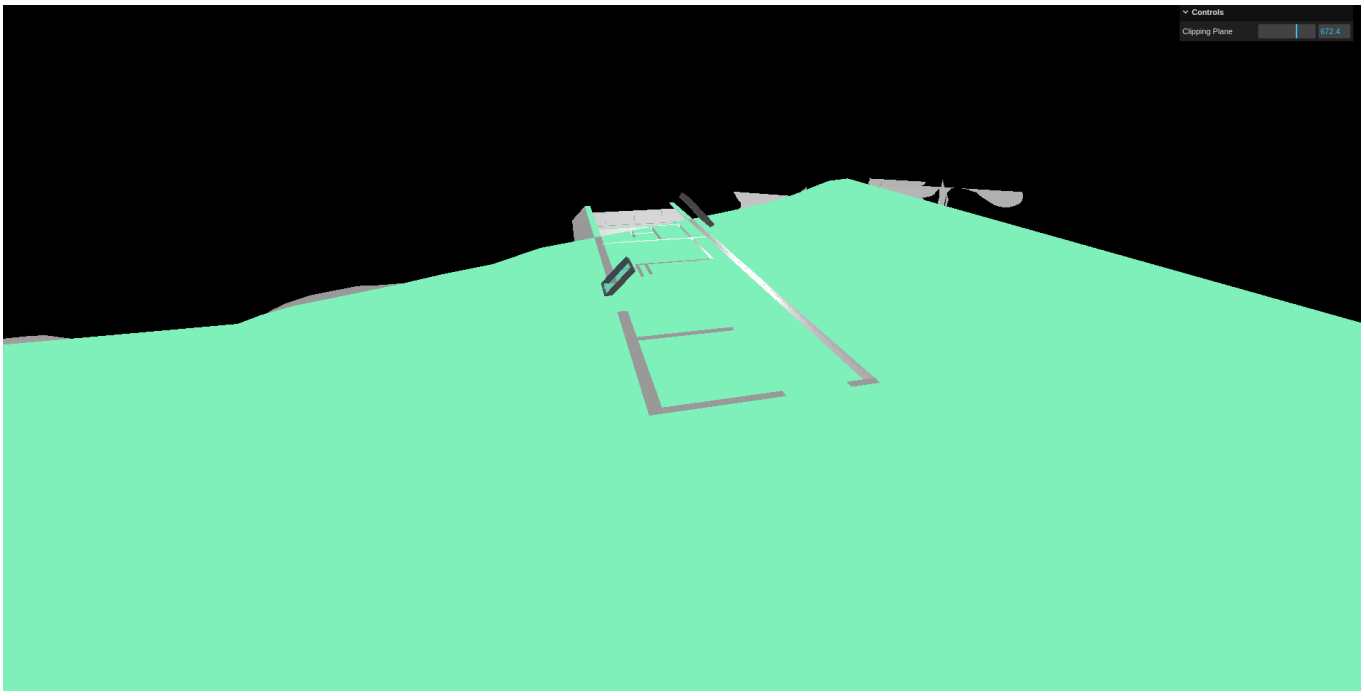
For Speckle I've been tasked to work on a solution for capping clipped geometry used for visualizing cross-sections which Speckle has not been able to fully solve yet. For this I've been giving a briefing which includes a link to a devlog describing the approaches that have been tried.

Getting Familiar

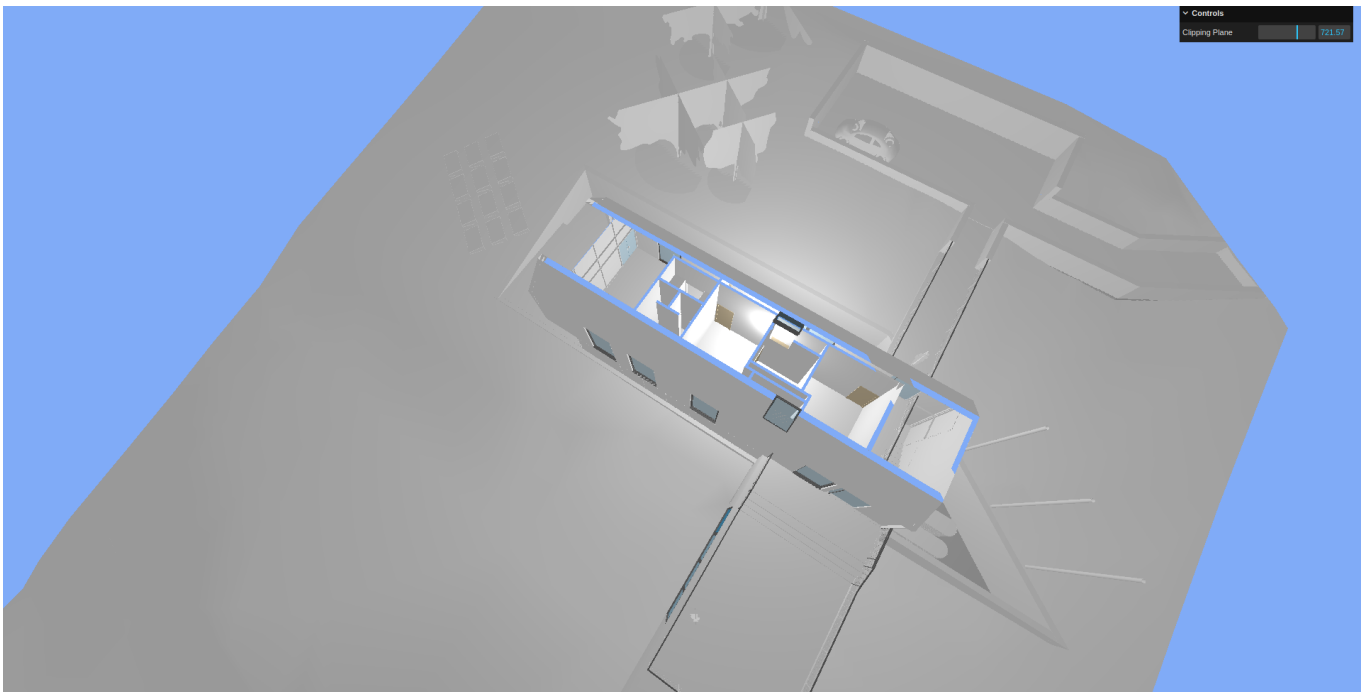
My knowledge of the problem is almost zero. I've never looking into this kind of problem before and also the approaches tried seem unfamiliar to me. Only the last post in the devlog seems familiar: generating a mesh from a bunch of points. This I have done before during my graduation internship. As I am unfamiliar with the problem my first step is understanding the problem better to make sure I solve the right thing. I also asked Alex to clarify some things to make sure I know what is expected.

After doing some research and got a good idea about what to build, I started with experimentation. I watched some videos about how the stencil buffer works and stencil capping. Then I wanted to try this method for myself to see what the problem is exactly and why Alex couldn't use this method.

I created a basis scene in Three.js and by reading the Three.js example's source code I implemented stencil capping on the house.fbx. I noticed that meshes that have no depth (like the terrain plane) messed up the stencil buffer and that the technique only works on meshes that are solid.

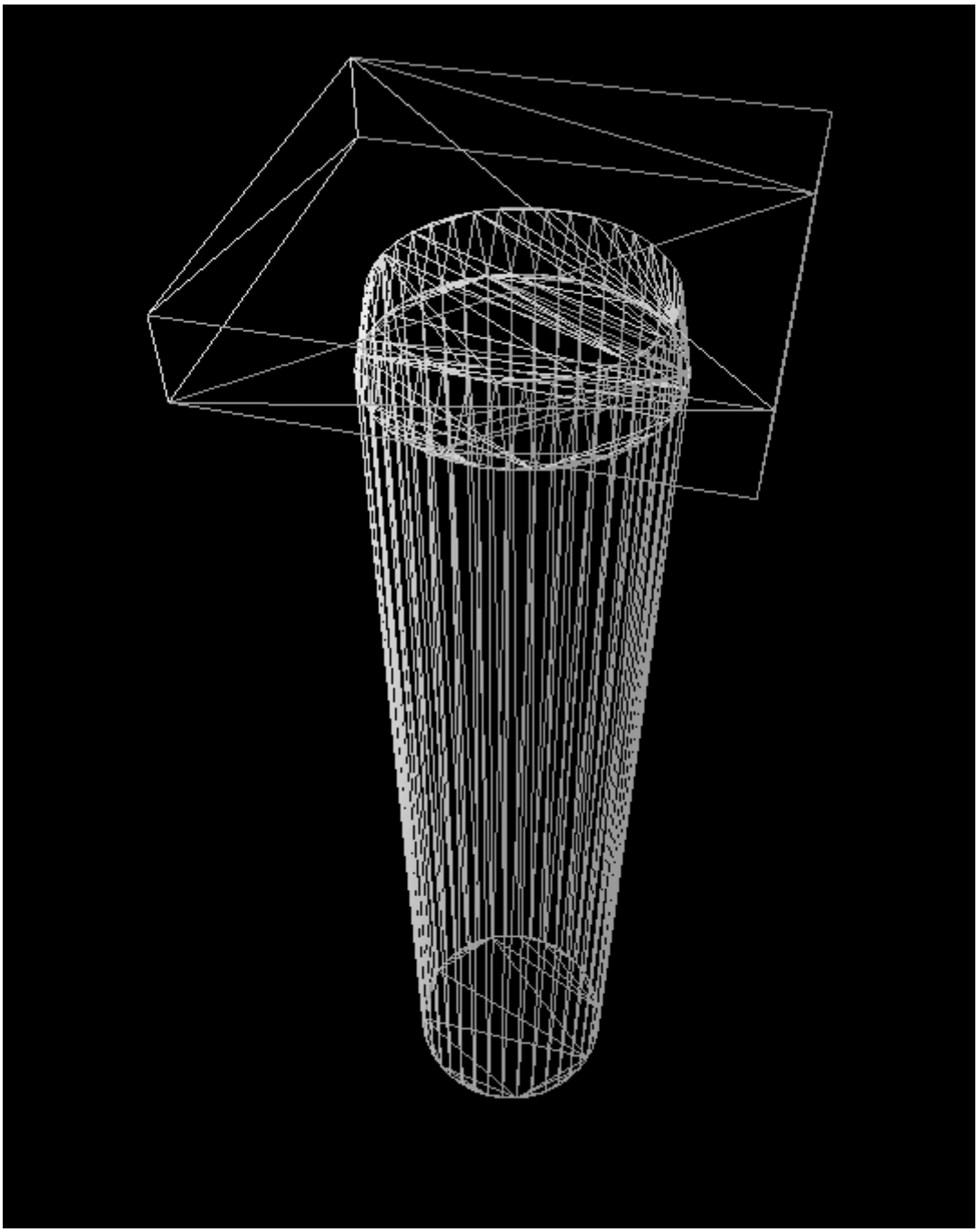


One solution to this problem is to try to detect meshes that are not solid and filter those out of the stencil capping method. Thus only applying the stencil capping method on valid solid meshes. I learned about manifold and non-manifold meshes and figured that manifold meshes are solid meshes. I utilized a doubly-connected-edge-list (DCEL) library and applied it on the meshes. I could then query how many neighbor triangles the edges between the vertices have. If every edge has two triangles it means the mesh is manifold and there are no borders or holes in the mesh. This technique worked, however there were problems.



It so happens that in the house.fbx scene there are meshes that are non-manifold but would still be valid to apply the stencil capping technique on because they are solid objects. For instance

take this pillar in the middle of the house pictured below. This pillar is solid but it's non-manifold because there are edges that share 3 triangles (the section in the pillar at the top, below the top). And so this mesh would be filtered out of the stencil capping technique. I tried to refine my method of detecting which non-manifold meshes to keep and which to filter out and in the end I decided to ditch the stencil capping approach as this would have taken me too much time. I would need to learn all about the DCEL data structure and try to change it to fit my purpose. I am sure with clever computational geometry techniques you can get this approach to work though and have a fast real-time method for capping geometry. I would give this method more thought in the future.



Real Geometry

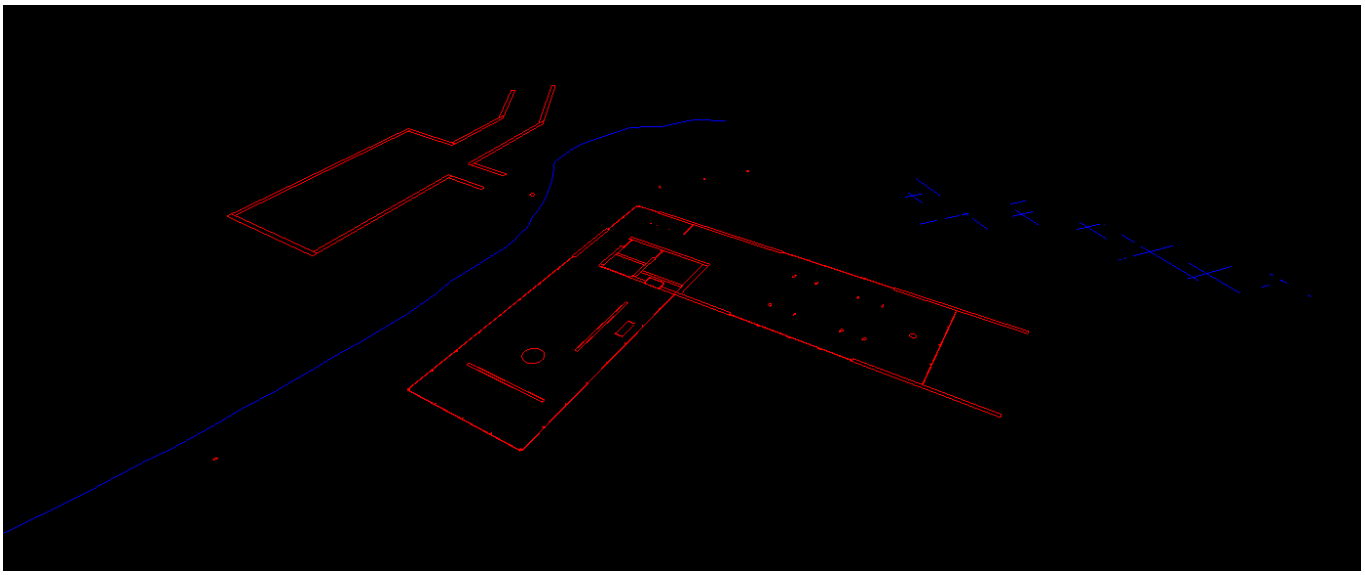
My next approach would work on literally every kind of geometry. The method is described in the paper by Scherzinger, Brix and Hinrichs. 2011. The idea is to check for intersections between every face of every mesh against the clipping plane. If a face intersects you'll get 2 intersection points. Those 2 points form a segment. If you do this for every face you'll get a whole bunch of segments, at least for the faces that intersect with the clipping plane. Then in those segments you can check for loops. If the end of a segment is the beginning of another segment, they are connected. If the last segment connects with the first, you get a loop (make sure to check for distance < epsilon because of precision errors). The loops that don't have any inner loops form the outside of a polygon. The first loop inside the outside loop means it represents a hole. Another loop inside that one is again the outside of a new polygon etc.

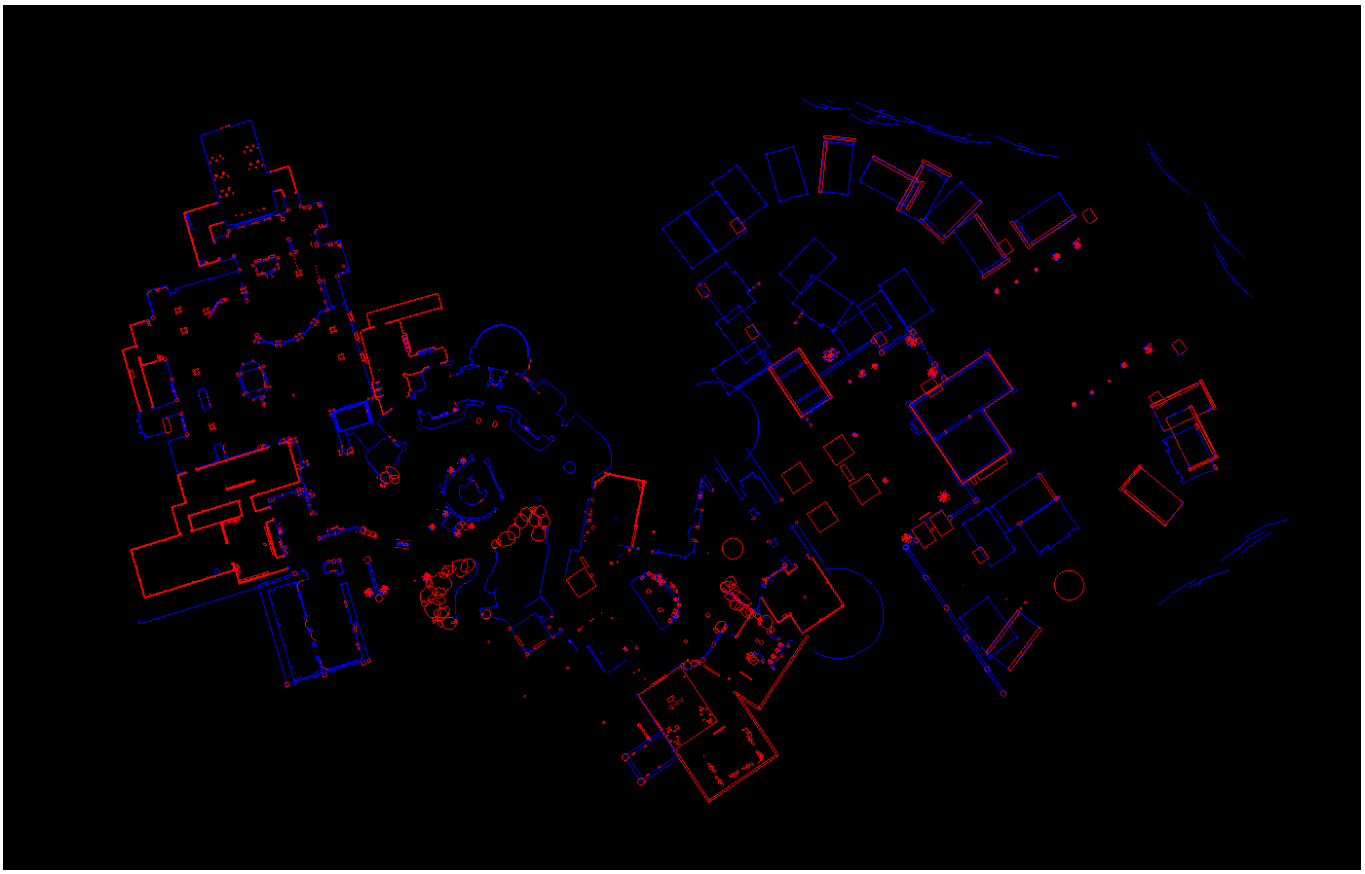
With this method you can successfully detect solid geometry because geometry that doesn't form a closed loop once it intersects with the clipping plane is not solid. This method works really great but the trade-off is that it's slower than the stencil capping method. Because of this I only execute this method once the clipping plane stops moving, just like the Autodesk viewer. Also keep in mind due to time constraints I implemented everything on the CPU in a brute-force manner. Some things like checking if a face intersects with the clipping plane in order to get all the segments can be moved to the GPU to increase performance.

Also the method of detecting which loops form the outside of a polygon and which form the holes is programmed in a brute-force manner due to time constraints. It can be optimized by using a sweep line algorithm as described in the paper. I however have no experience with this method yet (although I watched some lectures on the topic + read some parts of my computation geometry book) and it would have taken me too much time to implement this properly.

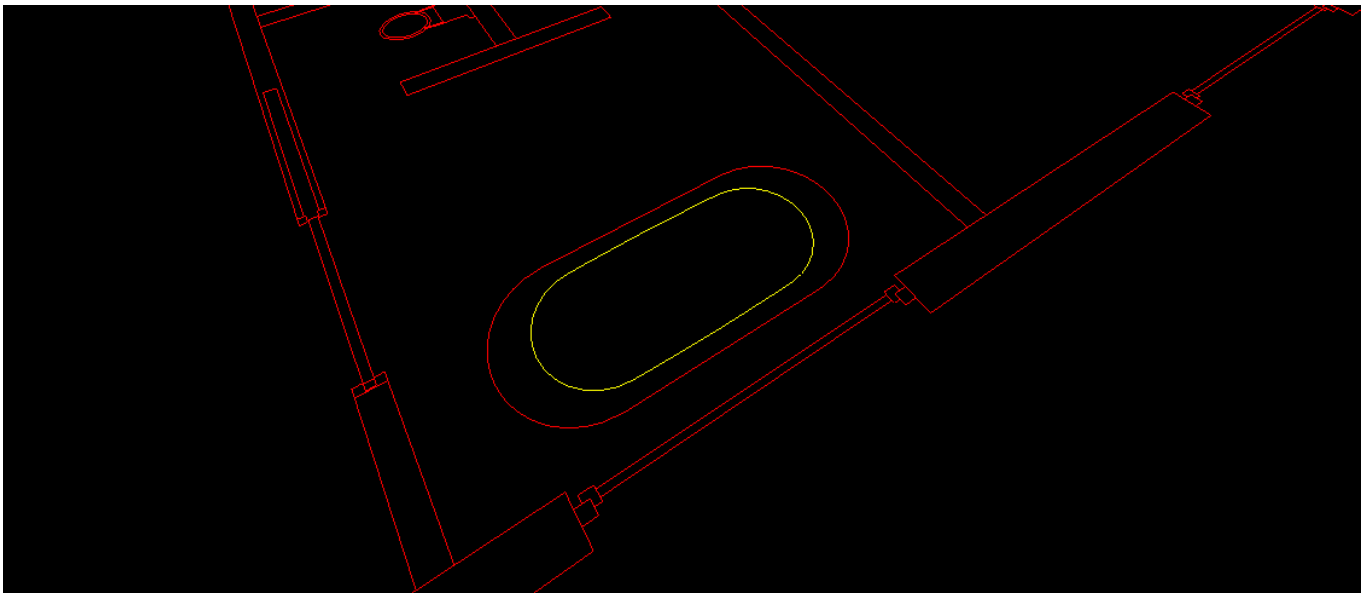
For the triangulation I used the earcut algorithm. It's a fast algorithm and from experience way faster than constrained delaunay. The trade-off is that you can't use steiner points but I saw no need for supporting that at this time.

Segments & Loop Construction

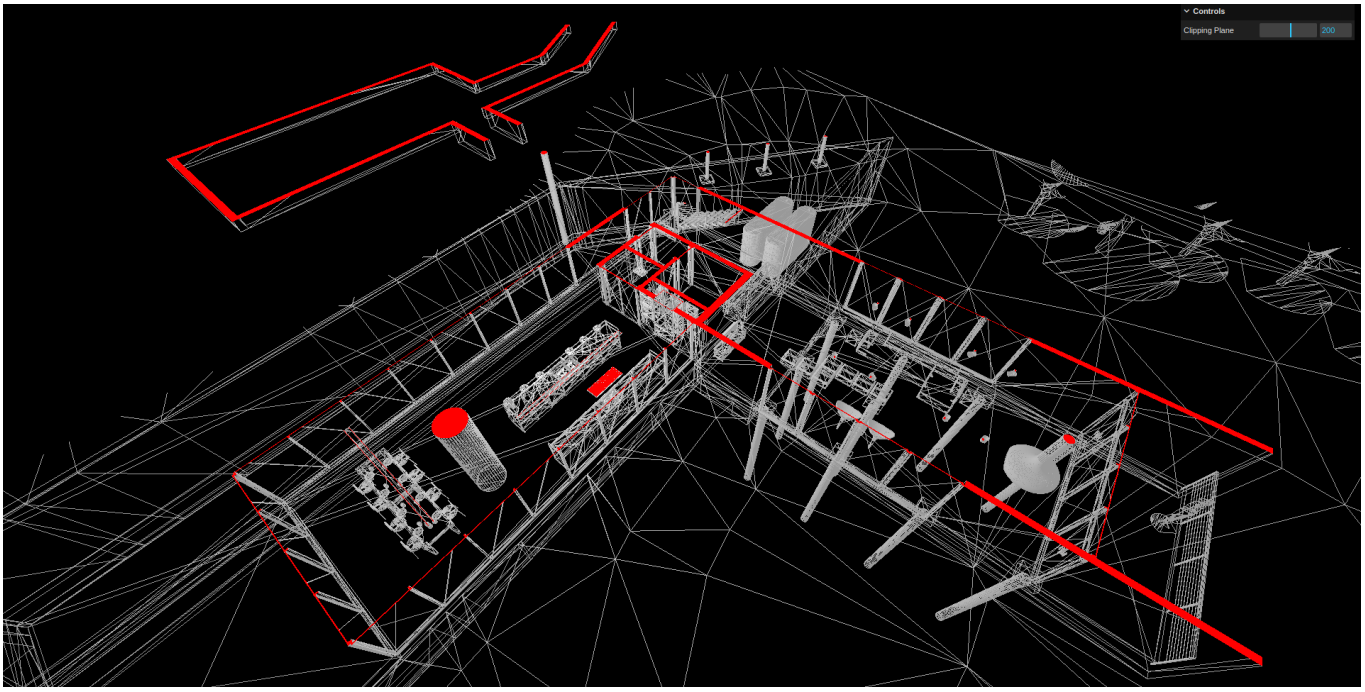


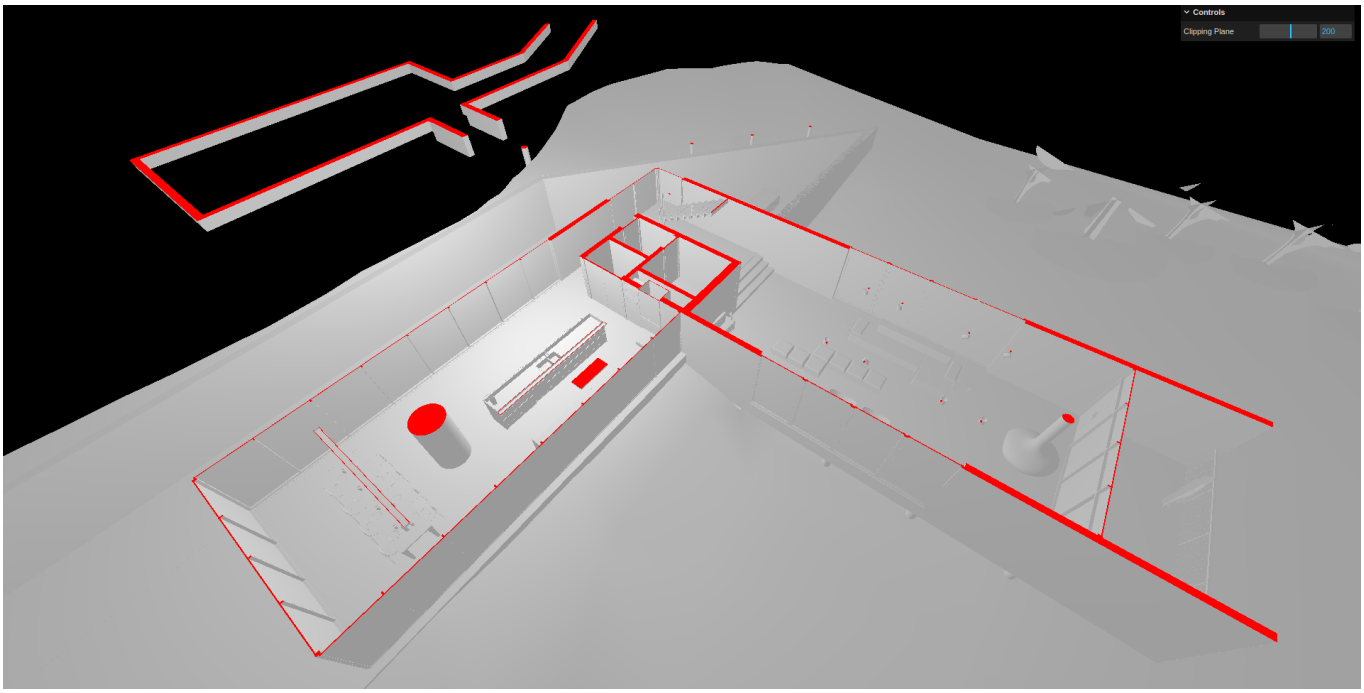


Inner Loop Detection



Triangulation





Future Work

- **STENCIL CAPPING**: Find a solution to effectively identify solid manifold AND non-manifold geometry on which to apply the stencil capping method.
- **REAL CAPPING**: Move clipping / segment creation to the GPU
- **REAL CAPPING**: Improve inner/outer loop identification algorithm with plane sweep / sweep line algorithm
- **REAL CAPPING:** Due to time constraints I only apply capping on loops without inner loops/holes in them. Improve the capping method to include geometry with holes (bathtub etc).

Demo

https://nickvanurk.com/capping/