Poseidon CSG

DYNAMIC LEVEL DESIGN FOR UNITY

This readme is here to help you, but is more of a general overview rather than an in-depth explanation.

We *highly* recommend you check out our documentation pages on
poseidon.cinderflame.com

You'll probably find videos containing better explanations on the **Cinderflame Studios** Youtube channel, or on our Discord Server at cinderflame.com/discord. If you have questions, message us on Discord 😊

1. Why Use Poseidon?

Creating dynamic spaces in games can be tricky. When our team first started working on games, we kept running into the age-old problem: How do I connect *this area* to this *other area*? We'd created a nice mesh for an area in our 3D Modeler, but we hadn't punched holes in it. Sigh… back to the modeler.

If we wanted to be more modular, reusing that area in different ways, we'd constantly run into the issue of having to make "possible" holes. Who knew how we'd want to reuse that space later, and where we'd need to connect a tunnel or something else that we hadn't even anticipated yet. Our 3D models grew increasingly more and more complicated with lots of holes/walls that we could toggle on and off depending on our needs… and those holes were baked into our models and could never change.

That's how **Poseidon** started – we wanted to be able to iterate and create dynamic levels for our own purposes. We wanted to create 3D environments in our modeler of choice, and not worry about HOW those areas could be used when brought into Unity. We've gone through many, many iterations of level design tools over the last 5 years, and Poseidon is the simplest, cleanest, and most elegant solution we've been able to find.

Poseidon is useful if you want to be able to connect 3D spaces to each other dynamically, allowing you to create the meshes you want, and not have to anticipate every possible way they could ever be used by a level designer. Poseidon allows a level designer to make subtle tweaks to levels without having to send a mesh back to an artist. Poseidon allows you to create modular levels with very few pieces but without everything feeling repetitive. Poseidon allows you to create unique geometry.

Poseidon does not force you to use it for everything, unlike other level design tools. You can delete it when you're done if you don't want it anymore.

Why is it called Poseidon? We have no idea. It was a codename that stuck. We're open to suggestions if you think you have a better name for it.

2. How does it work?

At its core, Poseidon is a geometry intersection tool that allows you to take two meshes in Unity, intersect them together, and then magic happens. It works something like this:

1. We detect that something has changed (a carver was moved, resized, enabled/disabled, its mesh was changed outside of Unity, etc).
2. We analyze all the Poseidons in the current scene to figure out which ones need to be updated.
3. We rerun our algorithm across each dirty Poseidon and its sisters (the ones it's intersecting).
4. We slice the meshes by each other.
5. We take any polygons from one Mesh that are inside the other and discard them.
6. We take the result and store it in a scene-copy of the Mesh.
7. We repeat the operation (4-6) until all affected Poseidons have successfully carved each other.

It's a little more complicated than that, but that is the general overview. This approach is the result of many years of iteration and slow improvement until we've created something that is both performant and clean (in most circumstances*).

3. Technical Requirements

- Unity 2019.x or newer (2018.x work but have problems).
- Runtime Version: .NET 4.x or .NET Standard 2.0 (Preferred)
- SRP Agnostic (Normal, LWRP, HDRP)

Because our code is packaged in DLLs, we require at least the .NET 4.x framework, and do not support anything before Unity 2018.3. Technically, Poseidon may work even on older builds (2017.4), but a lot of the included UI code will not be very functional and may need to be removed. If you're trying to get Poseidon working on an older build of Unity, please contact us and we will see if we can give you specially built DLLs that work on older builds of Unity.

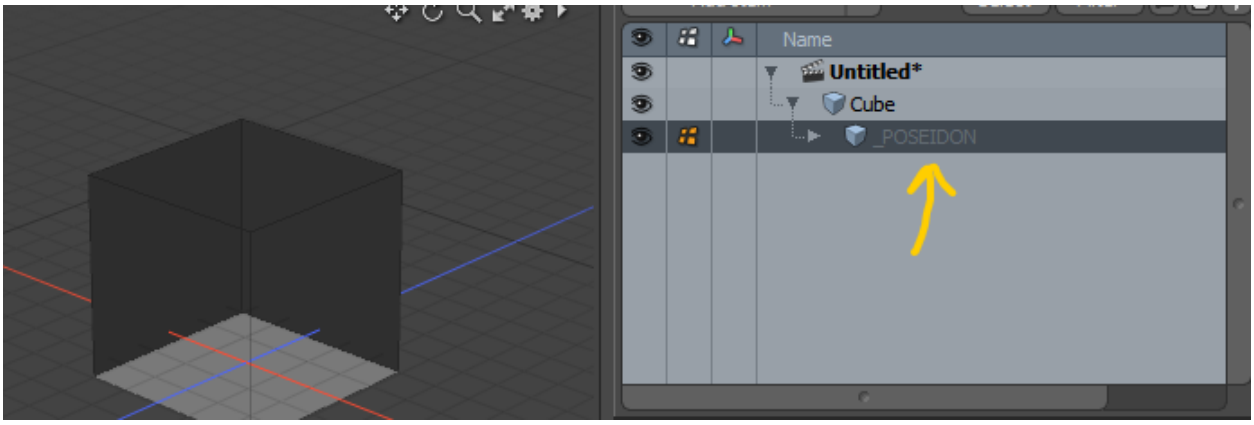
The Examples were built on 2019.2, and as a result do not work on any of the 2018.x branches of Unity. The prefabs in the Examples seem to break when imported into older projects because Unity changed the Prefab schema in 2019.1. If you are targeting a 2018 build of Unity, you can check out the examples in a separate 2019.2 project just to see the examples coming together. Sorry about that. We blame Unity.

4. Creating new Poseidons

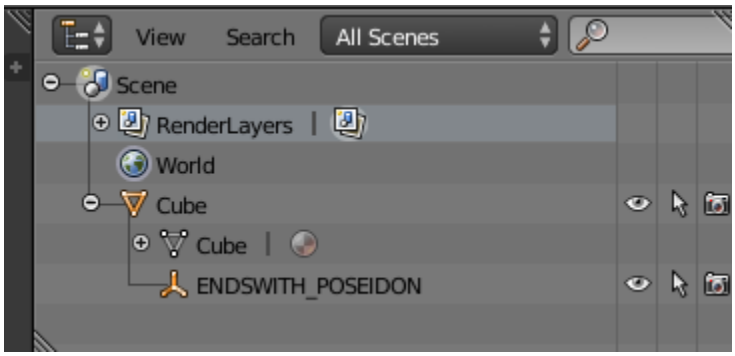
Creating a new Poseidon is easy – simply add the Poseidon script to the object with the Mesh Filter you are targeting. Our recommendation is that Poseidons should really only be added in Prefab mode, as it makes things cleaner, if you're going to prefab an object later. Poseidon does not run in Prefab Mode at the moment.

NOTE: Poseidon works with closed, interior facing geometry. (**Closed** matters as long as all the bottoms are closed. Sides and tops can be open, but may have unexpected consequences.). If you are adding to an exterior facing Mesh, you can utilize the Flip All Faces parameter to flip all polygons in your Base Mesh.

The other, recommended way of working with Poseidon is to use the Poseidon Auto ***ModelImporter***. Simply create a Mesh in your 3D modeler of choice, and if a portion of it is to be the Poseidon, simply place an empty object ending with _POSEIDON, and our importer will convert it automatically!



If your mesh in your 3D Modeler of choice has an empty child ending with _POSEIDON, it will automatically be imported as a Poseidon in the Mesh Prefab generated by Unity.



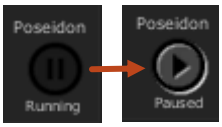
In general, we recommend using the auto-importer for almost everything when making new Poseidons. It's much easier to manage models in your own 3D modeler. Our algorithm detects if you've updated a mesh in the modeler and saved over the old mesh*.

* Kind of. We just look at the number of vertices and submeshes. If you take a Poseidon mesh, and just move a vert, we won't recognize it as changed and requiring a recompute. A more robust solution is on our roadmap as this is something that we want fixed soon for our own purposes.

5. Using Poseidon - Workflow

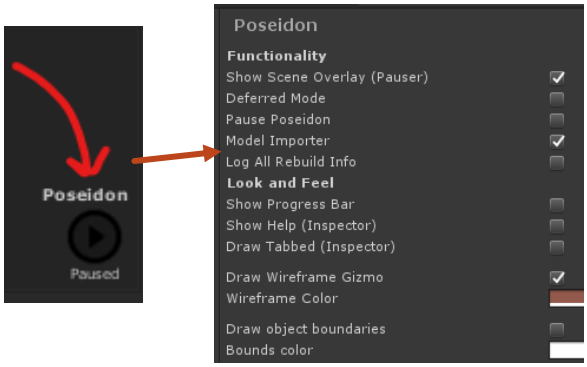
Once an object with an inward-facing Mesh has been marked as a Poseidon, it is simple to use it. ***Simply move it around in the scene and intersect it with other Poseidons in the scene, and watch the magic.*** Now, while that workflow might seem easy, there is a cost to doing high volume computations like this.

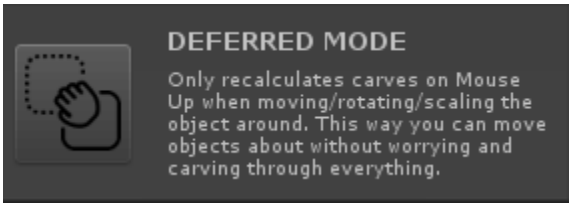
When you're in a scene with hundreds of potential intersecting objects, the math can become complicated and super slow. Thus, you may want to adjust something many times before letting it carve into place. We've exposed a button that shows up on bottom right side of the scene view that lets users **PAUSE** Poseidon. When paused, Poseidons will not recompute themselves under any circumstances. This allows a user to move multiple objects, positioning them just where they want, and finally unpause to let Poseidon recompute the recent changes.



We've found that when working in extremely large scenes, we're constantly toggling between PAUSED and RUNNING in order to get the perfect set of tweaks. This overlay can be turned off in the Poseidon Preferences window.

Clicking on the word **Poseidon** in the Scene View overlay can pull up the preferences window where you can tweak Poseidon specific settings, or this can be invoked from a Poseidon Inspector itself. Many global settings can be adjusted here.

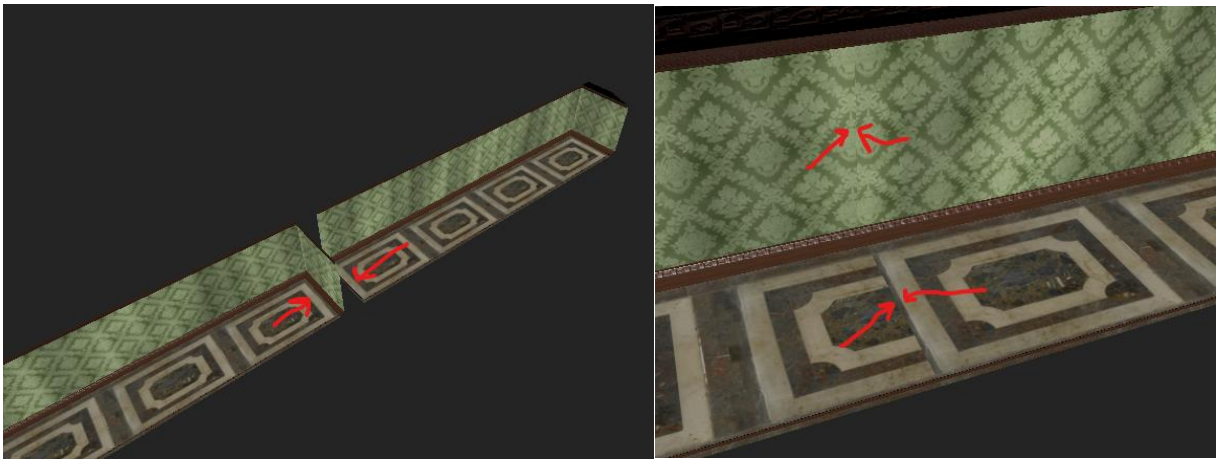




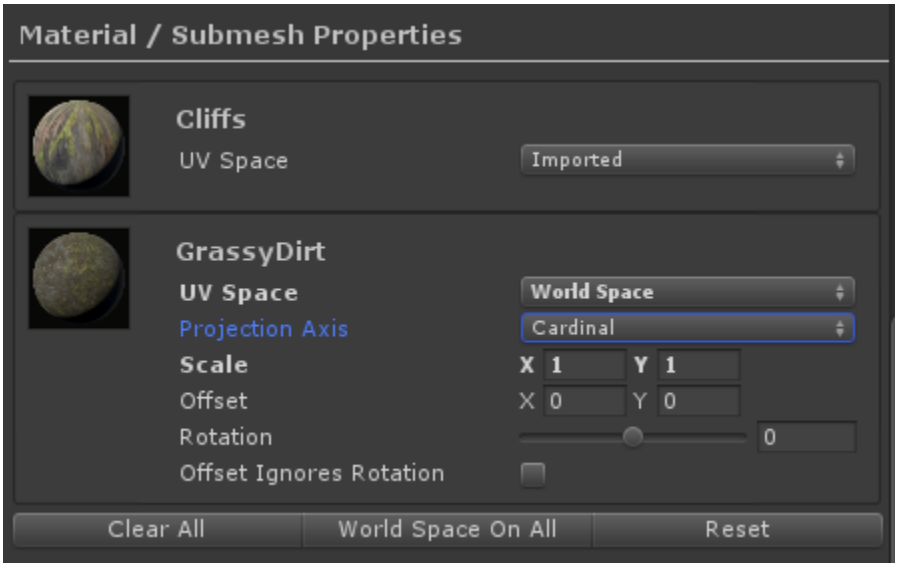
Deferred Mode is another addition we've added in place to help speed things up. When using Poseidon, it is nice to see things carving in realtime – but it is not nice if that "realtime" aspect takes forever to actually compute. Deferred Mode uses the "Mouse Up" property to decide if it should be recomputing Poseidons or not. This is in order to help with transform changes (moving/scaling/rotating) so that it's only once you let go of the mouse that we actually recalculate the Poseidons. We strongly suggest leaving this on most of the time, unless you *really* want to see everything carving in realtime.

6. Material / Submesh Properties

When you're dealing with properly textured objects, and pushing them together, it is really easy for seams to occur. As can be seen in the example below, the wallpaper and the marble floor produce ugly seams when pushed together.



If your goal is to eliminate generated seams, one possible solution is to use our UV remapper that can be found on the Poseidon inspector.



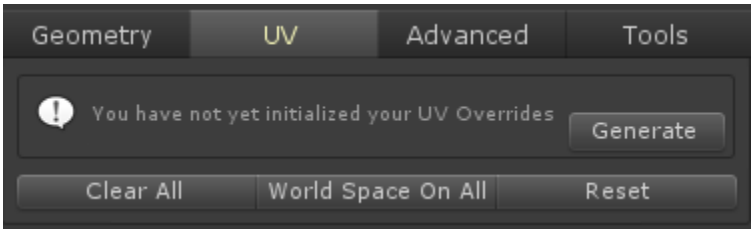
The Material / Submesh Properties generates UV overrides for each material/submesh in your mesh. When recomputing Poseidons, we can either use Imported UVs (the ones you created in your modeler), or World Space UVs, which will allow us to remap a specific submesh's UVs to a very basic projection axis. It's nothing too fancy, but it allows us to tile based on where something is in the world, not based on how it was UV mapped in the modeler.

In the image to the right, the floor tiles were remapped with World Space UVs, so that no seam occurs when these two rooms are connected together.

Because we've found using World Space UVs so useful, we've adopted the habit of almost always using it for ground textures and sometimes for walls, depending on the complexity of the model. This means that even if the floors/walls/roof of a room all use the same material, sometimes, we may want to explicitly place them on separate submeshes in order to get custom tiling on the generated World Space UVs.

The **Scale** property allows you to scale a World Space Texture, to make it larger or smaller on the UVs. **Rotation** rotates generated UVs by a certain amount in degrees, and **Offset** allows you to shift the UVs by a certain amount in order to get things to line up right. **Offset Ignores Rotation** is just a property that allows you to shift the UVs before doing rotation. Otherwise, rotation is calculated first by default.

Honestly, in many years of using Poseidon, our team has mostly only ever used the **Scale** property on these, but we wanted to expose everything, just in case your specific use cases need **Offset** and **Rotation**.

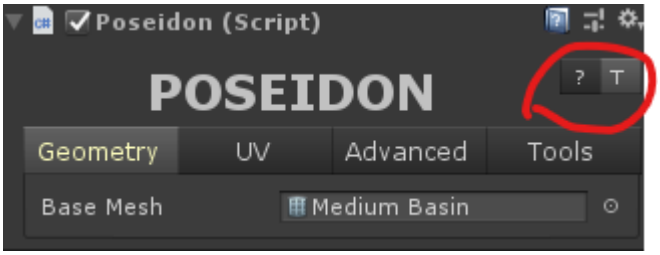


In order to start using these settings, you have to "Generate" the UV Overrides. This is just an annoying artifact of how we need to serialize the UV override data in the object, since meshes can change on us. If there are no overrides, then the system will use Imported UVs.

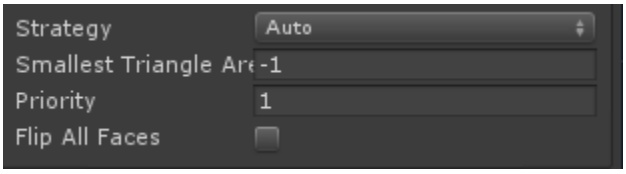
The buttons on the bottom do simple things. **Clear All** just clears the Overrides and everything is back to being Imported. **World Space On All** makes sure all of the UV Space properties are set to World Space. It's a fancy shortcut for making everything World Space. **Reset** does the opposite and sets everything to being Imported UVs. This whole thing will probably be cleaned up and simplified in future releases.

7. Other Inspector Features

The Poseidon Inspector has a question mark button which will toggle a bunch of help text above most options on the inspector. The "T" button will toggle between a tabbed mode, and a list-mode for the categories in the inspector.



Under advanced, we recommend reading the help text listed in the inspector, but think that most people shouldn't really be messing around with these properties, except maybe Priority. Flip All Faces can allow you to flip all the normal of a given mesh, but depending on the mesh, it may have unexpected results. We don't really recommend using it, but it's there if you really need to flip the default Cube.



The Poseidon inspector offers a number of warning boxes when things aren't correct. We strongly suggest resolving those using the "Fix Now" buttons. These include things like, changing Box Colliders to Mesh Colliders, or if your UV overrides were generated at some point in the past, and the Mesh has a different number of materials now, etc. Please pay attention to these in order for your carves to work correctly.

8. Runtime Support

Poseidon was originally created to be an editor-time tool for designers to be able to quickly create levels and be able to modify them dynamically. Due to significant requests, we have rewritten our core algorithm to support runtime support but are working on improving how this works for the future.

In the Editor, we do a lot of detection to know if any carver needs to be recomputed (the object was moved, the underlying mesh was changed, some settings on the object changed, etc.), but at runtime we do not *yet* want to be in control of managing when objects should change. At this point in the process, since we have exposed that logic to you, *you are responsible* for deciding when Poseidons should carve – we just handle the carve operation.

We are planning on eventually switching our system to the Unity Job System, but currently Poseidon is a Frame Divided operation. We are crunching tons of numbers and comparing numerous triangles, but we only do a certain number of operations per frame until we've hit our time budget. So in V1 of our Runtime Support, we're running on the Main Unity Thread, but are running on a budget that you decide.

```
PoseidonRuntime.GetCarveOperation(IEnumerable<PoseidonBase> allPoseidons,
    CarveParameters parameters) => PoseidonOperation
```

`GetCarveOperation` is the main function you need to call in order to generate a `PoseidonOperation` which will contain all the data for the carve. You're then responsible for calling `PoseidonOperation.RunFrame()`, which will run one frame tick of the carve operation for however long was set up in the `CarveParameters`. We recommend something like 15ms-20ms per frame, but if you're running this on a loading screen or somewhere where frame rate doesn't matter, you can pass a bigger number to the operation. You can even get the runtime system to compute the entire operation in one frame if you pass `float.PositiveInfinity` as the frame budget.

Example Scene "5. Runtime Support" contains an example of how to possibly run a carve operation for objects that are dynamically created. In that example, we run the operation inside of a Coroutine, but it could just as easily be called as part of an Update loop. In case this is being used as part of a loading scene for generating a level dynamically, we also expose a progress value on the operation that runs from 0f – 1f. The example scene uses that to show a scene-level progress bar for the operations happening in the scene.

For more details on how to actually use the Poseidon Runtime, check out the documentation on the Poseidon website, as that will go into more depth about how to use our runtime support, or message us on our discord and we're more than happy to help you work through any questions you may have.

9. Layered Rebuilding (Outward Facing)

Poseidon has a hidden experimental "layered mode". It works well, but is not the solution we want long term, so using it means that any Poseidons you build with it might not work with future versions. However, if you need the CSG Difference operation, want to carve inward facing objects with outward facing objects, or want systems of Poseidons inside of other systems that don't affect one another, feel free to turn on the Layered Mode in the Settings. Information about how to actually use it can be found on the documentation website.

10. Roadmap

- **Subtractive Operations** – Currently Poseidon adds Meshes together. We can kind of cheat the system by just throwing in non-Poseidon objects and call it subtraction, but we've got some things in the pipeline for a proper hierarchical operation tree. For now, you can use the Layered Poseidon mode if you really need the Difference CSG operation.
- **UI Improvements** – Our UI isn't the best, so we will make it better.
- **KDOP26 vs AABB** – When determining if two Poseidons are sisters and should carve one another, we use Axis-Aligned-Bounding-Boxes for the Intersection calculation, which has problems – namely, that it over includes things that maybe shouldn't have been part of the Poseidon. We're going to be switching to a KDOP26 approach to tighten the bounds around when two objects are actually intersecting with each other.
- **Mesh Changed Detection** – We need to better detect when meshes change in a modeler. Currently we only look at counts of vertices/submeshes, in the future Unity will be providing a more accessible asset/importer hash.
- **Open Poseidons** – Currently, one of the restrictions or problems with Poseidon are that they mostly need to be closed meshes. We're looking into ways of removing some of those restrictions by supporting slightly more open meshes.

11. Frequently* Asked Questions

*We just released this tool… so these aren't actually questions people have asked, but questions we think people might have. But if you have actual questions please join our discord group at cinderflame.com/discord and we'll gladly help you out there!

What happens if I have multiple scenes open at the same time?

Poseidons only intersect with objects in their current scene and *will only be computed for the Active Scene* if multiple scenes are open. Poseidons from inactive scenes remain dormant, so if moved around, they will not update. This might be something we'll resolve in the future.

Does Poseidon work with prefabs?

Yes. In fact, we'd rather that most things done with Poseidon be done using the prefab flow developed in Unity 2018.3+. We'd rather most configuration be done in Prefab Mode rather than tweaked on a dynamic object and applied back.

Does it work at runtime?

Yes, but not automatically. We have an initial version of our runtime support.

How do I do _____ with Poseidon?

Poseidon is a very dynamic tool that can do a lot of stuff. It can make doors, windows, tunnels, basins in addition to 3D environments. If you want to use Poseidon for those purpose, we're more than happy to help you figure those out on our Discord Server! Join us there and we'll walk you through creating more complicated carvers with Poseidon.

I found a bug / have a suggestion

Please let us know on our Discord server or email us at cinderflamestudios@gmail.com and we'll take a look at it for you. Thanks for letting us know!