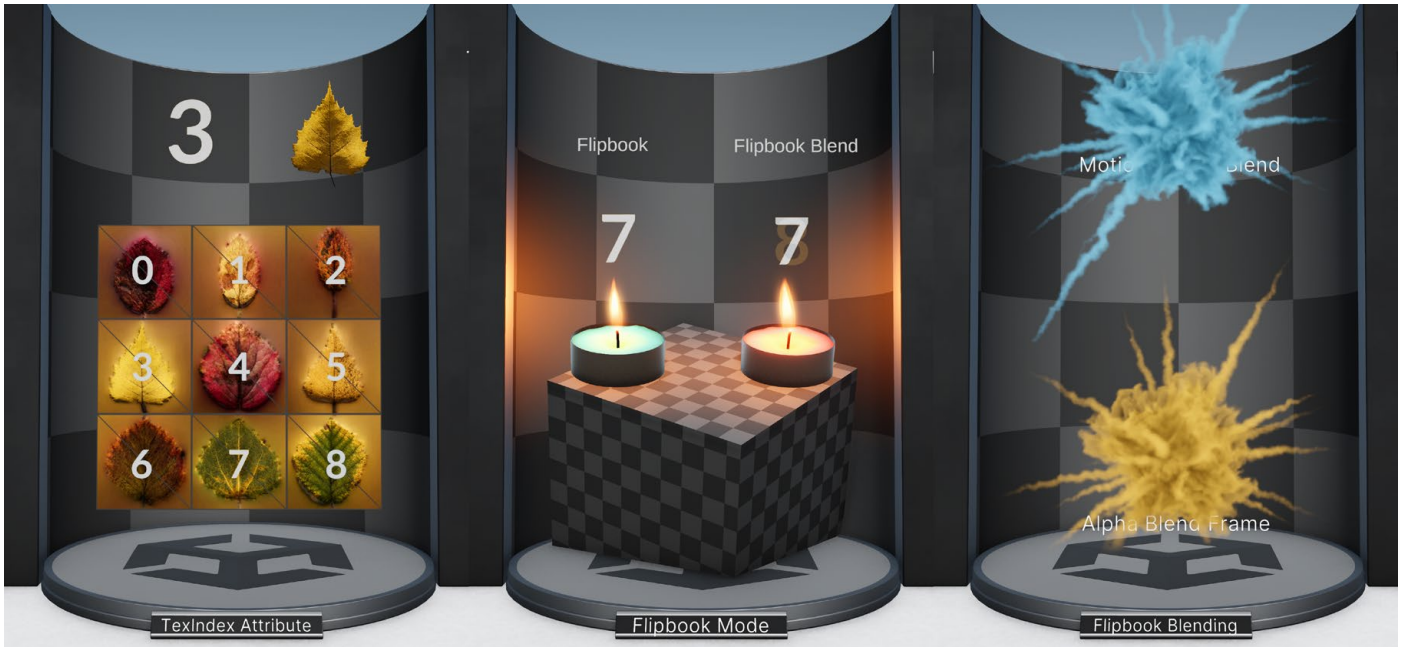
텍스처링과 플립북

실시간 VFX는 스프라이트 시트에 의존하는 경우가 많습니다. **TexIndex Attribute** 샘플은 출력의 UV 모드를 Flipbook으로 설정하여 스프라이트 시트를 사용하는 방법을 보여 줍니다.

Flipbook 블록은 스프라이트 시트를 애니메이션화하는 데 도움이 되며, 스프라이트 시트의 어느 부분을 표시할지 결정하는 플롯인 **texIndex** 속성을 조작하여 수동으로 제어할 수 있습니다.

Flipbook Mode VFX Graph에서는 **Flipbook Blend Frames**를 활성화하여 기본 애니메이션 플립북 텍스처보다 더 매끄럽고 보간된 애니메이션이 생성되는 것을 확인할 수 있습니다.

Flipbook Blending VFX Graph에서는 기존 프레임 블렌딩과 모션 벡터를 사용한 프레임 블렌딩의 차이점을 나타냅니다. 모션 벡터 블렌딩은 텍스처를 사용하여 프레임 사이의 픽셀 변위를 나타내므로 플립북의 프레임 수를 줄이거나 슬로우 모션으로 효과를 표시하는 데 유용할 수 있습니다.



이 샘플에서는 플립북 텍스처의 사용법을 확인할 수 있습니다.

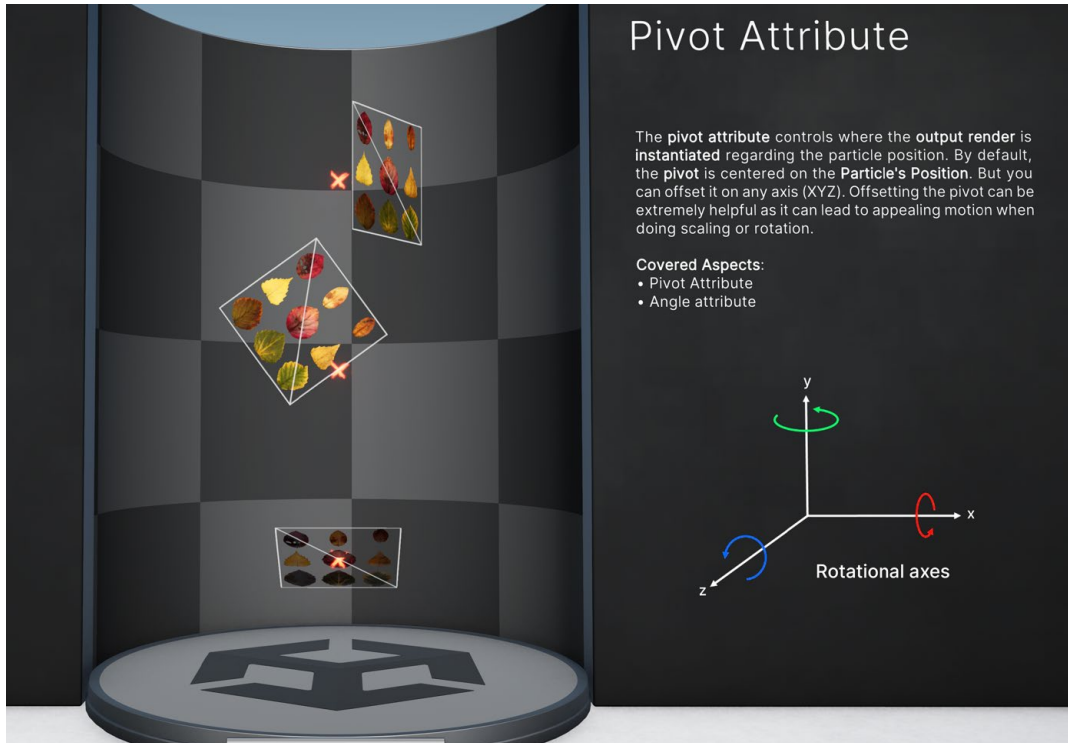
TexIndex Advanced VFX Graph는 **texIndex** 속성을 창의적으로 사용하는 여러 시스템으로 구성됩니다. 시간, 노이즈, 심지어 파티클 위치도 속성 값을 구동하는 데 사용됩니다. 이 설정은 여러 파티클을 함께 애니메이션화하여 멀티레이어 모션 그래픽 효과를 만들어 냅니다.



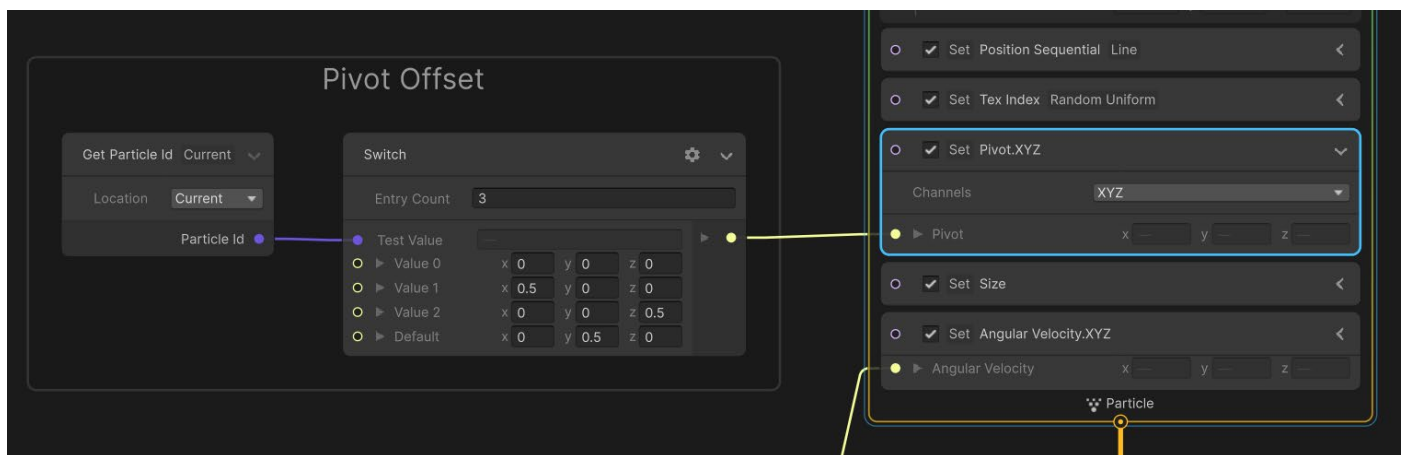
모션 그래픽 효과에 여러 시스템을 결합합니다.

파티클 피벗

기본적으로 피벗은 파티클의 위치를 중심으로 하지만 모든 축(XYZ)에서 오프셋할 수 있습니다. **Pivot Attribute** VFX Graph에서 각 파티클은 서로 다른 피벗 오프셋과 각속도를 갖습니다.



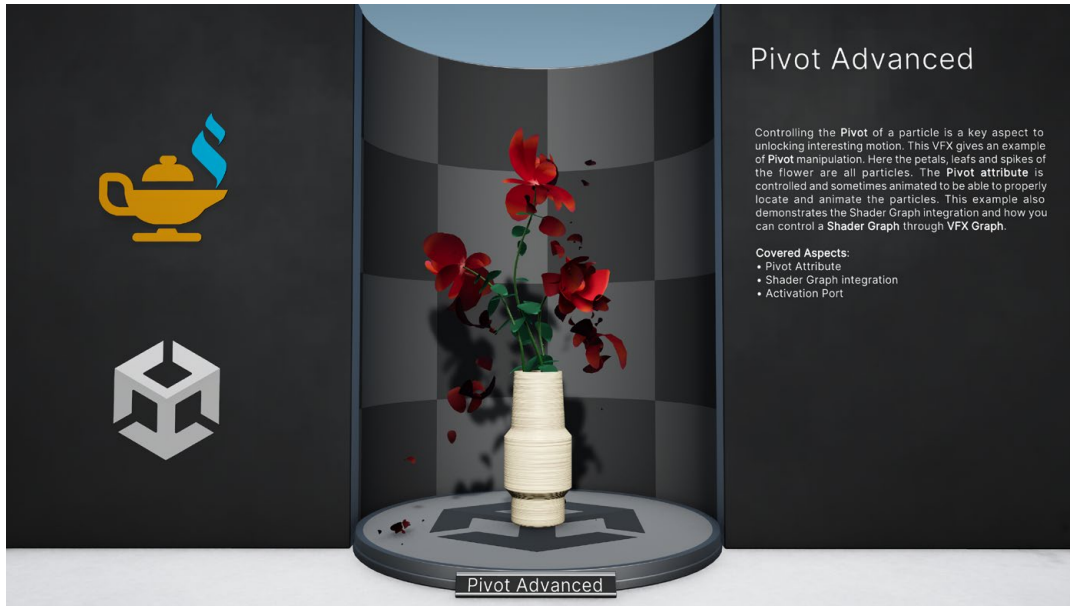
Pivot Attribute VFX Graph에는 서로 다른 피벗을 가진 세 개의 파티클이 표시됩니다.



파티클 피벗을 조정합니다.

파티클의 **Pivot** 속성을 제어하면 흥미로운 모션을 구현할 수 있습니다. **Pivot Advanced** VFX Graph는 피벗 조작의 예시를 보여 줍니다. 여기서는 꽃의 꽃잎, 잎사귀, 가시가 모두 파티클입니다.

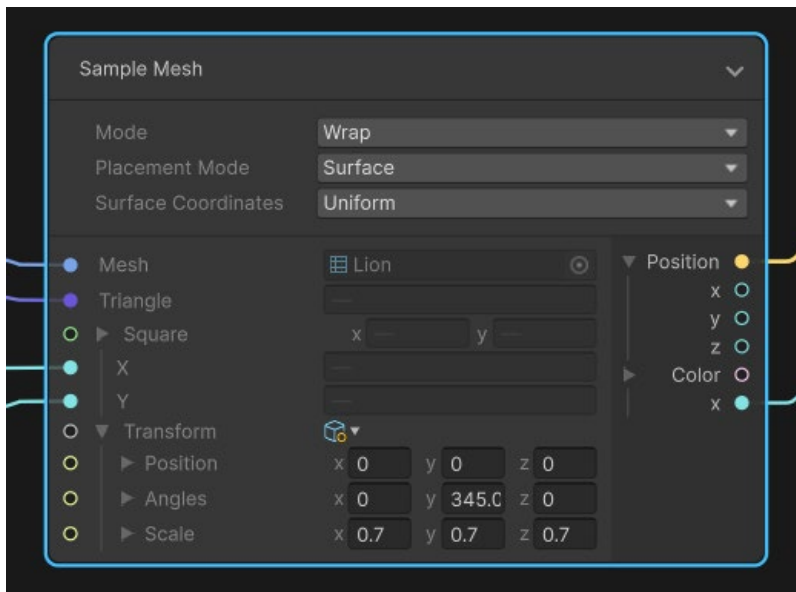
처음에 꽃잎의 피벗은 꽃잎이 붙어 있는 동안 적절하게 구부러질 수 있도록 뿌리에 설정되어 있습니다. 그리고 파티클의 수명이 진행되면서 피벗이 중앙으로 이동하도록 애니메이션화하여 꽃잎이 자연스럽게 떨어지고 떠내려갈 수 있게 됩니다.



피벗을 애니메이션화하면 흥미로운 움직임이 생성됩니다.

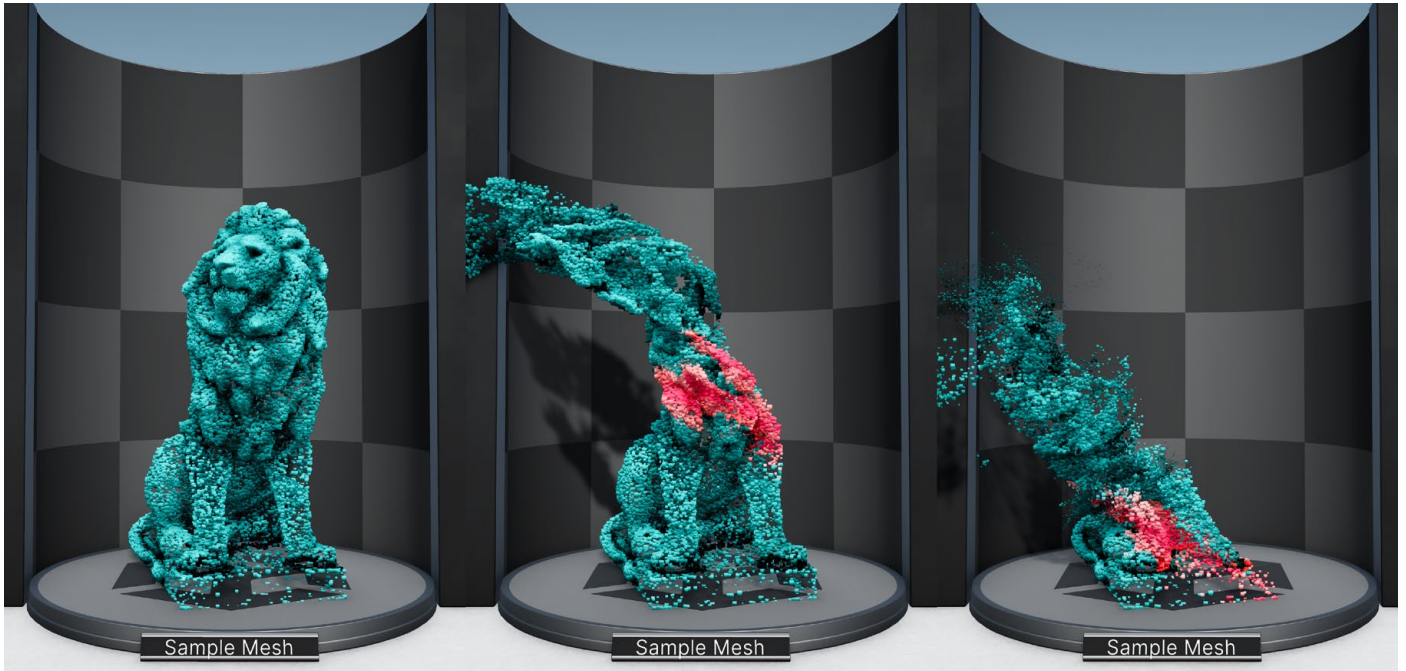
메시 및 텍스처 샘플링

Mesh Sample VFX Graph는 파티클을 생성하기 위해 메시를 샘플링하는 방법을 보여 줍니다. 그래프는 **Sample Mesh** 연산자를 사용하며, **Mesh** 입력은 적절한 메시를 참조하고 **Placement Mode**는 **Surface**로 설정되어 있습니다.



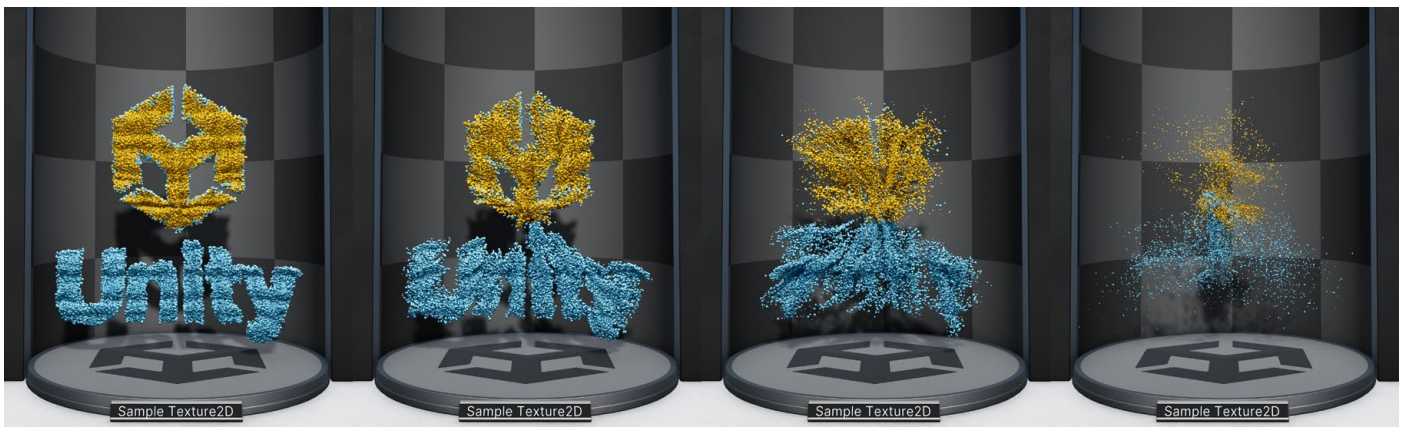
Sample Mesh 노드를 사용하여 표면에 파티클을 배치합니다.

이렇게 하면 메시 표면에 파티클이 무작위로 생성됩니다. **Position** 및 **Color** 출력은 향후 애니메이션 작업을 위한 파티클 프로퍼티를 저장하는 Initialize Particle 컨텍스트로 전달됩니다. 이 예시에서는 외부에서 불어오는 바람과 난기류로 인해 사자상이 먼지 구름으로 분해됩니다.



메시 표면에 파티클을 생성합니다.

2D 이미지를 사용하여 비슷한 효과를 얻을 수 있습니다. **Sample Texture 2D** VFX Graph에서는 Texture2D Sample 연산자를 사용하여 파티클의 컬러를 결정하고 거부 샘플링을 수행하는 방법을 확인할 수 있습니다. 그래프는 2D 그리드에 파티클을 생성하고 XY 좌표를 다시 매핑하여 텍스처를 샘플링합니다. 그런 다음 샘플링된 텍스처 값의 임계값에 따라 파티클이 소멸합니다.



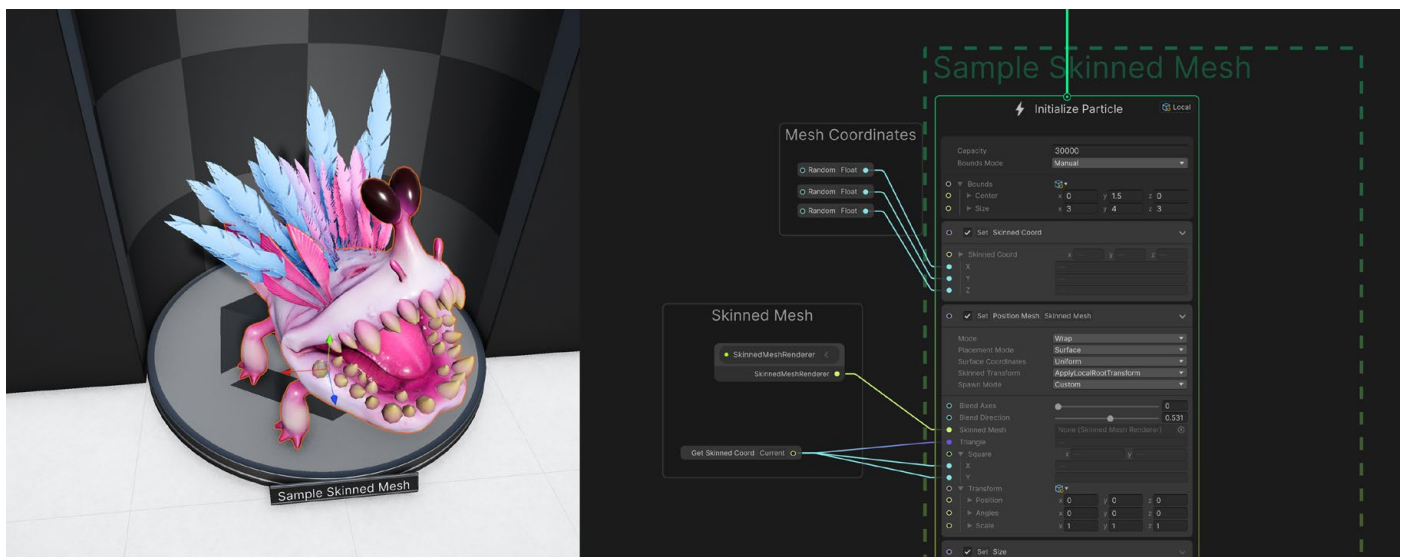
2D 이미지에 파티클을 생성합니다.

2D 또는 3D 오브젝트의 셰이프와 윤곽을 표현하는 기법인 **SDF(Signed Distance Field)**를 샘플링할 수도 있습니다. SDF는 공간의 특정 지점에서 오브젝트 표면의 가장 가까운 지점까지의 거리를 계산합니다. **SampleSDF** VFX Graph에서는 SDF를 사용하여 파티클이 메시 표면을 따라 기어다니도록 만드는 방법을 확인할 수 있습니다.



Signed Distance Field에 파티클을 생성합니다.

리깅된 캐릭터나 프랍(prop)에 효과를 적용해야 하는 경우 스킨드 메시를 샘플링하여 표면 위치, 버텍스 컬러, UV, 노멀 등에 대한 정보를 얻습니다. **Sample Skinned Mesh** VFX Graph는 크리처의 등에 깃털을 생성하기 위해 표면 UV를 가져옵니다.



스킨드 메시를 샘플링하면 이 효과가 구동됩니다.

충돌 및 상호 작용

VFX Graph는 GPU에서 파티클을 시뮬레이션하기 때문에 파티클이 일반 리지드바디 콜라이더와 충돌할 수 없습니다. 하지만 지오메트리 프리미티브(박스, 원뿔, 평면, 구)와 충돌하거나 [Collide with Signed Distance Field](#) 또는 [Collide with Depth Buffer](#) 블록을 사용하여 복잡한 셰이프와 충돌할 수 있습니다.



Collision Simple VFX Graph

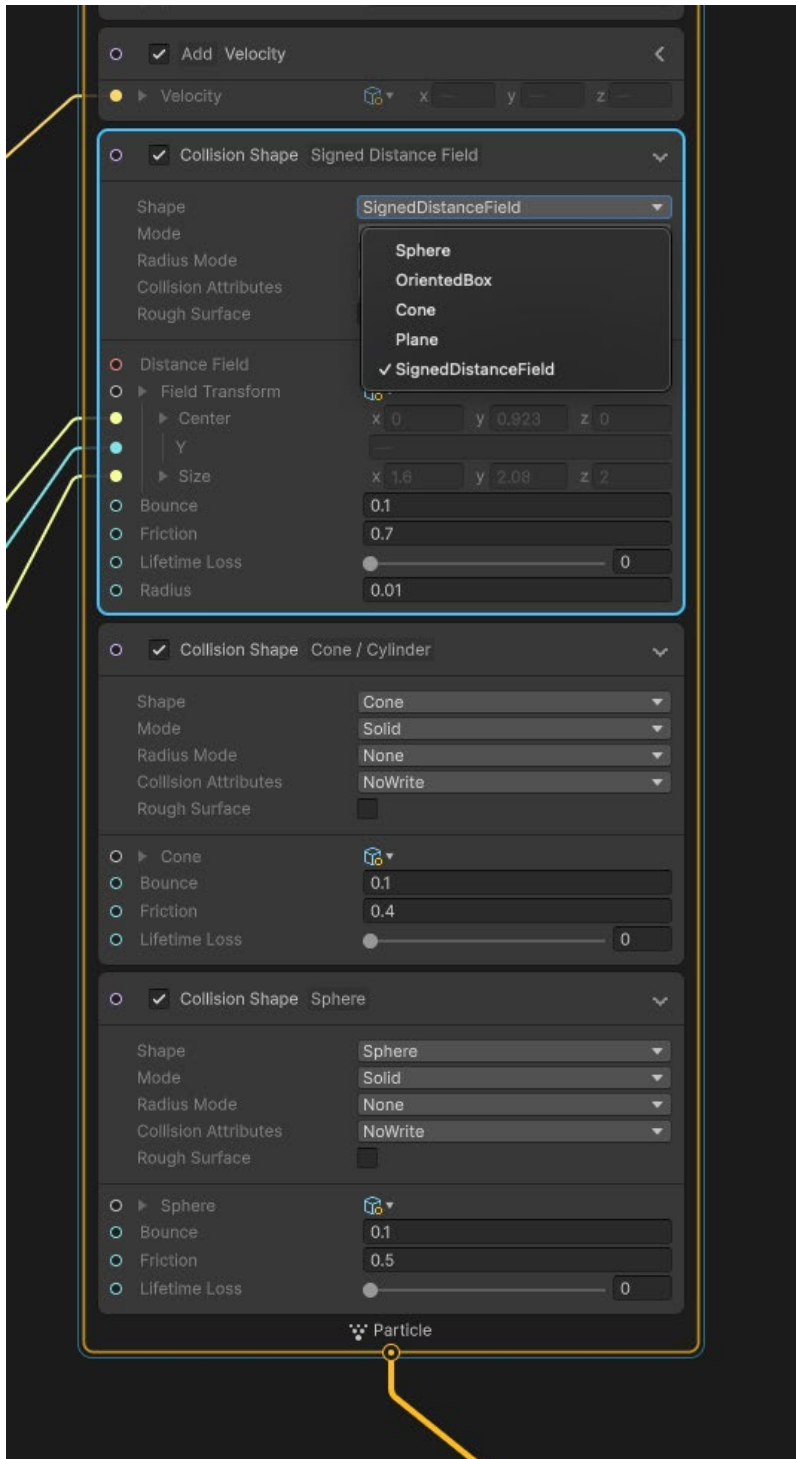
Collision Simple VFX Graph를 열면 기본 충돌 그래프 로직을 설정하는 방법을 확인할 수 있습니다.

Collision Properties VFX Graph는 Bounce, Friction, Lifetime Loss, Roughness 같은 프로퍼티가 파티클의 충돌 반응에 어떤 영향을 미치는지를 나타냅니다.



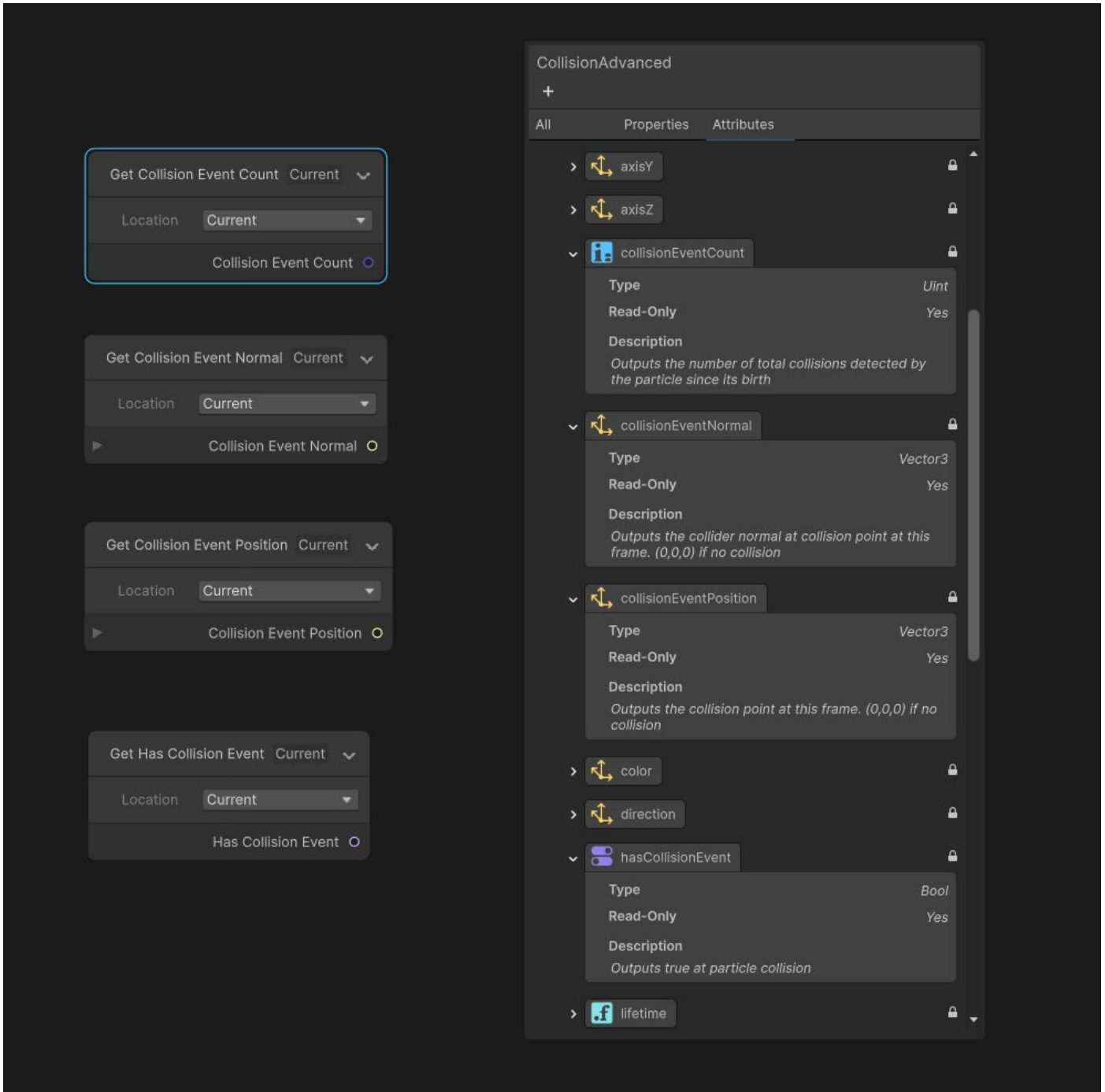
Bounce, Friction, Roughness가 충돌에 영향을 미칩니다.

Unity 6에서는 개별 **Collider** 블록이 Collision Shape 블록으로 결합되어 여러 셰이프를 쉽게 전환할 수 있습니다. 몇 가지 옵션을 추가하여 안정성과 정확도가 모두 개선되었습니다. 컨텍스트 내에서 여러 개의 Collision Shape 블록을 연결하여 원하는 효과를 얻을 수 있습니다.



Collision Shape 블록을 사용하여 콜라이더 셰이프를 전환합니다.

몇 가지 새로운 충돌 속성(**collisionEventCount**, **collisionEventNormal**, **collisionEventPosition**, **hasCollisionEvent**)은 충돌 이벤트에 대해 자세한 정보를 제공하여 보다 정밀한 제어와 상호 작용을 지원합니다.



새로운 충돌 속성을 사용하면 파티클 충돌을 더욱 정밀하게 제어할 수 있습니다.

단순한 충돌 셰이프만으로 환경과의 충돌을 충분히 정밀하게 구현할 수 없는 경우, Signed Distance Field를 사용하면 복잡한 지오메트리를 추정해 적용할 수 있습니다. **Collision Advanced** VFX Graph는 **Collide with Signed Distance Field** 블록을 사용하여 손 조각과의 파티클 충돌을 구현하는 방법을 보여 줍니다.



Signed Distance Field는 복잡한 지오메트리를 추정해 충돌에 활용할 수 있습니다.

데칼 파티클

데칼을 사용하면 환경과 동적 오브젝트 또는 스킨드 메시에도 텍스처를 투사할 수 있습니다. 이러한 방식으로 성능에 큰 영향을 주지 않으면서 씬에 시각적 복잡성을 더할 수 있습니다. 데칼의 일반적인 용도는 다음과 같습니다.

- 총알 구멍 또는 그을린 자국
- 먼지 및 때
- 그래피티, 포스터, 간판
- 트랙 및 발자국
- 초목 및 이끼

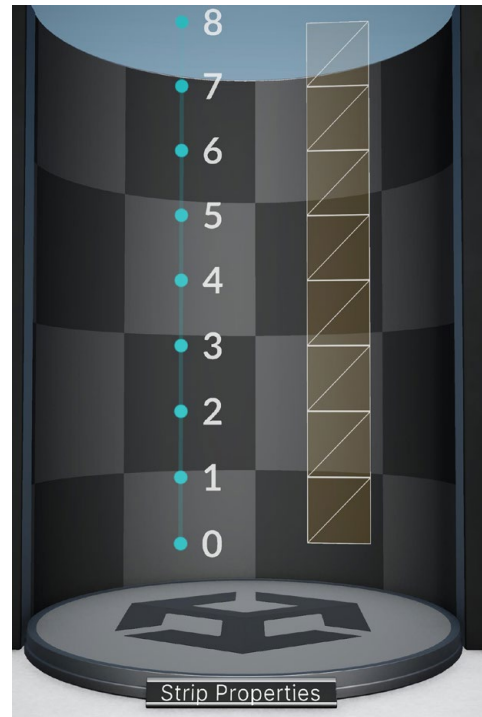
출력 데칼을 사용하면 파티클을 데칼로 렌더링하고 기본 컬러 맵(알베도), 노멀 맵, 마스크 맵을 사용하여 표면에 프로퍼티를 투사할 수 있습니다. 이 예시는 애니메이션화된 Skinned Mesh Renderer 컴포넌트에 데칼을 투사하는 방법을 보여 줍니다.



데칼은 동적 오브젝트에도 시각적 복잡성을 더합니다.

파티클 스트립

파티클 스트립 시스템은 연결된 파티클 그룹으로, 파티클 사이에 사각형을 그려서 리본이나 트레일 같은 효과를 만들 수 있습니다. **Strip Properties** 템플릿에서 두 개의 간단한 파티클 스트립을 가져올 수 있는데, 왼쪽에는 선이 있고 오른쪽에는 더 넓은 리본이 있습니다.



Strip Properties 템플릿으로 두 개의 파티클 스트립을 생성할 수 있습니다.

스트립을 생성하려면 Initialize Particle Strip 컨텍스트를 사용하여 Strip Capacity 및 Particles Per Strip Count를 설정합니다. 이러한 설정은 시스템에서 처리할 최대 스트립 수와 각 스트립을 구성할 파티클 수를 결정합니다. 두 설정 모두 스트립의 외형과 동작에 영향을 줍니다.

그래프 내에서 다음과 같은 몇 가지 속성은 파티클 스트립에만 해당됩니다.

- **Strip Index**는 각 파티클이 속한 스트립을 정의합니다. 이와 같은 간단한 설정에서는 모든 파티클이 하나의 파티클 스트립에 속하므로 Strip Index가 0이 됩니다.
- **ParticleIndexInStrip** 속성은 파티클 스트립 버퍼 내 파티클 위치를 나타내는 uint입니다. 이 값은 0부터 스트립 내 파티클의 총 개수 사이에 해당하는 숫자입니다(이 예시에서는 0~8 사이).
- 이 **SpawnIndexInStrip** 속성은 스트립의 Spawn Index를 나타내는 uint입니다. ParticleIndexInStrip 속성과 달리, SpawnIndexInStrip은 고유 ID가 아닙니다. 같은 스트립에서 다른 프레임에 생성된 두 파티클은 같은 인덱스 값을 가질 수 있습니다.

이 예시에서는 모든 파티클이 단일 버스트를 사용하여 같은 프레임에 생성되므로 SpawnIndexInStrip과 ParticleIndexInStrip의 값이 동일합니다.

Initialize ParticleStrip 컨텍스트는 Update ParticleStrip 컨텍스트에 연결됩니다. 마지막으로 Output ParticleStrip 컨텍스트는 ParticleStrip을 렌더링합니다. 여기서 Stretch, Repeat Per Segment, Custom 같은 Tiling Mode 옵션을 사용하여 스트립을 따라 텍스처가 매핑되는 방식을 제어할 수 있습니다.

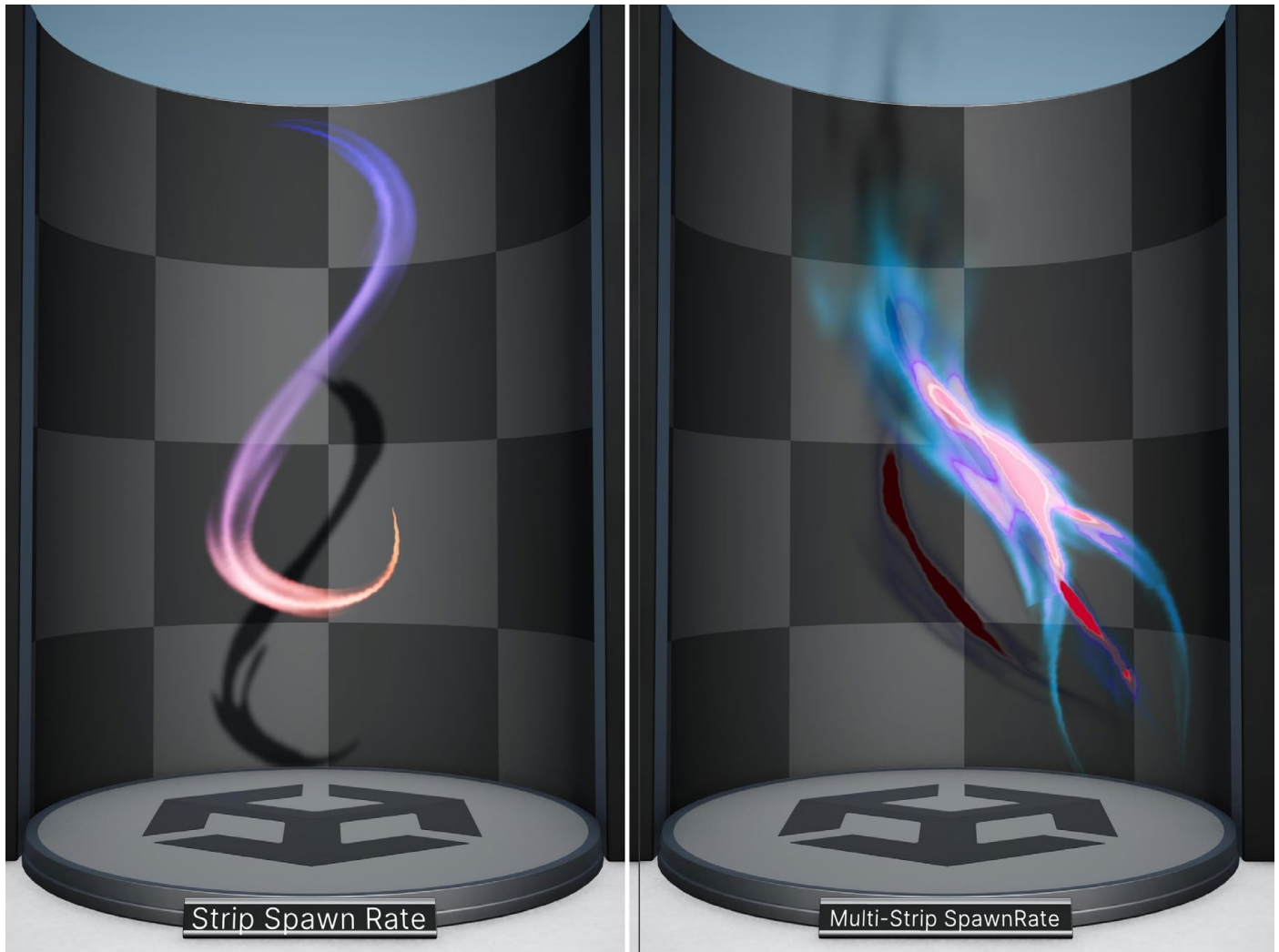
파티클 스트립을 설정합니다.



파티클을 생성하는 방법에는 여러 가지가 있으며, 이는 파티클 스트립을 사용할 때 VFX Graph를 설정하는 방식에 영향을 미칠 수 있습니다. 이 예시에서는 다양한 설정을 확인할 수 있습니다.

Strip SpawnRate 템플릿에는 파티클의 연속 생성 속도로 단일 트레일을 만드는 방법이 나와 있습니다.

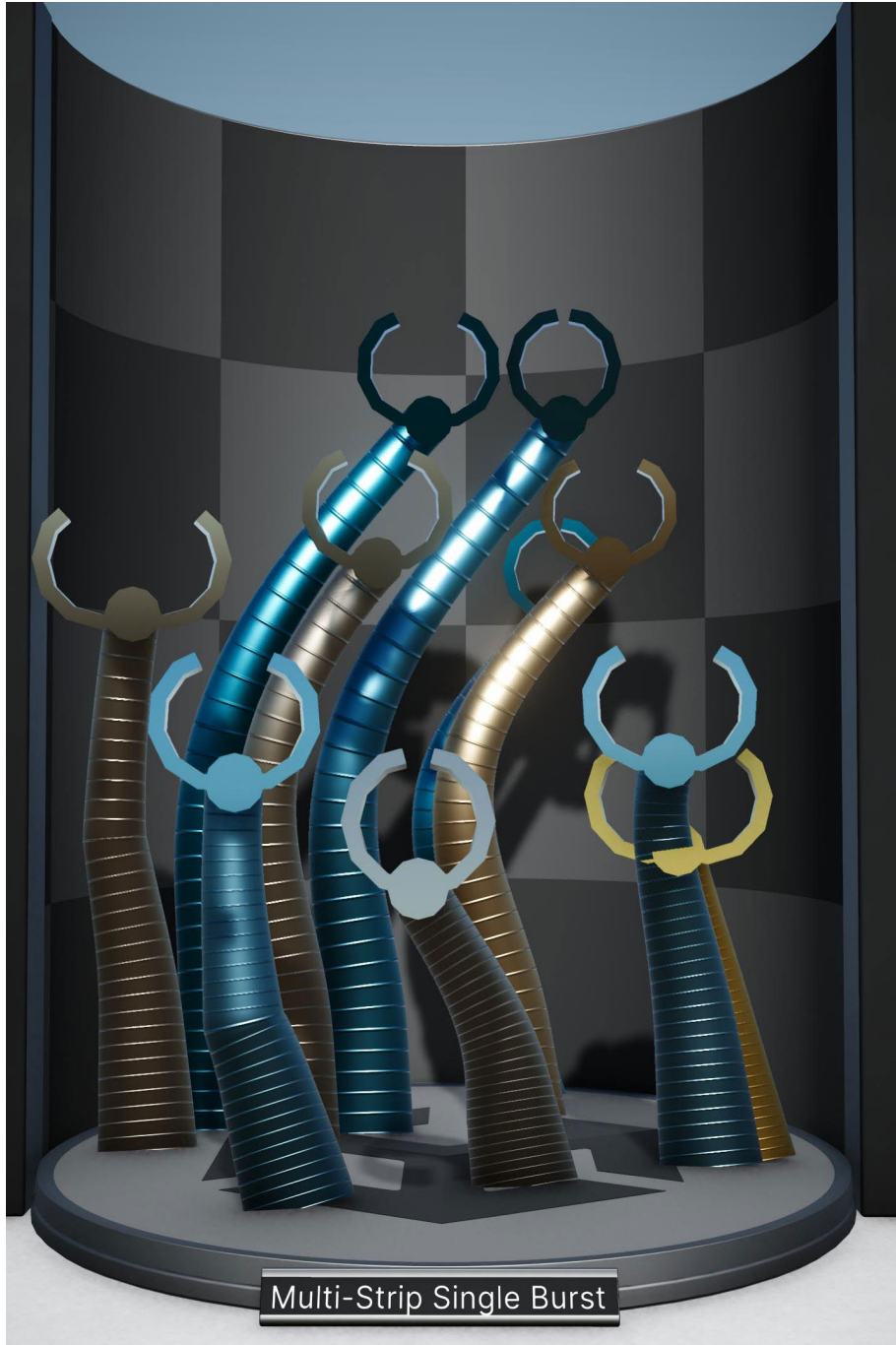
Multi-Strip SpawnRate 템플릿에는 연속 생성 속도로 여러 개의 트레일을 만드는 방법이 나와 있습니다.



이 예시에서는 연속 생성 속도를 사용하여 트레일을 생성합니다.

이 예시에서는 한 번의 버스트로 여러 로봇 팔을 생성합니다. Strip Index 설정 방법은 **Multi-Strip Single Burst** 템플릿을 참고하세요(Spawn Index를 스트립당 파티클 수로 나누기).

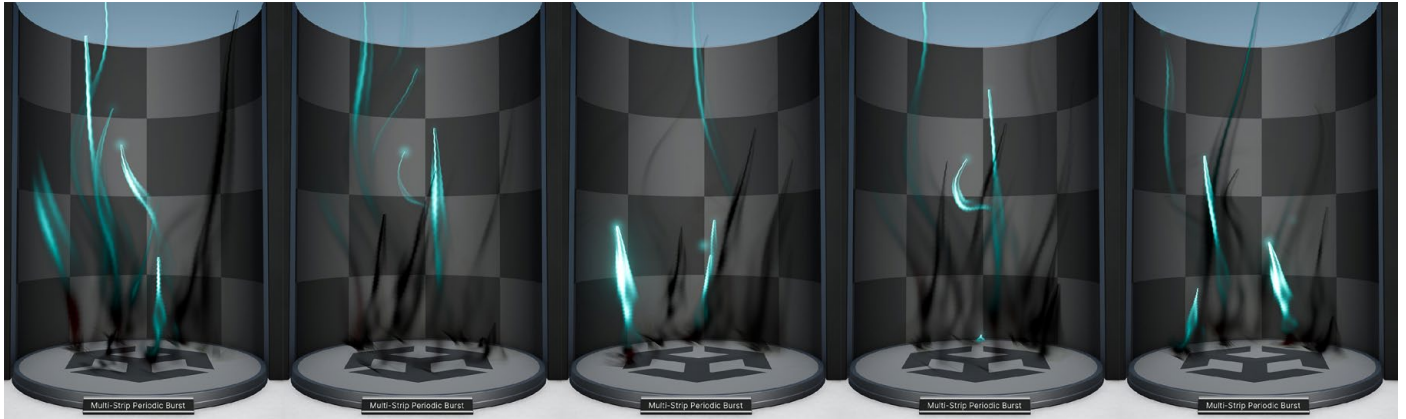
로봇 축수는 파티클 스트립이고 끝부분의 집게는 릿 메시 파티클입니다.



한 번의 버스트로 여러 파티클 스트립을 생성합니다.

한편 **Multi-Strip Periodic Burst** 예제에는 시스템에서 각 주기적 버스트에 새로운 트레일을 생성하는 방법이 나와 있습니다. Loop Index는 Spawn 컨텍스트 외부에서 사용하기 위해 texIndex로 저장됩니다. 그런 다음 이 값은 초기화 중에 Strip Index로 사용되고 ‘스트립 비율’로 정규화됩니다.

이 스트립 비율을 계산하면 파티클 스트립을 유용하게 다룰 수 있습니다. 이 예시에서는 스트레치, 크기 변경, 중력, 난기류 같은 힘에 반응하는 방식 등의 동작과 외형에 영향을 미칩니다.



스트립 비율을 계산하여 파티클 스트립 동작을 관리합니다.

이 **Strip GPU Event** 예시는 자라나는 버섯 VFX의 예를 보여 줍니다. 버섯의 머리(Mushroom Hat System)는 파티클 메시를 렌더링한 것이고, 버섯의 줄기(Mushroom Foot System)는 파티클 스트립으로 만들어집니다.

한 시스템은 다른 시스템에 대한 부모 역할을 하며, GPU 이벤트를 사용하여 다음 시스템을 트리거합니다.

이 메커니즘은 버섯이 완전히 자라면 먼지와 연기 구름을 생성하여 효과를 더합니다.



GPU 이벤트로 버섯의 성장을 구현합니다.

Multi-Strips GPU Event 템플릿에는 파티클 스트립으로 더 복잡한 효과를 만드는 방법이 나와 있습니다. 이 예시에서 각 헤드폰 잭은 파티클 메시로, 경로를 따라 파티클을 생성하여 일반적으로 부모 파티클당 하나의 스트립을 생성하는 방식을 우회합니다.

각 잭은 바닥에서 스피커 아래의 플러그까지 베지어 경로를 따라 뱀처럼 유기적으로 움직이는 케이블 트레일을 형성합니다. 잭이 성공적으로 연결되면 녹색 불빛 파티클이 추가되어 효과에 역동적인 느낌을 더합니다.



헤드폰 잭 뒤의 파티클 스트립.

VFX Graph 샘플(HDRP)

Visual Effect Graph 샘플은 VFX Graph와 관련된 다양한 시나리오를 선보입니다. 활용 사례로 몇 가지 예시 그래프를 확인할 수 있으므로, 프로젝트를 다운로드하여 작업 참고 자료로 활용하세요.

참고: 이 샘플은 HDRP만 지원하며 URP와 호환되지 않습니다.

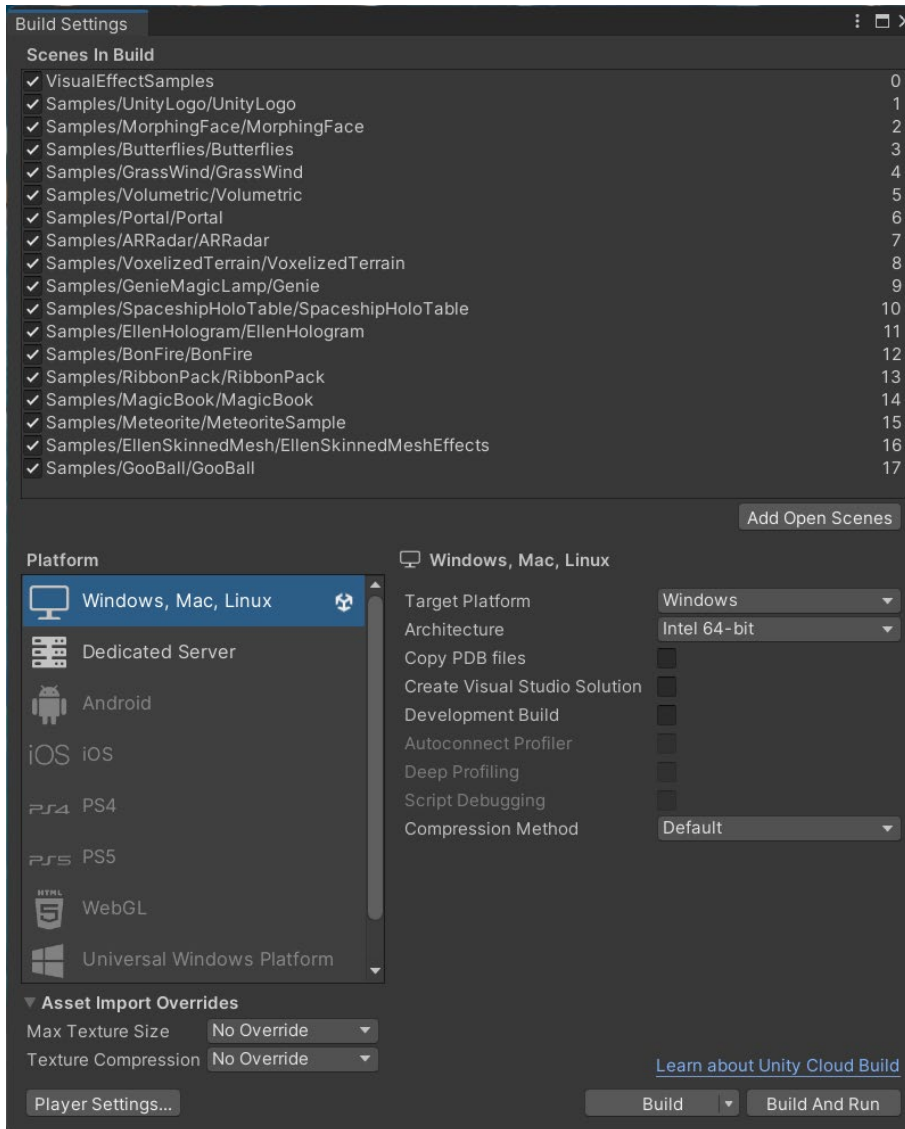


저장소에서 제공하는 VFX Graph 샘플 예시

Release 탭에는 샘플의 스냅샷과 미리 제작된 바이너리에 대한 링크가 준비되어 있습니다. 전체 저장소를 클로닝할 수도 있습니다.

각 샘플은 프로젝트의 **Assets/Samples** 폴더에 있는 하위 디렉토리에 있습니다. 메인 **VisualEffectsSample** 씬은 루트에 있습니다.

플레이어를 만들어야 하는 경우, **Build Settings**에서 이 메인 씬을 인덱스 0으로 설정하세요. 그런 다음 나중에 사용할 다른 모든 씬을 추가하세요.



Build Settings 창

VFX Graph 샘플은 고유의 효과를 선보이는 여러 씬으로 구성됩니다. 그중 몇 가지를 자세히 살펴보겠습니다.

연기 포털

연기 포털은 밀도 높은 볼류메트릭 연기가 소용돌이치며 신비로운 포털을 형성하는 효과입니다. 이 효과는 새로운 **6방향 연기 조명** 머티리얼(아래 참조)을 사용하여 Houdini로 시뮬레이션된 연기 애니메이션과 실시간 조명을 결합합니다. VFX Graph는 이 기법을 통해 6개의 축에서 베이킹된 라이트 맵을 사용할 수 있습니다.



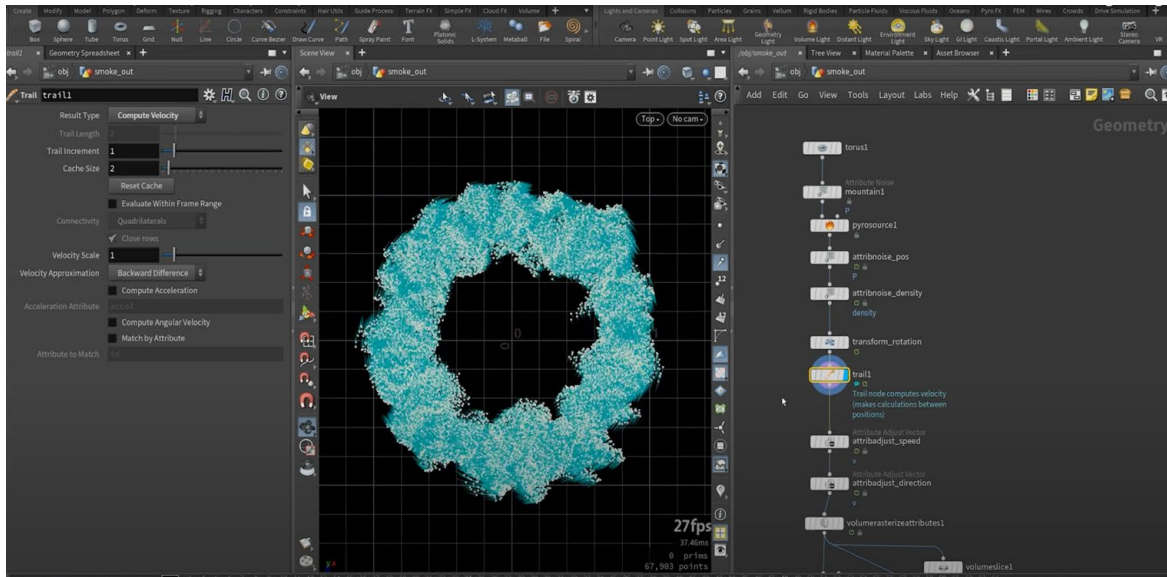
메인 포털을 둘러싸고 있는 추가 요소는 신비로운 외관을 더욱 돋보이게 합니다. 바로 깜박이는 불꽃, 스파크, 떠다니는 바위, 포털 주변의 공간을 뒤틀려 보이게 하는 미묘한 왜곡 효과들이 그렇습니다.



연기 포털은 6방향 연기 조명 재터리얼을 선보입니다.

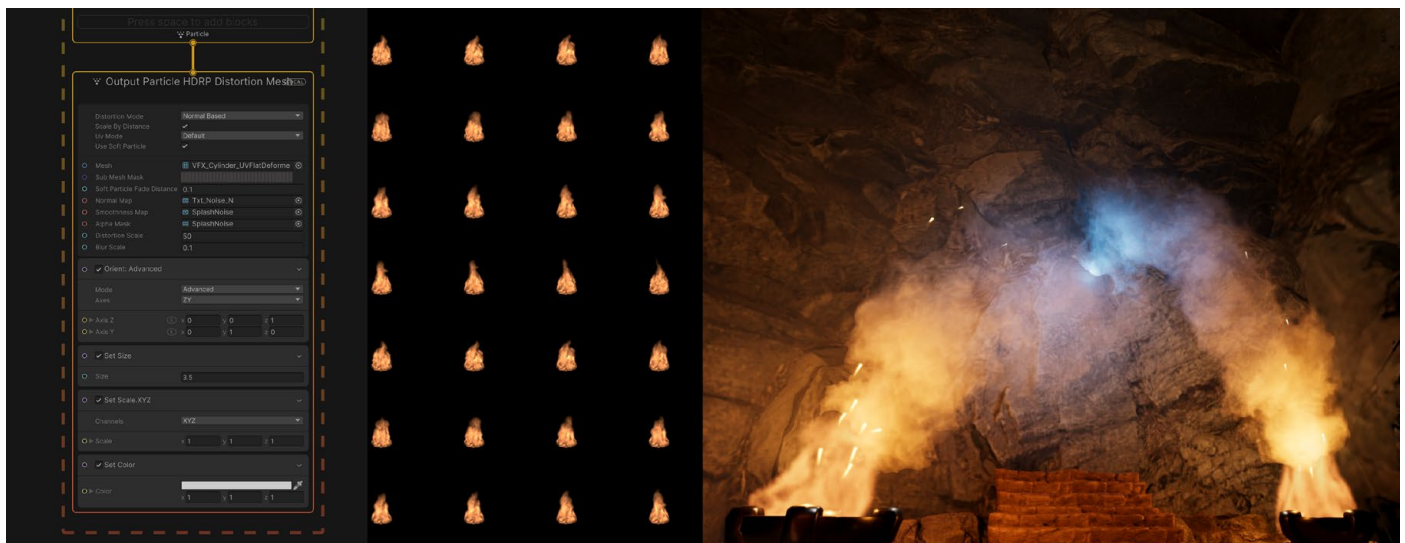
이 **제작 동영상**에서 이 샘플의 전체 워크플로를 살펴볼 수 있습니다. 제작 과정에서 몇 가지 중요한 단계는 다음과 같습니다.

- **Houdini 시뮬레이션:** 연기 포털 VFX는 Houdini에서 익스포트한 연기 시뮬레이션을 포함한 레이어로 구성됩니다. 원래의 소용돌이는 Houdini의 노이즈가 있는 토러스에서 시작됩니다. 밀도, 온도, 속도 속성이 연기 시뮬레이션을 구동한 다음 8×8 플립북 텍스처로 익스포트됩니다.



연기 포털은 Houdini 시뮬레이션으로 시작됩니다.

- **불꽃 및 추가 효과:** 불꽃 효과는 플립북 텍스처를 사용하여 개발되고 생생한 느낌을 위해 연기 링과 블렌딩됩니다. 바위와 스파크가 씬에 추가되어 중력으로 끌어당기는 느낌을 연출하고 주변 환경과 효과를 통합합니다.



불꽃은 플립북 텍스처입니다.

- **6방향 조명:** 6방향 조명 기능은 6개의 서로 다른 축에서 베이킹된 라이트 맵을 사용합니다. 이 기법은 계산 비용이 큰 볼류메트릭 렌더링을 피하기 위해 스프라이트를 대신 사용합니다. 실시간 광원이 연기와 상호 작용하여 실제 볼류메트릭 렌더 없이도 사실적인 조명과 그림자를 생성할 수 있습니다.
- **플립북 블렌딩:** 모션 벡터는 플립북의 프레임을 블렌딩하는 데 도움을 줍니다. 에셋 스토어에서 제공하는 **TFlow**는 익스포트된 플립북 텍스처에서 모션 벡터를 생성하는 데 도움이 되며, 텍스처 해상도를 낮출 때 TFlow 모션 벡터 맵을 사용하면 품질 저하를 보완할 수 있습니다. 플립북 텍스처는 포물선 모양의 메시에 적용되어 볼륨 및 환경과의 통합을 개선합니다.
- **절차적 크리스탈:** VFX Graph는 주변 환경에 다양한 크리스탈 셰이프를 생성하는 데 도움이 됩니다. **인스턴싱**은 VFX 크리스탈을 단일 배치로 그룹화하여 계산 부하를 줄입니다.



VFX Graph 인스턴싱은 크리스탈 요소를 최적화하는 데 도움이 됩니다.

- **HDRP 조명:** 스폿 광원은 불꽃 주위에 조명을 집중적으로 비추고, 면 광원은 경로를 따라 주변광을 완성시킵니다. 동적 조명 효과는 Light Flicker 스크립트로 조명 위치에 애니메이션을 추가하여 씬을 더 생생하게 만듭니다. 점 광원은 반경을 작게 유지하여 시각적 품질 저하 없이 설정을 최적화합니다.

- **최적화:** VFX Graph의 성능을 개선하는 몇 가지 전략이 있습니다. 여기에는 포털의 플립북 텍스처를 8K에서 2K 해상도로 줄이고, 알파 클리핑을 사용하여 투명 오버드로우를 줄이고, 크리스탈 요소에 [VFX Graph의 인스턴싱](#) 기능을 활용하는 것 등이 포함됩니다.

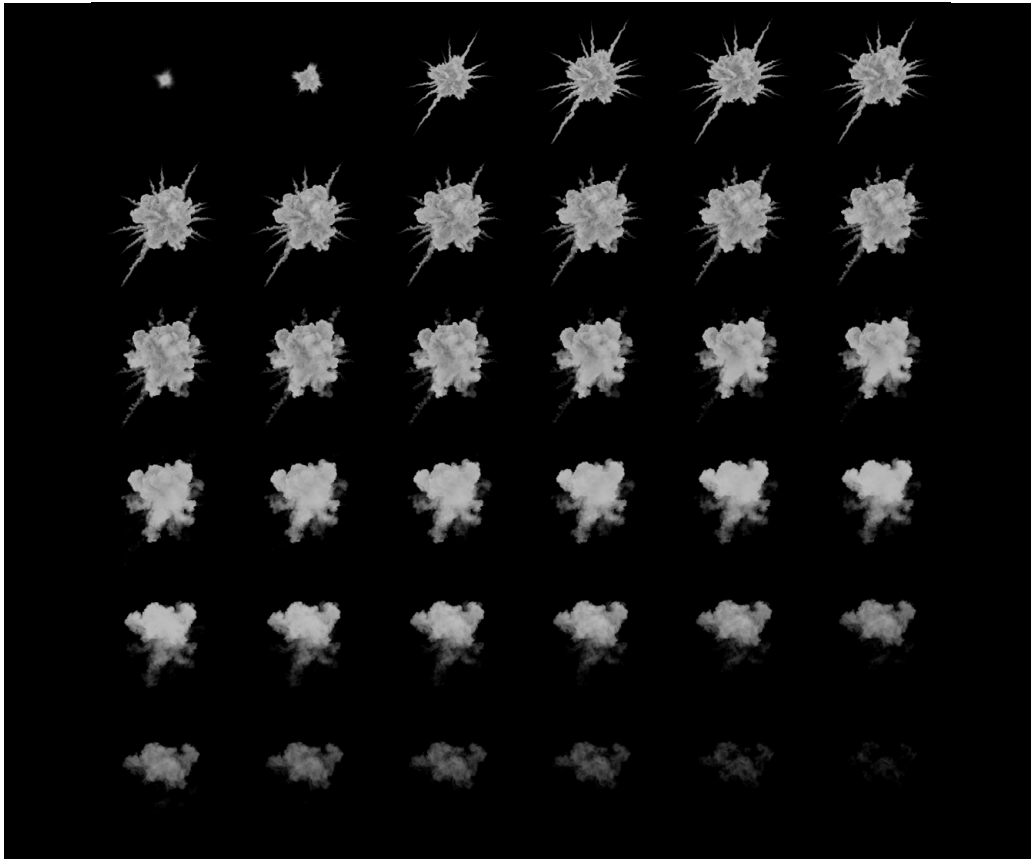


이 포털에 여러 기법이 결합되어 최종 효과를 만들어 냅니다.

자세한 내용을 알아보려면 [분석 동영상](#)을 시청하세요.

실시간 연기를 위한 6방향 조명

대부분의 게임에서 완전한 볼류메트릭 렌더링은 비용이 너무 많이 들기 때문에 사실적인 연기 효과를 실시간으로 렌더링하기는 어려울 수 있습니다. 이 문제를 해결하기 위해 많은 개발자가 플립북 텍스처를 사용하는데, 이는 일반적으로 텍스처를 $n \times n$ 그리드로 분할하는 방식입니다. 하지만 이 경우 해상도의 일부만 표시되므로 해상도와 메모리 측면에서 손해를 보게 됩니다.



연기 효과를 위한 일반적인 8×8 플립북 텍스처

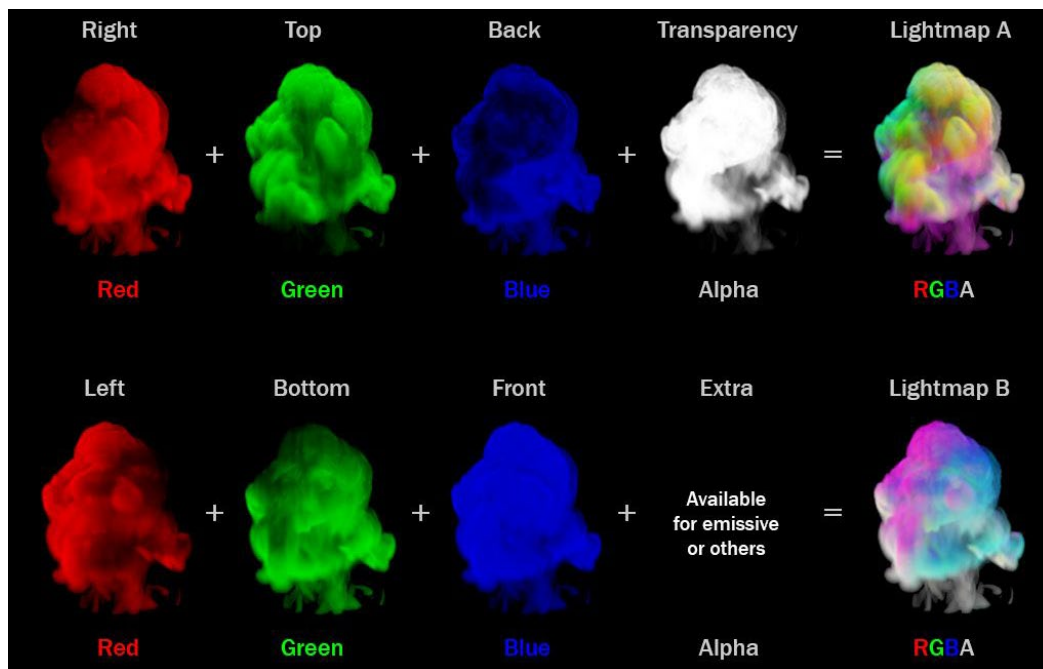
여기서 일반적으로 생기는 의문이 있습니다. 플랫 2D 지오메트리를 기반으로 연기 효과에 조명을 비추려면 어떻게 해야 할까요? 다양성을 위해 텍스처를 더 추가하면 메모리 할당량이 빠르게 소모되어 텍스처 품질과 폭발 효과의 다양성이 모두 감소하게 됩니다. 노멀 매핑이나 완전히 베이킹된 컬러 맵 같은 방법은 사실적이지 않거나 충분히 유연하지 않은 경우가 많습니다.

여기서 필요한 것이 바로 6방향 조명입니다. 이 방법을 사용하면 베이킹된 시뮬레이션에서 연기를 렌더링할 수 있으며, 다양한 조명 환경에서 원활하게 작동합니다. 비용 효율적인 프로세스로 볼류메트릭 연기 효과를 근접하게 구현할 수 있는 것입니다.

6방향 조명의 비결은 서로 다른 6개의 방향(위, 아래, 왼쪽, 오른쪽, 앞, 뒤)에서 조명을 비출 때 연기가 어떻게 보이는지 캡처하는 라이트맵 세트를 사용하는 데 있습니다.

그런 다음 이 라이트맵을 두 개의 RGBA 텍스처로 베이킹합니다. 첫 번째 텍스처의 RGB 채널은 상단, 왼쪽, 오른쪽 맵을 저장하고 알파 채널은 하단 맵을 저장합니다.

두 번째 텍스처의 RGB 채널은 앞, 뒤, 추가 맵을 저장하고 알파 채널은 이미시브 마스크(선택 사항)에 사용할 수 있습니다.

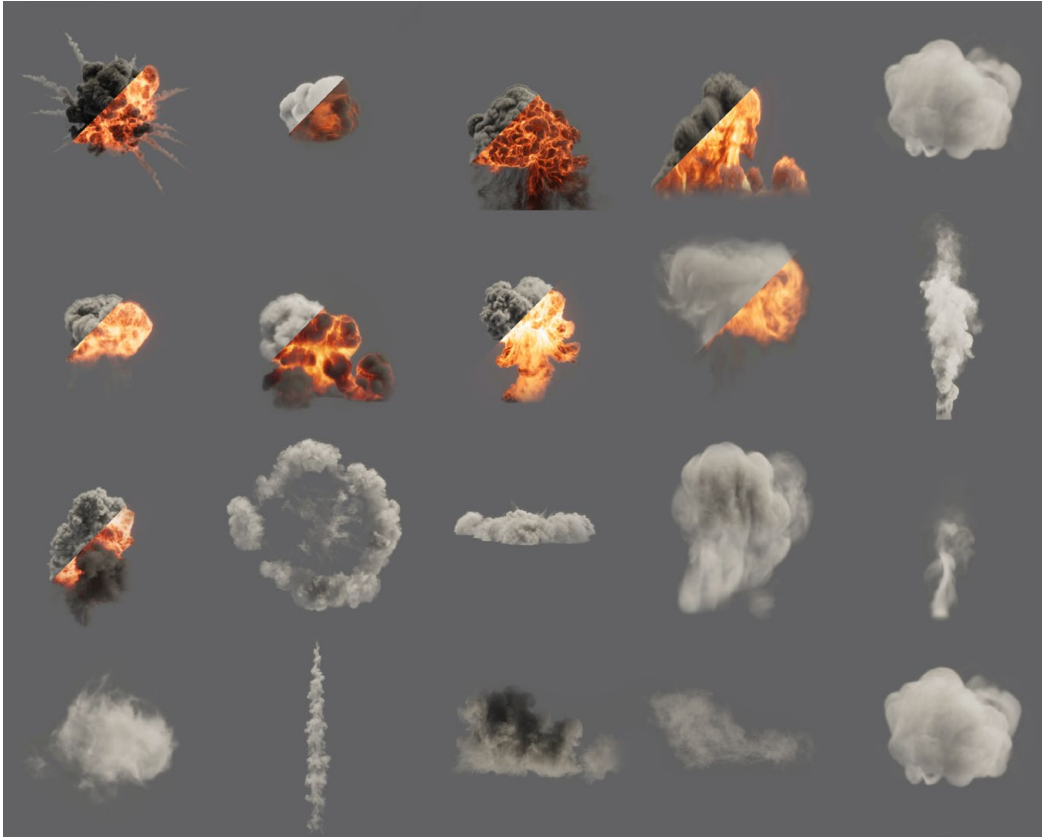


6개의 라이트맵은 서로 다른 광원의 방향을 캡처합니다.

렌더링할 때 셰이더는 각 파티클에 대한 광원의 방향에 따라 6개의 라이트맵을 블렌딩합니다. 이렇게 모든 조명 유형의 직접 조명과 라이트 프로브 및 기타 전역 조명 기법의 간접 조명을 사용하여 다양한 조명 조건에 맞게 연기를 동적으로 음영 처리할 수 있습니다.

6방향 조명의 장점은 다음과 같습니다.

- **복잡한 광원 상호 작용 및 산란 효과:** 연기에서 사실적인 빛의 산란을 시뮬레이션하여 시각적 복잡성을 더하고 연기가 더욱 자연스럽게 보이게 합니다.
- **환경과의 시각적 통합 개선:** 동적 셰이딩을 통해 연기 효과가 주변 환경과 블렌딩되며 변화하는 조명 조건에 적응할 수 있게 됩니다.



6방향 조명으로 다양한 조명 조건에서 다양한 연기 렌더링이 가능합니다.

- **비용 효율적인 렌더링:** 6방향 조명은 사실적인 외형을 제공하면서도 성능 친화적이며, 실시간 애플리케이션에 적합합니다.
- **메모리 효율성:** 다양한 조명 환경에 라이트맵 텍스처를 재사용함으로써 여러 텍스처를 사용하는 것보다 메모리를 절약할 수 있습니다.

6방향 조명은 실시간 연기 효과를 렌더링할 때 시각적 품질, 성능, 메모리 사용량 간의 균형을 맞추는 유용한 기법이 될 수 있습니다.

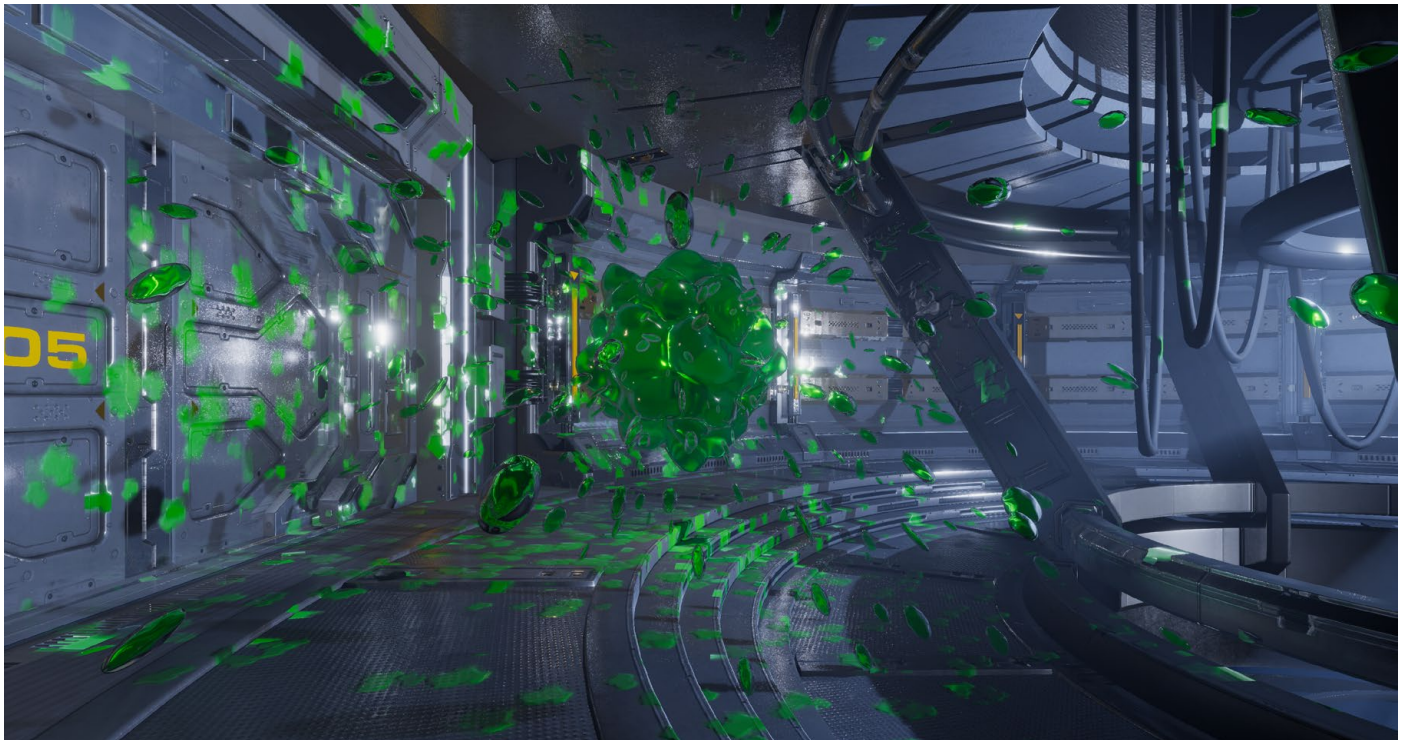


사실적인 연기와 폭발 효과를 구현하려면 6방향 조명을 사용하세요.

VFX Graph: 6방향 조명 워크플로에서 이 기법에 대한 내용을 자세히 알아볼 수 있습니다. 자세한 내용은 이 [블로그 게시물](#)에서도 확인할 수 있습니다.

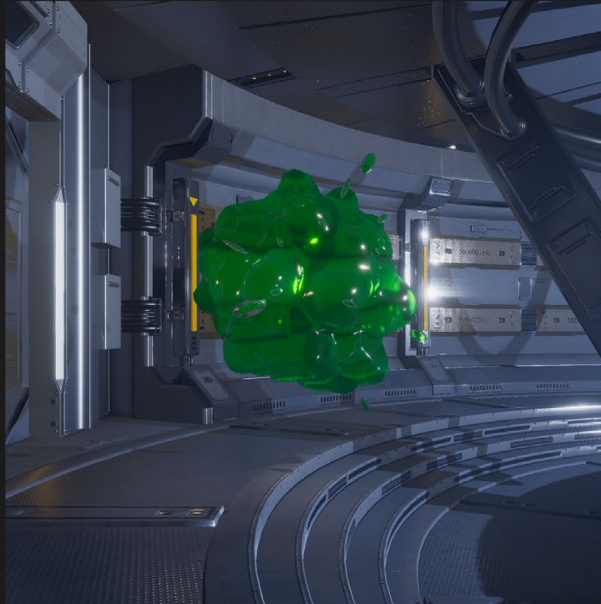
구볼(GooBall)

Unity에서 페인트를 흩뿌려 보고 싶던 적이 있으신가요? 장난스럽고 통통 튀는 느낌을 주는 이 SF 데모의 데칼은 주변 환경과 상호 작용하는 끈적끈적한 물질을 시뮬레이션합니다. 다른 예제들과 마찬가지로, 이 멀티레이어 효과는 몇 가지 시스템을 활용하여 최종 결과물을 만듭니다.

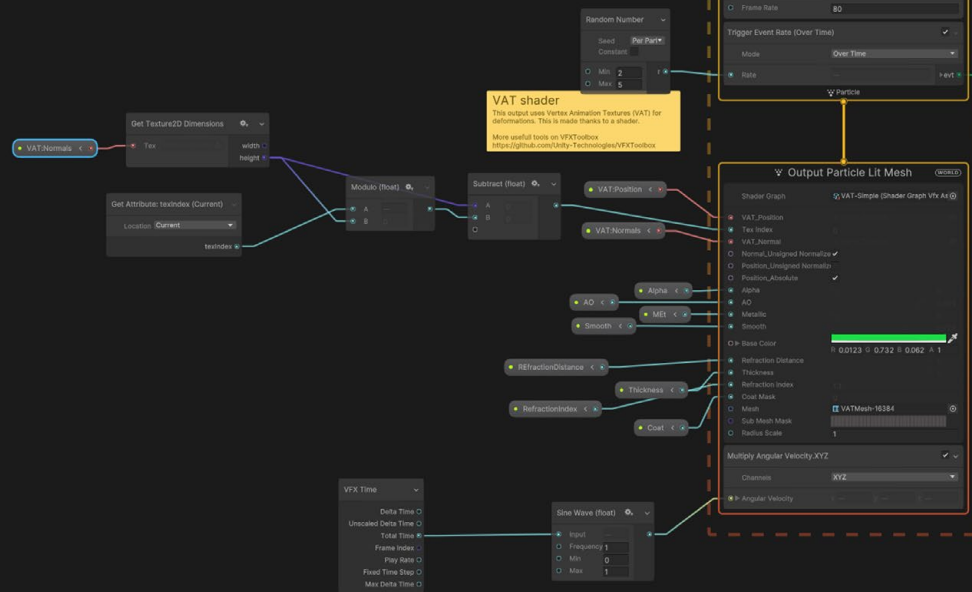


구볼에 적용된 효과

중앙에 위치한 방울은 유체 같은 움직임을 연출하는 **VAT(버텍스 애니메이션 텍스처) Shader Graph**로 시작합니다. 세이더의 스크롤링 텍스처 시트가 메시의 3D 포인트를 계속 위아래로 요동치게 만들어 SF 슬라임에 맞는 효과를 연출합니다.

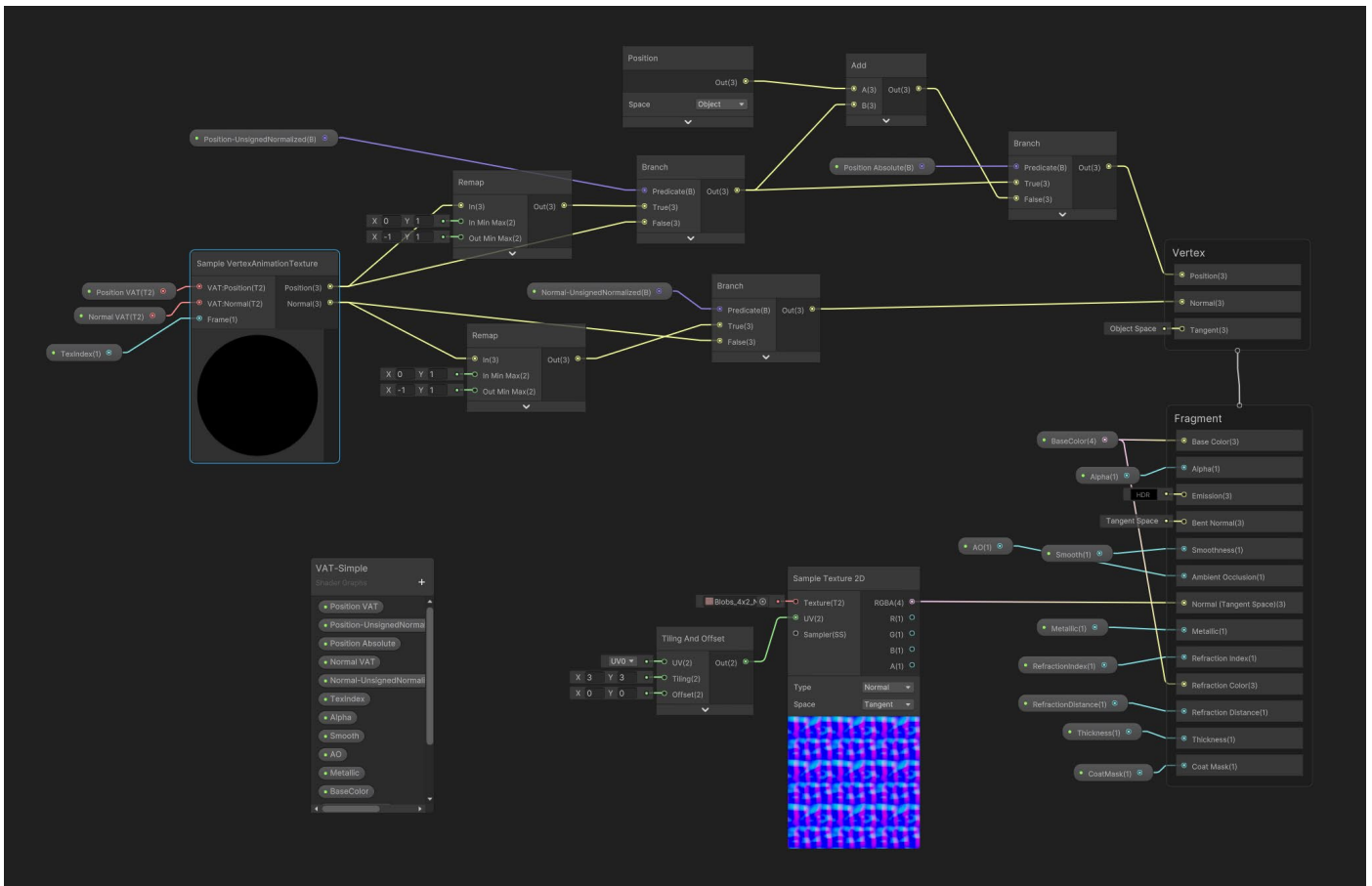


Main blob



주인공 방울

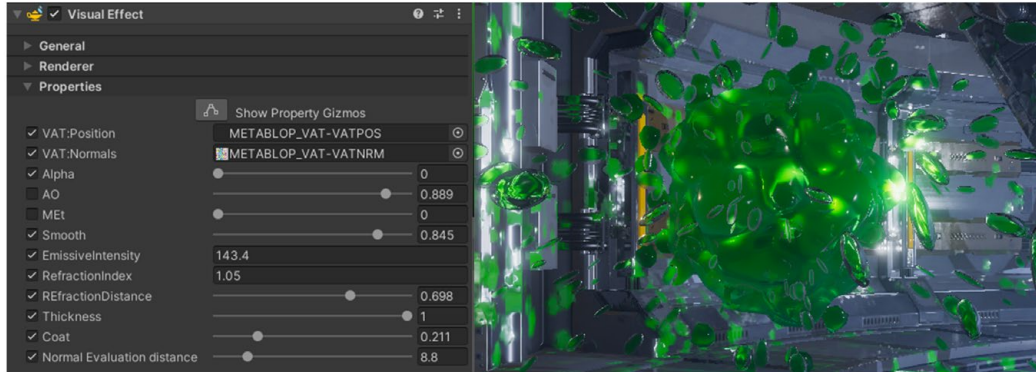
Shader Graph가 **Output Particle Lit Mesh 컨텍스트**에 직접 연결되므로, 셰이더에 영향을 주는 여러 개의 입력 포트가 있습니다. 다음은 VAT Shader Graph를 한눈에 본 모습입니다.



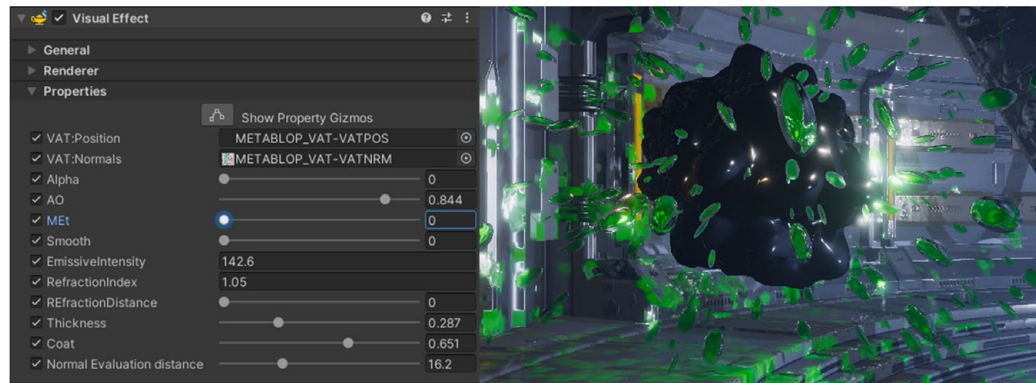
VAT Shader Graph

버텍스 모션 외에도 이 Shader Graph는 투명한 초록색 유리 같은 머티리얼의 형상을 생성합니다. 이 머티리얼이 슬라임의 모습을 만듭니다.

Shader Graph는 VFX Graph와 **밀접하게 연동**되어 있습니다. 이 예시에서는 VFX Graph의 Blackboard에서 GooBall의 셰이딩 파라미터 중 일부를 편집할 수 있습니다. **인스펙터**로 이동하면 통합된 여러 슬라이더와 필드를 사용해 슬라임의 형상을 조정할 수 있습니다.



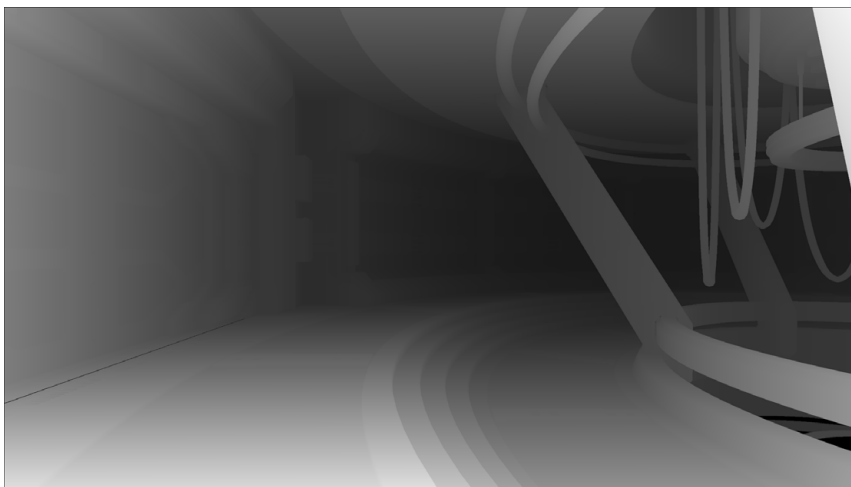
Adjust shader parameters



Blackboard 프로퍼티를 조정하여 슬라임의 형상 수정

주인공 방울은 GPU 이벤트를 트리거하여 구체 표면에 약간의 파티클을 생성합니다. 적절한 맵을 통해 아래쪽으로 힘이 적용되고, 슬라임에서 나오는 물방울이 주기적으로 떨어집니다.

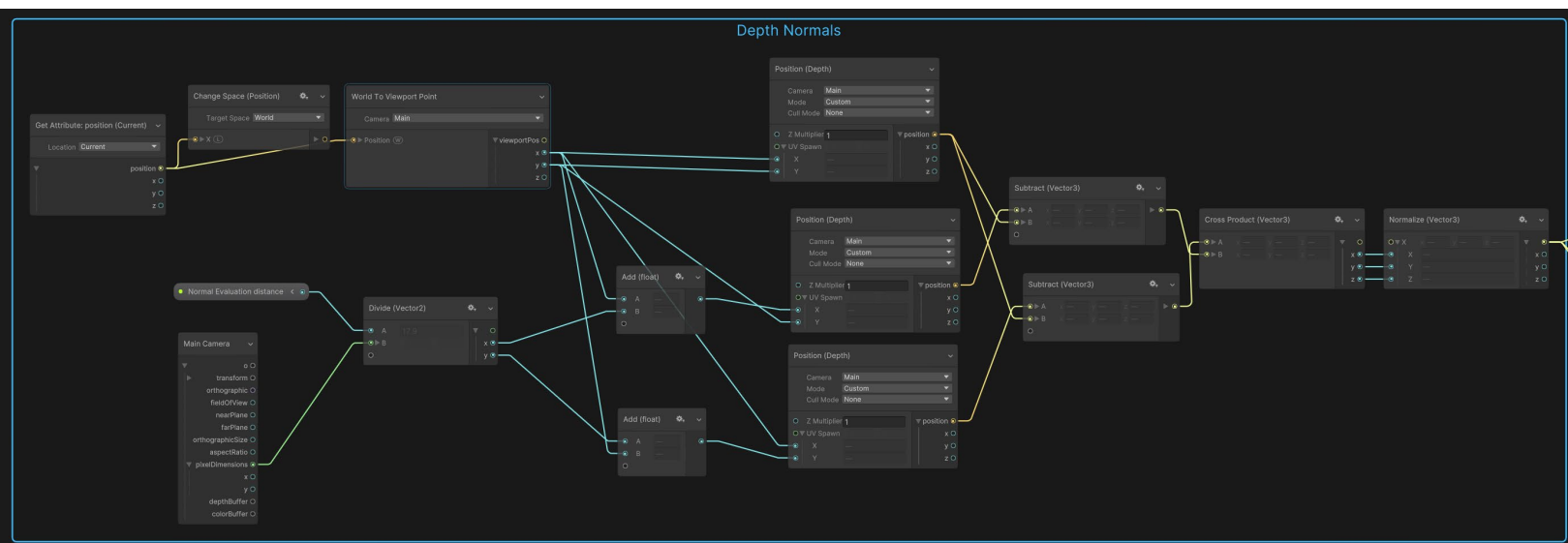
VFX Graph가 바닥의 콜라이더와 직접 상호 작용하지 않더라도, 카메라의 **덱스 버퍼**를 사용해 환경을 근사적으로 표현할 수 있습니다. 정밀도가 중요하지 않다면 괜찮은 방법입니다.



카메라의 덱스 버퍼 시각화

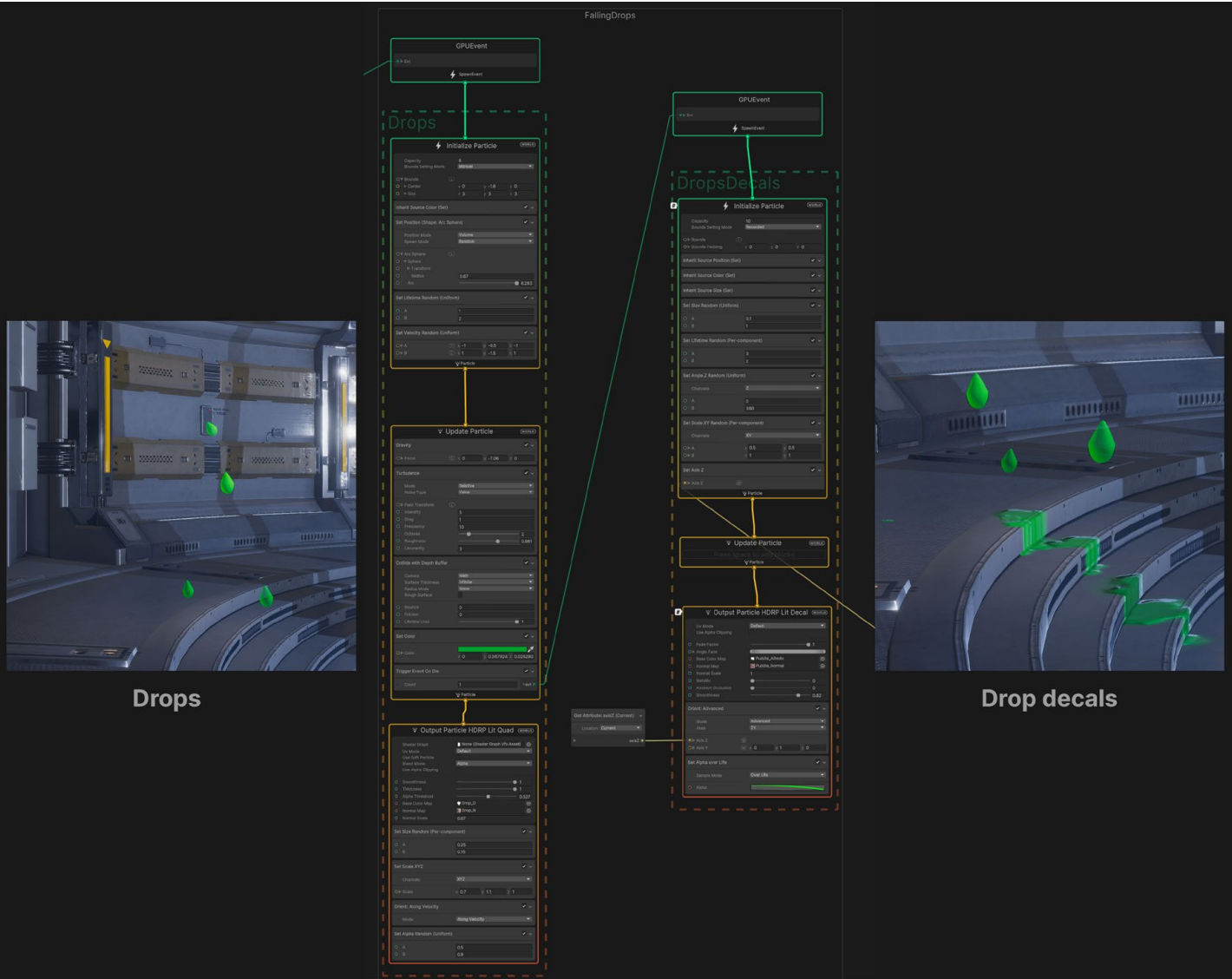
물방울이 **덱스 버퍼**에 닿으면 GPU 이벤트가 트리거됩니다. 이 이벤트는 데칼을 처리하기 위해 위치, 컬러, 크기 속성을 별도의 시스템으로 전달합니다.

데칼이 지오메트리에 올바른 방향으로 흩뿌려질 수 있도록, 다음과 같은 방식으로 버퍼의 덱스 노멀이 산출됩니다.



덱스 노멀

산출된 덱스 노멀은 이후 모든 데칼이 올바른 방향을 향하도록 Z축 속성의 일부가 됩니다. 새로운 웅덩이 텍스처가 생성되어 각 데칼이 고르지 않은 바닥 표면 전체에 실감 나게 흩뿌려집니다.



물방울은 표면에 떨어져 데칼을 남깁니다.

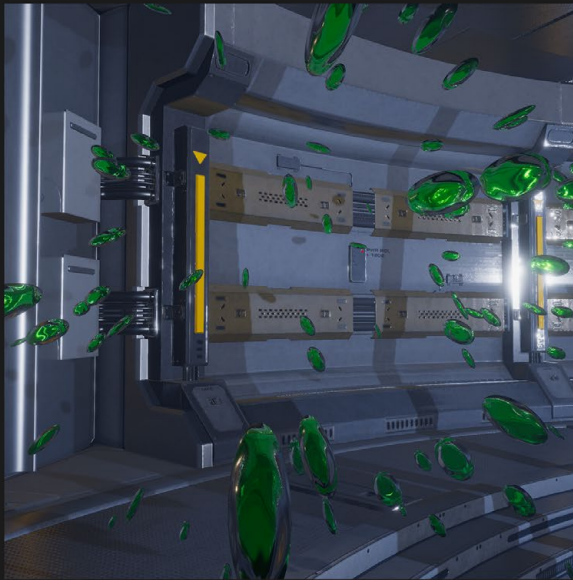
구형 표면에서 무작위로 발사되는 발사체가 더해지며 씬은 더 무질서하게 변합니다. VFX의 장점이 돋보이는 예시입니다.

물리 기반 효과

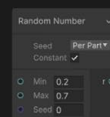
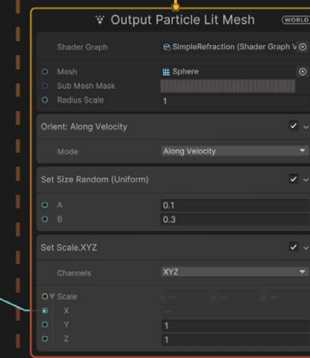
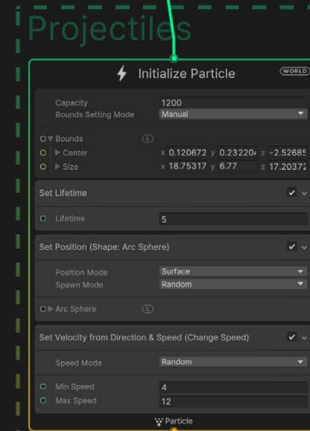
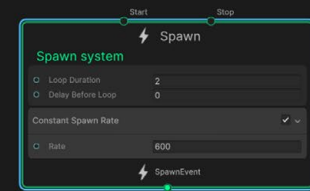
VFX Graph는 복잡한 시뮬레이션을 계산하고 프레임 버퍼를 읽을 수 있습니다. 그러나 파티클 데이터를 C#으로 가져오거나 기본 물리 시스템에 연결하는 기능은 지원하지 않습니다.

따라서 물리 기반 효과를 생성하려면 다음과 같은 차선책을 사용해야 합니다.

- 템스 버퍼
- 프리미티브 표현(구, 상자, 토러스 등)
- **Signed Distance Fields, 포인트 캐시, 벡터 필드** 같은 3D 텍스처

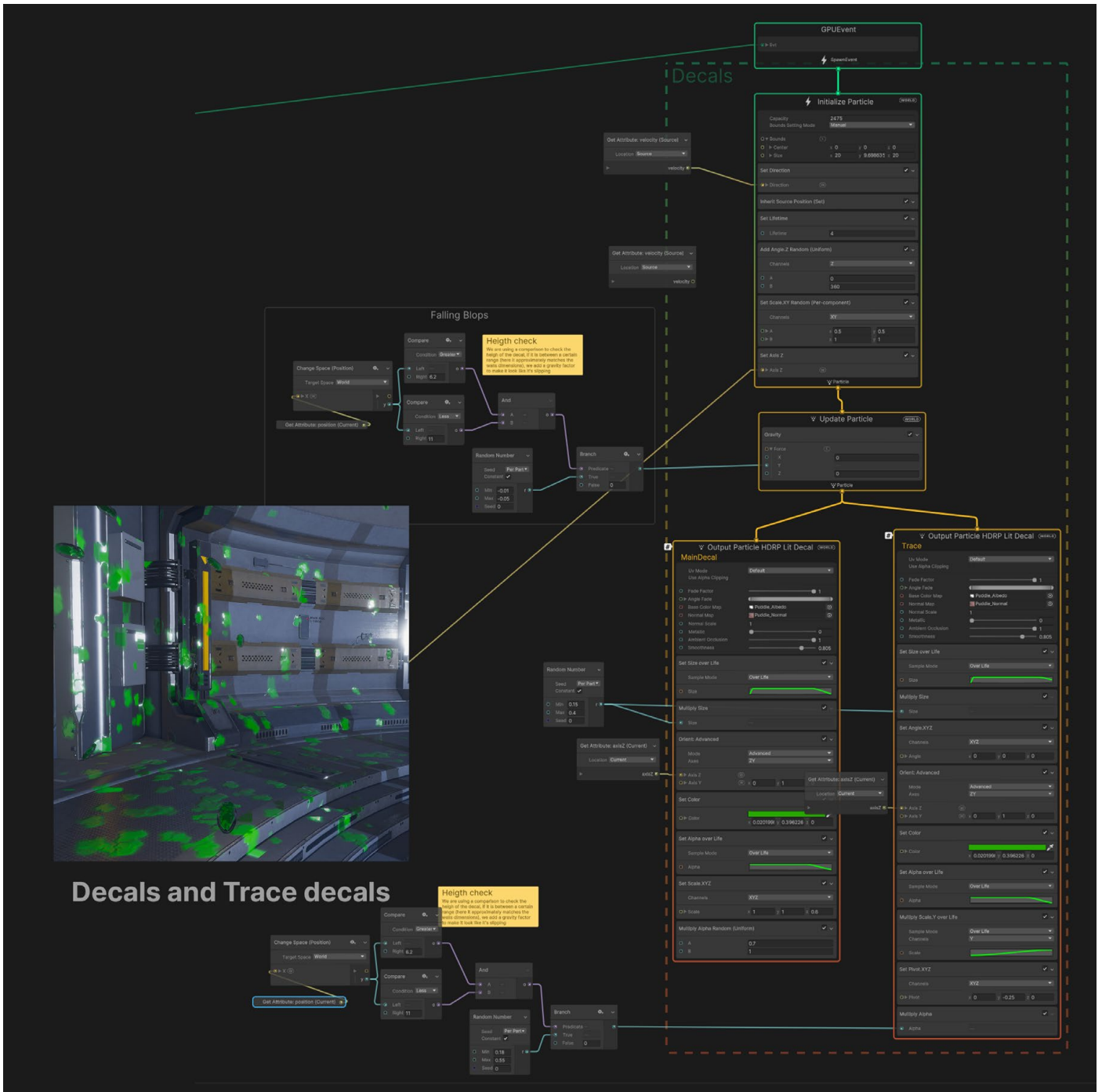


Projectiles



구볼에서 무질서하게 발사되는 발사체

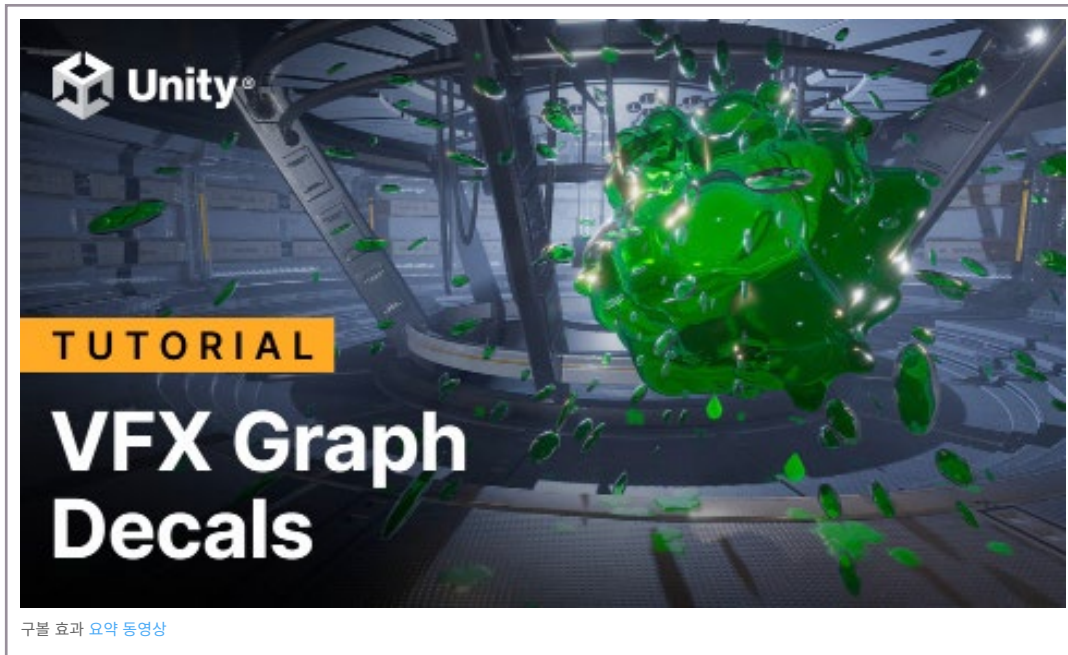
발사체가 공간 지오메트리와 충돌할 때, 유사한 기법으로 슬라임이 흠뻑러지는 모습을 계산합니다. 이번에는 두 번째 Trace 데칼이 벽의 높이 범위 내에 있는지 여부도 확인해야 합니다. 만약 그렇다면 Y 스케일이 약간 움직이며 슬라임이 매끄러운 금속 표면을 따라 미끄러지는 착시 현상이 연출됩니다.



Splat 및 Trace 데칼이 효과를 완성합니다.

정확히 같은 효과를 재현하려 하지 않더라도 이 구불 씬을 통해 다음 내용을 확인할 수 있습니다.

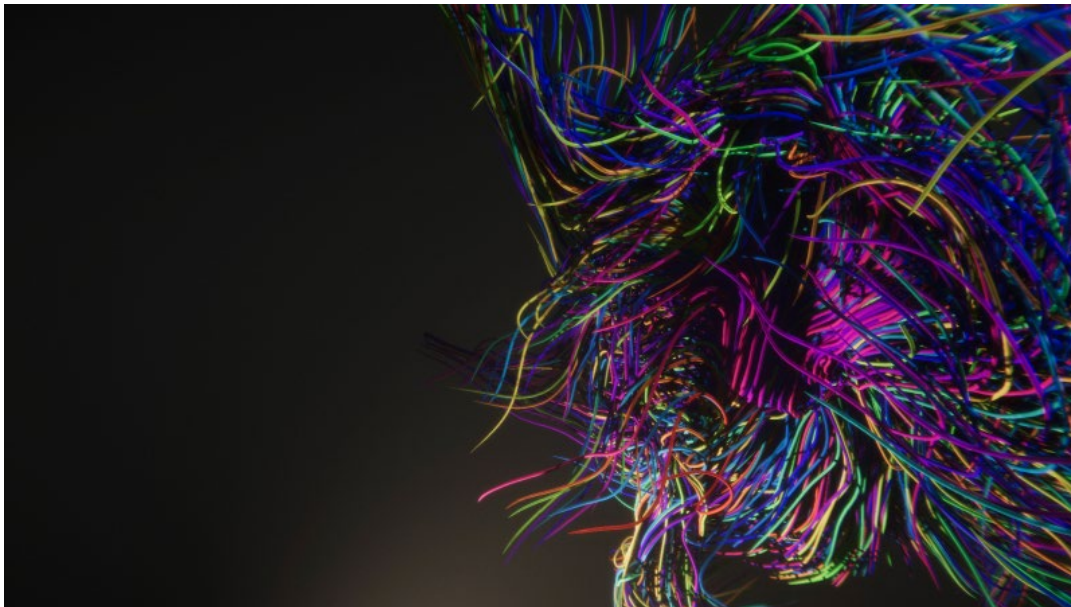
- Shader Graph를 Output 컨텍스트에 통합하는 방법
- GPU 이벤트를 사용해 같은 그래프에 있는 다른 시스템을 트리거하는 방법
- 덤스 노멀을 사용하여 환경 위에 데칼을 적용하는 방법



Ribbon Pack

이 추상적인 효과는 **파티클 스트립**, 즉 선 또는 사각형 스트립으로 렌더링된 파티클 체인의 사용법을 시연합니다.

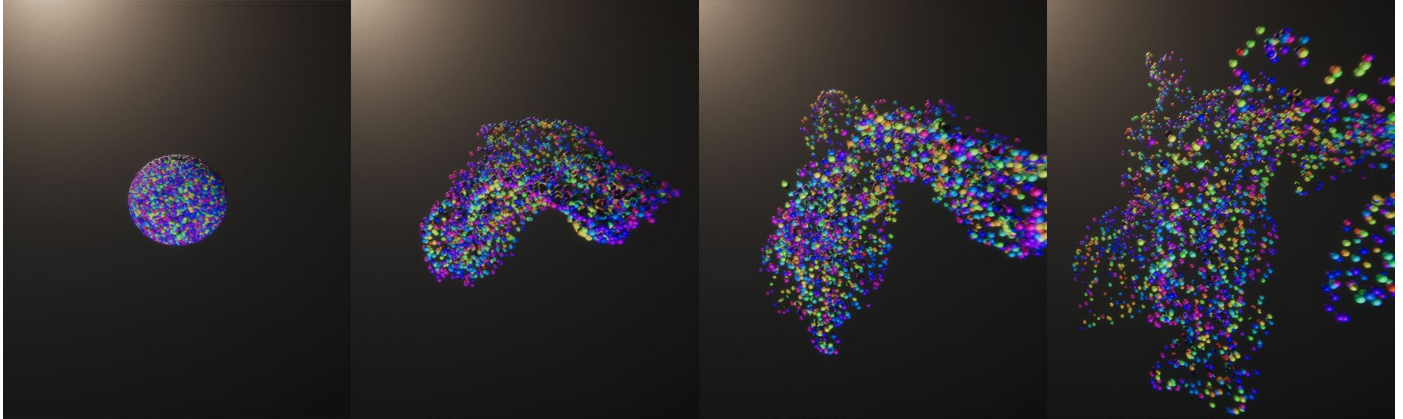
이 효과는 애니메이션 트레일의 시뮬레이션에 자주 사용됩니다. GPU 이벤트를 사용하여 다른 파티클과 함께 움직이도록 유도하고, 트레일의 모든 점이 개별적으로 움직이는 방식을 확인하여 바람과 힘, 난류를 적용할 수 있습니다.



파티클 스트립을 보여 주는 Ribbon Pack

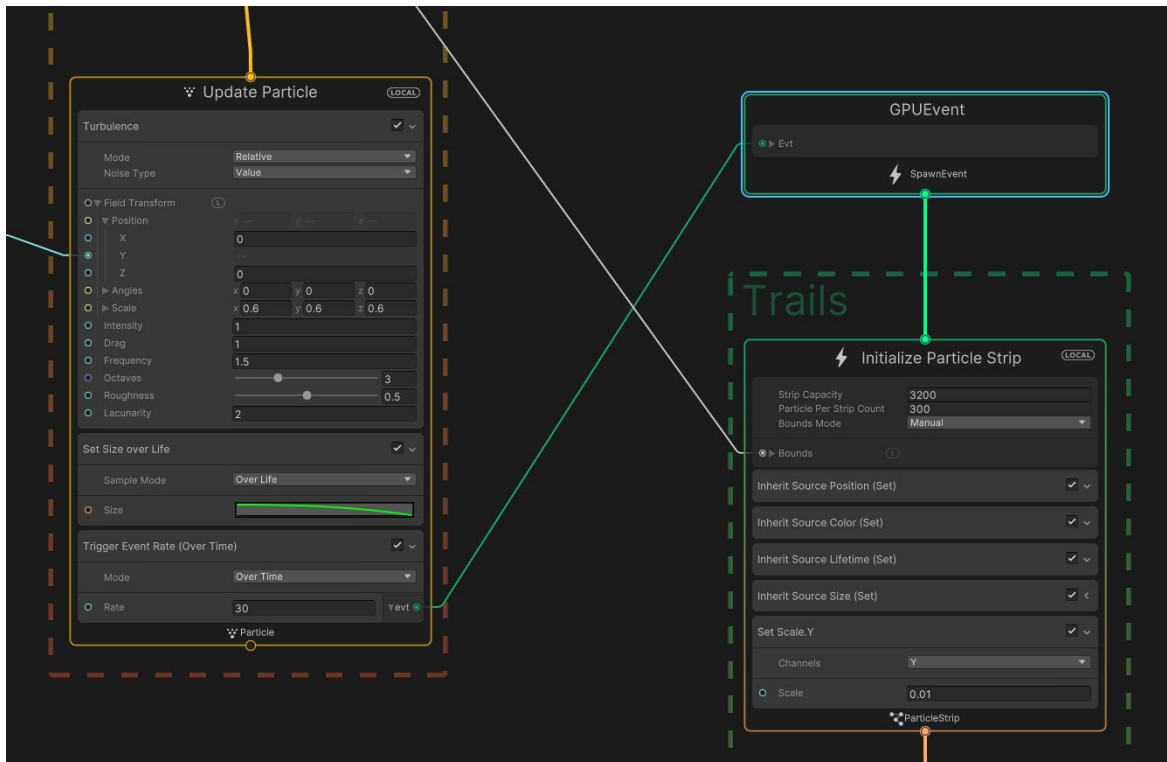
Ribbon Pack 그래프는 구형 호 표면에 다양한 컬러의 파티클을 생성합니다. **노이즈**를 사용하여 유기적인 모션을 추가하거나, 사용할 수 있는 **블록**을 수정하여 각 파티클 스트립의 텍스처 매핑, 생성, 방향을 커스터마이징할 수 있습니다.

메시 파티클을 사용하여 이 효과를 기존 방식으로 렌더링하면 다음과 유사한 모습이 됩니다.



파티클 스트립 없이 Ribbon Pack을 렌더링한 모습

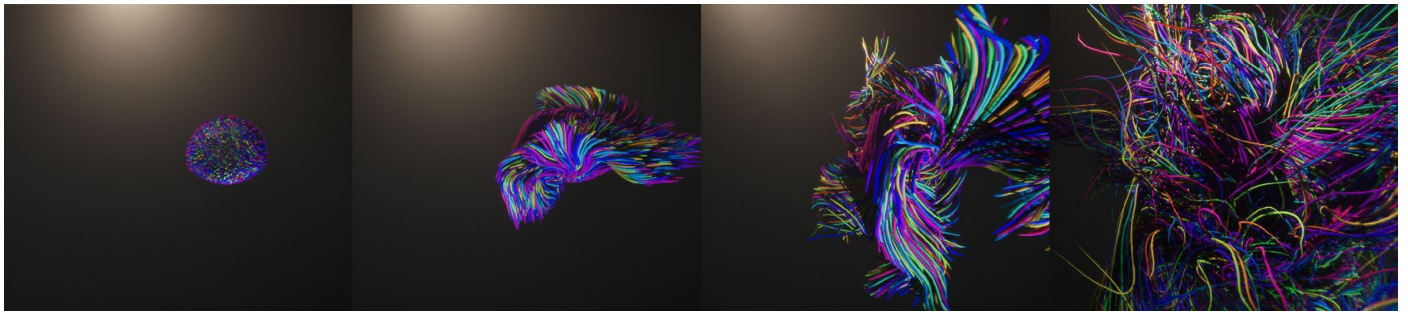
대신 첫 번째 시스템은 숨김 상태가 되어 화면에 나타나지 않습니다. **Trigger Event Rate** 블록은 GPU 이벤트를 호출하는 데 사용됩니다.



GPU 이벤트를 호출하는 Trigger Event Rate 블록



이 블록은 파티클 스트립을 초기화, 업데이트, 렌더링하는 일련의 컨텍스트에 메시지를 전송합니다. 그 결과 케이블이나 섬유와 같은 것이 프리즘처럼 얹힌 모습이 나타납니다.



많은 수의 리본

파티클 스트립은 마법 지팡이의 빛줄기, 무기의 검기, 와이어 등 다양한 용도로 활용됩니다. Magic Book 샘플에서는 각 빛줄기 주변에서 소용돌이치는 트레일에 파티클 스트립이 사용되었습니다.



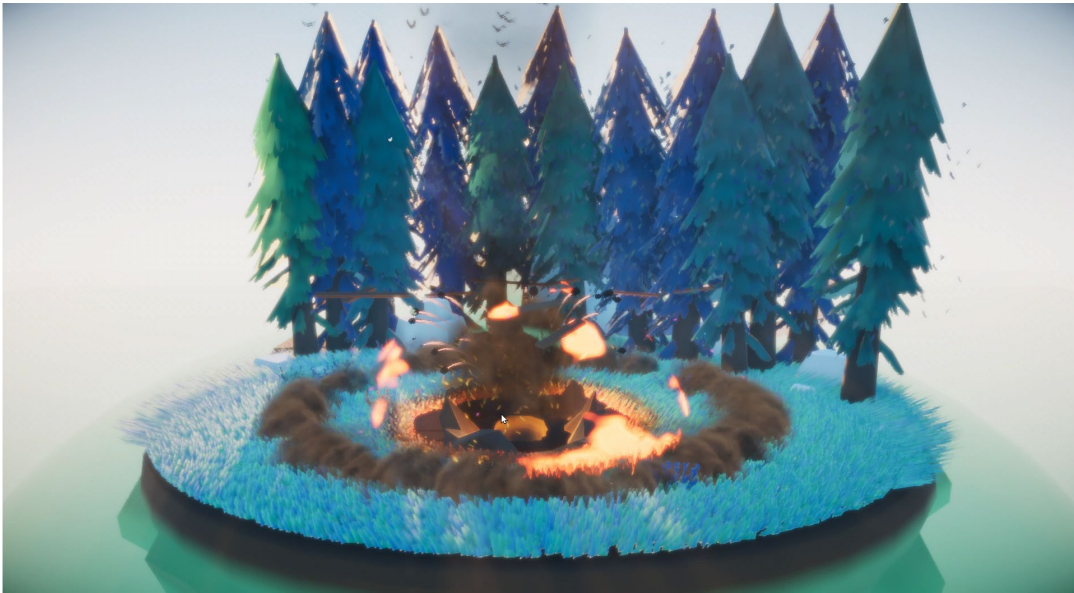
Magic Book 샘플의 파티클 스트립 트레일

아래 섹션에서 설명할 Meteorite 샘플에서 풀잎을 대신하는 효과로 사용되기도 합니다.

Meteorite 샘플

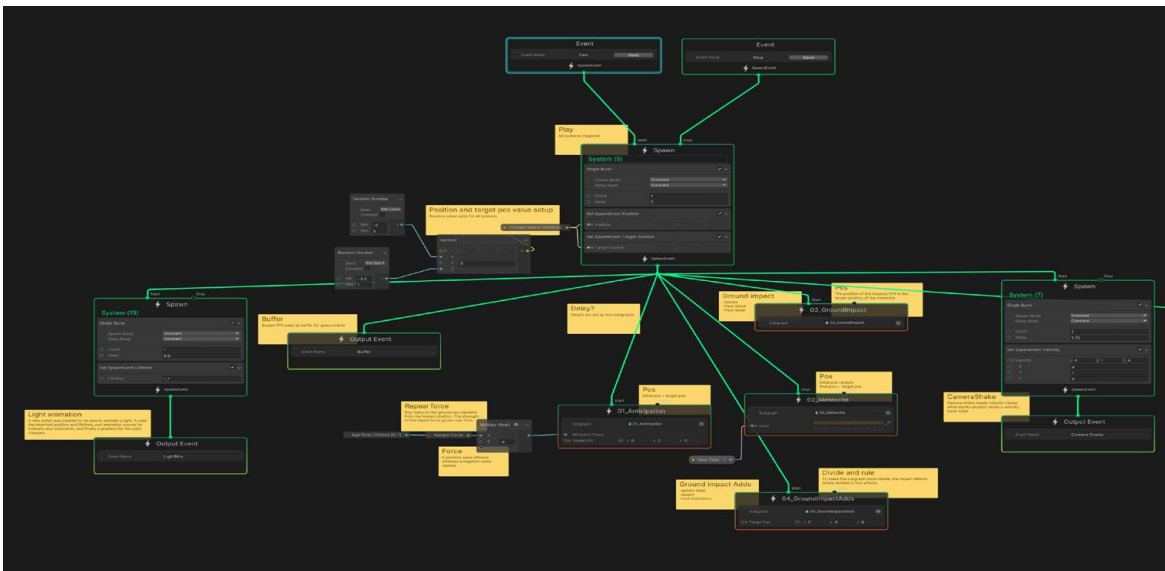
Meteorite 샘플은 여러 효과를 결합하여 지구에 충돌하는 운석의 충격을 강렬하게 표현합니다. **Timeline** 오브젝트의 **MeteoriteControl** 스크립트는 키보드 키 누르기 또는 마우스 클릭 이벤트를 수신 대기한 후 운석 효과를 활성화합니다.

이 씬에서는 지면과 주변의 나무가 폭발에 반응합니다. 하지만 이 효과를 만드는 과정에서 피해를 입은 VFX 동물은 없으므로 걱정하지 마세요.



Meteorite 샘플

여기서는 **MeteoriteMain**이라는 단일 그래프가 다른 여러 그래프를 구동합니다. 아래에서 볼 수 있는 것처럼 그래프 자체는 깔끔하게 정리되어 있습니다.



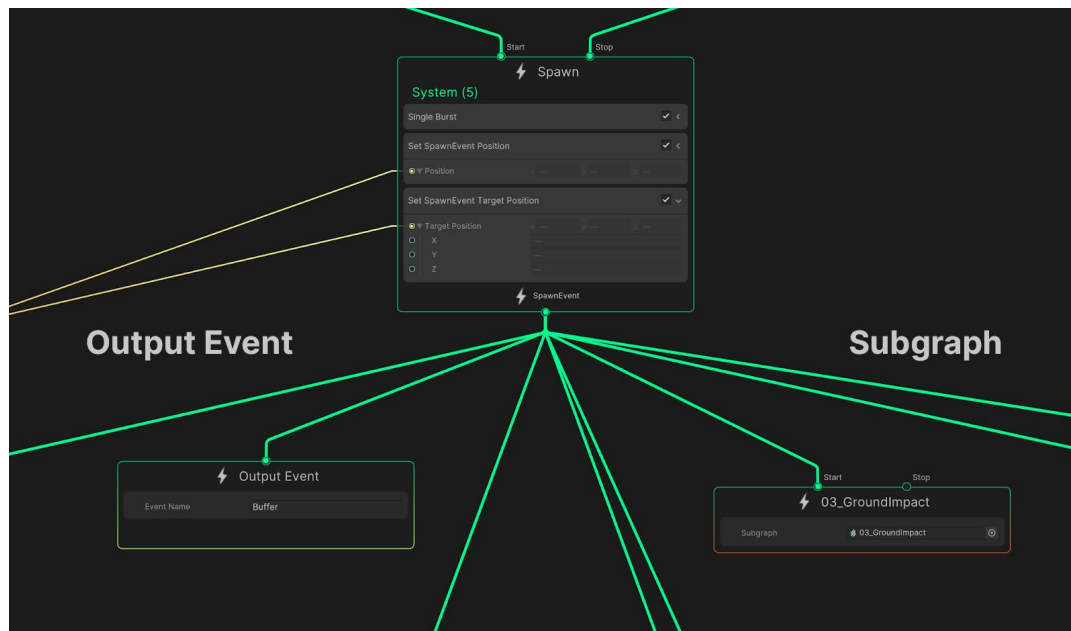
MeteoriteMain 그래프

하늘에서 운석이 떨어질 때마다 효과의 연쇄 반응이 플레이됩니다. 그래프 중앙에는 다음과 같은 여러 효과를 트리거하는 Spawn 컨텍스트가 있습니다.

- 운석 충돌에 앞서 흔들리는 바위 및 광원 효과
- 연기 트레일을 일으키며 충돌하는 운석
- 충돌 시 보는 사람을 놀라게 하는 폭발 광원 애니메이션
- 지면과 상호 작용하는 리지드바디 판자 및 파편
- 충돌의 힘을 더 실감 나게 표현하기 위한 카메라 흔들림

참고: MeteoriteMain 그래프는 많은 부분이 하위 그래프로 나뉘어져 있으므로 비교적 깔끔해 보입니다. 하위 그래프 대다수의 시작 포트에는 생성 이벤트가 직접 연결됩니다. 개별 하위 그래프를 자세히 살펴보며 특별하게 구현된 요소를 살펴보세요.

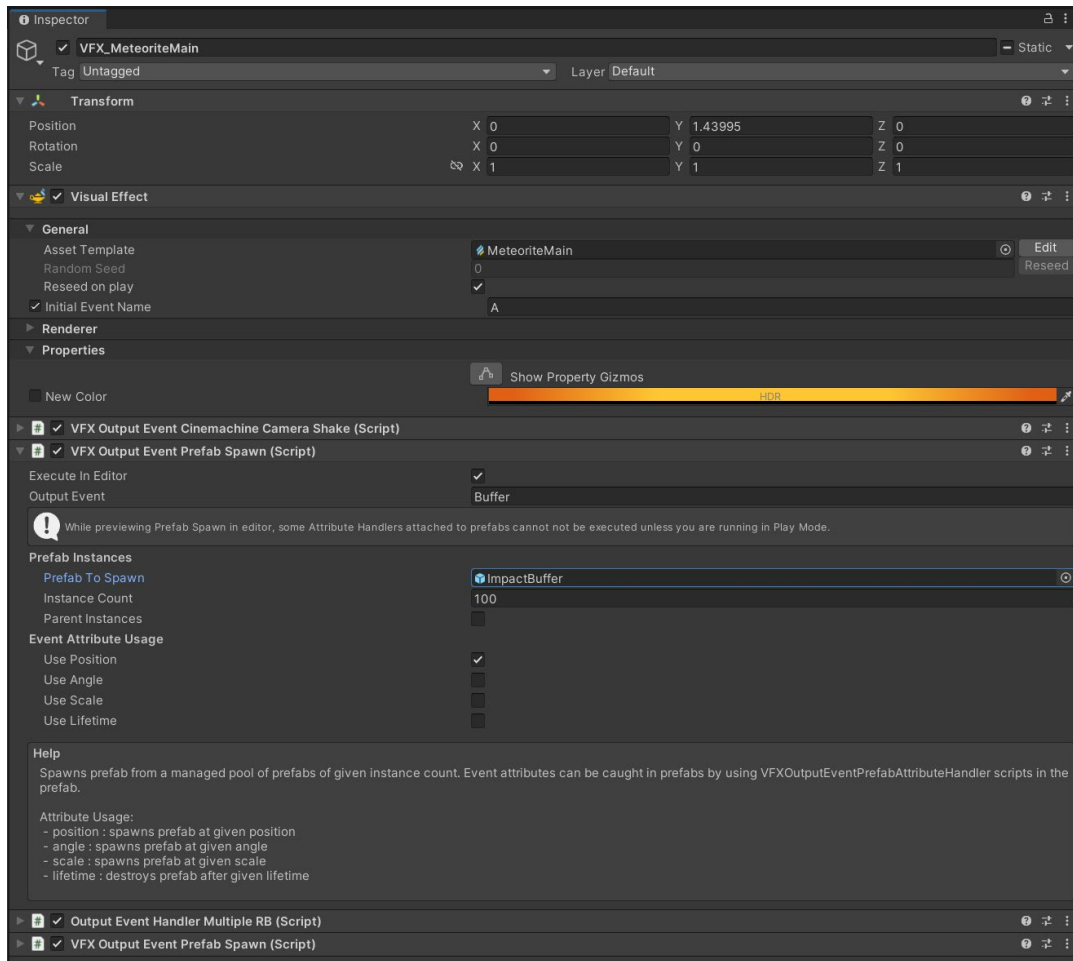
이 구조는 많은 세부 요소를 더 관리하기 편한 작은 부분들로 나눕니다. 따라서 거미줄처럼 혼란스러운 노드를 힘들게 살펴볼지 않아도 쉽게 그래프를 탐색할 수 있습니다.



생성 이벤트는 하위 그래프와 출력 이벤트에 연결됩니다.

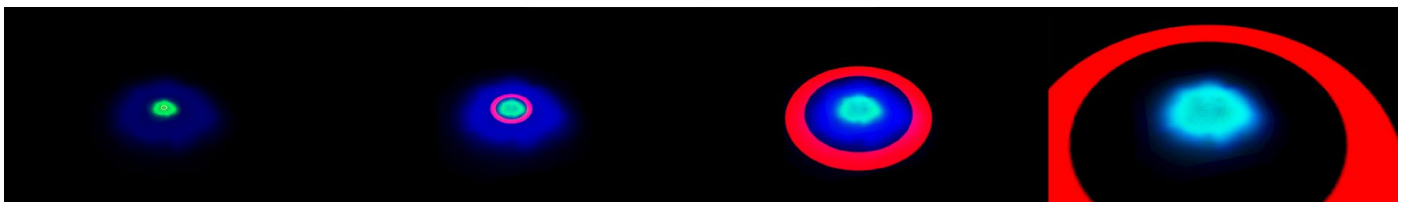
출력 이벤트는 그래프 외부의 다른 컴포넌트와 커뮤니케이션하는 데 사용됩니다. 특히 광원 애니메이션, 카메라 흔들림, 판자 파편에는 자체 출력 이벤트가 있습니다.

VFX_MeteoriteMain 게임 오브젝트를 선택하면 이러한 이벤트를 수신하고 적절하게 응답하는 다양한 출력 이벤트 핸들러 스크립트를 확인할 수 있습니다.



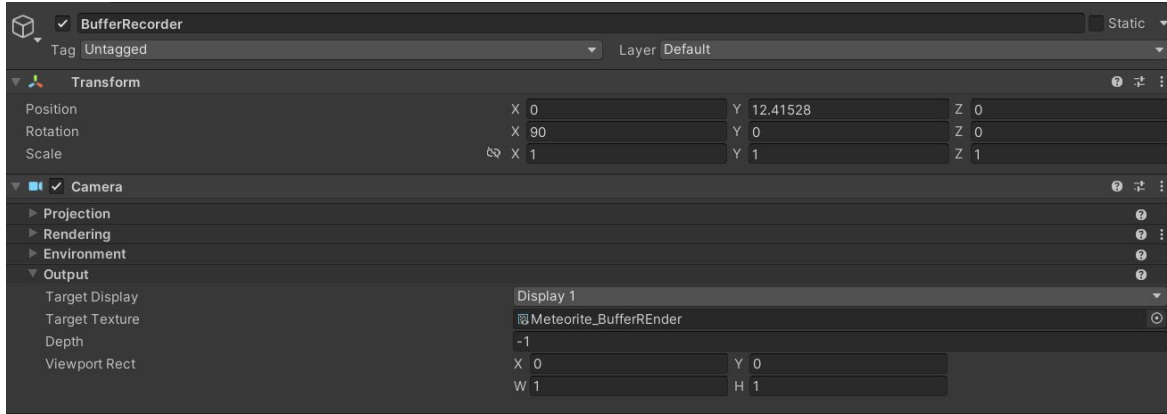
게임 오브젝트의 출력 이벤트 핸들러

한편 그래프의 **버퍼 출력 이벤트**는 자체 VFX Graph가 있는 **ImpactBuffer**라는 프리팹을 생성합니다. 이 프리팹은 빨간색, 파란색, 초록색 컬러 채널로 구분된 지면 데칼을 애니메이션화합니다. ImpactBuffer 효과는 아래 그림처럼 보입니다.



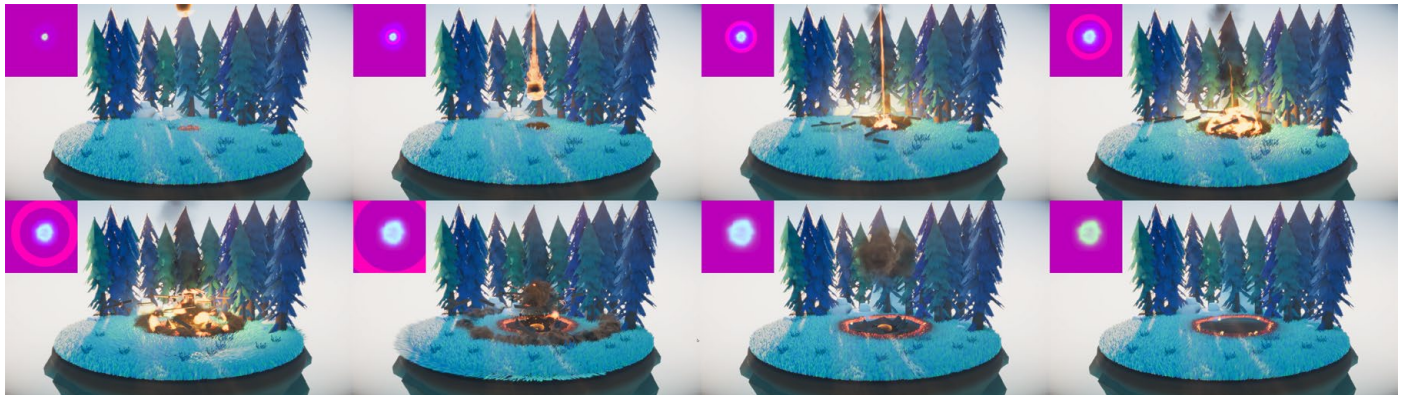
빨간색, 파란색, 초록색의 ImpactBuffer 프리팹

효과의 2D 녹화본만 있으면 됩니다. 아래쪽을 바라보는 **Buffer Recorder**라는 별도의 카메라가 **Meteorite_BufferRender**라는 렌더 텍스처를 생성합니다. 이 텍스처 버퍼는 **GrassStrip**이라는 별도의 그래프로 데이터를 전송합니다.



BufferRecorder

이 텍스처는 GrassStrip 그래프에 연결되어 파티클 스트립의 움직임과 렌더링을 구동합니다. 여기에서 텍스처 버퍼가 어떻게 효과를 만들어 내는지 볼 수 있습니다.

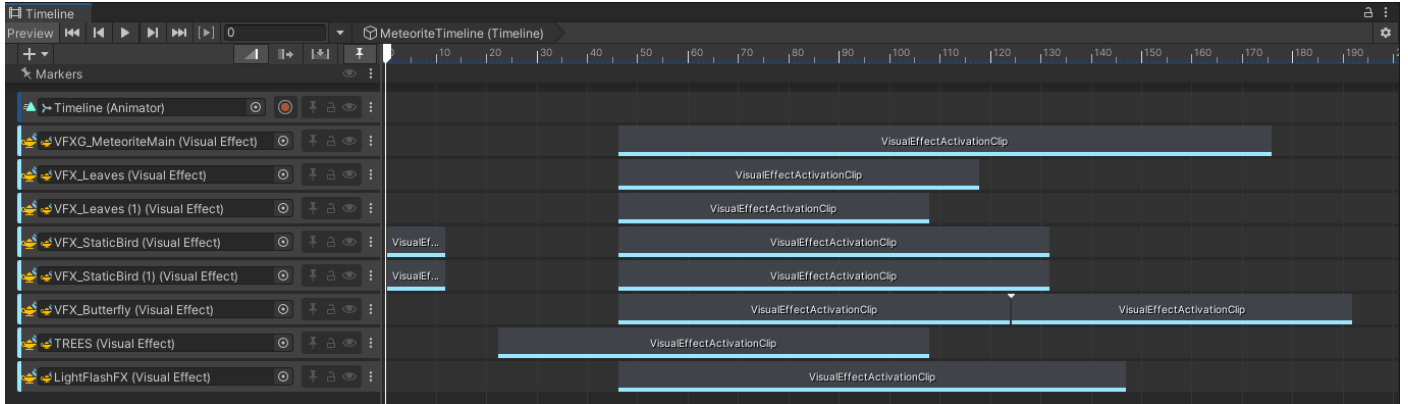


텍스처 버퍼는 잔디 효과를 제어합니다.

떨어지는 운석 외에도 다음과 같은 여러 보조 효과가 있습니다.

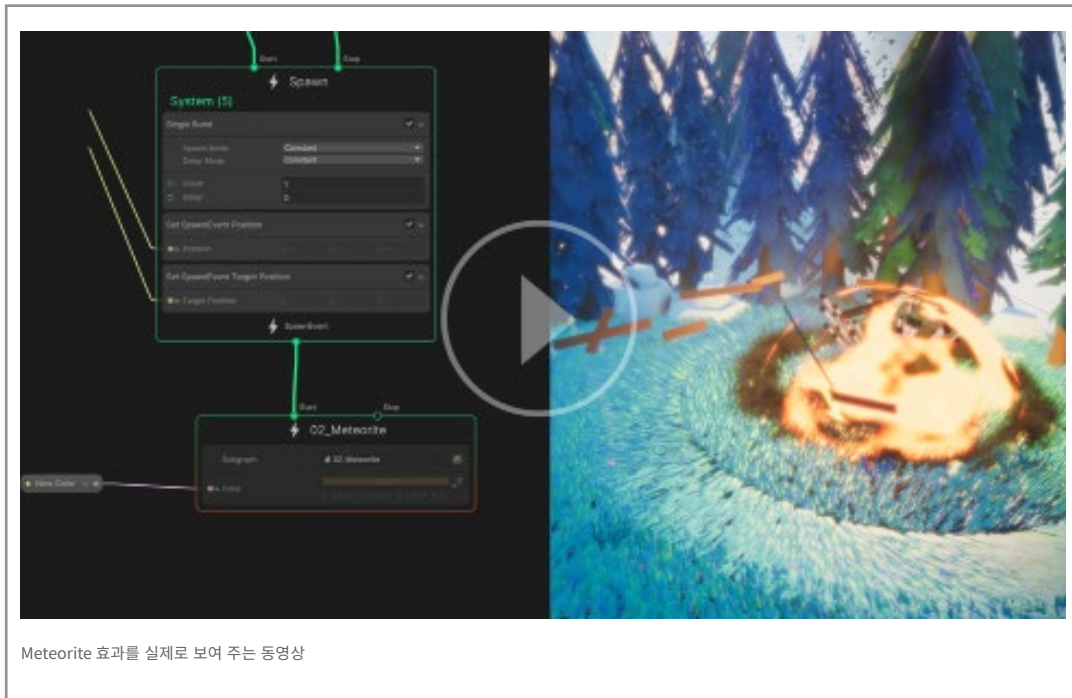
- 나무에서 떨어지는 나뭇잎
- 전경과 배경에서 날아가는 새 무리
- 사라졌다가 다시 나타나는 나비

이러한 보조 효과는 **Meteorite 타임라인**을 통해 구현되며, 이 타임라인은 여러 개의 **시각 효과 활성화 트랙**으로 구성됩니다. MeteoriteControl 스크립트가 **Playable Director**를 시작하면 각 보조 효과가 미리 지정된 타이밍에 맞게 재생됩니다.



Meteorite 타임라인

타임라인 및 출력 이벤트를 설정하는 자세한 방법은 [상호 작용 챕터](#)를 참조하세요.

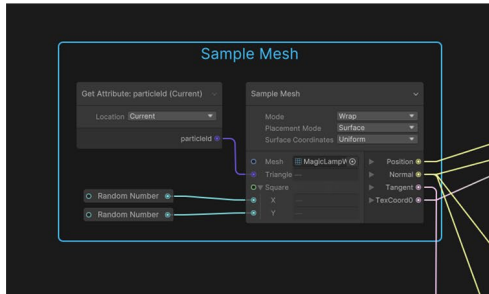


Meteorite 효과를 실제로 보여 주는 동영상

메시 샘플링 효과

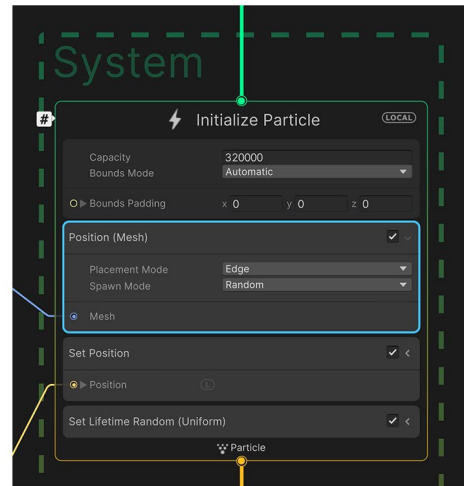
메시 샘플링은 메시에서 데이터를 가져와 그 결과를 그래프에서 사용할 수 있는 실험 단계 기법입니다. 메시를 다음 중 하나로 샘플링할 수 있습니다.

- [Position \(Mesh\) 블록](#)
- [Sample Mesh 연산자](#)



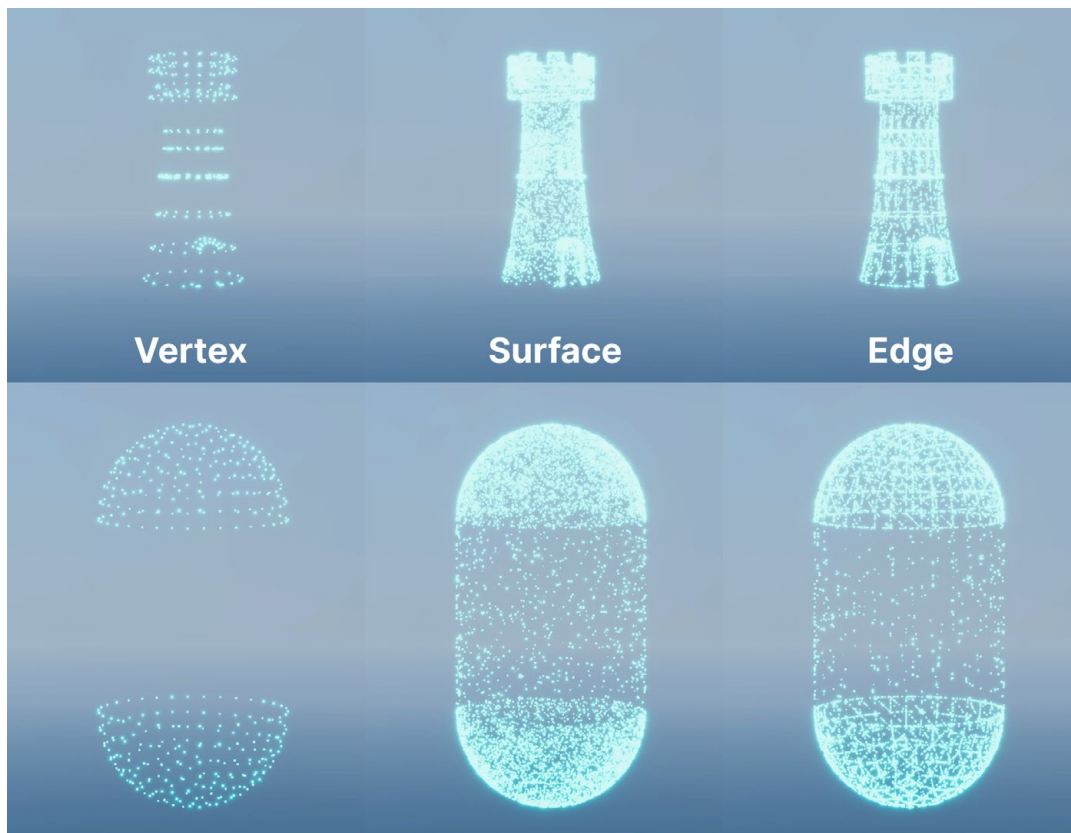
Sample Mesh Operator

Sample Mesh 연산자와 Position (Mesh) 블록



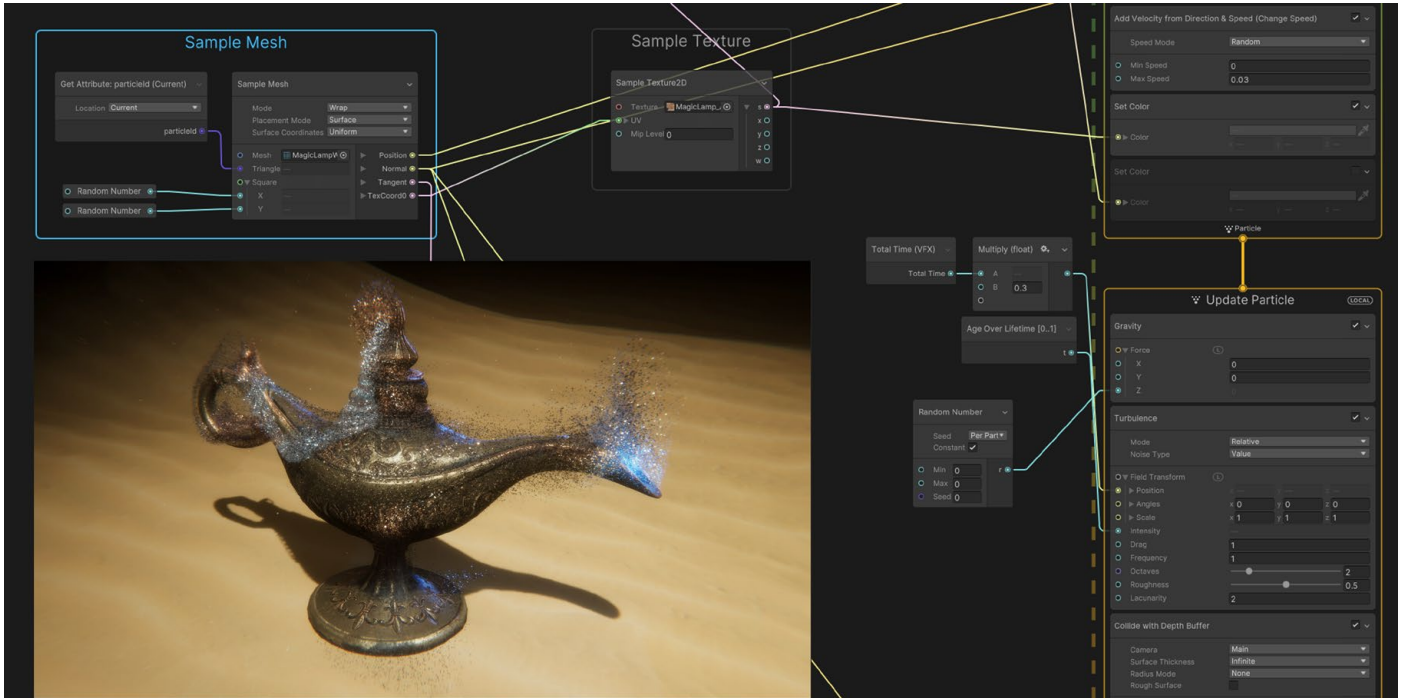
Position (Mesh) Block

Placement Mode는 **Vertex**, **Edge**, **Surface**로 설정할 수 있습니다. 다음은 간단한 메시에서 **Position (Mesh)** 블록을 사용하는 모습입니다.



Vertex, Surface, Edge 샘플링

이 배리에이션은 샘플의 원본 Magic Lamp 씬을 수정합니다. 몇 가지 추가 작업을 하면 **Sample Mesh** 연산자를 사용하여 램프 표면의 파티클을 초기화하고 방향을 바꿔 파티클이 흩어져 사라지는 효과를 낼 수 있습니다. Magic Lamp의 **샘플 텍스처**를 사용하여 파티클 컬러와 메시 표면을 자연스럽게 통합해 보세요.



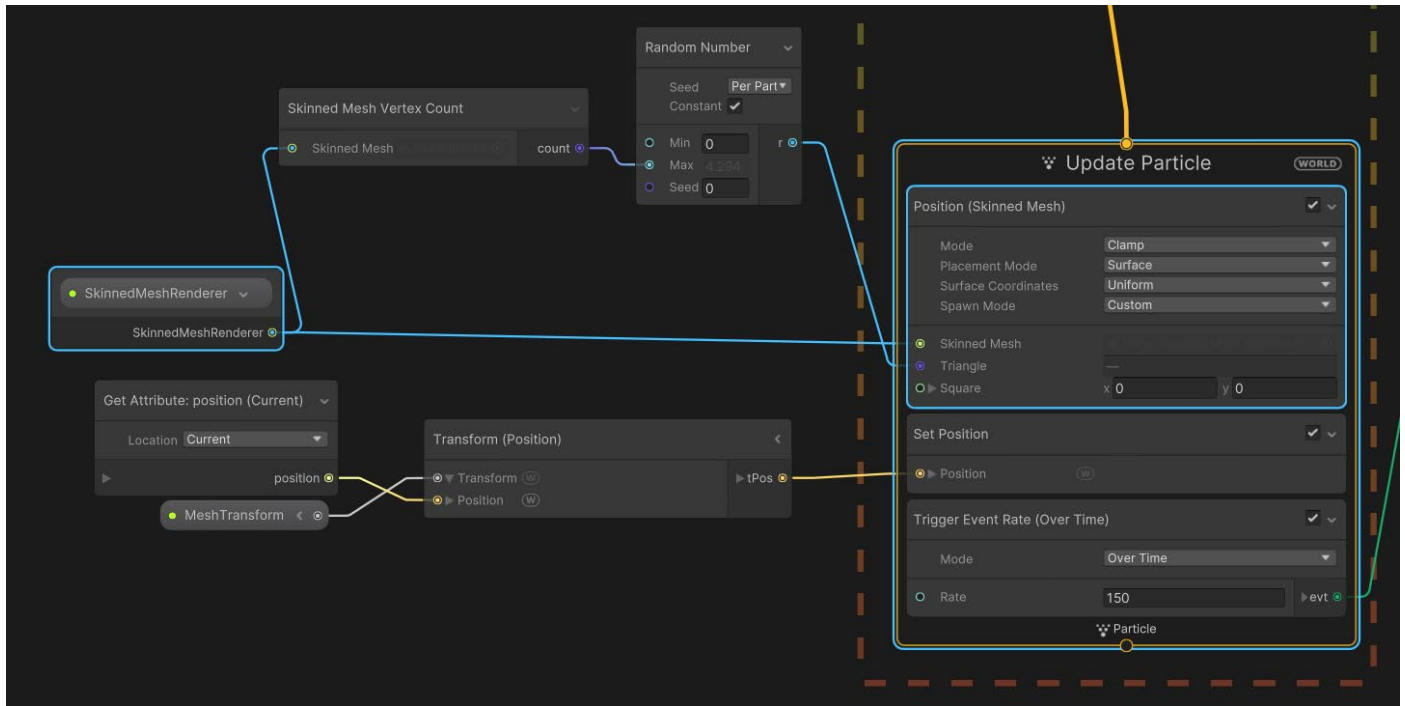
Magic Lamp 샘플 씬

스킨드 메시 샘플링

게임의 캐릭터에 효과를 적용하여 게임플레이에 재미있고 눈길을 끄는 요소를 추가할 수 있습니다. 심장이 꿰뚫려 모래알처럼 흩어지거나 화염에 휩싸이는 뱀파이어를 구현해야 하나요? [스킨드 메시 렌더러](#)를 샘플링하여 두 가지 모두 구현할 수 있습니다.

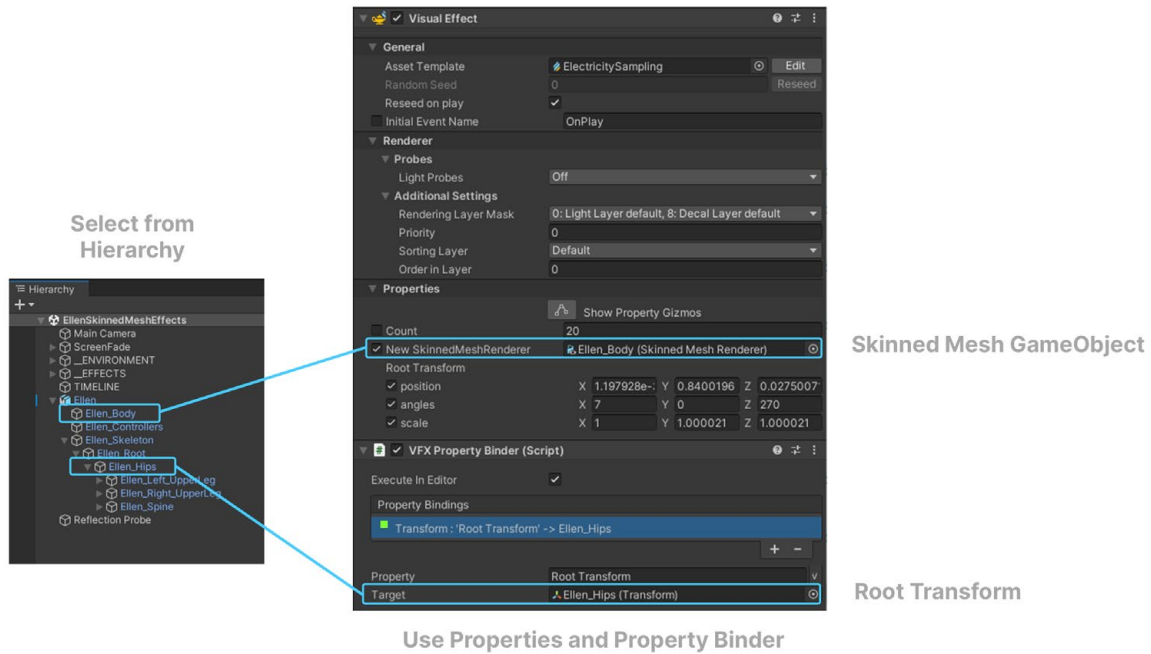
EllenSkinnedMeshEffects라는 샘플 효과는 여주인공과 관련된 다양한 효과를 선보입니다. 하지만 각 사례에서 스킨드 메시 데이터를 샘플링하는 일반적인 프로세스는 다음과 같이 비슷합니다.

- **Position (Skinned Mesh)** 블록을 컨텍스트(원하는 효과에 따라 **Initialize**, **Update**, **Output**)에 추가합니다.
- **Blackboard**에 **Skinned Mesh Renderer** 프로퍼티를 표시하고 스킨드 메시 플로 포트에 연결합니다.
- **Surface, Edge, Vertex** 데이터에 추가로 액세스해야 하는 경우 **Skinned Mesh 연산자**를 사용합니다. 사용 예시는 Hologram 및 Disintegration 그래프를 참조하세요.



그래프에서 Position (Skinned Mesh) 블록 설정하기

- 인스펙터에서 **Property**를 씬의 **Skinned Mesh Renderer**로 설정합니다.
- **프로퍼티 바인더**를 사용([상호 작용 챕터 참조](#))하여 캐릭터의 골격에서 효과의 위치와 방향을 변환할 수 있습니다. 최적의 결과를 위해 **Transform**과 해당 **Set Position Block**을 **World** 공간으로 설정합니다.



Properties 및 VFX Property Binder 설정

또한 골격 자체의 **기본 트랜스폼**을 처리하는 **메시 트랜스폼 프로퍼티 바인더**를 추가해야 합니다.¹ 캐릭터마다 다르지만 이 경우에는 캐릭터의 고관절을 연결합니다. 이제 파티클이 스킨드 메시 렌더러를 따라갈 수 있습니다.



EllenSkinnedMeshEffects 예시

¹ Unity 2021 LTS에 관한 설명입니다. Unity 2022.2 이상에서는 이 단계가 필요하지 않습니다.



캐릭터와 오브젝트에 비슷한 효과를 생성할 때 스킨드 메시 샘플링을 다양한 방식으로 활용하세요. 샘플에서 엘렌 캐릭터는 다음과 같이 묘사됩니다.

- SF 홀로그램으로 투사된 모습
- 몸에 불이 붙고 연기와 불꽃이 발생하는 모습
- 몸이 젖어 피부에서 물방울이 떨어지는 모습
- 뒤쪽에 컬러 잔상을 남기며 매우 빠른 속도로 질주하는 모습
- 전기 충격을 받은 모습
- 재로 변하는 모습

이러한 효과는 저마다 스킨드 메시 렌더러를 샘플링하는 별도의 그래프로 구현됩니다. 구현된 세부 요소를 살펴보면 파티클이 캐릭터 메시와 어떻게 상호 작용하는지 확인해 보세요.

더 많은 예시

VFX Graph 샘플에서 더 많은 예시를 살펴볼 수 있습니다. 프로젝트에 있는 다른 씬도 함께 확인하세요. 각 씬에서 특정 시각 효과를 만드는 다양한 기법을 볼 수 있습니다.

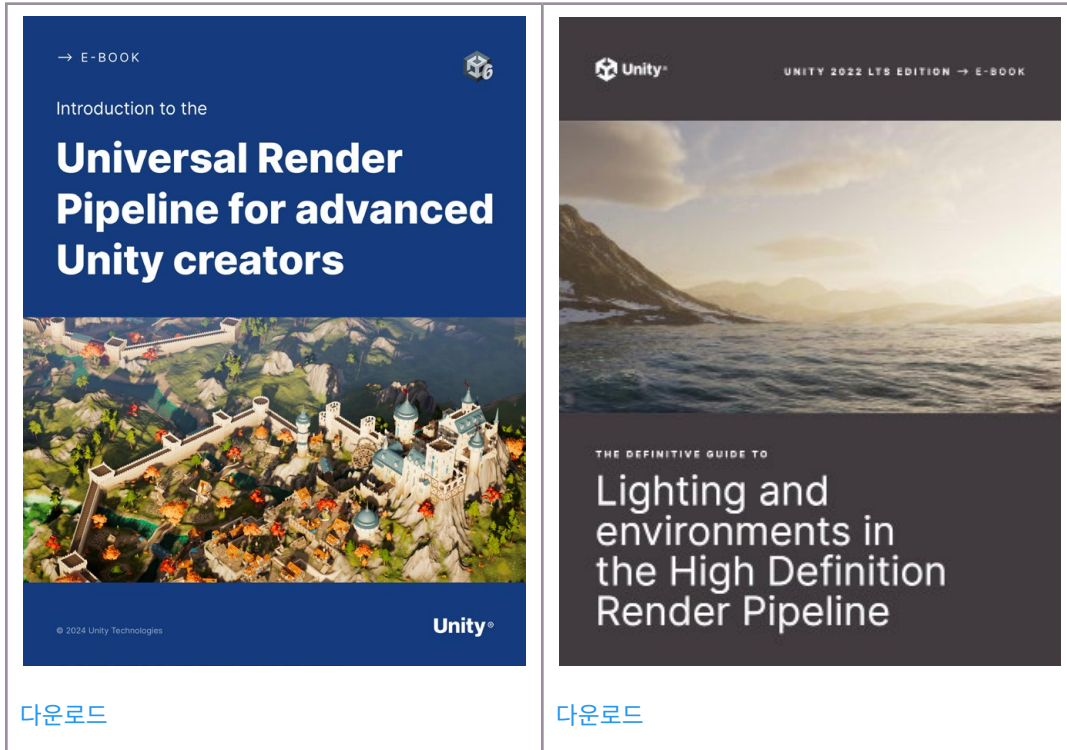
렌더 파이프라인 및 VFX Graph

선택한 렌더 파이프라인은 VFX Graph에서 사용 가능한 출력 컨텍스트에 영향을 줍니다. 디자인 요구 사항과 타겟 플랫폼에 따라 애플리케이션에 가장 적합한 렌더 파이프라인이 최종적으로 결정됩니다.

[URP\(유니버설 렌더 파이프라인\)](#)는 저사양 모바일부터 고사양 콘솔 및 PC에 이르기까지 광범위한 디바이스의 성능에 최적화되어 있습니다. 단순한 조명과 싱글 패스 포워드 렌더링으로 간소화된 기능 세트를 제공하기도 합니다. URP는 VFX Graph를 지원하지만 일부 고급 기능은 제한되거나 사용이 불가할 수 있습니다.

[HDRP\(고해상도 렌더 파이프라인\)](#)는 복잡한 조명과 셰이딩을 처리하는 디퍼드 렌더링 경로를 사용하여 고사양 PC 및 콘솔에 최적화되어 있습니다. HDRP에는 레이트레이싱, 볼류메트릭 조명, 피하 산란, 스크린 공간 반사 같은 고급 기능이 포함되어 있습니다. 하지만 HDRP는 높은 성능 오버헤드와 제한된 플랫폼 지원으로 인해 저사양 디바이스 및 모바일 플랫폼에는 적합하지 않습니다.

모든 렌더 파이프라인 기능은 [여기](#)에서 비교해 볼 수 있습니다.



디바이스 지원

VFX Graph는 디바이스 호환성을 위해 컴퓨트 셰이더가 필요합니다. 모바일 디바이스의 컴퓨트 지원은 브랜드, 모바일 GPU 아키텍처, 운영 체제에 따라 매우 다양합니다. 사용 중인 플랫폼이 컴퓨트 셰이더를 지원하지 않는다면 Unity [빌트인 파티클 시스템](#)을 사용하는 것이 좋습니다.

렌더 파이프라인 호환성에 대한 자세한 내용은 [Unity VFX Graph 시스템 요구 사항](#)을 참조하세요.

URP용 VFX Graph

VFX Graph는 컴퓨팅이 가능한 데스크톱, 모바일, XR 플랫폼에 유니버설 렌더 파이프라인을 지원합니다.

참고: URP에서 Visual Effect Graph는 [감마 색 공간](#)을 지원하지 않습니다.

Unity 6에는 이전에 HDRP에서만 사용할 수 있었던 여러 기능의 URP(유니버설 렌더 파이프라인) 지원이 추가되었습니다.

릿 출력

VFX Graph는 이제 URP에서 릿 출력을 지원합니다. 릿 출력을 사용해 씬의 조명에 직접 반응하는 효과를 만들 수 있습니다.



URP에서 이제 릿 출력을 지원합니다.

이제 VFX Graph로 URP 데칼을 생성하고 Shader Graph를 사용하여 HDRP 및 URP에 모두 데칼을 커스터마이징할 수 있습니다.

2D 렌더러 및 Shader Graph 지원

VFX Graph는 파티클을 스프라이트로 렌더링하는 데 사용할 수 있는 Shader Graph의 2D 하위 타겟을 지원합니다. 새로운 하위 타겟을 찾으려면 **Create > Shader Graph > URP**로 이동합니다. 이 기능은 유니버설 렌더 파이프라인 전용 기능입니다.



URP 2D 렌더러를 사용한 VFX Graph의 비 효과

데칼

이제 VFX Graph로 URP 데칼을 생성하고 Shader Graph를 사용하여 HDRP 및 URP에 맞게 커스터마이징할 수 있습니다. 데칼은 총알 자국, 발자국, 표면 손상 또는 기타 동적 효과를 환경에 추가하는 데 사용할 수 있습니다.

6방향 연기 조명

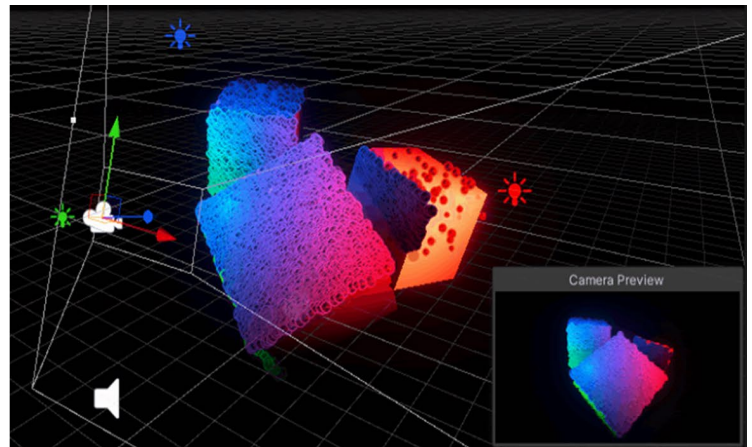
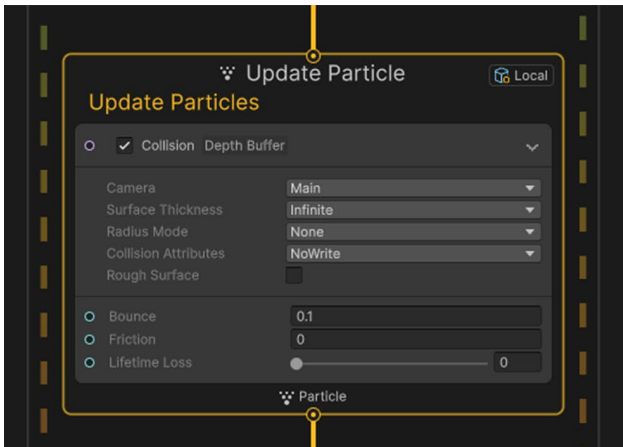
사실적인 연기 효과를 위한 6방향 조명은 SideFX의 Houdini, Blender, JangaFX의 EmberGen 같은 툴의 라이트맵으로 커스터마이징할 수 있습니다. 이 조명 기법을 사용하면 높은 연산 비용을 들이지 않고도 실시간 동적 조명으로 연기 파티클을 통해 볼류메트릭 연기 효과를 거의 정확하게 구현할 수 있습니다. 화재, 폭발 또는 연기와 유사한 기타 효과에 6방향 조명을 사용하면 씬 조명이 파티클 시뮬레이션을 환경에 통합하는 데 도움이 될 수 있습니다.

모션 벡터

VFX Graph 파티클은 URP에서 모션 벡터를 생성할 수 있으며, 이는 TAA(Temporal Anti-Aliasing) 또는 모션 블러 같은 효과에 유용합니다. URP는 불투명 파티클만 지원하지만 HDRP에서는 투명과 불투명을 모두 사용할 수 있습니다.

카메라 버퍼

이제 URP 카메라 버퍼를 샘플링하여 씬의 텍스처와 컬러를 얻을 수 있습니다. 이 기능을 사용하면 GPU에서 빠른 충돌을 수행하거나 텍스처 버퍼에 파티클을 생성하고 씬 컬러를 상속할 수 있습니다. VFX 효과와 액체가 터지는 효과, 파티클로 분해되는 캐릭터, 기본 오브젝트에 따라 컬러가 변하는 효과를 만들 수 있죠.



카메라 버퍼는 GPU에서 충돌을 수행할 수 있습니다.



HDRP용 VFX Graph

레이트레이싱 및 볼류메트릭 렌더링과 같은 일부 VFX Graph 기능은 HDRP에서만 사용할 수 있습니다. 애플리케이션에 고사양 그래픽스가 필요한 경우 다음과 같은 HDRP 전용 기능을 고려해 보세요.

레이트레이싱 지원

고해상도 렌더 파이프라인에서 다음의 레이트레이싱 기능과 함께 VFX Graph 파티클을 사용하세요.

- **레이트레이싱 반사:** [스크린 공간 반사](#)의 대안으로 화면 외부 데이터를 사용하여 보다 정확한 반사를 구현할 수 있습니다.
- **레이트레이싱 그림자:** 레이트레이싱은 기존 색도우 맵을 대체하여 더욱 정확하고 사실적인 그림자를 구현합니다.
- **레이트레이싱 앰비언트 오클루전:** 화면 외부 데이터를 사용할 수 있는 보다 정확한 레이트레이싱 솔루션으로, HDRP의 스크린 공간 앰비언트 오클루전의 대안입니다.
- **레이트레이싱 전역 조명:** 표면에서 반사되는 광원을 고려하여 간접광을 보다 정확하게 표현합니다.

RTX 지원으로 레이트레이싱 패스에서 VFX를 렌더링하는 기능이 추가되어 레이트레이싱 반사에서 VFX를 고려할 수 있게 됩니다.



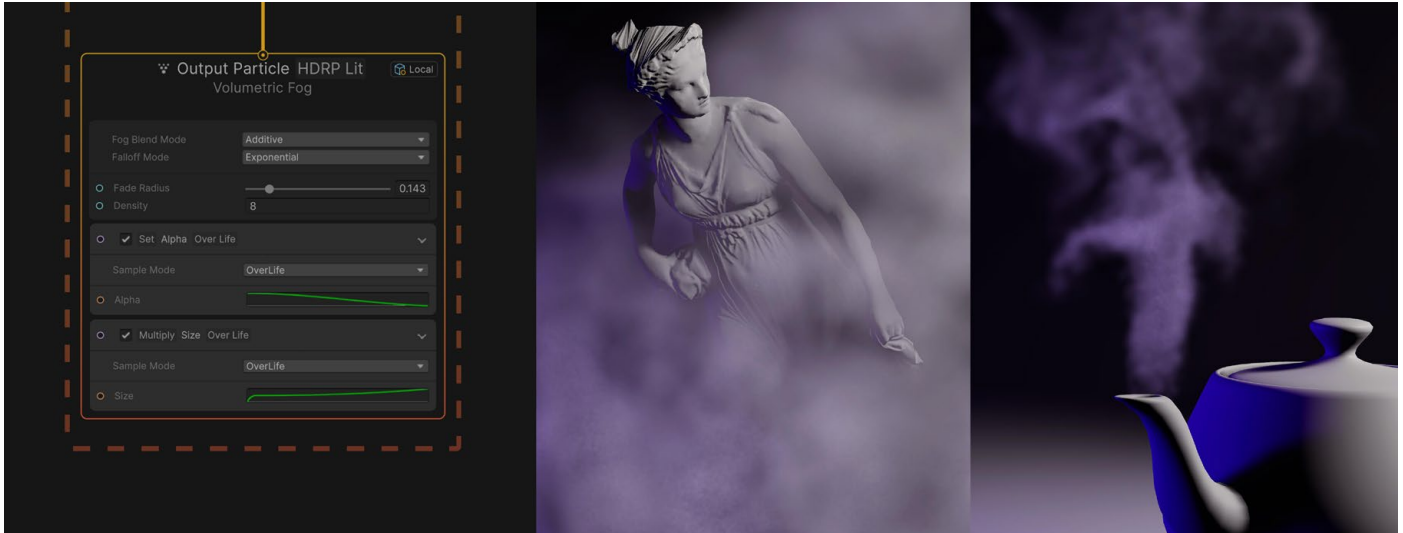
RTX 지원으로 레이트레이싱 패스에서 VFX Graph를 렌더링하는 기능이 추가됩니다.

사각형, 삼각형, 팔각형을 사용하는 VFX Graph에서 레이트레이싱을 활성화할 수 있습니다. VFX Graph는 메시 또는 스트립을 사용한 레이트레이싱을 지원하지 않습니다.

HDRP 씬에서 레이트레이싱을 사용하려면 [레이트레이싱 시작하기](#) 페이지를 참고하세요.

볼류메트릭 안개 출력

Output Particle HDRP Volumetric Fog 출력 노드는 HDRP에서 **로컬 볼류메트릭 안개**를 샘플링합니다. 이를 사용하여 역동적인 구름, 연기, 화재 효과를 시뮬레이션하거나 더욱 역동적인 볼류메트릭 안개를 만들 수 있습니다.



HDRP는 볼류메트릭 안개 출력을 지원합니다.

VFX Graph와의 렌더 파이프라인 호환성에 대한 자세한 내용은 Unity [그래픽스 제품 로드맵](#)을 참조하세요.

Shader Graph 연동

시각 효과는 현실 세계의 규칙을 거스르는 경우가 많기 때문에, 표준 URP 및 HDRP 셰이더가 제공하는 것을 넘어서 고유한 셰이딩과 렌더링이 필요합니다. SF 장르의 초자연적 필드나 신비로운 마법 오라 같은 경우 현실 세계에 대응하는 것이 없기 때문에 미리 준비된 셰이더로는 이를 묘사하기 어려울 수 있죠. 이러한 경우 [Shader Graph](#)를 사용하여 셰이더를 커스터마이징할 수 있습니다.

Shader Graph 연동의 주요 장점 중 하나는 파티클 단위로 셰이더 동작을 구동할 수 있다는 점입니다. 이를 통해 파티클마다 다른 값으로 배리어이션, 컬러 무작위화 및 기타 동적 효과를 생성하여 매우 복잡한 비주얼을 구현할 수 있습니다.

빌트인 출력

커스텀 셰이더를 살펴보기 전에, Unity는 다음과 같이 여러 렌더 파이프라인에 최적화된 다양한 빌트인 출력 컨텍스트를 제공한다는 점을 알아 두세요.

- **HDRP/URP Lit Output:** 이 출력은 사실적인 조명과 셰이딩으로 파티클을 렌더링하는 데 최적화되어 있습니다. 릿 출력은 시각 효과가 씬의 조명에 반응해야 하는 시나리오에서 유용합니다.
- **Unlit Output:** 이 출력은 조명이 없는 단순 효과용이며 씬 조명과 상호 작용하지 않습니다. 조명 계산이 필요 없는 스타일라이즈드 또는 2D 같은 효과에 사용합니다.
- **Six-Way Smoke Lit:** HDRP Lit 및 URP Lit 출력용 Material Type 옵션입니다. 이 출력은 [6방향 조명 기법](#)을 지원하여 사실적인 볼류메트릭 같은 셰이딩을 구현할 수 있습니다.

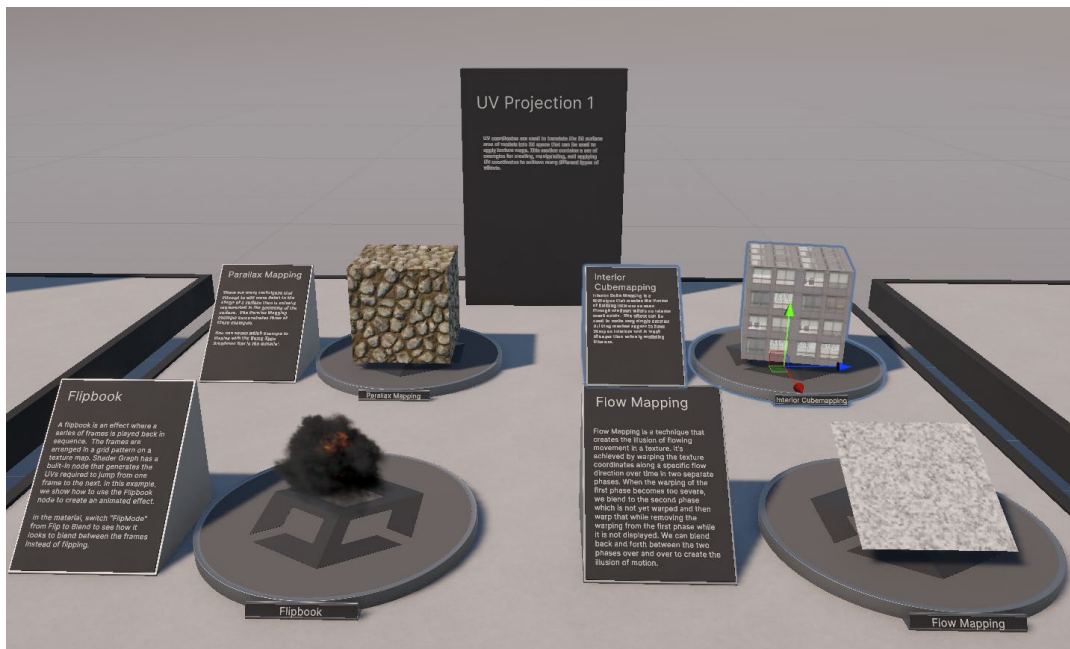
- **HDRP Distortion Output:** 이 HDRP 출력은 열로 인한 아지랑이, 잔물결, 유리 굴절 같은 현상을 시뮬레이션하여 배경을 와핑 또는 왜곡하는 효과를 생성합니다.
- **Decals Output:** 씬의 표면에 텍스처를 투사하는 데 사용되는 URP 및 HDRP용 출력이며 총알 구멍, 그래피티, 표면 마모 같은 디테일을 기존 지오메트리에 직접 추가할 수 있습니다.

이 출력에는 프레임 블렌딩, UV 플립북, 이미시브 프로퍼티 등의 기능이 포함된 통합 셰이더 기능이 함께 제공됩니다. 따라서 커스텀 셰이더를 처음부터 만들지 않고도 정교한 효과를 구현할 수 있습니다.

셰이더 효과

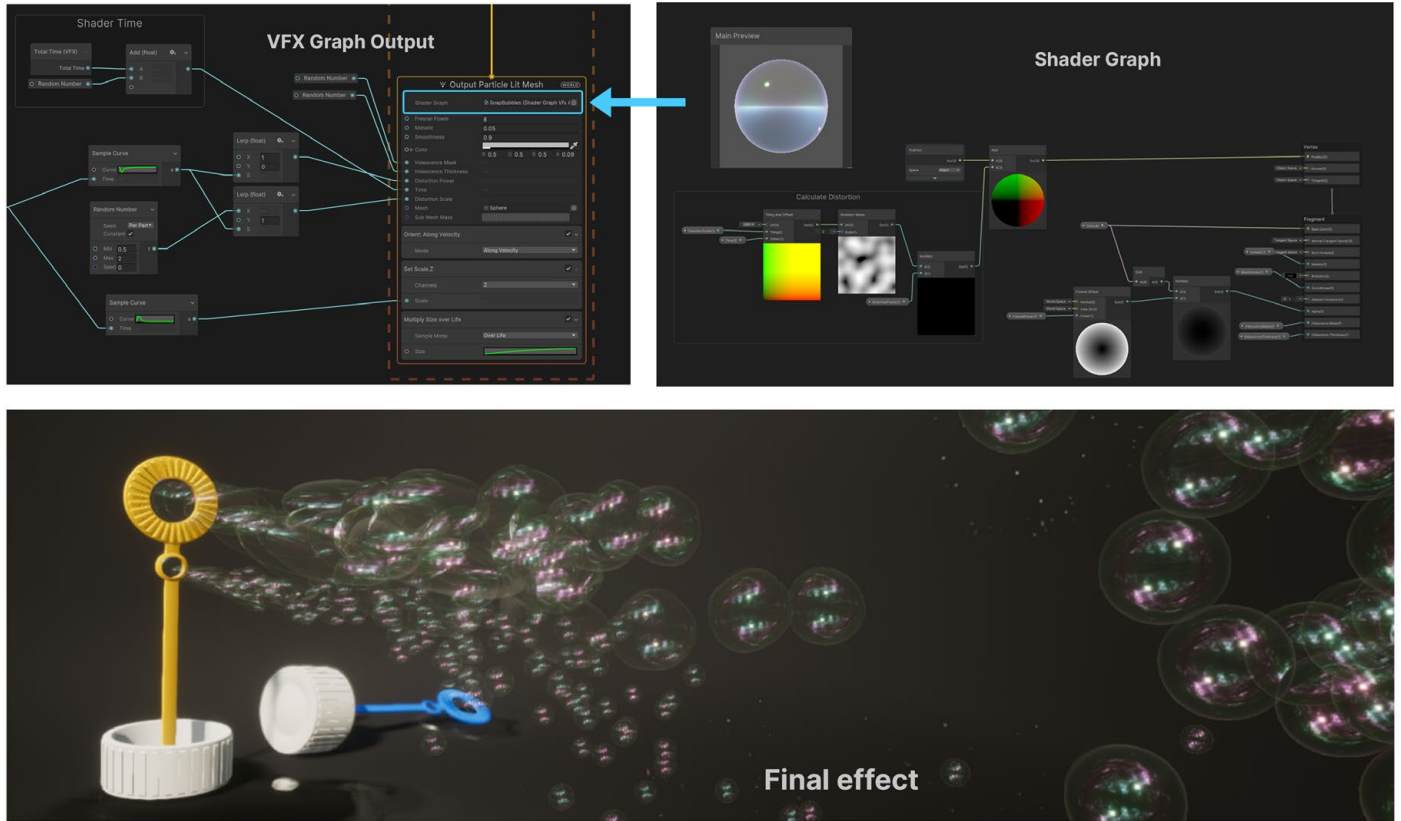
테크니컬 아티스트는 Shader Graph의 그래프 연결망으로 커스텀 셰이딩을 만들 수 있습니다. Shader Graph와 셰이더 저작은 그 자체로 하나의 하위 주제지만, 셰이더에 대한 실무 지식이 있으면 VFX Graph 활용을 보충할 수 있습니다. 셰이더를 사용하면 광원과 컬러를 조작하여 효과를 한층 더 강화할 수 있습니다.

Shader Graph를 사용하면 UV를 와핑 및 애니메이션화하거나 표면의 외형을 점차적으로 변경할 수 있습니다. 셰이더는 전체 화면 이미지 필터처럼 작동하거나 월드 위치, 노멀, 카메라 거리 등에 따라 오브젝트의 표면을 변경하는 데 유용하게 사용할 수 있습니다. Shader Graph의 시각적 인터페이스를 실시간 피드백 삼아 더 빠르게 반복 수정할 수 있습니다.



셰이더는 UV를 사용하여 표면의 외형을 변경할 수 있습니다.

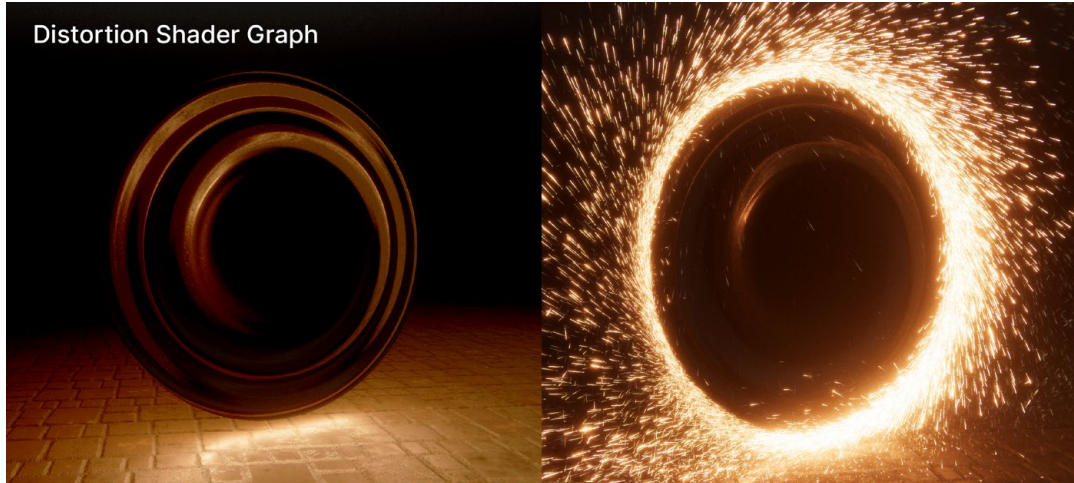
셰이더를 사용하면 깜박이는 불빛이나 통과하는 오브젝트에 반응하는 물과 같은 동적 효과를 만들 수 있습니다. 구름, 연기, 유체 같은 자연 현상의 경우 셰이더를 사용하면 복잡하고 반복적이지 않은 패턴을 만들어 깊이감과 배리어이션을 더할 수 있습니다.



HDRP의 Shader Graph 연동

셰이더가 VFX Graph를 보완하는 몇 가지 다른 방법은 다음과 같습니다.

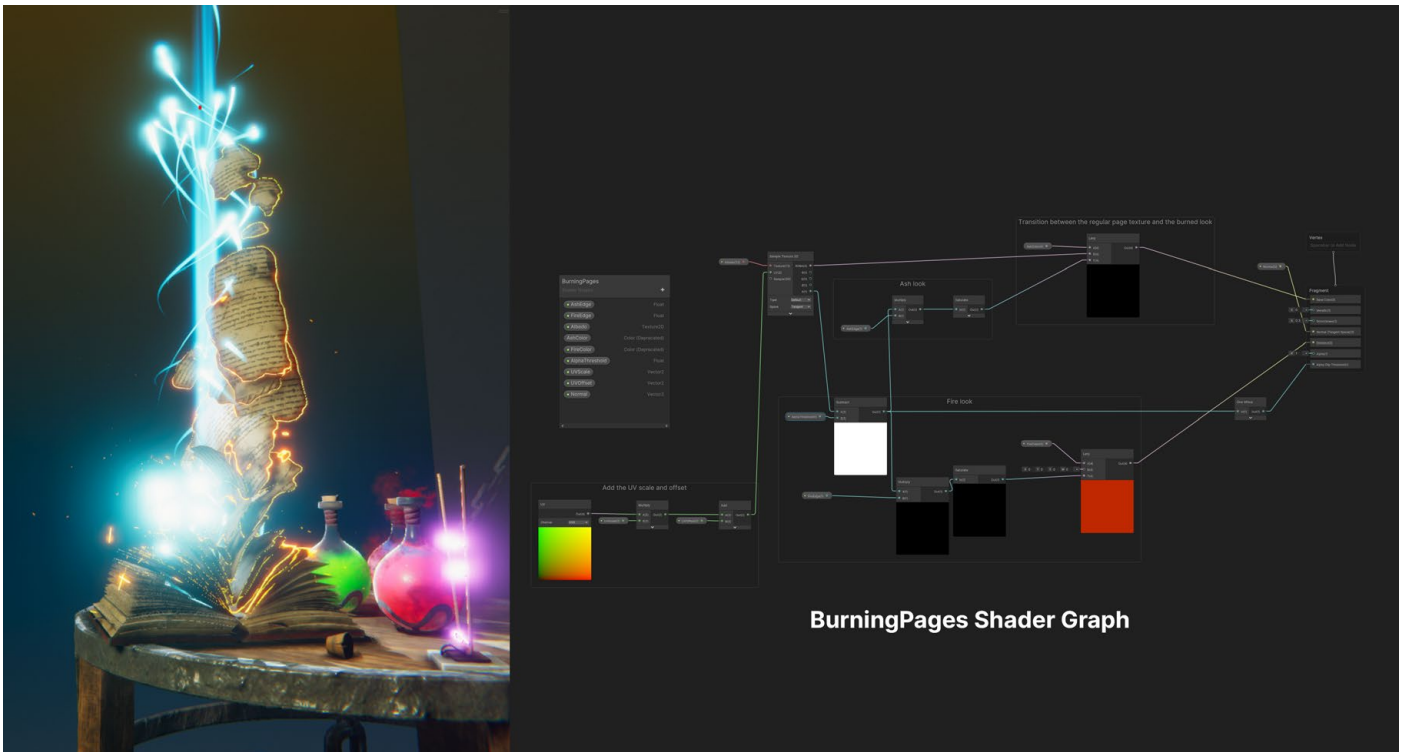
- **왜곡 효과:** 셰이더는 픽셀을 조작하여 광원의 굴절을 시뮬레이션할 수 있습니다. 모닥불이나 제트 엔진과 같은 열원 위에 물결 모양 효과를 만들려면 왜곡 셰이더를 사용합니다. 왜곡을 사용하면 물이나 다른 액체의 잔물결 모양을 모방할 수도 있습니다.
- 예를 들어 [Visual Effect Graph 샘플](#)의 포털 효과는 Shader Graph를 사용하여 포털 내부의 중심을 왜곡합니다.



셰이딩은 광원의 굴절을 시뮬레이션할 수 있습니다.

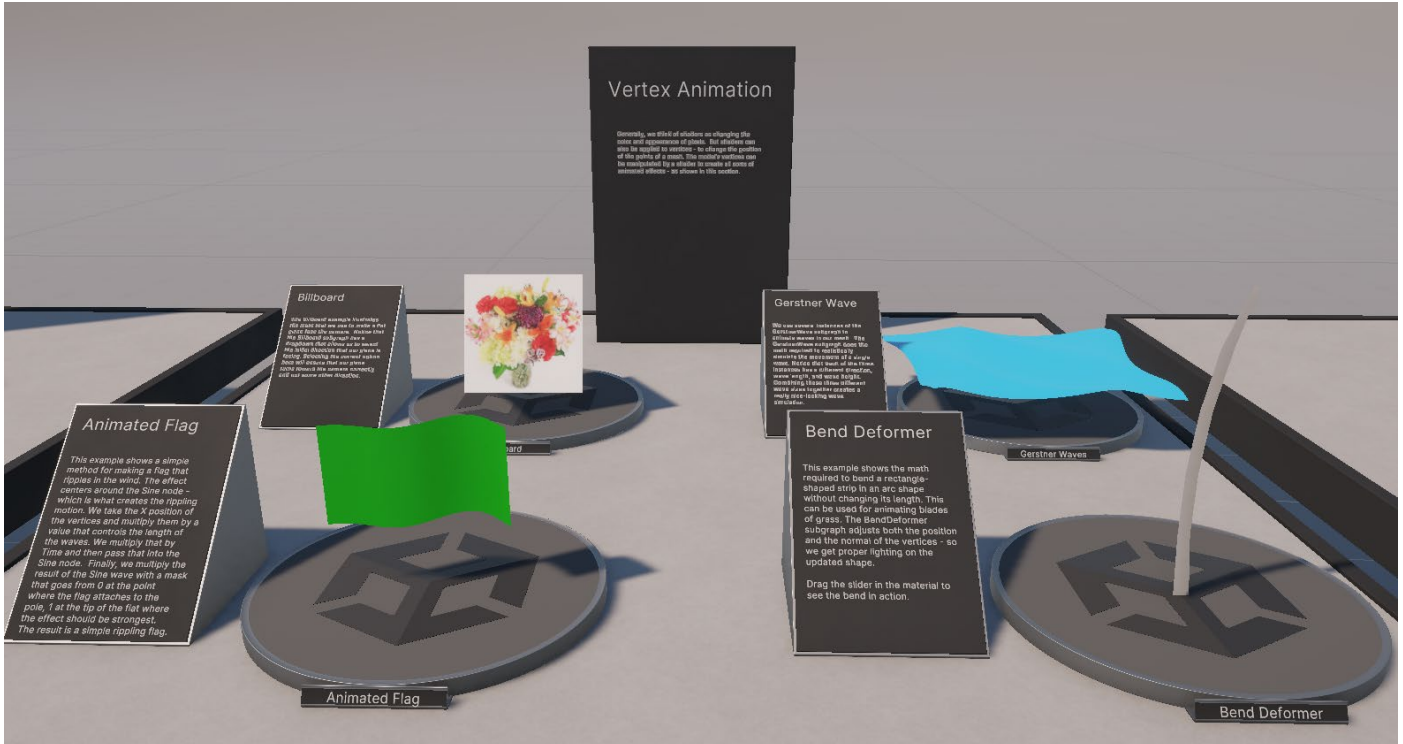
- **알파 침식:** 오브젝트나 텍스처를 분해하는 것은 텍스처를 불균일하게 페이드인/아웃 처리하여 무언가를 연기나 재로 사라지게 하거나 숨겨진 구조물이 드러나는 효과를 만드는 일반적인 방법입니다. 순간 이동, 홀로그램 또는 분해할 수 있는 다른 모든 요소를 생각해 보세요.

Magic Book 샘플의 경우, 날아다니는 페이지에 특수 셰이더를 적용하여 종이가 불씨로 사라져 버리게 만들었습니다.



셰이더로 분해 효과를 만들 수 있습니다.

- **버텍스 애니메이션:** 버텍스 애니메이션으로 유기적인 움직임이나 역동적인 표면을 시뮬레이션하세요. 버텍스 셰이더는 메시 버텍스를 움직여 GPU에서 직접 애니메이션을 만들 수 있습니다. 예를 들어 하나의 메시가 다른 메시로 변하도록 만들거나, 파티클 스트립을 호(arc)로 구부리거나, 표면이 흔들리는 깃발이나 불덩이의 꼬리처럼 물결치도록 만들 수 있습니다.

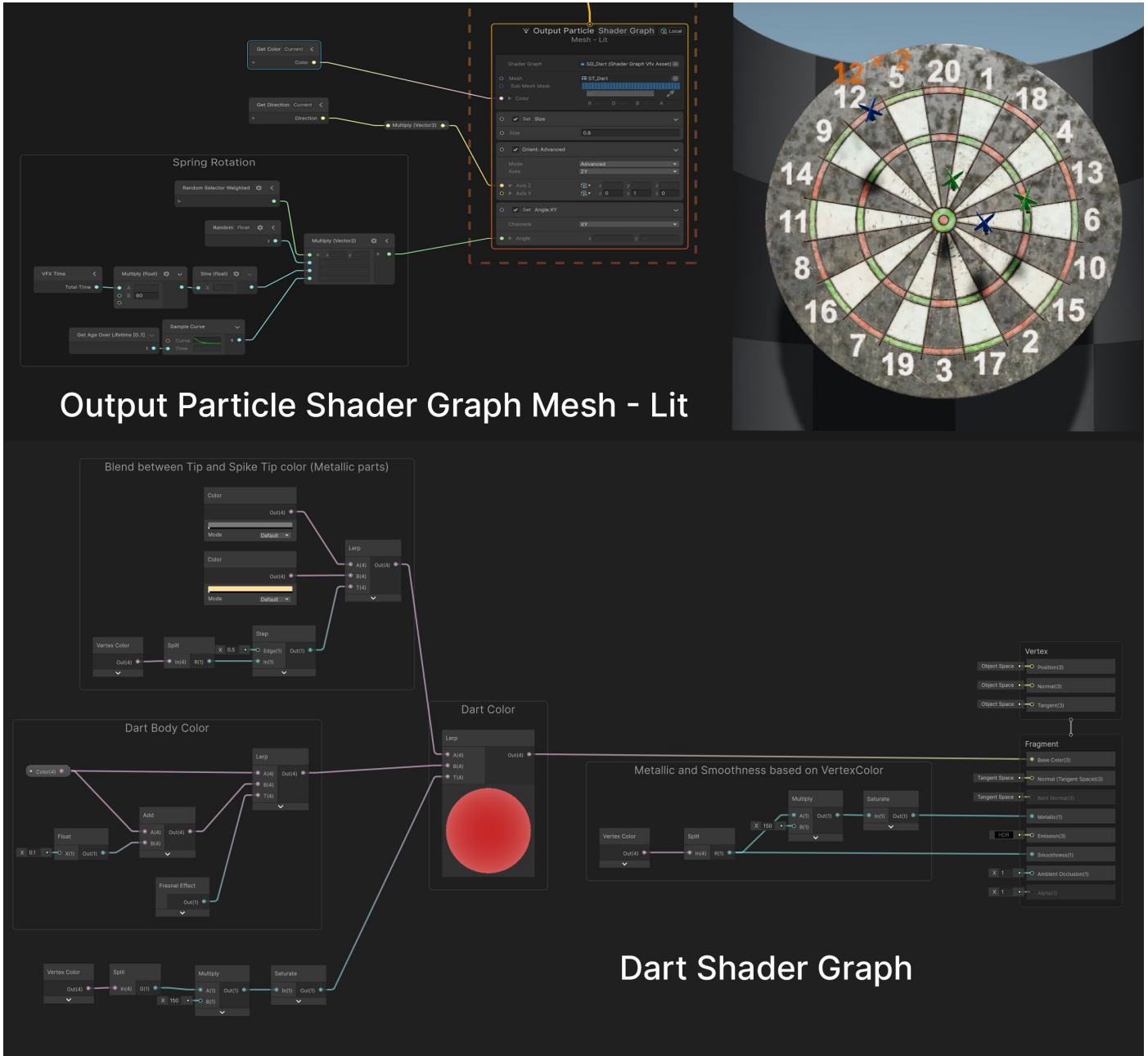


버텍스 애니메이션은 유기적인 움직임을 시뮬레이션할 수 있습니다.

- **조명 및 셰이딩:** 고급 조명 효과로 사실감이나 스타일을 추가합니다. VFX Graph는 빛나는 셰이더 또는 반사, 굴절, 그림자와 같은 복잡한 조명 상호 작용과 함께 사용하면 더 효과적인 경우가 많습니다.

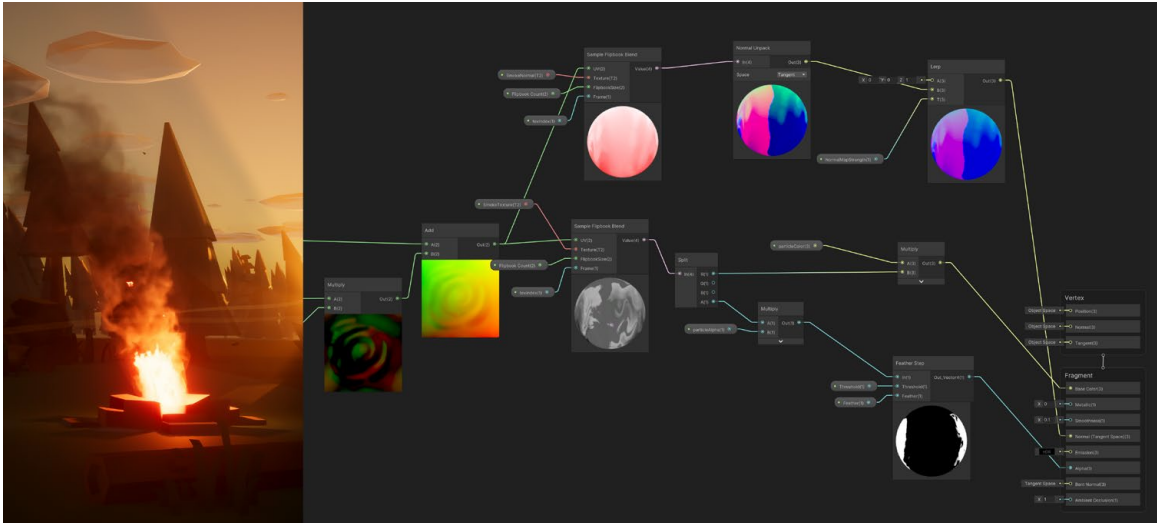
예를 들어 유리, 물, 금속을 모방하는 마법 효과를 구현하려는 경우 셰이더가 정확한 반사를 시뮬레이션해야 합니다. 툰 셰이딩 게임을 제작하시나요? Shader Graph를 사용하면 애니메이션 같은 스타일라이즈드 효과를 렌더링할 수 있습니다.

이 예시에서 다트는 Shader Graph를 사용하여 릿 메시를 탄력 있는 발사체처럼 렌더링하는 파티클입니다.



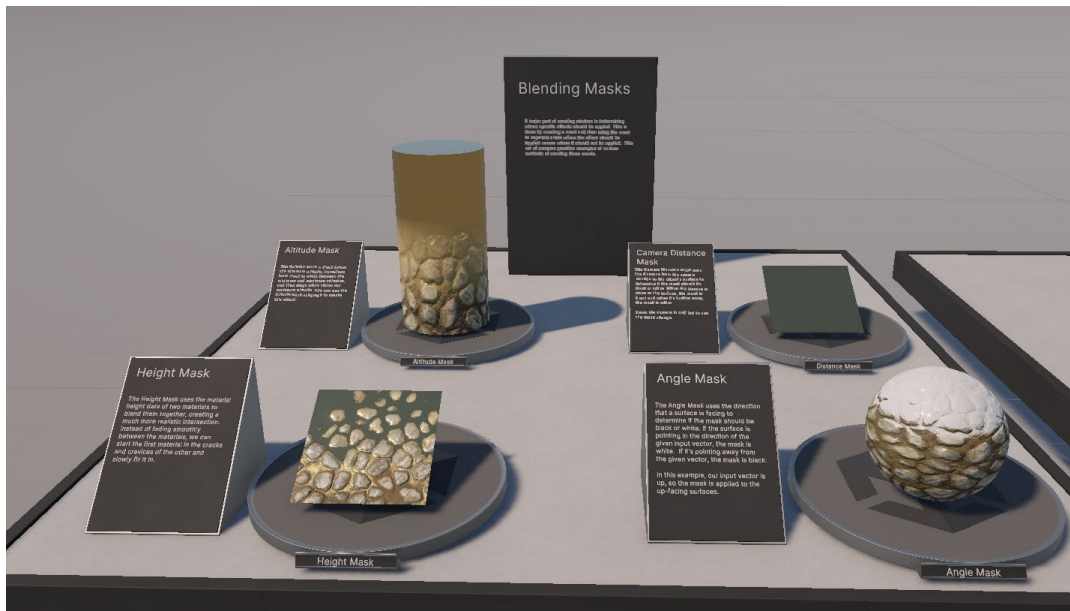
셰이딩은 메시 파티클을 개선합니다.

- **컬러 전환:** 셰이더는 시간, 위치, 사용자 입력에 따라 컬러를 보간할 수 있습니다. 램프 또는 그레디언트를 사용하여 전체 표면이나 애니메이션 시퀀스에서 컬러를 부드럽게 전환해 보세요. 가장 뜨거운 중심부의 파란색에서 바깥쪽 레이어의 빨간색과 노란색으로 바뀌는 불꽃처럼 더 역동적인 효과를 만들 수 있습니다.



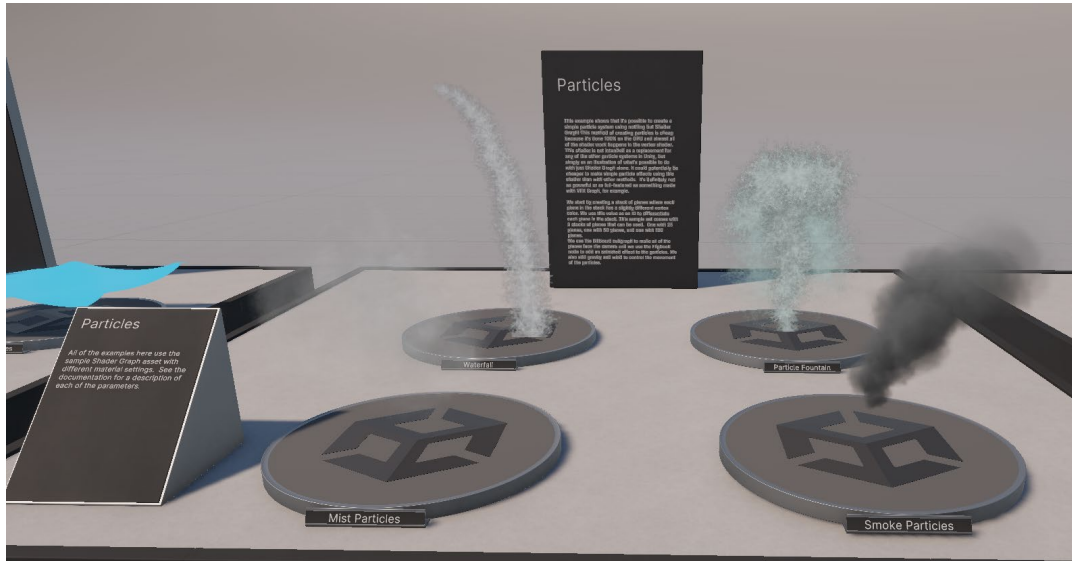
컬러 전환을 사용하면 더 역동적인 효과를 만들 수 있습니다.

- **마스크와 투명도 블렌딩:** 여러 수준의 투명도를 적용해야 하는 효과는 커스텀 셰이딩을 활용해 볼 수 있습니다. Shader Graph를 사용하여 픽셀의 알파 값을 제어하세요. 블렌딩 마스크는 높이, 각도, 카메라 거리에 따라 페이드될 수 있습니다. 이 기법은 유리, 얼음 또는 광원을 투과해야 하는 다른 요소에도 유용합니다.



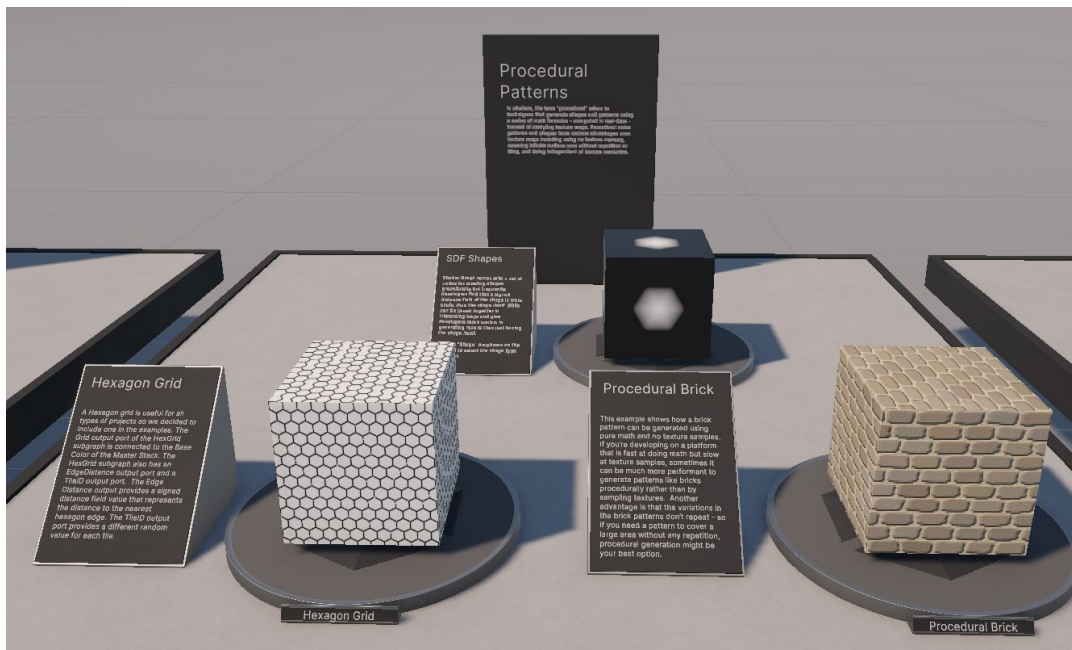
블렌딩 마스크는 거리 또는 각도에 따라 페이드될 수 있습니다.

- **파티클 셰이딩:** 셰이더는 작은 시각적 디테일을 추가하거나 다양한 파라미터에 따라 외형을 조작하는 방식으로 파티클 효과에 복잡성을 더할 수 있습니다. 예를 들어 Shader Graph를 사용하면 불꽃놀이에서 파티클의 컬러, 크기, 밝기를 제어할 수 있죠. 또는 셰이더를 적용하여 마법 주문으로 컬러를 변경하거나 광원 방출 효과를 애니메이션화할 수 있습니다.



셰이더를 사용하면 파티클의 컬러를 변경하거나 애니메이션화할 수 있습니다.

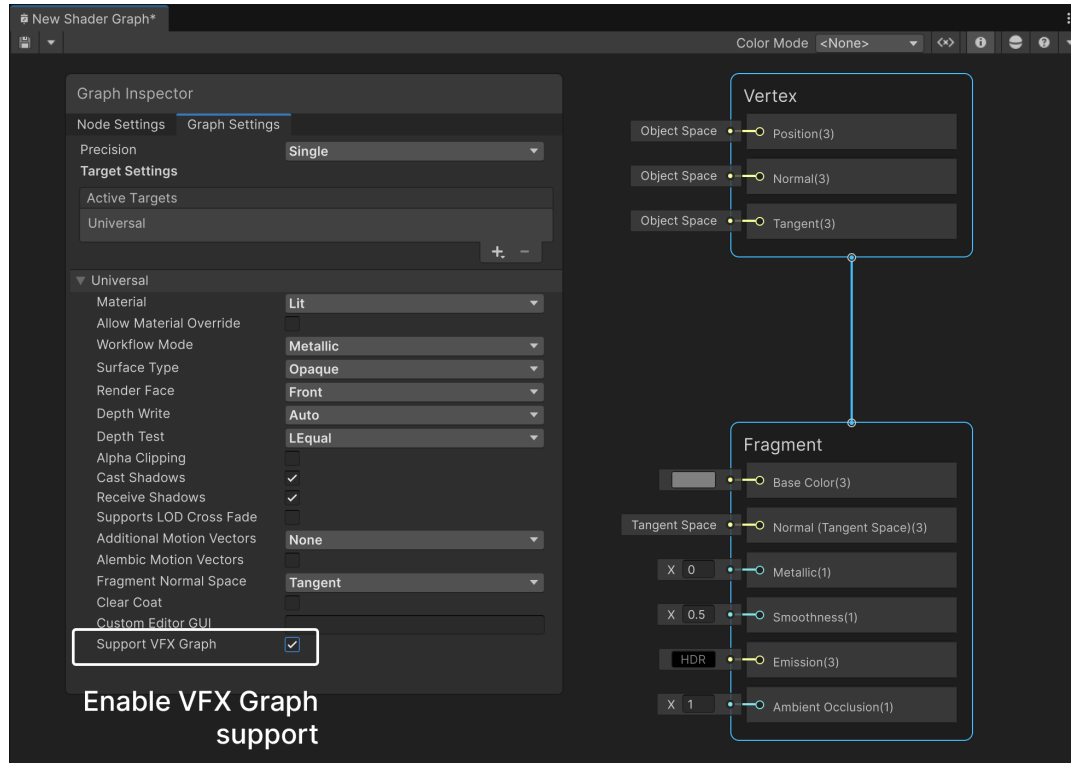
- **절차적 패턴 및 텍스처:** 셰이더는 텍스처를 절차적으로 생성할 수 있으므로 미리 만들어진 텍스처가 필요하지 않습니다. 따라서 대용량 텍스처 파일의 필요성이 줄어들고 메모리 사용량을 줄이면서 고유한 패턴을 만들 수 있게 됩니다. 텍스처는 수학적 함수를 기반으로 하기 때문에 카메라에 가까이 있어도 픽셀화나 블러 현상 없이 디테일을 표현할 수 있습니다.



셰이더는 텍스처와 패턴을 절차적으로 생성할 수 있습니다.

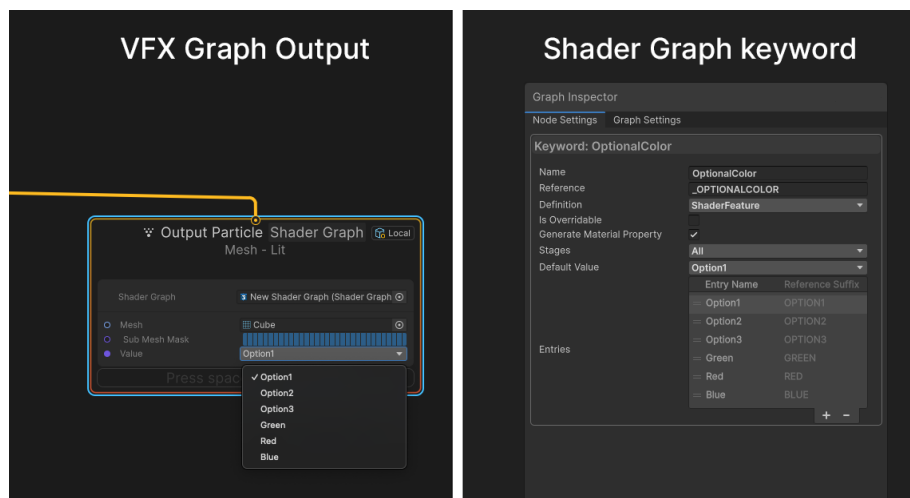
Shader Graph와 VFX Graph

Shader Graph를 새로 생성할 때 **Support VFX Graph** 옵션을 선택한 다음 VFX Graph의 Output에서 해당하는 필드에 할당하면 됩니다. 이를 통해 셰이딩과 VFX 파라미터를 함께 유지할 수 있습니다.



셰이더에서 VFX Graph 지원을 활성화합니다.

이제 VFX Graph에 Shader Graph 키워드와의 연동이 포함됩니다. 이를 통해 하나의 Shader Graph를 생성하여 여러 VFX Graph에서 사용할 수 있습니다. 그런 다음 VFX Graph Out Particle Mesh 노드에서 해당 키워드에 기반한 기능을 활성화합니다.

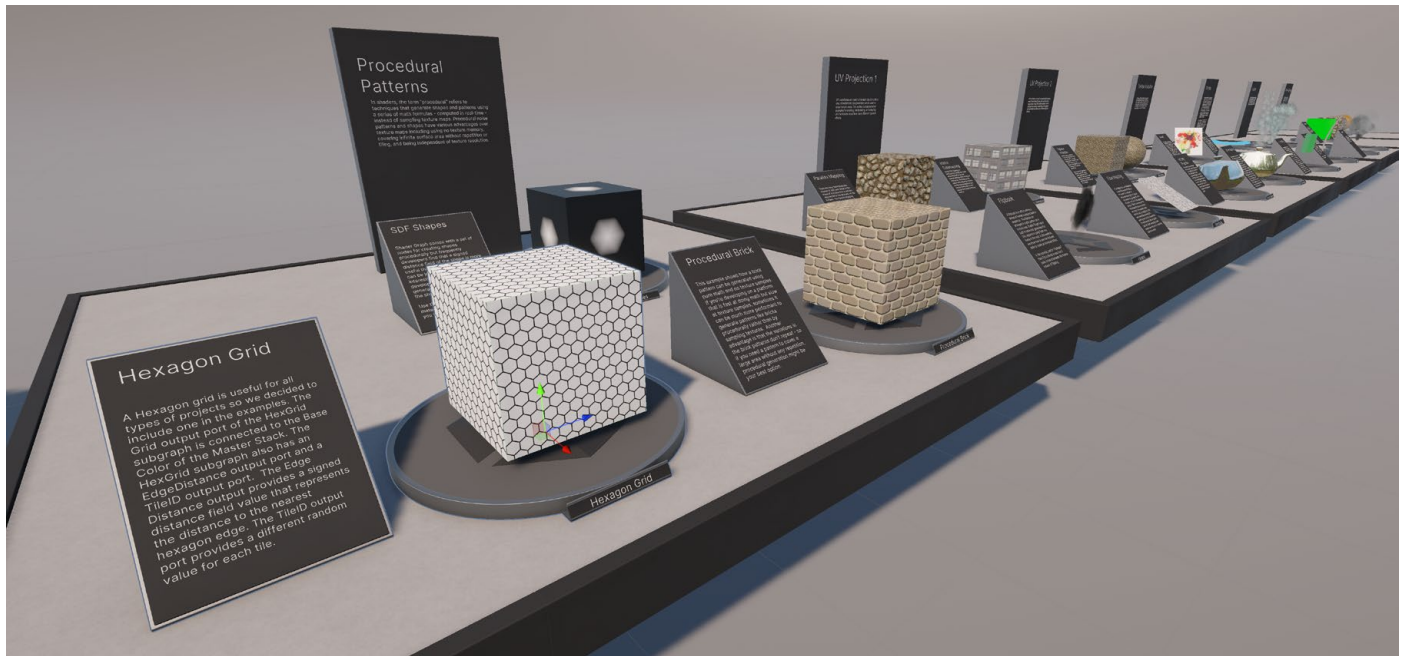


VFX Graph는 Shader Graph 키워드에 기반한 동작을 활성화할 수 있습니다.

VFX Graph Learning Templates에서 예시를 살펴보거나, 패키지 관리자에서 제공되는 Shader Graph Feature Examples에서 더 일반적인 예시를 찾아볼 수 있습니다.

Shader Graph 기능 예시

[Shader Graph Feature Examples](#) 샘플 콘텐츠는 Shader Graph에서 일반적인 기법과 효과를 구현하는 방법을 보여 주는 Shader Graph 에셋 모음입니다. 이 샘플 팩의 목표는 특정 효과를 구현하기 위해 무엇이 필요한지 알 수 있도록 지원하고 더 쉬운 학습을 위한 예시를 제공하는 것입니다.



Shader Graph Feature Examples를 살펴보세요.

상호 작용

시각 효과는 다수의 움직이는 요소를 포함하는 경우가 많습니다. 해당 요소를 런타임에 통합하려면 애플리케이션 내의 적절한 지점에 요소를 연결하는 것이 필수적입니다.

접촉하면 폭발하는 발사체를 만들거나 마우스 포인터에서 전기가 번쩍이는 효과를 구현하려는 경우 다음 툴 중 하나를 사용해 Unity 씬에서 상호 작용하는 효과를 만들 수 있습니다.

- **이벤트 바인더:** 이벤트 바인더는 씬에서 발생하는 다양한 상황을 수신 대기하며, 런타임 시 특정 행동에 반응합니다.
- **타임라인:** 활성화 트랙으로 시각 효과를 배열해서 지정된 순간에 이벤트를 그래프로 전송할 수 있습니다. 컷씬 중에 효과를 재생하는 등 미리 스크립팅된 타이밍으로 정밀하게 시각 효과를 제어할 수 있습니다.
- **프로퍼티 바인더:** 씬 또는 게임플레이 값을 Blackboard의 **Exposed Properties**에 연결하여 효과가 씬의 변화에 실시간으로 반응하도록 합니다.
- **출력 이벤트:** VFX Graph에서 스크립트나 다른 씬 컴포넌트로 메시지를 전송할 때 사용합니다.

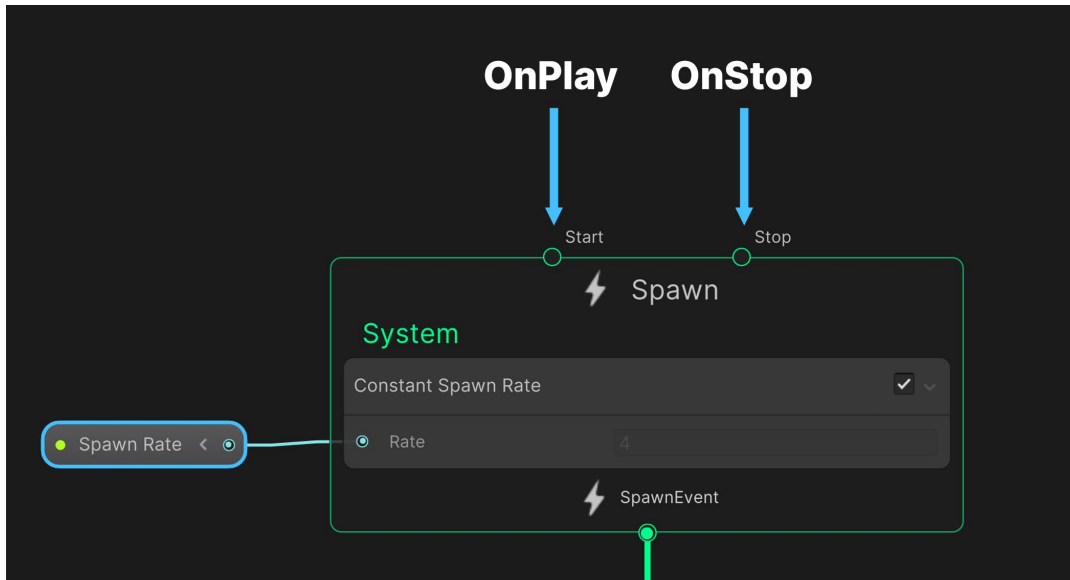
게임 오브젝트와 VFX Graph를 연결하는 핵심 기법인 각 툴을 더 자세히 살펴보겠습니다.

이벤트 바인더

이벤트 바인더는 VFX Graph 내에서 **이벤트**를 호출할 수 있는 **MonoBehaviour** 스크립트입니다. 효과가 씬의 마우스 동작, 충돌, 트리거, 가시성 이벤트에 반응하도록 합니다.

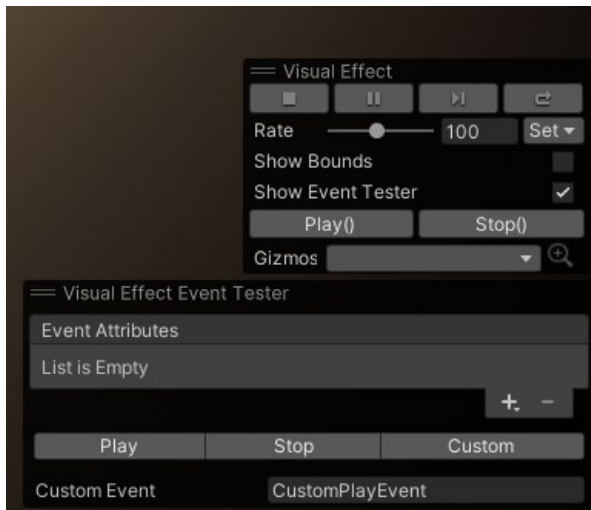
OnPlay 및 OnStop 이벤트

기본적으로 VFX Graph의 Spawn 컨텍스트에는 OnPlay 또는 OnStop 이벤트가 있습니다. 각 Spawn 컨텍스트의 Start 및 Stop 플로 슬롯은 특정 이벤트를 연결하지 않으면 묵시적으로 OnPlay 또는 OnStop 이벤트를 수신합니다.



Spawn 컨텍스트에 묵시적으로 유입되는 OnPlay 및 OnStop 이벤트

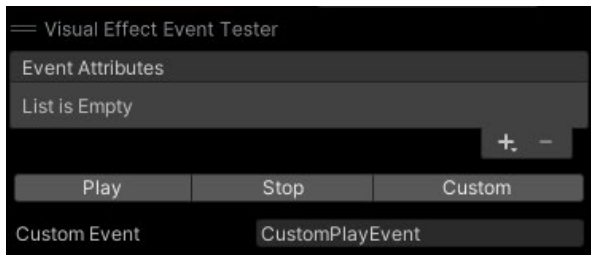
씬 뷰의 게임 오브젝트 재생 제어에 익숙하다면 알 수 있듯이, 하단의 **Play()** 버튼과 **Stop()** 버튼이 각각 OnPlay 이벤트와 OnStop 이벤트를 전송합니다.



Play 버튼과 Stop 버튼을 사용하여 OnPlay 이벤트와 OnStop 이벤트를 전송합니다.

이벤트는 오브젝트 간에 메시지를 전송하는 프로세스를 더 원활하게 만듭니다. VFX Graph에서 이벤트는 문자열로 전달됩니다. OnPlay 또는 OnStop을 눌러도 효과가 즉시 변경되지 않으며, 상단의 재생 아이콘을 사용하는 경우와 비교하면 더욱 그렇습니다. 두 이벤트는 생성 시스템에 신호만 제공합니다.

Visual Effect Event Tester 다이얼로그 창을 열면, Play 버튼 및 Stop 버튼을 사용해 동일한 효과를 얻을 수 있습니다. 이러한 유연성을 활용해 **커스텀 이벤트**를 지정하고 Custom 버튼으로 호출할 수 있습니다.

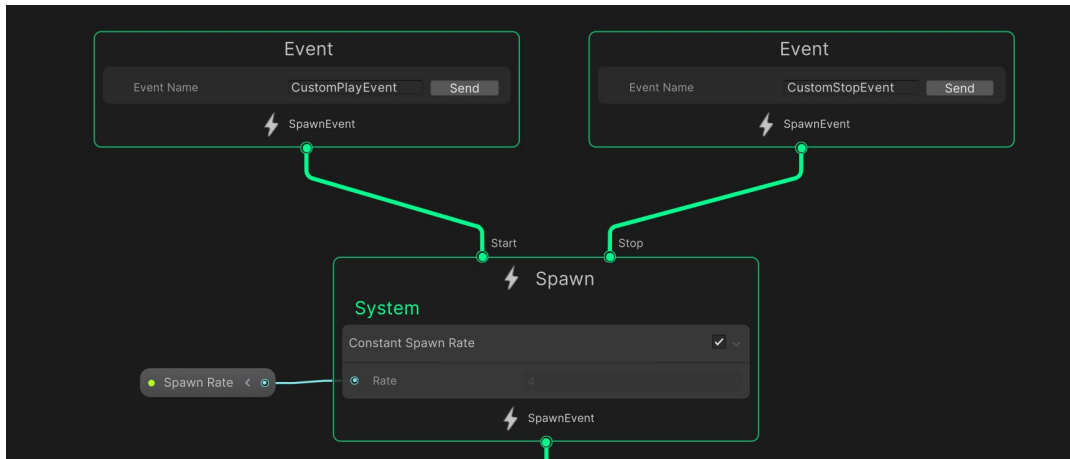


Visual Effect Event Tester

Node > Context > Event를 사용해 그래프에 고유의 이벤트를 추가할 수 있습니다. 테스트 시 **Send** 버튼을 누르면 수동으로 이벤트를 일으킬 수 있습니다.

커스텀 이벤트를 만들려면 **Event Name** 문자열을 변경하고 런타임에 **이벤트 바인더** 또는 **타임라인 활성화 클립**으로 이벤트를 호출하면 됩니다.

이 예시에서는 **CustomPlay**와 **CustomStop**이라는 두 개의 이벤트를 추가했습니다.

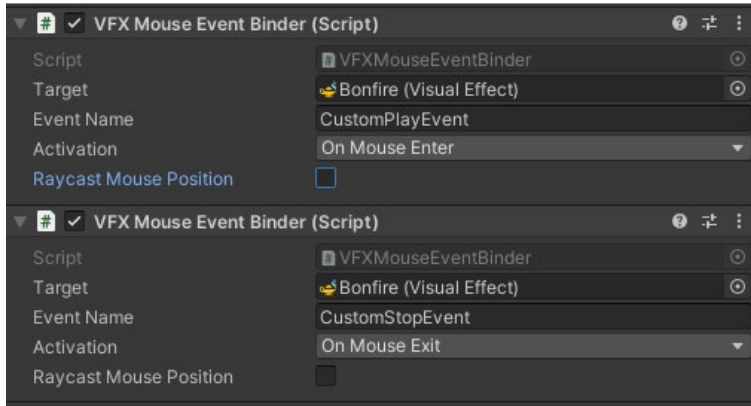


그래프에서 커스텀 이벤트를 설정하는 방법

Mouse Event Binder

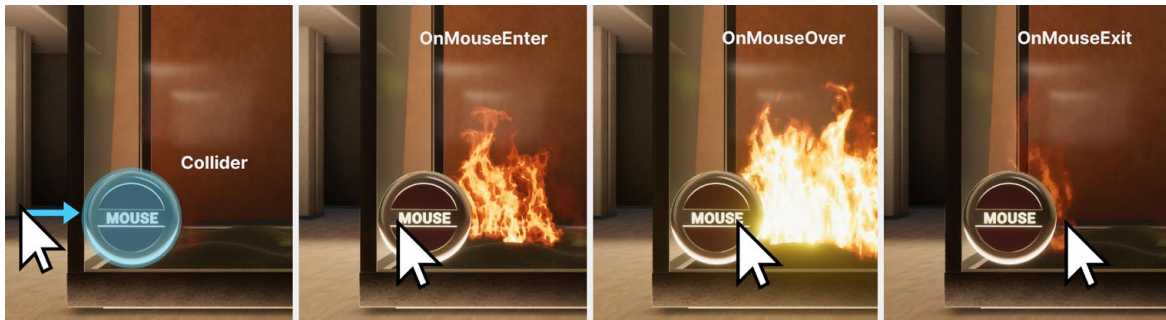
VFX Mouse Event Binder를 사용하면 마우스 포인터를 클릭, 드래그하거나 올려놓는 방법으로 그래프에 메시지를 전송할 수 있습니다. 이때 필요한 것은 콜라이더가 있는 게임 오브젝트뿐입니다.

이 예시에서 두 개의 이벤트 바인더는 **CustomPlay 이벤트**와 **CustomStop 이벤트**를 Bonfire 효과에 연결합니다.



Mouse Event Binder로 효과 활성화

마우스 포인터가 화면의 콜라이더에 들어가면 CustomPlay 이벤트가 그래프로 전송됩니다. 이 이벤트는 불길과 연기, 불꽃을 생성합니다. 마우스 포인터가 콜라이더를 벗어나면 CustomStop 이벤트가 Spawn 컨텍스트에 중지하라고 알립니다.

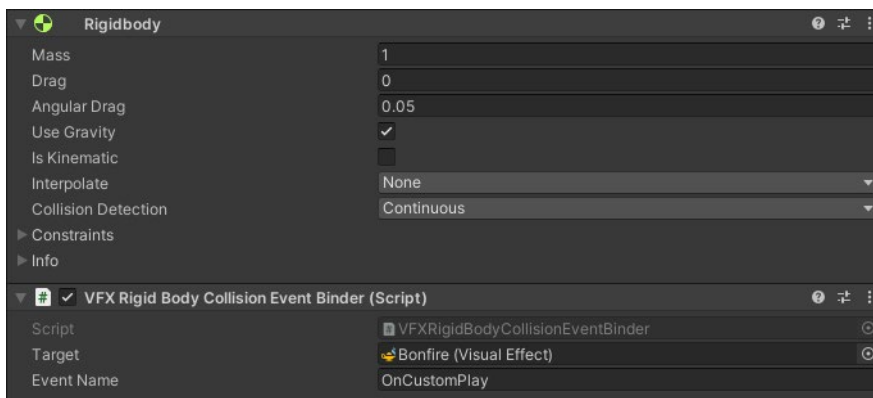


마우스 포인터가 콜라이더에 들어가거나 콜라이더를 벗어날 때 이벤트가 발생합니다.

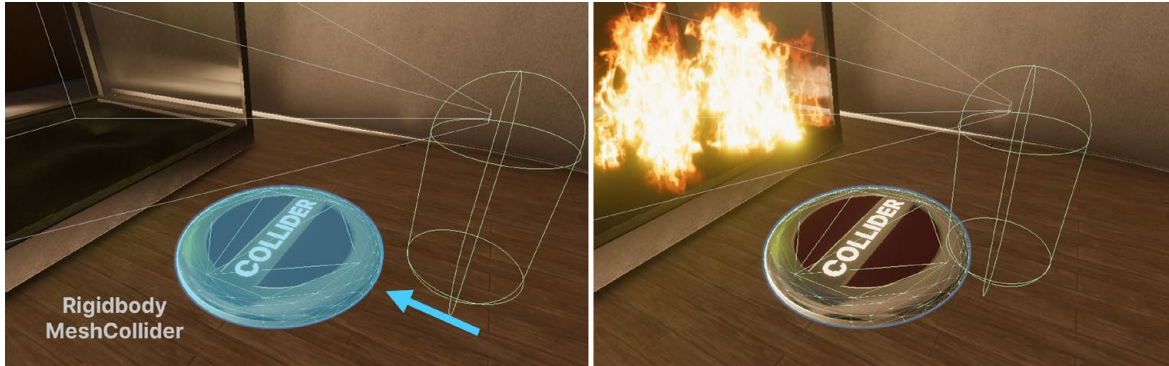
모든 표준 마우스 동작(**Up, Down, Enter, Exit, Over, Drag**)에 이벤트를 바인드할 수 있습니다. **Raycast Mouse Position** 옵션은 포인터의 3D 위치를 **이벤트 속성**으로 전달합니다.

Rigidbody Collision Event Binder

Rigidbody Collision Event Binder를 사용하면 충돌 발생 시 물리 오브젝트가 그래프에 경고를 표시할 수 있습니다. **리지드바디**와 **콜라이더**가 있는 적절한 게임 오브젝트에 이벤트 바인더를 연결하면 됩니다.



충돌하면 VFX Graph에 이벤트가 전송됩니다.



리지드바디 충돌

접촉하는 모든 콜라이더는 지정된 이벤트 이름을 가진 이벤트를 발생시키고, 메시지 형태로 **시각 효과 타겟**에 전송합니다.

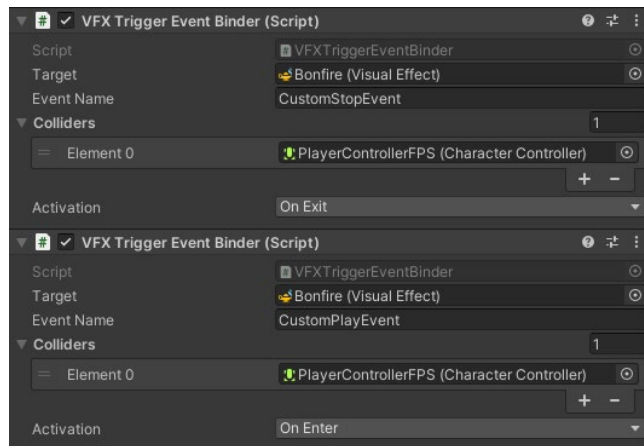
이 예시에서는 리지드바디가 있는 **메시 콜라이더**가 버튼으로 작동합니다. 이 효과는 플레이어가 접촉하면 생성되기 시작합니다.

이 이벤트 바인더는 리지드바디와의 충돌에 반응하므로, 다양한 형태의 상호 작용을 생성할 수 있습니다. 반사될 때마다 파티클을 방출하는 구체 또는 발사체에 맞으면 왜곡되는 역장(force field) 등을 상상해 보세요.

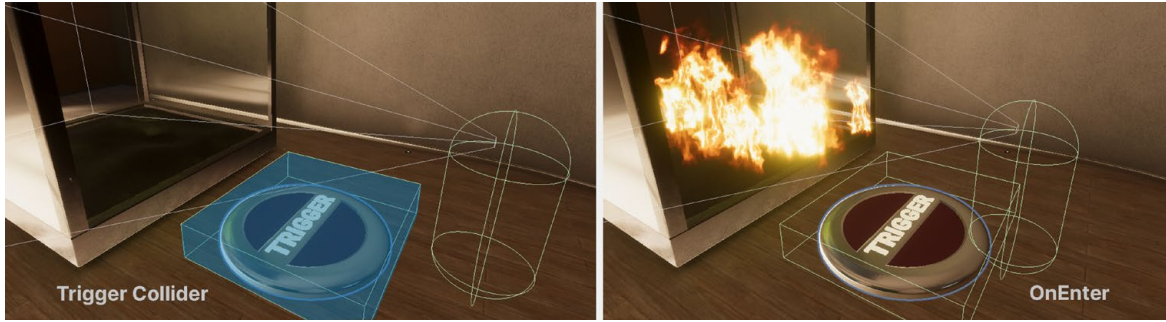
Trigger Event Binder

마찬가지로 콜라이더가 트리거로 설정된 게임 오브젝트에 VFX Trigger Event Binder를 추가할 수 있습니다. 이 컴포넌트는 **OnEnter**, **OnExit**, **OnStay** 등의 특정 **활성화 이벤트**를 등록할 수 있으며, 이벤트 이름이 지정된 이벤트를 **Target** 그래프로 전송합니다.

Trigger Event Binder를 사용하여 플레이어 또는 다른 게임 오브젝트가 레벨의 특정 부분에 도달했을 때 효과를 트리거할 수 있습니다.



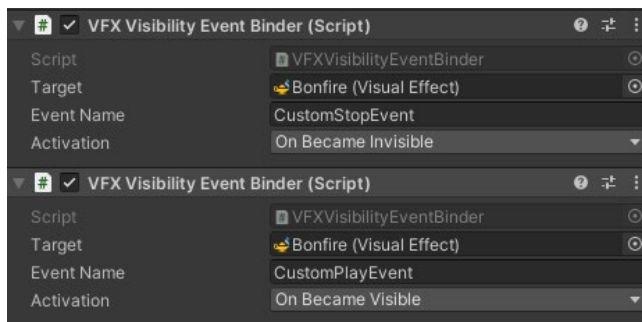
Trigger Event Binder



Trigger Event Binder는 특정 콜라이더를 감지합니다.

Visibility Event Binder

Visibility Event Binder를 사용하면 오브젝트가 보이거나 보이지 않게 될 때 이벤트를 발생시킬 수 있습니다. 여기에서는 몇 가지 렌더러 유형을 사용하여 게임 오브젝트에 Visibility Event Binder를 연결합니다.



Visibility Event Binder

Activation 설정(**OnBecameVisible** 또는 **OnBecameInvisible**)에 따라 특정 이벤트 이름을 Target 그래프에 전달합니다. 그러면 렌더러가 카메라 절두체에 들어오거나 나갈 때 또는 렌더러를 켜고 끌 때 그래프에 이를 알립니다.



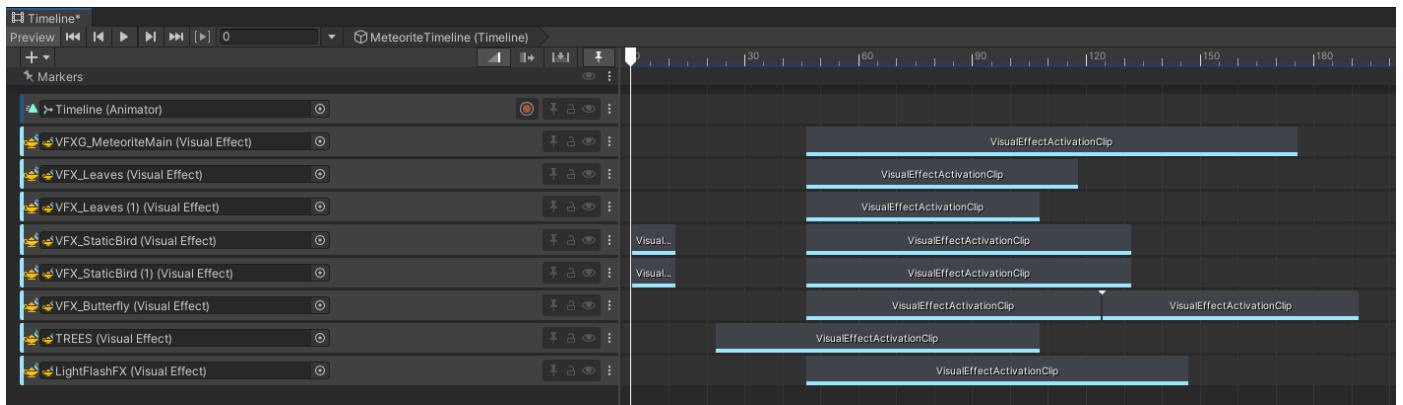
시각 효과는 타겟이 보이게 될 때만 활성화됩니다.



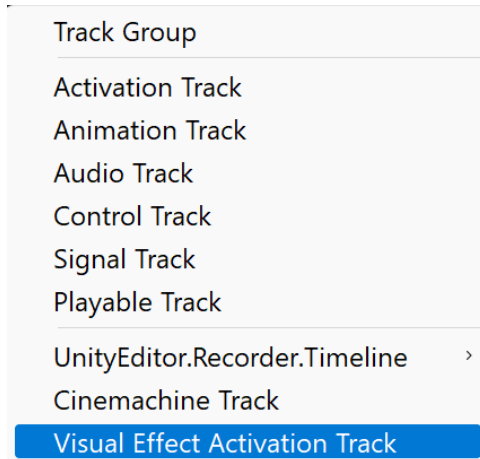
동영상 클립: 이벤트 바인더

타임라인

타임라인은 시각 효과를 정확한 타이밍에 맞춰 켜고 꺼야 할 때, 그래프와 커뮤니케이션할 수 있는 또 다른 방법을 제공합니다. 시각 효과 활성화 트랙을 사용해 여러 효과 레이어를 편성할 수 있습니다.

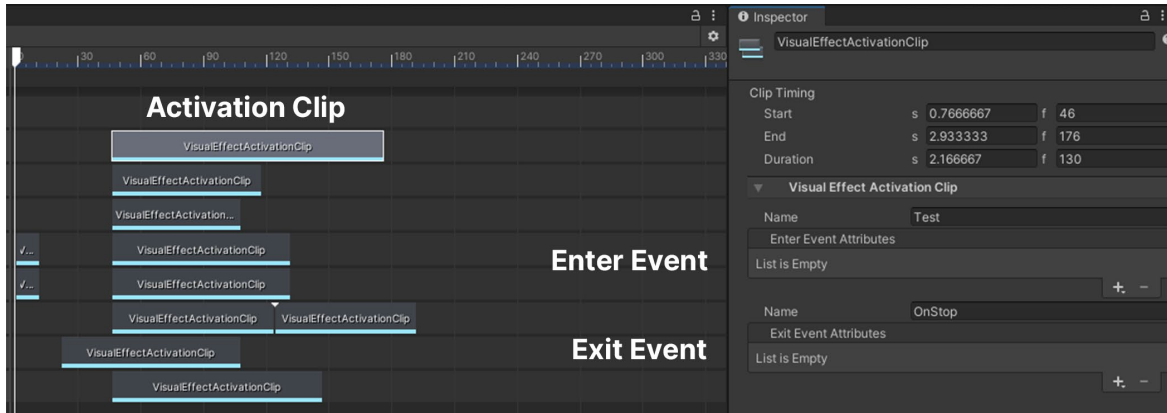


시각 효과 활성화 트랙

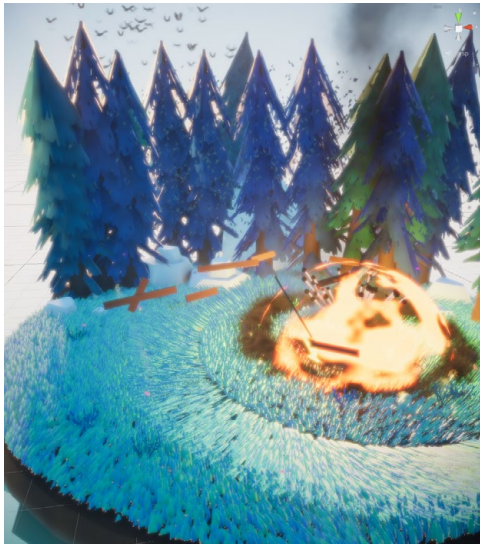


타임라인을 오른쪽 클릭하여 새 활성화 트랙을 만들 수 있습니다.

활성화 트랙에 효과를 할당하고 타임라인 트랙에 하나 이상의 **활성화 클립**을 생성합니다. 각 클립은 클립이 시작할 때와 끝날 때 각각 하나씩 두 개의 이벤트를 전송합니다.



각 활성화 클립은 두 개의 이벤트를 해당 VFX Graph에 전송합니다.

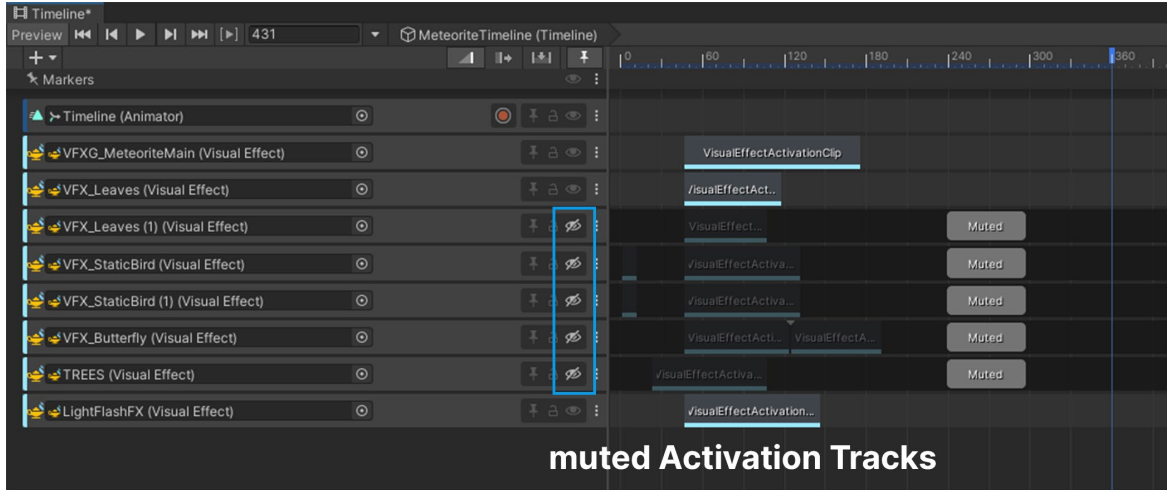


VFX Graph 샘플 프로젝트의 Meteorite 샘플

Meteorite 샘플은 함께 조화를 이루며 재생되는 다수의 효과를 타임라인으로 제어하는 방법을 보여 줍니다.

타임라인을 이용하면 함께 모여 전체 효과를 생성하는 각 효과의 요소를 더 쉽게 정리할 수 있습니다. 여기서는 각 하위 그래프에 별도의 활성화 트랙을 사용하며 활성화 클립을 통해 이벤트를 전달합니다.

타임라인 내에서 클립을 슬라이드하면 상호 작용하며 타이밍을 조정할 수 있습니다. 커스텀 MeteoriteControl 스크립트는 Playable Director 컴포넌트를 호출합니다.



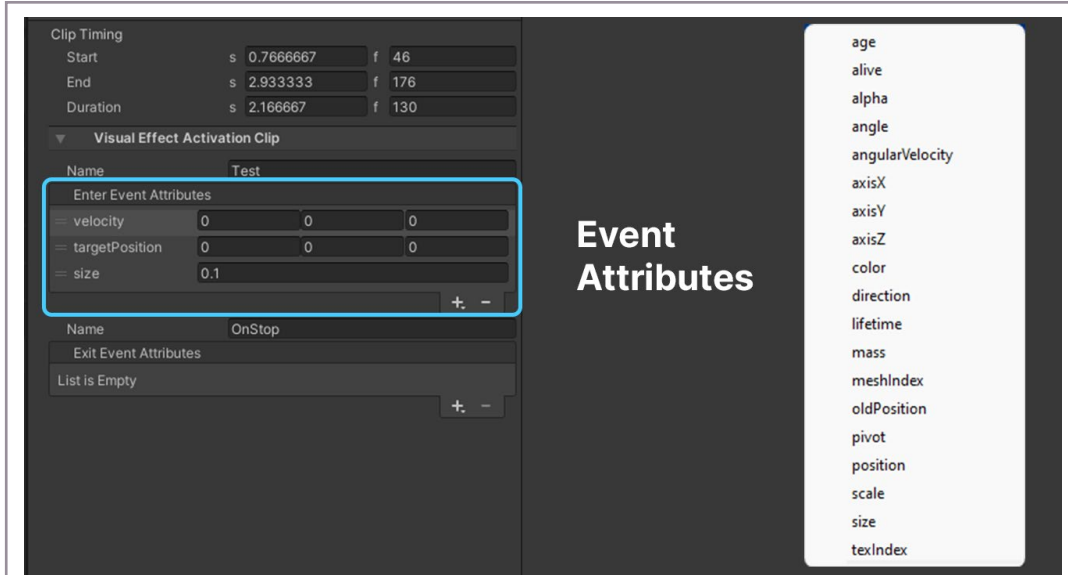
활성화 트랙 뮤트

타임라인을 사용하여 특정 시각 효과 활성화 트랙을 뮤트할 수도 있으며, 이렇게 하면 일시적으로 이벤트 수신에 중단됩니다. 문제를 해결해야 할 때 유용한 기능입니다.

이벤트 속성

이벤트 바인더와 타임라인 활성화 클립은 이벤트에 [이벤트 속성 페이로드](#)를 연결할 수 있습니다. 그러면 이벤트가 호출될 때 추가 정보를 이벤트와 함께 전달할 수 있습니다.

예를 들어 효과를 인스턴스화할 위치를 그래프에 알리는 노출 **Vector3** 프로퍼티를 가지는 이벤트 속성을 만들 수 있습니다.



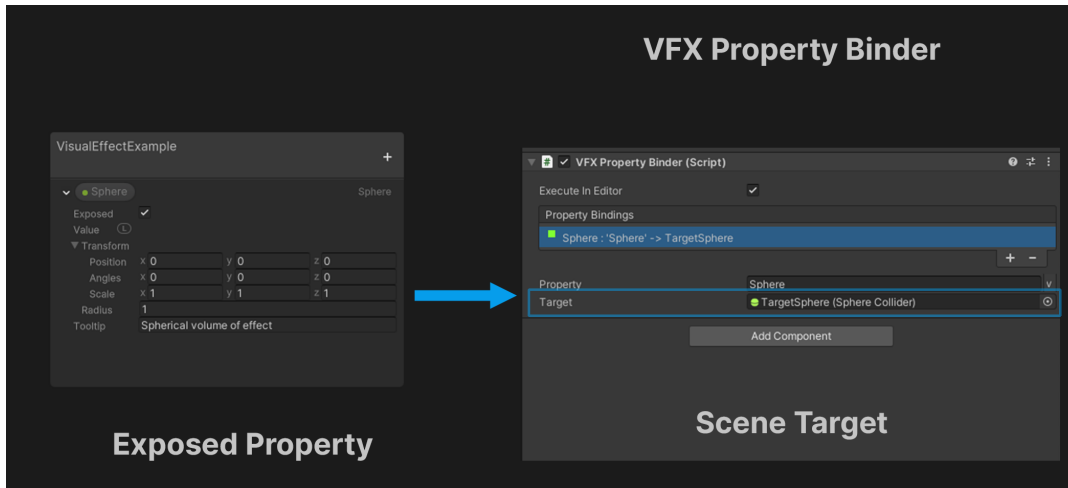
이벤트 속성을 사용하여 이벤트를 트리거할 때 데이터를 전달합니다.

VFX Graph에서 이러한 속성을 설정하려면 **Spawn 컨텍스트**에서 **Set Attribute Blocks**를 사용하면 됩니다. C# 스크립트에서 전송된 이벤트에도 해당 속성을 연결할 수 있습니다. 자세한 내용은 Visual Effect [컴포넌트 API](#)를 참조하세요.

프로퍼티 바인더

프로퍼티 바인더는 씬이나 게임플레이 값을 VFX Graph의 [Exposed Properties](#)에 연결할 수 있는 C# 동작입니다. **VFX Property Binder**라는 일반 MonoBehaviour를 통해 프로퍼티 바인더를 추가할 수 있습니다.

예를 들어 **스피어 콜라이더 바인더**는 씬의 스피어 콜라이더 값을 사용하여 **Sphere Exposed 프로퍼티**의 위치와 반지름을 자동으로 설정할 수 있습니다.

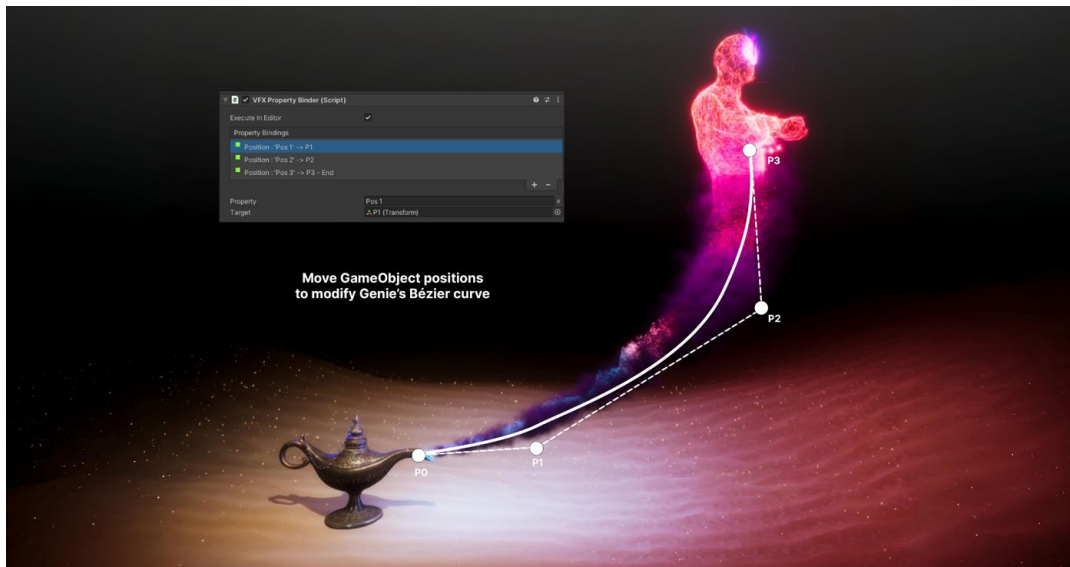


씬 값을 노출 프로퍼티에 연결하는 프로퍼티 바인더

런타임 시 효과에 영향을 줄 수 있는 광원이나 카메라를 씬에 배치해야 하거나, 효과가 트랜스폼이나 Vector3를 따라가야 하는 경우, 프로퍼티 바인더를 사용하면 여러 노출 프로퍼티 유형을 씬의 값과 동기화할 수 있습니다. 사용 가능한 전체 항목은 **Add Component > VFX > Property Binders**에서 확인할 수 있습니다.

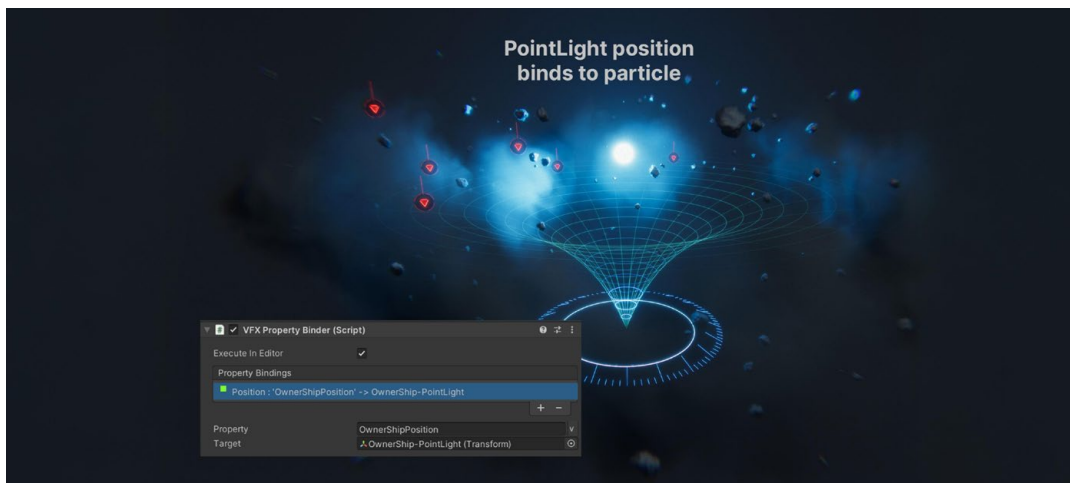
시각 효과 샘플에는 프로퍼티 바인더가 씬 계층 구조를 그래프에 연결하는 방법을 보여 주는 인스턴스가 있습니다.

- Magic Lamp 샘플에서 프로퍼티 바인더는 여러 씬 오브젝트(P1, P2, P3)의 위치를 그래프의 Blackboard 프로퍼티(Pos1, Pos2, Pos3)에 연결합니다. 그러면 이 **Vector3 노드**는 지니의 전반적인 형상을 정의하는 **베지어(Bézier) 스플라인**을 형성합니다. 씬에서 P1, P2, P3 **트랜스폼**을 움직이면 지니의 연기 트레일이 실시간으로 반응합니다.



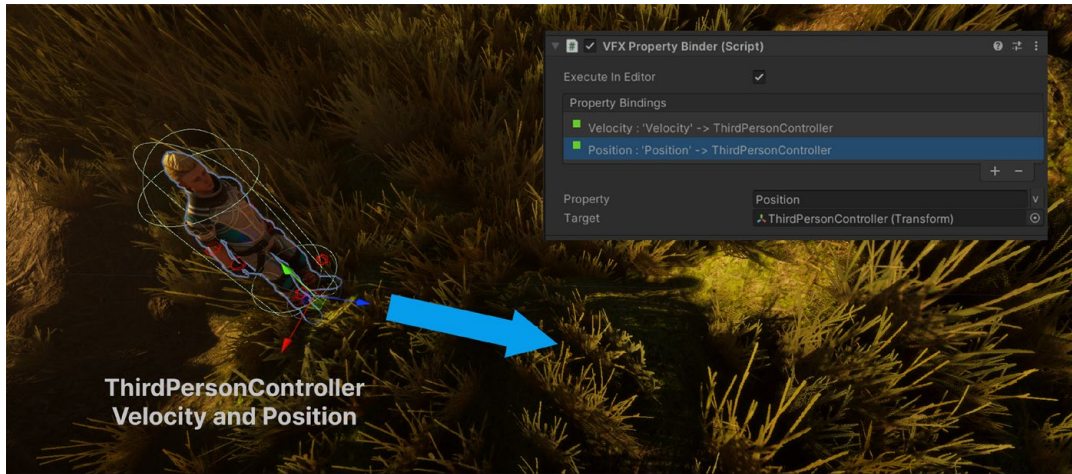
결과적으로 생성되는 베지어 커브가 지니의 형상을 결정합니다.

- ARRadar에서 **PointLight Transform**은 3D 레이더 화면에서 플레이어의 함선이 나타나는 위치를 결정하며, 빛나는 신호를 실시간 광원과 동기화합니다.



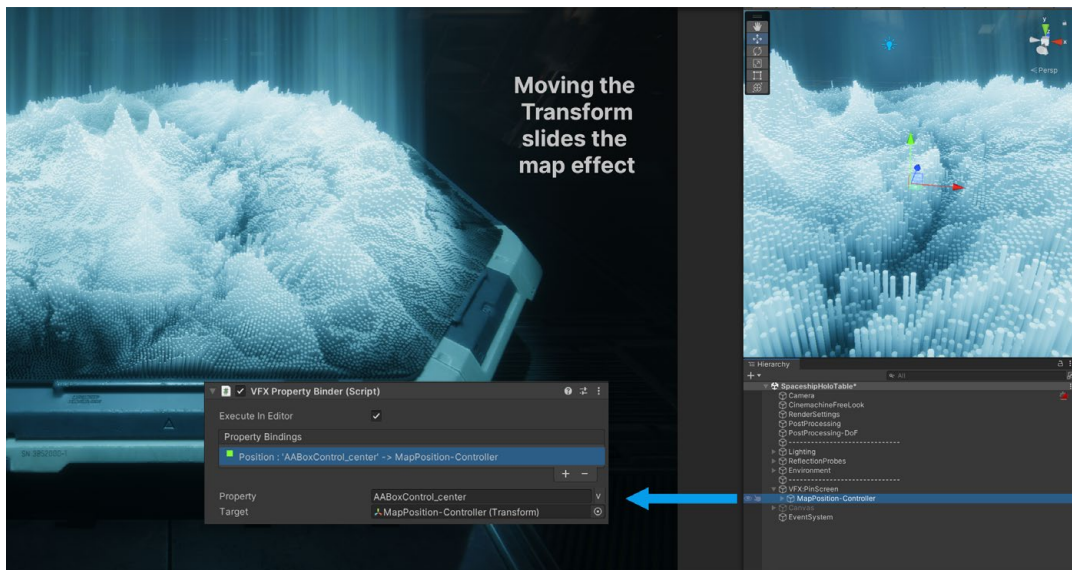
ARRadar는 PointLight를 함선 아이콘에 연결합니다.

- Grass Wind 샘플에서는 프로퍼티 바인더가 **ThirdPersonController**라는 트랜스폼의 위치와 속도를 캡처하여 잔디를 밀어냅니다.



Grass Wind 샘플

- SpaceshipHoloTable에서 **MapPosition-Controller**라는 트랜스폼을 사용하여 테이블 위의 PinScreen 효과 위치를 정합니다. 애니메이션이 미리 정해진 모션에 따라 효과를 움직이게 하거나 런타임에 드래그하도록 하면 홀로그램이 생생하게 움직이는 모습이 재생됩니다.



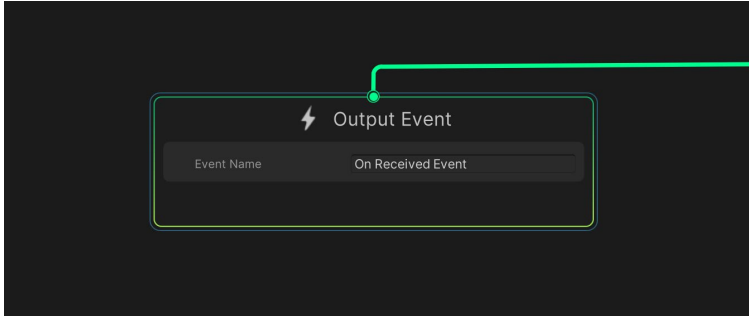
SpaceshipHoloTable

이는 프로퍼티 바인더가 그래프와 씬의 관계를 강화하는 다양한 사례 중 일부에 불과합니다. 오디오, 입력, 물리, UI 등 다양한 컴포넌트에 대한 **빌트인 프로퍼티 바인더**를 확인해 보세요.

필요한 프로퍼티 바인더를 찾을 수 없다면 **UnityEngine.VFX.Utility.VFXBinderBase** 클래스를 사용하여 **직접** 프로퍼티 바인더를 작성해도 됩니다.

출력 이벤트

이벤트를 활용하면 VFX Graph에 메시지를 보내는 방법과 유사한 방법으로 메시지를 내보낼 수 있습니다. **출력 이벤트를 사용하면, Spawner 컨텍스트**에서 새 파티클의 속성을 얻을 수 있습니다. 이를 **출력 이벤트 핸들러**와 함께 사용하여 씬의 C# 스크립트에 알릴 수 있습니다.



출력 이벤트

카메라 흔들기, 사운드 재생, 프리팹 생성 등 게임플레이 로직에 따라 반응하는 다양한 동작을 얼마든지 만들 수 있습니다.

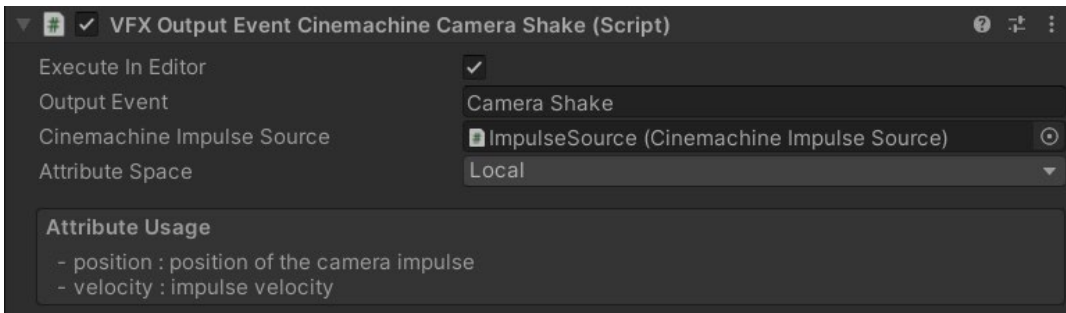
이벤트를 수신하려면 **UnityEngine.VFX.Utility** 네임스페이스의 **VFXOutputEventAbstractHandler**에서 **스크립팅된 클래스를 상속** 합니다. 그리고 다음과 같은 메서드를 오버라이드합니다.

```
override void OnVFXOutputEvent(VFXEventAttribute eventAttribute)
```

Unity는 이벤트가 트리거될 때마다 이벤트 속성을 파라미터로 전달하여 **OnVFXOutputEvent**를 호출합니다. VFX Graph에 포함되어 **Output Event Helper**에서 제공하는 구현 사항을 확인해 보세요. 패키지 관리자에서 설치하여 예제 스크립트를 살펴볼 수 있습니다.

Meteorite 샘플을 다시 살펴보며 실제 그래프 내에서 어떻게 작동하는지 확인할 수도 있습니다. 다음은 **VFX_MeteoriteMain 게임 오브젝트**에 연결할 수 있는 여러 출력 이벤트 핸들러입니다.

- **카메라 흔들림: VFX Output Event Cinemachine Camera Shake** 컴포넌트는 **Camera Shake 이벤트**가 발생했을 때 카메라를 흔들니다.



Camera Shake 스크립트

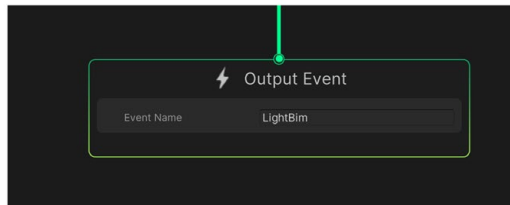
- **보조 효과: VFX Output Event Prefab Spawn** 컴포넌트에 의해 **Buffer** 및 **PlankImpulse** 이벤트가 발생합니다. 출력 이벤트 핸들러에 의해 발생한 충격파는 잔디를 거쳐 나무판자를 날려 보냅니다.



보조 효과를 생성하는 출력 이벤트 핸들러

- **광원 애니메이션:** 또 하나의 VFX Output Event Prefab Spawn 컴포넌트는 운석이 충돌할 때 광원 애니메이션을 생성합니다. 커스텀 **Output Event Handler Light Update** 스크립트는 볼류메트릭 스케일, 밝기, 컬러를 수정하여 더 극적인 느낌의 충돌을 구현합니다.

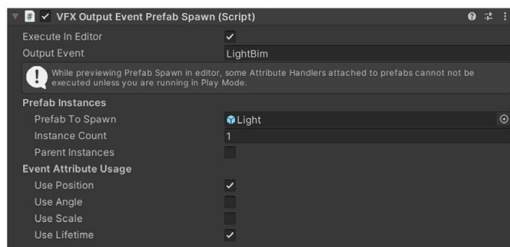
VFX Graph Output Event



GameObject



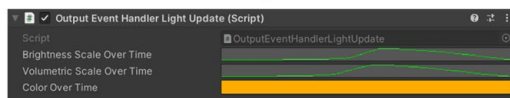
Output Event Handler spawns Prefab



Prefab instance



Light animation from another Output Event Handler



커스텀 Light Update Output Event Handler



운석 충돌 효과에 반응하는 광원

우선 이러한 아이디어 중 몇 개를 살펴보세요. 코드를 작성하지 않고 샘플 스크립트를 바로 사용해도 되고, 자체 스크립트를 제작할 때 참고할 수도 있습니다. 그러다 보면 곧 애플리케이션에 필요한 모든 출력 이벤트 핸들러를 직접 만들 수 있게 될 것입니다.

파이프라인 툴

주변과 단절된 효과란 있을 수 없습니다. 원하는 모습을 구현하려면 효과에 외부 데이터를 제공해야 하는 경우가 많습니다.

요술 램프에서 나오는 지니를 구현하거나, 홀로그램과 SF 우주선을 통합하려는 경우를 생각해 보세요. 수학 함수와 연산자를 사용해 대부분 구현할 수도 있겠지만, 더 복잡한 모양 및 형태와 상호 작용하는 효과가 필요할 수도 있습니다.

이러한 이유로 Unity는 다음과 같이 다양한 데이터 유형을 지원합니다.

- **포인트 캐시:** 트랜스폼, 노멀, 컬러, UV 등 공간에 있는 점의 속성을 저장합니다.
- **Signed Distance Fields:** 볼류메트릭 표현을 사용하여 파티클을 끌어당기고 충돌합니다.
- **벡터 필드:** 파티클의 위치를 샘플링한 후 3D 공간에서 파티클을 밀어냅니다.

Unity는 이러한 파일 포맷을 쉽게 생성할 수 있도록 여러 지원 유틸리티도 함께 제공합니다.

포인트 캐시

포인트 캐시는 점과 그 위치, 노멀, 컬러 등 **파티클 속성** 데이터의 고정 목록을 저장하는 에셋입니다.

포인트 캐시 에셋은 오픈 소스 **포인트 캐시 사양**을 따르고 **.pCache** 파일 확장자를 사용합니다. 내부적으로 포인트 캐시는 중첩된 **스크립터블 오브젝트**이며, 파티클 속성 맵을 나타내는 다양한 텍스처를 모두 포함합니다.

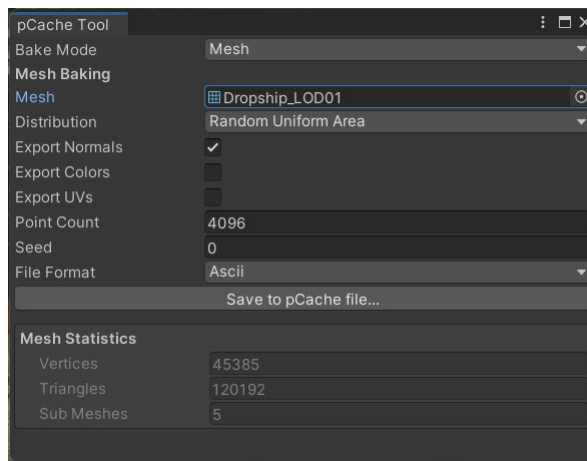
또한 SDF보다 적은 리소스를 소모합니다.

다음은 시각 효과에 사용할 포인트 캐시를 생성하는 방법입니다.

- **Window > Visual Effects > Utilities > Point Cache Bake Tool**에서 빌트인 [포인트 캐시 베이킹](#) 툴을 사용합니다.
- **VFXToolbox**와 함께 제공되는 **Houdini pCache Exporter**를 선택합니다.
- 자체 커스텀 익스포터를 제작합니다. [pCache README](#)에서 예셋의 형식과 사양에 대한 자세한 내용을 볼 수 있습니다.

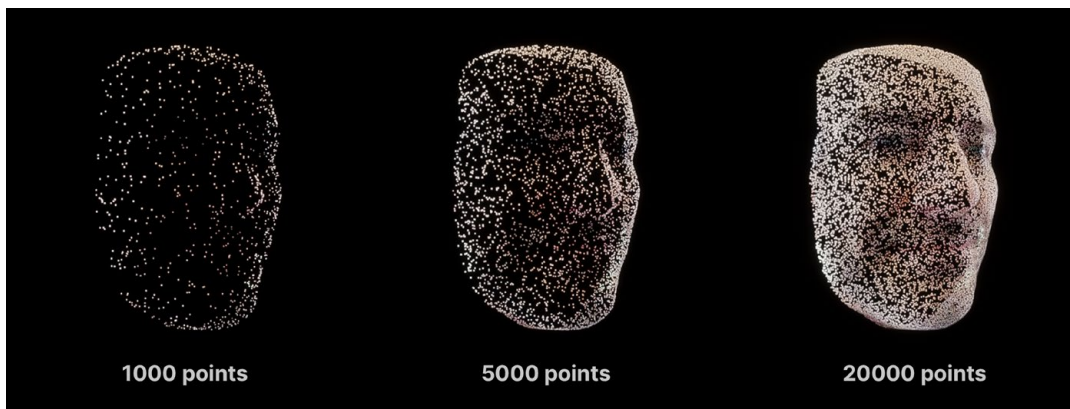
포인트 캐시 베이킹 툴

포인트 캐시 베이킹 툴을 사용하면 입력 메시 또는 2D 텍스처로부터 포인트 캐시를 생성할 수 있습니다.



포인트 캐시 베이킹 툴

포인트 캐시는 3D 메시 또는 2D 텍스처에서 생성된 점의 목록을 저장하지만, 실제 지오메트리를 저장하지는 않습니다. 베이킹 중에 추가 필터링을 사용하면 점을 분산시켜 더 고르게 분리하고 점이 중복되는 횟수를 줄일 수 있습니다. **Mesh** 또는 **Texture**를 선택하고 적절한 **Point Count**를 설정한 후, **Save to pCache file**을 선택합니다.



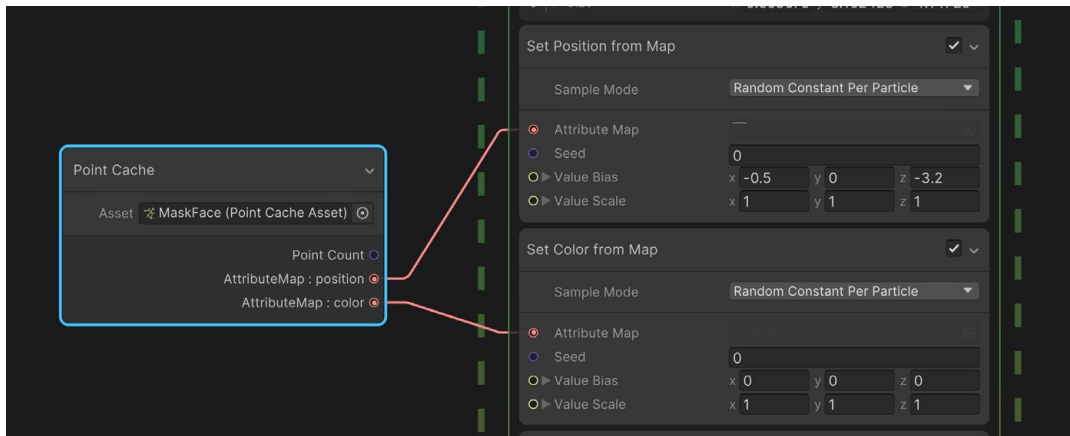
다양한 샘플로 구현한 포인트 캐시의 시각화

포인트 캐시는 **Stanford PLY** 파일과 유사하지만 .pCache 파일 포맷은 폴리곤을 제거하고 벡터에 대한 추가 지원을 제공합니다. 따라서 Python이나 C#에서 더 쉽게 읽고 쓸 수 있습니다.

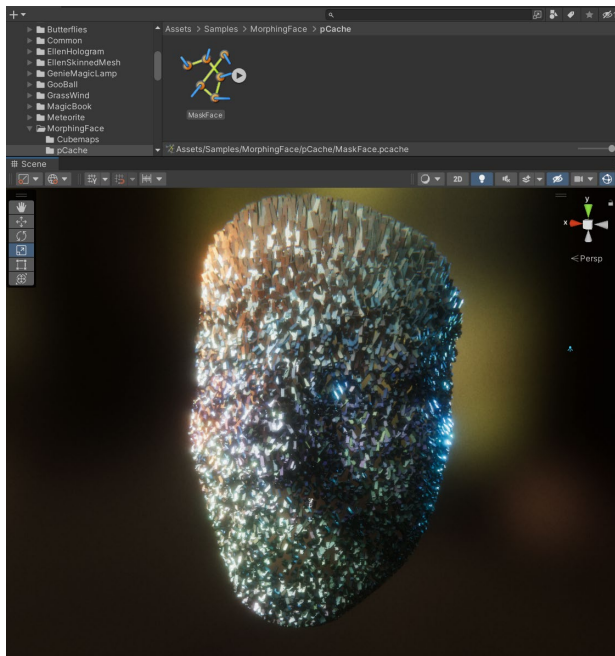
포인트 캐시 사용

VFX Graph에서 **포인트 캐시 연산자**는 포인트 캐시 에셋에서 파티클의 수와 그 속성을 추출합니다. 그런 다음 연산자에서 출력 포트에 노출합니다.

Morphing Face 샘플에서 연산자는 Point Count에 출력 슬롯 1개를 생성하고 Attribute Map에 별도의 텍스처 슬롯을 생성합니다. 출력을 **Set Attribute from Map 블록** 등의 다른 노드에 연결할 수 있습니다.



VFX Graph에서 포인트 캐시 사용



포인트 캐시 에셋에 적용된 Morphing Face 효과

MaskFace 포인트 캐시는 효과의 기본적인 구조를 구성합니다.

자세한 내용은 **포인트 캐시 기술 자료**를 참조하세요.

Signed Distance Fields

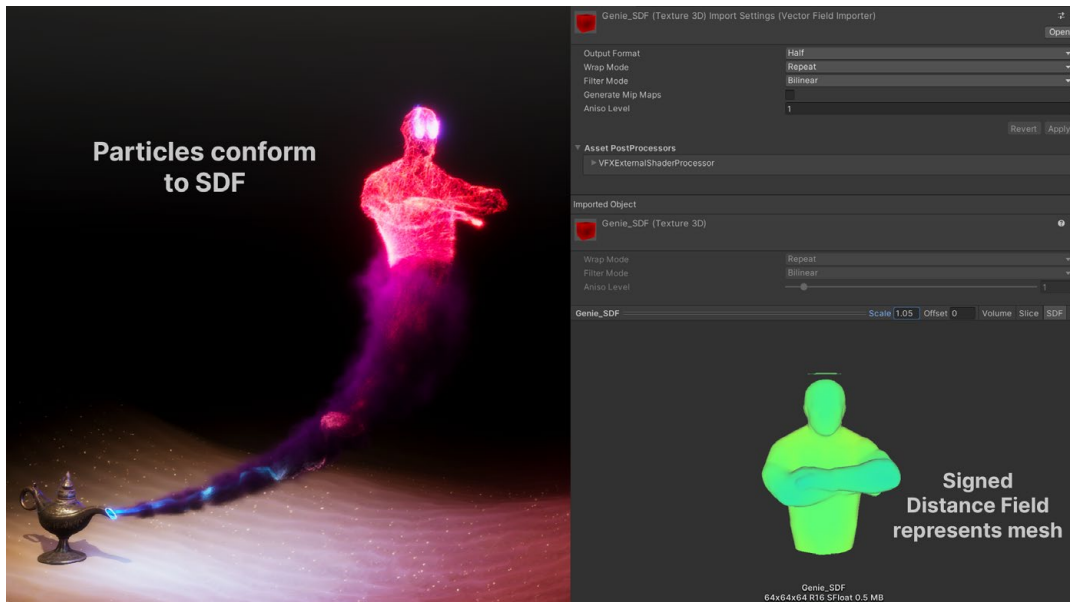
SDF(**Signed Distance Field**)는 메시 지오메트리의 3D 텍스처 표현입니다. 각 텍셀은 메시 표면과 가장 가까운 거리 값을 저장합니다.

일반적으로 이 거리는 메시 안쪽이 음수, 메시 바깥쪽이 양수입니다. 따라서 파티클을 표면 위 임의의 지점, 지오메트리의 경계 안쪽, 지정된 거리 등에 배치할 수 있습니다.

SDF는 포인트 캐시에 비해 많은 리소스를 계산에 소모하는 반면, 여러 추가 기능을 제공합니다. 메시가 상당히 세밀한 경우 높은 해상도의 텍스처가 필요하며, 그러면 보통 더 많은 메모리를 소모합니다.

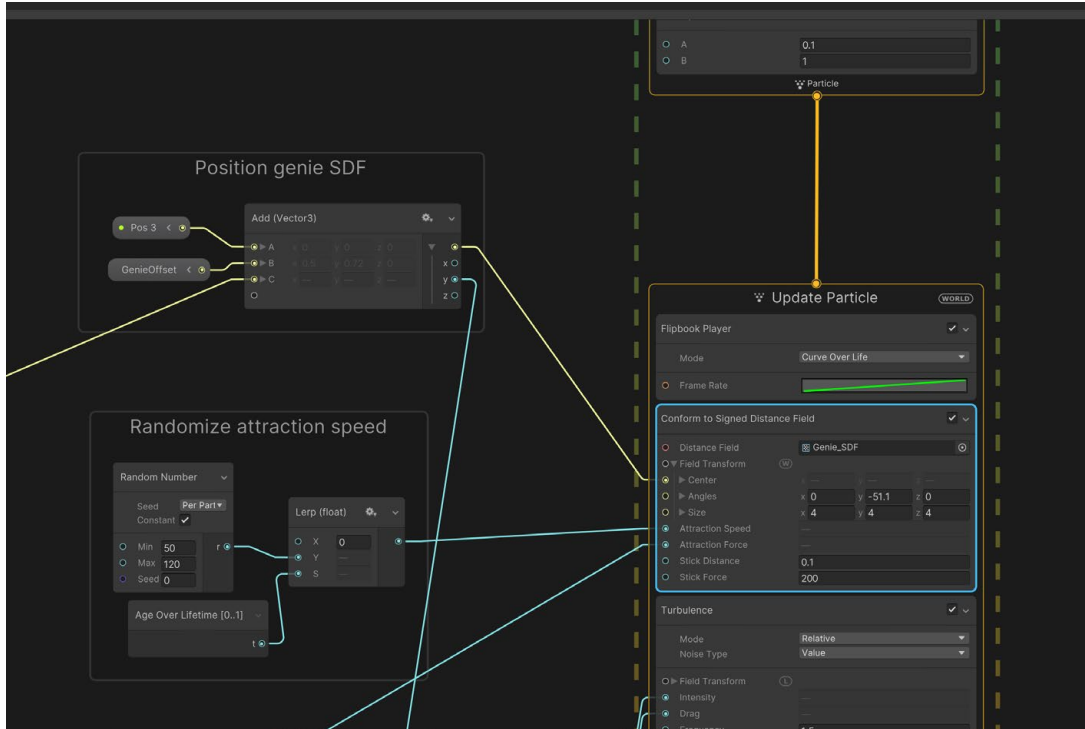
SDF 사용

시각 효과에 SDF를 사용하면 파티클의 위치를 지정하거나, 파티클을 모양에 맞추거나, 다른 파티클과 충돌시킬 수 있습니다. Magic Lamp 샘플에서는 지니의 바디에 SDF를 사용했습니다. SDF 에셋의 프리뷰는 다음과 같습니다.

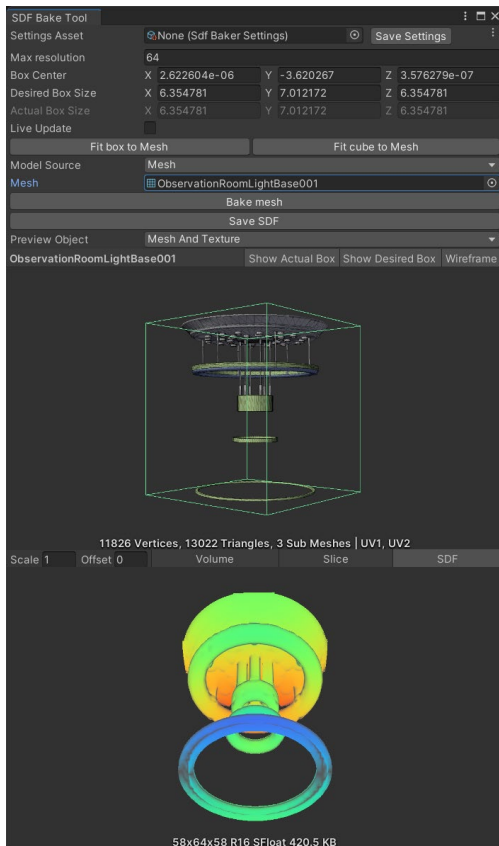


SDF의 시각적 표현

Unity는 SDF 에셋을 3D 텍스처 **블룸 파일(.vf)**로 импорт합니다. 그런 다음 호환되는 VFX Graph 블록과 연산자는 파티클 시스템이 샘플링된 점과 상호 작용하도록 합니다. 이 예시에서 **Conform to Signed Distance Field** 블록은 파티클을 지니의 몸통이 나타나는 영역으로 끌어당깁니다.



VFX Graph에서 SDF 사용



SDF 베이킹 툴

SDF 베이킹 툴

SideFX Houdini 등의 외부 DCC 툴을 사용하여 SDF를 만들 수 있습니다. [VFXToolbox](#) 유틸리티는 **Houdini Volume Exporter**에서 볼륨 파일을 베이킹할 수 있습니다.

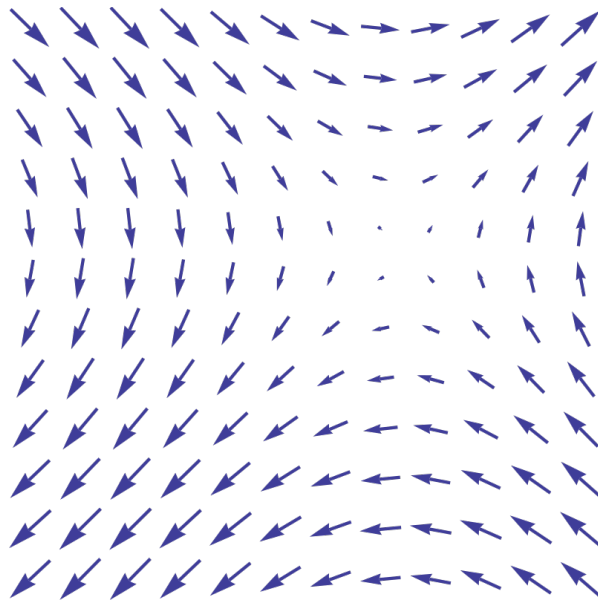
Unity에는 이 프로세스를 간소화하는 유틸리티가 있습니다. SDF 베이킹 툴(**Window > Visual Effects > Utilities > SDF Bake Tool**)은 입력 **메시** (또는 하나의 프리팹에 있는 여러 메시)를 가져와서 3D 텍스처 표현을 생성합니다.

런타임 및 Unity 에디터에서 [SDF Bake Tool API](#)를 사용하여 SDF를 베이킹할 수도 있습니다. 런타임 베이킹에 많은 리소스가 소모되는 점에 유의하세요. 가장 좋은 방법은 저해상도 SDF를 사용하여 특정 순서의 프레임에서만 처리하는 것입니다.

SDF를 생성하고 사용하는 방법은 [VFX Graph의 Signed Distance Fields](#)를 참조하거나 [이 저장소](#)에서 추가 샘플을 살펴보세요.

벡터 필드

벡터 필드는 파티클의 속도 또는 가속도를 제어하는 균일한 벡터 그리드입니다. 아래 그림에서 화살표는 각각의 벡터를 나타냅니다. 벡터의 크기가 클수록 파티클은 벡터를 더 빠르게 통과합니다.



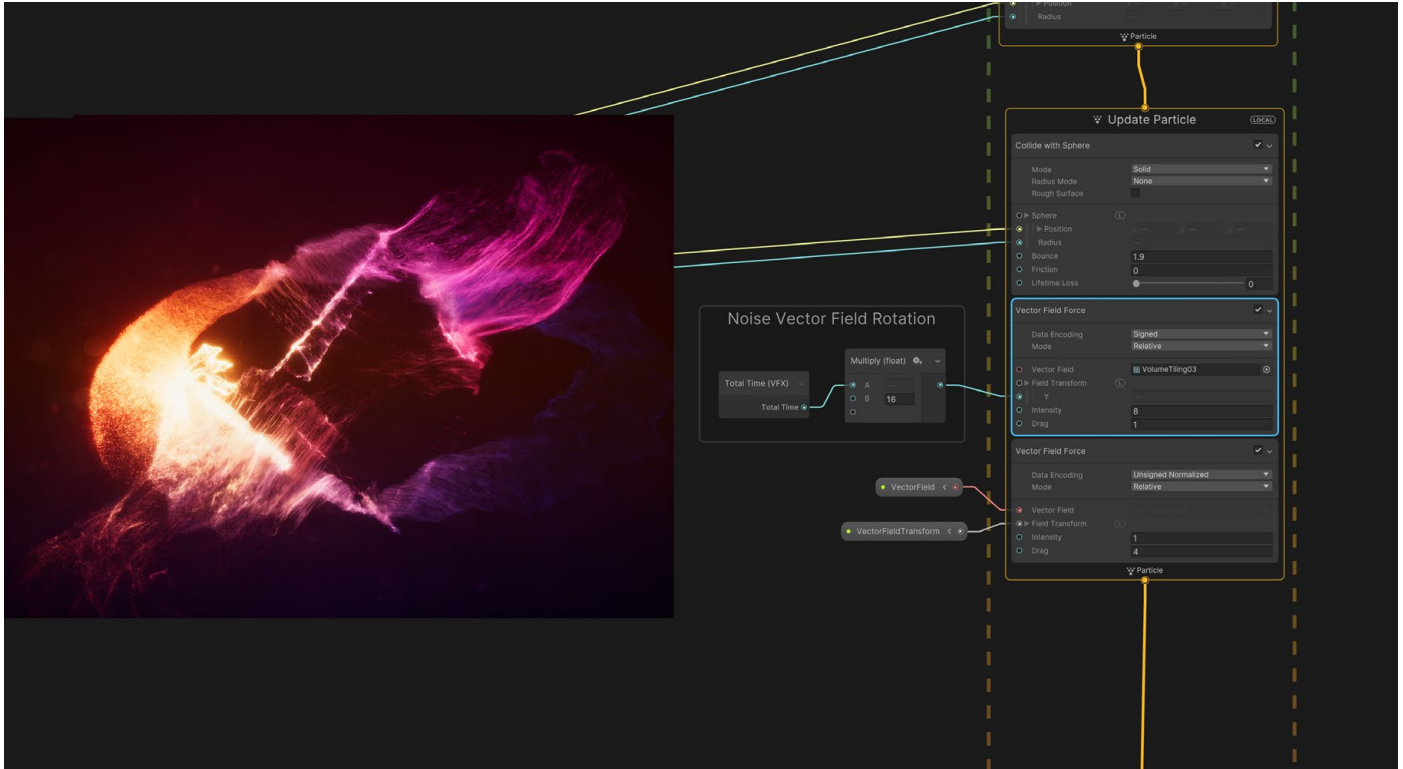
2D 벡터 필드(출처: [Wikipedia](#))

SDF와 마찬가지로, 오픈 소스 [볼륨 파일\(.vf\)](#) 포맷을 사용하여 벡터 필드를 표현하거나, [VFXToolbox](#)에서 함께 제공되는 Houdini VF Exporter에서 벡터 필드를 생성할 수 있습니다. 볼륨 파일 사양을 따르는 자체 VF 파일 익스포터를 작성해도 됩니다.

UnityLogo 씬에서 파티클은 보이지 않는 벡터 필드의 흐름에 의해 밀리듯이 흘러갑니다.

TFlow(에셋 스토어)

TFlow는 [Unity 에셋 스토어](#)에서 다운로드해 사용할 수 있으며, 플립북의 활용도와 품질을 높일 수 있는 모션 벡터 및 모션 블러 제너레이터입니다. [빌트인 파티클 시스템](#) 및 [VFX Graph](#) 모두와 함께 사용할 수 있습니다.



UnityLogo 효과를 구동하는 벡터 필드

VFXToolbox

VFXToolbox에는 Unity 시각 효과 아티스트를 위한 추가 툴이 담겨 있습니다. SideFX의 **Houdini Point Cache Exporter**와 **Volume Exporter**에서 .pCache 파일과 .vf 파일을 익스포트할 수 있습니다.

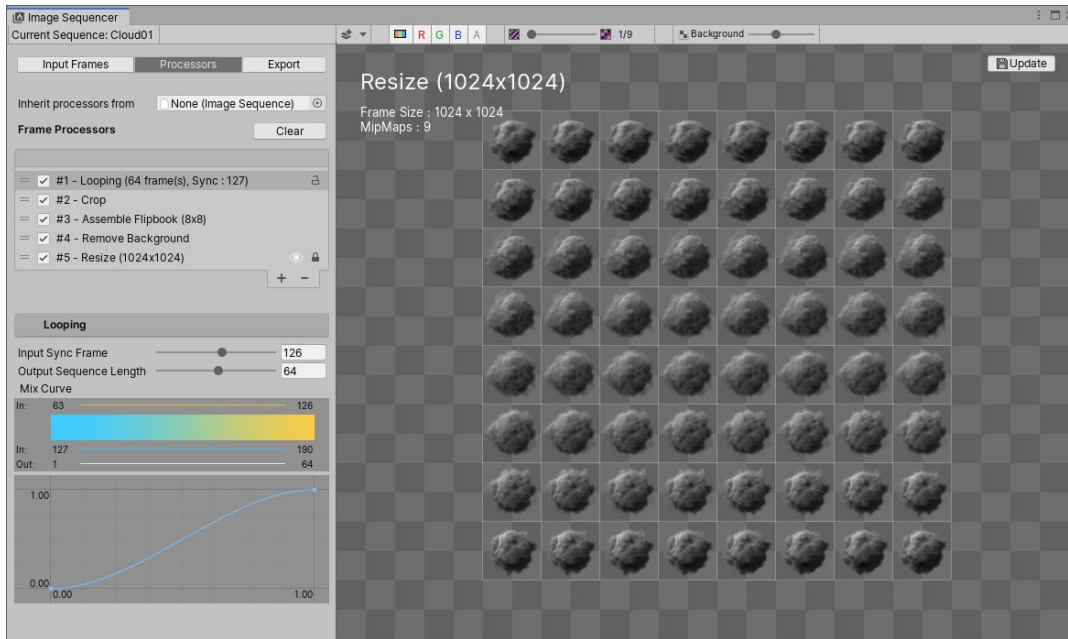
[GitHub](#)의 [저장소](#)를 [다운로드](#)하고 패키지 관리자로 설치해 사용할 수 있습니다.

Image Sequencer

플립북 **텍스처 시트**를 사용하여 애니메이션 효과를 스프라이트로 베이킹할 수 있습니다. 연기, 불, 폭발 등의 효과를 시뮬레이션할 프레임 할당량이 없는 경우, 이미지를 플립북 텍스처 시트로 저장하면 리소스를 많이 사용하지 않고도 비슷하게 베이킹된 효과를 구현할 수 있습니다.

먼저 Unity나 다른 DCC 패키지를 사용하여 효과의 이미지 시퀀스를 프로젝트 폴더에 렌더링합니다. 그런 다음 Image Sequencer를 사용하여 개별 이미지를 하나의 텍스처 시트로 전환합니다. 이미지를 재생하기 전에 [Flipbook Player](#) 블록을 사용하여 원하는 속도로 시간을 변경하고 이미지를 반복합니다.

이 툴을 사용하여 만든 [샘플 플립북](#)도 함께 확인해 보세요.



플립북 텍스처 예시

디지털 콘텐츠 제작 툴

게임을 개발할 때 Unity를 핵심 툴로 사용하기는 하지만, 오직 Unity만 사용하지는 않습니다. 실시간 시각 효과 제작은 복잡한 작업이 될 수 있으므로 Unity 외에도 전문적인 DCC(디지털 콘텐츠 제작) 소프트웨어의 지원을 받는 것이 좋습니다. Unity 및 VFX Graph를 보완하고자 많은 아티스트가 사용하는 몇 가지 툴은 다음과 같습니다.

SideFX Houdini

Houdini는 오래 전부터 시뮬레이션 및 시각 효과의 업계 표준 툴로 사용되고 있습니다. 절차적 워크플로와 노드 기반 인터페이스를 통해 텍스처, 셰이더, 파티클을 비교적 간단하고 간편하게 제작할 수 있습니다. 연산자 중심의 구조라서 비선형 개발에 편하고 3D 제작의 모든 주요 영역에 사용할 수 있습니다.

Autodesk Maya

Maya는 많은 게임 개발 스튜디오의 기반이 되는 툴입니다. 비교적 최근에 출시된 Bifrost 시스템을 사용하면 시각적 프로그래밍 환경에서 물리적으로 정확하고 놀랍도록 세밀한 시뮬레이션을 제작할 수 있습니다.



Blender

Blender는 무료 오픈 소스 3D 제작 툴입니다. 모델링과 리깅부터 애니메이션, 시뮬레이션, 렌더링까지 3D 제작의 모든 측면을 아우르는 기능을 제공합니다. Blender는 크로스 플랫폼 소프트웨어이며 Linux, Windows, macOS 등 어디에서나 원활하게 실행되므로 커뮤니티에서 꾸준히 폭넓게 사용됩니다.

Adobe Photoshop

앞서 설명한 다른 3D 툴과 함께 Adobe Photoshop 같은 이미지 편집 소프트웨어를 유용하게 활용할 수도 있습니다. Photoshop을 사용해 여러 레이어로 래스터 이미지를 편집하고 만들 수 있으며, [마스크](#), [알파 합성](#), 여러 [컬러 모델](#)을 사용할 수 있습니다. Photoshop은 이러한 기능을 지원하는 자체 PSD 및 PSB 파일 포맷을 사용합니다.

위에서 소개한 툴은 물론 DCC 툴의 일부에 불과합니다. 그래프를 제작하면서 DCC 툴을 사용하면 플립북 텍스처나 메시를 비롯한 매우 다양한 요소를 더 원활하게 만들 수 있습니다.

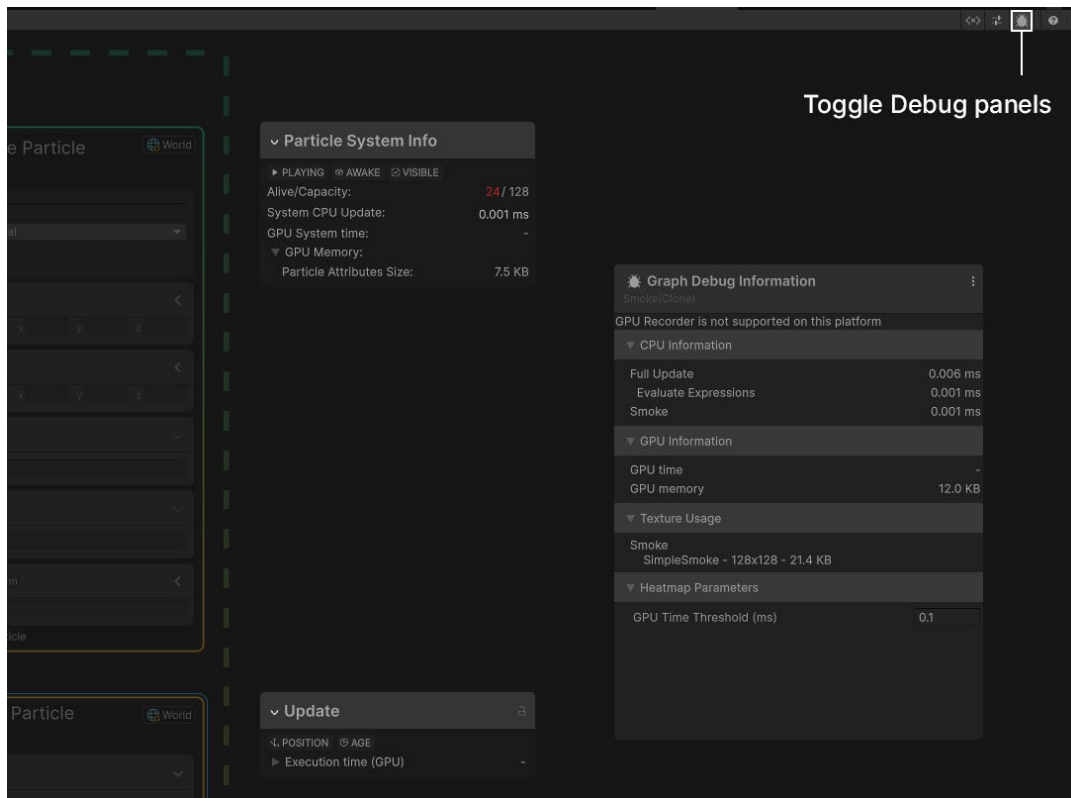
최적화

프로그래머가 코드를 프로파일링하고 성능을 확인하는 것처럼, VFX Graph로 시각 효과를 만든 다음 그래프를 재구성하고 최적화하는 것이 좋습니다. 원하는 효과를 만들었다면 최종 게임이나 애플리케이션에 시각 효과를 배포하기 전에 그러한 효과가 리소스를 지나치게 사용하는 것은 아닌지 확인하세요.

VFX Graph의 Profiling 및 Debug 패널

Unity 6에는 실행 중인 시각 효과에 대한 필수 정보를 제공하는 Profiling 및 Debug 패널이 포함되어 있습니다. 이러한 톨은 CPU 및 GPU 타이밍, 메모리 사용량, 텍스처 사용량, 다양한 상태 등의 정보를 제공할 수 있습니다. 이를 사용하여 VFX Graph의 성능을 모니터링하고 최적화하세요.

Visual Effect Graph 창의 오른쪽 상단에 있는 디버그 아이콘을 클릭합니다. 이 패널은 VFX Graph 창의 Visual Effect 컴포넌트가 있는 게임 오브젝트에 연결된 경우에만 작동합니다.



Profiling 및 Debug 패널을 사용하여 VFX Graph를 최적화하세요.

Graph Debug Information 패널

Graph Debug Information 패널은 다음 정보를 제공합니다.

- **CPU Information:** CPU Information 패널에는 전체 그래프의 전체 업데이트 시간(밀리초), CPU에서 계산된 그래프 파라미터 평가 시간, 그래프 내 특정 시스템의 업데이트 시간(밀리초) 같은 중요한 성능 지표가 표시됩니다.
- **GPU Information:** GPU Information 패널에는 GPU에서 VFX Graph의 실행 시간(밀리초)과 GPU 메모리 사용량이 표시됩니다.

- **Texture Usage:** 사용된 텍스처와 그 크기, 메모리 크기가 나열됩니다.
- **Heatmap Parameters:** GPU 시간 임계값을 조정하여 그래프에서 비용이 많이 드는 부분을 강조 표시합니다.

오른쪽 상단 세로 줄임표() 메뉴에서 **렌더링 디버거**(URP/HDRP) 및 **Unity 프로파일러**에 액세스할 수도 있습니다.

Particle System Info 패널

Particle System Info 패널은 각 시스템의 Initialize 컨텍스트에 연결되어 있습니다. 이 패널은 특정 시스템과 관련된 정보를 제공합니다.

- **Particle state:** 시스템이 재생/일시 정지, 활성화/절전, 표시/컬링 중인지 보여 줍니다.
- **Alive/capacity:** Initialize 컨텍스트에서 활성 상태인 파티클의 수와 사용자가 설정한 용량을 나타냅니다. 이 용량을 최대 활성 파티클 수에 최적화하면 메모리 할당 공간을 절약하는 데 도움이 됩니다.

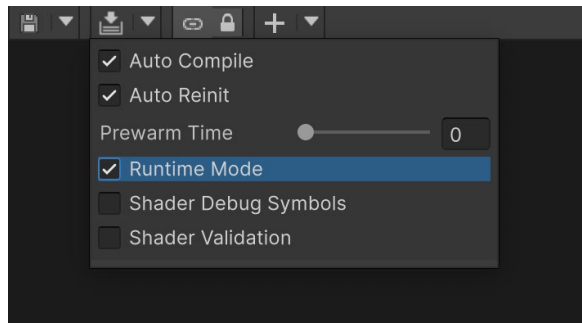
Context Debug 패널

컨텍스트를 선택하면 이 패널이 나타납니다. 오른쪽 상단의 자물쇠 기호를 사용하여 화면에 고정할 수 있습니다. 이 패널은 컨텍스트마다 다른 정보를 표시할 수 있습니다.

- **Spawn 컨텍스트:** 생성자가 재생 중인지 또는 중지되었는지 나타냅니다.
- **Initialize, Update, Output 컨텍스트:** 이 패널은 Update 컨텍스트에서 암시적으로 업데이트된 속성을 표시하고, 작업별로 GPU 실행 시간을 세분화하며, 텍스처 사용량과 함께 치수 및 메모리 크기를 나열합니다.

이 패널의 프로파일링 타이밍은 에디터에 기록되므로 정확한 측정값이 아니라 비교용이라는 점에 유의하세요. 병목 현상을 파악하고 성능을 최적화하는 데 사용하기를 추천합니다.

또한 툴바에서 Runtime Mode를 활성화하여 런타임 컴파일을 실행할 수 있습니다. 이렇게 하면 효과가 저작 버전이 아닌 최적화된 런타임 버전을 사용합니다. 효과를 완전히 다시 저작할 수 있도록 유지하려면 이 옵션을 선택 취소합니다.



툴바에서 Runtime Mode를 활성화합니다.

참고: Apple Silicon에서는 GPU 실행 타이밍을 사용할 수 없으며, 프로파일링 패널은 연결된 시각 효과에 대한 인스턴싱을 비활성화합니다. 정확한 성능 평가를 위해서는 항상 Unity 프로파일러를 사용하여 타겟 디바이스에서 프로파일링하세요.

Unity 프로파일러 및 프레임 디버거

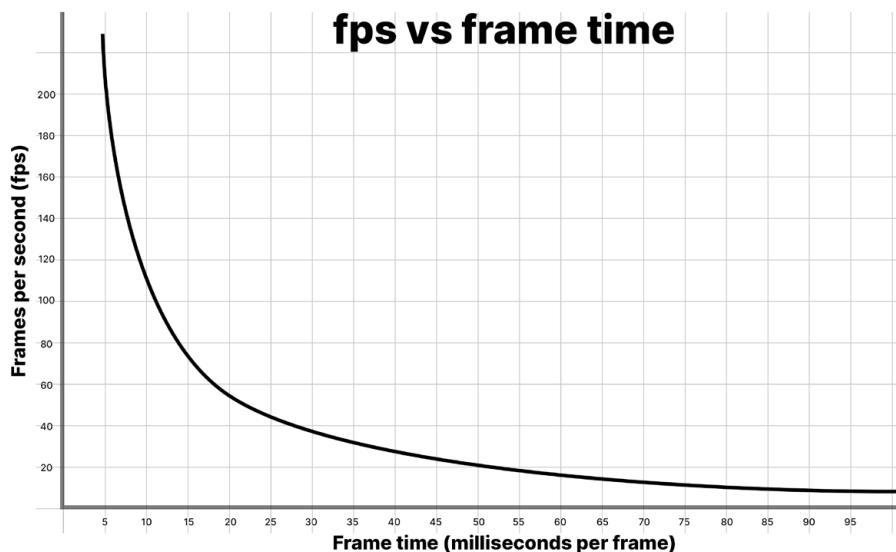
Unity 프로파일러 (Window > Analysis > Profiler) 및 **프레임 디버거** (Window > Analysis > Frame Debugger)를 사용하여 그래프를 최적화하고 성능을 개선할 수 있습니다. 그런데 Unity 에디터가 프로파일링 정보에 영향을 미쳐 부정확한 결과가 나올 수도 있습니다.

실제 성능을 측정해야 하는 경우 **Profiler Standalone Process** 옵션을 사용하거나 별도의 빌드를 만들어야 합니다. **그래픽스 성능의 기초**를 고려하여 높은 프레임 속도를 유지하고, 궁극적으로 플레이어에게 최상의 경험을 제공해야 합니다.



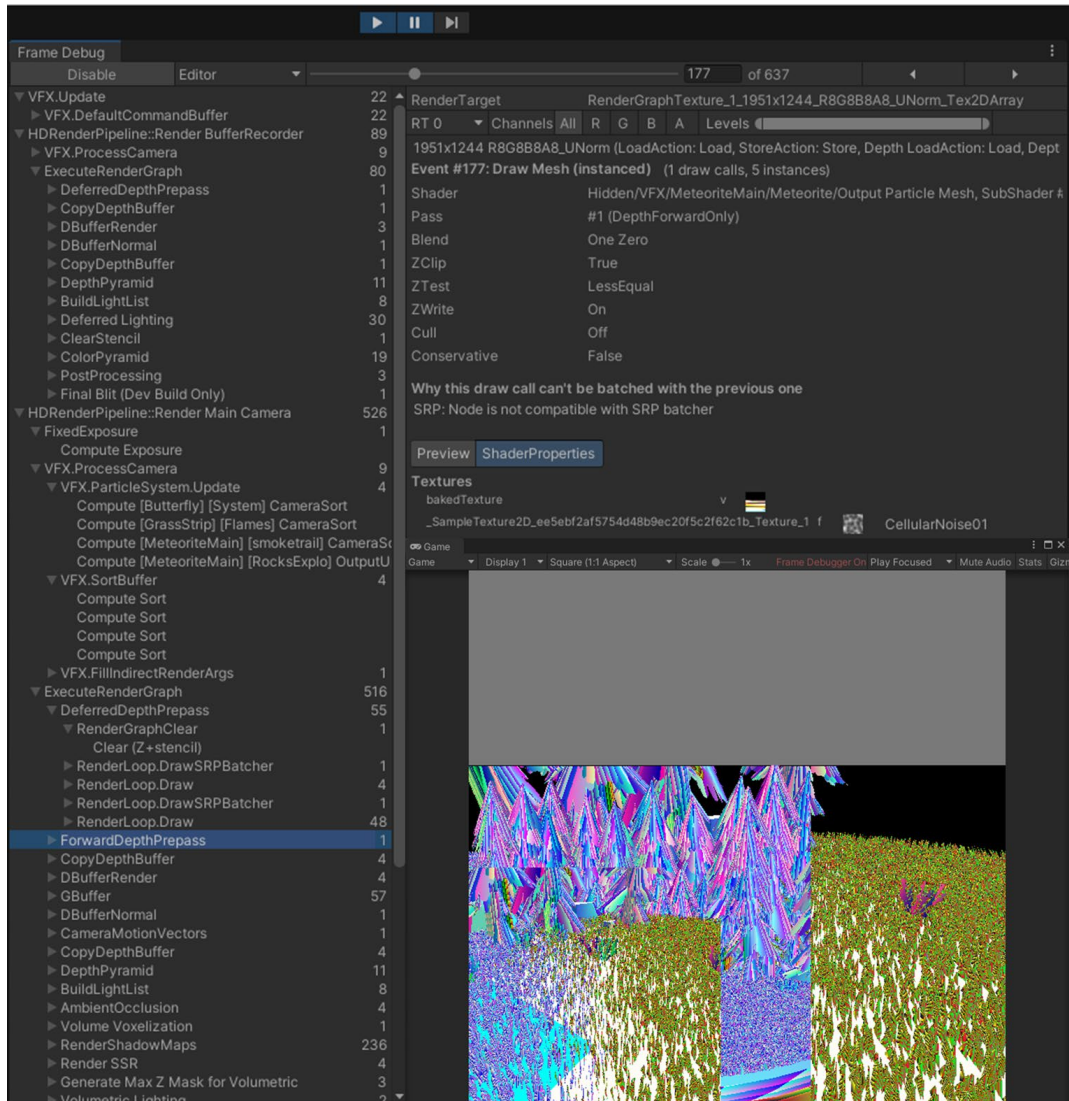
Unity 프로파일러

렌더링 통계를 살펴볼 때는 초당 프레임 수보다 프레임당 시간 비용을 확인해야 합니다. FPS는 비선형적이므로 벤치마크로 사용하기에는 오해의 소지가 있을 수 있습니다(아래 그래프 참조). 목표가 60FPS라면 프레임당 16ms를 프레임 할당량으로 사용하고, 30FPS라면 프레임당 33ms를 사용해야 합니다.



프레임 시간을 최적화 가이드로 사용하세요.

프레임 디버거에는 드로우 콜 정보가 표시되므로 프레임 구성 방법을 제어할 수 있습니다.



프레임 디버거

위 이미지의 왼쪽 패널에는 드로우 콜 외에도 여러 렌더링 이벤트의 시퀀스가 계층적으로 정렬되어 있습니다. 한편 오른쪽 패널에는 셰이더 패스 및 텍스처를 포함하여 선택한 드로우 콜의 세부 정보가 표시됩니다. 이를 통해 프레임을 살살이 살펴보고 Unity가 리소스를 어디에 사용하고 있는지 확인할 수 있습니다.

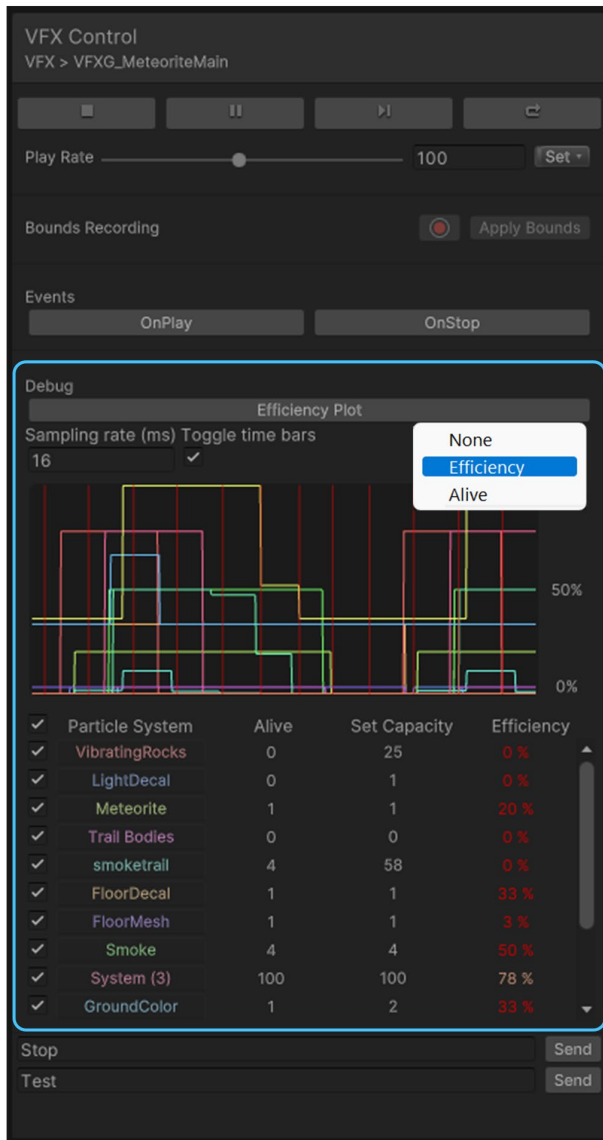
다음 요소는 일반적으로 리소스를 많이 사용하므로 효과를 최적화할 때 꼭 확인하는 것이 좋습니다.

- **텍스처 크기:** 카메라에 근접하는 일이 없는 에셋이라면 해상도를 낮춰야 합니다.
- **용량:** 파티클 수가 적을수록 리소스 사용량이 줄어듭니다. 초기화 블록에서 용량을 설정하여 시스템의 최대 파티클 수를 제한하는 것이 좋습니다.
- **가시성 및 수명:** 일반적으로 화면에서 보이지 않는 것이 있다면 비활성화하는 것이 좋습니다.

VFX Control의 디버그 모드

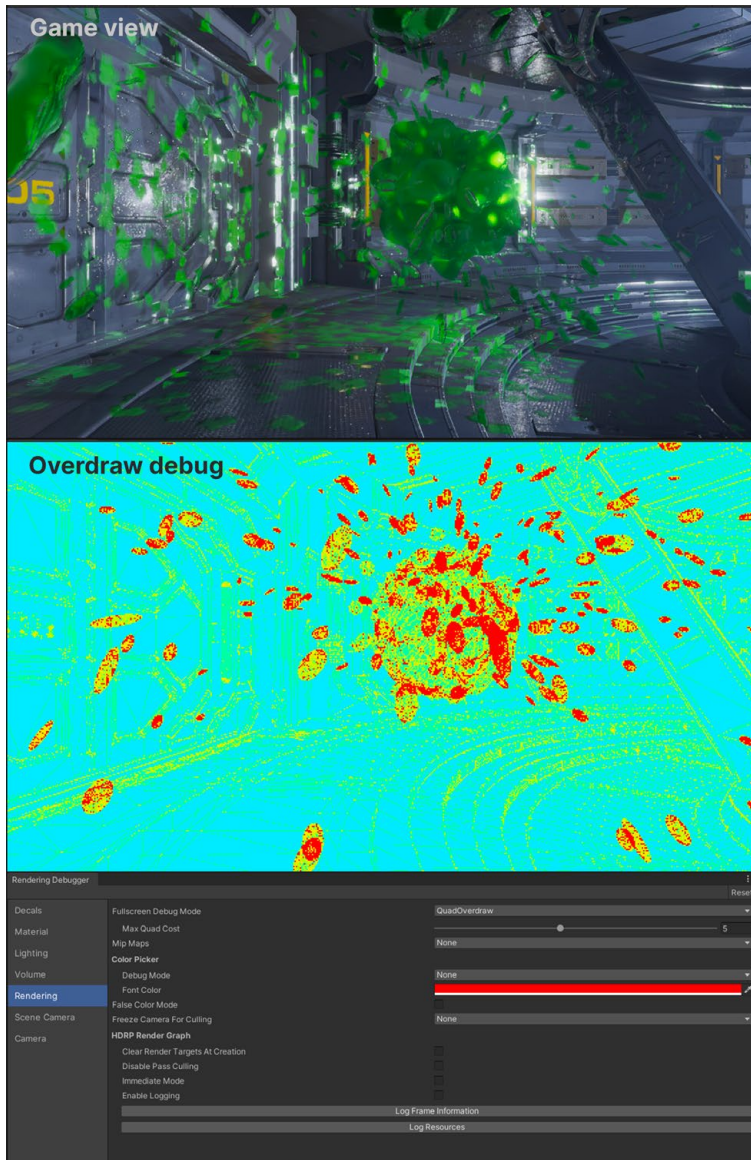
VFX 제어판의 디버그 모드를 사용해 파티클 수명과 용량을 결정할 수 있습니다. 이 설정은 성능과 메모리 사용량에 영향을 줄 수 있습니다. 인스펙터에서 Visual Effect 인스턴스를 편집한 다음, Debug 옵션을 **Alive** 또는 **Efficiency**로 설정하세요.

그 결과에 따른 플롯은 얼마나 많은 파티클이 남아 있는지 또는 그 수가 시스템의 설정 용량과 비교하여 얼마나 다른지 표시합니다. 파티클 수와 용량 설정을 조정하여 VFX Graph의 효율성을 개선할 수 있습니다.



VFX Control의 디버그 모드

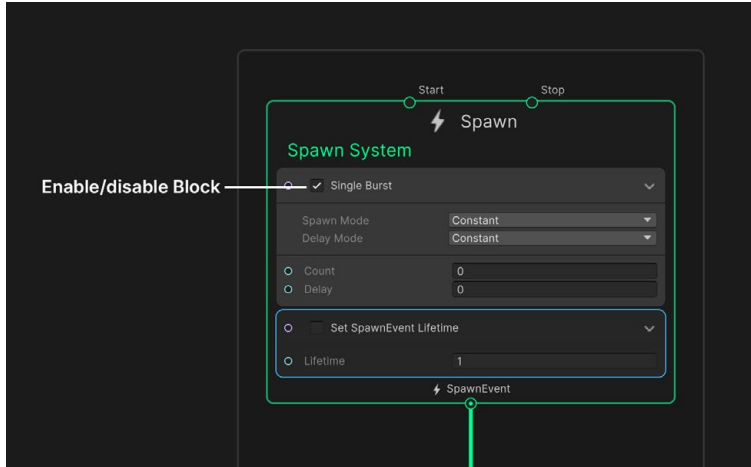
- **연산자 및 메모리:** 불필요한 연산자를 단순화해야 합니다. 눈에 띄게 다르지 않다면 반복 작업을 줄여야 합니다.
- **플립북:** 모든 파티클을 시뮬레이션하는 대신 특정 효과를 텍스처 플립북에 미리 렌더링해 두는 편이 좋습니다. 그런 다음 전체 시뮬레이션이 필요하지 않은 상황에서는 항상 애니메이션 텍스처를 재생하면 됩니다.
- **메시 크기:** 파티클이 출력 메시라면 폴리곤 수를 줄여야 합니다.
- **초과 오버드로우:** 투명한 표면이 여러 개인 경우 렌더링 리소스가 소모됩니다. 렌더링 디버거(**Window > Analysis > Rendering Debugger**)를 사용하여 초과 오버드로우를 확인하고 그래프를 적절히 조정하세요. 또한 가능하면 팔각형 파티클을 사용하는 것이 좋습니다.



오버드로우(빨간색)를 최소화하여 성능 개선

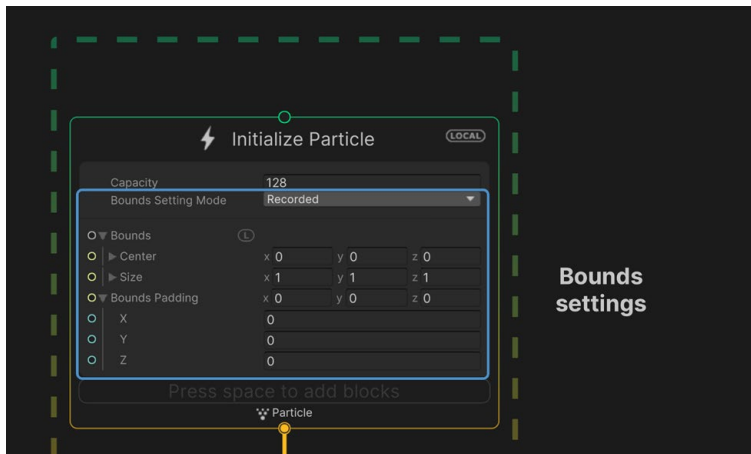
성능 문제를 해결할 때 오른쪽 상단의 체크박스를 사용하여 각 블록을 활성화 또는 비활성화할 수 있습니다. 이를 통해 빠르게 A/B 테스트를 수행하여 전후 성능을 측정하고 그래프의 일부를 분리할 수 있습니다. 테스트를 완료하면 블록을 다시 활성 상태로 복원해야 합니다.

테스트를 위해 블록을 비활성화/활성화합니다.



경계

시각 효과의 경계는 가시성에 기반한 빌트인 최적화로 구성됩니다. 모든 Initialize 컨텍스트에서 다음과 같은 여러 설정을 보셨을 것입니다.



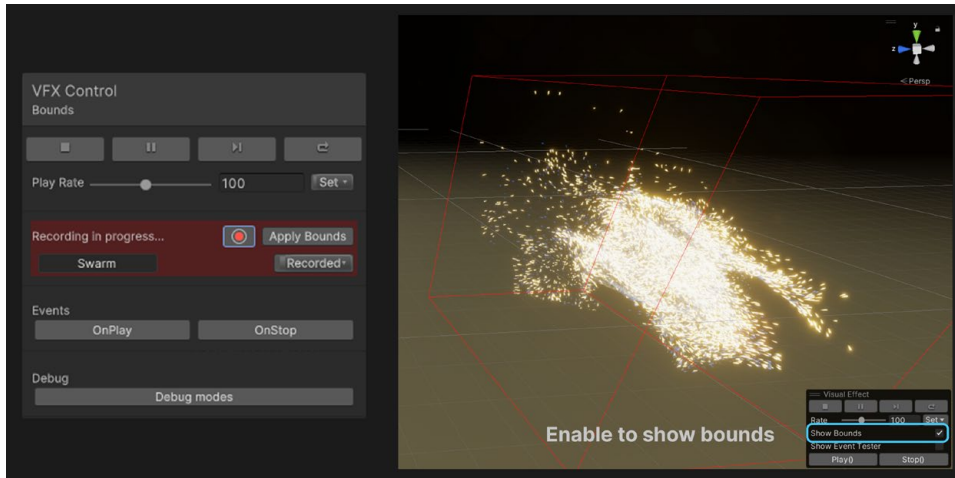
Bounds 설정을 사용하여 효과가 렌더링되는 위치를 정의합니다.

카메라가 경계를 볼 수 없다면, Unity는 효과를 컬링하여 렌더링하지 않습니다. 다음 가이드라인을 따라 각 시스템의 경계를 설정할 수 있습니다.

- 경계가 너무 크면 개별 파티클이 화면 밖으로 나가더라도 카메라가 시각 효과를 처리하며, 이는 리소스 낭비로 이어집니다.
- 경계가 너무 작으면 일부 파티클이 아직 화면 내에 있는 경우에도 Unity가 시각 효과를 컬링할 수 있습니다. 그러면 팝핑(popping) 현상이 발생할 수 있습니다.

기본적으로 Unity는 각 시스템의 경계를 자동으로 계산합니다. 하지만 Bounds Setting Mode를 다음과 같이 변경해도 됩니다.

- **Automatic:** Unity가 경계를 확장하여 효과를 계속 표시합니다. 이 옵션이 가장 효율적이지 않다면, 아래 옵션 중 하나를 사용하여 경계를 최적화해야 합니다.
- **Manual:** Bounds와 **Bounds Padding**을 사용하여 Initialize 컨텍스트에서 영역을 정의합니다. 간단한 옵션이지만 시스템 전체에 적용하는 데 오랜 시간이 걸립니다.
- **Recorded:** 이 옵션을 사용하면 VFX Control 패널에서 경계를 녹화할 수 있습니다. 경계는 녹화 시 빨간색으로 표시되며 효과를 재생하는 동안 확장됩니다. **Apply Bounds**를 누르면 크기가 저장됩니다.



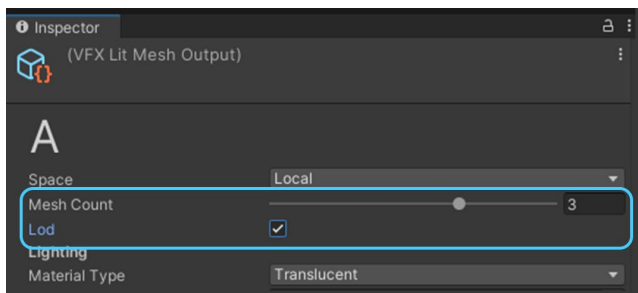
런타임에 경계 녹화

Manual 또는 **Recorded** 모드에서 런타임에 연산자를 사용하여 각 시스템에 대한 경계를 계산할 수 있습니다. Initialize 컨텍스트에는 Bounds Padding 입력이 포함되며, 이 Vector3를 사용하여 Bounds 값을 키울 수 있습니다.

메시 LOD

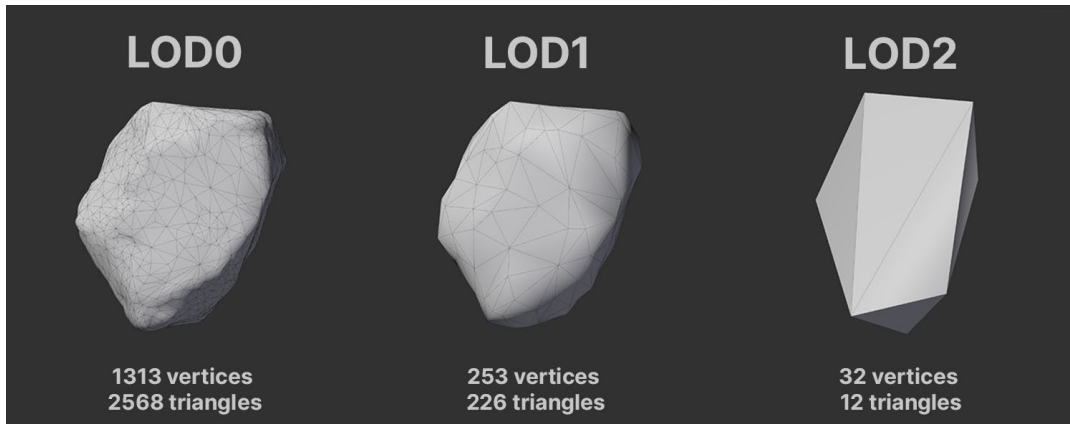
파티클이 메시를 출력하는 경우 **디테일 수준(LOD)**을 활용하는 것이 좋습니다. 여기에서 원거리 파티클에 더 간단한 메시를 직접 지정할 수 있습니다.

파티클 메시 출력에는 인스펙터에 표시되는 **Mesh Count** 파라미터가 있으며, 이를 통해 출력당 최대 4개의 메시를 지정할 수 있습니다. 이 파라미터와 LOD 체크박스를 함께 사용하면 화면에 표시되는 메시의 크기에 따라 메시를 자동으로 전환할 수 있습니다.



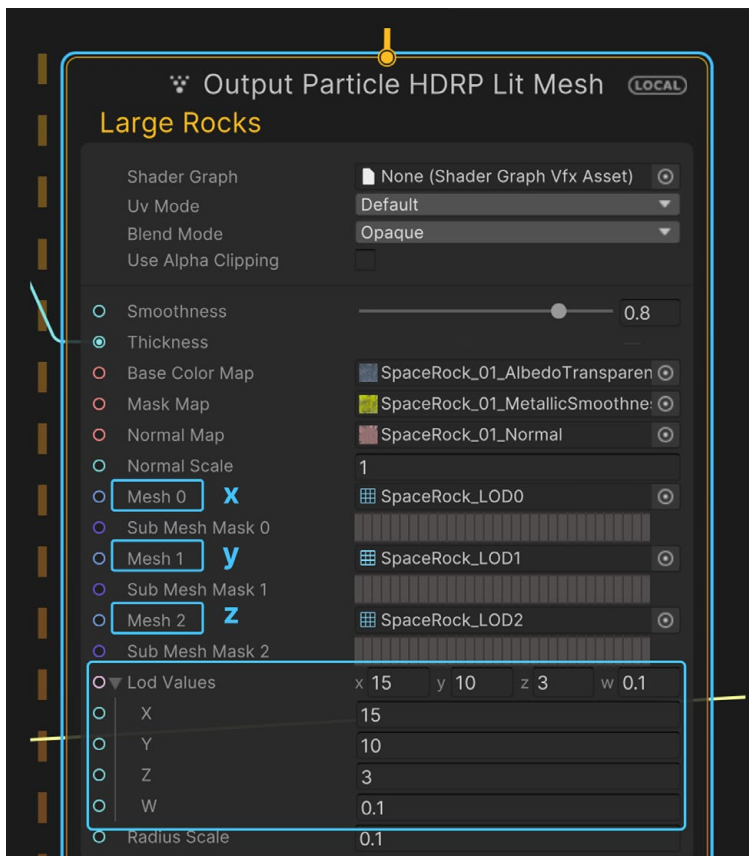
인스펙터의 Mesh Count 및 LOD 설정

고해상도 모델은 Output 컨텍스트의 LOD 값에 설정된 스크린 공간 백분율에 따라 저해상도 모델로 전환될 수 있습니다.



LOD 해상도

이 예시에서는 메시가 화면의 15% 미만을 차지할 때 SpaceRock_LOD0 모델이 더 작은 SpaceRock_LOD1 모델로 바뀝니다.



LOD 값

메시 파티클을 대량으로 만드는 경우 수백만 개의 폴리곤을 렌더링할 필요가 없으며, 따라서 프레임 시간을 크게 단축할 수 있습니다.

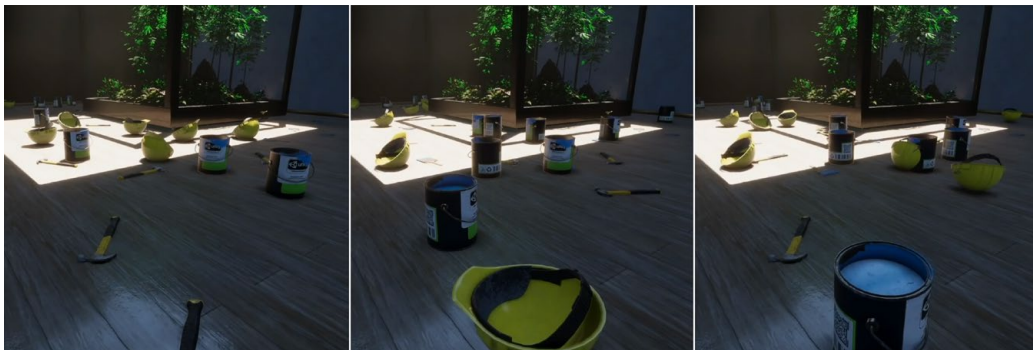


렌더링 리소스를 줄이는 메시 LOD

[이 프로젝트](#)의 PlanetaryRing 예시에서 메시 LOD를 직접 확인해 보세요.

Mesh Count

Mesh Count를 LOD 없이 비슷하게 활용할 수 있습니다. 여기서는 다음과 같이 여러 개의 메시를 사용해 무작위성을 추가합니다. Output Particle Lit Mesh의 네 가지 메시를 사용하면 바닥에 다양하게 흩어진 프랍(prop)을 만들 수 있습니다.



메시 파티클을 무작위화하는 Mesh Count

파티클 렌더링

목표 프레임 속도와 프레임 할당량을 달성하려면 파티클이나 메시를 렌더링할 때 다음 최적화 팁을 고려해야 합니다.

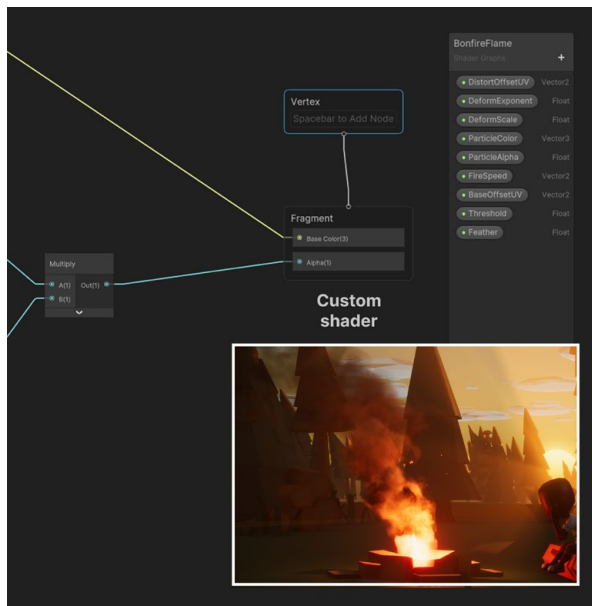
- **삼각형 파티클:** 사각형 파티클의 지오메트리 절반으로, 빠르게 움직이는 효과와 대량의 파티클 렌더링에 효과적입니다.
- **단순화된 조명:** 완전한 **빛 HDRP** 셰이더가 필요하지 않다면 리소스 사용량이 적은 셰이더로 전환하세요. Shader Graph에서 출력을 커스터마이징하여 특정 효과에 필요하지 않은 기능을 삭제할 수 있습니다. 예를 들어 Bonfire 샘플 씬은 스타일라이즈드 Shader Graph를 사용하는데, 이는 출력을 크게 단순화합니다.



출력 파티클 프리미티브(사각형, 삼각형, 팔각형)

다음 팁은 HDRP에만 해당합니다.

- **저해상도 투명도:** **HDRP Rendering** 프로퍼티에서 Low Res Transparency를 활성화하면 투명 파티클을 더 낮은 해상도로 렌더링할 수 있습니다. 그러면 파티클이 약간 흐리게 표시되는 대신 성능을 4배 향상할 수 있습니다. 적절하게 사용하면 최대 해상도로 렌더링한 결과와 거의 구별할 수 없습니다.
- **팔각형 파티클:** 팔각형 파티클은 사각형 파티클의 모서리를 자른 형태입니다. 파티클 텍스처의 모서리가 투명한 경우, 팔각형 파티클을 사용하면 오버드로우를 줄이거나 방지할 수 있습니다. 투명 영역이 겹치더라도 여전히 약간의 계산은 필요하므로, 팔각형 파티클을 사용하는 경우 사각형의 모서리가 교차하는 부분의 불필요한 컴퓨팅을 줄일 수 있습니다.



시각 효과 샘플의 Bonfire 씬

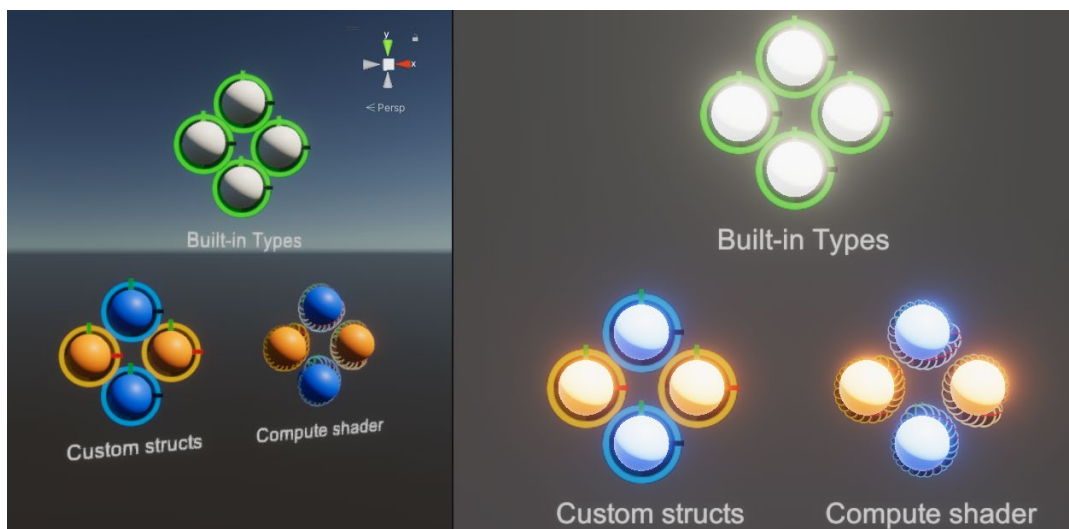
고급 기능

그래픽스 버퍼 지원

그래픽스와 컴퓨트 버퍼를 통해 이제 C#에서 VFX Graph로 대량의 데이터를 더 쉽게 처리하고 전송할 수 있습니다.

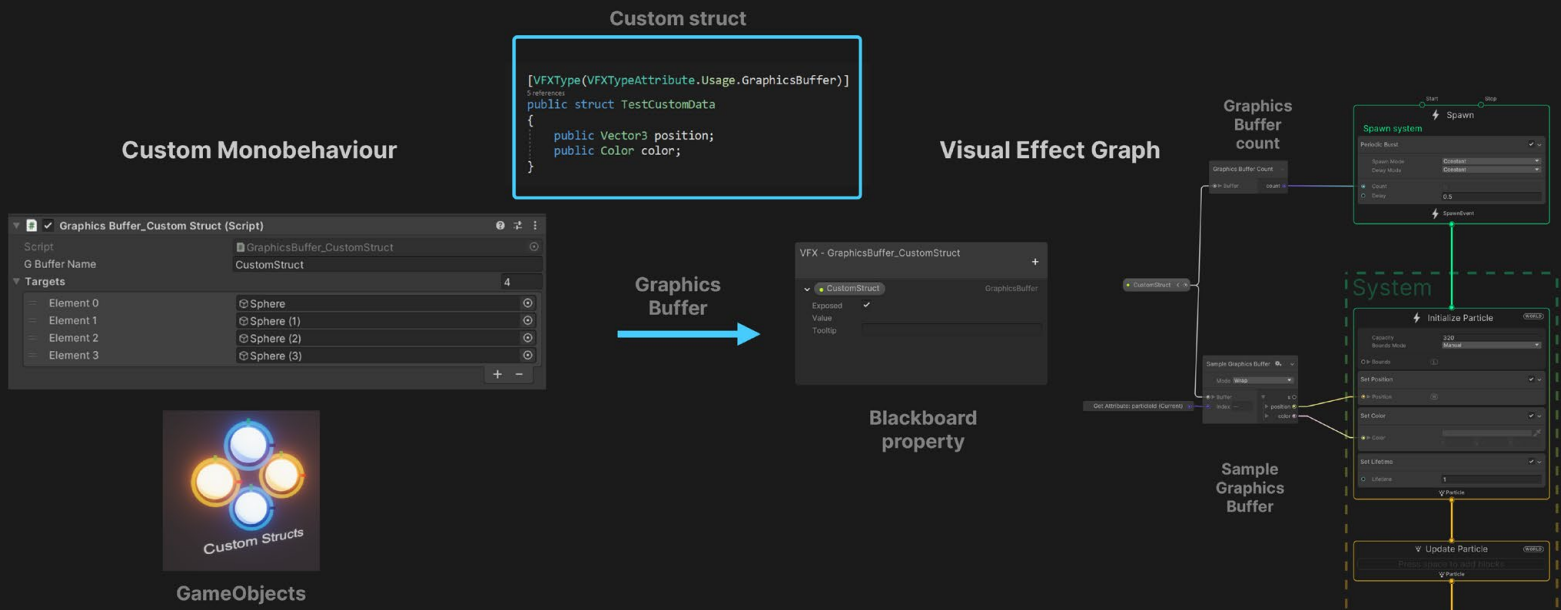
그래프에서 여러 게임 오브젝트 위치를 추적하는 경우, 스크립팅을 통해 시뮬레이션에서 해당 정보에 액세스하도록 할 수 있습니다. 그래픽스 버퍼는 앞의 샘플에서 볼 수 있듯이 텍스처 내에 데이터를 저장하는 기능과 유사한 대체 기능을 제공합니다.

이 예시에서는 게임 오브젝트의 위치와 컬러 데이터를 빌트인 유형, 커스텀 구조체, 컴퓨트 셰이더에서 VFX Graph로 전달합니다.



그래픽스 버퍼는 게임 오브젝트 데이터를 VFX Graph로 전송합니다.

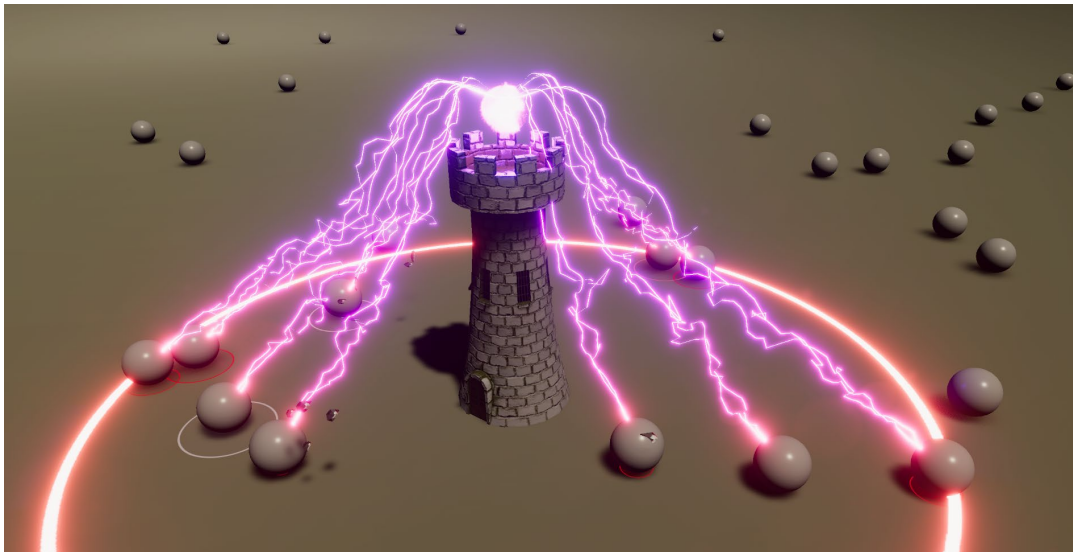
스크립트를 사용하여 그래픽스 버퍼를 정의하고 게임 오브젝트의 데이터로 채울 수 있습니다. 그런 다음 Blackboard 프로퍼티를 통해 VFX Graph에 전달하면 됩니다.



커스텀 구조체를 통한 데이터 전달

이 예제는 보다 복잡한데, 전기 타워가 그래픽스 버퍼를 통해 접근하는 구체의 위치에 액세스할 수 있습니다. 구체가 많으면 현실적으로 모든 게임 오브젝트에 프로퍼티를 노출할 수 없게 됩니다.

Blackboard에는 그래픽스 버퍼 내에서 사용할 수 있는 단 하나의 커스텀 구조체가 있습니다. 이 타워는 잠재적으로 수백 개의 타겟에 도달하여 단 몇 개의 연산자로 타겟의 데이터에 액세스할 수 있습니다.



그래픽스 버퍼 사용 사례



이를 통해 보이드(boid), 유체, 헤어 시뮬레이션, 군중 등 복잡한 시뮬레이션 상황에서 VFX Graph가 씬 내에서 어떤 식으로 상호 작용할 수 있는지 볼 수 있습니다. 그래픽스 버퍼를 사용하려면 C# API에 대한 지식이 필요하지만, 데이터를 게임 오브젝트와 훨씬 편리하게 교환할 수 있습니다.

이 [프로젝트](#)를 통해 Unity에서 VFX Graph와 그래픽스 버퍼를 함께 사용하는 다른 예시도 살펴보세요.

Custom HLSL

이제 숙련된 VFX 아티스트와 개발자는 [Custom HLSL 블록](#)을 활용할 수 있습니다. 이 기능을 사용하면 Unity에서 아직 기본으로 지원되지 않는 고유한 효과를 만들 수 있습니다. Custom HLSL을 사용하면 고급 물리 시뮬레이션, 군집 동작, 실시간 데이터 시각화를 생성할 수 있습니다.

Custom HLSL 노드를 사용하면 파티클 시뮬레이션 중에 커스텀 HLSL 코드를 실행할 수 있습니다. 컨텍스트에서 수평 플로에는 연산자를, 수직 플로에는 블록을 사용할 수 있습니다.

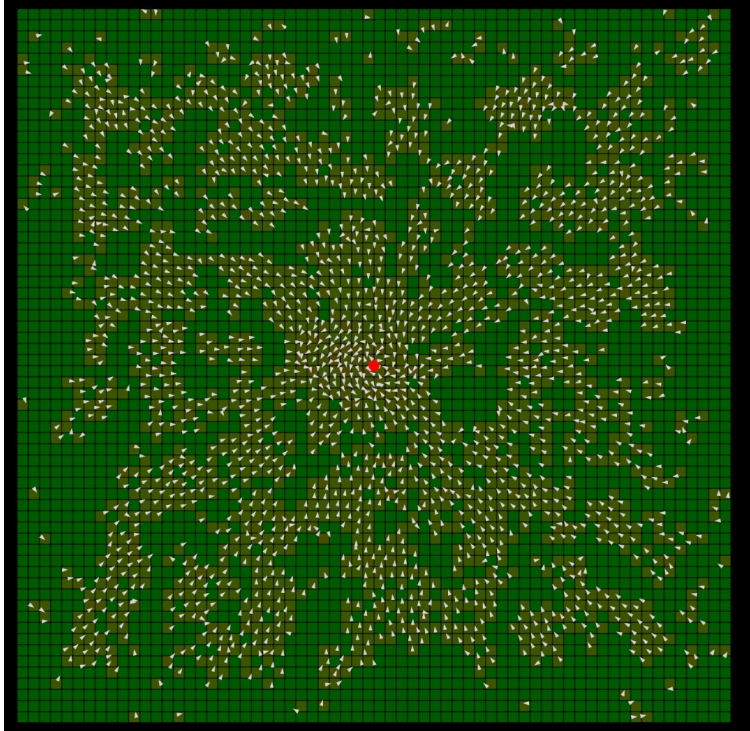
VFX Graph와 호환되려면 Custom HLSL 블록이 다음 요구 사항을 충족해야 합니다.

- 반환 유형은 `void`입니다.
- 액세스 한정자 `inout`이 있는 `VFXAttributes` 유형의 파라미터가 하나 있어야 합니다.

Custom HLSL 노드를 생성한 다음 HLSL 코드를 포함하거나 HLSL 텍스트 파일을 소싱합니다. 전체 요구 사항은 이 [기술 자료 페이지](#)를 참조하세요.

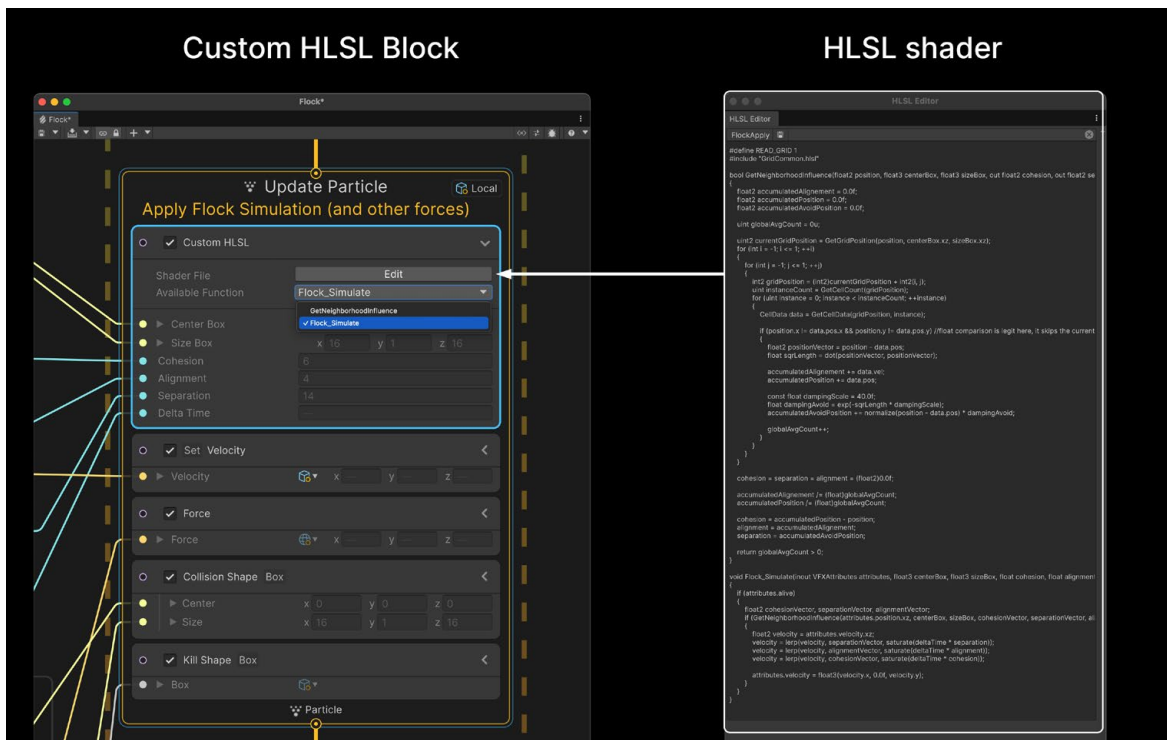
로우레벨 HLSL 코드를 작성하는 것이 복잡한 VFX Graph 노드 네트워크를 사용하는 것보다 더 효율적일 수 있습니다. 특정 계산을 HLSL에서 직접 구현하면 특히 계산을 많이 사용하는 효과의 성능을 향상시킬 수 있습니다.

Custom HLSL은 수학적 함수, 노이즈 알고리즘, 기타 절차적 기법을 기반으로 파티클 속성이나 동작을 절차적으로 생성하는 데 사용할 수 있습니다. 이를 통해 더 유기적이거나 다양한 효과를 만들 수 있죠.



Custom HLSL은 절차적 동작을 생성할 수 있습니다.

이 프로젝트는 이웃 검색을 사용하여 Custom HLSL 블록 및 그래픽스 버퍼를 사용하는 Visual Effect Graph로 2D 군집을 시뮬레이션하는 방법을 보여 줍니다(GridManager 스크립트 참조).

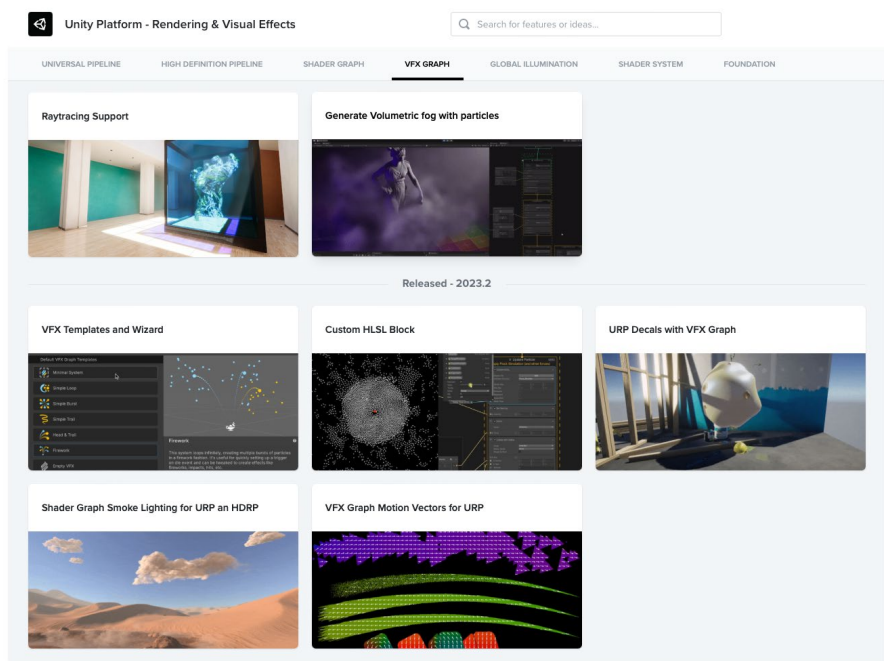


이 Custom HLSL은 군집을 만들기 위해 이웃 검색을 구현합니다.

추가 리소스

본 가이드가 VFX Graph와 Unity의 실시간 시각 효과 툴셋을 더 깊게 파악하는 데 도움이 되었기를 바랍니다. 유니티의 사명은 모든 크리에이터가 만들고 싶은 아트를 만들 수 있도록 돕는 것입니다.

VFX Graph를 사용하면 초현실적인 시뮬레이션과 놀라운 그래픽으로 플레이어의 마음을 사로잡을 수 있습니다. 완성도 높은 작품을 만들어 보세요.



그래픽스 제품 로드맵



VFX Graph에 대해 학습할 수 있는 여러 추가 자료를 아래에 소개합니다.

동영상 튜토리얼

- [VFX Graph로 놀라운 VFX 만들기](#): 자체 VFX Graph를 설정하는 데 필요한 기본을 다룹니다.
- [아티스트의 창작을 지원할 기능](#): 이 동영상에서는 메시 LOD, 그래픽스 버퍼, Shader Graph 연동 등 최근에 업데이트된 VFX Graph 기능을 중점적으로 소개합니다.
- [VFX Graph: 스페이스쉽 데모에서 시각적 요소 구축](#): 이 세션에서는 스페이스쉽(Spaceship) 데모에 사용된 여러 기법을 살펴봅니다.
- [VFX Graph로 포털 효과 만들기](#): VFX Graph를 사용하여 파티클 링을 더 역동적인 효과로 변환하는 방식을 통해 포털 효과를 생성합니다.



GitHub의 VFX 프로젝트

오디오 반응 효과, 데이터 시각화, 포인트 클라우드, 볼류메트릭 데이터 재생 등 VFX Graph를 사용하여 만들 수 있는 더 다양한 효과를 살펴보세요. 유니티의 시니어 크리에이터 애드버킷인 케이지로 다카하시가 제작한 아래의 프로젝트를 확인해 보세요.

- 카메라 트래킹 + Lidar + VFX Graph: [여기](#) 및 [여기](#)
- [포인트 클라우드\(pcx\) + VFX Graph](#)
- [Azure Kinect + VFX Graph](#)
- [Intel RealSense 카메라 + VFX Graph](#)
- [DepthKit + VFX Graph](#)
- [DepthKit 볼류메트릭 동영상 + VFX Graph](#)
- [4DViews 볼류메트릭 동영상 + Alembic + VFX Graph](#)
- [Alembic + HAP 동영상 + VFX Graph](#)
- [오디오 반응 + VFX Graph](#)
- [VFX Graph + Midi 컨트롤러](#)
- [스킨드 메시 샘플링](#)
- [SDF + VFX Graph](#)
- [VAT\(버텍스 애니메이션 텍스처\) + VFX Graph](#)
- [그래픽스 버퍼 테스트](#)
- [그래픽스 버퍼를 사용한 지리 데이터 렌더링 + VFX Graph](#)
- [덱스오브필드\(피사계심도\) 파티클 샘플](#)
- [VFX Graph를 사용한 절차적 모델링](#)
- [VFX Graph + 절차적 Shader Graph 스프라이트 생성 셰이더](#)
- [VFX Graph 테스트 씬](#)
- [VFX Graph 인터랙티브 불꽃놀이](#)
- [초밥 + VFX Graph](#)



unity.com/kr