[Unity] Shatter Stone Documentation

Version: 3.0

Last Update: 16 June 2025

Introduction

What is *Shatter Stone*

Shatter Stone is a modular asset series for Unity that enables destructible resource nodes—stone, ore, metal, gemstone, and crystal—with a unified interaction and pickup framework. Designed for flexibility and extensibility, each pack in the series adds new themed content (e.g., metals, gems, stylized variants) while sharing a common logic base for interaction, fragmentation, and resource collection.

Shatter Stone is built around the concept of modularity. Every pack adds new content (e.g., ores, metals, crystals) but does not require additional scripts or setup changes—meaning you can drop in new packs without worrying about compatibility.

Whether you're building an RPG mining system, a survival crafting mechanic, or just adding rich visual feedback to resource harvesting, *Shatter Stone* provides both visual assets and functional systems that scale with your project.

Core Features:

- 🔨 **Modular Node System**

  Reusable Ore Node logic handles interaction, damage tracking, drop spawning, and respawn support.

- 💎 **Themed Packs**

Each pack includes detailed node meshes, refined pickup objects, particle FX, and UI sprites specific to its material type. Expand your collection to suit your game.

- 🎮 **Click or Collision Interactions**

  Use raycast input for traditional games or physics triggers for VR/melee harvesting.

- 💥 **Dynamic Destruction**

  On interaction, nodes shatter into pieces and spawn pickups with randomized movement and audio.

- ♻️ **Clean Debris Management**

  Detached fragments fade out and reset automatically with optional timing.

Supported Use Cases:

- Open-world mining and crafting.
- Resource harvesting in RPGs and survival games.
- VR/AR interaction systems (via collision triggers).
- Educational or simulation tools involving geology/resource types.

Contacting Support

Online Documentation - https://nv3d.notion.site/

If you encounter any issues or have questions not covered in this documentation you can contact Matt directly, or join our active Discord server for help, tips and community events.

✉️ matt.nv3d@gmail.com

🗨️ https://discord.gg/TQ7ADa8zFu

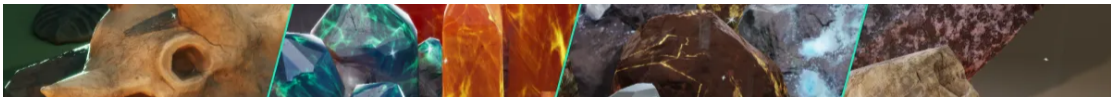
Pack Overview

The
Shatter Stone series is divided into thematic packs that build upon a shared core system. Each pack contains high-quality destructible node meshes, matching refined pickups, FX, UI icons, SFX and integration-ready prefabs. You can use each pack independently or mix and match them within a unified project.

> ✅ Note
> All packs use the same logic. No additional scripting is required when adding new packs.

> ✅ Note
> All packs include support for BiRP, URP, and HDRP via switch packages. See the ***Render Pipeline Compatibility*** section for details on how to apply the appropriate pipeline for your project.

▼ Shatter Stone: Bundle



Starter Pack

Essentials for setup, testing, and extension.

- Pyrite ore node with matching refined pickup and UI icon.
- Collision-based Pickaxe prefab.
- Demo Scene and prefabs.
- 🪵 *Includes full access to shared scripts.*
- ✂️ *Includes a **reduced selection** of SFX from the shared audio library.*

Minerals & Crystals

- 6 Unique crystal nodes with matching refined crystal and UI icon.
- Demo Scene and prefabs.
- 🪵 *Includes full access to shared scripts.*

- 🎧 *Includes full access to shared SFX library.*

Metal Ores

- 6 Unique metal ore nodes with matching refined metal and UI icon.
- Demo Scene and prefabs.
- 📜 *Includes full access to shared scripts.*
- 🎧 *Includes full access to shared SFX library.*

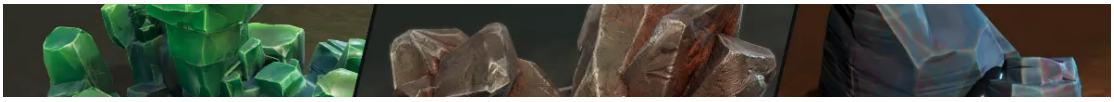
Stones & Rocks

- 6 Unique stone nodes with matching chiseled stone block and UI icon.
- Demo Scene and prefabs.
- 📜 *Includes full access to shared scripts.*
- 🎧 *Includes full access to shared SFX library.*

Fossils

- 6 Unique rockbound fossil nodes with matching freed fossil and UI icon.
- Demo Scene and prefabs.
- 📜 *Includes full access to shared scripts.*
- 🎧 *Includes full access to shared SFX library.*

▼ Shatter Stone: Stylized Resource Bundle



Stylized Gemstones

- 6 Unique gem types with matching refined gems and UI icons.
- Modular design with 3 mineable clusters per gem type - 18 total.
- Individual pieces included for generating new or even mixed clusters, as well as environment decoration.
- Demo Scene and prefabs.
- 📜 *Includes full access to shared scripts.*
- 🎧 *Includes full access to shared SFX library.*

Stylized Metals

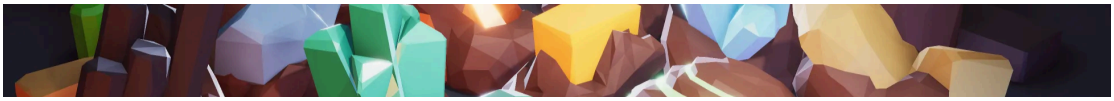
- 6 Unique metal types with matching refined metals and UI icons.

- Modular design with 3 mineable clusters per metal type - 18 total.
- Individual pieces included for generating new or even mixed clusters, as well as environment decoration.
- 🟫 *Includes full access to shared scripts.*
- 🎧 *Includes full access to shared SFX library.*

Stylized Fuel & Earth

- 6 Unique resource types with matching refined resources and UI icons.
- Modular design with 3 mineable clusters per resource type - 18 total.
- Individual pieces included for generating new or even mixed clusters, as well as environment decoration.
- 🟫 *Includes full access to shared scripts.*
- 🎧 *Includes full access to shared SFX library.*

▼ Shatter Stone: Low Poly



- Low Poly version of all the standard packs - Starter + Minerals&Crystals + Metal Ores + Stones&Rocks + Fossils.
- 25x Mining Nodes with matching resource and UI icon.
- Demo Scene and prefabs.
- 🟫 *Includes full access to shared scripts.*
- 🎧 *Includes full access to shared SFX library.*

Getting Started

This section will help you quickly install and test the Shatter Stone system. Whether you've purchased the Starter Pack or a full bundle, the process remains consistent thanks to the shared logic and prefab structure.

> ✅ Note
> New to Unity? While this documentation assumes familiarity with the Unity Editor, you don't need advanced scripting knowledge to use Shatter Stone. All interactions are driven via prefabs and drag-and-drop setup.

🚨 Important
Shatter Stone 3.0 introduces significant structural changes, including new file locations and renamed assets. To avoid conflicts, duplicate files, or broken references, it is **strongly recommended** to:

1. Back up any existing Shatter Stone content
2. Delete the entire `ShatterStone` folder from your project
3. Import the new version cleanly via the Unity Package Manager

▼ Import the Pack(s)

Download and import any Shatter Stone pack from the Unity Asset Store via the Package Manager:

- Open Unity and go to **Window > Package Manager**
- Select **My Assets** and locate your pack
- Click **Download** then **Import**
- Ensure all folders (Demo, Prefabs, Scripts, Shared) are selected

✅ Note
All packs share a common folder structure and auto-configure via prefab settings.

⚠️ Warning
If you are getting errors check the pack version is the latest and re-download + re-import.
Before placing nodes in your scene, make sure the correct render pipeline support package has been imported (URP/HDRP), if applicable. See ***Render Pipeline Compatibility*** for instructions.

▼ Render Pipeline Compatibility

As of
Unity 6, **URP** is the default render pipeline. **BiRP** and **HDRP** are supported via included switch packages. In earlier versions (e.g., Unity 2022), **BiRP** was the default pipeline.

Pipeline	Unity 2022	Unity 6	Unity 6.1
BiRP	✅ Default	✅ Use switch package	✅ Use switch package

URP	✅ Use switch package	✅ Default	✅ Default
HDRP	✅ Use switch package	✅ Use switch package	✅ Use switch package

To switch render pipelines simply open the corresponding .unitypackage in `Assets/NV3D/ShatterStone/<PackName>/`.

> ✅ Note
> If you switch to the wrong pipeline or encounter rendering issues, you can always re-import the default package to restore original settings.

▼ Running the Demo Scene

Each pack includes a demo scene that showcases key features.

To load a demo scene:

1. Navigate to:
   `Assets/NV3D/ShatterStone/<PackName>/Demo/`
2. Open `Demo.unity`
3. Press **Play** in the Unity Editor.

Clicking on nodes or using the Pickaxe will trigger the interaction, spawning pickups and ultimately shattering the node.

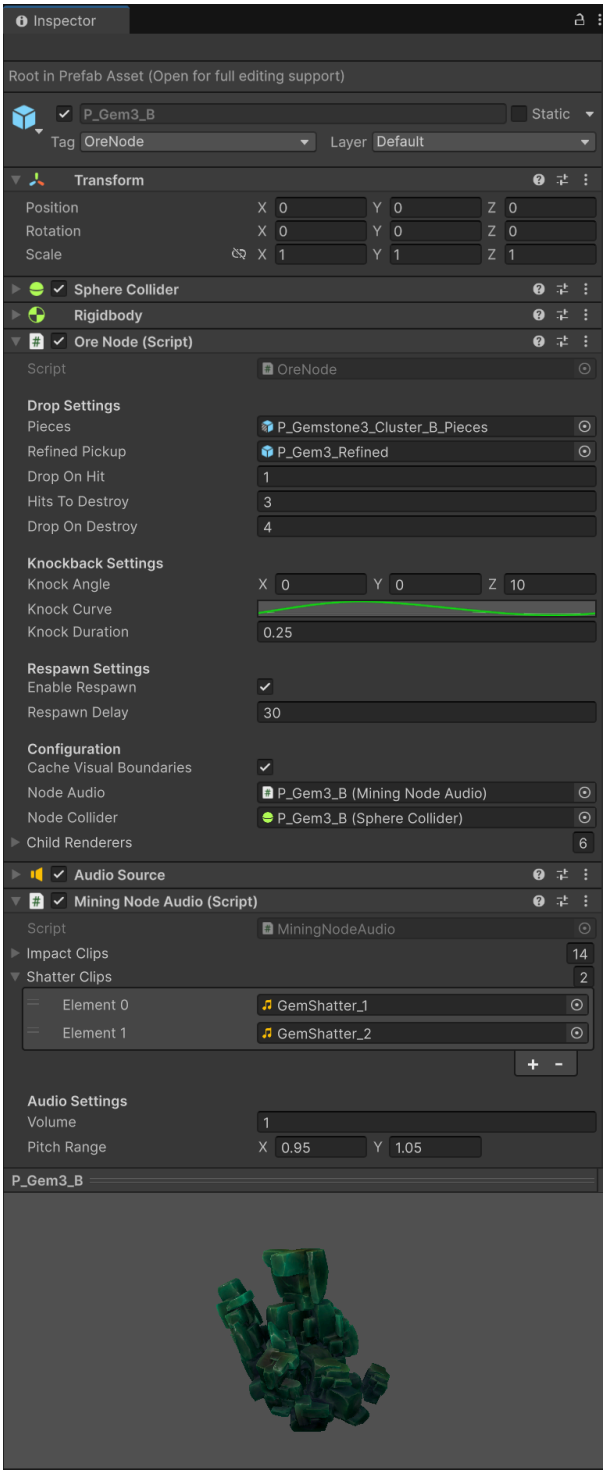
> ⚠️ Warning
> The click interaction controller `DemoClickInteraction.cs` will only work with Unity's new input systems. Set input to new or both in the project settings to use.

▼ Placing Nodes in Your Scene

You can place mineable nodes anywhere in your scene by dragging them from the prefab folder.

Steps:

1. Open the Prefab folder:
   `Assets/NV3D/ShatterStone/<PackName>/Prefabs/`
2. Drag a node (e.g., `PyriteOre.prefab`) into your scene.
3. Optionally add a `Pickaxe` or other trigger collider to interact via physics.



Inspector view of ore node prefab.

▼ Click-Based Interaction (Demo)

By default, demo scenes use `DemoClickInteraction.cs` for mouse input:

- Click on any object tagged `"OreNode"` to trigger mining.
- Click on any `"OrePickup"` to collect the item.

This script is intended for demonstration and may be replaced or extended based on your input system.

▼ Collision-Based Interaction

The included `Pickaxe` or `TriggerSphere` objects demonstrate a collision-based approach:

- When the pickaxe's collider enters a node tagged `"OreNode"`, it triggers the `PickaxeTrigger.cs` script.
- This is ideal for VR tools, melee weapons, or gesture-based gameplay.

> 💡 Tip
> Attach the `PickaxeTrigger.cs` to any tool or object with a collider to make it capable of mining.

▼ Audio & VFX

All prefabs are preconfigured with:

- Particle systems for hit and shatter effects
- Randomized audio via `MiningNodeAudio.cs` and `PickupController.cs`

> ✅ Note
> 🎧 Full SFX library is included in all packs except the Starter Pack, which provides a reduced set.

Core Architecture

The Shatter Stone series is built on a modular, shared codebase designed for plug-and-play extensibility. All packs—Starter, Standard, and Stylized—use the same foundational system to handle node interaction, destruction, pickup behavior, and audio feedback.

✅ Note
No scripting changes are needed when combining packs. All logic is centrally maintained in the shared scripts folder and configured through the Unity Inspector.

▼ `OreNode.cs`

Purpose: Controls mining interactions, damage accumulation, drop spawning, and node lifecycle (shatter, cleanup, respawn).

Key Features:

- Hit tracking and configurable hit thresholds.
- Spawns pickups at randomized positions.
- Optionally replaces the node with shatter debris.
- Handles respawn logic with delay.

Inspector Fields:

- `dropOnHit` + `dropOnDestroy`
- `hitsToDestroy`
- `refinedPickup` (spawned collectible)
- `pieces` (shattered prefab)
- `enableRespawn`, `respawnDelay`
- 🎧 **Audio**: Connected via `MiningNodeAudio` for impact/shatter sounds

▼ `ShatterCleanUp.cs`

Purpose: Handles post-shatter debris cleanup.

Key Features:

- Shrinks fragments over time with optional random delays.
- Excludes particle systems to preserve visual FX.
- Destroys the parent GameObject after cleanup.

Inspector Fields:

- `delayBeforeShrink`
- `shrinkDuration`
- `randomStartOffset`

▼ `PickupController.cs`

Purpose: Manages spawned pickup behavior and collection logic.

Key Features:

- Spawn animation: forward jump + sine curve arc.
- Continuous spin animation.
- Shrinks and disappears upon collection.
- Optional despawn timer for uncollected items.
- 🎧 **Audio**: Randomized pickup SFX from `pickupClips[]`.

Inspector Fields:

- `jumpDistance`, `jumpHeight`, `jumpCurve`
- `spinRate`, `enableDespawn`, `timeToDespawn`
- `pickupClips[]`, `volume`, `pitchRange`

▼ DemoClickInteraction.cs

Purpose: Enables interaction with nodes and pickups using screen-space raycasts (mouse clicks).

Key Features:

- Compatible with Unity's new input system.
- Raycasts from camera to click position.
- Interacts with tagged objects: `"OreNode"` and `"OrePickup"`.

> 💡 Tip
> Used only in demo scenes. You may replace this with your custom interaction logic.

▼ PickaxeTrigger.cs

Purpose: Enables collision-based interactions, ideal for melee weapons, VR tools, or physical harvesting.

Key Features:

- Triggers `OreNode.Interact()` on collider entry.
- Designed for tools like pickaxes or hammers.
- Resets interaction on trigger exit.

▼ MiningNodeAudio.cs

Purpose: Manages audio playback for node-related actions.

Key Features:

- Plays randomized clips for:
  - Pickaxe impact

    - Node shatter
    - Collecting items
- Automatically integrated with `OreNode` on prefab.

> ✅ Note
> Audio clips are assigned via the Inspector from the Shared SFX Library.

Integration

This section covers how to integrate Shatter Stone into your scenes and systems, including node placement, interaction setup, and tagging requirements.

> ✅ Note
> Shatter Stone is designed for modular integration. Each node is self-contained and can be dropped into your scene or extended to support game-specific systems.

▼ Scene Integration

Adding Nodes

1. Open the prefab folder:
   `Assets/NV3D/ShatterStone/<PackName>/Prefabs/`
2. Drag a node (e.g., `PyriteOre.prefab`) into your scene.
3. Position and scale freely, logic is handled via the prefab.

Adding Tools

Use the included `Pickaxe.prefab` or create your own trigger-based tool:

- Add a **Collider** set to **Is Trigger**
- Attach the `PickaxeTrigger.cs` script

▼ Required Tags

Shatter Stone uses Unity tags to identify interactive objects:

Tag	Usage
`"OreNode"`	All mineable nodes
`"OrePickup"`	All collectible items

Add these via Edit > Project Settings > Tags and Layers.

▼ Input Options

Click-to-Interact

Handled via `DemoClickInteraction.cs` in demo scenes:

- Left-click on `"OreNode"` to mine
- Left-click on `"OrePickup"` to collect

Collision-to-Interact

Handled via `PickaxeTrigger.cs`:

- Tool collider enters `"OreNode"`
- Triggers `OreNode.Interact()`

Works well for melee combat, VR, and gesture-based tools.

▼ Audio

Audio logic controlled via `MiningNodeAudio.cs` and `PickupController.cs`

To assign custom clips:

- Select the ore or pickup prefab.
- Assign clips for ore nodes via `MiningNodeAudio.cs` → `impactClips[]` + `shatterClips[]` in the Inspector.
- Assign clips for pickups via `PickupController.cs` → `pickupClips[]` in the inspector

Customization & Extension

This section explains how to tailor the Shatter Stone framework to your game's specific systems, such as inventory, quests, or alternative pickup behavior.

▼ Customizing Drops

Use the following fields in `OreNode.cs` to control how nodes drop items:

- `dropOnHit`: Drops per hit
- `dropOnDestroy`: Bonus drops when node is destroyed
- `refinedPickup`: The item prefab to spawn

Example - Custom Drop Logic

Override `InflictHit(int dropCount)` to drop additional items, currency, or rare materials:

```
protected override void InflictHit(int dropCount)
{
    base.InflictHit(dropCount);

    if (Random.value < 0.05f) // 5% chance to spawn a rare item
    {
        Instantiate(rareDropPrefab, transform.position + Vector3.up, Quaternion.identity);
    }
}
```

▼ Connecting to Inventory

`PickupController.cs` handles item collection via:

```
public void CollectItem()
```

To integrate with your inventory system:

- Subclass `PickupController`, or
- Modify the `CollectItem()` method to trigger an event or call your inventory logic:

```
public override void CollectItem()
{
    inventoryManager.Add(itemID); //Example
    base.CollectItem();
}
```

▼ Extending Node Logic

For advanced functionality (e.g., mining level, durability, quest integration):

- Subclass `OreNode`
- Override `Interact(int hits)` or `InflictHit(int dropCount)`
- `OreNodeBounds` and `CalculateRandomDropPosition()` are available for custom spawn behavior.
- Add custom spawn conditions, animations, or events.

Example - Override Interactions Based on Conditions

To modify how nodes react to hits:

```
public class CustomOreNode : OreNode
{
    public int requiredMiningLevel = 2;
```

```
    public override void Interact(int hits)
    {
        if (player.MiningLevel < requiredMiningLevel)
        {
            Debug.Log("Mining level too low!");
            return;
        }

        base.Interact(hits);
    }
}
```

This ensures only qualified players can interact with specific nodes.

Example - Customize Drop Positioning

All pickups are spawned within a bounding area defined by `OreNodeBounds`. You can override the calculation or modify drop placement logic for non-uniform spawns.

```
protected override Vector3 CalculateRandomDropPosition(OreNodeBounds bounds)
{
    float radius = 1.5f;
    Vector2 offset = Random.insideUnitCircle * radius;

    return new Vector3(
        transform.position.x + offset.x,
        bounds.centerY,
        transform.position.z + offset.y
    );
}
```

To adjust the spawn volume, override `CalculateNodeBounds()`:

```
csharp
CopyEdit
protected override OreNodeBounds CalculateNodeBounds()
{
    return new OreNodeBounds(-1, 1, -1, 1, transform.position.y);
}
```

This is especially useful for unusually shaped nodes or dynamic node generation.

▼ Custom Shaders

Each pack in the Shatter Stone series uses **custom shaders** to support distinctive material behaviors and stylized visual presentation. These shaders are optimized for performance and flexibility across BiRP, URP, and HDRP.

Standard packs utilize a **rock mask system** that enables separate tinting of the base rock and embedded ore from a single material and texture set. This allows seamless environmental blending (rock) while maintaining visual control over ore coloration.

In addition to standard **albedo**, **normal**, and **emission** maps, each material uses a packed **mask texture**. The packing varies slightly by pack type, as shown below:

Standard Pack	R Channel	G Channel	B Channel	A Channel	Notes
Mineral & Crystal	Occlusion	Metallic	Smoothness	Rock Mask	Additional noise pattern used to customize crystals
Metal Ore	Occlusion	Metallic	Smoothness	Rust Mask	Metal(G) used as rock mask.
Stone & Rock	Occlusion	Metallic	Smoothness	Rock Mask	Additional noise pattern used to customize stone.
Fossils	Occlusion	Metallic	Smoothness	Rock Mask	Additional noise pattern used to customize fossils.

✅ Note
Packed masks reduce texture calls and GPU load, especially with instancing enabled.

Shader Editing

Shaders were built using **Amplify Shader Editor (ASE)**. ASE is **not required** to use the assets, but if you have ASE installed, you can open and customize the node graphs directly.

Further Optimization

Shatter Stone is designed for scalability, but large-scale use benefits from a few performance strategies.

▼ Pooling Pickups & Debris

Avoid frequent `Instantiate()` and `Destroy()` by pooling. Use Unity's built-in `ObjectPool<T>` or a third-party pooling system.

▼ Fragment Cleanup

`ShatterCleanUp.cs` ensures fragments shrink and despawn automatically:

- Tune `delayBeforeShrink` and `shrinkDuration` for performance vs. realism.
- Disable particle collisions unless needed.

▼ Batching & Shadows

- Mark debris static if it doesn't move after shattering.
- Disable shadows on pickups and debris where appropriate.
- LODs included inside LOD Groups, altering LOD bias can improve performance.

▼ Audio & FX Optimization

- Limit audio overlap: use randomized volume/pitch for variation.
- Reduce or pool expensive VFX for large-scale scenes.

> 💡 Tip
> If you're generating hundreds of nodes procedurally, consider disabling VFX and audio on distant objects.

FAQ

▼ Materials look pink or broken after importing. What should I do?

This usually means the wrong render pipeline is active. Import the correct support package from `Assets/NV3D/ShatterStone/<PackName>/`, or revert by re-importing the base asset package.

See ***Render Pipeline Compatibility*** for details.

▼ Why isn't my node shattering?

✅ Checklist:

- Ensure the GameObject is tagged `"OreNode"`.

- Verify that `OreNode.cs` is attached.
- Make sure the prefab has a collider (non-trigger).
- Confirm the `hitsToDestroy` is greater than 0.
- If using the demo, confirm `DemoClickInteraction.cs` is referencing the correct camera.

🔧 Fix:

- Double-check prefab setup.
- Place a known-working prefab (e.g., from a demo scene) to isolate the issue.

▼ Pickups are not spawning or are invisible

✅ Checklist:

- Check that `refinedPickup` is assigned in the `OreNode` Inspector.
- Ensure the spawn position isn't clipped into geometry.
- Confirm the pickup prefab has visible MeshRenderer + material.
- Verify correct scale and spawn logic if using custom bounds.

🔧 Fix:

- Try replacing the `refinedPickup` with one from the default Starter Pack.
- Use `Debug.DrawRay` in custom drop logic to verify positions.

▼ Clicking does nothing in demo scenes

✅ Checklist:

- `DemoClickInteraction.cs` must have a valid reference to the main camera.
- Make sure Unity's **Input System Package** is installed (if using new input system).
- Tags `"OreNode"` and `"OrePickup"` must exist and be correctly assigned.

🔧 Fix:

- Reassign camera in the `DemoClickInteraction` script component.
- Check `Project Settings > Input System Package` compatibility.

▼ No sound plays when collecting or shattering

✅ Checklist:

- Check both `MiningNodeAudio.cs` and `OreNode.cs` are attached to the prefab root.
- Confirm audio clips are assigned in the `MiningNodeAudio.cs` and/or `PickupController.cs`.
- Ensure an `AudioSource` is present and enabled on the object.
- Check audio mixer routing (e.g., master volume).

- Make sure your prefab isn't being destroyed before the clip finishes.

🔧 Fix:

- Use longer-lived collection animations (e.g., delay before `Destroy()`).
- Test playback directly via `AudioSource.PlayOneShot()` in the Inspector.

▼ Shatter debris doesn't disappear

✅ Checklist:

- Ensure `ShatterCleanUp.cs` is attached to the shattered object prefab.
- Confirm that the debris hierarchy contains child objects (not just effects).
- `shrinkDuration` and `delayBeforeShrink` must be set to reasonable values.

🔧 Fix:

- Reattach or reconfigure `ShatterCleanUp.cs`.
- Use Unity's **Gizmos view** to inspect hierarchy during runtime.

▼ I'm getting errors after importing a new pack

✅ Checklist:

- Confirm you're using the latest Unity LTS version.
- Check the Asset Store for updates to Shatter Stone.
- Ensure your tag list includes `"OreNode"` and `"OrePickup"`.
- Avoid importing multiple versions of shared scripts across different packs.

🔧 Fix:

- Update all installed Shatter Stone packs to the latest version.
- Delete and reimport the affected pack if needed.

> ⚠️ Warning
> When using multiple packs, do not modify the contents of the `Shared` folder. Doing so may create version conflicts or prefab mismatches. Re-import packs to revert changes.

▼ How do I reset a node after mining?

Shatterable nodes can automatically respawn if `enableRespawn` is enabled in the `OreNode` inspector.

- Set `enableRespawn = true`.

- Configure the `respawnDelay` in seconds.
- No extra scripting is required.

▼ Where did the Script Graphs go?

As of version 3.0 Script Graphs have been depreciated and no longer supported in the Shatter Stone series. This is largely due to them not being utilized by users. If you are using them in your project, you can still find the latest supported version of the graphs and prefabs inside `Assets/NV3D/ShatterStone/Shared/Legacy/` and `Assets/NV3D/ShatterStone/<PackName>/Legacy/.`

▼ Can I use this in multiplayer?

Yes, with some customization.

Shatter Stone is **not network-aware out of the box**, but it can be used with:

- Mirror
- Photon PUN 2
- Netcode for GameObjects

You will need to:

- Sync node state and pickup spawning through RPCs or commands.
- Prevent duplicate spawns on clients.
- Disable client-side `Interact()` if not owning the object.

Changelog & Versioning

▼ v3.0.0 – Shatter Stone

New Features

- Added Unity 6.1 version.
- Audio has been added across the series.
- New clean/destroy function added to shatter pieces.
- New Respawn/Reset functionality added to Ore Nodes.
- Performance improvements including LODs, poly reduction and serialized loading.
- Script Graphs have been depreciated in 3.0. Legacy graphs can still be found in `Assets/NV3D/ShatterStone/Shared/Legacy/` along with the premade prefabs for each pack in `Assets/NV3D/ShatterStone/<PackName>/Legacy/`.

Content Additions

- SFX library added to `Assets/NV3D/ShatterStone/Shared` for pick hit, shatter and item collect.
- New script `MiningNodeAudio.cs` and logic added to play the new SFX at the correct time.
- New script `ShatterCleanUp.cs` added to support clean up.

- New script `DemoClickInteraction.cs` added to support Unity's new input system and enable on-click interactions in the demo scene.
- HDRP Switches added to packs in the Stylized Bundle.
- LODs added to all packs.
- Full documentation now available.

Fixes

- Removed ASE reference which was causing warnings in some projects.
- Cleaned up certain meshes that were triggering a convex hull warning when used as mesh colliders.
- Fossils - Fixed texture settings on Masks.
- Fossils + Minerals + Stones - Fixed texture settings on Noises.
- Stylized Metals - Removed duplicate textures.
- Fossils - Added Smoothness control to shader.
- Gemstones - Refined gems now correctly have scripts as opposed to script graphs.
- Minerals - Added Rigidbody and Collider components to Mineral 6 pieces.
- Fuel+Earth - Scenes folder renamed to Demo, and removed baked lighting data.
- Fixed an on import transform issue with the Mineral pieces.

Improvements

- Replaced Mesh Colliders with primitive colliders for better performance.
- `PickupSpawnAnim.cs` renamed to `PickupController.cs` to better reflect its expanded logic.
- Prefab hierarchies have been simplified to allow for better clean-up.
- Poly count reductions in the Fuel&Earth pack.
- Fossils - Improved trail renderer.
- Updated logic to support finding 'Mesh Renderer/Filter' in LOD Groups.
- Updated Demo scenes to be more consistent across packs.
- Standardized the trigger object across all packs - `TriggerSphere`
- Moved PFX texture to shared folder to avoid unnecessary duplication across packs.
- PFX materials now use standard Unity shaders in BiRP and URP projects.
- Dedicated PFX shader for HDRP moved to `Assets/NV3D/ShatterStone/Shared` to avoid duplication.
- Unity 2023 no longer supported with a dedicated version.

▼ Script Reference (Appendix)

This section provides technical reference for all major scripts used in the Shatter Stone system. These scripts are located in `Assets/NV3D/ShatterStone/Shared/Scripts/`

> ✅ Note
> All scripts are written in C#, fully serializable, and ready for extension via inheritance.

OreNode.cs

Purpose: Handles mining interaction, drop logic, node destruction, and optional respawn.

Key Public Members:

```
public virtual void Interact();
public virtual void Interact(int hits);
public virtual void ResetNode(float respawnDelay);
public virtual IEnumerator ResetAsync(float respawnDelay);
```

Key Serialized Fields:

- `int dropOnHit`
- `int dropOnDestroy`
- `int hitsToDestroy`
- `GameObject refinedPickup`
- `GameObject pieces`
- `bool enableRespawn`
- `float respawnDelay`

Extension Points:

- Override `Interact(int hits)` or `InflictHit(int dropCount)` for custom behavior
- Override `CalculateNodeBounds()` or `CalculateRandomDropPosition()` to customize pickup spawn regions

PickupController.cs

Purpose: Controls pickup animations, audio, despawning, and collection logic.

Key Public Members:

```
public void CollectItem();
public void DespawnItem();
```

Key Serialized Fields:

- `AnimationCurve jumpCurve`
- `float jumpDistance`, `jumpHeight`, `jumpDuration`
- `float spinRate`
- `bool enableDespawn`, `float timeToDespawn`
- `AudioClip[] pickupClips`
- `Vector2 pitchRange`
- `float volume`

Extension Points:

- Override `CollectItem()` to integrate with custom inventory systems
- Modify spawn animation parameters for varied motion

ShatterCleanUp.cs

Purpose: Shrinks and removes shattered debris over time.

Key Serialized Fields:

- `float delayBeforeShrink`
- `float shrinkDuration`
- `float randomStartOffset`

Notes:

- Automatically traverses child hierarchy to shrink meshes
- Ignores particle systems
- Destroy timing is handled internally—no user intervention needed

---

DemoClickInteraction.cs

Purpose: Demo-only script for mouse-based interactions.

Key Serialized Fields:

- `Camera mainCamera`
- `float maxDistance`
- `string oreNodeTag = "OreNode"`
- `string orePickupTag = "OrePickup"`

Notes:

- Compatible with the new Unity Input System

- Replace or remove in production for custom input logic

PickaxeTrigger.cs

Purpose: Triggers mining via collision (e.g., melee, VR, AR tools).

Behavior:

- On `OnTriggerEnter`, checks for tag `"OreNode"` and calls `OreNode.Interact()`
- Resets trigger with `OnTriggerExit`

Notes:

- Ideal for physical interactions, VR rigs, or gesture-based inputs
- Can be attached to any tool with a collider marked as “Is Trigger”

MiningNodeAudio.cs

Purpose: Manages audio playback for mining events.

Key Serialized Fields:

- `AudioClip[] impactClips`
- `AudioClip[] shatterClips`
- `AudioSource audioSource`

Public Methods:

```
public void PlayImpactSound();
public void PlayShatterSound();
```

Notes:

- Used internally by `OreNode`
- Randomizes clip pitch and selection
- Audio is spatialized (3D) by default

Optional Utility Structures

OreNodeBounds

Struct used to cache the bounds of a node’s visual area.

```
public struct OreNodeBounds
{
```

```
    public float minX, maxX, minZ, maxZ, centerY;
}
```

Use With:

- `CalculateNodeBounds()`
- `CalculateRandomDropPosition(OreNodeBounds bounds)`