The Challenge
Find a path through a volume (described by a mesh) connecting multiple endpoints (which are features of the mesh).

The Solution
My solution consists of the following stages:

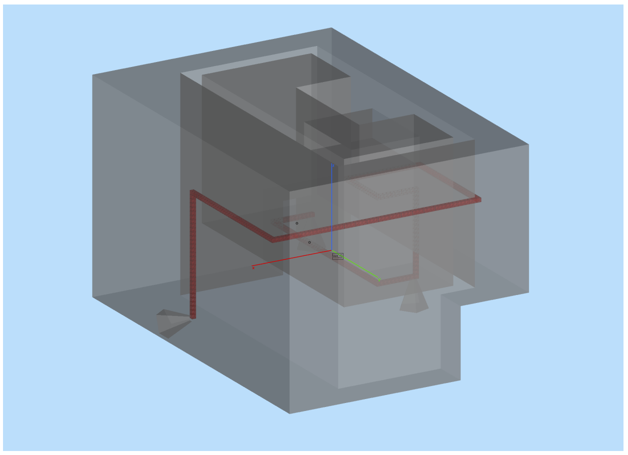
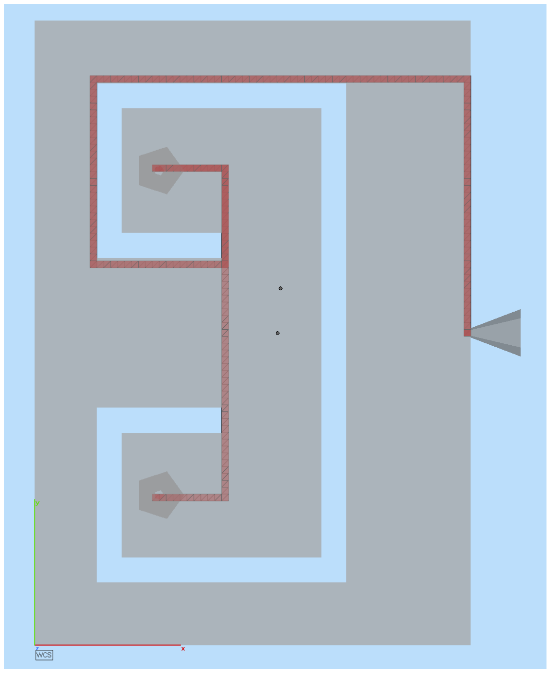
1. Extract the endpoint locations from the mesh
2. Construct a voxel representation of the mesh
3. Perform 3D pathfinding on the voxels
4. Construct mesh of the pipe voxels and output

Justification
Step 1 must be performed regardless. For step 2, some kind of intermediate representation will be required if we hope to achieve anything. I did decide to go with the voxel IR as it:

- Is trivial to construct from mesh (and convert back)
- Allows the solution to be directly computed by performing some kind of 3d pathfinding on the voxel grid

Potential downsides would be performance: 3D pathfinding is difficult. Thankfully, I found a pretty efficient algorithm. It's a relatively brute force approach which works well on 2023 computers.

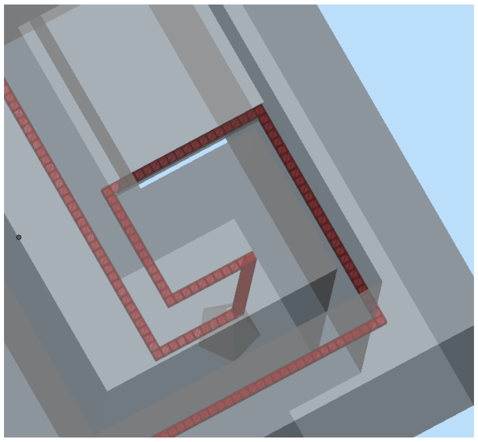
Results
Outputting the pipe mesh to an STL file allows it to be viewed alongside the original volume:

Tools Used
As my solution was to be voxel-based I knew I would need to maximize my efficiency. I was also feeling the heat and wanted to be at full power. For these reasons I chose to use Rust.

The only noteworthy Rust library used was stl_io for reading/writing STL files.
There was no usable 3MF library so the solution does not have 3MF capability. The maturity of the C/C++ ecosystems is a good reason to choose them over Rust.

3D Pathfinding Algorithm
The 90 degrees constraint makes this much easier. My solution explores in straight lines first. If it ever encounters an explored cell, it does not need to explore it again (no decrease-key operation). It uses an array of ring-buffers to store the exploration frontier. Each extra level of ring buffer means an extra turn. It takes about 10 seconds with 3x10^9 voxels.

Issues
The primary issue with this solution is the suboptimal trunking:

The pipes could have joined earlier, saving pipe and junctions.
The reason for this is that the path found is the shortest path tree from an arbitrary node to each other node.
One solution would be to first compute this route, and then compute the shortest path tree from each position on the route, with the one that minimizes the overall number of pipes being optimally trunked. But this is a lot of extra pathfinding of course.

Also the constraint of 3-junctions only being T's is not enforced, it just didn't come up in this route.

I ran out of time to do individual pipe lengths but I would have just used a similar algorithm to the one I used to assign vertices to their corresponding endpoint.

I think it fails to detect the one T-junction (which is at the one of the sprinklers -- the one in the image above)

Instructions to build & run
- Required Rust: install rustup
    - Rust requires Visual Studio C++ build tools to be installed on windows
- git clone https://github.com/kennoath/piperouter
- cd piperouter

- cargo run --release
- The program will check the working directory for “Volume.stl” which it will load and use
- The program will output “pipes.stl” and “elbows.stl” which can be viewed in your favorite cad program alongside the original “Volume.stl” and you will see that they match up
- The program will need about 2gb of ram and 13 seconds on my PC to run
- The binary (for windows) and sample output are also included

Possible Improvements
- Performance: probably at least a 10x speedup available
    - Directions could be packed into 8 bits instead of 24 bits
    - Could ensure the voxel grid is enclosed with ‘1’ voxels and avoid a lot of bounds checking in tight loops
- Mesh optimization: the output meshes could have adjacent quads combined together to reduce triangle count
- Proper frontend: rather than just searching for ‘Volume.stl’ it could be a command line arg

Other Possible Approaches
I considered another approach:
1. Decompose the volume into several connected rectangular prisms
2. Pathfind across the prisms
3. Lay pipes through prisms which are on the path

This method promises pathfinding on ~20 nodes instead of 3 billion, however the issue is I’m actually not sure how to do parts 1 & 3.
Part 1 is kind of like ‘Inverse-CSG’.
Part 3 would be kind of like an optimization problem. It’s not so spooky, and might play nicely for some of the constraints and overall this approach might yield better trunking.
But it’s simply so much more ambitious than voxels in terms of me actually being able to do it. It’s definitely one that merits future investigation.

I also tried to implement an agent based 3D pathfinding algorithm based on ants, and there may be some potential there however the curse of the extra dimension is obvious as the agents are much less likely to find their targets. But it could have some potential, as with other 3D pathfinding algorithms. Again, I thought this might solve the trunking issue.