Path Optimization for the Rough Carving of 3D models

Chris Allum

***Abstract*— Carving a 3D model using subtractive methods is a difficult task. Artists can can spend years learning the art. In this paper, I propose an algorithm that would allow a robotic arm equipped with a cutting tool such as a chainsaw to cut away at a material, leaving behind a carving resembling the desired 3D model. The general process of calculating the carving process of a model can be broken into three sections: (i) computing a simplified geometric model based on the input 3D model, (ii) planning the order and optimal path of cuts that a robotic arm should preform in order to efficiently carve the desired shape, and (iii) using inverse kinematics and control methods to move a robot arm to preform the cuts. In this paper, I will explore geometric simplification, cut generation, and cut ordering. However, due to time constraints, I was not able to explore the path planning of the robotic arm after the order of cuts have been generated and ordered.**

I. INTRODUCTION

The use of a robotic arm wielding a chainsaw to cut 3D models out of a starting piece of material, while humorous, presents a wide range of technical challenges. Issues such as how to represent a 3D model using simplified geometry, assessing whether a given geometry can be successfully cut by a chainsaw, how one should can preform a series of cuts to yield a desired shape, and the efficiency and optimality of different cutting orders are all non trivial. Such challenges are typically regarded as part of the artistic sculpting process. The challenge of carving an object with straight cuts has an infinitely large configuration space, with no clear ”optimal” solution. However, in this paper, I aim to explore these challenges, and demonstrate a possible set of algorithms that would allow a robotic arm to cut a 3D model with decent efficiency.

II. PRIOR WORK

A. *Robotic Cutting*

The challenge of using a robotic chainsaw to cut a 3D model was first brought to my attention in a youtube video by Shane Wighton where he builds exactly such a robot [6]. In his video, he uses a process of gradient descent to calculate a set of wedge shapes that a robot can cut in order to remove material, leaving behind the desired shape. Work similar to this problem was done by Hattab *et al.* [5] in a paper in which the authors explored using a projector to display the optimal cutting order on a piece of wood that a user would then carve. In this paper, the authors decide a cutting order by first looking at an input object at a set of discrete angles and determining which series of cuts would remove the most material at any given time. Similar work has also been done by Zhu *et al.* [9] in which the authors explore cutting 3D objects using a robotic hot wire. In their work, the authors first compute a simplified geometry, and then determine the cut order by top to bottom, prioritizing faces that share a common edge.

All of these approaches, while functional, do not necessarily provide the most efficient cutting. The methods proposed by Wighton [6] and Hattab [5] both attempt to achieve efficiency by removing the most material with each cut. However, if sequential cuts are on different sides of the model, there could potentially be wasted time in waiting for the robot to move to the other side of the model. Removing the most material at a time may also not necessarily be the most time efficient method. The area of material removed is not necessarily directly correlated with the time it takes to make the cut. A better method might be to minimize the surface area of cuts preformed. Zhu *et al.* [9] tried to solve the problem of efficiency by cutting adjacent faces, working from the top down. This may be efficient for some geometries, but this approach is limiting.

B. *3D model simplification*

The first step to cutting an object is to determine what the final shape should look like. For most models, given that they are made up of thousands faces, it would take too long to cut each individual face. Therefore, it is necessary to first simplify the model. Work on geometric model simplification has also been done by Zhu *et al.* specifically for the process of rough cutting [7] [8]. A comprehensive survey on different mesh simplification algorithms by Cignoni *et al.* [1] contains many examples of model simplification. Talton *et al.* also provide a survey on mesh simplification. As mentioned earlier, Hattab *et al.* [5] solved this issue by flattening the 3D model into a 2D shape from a discrete set of perspectives, and then computing a set of cuts to remove volume from these perspectives.

Of these methods, while the general process of mesh simplifications proposed by [7] and [8] work to reduce the number of triangles on a 3D mesh, they don’t necessarily ensure that the resulting shape can be actually cut. There may be sections that a robotic arm may not reach, or sections that are not possible to cut. Therefore, I believe, the geometric simplification methods by Hattab *et al.* [5] is the most applicable to this problem. Not only does it simplify the geometry, but is simultaneously ensures that all cuts are valid. The only potential problem with this method is that since we are looking at the model from a discrete number of perspectives, there may be regions where material is not removed if it is not in any of the perspectives.

C. Cut Ordering

Determining an order of faces to cut is a complex problem. For a detailed model, there may be over a hundred different cuts. For n cuts, there are $n!$ different ways those cuts could be ordered. Therefore, trying to outright determine the best order is computationally not possible. One possible way to determine the order of cuts is to represent each cut as a node on a graph. If each cut has a weight associated with the surface area of that cut, and we are trying to minimize the total surface area, the problem of cut ordering could be constructed as a traveling salesman problem were the sum of node weights should be minimized. This is still a difficult problem. One approach to solving such a problem is through using machine learning. Work on similar techniques have been explored by Gambradella *et al.* [3] in their paper on Q-learning and the Traveling Salesman Problem. The problem has also been explored by Cost *et al.* [2] in their paper on the traveling salesman problem and deep reinforcement learning.

Another possible solution to this challenge is some sort of policy that, while not necessarily optimal, results in a close to minimal cutting time. I will explore such policies later in this paper.

III. Methods

A. 3D Model Simplification and Admissible Cut Generation

To generate a simplified model and admissible cuts, I am using a method similar to that which Hattab *et al.* [5] proposed. The first step I perform is to convert a stl mesh model into a point cloud. Figures 1 and 2 show the conversion from an stl mesh to a point cloud using the vertices of the mesh as points.

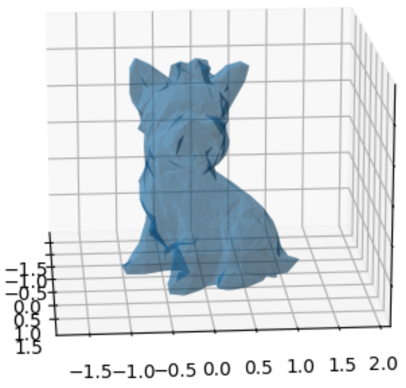

Fig. 1: Original stl model of a dog

After converting the model into point cloud, I then generate a set of perspectives as defined by points evenly spaced on the surface of a sphere. To generate an evenly spaced set of perspectives, I am using the Fibonacci sphere algorithm [4] as seen in algorithm 1.

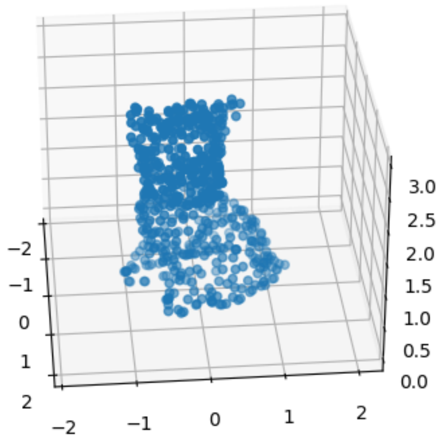

Fig. 2: Point cloud generated using the vertices of the original stl

Algorithm 1 Fibonacci Sphere Algorithm

```
procedure GENERATE_PONTS(n_samples)
    Let $points$ = []
    Let $\phi = \pi(\sqrt{(1/2)} - 1)$                ▷ Golden ratio
    Let $idx = 0$
    while idx ≤ n_samples do
        $y = 1 - (idx/(samples - 1)) * 2$
        $radius = \sqrt{1 - y^2}$
        $\theta = \phi \cdot idx$
        $x = \cos\theta \cdot radius$
        $z = \sin\theta \cdot radius$
        points[idx] = $(x, y, z)$
        $idx = idx + 1$
    end while
    return $points$
end procedure
```

From each x, y, z location of a point on a sphere, I can then define a set of rotation matrices in terms of θ and ϕ where θ is a rotation around the z axis and ϕ is a rotation around the x axis. By rotating the point cloud using these rotation matrices, I can get a new point cloud where the (x, y) coordinates of the point cloud are a flattened view of the original point cloud. Examples of this flattened view can be seen in figures 3 and 4 which depict the the view from the side and the top respectively.

After generating a set of perspective views, I then generate a set of cuts. To do this, I am deviating from the method proposed by Hattab *et al.* [5]. I am instead calculating the convex hull of the points as well as the alpha shape of the points. Each line segment in the convex hull represents a valid slice that one could make to remove material, while line segments in the alpha shape include line segments that could be concave cuts. The convex hull and alpha shape of the point cloud seen in Figure 3 can be see in figures 5 and

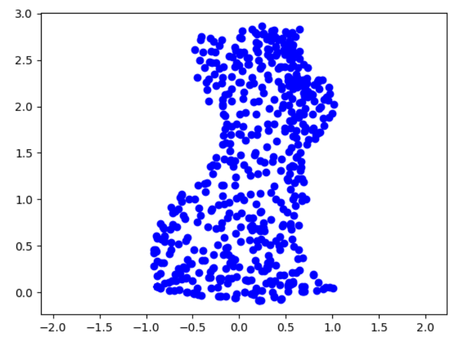
Fig. 3: Flattened point cloud from the side perspective

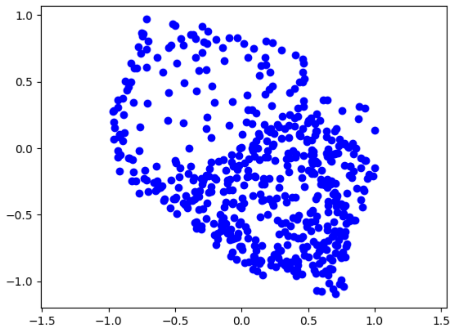
Fig. 4: Flattened point cloud from the top perspective

6 respectively.

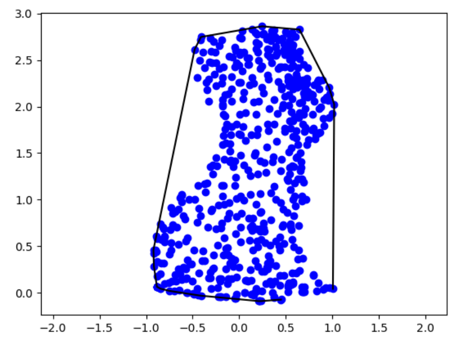
Fig. 5: Convex hull of the side perspective points

One thing to note about the convex hull and the alpha shape is that every line segment in the convex hull is a valid cut, but it doesn't cut any concave geometry, while the line segments in the alpha shape can cut concave geometry, but not all line segments are valid cuts. The reason that some line segments are not valid in the alpha shape is that they contain consecutive concave points. This means that the ends of such line segments cannot be reached from outside the model. For a cut to be valid, at least one end of the line segment must be assessable from outside of the mode.

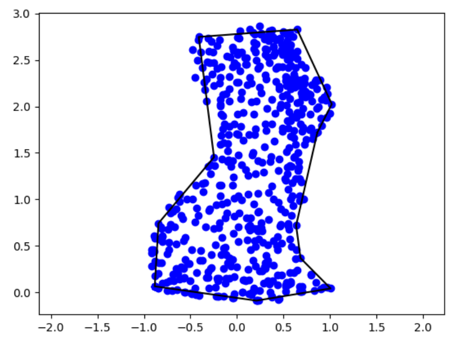
Fig. 6: Alpha shape of the side perspective points

In order to get the best of both worlds, we take points from both the convex hull and the alpha shape. We take points from the alpha shape if they are also in the convex hull. We also take some concave points from the alpha shape, but we also ensure that we do not have any consecutive concave points. This results in a set of line segments that are fully capable of being cut, but also provide a decent fit to the geometry. The final combination of the convex hull and alpha shape can be seen in 7.

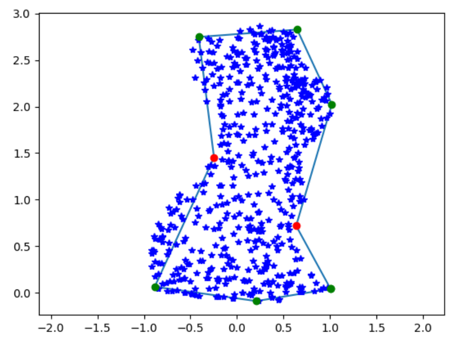
Fig. 7: Final outline of the model using points from both the convex hull and alpha shape. Convex points are green and concave points are red.

Using this outline, we can generate distinct valid cuts to the model. A cut that is convex is defined by a single line. While a concave cut is defined by a set of two lines. This can be seen in Figure 8. We can see that each concave section is defined by two lines that meet at a point, while a convex cut is just a single line.

Using these cut lines, we can project them into 3D and then perform the inverse rotation of that which we used to obtain the perspective point cloud. This gives us a set of 3D planes that define a set of cuts. This can be see in Figure 9

If we start our original model as a rectangular prism, with each successive set of cuts, we can slowly form the shape of the original mesh. The cut planes and the starting rectangular block can be seen in figure 10.

After removing the material from the starting un-carved

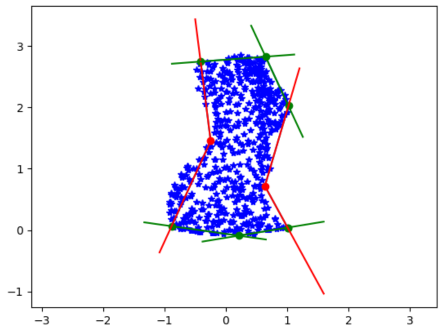
Fig. 8: Cut lines. Convex cuts are green, concave cuts are red

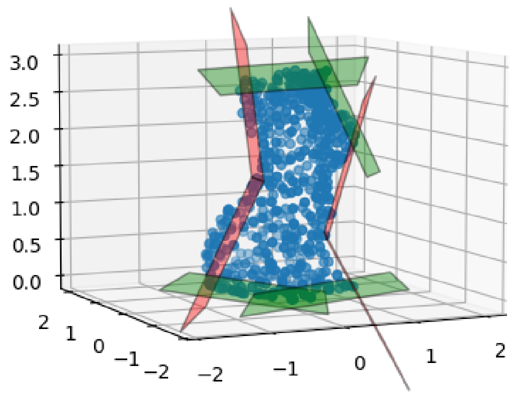
Fig. 9: Cut planes. Convex cuts are green, concave cuts are red

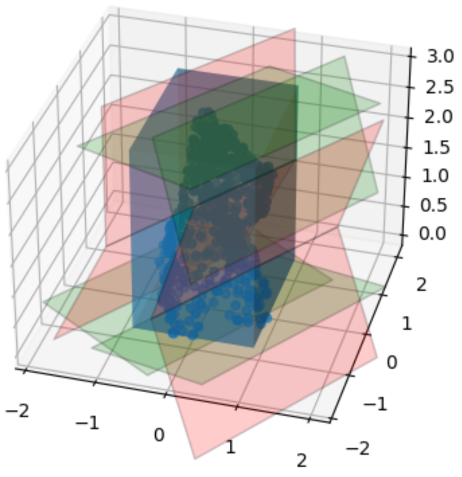
Fig. 10: Cut planes overlaying the starting un-carved block

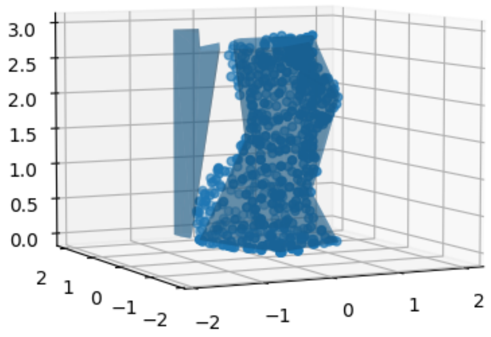
Fig. 11: Model after a cuts from a single perspective

block using the set of planes seen in Figures 9 and 10, we are left with the the material seen in Figure 11. After the set of cuts from all of the perspectives, we get a final simplified geometry. This can be seen in figure 12.

This simplified geometric shape is roughly the same as the source stl, but it is fully made by removing material through admissible convex and concave cuts.

B. *Cut Ordering*

After we have generated a set of valid cuts that will remove material from our starting rectangular prism and will yield a final geometry that is similar to the source stl geometry, we need to determine the order of cuts. As mentioned earlier, for n cuts, there are $n!$ different orderings of the cuts, yet no matter the order, we will get the same end result. Therefore, the challenge is to determine what order to cut the faces in.

The approach that Hattab *et al.* [5] and Wighton [6] took was to perform the cut at each stage that removes the most material. This makes intuitive sense. The quickest way to remove material would be to remove large chunks of material at a time. However, for this is not necessarily the quickest way to remove material. If we make the assumption that the the time it takes to make any given cut is mostly proportional to the surface area of the performed cut, then the method to reduce the total cutting time would be to perform a set of cuts such that the sum of all surface area cut is minimized. While removing the largest pieces of material with each successive cut often also tends to perform cuts that are surface area-efficient, it is not guaranteed to be optimal. Therefore, I wanted to determine if there is a policy that if followed, will always result in the geometry being cut such that the cumulative surface area that was cut is minimized.

C. *Optimal Cut Ordering Hypothesis*

In this section, I will propose a cut ordering policy that I believe is optimal. If we assume that we have order of cuts O, then the total surface are that is cut is

$$SA_{total} = \sum_{i=0}^{i=len(O)} SA(O_i) \tag{1}$$

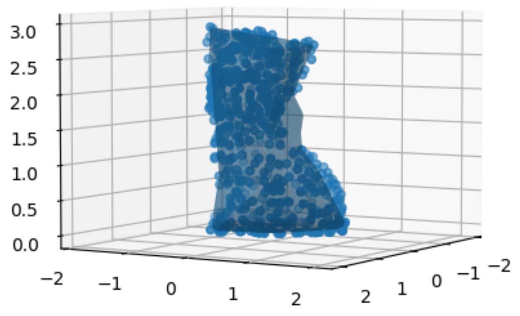

Fig. 12: Model after a cuts from a all perspectives

where $SA(\cdot)$ the surface area of a cut, and O_i is the i^{th} cut in the ordered list of cuts O. Furthermore,

$$SA(O_i) = SA_{unique}(O_i) + SA_{shared}(O) \tag{2}$$

In equation 2, the surface are of any given cut is actually a summation of the surface area that is unique to the face that it is carving and a the surface are of the cut that is in a region that is shared with other cuts. This shared surface area can be seen in Figure 8. Each part of the cut line that extends past the bounded region is a cut that extends into a region that could also be removed by another cut. As a result, as other cuts are made, the SA_{unique} associated with each cut remains the same. However, SA_{shared} will either remain the same or decrease. This makes intuitive sense. For two cuts A and B that are close to each other, by performing cut A, you are removing material that cut B would have had to also remove. Therefore, the $SA_{shared}(B)$ has decreased. Likewise, if cut B is performed first, the $SA_{shared}(A)$ will decrease.

Now, if we are trying to minimize total surface are cut, we are trying to minimize the following equation:

$$SA_{total} = \sum_{i=0}^{i=len(O)} \left[SA_{unique}(O_i + SA_{shared}(O_i)\right]) \tag{3}$$

$$SA_{total} = \sum_{i=0}^{i=len(O)} SA_{unique}(O_i +) \sum_{i=0}^{i=len(O)} SA_{shared}(O_i) \tag{4}$$

Since $SA_{unique}(\cdot)$ is a constant with respect to each cut and does not change depending on the order of cuts, we can simplify equation 4. We can minimize the total surface are cut by finding an order O such that

$$SA_{total} = \sum_{i=0}^{i=len(O)} SA_{shared}(O_i) \tag{5}$$

is minimized.

In order to minimize the sum of the shared surface area over a series of cuts, I believe that the optimal policy would be to, with each iteration, select the cut which reduces the total shared surface are of all cuts that have not yet been cut. This intuitively makes sense. If we can, with each step, remove the most shared surface area, in the long run, the total summed shared surface will be minimal. In a way, this is an altruistic policy. The optimal cut at each iteration is the one that reduces other cuts' future surface area that would have needed to be cut. The algorithm for which I am using is seen below in Algorithm 2

Algorithm 2 Optimal Cut Policy

```
procedure MAKE_CUTS($C$: List[Cuts])
    Let $mesh$ = starting_mesh
    Let $n\_cuts = 0$
    while $n\_cuts \leq$ length($C$) do
        $optimal\_idx$ = GET_OPTIMAL_IDX($C$, $mesh$)
        cut mesh using C[$optimal\_idx$]
        $n\_cuts$ = $n\_cuts + 1$
    end while
    return
end procedure
procedure GET_OPTIMAL_IDX($C$: List[Cuts], $mesh$)
    Let $weights$ = []
    Let $cut\_idx = 0$
    while $cut\_idx \leq length(C)$ do
        Let $cut = C[cut\_idx]$
        if cut.is_cut then $weights.append(-1)$
        else
            let $temp\_mesh$ = copy($mesh$)
            Let $initial\_shared\_sa$ = 0
            Let $other\_cut\_idx = 0$
            while $other\_cut\_idx \leq length(C)$ do
                $other\_cut$ = C[other_cut_idx]
                $initial\_shared\_sa$ = $initial\_shared\_sa$ + SA($other\_cut$)
                $other\_cut\_idx$ = $other\_cut\_idx$ + 1
            end while
            temp_mesh.make_cut(cut)
            Let $end\_shared\_sa$ = 0
            Let $other\_cut\_idx = 0$
            while $other\_cut\_idx \leq length(C)$ do
                $other\_cut$ = C[other_cut_idx]
                $end\_shared\_sa$ = $end\_shared\_sa$ + SA($other\_cut$)
                $other\_cut\_idx$ = $other\_cut\_idx$ + 1
            end while
            $weights.append(initial\_shared\_sa - end\_shared\_sa)$
        end if
        $cut\_idx = cut\_idx + 1$
    end while
    Let $optimal\_idx$ be index of largest value in $weights$
    return $optimal\_idx$
end procedure
```

D. Optimal Cut Policy Results

In order to test the proposed optimal cut policy, I also implemented a policy which, at each iteration, chooses the cut which removes the most volume. I also implemented a policy which chooses the cut order randomly. I then generated a range of geometries using the same source model. Geometries where I allowed more flattening perspectives had more detail and more total needed cuts. The results for each policy are below in Tables I and II.

Cut Policy Surface Area Comparison			
Number of Cuts	Surface Area Policy	Volume Policy	Random
64 cuts	35.556	34.28	49.79
70 cuts	35.86	32.83	53.08
108 cuts	38.85	36.94	51.89
128 cuts	34.82	34.51	57.92

TABLE I: The total surface area cut by each policy for a range of number of total cuts

Cut Policy Avg. Time Comparison			
Number of Cuts	Surface Area Policy	Volume Policy	Random
64 cuts	3553 ms	34.25 ms	8.33 ms
70 cuts	4649 ms	38.32 ms	9.81 ms
108 cuts	10003 ms	55.10 ms	7.51 ms
128 cuts	15020 ms	63.94 ms	8.28 ms

TABLE II: The average time it took for each policy to compute the optimal cut for a range of number of total cuts.

E. Results of Policies

I believed that a policy that optimizes for surface area would perform the best in terms of minimizing total surface area cut. However, the results in Table I indicate that this isn't the case. In fact, the policy that minimized total cut surface area was the policy of removing the most material at any given time. However, between my proposed policy and the volume removal policy, the end results are very similar. Especially compared to picking a cut randomly, both optimization policies performed significantly better.

There are many reasons why my proposed policy did not out-perform the volume policy. The first reason is that my proposed policy is wrong. I have not done a rigorous proof that demonstrates that my proposed policy actually results in the lowest total surface area cut in the long term, so this is possible. Another option is that my simulation is faulty. This is very likely. Calculating the surface area for a cut in a complex mesh is not easy, and it is quite possible that I have make a mistake somewhere in my simulation code.

Regardless of whether my proposed policy is optimal for minimizing cut surface area, there are other considerations to take into account. For one, the computation time of my policy is vastly greater than a volume policy. As seen in Table II, a surface area policy takes over 100 times longer to compute. This might be due to inefficient programming of the policy. However, even if that were the case, it's clear that this policy is not vastly better than the volume policy and the volume policy is vastly quicker to compute. Therefore, I conclude that my proposed policy, while it seems like it would work on paper, in all practicality, is inferior to a policy of removing the most volume at any given time.

F. Robotic Arm Motion

I unfortunately was not able to do much work on calculating optimal robot arm motion to perform a series of cuts. However, the next steps would be to use the the order of cuts generated by one of the policies listed above and calculate the shortest path for a robotic arm to move in a way such that each cut is complete. This is also a non-trivial task. There are any infinite number of ways to perform the cuts, from the angle at which the makes the cut, to the path the robot takes from cut to cut.

IV. Future Work

If I were to continue this work, there are some clear next steps. The first step would re-work the simplified geometry and cut generation algorithm I am using. While functional, the end geometry doesn't quite look like the source geometry. There are sections where the algorithm removes too much material, which causes features to be lost, and there are also sections where the algorithm does not not remove enough material, and features are hidden.

Beyond a better geometry and cut generation, as mentioned above, I would need to work on generating the path for the robotic arm to perform all the cuts. I think it would also be worthwhile to explore using a physics simulator to test the robot arm movements.

References

[1] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998.

[2] P. R. d O Costa, J. Rhuggenaath, Y. Zhang, and A. Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian conference on machine learning*, pages 465–480. PMLR, 2020.

[3] L. M. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In A. Prieditis and S. Russell, editors, *Machine Learning Proceedings 1995*, pages 252–260. Morgan Kaufmann, San Francisco (CA), 1995.

[4] Á. González. Measurement of areas on a sphere using fibonacci and latitude–longitude lattices. *Mathematical Geosciences*, 42:49–64, 2010.

[5] A. Hattab and G. Taubin. Rough carving of 3d models with spatial augmented reality. In *Proceedings of the 3rd Annual ACM Symposium on Computational Fabrication*, pages 1–10, 2019.

[6] S. Wighton. Chainsaw + robotic arm = art? `https://www.youtube.com/watch?v=ix68oRfI5Gw`. Accessed: 2023-11-07.

[7] J. Zhu, T. Tanaka, and Y. Saito. 1 3 d mesh simplification for freeform surfacing. 2005.

[8] J. Zhu, T. Tanaka, and Y. Saito. A rough cutting model generation algorithm based on multi-resolution mesh for sculptured surface machining. *Journal Template*, 00, 01 2005.

[9] J. Zhu, T. Tanaka, and Y. Saito. Rough machining process and its simulation for robot integrated surface sculpturing system. 01 2006.