LIQUID PROPELLANT POSITIONING AND CONTROL IN EXAMPLE PROPELLANT TANK

by

Logan Walters

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Aeronautics and Astronautics

School of Aeronautics and Astronautics

West Lafayette, Indiana

December 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF COMMITTEE APPROVAL

Dr. Steven Collicott, Chair

School of Aeronautics and Astronautics

Dr. Haifeng Wang

School of Aeronautics and Astronautics

Dr. Timothee Pourpoint

School of Aeronautics and Astronautics

Approved by:

Dr. Gregory Blaisdell

Dedicated to my parents Tammy and Richard and sister Aada for their continual support

ACKNOWLEDGMENTS

This research was funded as part of a Teaching Assistantship offered by Purdue University's Gambaro Graduate Program of Aeronautics and Astronautics. The Rotational Slosh project is funded by the grant "Spacecraft Pointing Control and Zero-Gravity Slosh" from NASA Flight Opportunities (sponsor award number: 80NSSC20K0097). The Rotational Slosh project also features contributions from Purdue students taking AAE 418: Zero-Gravity Flight Experiment.

I would also like to thank my thesis advisor Dr. Collicott for the opportunity to work on this project. His assistance and guidance over the past year and a half while I have completed this research has been invaluable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Two topics relating to low gravity fluid behavior in satellite propellant tanks are considered. In the first, static case, the problem of liquid trapping is examined. Satellite propellant tank end caps optimized for weight are generally shallower and more oblate than hemispherical end caps of the same radius. However, these shallower end caps pose an interesting challenge for propellant management. In the absence of vanes, it is possible for liquid propellant to be trapped in the tank and become unusable. Understanding of how propellant tends to distribute itself in the bare, vaneless tank can be used to drive vane design to counteract these tendencies and ensure propellant remains where desired. The first section of this thesis aims to demonstrate methods that can be used to identify when, how, and why liquid trapping occurs in a given tank geometry. A fluid statics code called Surface Evolver is used to calculate possible fluid configurations for different propellant volumes, contact angles, and end cap designs. The specific case of a cylindrical tank with 2:1 ellipsoidal end caps is studied extensively for ranges of fill fractions and contact angles to illustrate the methods used. Results are computed for each possible propellant configuration: a spherical liquid-gas interface, an asymmetric liquid-gas interface, and a liquid ring. Analytical solutions are found and compared against Surface Evolver results for the spherical liquid-gas interface and liquid ring, showing excellent agreement. Results are also found for other aspect ratio ellipsoidal end caps, superellipsoidal end caps, and torispherical end caps. Each non-hemispherical dome design is found to be able to trap liquid away from the axis of the tank regardless of contact angle. The second part of this thesis, focusing on the dynamic case, details the development of an experimental payload designed to fly on Virgin Galactic's SpaceShipTwo. This experiment is designed to obtain data on sloshing behavior of liquids in microgravity in response to rotation. The payload contains eight scaled down propellant tanks that are rotated while in microgravity, and the resulting slosh is recorded by video cameras inside the payload. The video will be analyzed after the experiment to extract data on damping rates and potentially positional data of the liquid-gas interface. The impact of constraints on the design of the overall experiment are discussed. The purpose of each component in the experiment is explained and justified relative to the design constraints. The remaining work that must be completed before flight on SpaceShipTwo is reviewed, highlighting the most significant unknowns.

1. INTRODUCTION

Often, it is desired to control the motion and position of propellant inside satellite propellant tanks while in orbit. However, propellant behavior can be difficult to predict and model in low gravity. The first half of this thesis examines the static behavior of liquid propellants in low gravity through the lens of the problem of liquid trapping. The second half of this thesis details the design of an experimental payload that will fly on Virgin Galactic's SpaceShipTwo to collect data on rotational sloshing of propellant in satellite tanks to improve modeling capabilities for the dynamic case.

The liquid trapping problem describes a way in which liquid can become trapped and inaccessible when draining liquid propellant from a propellant tank while in low gravity. If liquid trapping occurs on a satellite, the satellite's operable life may be cut short by preventing the trapped propellant from being accessed for course correction, attitude adjustment, or any other required maneuver. Liquid trapping can be avoided by using propellant management devices (PMDs) like vanes. Vanes use the surface tension of the liquid propellant to control the propellant distribution while in zero gravity. In zero-g, liquid tends to wick into corners, so vanes can be strategically added to create corners near where the liquid needs to be. This makes vanes a particularly attractive option for most applications since they have no moving parts and require no energy input.

However, to design vanes capable of controlling the propellant distribution, it is first necessary to understand the fluid behavior inside the bare, vaneless tank. This can be used to identify which problems the vanes need to solve. Without vanes, the propellant distributions are heavily dependent on the shape of the end cap. For a hemispherical end cap, characterization of propellant distributions is easy, and no liquid trapping can occur. These hemispherical end caps are not generally the minimum weight solutions, however. Depending on the mission specifications, other geometries, like ellipsoidal or torispherical end caps, can save a significant amount of weight over hemispherical domes (Bert & Hyler, 1966). As a result, most satellite tanks use non-hemispherical end caps to store propellant, allowing for the possibility of liquid trapping. Modeling of the conditions under which liquid trapping occurs can be used to help drive vane design.

The second half of this thesis considers the problem of rotational slosh and how data can best be collected to provide better understanding of and modeling capabilities for liquid propellant

behavior in these environments. Satellites are sometimes required to undergo pointing maneuvers while in orbit, such as for earth imaging. During this process, the satellite changes orientation, rotating the propellant tank along with it. This induces sloshing motion in the propellant stored within, which can cause unwanted side loads that may change the satellite's orientation. In such cases, additional fuel must be spent to correct the error in orientation. This wasted fuel can be saved and pointing accuracy improved by better understanding the dynamic sloshing of the propellant in the satellite tank in response to external forces and incorporating this understanding into control algorithms. However, modeling of low gravity propellant slosh is very complex. Analytical solutions have very limited practicality due to the strong simplifying assumptions required. Computational Fluid Dynamics (CFD) can be used to obtain acceptable results, capturing general trends when compared to experiment but losing some of the fine detail of propellant behavior. To improve performance, more experimental data is needed to calibrate models. Much of the existing experimental slosh data has focused on the problem of sloshing in response to linear movement, but little slosh data exists for tanks undergoing rotation.

An experimental payload is designed to fly on Virgin Galactic's SpaceShipTwo to help fill this knowledge gap and provide initial data for rotational sloshing of liquid propellants. The experiment will contain eight different tanks. Four of them will contain a fluid with contact angle near 0°, while the other four will contain a fluid with contact angle greater than 20°. Each set of tanks will have a different liquid volume to allow investigation into how fill fraction influences propellant slosh. The tanks are rotated 45° about an axis perpendicular to the longitudinal axis while in microgravity, inducing slosh. The liquid response is recorded using video cameras on board the payload, from which damping rates will be extracted. In addition, it may be possible to extract the 3D position of the liquid-gas interface. Both can then be used to serve as points of comparison for future analysis.

2. LIQUID TRAPPING INTRODUCTION

2.1 Zero Gravity Fluid Mechanics

A quasi-static fluids approach may be used to analyze liquid trapping. One way to find propellant configurations in the tank is to find the liquid-gas interface that forms a surface of constant mean curvature intersecting the wall at the appropriate contact angle and encloses the liquid volume. Liquid volumes are generally expressed relative to the total tank volume as the fill fraction. However, when considering a single tank geometry of fixed size, liquid volume can be used interchangeably with fill fraction. While this approach can be used to find analytical solutions for certain cases, an energy minimization approach is generally simpler to work with. Assuming there is no gravity or other acceleration so Bond number is zero, that the tank is perfectly rigid and smooth, and that the contact angle is uniform on the tank wall, the relevant nondimensional energy to minimize can be expressed in terms of the non-dimensional free surface area A_{FS}, which is the surface area of the liquid gas interface, the non-dimensional wetted area A_w, which is the area of the wall wetted by the liquid, and contact angle θ.

$$E = A_{FS} - \cos\theta\, A_w + const \tag{2.1}$$

Here, *const* is a constant that depends only on the contact angle and tank shape and size. This constant can be ignored since it does not change during the energy minimization process. Assuming there is no phase change so the liquid volume remains constant, minimization of this energy while conserving volume will yield static equilibria solutions for the propellant configurations. This energy minimization process can be performed numerically with a scalar minimization code called Surface Evolver (Brakke, 2013). After defining how the free surface relates to the tank geometry, Surface Evolver can be used to iteratively deform the free surface to minimize energy while conserving volume at each step.

It is assumed that the liquid exists as a single volume that is at least partially touching one end cap but is not touching both end caps. When filling and draining is discussed, a slow, quasi-static process is assumed such that the dynamic fluid motion can be ignored. When discussing liquid trapping, it is assumed that the outlet is small and located at the center of the end cap.

For simplicity, a tank of radius $R = 1$ is chosen, though results can be easily scaled to different tank sizes. The fill fraction can be expressed in terms of liquid volume V_{liq}, tank volume V_{tank}, end cap volume V_{dome}, tank radius R, and cylinder length L as

$$f = \frac{V_{liq}}{V_{tank}} = \frac{V_{liq}}{\pi R^2 L + 2V_{dome}} \tag{2.2}$$

Since the propellant volume is isolated at one end of the tank, the length of the cylindrical section can be selected arbitrarily to be twice the tank radius, as shown in Figure 2.1. Fill fractions can be scaled to other selections for cylinder length by multiplying by the tank volume for the assumed cylinder length to obtain the liquid volume, then normalizing by the tank volume corresponding to the new cylinder length.

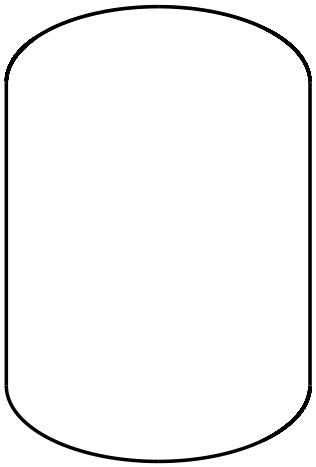

Figure 2.1. A tank that has a cylindrical section with length equal to two tank radii. In this example, the domes are 2:1 ellipsoidal end caps.

3. LIQUID TRAPPING METHODS

One dome design that is commonly used to reduce weight is a torispherical end cap, which consists of a spherical head with radius larger than the tank radius, connected to the tank cylinder by a toroidal "knuckle". However, this dome geometry can be difficult to model in Surface Evolver. The mean curvature, which is a significant contributor to the behavior of fluids on a surface, has two discontinuities, one where the dome meets the knuckle and another where the knuckle meets the cylinder. The contact line may get stuck or otherwise behave strangely when moving across these discontinuities, and so special care must be taken when running the Surface Evolver code.

Another dome design that is used in satellite propellant tanks is an ellipsoidal end cap, created by revolving an ellipse about the longitudinal axis. Ellipsoidal end caps have continuous mean curvature on the end cap itself, though a discontinuity in mean curvature still exists at the junction. This overall improvement in continuity of mean curvature, combined with the simpler equations used to define the geometry, can make ellipsoidal end caps easier to model in Surface Evolver than torispherical end caps. Importantly, the ellipsoidal end cap can smoothly morph into a hemispherical end cap by letting the end cap depth be equal to the tank radius R. This makes it serve as a good point of comparison against the far simpler to model hemispherical end cap design.

The model of an ellipsoidal end cap can also be extended to capture the effect of tighter corners near the junction by considering a superellipsoidal end cap. This type of geometry replaces the exponent of 2 used in the equation for an ellipse with a parameter $n > 2$ and revolving about an axis, creating a tighter corner at the junction. This can be used to emulate the effects of a small knuckle radius on a torispherical dome by introducing a high curvature region at the junction. Like the ellipsoidal end cap, a superellipsoidal end cap has continuous mean curvature on the dome, but it also has continuous mean curvature at the junction between the end cap and cylinder. As a result, a superellipsoidal end cap has no discontinuities in mean curvature anywhere. Despite this, the mean curvature does change significantly around the junction, especially for high values of n.

Examples of each end cap geometry are shown in Figure 3.1, namely a hemispherical end cap, an ellipsoidal end cap with minor axis half the major axis (a 2:1 ellipsoid), a 2:1 superellipsoidal end cap with exponent $n = 3$, and a torispherical end cap with overall depth half the tank radius and knuckle radius 60% of depth, corresponding to a dome radius of 1.625 R and a knuckle radius of 0.3 R. The mean curvature for each of these dome geometries is plotted in

Figure 3.2, showing the location of discontinuities in mean curvature for each dome geometry. All dimensions are nondimensionalized by the tank radius, including the mean curvature.

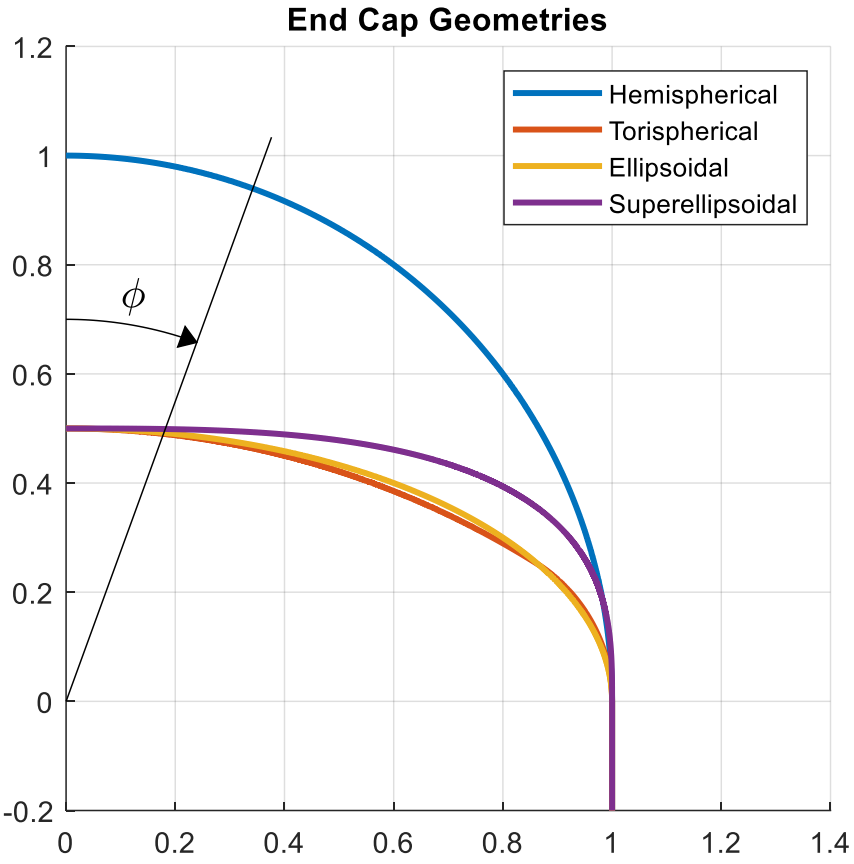

Figure 3.1. Example hemispherical, torispherical, ellipsoidal, and superellipsoidal end caps with azimuthal angle ϕ.

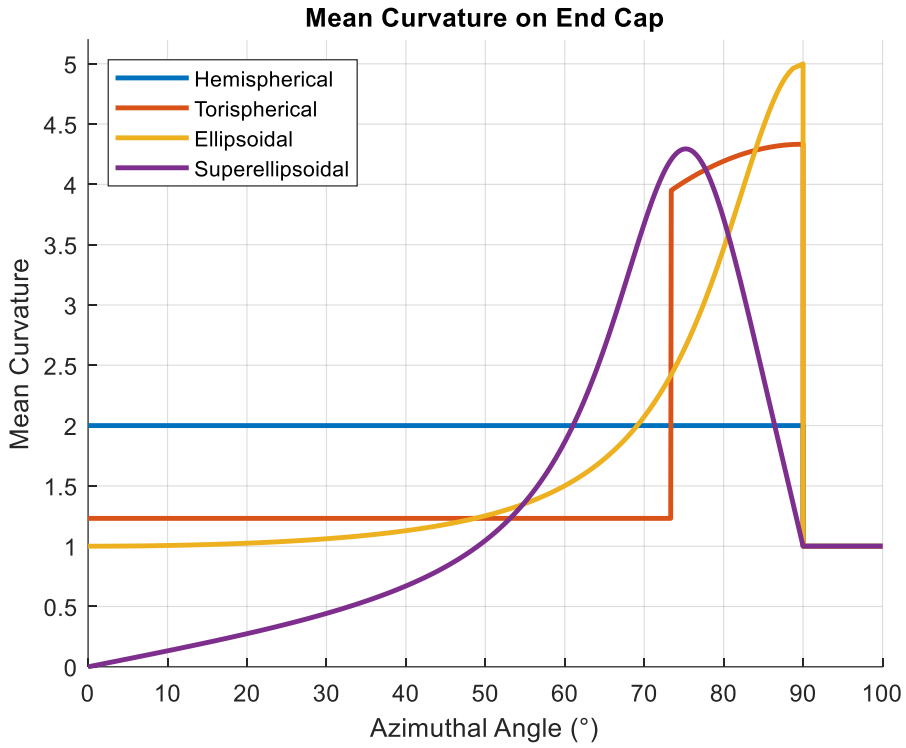

Figure 3.2. Mean curvature of the example hemispherical, torispherical, ellipsoidal, and superellipsoidal end caps is plotted against azimuthal angle ϕ.

3.1 Surface Evolver File Setup

The coordinate system used in all Surface Evolver models presented places the z-axis along the tank's axis of rotation, with the z = 0 plane passing through the junction between the end cap and cylinder. The end cap is above the z = 0 plane, and the cylindrical section is below this plane. Before converging, the free surface is initialized to a simple geometry in the Surface Evolver model. When considering droplet-like solutions, the free surface is initialized as a flat square on the x-y plane. The vertices and contact line are constrained to the tank wall, and all facets are constrained to be inside the tank. For ring-like solutions, the free surface is initialized as a truncated pyramid that omits the top and bottom faces. All initial vertices and both contact lines are constrained to the tank wall, and all facets and internal edges are constrained to be inside the tank. After converging, between 7,500 and 15,000 facets are used for most models, depending on the liquid volume and contact angle. For some cases, particularly those with low volume and contact angle, facet counts as high as 300,000 facets are required to achieve convergence.

When setting up the Surface Evolver model, an equation is needed to define the tank geometry so the contact line can stay coincident with the tank wall. When z < 0, the contact line is on the cylinder, so the contact line constraint equation is

$$x^2 + y^2 = R^2, \quad z < 0 \tag{3.1}$$

where R is the tank radius. When z > 0, the contact line is on the dome, and so the equation will depend on the end cap geometry. The formulas for a hemispherical end cap, ellipsoidal end cap, and superellipsoidal end cap are listed below, where z_0 is the depth of an ellipsoidal or superellipsoidal end cap and n is the exponent for a superellipsoidal end cap.

Hemisphere: $$x^2 + y^2 + z^2 = R^2, \quad z > 0 \tag{3.2}$$

Ellipsoid: $$\left(\frac{x}{R}\right)^2 + \left(\frac{y}{R}\right)^2 + \left(\frac{z}{z_0}\right)^2 = 1, \quad z > 0 \tag{3.3}$$

Superellipsoid: $$\left[\left(\frac{x}{R}\right)^2 + \left(\frac{y}{R}\right)^2\right]^{\frac{n}{2}} + \left|\frac{z}{z_0}\right|^n = 1, \quad z > 0 \tag{3.4}$$

To make comparison with other tank geometries easier, the torispherical end cap is defined by the end cap depth z_0 and knuckle radius r_N, expressed as a percentage of the end cap depth. However, it is more convenient to work with the actual knuckle radius $r_s = r_N z_0$. In addition, other intermediate variables convenient for the torispherical end caps are

$$R_H = \frac{2r_s R - R^2 - {z_0}^2}{2(r_s - z_0)} \tag{3.5}$$

$$z_{center} = -\sqrt{{R_H}^2 - R^2 + 2Rr_s - 2R_H r_s} \tag{3.6}$$

$$z_{cr} = \frac{r_s\sqrt{(R_H - R)(R_H - 2r_s + R)}}{R_H - r_s} \tag{3.7}$$

$$r_{cr} = \sqrt{{r_s}^2 - {z_{cr}}^2} + R - r_s \tag{3.8}$$

Using these variables, the equation of the torispherical end cap is

Torispherical: $$\begin{cases} x^2 + y^2 + (z - z_{center})^2 = {R_H}^2, & z > z_{cr} \\ \sqrt{{r_s}^2 - z^2} + R - r_s - \sqrt{x^2 + y^2} = 0, & 0 < z < z_{cr} \end{cases} \tag{3.9}$$

For Surface Evolver to compute the liquid volume, a slightly modified version of Surface Evolver's default volume method can be used. This method uses a facet vector integral applied to the free surface with components

$$\begin{aligned} q_1 &= 0 \\ q_2 &= 0 \\ q_3 &= f(x,y) - z \end{aligned} \tag{3.10}$$

where $f(x,y)$ is the height of the dome above the x-y plane, which depends on the end cap geometry, determined by solving the tank geometry equations for z.

Hemisphere: $$f(x,y) = \sqrt{R^2 - x^2 - y^2} \tag{3.11}$$

Ellipsoid: $$f(x,y) = \frac{z_0}{R}\sqrt{R^2 - x^2 - y^2} \tag{3.12}$$

Superellipsoid: $$f(x,y) = z_0\left[1 - \left(\left(\frac{x}{R}\right)^2 + \left(\frac{y}{R}\right)^2\right)^{\frac{n}{2}}\right]^{\frac{1}{n}} \tag{3.13}$$

Torispherical: $$f(x,y) = \begin{cases} \sqrt{{R_H}^2 - x^2 - y^2} + z_{center}, & \sqrt{x^2+y^2} < r_{cr} \\ \sqrt{{r_s}^2 - \left(\sqrt{x^2+y^2} - R + r_s\right)^2}, & \sqrt{x^2+y^2} > r_{cr} \end{cases} \tag{3.14}$$

Finally, to compute the total energy, equation 3.15 can be used.

$$E = A_{FS} - \cos\theta\, A_W \tag{3.15}$$

The free surface area is computed automatically by Surface Evolver. The wetted area can be computed by integrating an edge vector integral along the contact line, and this vector field depends on the geometry of the tank the contact line is touching. On the cylindrical section of the tank, the vector field to obtain the wetted area is

$$\begin{aligned} q_1 &= \frac{-y}{x^2+y^2}\left(\frac{S_{end\ cap}}{2\pi} - Rz\right) \qquad z < 0 \\ q_2 &= \frac{x}{x^2+y^2}\left(\frac{S_{end\ cap}}{2\pi} - Rz\right) \qquad z < 0 \\ q_3 &= 0 \end{aligned} \tag{3.16}$$

On the dome, the vector field is

$$\begin{aligned} q_1 &= \frac{-y}{x^2+y^2}\left(\frac{S_{end\ cap}}{2\pi} - I(z)\right) \qquad z > 0 \\ q_2 &= \frac{x}{x^2+y^2}\left(\frac{S_{end\ cap}}{2\pi} - I(z)\right) \qquad z > 0 \\ q_3 &= 0 \end{aligned} \tag{3.17}$$

where $S_{end\ cap}$ is the total surface area of the end cap and $I(z)$ is a function that depends on the geometry of the end cap. For each end cap, these correspond to

Hemisphere: $$\frac{S_{end\ cap}}{2\pi} = R^2 \tag{3.18}$$

$$I(z) = Rz \tag{3.19}$$

Ellipsoidal: $$\frac{S_{end\ cap}}{2\pi} = \frac{R}{2}\left(R + \frac{\sinh^{-1}(bz_0)}{b}\right) \tag{3.20}$$

$$b^2 = \frac{R^2 - {z_0}^2}{{z_0}^4}, \qquad I(z) = R\left(\frac{z}{2}\sqrt{1+b^2z^2} + \frac{\sinh^{-1}(bz)}{2b}\right) \tag{3.21}$$

Superellipsoidal: $$\frac{S_{end\ cap}}{2\pi} = R\, I(z_0) \tag{3.22}$$

$$I(z) = R\int_0^z \left(1 - \left(\frac{z}{z_0}\right)^n\right)^{1/n} \sqrt{1 + \left(\frac{R}{z_0}\right)^2 \left(\frac{z}{z_0}\right)^{2n-2} \left(1 - \left(\frac{z}{z_0}\right)^n\right)^{\frac{2}{n}-2}}\, dz \tag{3.23}$$

Torispherical: $$\frac{S_{end\ cap}}{2\pi} = r_s\left[z_{cr} + (R - r_s)\sin^{-1}\left(\frac{z_{cr}}{r_s}\right)\right] + R_H(z_0 - z_{cr}) \tag{3.24}$$

$$I(z) = \begin{cases} r_s\left[z + (R - r_s)\sin^{-1}\left(\frac{z}{r_s}\right)\right] & z < z_{cr} \\ r_s\left[z_{cr} + (R - r_s)\sin^{-1}\left(\frac{z_{cr}}{r_s}\right)\right] + R_H(z - z_{cr}) & z \geq z_{cr} \end{cases} \tag{3.25}$$

For a derivation showing how the wetted energy formulas are obtained, see Appendix B. Note that for the case of the superellipsoidal dome, $I(z)$ is left in terms of an integral since there is no analytical solution to this integral. Instead, this integral is approximated using a cubic spline. First, pulling out constants, normalizing z as $\bar{z} = \frac{z}{z_0}$, and defining $AR = \left(\frac{R}{z_0}\right)^2$ to obtain an easier integral to work with,

$$I'(\bar{z}) = \frac{I(\bar{z}\, z_0)}{R z_0} = \int_0^{\bar{z}} (1-\bar{z}^n)^{1/n} \sqrt{1 + AR\, \bar{z}^{2n-2} (1-\bar{z}^n)^{\frac{2}{n}-2}}\, d\bar{z} \tag{3.26}$$

Since small changes in z result in large changes to the integrand of $I(z)$ near the top of the dome, the integral can be equivalently expressed in terms of r by the substitution

$$z = z_0 \left[1 - \left(\left(\frac{x}{R}\right)^2 + \left(\frac{y}{R}\right)^2 \right)^{\frac{n}{2}} \right]^{\frac{1}{n}} \tag{3.27}$$

resulting in

$$I'(r) = \int_0^r r \sqrt{1 + AR(1-r^n)^{\left(2-\frac{2}{n}\right)} r^{(2-2n)}}\; dr \tag{3.28}$$

However, this integral suffers from a similar problem near the junction, so each integrand is used in the range of values where they are most accurate. The point at which this switch takes place is $(r_{cr}, \bar{z}_{cr}) = \left(2^{-\frac{1}{n}}, 2^{-\frac{1}{n}}\right)$. The equations used to define the cubic spline representing $\tilde{I}(\bar{z})$ and $\tilde{I}(r)$ are

$$I'(\bar{z}) = a_i^{\bar{z}} t^3 + b_i^{\bar{z}} t^2 + c_i^{\bar{z}} t + d_i^{\bar{z}} \qquad \frac{z}{z_0} < \bar{z}_{cr} \tag{3.29}$$

$$t = z \bmod \frac{z_0\, \bar{z}_{cr}}{N^{\bar{z}} - 1} \tag{3.30}$$

$$i = \text{ceil}\left(\frac{N^{\bar{z}} - 1}{z_0\, \bar{z}_{cr}}\, z \right) \tag{3.31}$$

$$I'(r) = a_i^r t^3 + b_i^r t^2 + c_i^r t + d_i^r \qquad r > r_{cr} \tag{3.32}$$

$$t = r \bmod \frac{r_{cr}}{N^r - 1} \tag{3.33}$$

$$i = \text{ceil}\left(\frac{N^r - 1}{r_{cr}}\, r \right) \tag{3.34}$$

where ceil(x) is the ceiling function, the coefficients a_i, b_i, c_i, and d_i can be computed using the MATLAB script found in Appendix C, the variable t ranges between 0 and 1 for each segment of the spline, and the variable i is the index of the coefficient for that segment. For details of this approximation, see Appendix B.

4. LIQUID TRAPPING ANALYSIS

For bare, axisymmetric tanks with these end cap geometries, there are three possible types of propellant configurations: a spherical interface, a liquid ring, and an asymmetric droplet. The first and simplest of these configurations is a spherical interface where the liquid forms a single axisymmetric droplet. This results in the free surface of the liquid taking the form of a section of a sphere. Configurations of this type are quite easy to compute analytically. For a given contact line height, the necessary radius of the spherical interface can be computed to ensure the interface intersects with the wall (either cylinder or end cap) at the appropriate contact angle. Once the exact shape of the interface is known, it is a relatively simple geometry problem to compute the liquid volume, free surface area, and wetted area, from which fill fraction and energy can be computed. This analytical solution serves as a convenient method for verifying accuracy of the Surface Evolver models and is also used to rapidly calculate solutions for this propellant configuration.

The second type of configuration is a liquid ring around the junction between the end cap and cylinder. This propellant configuration is axisymmetric like in the case with a spherical interface, but has a second contact line located higher on the end cap. Since there is no liquid at the center of the end cap, no liquid can be drained, and the remaining propellant volume will be trapped. It is also possible to develop analytic solutions for this propellant configuration. The process is like the process for a spherical interface, however instead of spherical sections, sections of either unduloids or nodoids are used to define the shape of the free surface. The MATLAB code used to compute these solutions is listed in Appendix H. This method is useful for quickly computing solutions for the liquid ring for a range of volumes and contact angles, though Surface Evolver results are also used to verify accuracy of these results.

The third and final possible propellant configuration is an asymmetric droplet, which encompasses any distribution without rotational symmetry. Though some of these distributions have liquid at the center of the end cap allowing draining to occur, liquid will tend to coalesce at the junction when draining for any non-spherical tank geometry. As a result, liquid trapping will eventually occur even for distributions that are initially able to drain. Unlike the previous cases, analytical solutions are not feasible, and so Surface Evolver must be used.

4.1 Specific Case: Characterization of Configurations

All propellant configurations are found for a range of contact angles and fill fractions for one specific tank geometry to demonstrate how liquid trapping can occur. A 2:1 ellipsoidal end cap, where the depth is half the radius, is chosen as it demonstrates all the most common propellant distributions. This relatively short end cap accentuates the differences between an ellipsoidal end cap and hemispherical end cap.

When calculating solutions for this tank geometry, analytical solutions are used instead of Surface Evolver results whenever possible to speed up computations. As a result, energies for the spherical interface and liquid ring are calculated analytically, while Surface Evolver is only used for the asymmetric droplet. When fully converged, most of these asymmetric models use between 7,500 and 15,000 facets, depending on the fill fraction and contact angle. Configurations with low volumes and low contact angles require more facets to fully converge, in some cases requiring over 300,000 facets.

In general, it is possible for all three of these propellant configurations to exist for a single combination of tank geometry, contact angle, and liquid volume. For example, in the case of a 2:1 ellipsoidal end cap at a contact angle of 20° and fill fraction of 0.06, either a spherical interface, liquid ring, or asymmetric droplet is possible depending on the fluid history, as shown in Figure 4.1. Therefore, it is useful to characterize the range of existence for each propellant configuration, which may overlap with each other. When calculating results for the asymmetric droplet using Surface Evolver, contact angle is changed in increments of 1° and fill fraction in increments of 0.0005 until the boundary of the range of existence is found.

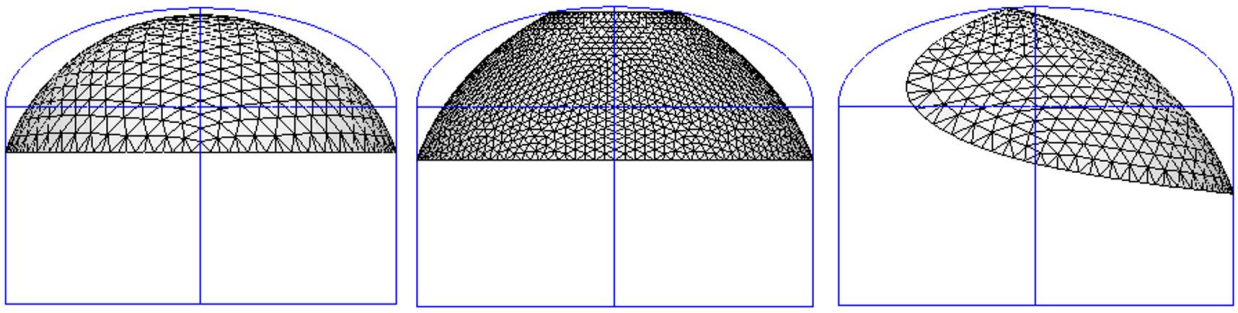

Figure 4.1. Possible configurations for a liquid volume inside a 2:1 ellipsoidal tank at a contact angle of 20° and fill fraction of 0.06 are a spherical interface (left), liquid ring (middle), and asymmetric droplet (right).

One method to visualize these results is to consider a single end cap geometry and label the range of existence on a diagram with fill fraction and contact angle on the abscissa and ordinate, respectively. Such a diagram is shown for the spherical interface with a 2:1 ellipsoidal end cap in Figure 4.2, computed using the analytical solution for a spherical interface. Here, the region labeled with a black S denotes the combinations of contact angles and fill fractions where a spherical interface is possible, while the region labeled with a black NS denotes the opposite, where no spherical interface is possible. A black curve separates these two regions, denoting the boundary between them. The region where no spherical interface is possible is only marked NS here for convenience, in all future diagrams of this style only the region of existence will be labeled. For any spherical interface propellant configuration near this boundary, any change in fill fraction that crosses the boundary causes the top of the bubble to touch the dome, resulting in the propellant reorienting as a liquid ring.

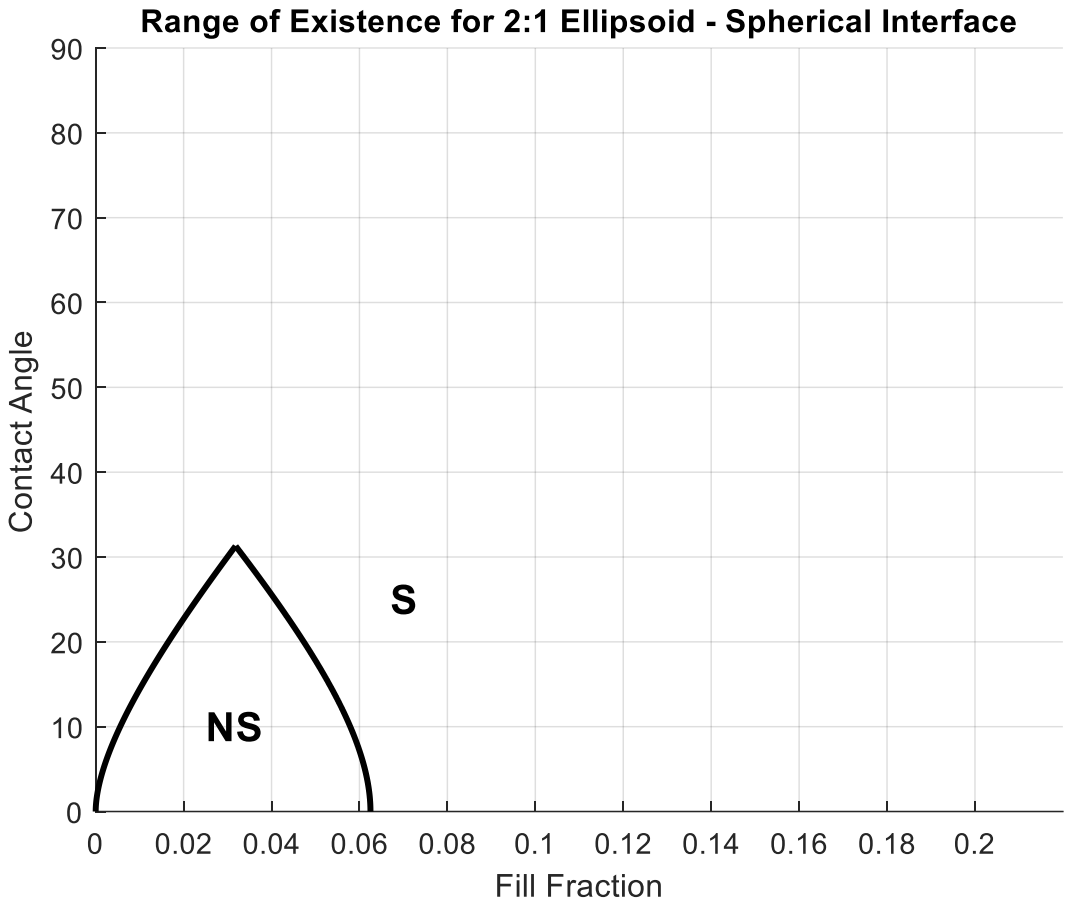

Figure 4.2. Range of existence for spherical interface solutions inside a tank with a 2:1 ellipsoidal dome. In the region labeled S, spherical interface solutions are possible, while in the region labeled NS no spherical interface solutions are possible.

A similar diagram can be made for the liquid ring, shown in Figure 4.3. The range of existence for the liquid ring configuration is shown for a 2:1 ellipsoidal end cap, computed using the analytical solution for a liquid ring configuration. The region labeled with a blue R denotes the combinations of contact angle and fill fraction for which a liquid ring solution is possible, bounded by the blue curve. For any combination of fill fraction and contact angle outside of this region, no liquid ring solution is possible. For a liquid ring distribution on the right boundary, any further increase in fill fraction would result in the radius of the upper contact line going to zero, causing the propellant to reorient with a spherical interface. Likewise, on the left boundary, any decrease in volume from a valid liquid ring solution would cause the ring to split, becoming an asymmetric droplet.

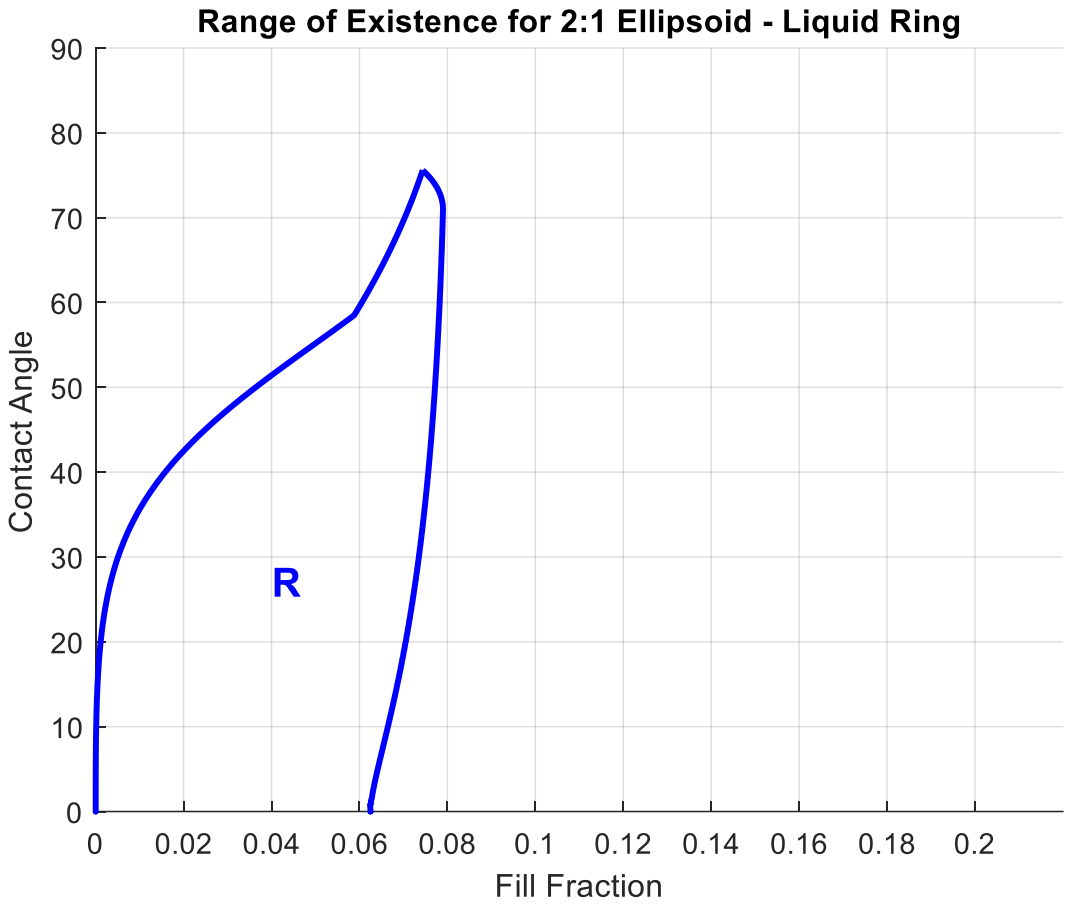

Figure 4.3. Range of existence for liquid ring solutions inside a tank with a 2:1 ellipsoidal dome. In the region labeled R, liquid ring solutions are possible, while outside of it no liquid ring solutions are possible.

Finally, a diagram showing the range of existence for an asymmetric droplet in a 2:1 ellipsoidal end cap can be seen in Figure 4.4, computed using Surface Evolver. The region labeled with a red A indicates the combinations of contact angles and fill fractions for which it is possible for an asymmetric droplet to form in this tank geometry, and the red curve is the boundary of this region. The boundary demonstrates a noticeable change in behavior near a contact angle of roughly 20°. This is due to a change in the type of instability exhibited by the asymmetric droplet. For an asymmetric droplet with contact angle below roughly 20°, increasing the liquid volume past the boundary results in the droplet extending around the junction between the end cap and cylinder, becoming a liquid ring. However, for an asymmetric droplet with contact angle above roughly 20°, increasing the liquid volume past the boundary results in the droplet shifting towards the center of

the dome, becoming a droplet with a spherical interface. Examples of asymmetric droplets near the boundary in each of these states can be seen in Figure 4.5.

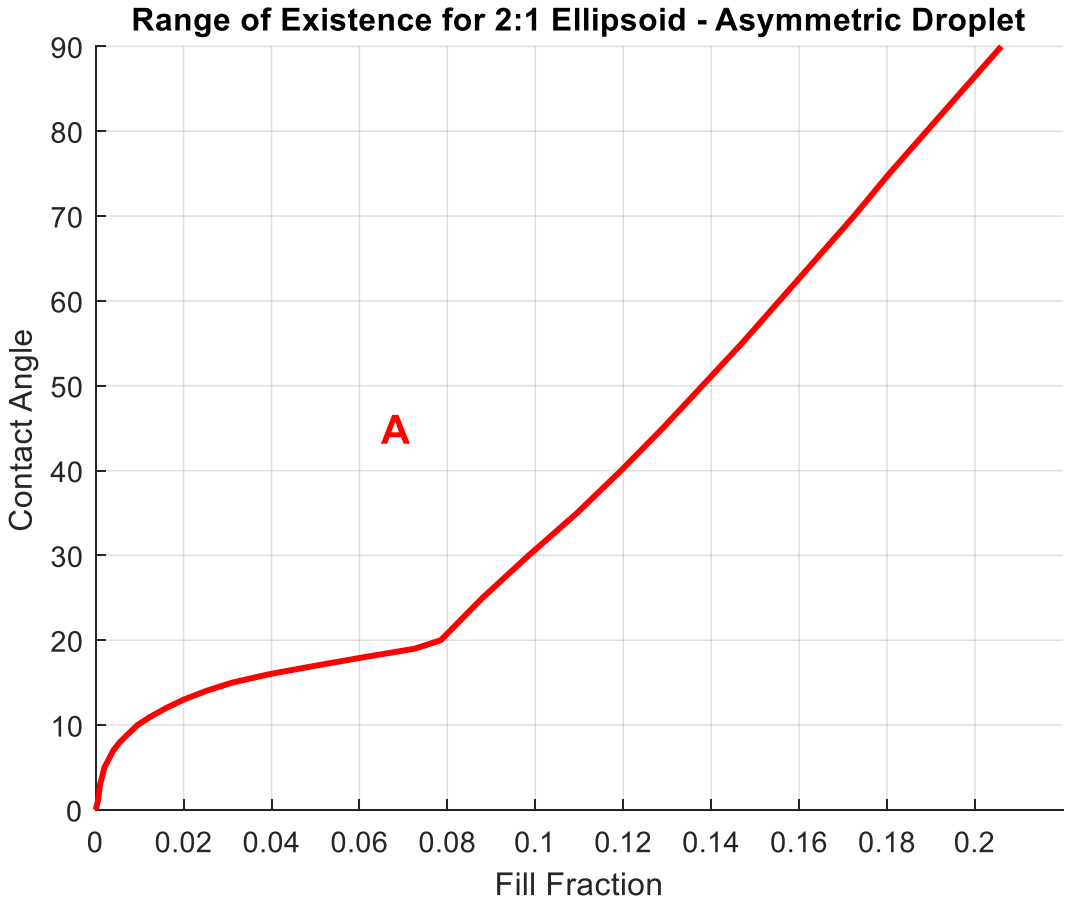

Figure 4.4. Range of existence for asymmetric droplet solutions inside a tank with a 2:1 ellipsoidal dome. In the region labeled A, asymmetric droplet solutions are possible, while outside of it no asymmetric droplet solutions are possible.

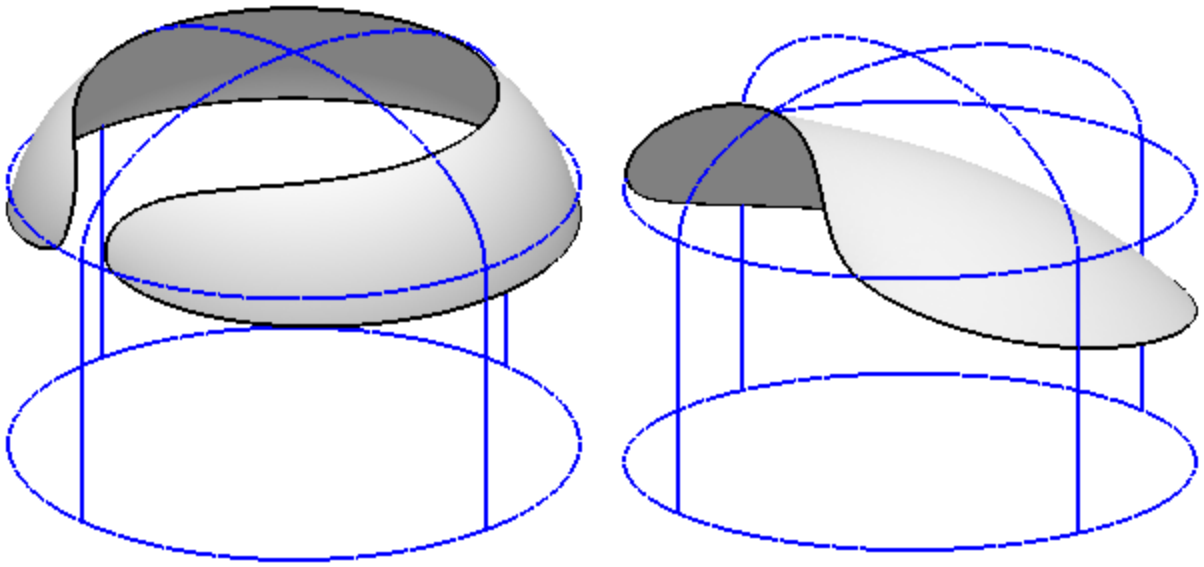

Figure 4.5. An asymmetric droplet with a contact angle of 10° and fill fraction of 0.0095 (left) and an asymmetric droplet with a contact angle of 45° and fill fraction of 0.129 (right) in a tank with 2:1 ellipsoidal end caps. For the lower contact angle case, a small increase in fill fraction would cause the propellant to transition to a liquid ring, while for the higher contact angle case it would cause the propellant to reorient with a spherical interface.

The diagrams in Figure 4.2, Figure 4.3, and Figure 4.4 can be combined into a single figure to show the overlap in range of existence for each type of propellant configuration. Such a diagram is shown in Figure 4.6 for the 2:1 ellipsoidal end cap. A black S denotes any region where a spherical interface can exist, a blue R denotes any region where a liquid ring can exist, and a red A denotes any region where an asymmetric droplet can exist. For example, in the region labeled SRA, all three solution types are possible, while in the region labeled SA, both spherical and asymmetric solutions are possible, but not a liquid ring. Some insights can readily be drawn from this diagram, such as the fact that only the spherical interface solution can exist at high liquid volumes where the contact line is far from the end cap, or that the liquid ring solutions are more prevalent at low contact angles, where the propellant can more easily wick into the corner at the junction. Interestingly, the region where the liquid ring exists fully covers the region where a spherical interface does not exist, meaning some form of axisymmetric solution is always possible. This also implies that there is no region where exclusively asymmetric droplet solutions are possible.

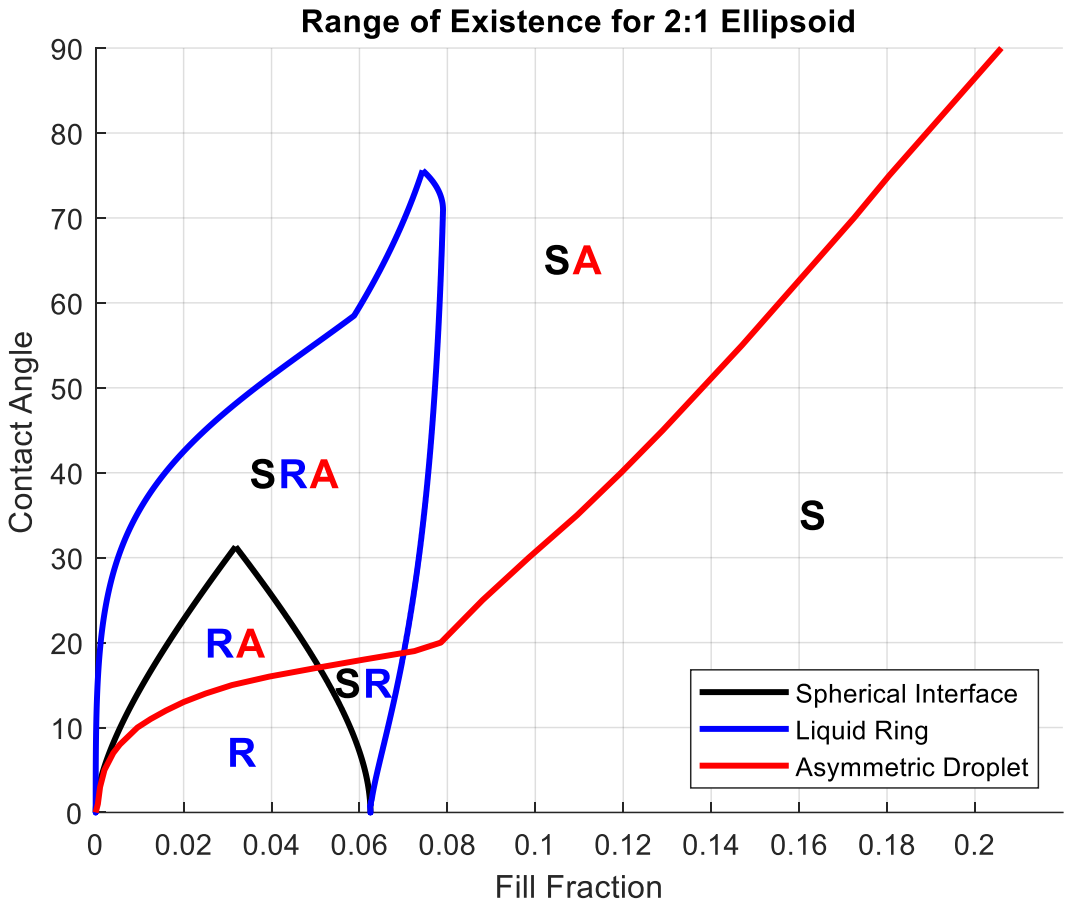

Figure 4.6. Range of existence for each propellant configuration in a tank with 2:1 ellipsoidal end caps. In regions labeled with S, spherical interface solutions are possible, in regions labeled with R, liquid ring solutions are possible, and in regions labeled with A, asymmetric droplet solutions are possible.

For the special case of a 0° contact angle propellant, which is a good approximation for many traditional propellants, there is no overlap in the range of existence for each propellant configuration. There is only one solution type possible for a given fill fraction: at low fill fractions, only the liquid ring solution is possible, while at high fill fractions only the spherical interface solution is possible. As a result, the range of existence for all propellant configurations at a 0° contact angle can be found easily by using only the analytical solution for a spherical interface. This remains true regardless of end cap geometry, so long as the tank is axisymmetric and has no vanes.

4.2 Specific Case: Filling and Draining

This type of diagram can be used to better understand liquid behavior during filling and draining by assuming a slow, quasistatic process. Since only liquid volume changes during filling and draining, these processes can be represented on the diagram as a horizontal line moving left or right at constant contact angle. Assuming no major perturbations, the type of propellant configuration will remain the same during the draining or filling process up until that configuration type no longer exists, at which point the propellant will reconfigure to one of the other types. If multiple other solution types exists when the propellant is forced to reconfigure in this way, the new configuration will depend largely on the type of instability present in the previous configuration. For example, when starting with a spherical interface and draining, the only way a further decrease in volume could make a spherical interface impossible is when the top of the bubble touches the dome, which would result in a transition to the liquid ring.

An example of draining can be seen in Figure 4.7, where the horizontal arrow represents the draining process of a propellant with contact angle of 45° that starts at a fill fraction of 0.15. Since the initial volume is a spherical interface, and there exist valid configurations with a spherical interface at every point along this line, the liquid will remain in the spherical interface configuration and no liquid trapping will occur. A similar example of draining can be seen in Figure 4.8, where a lower contact angle of 25° is considered with the same starting fill fraction of 0.15. The configuration starts with a spherical interface and remains spherical as it drains until it reaches (1), at which point no spherical interface solution exists. Since the bubble touching the top of the dome is the reason no spherical interface solution exists beyond this point, the propellant reconfigures as a liquid ring. At this point, the remaining propellant volume is trapped away from the drain hole, and so in a spacecraft no further liquid would be drained. If the liquid volume were to be decreased further, the liquid ring would shrink until reaching (2), at which point no liquid ring solution exists. The liquid ring breaks apart, becoming an asymmetric droplet and remaining that way as volume goes to zero.

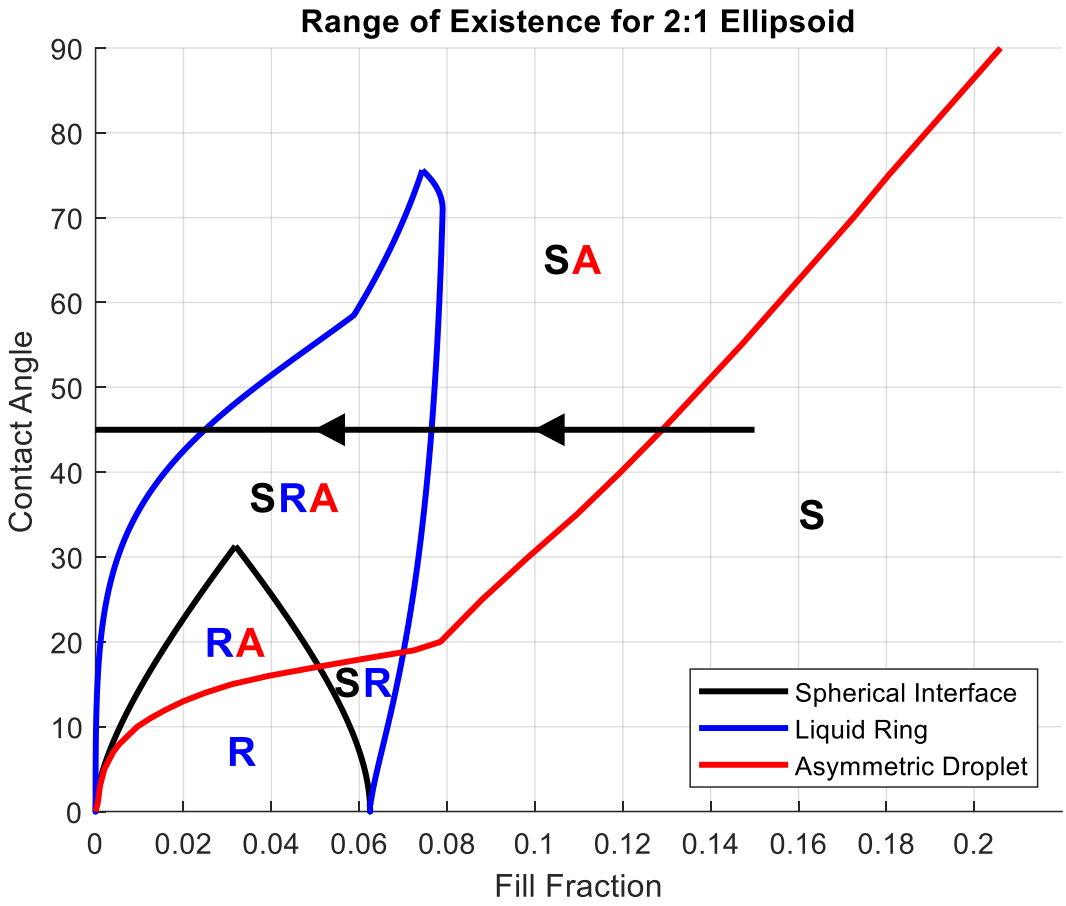

Figure 4.7. An example draining procedure for a tank with 2:1 ellipsoidal end caps containing a liquid with 45° contact angle is overlaid on a diagram showing range of existence for each propellant configuration.

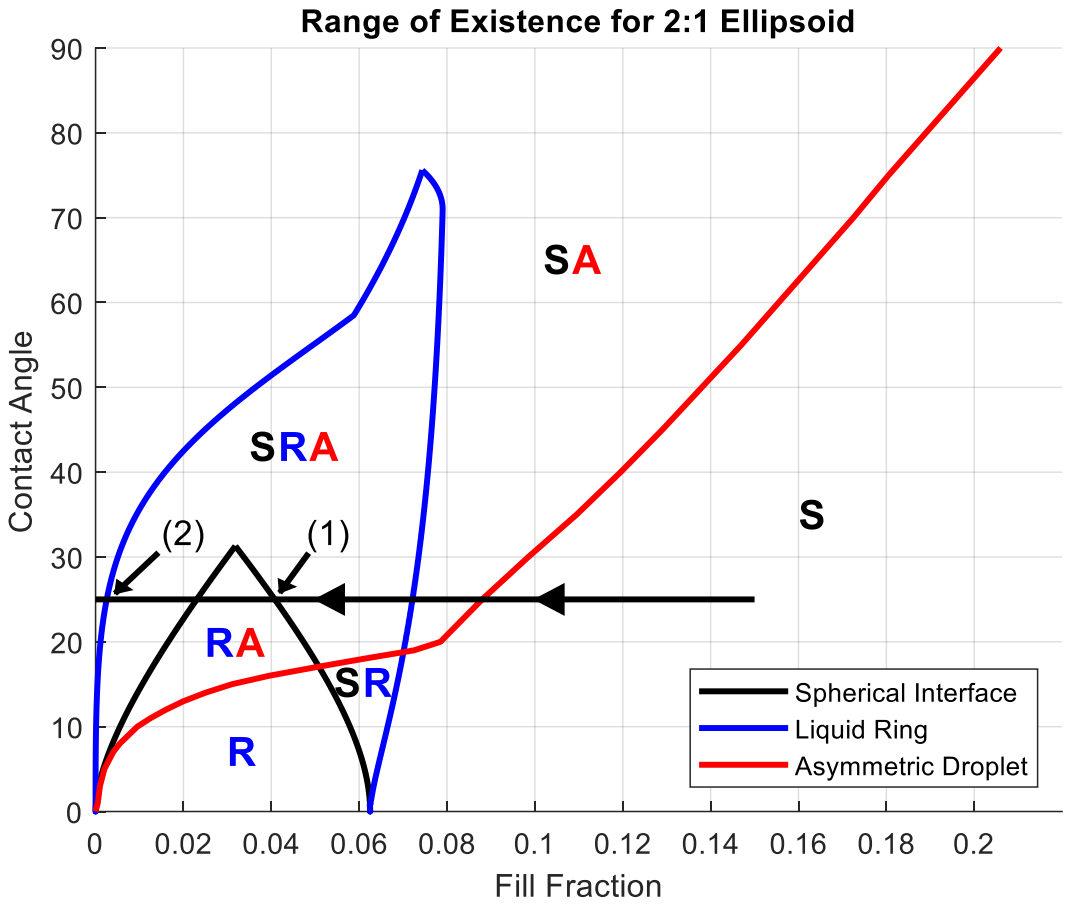

Figure 4.8. An example draining procedure for a tank with 2:1 ellipsoidal end caps containing a liquid with 25° contact angle is overlaid on a diagram showing range of existence for each propellant configuration.

A similar analysis can be carried out to consider filling in zero gravity, as might occur when refueling. In Figure 4.9, the tank starts empty and fills with a propellant with a contact angle of 45°. Assuming the tank fills from the center of the dome, a droplet with a spherical interface will form around the inlet. As the tank continues to fill, this droplet will grow, maintaining a spherical interface until filled to the desired level. A similar process occurs in Figure 4.10, except this time a propellant with a lower contact angle of 25° is used. Again, as the tank begins to fill, a droplet with a spherical interface forms near the inlet. As the droplet grows, the interface remains spherical until (1), at which point the top of the bubble touches the dome, resulting in a reconfiguration to a liquid ring. If more liquid is added, a new droplet will form at the inlet with a spherical interface, which will grow until the contact line reaches the upper contact line of the liquid ring. Once the

two liquid volumes meet, they combine, increasing the volume of the liquid ring. Adding more liquid will result in this process repeating, forming a new spherical interface droplet that grows and joins the liquid ring until reaching (2), at which point a liquid ring solution is no longer possible. This occurs because the radius of the upper contact line goes to zero, causing the propellant to reconfigure as a droplet with a spherical interface. The propellant configuration will remain a spherical interface as more propellant is added until the desired level is reached. Therefore, care should be taken when designing vanes if refueling is required to ensure that either the liquid does not form multiple volumes during the filling procedure, or that enough propellant is added to force the formation of a spherical interface.

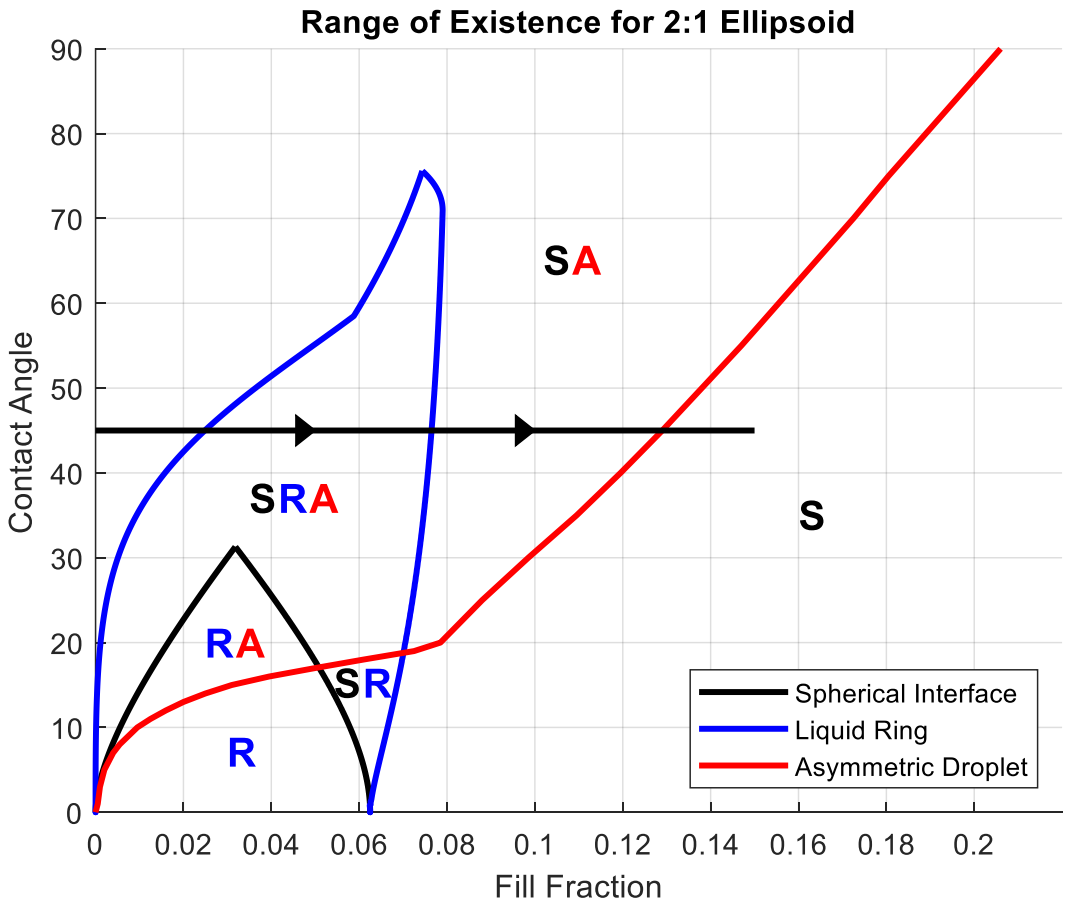

Figure 4.9. An example filling procedure for a tank with 2:1 ellipsoidal end caps containing a liquid with 45° contact angle is overlaid on a diagram showing range of existence for each propellant configuration.

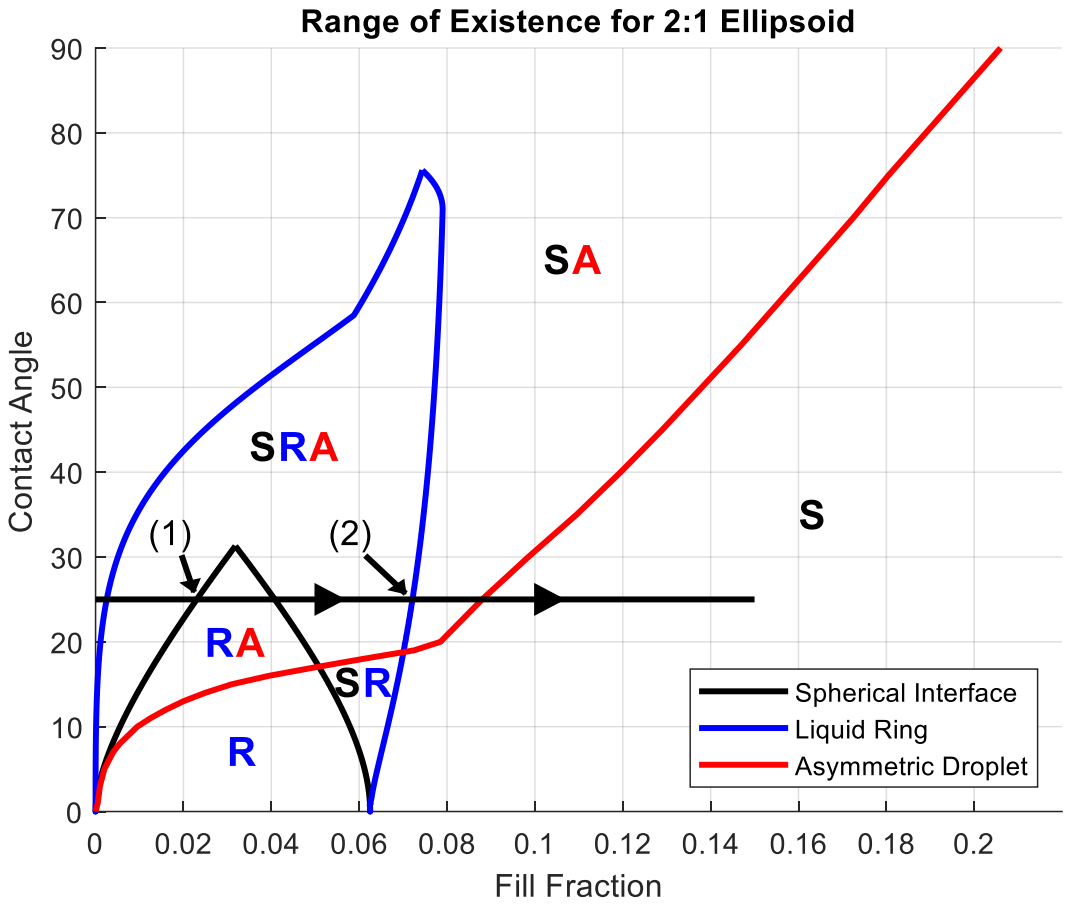

Figure 4.10. An example filling procedure for a tank with 2:1 ellipsoidal end caps containing a liquid with 25° contact angle is overlaid on a diagram showing range of existence for each propellant configuration.

4.3 Specific Case: Minimum Energy

In addition to computing the range of existence for each type of solution, the energies of each configuration can be compared to determine the minimum energy solution for a given combination of fill fraction, contact angle, and tank geometry. Such a plot is shown for the 2:1 ellipsoidal dome in Figure 4.11. The notation used here is like that used in the range of existence plots, where a black S denotes the combinations of fill fraction and contact angle for which a spherical interface is the minimum energy, a blue R denotes the combinations for which a liquid ring is the minimum energy, and a red A denotes the combinations for which an asymmetric droplet is the minimum energy.

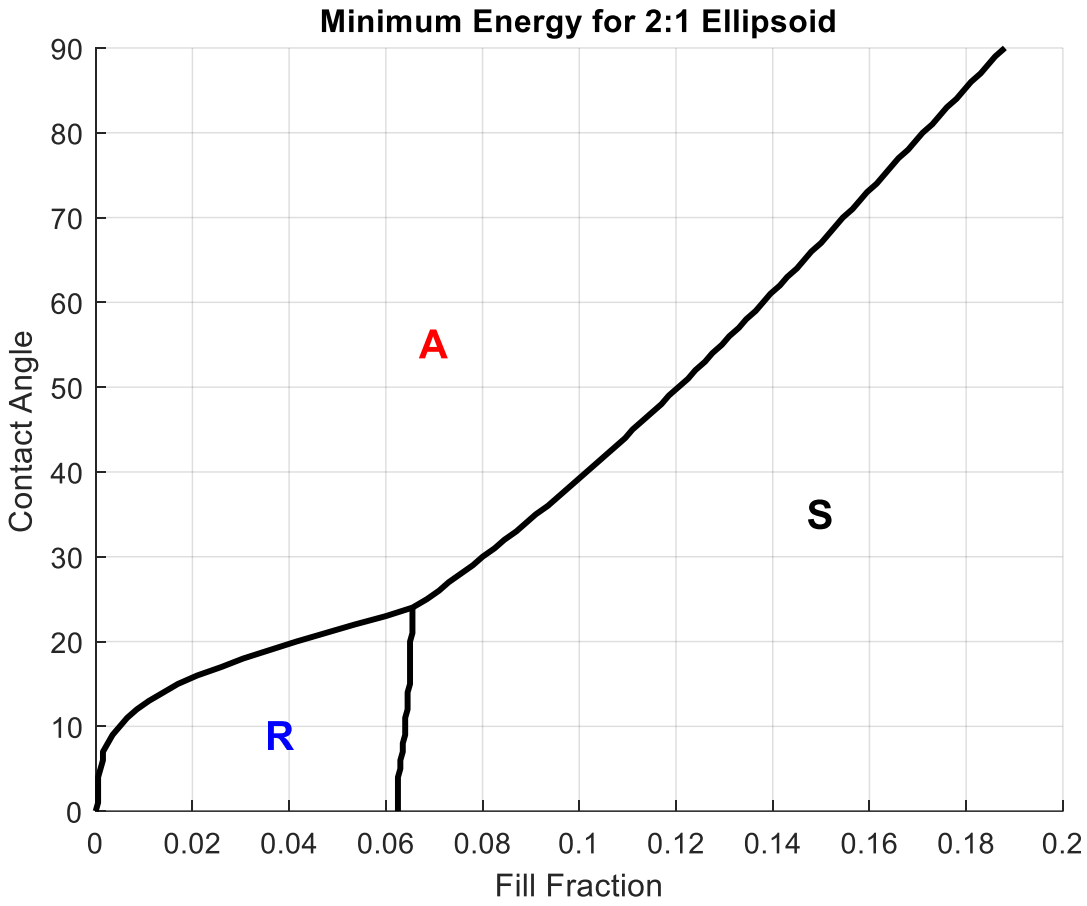

Figure 4.11. Minimum energy propellant configurations for a tank with 2:1 ellipsoidal end caps. The region marked with S denotes the combinations of fill fraction and contact angle where a spherical interface is the minimum energy, the region marked with R denotes the combinations where a liquid ring is the minimum energy, and the region marked with A denotes the combinations where an asymmetric droplet is the minimum energy.

Interestingly, for low volumes the spherical interface is never the minimum energy solution, indicating that low volume spherical interface solutions may be particularly susceptible to perturbations. This is particularly noteworthy because liquid trapping occurs in all propellant configurations besides spherical, suggesting that particular care needs to be taken when designing vanes to ensure that liquid is held near the outlet even at low volumes.

The energies are computed analytically for the spherical interface and liquid ring and using Surface Evolver for the asymmetric droplet. Since the free surface in Surface Evolver is discretized and not exact, the energy computed using Surface Evolver will be a bit higher than its true value. This result is then compared directly with the exact energies computed analytically for the

spherical interface and liquid ring, which will result in some error in the location of the minimum energy boundary. The true surface can be better approximated by using more facets, especially when the free surface has high curvature or when discretization errors in volume calculations are large relative to the overall volume, as is the case at low fill fractions. The result of this error can be estimated by further refining the surface after fully converged and iterating until the new surface is fully converged. The new energy can then be compared to the analytical solutions to find a more accurate estimate of the minimum energy boundary. This process is done manually for contact angles of 10°, 30°, 50°, and 70°, and it is found that the effect of this error is negligible, with the boundary fill fraction increasing by a maximum of 0.0005.

The range of existence diagram and minimum energy diagram shown above can be combined into a single plot to see relationships between the two. The two are combined in Figure 4.12, however the region labels are omitted for legibility. The solid lines define the boundaries for range of existence of each type of propellant configuration, while the dashed lines are the boundary between the minimum energy solutions for each type of configuration. One notable thing about this diagram is that the boundary for range of existence of an asymmetric droplet is consistently close to its boundary for minimum energy. This is because the energy of the asymmetric droplet is almost always near the global minimum energy. If an asymmetric droplet is possible, it will always be the minimum energy or close to the minimum energy. This is not true for the liquid ring, which for some combinations of fill fraction and contact angle can exist at a much higher energy than the minimum energy solution. This is shown in the large disparity in size of the range of existence compared to minimum energy for a liquid ring. Meanwhile, the spherical interface is only near the minimum energy when neither of the other two solutions exist, limiting the minimum energy region for the spherical interface to only high fill fractions, despite its existence at low fill fractions.

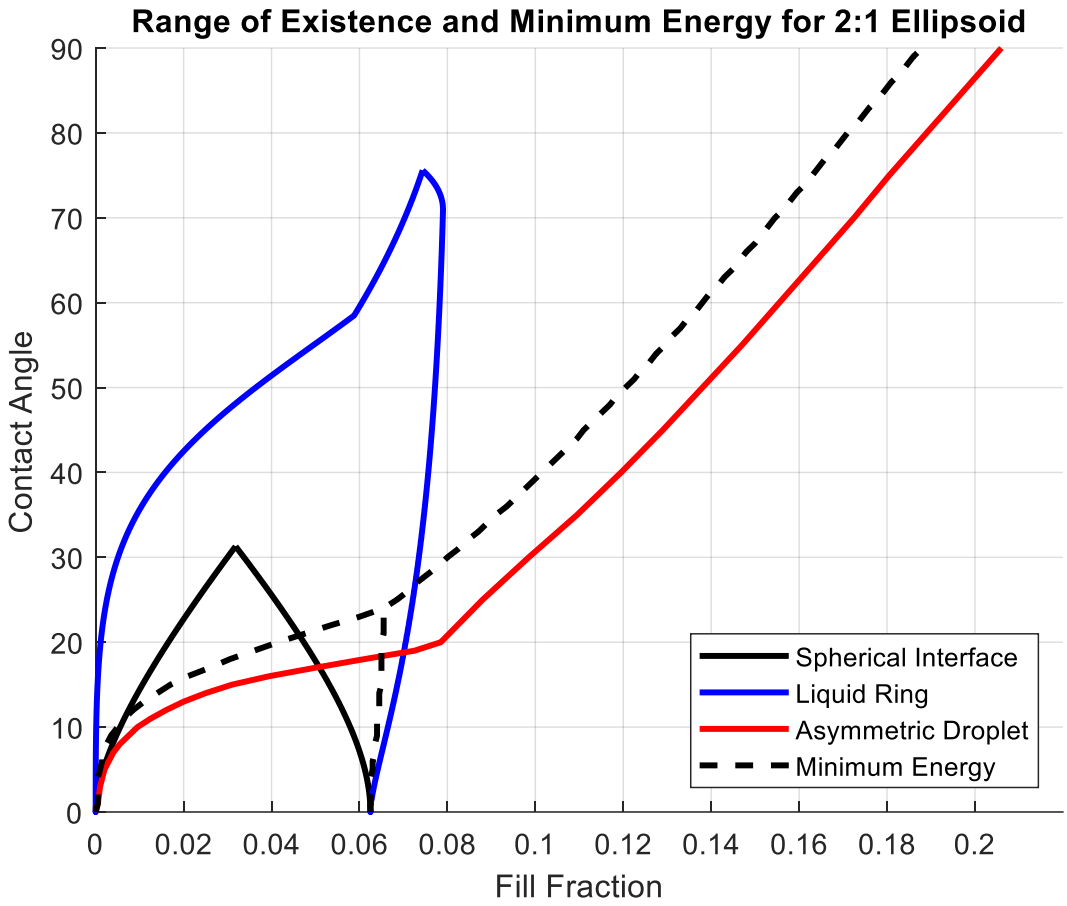

Figure 4.12. Range of existence and minimum energy for a tank with 2:1 ellipsoidal end caps are overlaid on the same diagram. Solid lines denote boundaries for range of existence, while dashed lines denote boundaries for minimum energy.

4.4 Specific Case: Energy vs Fill Fraction

Considering a single contact angle can help understand the relationship between range of existence and minimum energy for each propellant configuration. Energy is plotted against fill fraction at a contact angle of 15° in Figure 4.13 for each propellant configuration, omitting the constant factor in Equation 2.1. The range of existence for each type of configuration can be seen as well as the minimum energy configuration. This plot provides additional information in the form of the energy difference between configurations, which corresponds roughly to the instability of a configuration. The energy difference is plotted against fill fraction in Figure 4.14 to make these differences easier to see. At low fill fractions, the spherical interface configuration has a much higher energy than either the asymmetric droplet or liquid ring. Therefore, even though it

may be possible for a spherical interface to form at low fill fractions, such configurations may be particularly unstable and susceptible to perturbations. This is especially noteworthy because the spherical interface is the only configuration that does not exhibit liquid trapping, showing the need for special care to be taken when designing vanes to avoid liquid trapping at low fill fractions.

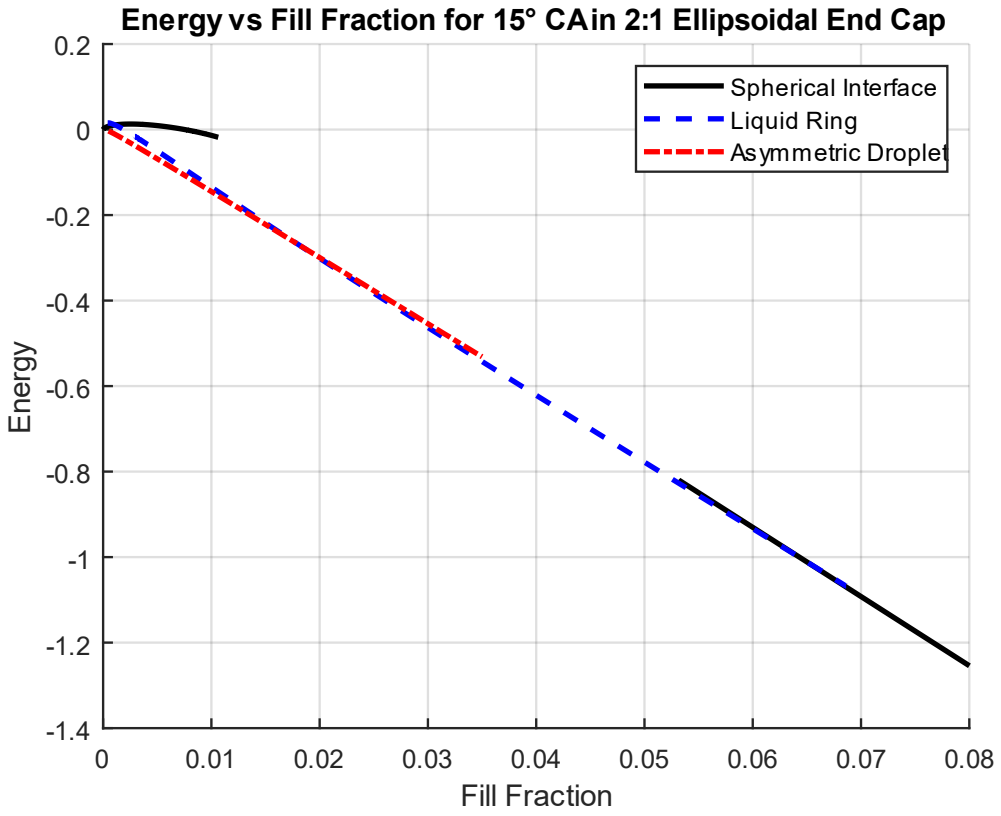

Figure 4.13. Energy vs fill fraction for each propellant configuration with a 15° contact angle in a tank with 2:1 ellipsoidal end caps.

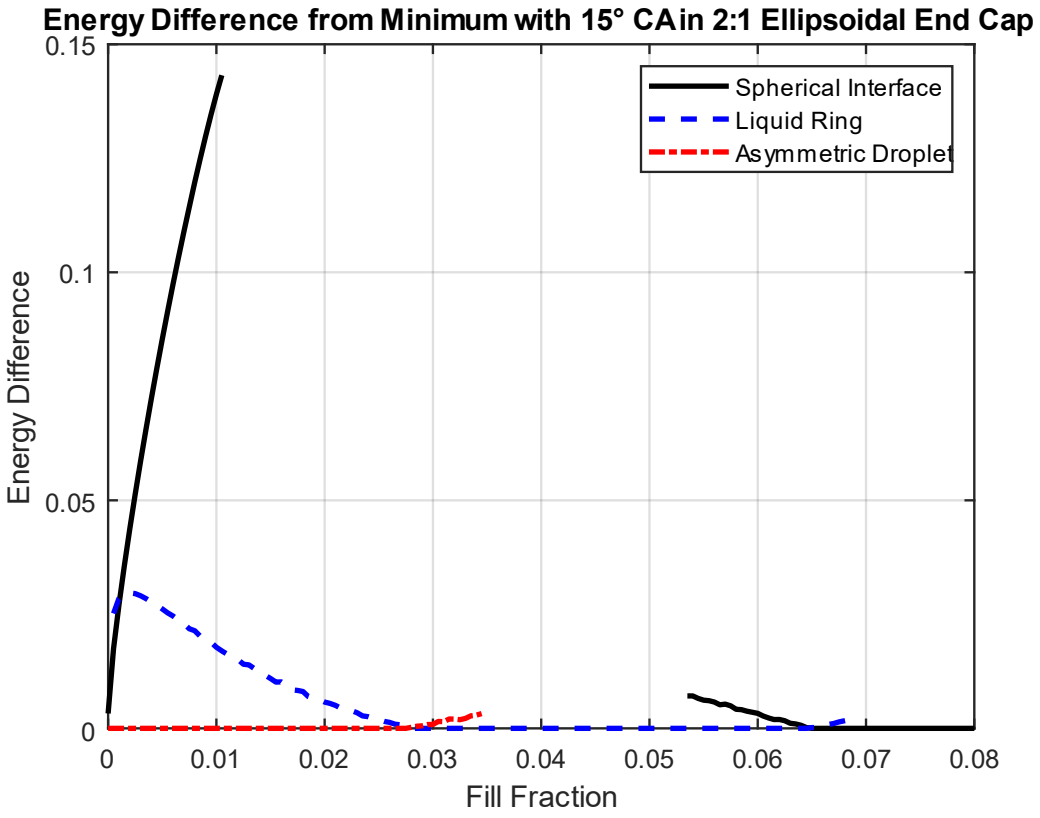

Figure 4.14. Energy difference relative to the minimum energy vs fill fraction for each propellant configuration with a contact angle of 15° in a tank with 2:1 ellipsoidal end caps.

4.5 Other Geometries

In addition to the detailed analysis performed for the 2:1 ellipsoidal end cap above, solutions are computed for several other tank geometries to evaluate trends in trapping behavior. A 4:3 ellipsoidal end cap (with depth equal to 0.75 times the radius), 2:1 superellipsoidal end cap with exponent n = 3, 4:3 superellipsoidal end cap with n = 3, 2:1 torispherical end cap with $r_N = 0.6$, and 4:3 torispherical end cap with $r_N = 0.6$ are also considered. These tank geometries can be seen in Figure 4.15. Results are computed for a spherical interface, liquid ring, and asymmetric droplet for each end cap geometry. The energies for each solution are compared to determine the minimum energy configuration for a given fill fraction and contact angle. The full range of existence is not computed for each propellant configuration since special care is required for Surface Evolver to converge when the configuration is barely stable, as is the case at the edge of a configuration's range of existence.

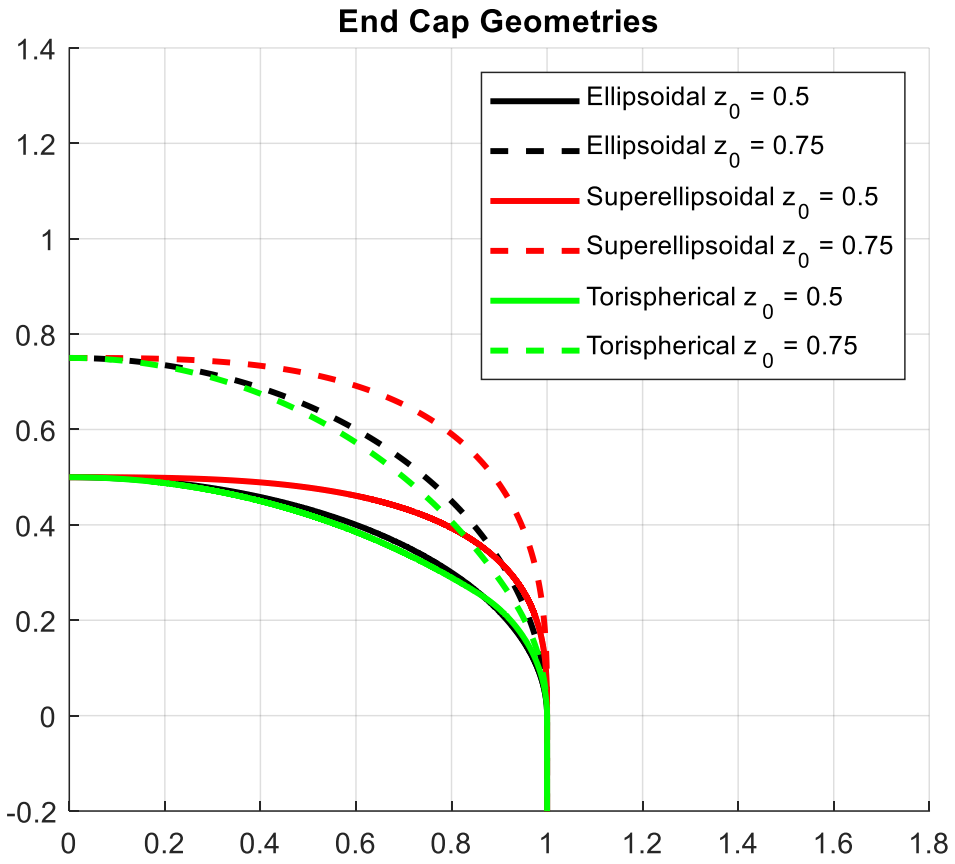

Figure 4.15. Ellipsoidal, superellipsoidal, and torispherical end cap geometries considered.

A larger increment in contact angle and fill fraction is used to obtain results for the increased number of end cap geometries. Contact angle starts at 5°, increasing in 5° increments until 90°. Fill fraction starts at 0.005, increasing in increments of 0.005 until the spherical interface represents the minimum energy solution. Surface Evolver is used to compute liquid ring and asymmetric droplet solutions for each geometry, while the analytical solution is used for the spherical interface to save computation time. Results for a contact angle of 0° are computed analytically using the methods discussed in Section 4.1. Since the Surface Evolver results are discretized, the computed energies are higher than their true values. These approximate energies for the liquid ring and asymmetric droplet are compared directly with the exact energies from the analytical spherical interface solution, so the minimum energy boundaries presented here will be shifted slightly from their true values. While this error source is also present in the discussion of the 2:1 ellipsoid in Section 4.3, it is far more pronounced here since all models use exactly 1088

facets each, roughly a factor of 10 less than those used previously. However, due to the increased fill fraction step size, the result of this error is still estimated to be less than the increment in fill fraction.

Figure 4.16 shows the minimum energy propellant configuration for a 2:1 superellipsoidal end cap with n = 3. The notation used is the same as Figure 4.11, where a black S represents the combinations of fill fraction and contact angle where the spherical interface is the minimum energy solution, a blue R denotes the region where the liquid ring is the minimum energy solution, and a red S denotes the region where the asymmetric droplet is the minimum energy solution. The curve separating these regions is the boundary at which two different configurations have the same energy. Horizontal error bars on the boundary represent the size of the fill fraction increment used. Similar diagrams for the other end cap geometries discussed in the section can be seen in Appendix A.

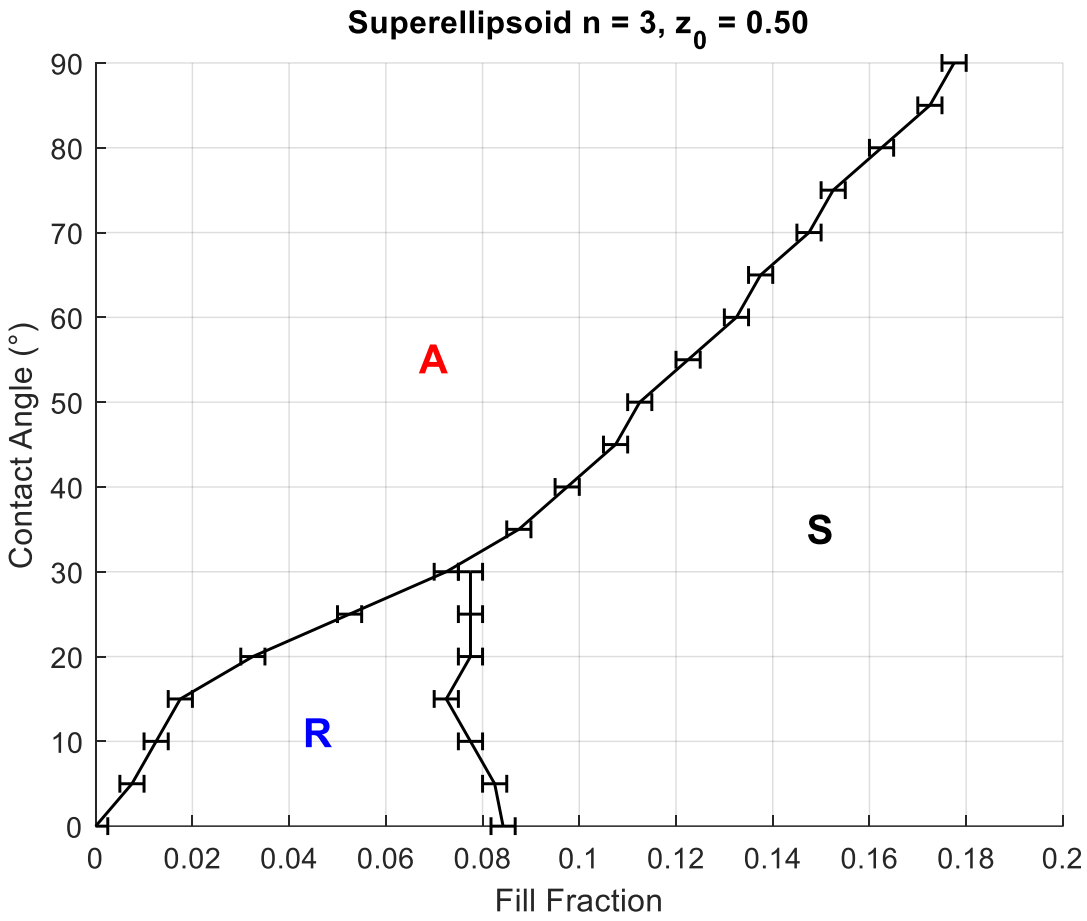

Figure 4.16. Minimum energy propellant configurations for a tank with 2:1 superellipsoidal end caps with n = 3. The region marked with S denotes the combinations of fill fraction and contact angle where the spherical interface is the minimum energy, the region marked with R denotes the combinations where the liquid ring is the minimum energy, and the region marked with A denotes the combinations where the asymmetric droplet is the minimum energy. The horizontal error bars show the fill fraction step size.

An interesting comparison can be made regarding the behavior of propellant in end caps with depth $z_0 = 0.5$. The minimum energy configurations for these end caps can be seen in Figure 4.16 and Figure A.1 of Appendix A, along with the minimum energy for the 2:1 ellipsoidal end cap which can be seen in Figure 4.11. These diagrams are combined into one figure in Figure 4.17, with the different color curves representing the boundary for the corresponding end cap geometry. The same general trends can be observed for each geometry, with the spherical interface as the minimum energy for high volumes, asymmetric droplet as the minimum for low volumes, and the liquid ring solution as the minimum energy for low contact angles, with fill fraction nestled

between the two. The behavior exhibited by the torispherical end cap is very similar to that of the ellipsoidal end cap. This is largely due to the torispherical knuckle radius being specifically chosen to result in nearly the same geometry as the 2:1 ellipsoidal end cap. Even though the geometries are nearly identical, there are some discrepancies that are discernible even with the relatively large steps in fill fraction and contact angle used. This indicates that mean curvature, which differs more substantially between these two geometries, plays a notable role in the boundaries of the minimum energy propellant configuration. Compared to the ellipsoidal and torispherical end cap, the superellipsoidal end cap has a more prominent liquid ring region. This is due to the tighter corner at the junction, which allows for higher volume liquid ring solutions to wick into the junction. The boundary between spherical interface and asymmetric droplet solutions is quite similar for all cases. At high contact angles, the boundary for the superellipsoidal end cap deviates from the other two, transitioning to a spherical interface at a lower fill fraction. This indicates that the asymmetric droplet is less stable at high fill fractions for the superellipsoidal end cap than for the ellipsoidal or torispherical end caps.

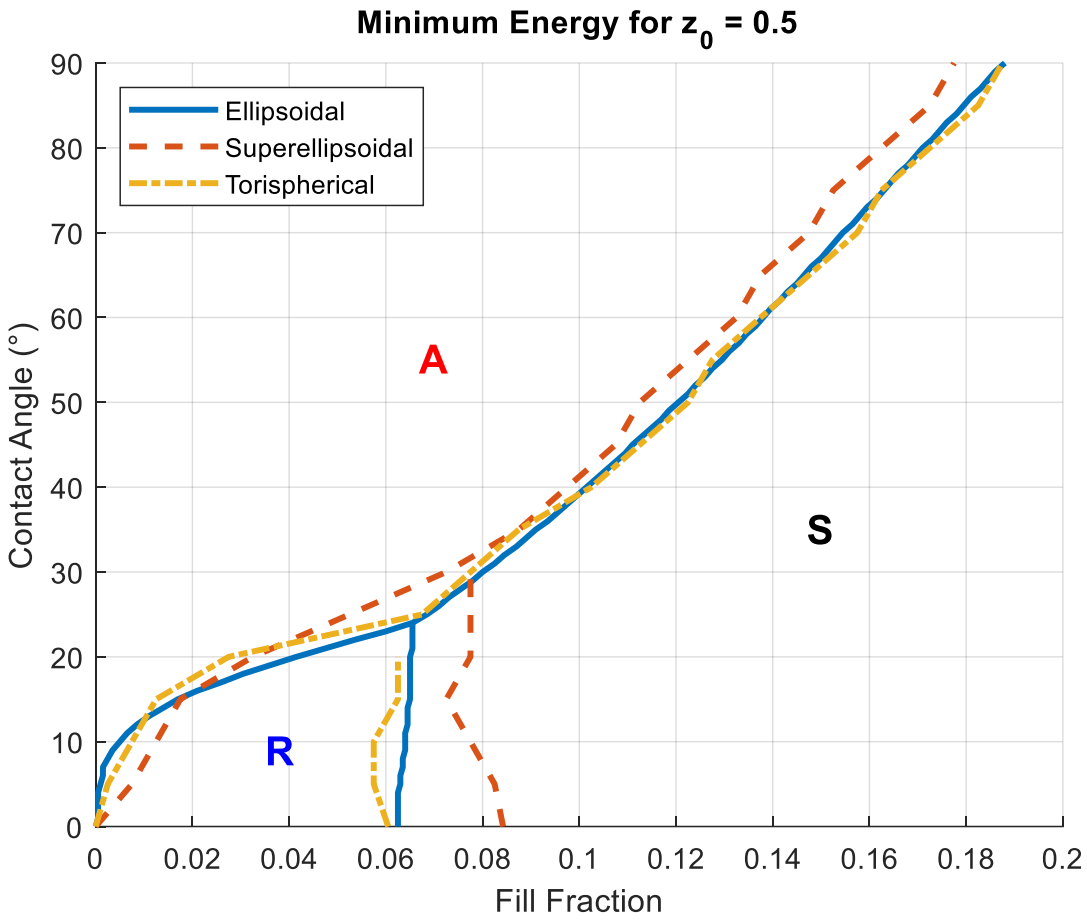

Figure 4.17. Minimum energy configurations for tanks with 2:1 ellipsoidal, superellipsoidal, and torispherical end caps overlaid on the same diagram. The region marked with S denotes where the spherical interface is the minimum energy, R where the liquid ring is minimum, and A where the asymmetric droplet is minimum. The blue curve is the boundary for the tank with ellipsoidal end caps, the red curve is the boundary for the tank with superellipsoidal end caps, and the yellow curve is the boundary for the tank with torispherical end caps.

A similar comparison can be performed for an end cap depth of $z_0 = 0.75$. Minimum energy diagrams for each individual end cap geometry can be seen in Figures A.2, A.3, and A.4 of Appendix A. These diagrams are combined into a single diagram in Figure 4.18, showing the minimum energy configuration for each end cap geometry. The same trends as discussed before can be seen, with the spherical interface solution representing the minimum energy at large fill fractions, the asymmetric droplet representing the minimum at low fill fractions, and the liquid ring representing the minimum between the two at low contact angles only. The most noticeable effect of increasing the end cap depth to 0.75 is decreasing the prominence of liquid ring solutions.

By increasing the end cap depth, the curvature at the junction decreases. As a result, liquid is wicked into the junction less strongly, causing the liquid ring configuration to be less stable for end cap geometries with $z_0 = 0.75$ than with $z_0 = 0.5$. Between the three end cap geometries here, the superellipsoidal end cap has the most prominent liquid ring region, followed by the ellipsoidal end cap, then torispherical end cap. The liquid ring region is most prominent for the superellipsoidal end cap since it has a tighter corner at the junction, while the torispherical end cap has the least prominent liquid ring region because it has the lowest curvature at the junction. The differences in the liquid ring region are more significant for the deeper end caps, however this is primarily because the end cap geometries themselves differ more for the $z_0 = 0.75$ end caps. The boundary between the spherical interface and asymmetric droplet minimum energy configurations is similar for all three cases, though they deviate a bit, particularly at high contact angle and fill fraction. This deviation is most significant for the superellipsoidal end cap, indicating that the asymmetric droplet is less stable for large volumes in the superellipsoidal end cap than for the ellipsoidal or torispherical end cap.

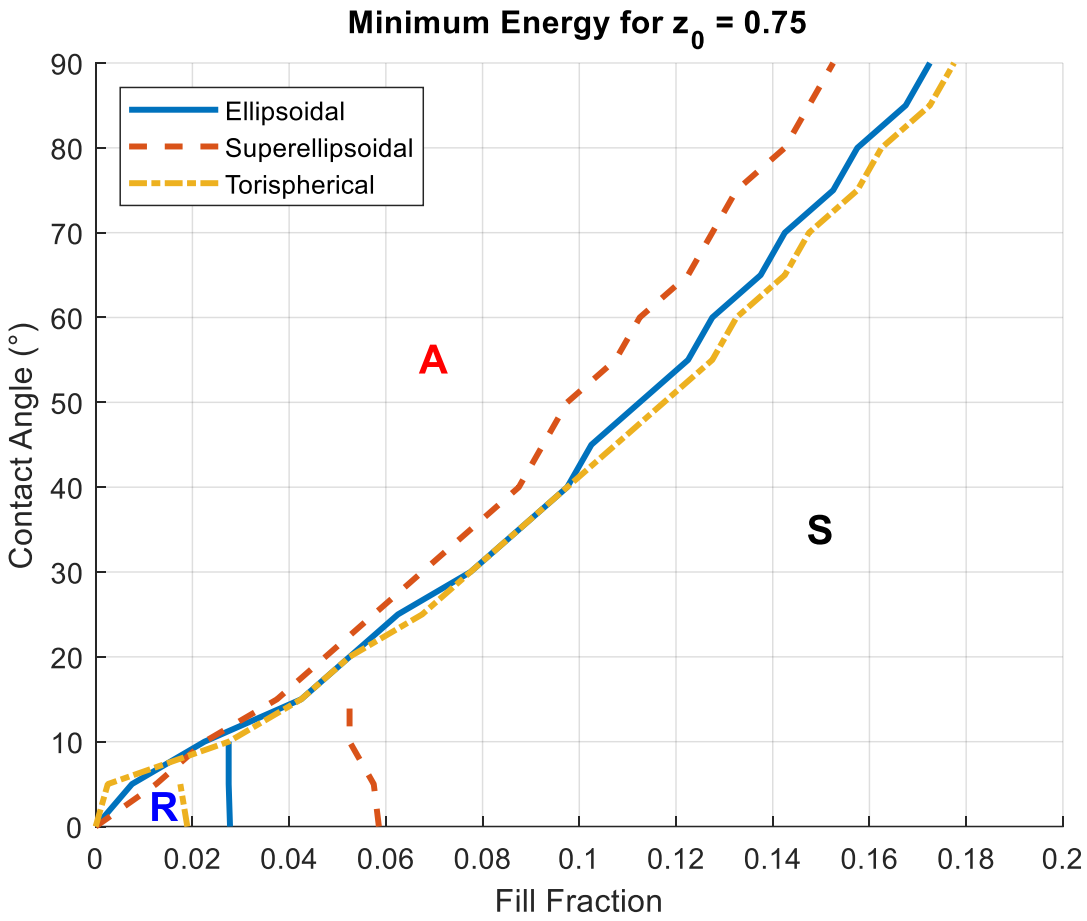

Figure 4.18. Minimum energy configurations for tanks with 4:3 ellipsoidal, superellipsoidal, and torispherical end caps overlaid on the same diagram. The region marked with S denotes where the spherical interface is the minimum energy, R where the liquid ring is minimum, and A where the asymmetric droplet is minimum. The blue curve is the boundary for the tank with ellipsoidal end caps, the red curve is the boundary for the tank with superellipsoidal end caps, and the yellow curve is the boundary for the tank with torispherical end caps.

5. LIQUID TRAPPING SUMMARY

The problem of liquid trapping is analyzed using analytical methods and Surface Evolver. Results are computed and analyzed for bare, vaneless tanks with ellipsoidal, superellipsoidal, and torispherical end caps. For these geometries, three different possible propellant configurations are identified: a spherical interface, a liquid ring, and an asymmetric droplet. Propellant configurations in these tank geometries are computed using Surface Evolver for a wide range of fill fractions and contact angles. Detailed results are computed for the case of a tank with 2:1 ellipsoidal end caps, which are used to discuss range of existence and minimum energy configurations for each solution type. Propellant filling and draining behavior are also discussed for this end cap geometry. Results for other end cap geometries are compared to evaluate general trends in minimum energy liquid configurations.

6. ROTATIONAL SLOSH INTRODUCTION

Each tank inside the rotational slosh payload will contain one of two liquids. The first is silicone oil, which is a nontoxic liquid that has a contact angle near 0° when used with acrylic. Most traditional propellants have contact angles near 0° as well, making it an excellent option for simulating these propellants. The other liquid that will be used is tripropylene glycol diluted with water, which is also nontoxic and easy to work with. The contact angle of tripropylene glycol when used with acrylic varies depending on the level of dilution but will be chosen so the contact angle is near 20°. Tripropylene glycol is a good point of comparison for propellants with non-zero contact angle such as hydrogen peroxide and more recent green propellants while also serving as a great benchmark for computational models.

Before the start of the experiment, the tanks are initially rotated 22.5° from the vertical. The experiment then waits to receive power from SpaceShipTwo, at which point the Arduinos used to control the electronics will power on and turn the cameras and accelerometer on. The accelerometer will then begin recording data, waiting for a sustained period of near zero gravity to indicate the start of the experiment. At this point the cameras are triggered to begin recording, and, after allowing the liquid to reach equilibrium, the first tank rotations can occur. The tanks rotate 45°, ending up at 22.5° opposite their starting location, mirrored about the vertical. The tank rotations will occur at one of two speeds about an axis perpendicular to the longitudinal axis going through either the middle of the tank or top of the tank. There will be 20 seconds between rotations to allow time for the liquid slosh to dampen and return to its equilibrium position. Sloshing behavior of fluids in microgravity is very difficult to predict, so the waiting time between rotations is chosen based on experience from previous low gravity fluid experiments, though may be subject to updating based on ground testing of the payload. The first rotation will be slow and about the middle pivot, holding for 20 seconds at the new location before rotating back to their original location at the faster speed, still rotating about the middle pivot. After waiting for oscillations to dampen again, the process is repeated for the other pivot about the top of the tanks. This process is then repeated until zero-g time concludes. Once the accelerometer detects acceleration consistent with the end of zero-g time, the motors will stop, and the cameras will stop recording and power off.

An order of magnitude estimate for tank rotation speed can be found by scaling satellite data. The propellant slosh inside the tanks is driven primarily by the acceleration experienced by the liquid in response to tank rotation, so the angular acceleration α will be one of the most important quantities to consider. Other relevant variables are the rotational velocity Ω, tank radius R, fluid kinematic viscosity ν, density ρ, and surface tension σ. Using the Buckingham π theorem gives three nondimensional quantities, one roughly equivalent to a Bond number, another roughly equivalent to a Reynolds number, and the last roughly equivalent to a Weber number. These equivalences can be shown by noting that velocities inside the tank are proportional to RΩ and accelerations to Rα.

$$\Pi_1 = \frac{\rho R^3 \alpha}{\sigma} \sim \frac{\Delta \rho R^2 a}{\sigma} = B \tag{6.1}$$

$$\Pi_2 = \frac{R^2 \Omega}{\nu} \sim \frac{Ru}{\nu} = Re \tag{6.2}$$

$$\Pi_3 = \frac{\rho R^3 \Omega^2}{\sigma} \sim \frac{\rho R u^2}{\sigma} = We \tag{6.3}$$

The satellite angular velocity and tank radius could be obtained for NASA's Plankton, Aerosol, Cloud, ocean Ecosystem (PACE) satellite, but not the angular acceleration or fluid properties. Experimental payload tank angular velocities are computed based on the Reynolds number and Weber number assuming the fluid properties are of the same order of magnitude for both the experiment and PACE. The subscript exp represents variables for the experimental payload and the subscript PACE represents variables from the PACE satellite.

$$\left(\Omega_{exp}\right)_2 = \left(\frac{R_{PACE}}{R_{exp}}\right)^2 \Omega_{PACE} = 40°/\text{sec} \tag{6.4}$$

$$\left(\Omega_{exp}\right)_3 = \left(\frac{R_{PACE}}{R_{exp}}\right)^{3/2} \Omega_{PACE} = 10°/\text{sec} \tag{6.5}$$

A very rough estimate for the angular acceleration of the PACE satellite can be obtained by combining rotation rates with an estimate for thruster fire duration and assuming a single firing of a thruster accelerates the satellite to its maximum rotational velocity. An experiment on the PACE propulsion subsystem recorded thruster firing data for a few seconds (Mikhailovsky, Stahl, & Mulkey, 2020), indicating that the fire length is of a similar order of magnitude. Assuming the

thrust from the thruster is constant during its fire time, $\alpha = \frac{\Omega}{t}$ can be substituted into the equation for Π_1. The experimental payload tanks likely take between 0.01 s and 0.1 s to accelerate to full speed, which can be used to estimate the angular acceleration of the experimental payload tanks. A scaling based on Bond number can then be performed to estimate the tank rotation rate.

$$\Pi_1 = \frac{\rho R^3 \alpha}{\sigma} = \frac{\rho R^3 \Omega}{\sigma t} \tag{6.6}$$

$$\left(\Omega_{exp}\right)_1 = \frac{t_{exp}}{t_{PACE}} \left(\frac{R_{PACE}}{R_{exp}}\right)^3 \Omega_{PACE} = 4.7°/\text{sec} - 47°/\text{sec} \tag{6.7}$$

The lower bound of this range corresponds to the assumption that the experimental payload tanks take 0.01 s to fully accelerate, while the upper bound assumes that the tanks take 0.1 s to fully accelerate. These results are of the same order of magnitude as those computed using the Reynolds number and Weber number, lending some confidence to these approximate results. As a result, tank rotation rates of 10°/s and 40°/s were chosen to be the fast rotation speed and slow rotation speed, respectively.

6.1 Low Gravity Slosh Modeling

The problem of liquid sloshing in low gravity is complex and difficult to handle analytically. As discussed in the liquid trapping sections, analysis of even static cases can be difficult in low gravity, where the dominance of surface tension and wetting can result in a highly curved free surface. The problem is even more complicated in the dynamic case, where analytical solutions are nearly impossible. Those that do exist require very strong simplifying assumptions to obtain any results, and most works that discuss low-g sloshing focus on linear slosh, neglecting rotational slosh entirely. Analytical treatment of the topic generally requires assuming that the flow is inviscid and incompressible, and only subject to small perturbations, allowing the problem to be reduced to a tractable form. Results can only be obtained for the simplest tank geometries due to the complex boundary conditions involved. However, these analytical solutions only predict the natural frequencies of the sloshing motion and does not predict any damping at all (Reynolds & Satterlee, 1966). These results can be used to find an equivalent mechanical model to which linear damping can be added if rough estimates of damping times are required, though it is unclear how accurate such an approximation is (Dodge & Garza, 1967).

As a result, a different approach needs to be taken to consider real world tank geometries and to estimate damping. CFD is a particularly attractive option, as it allows prediction of the full 3D fluid motion without the need for strong simplifying assumptions. However, this benefit does come at the cost of significant computation time. To help build confidence in results, more experimental data needs to be gathered to compare to CFD. One recent example of the success of this approach is the NASA SPHERES-Slosh Experiment. The SPHERES-Slosh experiment flew on the International Space Station and was used to collect long term linear slosh behavior for liquids in propellant tanks in response to acceleration. Acceleration history of the tank was recorded during the experiment, which could be used as an input to CFD models. A CFD model using data from the SPHERES-Slosh experiment shows general agreement with experimental results but is not able to completely capture the details of the fluid motion (Lapilli, et al., 2015). A later analysis of the SPHERES-Slosh data by members of the same group using a different CFD model yielded similar results (Kirk, Storey, Marsell, & Schallhorn, 2017). These results are consistent with other studies comparing CFD to experiment, where CFD can capture general trends and some of the details of fluid motion, but significant discrepancies remain (Walls, Kirk, de Luis, & Haberbusch, 2011). While these results demonstrate how CFD can be used effectively when modeling low-g slosh, they also show some of its limitations. Currently, the complete physics of how fluids behave in low gravity is not fully understood, being unable to account for phenomena such as contact angle hysteresis. For these reasons, many more low gravity fluids experiments will be required so they can continue to provide test cases for CFD models. One avenue that has had minimal exploration yet is low-g sloshing in response to tank rotations about an axis perpendicular to the longitudinal axis. No major experimental or numerical studies have been performed investigating liquid response to this motion, despite applicability to satellite pointing maneuvers. There is little reason to expect CFD models to handle rotational sloshing significantly differently than linear sloshing, the only thing lacking is experimental data.

7. ROTATIONAL SLOSH EXPERIMENT DESIGN

Several design constraints are imposed on the experiment design by Virgin Galactic to operate on SpaceShipTwo. The two that are the most significant drivers of experiment design are the volume and weight limits. The maximum dimensions of the experiment, including payload box, are 18.50” W x 11.25” H x 21.50” D (Virgin Galactic, 2019). This is what drove the separation of the experiment into a tall and short side, using mirrors to view tanks on the opposite side. Mirrors allow the cameras to be located out of the way of other components while still being sufficiently far from the tanks to have each camera able to see two tanks without a high field of view lens that would introduce necessary distortion. A sketch of the experiment layout is shown in Figure 7.1. The weight limit for the experiment, including the payload box, is 50 lbs. (Virgin Galactic, 2019). Since the payload box design is not yet finalized, a conservative estimate of 20 lbs. is assumed, leaving a 30 lbs. weight budget for the rest of the experiment. This had to be considered for every component designed, most of which feature additional holes solely for weight reduction purposes. Though not as significant to overall experiment design, the maximum power draw of 50 W at 24-28 V (Virgin Galactic, 2019) serves as a constraint on the electrical system that had to be accommodated.

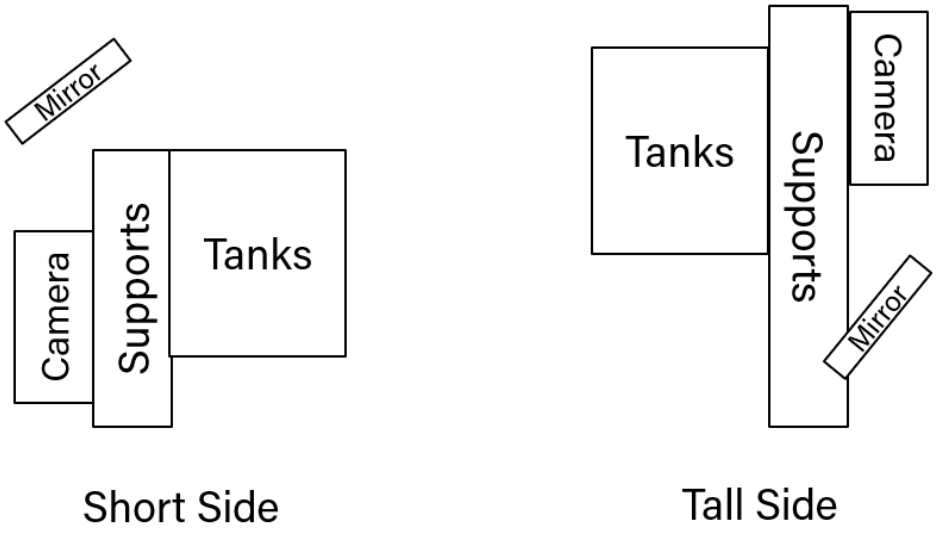

Figure 7.1. A sketch of the experiment layout.

7.1 Mechanical Design

7.1.1 Tank Design

The tank and vane geometries are scaled down from the propellant tanks used on the Cassini mission, which are typical of interplanetary satellite tanks. The design is also not export controlled, making it convenient to serve as a basis for the tank design in this experiment. The Cassini tanks have a diameter of 0.62 m (Enright & Wong, 1994), while the tanks used in this experiment have a diameter of 1.33”, resulting in a scaling factor of 18.35. The tanks have two hemispherical end caps joined by a cylindrical section. Eight vanes are placed inside the tank to hold propellant near the outlet, shown in Figure 7.2. The bottom of each individual vane lies flush against the hemispherical tank wall, shown in Figure 7.3. No filling or draining occurs through the outlet since the propellant volumes are fixed during flight. Instead, the tanks are filled while on the ground through a hole on the top of the tanks and sealed with a screw and gasket to prevent leakage. The vanes are fixed to the end cap using a wire inserted from the side of the tank, which is kept in place by gluing a small acrylic rod in from the side. Additional glue is added at the base of the vanes to prevent twisting. Both features can be seen in Figure 7.3. Each half of the tank is glued together once the vanes are fixed in place, then polished to ensure clear view of the liquid inside. Acrylic brackets are glued to the tanks, which are used to hold bolts running the length of the tank to ensure that the two tank halves do not separate during flight, which would leak liquid inside the experiment. The brackets are also used to mount the tanks to an aluminum backplate, which is shown along with the brackets in Figure 7.4. Slots are added to the top and bottom of the tank CAD model to serve as additional weight reduction if needed. These will only be machined if the total weight exceeds the weight limit and can be seen in Figure 7.4. The two outer tanks require chamfers on their outer edges to ensure the tanks fit completely within the second containment when rotated. Figure 7.5 shows the rotated tanks relative to the second containment wall.

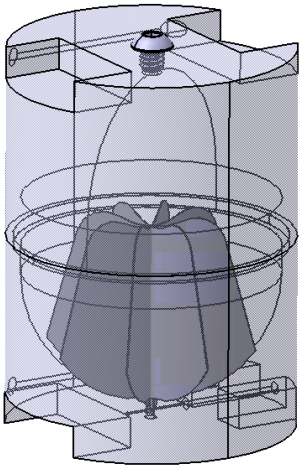

Figure 7.2. A CAD model of one of the tanks and PMD vanes used on this experiment.

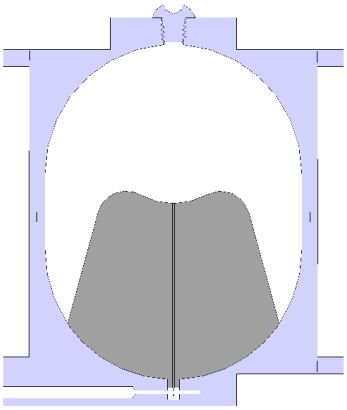

Figure 7.3. A section view of the tank demonstrating how the vanes lie flush against the tank wall and how the vanes are connected to the tank.

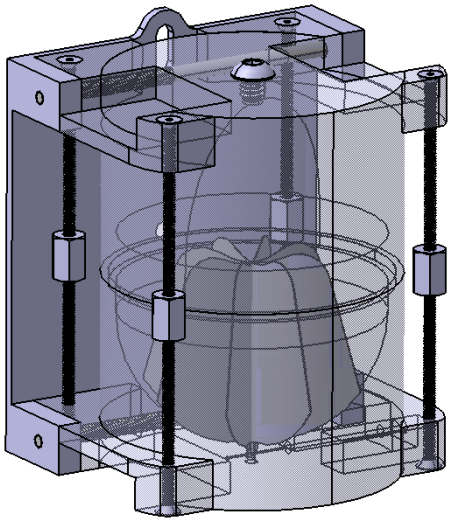

Figure 7.4. A tank with mounting hardware attached.

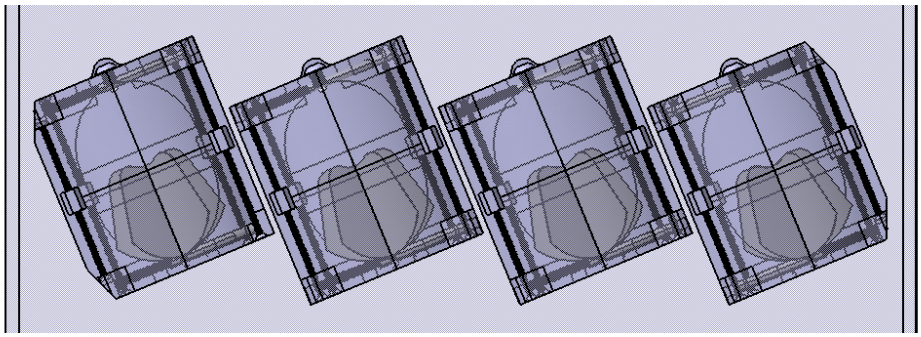

Figure 7.5. Tanks rotated inside the second containment, demonstrating the need for chamfers on the outer two tanks.

7.1.2 Moving Components

The tank backplates are attached to the rest of the assembly by two pivots that are used to rotate the tanks. One pivot is a simple hole located at the center of the backplate, while the other is a slot located near the top of the tank and can be seen in Figure 7.6. The pivots are each attached to bars that can move independently from side to side. By moving one bar horizontally while holding the other stationary, the tanks can be rotated about either pivot. The tanks are attached to the bar using shoulder bolts, providing a smooth surface to help minimize friction. To further minimize friction, PTFE washers are used on either side of the backplate, helping ensure consistent tank rotation. A section view is shown in Figure 7.7.

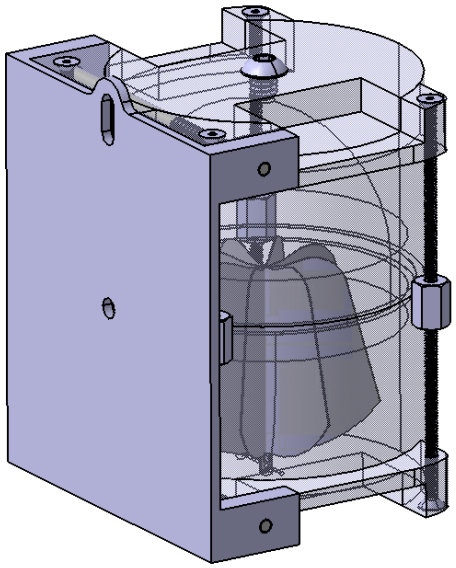

Figure 7.6. Rear view of the tank mounting plate showing the two pivots that are used to rotate the tank.

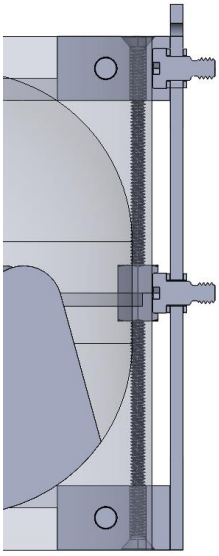

Figure 7.7. A section view of a tank and mounting plate showing the shoulder bolts and PTFE washers.

Each bar is attached to a stationary truss wall located on the opposite side of the tanks. Two linear guides per bar are used to affix the bars to the truss wall, with the rail attached to the bars and guide attached to the truss wall. The locations of the linear guides can be seen in Figure 7.8, and the way they interface with the truss wall is shown in Figure 7.9. A view of how the bars move

when the tanks are rotated can be seen in Figure 7.10. Ball bearings fit between the rail and guide to minimize friction and allow smooth translation of the bars. The two bars interface with the motors by means of a rack and pinion. The rack is screwed onto the bars, while the pinion is attached to the motor gearbox output. The motors are oriented in opposite directions to reach each bar as shown in Figure 7.11. The motors themselves are attached to the truss wall using a custom machined bracket that is screwed onto the end of the gearbox, shown in Figure 7.12.

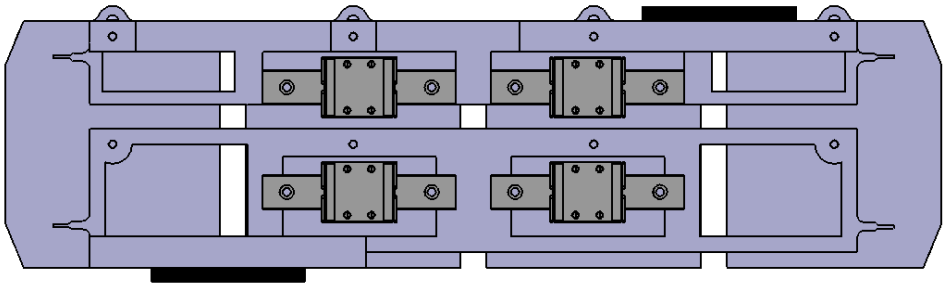

Figure 7.8. Bar assembly orientation and linear guide positioning.

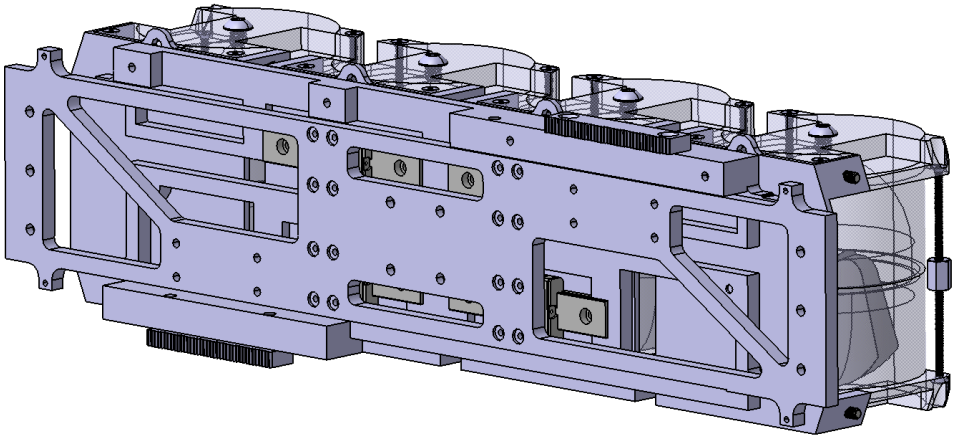

Figure 7.9. Truss wall location and attachment to linear guides.

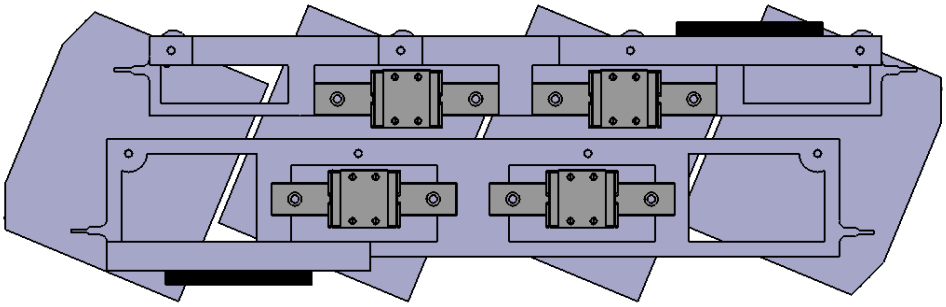

Figure 7.10. Bar assembly position when tanks are rotated.

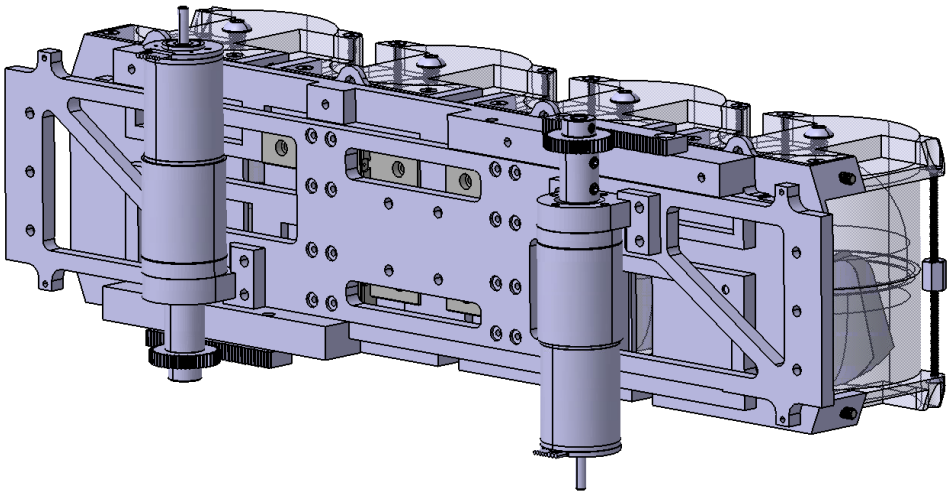

Figure 7.11. Motor attachment to the truss wall and rack and pinion interface with the bar assemblies.

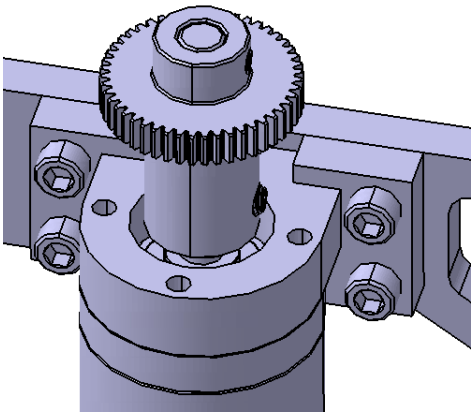

Figure 7.12. Closeup of a motor mount.

Limit switches provide absolute positional feedback to rotate the motors consistently. Four limit switches are attached to each truss wall for a total of eight, one for each direction and for each bar. An extension on each bar slides inside the limit switch, interrupting a laser and sending a signal to the Arduinos controlling the motors, which then stops the motor rotation. This set up is shown in Figure 7.13.

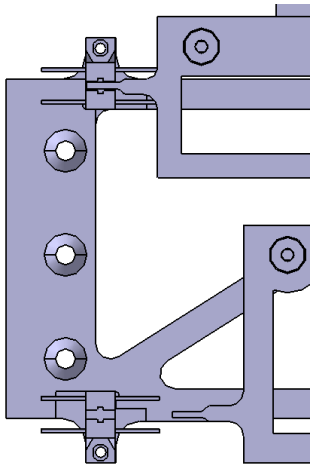

Figure 7.13. Limit switch mounting and interface with the bar assembly.

7.1.3 Structural Components

The truss wall is attached to three pylons, which rigidly connect the whole assembly to a baseplate and serve as mounting points for the cameras and mirror brackets. This is also where the asymmetries between the tall and short side become noticeable. For the tanks to be visible to the opposite side's cameras, the entire assembly discussed up to this point, including the truss wall, tanks, motors, and bar assemblies, are located at different heights for each side. On the tall side, tanks are placed near the top of the payload box and away from the baseplate as shown in Figure 7.14, while on the short side the tanks are placed near the baseplate, as shown in Figure 7.15. This allows the height of the mirrors to align with the height of the tanks on the opposite side, as shown in Figure 7.16. Cameras are attached to the top of the tall side middle pylon using a custom machined aluminum bracket, while the mirror brackets are attached near the bottom of the tall side middle pylon and can be seen in Figure 7.17. The short side assembly is the exact opposite, with the cameras attached to the bottom of the middle pylon, while the mirror brackets are attached to the top, shown in Figure 7.18. The pylons on the left and right of the short side assembly also have

notches removed from their bases to accommodate the dimensions of the payload box, which will be discussed in more detail in the second containment section.

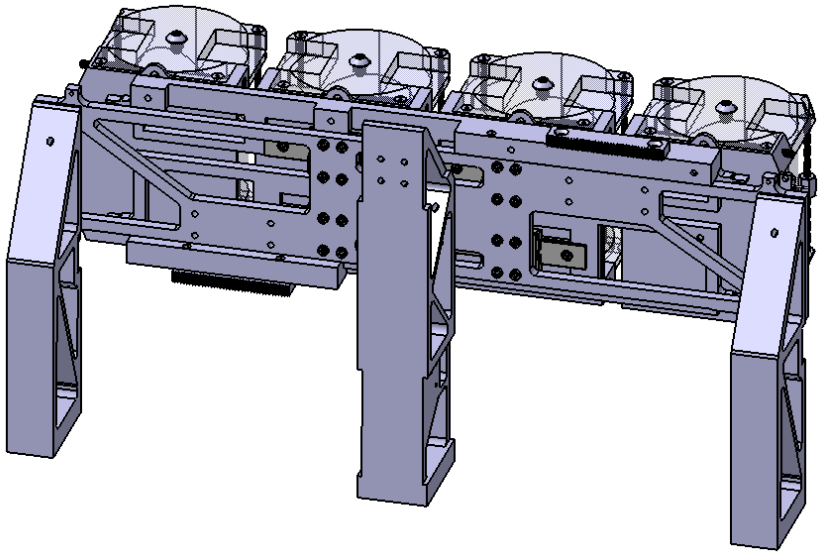

Figure 7.14. Tall side pylon attachment to truss wall.

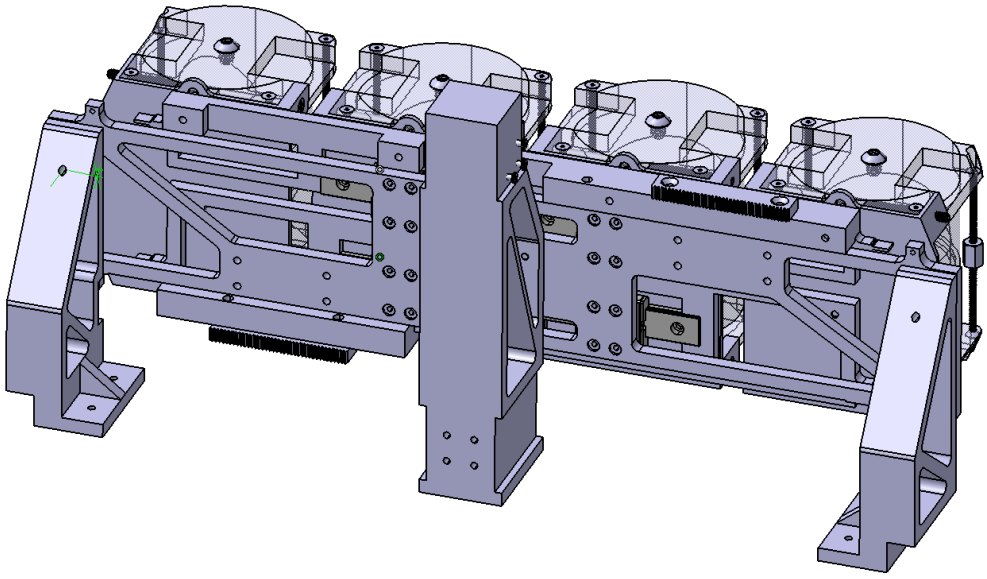

Figure 7.15. Short side pylon attachment to truss wall.

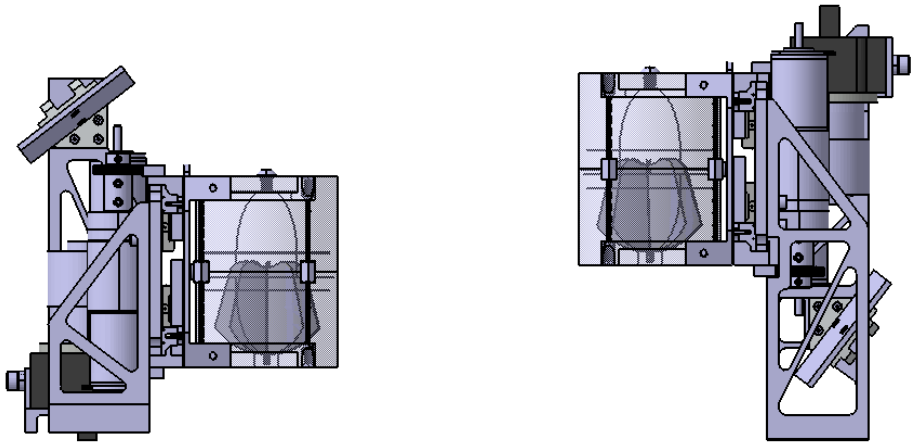

Figure 7.16. Orientation of the two side assemblies relative to each other.

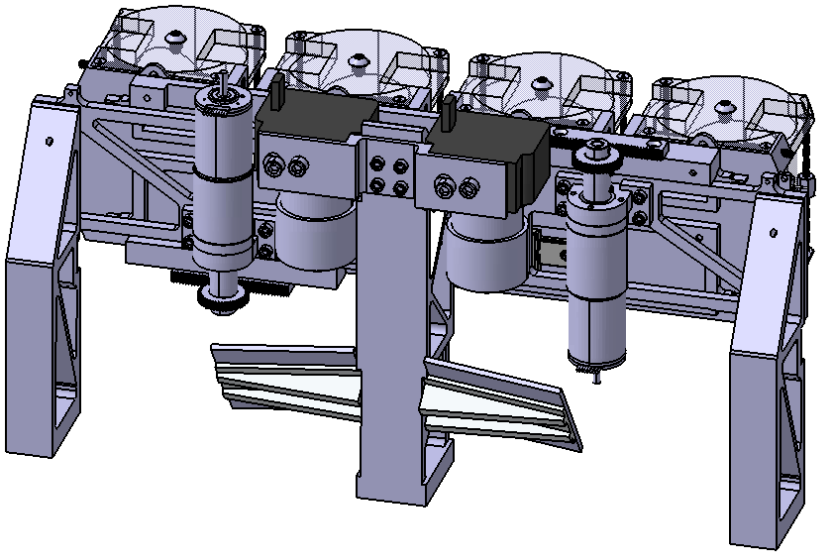

Figure 7.17. Camera and mirror attachment to the tall side middle pylon.

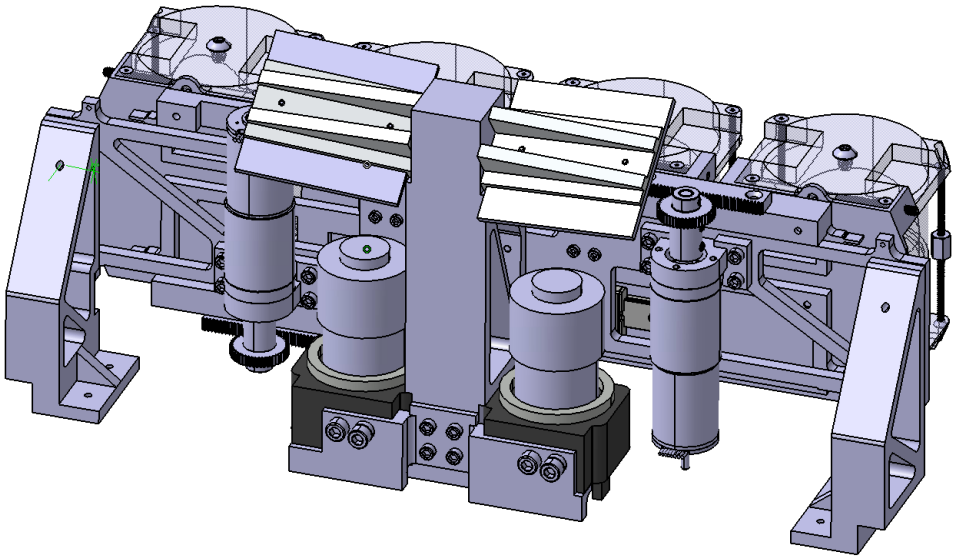

Figure 7.18. Camera and mirror attachment to the short side middle pylon.

Due to their large size, special care had to be taken to minimize the weight of the pylons and truss walls. As much material is removed as possible from these components by adding pockets, leaving only material where needed. An example of this is shown in Figure 7.19 for the tall side middle pylon. For each camera to have an unobstructed view of two tanks on the opposite side, the mirror brackets must be angled outward since they are closer to the middle pylon than the tanks that they need to view. Initial estimates for the required angles are estimated using the CAD model and known camera field of view and dimensions. The parts involved are then 3D printed to test the angles with the actual camera and lens. Based on the camera view, angles are iteratively adjusted until the entirety of the opposing tanks could be seen. This method helped minimize uncertainty, lending more confidence to the mirror setup. Another problem faced by the mirror brackets is that they are relatively thin and span far from the middle pylon from a cantilevered support. This led to concerns regarding vibrations in the mirror brackets during launch. A cursory vibration analysis using SolidWorks resulted in the addition of stiffening ribs to ensure the natural frequencies of the mirror bracket are outside the range of those that will be experienced during launch. These stiffening ribs are shown in Figure 7.20.

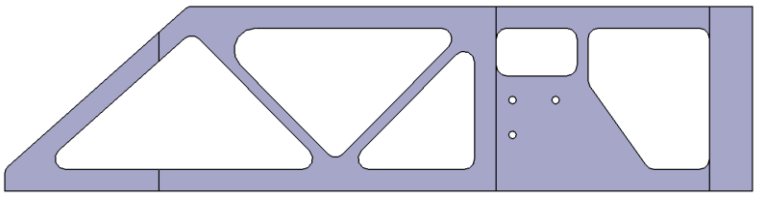

Figure 7.19. Tall side middle pylon weight reduction holes.

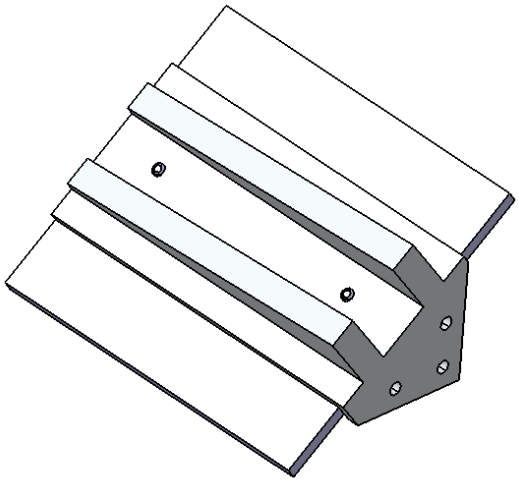

Figure 7.20. Mirror bracket with stiffening ribs with mirror attached.

7.1.4 Second Containment

Some form of second containment is required to capture any stray liquid in the rare event that a leak occurs in one of the tanks. The tall side and short side each have their own second containment, enclosing that side's tanks, cameras, and motors. Each second containment seals against a 0.5" thick aluminum baseplate, which is also used to mount the experiment to the payload box. The payload box has an array of holes that screw on from under the baseplate. These are blind tapped holes to maintain a watertight seal and are the main driver of the baseplate thickness. This thickness also makes the baseplates an ideal location to pass wires into the second containment needed for power and communication with external electronics. A small blind hole will be drilled in the side of the baseplate that matches up with a similar blind hole on the top of the baseplate. This creates a passage for wires to enter that can be sealed with a connector. The baseplates are some of the heaviest components in the experiment, and so special care is taken to minimize their weight as much as possible. Luckily, the full thickness of the baseplate is only needed at hole

locations, along with a few connecting ribs to provide additional rigidity. Everywhere else, the thickness is reduced to 1/8”, saving approximately 2.3 lbs. for each side. This configuration is shown for the short side in Figure 7.21.

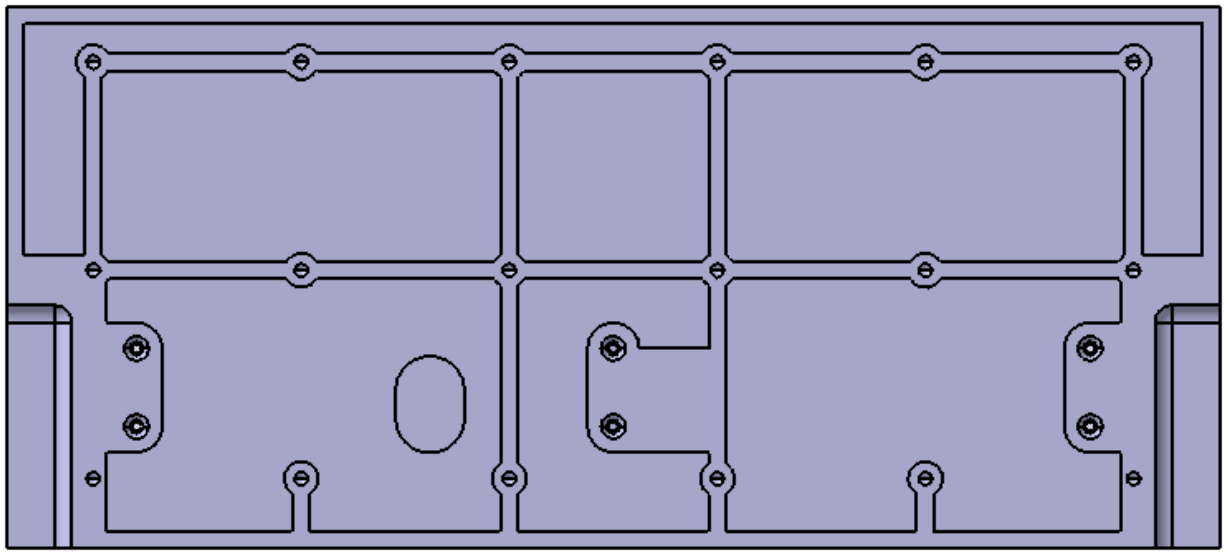

Figure 7.21. Weight reduction pockets on the short side baseplate.

The rest of the experiment is connected to the baseplate by the pylons, which are screwed in from the bottom of the baseplate. Since they are attached using through holes, gaskets are compressed underneath the pylons to maintain a watertight seal. The main wall of the second containment is made from 10 thousandths of an inch thick acrylic sheet. Since the second containment wall carries essentially no load, the wall can be very thin to minimize weight. Each second containment will have the acrylic wall cut out of a single large sheet, which can be bent to the appropriate shape. The acrylic wall is attached to the baseplate with 90° aluminum angles along the baseplate perimeter. The aluminum angle’s vertical face is glued to the acrylic wall, while the other face is screwed onto the baseplate, compressing a gasket to form a watertight seal. The second containment for the tall side can be seen in Figure 7.22.

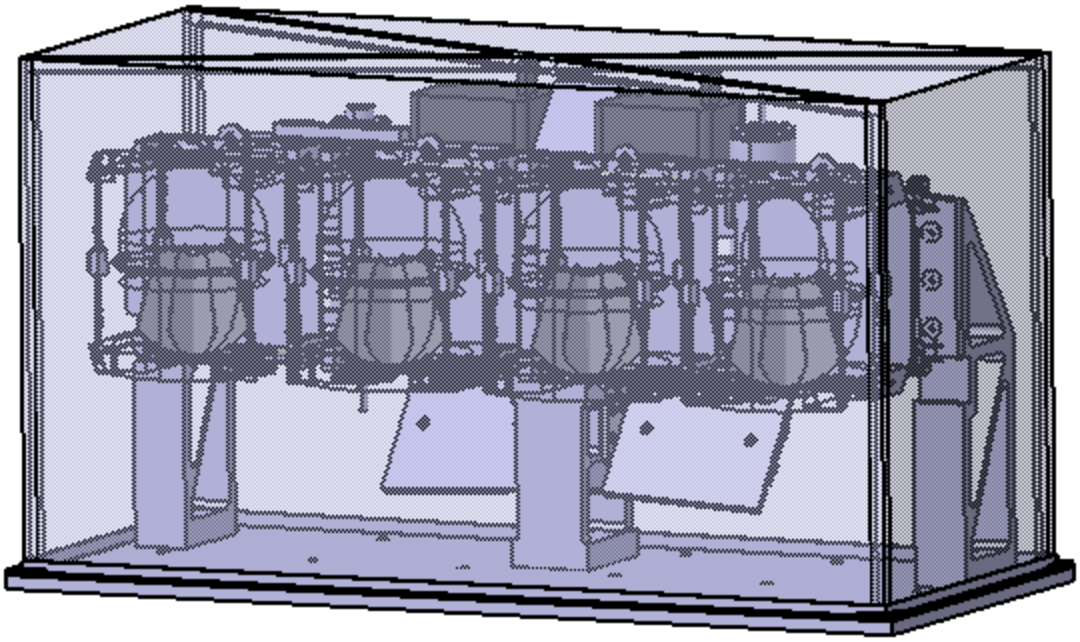

Figure 7.22. Tall side assembly with the second containment and baseplate.

The short side second containment is complicated by two corner notches that accommodate the mounting of the payload box to SpaceShipTwo. The two side pylons go over the top of the notch, preventing the second containment from simply going around the notches. Instead, the second containment goes over and seals against the top of the notches. These elevated sealing surfaces are connected to the baseplate using sloped sealing surfaces to create a single continuous seal along the perimeter of the second containment as shown in Figure 7.23. Each notch has one sloped sealing surface in front of the notch on the side of the second containment and another to the side of the notch on the rear of the second containment. The short side baseplate will be machined out of a single piece of aluminum and will include the notches. The short side second containment and baseplate can be seen in Figure 7.24.

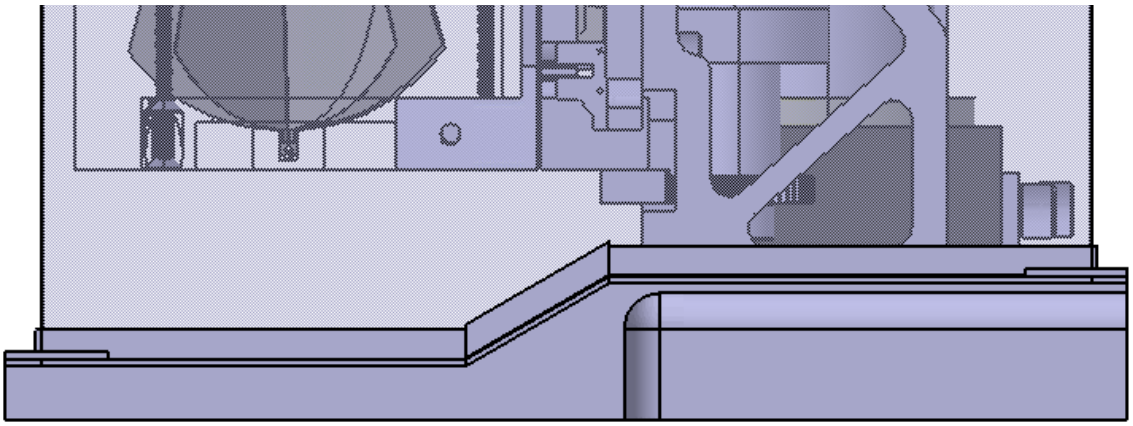

Figure 7.23. The short side second containment accommodates the notch by adding a vertical sealing surface.

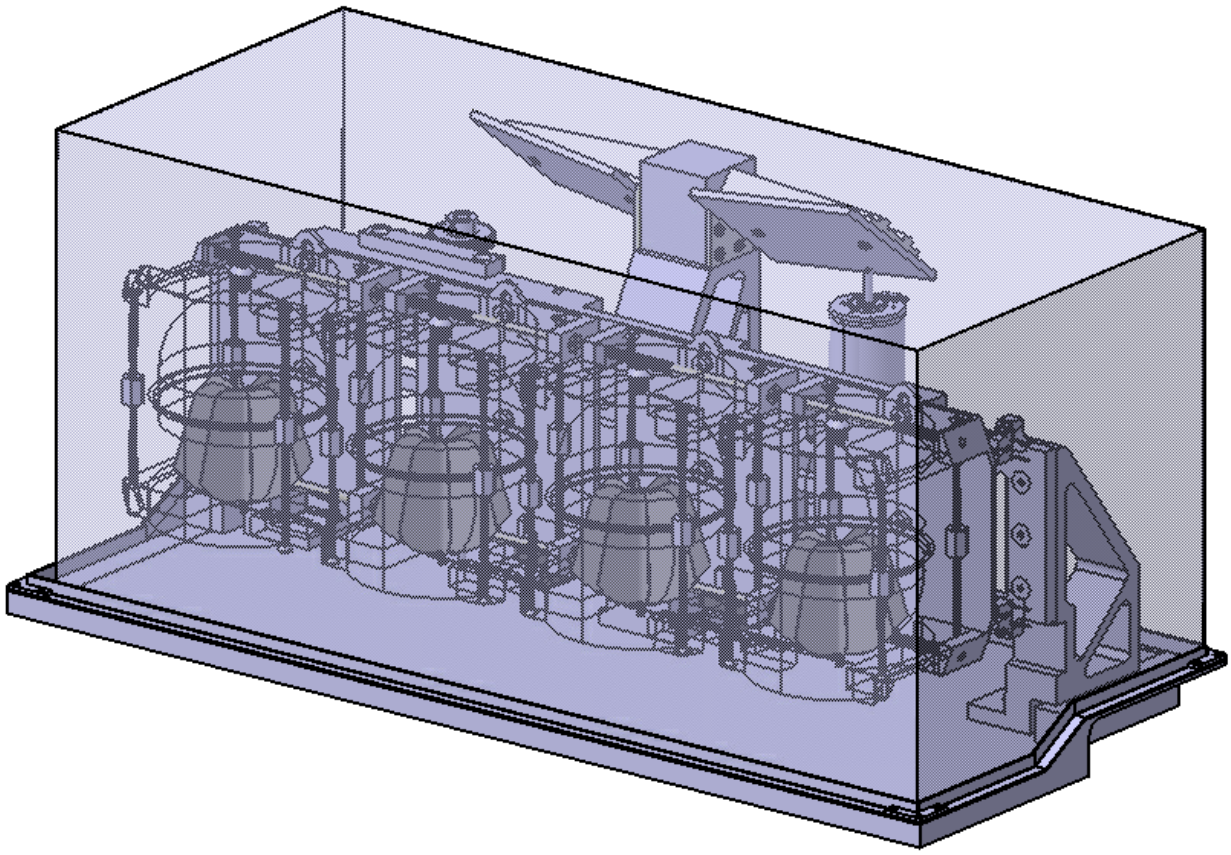

Figure 7.24. The short side assembly with the second containment and baseplate.

7.2 Electrical Systems

The primary functions of the electrical system are to power and control the motors and cameras. The four cameras need to be able to be controlled remotely while being charged. The four motors are each controlled by its own motor control board that handles routing of power and provides a simple interface for controlling the motors with an external device. Controls are handled by three Arduinos that connect to the motor control boards, cameras, and to each other. An accelerometer is used to detect when zero-g time begins and determine when to trigger the cameras to record and the motors to move. Eight limit switches are also used to provide absolute positional feedback to the motors, increasing accuracy of rotations. The motor control boards and Arduinos are located at the bottom of the gap between the tall side and short side second containments, making wiring through the second containment baseplate simple.

The electrical system must be able to route the power provided by Virgin Galactic's SpaceShipTwo to each component at the appropriate voltage. The SpaceShipTwo provides up to 50W power at 24-28V for use by the experiment (Virgin Galactic, 2019). The motors can run on any voltage between 9 and 50 V DC, so the provided power can be connected directly to the motor control board. The Arduinos all run at 9 V DC, so a 9 V DC-DC converter is required to power them. Another 5 V DC-DC converter is used both to charge the cameras and to power the accelerometer. High efficiency DC-DC converters are required to minimize the waste heat produced by the experiment.

The Arduino Uno Rev3 is chosen for its small footprint, large number of ports, extensive documentation, and general ease of use. One Arduino will be used to control all cameras, another will be used to control half the motors and limit switches, and the last serves as a master controlling the other two Arduinos while also handling the other half of the motors and limit switches. The master Arduino also manages the accelerometer, which it uses to determine when to signal to the other two Arduinos when to begin recording or running the motors. The accelerometer used is the ADXL345, which is chosen primarily for its ease of use, small size, and minimal IO requirements. High precision is not required since it is only checking for a continuous period of low gravity to begin the experiment, though the acceleration history may be logged as well to aid in data analysis. The camera control Arduino's fourteen digital IO pins are enough to control all cameras while communicating with the master Arduino, because each camera needs only three digital IO pins. Each motor control board also requires three digital IO pins from the Arduino; however, this does

not leave enough available IO for the limit switches. Rather than connect the limit switches to the master Arduino and then send a signal to the motor control Arduino to stop running the corresponding motor, half the motors and limit switches are assigned to the motor control Arduino while the other half is assigned to the master Arduino. This minimizes the delay between a limit switch being triggered and the motor receiving the signal to stop rotating, helping keep rotations consistent. A diagram showing how each component is powered and connected is shown in Figure 7.25.

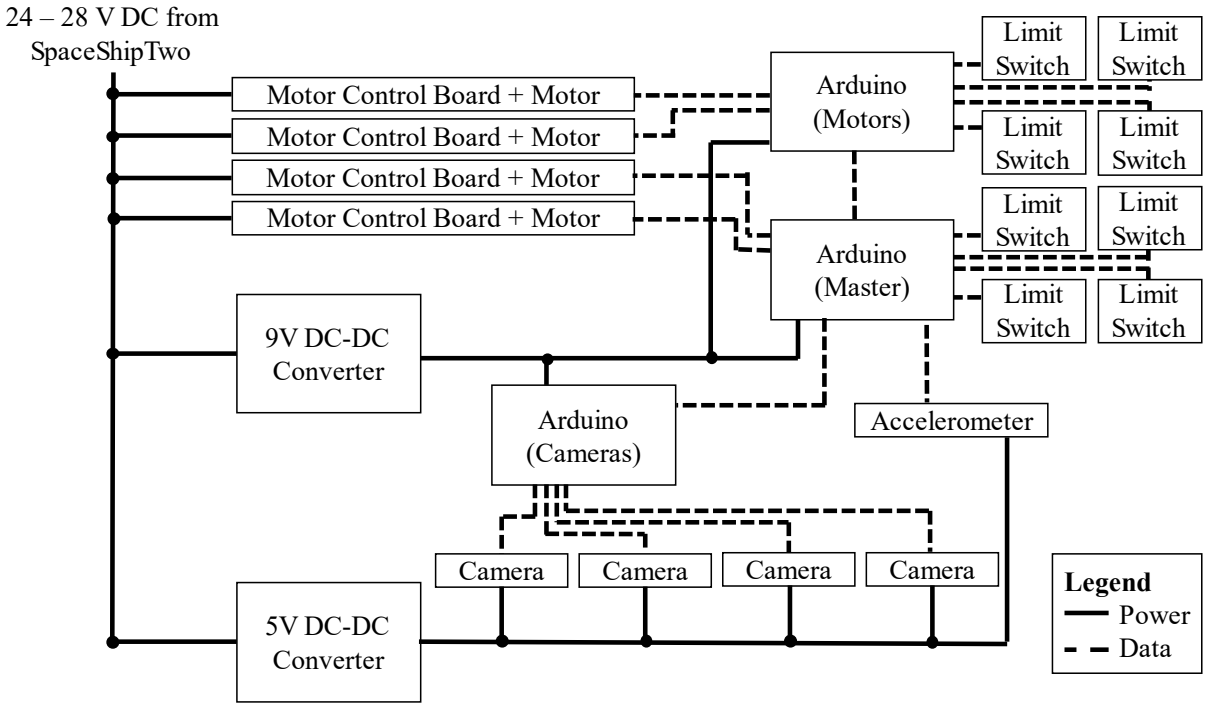

Figure 7.25. Diagram showing power distribution to each component in the electrical system along with data transfer between components.

7.2.1 Camera System

This experiment will use four Sony RX0 II cameras with the Ribcage modification from Back-Bone. These cameras can record 4K footage and come with a small form factor. The Ribcage modification offers a CNC machined mounting plate that provides a simple and secure connection point. Three major challenges needed to be solved to use these cameras during the experiment. The first and simplest is to be able to charge the camera's battery using power supplied by SpaceShipTwo. The experiment may sit idle for a few days on SpaceShipTwo after the payload is

loaded, which could completely drain the camera batteries. The second is to power on or off the camera based on an Arduino signal. The last is to remotely trigger the camera to begin or stop recording based on an Arduino signal. These cameras come with simple built-in ways to accomplish all of these, allowing charging over a Micro-USB cable and power on/off and recording capabilities over Bluetooth. However, Bluetooth cannot be used on the flight due to electromagnetic interference, eliminating the normally recommended remote control method for the RX0 II.

Unexpectedly, the Micro-USB compatible port on the camera that is used for charging and communicating with the computer is not a Micro-USB connector, and instead is a proprietary multiport connector from Sony. This connector adds a few extra pins in addition to the normal USB connections. Sony does not provide specifications for these connectors; however, others have reverse engineered their function. A third party supplier of breakout boards for the Sony multiport connectors lists the function of each pin on their website (STUDIO1productions, n.d.), shown in Table 7.1. These breakout boards also provide a convenient way to interface with the cameras since wires can be soldered to the breakout boards to interface with the rest of the electrical system.

Table 7.1. Functions of each pin on the Sony multiport connector.

Pin	**Function**
A1	VBUS
A2	USB D-
A3	USB D+
A4	USB ID
A5	Ground
1	Power On / Off
2	Ground
3	Composite Video Out
4	Audio Out L / Shutter Release
5	Audio Out R Audio / Activate Camera / Focus
6	Select
7	UART RX
8	UART TX
9	XReset Req
10	3.3 Volt Output

In addition to the usual USB connections, these connectors include a pin for power on/off and a pair of wires for serial communication with the cameras. When the power on/off pin is shorted to ground, the camera will turn on or off. The serial connection is used to trigger the camera to begin or stop recording, however no official documentation is available from Sony on the communication format. Sony sells camera remote control units that are compatible with the RX0 II and use the multiport connector to remotely control the camera's functions. These functions include powering on/off and recording, activated by pressing physical buttons on the remote control unit. The physical buttons on the remote control can be bypassed by opening the case to access the circuit board inside. Based on the board layout obtained from a YouTube video by user Elliot Lowndes (Lowndes, 2017), which can be verified by probing the circuit board during use, wires can be soldered to the circuit board to control the cameras electronically. By shorting these wires to ground, the effects of pressing the buttons on the remote control unit can be replicated exactly. The Arduino can control when these are shorted by connecting the wires to its digital IO pin and ground. This allows the camera to be turned on/off or begin/stop recording remotely, however there is no way to charge the camera while using the remote control as intended because the camera's charging port and communications share the same connector. This problem is solved by cutting off the remote control's connector and soldering them to the multiport breakout board. Of course, this requires knowing what each individual wire on the remote control board needs to connect to, which can be determined by manually probing each pin with a multimeter while using the remote control. Once the wires are identified and connector soldered to the remote control unit, extra wires can be added for charging following the USB Battery Charging Specification. Since no USB communication is needed, the connector is designated as a dedicated charging port by shorting the two USB data lines D+ and D- together. The USB power pin can be connected to 5V and the USB ground to ground. These connections are summarized in Table 7.2.

Table 7.2. Connections from the multiport connector to power and the remote control unit.

Multiport Pin	Remote Control Pin	Function
1	P4	Power On / Off
2	P13	Ground
6	P7	Select
7	P6	UART RX
8	P5	UART TX
10	P1	3.3 V

Using the camera remote control unit is simple and can consistently trigger the camera, however it does require the payload to contain four of the remote control boards, adding unnecessary weight. The Arduinos that are used to determine when to start recording are already capable of serial communication that could be used trigger the camera recording and they are also capable of shorting the power on/off pin to ground. A GitHub page from Anton Slooten was invaluable in this process, as they had already reverse engineered the serial communications sent between the camera and computer during the recording process. When beginning communications, the camera first checks for a 100 kΩ resistor between a selector pin and ground (Slooten, 2017). With the addition of this resistor, the two serial communication lines can be connected directly to two of the Arduino's digital IO pins, allowing serial communication between the camera and Arduino using the Arduino's SoftwareSerial library using commands reverse engineered by Slooten. These connections are summarized in Table 7.3.

Table 7.3. Connections from the multiport connector to power and the Arduino.

Multiport Pin	Destination	Function
A1	5V in	VBUS
A2	Shorted to A3	USB D-
A3	Shorted to A2	USB D+
A5	Ground	Ground
1	Arduino Digital IO	Power On / Off
2	Ground	Ground
6	100 kΩ to Ground	Selector
7	Arduino Digital IO	UART RX
8	Arduino Digital IO	UART TX

7.2.2 Motor System

This experiment uses four Maxon EC-I 30 45W brushless motors. These motors are primarily selected for their known low electromagnetic interference, small profile, high torque, and familiarity. These motors can provide much more torque than is required to rotate the tanks, preventing the motors from being moved by forces caused by the liquid sloshing inside the tanks. The required gearbox reduction can be easily calculated from the desired tank rotational speed discussed at the beginning of this section. The tank rotational speed can be used to compute the linear speed of the bar, which can then be used to find the motor speed. Setting the motor speed to be near its maximum for the fast tank rotation speed yields a reduction of 318.

The motors are controlled using a separate motor controller board from the motor manufacturer Maxon, one board for each of the four motors. Each motor board takes in power to route to a motor along with signals indicating if the motor is enabled, and if so its direction and speed. The direction and enable pins are simple digital inputs that can be set to either high or low by the Arduino. The speed pin, on the other hand, takes in an analog signal from 0V to 5V, with 0V corresponding to no rotation and 5V corresponding to maximum speed. However, the motor behaves poorly at low commanded speeds, so the enable pin must be disabled when the motor is not in use rather than simply setting the speed pin to 0V. Based on testing, a PWM signal can be used for the motor speed input instead of an analog signal without degrading motor performance. This allows the speed pin to be connected directly to the Arduino PWM out rather than through a digital-analog converter first, saving weight. The connections between the motor control board and Arduino are summarized in Table 7.4.

Table 7.4. Connections from the motor control board to power and the Arduino.

Motor Control Board	**Destination**	**Function**
J1 – 1	Ground	Power Ground
J1 – 3	Power In	Vcc 9-50VDC
8	Arduino PWM Out	Setspeed
5	Arduino Digital IO	Direction
4	Arduino Digital IO	Enable
1	Ground	Ground

8. ROTATIONAL SLOSH SUMMARY

Accurate predictions about liquid slosh behavior in low gravity are difficult to make. Currently, very little data exists regarding rotational slosh in satellite propellant tanks while in orbit, making improvements on existing models difficult. Additional experimental data can be used to help calibrate models and improve their accuracy, which can translate into improved fuel efficiency for satellites undergoing pointing maneuvers in orbit. The rotational slosh payload will record video of liquid slosh in scaled down tanks while being rotated about different pivots and at different speeds. Damping rates and fluid positional information can then be extracted from the video feed to serve as a point of comparison for future computational models

9. CONCLUSION

Liquid trapping can occur in any vaneless tank with non-hemispherical end caps regardless of contact angle. Characterization of different propellant distributions can be used to help understand why liquid trapping occurs, and therefore how it can be avoided. A combination of Surface Evolver and analytical solutions are used to identify these conditions for a wide range of contact angles and a few end cap geometries, including hemispherical, ellipsoidal, superellipsoidal, and torispherical domes. Surface Evolver in particular is a useful tool for generating these results, since once set up it can be used to compute each type of solution quickly and easily for a wide range of fill fractions and contact angles. Surface Evolver does suffer from some amount of discretization error; however, this can be minimized by using an appropriate number of facets and performing sufficient iterations to ensure the free surface is fully converged. When using non-hemispherical end caps, understanding of how liquid propellants interact with the bare tank, especially with respect to liquid trapping, is critical for successful vane design.

Additional work is still required on the rotational slosh payload before flight. Several parts still need to be machined, including the entirety of secondary containment. Tanks need to be polished and assembled. The rest of the experiment also needs to be assembled, and weight estimates need to be verified using final parts to ensure the weight budget is not exceeded. Testing of camera remote triggering still needs to be completed. Arduino code needs to be written for the mission plan, which itself stills needs to be finalized. The details of how features will be extracted from the recorded video need to be determined. Since the primary time of interest is when the fluid is damping after the tanks have already rotated, the liquid should be the only thing moving in the camera feeds. This could make edge detection possible on the fluid to track the free surface position and find damping rates. The presence of bubbles in the liquid would make such an approach difficult, but the slow rotation speeds involved may prevent bubble formation. Video can be recorded during ground tests once the experiment is assembled to check lighting conditions and test the accuracy of these methods. Another avenue that may be worth pursuing is the addition of extra data logging to the experiment. Inertial Measurement Units (IMUs) could be added to the tanks to provide a more detailed acceleration history, isolating the movement of the tanks from the experiment and any play involved in the mechanical connections with the motor. If this data is used as a point of comparison for any computational models, an acceleration history that can serve

as an input into the model would reduce sources of error, providing a more accurate understanding of model performance. IMUs do not need to be added to every tank to be useful, either. If during ground testing it is determined that the tank motion is very similar across tanks, a single IMU on each side might be used to obtain accurate motion data for all tanks. The usefulness of such a feature depends on the backlash present in the system and repeatability of the motors and limit switches in rotating the same amount each time. The inclusion of IMUs for additional data logging will largely be decided by whether there is sufficient weight in the weight budget and IO on the Arduinos.

Modeling of static equilibrium propellant configurations can be used to avoid liquid trapping and aid in vane design. A lack of experimental data for the dynamic case of rotational slosh poses a problem for validation of future models making predictions about liquid response to this motion. The rotational slosh experiment aims to collect data that will fill part of this knowledge gap. Prediction of low-g propellant motion in satellite propellant tanks can be difficult. However, successful modeling of these phenomena can lead to improved vane designs, better fuel efficiency, and many more advancements, making the significant effort required worthwhile.

APPENDIX A. ADDITIONAL FIGURES

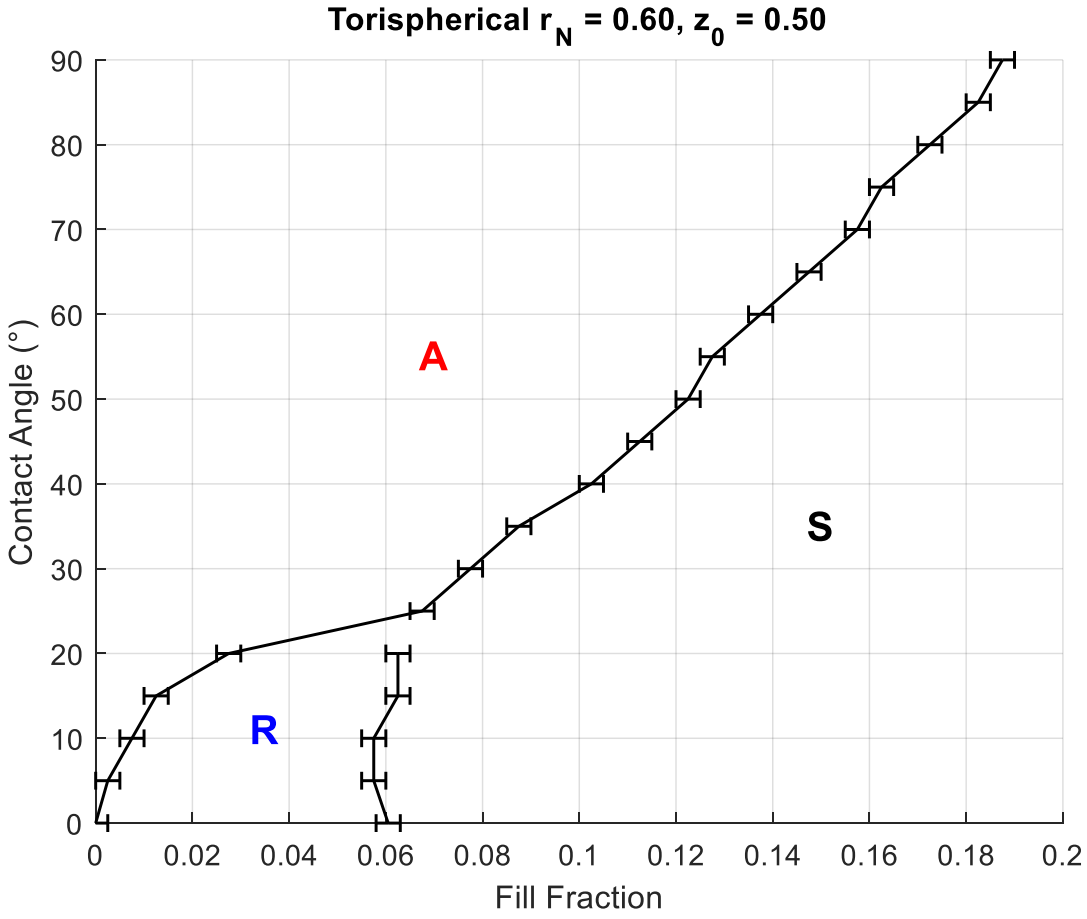

Figure A.9.1. Minimum energy propellant configurations for a tank with 2:1 torispherical end caps with $r_N = 0.6$. The region marked with S denotes the combinations of fill fraction and contact angle where the spherical interface is the minimum energy, the region marked with R denotes the combinations where the liquid ring is the minimum energy, and the region marked with A denotes the combinations where the asymmetric droplet is the minimum energy. The horizontal error bars show the fill fraction step size.

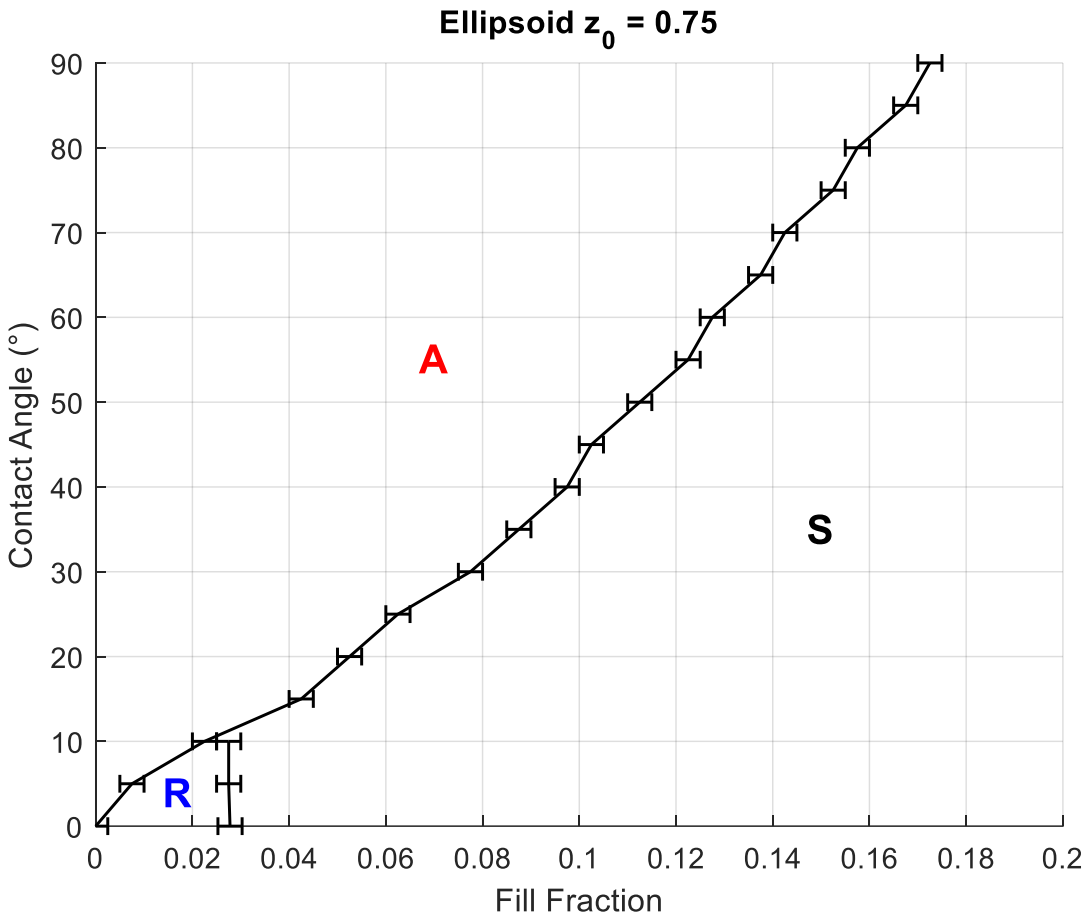

Figure A.9.2. Minimum energy propellant configuration for a tank with 4:3 ellipsoidal end caps. The region marked with S denotes the combinations of fill fraction and contact angle where the spherical interface is the minimum energy, the region marked with R denotes the combinations where the liquid ring is the minimum energy, and the region marked with A denotes the combinations where the asymmetric droplet is the minimum energy. The horizontal error bars show the fill fraction step size.

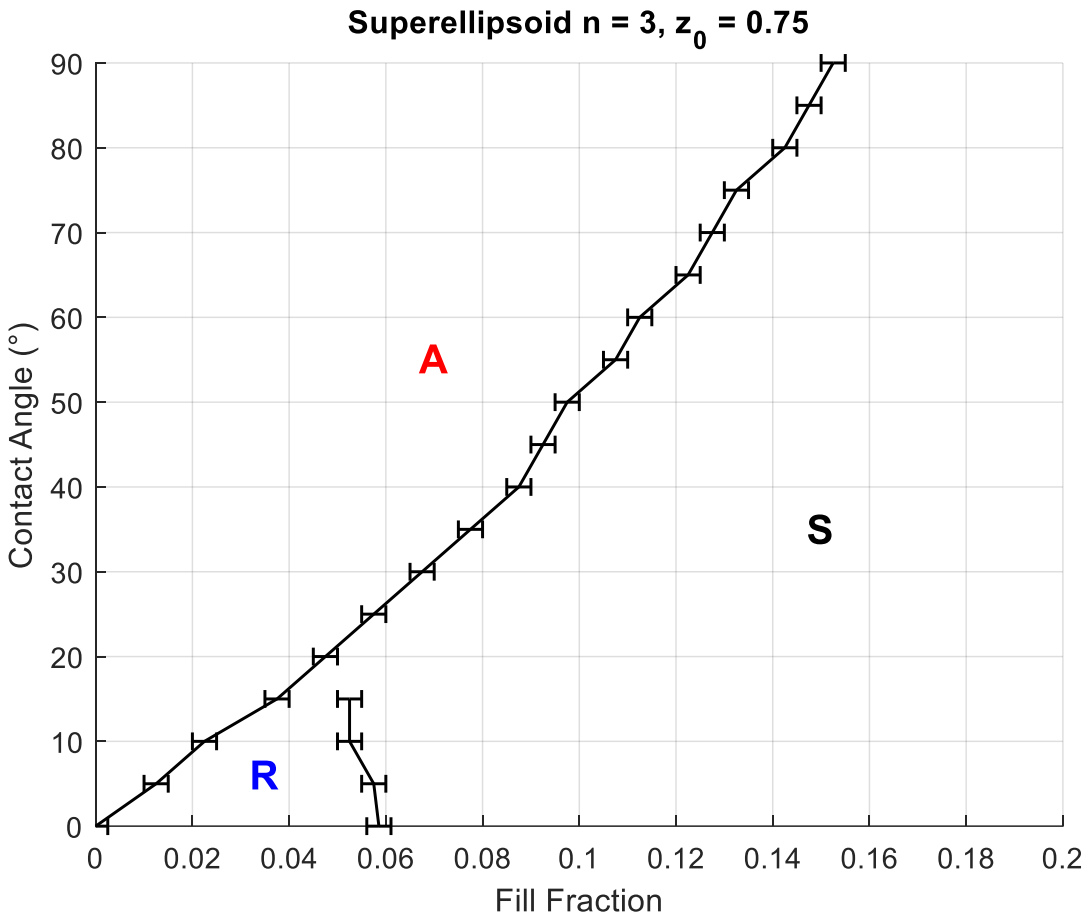

Figure A.9.3. Minimum energy propellant configurations for a tank with 4:3 superellipsoidal end caps with n = 3. The region marked with S denotes the combinations of fill fraction and contact angle where the spherical interface is the minimum energy, the region marked with R denotes the combinations where the liquid ring is the minimum energy, and the region marked with A denotes the combinations where the asymmetric droplet is the minimum energy. The horizontal error bars show the fill fraction step size.

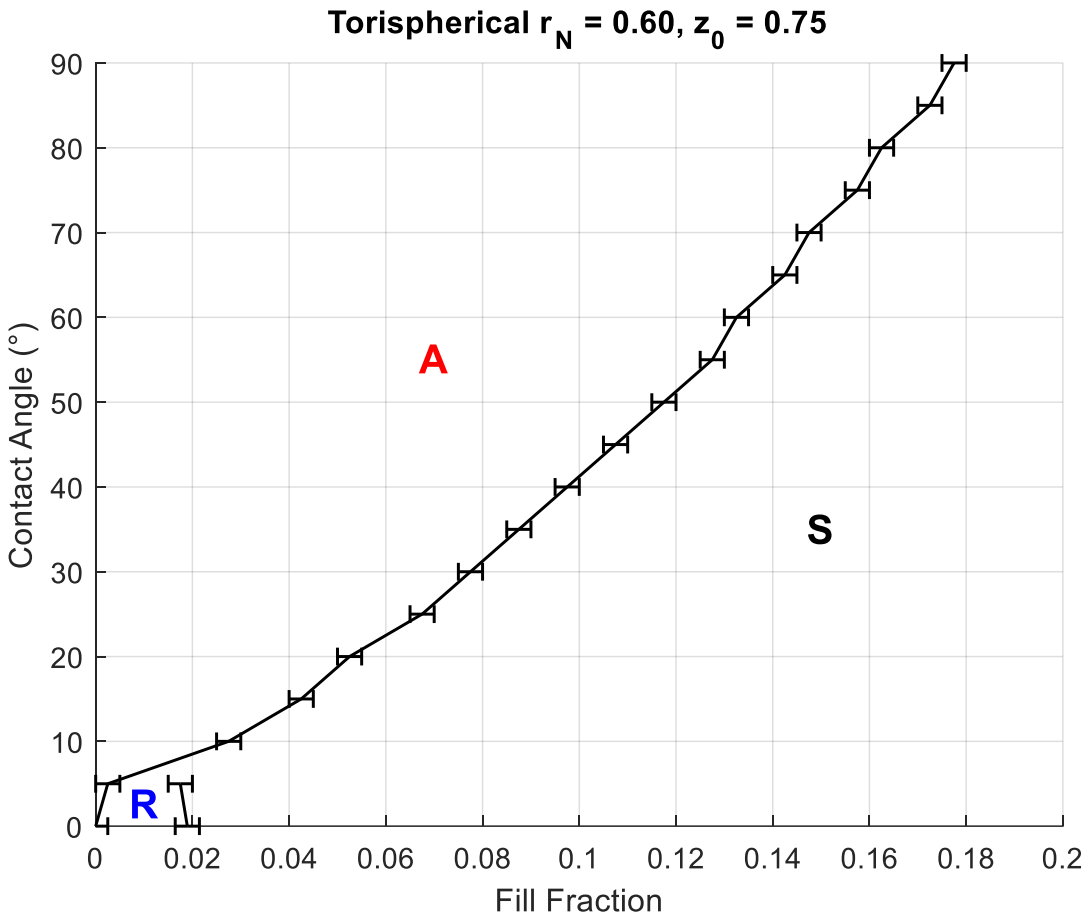

Figure A.9.4. Minimum energy propellant configurations for a tank with 4:3 torispherical end caps with $r_N = 0.6$. The region marked with S denotes the combinations of fill fraction and contact angle where the spherical interface is the minimum energy, the region marked with R denotes the combinations where the liquid ring is the minimum energy, and the region marked with A denotes the combinations where the asymmetric droplet is the minimum energy. The horizontal error bars show the fill fraction step size.

APPENDIX B: DERIVATION OF WETTING EQUATIONS FOR SURFACE EVOLVER

GENERAL WETTING EQUATION

The wetted area can be computed using the definition of surface area for a surface defined by $\vec{r}(z, \phi)$.

$$A_w = \int \left(\int_z^{z_0} \left| \vec{r}_z \times \vec{r}_\phi \right| dz \right) d\phi$$

Looking at the inner integral,

$$\int_z^{z_0} \left| \vec{r}_z \times \vec{r}_\phi \right| dz = \int_0^{z_0} \left| \vec{r}_z \times \vec{r}_\phi \right| dz - \int_0^{z} \left| \vec{r}_z \times \vec{r}_\phi \right| dz$$

Defining

$$I(z) = \int_0^{z} \left| \vec{r}_z \times \vec{r}_\phi \right| dz$$

$$C = \int_0^{z_0} \left| \vec{r}_z \times \vec{r}_\phi \right| dz = I(z_0)$$

and noting that

$$S_{end\ cap} = \int_0^{2\pi} \int_z^{z_0} \left| \vec{r}_z \times \vec{r}_\phi \right| dz\, d\phi = \int_0^{2\pi} C\, d\phi = 2\pi C$$

$$\Rightarrow C = \frac{S_{end\ cap}}{2\pi}$$

where $S_{end\ cap}$ us the end cap surface area. Substituting,

$$A_w = \int \left(\frac{S_{end\ cap}}{2\pi} - I(z) \right) d\phi$$

Since $\phi = \text{atan2}(y, x)$,

$$d\phi = \frac{-y}{x^2 + y^2} dx + \frac{x}{x^2 + y^2} dy$$

$$A_w = \int \frac{-y}{x^2 + y^2} \left(\frac{S_{end\ cap}}{2\pi} - I(z) \right) dx + \int \frac{x}{x^2 + y^2} \left(\frac{S_{end\ cap}}{2\pi} - I(z) \right) dy + 0\, dz$$

The wetted energy can then be found by multiplying by the negative cosine of the contact angle CA, splitting into x, y, and z components for input into Surface Evolver.

$$e_1 = \cos(CA)\frac{y}{x^2+y^2}\left(\frac{S_{end\ cap}}{2\pi} - I(z)\right)$$

$$e_2 = -\cos(CA)\frac{x}{x^2+y^2}\left(\frac{S_{end\ cap}}{2\pi} - I(z)\right)$$

$$e_3 = 0$$

Therefore, for each type of end cap geometry, it is only necessary to determine $I(z)$ and $\frac{S_{end\ cap}}{2\pi}$, which can be substituted into these formulas to find the necessary wetting equations.

CYLINDRICAL SECTION

For the cylindrical section where z < 0, a parameterization of a cylinder is

$$\vec{r}(z,\phi) = [R\cos\phi\,,\quad R\sin\phi\,,\quad z]$$

Taking partial derivatives,

$$\vec{r}_z = [0,\quad 0,\quad 1,]$$

$$\vec{r}_\phi = [-R\sin\phi\,,\quad R\cos\phi\,,\quad 0]$$

The cross product is

$$\vec{r}_z \times \vec{r}_\phi = [-R\cos\phi\,,\quad -R\sin\phi\,,\quad 0]$$

which has magnitude

$$\left|\vec{r}_z \times \vec{r}_\phi\right| = R$$

Substituting, $I(z)$ is

$$I(z) = \int_0^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz = \int_0^z R\,dz = Rz$$

This, combined with the value of $\frac{S_{end\ cap}}{2\pi}$ for the end cap used, can be used to define the wetting equation on the cylindrical section where z < 0.

HEMISPHERICAL END CAP

A parameterization of a sphere is

$$\vec{r}(z,\phi) = \left[\sqrt{R^2 - z^2}\cos\phi\,,\quad \sqrt{R^2 - z^2}\sin\phi\,,\quad z\right]$$

Taking partial derivatives,

$$\vec{r}_z = \left[\frac{z}{\sqrt{R^2 - z^2}}\cos\phi\,,\quad \frac{z}{\sqrt{R^2 - z^2}}\sin\phi\,,\quad 1\right]$$

$$\vec{r}_\phi = \left[-\sqrt{R^2 - z^2}\sin\phi\,,\quad \sqrt{R^2 - z^2}\cos\phi\,,\quad 0\right]$$

The cross product is

$$\vec{r}_z \times \vec{r}_\phi = \left[-\sqrt{R^2 - z^2}\cos\phi\,,\quad \sqrt{R^2 - z^2}\sin\phi\,,\quad z\right]$$

The magnitude is

$$\left|\vec{r}_z \times \vec{r}_\phi\right| = R$$

Substituting, $I(z)$ is

$$I(z) = \int_0^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz = \int_0^z R\,dz = Rz$$

The end cap surface area is, since $z_0 = R$,

$$\frac{S_{end\ cap}}{2\pi} = I(z_0) = I(R) = R^2$$

ELLIPSOIDAL END CAP

A parameterization of an ellipsoid is

$$\vec{r}(z,\phi) = \left[R\sqrt{1 - \left(\frac{z}{z_0}\right)^2}\cos\phi\,,\quad R\sqrt{1 - \left(\frac{z}{z_0}\right)^2}\sin\phi\,,\quad z\right]$$

Taking partial derivatives,

$$\vec{r}_z = \left[\frac{-Rz}{{z_0}^2\sqrt{1 - \left(\frac{z}{z_0}\right)^2}}\cos\phi\,,\quad \frac{-Rz}{{z_0}^2\sqrt{1 - \left(\frac{z}{z_0}\right)^2}}\sin\phi\,,\quad 1\right]$$

$$\vec{r}_\phi = \left[-R\sqrt{1 - \left(\frac{z}{z_0}\right)^2}\sin\phi\,,\quad R\sqrt{1 - \left(\frac{z}{z_0}\right)^2}\cos\phi\,,\quad 0\right]$$

The cross product is

$$\vec{r}_z \times \vec{r}_\phi = \left[R\sqrt{1-\left(\frac{z}{z_0}\right)^2}\cos\phi\,, \quad R\sqrt{1-\left(\frac{z}{z_0}\right)^2}\sin\phi\,, \quad \frac{R^2 z}{{z_0}^2} \right]$$

The magnitude is

$$\left|\vec{r}_\theta \times \vec{r}_\phi\right| = R\sqrt{1+\frac{R^2-{z_0}^2}{{z_0}^4}z^2}$$

Defining $b^2 = \frac{R^2-{z_0}^2}{{z_0}^4}$ and substituting into $I(z)$,

$$I(z) = \int_0^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz = \int_0^z R\sqrt{1+b^2z^2}\,dz = R\left(\frac{z}{2}\sqrt{1+b^2z^2} + \frac{\sinh^{-1}(bz)}{2b}\right)$$

The surface area is then

$$\frac{S_{end\ cap}}{2\pi} = I(z_0) = R\left(\frac{z_0}{2}\sqrt{1+b^2{z_0}^2} + \frac{\sinh^{-1}(bz_0)}{2b}\right) = \frac{R}{2}\left(R + \frac{\sinh^{-1}(bz_0)}{b}\right)$$

REDUCTION OF ELLIPSOID TO HEMISPHERE

In the limit as $z_0 \to R$, $b \to 0$, and $a = \sqrt{R^2-{z_0}^2} \to 0$,

$$\lim_{z_0\to R} I(z) = \lim_{z_0\to R} R\left(\frac{z}{2}\sqrt{1+b^2z^2} + \frac{\sinh^{-1}(bz)}{2b}\right) = \frac{R}{2}\lim_{b\to 0} z\sqrt{1+b^2z^2} + \frac{R}{2}\lim_{b\to 0}\frac{\sinh^{-1}(bz)}{b}$$

$$= \frac{Rz}{2} + \frac{R}{2}\lim_{b\to 0}\frac{z/\sqrt{b^2z^2+1}}{1} = \frac{Rz}{2} + \frac{Rz}{2} = Rz$$

$$\lim_{z_0\to R}\frac{S_{end\ cap}}{2\pi} = \lim_{z_0\to R}\frac{R}{2}\left(R + \frac{\sinh^{-1}(bz_0)}{b}\right) = \frac{R^2}{2} + \frac{R}{2}\lim_{z_0\to R}\left(\frac{{z_0}^2\sinh^{-1}\left(\frac{\sqrt{R^2-{z_0}^2}}{z_0}\right)}{\sqrt{R^2-{z_0}^2}}\right)$$

$$= \frac{R^2}{2} + \frac{R}{2}\lim_{a\to 0}\left(\frac{(R^2-a^2)\sinh^{-1}\left(\frac{a}{\sqrt{R^2-a^2}}\right)}{a}\right)$$

$$= \frac{R^2}{2} + \frac{R}{2}\lim_{a\to 0}\left(\frac{R - 2a\sinh^{-1}\left(\frac{a}{\sqrt{R^2-a^2}}\right)}{1}\right) = \frac{R^2}{2} + \frac{R^2}{2} = R^2$$

which is the same result as obtained in the hemispherical case.

SUPERELLIPSOIDAL END CAP

A parameterization of a superellipsoid is

$$\vec{r}(z,\phi) = \left[R\left(1-\left(\frac{z}{z_0}\right)^n\right)^{1/n}\cos\phi\,,\quad R\left(1-\left(\frac{z}{z_0}\right)^n\right)^{1/n}\sin\phi\,,\quad z\right]$$

Taking partial derivatives,

$$\vec{r}_z = \left[-\frac{R}{z_0}\left(\frac{z}{z_0}\right)^{n-1}\left(1-\left(\frac{z}{z_0}\right)^n\right)^{\frac{1}{n}-1}\cos\phi\,,\quad -\frac{R}{z_0}\left(\frac{z}{z_0}\right)^{n-1}\left(1-\left(\frac{z}{z_0}\right)^n\right)^{\frac{1}{n}-1}\sin\phi\,,\quad 1\right]$$

$$\vec{r}_\phi = \left[-R\left(1-\left(\frac{z}{z_0}\right)^n\right)^{\frac{1}{n}}\sin\phi\,,\quad R\left(1-\left(\frac{z}{z_0}\right)^n\right)^{1/n}\cos\phi\,,\quad 0\right]$$

The cross product is

$$\vec{r}_z \times \vec{r}_\phi = \left[R\left(1-\left(\frac{z}{z_0}\right)^n\right)^{1/n}\cos\phi\,,\quad R\left(1-\left(\frac{z}{z_0}\right)^n\right)^{1/n}\sin\phi\,,\quad \frac{R^2}{{z_0}^2}z^{n-1}({z_0}^n - z^n)^{\frac{2}{n}-1}\right]$$

The magnitude is

$$\left|\vec{r}_\theta \times \vec{r}_\phi\right| = R\left(1-\left(\frac{z}{z_0}\right)^n\right)^{1/n}\sqrt{1+\left(\frac{R}{z_0}\right)^2\left(\frac{z}{z_0}\right)^{2n-2}\left(1-\left(\frac{z}{z_0}\right)^n\right)^{\frac{2}{n}-2}}$$

Substituting into $I(z)$,

$$I(z) = \int_0^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz = \int_0^z R\left(1-\left(\frac{z}{z_0}\right)^n\right)^{1/n}\sqrt{1+\left(\frac{R}{z_0}\right)^2\left(\frac{z}{z_0}\right)^{2n-2}\left(1-\left(\frac{z}{z_0}\right)^n\right)^{\frac{2}{n}-2}}\,dz$$

Defining $\bar{z} = \frac{z}{z_0}$ and $AR = \left(\frac{R}{z_0}\right)^2$,

$$I(z) = Rz_0\int_0^{\bar{z}}(1-\bar{z}^n)^{1/n}\sqrt{1+AR\;\bar{z}^{2n-2}(1-\bar{z}^n)^{\frac{2}{n}-2}}\,d\bar{z}$$

The inner integral with respect to $\bar{z}$ cannot be evaluated analytically and so must be approximated. This integral is defined as $I'(\bar{z})$.

$$I'(\bar{z}) = \int_0^{\bar{z}} f(\bar{z})\,d\bar{z} = \int_0^{\bar{z}}(1-\bar{z}^n)^{1/n}\sqrt{1+AR\;\bar{z}^{2n-2}(1-\bar{z}^n)^{\frac{2}{n}-2}}\,d\bar{z}$$

$$I(z) = Rz_0\,I'(\bar{z})$$

The end cap surface area is

$$\frac{S_{end\ cap}}{2\pi} = I(z_0) = Rz_0\,I'(1)$$

APPROXIMATION OF SUPERELLIPSOIDAL WETTING EQUATION

When approximating the integral $I'(\bar{z})$, $I'(\bar{z})$ is first evaluated numerically for a range of values of $\bar{z}$. These values are then used to construct a cubic spline, which can be implemented as a piecewise function into Surface Evolver. By using a cubic spline to connect values of $I'(\bar{z})$, the second derivative of the approximated $I'(\bar{z})$ can remain continuous when moving from one interval to another. This helps prevent the contact line from sticking at the boundaries of these piecewise intervals. To further improve accuracy, a second approximation is used near the dome, where small changes in z result in large changes in location. By using a second expression for $I'(\bar{z})$ in terms of r near the dome, from here on referred as $I'(r)$, similar levels of accuracy can be maintained at each point on the dome. Expressing $I'(\bar{z})$ in terms of r gives $I'(r)$ as

$$I'(r) = \int_0^r r\sqrt{1 + AR(1 - r^n)^{\left(2-\frac{2}{n}\right)} r^{(2-2n)}}\, dr$$

The point on the dome to switch between these approximations is chosen to be $(r_{cr}, \bar{z}_{cr}) = \left(2^{-\frac{1}{n}}, 2^{-\frac{1}{n}}\right)$, which corresponds to the point at which the slope of $\bar{z}$ vs r is -1. In total, $N^{\bar{z}} = 1000$ points are used for the approximation of $I'(\bar{z})$ near the junction and $N^r = 1000$ points are used for the approximation of $I'(r)$ near the top of the dome. For $I'(\bar{z})$, the cubic spline approximation is

$$I'(\bar{z}) = a_i^{\bar{z}} t^3 + b_i^{\bar{z}} t^2 + c_i^{\bar{z}} t + d_i^{\bar{z}}$$

$$t = z \bmod \frac{z_0\, \bar{z}_{cr}}{N^{\bar{z}} - 1}$$

$$i = \left\lceil \frac{N^{\bar{z}} - 1}{z_0\, \bar{z}_{cr}}\, z \right\rceil$$

Similarly, for $I'(r)$,

$$I'(r) = a_i^r t^3 + b_i^r t^2 + c_i^r t + d_i^r$$

$$t = r \bmod \frac{r_{cr}}{N^r - 1}$$

$$i = \left\lceil \frac{N^r - 1}{r_{cr}}\, r \right\rceil$$

The coefficients a_i, b_i, c_i, and d_i can be computed by enforcing continuity of the 0th, 1st and 2nd derivatives at each point. For the end points at the top of the dome and at the junction between the dome and cylinder, these derivatives are computed analytically and enforced to create

a complete system of equations which can be solved with linear algebra. The values of these coefficients for the case of $n = 3$, $z_0 = 0.5$ and the MATLAB script used to generate them can be found in Appendix C.

REDUCTION OF SUPERELLIPSOID TO ELLIPSOID

The equations for the superellipsoidal case reduce to the equations for the ellipsoid when letting $n = 2$. The integral $I(z)$ becomes

$$I(z) = Rz_0\, I'^{(\bar{z})} = Rz_0 \int_0^{\bar{z}} \sqrt{1-\bar{z}^2}\sqrt{1+\left(\frac{R}{z_0}\right)^2 \frac{\bar{z}^2}{1-\bar{z}^2}}\, d\bar{z} = Rz_0 \int_0^{\bar{z}} \sqrt{1+\left(\left(\frac{R}{z_0}\right)^2 - 1\right)\bar{z}^2}\, d\bar{z}$$

$$= R\int_0^z \sqrt{1+\left(\frac{R^2 - {z_0}^2}{{z_0}^2}\right)\left(\frac{z}{z_0}\right)^2}\, dz = R\int_0^z \sqrt{1+b^2z^2}\, dz$$

$$= R\left(\frac{z}{2}\sqrt{1+b^2z^2} + \frac{\sinh^{-1}(bz)}{2b}\right)$$

where $b^2 = \frac{R^2 - {z_0}^2}{{z_0}^4}$, and is the same as for the ellipsoidal end cap wetting equation.

TORISPHERICAL END CAP

For the torispherical end caps, the knuckle and dome need to be considered separately. On the knuckle, where $z < z_{cr}$, the parameterization of a torus can be used.

$$\vec{r}(\theta, \phi) = [(R - r_s + r_s\cos\theta)\cos\phi\,, \quad (R - r_s + r_s\cos\theta)\sin\phi\,, \quad r_s\sin\theta]$$

Taking derivatives,

$$\vec{r}_\theta = [-r_s\sin\theta\cos\phi\,, \quad -r_s\sin\theta\sin\phi\,, \quad r_s\cos\theta]$$

$$\vec{r}_\phi = [-(R - r_s + r_s\cos\theta)\sin\phi\,, \quad (R - r_s + r_s\cos\theta)\cos\phi\,, \quad 0]$$

The cross product is

$\vec{r}_\theta \times \vec{r}_\phi$

$= [-r_s\cos\theta\,(R - r_s + r_s\cos\theta)\cos\phi\,, \quad -r_s\cos\theta\,(R - r_s + r_s\cos\theta)\sin\phi\,, \quad r_s\sin\theta\,(R - r_s + r_s\cos\theta)]$

The magnitude is

$$\left|\vec{r}_\theta \times \vec{r}_\phi\right| = r_s(R - r_s + r_s\cos\theta)$$

Substituting,

$$I(z) = \int_0^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz = \int_0^\theta \left|\vec{r}_\theta \times \vec{r}_\phi\right| d\theta = \int_0^\theta r_s(R - r_s + r_s\cos\theta)\, d\theta$$

$$= r_s[(R - r_s)\theta + r_s \sin\theta]$$

Since $z = r_s \sin\theta$, $\theta = \sin^{-1}\left(\frac{z}{r_s}\right)$,

$$I(z) = r_s\left[z + (R - r_s)\sin^{-1}\left(\frac{z}{r_s}\right)\right]$$

On the spherical dome, where $z \geq z_{cr}$, $I(z)$ is

$$I(z) = \int_0^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz = \int_0^{z_{cr}} \left|\vec{r}_z \times \vec{r}_\phi\right| dz + \int_{z_{cr}}^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz = I(z_{cr}) + \int_{z_{cr}}^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz$$

$$= r_s\left[z_{cr} + (R - r_s)\sin^{-1}\left(\frac{z_{cr}}{r_s}\right)\right] + \int_{z_{cr}}^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz$$

On the dome, the parameterization of a sphere can be used.

$$\vec{r}(z, \phi) = \left[\sqrt{{R_H}^2 - z^2}\cos\phi\,,\quad \sqrt{{R_H}^2 - z^2}\sin\phi\,,\quad z\right]$$

Taking partial derivatives,

$$\vec{r}_z = \left[\frac{z}{\sqrt{{R_H}^2 - z^2}}\cos\phi\,,\quad \frac{z}{\sqrt{{R_H}^2 - z^2}}\sin\phi\,,\quad 1\right]$$

$$\vec{r}_\phi = \left[-\sqrt{{R_H}^2 - z^2}\sin\phi\,,\quad \sqrt{{R_H}^2 - z^2}\cos\phi\,,\quad 0\right]$$

The cross product is

$$\vec{r}_z \times \vec{r}_\phi = \left[-\sqrt{{R_H}^2 - z^2}\cos\phi\,,\quad \sqrt{{R_H}^2 - z^2}\sin\phi\,,\quad z\right]$$

The magnitude is

$$\left|\vec{r}_z \times \vec{r}_\phi\right| = R_H$$

Integrating,

$$\int_{z_{cr}}^z \left|\vec{r}_z \times \vec{r}_\phi\right| dz = \int_{z_{cr}}^z R_H\, dz = R_H(z - z_{cr})$$

Substituting,

$$I(z) = r_s\left[z_{cr} + (R - r_s)\sin^{-1}\left(\frac{z_{cr}}{r_s}\right)\right] + R_H(z - z_{cr})$$

For the entire end cap,

$$I(z) = \begin{cases} r_s \left[z + (R - r_s) \sin^{-1}\left(\frac{z}{r_s}\right) \right] & z < z_{cr} \\ r_s \left[z_{cr} + (R - r_s) \sin^{-1}\left(\frac{z_{cr}}{r_s}\right) \right] + R_H (z - z_{cr}) & z \geq z_{cr} \end{cases}$$

The end cap surface area is

$$\frac{S_{end\ cap}}{2\pi} = I(z_0) = r_s \left[z_{cr} + (R - r_s) \sin^{-1}\left(\frac{z_{cr}}{r_s}\right) \right] + R_H (z_0 - z_{cr})$$

APPENDIX C. SUPERELLIPSOID WETTING APPROXIMATION CODE

SAMPLE OUTPUT

Table C.1. A small selection of the coefficients used to approximate the superellipsoid wetting equation.

i	$a_i^{\bar{z}}$	$b_i^{\bar{z}}$	$c_i^{\bar{z}}$	$d_i^{\bar{z}}$	j	a_i^r	b_i^r	c_i^r	d_i^r
1	-0.00013	5.23E-08	1	0	1	-4.88E-07	-1	0	1.549038
50	-0.00692	-0.00052	0.999985	0.038891	50	-5.02E-05	-1	-0.07778	1.547526
100	-0.00136	-0.00115	0.999915	0.078574	100	-0.00041	-1.00002	-0.15715	1.542864
150	0.016934	-0.00037	0.99984	0.118254	150	-0.0014	-1.00012	-0.23653	1.535053
200	0.048265	0.003344	0.999934	0.157934	200	-0.00336	-1.00039	-0.31592	1.524091
250	0.093165	0.011572	1.00049	0.197625	250	-0.00666	-1.00097	-0.39534	1.509978
300	0.152396	0.025972	1.001934	0.237355	300	-0.01176	-1.00204	-0.47482	1.492712
350	0.226908	0.048306	1.004823	0.277168	350	-0.01925	-1.00385	-0.55442	1.472289
400	0.317763	0.080453	1.009862	0.317136	400	-0.0299	-1.00673	-0.6342	1.448705
450	0.426054	0.124424	1.017907	0.35736	450	-0.04481	-1.01111	-0.71427	1.421949
500	0.552866	0.182356	1.029982	0.39798	500	-0.06556	-1.01759	-0.79476	1.392008
550	0.699356	0.256522	1.047284	0.439179	550	-0.09448	-1.02698	-0.87588	1.35886
600	0.867115	0.349349	1.071196	0.481191	600	-0.13515	-1.04046	-0.95789	1.322477
650	1.058988	0.463516	1.103304	0.524308	650	-0.19315	-1.05973	-1.04119	1.282815
700	1.280603	0.602186	1.145422	0.568892	700	-0.2775	-1.08732	-1.12633	1.239813
750	1.542994	0.769471	1.19965	0.61538	750	-0.40331	-1.12719	-1.21412	1.193383
800	1.867012	0.971314	1.268479	0.664301	800	-0.59699	-1.18564	-1.30575	1.143398
850	2.291185	1.217068	1.354993	0.716293	850	-0.90711	-1.2733	-1.40309	1.089671
900	2.887336	1.522397	1.463243	0.772134	900	-1.42872	-1.40891	-1.50913	1.03192
950	3.795614	1.914984	1.598948	0.832793	950	-2.363	-1.62783	-1.62892	0.969711
1000	5.313934	2.447014	1.770874	0.899519	1000	-4.18138	-2.00249	-1.77158	0.902334

MATLAB Source Code

```
z0 = 0.5;
n = 3;

R0 = 1;
AR = (R0/z0)^2;
zcyl = 1;


%% Compute Comparison Solution
N = 1001;

z_cr = 2^(-1/n);
r_cr = 2^(-1/n);
z = linspace(0, z_cr, N);
r = linspace(r_cr, 0, N);
I_z = z_computeIntegral(z, n, AR);
I_r = I_z(end) + r_computeIntegral(r, n, AR);
```

```
%% Compute z Spline
N_zSpline = 1001;
zSpline = linspace(0, z_cr, N_zSpline);
I_zSpline = zeros(size(zSpline));
for i = 1:length(zSpline)
    [~, ind] = min(abs(z - zSpline(i)));
    I_zSpline(i) = I_z(ind);
end
slope_zSpline_1 = 1;
slope_zSpline_end = (1-z_cr.^n).^(1/n) .* sqrt(1 + AR.*(1-z_cr.^n).^(2/n-
2).*z_cr.^(2*n-2));
p_zSpline = spline(zSpline, [slope_zSpline_1, I_zSpline, slope_zSpline_end]);

%% Compute r Spline
N_rSpline = 1001;
rSpline = linspace(r_cr, 0, N_rSpline);
I_rSpline = zeros(size(rSpline));
for i = 1:length(zSpline)
    [~, ind] = min(abs(r - rSpline(i)));
    I_rSpline(i) = I_r(ind);
end
slope_rSpline_1 = -r_cr.^n.*(1-r_cr.^n).^(1/n-1).*sqrt(1 + AR*r_cr.^(2-
2*n).*(1 - r_cr.^n).^(2-2/n));
slope_rSpline_end = 0;
p_rSpline = spline(rSpline, [slope_rSpline_end, I_rSpline, slope_rSpline_1]);

%% Compute Spline Error
I_zSpline_Comp = ppval(p_zSpline, z);
I_rSpline_Comp = ppval(p_rSpline, r);

zTotal = [z, (1-r.^n).^(1/n)];
ITotal = [I_z, I_r];
ITotalSpline = [I_zSpline_Comp, I_rSpline_Comp];

error_spline = sqrt(sum((ITotalSpline - ITotal).^2)/numel(ITotal));
emax_spline = max(abs(ITotalSpline-ITotal));

integrand_z = @(z1) (1-z1.^n).^(1/n) .* sqrt(1 + AR.*(1-z1.^n).^(2/n-
2).*z1.^(2*n-2));
error_Integral = (integral(@(z1) integrand_z(z1), 0, z_cr) - integral(@(r1)
integrand_r(r1, n, AR), r_cr, 0) - ITotal(end));

%% Plot Results
figure
hold on; grid on
plot(z, I_z)
plot(z, I_zSpline_Comp)

figure
hold on; grid on
plot(r, I_r)
plot(r, I_rSpline_Comp)

figure
hold on; grid on
plot(zTotal, ITotal)
plot(zTotal, ITotalSpline)
```

```
figure
hold on; grid on
plot(zTotal, ITotalSpline - ITotal)

%% Output for SE
% z wetting constraint
zVals = linspace(1, 1+z0*z_cr, 1000);
zCoeffs = p_zSpline.coefs';
z_ind = max(1, ceil((zVals-1)/(z0*z_cr)*(N_zSpline-1)));
zCalc = mod((zVals-1)/(z0), z_cr/(N_zSpline-1));
zCalc(zVals >= 1+z0*z_cr) = z_cr/(N_zSpline-1);
Icalc_zSpline = zCoeffs(1, z_ind).*zCalc.^3 + zCoeffs(2, z_ind).*zCalc.^2 +
zCoeffs(3, z_ind).*zCalc + zCoeffs(4, z_ind);

zCalcString = sprintf("((%.16g*(z-1)) %% (%.16g))", 1/z0, z_cr/(N_zSpline-1));
zIndString = sprintf("maximum(1, ceil(%.16g*(z-1)))", (N_zSpline-1)/(z0*z_cr));
zWettingString = sprintf("zCoeffs3[%s]*%s^3 + zCoeffs2[%s]*%s^2 +
zCoeffs1[%s]*%s + zCoeffs0[%s]", zIndString, zCalcString, zIndString,
zCalcString, zIndString, zCalcString, zIndString);

zCoeffs3 = sprintf("define zCoeffs3 real[%d] = {", length(zCoeffs));
zCoeffs2 = sprintf("define zCoeffs2 real[%d] = {", length(zCoeffs));
zCoeffs1 = sprintf("define zCoeffs1 real[%d] = {", length(zCoeffs));
zCoeffs0 = sprintf("define zCoeffs0 real[%d] = {", length(zCoeffs));

for i = 1:length(zCoeffs)
    if i == length(zCoeffs)
        additionalString = " }";
    else
        additionalString = ",";
    end
    zCoeffs3 = sprintf("%s %.16f%s", zCoeffs3, zCoeffs(1, i), additionalString);
    zCoeffs2 = sprintf("%s %.16f%s", zCoeffs2, zCoeffs(2, i), additionalString);
    zCoeffs1 = sprintf("%s %.16f%s", zCoeffs1, zCoeffs(3, i), additionalString);
    zCoeffs0 = sprintf("%s %.16f%s", zCoeffs0, zCoeffs(4, i), additionalString);
end

% r wetting constraint
rVals = linspace(0, r_cr, 1000);
rCoeffs = p_rSpline.coefs';
r_ind = max(1, ceil(rVals/r_cr*(N_rSpline-1)));
rCalc = mod(rVals, r_cr/(N_rSpline-1));
rCalc(rVals >= r_cr) = r_cr/(N_rSpline-1);
Icalc_rSpline = rCoeffs(1, r_ind).*rCalc.^3 + rCoeffs(2, r_ind).*rCalc.^2 +
rCoeffs(3, r_ind).*rCalc + rCoeffs(4, r_ind);

rCalcString = sprintf("((r_calc) %% (%.16g))", r_cr/(N_rSpline-1));
rIndString = sprintf("maximum(1, ceil(%.16g*(r_calc)))", (N_rSpline-1)/r_cr);
rWettingString = sprintf("rCoeffs3[%s]*%s^3 + rCoeffs2[%s]*%s^2 +
rCoeffs1[%s]*%s + rCoeffs0[%s]", rIndString, rCalcString, rIndString,
rCalcString, rIndString, rCalcString, rIndString);

rCoeffs3 = sprintf("define rCoeffs3 real[%d] = {", length(rCoeffs));
rCoeffs2 = sprintf("define rCoeffs2 real[%d] = {", length(rCoeffs));
rCoeffs1 = sprintf("define rCoeffs1 real[%d] = {", length(rCoeffs));
rCoeffs0 = sprintf("define rCoeffs0 real[%d] = {", length(rCoeffs));
```

```
for i = 1:length(rCoeffs)
    if i == length(rCoeffs)
        additionalString = " }";
    else
        additionalString = ",";
    end
    rCoeffs3 = sprintf("%s %.16f%s", rCoeffs3, rCoeffs(1, i), additionalString);
    rCoeffs2 = sprintf("%s %.16f%s", rCoeffs2, rCoeffs(2, i), additionalString);
    rCoeffs1 = sprintf("%s %.16f%s", rCoeffs1, rCoeffs(3, i), additionalString);
    rCoeffs0 = sprintf("%s %.16f%s", rCoeffs0, rCoeffs(4, i), additionalString);
end

% Plot SE Output
figure
hold on; grid on
plot(1+zTotal*z0, ITotal)
plot(zVals, Icalc_zSpline)
plot(1+z0*(1-rVals.^n).^(1/n), Icalc_rSpline)

% Combine for SE output
coeffString = sprintf("%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n", zCoeffs3, zCoeffs2,
zCoeffs1, zCoeffs0, rCoeffs3, rCoeffs2, rCoeffs1, rCoeffs0);
wettingString = sprintf("#define super_wetting ((super_SA) - RT*zMax_sup*((z
< %.16g) ? (%s) : (%s)))\n\n", 1+z_cr*z0, zWettingString, rWettingString);

syms nSym
betaFunc = double(subs(beta(1/nSym, 2/nSym), nSym, n));

defineString = sprintf("#define zMax_sup (%.3f)\n#define n (%d)\n", z0, n);
betaFuncString = sprintf("#define betaFunc (%.16f)\n", betaFunc);
SA_String = sprintf("#define super_SA (%.16f)\n", z0*ITotal(end));
outString = defineString + betaFuncString + SA_String + coeffString +
wettingString;
clipboard("copy", outString)

function res = integrand_r(r1, n, AR)
    res = r1.^n.*(1-r1.^n).^(1/n-1).*sqrt(1 + AR*r1.^(2-2*n).*(1 -
r1.^n).^(2-2/n));
    res(r1 == 0) = 0;
end

function I_z = z_computeIntegral(z, n, AR)
    integrand_z = @(z1) (1-z1.^n).^(1/n) .* sqrt(1 + AR.*(1-z1.^n).^(2/n-
2).*z1.^(2*n-2));

    for i = 1:length(z)
        I_z(i) = integral(@(z1) integrand_z(z1), 0, z(i));
    end
end

function I_r = r_computeIntegral(r, n, AR)

    for i = 1:length(r)
        I_r(i) = integral(@(r1) integrand_r(r1, n, AR), r(i), r(1));
    end
end
```

APPENDIX D. SURFACE EVOLVER SOURCE CODE – SUPERELLIPSOIDAL DOME, NON-RING SOLUTION

Note: {..} is used to represent the extremely large arrays output from the MATLAB code in Appendix C to save space.

```
#define RT 1
#define ZC RT
PARAMETER ang = 15
#define cosca cos(ang*pi/180)
#define sinca sin(ang*pi/180)
#define eps 1e-9
PARAMETER outputCLFile = ""
PARAMETER outputHeightFile = ""
PARAMETER outputEnergyFile = ""
PARAMETER r_std = 0
PARAMETER z_std = 0
PARAMETER maxInd = 6
define maxFrac_mat real[maxInd]
define minFrac_mat real[maxInd]
define theta_mat real[maxInd]
PARAMETER index = 1
PARAMETER asymmetric = 1;
PARAMETER toroidal = 0;
PARAMETER diff = 100;
PARAMETER theta1 = 0;
PARAMETER theta2 = 0;
PARAMETER cyl_vol = (pi*RT^2*ZC)
#define cyl_eqn (RT^2 - (x^2 + y^2) = 0)

// HEMISPHERE
#define hemi_eqn (z > ZC ? RT^2 - (x^2 + y^2 + (z-ZC)^2) = 0 :\
                 cyl_eqn)
#define hemi_vol (2/3*pi*RT^3 + cyl_vol)
#define hemi_H RT
#define hemi_z (sqrt(RT^2 - (x^2 + y^2)) + ZC)
```

```
#define hemi_SA (RT^2)
#define hemi_wetting (hemi_SA - RT*(z-ZC))
#define hemi_n 2
#define hemi_z0 1
#define hemi_diff (0)

// ELLIPSE
#define zMax_ell (0.5*RT)
PARAMETER CONST_z0_ell = zMax_ell
#define ellipse_eqn (z > ZC ? 1 - ((x/RT)^2 + (y/RT)^2 + ((z-ZC)/zMax_ell)^2)
= 0 :\
                    cyl_eqn)
#define ellipse_vol (2/3*pi*RT^2*zMax_ell + cyl_vol)
#define ellipse_H (zMax_ell + ZC - RT)
#define ellipse_z (zMax_ell*sqrt(1 - (x^2+y^2)/RT^2) + ZC)
#define bVal ((RT^2 - zMax_ell^2)/zMax_ell^4)
#define ellipse_SA (RT/2*(zMax_ell*sqrt(bVal*zMax_ell^2+1) +
asinh(sqrt(bVal)*zMax_ell)/sqrt(bVal)))
#define ellipse_wetting ((ellipse_SA) - 1/2*RT*((z_calc)*sqrt(bVal*(z_calc)^2
+ 1) + asinh(sqrt(bVal)*(z_calc))/sqrt(bVal)))
#define ellipse_n 2
#define ellipse_z0 zMax_ell

// SUPERELLIPSE
#define zMax_sup (0.750)
#define n (3)
#define betaFunc (3.6275987284684357)
#define super_SA (0.9563491000262874)
define zCoeffs3 real[1000] = {...}
define zCoeffs2 real[1000] = {...}
define zCoeffs1 real[1000] = {...}
define zCoeffs0 real[1000] = {...}
define rCoeffs3 real[1000] = {...}
define rCoeffs2 real[1000] = {...}
define rCoeffs1 real[1000] = {...}
define rCoeffs0 real[1000] = {...}
#define super_wetting ((super_SA) - RT*zMax_sup*((z < 1.595275394488075) ?
(zCoeffs3[maximum(1, ceil(1679.894733193164*(z-1)))]*((1.333333333333333*(z-
```

```
1)) % (0.0007937005259840997))^3 + zCoeffs2[maximum(1,
ceil(1679.894733193164*(z-1)))]*((1.333333333333333*(z-1)) %
(0.0007937005259840997))^2 + zCoeffs1[maximum(1, ceil(1679.894733193164*(z-
1)))]*((1.333333333333333*(z-1)) % (0.0007937005259840997)) +
zCoeffs0[maximum(1, ceil(1679.894733193164*(z-1)))]) : (rCoeffs3[maximum(1,
ceil(1259.921049894873*(r_calc)))]*((r_calc) % (0.0007937005259840997))^3 +
rCoeffs2[maximum(1, ceil(1259.921049894873*(r_calc)))]*((r_calc) %
(0.0007937005259840997))^2 + rCoeffs1[maximum(1,
ceil(1259.921049894873*(r_calc)))]*((r_calc) % (0.0007937005259840997)) +
rCoeffs0[maximum(1, ceil(1259.921049894873*(r_calc)))])))

#define super_n n
#define super_z0 zMax_sup
#define super_eqn (z > ZC ? 1 - (((x/RT)^2 + (y/RT)^2)^(n/2) + abs((z-
ZC)/zMax_sup)^n) = 0 :\
                  cyl_eqn)
#define super_vol (2*pi/(3*n)*RT^2*zMax_sup*betaFunc + cyl_vol)
#define super_H (zMax_sup + ZC - RT)
#define super_z (zMax_sup*(1 - ((x/RT)^2 + (y/RT)^2)^(n/2))^(1/n) + ZC)

// Poles
#define r_calc ((((z-1)/(sqrt(x^2+y^2)*CONST_z0))^CONST_n + 1)^(-1/CONST_n))
#define z_calc ((z-1)/sqrt(x^2+y^2)*r_calc)
#define x_calc (r_calc*x/sqrt(x^2+y^2))
#define y_calc (r_calc*y/sqrt(x^2+y^2))

// DEFINE MODEL
parameter frac := 0.45
parameter CONST_TANKVOL = ellipse_vol
#define           tankeqn ellipse_eqn
#define              delH ellipse_H
#define           capzEqn ellipse_z
#define    wettingEqnPart ellipse_wetting
#define          endCapSA ellipse_SA
PARAMETER       CONST_n = ellipse_n
PARAMETER      CONST_z0 = ellipse_z0
```

```
parameter maxFrac = maximum(0.1, ceil((1*(CONST_TANKVOL - cyl_vol - pi/6*(1-
sinca)/cosca*(3 + ((1-sinca)/cosca)^2))/CONST_TANKVOL)/0.01)*0.01 + 0.05)
parameter minFrac = 0.01

constraint 1 // draw the tank
formula: tankeqn

constraint 2 convex // on tank wall
formula: tankeqn

constraint 3 nonnegative // inside the tank
formula: tankeqn

constraint 4 // on center axis
formula: x = 0

constraint 5 // on center axis
formula: y = 0

constraint 9
formula: 0

constraint 10
formula: 0

quantity myvol fixed = (CONST_TANKVOL - cyl_vol) method facet_vector_integral
vector_integrand:
q1: 0
q2: 0
q3: (capzEqn - z)

quantity xmom info_only method facet_vector_integral
vector_integrand:
q1: 0
q2: 0
q3: x*(capzEqn - z)
```

```
quantity ymom info_only method facet_vector_integral
vector_integrand:
q1: 0
q2: 0
q3: y*(capzEqn - z)

quantity wettedEnergy energy method edge_vector_integral
vector_integrand
q1: (z > ZC ?  cosca*(y_calc)/(r_calc)^2 * (wettingEqnPart) :
RT*cosca*y/(x^2+y^2) * ((ZC-z) + (endCapSA)))
q2: (z > ZC ? -cosca*(x_calc)/(r_calc)^2 * (wettingEqnPart) : -
RT*cosca*x/(x^2+y^2) * ((ZC-z) + (endCapSA)))
q3: 0

quantity wettedArea info_only method edge_vector_integral
vector_integrand:
q1: (z > ZC ? -(y_calc)/(r_calc)^2 * (wettingEqnPart) :  RT*cosca*y/(x^2+y^2)
* ((ZC-z) + (endCapSA)))
q2: (z > ZC ?  (x_calc)/(r_calc)^2 * (wettingEqnPart) : -RT*cosca*x/(x^2+y^2)
* ((ZC-z) + (endCapSA)))
q3: 0

vertex
 1  RT   0       ZC fixed constraint 1  // outline only
 2   0  RT       ZC fixed constraint 1  // outline only
 3 -RT   0       ZC fixed constraint 1  // outline only
 4   0 -RT       ZC fixed constraint 1  // outline only
 5  RT   0        0 fixed constraint 1  // outline only
 6   0  RT        0 fixed constraint 1  // outline only
 7 -RT   0        0 fixed constraint 1  // outline only
 8   0 -RT        0 fixed constraint 1  // outline only
 9   0   0  (ZC+delH) fixed constraint 1   // outline only

11  RT   0  0 constraint 2  9 // bubble
12   0  RT  0 constraint 2    // bubble
13 -RT   0  0 constraint 2 10 // bubble
14   0 -RT  0 constraint 2    // bubble
```

```
edge
 1 1  2 constraint 1 fixed bare no_refine color blue // outline only
 2 2  3 constraint 1 fixed bare no_refine color blue // outline only
 3 3  4 constraint 1 fixed bare no_refine color blue // outline only
 4 4  1 constraint 1 fixed bare no_refine color blue // outline only
 5 5  6 constraint 1 fixed bare no_refine color blue // outline only
 6 6  7 constraint 1 fixed bare no_refine color blue // outline only
 7 7  8 constraint 1 fixed bare no_refine color blue // outline only
 8 8  5 constraint 1 fixed bare no_refine color blue // outline only
 9 1  9 constraint 1 fixed bare no_refine color blue // outline only
10 2  9 constraint 1 fixed bare no_refine color blue // outline only
11 3  9 constraint 1 fixed bare no_refine color blue // outline only
12 4  9 constraint 1 fixed bare no_refine color blue // outline only
13 1  5              fixed bare no_refine color blue // outline only
14 2  6              fixed bare no_refine color blue // outline only
15 3  7              fixed bare no_refine color blue // outline only
16 4  8              fixed bare no_refine color blue // outline only

21 11 12 constraint 2  9 wettedArea wettedEnergy // bubble
22 12 13 constraint 2 10 wettedArea wettedEnergy // bubble
23 13 14 constraint 2 10 wettedArea wettedEnergy // bubble
24 14 11 constraint 2  9 wettedArea wettedEnergy // bubble

face
1 21 22 23 24 constraint 3 myvol xmom ymom // bubble

body
1 1 density 0

read
lh := histogram(edge where not fixed, length)
ah := histogram(face, area)
vug := {{V 3; u 3; g3}3;}
r_outline := {refine edge where on_constraint 1}
changeVol := {myvol.target := (frac*CONST_TANKVOL)}
calcFrac := {print (myvol.value/CONST_TANKVOL)}
//drain := {frac := frac - 0.001; changeVol; vug; g5; {vug; g50}3}
//fill := {frac := frac + 0.001; changeVol; vug; g5; {vug; g50}3}
```

```
eraseHeightFile := {printf """>>>outputHeightFile}
eraseCLFile := {printf """>>>outputCLFile}
eraseEnergyFile := {printf """>>>outputEnergyFile}
outputHeight := {printf "%.16g, %.16g\n", frac,
vertex[14].z>>outputHeightFile}
outputCL := {foreach vertex vv where on_constraint 2 do {
              printf "%.2g %.16g %.16g %.16g\n ",frac,x,y,z>>outputCLFile
              };
            }
outputCLFile := "test.txt"
pertr := {set vertex where on_constraint 2 or on_constraint 3 z z-0.01*y};
pertl := {set vertex where on_constraint 2 or on_constraint 3 z z+0.01*y};
ar := {printf "%.4f\n", (wettedArea.value)}

drain := {frac := maximum(0.001, ceil((frac-0.001)*1000-0.5)/1000);
changeVol}
fill  := {frac := ceil((frac+0.01)*1000-0.5)/1000; changeVol}

averageVert := {fix vertex where on_constraint 2; {V;u}100; unfix vertex
where on_constraint 2}
averageEdge := {fix vertex where on_constraint 3; {V;u}10; unfix vertex where
on_constraint 3}
converge := {{averageVert; g20}10; averageVert}
slow_converge := {{{V;u}10; vug}20; {vug; g20}10};
//{vug 20; {averageVertSmall; g20}20; averageVert; {averageVertSmall;
g100}20; averageVert; g1000}
//converge := {quiet on; autodisplay off; vug 20; {vug; g20}50; {vug; g50}20;
{vug; g100}10; g1000; quiet off; autodisplay on}
//{quiet on; autodisplay off; vug 3; {{V;u}100; vug 10}20; {vug; g5}5;
{V;u}100; {vug; g20}20; {V;u}100; {vug; g50}20; g1000; quiet off; autodisplay
on};

checkAsymmetric := {r_1 := 0; r_2 := 0; z_1 := 0; z_2 := 0; numVert := 0;
                    foreach vertex vv where on_constraint 2 do {
                      r_1 := r_1 + sqrt(x^2 + y^2);
                      r_2 := r_2 + x^2 + y^2;
                      z_1 := z_1 + z;
                      z_2 := z_2 + z^2;
```

```
                    numVert := numVert + 1;
                  };
                  r_std_old := r_std; z_std_old := z_std;
                  r_std := sqrt((r_2/numVert) - (r_1/numVert)^2); z_std :=
sqrt((z_2/numVert) - (z_1/numVert)^2);
                  if (r_std + z_std) >= 0.05 then {asymmetric := 1} else
{asymmetric := 0};
                  diff := maximum(abs(r_std - r_std_old), abs(z_std -
z_std_old));
                 };
checkToroidal :=  {theta0 := sum(vertex where original == 12,  atan2(y,
x));//theta0 := atan2(ymom.value, xmom.value);
                  theta1_old := theta1; theta2_old := theta2;
                  theta1 :=  max(vertex where on_constraint 10 and
sqrt(x^2+y^2) > 0.5, theta0/abs(theta0)*(atan2(-x*sin(theta0) +
y*cos(theta0), x*cos(theta0) + y*sin(theta0)) + pi/4) % (2*pi) - pi/4);
                  theta2 :=  max(vertex where on_constraint 9 and
sqrt(x^2+y^2) > 0.5, -theta0/abs(theta0)*(atan2(-x*sin(theta0) +
y*cos(theta0), x*cos(theta0) + y*sin(theta0)) - pi/4) % (2*pi) - pi/4);
                  diff2 := (theta1+theta2) - (theta1_old+theta2_old);
                  if ( theta1 + theta2 - 2*pi) > 0 then {
                    toroidal := 1
                  } else {
                    toroidal := 0
                 }};

outputValues := { hit_vert := 0;
                  checkAsymmetric;
                  hit_vert := 0; toroidal := 0;
                  foreach vertex vv where hit_constraint 3 do {
                    hit_vert := hit_vert + 1;
                  };
                  checkToroidal;
                  printf "%.2f %.3f %d %d %d %.16f %.16f %.16f\n", ang, frac,
asymmetric, toroidal, hit_vert, r_std, z_std, total_energy>>outputEnergyFile
               }
```

```
tilt := {checkAsymmetric; if (asymmetric == 0) then {pertr 20; vug}5;
slow_converge; checkAsymmetric};
convergeLoopSlow := {conj_grad on; diff := 100; energyDiff := -100;
oldEnergy := total_energy; while ((diff > 10^-4) && (energyDiff < 0) &&
(toroidal == 0)) do { vug; checkAsymmetric; checkToroidal; energyDiff :=
total_energy - oldEnergy; oldEnergy := total_energy}; conj_grad off};

maxLength := 0.05;
r2 := {refine edge where length > maxLength and not fixed};

maxDiff := 10^-8;
//convergeLoopRefine := {quiet on; conj_grad on; diff := 100; while ((diff >
maxDiff) && (toroidal == 0) && (asymmetric == 1)) do { r2; averageVert; g20;
checkAsymmetric; checkToroidal; }; conj_grad off; quiet off};
convergeLoopRefine := {quiet on; conj_grad on; diff := 100; numNoChange := 0;
while ((numNoChange < 5) && (toroidal == 0) && (asymmetric == 1)) do { r2;
averageVert; g20; checkAsymmetric; checkToroidal; if (diff < maxDiff) then
{numNoChange := numNoChange + 1; print numNoChange} else {numNoChange := 0}};
conj_grad off; quiet off};
convergeLoopRefineNoGradLong := {quiet on; conj_grad off; diff := 100;
numNoChange := 0; while ((numNoChange < 5) && (toroidal == 0) && (asymmetric
== 1)) do { r2; averageVert; g20; checkAsymmetric; checkToroidal; if (diff <
maxDiff) then {numNoChange := numNoChange + 1; print numNoChange} else
{numNoChange := 0}}; conj_grad off; quiet off};
convergeLoopRefineNoGrad := {quiet on; conj_grad off; diff := 100; while
(diff > 10^-7) do { r2; averageVert; g20; checkAsymmetric; }; quiet off};
convergeLoop := {quiet on; conj_grad on; diff := 100; while ((diff > 10^-8)
&& (toroidal == 0) && (asymmetric == 1)) do { averageVert; g20;
checkAsymmetric; checkToroidal; }; conj_grad off; quiet off};
averageLoop := {diff := 100; while (diff > 0.0003) do { {{V;u}10; vug};
checkAsymmetric; checkToroidal; }};

endLength := 0.05;
convergeLoopRefineLoop := {maxLength := 0.08; while (maxLength > endLength -
0.005) do {convergeLoopRefine; maxLength := maxLength - 0.015};};

numFile := 4; maxAng := 90;
```

```
setNoDumpOn  := {ang.no_dump on;  frac.no_dump on};
setNoDumpOff := {ang.no_dump off; frac.no_dump off};
tempDumpFileName := sprintf "AsymmetricBoundary/tempDump/n%d_z0_%.3f_4.dmp",
CONST_n, CONST_z0
outputEnergyFile := sprintf "AsymmetricBoundary/n%d_z0_%.3f_%d.txt", CONST_n,
CONST_z0, numFile
dumpCurr := {setNoDumpOff; dump sprintf
"Asymmetric/Dump/n%d_z0_%.3f_theta%d_f%.1f_4.dmp", CONST_n, CONST_z0, ang,
frac*100; setNoDumpOn}
dumpTempDumpFile := {setNoDumpOn; asymmetric.no_dump on; toroidal.no_dump on;
dump tempDumpFileName; setNoDumpOff; asymmetric.no_dump off};

tilt2 := {fix vertex where on_constraint 2; {pertr 5; vug; averageVert}20;
unfix vertex where on_constraint 2; vug 50; {vug; g20}50};
//findBoundary := {while ((asymmetric == 1) && (toroidal == 0)) do { vug 20;
dumpTempDumpFile; convergeLoopRefineLoop; outputValues; dumpCurr;
replace_load tempDumpFileName; fill }};
prepSurface := {r; {r; vug}2; {vug; g20}5; {r; vug}; refine edge where
on_constraint 2; vug 20; tilt2; checkAsymmetric; if (asymmetric == 0) then
{ convergeLoopRefineNoGrad; tilt2; printf
"n%d_z0_%.3f_theta%d_f%g_4_prepSurface\n", CONST_n, CONST_z0, ang,
frac*100>>outputEnergyFile}}
//findBoundary := {while ((asymmetric == 1) && (toroidal == 0)) do
{ conj_grad off; prepSurface; {convergeLoopRefineNoGrad; vug}2;
convergeLoopRefine; outputValues; dumpCurr; replace_load tempDumpFileName;
fill }};
findBoundary := {while ((asymmetric == 1) && (toroidal == 0)) do { conj_grad
off; prepSurface; {r2; averageVert; g20}100; averageEdge 10; vug;
convergeLoopRefineNoGradLong; outputValues; dumpCurr; replace_load
tempDumpFileName; fill }};
runBoundary := {findBoundary; while ang < maxAng do {drain 3; ang := ang + 1;
asymmetric := 1; toroidal := 0; findBoundary}};

runIndividual := {convergeLoopRefineLoop; endLength := 0.015;
convergeLoopRefineLoop; maxLength := 0.015; convergeLoopRefine; maxLength :=
0.01; convergeLoopRefine; dumpCurr}
convergeToroidal := {quiet on; conj_grad on; diff2 := 100; diff := 100; while
(((diff2 > 10^-6) && (diff > 10^-6)) && (toroidal == 0) && (asymmetric == 1))
```

```
do { r2; averageVert; g20; checkAsymmetric; checkToroidal; }; conj_grad off;
quiet off};

blankDumpFile := sprintf "AsymmetricBoundary/tempDump/n%d_z0_%.3f_blank.dmp",
CONST_n, CONST_z0;
runUp   := {runIndividual; outputValues; while ((asymmetric == 1) &&
(toroidal == 0)) do {replace_load blankDumpFile; r; {r; vug}2; {vug;g 20}20;
{r; vug}; refine edge where on_constraint 2; vug 20; tilt; runIndividual;
outputValues; fill}};
runDown := {runIndividual; outputValues; while ((asymmetric == 0) ||
(toroidal == 1)) do {replace_load blankDumpFile; r; {r; vug}2; {vug;g 20}20;
{r; vug}; refine edge where on_constraint 2; vug 20; tilt; runIndividual;
outputValues; fill}};

test := {foreach edge ee where on_constraint 2 do {
          foreach ee.vertex do {
            printf "%.2g %.16g %.16g %.16g\n ",frac,x,y,z>>outputCLFile
          }
        };
        };

diff := 100; diff2 := 100; diff3 := 100; currMaxOld := 0;
checkMaxHeight := {currMax := max(vertex where on_constraint 2, z);
                   diff3 := currMax - currMaxOld;
                   currMaxOld := currMax;
                  }
outputDiffTestFile := "";
updateDiffTestFile := {outputDiffTestFile := sprintf
"AsymmetricBoundary/Test/n%d_z0_%.3f_theta%d_f%.1f.txt", CONST_n, CONST_z0,
ang, frac*100};
outputDiffTest := {checkAsymmetric; checkToroidal; checkMaxHeight; printf
"%.16f %.16f %.16f\n", diff, diff2, diff3>>outputDiffTestFile};
diffTest :=  {quiet on; conj_grad on; while (1) do { r2; averageVert; g20;
outputDiffTest }; conj_grad off; quiet off};
runDiffTest := {prepSurface; updateDiffTestFile; diffTest};

fixVert := {fix vertex where on_constraint 2};
unfixVert := {unfix vertex where on_constraint 2};
```

```
increment := {outputValues; dumpCurr; fill}
avgv := {{V;u}100; vug};
avgvFix := {fixVert; {V;u}100; unfixVert};
converge := {vug 20; {vug; g20}5; conj_grad on; avgv 2; {vug; g20}20; U; U;
{vug; g100}; U; fixVert; {V;u}1000; unfixVert; U; {vug; g100}; U};

runAsymmetric := {};
runAsymmetric := {converge; increment; runAsymmetric};

runAsymmetricUntil := {};
runAsymmetricUntil := {avgv 10; converge; increment; if (((asymmetric == 1)
&& (toroidal == 0))) then {runAsymmetricUntil}};

shrink := {avgv 10; vug 5; frac := 0.05; changeVol; vug 5; avgv 10; vug 5;
frac := 0.03; changeVol; vug 5; avgv 10; vug 5; frac := 0.01; changeVol; vug
5; avgv 10; vug 5}

runCase := {};
runCase := {frac := 0.05; changeVol; avgv 3; tilt; avgv 5; shrink; avgv 100;
converge; avgv 10; converge; runAsymmetricUntil; ang := ang + 5; runCase};
```

APPENDIX E. SURFACE EVOLVER SOURCE CODE – SUPERELLIPSOIDAL DOME, RING SOLUTION

Note: {..} is used to represent the extremely large arrays output from the MATLAB code in Appendix C to save space.

```
#define RT 1
#define ZC RT
PARAMETER ang = 90
#define cosca cos(ang*pi/180)
#define sinca sin(ang*pi/180)
#define eps 1e-9
PARAMETER outputCLFile = ""
PARAMETER outputHeightFile = ""
PARAMETER outputEnergyFile = ""
PARAMETER r_std = 0
PARAMETER z_std = 0
PARAMETER maxInd = 2
define maxFrac_mat real[maxInd]
define minFrac_mat real[maxInd]
define theta_mat real[maxInd]
PARAMETER index = 0
PARAMETER cyl_vol (pi*RT^2*ZC)
#define cyl_eqn (RT^2 - (x^2 + y^2) = 0)

// HEMISPHERE
#define hemi_eqn (z > ZC ? RT^2 - (x^2 + y^2 + (z-ZC)^2) = 0 :\
                  cyl_eqn)
#define hemi_vol (2/3*pi*RT^3 + cyl_vol)
#define hemi_H RT
#define hemi_z (sqrt(RT^2 - (x^2 + y^2)) + ZC)
#define hemi_SA (RT^2)
#define hemi_wetting (hemi_SA - RT*(z-ZC))
#define hemi_n 2
#define hemi_z0 1
#define hemi_diff (0)
```

```
// ELLIPSE
#define zMax_ell (0.5*RT)
PARAMETER CONST_z0_ell = zMax_ell
#define ellipse_eqn (z > ZC ? 1 - ((x/RT)^2 + (y/RT)^2 + ((z-ZC)/zMax_ell)^2)
= 0 : cyl_eqn)
#define ellipse_vol (2/3*pi*RT^2*zMax_ell + cyl_vol)
#define ellipse_H (zMax_ell + ZC - RT)
#define ellipse_z (zMax_ell*sqrt(1 - (x^2+y^2)/RT^2) + ZC)
#define bVal ((RT^2 - zMax_ell^2)/zMax_ell^4)
#define ellipse_SA (RT/2*(zMax_ell*sqrt(bVal*zMax_ell^2+1) +
asinh(sqrt(bVal)*zMax_ell)/sqrt(bVal)))
#define ellipse_wetting ((ellipse_SA) - 1/2*RT*((z_calc)*sqrt(bVal*(z_calc)^2
+ 1) + asinh(sqrt(bVal)*(z_calc))/sqrt(bVal)))
#define ellipse_n 2
#define ellipse_z0 zMax_ell

// SUPERELLIPSE
#define zMax_sup (0.750)
#define n (3)
#define betaFunc (3.6275987284684357)
#define super_SA (0.9563491000262874)
define zCoeffs3 real[1000] = {...}
define zCoeffs2 real[1000] = {...}
define zCoeffs1 real[1000] = {...}
define zCoeffs0 real[1000] = {...}
define rCoeffs3 real[1000] = {...}
define rCoeffs2 real[1000] = {...}
define rCoeffs1 real[1000] = {...}
define rCoeffs0 real[1000] = {...}
#define super_wetting ((super_SA) - RT*zMax_sup*((z < 1.595275394488075) ?
(zCoeffs3[maximum(1, ceil(1679.894733193164*(z-1)))]*((1.333333333333333*(z-
1)) % (0.0007937005259840997))^3 + zCoeffs2[maximum(1,
ceil(1679.894733193164*(z-1)))]*((1.333333333333333*(z-1)) %
(0.0007937005259840997))^2 + zCoeffs1[maximum(1, ceil(1679.894733193164*(z-
1)))]*((1.333333333333333*(z-1)) % (0.0007937005259840997)) +
zCoeffs0[maximum(1, ceil(1679.894733193164*(z-1)))]) : (rCoeffs3[maximum(1,
ceil(1259.921049894873*(r_calc)))]*((r_calc) % (0.0007937005259840997))^3 +
rCoeffs2[maximum(1, ceil(1259.921049894873*(r_calc)))]*((r_calc) %
```

```
(0.0007937005259840997))^2 + rCoeffs1[maximum(1,
ceil(1259.921049894873*(r_calc)))]*((r_calc) % (0.0007937005259840997)) +
rCoeffs0[maximum(1, ceil(1259.921049894873*(r_calc)))]))))

#define super_n n
#define super_z0 zMax_sup
#define super_eqn (z > ZC ? 1 - (((x/RT)^2 + (y/RT)^2)^(n/2) + abs((z-
ZC)/zMax_sup)^n) = 0 :\
                   cyl_eqn)
#define super_vol (2*pi/(3*n)*RT^2*zMax_sup*betaFunc + cyl_vol)
#define super_H (zMax_sup + ZC - RT)
#define super_z (zMax_sup*(1 - ((x/RT)^2 + (y/RT)^2)^(n/2))^(1/n) + ZC)

// Poles
#define r_calc ((((z-1)/(sqrt(x^2+y^2)*CONST_z0))^CONST_n + 1)^(-1/CONST_n))
#define z_calc ((z-1)/sqrt(x^2+y^2)*r_calc)
#define x_calc (r_calc*x/sqrt(x^2+y^2))
#define y_calc (r_calc*y/sqrt(x^2+y^2))

// DEFINE MODEL
parameter frac := 0.45
parameter CONST_TANKVOL = ellipse_vol
#define           tankeqn ellipse_eqn
#define              delH ellipse_H
#define           capzEqn ellipse_z
#define    wettingEqnPart ellipse_wetting
#define          endCapSA ellipse_SA
PARAMETER       CONST_n = ellipse_n
PARAMETER      CONST_z0 = ellipse_z0

parameter maxFrac = maximum(0.1, ceil((1*(CONST_TANKVOL - cyl_vol - pi/6*(1-
sinca)/cosca*(3 + ((1-sinca)/cosca)^2))/CONST_TANKVOL)/0.01)*0.01 + 0.05)
parameter minFrac = 0.01

constraint 1 // draw the tank
formula: tankeqn
```

```
constraint 2 convex // on tank wall
formula: tankeqn

constraint 3 nonnegative // inside the tank
formula: tankeqn

constraint 4 // on center axis
formula: x = 0

constraint 5 // on center axis
formula: y = 0

quantity myvol fixed = (CONST_TANKVOL - cyl_vol) method facet_vector_integral
vector_integrand:
q1: 0
q2: 0
q3: (capzEqn - z)

quantity wettedEnergy energy method edge_vector_integral
vector_integrand
q1: (z > ZC ?  cosca*(y_calc)/(r_calc)^2 * (wettingEqnPart) :
RT*cosca*y/(x^2+y^2) * ((ZC-z) + (endCapSA)))
q2: (z > ZC ? -cosca*(x_calc)/(r_calc)^2 * (wettingEqnPart) : -
RT*cosca*x/(x^2+y^2) * ((ZC-z) + (endCapSA)))
q3: 0

quantity wettedArea info_only method edge_vector_integral
vector_integrand:
q1: (z > ZC ? -(y_calc)/(r_calc)^2 * (wettingEqnPart) :  RT*cosca*y/(x^2+y^2)
* ((ZC-z) + (endCapSA)))
q2: (z > ZC ?  (x_calc)/(r_calc)^2 * (wettingEqnPart) : -RT*cosca*x/(x^2+y^2)
* ((ZC-z) + (endCapSA)))
q3: 0

vertex
 1  RT   0       ZC fixed constraint 1   // outline only
 2   0  RT       ZC fixed constraint 1   // outline only
 3 -RT   0       ZC fixed constraint 1   // outline only
```

```
 4   0 -RT       ZC fixed constraint 1   // outline only
 5  RT   0        0 fixed constraint 1   // outline only
 6   0  RT        0 fixed constraint 1   // outline only
 7 -RT   0        0 fixed constraint 1   // outline only
 8   0 -RT        0 fixed constraint 1   // outline only
 9   0   0  (ZC+delH) fixed constraint 1   // outline only

11  RT   0  ZC constraint 2 // bubble
12   0  RT  ZC constraint 2 // bubble
13 -RT   0  ZC constraint 2 // bubble
14   0 -RT  ZC constraint 2 // bubble

15  0.5    0 (ZC+delH) constraint 2 // bubble
16    0  0.5 (ZC+delH) constraint 2 // bubble
17 -0.5    0 (ZC+delH) constraint 2 // bubble
18    0 -0.5 (ZC+delH)constraint 2 // bubble

edge
 1 1  2 constraint 1 fixed bare no_refine color blue // outline only
 2 2  3 constraint 1 fixed bare no_refine color blue // outline only
 3 3  4 constraint 1 fixed bare no_refine color blue // outline only
 4 4  1 constraint 1 fixed bare no_refine color blue // outline only
 5 5  6 constraint 1 fixed bare no_refine color blue // outline only
 6 6  7 constraint 1 fixed bare no_refine color blue // outline only
 7 7  8 constraint 1 fixed bare no_refine color blue // outline only
 8 8  5 constraint 1 fixed bare no_refine color blue // outline only
 9 1  9 constraint 1 fixed bare no_refine color blue // outline only
10 2  9 constraint 1 fixed bare no_refine color blue // outline only
11 3  9 constraint 1 fixed bare no_refine color blue // outline only
12 4  9 constraint 1 fixed bare no_refine color blue // outline only
13 1  5              fixed bare no_refine color blue // outline only
14 2  6              fixed bare no_refine color blue // outline only
15 3  7              fixed bare no_refine color blue // outline only
16 4  8              fixed bare no_refine color blue // outline only

21 11 12 constraint 2 wettedArea wettedEnergy // bubble
22 12 13 constraint 2 wettedArea wettedEnergy // bubble
23 13 14 constraint 2 wettedArea wettedEnergy // bubble
```

```
24 14 11 constraint 2 wettedArea wettedEnergy // bubble

25 16 15 constraint 2 wettedArea wettedEnergy // bubble
26 17 16 constraint 2 wettedArea wettedEnergy // bubble
27 18 17 constraint 2 wettedArea wettedEnergy // bubble
28 15 18 constraint 2 wettedArea wettedEnergy // bubble

29 11 15 constraint 3 // bubble
30 12 16 constraint 3 // bubble
31 13 17 constraint 3 // bubble
32 14 18 constraint 3 // bubble

face
1 21 30 25 -29 constraint 3 myvol // bubble
2 22 31 26 -30 constraint 3 myvol // bubble
3 23 32 27 -31 constraint 3 myvol // bubble
4 24 29 28 -32 constraint 3 myvol // bubble

body
1 1 2 3 4 density 0

read
theta_mat := {30, 15};
minFrac_mat := {0.11, 0.09};
maxFrac_mat := {0.21, 0.22};

ang := theta_mat[1];
maxFrac := maxFrac_mat[1]; minFrac := minFrac_mat[1];

lh := histogram(edge where not fixed, length)
ah := histogram(face, area)
vug := {{V 3; u 3; g3}3;}
r_outline := {refine edge where on_constraint 1}
changeVol := {myvol.target := (frac*CONST_TANKVOL)}
calcFrac := {print (myvol.value/CONST_TANKVOL)}
eraseHeightFile := {printf """>>>outputHeightFile}
eraseCLFile := {printf """>>>outputCLFile}
eraseEnergyFile := {printf """>>>outputEnergyFile}
```

```
outputHeight := {printf "%.16g, %.16g\n", frac,
vertex[14].z>>outputHeightFile}
outputCL := {foreach vertex vv where on_constraint 2 do {
              printf "%.2g %.16g %.16g %.16g\n ",frac,x,y,z>>outputCLFile
              };
            }
outputCLFile := "test.txt"
pertr := {set vertex where on_constraint 2 or on_constraint 3 z z-0.01*y};
pertl := {set vertex where on_constraint 2 or on_constraint 3 z z+0.01*y};
ar := {printf "%.4f\n", (wettedArea.value)}

drain := {frac := ceil((frac-0.01)*100-0.5)/100; changeVol}
fill  := {frac := ceil((frac+0.01)*100-0.5)/100; changeVol}

averageVert := {fix vertex where on_constraint 2; {V;u}100; unfix vertex
where on_constraint 2}
converge := {vug 20; {averageVert; g20}20; {averageVert; g100}20;
averageVert; g1000; averageVert}
slow_converge := {{{V;u}10; vug}20; {vug; g20}10};
//{quiet on; autodisplay off; vug 3; {{V;u}100; vug 10}20; {vug; g5}5;
{V;u}100; {vug; g20}20; {V;u}100; {vug; g50}20; g1000; quiet off; autodisplay
on};
tilt := {if ((r_std + z_std) < 0.05) then {pertr 20; vug}3 }

checkAsymmetric := {r_1 := 0; r_2 := 0; z_1 := 0; z_2 := 0; numVert := 0;
                    foreach vertex vv where on_constraint 2 do {
                      r_1 := r_1 + sqrt(x^2 + y^2);
                      r_2 := r_2 + x^2 + y^2;
                      z_1 := z_1 + z;
                      z_2 := z_2 + z^2;
                      numVert := numVert + 1;
                    };
                    r_std := sqrt((r_2/numVert) - (r_1/numVert)^2); z_std :=
sqrt((z_2/numVert) - (z_1/numVert)^2);
                   };
outputValues := { hit_vert := 0;
                  checkAsymmetric;
                  hit_vert := 0; toroidal := 0;
```

```
                  foreach vertex vv where hit_constraint 3 do {
                    hit_vert := hit_vert + 1;
                  };
                  printf "%.2f %.3f %d %.16f %.16f %.16f\n", ang, frac,
hit_vert, r_std, z_std, total_energy>>outputEnergyFile
                }

setNoDumpOn  := {ang.no_dump on;  frac.no_dump on;  index.no_dump on;
maxFrac.no_dump on;  minFrac.no_dump on};
setNoDumpOff := {ang.no_dump off; frac.no_dump off; index.no_dump off;
maxFrac.no_dump off; minFrac.no_dump off};
outputEnergyFile := sprintf
"Energy/Superellipsoid/ToroidalFill/n%d_z0_%.3f_theta%d.txt", CONST_n,
CONST_z0, ang
dumpCurr := {dump sprintf "Ring/Dump/n%d_z0_%.3f_theta%d_f%.2g.dmp", CONST_n,
CONST_z0, ang, frac*100}
takeStep := {converge; outputValues; setNoDumpOff; dumpCurr; setNoDumpOn;
fill}
run := {while (frac < (maxFrac+0.005)) do {takeStep} };

outputVertices := {printf """>>>"test.txt"; foreach vertex vv where
on_constraint 3 or on_constraint 2 do { printf "%.16g %.16g %.16g\n", x, y,
z>>"test.txt" }};

tempDumpFile := sprintf "TempDumpFiles/ToroidalFill_n%d_z0_%.3f.dmp",
CONST_n, CONST_z0
incrementAng := { index := index + 1;
                  ang := theta_mat[index];
                  maxFrac := maxFrac_mat[index]; minFrac :=
minFrac_mat[index];
                  outputEnergyFile := sprintf
"Energy/Superellipsoid/ToroidalFill/n%d_z0_%.3f_theta%d.txt", CONST_n,
CONST_z0, ang;
                  dumpCurr := {dump sprintf
"Energy/Superellipsoid/ToroidalFill/Dump/n%d_z0_%.3f_theta%d_f%d.dmp",
CONST_n, CONST_z0, ang, frac*100};
                  frac := minFrac;
                };
```

```
runCA := {while index < maxInd do {incrementAng; eraseEnergyFile;
replace_load tempDumpFile; changeVol; slow_converge; run}};

fixVert := {fix vertex where on_constraint 2};
unfixVert := {unfix vertex where on_constraint 2};
increment := {outputValues; dumpCurr; fill}
decrement := {outputValues; dumpCurr; drain}
avgv := {{V;u}100; vug};
converge := {avgv 3; vug 20; {vug; g20}5; conj_grad on; avgv 2; {vug; g20}20;
U; U; {vug; g100}; U; fixVert; {V;u}1000; unfixVert; U; {vug; g100}; U};

runRingUp := {};
runRingUp := {fill; converge; increment; runRingUp};
runRingDown := {};
runRingDown := {converge; decrement; runRingDown};

incrementAng := {outputValues; dumpCurr; ang := ang + 1; outputEnergyFile :=
sprintf "Ring/n%d_z0_%.3f/n%d_z0_%.3f_t%d.txt", CONST_n, CONST_z0, CONST_n,
CONST_z0, ang}
runRingAng := {};
runRingAng := {converge; incrementAng; runRingAng};

shrink := {avgv 10; vug 5; frac := 0.05; changeVol; vug 5; avgv 10; vug 5;
frac := 0.03; changeVol; vug 5; avgv 10; vug 5; frac := 0.01; changeVol; vug
5; avgv 10; vug 5}
```

APPENDIX F. SURFACE EVOLVER SOURCE CODE – TORISPHERICAL DOME, NON-RING SOLUTION

```
#define RT 1
#define ZC RT
PARAMETER ang = 90
#define cosca cos(ang*pi/180)
#define sinca sin(ang*pi/180)
#define eps 1e-9
PARAMETER outputCLFile = ""
PARAMETER outputHeightFile = ""
PARAMETER outputEnergyFile = ""
PARAMETER r_std = 0
PARAMETER z_std = 0
define maxFrac_mat real[6]
define minFrac_mat real[6]
PARAMETER index = 1
PARAMETER asymmetric = 1;
PARAMETER toroidal = 0;
PARAMETER diff = 100;
PARAMETER theta1 = 0;
PARAMETER theta2 = 0;
PARAMETER cyl_vol = (pi*RT^2*ZC)
#define cyl_eqn (RT^2 - (x^2 + y^2) = 0)

// TORISPHERICAL
#define z_max  (0.75)
#define r_norm (0.6)

#define rS (r_norm*z_max)
#define RH ((2*rS*RT - RT^2 - z_max^2)/(2*(rS-z_max)))

#define zCenter (-sqrt(-RT^2 + 2*RT*rS - 2*RH*rS + RH^2)) //z of sphere
center
#define zcr (rS*sqrt((RH-RT)*(RH - 2*rS + RT))/(RH-rS)) // r of start of
sphere
#define rcr (sqrt(rS^2 - zcr^2) + RT - rS)
```

```
#define toris_eqn (z > ZC ? (z > (zcr+ZC) ? RH^2 - (x^2 + y^2 + (z-zCenter-
ZC)^2 = 0) :\
                          ((sqrt(rS^2 - (z-ZC)^2) + RT - rS -
sqrt(x^2+y^2)) = 0)) :\
                  cyl_eqn)
#define toris_vol (pi/3*(2*(RH+zCenter)*RH^2 - (2*rS^2+(RT-
rS)^2+2*rS*RH)*(RH-(RH+zCenter)) \
                  + 3*rS^2*(RT-rS)*asin((RH-(RH+zCenter))/(RH-rS))) +
cyl_vol)
#define toris_H (RH + zCenter + ZC - RT)
#define toris_z (sqrt(x^2+y^2) < rcr ? (sqrt(RH^2 - (x^2+y^2)) + ZC
+zCenter) : (sqrt(rS^2 - (sqrt(x^2+y^2)-RT+rS)^2) + ZC))
#define toris_SA ((rcr^2 + (RH-zcr+zCenter)^2)/2 + rS*(zcr + (RT-
rS)*asin(zcr/rS)))
#define toris_wetting ((toris_SA) + (z_calc < (zcr) ? -(rS*((z_calc) + (RT-
rS)*asin((z_calc)/rS))) : RH/RT*(zcr-z_calc) -(rS*((zcr) + (RT-
rS)*asin((zcr)/rS)))))
parameter CONST_zMax = z_max
parameter CONST_rNorm = r_norm

// Poles
#define t_calc ((z-ZC)/sqrt(x^2+y^2))
#define r_calc (t_calc >= zcr/rcr ? (t_calc*zCenter+sqrt(RH^2*(t_calc^2+1)-
zCenter^2))/(t_calc^2+1) : (sqrt(-RT^2*t_calc^2+2*RT*rS*t_calc^2+rS^2) + RT -
rS)/(t_calc^2+1) )
#define z_calc (t_calc*r_calc)
#define x_calc (r_calc*x/sqrt(x^2+y^2))
#define y_calc (r_calc*y/sqrt(x^2+y^2))

// DEFINE MODEL
parameter frac := 0.45
parameter CONST_TANKVOL = toris_vol
#define           tankeqn toris_eqn
#define              delH toris_H
#define           capzEqn toris_z
#define    wettingEqnPart toris_wetting
#define          endCapSA toris_SA
```

```
parameter maxFrac = maximum(0.1, ceil((1*(CONST_TANKVOL - cyl_vol - pi/6*(1-
sinca)/cosca*(3 + ((1-sinca)/cosca)^2))/CONST_TANKVOL)/0.01)*0.01 + 0.05)
parameter minFrac = 0.01

constraint 1 // draw the tank
formula: tankeqn

constraint 2 convex // on tank wall
formula: tankeqn

constraint 3 nonnegative // inside the tank
formula: tankeqn

constraint 4 // on center axis
formula: x = 0

constraint 5 // on center axis
formula: y = 0

constraint 9
formula: 0

constraint 10
formula: 0

quantity myvol fixed = (CONST_TANKVOL - cyl_vol) method facet_vector_integral
vector_integrand:
q1: 0
q2: 0
q3: (capzEqn - z)

quantity wettedEnergy energy method edge_vector_integral
vector_integrand
q1: (z > ZC ?  cosca*(y_calc)/(r_calc)^2 * (wettingEqnPart) :
RT*cosca*y/(x^2+y^2) * ((ZC-z) + (endCapSA)))
q2: (z > ZC ? -cosca*(x_calc)/(r_calc)^2 * (wettingEqnPart) : -
RT*cosca*x/(x^2+y^2) * ((ZC-z) + (endCapSA)))
q3: 0
```

```
quantity wettedArea info_only method edge_vector_integral
vector_integrand:
q1: (z > ZC ? -(y_calc)/(r_calc)^2 * (wettingEqnPart) :  RT*cosca*y/(x^2+y^2)
* ((ZC-z) + (endCapSA)))
q2: (z > ZC ?  (x_calc)/(r_calc)^2 * (wettingEqnPart) : -RT*cosca*x/(x^2+y^2)
* ((ZC-z) + (endCapSA)))
q3: 0

vertex
 1  RT   0       ZC fixed constraint 1   // outline only
 2   0  RT       ZC fixed constraint 1   // outline only
 3 -RT   0       ZC fixed constraint 1   // outline only
 4   0 -RT       ZC fixed constraint 1   // outline only
 5  RT   0        0 fixed constraint 1   // outline only
 6   0  RT        0 fixed constraint 1   // outline only
 7 -RT   0        0 fixed constraint 1   // outline only
 8   0 -RT        0 fixed constraint 1   // outline only
 9   0   0  (ZC+delH) fixed constraint 1   // outline only

11  RT   0  0 constraint 2  9 // bubble
12   0  RT  0 constraint 2    // bubble
13 -RT   0  0 constraint 2 10 // bubble
14   0 -RT  0 constraint 2    // bubble

edge
 1 1  2 constraint 1 fixed bare no_refine color blue // outline only
 2 2  3 constraint 1 fixed bare no_refine color blue // outline only
 3 3  4 constraint 1 fixed bare no_refine color blue // outline only
 4 4  1 constraint 1 fixed bare no_refine color blue // outline only
 5 5  6 constraint 1 fixed bare no_refine color blue // outline only
 6 6  7 constraint 1 fixed bare no_refine color blue // outline only
 7 7  8 constraint 1 fixed bare no_refine color blue // outline only
 8 8  5 constraint 1 fixed bare no_refine color blue // outline only
 9 1  9 constraint 1 fixed bare no_refine color blue // outline only
10 2  9 constraint 1 fixed bare no_refine color blue // outline only
11 3  9 constraint 1 fixed bare no_refine color blue // outline only
12 4  9 constraint 1 fixed bare no_refine color blue // outline only
```

```
13 1  5              fixed bare no_refine color blue // outline only
14 2  6              fixed bare no_refine color blue // outline only
15 3  7              fixed bare no_refine color blue // outline only
16 4  8              fixed bare no_refine color blue // outline only

21 11 12 constraint 2  9 wettedArea wettedEnergy // bubble
22 12 13 constraint 2 10 wettedArea wettedEnergy // bubble
23 13 14 constraint 2 10 wettedArea wettedEnergy // bubble
24 14 11 constraint 2  9 wettedArea wettedEnergy // bubble

face
1 21 22 23 24 constraint 3 myvol // bubble

body
1 1 density 0

read
lh := histogram(edge where not fixed, length)
ah := histogram(face, area)
vug := {{V 3; u 3; g3}3;}
r_outline := {refine edge where on_constraint 1}
changeVol := {myvol.target := (frac*CONST_TANKVOL)}
calcFrac := {print (myvol.value/CONST_TANKVOL)}
eraseHeightFile := {printf """>>>outputHeightFile}
eraseCLFile := {printf """>>>outputCLFile}
eraseEnergyFile := {printf """>>>outputEnergyFile}
outputHeight := {printf "%.16g, %.16g\n", frac,
vertex[14].z>>outputHeightFile}
outputCL := {foreach vertex vv where on_constraint 2 do {
              printf "%.2g %.16g %.16g %.16g\n ",frac,x,y,z>>outputCLFile
              };
            }
outputCLFile := "test.txt"
pertr := {set vertex where on_constraint 2 or on_constraint 3 z z-0.01*y};
pertl := {set vertex where on_constraint 2 or on_constraint 3 z z+0.01*y};
ar := {printf "%.4f\n", (wettedArea.value)}
```

```
drain := {frac := maximum(0.001, ceil((frac-0.001)*1000-0.5)/1000);
changeVol}
fill  := {frac := ceil((frac+0.01)*1000-0.5)/1000; changeVol}

averageVert := {fix vertex where on_constraint 2; {V;u}100; unfix vertex
where on_constraint 2}
averageEdge := {fix vertex where on_constraint 3; {V;u}10; unfix vertex where
on_constraint 3}
converge := {{averageVert; g20}10; averageVert}
slow_converge := {{{V;u}10; vug}20; {vug; g20}10};
//{vug 20; {averageVertSmall; g20}20; averageVert; {averageVertSmall;
g100}20; averageVert; g1000}
//converge := {quiet on; autodisplay off; vug 20; {vug; g20}50; {vug; g50}20;
{vug; g100}10; g1000; quiet off; autodisplay on}
//{quiet on; autodisplay off; vug 3; {{V;u}100; vug 10}20; {vug; g5}5;
{V;u}100; {vug; g20}20; {V;u}100; {vug; g50}20; g1000; quiet off; autodisplay
on};

checkAsymmetric := {r_1 := 0; r_2 := 0; z_1 := 0; z_2 := 0; numVert := 0;
                    foreach vertex vv where on_constraint 2 do {
                      r_1 := r_1 + sqrt(x^2 + y^2);
                      r_2 := r_2 + x^2 + y^2;
                      z_1 := z_1 + z;
                      z_2 := z_2 + z^2;
                      numVert := numVert + 1;
                    };
                    r_std_old := r_std; z_std_old := z_std;
                    r_std := sqrt((r_2/numVert) - (r_1/numVert)^2); z_std :=
sqrt((z_2/numVert) - (z_1/numVert)^2);
                    if (r_std + z_std) >= 0.05 then {asymmetric := 1} else
{asymmetric := 0};
                    diff := maximum(abs(r_std - r_std_old), abs(z_std -
z_std_old));
                   };
checkToroidal :=  {theta0 := sum(vertex where original == 12,  atan2(y,
x));//theta0 := atan2(ymom.value, xmom.value);
                   theta1_old := theta1; theta2_old := theta2;
```

```
                  theta1 :=  max(vertex where on_constraint 10 and
sqrt(x^2+y^2) > 0.5, theta0/abs(theta0)*(atan2(-x*sin(theta0) +
y*cos(theta0), x*cos(theta0) + y*sin(theta0)) + pi/4) % (2*pi) - pi/4);
                  theta2 :=  max(vertex where on_constraint 9 and
sqrt(x^2+y^2) > 0.5, -theta0/abs(theta0)*(atan2(-x*sin(theta0) +
y*cos(theta0), x*cos(theta0) + y*sin(theta0)) - pi/4) % (2*pi) - pi/4);
                  diff2 := (theta1+theta2) - (theta1_old+theta2_old);
                  if ( theta1 + theta2 - 2*pi) > 0 then {
                    toroidal := 1
                  } else {
                    toroidal := 0
                 }};

outputValues := { hit_vert := 0;
                  checkAsymmetric;
                  hit_vert := 0; toroidal := 0;
                  foreach vertex vv where hit_constraint 3 do {
                    hit_vert := hit_vert + 1;
                  };
                  checkToroidal;
                  printf "%.2f %.3f %d %d %d %.16f %.16f %.16f\n", ang, frac,
asymmetric, toroidal, hit_vert, r_std, z_std, total_energy>>outputEnergyFile
                }

tilt := {checkAsymmetric; if (asymmetric == 0) then {pertr 20; vug}5;
slow_converge; checkAsymmetric};
convergeLoopSlow := {conj_grad on; diff := 100; energyDiff := -100;
oldEnergy := total_energy; while ((diff > 10^-4) && (energyDiff < 0) &&
(toroidal == 0)) do { vug; checkAsymmetric; checkToroidal; energyDiff :=
total_energy - oldEnergy; oldEnergy := total_energy}; conj_grad off};

maxLength := 0.05;
r2 := {refine edge where length > maxLength and not fixed};

maxDiff := 10^-8;
//convergeLoopRefine := {quiet on; conj_grad on; diff := 100; while ((diff >
maxDiff) && (toroidal == 0) && (asymmetric == 1)) do { r2; averageVert; g20;
checkAsymmetric; checkToroidal; }; conj_grad off; quiet off};
```

```
convergeLoopRefine := {quiet on; conj_grad on; diff := 100; numNoChange := 0;
while ((numNoChange < 5) && (toroidal == 0) && (asymmetric == 1)) do { r2;
averageVert; g20; checkAsymmetric; checkToroidal; if (diff < maxDiff) then
{numNoChange := numNoChange + 1; print numNoChange} else {numNoChange := 0}};
conj_grad off; quiet off};
convergeLoopRefineNoGradLong := {quiet on; conj_grad off; diff := 100;
numNoChange := 0; while ((numNoChange < 5) && (toroidal == 0) && (asymmetric
== 1)) do { r2; averageVert; g20; checkAsymmetric; checkToroidal; if (diff <
maxDiff) then {numNoChange := numNoChange + 1; print numNoChange} else
{numNoChange := 0}}; conj_grad off; quiet off};
convergeLoopRefineNoGrad := {quiet on; conj_grad off; diff := 100; while
(diff > 10^-7) do { r2; averageVert; g20; checkAsymmetric; }; quiet off};
convergeLoop := {quiet on; conj_grad on; diff := 100; while ((diff > 10^-8)
&& (toroidal == 0) && (asymmetric == 1)) do { averageVert; g20;
checkAsymmetric; checkToroidal; }; conj_grad off; quiet off};
averageLoop := {diff := 100; while (diff > 0.0003) do { {{V;u}10; vug};
checkAsymmetric; checkToroidal; }};

endLength := 0.05;
convergeLoopRefineLoop := {maxLength := 0.08; while (maxLength > endLength -
0.005) do {convergeLoopRefine; maxLength := maxLength - 0.015};};

numFile := 4; maxAng := 90;

setNoDumpOn  := {ang.no_dump on;  frac.no_dump on};
setNoDumpOff := {ang.no_dump off; frac.no_dump off};
outputEnergyFile := sprintf "Asymmetric/r%.3f_z0_%.3f/r%.3f_z0_%.3f_t%d.txt",
CONST_rNorm, CONST_zMax, CONST_rNorm, CONST_zMax, ang
dumpCurr := {setNoDumpOff; dump sprintf
"Asymmetric/Dump/r%.3f_z0_%.3f_theta%d_f%.1f_4.dmp", CONST_rNorm, CONST_zMax,
ang, frac*100; setNoDumpOn}

tilt2 := {fix vertex where on_constraint 2; {pertr 5; vug; averageVert}20;
unfix vertex where on_constraint 2; vug 50; {vug; g20}50};

runIndividual := {convergeLoopRefineLoop; endLength := 0.015;
convergeLoopRefineLoop; maxLength := 0.015; convergeLoopRefine; maxLength :=
0.01; convergeLoopRefine; dumpCurr}
```

```
test := {foreach edge ee where on_constraint 2 do {
          foreach ee.vertex do {
            printf "%.2g %.16g %.16g %.16g\n ",frac,x,y,z>>outputCLFile
          }
        };
        };


diff := 100; diff2 := 100; diff3 := 100; currMaxOld := 0;
checkMaxHeight := {currMax := max(vertex where on_constraint 2, z);
                   diff3 := currMax - currMaxOld;
                   currMaxOld := currMax;
                  }

fixVert := {fix vertex where on_constraint 2};
unfixVert := {unfix vertex where on_constraint 2};
increment := {outputValues; dumpCurr; fill}
avgv := {{V;u}100; vug};
avgvFix := {fixVert; {V;u}100; unfixVert};
converge := {avgv 10; vug 20; {vug; g20}5; conj_grad on; avgv 2; {vug;
g20}20; U; U; {vug; g100}; U; fixVert; {V;u}1000; unfixVert; U; {vug; g100};
U};

runAsymmetric := {};
runAsymmetric := {converge; increment; runAsymmetric};

runAsymmetricUntil := {};
runAsymmetricUntil := {converge; increment; if (((asymmetric == 1) &&
(toroidal == 0))) then {runAsymmetricUntil}};

shrink := {avgv 10; vug 5; frac := 0.05; changeVol; vug 5; avgv 10; vug 5;
frac := 0.03; changeVol; vug 5; avgv 10; vug 5; frac := 0.01; changeVol; vug
5; avgv 10; vug 5}

runCase := {};
runCase := {frac := 0.05; changeVol; avgv 3; tilt; avgv 5; shrink; avgv 100;
converge; avgv 10; converge; runAsymmetricUntil; ang := ang + 5; runCase};
```

APPENDIX G. SURFACE EVOLVER SOURCE CODE – TORISPHERICAL DOME, RING SOLUTION

```
#define RT 1
#define ZC RT
PARAMETER ang = 90
#define cosca cos(ang*pi/180)
#define sinca sin(ang*pi/180)
#define eps 1e-9
PARAMETER outputCLFile = ""
PARAMETER outputHeightFile = ""
PARAMETER outputEnergyFile = ""
PARAMETER r_std = 0
PARAMETER z_std = 0
PARAMETER maxInd = 6
define maxFrac_mat real[maxInd]
define minFrac_mat real[maxInd]
define theta_mat real[maxInd]
PARAMETER index = 1
PARAMETER asymmetric = 1;
PARAMETER toroidal = 0;
PARAMETER diff = 100;
PARAMETER theta1 = 0;
PARAMETER theta2 = 0;
PARAMETER cyl_vol (pi*RT^2*ZC)
#define cyl_eqn (RT^2 - (x^2 + y^2) = 0)

// TORISPHERICAL
#define z_max  (0.75)
#define r_norm (0.6)

#define rS (r_norm*z_max)
#define RH ((2*rS*RT - RT^2 - z_max^2)/(2*(rS-z_max)))

#define zCenter (-sqrt(-RT^2 + 2*RT*rS - 2*RH*rS + RH^2)) //z of sphere
center
```

```
#define zcr (rS*sqrt((RH-RT)*(RH - 2*rS + RT))/(RH-rS)) // r of start of
sphere
#define rcr (sqrt(rS^2 - zcr^2) + RT - rS)
#define toris_eqn (z > ZC ? (z > (zcr+ZC) ? RH^2 - (x^2 + y^2 + (z-zCenter-
ZC)^2 = 0) :\
                          ((sqrt(rS^2 - (z-ZC)^2) + RT - rS -
sqrt(x^2+y^2)) = 0)) :\
                  cyl_eqn)
#define toris_vol (pi/3*(2*(RH+zCenter)*RH^2 - (2*rS^2+(RT-
rS)^2+2*rS*RH)*(RH-(RH+zCenter)) \
                  + 3*rS^2*(RT-rS)*asin((RH-(RH+zCenter))/(RH-rS))) +
cyl_vol)
#define toris_H (RH + zCenter + ZC - RT)
#define toris_z (sqrt(x^2+y^2) < rcr ? (sqrt(RH^2 - (x^2+y^2)) + ZC
+zCenter) : (sqrt(rS^2 - (sqrt(x^2+y^2)-RT+rS)^2) + ZC))
#define toris_SA ((rcr^2 + (RH-zcr+zCenter)^2)/2 + rS*(zcr + (RT-
rS)*asin(zcr/rS)))
#define toris_wetting ((toris_SA) + (z_calc < (zcr) ? -(rS*((z_calc) + (RT-
rS)*asin((z_calc)/rS))) : RH/RT*(zcr-z_calc) -(rS*((zcr) + (RT-
rS)*asin((zcr)/rS)))))
parameter CONST_zMax = z_max
parameter CONST_rNorm = r_norm

// Poles
#define t_calc ((z-ZC)/sqrt(x^2+y^2))
#define r_calc (t_calc >= zcr/rcr ? (t_calc*zCenter+sqrt(RH^2*(t_calc^2+1)-
zCenter^2))/(t_calc^2+1) : (sqrt(-RT^2*t_calc^2+2*RT*rS*t_calc^2+rS^2) + RT -
rS)/(t_calc^2+1) )
#define z_calc (t_calc*r_calc)
#define x_calc (r_calc*x/sqrt(x^2+y^2))
#define y_calc (r_calc*y/sqrt(x^2+y^2))

// DEFINE MODEL
parameter frac := 0.45
parameter CONST_TANKVOL = toris_vol
#define           tankeqn toris_eqn
#define              delH toris_H
#define           capzEqn toris_z
```

```
#define    wettingEqnPart toris_wetting
#define          endCapSA toris_SA

parameter maxFrac = maximum(0.1, ceil((1*(CONST_TANKVOL - cyl_vol - pi/6*(1-
sinca)/cosca*(3 + ((1-sinca)/cosca)^2))/CONST_TANKVOL)/0.01)*0.01 + 0.05)
parameter minFrac = 0.01

constraint 1 // draw the tank
formula: tankeqn

constraint 2 convex // on tank wall
formula: tankeqn

constraint 3 nonnegative // inside the tank
formula: tankeqn

constraint 4 // on center axis
formula: x = 0

constraint 5 // on center axis
formula: y = 0

quantity myvol fixed = (CONST_TANKVOL - cyl_vol) method facet_vector_integral
vector_integrand:
q1: 0
q2: 0
q3: (capzEqn - z)

quantity wettedEnergy energy method edge_vector_integral
vector_integrand
q1: (z > ZC ?  cosca*(y_calc)/(r_calc)^2 * (wettingEqnPart) :
RT*cosca*y/(x^2+y^2) * ((ZC-z) + (endCapSA)))
q2: (z > ZC ? -cosca*(x_calc)/(r_calc)^2 * (wettingEqnPart) : -
RT*cosca*x/(x^2+y^2) * ((ZC-z) + (endCapSA)))
q3: 0

quantity wettedArea info_only method edge_vector_integral
vector_integrand:
```

```
q1: (z > ZC ? -(y_calc)/(r_calc)^2 * (wettingEqnPart) :  RT*cosca*y/(x^2+y^2)
* ((ZC-z) + (endCapSA)))
q2: (z > ZC ?  (x_calc)/(r_calc)^2 * (wettingEqnPart) : -RT*cosca*x/(x^2+y^2)
* ((ZC-z) + (endCapSA)))
q3: 0

vertex
 1  RT   0       ZC fixed constraint 1   // outline only
 2   0  RT       ZC fixed constraint 1   // outline only
 3 -RT   0       ZC fixed constraint 1   // outline only
 4   0 -RT       ZC fixed constraint 1   // outline only
 5  RT   0        0 fixed constraint 1   // outline only
 6   0  RT        0 fixed constraint 1   // outline only
 7 -RT   0        0 fixed constraint 1   // outline only
 8   0 -RT        0 fixed constraint 1   // outline only
 9   0   0  (ZC+delH) fixed constraint 1   // outline only

11  RT   0  ZC constraint 2 // bubble
12   0  RT  ZC constraint 2 // bubble
13 -RT   0  ZC constraint 2 // bubble
14   0 -RT  ZC constraint 2 // bubble

15  0.5    0 (ZC+delH) constraint 2 // bubble
16    0  0.5 (ZC+delH) constraint 2 // bubble
17 -0.5    0 (ZC+delH) constraint 2 // bubble
18    0 -0.5 (ZC+delH)constraint 2 // bubble

edge
 1 1  2 constraint 1 fixed bare no_refine color blue // outline only
 2 2  3 constraint 1 fixed bare no_refine color blue // outline only
 3 3  4 constraint 1 fixed bare no_refine color blue // outline only
 4 4  1 constraint 1 fixed bare no_refine color blue // outline only
 5 5  6 constraint 1 fixed bare no_refine color blue // outline only
 6 6  7 constraint 1 fixed bare no_refine color blue // outline only
 7 7  8 constraint 1 fixed bare no_refine color blue // outline only
 8 8  5 constraint 1 fixed bare no_refine color blue // outline only
 9 1  9 constraint 1 fixed bare no_refine color blue // outline only
10 2  9 constraint 1 fixed bare no_refine color blue // outline only
```

```
11 3  9 constraint 1 fixed bare no_refine color blue // outline only
12 4  9 constraint 1 fixed bare no_refine color blue // outline only
13 1  5              fixed bare no_refine color blue // outline only
14 2  6              fixed bare no_refine color blue // outline only
15 3  7              fixed bare no_refine color blue // outline only
16 4  8              fixed bare no_refine color blue // outline only

21 11 12 constraint 2 wettedArea wettedEnergy // bubble
22 12 13 constraint 2 wettedArea wettedEnergy // bubble
23 13 14 constraint 2 wettedArea wettedEnergy // bubble
24 14 11 constraint 2 wettedArea wettedEnergy // bubble

25 16 15 constraint 2 wettedArea wettedEnergy // bubble
26 17 16 constraint 2 wettedArea wettedEnergy // bubble
27 18 17 constraint 2 wettedArea wettedEnergy // bubble
28 15 18 constraint 2 wettedArea wettedEnergy // bubble

29 11 15 constraint 3 // bubble
30 12 16 constraint 3 // bubble
31 13 17 constraint 3 // bubble
32 14 18 constraint 3 // bubble

face
1 21 30 25 -29 constraint 3 myvol // bubble
2 22 31 26 -30 constraint 3 myvol // bubble
3 23 32 27 -31 constraint 3 myvol // bubble
4 24 29 28 -32 constraint 3 myvol // bubble



body
1 1 2 3 4 density 0

read
lh := histogram(edge where not fixed, length)
ah := histogram(face, area)
vug := {{V 3; u 3; g3}3;}
r_outline := {refine edge where on_constraint 1}
changeVol := {myvol.target := (frac*CONST_TANKVOL)}
```

```
calcFrac := {print (myvol.value/CONST_TANKVOL)}
//drain := {frac := frac - 0.001; changeVol; vug; g5; {vug; g50}3}
//fill := {frac := frac + 0.001; changeVol; vug; g5; {vug; g50}3}
eraseHeightFile := {printf """>>>outputHeightFile}
eraseCLFile := {printf """>>>outputCLFile}
eraseEnergyFile := {printf """>>>outputEnergyFile}
outputHeight := {printf "%.16g, %.16g\n", frac,
vertex[14].z>>outputHeightFile}
outputCL := {foreach vertex vv where on_constraint 2 do {
              printf "%.2g %.16g %.16g %.16g\n ",frac,x,y,z>>outputCLFile
              };
            }
outputCLFile := "test.txt"
pertr := {set vertex where on_constraint 2 or on_constraint 3 z z-0.01*y};
pertl := {set vertex where on_constraint 2 or on_constraint 3 z z+0.01*y};
ar := {printf "%.4f\n", (wettedArea.value)}

drain := {frac := maximum(0.001, ceil((frac-0.001)*1000-0.5)/1000);
changeVol}
fill  := {frac := ceil((frac+0.01)*1000-0.5)/1000; changeVol}

averageVert := {fix vertex where on_constraint 2; {V;u}100; unfix vertex
where on_constraint 2}
averageEdge := {fix vertex where on_constraint 3; {V;u}10; unfix vertex where
on_constraint 3}
converge := {{averageVert; g20}10; averageVert}
slow_converge := {{{V;u}10; vug}20; {vug; g20}10};
//{vug 20; {averageVertSmall; g20}20; averageVert; {averageVertSmall;
g100}20; averageVert; g1000}
//converge := {quiet on; autodisplay off; vug 20; {vug; g20}50; {vug; g50}20;
{vug; g100}10; g1000; quiet off; autodisplay on}
//{quiet on; autodisplay off; vug 3; {{V;u}100; vug 10}20; {vug; g5}5;
{V;u}100; {vug; g20}20; {V;u}100; {vug; g50}20; g1000; quiet off; autodisplay
on};

checkAsymmetric := {r_1 := 0; r_2 := 0; z_1 := 0; z_2 := 0; numVert := 0;
                    foreach vertex vv where on_constraint 2 do {
                      r_1 := r_1 + sqrt(x^2 + y^2);
```

```
                    r_2 := r_2 + x^2 + y^2;
                    z_1 := z_1 + z;
                    z_2 := z_2 + z^2;
                    numVert := numVert + 1;
                  };
                  r_std_old := r_std; z_std_old := z_std;
                  r_std := sqrt((r_2/numVert) - (r_1/numVert)^2); z_std :=
sqrt((z_2/numVert) - (z_1/numVert)^2);
                  if (r_std + z_std) >= 0.05 then {asymmetric := 1} else
{asymmetric := 0};
                  diff := maximum(abs(r_std - r_std_old), abs(z_std -
z_std_old));
                 };
checkToroidal :=  {theta0 := sum(vertex where original == 12,  atan2(y,
x));//theta0 := atan2(ymom.value, xmom.value);
                  theta1_old := theta1; theta2_old := theta2;
                  theta1 :=  max(vertex where on_constraint 10 and
sqrt(x^2+y^2) > 0.5, theta0/abs(theta0)*(atan2(-x*sin(theta0) +
y*cos(theta0), x*cos(theta0) + y*sin(theta0)) + pi/4) % (2*pi) - pi/4);
                  theta2 :=  max(vertex where on_constraint 9 and
sqrt(x^2+y^2) > 0.5, -theta0/abs(theta0)*(atan2(-x*sin(theta0) +
y*cos(theta0), x*cos(theta0) + y*sin(theta0)) - pi/4) % (2*pi) - pi/4);
                  diff2 := (theta1+theta2) - (theta1_old+theta2_old);
                  if ( theta1 + theta2 - 2*pi) > 0 then {
                    toroidal := 1
                  } else {
                    toroidal := 0
                 }};

outputValues := { hit_vert := 0;
                 checkAsymmetric;
                 hit_vert := 0; toroidal := 0;
                 foreach vertex vv where hit_constraint 3 do {
                   hit_vert := hit_vert + 1;
                 };
                 checkToroidal;
                 printf "%.2f %.3f %d %d %d %.16f %.16f %.16f\n", ang, frac,
asymmetric, toroidal, hit_vert, r_std, z_std, total_energy>>outputEnergyFile
```

```
                }

tilt := {checkAsymmetric; if (asymmetric == 0) then {pertr 20; vug}5;
slow_converge; checkAsymmetric};
convergeLoopSlow := {conj_grad on; diff := 100; energyDiff := -100;
oldEnergy := total_energy; while ((diff > 10^-4) && (energyDiff < 0) &&
(toroidal == 0)) do { vug; checkAsymmetric; checkToroidal; energyDiff :=
total_energy - oldEnergy; oldEnergy := total_energy}; conj_grad off};

maxLength := 0.05;
r2 := {refine edge where length > maxLength and not fixed};

maxDiff := 10^-8;
//convergeLoopRefine := {quiet on; conj_grad on; diff := 100; while ((diff >
maxDiff) && (toroidal == 0) && (asymmetric == 1)) do { r2; averageVert; g20;
checkAsymmetric; checkToroidal; }; conj_grad off; quiet off};
convergeLoopRefine := {quiet on; conj_grad on; diff := 100; numNoChange := 0;
while ((numNoChange < 5) && (toroidal == 0) && (asymmetric == 1)) do { r2;
averageVert; g20; checkAsymmetric; checkToroidal; if (diff < maxDiff) then
{numNoChange := numNoChange + 1; print numNoChange} else {numNoChange := 0}};
conj_grad off; quiet off};
convergeLoopRefineNoGradLong := {quiet on; conj_grad off; diff := 100;
numNoChange := 0; while ((numNoChange < 5) && (toroidal == 0) && (asymmetric
== 1)) do { r2; averageVert; g20; checkAsymmetric; checkToroidal; if (diff <
maxDiff) then {numNoChange := numNoChange + 1; print numNoChange} else
{numNoChange := 0}}; conj_grad off; quiet off};
convergeLoopRefineNoGrad := {quiet on; conj_grad off; diff := 100; while
(diff > 10^-7) do { r2; averageVert; g20; checkAsymmetric; }; quiet off};
convergeLoop := {quiet on; conj_grad on; diff := 100; while ((diff > 10^-8)
&& (toroidal == 0) && (asymmetric == 1)) do { averageVert; g20;
checkAsymmetric; checkToroidal; }; conj_grad off; quiet off};
averageLoop := {diff := 100; while (diff > 0.0003) do { {{V;u}10; vug};
checkAsymmetric; checkToroidal; }};

endLength := 0.05;
convergeLoopRefineLoop := {maxLength := 0.08; while (maxLength > endLength -
0.005) do {convergeLoopRefine; maxLength := maxLength - 0.015};};
```

```
numFile := 4; maxAng := 90;

setNoDumpOn  := {ang.no_dump on;  frac.no_dump on};
setNoDumpOff := {ang.no_dump off; frac.no_dump off};
outputEnergyFile := sprintf "Ring/r%.3f_z0_%.3f/r%.3f_z0_%.3f_t%d.txt",
CONST_rNorm, CONST_zMax, CONST_rNorm, CONST_zMax, ang
dumpCurr := {setNoDumpOff; dump sprintf
"Ring/Dump/r%.3f_z0_%.3f_theta%d_f%.1f_4.dmp", CONST_rNorm, CONST_zMax, ang,
frac*100; setNoDumpOn}

tilt2 := {fix vertex where on_constraint 2; {pertr 5; vug; averageVert}20;
unfix vertex where on_constraint 2; vug 50; {vug; g20}50};

runIndividual := {convergeLoopRefineLoop; endLength := 0.015;
convergeLoopRefineLoop; maxLength := 0.015; convergeLoopRefine; maxLength :=
0.01; convergeLoopRefine; dumpCurr}

test := {foreach edge ee where on_constraint 2 do {
          foreach ee.vertex do {
            printf "%.2g %.16g %.16g %.16g\n ",frac,x,y,z>>outputCLFile
          }
        };
        };

diff := 100; diff2 := 100; diff3 := 100; currMaxOld := 0;
checkMaxHeight := {currMax := max(vertex where on_constraint 2, z);
                   diff3 := currMax - currMaxOld;
                   currMaxOld := currMax;
                  }

fixVert := {fix vertex where on_constraint 2};
unfixVert := {unfix vertex where on_constraint 2};
increment := {outputValues; dumpCurr; fill}
avgv := {{V;u}100; vug};
avgvFix := {fixVert; {V;u}100; unfixVert};
converge := {avgv 10; vug 20; {vug; g20}5; conj_grad on; avgv 2; {vug;
g20}20; U; U; {vug; g100}; U; fixVert; {V;u}1000; unfixVert; U; {vug; g100};
U};
```

```
shrink := {avgv 10; vug 5; frac := 0.05; changeVol; vug 5; avgv 10; vug 5;
frac := 0.03; changeVol; vug 5; avgv 10; vug 5; frac := 0.01; changeVol; vug
5; avgv 10; vug 5}
incrementAng := {outputValues; dumpCurr; ang := ang + 1; outputEnergyFile :=
sprintf "Ring/r%.3f_z0_%.3f/r%.3f_z0_%.3f_t%d.txt", CONST_rNorm, CONST_zMax,
CONST_rNorm, CONST_zMax, ang}
runRingAng := {};
runRingAng := {converge; incrementAng; runRingAng};

setNoDumpOn;

r_outline 6

showq
ang := 5;
outputEnergyFile := sprintf "Ring/r%.3f_z0_%.3f/r%.3f_z0_%.3f_t%d.txt",
CONST_rNorm, CONST_zMax, CONST_rNorm, CONST_zMax, ang

frac := 0.05; changeVol;

r;
{r; vug}
{vug;g 20}10
{r; vug}
refine edge where on_constraint 2
frac := 0.02; changeVol;
vug 5; {vug; g20}10
```

APPENDIX H. LIQUID RING ANALYTICAL SOLUTION CODE

EQUATIONS

The generatrix of an unduloid is parameterized by the equations $x(u)$ and $y(u)$ with free parameters a and b, obtained from the Wolfram Demonstrations Project (Zeleny, The Unduloid - Wolfram Demonstrations Project, 2014).

$$x(u) = \int_0^u \sqrt{a^2 \sin^2\phi + b^2 \cos^2\phi}\, d\phi + \frac{\sqrt{a^2 - b^2}\sin u \left(\sqrt{a^2 - b^2}\cos u + a\right)}{\sqrt{a^2 \sin^2 u + b^2 \cos^2 u}}$$

$$y(u) = \frac{b\left(\sqrt{a^2 - b^2}\cos u + a\right)}{\sqrt{a^2 \sin^2 u + b^2 \cos^2 u}}$$

The generatrix of a nodoid is parameterized by the equations $x(u)$ and $y(u)$ with free parameters a and β, obtained from the Wolfram Demonstrations Project (Zeleny, Delaunay Nodoids - Wolfram Demonstrations Project, 2014).

$$x(u) = a\left(1 - \cos u + \int_0^u \frac{\sin^2\alpha}{\sqrt{\sin^2\alpha + \beta}}\, d\alpha\right)$$

$$y(u) = a\left(\sin u + \sqrt{\sin^2 u + \beta}\right)$$

MAIN

```
clear; clc; close all; tic

N_b = 201;
N_a = 201;

outputUnduloidValues = 1;
outputNodoidValues   = 1;

n_mat = 2;
z0_mat = 0.5;
theta_deg_mat = 0:0.001:90;
theta_deg_mat = theta_deg_mat(mod(theta_deg_mat, 0.01)~=0);

numWrite = 0;
for n = n_mat
```

```
for z0 = z0_mat
for theta_deg = theta_deg_mat
    if outputUnduloidValues
        b_mat = linspace(0, 2, N_b);
        b_mat = [b_mat(1:end-1), linspace(2, 10, N_b), 100, 1000, 10000];
        numWrite = calcUnduloidCase(b_mat, theta_deg, n, z0, 1) + numWrite;
    end
    if outputNodoidValues
        a_range = linspace(0, 1/(2*cosd(theta_deg)), N_a);
        dif_range = calcD(a_range, z0, n, theta_deg);
        [~, ind] = min(abs(dif_range));
        a_mat = linspace(a_range(max(ind-2,1)), 1/(2*cosd(theta_deg)), N_a);
        numWrite = calcNodoidCase(a_mat, theta_deg, n, z0, 1) + numWrite;
    end
end
end
end

function res = calcD(a_mat, z0, n, theta_deg)
    theta = theta_deg*pi/180;
    for i = 1:length(a_mat)
        a = a_mat(i);
        B = 1/a^2 - 2/a*cos(theta);

        u_cr = pi - asin((1-a^2*B)/(2*a));
        u = linspace(u_cr, pi, 10001);

        r = a*(sin(u) + sqrt(sin(u).^2 + B));
        z_tank_der = z0*(1-r.^n).^(1/n-1).*r.^(n-1);
        D = pi - u - theta - atan(z_tank_der);
        res(i) = max(real(D));
    end
end
```

calcUnduloidCase.m

```
function numWrite = calcUnduloidCase(b_mat, theta_deg, n, z0, outputValues)
    eps = 1e-9;
    maxIter = 100000;
    N = 10001;

    theta = theta_deg*pi/180;
    j = 1;

    f = []; E = []; b_res = []; zLower = []; zUpper = [];
    for b = b_mat
        a = (b^2+1)/(2*cos(theta));

        u_cr = acos((1-b^2)/(1+b^2)*a/sqrt(a^2-b^2));
        u_range = linspace(u_cr, pi, N); u_range = u_range(2:end-1);
        r_range = b*(sqrt(a^2-b^2)*cos(u_range)+a)./sqrt(a^2*sin(u_range).^2+
b^2*cos(u_range).^2);

        dz_dr_inv = sin(u_range)*sqrt(a^2-b^2)/b;
        dz_dr_tank = z0*(1-r_range.^n).^(1/n-1).*r_range.^(n-1);
        dz_dr_tank(1) = -inf;

        dif_range = pi/2-theta-atan(dz_dr_inv)-atan(dz_dr_tank);
        indices = islocalmin(abs(dif_range));
        X = 1:N; indices = X(indices);

        for ind = indices
            if (dif_range(ind)*dif_range(ind+1)>0 &&
dif_range(ind)*dif_range(ind-1)>0)
                continue
            end

            u_max_upper = max([u_range(ind-1), u_range(ind+1)]);
            u_max_lower = min([u_range(ind-1), u_range(ind+1)]);

            r = b*(sqrt(a^2-b^2)*cos(u_max_upper)+a)./
sqrt(a^2*sin(u_max_upper).^2+b^2*cos(u_max_upper).^2);
            dz_dr_inv = sin(u_max_upper)*sqrt(a^2-b^2)/b;
```

```
            dz_dr_tank = z0*(1-r.^n).^(1/n-1).*r.^(n-1);
            dif = pi/2-theta-atan(dz_dr_inv)-atan(dz_dr_tank);
            s = sign(dif);

            dif = 100; iter = 1;
            while abs(dif) > eps && iter < maxIter
                u_max = (u_max_upper+u_max_lower)/2;

                r = b*(sqrt(a^2-b^2)*cos(u_max)+a)./
sqrt(a^2*sin(u_max).^2+b^2*cos(u_max).^2);
                dz_dr_inv = sin(u_max)*sqrt(a^2-b^2)/b;
                dz_dr_tank = z0*(1-r.^n).^(1/n-1).*r.^(n-1);
                dif = pi/2-theta-atan(dz_dr_inv)-atan(dz_dr_tank);

                if s*dif > 0
                    u_max_upper = u_max;
                else
                    u_max_lower = u_max;
                end
                iter = iter + 1;
            end

            if abs(dif) > 1e-6
                continue
            end

            if u_max < u_cr
                continue
            end

            u = linspace(u_cr, u_max, N);
            r = b*(sqrt(a^2-
b^2)*cos(u)+a)./sqrt(a^2*sin(u).^2+b^2*cos(u).^2);

            diffR = abs(diff(r));

            maxDiffR = 1e-3;
            maxN = 1000000;
```

```
            while max(diffR) > maxDiffR && length(u) < maxN
                u_new = (u([diffR, 0] > maxDiffR) + u([0, diffR] >
maxDiffR))/2;
                r_new = b*(sqrt(a^2-b^2)*cos(u_new)+a)./
sqrt(a^2*sin(u_new).^2+b^2*cos(u_new).^2);
                res = sortrows([[u, u_new]', [r, r_new]']);
                u = res(:, 1)'; r = res(:, 2)';
                diffR = abs(diff(r));
            end

            if max(abs(diff(r))) > maxDiffR
                disp(max(abs(diff(r))))
                continue;
            end

            dz_dr_inv = sin(u)*sqrt(a^2-b^2)/b;

            z_integrand = sqrt(a^2*sin(u).^2 + b^2*cos(u).^2);
            z_integral_0 = integral(@(u) sqrt(a^2*sin(u).^2 + b^2*cos(u).^2),
0, u_cr);
%{
            S = zeros(size(u));
            S(1) = z_integrand(1);
            for i = 2:length(u)
                S(i) = S(i-1) + z_integrand(i);
            end

            z_integral = z_integral_0 + (u(2)-u(1))*(S - 0.5*(z_integrand +
z_integrand(1)));

            z = z_integral + sqrt(a^2-b^2)*sin(u).*(sqrt(a^2-
b^2)*cos(u)+a)./sqrt(a^2*sin(u).^2 + b^2*cos(u).^2);
            z = z - z(end) + z0*(1-r(end)^n)^(1/n);
%}
            z_integral = zeros(size(u));
            z_integral(1) = z_integral_0;
            du = diff(u);
            for i = 2:length(u)
```

```
                z_integral(i) = z_integral(i-1) + (z_integrand(i) +
z_integrand(i-1))*du(i-1)/2;
            end

            z = z_integral + sqrt(a^2-b^2)*sin(u).*(sqrt(a^2-
b^2)*cos(u)+a)./sqrt(a^2*sin(u).^2 + b^2*cos(u).^2);

            z = z - z(end) + z0*(1-r(end)^n)^(1/n);

            if z(1) > 0
                continue
            end

            %% Calculate f
            r_tank = (1 - (z/z0).^n).^(1/n);
            r_tank(z <= 0) = 1;
            V = pi*trapz(z, r_tank.^2 - r.^2);
            R0 = 1; z_cyl = 2;
            Vend_total = 2*pi/(3*n)*R0^2*z0*beta(1/n, 2/n);
            Vtank = (2*Vend_total + pi*R0^2*z_cyl)/2;
            f(j) = V/Vtank;

            %% Calculate Energy
            A_FS = 2*pi*trapz(z, r.*sqrt(1 + dz_dr_inv.^2));
            integrand = @(z1) (1-(z1/z0).^n).^(1/n) .* sqrt(1 + (R0/z0)^2*(1-
(z1/z0).^n).^(2/n-2).*(z1/z0).^(2*n-2));
            A_wet = 2*pi*integral(@(z1) integrand(z1), 0, z(end)) -
2*pi*R0*z(1);
            E(j) = A_FS - cos(theta)*A_wet;
            b_res(j) = b;
            zLower(j) = min(z);
            zUpper(j) = max(z);
            j = j+1;
        end
    end

    numWrite = length(f);
    if outputValues
```

```
        dir = sprintf("Data/n%d_z0_%.3f", n, z0);
        fName = sprintf("Data/n%d_z0_%.3f/Unduloid_theta%g.txt", n, z0,
theta_deg);
        if ~isempty(f)
            mkdir(dir);
            fprintf("Writing %d values to %s.\n", length(f), fName);
            fid = fopen(fName, 'a');
            for i = 1:length(f)
                fprintf(fid, "%.16f %.16f %.16f %.16f %.16f\n", b_res(i),
f(i), E(i), zLower(i), zUpper(i));
            end
            fclose('all');
        end
    end
    toc
end
```

calcNodoidCase.m

```
function numWrite = calcNodoidCase(a_mat, theta_deg, n, z0, outputValues)
    eps = 1e-9;
    N = 10001;
    maxIter = 100000;

    theta = theta_deg*pi/180;
    j = 1;

    f = []; E = []; a_res = []; zLower = []; zUpper = [];
    for a = a_mat
        B = 1/a^2 - 2/a*cos(theta);
        u_cr = pi - asin((1-a^2*B)/(2*a));

        u_range = linspace(u_cr, pi, N);
        r_range = a*(sin(u_range) + sqrt(sin(u_range).^2 + B));

        dz_dr_tank = z0*(1-r_range.^n).^(1/n-1).*r_range.^(n-1);

        dif_range = pi - u_range - theta - atan(dz_dr_tank);
        indices = islocalmin(abs(dif_range));
```

```
        X = 1:N; indices = X(indices);

        for ind = indices
            if (dif_range(ind)*dif_range(ind+1)>0 &&
dif_range(ind)*dif_range(ind-1)>0)
                continue
            end

            u_max_upper = max([u_range(ind-1), u_range(ind+1)]);
            u_max_lower = min([u_range(ind-1), u_range(ind+1)]);

            r = a*(sin(u_max_upper) + sqrt(sin(u_max_upper).^2 + B));
            dz_dr_tank = z0*(1-r.^n).^(1/n-1).*r.^(n-1);
            dif = pi - u_max_upper - theta - atan(dz_dr_tank);
            s = sign(dif);

            dif = 100;
            iter = 0;
            while abs(dif) > eps && iter < maxIter
                u_max = (u_max_upper+u_max_lower)/2;

                r = a*(sin(u_max) + sqrt(sin(u_max).^2 + B));
                dz_dr_tank = z0*(1-r.^n).^(1/n-1).*r.^(n-1);
                dif = pi - u_max - theta - atan(dz_dr_tank);

                if s*dif > 0
                    u_max_upper = u_max;
                else
                    u_max_lower = u_max;
                end
                iter = iter+1;
            end

            if abs(dif) > 1e-6
                continue;
            end

            u = linspace(u_cr, u_max, N);
```

```
            r = a*(sin(u) + sqrt(sin(u).^2 + B));

            diffR = abs(diff(r));

            maxDiffR = 1e-3;
            maxN = 1000000;
            while max(diffR) > maxDiffR && length(u) < maxN
                u_new = (u([diffR, 0] > maxDiffR) + u([0, diffR] >
maxDiffR))/2;
                r_new = a*(sin(u_new) + sqrt(sin(u_new).^2 + B));
                res = sortrows([[u, u_new]', [r, r_new]']);
                u = res(:, 1)'; r = res(:, 2)';
                diffR = abs(diff(r));
            end

            if max(abs(diff(r))) > maxDiffR
                disp(max(abs(diff(r))))
                continue;
            end

            dz_dr = tan(u);

            z_integrand = sin(u).^2./sqrt(sin(u).^2 + B);
            z_integral_0 = integral(@(u) sin(u).^2./sqrt(sin(u).^2 + B), 0,
u_cr);

            S = zeros(size(u));
            S(1) = z_integrand(1);
            for i = 2:length(u)
                S(i) = S(i-1) + z_integrand(i);
            end

            z_integral = z_integral_0 + (u(2)-u(1))*(S - 0.5*(z_integrand +
z_integrand(1)));

            z = a*(1 - cos(u) + z_integral);
            z = z - z(end) + z0*(1-r(end)^n)^(1/n);
```

```
            if z(1) > 0
                continue
            end

            %% Calculate f
            r_tank = (1 - (z/z0).^n).^(1/n);
            r_tank(z <= 0) = 1;
            V = pi*trapz(z, r_tank.^2 - r.^2);
            R0 = 1; z_cyl = 2;
            Vend_total = 2*pi/(3*n)*R0^2*z0*beta(1/n, 2/n);
            Vtank = (2*Vend_total + pi*R0^2*z_cyl)/2;
            f(j) = V/Vtank;

            %% Calculate Energy
            A_FS = 2*pi*trapz(z, r.*sqrt(1 + dz_dr.^-2));

            integrand = @(z1) (1-(z1/z0).^n).^(1/n) .* sqrt(1 + (R0/z0)^2*(1-
(z1/z0).^n).^(2/n-2).*(z1/z0).^(2*n-2));
            A_wet = 2*pi*integral(@(z1) integrand(z1), 0, z(end)) -
2*pi*R0*z(1);
            E(j) = A_FS - cos(theta)*A_wet;

            a_res(j) = a;
            zLower(j) = min(z);
            zUpper(j) = max(z);
            j = j+1;

        end
    end

    numWrite = length(f);
    if outputValues
        dir = sprintf("Data/n%d_z0_%.3f", n, z0);
        fName = sprintf("Data/n%d_z0_%.3f/Nodoid_theta%g.txt", n, z0,
theta_deg);
        if ~isempty(f)
            mkdir(dir);
            fprintf("Writing %d values to %s.\n", length(f), fName);
```

```
            fid = fopen(fName, 'a');
            for i = 1:length(f)
                fprintf(fid, "%.16f %.16f %.16f %.16f %.16f\n", a_res(i),
f(i), E(i), zLower(i), zUpper(i));
            end
            fclose('all');
        end
    end
    toc
end
```

REFERENCES

Bert, C. W., & Hyler, W. S. (1966). Structures and Materials for Solid Propellant Rocket Motor Cases. In *Advances in Space Science and Technology* (Vol. 8, pp. 87-91). Elsevier.

Brakke, K. (2013, August 25). *Surface Evolver*. Retrieved from facstaff.susqu.edu: https://facstaff.susqu.edu/brakke/evolver/evolver.html

Dodge, F. T., & Garza, L. R. (1967). Experimental and Theoretical Studies of Liquid Sloshing at Simulated Low Gravities. *Journal of Applied Mechanics*, 555-562.

Enright, P. J., & Wong, E. C. (1994). Propellant Slosh Models for the Cassini Spacecraft. *Astrodynamics Conference.* American Institute of Aeronautics and Astronautics.

Kirk, D., Storey, J. M., Marsell, B., & Schallhorn, P. (2017). Progress Towards a Microgravity CFD Validation Study Using the ISS SPHERES-SLOSH Experiment. *AIAA/SAE/ASEE Joint Propulsion Conference.* Atlanta: American Institute of Aeronautics and Astronautics.

Lapilli, G., Kirk, D., Gutierrez, H., Schallhorn, P., Marsell, B., Roth, J., & Moder, J. (2015). *Results of Microgravity Fluid Dynamics Captured with the SPHERES-SLOSH Experiment.* Jerusalem: Internautional Astronautical Congress.

Lowndes, E. (2017, March 13). *How to hack the Multi-Terminal: Taking (remote) control of the Sony A7/S/R.* Retrieved from youtube.com: https://www.youtube.com/watch?v=4P63OWKaa0g

Mikhailovsky, C. Y., Stahl, J. M., & Mulkey, H. W. (2020). PACE Propulsion Subsystem Surge Analysis and Testing. *AIAA Propulsion and Energy Forum.* American Institute of Aeronautics and Astronautics.

Reynolds, W. C., & Satterlee, H. M. (1966). *The Dynamic Behavior of Liquids in Moving Containers.* Washington, D.C.: NASA Scientific and Technical Information Division SP-106.

Slooten, A. (2017, June 1). *Research Sony Multi Interface • Aeronavics/ValleyForge Wiki • GitHub*. Retrieved from github.com: https://github.com/Aeronavics/ValleyForge/wiki/Research-Sony-Multi-Interface-Research

STUDIO1productions. (n.d.). *Sony 15 pin Multiport Connector Multi Terminal Pinout*. Retrieved from studio1productions.com: https://www.studio1productions.com/Articles/sony-pinout.htm

Virgin Galactic. (2019). *SpaceShipTwo Payload User's Guide.*

Walls, L. K., Kirk, D., de Luis, J., & Haberbusch, M. S. (2011). Experimental and Numerical Investigation of Reduced Gravity Fluid Slosh Dynamics for the Characterization of Cryogenic Launch and Space Vehicle Propellants. *Cryogenic Engineering Conference.* Spokane: American Institude of Physics.

Zeleny, E. (2014, July). *Delaunay Nodoids - Wolfram Demonstrations Project*. Retrieved from wolfram.com: https://demonstrations.wolfram.com/DelaunayNodoids/

Zeleny, E. (2014, May). *The Unduloid - Wolfram Demonstrations Project*. Retrieved from wolfram.com: https://demonstrations.wolfram.com/TheUnduloid/