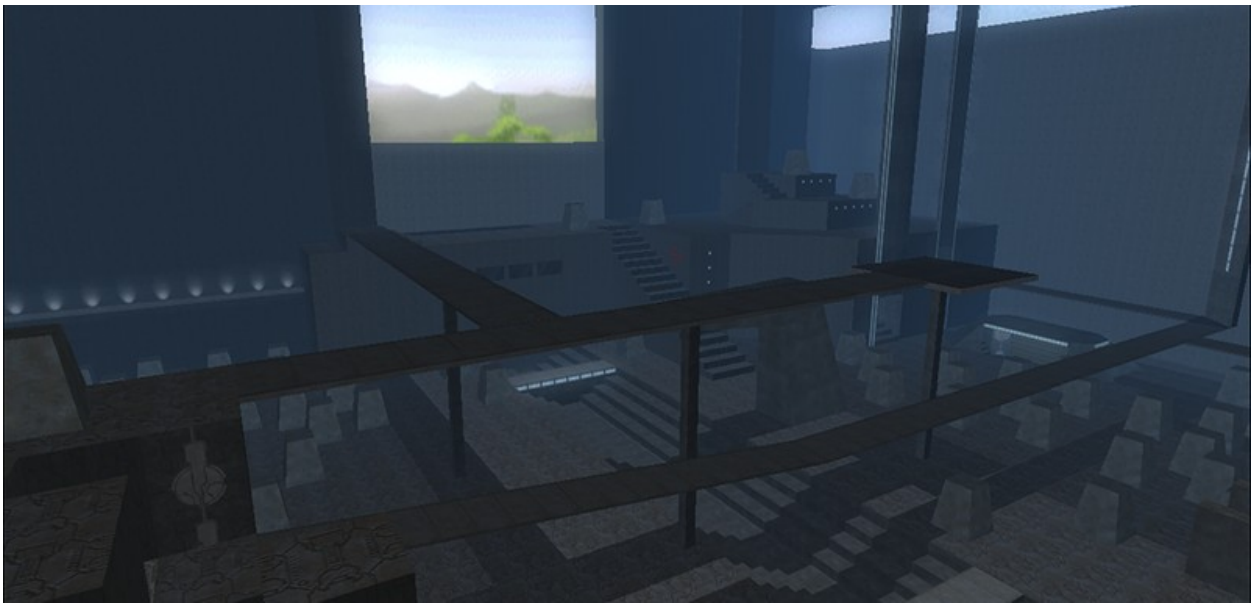
Better Bridges 1.0



Overview

Better Bridges is a new technique for simulating bridges in Aleph One levels as if the engine supported them natively. It's a generalized Lua script that can work with any level that's set up in a specific way. This guide explains how to use Better Bridges, and includes a working sample. To get started right away, skip to the detailed tutorial beginning on page 4.

Features

- Players can move over, under, onto, and off of bridges as if they were normal polygons.
- Projectiles can spawn on, pass over, pass under, and detonate on bridges.
- Players on bridges hear projectiles and environmental sounds.
- Mapmakers can implement Better Bridges without little to no scripting.
- Better Bridges works for single player and multiplayer games.

Limitations

- Players can’t pick up items on bridges.
- Monsters don’t use bridges or find players on bridges.
- A polygon can’t contain more than one bridge.
- Adjacent bridges can’t be at different heights, unless they function as stairs.
- The space under a bridge can’t be a platform, hill, or other special polygon type.
- The 3rd person camera isn’t accurate while on a bridge.
- Players on bridges don’t hear other, rarer sounds like other players teleporting.
- In netgames, players on bridges may appear extra ‘jittery’.

Background & History

Aleph One’s 2.5D, portal-based engine supports some interesting geometry features like 5D space, but not others like *bridges and balconies*. Each polygon has one floor and one ceiling, so there’s no way to specify a second floor on which to stand within the same polygon.

There have been several efforts to overcome this. The earliest, seen in the original games and available to anyone with Forge or Weland, is to carefully wrap two sets of polygons around some pillars, so that there’s a path over and a path under the same spot. Some people have used more advanced map tricks that take advantage of how Aleph One renders a scene, including Even Steven (pre-2005), goran (2009), and quartz (2011). Non-map-based solutions include Inio’s Aleph One B&B (2001), with engine changes that allowed bridges to be specified in MML; and my (Crater Creator’s) earlier attempt at a Lua script (2003), which counteracted gravity in certain regions to create a virtual bridge.

All these methods have their downsides. The bridges don’t look right, or they only look correct in narrow circumstances. The player can’t do everything while over or under a bridge that they could do on regular ground, or in B&B’s case the code has become vaporware. Mapmakers have long desired a better technique.

When heliomass started experimenting with “solid ledges and fully moving 3D platforms” in 2013, that’s when I realized Lua could move a player to a surface, have it interact with the ground, and move it back before the scene renders, such that the player wouldn’t see any movement. Later, as thedoctor45 led development of Halathon, I saw the opportunity to use this idea in maps like Hang ‘Em High, which translated well to Aleph One except for their bridges.

I developed the script for Halathon and extracted it out into this standalone version. Although it, too, has limitations, I hope mapmakers will find Better Bridges useful and easy enough to use in their scenarios.

Acknowledgments

I'd like to thank all current and former Aleph One engine developers, tool makers, and scenario makers that have sought to push the envelope over the years, and specifically everyone who's contributed to Halathon, for directly or indirectly helping Better Bridges get this far.

Version History

1.0 First public release

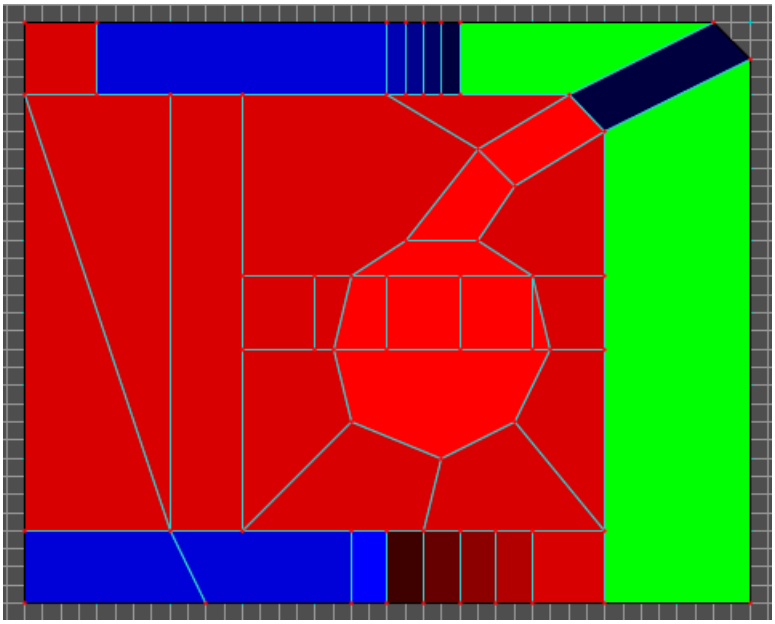
- Extracted script from Halathon
- Renamed Better Bridges
- Allowed proxy offsets in any xyz direction
- Fixed misnamed and disappearing projectiles

Making Maps with Better Bridges

Part 1: Level Construction & Bridge Visibility

As you draw out your level as usual in Weland, carve the shapes of the bridges into separate polygons. A bridge can be any height above the floor and use any size, shape, or number of polygons. But practically speaking, a simple, blocky bridge will be easier to make than a winding, organic bridge. This is because we'll make the bridges visible using 3D models, which Aleph One renders best as small, repeating segments instead of one big model.

If a bridge will have stairs, draw a polygon for each stair, the same way you'd otherwise draw stairs. Then, set all the floor and ceiling heights as if the level was an open space with no bridges.

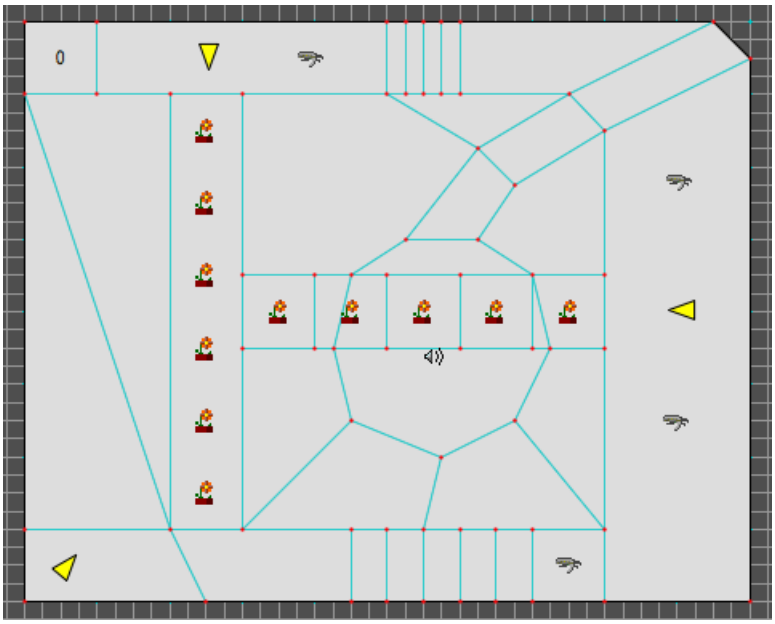


Next, add the scenery objects that will later become 3D models (using MML) to form the visual component of the bridges. The sample map replaces 'light dirt' with a 1 WU square block, placed at 1 WU intervals. Use these tips while placing the scenery:

- Set each object's x/y coordinates precisely, down to 0.001 WU, to avoid seams.
- You can set the facing angle of scenery objects in Weland, so the same bridge

segment can be used in a north-south and east-west configuration.

- Set the height of these objects appropriately. If your bridge is 2 WU off the ground, set the height of the scenery object to 2 WU. If your ground is complex, it may be easier to set the heights relative to the ceiling instead.



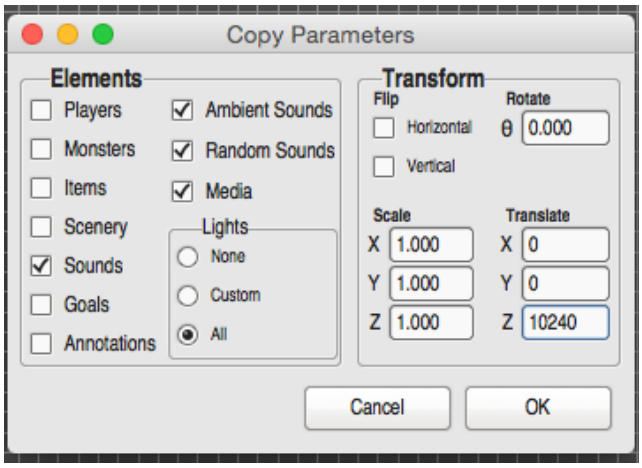
Save your level. Now take note of how the provided `Split Map Folder` is organized to merge a level with the MML and Lua that make Better Bridges work. The first file in `Resources/TEXT`, `00128.txt`, specifies that the first level in the scenario will use the other two files, `00129.txt` and `01000.txt`, for Lua and MML, respectively. Organize your files for merging similarly, depending on what kinds of files are already part of your scenario. For multi-level scenarios, edit `00128.txt` so that `00129.txt` and `01000.txt` are used in each level with bridges.

Merge the map with Atque and try it in Aleph One. Verify your bridges look the way you want. They won't be 'solid' yet, so Visual Mode Lua or Vasara may be useful so you can fly off the ground and see the bridges from above and below.

Before proceeding further, at a minimum your geometry should be finalized and paved, and sounds should be in place. You'll be able to change textures, lights, and items later. But any geometry changes – especially adding or removing polygons – will require repeating the parts after this one.

Part 2: Constructing The Proxy

When your level is ready, the next step is to create a *proxy*. The proxy is a second copy of the whole level's geometry. For this you need jonny's Copy & Paste plugin for Weland, available from Simplici7y.



- Save your level.
- Go to Plugins -> Copy & Paste.
- Select the same level from the open file dialog, since you want a second copy in the same level.
- At the parameters screen, uncheck all boxes except for media and sound-related categories (see image). Do not flip, rotate, or scale the level.
- Add an offset using the translation fields, where 1024 units = 1 WU. Technically you can put the proxy anywhere. But without an offset, it will be very difficult to make changes later, since the proxy will overlap everything. Putting the proxy directly above or below the original is a viable option thanks to Weland's min/max height sliders.

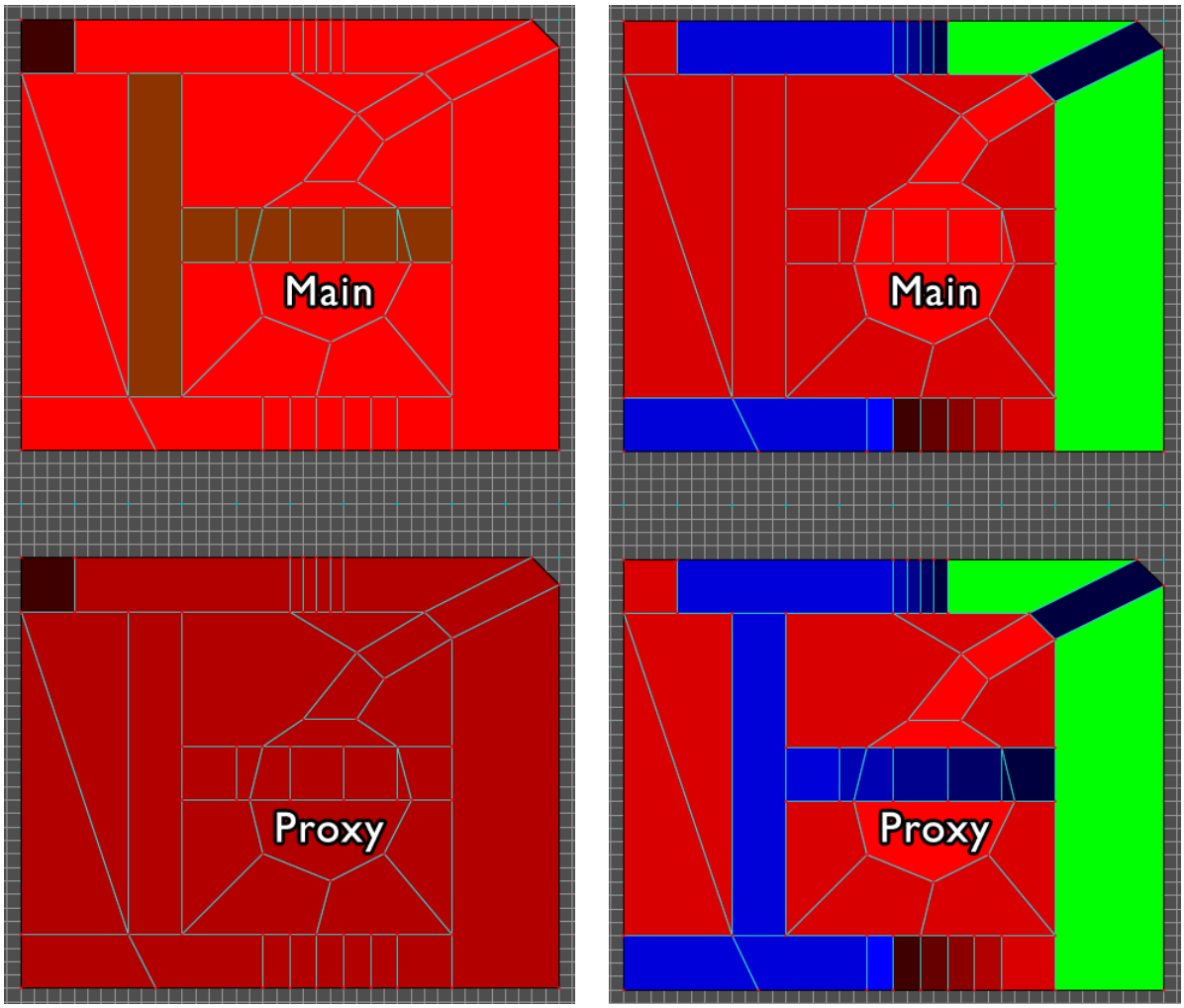
Part 3: Bridge Functionality

From now on we will refer to the *main* and *proxy* parts of the level separately.

- In the main part of the level, set all polygons where there is to be a bridge to the "Glue" polygon type. The script looks for glue polygons to know where the

bridges are, and if the level uses bridges in the first place.

- In the proxy part of the level, raise the floor heights of bridge polygons to where you want the floor of each bridge polygon. If you were to look around in the proxy, it would look like the main except for solid blocks rising out of the ground up to the floor of each bridge. Remember, if you offset the proxy above or below the main, all the heights will be different by that offset.
- Pave the level, because raising the floors creates untextured sides.
- If you have items or monsters that spawn at random locations, change all proxy polygons to monster & item impassable.
- Save the level, preferably as a new file.
- Merge your map with Atque and try it in Aleph One (again).



Left: finished map, polygon types. Right: finished map, floor heights.

You should now have visible, functional bridges. You should be able to walk, shoot, etc. while on the bridges, as detailed in the features section.

Part 4: Modifications & Troubleshooting

- If you play the level and it looks like it's been nuked & paved, with solid space under all the bridges, you are seeing the proxy, and something has gone wrong. You need to insure the proxy is a complete duplicate of the main, with a constant offset of polygon numbers.
- If you jitter up and down when on a bridge, one or more proxy polygons are probably misaligned relative to the main. If you can't find the misalignment, delete and reconstruct the proxy. Jittering in netgames, though, may persist.
- If you need to create or delete polygons, whether part of a bridge or not, you need to **delete and reconstruct the proxy**. Unfortunately, while *constructing* the proxy is easy thanks to the Copy & Paste plugin, there's no quick way I've found to delete the proxy.
    - If the proxy overlaps the x/y coordinates of the main, set your min/max heights in Weland so only the proxy is shown, and set the preferences to not show hidden vertices. This will prevent you from deleting the main in the next step.
    - Manually delete everything visible. Click on a point and hold the delete key until you stop deleting things. Then pick another point and repeat until the entire proxy is gone.
    - Now, reset the min/max heights and make whatever changes you need to make in the main (which is the only part left in the level).
    - With the main finished (filled, paved, and with sounds), repeat parts 2 & 3.
- If your scenario already uses Lua, you will need to integrate any functions defined in both places. For instance, your script may already have its own `Triggers.projectile_detonated()` function. To facilitate integration, I've compartmentalized the script into handler functions like `handle_bridges_proj_det()` that can be called from the standard trigger functions in your custom script.