Computational spoon shape optimization

for use by patients with motor impairments

Martin Mossler Rockström

[Various information about institution?]
[style guide unclear]

(1) Abstract

This thesis investigates whether the shape of eating utensils can be optimized using computational methods. This is done by creating a system to do so for a specific case, spoons, and see if any insurmountable problems are encountered.

As this system is meant to optimize spoons rather than eating utensils in general, there are a few simplifications that can be done when modelling the eating process. The spoons are represented by a set of parameters that are further divided into groups. The first group representing the bowl, followed by a variable number of handle segments.

The parameter sets are evaluated by calculating the volume between the lowest point on the rim and the bottom of the bowl, giving both the volume remaining at the end as well as the volume spilled over the course of the motion. This, together with the final direction and tilt of the bowl as well as a measure of spoon complexity, is processed into a score value.

The chosen optimization method is gradient descent, with a variable step length.

The results show that the optimization process does work, and the resulting spoons look plausible. The conclusion is that the approach is viable, but there are a number of changes and improvements needed to make things work for practical purposes.

Table of contents

(1) Abstract 2
(2) Introduction 5
(2.1) Purpose of thesis 7
(2.2) Background & related work 8
(2.2.1) Healthcare 8
(2.2.2) Optimization 8
(2.2.3) Optimization in healthcare 9
(3) Theory 10
(3.1) Terminology 10
(3.2) Optimization 11
(3.2.1) General optimization theory 11
(3.2.2) Gradient descent 12
(3.2.3) Other methods 14
(3.3) Geometric transformations 17
(3.3.1) Homogenous coordinates 17
(3.3.2) Order of operations 18
(3.3.3) Tait-Bryan angles 18
(3.4) Triangle meshes 19
(4) Method 21
(4.1) Overview 21
(4.2) Spoon modeling 21
(4.2.1) Representation goal 22
(4.2.2) The chosen representation 24
(4.2.3) Movement representation 26
(4.2.4) Volume calculation 27
(4.3) Mesh creation 29
(4.3.1) Vertices 29
(4.3.2) Edges and Surfaces 30
(4.3.3) Coordinate transforms 30
(4.4) Optimization 31
(4.4.1) Objective function 31

(4.4.2) Optimization methods 35
(5) Implementation 37
(5.1) Software used 37
(5.2) Representations 37
(5.2.1) Parameter formats 37
(5.2.2) Parameter space 38
(5.2.3) Meshes 39
(5.3) Optimization 39
(6) Results 39
(6.2) Optimization 41
(6.2.1) No movement 42
(6.2.2) Tilted initial spoon 43
(6.2.3) Upright grip 45
(6.2.4) Turned left 46
(6.2.5) General tendencies 48
(6.3) Printed spoon 48
(7) Conclusions and Discussion 49
(7.1) Discussing the results 49
(7.1.1) Spoon parameters 49
(7.1.2) Objective function 50
(7.2) Ethical and societal implications 50
(7.3) Further work 51
(7.3.1) Improvements for the current goals 51
(7.3.2) Work for the greater project 51
(8) Bibliography 52

(2) Introduction

Motor impairments, as the name implies, impair the ability of a person to move their body. However, given the right tools, it does not necessarily need to impair their daily life, but the right tools are not necessarily the standard tools for any given task. This includes the silverware we use at mealtimes.

Currently, there are two main ways of alleviating this problem: having a personal assistant or using some sort of assistive technology. While the former can solve pretty much all problems caused by motor impairments, it can be expensive to hire a personal assistant. It also makes the person with the impairments dependent on the assistant(s), when many people would prefer to be self-sufficient.

The other alternative, using assistive technology, can be done in two primary ways. Either the utensils are adapted to the disability, or the patient holds or wears something that helps them use the standard utensils. In both cases, there are a number of commercially available tools that help with more common problems. However, these standardized solutions might be insufficient, with the patient needing more personalised tools. In these cases, a physical therapist might be able to manually customise something for the use of the patient, but otherwise they are out of luck. From personal conversation with a physical therapist at KI, we know that personal customization of tools is not uncommon.

This thesis explores whether it is possible to optimize the shape of eating utensils using computational methods. The hypothesis is that it is possible, despite the difficulties posed by the high-dimensional design space, but it needs care and thorough evaluation of the optimization processes.

To do this, a way to model the spoons also needs to be created, as well as a simple way of representing the motions and a method of evaluating the suitability of the spoons. The spoons also need to be visualized, and be able to be exported to a 3D-printer.

A list of the notations used will be provided in the theory section.

The motions are represented as a series of rotation matrices corresponding to the orientation of the hand. The spoons are represented in a parameterised model, and their suitability is determined by how much content they can deliver, how much they spill, the orientation of the spoon at the end of the motion and the complexity of the spoon.

The parameterised model can be converted to a triangle mesh, suitable for display and 3D-printing.

The shape adaptation is done through computational optimization, and is evaluated through running several different examples of varying complexity. The results are then examined with parallel coordinate visualization, as to give a better understanding of the optimization landscape.

By monitoring the state of the spoons at different stages of the optimization process for several different motions, the concept does seem viable. We observed an intuitively well-working optimization process with our example data-sets. There are some problems, including one that prevents using the printed spoons, but those stem from the implementation rather than the idea itself.

The thesis is also done in connection to the larger project *Patient-specific design of silverware for patients with motor impairments*, serving as a platform for future improvements. The goal of said project is to create a process allowing a patient with physical disabilities to visit a physical therapist, record how they move their hands and/or arms when they eat, and leave some time later with 3D-printed utensils optimized for their use.

(2.1) Purpose of thesis

This thesis investigates whether computational optimization can be used to adapt the shape of eating utensils to suit given eating motions. The main part of the thesis will document the creation of a program that accomplishes said task, thereby proving that it is possible.

The problem this thesis is to solve is as follows:

"Find a way of creating an adapted spoon that suits a given eating motion and show it in action. The process should be implement it in Inviwo, and the resulting spoons should be in the form of triangle meshes."

We can assume that representations of both how the patient holds the spoon and how they move it when eating will both be given. The format of the end results are also specified, there should be a triangle mesh of the spoon. Anything else is to be determined as part of solving the problem.

From this, it has been determined that the problem has the following parts:

- **Spoon model design:**
  A parameterized spoon model needs to be developed on a conceptual level and implemented. One example of the necessary capabilities of this parameterization is that the neck and bowl of the spoon need to take on different forms for different eating motions.
- **Optimization Goals:**
  A set of criteria for evaluating the spoons need to be defined, as well as methods for evaluating the spoon model according to said criteria. This can be things like amount of food delivered over the motion, and how much of the initial contents is spilled.
- **Optimization Method:**
  A suitable optimization method has to be selected and implemented among those deemed viable. To do this, some research has to be done on different methods. The computation time is of concern, since this would ideally be the starting point for software meant to run as the patient watches.
- **Visualization:**
  The optimized spoon needs to be visualized. To do so, a triangle mesh has to be created from the optimized design. An animation demonstrating the spoon in use should be available with the choice of exporting the results. The mesh should be able to be interpret by a 3D-printer.
- **Prototype implementation:**
  All methods are to be implemented in Inviwo, and be structured into a working prototype.
- **Result evaluation:**
  The behaviour of the prototype is to be evaluated to make sure it accomplishes the task it is designed for.

The reason Inviwo was chosen as the platform for this program was that the framework it provides is very useful for this project. For example, Inviwo has methods for geometric transformations and mesh structures to use, as well as a method of visualizing them.

(2.2) Background & related work

(2.2.1) Healthcare

One might ask, how widespread is the problem this project deals with? Is it worth the effort required for a project like this? As it turns out, it is. It is a fairly common problem.

Even though it is not the main point of the study, *Hospital inpatients' experiences of access to food: a qualitative interview and observational study* (Naithani, Whelan, Thomas, Gulliford, & Morgan, 2008) shows that food troubles stemming from not being able to use the utensils occur even at hospitals. The study was made to investigate why undernutrition in hospitals were not less common, as most patients were content with the hospital food.

The cause was that while the quality of the food was high, many patients had difficulties accessing the food. The reasons for this was divided into three categories: organizational, physical and environmental barriers. One of the physical barriers was not being able to use the utensils provided.

So the problem is a valid one, how about our chosen solution? Are customized utensils a good way to help people with physical disabilities? Once more, the answer is yes.

Effectiveness of adaptive silverware on range of motion of the hand (McDonald, Levine, Richards, & Aguilar, 2016) actually investigates if customized utensils actually help people with disabilities, focusing on the ability to grip and hold the utensils. This is done through defining the measurable characteristic of the Range Of Motion (ROM); how much different joints in the hand need to move and/or rotate to hold the utensil.

The utensils were evaluated using healthy individuals, as they would have full ROM available. The study concluded that the adapted utensils required a lower ROM than standard utensils.

The literature shows that adapted utensils are useful for people with motor impairments. This thesis will evaluate if this concept is possible to implement in practice.

(2.2.2) Optimization

The shape of an arbitrary object can be difficult to represent, let alone alter to suit specific purposes. If the general form of the object is known, a parameterized representation is often possible, albeit sometimes being very high-dimensional. Other times, one has to rely on non-parametric models, being harder still to predict. As such, the existence of shape optimization as its own sub-branch of the optimization field should be no surprise.

Shape optimization is, as the name implies, the optimization of shapes. One of the more common uses is to alter shapes to be more aerodynamic. It is easy to simulate fluid dynamics, and thereby evaluate a shape, but hard to "solve" as to get an optimal shape from the get-go, making it an area well suited to optimization algorithms.

The use of shape optimization in this area is both common enough and sufficiently efficient that entire textbooks has been written on the subject, such as *Applied Shape Optimization for Fluids* (Mohammadi & Pironneau, 2009) and *Conceptual Shape Optimization of Entry Vehicles Applied to Capsules and Winged Fuselage Vehicles* (Dirkx, Mooij, & SpringerLink, 2017).

That is not to say that research in this area has stopped. *CFD-Based Shape Optimization for Optimal Aerodynamic Design* (Gabbasa, Jawad, & Koutsavdis, 2012), for example,

investigates the best way of utilizing this field. Using a commercially available tool (Fluent) based on a polynomial mesh-morphing algorithm, the study investigates how to best use said tool to influence and/or alter aerodynamic properties, in this case focusing on the lift-to-drag ratio in a simple test case.

In other cases, the shapes used might be more regular, allowing for parameterized models. The one studied in *Particle swarm optimization (PSO) for reflector antenna shaping* (Gies & Rahmat-Samii, 2004) is one such case. Reflector antennas do not differ that much from one another, with the exception of a few discrete variations in, for example, the general shape of the reflector dish (parabolic, flat, cylindrical etc.). As such, they are very well suited for a parameterized model.

In *Particle swarm optimization (PSO) for reflector antenna shaping*, Gies and Rahmat-Samii creates tools to use the named optimization method (see 3.2.3.5) to, as a proof-of-concept, optimize an antenna shape for better scanning performance. The results show that particle swarm optimization does work in this case, and there is an implication that this could be generalized for other reflector antenna properties.

Another usage is to alter existing objects to allow/alter a specified property. As an example, we have *Stackabilization* (Li, Alhashim, Zhang, Shamir, & Cohen-Or, 2012) and *Foldabilizing Furniture* (Li, Hu, Alhashim, & Zhang, 2015).

Stackabilization (Li, Alhashim, Zhang, Shamir, & Cohen-Or, 2012) investigates how to alter a 3D object to make it easier to stack copies of said object. The stackability score used for energy minimization is based on the space in the stacking direction between two stacked objects, and the goal is to minimize this space while also keeping the alterations as small as possible.

The goal of *Foldabilizing Furniture* (Li, Hu, Alhashim, & Zhang, 2015) is to turn a 3D object, presumed to be a piece of furniture, into a foldable version of said object. The goal is to minimize the volume of the folded object, whilst making as few alterations as possible. The study focuses on a specific case, where there is a pre-defined folding direction and the only allowed alterations are hinge insertions and part shrinking.

The main problem in *Foldabilizing Furniture* is similar to the one in *Stackabilization*, but with a lot of constraints not present in the latter. The folding process needs to be modelled, both to calculate the volume of the folded object and to make sure that hinge placement allows for folding the object without collisions.

Furthermore, the way the studies approach their problems are rather different. In *Stackabilization*, the problem is solved by first creating slight variations on the current shape, defining those as neighbouring state, and searching through the solution space with a “best first” algorithm. *Foldabilizing Furniture*, on the other hand, decomposes their input shapes into scaffolds and “hinge graphs”, using graph theory to find the best modifications that avoids collisions.

(2.2.3) Optimization in healthcare

While the utility of using optimization and computer science in healthcare in general has not gone unnoticed, it is not always trivial to combine them. In *Interdisciplinary partnerships between rehabilitation therapists and computer scientists: a proposed model* (Boisselle & Simmonds, 2014), the authors investigates how to integrate the two.

The proposed model mentioned in the title is a prototype method/framework for how to successfully set up a partnership between computer scientists and rehabilitation therapists.

That being said, the use of shape optimization to adapt physical aids for people with disabilities is somewhat rarer, but there are some studies and projects that come close.

Optimization of the Design of a Discus for People with Disabilities (Seo, Takahashi, Kawabata, & Mitsui, 2016), as the name implies, is about customizing the shape of a discus for athletes with physical disabilities. Using a parameterized model and a few different throwing motions, Seo et al. concluded that the optimal shape was a thicker and less wide discus than the standard.

If we forego the criteria of the disability aid being a physical thing, the SUPPLE project (Gajos, o.a., 2010) has an extremely similar goal to ours.

SUPPLE is a project developing a method for creating custom user interfaces to suit the needs of the user(s). More specifically, this allows the UI to compensate for disabilities, such as having larger buttons and fewer instances requiring precision for people with poor motor control or having larger text for people with poor eyesight.

The project has three main parts; SUPPLE, ARNAULD and the ABILITY MODELER. The ABILITY MODELER is fairly self-explanatory; it models the abilities of the user. ARNAULD allows the user to adapt the optimization processes to their liking. And last, but not least, SUPPLE itself contains the actual optimization process.

(3) Theory

Since determining the specifics of the optimization problem is part of the solution, rather than something known at the start, there is very little theory behind the problem that is relevant to the rest if the thesis.

As such, this section will focus on the theory behind the chosen formulae, representations and methods. In the case of the optimization methods, those that were considered will also be briefly explained, together with the reasons they were not chosen.

(3.1) Terminology

This thesis will use the following notation throughout:

X: Set of parameters representing a spoon. A subscript will, unless otherwise stated, stand for the iteration number while a superscript will denote its place in a set. Special properties, such as being the best set found by the method, will also be noted in the superscript.

S is the solution space. If X has n parameters, then $S \subset \mathbb{R}^n$.

f is the objective function for the optimization. $f: S \rightarrow \mathbb{R}$

δ and ω are scaling constants.

ε is a very small scalar, used to check when the optimization algorithms can stop iterating. $X \sim U(S)$ is a set of parameters sampled from a uniform distribution over the entire solution space.

R denotes a rotation matrix

γ, θ and φ are rotation angles. γ is the rotation around the z-axis, θ the y-axis and φ the x-axis. More on this in section 3.3.3.

(3.2) Optimization

(3.2.1) General optimization theory

Optimization, as used in this project, is the process of finding the best solution out of all possible solutions.

As such, it is often used for problems where it is easy to construct a solution and test its quality, but difficult to find the best one due to the large number of possible solutions. For example, finding the point in an N-dimensional scalar function with the lowest value: it is easy to choose an arbitrary point, but difficult to find the best one unless the generating function is known.

The suitability of a solution is determined by an objective function. The goal of any optimization process is to find the solution that minimizes the value of the objective function. To find the highest value in a scalar field, the objective function would be the inverse of the scalar value, i.e., multiplied with -1.

Another way of expressing it is that the objective function maps the variables to a scalar field, and the goal of the optimization process is to find the lowest scalar value in the field.

The basic pattern for an optimization process is as follows:

1. Find a feasible solution. That is to say, choose any possible solution that solves the problem.
   $$X_0 \in S$$
   In the case of the scalar field above, it is any point in the field that fulfils the conditions.

2. Use the found solution to somehow get a better feasible solution.
   $$g: S \to S \text{ such that}$$
   $$\text{if } X_{i+1} = g(X_i) \text{ then } f(X_{i+1}) < f(X_i)$$

3. Repeat step 2 until some criteria is met, usually when the solution is deemed good enough or the process has gone through a set number of iterations.
   $$X_{i+1} = g(X_i)$$
   $$\text{Stop if } f(X_i) < \varepsilon$$
   $$\text{or if } i > N$$

As such, optimization is solved through some manner of iterative methods, be they convergent, heuristic-based or even algorithms.

Of course, this process only works if there is an optimal solution to find. It is also possible that the objective function can approach negative infinity, meaning that there is no “best” solution, or that there is an infinite number of equally good solutions.

Determining if a problem is possible to solve can be surprisingly difficult. Should the problem be convex, it is a non-issue; the problem will have a single minimum point, making said point the optimum.

Otherwise, certain types of optimization problems can be solved, as mentioned above, through algorithms. In those cases, the algorithms have conditions on when a problem, otherwise appropriate for said algorithm, cannot be solved for any reason.

In the case of other iterative methods, however, it can be much harder. For continuous functions operating on a compact set, the extreme value theorem proves that there exists a point for which the function attains its lowest possible value. Unfortunately, it says nothing of the uniqueness of said point, or if a found point is the global minimum, or merely a local minimum.

(3.2.2) Gradient descent

As the name implies, gradient descent is the optimization method based on calculating the gradient of the objective function with respect to the variables, and then adjusting the solution based on that.

The basic process is as follows:

1. Create a set of variables.

2. Calculate the gradient for this set.

3. Take a step in the direction of the gradient.

4. Repeat steps 2 and 3 as necessary until the gradient approaches zero.

Or, in mathematical terms:

$$X_{i+1} = X_i + \delta \cdot \nabla f(X_i)$$
$$X_{best} = X \text{ if } (\nabla f(X))^2 \leq \varepsilon$$

Where $\delta \in [0,1]$ and ε is most typically 10^{-n} where $n \geq 5$.

The basic gradient descent function does not take any form of constraints into account. It assumes that adjusting the variables according to the gradient will always be an allowed action. While it still works tolerably well on convex sets, albeit with a risk of odd behaviour and/or getting stuck on the boundary, anything more involved would require a projection onto the constraints.

Gradient descent also works best when the objective function is convex. In that case, the function has no local minima or saddle points, meaning that when the optimization process finds a minimum point, it will be the global minimum.

When the objective function is not convex, the Gradient Descent algorithm runs a risk of getting stuck in local minima. One way of rectifying this is to check a number of surrounding points whenever a minimum point is encountered. Should any of the surrounding points have a lower score than the found minimum, the optimization process will continue from there.

Another possible problem stems from the step length. If the step length is too small, the process of finding an optimum will be very slow. On the other hand, if the step length is too

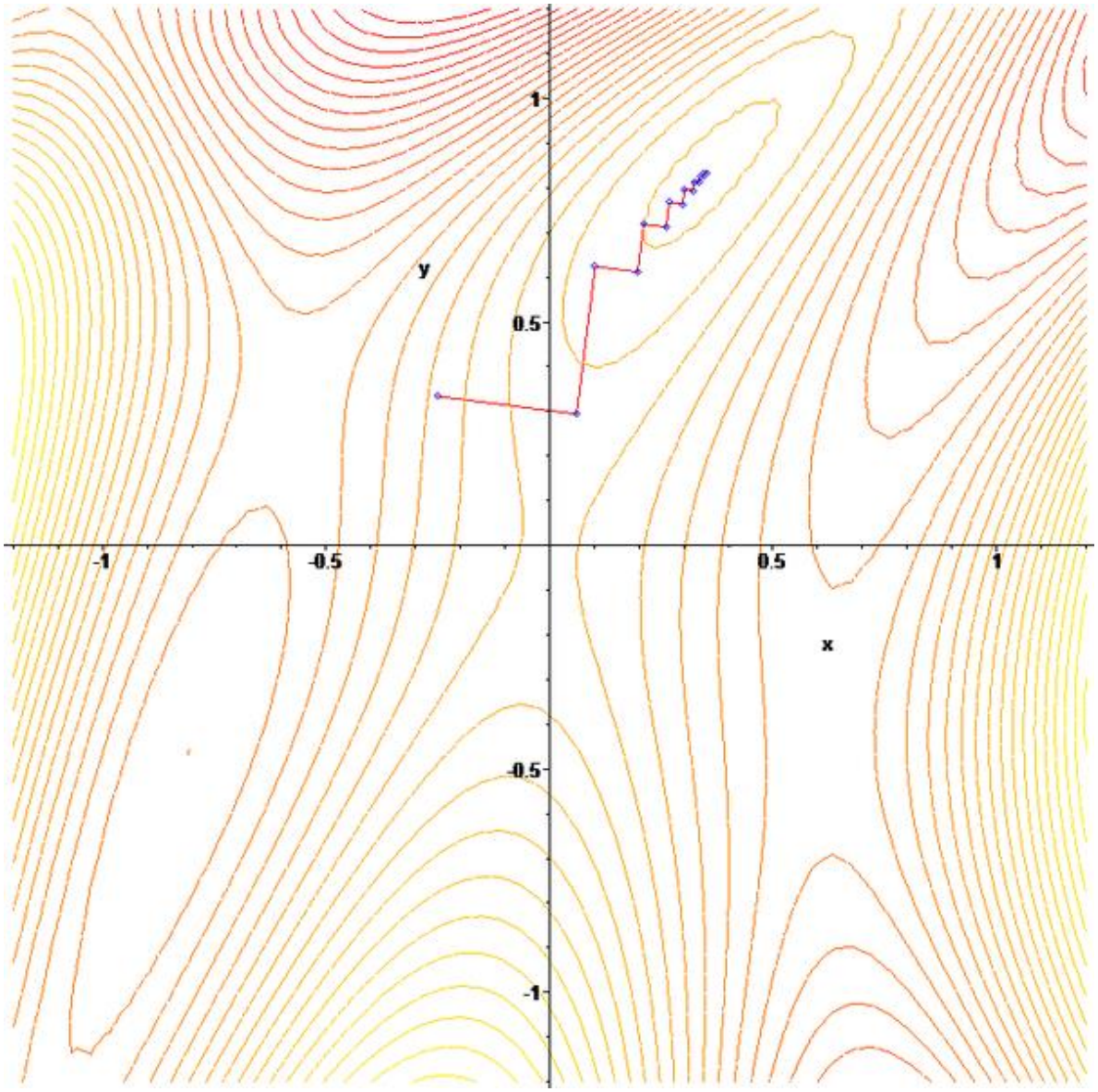

Figure 1: Example of gradient descent. Note the overshoot, resulting in a zig-zaging motion. Figure source: public domain picture from Wikipedia, https://en.wikipedia.org/wiki/File:Gradient_ascent_(contour).png, accessed 08-08-2017, created by Wikipedia user Jor

large the process is liable to overshoot, either circling the minimum or zig-zagging when it would be more prudent to travel through the middle.

One way to solve the step length problem is to do a linear search along the direction specified by the gradient, and then choose the step-length that gives the best results. While this does give good results, it does take up a bit more processing power. Especially in cases like this project, when every possible result requires quite a few calculations.

A related way of solving it is to start out with a fairly large step-length. If using that step-length gives a result that is equal to or worse than the previous one, the step-length is halved. If it still is not better than the previous one, it is halved again. This is repeated until the score is better than the previous step, or as a fail-safe until a specific number of iterations have been reached.

Finally, the problem of overshooting might also be solved by adding a factor that is somewhat resistant to directional changes, like momentum for a kinetic particle, when applying the gradient. In that case, the zig-zagging would counter-act itself, since the new gradient and the momentum would partially cancel out (in the overshoot direction) and partially reinforce each other (in the correct direction).

The momentum part might also help with the process getting stuck in a local minimum; should the forward momentum be strong enough, it might be enough to overcome the pull of the local minimum until it descends towards a stronger minimum.

(3.2.3) Other methods

While this project mainly uses gradient descent, there are other methods that could be used instead.

(3.2.3.1) Exhaustive search

Exhaustive search is more or less the brute force approach to optimization. As its name implies, exhaustive search consists of evaluating every single possible solution, and choosing the best one. For methods with a continuous solution space, and thereby an infinite number of solutions, some manner of discretizing the solution space is applied.

In mathematical terms:

$$X^{best} = \min_{X \in S} f(X)$$

Exhaustive search is guaranteed to find the best possible solution present in the searched solution space, but since it takes an extraordinary amount of time to do so for large problems it is seldom used in practice.

As mentioned later in this report, exhaustive search was deemed inconvenient for this project. However, an adaptation of exhaustive search, where the discretization of the solution space consisted of choosing N random solutions and letting those represent the entire space, was used under the name of “Best of random choice.”

(3.2.3.2) Simulated annealing

Simulated annealing is a probabilistic method inspired by the thermodynamic process behind metallurgical annealing. Annealing involves using the diffusion of atoms during controlled heating and/or cooling to influence the properties of an object.

Simulated annealing is at its most basic similar to many other heuristics; it uses a candidate solution and evaluates its neighbours, switching to an evaluated neighbour if it is better. However, simulated annealing sometimes switches to a worse candidate solution, as to allow for a more thorough exploration of the solution space.

The annealing part comes into play through how solutions are accepted and discarded. The method keeps track of a “temperature” value that starts high and sinks as the method iterates.

$$T_i = T(i, M) : \begin{cases} T_0 > 0 \\ T_i \geq T_{i+1} \\ \lim_{i \to M} T_i = 0 \end{cases}$$

Where T is the “temperature” function and M is the maximum number of iterations. The higher the temperature, the higher the chance of accepting a worse solution.

This probability should also be proportional to how much worse the new solution is.

$$P(f(X_a), f(X_b), T) \in [0,1]$$
$$P(f(X_a), f(X_b), T_i) \geq P(f(X_a), f(X_b), T_{i+1})$$
$$\text{if } f(X_a) > f(X_b) > f(X_c)$$
$$\text{then } P(f(X_b), f(X_a), T) = 1,$$
$$\text{and } P(f(X_a), f(X_b), T) > P(f(X_a), f(X_c), T)$$

Where P is the probability of accepting a worse solution.

This method is mainly changed and adapted through the acceptance probabilities of solutions, and through the annealing scheme used to determine how much the temperature lowers with each iteration. Given the above functions, the algorithm itself is very simple:

$$X_n = \text{Neighbour}(X_i)$$
$$X_{i+1} = \begin{cases} X_n, & P(f(X_i), f(X_n), T_i) \geq r, \quad r \in U(0,1) \\ X_i \text{ otherwise} \end{cases}$$

Simulated annealing is fairly well suited for projects like this, provided a way of determining what a neighbouring state is. Code was written for an implementation, but due to some hard-to-track errors, presumably created by a faulty neighbouring-states interpretation, it was abandoned as too time consuming when other optimization methods had already been implemented successfully.

(3.2.3.3) Nelder-Mead method

The Nelder-Mead method is based on creating a simplex in the solution space, and then manipulating its size and position through replacing the vertex with the worst score.

Each iteration step, the vertices are ordered according to score, and the centroid is calculated. Then, points are evaluated along the line defined by the centroid and the worst vertex. First test is a point at the other side of the centroid from the worst vertex, usually at the same distance as the worst vertex.

If this new point is comparable to the other vertices, the worst vertex is replaced and a new iteration starts.

If it is better than the best point, a new one is evaluated further out along the same line, and the worst vertex is replaced by the best of these new points.

If the new point is worse than the second-worst vertex, a point between the worst vertex and the centroid is evaluated, usually at half the distance between them but at times closer to the centroid. If this point is better than the worst vertex, it is replaced. If it is worse, all vertices except the best one are updated to be closer to the centroid.

As a method meant to operate on an N-dimensional space, the Nelder-Mead method would be well suited for this project. Due to not needing a gradient to be calculated, it might even be faster than gradient descent. However, since this project focuses on finding a way rather than the best possible way of solving the problem, gradient descent was chosen for familiarity reasons; using an unfamiliar method would make it more difficult for the programmer.

(3.2.3.4) Genetic algorithm

Genetic algorithm is a method inspired by evolution. A "population" of candidate solutions are created and each iteration the best candidate solutions, the most fit solutions, have their "genes" recombined to form a new population of candidate solutions.

In order for this method to work, a genetic representation of the solution domain is needed. As the solution space for this project is continuous rather than discrete, let alone a series of binary features, this optimization method was deemed too inconvenient for this task.

(3.2.3.5) Particle swarm

Particle swarm is a method of exploring the solution space through, as the name implies, a swarm of particle.

Each particle consists of a position, velocity and a record of the best-scoring position that particle has visited.

$$P^i = \left[X^i, \boldsymbol{v}^i, X^i_{best}\right]$$

Where X^i is the particle with index i, X is a position in the solution space, and X_{best} is the position where the particle has found its best score so far. The best position found by any particle is also stored separately, and all particles have access to this value.

$$X^{swarm}_{best} = \min_i f(X^i_{best})$$

The process is initialized with a large number of particles with uniformly distributed random positions and velocities. Every iteration, the particle positions are updated with their respective velocities, and the velocities are updated to a weighted sum between the particles' current velocity, a vector pointing towards that particles' best position and a vector pointing towards the swarms' best position. All terms are appropriately scaled, and the last two have a random component. In other words:

$$\begin{aligned}
&r_p, r_s \sim U(0,1)\\
&\boldsymbol{v}^i = \delta \boldsymbol{v}^i + \omega_p r_p\left(X^i_{best} - X^i\right) + \omega_s r_s\left(X^{swarm}_{best} - X^i\right)\\
&X^i = X^i + \boldsymbol{v}^i\\
&f\left(X^i\right) < f\left(X^i_{best}\right) \rightarrow X^i_{best} = X^i\\
&f\left(X^i\right) < f\left(X^{swarm}_{best}\right) \rightarrow X^{swarm}_{best} = X^i
\end{aligned}$$

Where δ, ω_p and ω_s are user-defined scaling constants.

This results in a particle swarm that explores the solution space in an initially random pattern, but with particles gradually converging should the swarms' best known position remain in the same area for long enough.

While the particle swarm method does not require calculating a gradient, it does require a lot of particles to cover the entire solution domain. As such, it is hard to tell if it would suit this project better or worse than gradient descent. In the end, this method was not chosen due to the programmer being much more familiar with gradient descent.

(3.3) Geometric transformations

To borrow a definition, "A geometric transformation is a function whose domain and range are sets of points. Most often the domain and range of a geometric transformation are either both R^2 or both R^3. Often geometric transformations are required to be 1-1 functions, so that they have inverses." (Usiskin, Peressini, & Marchisotto, 2003)

For the purposes of this project, a geometric transformation is an operation that moves, rotates or similarly alters an object (points, vectors and entire spoons). Only translations and rotations are used, as those are the only parts needed for either creating the spoon, adjusting the position of the spoon with respect to the origin or modelling the motion.

Geometric transformations are commonly represented through matrix operations. For example, a 90° rotation around the z-axis would be written:

$$R\boldsymbol{x} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -y \\ x \\ z \end{pmatrix}$$

While matrix notation does require the transformations to be construed as linear transformations, most are linear to begin with. Those that are not do require some extra measures for the approximation to work.

(3.3.1) Homogenous coordinates

Translation is one of the transformations that need special measures. Normally, matrix transformations assume that there is a fixed point, and that all other coordinates are altered in reference to said point. This works fine for rotation and scaling, but the idea behind translation is that all points of the translated object are altered equally, no matter where they are in reference to each other.

Homogenous coordinates is a way of solving this. In essence, an additional dimension is added to the coordinate system, allowing the actual coordinates to be altered with respect to the extra coordinate. For example, translating the example from the previous section would require two separate operations with normal coordinates:

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} + R\boldsymbol{x} = \begin{pmatrix} a - y \\ b + x \\ c + z \end{pmatrix}$$

But with homogenous coordinates, only one operation is needed:

$$\begin{pmatrix} a \\ b \\ c \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} a - y \\ b + x \\ c + z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & -1 & 0 & a \\ 1 & 0 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} a - y \\ b + x \\ c + z \\ 1 \end{pmatrix}$$

The actual value of the extra coordinate is mainly relevant for projective transformations. In those cases, the vectors are "normalized" by dividing all coordinate values by the last value, ensuring that it ends with a 1. However, this is not relevant for this thesis. The last coordinate value for any given vector should not be altered by any algorithm.

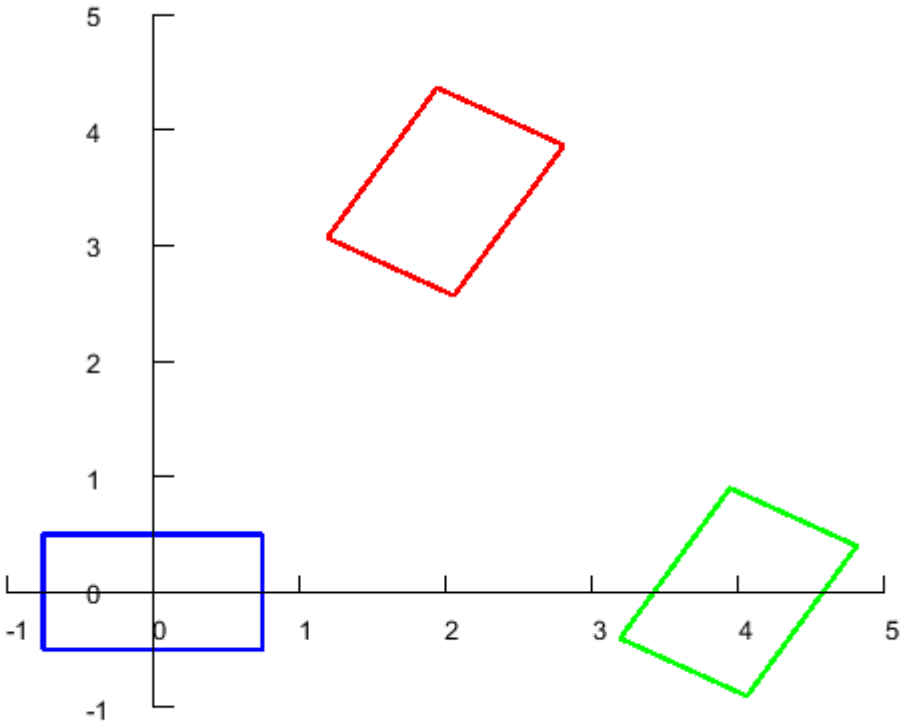

Figure 2: green is rotation before translation, red is translation before rotation

(3.3.2) Order of operations

Matrix operations are not generally commutative, and our case is not an exception. Rotations are performed around a fixed point, assumed to be the origin. Rotation around some other point would either require translating the object so that the correct point winds up at the origin, or reformulating the coordinate system.

As such, translating the object before a rotation is rather different from translating it afterwards.

Given how it can affect the end results, the matrix transformations used in this thesis are all applied in the following order: scaling first, followed by rotation and translations applied last. Transformations relative to a local frame of reference are also applied before the transformation to a global reference frame, followed by any transformations relative to said global frame.

(3.3.3) Tait-Bryan angles

One of the two main types of conventions for Euler angles, Tait-Bryan angles are a way of describing the orientation of an object or a vector in a 3D-space. They differ from the other group, Proper Euler angles, in what axes the rotations are made around.

Any orientation can be reproduced through three rotations around the axes of the coordinate system. With Proper Euler angles, the first and the last angle are around the same axis, before and after the middle rotation takes place. Tait-Bryan angles, on the other hand, match one angle to each axis.

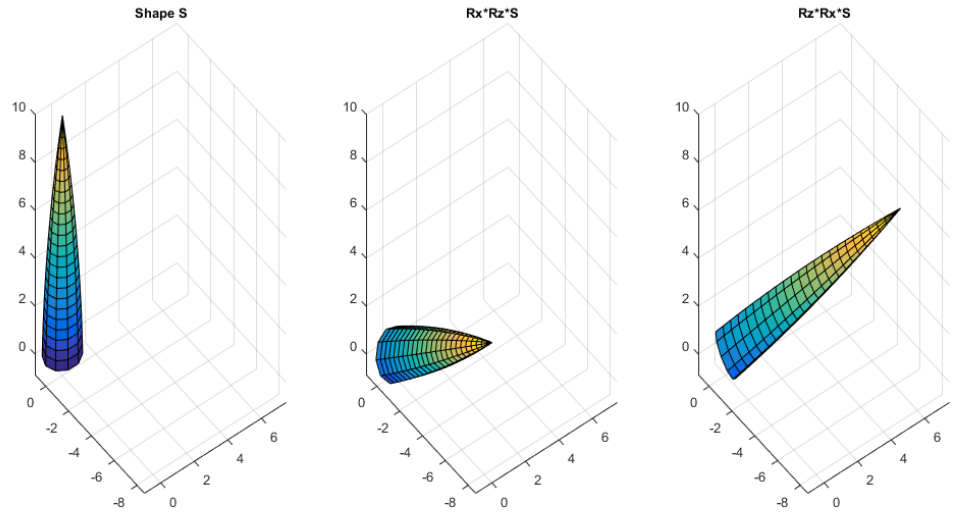

Figure 3: From left to right: untransformed shape, Rotation around z axis before rotation around x axis, and rotation around x axis before rotation around z axis

Both of these groups have six possible variants, based on what axes are used, and in what order. Each of those variants can also be applied as either extrinsic (all rotations are according to the original axes) or intrinsic (rotation happens around axes that have been rotated along the object).

The order of the rotations is important, due to the fact that unless specifically compensated for, each new applied rotation turns the previous ones into intrinsic rotations. In other words, each new rotation is applied to the previous rotations as well as the object.

The order used in this project is referred to as z-y'-x'', meaning rotation around the original z-axis, along a rotated y-axis and a twice-rotated x-axis.

$$R(\gamma, \theta, \varphi) = R_x(\varphi)\, R_y(\theta)\, R_z(\gamma)$$

Where γ, θ and φ are the rotation angles around the z-, y- and x-axis respectively. This configuration is often used in aeronautics, and is also known as yaw-pitch-roll. This notation will for the rest of this thesis be referred to as RPY-angles.

(3.4) Triangle meshes

Polygonal meshes, of which triangle meshes are a subset, are a way of mathematically defining shapes. More specifically, complex shapes that are not easily reduced to simple shapes with known generating functions, such as spheres or cubes.

A polygonal mesh consists of vertices, edges and surface elements. The vertices are points on the surface of the shape, the edges define what vertices are connected with each other, and the surface elements are the polygons defined by the edges.

In a triangle mesh, all of the surface polygons are triangles, meaning that the vertices are connected in groups of three, where one vertex can belong to several different groups. It is one of the more common mesh variants, due to methods taking advantage of the vertex groupings.

There are a number of possible mesh representations to use. All of them keep a list of what vertices exists, and the positions of the same, but the way the polygon information is stored differs. This project uses a face-vertex representation, meaning that the surface elements are represented by the vertices in its corners.

Figure 4: Start and end position of a spoon with a full fist grip.

(4) Method

(4.1) Overview

The first step in solving the problem is to create concrete steps for doing so – steps that can be solved, unlike the initial vague problem statement. In this case, the steps can be labelled input, processing and output;

1. **Input: importing the movement.**
   While recording how the patients move when eating is, strictly speaking, beyond the scope of this thesis, the optimization process does need something to adapt the utensils to. As such, the program runs on the assumption that a representation of the movements are available on a file, and said data is then imported and converted to the format used in the program.

2. **Processing: adapting the spoon to the motion.**
   This is the main part of the program. To do this, the program needs both an optimization algorithm and something for said algorithm to optimize. In other words, a way to model the spoons is needed, with a good degree of flexibility for different spoon shapes, as well as a way to optimize said model to the imported movements.

3. **Output: converting the spoon model to a mesh.**
   The problem formulation does specify that the adapted spoons should be in the form of meshes. While it would have been possible to skip the spoon model by optimizing the mesh directly, implementing a way of making sure that the results were spoon-shaped was estimated to require more work than it was worth. As such, the program needs a way to convert its own spoon model to a mesh format, before exporting it to file.

This section will detail how these three steps are implemented.

(4.2) Spoon modeling

How to represent that which is being modelled is one of the most fundamental decisions to make in a project, as it influences every single operation made on the representations. In the case of optimization tasks, such as this project, the representation can be tailor-made to a specific optimization method, albeit with the drawback of making other methods slightly more difficult to implement.

As this project does not have a set optimization method, the chosen representation is fairly general.

(4.2.1) Representation goal

The spoon representation should be able to recreate a large number of commercially available specialty spoons while still being as simple as possible. It should also be able to be converted into a form that can be interpreted, and used, by a 3D-printer.

(4.2.1.1) Determining the desired capabilities

To determine what the spoon representation should be able to model, one needs to analyse what kinds of disabilities impair the usage of standard spoons. To visualize this, we show a series of pictures illustrating simplified conditions that would require the use of different specialty spoons.

The first of these examples are based on not being able to move the hand and/or wrist with respect to the arm. To begin with, just to have as a basis for comparisons, a case where an ordinary spoon would work.

While not being able to adjust the spoon without moving the entire arm is inconvenient, having the hand frozen in this position does not impair the use of an ordinary spoon significantly. Should the hand instead be twisted 90°, as if to give a “thumbs up” gesture, a normal spoon would not work. The spoon would need a bowl that is angled with respect to the handle, much like a ladle.

Figure 5: By tilting the bowl upwards with reference to the handle, spillage is minimized

Figure 6: While getting the bowl close enough to the mouth is still difficult when one is not able to move the wrist, having the spoon bowl turned towards you (right) is helpful.

For our third example, the hand is tilted to the side, as if to point with the thumb. While it still allows regular spoons to not spill their contents, unlike the previous example, it is much harder to actually bring the contents to the mouth. The angling of the hand needs to be compensated for with an angling of the spoon bowl.

As we can see, the bowl needs to be able to be angled in any direction with respect to the handle. However, there are other types of disabilities that cannot be solved as easily. For example, if the hand itself is irregularly shaped, by muscles seizing up or otherwise, then a straight spoon handle will not be sufficient. The handle needs to conform to the hand.

As seen in figure 7, the handle should be allowed to bend in several places. It should also allow for variable thickness, as there is no guarantee that the disability would allow for adjusting the hand enough to grip a too-thin handle.

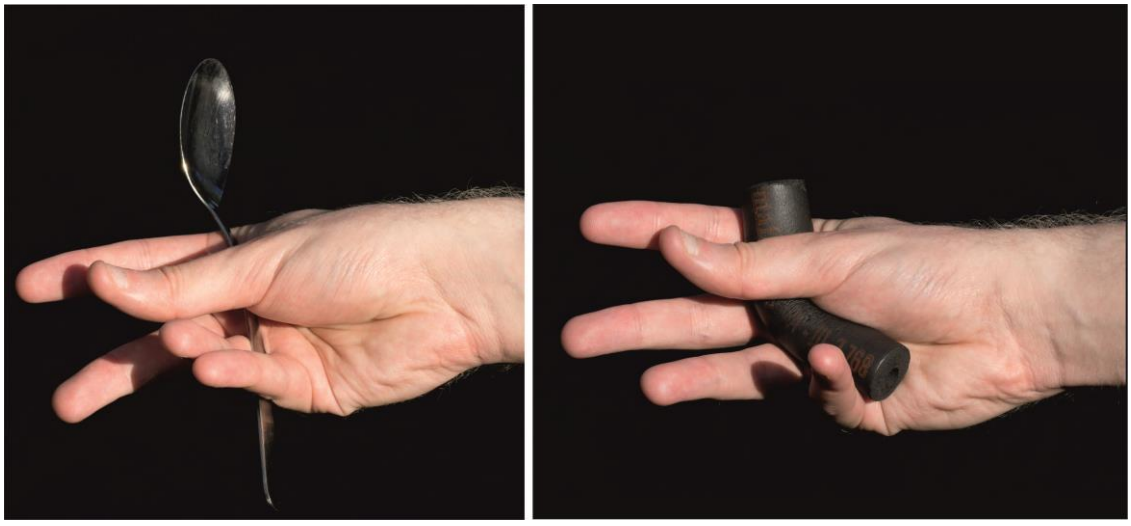

Figure 7: When the hand is unable to properly close around a shape, adjusting the shape can help with the grip

From this, the following criteria for alterable spoon characteristics were chosen for the spoon representation:

- For the bowl shape
  - Dimensions
    - The representation must be able to model several different bowl shapes and sizes
  - Angle of connection
    - The representation must be able to model bowls connecting to the handle at different angles.
- For the handle
  - Length
    - The representation must be able to model handles of different lengths
    - The length of the handle is defined as the length of the curve defining the middle of the cross section of the handle
  - Thickness
    - The representation must be able to model different handle thicknesses
    - Thickness is here represented as the size of the cross section
  - Bends
    - The representation must be able to model the handle bending in different directions
    - A bend is defined as a length of the handle where the tangent of the handle changes direction
  - Properties altering along the handle
    - The representation must be able to model a handle with more than one value of the above two properties along its length
      - The handle should be able to have different thicknesses at different places along the handle
      - The handle should be able to bend in more than one place

(4.2.2) The chosen representation

In this project, the spoon shapes are represented as a series of parameters, a number of continuous variables. The number of parameters can vary between spoons, for reasons that will be discussed shortly, and the code has been written to take this into account.

The spoon bowls are modelled as ellipsoid cuts. As such, the shape of the bowl is determined by the radii of the ellipsoid, as well as the height of the cut, measured from the lowest point (lowest z-coordinate) on the ellipsoid. All four of these are measured in millimetres. The angle of connection between bowl and handle is represented as RPY-angles, measured in radians. The bowl is thus represented by seven parameters altogether.

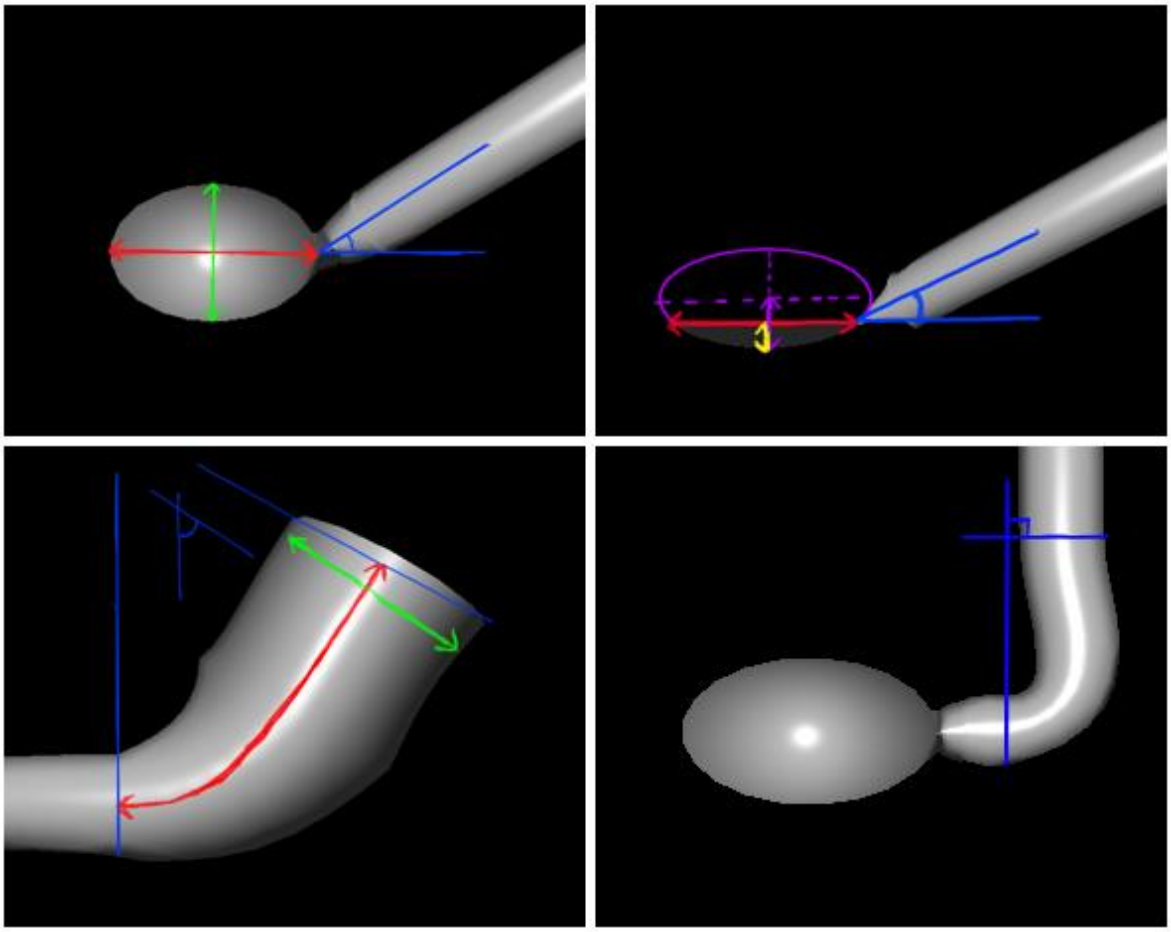

Figure 8: red denotes length, green width, blue angle differences, yellow the bowl height and purple the height of the original ellipsoid

In order to model the handle having more than one bend, the handle is divided into segments. Each segment is able to model one bend and one alteration of the handle thickness.

Each segment has parameters that describe the shape of the handle at the end-point of the segment. The handle length is represented through a parameter describing the length of the current segment in millimetres. The thickness is represented through two parameters, the radii of the cross-section ellipsis at the end of the segment, also in millimetres. The bend, or absence of the same, is represented through RPY-angles describing how the direction of the handle tangent changes from the beginning to the end of the segment. These angles are given in radians.

As using too few segments restricts the number of possible handles it can model, one does not want the number of segments to be too low. At the same time, having too many segments leads to a much higher strain on the optimization process, and can lead to a lack of generality. Determining a balance between these two concerns can be a complicated process, and is not in the scope of this project. As such, the number of handle segments has been left as a user-defined variable.

All in all, each spoon is represented by $7 + 6n$ parameters. For most of the testing, n has been set to either 4, meaning 31 parameters, or 6, meaning 43 parameters.

(4.2.2.1) Simplifications

As mentioned above, the bowl is modelled as an ellipsoid cut. While it is mostly the right shape, it is not completely able to re-create all possible spoon bowls out there.

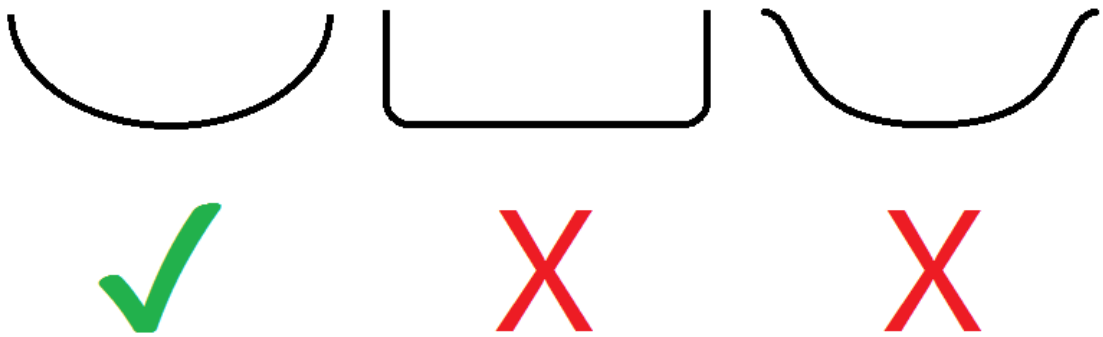

Figure 9: The used model only allows for bowls that are ellipsoid cuts, i.e. the cross section of the bowl must be an ellipsis segment

For one, the ellipsoid cut approximation makes it a requirement that the deepest point of the bowl is in the middle with respect to the rim ellipsis. As another example in the same vein, a real spoon bowl could theoretically be asymmetrical, with a rim that would require four different radii to describe, instead of the two of the current model.

There are also several bowl shape adjustments made impossible by the approximation, figure 9 shows some examples.

The benefits of this approximation does, however, outweigh its flaws. Most of the above concerns would require a shape that could not be described easily with a general differentiable function. As such, calculating the volume would require going through all the bowl vertices of the spoon mesh, which in turn demands a mesh to be constructed for every single parameter set that gets evaluated.

Because of the way the in-built methods handles meshes in the software used (Inviwo), merely finding the correct vertices of the spoon would be a chore, as there is no in-built way of really telling them apart from the ones in the handle. Even if one were to use a mesh consisting only of the bowl for the calculations, actually finding what vertices belongs to the rim would need to go through them all.

As such, the ellipsoid cut representation makes up for its flaws by being considerably faster. Given that the flaws are not that serious, it is a more than acceptable trade-off.

(4.2.3) Movement representation

The movement of the spoon is represented as a series of transformation matrices. More specifically, it is represented as a series of rotation matrices. This is due to the fact that rotation alters how much the spoon could hold at that moment, whilst translation affects the physics of the contents.

For reasons discussed in the “Simplifications in the representation” part of this chapter, this project disregards the physics part of the process. As such, adding a translation part to the transformation matrices is an unnecessary complication.

The transformation assumes that the initial origin is supposed to be in the middle of where the hand holds the handle, with its cross-section being a plane with its normal along the x axis. As such, the movement matrices contain information about both how the spoon moves and how it is held.

(4.2.4) Volume calculation

The volume contained in the spoon bowl at any given step of the motion is calculated as the volume of the smaller part of an ellipsoid cut by a plane.

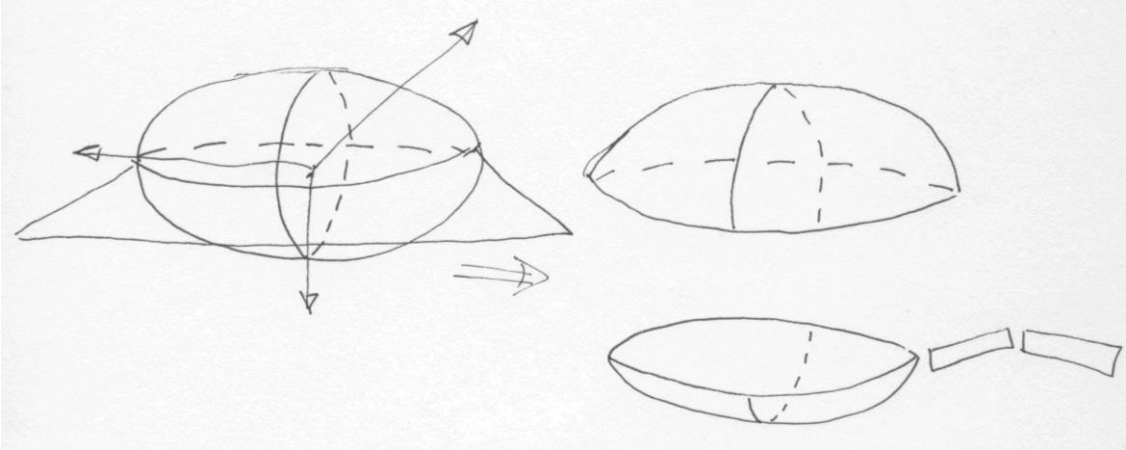

Figure 10: The volume below the cutting plane is repurposed to serve as the spoon bowl

However, calculating the volume of an ellipsoid cut by an arbitrary plane can be difficult as well. To solve this, a transformation is applied to simplify the matter. If an ellipsoid defined by

$$\frac{x^2}{r_x^2}+\frac{y^2}{r_y^2}+\frac{z^2}{r_z^2}=1$$

Is cut by a plane defined by

$$Ax+By+Cz=D$$

Then a simple change of variables to

$$x=r_x u, \qquad y=r_y v, \qquad z=r_z w$$

Turns it into a sphere

$$\frac{(r_x u)^2}{r_x^2}+\frac{\left(r_y v\right)^2}{r_y^2}+\frac{(r_z w)^2}{r_z^2}=u^2+v^2+w^2=1$$

Cut by a slightly different plane

$$(Ar_x)u+\left(Br_y\right)v+(Cr_z)w=D$$

The volume of a spheroid cut is fairly simple to calculate through an integral, and since the Jacobian determinant is as simple as

$$\left|\frac{d(x,y,z)}{d(u,v,w)}\right| = \begin{vmatrix} r_x & 0 & 0 \\ 0 & r_y & 0 \\ 0 & 0 & r_z \end{vmatrix} = r_x r_y r_z$$

The volume of the ellipsoid cut can be obtained through multiplying the volume of the spheroid cut by $r_x r_y r_z$.

This reasoning is further re-used, since cutting an ellipsoid cut by a plane gives the same results as cutting the original ellipsoid by the same plane. It merely becomes a question of how to define the plane.

Because of the assumption that the orientation of the spoon bowl is the only thing capable of affecting the bowl contents (i.e. no momentum, surface tension etc), the bowl contents will always have a surface that is parallel to the XY-plane. The height of this surface is limited by the lowest point on the bowl rim.

As such, the plane cutting the ellipsoid is defined by applying the reverse transform for the rotated bowl to the plane that pass through the lowest point on the rim whilst parallel to the XY-plane.

Said lowest point can easily be found as the rim of the bowl is an ellipsis, and any point on it can thus be described by a differentiable function dependent on the angle from the handle connection point:

$$\boldsymbol{p}_{\text{rim}}(\theta) = R\ (r_{\text{length}}(1 - \cos\theta) \quad r_{\text{width}} \sin\theta \quad 0)$$

Where R is the transformation matrix, and the origin before the transformation is placed at the point where the bowl connects to the handle. The height is along the z-axis, giving the following:

$$z_{\text{rim}}(\theta) = R_{3,1} r_{\text{length}}(1 - \cos\theta) + R_{3,2} r_{\text{width}} \sin\theta$$

From the values of θ where the derivative of the height function equals zero, we get two possible points. Comparing the z-coordinate of those two gives the correct angle, which can be used to calculate the lowest rim-point.

As mentioned above, the maximal possible volume at that step in the movement is limited by the plane $z = z_{\text{rim}}(\theta_{\text{lowest}})$ cutting the ellipsoid defined by the bowl parameters and rotated by R. This is also the same volume defined by cutting the initial ellipsoid and rotating the above cutting plane by R^{-1}.

(4.2.4.1) Simplifications

The volume calculation assumes that the spoon moves slowly enough that its contents can be considered at rest for the entire movement. This is often not the case, especially for those with physical disabilities. The current model does not take concept such as momentum, viscosity of the contents (for example, water or soup or pudding), fluid dynamics or surface tension.

However, all of these concepts require a lot of processing power to take into account, and some would doubtlessly require greater precision in the motion recording than the tools used in this project could provide. Not to mention it would take a lot of time and effort to implement, more than allotted for this thesis project.

As such, it would mostly serve as a time-sink with dubious value, for the same reasons why it only tries to optimize spoons. The goal of this thesis is to prove it possible to optimize spoons by finding a way of doing so.

(4.3) Mesh creation

The cross section of an ellipsoid is an ellipsis, no matter how the cut is aligned. The handle also has an ellipsis for its cross section, since it needs to take two thickness values into account. As such, it is natural to use the same basic process, using the ellipsis shape, to create the mesh for both the bowl and the handle.

The basic process consists of placing the mesh vertices along appropriately placed and aligned elliptical rings. Then the vertices from two adjacent rings are connected together in groups of three.

(4.3.1) Vertices

Even if we take advantage of the recurring elliptical shapes in the spoon, there is still a question of how to place the elliptical vertex rings. Naturally, the answer differs between the bowl and the handle shapes.

For the bowl, the rings are placed concentrically to the rim, with the first one actually conforming to the rim, with each new ring being slightly further down and slightly smaller. At the very bottom, a single vertex is placed, a "ring" of one so to speak.

The handle is less simple; since it is supposed to model turns and bends, it cannot be reduced to a simple geometrical figure in the same way. As such, the first step is to create a curve to model the mid-point of the cross section of the handle.

This is done by first expressing the local handle properties (thicknesses and angles) as a function of the curve length.

$$\boldsymbol{p}(t) = (r_1(t) \quad r_2(t) \quad \gamma(t) \quad \theta(t) \quad \varphi(t)), t \in [0, L]$$

Where r_1 and r_2 are the two thickness values and L is the total length of the handle as specified by the parameters.

From there, spline interpolation is used to convert it from a discrete function, defined only between two parameter segments, to a continuous function. This, in turn, is used to trace a curve, using the RPY at the current curve length parameters to determine the tangent, normal and binormal, creating a ring with the two radii along the normal and binormal, and then taking a step in the direction of the tangent.

$$R(t) = R_{rpy}(\gamma(t), \theta(t), \varphi(t))$$

$$\begin{aligned} \boldsymbol{v}_{tangent} &= R(t)\hat{x} \\ \boldsymbol{v}_{normal} &= R(t)\hat{y} \\ \boldsymbol{v}_{binormal} &= R(t)\hat{z} \end{aligned}$$

$$\begin{aligned} \boldsymbol{x}_{i+1} &= \boldsymbol{x}_i + \boldsymbol{v}_{tangent} dt \\ t &= t + dt \end{aligned}$$

One thing to note is that in both the bowl and the handle, the number of vertices in any given ring is dependent on how large the ring is. Had the rings all been circular, then the number of vertices would be chosen so that the arc length between two vertices would be constant. Since the rings are elliptical, the actual arc length differs, but the principle is the same.

(4.3.2) Edges and Surfaces

As mentioned above, the vertices for the spoons are grouped in rings. Unfortunately, the in-built functions for meshes does not have a way to take the rings into account, or even keep track of them. As such, the edges have to be defined in the same method that creates the vertices, using local variables to keep track of the rings.

As such, whenever a vertex is created, its coordinates is stored in a vector, hereafter called the ring-vector(s). Each such vector represents one ring, and they are in turn stored in another vector, called the shape-vector. From there, it is easy to calculate the index for each vertex in the mesh's "master list" from their index in the ring-vector, and the index of the ring-vector in the shape-vector, given the index of the first vertex in the first ring.

With the ability to convert indices from the mesh scope to the ring scope, the process of creating the surface triangles is as follows:

Let $a_i \in R_n, i = 0, 1 \dots N$ denote vertex i in the "first" ring, and $b_j \in R_{n+1}, j = 0, 1 \dots M$ vertex j in the second. Furthermore, let the indices wrap around, so that $a_{N+1} = a_0$.

1. Find the vertex pair (a^0, b^0) that minimizes the Euclidian distance between the vertex positions. Keep track of their indices, and let them initialize the iterators i and j.
2. Calculate the distance between a_i and b_{j+1} as well as between a_{i+1} and b_j.
3. Of those two "new" nodes, take the one that resulted in the shortest distance. Connect it together with a_i and b_j in a clockwise fashion with respect to the vertex normals. Store these values in the mesh's edge list.
4. Increment either i or j as appropriate.
5. Repeat steps 2 to 4 until $(a_i, b_j) = (a^0, b^0)$ once more.
6. Repeat steps 1 to 5 for all adjacent pairs of rings.

(4.3.3) Coordinate transforms

The motion transforms are meant to rotate the spoons around a point in the middle of where a theoretical hand would grip the handle. This is represented as a point in the middle of the handle, 2/3rds of the length of the handle from the bowl, with the x-axis aligned with the tangent of the curve in the middle of the spoon.

However, the spoon creation process and, to a lesser degree, the parameter sets themselves assume that the origin is placed at the point where the bowl meets the handle, the bowl rim in the XY-plane and the handle tangent along the positive x-axis. This makes the spoon creation much easier, but it does make it necessary to create a transform that moves and rotate the spoon to the correct place and orientation.

The first step of calculating this transform is to find the correct point in the handle. Due to the way the spoons are constructed, this is a non-trivial task requiring an iterative process outlined in 4.3.1. Stripping of the vertex creation, said process gives us the rotation matrix necessary to calculate the tangent, normal and binormal vectors:

$$R(t) = R_{rpy}(\gamma(t), \theta(t), \varphi(t))$$

This is then used to get the coordinates along the curve in the middle of the handle:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + R(t)\hat{x}dt$$

$$t = t + dt$$

In this case, the iteration is stopped when the curve reaches the appropriate length, i.e. when $t = L_{middle}$. For this project, L_{middle} has been set to $2/3$ of the total length of the handle.

Once the iteration has stopped, the coordinates and rotation matrix are used to create the transform that places the origin at that point in the handle, and makes the tangent vector parallel with the x-axis.

(4.4) Optimization

(4.4.1) Objective function

Implementing an objective function for this project follows the same process as most optimization tasks; Define desired and undesired behaviours and results, find a way to quantify them and find a function that map these values to scores as appropriate.

We will be referring to the contents of the bowl as soup for the sake of simplicity. We will be using the following terms:

Term	Meaning
s	Total score of the spoon
$s_{delivered}$	Score assessing the volume of the delivered soup
$s_{spilled}$	Score assessing the volume of the spilled soup
$s_{direction}$	Score assessing the end direction of the bowl
s_{tilt}	Score assessing the end tilt of the bowl
s_{angle}	Score assessing the complexity of the handle.
V_{lowest}	Lowest possible soup volume found for the entire movement
$V_{desired}$	Desired volume of delivered soup
$V_{initial}$	Initial volume of soup
$\boldsymbol{v}_{mouth}$	Vector denoting mouth position
$\boldsymbol{v}_{bowl}$	Vector denoting bowl direction
$\boldsymbol{v}_{rim}$	Vector denoting direction of the open side of the bowl
$\hat{z}$	Unit vector pointing upwards
$\theta_i^{roll,pitch,yaw}$	Angle difference around denoted RPY axis for handle segment i

The function used is as follows:

$$s = w_1 s_{delivered} + w_2 s_{spilled} + w_3 s_{direction} + w_4 s_{tilt} + w_5 s_{angle} \qquad \begin{matrix} w_i \geq 0 \\ \sum_i w_i = 1 \end{matrix}$$

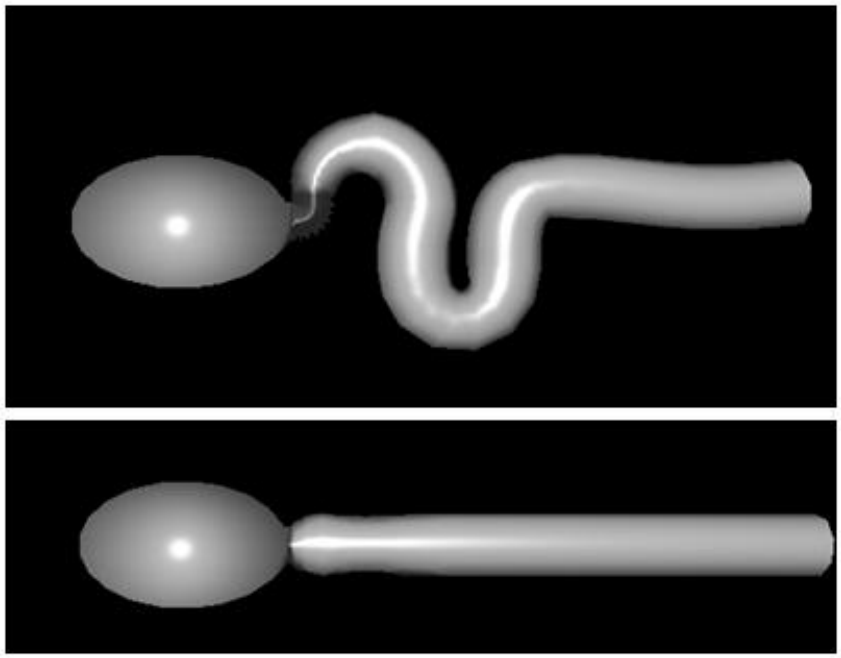

Figure 11: The complexity is measured as the total angle differences along the handle. As such, all else being equal, the simpler spoon (bottom) should be preferable to the complex spoon (top)

$$s_{delivered} = 1 - \exp(-c_{delivered}(V_{lowest} - V_{desired})^2)$$
$$s_{spilled} = 1 - \exp\left(-c_{spilled}(V_{initial} - V_{lowest})^2\right)$$
$$s_{direction} = \frac{1}{2}(1 - \boldsymbol{v}_{mouth} \cdot \boldsymbol{v}_{bowl})$$
$$s_{tilt} = \frac{1}{2}(1 - \hat{z} \cdot \boldsymbol{v}_{rim})$$
$$s_{angle} = \frac{\sum_{i=0}^{n} \theta_i^{roll} + \theta_i^{pitch} + \theta_i^{yaw}}{\frac{3(n+1)}{2}\pi}$$

Where V denotes volumes, $\boldsymbol{v}$ vectors/directions, c constants and w user-defined weights for the weighted sum of the scores s.

The rest of this section will detail why this function was chosen.

(4.4.1.1) Desired behaviour

We wish to create an eating utensil. We wish to optimize an implement to transport a volume from a bowl or plate to the mouth of the patient. The objective function should represent this, its goals should be taken from what one could desire while eating with a spoon.

For the sake of the argument, the meal in question is a soup.

The first goal that comes to mind is that the spoon should deliver soup to the mouth, $s_{delivered}$. This is, after all, the primary purpose of the spoon. However, one must not simply try to maximize the amount delivered, since that goal would be easier to fulfil by simply bringing the entire bowl of soup from the table to the mouth.

We want the bowl to deliver a reasonable amount of soup. Or to rephrase, since a "reasonable amount" is a very vague measure, we want the spoon to either deliver a

specified amount, or to deliver as much as possible while adhering to certain restrictions on bowl size.

The second thing that comes to mind is spillage, $s_{spilled}$. Spilling soup on the table or ones clothes is undesirable, thus we want to minimize how much is spilled during the movement.

Next, the location of the bowl at the end of the motion needs to be considered, $s_{direction}$ and s_{tilt}. In order to be able to eat the soup, the spoon needs to deliver it to the mouth. As such, we want the spoon to end up in a specific orientation. Since translation is a non-factor in this project, the position is irrelevant.

Finally, the shape of the spoon, s_{angle}. Should all else be equal, we would prefer a simple spoon before one with unnecessary twists and turns. As such, we wish to minimize how much the handle bends, albeit with a lower importance than the above criterions.

In other words, an optimal spoon should:

- Deliver a “reasonable” volume
- Minimize the volume spilled
- Have the spoon bowl conveniently oriented relative the mouth
- Not be needlessly complicated

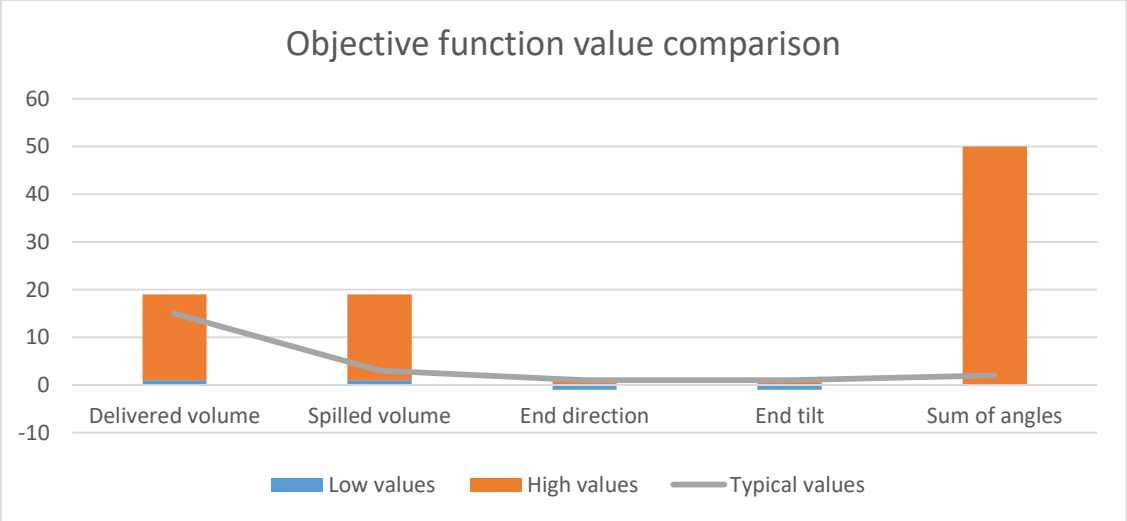

Figure 12: Plot of typical values. Please note that the high value for the angle sum has been truncated to allow the other values to be visible, the actual high value is supposed to be 180

(4.4.1.2) Choice of contributing factors

For this project, the objective function was chosen to be a weighted sum of contributing factors, rather than some non-linear multivariate function. The contributing factors are all quantifiable values related to the criterions named in the last section.

The delivery of a reasonable volume is quantified as the difference between the lowest spoon content volume found during the entire motion and a user-defined “desired” volume.

The minimization of spilled contents is quantified as the difference between the initial spoon content volume and the lowest volume found during the entire motion.

The end direction and end tilt are quantified as scalar products. The former is measured as the scalar product between a normalized vector pointing along the direction of the bowl and a user defined, normalized vector pointing towards where we want it to end up. The mouth vector can be defined to be in any direction, but is assumed to lie in the xy-plane.

The tilt is measured as the scalar product between the normalized spoon-bowl-direction vector projected onto the plane with the mouth vector as normal, and the z-axis itself. This is to make sure that the direction does not affect the tilt value.

The complexity of the handle is measured as the sum of the angle differences for all bends present in the spoon.

(4.4.1.3) Comparability

For a weighted sum to work properly, all parts of the sum must come from similar ranges of values. Otherwise, the ones that tend to larger values will dominate the sum. Trying to set the weights as to compensate for this will either result in terms being near constant (large term with very small weight), or have a rather violent reaction to otherwise small changes (small term with very large weight).

In our case, the contributing factors chosen are, sadly, of quite different sizes.

Both the volume spilled and the volume delivered can vary between 0 and the initial volume calculated. Since the measurement 1 tablespoon equals 15 ml, it would not be unreasonable to have a volumes anywhere between 0 and 30 ml. The only real upper bound on the volumes is determined by the (user defined) maximum bowl size.

Both the direction and the tilt are measured as dot products, and as such vary between -1 and 1.

Lastly, the angle sum varies with the number of segments. Each angle is capped at $\frac{\pi}{2}$ radians, meaning that the angle sum varies between 0 and $\frac{3(n+1)}{2}\pi$, where n is the number of handle segments.

As we can see in figure 12, the volume and angle sums could easily dominate the weighted sum as-is, with the volume parts being far more likely to do so. In order for the weighted sum to work, these values need to be processed to fall within comparable ranges, to be normalized so to speak.

(4.4.1.4) Normalization

All contributing factors have a normalization function applied to them before the weighted sum. These functions make sure that all the terms for the sum lie between 0 and 1.

For the volume terms, this is done through a bell-curve. More specifically, the volume contributions to the weighted sum are both on the form $1 - e^{-c \cdot x^2}$. This way, the greater the deviation from the desired values, the closer the result will be to 0.

More specifically, the two normalization functions are as follows, with s being the score:

$$s_{delivered} = 1 - \exp(-c_{delivered}(V_{lowest} - V_{desired})^2)$$

$$s_{spilled} = 1 - \exp\left(-c_{spilled}(V_{initial} - V_{lowest})^2\right)$$

The function that normalizes the direction and the tilt are much simpler, seeing as they already belong to an interval close to the desired one. The value is simply scaled and subtracted from 1 to get the corrected score:

$$s_{direction} = \frac{1}{2}(1 - \boldsymbol{v}_{mouth} \cdot \boldsymbol{v}_{bowl})$$

$$s_{tilt} = \frac{1}{2}\left(1 - \hat{z} \cdot (\boldsymbol{v}_{mouth} - (\boldsymbol{v}_{mouth} \cdot \boldsymbol{v}_{bowl})\boldsymbol{v}_{bowl})\right)$$

Please note that all vectors involved are normalized. Thus the division by the length of the bowl vector has been omitted.

The last contributing factor, the angle sum, is normalized through division with the maximum value of the angle sum for that number of segments. In other words:

$$s_{angle} = \frac{\sum_{i=0}^{n} \theta_i^{roll} + \theta_i^{pitch} + \theta_i^{yaw}}{\frac{3(n+1)}{2}\pi}$$

With $i = 0$ representing the bowl connection angles.

(4.4.2) Optimization methods

(4.4.2.1) Exhaustive search

As all the parameters are continuous, rather than discrete, a discretization would be required to use exhaustive search here.

The implementation for this specific project is simple, as it often is for exhaustive search. A method creates a list of all possible parameter combinations, which are then tested to find which one gives the best response from the objective function.

It turns out, however, that exhaustive search is unfeasible for this project. Between having a variable number of handle segments, and thereby a variable number of parameters, and the sheer number of parameters for each spoon, this method has problems accomplishing its task.

The former problem can be solved by simply allowing “null” be a valid parameter choice, albeit for entire segments at a time, with all parameters following a null parameters also being null. However, that would only exacerbate the second problem.

Due to the amount of parameters, and that small changes in one parameter can have a large effect on the result as a whole, exhaustive search simply takes too much time to be of any use. This will be discussed further in the time study section of the results chapter.

Let us assume that each parameter has been simplified to n choices. Evaluating parameter sets with a single handle segment would result in n^{13} possible spoons. Two handle segments would mean n^{19} possible spoons. If we use a general case, with m handle segments, we get n^{7+6m} possible spoons. If we try to allow for a variable number of handle segments, we get $\sum_{k=1}^{m} n^{7+6k}$ possible spoons.

In other words, no matter how fast each spoon is evaluated, either the search would take much too long even for debugging purposes, or n would need to be set too low to be of use.

(4.4.2.2) Best of random choice

Unlike the other two methods mentioned, this is not really a proper optimization method. It is more of an adaptation of exhaustive search, but applied to a random selection of possible spoons instead of all of them.

Since a real discretization would take too long to evaluate, the parameter space is instead represented by a large number of uniformly sampled random parameter sets. While a large number, it is nowhere close to the number of sets in a real discretization, and it is possible to evaluate them in a reasonable time.

While this method can give results fairly quickly, it is far from reliable. After all, the best choice need not be a good choice; the only requirement is that it is better than all the other choices.

As such, it is heavily dependent on random chance. There is no real guarantee that the random choice is representative of the entire parameter space, unless the number of sampled spoons is very large, approaching the size of a real discretization. At that point, it starts to run into the same problems as exhaustive search, albeit not quite as severely.

Because of these reasons, "best of random choice" is mostly used as a comparison method, not as something to actually use.

(4.4.2.3) Gradient descent

There is not much alteration or customization needed to use the standard gradient descent algorithm for this project.

To calculate the gradient, two new parameter sets are created for each parameter, with the parameter in question slightly increased and decreased respectively. The partial derivatives for each parameter is then calculated from the scores of these new parameter sets, and the partial derivatives together form the gradient.

The step along the gradient is taken with a variable step length, starting out fairly large and is halved with each attempt that results in a worse score.

There is also a momentum function implemented but inactivated. The momentum function works on the premise of altering the gradient as to get a better score than taking one step along the current gradient. The variable step length already makes sure that each step results in a better score, and makes it impossible to use the momentum part as a way of escaping local minima (as that would require accepting worse results before they get better).

This method does work well for this kind of problem, but requires quite a few calculations in every iteration. With n handle segments, each iteration requires the program to evaluate $2(7 + 6n) + 1 + x$ parameter sets, where x is the number of times the step length is halved.

Another problem is that the parameters have different ranges of values, which can result in some parameters changing more than they should, while others change less.

Finally, as mentioned in the theory segment, gradient descent also makes it difficult to tell if the parameter set found is optimal or merely a local minimum.

(5) Implementation

(5.1) Software used

This project does rely on external software for some tasks, simply because writing them from the ground up is not included in the scope of the project.

The framework the project is written for is called Inviwo. The name is an acronym for Interactive Visualization Workshop, and contains framework for working with linear algebra, meshes and visualization

For the recording and export of motion data, a free Android app called Physics Toolbox is used.

For the evaluation of the objective function, an adapted example code piece for the D3 javascript framework is used. More specifically, an example pertaining to parallel plots.

(5.1.1.1) Recording motion

The movement of the spoon is recorded on a smart phone or tablet using, as mentioned above, the free app Physics Toolbox from Vieyra. More specifically using the inclinometer function of the app, recording the angles as incline, tilt and azimuth.

In other words, it uses Tait-Bryan angles as well, making it well suited for this project.

Once recorded, the data is sent as a .csv file via e-mail, using the app's own export function. The main program has a method for importing the data from said files, with options to apply a low-pass filter to reduce noise and to adjust the starting azimuth of the spoon to a default value. The latter is due to the fact that Physics Toolbox seems to record the azimuth with respect to the North Pole, rather than to the starting orientation.

(5.2) Representations

(5.2.1) Parameter formats

The spoon representation can be stored in two different formats to suit different purposes. The reason for this is that the initial representation proved inconvenient and tedious to use for certain tasks roughly three-quarters through the project. While adding a second representation was easy, re-doing the rest of the program to use the new standard instead of the old one would take as much time and effort, if not more, as using the old one even in the inconvenient part.

As such, having two ways of store and access the spoon representations was considered as acceptable.

(5.2.1.1) Separate segment representation

As the name implies, in this representation all handle segments are stored separately. The bowl parameters are stored in their own vector, and each handle segment has its own vector of parameters. The handle parameter vectors are then stored in another vector, in their proper order.

This representation is the first one implemented, and is the more intuitive of the two representations used. It makes it easy to pick out a specific parameter of a specific segment, as well as quickly figuring out how many segments are in the handle.

However, it is somewhat inconvenient when it comes to repeatedly and systematically access and/or alter each parameter in turn, such as when calculating the gradient, or altering the parameters for the next step in the optimization process.

This representation is mostly used in the mesh creation process, as it makes it easy to use in “for each segment, do the following” style loops. It is also used when sending information between different Inviwo processors, partially because it makes it easy to manually create spoons but mostly as a legacy from earlier in the project.

(5.2.1.2) Single list representation

In this case, each spoon is stored as a vector of 7 + 6n parameters, n being the number of handle segments. The parameters are stored in order according to segment, so element 8 in the vector would be the length of segment 1, while element 14 would be the length of segment 2. The first 7 elements are the bowl parameters.

This representation was added to allow for easier parameter manipulation, around the time when gradient descent was implemented. To access each parameter in order would require a single for-loop in this representation, but at least three in the separate segment representation.

If not for how the rest of the program had already been written with the separate segment representation in mind, this representation would also make it much easier to send spoons between Inviwo processors.

The downside of this representation is that the number of segments is not as easily accessible, requiring some calculations rather than just accessing the length of a vector. It is also less convenient for accessing specific parameters, something that has been the source of many programming errors during the project.

(5.2.2) Parameter space

Rather than having several duplicate methods throughout the program, all methods related to the creation and manipulation of parameter sets are collected in their own class, the “parameter space” class. Especially when considering the two different methods of storing the spoon parameters.

This class contains the following:

- Random number generator
  - With methods to get and set the seed as well as re-set the seed without changing it
- Methods for altering parameters conditionally
  - Input is a positive or negative fraction, the parameter is altered by this fraction of the interval length
  - One method caps out should the final value exceed the allowed interval
  - One method wraps around should the final value exceed the allowed interval
- Methods for creating random spoons
  - Completely randomly, parameters chosen uniformly from their value ranges

- “In the vicinity of” another parameter set. Values chosen from a normal distribution, where the mean is the previous value and the standard deviation is an input value

- Methods for converting between parameter forms
  - From long-list to sets
  - From sets to long-list

(5.2.3) Meshes

The mesh creation is more or less as described in the method section, and Inviwo handles storing and transmitting the meshes as well as the visualization.

The in-built mesh representation in Inviwo consists of five lists. Please note that these are not the standard c++ lists, but rather an Inviwo specific format, referred to editable RAM representations in when calling them.

The first four of these five lists represent properties of the vertices; position, normal, texture coordinate and colour. The last one contains a list of indices for the vertex lists, and represent the triangle surfaces. The indices at 0, 1 and 2 are part of one surface while 3, 4 and 5 are part of the next one and so on.

(5.3) Optimization

The optimization implementation does not differ much from the standard case. The score of a parameter set is calculated, requiring a loop over all transformation matrices in the motion and a volume calculation for each. The gradient is calculated through partial derivatives, requiring two score calculations per parameter. Then, a check is made if a step in the direction of the gradient makes things better, and if not the step length is decreased for the next attempt.

The best of random choice method is also nigh-identical to what is outlined in the method section. A population of parameter sets are sampled from a uniform distribution, and exhaustive search is then applied to that population.

(6) Results

In order to approximate the time needed to evaluate any given problem, a time study was carried out. This time study focuses on three different parts of the process: generating random spoons, evaluating the score of a spoon and the gradient descent process.

All three cases can be simplified as one operation carried out a large number of times. When generating random spoons, this operation would be generating random parameters in the set. For both spoon evaluation and gradient descent, the operation is the volume calculation for a single step in the used motion.

For all of the following tests, the program has been forced to run single-threaded.

(6.1.1.1) *Generating random spoons*

The time needed to generate random spoons is linearly dependent on both the number of spoons to generate and the number of parameters per spoon.

From a series of tests, we can conclude the following:

Generating 500 000 spoons with 31 parameters (4 handle segments) each takes, on average, 6,8 seconds. In other words, 73 548 spoons with 31 parameters each can be generated per second.

Generalizing this, when handling spoons with p parameters, $\frac{2\ 280\ 000}{p}$ spoons can be generated per second.

(6.1.1.2) Evaluating spoons

The time it takes to evaluate the score of a spoon depends on how many times the volume needs to be calculated. In other words, the time it takes to evaluate a spoon is linearly dependent on the number of matrices that make up the movement used.

Using the same configuration as before (500 000 spoons, 31 parameters (4 segments)), and a movement consisting of 50 steps, this gives 3 450 000 volume calculations per second

When optimizing for an arbitrary movement consisting of m movement matrices the program can evaluate $\frac{3\ 450\ 000}{m}$ spoons per second. With $m = 50$, this becomes 69 000 spoons per second.

(6.1.1.3) Gradient descent

Calculating the gradient requires $2p \cdot m$ volume calculations, with m being the number of steps in the movement and p the number of parameters per spoon.

Given the results above, we expect the program to be able to calculate $\frac{3\ 450\ 000}{2p \cdot m}$ gradients per second. With the standard 31 parameters and 50 movement steps, going solely by the volume calculations the program should be able to calculate 1116 gradients per second.

Since there is other things in the gradient calculation function than just volume calculations, the actual number of gradients calculated per second is quite a bit lower. It

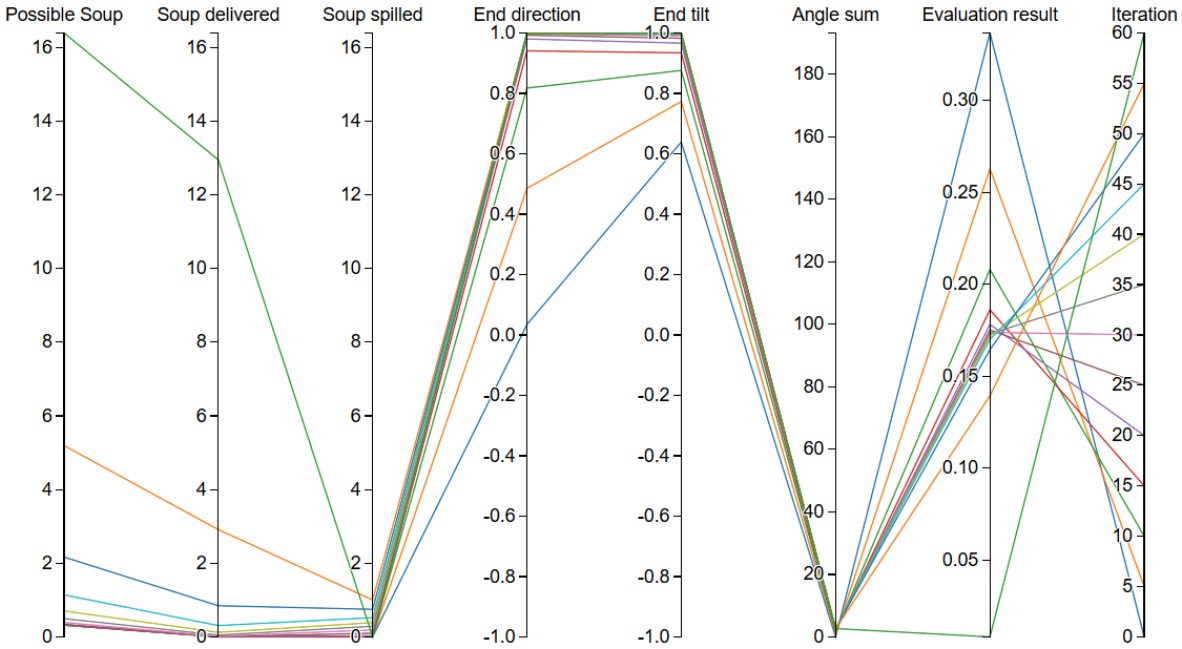

Figure 13: The parallel plot of an example optimization

takes around 870 seconds to calculate 500 000 gradients for 31 parameter spoons over 50 step movements. This translates to 570 gradients per second.

Since further calculations are needed to turn the gradient calculation into a gradient descent iteration, the number of iterations per second is slightly lower still. 500 000 iterations under the same conditions take around 1180 seconds, meaning approximately 425 gradient descent steps per second.

(6.2) Optimization

To evaluate the optimization process, a standard spoon was put through gradient descent for different movements, with the results being exported at different stages of the optimization.

All scenarios will use the same weights for the objective function, and the same standardized input spoon in all but one case.

The input spoon has four handle segments, a straight handle and a tiny bowl.

The weights are as follows:

Content delivered has a weight of 22%, content spilled 19%, bowl direction 28%, bowl tilt 17% and the angle sum 14%. These weights were chosen through trial-and-error, with the goal of letting all five parts of the objective function actually contribute.

The results are gathered through running the input spoon through the optimization process under the specified conditions several times, stopping at different points to export the results at said point.

The results of these different optimization lengths will be presented as both screenshots of the resulting meshes and as parallel plots of the values contributing to the objective function.

The parallel plots will look as in figure 13 Each line represents one exported result.

“Possible soup” represents the volume of the bowl, i.e. the largest possible volume that can be contained in that bowl. “Soup delivered” is the bowl contents at the end of the motion, “Soup spilled” is the difference between “Soup delivered” and the initial contents of the bowl.

The next three axes present their corresponding values as outlined in the objective function segments earlier in this report.

“Evaluation result” is the output of the objective function for the weights defined above, with a lower score being better.

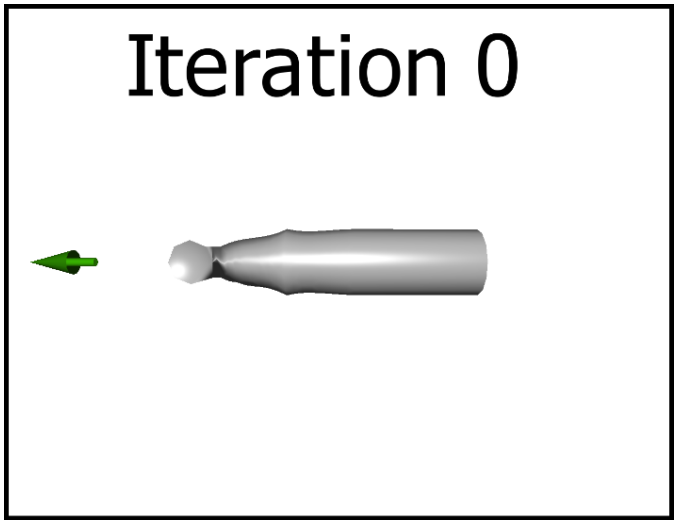

Figure 14: The initial spoon for all but one of the optimizations, as seen from above when held flat.

The number gradient descent steps used can be seen on the far-right axis.

The screenshots of the resulting spoons will be of the same form as figure 14. The spoon will either be seen from above, or from the side. The text at the top tells us how many gradient descent steps were taken to get this specific result. The green arrow at the left side of the frame shows the direction of the mouth of the patient.

(6.2.1) No movement

The first scenario tested is the most basic one possible; no movement other than noise, spoon already angled correctly. In this case, we expect the optimization to simply re-size the bowl until it hits the target volume of 15 ml.

(6.2.1.1) Resulting spoon

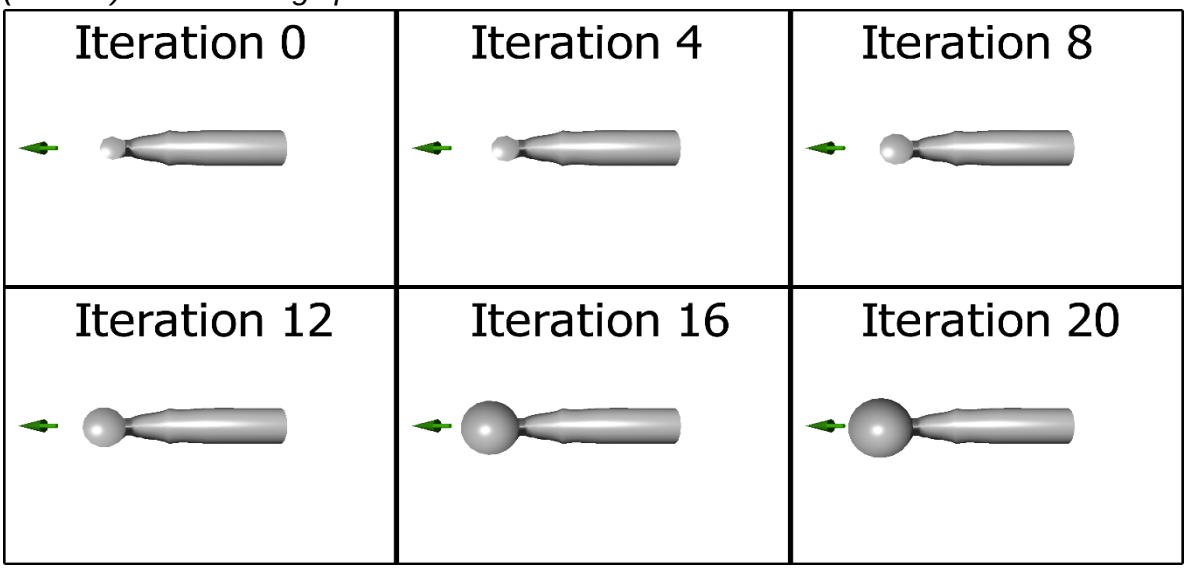

Figure 15: The spoon results for no motion

The actual results reflects our expectations perfectly. The only change is that the bowl increases in size until it matches the desired volume of 15 ml.

(6.2.1.2) Objective function

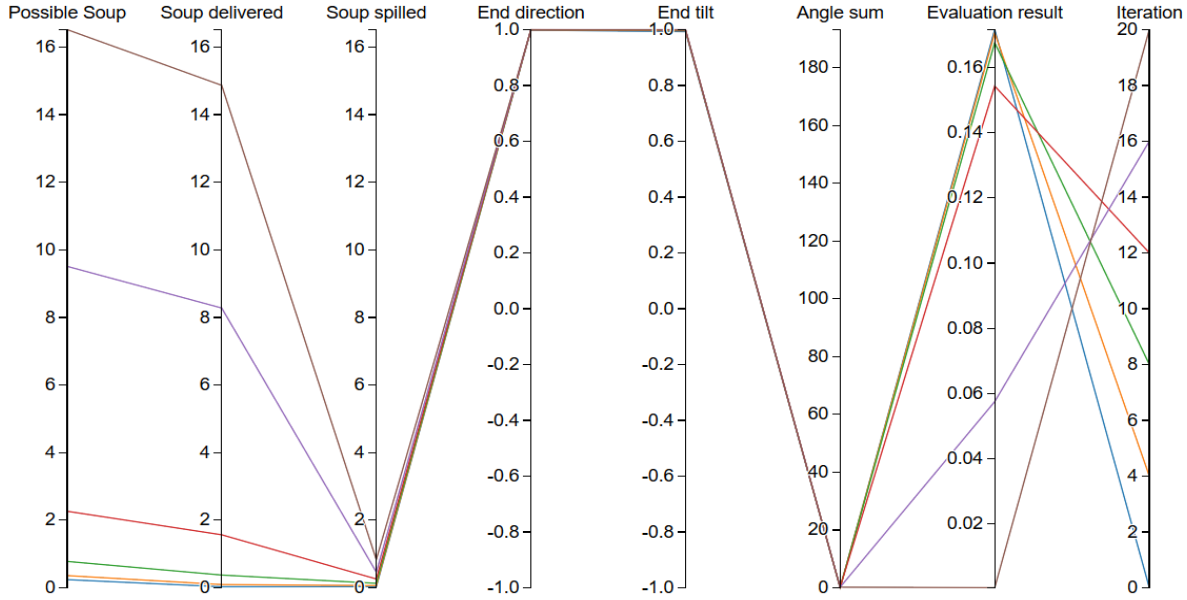

Figure 16: Parallel plot result for no motion

Again, the results match our expectations. The direction and tilt are correct from the beginning, and does not change at all. The noise in the movement causes some spillage, meaning that the delivered contents are slightly lower than the highest possible, and the spillage increases with the volume.

(6.2.2) Tilted initial spoon

This is the only case with an altered initial spoon. In this case, the spoon bowl has been tilted 90°, so that the initial contents should be 0. We expect the optimization to undo this rotation, and increase the size of the bowl.

(6.2.2.1) Resulting spoon

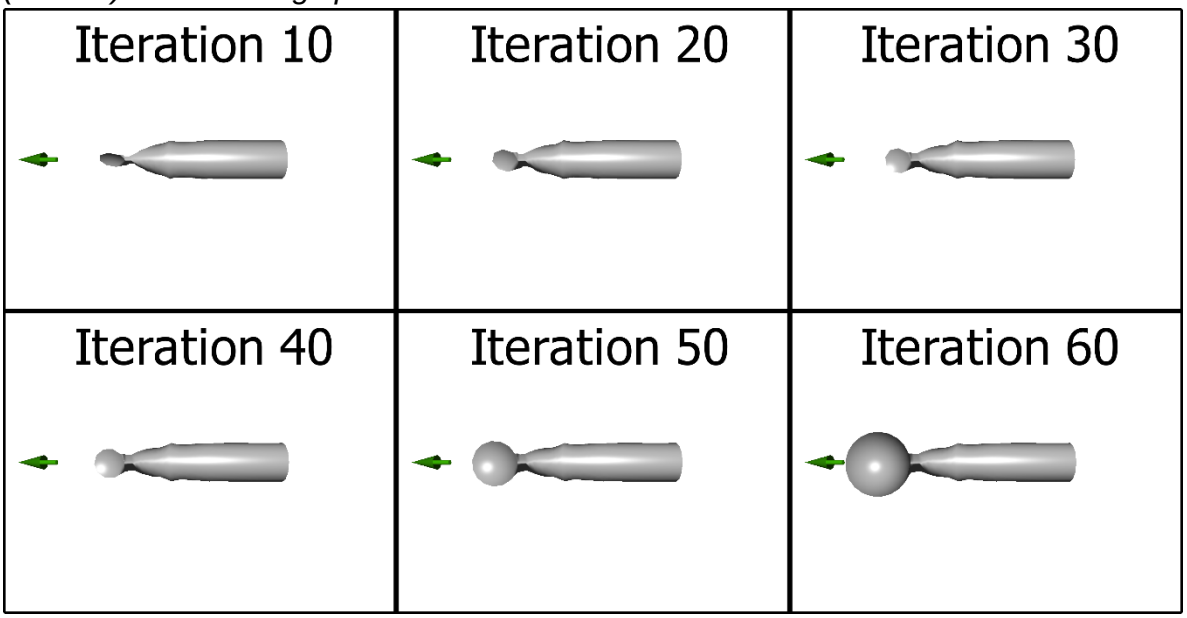

Figure 17: Spoon results for tilted bowl and no motion

The results are as expected, albeit very slow. The bottom row reflects the entirety of the "no movement" results. In other words, with little to no initial bowl contents and a low bowl volume, the optimization process is very slow.

(6.2.2.2) Objective function

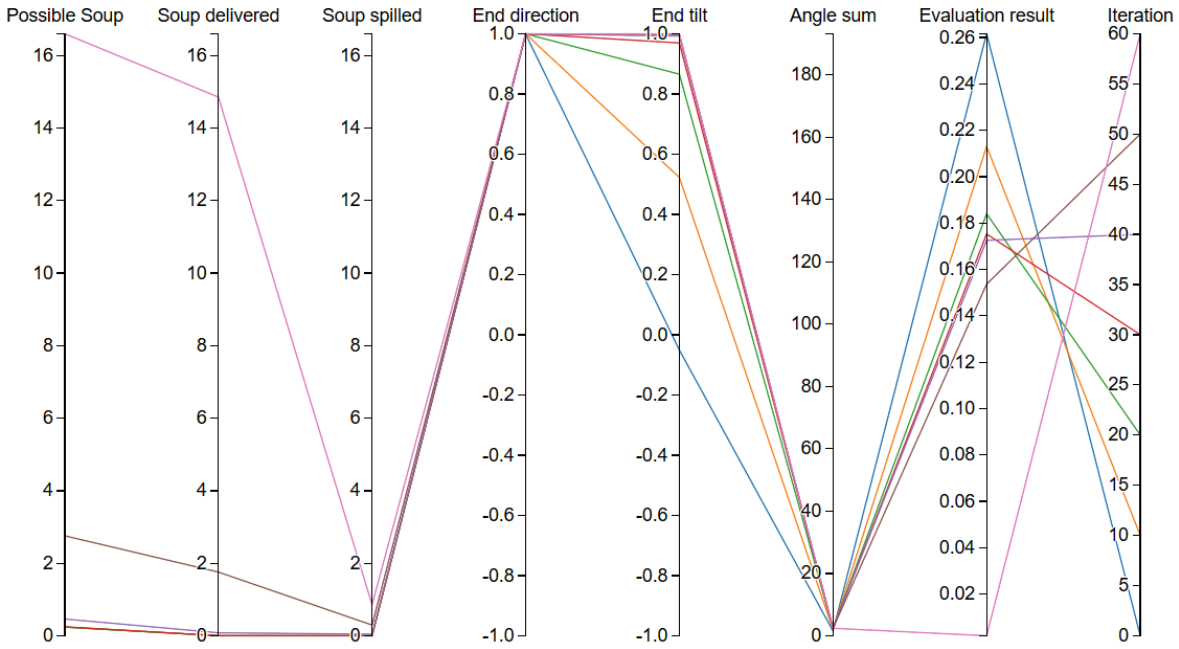

Figure 18: Parallel plot result for tilted initial spoon and no motion

These result are also mostly as expected. Notice that the volume is not altered at all until the tilt gets close to 1. This will be discussed more later on.

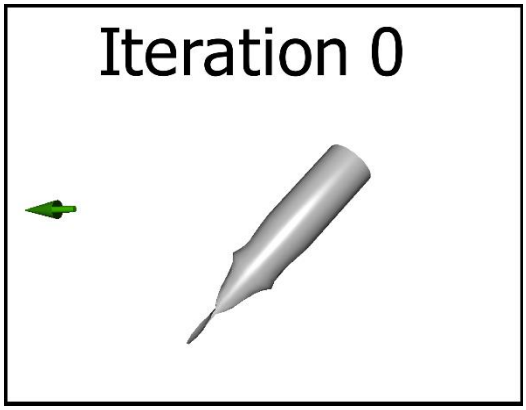

Figure 19: The initial state for upright grip

(6.2.3) Upright grip

In this case, the spoon is held in a mostly upright grip, halfway between how to hold a regular spoon and how to hold a ladle. There is no movement besides noise to take into account. Due to most of the corrections happening in the x-z plane, the spoon results will presented from the side, rather than from above.

We expect the optimization to bend the spoon so that the bowl does not spill its contents immediately and increase the size of the bowl.

(6.2.3.1) Resulting spoon

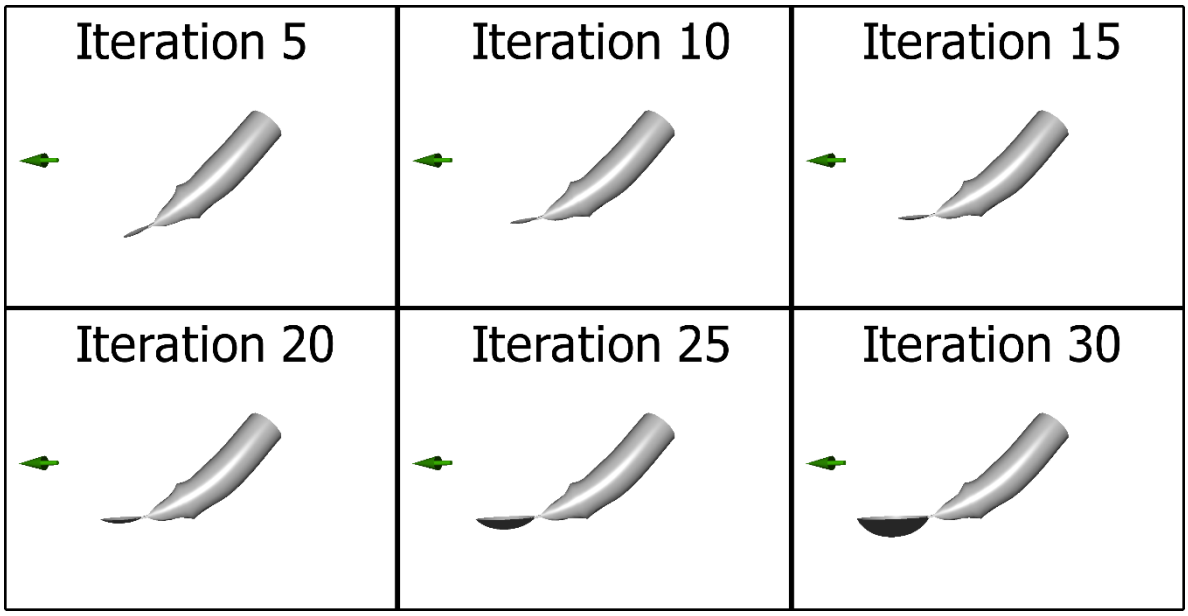

Figure 20: The spoon results for upright grip and no motion

The results are as expected. Once again, the orientation of the bowl is adjusted before the size is altered at all.

The adjustments do happen a lot faster than with the tilted bowl, even though it should not.

(6.2.3.2) Objective function

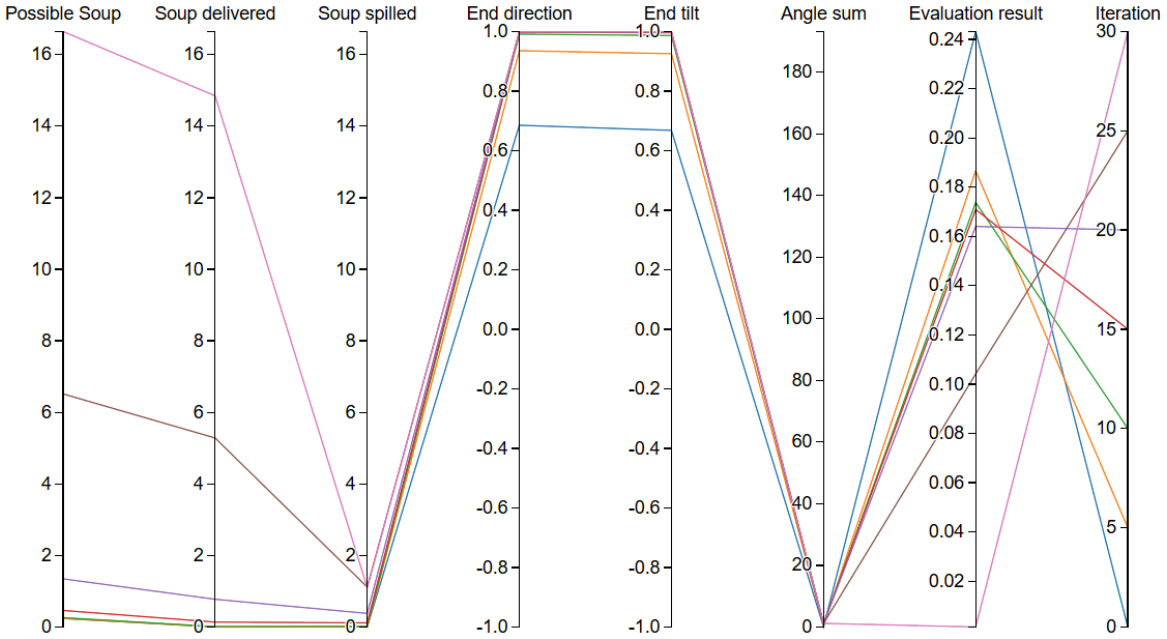

Figure 21: Parallel plot result for upright grip and no motion

The plot is mostly as expected, with an interesting exception. We would expect the diagram to be fairly similar to the one for the tilted bowl, but the end direction scores does not match up. This is due to the end direction and end tilt scores being the result of dot products. The end direction is the dot product of the direction along the spoon bowl (from handle connection straight across) and the direction from the origin to the mouth. In the previous case, the bowl was rotated around said vector, leaving the dot product unaffected. Here, however, it is affected by the tilt.

(6.2.4) Turned left

Here, the spoon is held flat like in the no-movement-case, but with an actual motion. Instead of being held still aside from some noise, the movement here consists of the bowl being turned almost 90° to the left.

(6.2.4.1) Resulting spoon

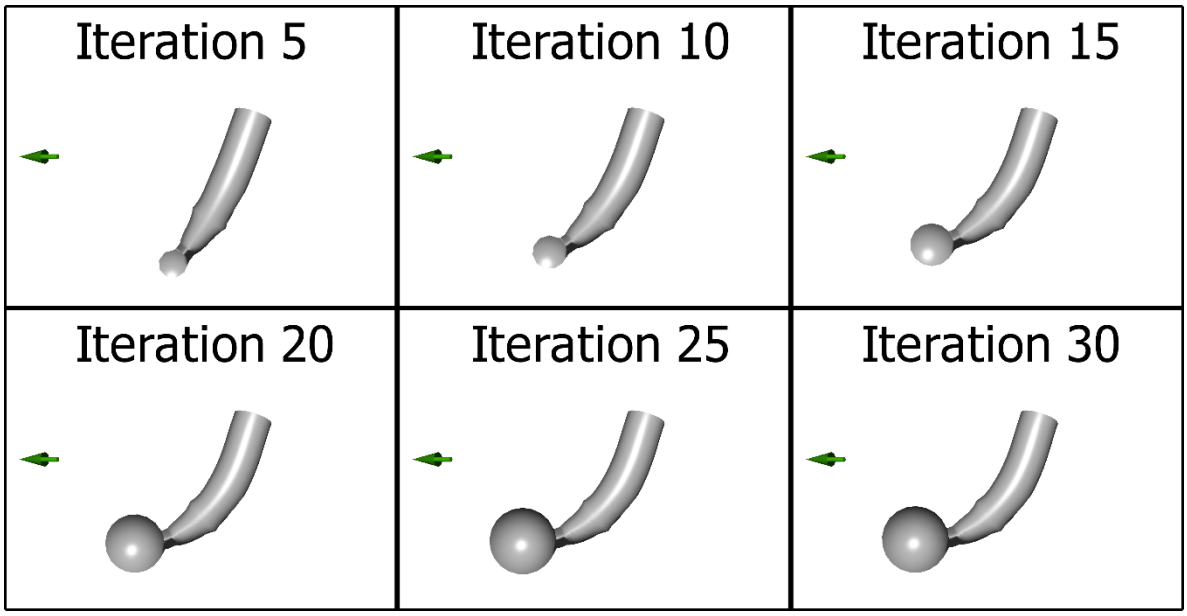

Figure 22: The spoon results for 90° positive rotation around z-axis

The results are as expected. The handle bends to make sure that the bowl is pointed towards the mouth, and its size increases towards the desired volume.

(6.2.4.2) Objective function

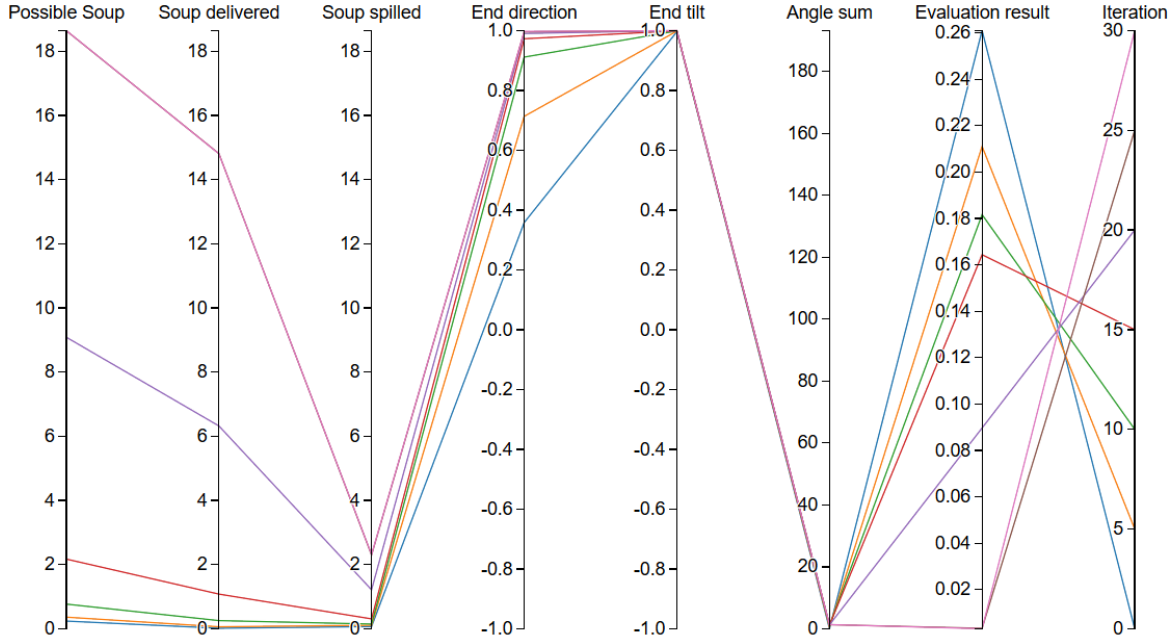

Figure 23: Parallel plot result for 90° positive rotation around z-axis

Once more, the results are as expected. The end tilt remains constant, whilst the end direction and volume parts improve.

(6.2.5) General tendencies

For both the upright grip and the tilted bowl, the orientation of the bowl is adjusted before the size of the bowl, but in the turned left case they are adjusted simultaneously. This is because in the latter case, the bowl starts in a position where it can deliver a non-zero volume.

In other words, the volume delivered only affects the gradient in cases when the bowl has something in it at the end of the motion.

Furthermore, the spillage part values smaller bowls above larger ones. This is due to the spillage part operating on raw volume, so a very small bowl spilling everything is considered preferable to a very large bowl spilling half its contents.

This also means that the spillage part will consider a zero-volume bowl to be optimal, since it will always deliver 100% of the initial 0 ml of contents.

(6.3) Printed spoon

One of the goals of this project is to have the spoons in a format that can be exported and used by a 3D-printer. This has been accomplished by modelling the spoon as a mesh. However, as the spoon model was written as part of this thesis, it needs to be evaluated like the other parts of the program.

As such, an optimized spoon mesh was exported and sent through a 3d-printer, to evaluate the results. Doing so proved that printing the optimized spoons works, but also revealed some problems.

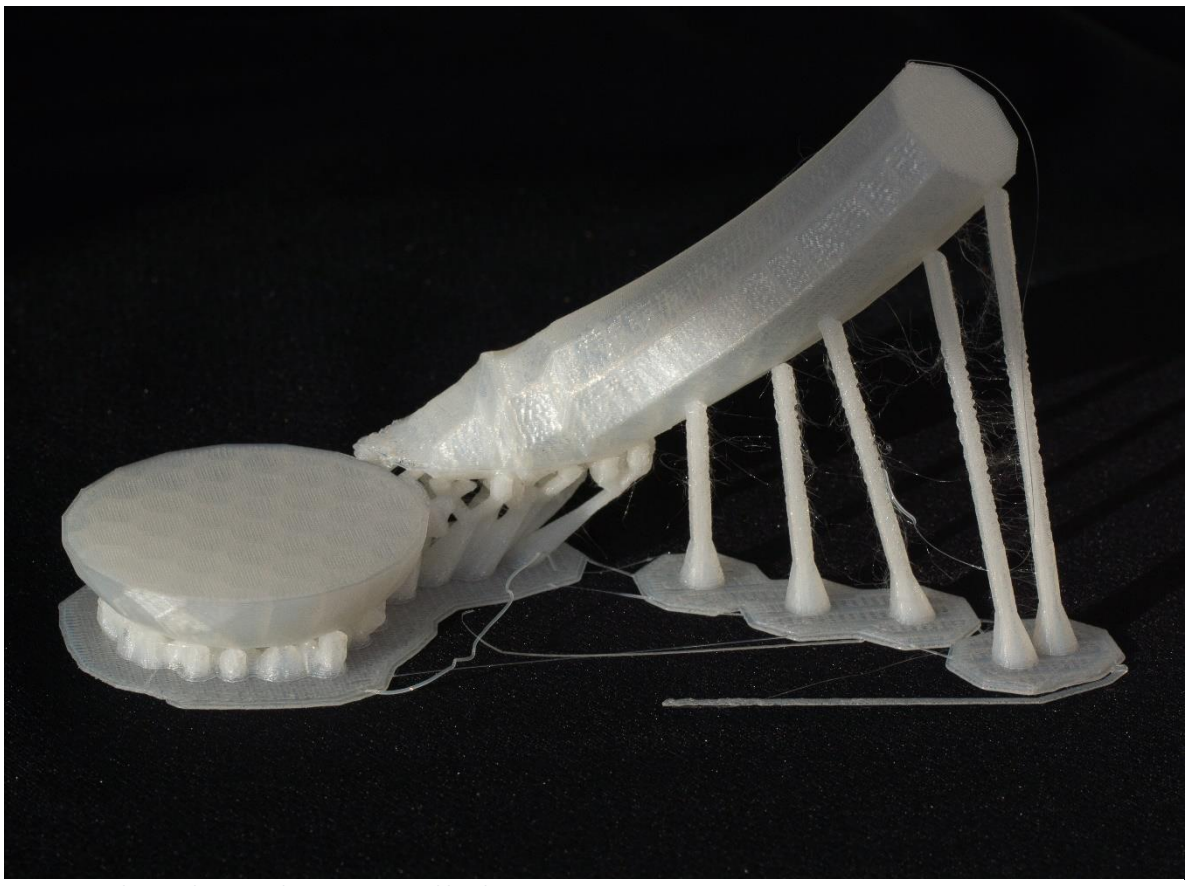

Figure 24: The resulting mesh, as interpreted by the 3D-printer

As can be seen despite the still-intact support structure, the general shape of the spoon does come across, but with a few glaring problems. For one, the bowl is not meant to be solid, and the handle is supposed to be attached to the bowl.

These errors are not the results of some incompatibility with the printer, but due to faults in the mesh creation process. Faults that, unfortunately, did not make themselves known until correcting them would cause an unacceptable delay, and thus would have to be left intact for the larger project to deal with.

The bowl is solid due to that part of the mesh consisting of a single layer of vertices. In short, the mesh describes the rim of an ellipsoid cut, the surface of the object rather than an actual object with a volume. Thus, when printing the spoon, the software auto-completes the surface into a volume the only way it knows; connecting the open ends.

This problem could be solved by creating a second, slightly larger bowl and connecting the bowl rims together. This would cause the bowl to have a volume, rather than just be a surface, and thus avoid the auto-completion.

The reason behind the disconnected handle is that the connection simply is too thin. In the mesh creation, the handle only connects to vertices in the bowl rim. In other words, the thickness of the handle approaches 0 when close to the bowl. As such, it is not too surprising that the bowl is disconnected; the printer does have a minimum thickness.

This problem could only be solved by re-working the way the handle connection is created. To do this, a better method of choosing what bowl vertices should be used for the connection. Since the current method was written early on in the project, it was made with the assumption that a mesh would need to be created for every parameter set evaluation, and thus favoured expediency. Since that assumption proved false, looping over all vertices to find the ones with appropriate coordinates is a much more reasonable prospect.

(7) Conclusions and Discussion

(7.1) Discussing the results

(7.1.1) Spoon parameters

One thing that can be noticed throughout the results section is that there are some spoon parameters that do not affect the outcome. More specifically, the handle thickness does not influence the score in any way. At the same time, the handle thickness could be what decides if a spoon actually could be used by a patient. Having the wrong handle thickness could mean that the spoon either does not fit in the hand, or that it slips out of the patients' grip and the current model has no way of measuring this.

This is not something that can be solved within the current model, as it only evaluates the content of the bowl over the movement. As such, additional measures are required to regulate the size of the handle.

One suggestion could be to, when recording the motion, use something with pressure sensors as well in order to determine the shape of the hand, and then adapt the shape of the handle to that data.

Another idea could be to manually adjust the thickness parameters after the optimization process, although this would likely require more time and effort on the part of the staff and could lead to several spoons being printed because of mistakes.

(7.1.2) Objective function

As mentioned in the “General tendencies” part of the results section, the objective function does have instances where it does not behave as intended.

First of all, the volume delivered part. Currently, said part is a bit too sensitive about when it reacts, as can be seen in the “Tilted initial spoon” and “Upright grip” sections. Part of this is probably caused by the normalization function.

The volume delivered part is normalized through an exponential curve, where hitting the desired volume gives the best score. However, adjusting the response to non-ideal volumes can be a bit tricky. If the function is adjusted to give better response for very low volumes, then the response for almost-correct volumes will decrease while the range for what counts as “almost correct” increases.

One way of mitigating this could be to instead use another type of function, where the score increases with the volume until it reaches the desired value, after which the score remains constant. Depending on the implementation, this could make it much easier to control the response at different volumes, at the expense of not actively discouraging too-large bowls.

The spillage function currently favours smaller bowls, due to them having a lower capacity for spillage. As it currently stand, the spillage part would consider an optimal spoon to have a bowl volume of 0.

The obvious approach to solve this problem would be to reformulate the spillage part to be about how much of the initial volume that has been lost over the movement, rather than the raw volume lost.

(7.2) Ethical and societal implications

Adapted utensils makes it much easier for those with motor impairments to feed themselves. For some, not having personalized utensils means that they need to have someone else feed them at every meal. As such, adapted utensils allows for much greater self-sufficiency and personal freedom for those with motor impairments.

Getting adapted utensils can be difficult, as they have to be manually customized by a physical therapist. The exception to this is if the patient is lucky enough that the ones they need can be found amongst the commercially available specialty utensils, but there is no guarantee of this happening. Because of this, creating a single utensil customized takes quite a bit of time and effort, especially when you consider that physical therapists have other patients and that it can be difficult to get an appointment within a reasonable time frame.

Automating the customization process makes it easier and faster to get adapted utensils. Instead of having to do the entire process by themselves, the physical therapist would only need to oversee the recording of the eating motion(s) and inspect the optimization results. This means that it would take less time per appointment, thus allowing more appointments for utensil customization.

Furthermore, since the optimization results can be saved and stored digitally, it is unnecessary to re-run the entire process to get additional utensils. As such, the patients would not necessarily need another appointment in order to order spare- or replacement utensils.

In short, automating the creation of adapted utensils would lead to more people with motor disabilities having access to adapted utensils, which allows them to be more self-sufficient. It also allows physical therapists to help more people in the same period of time, since it makes one of their undertakings much simpler.

(7.3) Further work

(7.3.1) Improvements for the current goals

First of all, there are several things that can be done to improve the optimization process. Some of these are flaws mentioned in earlier parts, while others are pure improvements.

The most obvious flaw would be the output mesh. Having a second layer for the bowl and a better connection between it and the handle would make the 3D printer create actual spoons rather than a handle and a filled ellipsoid cut.

Then there is the objective function. The current setup makes the delivered volume evaluation twitchy, while the spillage evaluation prefers not having any bowl contents to begin with. Ways to improve on this has been mentioned above; make the volume evaluation score go constant rather than decrease past the target value, and reformulate the spillage to some sort of fraction of the initial volume.

On the improvement side of things, there are a few things that could be done. There are, as mentioned in the theory section, other optimization methods that could be used. Some of which might be more efficient and/or give better results than gradient descent.

It might also be possible to switch over to a non-parameterized model, although that would require a near total re-work of the code. It would, however, allow much greater flexibility in both handle- and bowl-shape.

Finally, it would be a good idea to allow for handle thickness evaluation with respect to the hand. Unfortunately, this would require more data on the hand of the patient than is available at this stage.

(7.3.2) Work for the greater project

This thesis, as has been mentioned before, serves as a prototype and platform for improvements for the larger project. As such, there are quite a few things to add and/or alter to make the program reach the project goals.

The most immediately obvious thing would be to collect information on the hand of the patient, as mentioned above. This would allow for adapting the handle to fit the hand of the patient better, and actually make changes to the handle thickness affect the final score of the spoon.

As a related change, a dedicated tool for recording motion and hand configuration could be very useful. The current method of recording data via a third-party application on a smartphone is not particularly reliable, and requires that either patient or physical therapist has a phone with the required sensors.

Furthermore, while this thesis focuses on spoon creation, the larger project has as a goal to allow for knife and fork optimization as well. As such, the program would need a way of modelling these utensils.

It would also need methods of evaluating the downward force that the utensil can apply (which also puts further demands on the recording tool), as both knives and forks are meant to apply force to the food on the plate.

In the case of knives, the program would also need to model the cutting motion. Similarly, the forks would require some way of determining how good they are at piercing and holding different kinds of food, as well as the motion to deliver the food to the mouth.

(8) Bibliography

Boisselle, A., & Simmonds, M. (2014). Interdisciplinary partnerships between rehabilitation therapists and computer scientists: a proposed model. *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '14).* New York, NY, USA: ACM. doi:10.1145/2674396.2674433

Casio Computer Company. (n.d.). *volume of ellipsoid cap.* Retrieved 10 13, 2017, from http://keisan.casio.com/exec/system/1311572253

Dirkx, D., Mooij, E., & SpringerLink. (2017). *Conceptual Shape Optimization of Entry Vehicles Applied to Capsules and Winged Fuselage Vehicles.* Springer Aerospace Technology.

Gabbasa, A., Jawad, B., & Koutsavdis, E. (2012). CFD-Based Shape Optimization for Optimal Aerodynamic Design. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems, 5*, 227-237. doi:10.4271/2012-01-0507

Gajos, K., Weld, D., Wobbrock, J., Christianson, D., Henning, K., Hoffman, R., . . . Wu, A. (2010). *SUPPLE: Automatically Generating Personalized User Interfaces.* Harvard University, Intelligent Interactive Systems Group, Cambridge, Massachusetts. Retrieved 10 11, 2017, from http://www.eecs.harvard.edu/~kgajos/research/supple/

Gies, D., & Rahmat-Samii, Y. (2004). Particle swarm optimization (PSO) for reflector antenna shaping. *Antennas and Propagation Society International Symposium, 2004. IEEE.* Monterey, CA, USA, USA: IEEE. doi:10.1109/APS.2004.1331828

Li, H., Alhashim, I., Zhang, H., Shamir, A., & Cohen-Or, D. (2012). Stackabilization. *ACM Transactions on Graphics (TOG)*. doi:10.1145/2366145.2366177

Li, H., Hu, R., Alhashim, I., & Zhang, H. (2015). Foldabilizing furniture. *ACM Transactions on Graphics (TOG)*. doi:10.1145/2766912

McDonald, S., Levine, D., Richards, J., & Aguilar, L. (2016). Effectiveness of adaptive silverware on range of motion of the hand. *PeerJ*, 4:e1667. doi:10.7717/peerj.1667

Mohammadi, B., & Pironneau, O. (2009). *Applied Shape Optimization for Fluids.* Oxford: Oxford University Press.

Naithani, S., Whelan, K., Thomas, J., Gulliford, M. C., & Morgan, M. (2008). Hospital inpatients' experiences of access to food: a qualitative interview and observational study. *Health Expectations*, 294–303. doi:10.1111/j.1369-7625.2008.00495.x

Seo, K., Takahashi, N., Kawabata, K., & Mitsui, T. (2016). Optimization of the Design of a Discus for People with Disabilities. *Procedia Engineering*, 538-543. doi:10.1016/j.proeng.2016.06.234

user_777. (2013). *Volume of the smaller region of ellipsoid cut by plane.* Mathematics Stack Exchange (website). Retrieved 10 13, 2017, from https://math.stackexchange.com/questions/1145267/volume-of-the-smaller-region-of-ellipsoid-cut-by-plane

Usiskin, Z., Peressini, A. L., & Marchisotto, E. (2003). *Mathematics for High School Teachers: An Advanced Perspective.* Retrieved 10 11, 2017, from https://en.wikipedia.org/wiki/Geometric_transformation