Computational Hydrodynamic Analysis for Speed Maximization (CHASM)

2024 Supercomputing Challenge

Luke Rand, Greta Swanson, Nandita Ganesan

Final Report

Fluid Mechanics
Santa Fe Preparatory School
Santa Fe, NM, US
April 10 2023

Contents

Introduction

1 Executive Summary

With access to adequate coaching becoming an ever increasing issue, disproportionately affecting underprivileged groups and individuals, the need for technological coaching aids to cut expenses increases. Swimming faces this challenge, and thus requires remedies. Our project, Computational Hydrodynamic Analysis for Speed Maximization (CHASM) achieves a method to provide coaching at low expense for swimmers without access, as well as increase the efficacy of pre-existing coaching programs for the sport of competitive swimming, potentially allowing the growth and increased accessibility of aforementioned teams. CHASM provides machine augmentation and replacement for human coaching time through computational analysis of the movements of the swimmer. Our model takes captured video footage of a swimmer, analyzes it using hydrodynamic drag approximation, and outputs a simple visual to allow either the swimmer themselves or a coach to locate points of human error. Integration of the hardware and model into swim teams and local pools can expand access to the sport of competitive swimming and address inequality issues related to coaching access. Additionally, with many people around the world facing injuries or disabilities that limit their movement and change their bodies, individualized coaching for swimming, a commonly prescribed recovery sport, is more important than ever. CHASM addresses this as well, with direct individualization providing the ability to account for such differences.

2 Method Outline

- First, an inexpensive PVC frame is attached to the lane line on either side of a swimming lane. This is the Lane Videography Unit (LVU) and is used to capture footage of the swimmer from specific angles using Apple iPhones which are mounted in waterproof casings on the LVU. IPhones are used to capture video footage due to being a device many coaches and swimmers will already have, and thus do not need to buy.
- Next, a short script takes individual frames from the footage collected by the LVU at an inputted time, and passes them to the model, CHASM, to output data to reduce drag.
- First, CHASM uses a pretrained image model to develop two-dimensional wireframes from the two images associated with each angle.
- CHASM then combines these two-dimensional wireframes into a single three-dimensional wireframe.
- This wireframe is passed to a `Swimmer` object constructor.

- Included in the swimmer object is a function to calculate the approximate drag on the body represented by the wireframe. An optimization function uses the drag approximation function to determine the ideal wireframe for the swimmer object to minimize drag for the exact proportions and sizes of the swimmer.
- Now, both the original wireframe, derived from the swimmer's position, and the optimized wireframe, derived from drag calculations for the swimmer's body attributes, are visually displayed to the user.
- Either the user or a coach of the user can use the outputted data to correct errors and improve swimming performance. The CHASM model can then be used to verify progress and determine further steps towards improvement.

3 The Problem

Across the United States, access to swim coaching has been limited for a variety of reasons, one of the most pertinent reasons being cost. A private lesson with an instructor can cost on average one hundred to three hundred dollars per hour (Dougherty, 201) and most competitive swimmers take private lessons, on average, twice a week, with more dedicated swimmers taking well over two (Koury, 2024). This means a significant amount of money is dedicated to the sport from the swimmers family, and for many, the price is not worth the benefit (Dougherty, 201).

The majority of swimmers will instead join a club or school swim team instead as these have a much lesser price. However, a club team has, at least, upward of thirty athletes on average, and highschool swim teams have upward of forty plus on average, depending on the size of the highschool. Meanwhile, the number of coaches for these teams leads to an approximate ratio of 1:10, coaches to athletes respectively. This means that a coach's attention has to be split between several athletes and each athlete will not receive the attention of a coach in the same way that an athlete taking privates could (Here).

Our program helps to bridge the gap between these two issues. By having a program that is cost efficient and coaches students, it would allow students who are in a high school or club team to have easier and cheaper access to individualized coaching.

4 Current State of the Field

As the sport of swimming stands, technology plays a significant role. Electronic timing machines are used to collect results for races and heart rate monitors are used for training purposes (Wise, 2022). Existing companies provide side by side comparison of video footage and method to review details, such as STREAMLINE. However, human error and labor remains as a key crutch point, as coaches

are still the ones analyzing and watching any collected data. Despite innovations in technology such as glass sided pools for video recording and analysis, CHASM remains unique in a computational approach to the analysis itself, and provides quantitative corrections based purely on the laws of hydrodynamics, which are immutable and infallible.

5 The Streamline Position

The streamline position significantly reduces the amount of drag acting on a swimmer during the dive and push off state of a race or swim, occuring at the beginning and all turns, respectively. This reduction in drag can be advantageous to many swimmers, as reducing drag allows for the swimmer to reach faster speeds (The Importance). For example, a swimmer can spend up to 20 percent of breaststroke in a streamline position, and by having a quicker streamline, a swimmer is able to drastically improve their times for competitions (Poirier, 2023). Beyond this, the streamline allows a swimmer to have a quicker break at the start of the race and carry that momentum and speed throughout their race. If a swimmer has a subpar streamline, they will lose speed at the beginning of their race and have to regain that speed later on, wasting energy (Admin, 2021). The streamline also occurs off the wall at every turn, and lost energy can add up over longer races (Poirier, 2023). Many beginner swimmers are able to improve their streamline and show drastic improvement in their streamline within the first year, as a tight streamline does not require any additional energy or work to generate *(fig. 1)*.

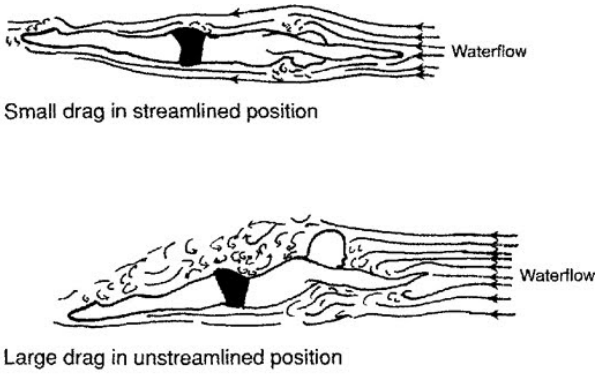


Figure 1: Streamline position of swimmers (360swim).

Beyond the importance of streamlines in competitive swimming, we were limited by our access to technology that would have allowed us to gain the data needed for strokes such as freestyle. Swimming strokes which involve adding energy to the system through movement are significantly harder computationally, and were not approachable within a year time frame. However, a streamline negated many of these concerns, facilitating ease of designing the apparatus and

simplification of computation, as a part of swimming that only involves energy loss through drag and not energy gain and representing an important part of the sport.

6 Importance of Individualization

An ideal swimming streamline consists of arms forward, hands overlapped, and body pointed straight (Holmes, 2022). With clearly defined and well tested data on optimal swimming position for a streamline, individualized calculated data for an optimal position seems unnecessary. However, human bodies differ in size and proportion, and while general technique may remain constant, exact angles and position differ immensely from person to person, and can often only be picked up visually. Furthermore, body types and abilities can differ even more when injuries and disabilities are introduced. Our model is especially able to address these issues, visually displaying data specific to a person's body type and proportions, providing accurate coaching for anyone, and accounting for individuals with differences, whether temporary or permanent. Additionally, if our model were to be expanded for use past just the streamline and into the domain of other strokes such as freestyle, individual body type influences correct swimming form multitudes more, and the technology we outline would become even more pertinent.

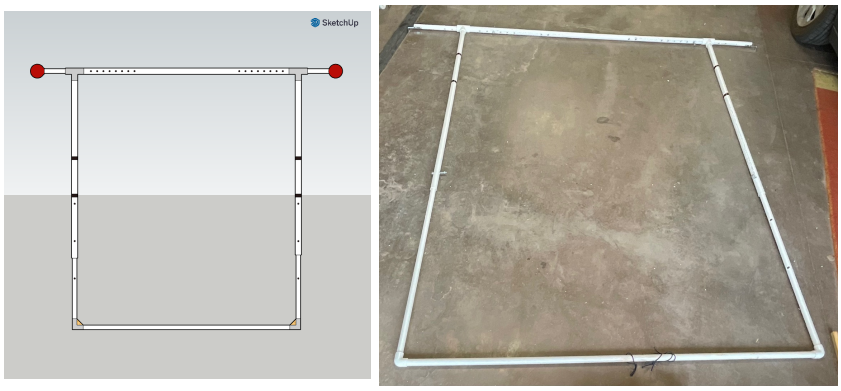
Computational Analysis for Drag Optimization (CHASM)

7 Capturing and Isolating Frames for Analysis

7.1 The Lane Videography Unit

The capture of video is performed using a custom built Lane Videography Unit (LVU). The frame is PVC, certain fasteners are steel, and the iPhone holding mounts are wooden. PVC is chosen for the main frame for its ease of construction, lightweight nature, and relatively low expense. The entire LVU contains under fifty dollars of material. Figure 2a shows the design for the LVU. The vertical axis contain two nested PVC pieces with holes drilled through both pieces. A screw and washer are then able to go through a set of holes decided by the user, creating adaptability for different depths of streamline. The black lines, denoted by tape, demarcate the optimal height of the streamline, most directly in the center of the path of the cameras. The horizontal axis also contains similar adaptability to account for different widths of the lane. The cameras are mounted at 45 degree angles for ease of computation, using a square root of two constant for conversion to a three dimensional wireframe as opposed to calculated trigonometry values. The decision to put cameras on angles arose from the understanding that the diagonal distance is longer than the perpendicular

and can more easily fit the entire person within the frame, given the constraints of the minimum width of a swim lane.



(a) (b)

Figure 2: (a) The LVU's blueprint (b) A completed frame of the LVU

7.2 Taking Individual Frames

Since video footage is taken, and not always started at the same time, a short python script determines the beginning of each video, is given a specific time from one of the videos, and adjusts a second time for the second video. A frame from both videos is then returned from the same world time, regardless of video start time.

8 Constructing a Three-Dimensional Wireframe

8.1 Clarifying Terminology

The CHASM model uses two two-dimensional wireframes of a human body to construct a single three-dimensional wireframe. The body of the swimmer passing through the water is horizontal and positioned perpendicular to the bottom of the pool. Between multiple wireframes and a body positioned horizontally, the terminology in respect to coordinate axis can prove unintuitive. Thus, in order to clearly describe position, it is necessary to clarify which axis each label refers to. For the purposes of this report, the Y axis will be used to describe the horizontal axis between a swimmer's feet and head, with lower numbers at the feet and higher numbers at the head. Similarly, the X axis runs from right to left (from the swimmer's perspective, meaning left to right from a viewer's perspective), and the Z axis runs from the swimmer's chest to back. X_1 and Z_1 are used to refer to the non-Y axis of the left and right two-dimensional wireframes, respectively.

8.2 OpenCV Pose Recognition

We performed two-dimensional pose recognition for two-dimensional wireframes from images using the OpenCV library and the MPII Human Pose Dataset (Andriluka et al., 2014). We use a pretrained model (Gupta, 2018) which integrates the MPII Dataset with the OpenCV image processing library to derive points at specific locations, which we use to construct our wireframe. Usage of the model consists of three parts. Initially, we load the model network to be used on our images. Two model networks must be initialized, one for each of the two angles in our image. Images are then rotated so the Y axis is vertical on the image. The MPII Dataset primarily includes upright bodies, meaning vertical rotation is important for accurate wireframe point location. Next, we determine the frames necessary for the model to find wireframe points. A frame is essentially OpenCV's interpretation of an image, and involves locating the images in a file structure and using a constructor to initialize each frame from the corresponding image. Lastly, the frames are given to the model, which outputs a two-dimensional wireframe for each frame *(fig. 3)*.

In order to use the wireframes computationally, it is necessary to make a few changes to each output wireframe. The location of the point corresponding to the chest is determined and its corresponding vector is subtracted from the vector of each wireframe point to "zero" the wireframe around the origin. The result is a translation of the wireframe to the chest point as the origin. This makes combination of wireframes significantly easier and improves compute time by minimizing the need for vector subtraction later on. Additionally, the wireframe is vertically inverted along the Y axis so that drag can later be computed from bottom to top, low Y values to higher numbers.

8.2.1 Code

```
#Method to translate points for computability
def translate_points(points):
    #determine chest point and initialize new points
    chestpoint = points[14]
    new_points = []
    #adjust each point
    for i in range(len(points)):
        new_point = [0, 0]
        #adjust each axis
        for j in range(2):
        new_point[j] = points[i][j] - chestpoint[j]
    new_points.append(new_point)
  return new_points
```

8.3 Combining Wireframes to add Depth

As previously stated, the two-dimensional wireframes from each angle of video are taken from diagonals, this makes constructing a three-dimensional wireframe

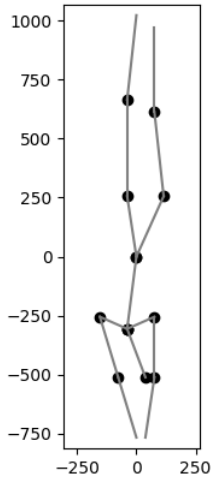


Figure 3: A two-dimensional wireframe determined from a streamlined swimmer.

mathematically more complex, yet reduces the margin of error and mitigates cost. Mathematical complexity results from converting the two diagonal coordinate systems geometrically into a single three-dimensional standard Cartesian coordinate system, but the diagonal approach is a better course than taking horizontal and vertical two-dimensional wireframes for two main reasons. First, diagonal perspectives on the 45 and 135 degrees ensure that all points are present in both angles, reducing the need for a third camera. Second, on a square frame, diagonals allow us to position the cameras further from the swimmer, mitigating perspective distortion, an image distortion phenomenon that intensifies near the edges of an image. Using the diagonals, we calculate the X, Y, and Z values of each point in three-dimensional Cartesian space from their two-dimensional counterparts using formulas determined geometrically. Due to the engineered 45 degree angle nature of the right triangles formed by the lines of view of the cameras, consistent algebra using a constant value of the square root of two can be used, significantly increasing performance times than what would be necessary for trigonometric computations. Each pair of corresponding two-dimensional points is used to calculate a single three-dimensional point, and each three-dimensional point is then appended onto an array to create a three-dimensional wireframe of the swimmer *(fig. 4)*.

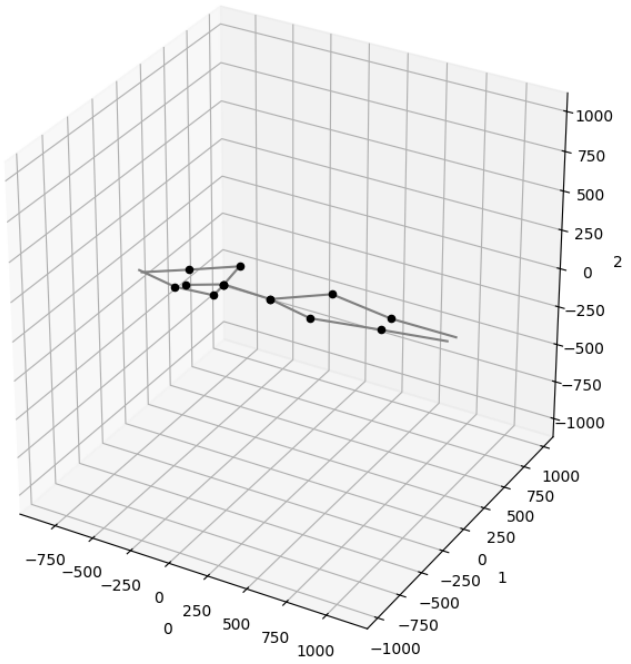


Figure 4: A three-dimensional wireframe generated from a two-dimensional wireframes.

8.3.1 Code

```
#method to calculate point in 3d from 2d points. cam1 and cam2 are 2d points in [y,x,z] format
def getPoint(cam1, cam2):
    #argument parsing
    #x1 and z1 refer to the 2d image while x and z refer to the 3d. Y is the same for both
    ys = [cam1[1], cam2[1]]
    x1 = cam1[0]
    z1 = cam2[0]
    #initialize point to return
    point = []
    #average y values for y
    point.append(sum(ys)/len(ys))
    #converting diagonals to horizontal and vertical coordinate system
    x = (z1+x1)/root2
    z = (z1-x1)/root2
```

```
        #append to point
        point.append(x)
        point.append(z)
        return point
```

9 Determining Drag Associated with a Wireframe

9.1 Determining Flow Type

Flow over a body can be characterized as one of two types: turbulent and laminar, with laminar flow characterized as separate layers with little interference, generally seen in denser, slower moving fluids, and turbulent flow characterized by high levels of interference, conversely with lighter and faster fluids. Drag created by a body moving with laminar flow along its edges is generated primarily by surface area drag, while turbulent flow creates form drag, produced by interfering layers of fluids ("Laminar and Turbulent," 1992). The distinction between flow is based on a Reynolds number, calculated using the velocity of the body, the "characteristic length" of the body, which is the head to toe Y-axis length of the swimmer due to it's relatively flat form, and the viscosity of the fluid (Ungerechts, 1982). For a body moving through fluid, a Reynolds number of below 1 indicates that flow is laminar, while a Reynolds number above roughly 10^6 indicates completely turbulent flow (OpenStax College, n.d.). The Reynolds number of a competitive number is in the interval $(2 * 10^5, 2 * 10^6)$, directly situated on the cusp of completely turbulent flow (Ungerechts, 1982). Therefore, our model must take into account both surface area drag and the interference of layers of fluid, commonly referred to as form drag. These two components become increasingly important in the creation of a drag approximation model, as it determines which forces contribute to total drag.

9.2 Drag Approximation Model

To determine drag, the force acting against the swimmer and the primary reason for energy loss in the streamline, we first determined constituent parts contributing to the drag force. As we determined based on the Reynolds number of a swimming body, drag acting on the swimmer will be frictional and form based, both in significant proportions (What is Drag?, 2022).

9.2.1 Approximating Skin Friction

Skin friction is directly proportional to surface area in contact with the fluid, in our case water, and can be calculated with the surface area equation for a geometric shape, and more complicated surface area determination techniques for more complex shapes. Since we do not yet know how much skin friction influences drag under our circumstances, we multiply the surface area by an

arbitrary constant, A. This finalizes our skin friction calculation equation:

$$A * SurfaceArea$$

9.2.2 Approximating Form Drag

Approximating form drag proves a more difficult challenge. Form drag is generated by the change in local pressures and flow around a body, resulting in mixing of streams of airflow and areas of high and low pressure. This varying pressure across the body generates a force which can be calculated by integrating the pressure difference across the body's surface area (What is Drag?, 2022). At a high level, differences in pressure are caused by deflection of fluid against the surface of a body, which increases with the angle of impact. Thus, a computationally simple approximation of form drag can be achieved by determining a metric associated with the angle between the surface of the body and the direction of flow. It is important to note that while we refer to a direction of flow, the water is relatively static, and the swimmer is moving through the water. This can, however, be visualized as a direction of flow opposite to the direction of movement of the body. We calculated such a metric and labeled it as the body's Pointiness Number, with a lower number representing higher pointiness, somewhat counter intuitively. However, pointiness has two forms, frontal and rear. Frontal pointiness represents the slope against the direction of flow for fluid colliding with the object, and rear pointiness represents the slope against the reverse direction of flow, representing how gradually fluid fills the void behind the object. The drag on spheres and hemispheres definitively shows this discretion *(tab. 1)*. For a sphere, front facing hemisphere, and rear facing hemisphere of the same maximum cross sectional area, their drag differs, as evidenced by their corresponding drag coefficients, a dimensionless quantity relating the frontal area of an object to the drag experienced by its form (Drag Coefficient, 2023). The sphere has the lowest drag coefficient, presumably due

Sphere	0.20
Frontal Hemisphere	0.42
Rear Hemisphere	1.17

Table 1: Drag coefficients of spheres and hemisphere("Table", 2023)

to a fairly low frontal and rear pointiness. The frontal and rear hemispheres, however, vary in drag coefficient, which led us to believe that frontal and rear pointiness contribute to overall drag in different amounts. Thus, we ascribed two coefficients to determine form drag, F for frontal pointiness, and B for rear pointiness. We then determined our equation for form drag to be:

$$F * FrontalPointiness + B * RearPointiness$$

To determine rear and frontal pointiness, we decided on a "slice and square" method. First, we take cross sectional slices of the body along the direction

of flow *(fig. 5)*. The distance between slices is determined by a resolution constant, and drastically changes compute time and slightly changes accuracy. Very large resolution values create lost data, while resolution values too small result in irrational compute times. We chose a resolution value of 1 unit, which experimentally yielded the most accurate results for objects with known drag. Initially, a `frontalArea` variable is initialized as a cross section of size 0, which will come to represent the area water has already impacted. For each cross section, moving along the direction of flow, the cross sectional area not covered by the frontal area is squared and added to the `FrontalPointiness` value, and cross sectional area that was removed from the previous cross section is squared and added to `RearPointiness`. Squaring the additional value means that larger added areas hold higher significance, causing more drag. This step is what ensures that shallower angles create lower pointiness values, as they will result in less additional area each time.

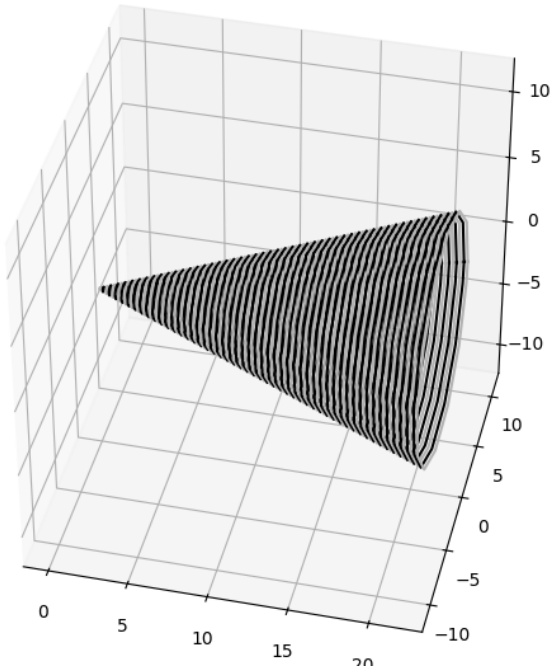


Figure 5: Cross sections taken from a cone.

9.2.3 Code

```
#algorithm to calculate pointiness numbers based on cross
↪  sections. Height is the height of the object
while(i <= height):
```

```
    #determine the next cross section down from the direction of flow
    crossSection2 = prism.getCrossSection(i)
    #increment pointiness numbers appropriately
    pointinessNumberF += (getNewAreaFront(frontalArea, crossSection2)**power)
    pointinessNumberB += (getNewAreaBack(crossSection1, crossSection2)**power)
    #save the current cross section to memory to compare against next time
    crossSection1 = crossSection2
    #add new frontal area to total frontal area
    frontalArea = union_all([frontalArea,crossSection2]).simplify(0.1)
    #increment i to calculate for the next cross section
    i+=resolution
```

9.2.4 Finalizing

Once the algorithms and equations for both frictional and form drag were determined, we amalgamated a finalized drag approximation equation.

$$A * SurfaceArea + F * FrontalPointiness + B * RearPointiness$$

It then became necessary to determine the values of the coefficients and verify the drag model using known values.

9.3 Tuning and Verifying the Drag Model

To verify the drag model and determine the coefficient to each subset of the drag number, we determined a set of geometric shapes with known drag coefficients *(tab. 2)* and tested the shapes with our model to determine the correct ratios between the front pointiness factor, surface area factor, and rear pointiness factor. Cones were chosen as a baseline for front pointiness, the angle corresponding to one half of the vertex, the sphere was chosen to account for surface area drag, and the two bidirectional hemispheres account for the ratio between

Sphere	0.20
Cone (20 degrees)	0.39
Frontal Hemisphere	0.42
Cone (40 degrees)	0.61
Cone (60 degrees)	0.83
Rear Hemisphere	1.17
Cone (90 degrees)	1.17

Table 2: Known drag coefficients used to tune the drag model ("Table", 2023)

rear and front surface area drag. We then ran code to determine which rear pointiness coefficient and surface area coefficient values, given a front surface area coefficient of 1, yielded the same order as experimental data, and chose the median as our coefficients for the project *(fig. 6)*. Thus, complete with the coefficients determined to yield the best approximation of drag, we determined our equation to approximate drag for an object with reference area of 200 square units:

$$1 * FrontalPointiness + 0.5 * RearPointiness + 0.6 * TotalSurfaceArea$$

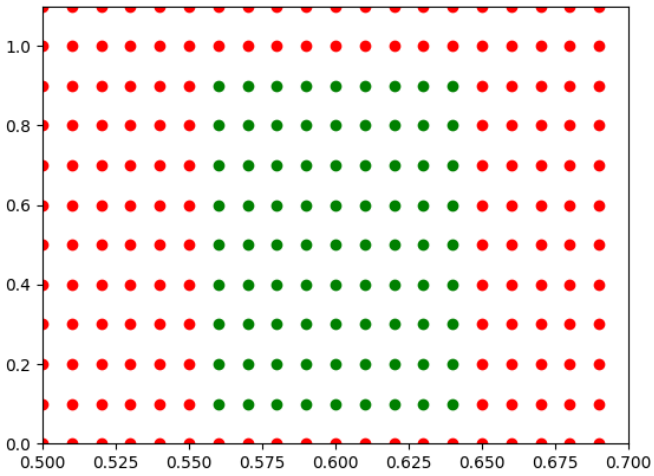


Figure 6: Coefficient values. Those yielding the same results as experimental data are shown in green. The X axis represents the `S` coefficient, and Y axis the `B`. The `F` coefficient is fixed at 1.

9.4 Applying the Drag Model to a Wireframe

With a `Swimmer` object successfully created based on the wireframe associated with an individual swimmer and a functional drag approximation model, the final step of determining the drag associated with an instance of the swimmer is extracting the necessary data from the `Swimmer` object to input into the drag approximation model. Having already determined the ratios between lengths of the swimmer's body by creating a wireframe, only one length must be manually measured. The left upper arm is thus measured from shoulder to elbow due to that measurement's ability to maximize comfort for the swimmer and minimize ambiguity in measurement points. With all lengths measured, widths are measured too for the limbs and torso of the swimmer. All wireframe lines are

assumed to have circular cross sections for ease of computing, with this being relatively true for limbs, and accounted for by other factors for the torso, such as the shoulders covering the frontal area of the chest. Measuring widths is extremely important, as it individualizes the data to account for different body types and proportions, a distinct advantage of a computational system. Most widths are measured sideways along the X axis, from left to right relative to the swimmer, with the exception of two. Chest is measured from front to back along the Z axis due to the shoulders covering the front cross sectional X axis width, and Z axis variation generally occurring most prominently in the center of the torso. Shoulders are additionally measured along the Z axis due to their position on the human body perpendicular to the torso. These measurements are taken in addition to the wireframe to create a cylinder based exterior model of the swimmer's form *(fig #)*. The entire swimmer's wireframe and modeled form are then adjusted to a frontal area of 200 for a more accurate output from the drag approximation model.

To determine cross sections to calculate frontal and rear pointiness, each cylinder associated with a wireframe line is assigned to one of three categories: vertical along the Y axis, horizontal perpendicular to the Y axis, or angled along both axes. For example, the chest cylinder is generally vertical, the shoulder cylinders horizontal, and arm cylinders angled. When taking cross sections perpendicular to the Y axis to approximate drag, vertical cylinders will yield a circle cross section, horizontal cylinders a rectangle, and angled cylinders an ellipse *(fig #)*. Vertical cylinders are defined as those corresponding to a wireframe line parallel to the Y axis. Horizontal cylinders are those whose difference in the y values of the endpoints of their corresponding line do not exceed the cylinder's diameter. Angled cylinders fit neither category. The cross section of vertical and angled cylinders is determined based on the wireframe initially to optimize runtime, while the cross sections of horizontal cylinders are dependent on the y value of the cross section, and thus must be calculated by the drag approximation program. When the drag approximation program queries a cross section, the `Swimmer` object's `getCrossSection()` method first determines horizontal cylinders that intersect the queried cross section and return the corresponding rectangles, then returns the ellipsis and circles generated by the intersection with vertical and angled cylinders, adjusted for the location in the cross section they would appear. The method then returns the collection of shapes making up the cross section as a `Multishape` in the Shapely polygon handling library *(fig. 7)*.

9.4.1 Code

```
#method to get cross section of swimmer object
def getCrossSection(self, height):
        #initialize all shapes of crosssection
        shapes = []
        #adjust height to shape height
        height = self.minheight + height
```

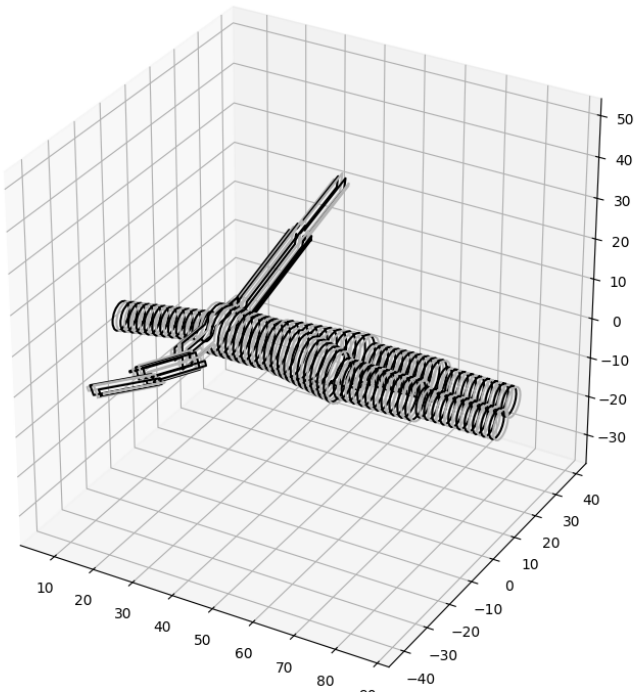


Figure 7: Cross sections taken from a wireframe of an improperly streamlined swimmer.

```
7           #iterate through lines of wireframe
8           for i in range(len(POSE_PAIRS)):
9               #get shape of cross section. The midpoint for
                ↪  horizontal cylinders, and the cross sectional
                ↪  shape otherwise
10              shape = self.xsections[i]
11              #get points of the line of wireframe
12              pair = POSE_PAIRS[i]
13              pointA = self.wireframe[pair[0]]
14              pointB = self.wireframe[pair[1]]
15              #get the diameter of the cylinder
16              diameter = diameters[i]
17              #if horizontal cylinder
18              if (type(shape) is list):
19                  #get distance of slice from midpoint
20                  #if less than radius add to shapes as rectangle
21                  xdistance = height - shape[0]
22                  ydistance = np.sqrt((diameter/2)*(diameter/2) -
                    ↪  xdistance*xdistance)
```

```
23                  if(ydistance < diameter/2):
24                      getPointsinRectangle(pointA, pointB)
25                      shapes.append(Polygon(points))
26              #if greater than radius add to shapes as ellipse at
                ↪  correct point
27              else:
28              #determine top and bottom point
29                  if(pointA[0] > pointB[0]):
30                      bottom = pointB
31                      top = pointA
32                  else:
33                      top=pointB
34                      bottom=pointA
35                  #insert ellipse if the cross section crosses the
                    ↪  line
36                  if(bottom[0] <= height <= top[0]):
37                      #determine at what fraction along the line
                        ↪  the cross section lies
38                      fraction = (height - bottom[0])/(top[0] -
                        ↪  bottom[0])
39                      #adjust x and z values of ellipse accordingly
40                      x = bottom[1]+(fraction*(top[1]-bottom[1]))
41                      z = bottom[2]+(fraction*(top[2]-bottom[2]))
42                      #move the ellipse to the determined center
                        ↪  and append
43                      shapetransformed = transform(shape, lambda a:
                        ↪  a + [x, z])
44                      shapes.append(shapetransformed)
45          #return shapes
46          return union_all(shapes)
```

To optimize computational runtime, all swimmer objects are assumed to have the same surface area, as lost surface area to adjacent touching limbs is minimal, and thus the arbitrary surface area value of 1 is returned.

10 Determining Optimal Position

10.1 Creating a Base Wireframe

Initially, a base wireframe is constructed using the lengths between points on the wireframe. The entire body is positioned flat on the XY plane, and the distance between points is used to construct an initial starting point for the wireframe with legs directly backwards along the Y axis and arms directly forwards *(fig #) (fig. 8)*. The purpose of this is twofold. First, constructing a base wireframe ensures that the optimization algorithm does not encounter any "valleys" of optimization. For example, despite pointed and slanted arms above the head

being a more hydrodynamic position, to achieve that position from arms at the side, a swimmer would need to pass their arms through a position with each arm extended to the side, less hydrodynamic than both the start and end positions. By starting with a base wireframe, we allow the use of an optimization algorithm that makes tweaks towards a final product, as opposed to testing every possible configuration of points, an incredibly computationally intensive task. Second, a tweak-based algorithm is incredibly susceptible to initial errors, which are not only unavoidable but exactly what we are seeking to address with our model. For example, a slightly upwards rotated left leg may influence a left arm to move upwards to address this error, especially if the left arm is tweaked before the leg. Consequently, the right arm may move downward to address left arm movement. Such an effect can quickly create undesired results, and a base wireframe addresses this. Within the `Swimmer` class, we define a `constructBaseWireframe()` method that returns a base wireframe based on the lengths taken from the original wireframe. This is done recursively by starting at the chest point, calculating distances, adjusting points accordingly, and performing a similar operation for each adjacent point, and each adjacent point after that.

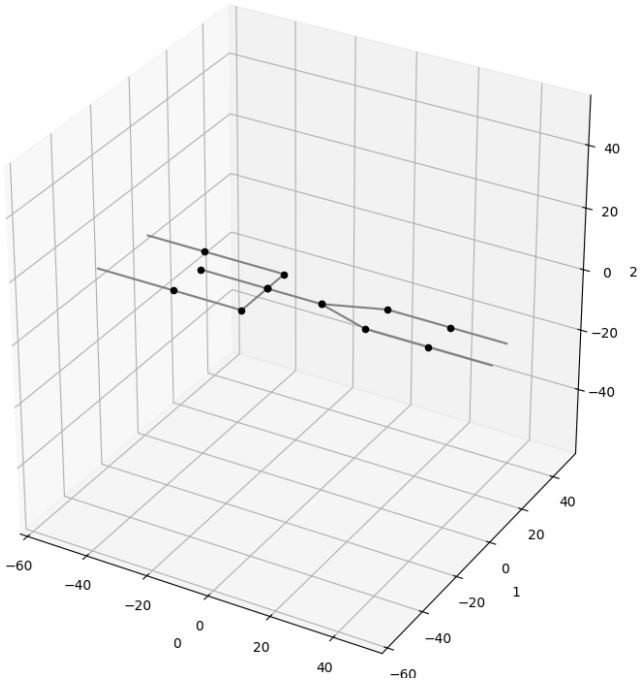


Figure 8: An example base wireframe.

10.2 Tweaking the Wireframe

Wireframe tweaking is accomplished using quaternions, mathematical operators for rotating vectors (Hughes, 2017). In general, they are a set of operations and specific rules based on the multiplication of hyper-complex numbers analogous to the i value that yield rotations of vectors. We use the PyQuaternion library to perform rotations of points due to the simplicity and ease of computation of quaternions. Within our `Swimmer` class, we define a `rotateWireframe()` method that takes the `Swimmer` object and rotates one point around another, subsequently rotating all connected points to maintain distance and allow rotation of entire limbs or sections. We wrote a constant `DEPENDANTS` tree that is used to determine the points that must be rotated with a point, and access those points recursively through a function.

10.2.1 Code

```
#method to rotate a section of wireframe. Basepoint is the point to rotate around, point2rotate the point to rotate, and y,x, and z the degrees of rotation
def rotateWireframe(self, basepoint, point2rotate, y,x,z):
        #retreive wireframe
        wireframe = self.wireframe.copy()
        #define axis for the quaternion
        yaxis = [1,0,0]
        xaxis = [0,1,0]
        zaxis = [0,0,1]
        #multiplication of quaternions results in a single quaternion with the same result as performing each rotation seperately
        quaternion = q.Quaternion(axis=yaxis,angle=np.deg2rad(y)) *q.Quaternion(axis=xaxis,angle=np.deg2rad(x)) * q.Quaternion(axis=zaxis,angle=np.deg2rad(z))
        #determine the points in the subsection that must be rotated along with point2rotate
        subsection = getSubsection(point2rotate)
        pointsInSubsection = getPointsInSubsection(subsection)
        #rotate both shoulder points together, and likewise with hips
        if(point2rotate == 2):

            pointsInSubsection.extend(getPointsInSubsection(getSubsection(5)))
        elif(point2rotate == 8):

            pointsInSubsection.extend(getPointsInSubsection(getSubsection(11)))
```

```
19          #rotate all points, first zeroing them around the
            ↪  basepoint, and then returning them to their original
            ↪  coordinates
20          for point in pointsInSubsection:
21              zeroedPoint = np.subtract(wireframe[point],
                ↪  wireframe[basepoint])
22              rotatedZeroedPoint = quaternion.rotate(zeroedPoint)
23              rotatedPoint = np.add(rotatedZeroedPoint,
                ↪  wireframe[basepoint])
24              wireframe[point] = rotatedPoint.tolist()
25          return wireframe
```

10.3 Optimization Function

To optimize a wireframe generated through computer vision into the hydrodynamically optimal wireframe, we include recursive iterations of the previously illustrated functions. Initially, we use the `constructBaseWireframe()` method and set the wireframe of the `Swimmer` object to the method's return value. Next, the optimization loop begins and is repeated until no change in wireframe is detected.

10.3.1 The Optimization Loop

One iteration of the optimization loop initially begins with the rotation of the sections connected to the chest point of the swimmer. First, each point connected directly to the chest point is rotated five degrees in either direction around the X axis and the magnitude of the calculated approximate drag is collected. The drag values are compared against each other and the initial value, and the most optimal result, yielding the lowest drag, is used. This process is repeated for rotation around the Z axis. Only two axis are chosen because any rotation of a line from an endpoint around the Y axis can be replicated by two turns around the X and Z axes. For each point rotated, the same process is recursively applied to each of their connected points, and so forth. In effect, the upper body and lower body are each rotated to their optimal position by five degrees, followed by the shoulders and hips, which are followed by the arms, legs, and head. Five degrees is chosen as the minimum value visible and correctable by a human swimmer, and is maximized to improve compute times. The optimization loop is then repeated.

10.3.2 Error Accounting

A few amends to the optimization function were made to account for common errors.

- The chest and shoulders are not rotated on the first pass through of the optimization loop. To create slant in the arms, a quality which is desirable through arm rotation, the algorithm would rotate the upper body an

arbitrary direction on the first pass through. Allowing the arms to rotate before the chest and shoulders ensures that arm slant is created through arm rotation, and preserves the ability to use a tweak-based optimization algorithm.

- The head is not optimized until after the rest of the body. As the arms rotated in towards a middle ground, the head was attempting to fill the space of the arms not yet rotated to cover the head, and was tilting drastically to the side.

10.3.3 Code

```
#function to optimize the wireframe
def minimizeDragNumber(swimmer : SC.Swimmer):
    #set the swimmer's wireframe to its base wireframe
    swimmer.setWireframe(swimmer.constructBaseWireframe())
    #initialize old wireframe and new wireframe to compare
    ↪  between iterations
    wireframeold = []
    wireframenew = swimmer.getWireframe()
    #conduct an initial improvement of the wireframe, excluding
    ↪  chest and shoulders. Each call of the improveWireframe
    ↪  method decrements the second argument by one. Due to the
    ↪  recursive nature of the function, the shoulders and chest
    ↪  are not optimized the first time through due to the
    ↪  positive second argument value
    improveWireframe(swimmer,2)
    #repeat iterations until no change is detected
    while(wireframenew != wireframeold):
        #print statement to see progress of optimization
        print(dN.getDragNumber(swimmer))
        wireframeold = wireframenew.copy()
        #the improve wireframe method tweaks each joint 5 degrees
        ↪  in the most optimal direction
        improveWireframe(swimmer,0)
        wireframenew = swimmer.getWireframe()
    return wireframenew
```

11 User Friendly Information Display

11.1 Wireframe Visualization

Lastly, we implemented a visualizer to relay the information to the user in a visually intelligible form. Both the three-dimensional wireframe taken from the swimmer's position and the hydrodynamically optimized wireframe are graphed, including the constituent points and the lines connecting them. The original

wireframe is graphed in red, and the optimized wireframe is graphed in green *(fig. 9)*.

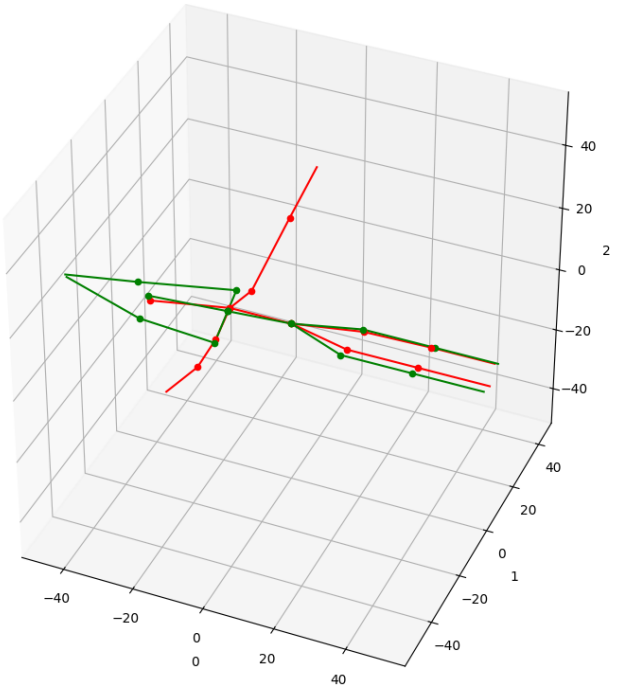


Figure 9: An example output.

Verifying the Efficacy of our Method

12 Criteria for an Optimal Streamline

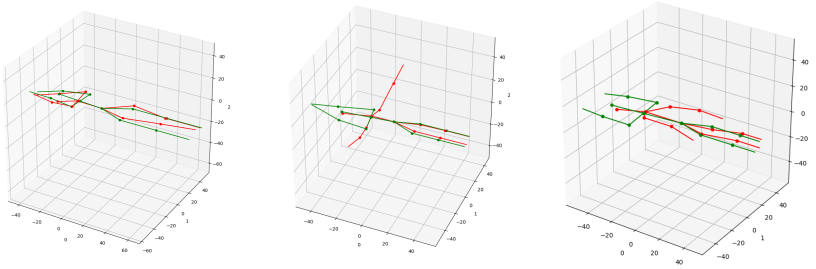
As previously outlined, an optimal streamline consists of arms above head, hands together and legs straight behind, the body forming a shape reminiscent of a torpedo or dolphin, a shape backed up by both engineering concepts and a multitude of years of evolution. Thus, we define criteria for an effective streamline accordingly. To determine the efficacy of our model, we compare results to the following set of criteria.

- The arms of the swimmer are outstretched above the head, and angled inwards towards the body's center line.
- The hands of the swimmer are connected or near connected, and the ends of the forearms are overlapping.

- The legs of the swimmer are stretched behind the swimmer, minimizing total frontal area within possible constraints of the human body.
- Frontal area appears to be minimized in all ways possible within the constraints of human form.

13 Verification of the Model

To verify the model, we provided 3 sets of images to the model and analyzed the outputs. To create a variety of test cases, we used two streamlines most commonly created due to errors, and one that was attempted to be correct *(fig. 10)*. Both optimized wireframe (a) and (b) fit the criteria above. Optimized



(a) (b) (c)

Figure 10: (a) A test case for a correct streamline (b) A test case for an incorrect streamline (c) A test case for an incorrect streamline with issues that will be discussed in the next section

wireframe (c), however, fails many of the criteria, but yet surprisingly still represents a successful algorithm, albeit highlighting errors in data collection.

13.0.1 Wireframe Optimization Inaccuracies

It is visible that optimized wireframe (c) does not fit the criteria of hands connected or near connected, and this issue mainly arises from data collection accuracy issues, further discussed in Section 14. The arms in this figure are collected as shorter than the other figures, while the shoulders are collected as wider. The main difference however, is that the shoulders were relaxed down, as opposed to hunched upwards as in wireframe (a) and (b). The image recognition software is unable to recognized hunched upwards shoulders as a change in the angle between shoulder center and the end of each shoulder, and just represents these intricacies as wider or skinnier shoulders, incorrectly ascribed intrinsic to the person. Therefore, the model neglects the ability to hunch shoulders upwards as a consequence of data collection limitations. The drag model, thus, determines that due to the short nature of the arms without shoulder movement coupled with wide shoulders, the drag minimization derived from moving arms

towards a center line is actually sub optimal. Shoulders have a natural slope to them, and moving the arms towards the center line would increase the drag on the arms by causing a flatter surface. The drag model has determined that by covering some percentage of the shoulders by a slope in the arms while leaving some shoulder exposed to the flow, drag is actually optimized *(fig. 11)*. Improvements to data collection as well as improvements to defining the shape and limitations of the human body would improve such an issue. Additionally, this highlights an interesting specific use case for the model.

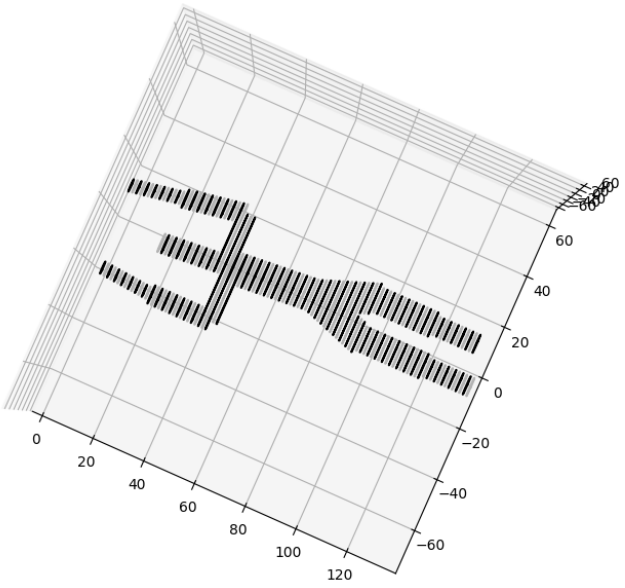


Figure 11: Cross sections of optimized wireframe (c). Note the slope of the shoulders and slope of the arms.

13.0.2 Ability of the Model to Account for Differences and Disabilities

Take a swimmer who had mobility issues due to a prior injury or surgery. Additionally, take a swimmer who had birth differences leading to decreased mobility and different body proportions. Scenarios such as these could easily be accounted for by our model. Wireframe (c) inadvertently could represent a person with an injured latissimus dorsi, the main muscle along a person's back, or rotatory cuff, the area of tendons around the shoulder, who is unable to lift their shoulders. Aforementioned fixes to the model could increase its ability to

accurately assist individuals without movement issues, but the reintroduction of restrictions could even further individualize the model for individuals with limitations. Wireframe (c) demonstrates that drastically different results are optimal for individuals with differences. Shoulder rotation limitations could be introduced to account for injuries. Differences in body proportions will be picked up by the computer vision algorithm. Individuals who could have previously struggled with coaching individualized to their needs would receive personalized results computationally by CHASM.

Finalizing Thoughts

14 Data Collection Improvement Needs

As previously stated, the CHASM model is incredibly susceptible to irregularities in position and wireframe. This is a desired quality, as such irregularities influence the position of an optimal hydrodynamic wireframe, and enforce the need for individualized computation as opposed to comparing to a universal baseline. This sensitivity, however, leaves the model highly sensitive to data collection inaccuracies. In our physical LVU, imperfections in measurement could extrapolate to noticeable inaccuracies. A better production method for the LVU than what was accessible to us using household power tools would improve data collection and address many of these inaccuracies, which included egregious differentiation between widths of right and left shoulders, and inaccuracies in point location due to improperly angled cameras.

15 Further Steps and Project Potential

With the production of a more reliable LVU, data collection accuracy would drastically improve. Further steps in improving computational accuracy could be taken by upgrading the wireframe model to a surface based model of the swimmer's form. This would allow the computation of more accurate drag, as well as determining forces added to the water, opening the doors to aspects of the sport of swimming other than the streamline. Freestyle optimization would be a logical, yet difficult, next step, and would likely require the use of more complicated hardware and software in order to collect accurate data for a body in relative motion to itself and account for forces added against the water. Additionally, introduction of restrictions to movement could specialize the model towards individuals with injuries or disabilities. With improvements, the steps and algorithms outlined above could provide coaching augmentation and replacement for almost all aspects of swimming, increasing accessibility and productivity of the sport.

16 Conclusion

As the project currently stands, CHASM provides visual feedback on the streamline portion of swimming. It can improve the ability of coaches to give meaningful feedback to their swimmers, and increase accessibility to a sport that involves paywalls due to coaching access. CHASM uses computer vision, mathematical coordinate operations, a drag approximation model, optimization functions, and visual output to accurately and effectively communicate feedback on a swimmer's streamline position. With improvement, CHASM could be expanded to provide feedback on all aspects of swimming, providing an invaluable coaching aid to augment the needs of competitive and recreational lap swimmers of all levels, and completely replace a coach for those who do not have the means to have one. Additionally, access to personalized coaching for individuals with disabilities or injuries that effect their body or abilities could exponentially increase. Adoption of CHASM, as it stands or as it could be, by swimming pools, coaching teams, individuals, and schools could prove an asset to creating greater depth of field and accessibility in the sport, all the while improving the form of preexisting participants. CHASM is a means to accessibility and inclusion for underprivileged, injured, and disabled individuals, effective coaching for swimmers of all levels, and speed maximization for the sport of competitive lap swimming.

17 Acknowledgements

Invaluable in our coding, building and development process were our wonderful sponsors Ms. Masoni and Ms. Comstock. Without their assistance, not only would we not have a designated time during the school day to work on our project, but we would not have the guidance and experience they bring to the table. Additionally, we would like to acknowledge Marlow Lichty, a prior team member from earlier in the year who elected to leave our team to invest his time elsewhere, but nonetheless contributed important work in the implementation of the image processing software.

18 Works Cited

References

[1] 360swim.com, C. @. (n.d.). Streamline explained (how do drag forces influence my body in swimming?). 360swim. https://360swim.com/blog/streamline-explained-how-forces-influence-swimming#google_vignette

[2] Admin, F. S. (2021, December 13). Why streamline position is so important. Precision Swim Training. https://www.precisionswimtraining.com/blogs/kicking-with-kaitlin/why-streamline-is-so-important

[3] Andriluka, M., Pishchulin, L., Gehler, P., & Bernt, S. (2014). 2D Human Pose Estimation: New Benchmark and State of the Art Analysis. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[4] Dougherty, A. (2017, June 14). A cost benefit analysis of a lifelong swimming career. Swimming World News. https://www.swimmingworldmagazine.com/news/a-cost-benefit-analysis-of-a-lifelong-swimming-career/

[5] Drag coefficient. (2023). Retrieved from https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/drag-coefficient-2/

[6] Gupta, V. (2018, May 29). Deep Learning based Human Pose Estimation using OpenCV. LearnOpenCV. Retrieved April 2, 2024, from https://learnopencv.com/deep-learning-based-human-pose-estimation-using-opencv-cpp-python/

[7] Here is how swimmers of all levels train (according to coaches). A3 Performance. (n.d.). https://www.a3performance.com/blogs/a3-performance/here-is-how-swimmers-of-all-levels-train-according-to-coaches

[8] Holmes, T. (2022). How to have perfect streamline in swimming. Retrieved from https://blog.myswimpro.com/2022/02/08/how-to-have-perfect-streamline-in-swimming/

[9] Hughes, M. (2017). Don't get lost in Deep space: Understanding quaternions - technical articles. Retrieved from https://www.allaboutcircuits.com/technical-articles/dont-get-lost-in-deep-space-understanding-quaternions/

[10] Koury, J. M. (2024, March 25). Individual coaching for competitive swimmers - swim technique coaching Lehigh Valley - swim articles. Swim Technique Coaching Lehigh Valley. https://www.swimtechniquecoaching.com/swim-articles/2023110private-coaching-for-competitive-swimming-athletes

[11] Laminar and turbulent flow. (1992). In (Ed.), DOE Fundamentals Handbook: Thermodynamics, Heat Transfer, and Fluid Flow. U.S. Department of Energy. https://engineeringlibrary.org/reference/laminar-and-turbulent-fluid-flow-doe-handbook

[12] OpenStax College. (n.d.). Motion of an Object in a Viscous Fluid. Lumen Learning. Retrieved April 2, 2024, from https://courses.lumenlearning.com/suny-physics/chapter/12-6-motion-of-an-object-in-a-viscous-fluid/

[13] Poirier-Leroy, O. (2023, December 19). How to streamline in swimming like a pro (swim faster and glide farther). SwimSwam. https://swimswam.com/streamline-in-swimming/

[14] Table 10.5 presents the drag coefficient, CD for numerous... (1 answer). (2022). Retrieved from https://www.transtutors.com/questions/10-23bg-table-10-5-presents-the-drag-coefficient-cd-for-numerous-geometric-shapes-at-6923663.html

[15] The importance of a good streamline. SAN. (n.d.). https://www.swimacademynetwork.com.au/resources/the-importance-of-a-good-streamline

[16] Ungerechts, Bodo. (1982). The Validity of the Reynolds Number for Swimming Bodies Which Change Form Periodically.

[17] What is drag? (2022). Retrieved from https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/what-is-drag/

[18] Wise, J. (2022). Role of technology in swimming: The good and bad. Retrieved from https://www.swimmingworldmagazine.com/news/role-of-technology-in-swimming-the-good-and-bad/

[19] Wynn, K. (n.d.). Welcome. Retrieved from https://kieranwynn.github.io/pyquaternion/