Imperial College London
Department of Earth Science and Engineering
MSc in Applied Computational Science and Engineering

Independent Research Project
Project Report

Generation and Refinement of Spherical Volumetric Shell Domains for FEA and IGA applications

by
Sotiris Gkoulimaris

sotiris.gkoulimaris19@imperial.ac.uk
GitHub login: acse-sg3219

Supervisors:
Dr. Adriana Paluszny

1 Literature Review

1.1 Motivation

The outer solar system has been the focus of a number of missions towards the end of the 20th century. The Pioneer and Voyager missions provided us a first glimpse into the numerous satellite worlds of Jupiter and Saturn; but it wouldn't be until later missions when enough data would be collected to begin unravelling the mysteries of those moons (Nimmo and Pappalardo 2016). The Galileo spacecraft reached the Jovian system in 1995. Part of its mission was to collect data from the Galilean satellites. Among them, Europa, stands out as an icy ocean body with striking surface features and the prospect of harboring life in its subsurface ocean (Nimmo and Pappalardo 2016; Vance et al. 2018).

Numerous studies have been conducted throughout the years in attempts to distinguish the moon's internal makeup (Golombek and Banerdt 1990; Billings and Kattenhorn 2005; Bray et al. 2014;), map and model the formation and evolution of its lineament fractures (Helfenstein and Parmentier 1985; Greeley et al. 2000; Kattenhorn 2002; Figueredo and Greeley 2004; Marshall and Kattenhorn 2005; Aydin 2006; Preblich et al. 2007; Dombard et al. 2013; Leonard, Pappalardo, and Yin 2018; Leonard, Yin, and Pappalardo 2020) and understand its geological features and processes (Kattenhorn and Marshall 2006; Goldreich and Mitchell 2010; Han and Showman 2010; Johnston and Montési 2014; Craft et al. 2016; Vance et al. 2018). Europa has a radius of 1561km with a density of about $2989kg/m^3$ (Vance et al. 2018). Gravity field measurements, magnetic induction and surface features, places the estimate for the outer water and trace minerals layers to be about 120km; a liquid ocean underneath an outer ice shell (Nimmo and Pappalardo 2016; Billings and Kattenhorn 2005). The water layer can be further divided into 3 sub-layers: an outer brittle/elastic ice layer, an underlying ductile layer of (potentially) convecting ice and a lower liquid layer (Billings and Kattenhorn 2005). An accurate estimation of the brittle layer thickness and its history is desirable, as many geological and biological processes depend on it (Vance et al. 2018). Thermal analyses making use of convection models, calculations of thermal equilibrium and temperature gradients through the ice shell, give an estimation of a thick icy layer ranging from 10-30km; impact studies infer a thickness of 2.4-19km; and mechanical studies such as: modeling the flexure of the ice shell due to loading at the surface, buoyancy of floating ice rafts and applications of fracture mechanics place the ice layer thickness at 0.1-10km (Billings and Kattenhorn 2005). The discrepancy between the estimates can be attributed to the methods implemented, as thermodynamic models estimate the thickness of the entire ice shell while mechanical models only capture the elastic portion of the shell (Billings and Kattenhorn 2005). There is evidence (Greeley et al. 2000; Leonard, Pappalardo, and Yin 2018; Leonard, Yin, and Pappalardo 2020) that points to a change in the thickness of the ice shell over time. However, further work is required to test this hypothesis.

Europa is not the only planetoid in the Solar System that displays those characteristics (Goldreich and Mitchell 2010; Vance et al. 2018; Skjetne et al. 2020; Weller and Lenardic 2018; Weller, Fuchs, et al. 2019) and modeling challenges. Nevertheless, there has not been much use of modern analysis techniques (FEA) to simulate parts of a moon or a planet. Such an endeavour can be quite tricky, as the required mesh would be quite large and thus would require noumerous elements and a lot of time wasted in the simulation routines. However, CAD applications based on spline objects counteract that issue by using as few elements as possible to accurately define a geometry. The intersection between FEA and CAD is called Isogeometric Analysis (Hughes, Cottrell, and Bazilevs 2005) and takes full advantage of all the capabilities of FEA and CAD. One of the advantages of IGA is the

fact that spheres are defined with a miniscule number of *control points* (Piegl and Tiller 1997) and is ideal for modeling shell objects. In this project a tool will be developed that makes use of all the advntages of IGA, in order to provide a software solution for generating and refining volumetric spherical shells. It will be implemented in pure C++.

2 Domain Generation and Refinement

The first task at hand is to construct a spherical volumetric shell domain or mesh that will be used for the modeling. There are a number of techniques one can implement to generate a spherical volumetric domain, however there are a number of constraints that need to be considered before selection. The estimated surface area of Europa is $3.06 \times 10^7\ km^2$ (Vance et al. 2018). To accurately represent a sphere of such a scale with a mesh, would require a forbidding number of elements. Remeshing of the domain to account for fracture growth would also increase the number of elements, so research on the types of fracture units and their history was conducted in order to estimate the level of resolution required to properly model such features.

2.1 Types of fracture units and limitations in domain generation

A number of geological mappings (Helfenstein and Parmentier 1985; Greeley et al. 2000; Figueredo and Greeley 2004; Leonard, Pappalardo, and Yin 2018) have been done throughout the years. Here I will present a summary of the classification done by Leonard, Pappalardo, and Yin 2018, as they enrich the existing classification efforts by taking into account the highest resolution images available in their mappings. The main categories of surface units are: ridge units, plain units, chaos units and band material. These are further subdivided into smaller groups according to their unique characteristics (eg. texture, morphology), but for now we are primarily concerned with the range of width or aperture size of the fractures, the determining factor for the desired resolution of the domain and size of the fracture elements. Ridge units can have an aperture size that ranges from 400m to more than 10km, plain units are characterised by "a series of small-scale (200-500m) high-albedo ridges", chaos units range from "tens of kilometers to tens of meters" in aperture size and band material can extend to more than 10 km in aperture. Thus, the generated domain will need to account for fractures with 0.1 to a few tens of kilometers in width.

These constraints (as it was aluded to in Section 1) indicate that using the standard Finite Element Method (FEM) would be computationally and memory expensive, as the mesh to be created would need to have a considerable number of elements in order to both model the domain and fractures. When it comes to describing surfaces and shapes, CAD systems have long been using spline entities like NURBS and other spline derivatives for their flexibility and ability to model geometries exactly. Signifact effort has been invested into implementing methods for bridging the gap between design and analysis, this is the field of Isogeometric Analysis (IGA). These advances are usefull for both analysis (when taking advantage of the IGA method) and design, as the need for creating proper shape parametrisations and develop techniques to refine meshes to be used in analysis, gives rise to new tools and algorithms that can be also employed in design. This project will implement B-Splines, NURBS and T-Splines in a effort to provide a framework for both design and analysis of volumetric spherical meshes that also support global and local mesh refinemnt. In the following sections I present the theoretical background required for the implementation of the algorithms described in Section 3.

2.2 B-splines and NURBS

2.2.1 Knot Vector

NURBS are constructed from B-Splines. We define a *knot vector* as an ordered set of increasing coordinate values for the B-Spline parametric space: $\Xi = \{ \xi_1, \xi_2, \ldots, \xi_{n+p+1} \}$, where ξ_i is the i_{th} knot, n is the number of number of basis functions and p the polynomial order. The knot vector splits the parametric space into intervals (*knot spans*) by repeating knot values in accordance to the knot's multiplicity value k. The knots can be *uniform* (evenly spaced) or *non-uniform* (unevenly spaced). A *knot vector* with $p+1$ repeated knots ($k = p+1$) in their beginning and end are called *open*. Basis functions from open knot vectors are interpolary at the edges of the parametric space interval $[\xi_1, \xi_{n+p+1}]$ and at knot values of multiplicity $k \geq p$ but not interpolary otherwise.

2.2.2 Basis Function

With a given knot vector Ξ, the set of B-spline basis functions $\{N_{i,p}\}^{n}_{i=1}$ can be defined recursively, starting with the zeroth polynomial order ($p = 0$):

$$N_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise,} \end{cases}$$

And for polynomial order $p \geq 1$:

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)$$

Note that in the case where the numerator and denominator are 0 (0/0) the fractions are set to 0 as well.

2.2.3 Curves, Surfaces & Volumes

B-Spline curves are defined in R^d and are constructed by taking a linear combination of B-Spline basis functions with their coefficients defined in R^d, referred to as control points (analogous to nodal coordinates in FEA). Given a *knot vector* $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, and n control points $\mathbf{P_i}$ in $\mathbb{R}^d$ for $i = 1, 2, ..., n$, a piecewise-polynomial B-Spline curve is defined by:

$$C(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) \boldsymbol{P_i}$$

Moving on to more dimensions; given a control net $\mathbf{P_{ij}}$ for $i = 1, ..., n$ and $j = 1, ..., m$ and *knot vector* $U = \{u_1, u_2, \ldots, u_{n+p+1}\}$ and $V = \{v_1, v_2, \ldots, v_{m+q+1}\}$, a tensor-product B-spline surface is defined:

$$S(v, u) = \sum_{i=1}^{n} \sum_{j=1}^{m} N_{i,p}(v) M_{j,q}(u) \boldsymbol{P_{ij}}$$

where $N_{i,p}(u)$ and $M_{j,q}(v)$ are univariate B-spline basis functions of order p and q, for the corresponding knot vectors U and V.

We can define solids in a similar fashion by using a control net $\mathbf{P_{ijk}}$ for $i = 1, ..., n$, $j = 1, ..., m$ and $k = 1, ..., l$ and knot vectors: U, V, H, as:

$$V(v,u,h) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} N_{i,p}(v) M_{j,q}(u) L_{k,r}(h) \boldsymbol{P_{ijk}}$$

where $\mathrm{N_{i,p}(v)}$, $\mathrm{M_{j,q}(u)}$ and $\mathrm{L_{k,r}(h)}$ are B-spline basis functions of order p, q and r respectively.

2.2.4 Rational B-splines

Although powerful, B-spline basis functions are unable to model geometries like circles, conic sections and ellipses. To circumvent this issue, we define any geometric entity in $\mathbb{R}^d$ through a projective transformation of a B-spline object in $\mathbb{R}^{d+1}$. The NURBS object is defined by the *control points* $\boldsymbol{P_i^w} = \{\mathrm{w_i x_1},\ \mathrm{w_i x_2},\ \ldots,\ \mathrm{w_i x_d},\ \mathrm{w_i}\}$ (projective *control points*) where $\mathrm{w_i}$ is the $\mathrm{i_{th}}$ *weight* and *knot vector* Ξ. The *control points* of the form $\boldsymbol{P^w}$ are referred to as homogeneous *control points* in $\mathbb{R}^{d+1}$. To derive the control points $\boldsymbol{P}$ in $\mathbb{R}^d$ we can apply relations:

$$(\boldsymbol{P_i})_j = (\boldsymbol{P_i^w})_j / w_i, \text{ for } j = 1, ..., d$$

$$w_i = (\boldsymbol{P_i^w})_{d+1}$$

where $(\boldsymbol{P_i})_j$ is the $\mathrm{j_{th}}$ component of vector $\mathbf{P_i}$ and $\mathrm{w_i}$ is the $\mathrm{i_{th}}$ *weight*.

Given the B-spline basis functions and the new *weights*, the new NURBS rational basis function is defined as:

$$R_i^p(\xi) = \frac{N_{i,p}(\xi) w_i}{\sum_{\hat{i}=1}^{n} N_{\hat{i},p}(\xi) w_{\hat{i}}}$$

Then using control points $\boldsymbol{P_i} = \{\mathrm{x_i},\ \mathrm{y_i},\ \mathrm{z_i}\}$, we can define a NURBS curve as:

$$C(\xi) = \sum_{i=1}^{n} R_i^p(\xi) \boldsymbol{P_i}$$

Following the same principles as in B-splines, tensor-product NURBS surfaces for *control net* $\boldsymbol{P_{ij}}$ with *knot vectors* U and V and volumes for *control net* $\boldsymbol{P_{ijk}}$ with *knot vectors* U, V and H are given by:

$$S(v,u) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q}(v,u) \boldsymbol{P_{ij}}$$

$$V(v,u,h) = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} R_{i,j,k}^{p,q,r}(v,u,h) \boldsymbol{P_{ijk}}$$

Whose rational basis functions have the form:

$$R_{i,j}^{p,q}(v,u) = \frac{N_{i,p}(v) M_{j,q}(u) w_{i,j}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m} N_{\hat{i},p}(v) M_{\hat{j},q}(u) w_{\hat{i},\hat{j}}}$$

$$R_{i,j,k}^{p,q,r}(v,u,h) = \frac{N_{i,p}(v) M_{j,q}(u) L_{k,r}(h) w_{i,j,k}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m}\sum_{\hat{k}=1}^{l} N_{\hat{i},p}(v) M_{\hat{j},q}(u) L_{,r}(h) w_{\hat{i},\hat{j},\hat{k}}}$$

2.3 Splines for Local Refinement

2.3.1 Anchors & index space (T-mesh)

As we mentioned in the sections above, when *open knot vectors* are used, there are $n + p + 1$ knots and n basis functions. However, knots $\{ \xi_i \}_{i=1}^{n+p+1}$ are not in direct correspondence with basis functions $\{ N_{j,p} \}_{j=1}^{n}$. It will prove beneficial to develop the means of identifying locations in the *parameter space* that are associated with specific basis functions. Those locations are called *anchors*. Let t_i be the anchor for $N_{i,p}$. Then the anchors for the basis functions $N_{i,p}$ are given by:

$$t_i = \begin{cases} \xi_{i+(p+1)/2} & \text{if p is odd} \\ \frac{1}{2}(\xi_{i+(p/2)} + \xi_{i+(p/2)+1}) & \text{if p is even.} \end{cases}$$

The concept of *anchors* becomes extremely useful when we make use of the *index space*. *Index space* is made by placing all the knots in a *knot vector* at equal spaced intervals and labeling them with their *index values*. *Index space* provides us with the means of defining unique locations for anchors that are associated with distinct basis functions. For a given one-dimensional B-spline basis function $N_{i,p}$, a unique *anchor* in *index space* s_i can be defined by:

$$s_i = i + (p+1)/2$$

When p is odd, the *anchors* have integer values, while when p is even, the *anchors* are real valued and the average of consecutive integers. *Anchors* in both parameter and *index space* can by expanded to multi-dimensional B-spline basis functions. When the *index space* of a Spline entity contains *T-junctions* (e.g. after local refinement), it is referred to as a *T-mesh* (Sederberg et al. 2004). *T-mesh* and *index space* will be used interchangeably for the rest of the report.

2.3.2 Local Knot vetors & T-Splines

A thing to note about a basis function $N_{i,p}$, is that its support is contained in the knot range $[\xi_i, \xi_{i+p+1}$ or, the knots that contribute to its definition are $\{\xi_i, \xi_{i+1}, \ldots, \xi_{i+p+1}\}$. Thus, we can make a shift from a global perspective of the *global knot vector* to a set of n *local knot vectors* $\Xi_i = \{ \xi_{i+j} \}_{j=0}^{p+1}$ for $i = 0, 1, ..., n-1$. Note that each basis function is located on an *anchor* point. Thus, through the use of the *T-mesh* of a Spline object, it is quite straight forward to build all the *local knot vectors* for all basis functions (Gu et al. 2015; Bazilevs et al. 2010; Sederberg et al. 2004).

The multivariate case requires the same number of *local knot vectors* as the number of dimensions d of the entity. With the *local knot vectors* of basis functions N^j for all given parametric directions, we can define a *blending function* B_i as:

$$B_i(\xi) = \prod_{j=1}^{d} N_i^j(\xi^j)$$

Now, for a given *blending function* B_i, its corresponding *control point* $\boldsymbol{P_i^w}$ we can define multivariate rational basis functions as:

$$R_i(\xi) = \frac{B_i(\xi) w_i}{\sum_{\hat{i}=1}^{n} B_{\hat{i}}(\xi) w_{\hat{i}}}$$

Note that in this representation n is is the number of *control points*. The spline in physical space is given by:

$$S(\xi) = \sum_{i=1}^{n} R_i(\xi) \boldsymbol{P_i}$$

Splines created in this way are referred to as *T-Splines* (Sederberg et al. 2004). This transition from global to local representations of spline entities, gives us access to local refinement as all the required information for the support of the basis and blending functions is stored in the *local knot vectors*. There are a number of refinement algorithms that makes use of *T-splines* (Sederberg et al. 2004; Bazilevs et al. 2010). In this report, we will focus on implementing *S-Splines* (Li and Sederberg 2019) for their optimal degrees of freedom and simplicity.

2.3.3 S-Splines

S-Spline entities are defined the same way as *T-Splines*, with the main difference being that the blending function is given by:

$$\hat{B}_i(\xi) = \beta_i B_i(\xi)$$

where β_i is a scalar constant that is computed by the local refinement algorithm (implemented in Section 3) and its purpose is to ensure partition of unity. An un-refined *S-Spline* is represented exactly as any *NURBS* or *T-Spline* as the scalar β_i will have the value of 1.

3 Algorithms

Creating NURBS objects is incremental in nature; in the sense that to properly define an NURBS object in $\mathbb{R}^n$. you only need n *knot vectors* and a *control net* in $\mathbb{R}^n$. Most algorithms (Piegl and Tiller 1997) progressively apply NURBS curves to form higher dimensional objects. One advantage of working with NURBS, is that once you have a curve, surface or volume, it is quite straightforward to apply transformations to it (e.g. translation, rotation, stretching) and to edit it (e.g. refine the knot vector). This added flexibility with higher dimensional structures, enables us to focus on creating a coarse meshe of desired entities that will then be refined to reach the desired final mesh; shifting the bulk of the algorithmic complexity from shape generators to editors. We can also use standardised primitives for some key generator routines, making input sanitation trivial. Efficient sampling, generating and geometric algorithms for NURBS entities do exist (Piegl and Tiller 1997), although most are limited to surfaces. In this section I will discuss and present algorithms for exapnding the set of routines to volumes. Moreover, simplified versions of object generation algorithms will be discussed, starting with a circle and leading up to and including a spherical shell.

3.1 Circles & Spheres

One way to create a sphere would be to define it parametrically through a *full circle* and a *half circle* (semi circle). The *semi circle* defines a half-circle curve in the xz plane and the full circle revolves that curve 360 degrees around the z axis, forming a spherical surface (Piegl and Tiller 1997). Thus, the primitive geometric object required to generate a spherical surface (closed or open) is a circular curve of arbitrary *sweep angle*. The NURBS book provides a number of algorithms for generating basic curves and surfaces. A7.1 MakeNurbsCircle produces a circular arc in three dimensions of arbitrary sweep angle theta. The arc is represented by:

$$C(u) = \boldsymbol{O} + r\cos u\boldsymbol{X} + r\sin u\boldsymbol{Y} \text{ for } \theta_{start} \leq u \leq \theta_{end}$$

where **O** is the coordinates of the origin, **X** and **Y** are *unit vectors* that define the plane of the curve.

The algorithm pieces together equal arcs of *sweep angle* dθ: $(min(\theta, 45) < d\theta \leq 90^{\circ}$, with knots of multiplicity $k = 2$ and weights $w = \cos(\theta)$. The result is C^1 continuous. A routine similar to **A7.1** was coded (**NURBSCircle**), with a few optimizations being implemented since we know the orientation and planes the two curves will need to enhabbit in order to properly define a sphere. A pseudocode snippet bellow outlines the changes made to **A7.1**. **NURBSCircle** uses a helper function for finding line intersection in 2 dimensions instead of 3 (*get intersect*):

```
T2 = -sin(angle)*X + cos(angle)*Y
if (X[0] == 1):
  P1 = get_intersect( P0(x, y), T0(x, y), P2(x, y), T2(x, y) )
if (X[2] == 1):
  P1 = get_intersect( P0(x, z), T0(x, z), P2(x, z), T2(x, z) )
Pw[index+1] = w1*P1
```

Figure 1: Code snippet for intersection hack.

For example if we want to create a circle in the xy plane: **X** = [1, 0, 0] and **Y** = [0, 1, 0] and if we want to create a circle in the xz plane: **X** = [0, 0, 1] = **Z** and **Y** = [1, 0, 0] = **X**. The output is given in the form of an array of the 4 dimensional homogeneous control points $\boldsymbol{P^w}$.

With this algorithm we generate a 9-point *full circle* and a 5-point *semi circle*, with *knot vectors* Uf = {0., 0., 0., 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1., 1., 1.} and Uh = {0., 0., 0., 0.5, 0.5, 1., 1., 1.}, weights wf = [1, $\sqrt{2}/2$, 1, $\sqrt{2}/2$, 1, $\sqrt{2}/2$, 1, $\sqrt{2}/2$, 1] and wh = [1, $\sqrt{2}/2$, 1, $\sqrt{2}/2$, 1] and of orders $p = q = 2$, respectively. Figure 2 shows the generated unit full circle and semi circle curves with their unweighted control points.

Now we have everything we need to generate a spherical surface. Algorithm **Sphere** is inspired by **A8.1**. The key differences are that it does no geometric calculations to find the control net but rather uses the control points from the semi and full circles that are both given as input. The input needs to be the homogeneous *control points* of two circular NURBS curves whose occupying planes are perpendicular to one another.

```
function Sphere(Pfw, Phw):
  n = length of Pfw
  m = length of Phw
  create output Pijw(n, m)
  for (i = 0; i < m; i++)
  { Pijw[0, i] = Ph[i]
    for (j = 0; j < n; j++)
    { if (i == m-1) || (i == 0) { Pijw[i, j] = Ph[i] }
      else { Pijw[i,j] = [Pf[j, x], Pf[j, y], Ph[i, z], Ph[i, w]*Pf[j, w]] }
    }
  }
  return Pijw
```

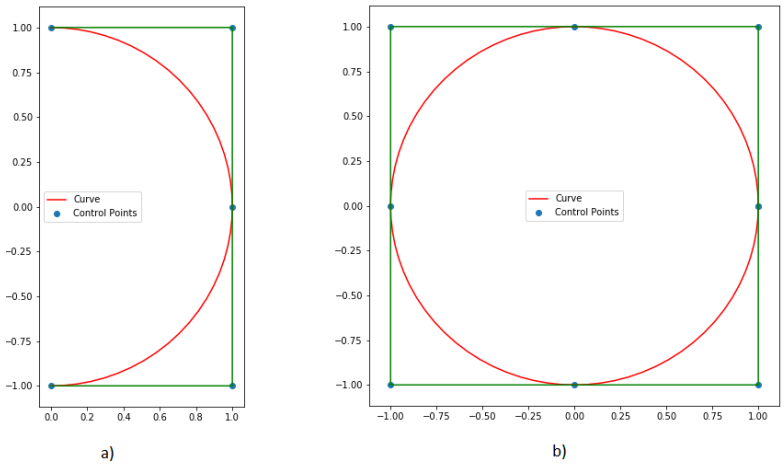

Figure 2: a) *Semi circle* control points with generated curve, b) *Full circle* control points with generated curve.

Figure 3: Pseudocode for sphere generation.

The output is the homogeneous *control net* $\boldsymbol{P_{ij}^{w}}$ which together with *knot vectors* Uf and Uh define the NURBS spherical surface. Figure 4left shows an evaluated spherical surface.

3.2 Spherical Shell

Given a NURBS spherical surface of arbitrary radius, creating a spherical volume is elementary. We need to define a *knot vector* H, *weights* wh and number of control points l (layers) for the thickness of the shell and then use the generated *control net* $\boldsymbol{P_{ij}^{w}}$ of the sphere to define a spherical shell volume *control net* $\boldsymbol{P_{ijk}^{w}}$. As input we need to provide the maximum r_{max} and minimum r_{min} radii of the shell as well as the number of *layers* to be created. For every entry in the thickness control points a new sphere will be generated; resulting in a spherical shell volume. So, for a *knot vector* H, with *weights* and l number of control points, function **sphericalShell()** bellow generates a volumetric spherical shell:

```
function sphericalShell( rmin, rmax, wh, Pijw, l):
  n, m = dimensions of Pijw
  create output Pijkw(l, n, m)
  step = (rmax - rmin)/l
  for (k = 0; k <= l; k++)
  { aspect = (rmax - k*step) / rmax
    Pijkw[k, :, :] = Pijw * aspect * wh[k]
  }
  return Pijkw
```

Figure 5: Pseudocode for spherical shell generation.

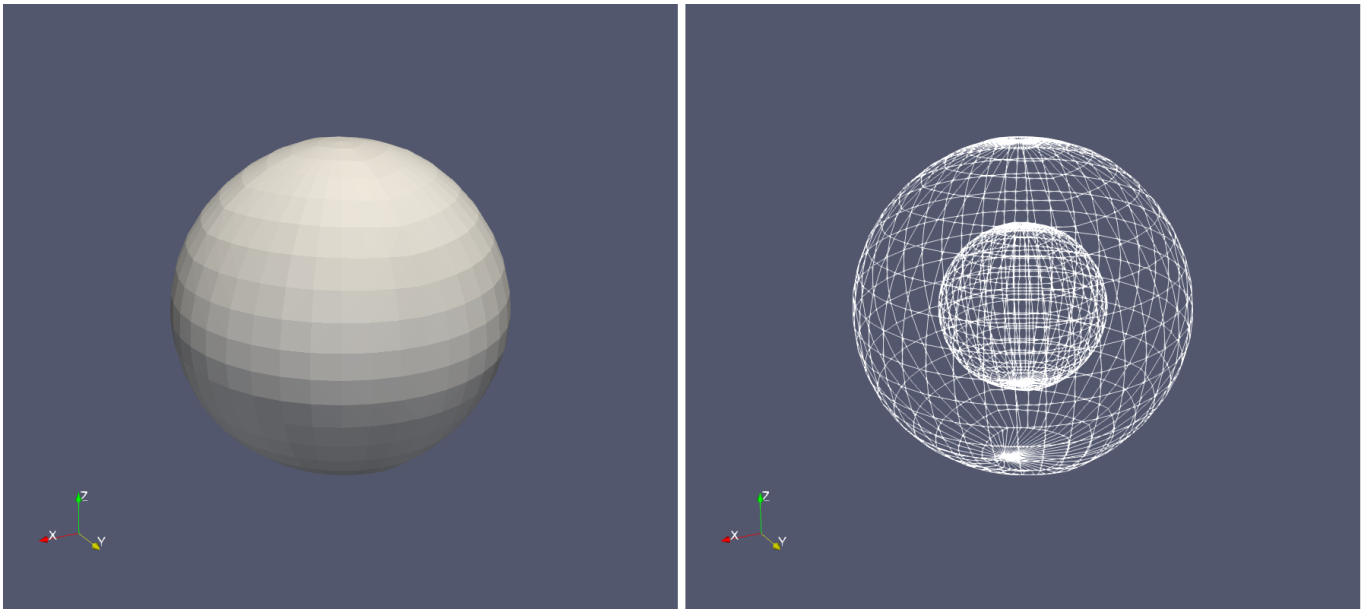

Figure 4: Left: NURBS Spherical Surface; Right: NURBS Spherical Shell Volume (in wireframe representation)

Thus, the complete NURBS structure for a single layer in a spherical volumetric shell, consists of: *control net* $\boldsymbol{P_{ijk}^{w}}$ and *knot vectors* U, V and H = [0, 0, 0.5, 1, 1] of degrees p, q, $r = 1$, respectively. Figure 4right shows an evaluated spherical shell volume.

3.2.1 Evaluate Volume

Now that we have defined the NURBS volume, we need a routine for computing it. Algorithm **VolumePoint()** expands on the algorithms from (Piegl and Tiller 1997) for curves and surfaces, and presents a version for volumes:

```
function VolumePoint(p, U, q, V, r, H, Pw, u, v, h):
  uspan = fspan(p, u, U)
  vspan = fspan(q, v, V)
  hspan = fspan(r, h, H)
  Nu = basisFun(uspan, u, p, U)
  Nv = basisFun(vspan, v, q, V)
  Nh = basisFun(hspan, h, r, H)
  create output Vw{0.0, 0.0, 0.0, 0.0}
  for (k = 0; k <= r; k++)
  { create array temp1{0.0, 0.0, 0.0, 0.0}
    for (j = 0; j <= q; j++)
    { create array temp2{0.0, 0.0, 0.0, 0.0}
      for (i = 0; i <= p; i++)
      { temp2 = temp2 + Nu[i] * Pw[wspan-r+k, vspan-q+j, uspan-p+i] }
      temp1 = temp1 + Nv[j] * temp2
    }
    Vw = Vw + Nw[k] * temp1
```

```
}
V = Vw/w
return V
```

Figure 6: Pseudocode for volume evaluation

3.2.2 Global Uniform Volume Refinement

Knot insertion is the analog to h-refinemnt in FEA. With NURBS, we are able to insert any number of knots witout affecting an entity geometrically or parametrically. If there exist a given *knot vector* $\Xi = \{ \xi_1, \xi_2, \ldots, \xi_{n+p+1} \}$, let $\hat{\xi} \in [\xi_k, \xi_{k+1}]$ be the knot to be added. We can form a the *n + 1* new basis functions recursively, with the new knot vector $\hat{\Xi} = \{ \xi_1, \ldots, \xi_k, \hat{\xi}, \xi_{k+1}, \ldots, \xi_{n+p+1} \}$. Of course: $\Xi \subset \hat{\Xi}$. New *control points* $\hat{\boldsymbol{P}}$ are formed from the original points $\boldsymbol{P}$, using:

$$\hat{\boldsymbol{P}}_{\boldsymbol{i}} = \alpha_i \boldsymbol{P}_{\boldsymbol{i}} + (1 - \alpha_i) \boldsymbol{P}_{\boldsymbol{i-1}}$$

where,

$$\alpha_i = \begin{cases} 1 & \text{for } 1 \leq i \leq k - p \\ \frac{\hat{\xi} - \xi_i}{\xi_{i+p} - \xi_i} & \text{for } k - p + 1 \leq i \leq k \\ 0 & \text{for } k + 1 \leq i \leq n + p + 2 \end{cases}$$

jewigvwegvojil;klk Knot values already in Ξ may be repeated, with reduction in continuity however. The continuity of the entity is preserved by selecting control points according to the equations above. Note that any interval knot value may not appear more than p times for the entity to not become discontinuous. The algorithms for global refinement for curves and surface can be found in Piegl and Tiller 1997, the tool expanded them for volumes.

3.3 T-Spline and Local Refinement

When knot refinement is applied to a NURBS volume, control points are being added for all other parametric directions as well. This not always ideal, since most of the times refinement will be initiated to tackle errors in a specific location on the domain (rather than a whole parametric direction). This is one limitation of NURBS that can be alleviated by transforming a *NURBS* entity into a *T-Spline* (or one of their derivatives) that support local refinement. Figure 7 makes clear the difference between local and global refinement.

3.3.1 Convert NURBS to T-Splines

Creating a *T-Spline* version of a *NURBS* object involves generating *local knot vectors* for every *achor* in the *index space* (see Section 2.3.2). The algorithm bellow generates volumetric *T-Splines* from a *NURBS* volume for one parametric direction; the same process can be repeated for the rest with the appropriate changes to variables. The input to the function are: degree, *global kont vector*, *anchor* indexes. The case for degree 1 is trivial and can be included with an simple if-statement.

```
function GenerateTSpline(p, V, Pw, anchs):
  n = length of anchs array
  initialize loc_knots
```

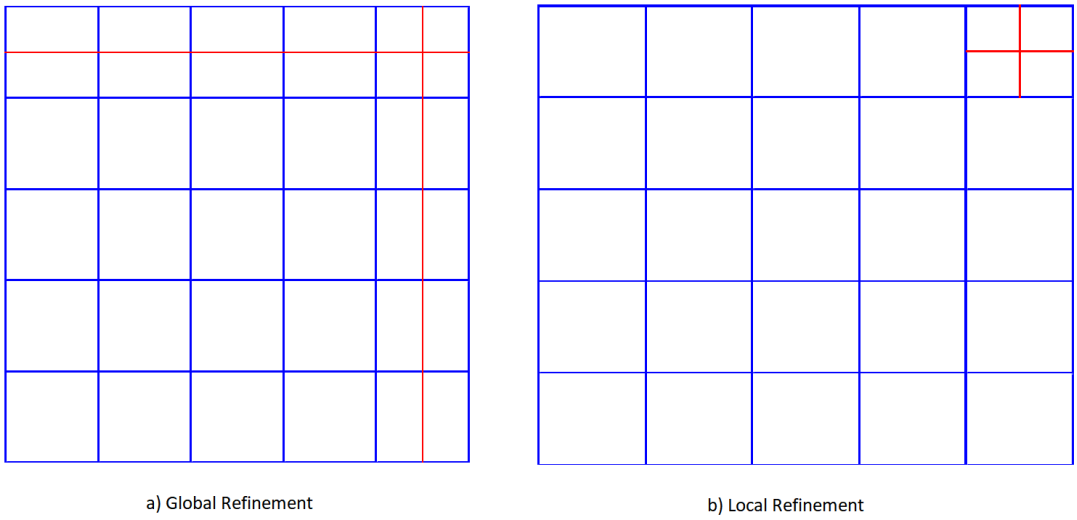

Figure 7: Index space of local and global refinement: a) NURBS *knot vector* refinement propagates to all parametric directions. b) Refinement of *local knot vectors* contains the refinement to specific locations of the domain.

```
for (i = 0; i < n; i++)
{ if (p % 2 == 0)
  { ind = floor(anchs[i]) - 1
    for (j = 0; j < p/2 + 1; j++)
    { if (ind - j <= 0) { loc_knots[i][p/2 - j] = V[0] }
      else { lock_knots[i][p/2 - j] = V[ind - j + p - 1] }
      m = anchs.size
      if (ind + j >= m) { loc_knots[i][p/2 + j + 1] = V[m] }
      else { loc_knots[i][p/2 + j + 1] = V[p + ind + j] }
    }
  } else {
     ind = anchs[i]
     loc_knots[p-1] = V[ind]
     for (j = 0; j < p-1; j++)
     { if (ind - j <= 0) { loc_knots[i][p - 2 - j] = V[0] }
       else { loc_knots[i][r - 2 - j] = V[ind - j - 1] }
       m = anchs.size
       if (ind + j + 1 >= m) { loc_knots[i][r + j] = V[m] }
       else { loc_knots[i][ind + j + 1] }
     }
   }
return loc_knots
```

Figure 8: Pseudocode for T-Spline generation.

3.3.2 Local Refinement with S-Splines

Although *T-Splines* offer a spline representation that enables local refinement, the process for accomplishing that can be quite cumbersome, both in algorithmic and time complexity. Out of all the *T-Spline* derivatives, *S-Splines* show the most promise for this task, as their version of local refinement is both easier and faster algorithmicly (Li and Sederberg 2019), while generating the least new *control points*.

The algorithm bellow implements Class 1 refinement to insert a new *control point* to a designated *index space* location (in all three parametric directions), given the knot to added and the chosen location are valid. The last input of the algorithm is the direction of refinement; only the case of even degree 2 is considered as (Li and Sederberg 2019) outlines the case for odd degrees.

```
function LocalRefine(x, y, z, knot, dir=0):
  anchors <- list of anchor indexes for all points
  Pw      <- control points
  betas   <- scalar value beta
  loc     <- list of local knot vectors

  find index: par s.t. -> x < anchors[i].x && y == anchors[i].y && z == anchors[i].z

  new beta nbet = (knot - loc[par][dir].0)/(loc[par][dir].2 - loc[par][dir].0) * betas[par]
  update betas[par] = (loc[par][dir].3 - knot)/(loc[par][dir].3 - loc[par][dir].1) * betas[par]

  new local knot = { loc[par][dir].0, loc[par][dir].1, knot, loc[par][dir].2 }
  update loc[par] = { loc[par][dir].1, knot, loc[par][dir].2, loc[par][dir].3 }

  new control point P = Pw[par]
  new set of anchors Anchs = { x, y, z }

  add all new entries before the parent control point
```

Figure 9: Pseudocode for S-Spline Class 1 refinement.

4 Implementation

Prototyping was done in Python and a proper implementation of the tool is written in C++. Functionality were included according to requirements of the domain generation pipeline. This section will discuss design decisions for the classes, validation, tests and general workflow of the tool.

4.1 Class Design

The main parts of the tool can be divided into the main categories of: Primitives, Generators, Operators and Samplers:

4.1.1 Primitives

The main primitive of the tool is the **spline** abstract class. It contains helper functions that are used by the derived objects, as well as objects and functions that are shared by all derived classes. All spline entities implemented by this tool are derived from **spline**. The choise to use an abstract class was to facilitate the definition and memory allocation of all the components of different sizes required to define and construct the different types of entities without creating overhead in the class definition. The derived classes are: **nurbsCur**, **nurbsSur**, **nurbsVol** and **tsplineVol**.

4.1.2 Generators

Generators construct splines (*NURBS* and *T-Splines*). They are implemented as the constructors of the derived classes. Note that there can be multiple constructors for each class. Figure 9 shows the relations and usage of different classes to define entities in higher dimensions.

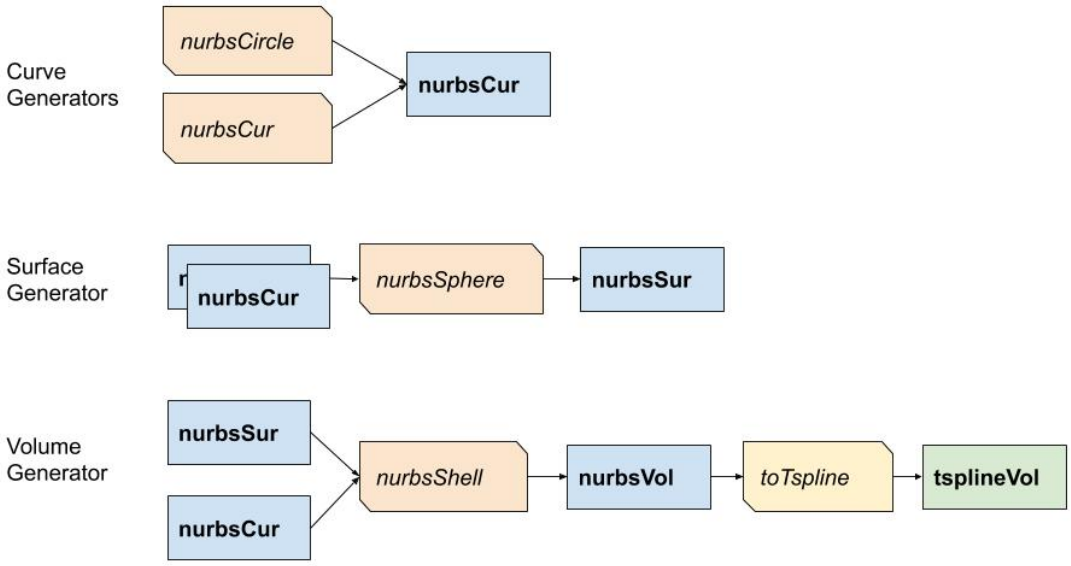

Figure 10: Relation of generators and their classes. Blue and Green are the class objects. Orange and Yellow are the class constructors (different names were used in the Figure to make the usage clear).

4.1.3 Operators & Samplers

Operators include all the geometric operations available for a given spline entity. This tool implements: refinement, evaluation, weighted projection. Samplers are essentially the tools exporters. The tool support exporting to *csv* and *vtk* formats according to the needs of the derived class. Table 1 summarizes the functionality of the derived classes.

Table 1: Functionality of the derived classes.

	nurbsCur	**nurbsSur**	**nurbsVol**	**tsplineVol**
Generator	Circles and Curves	Spheres	Spherical Shells	Shperical Shells
Refinement	-	-	Global	Local
Exporting	csv	csv	csv/vtk	csv/vtk

4.2 Tool Workflow

The code is implemented in a way to allow for incremental creation of the desired spherical shell entity and its refinement before and during the simulation runtime. Figure 10 shows a stadard workflow for generating and refinening splines.

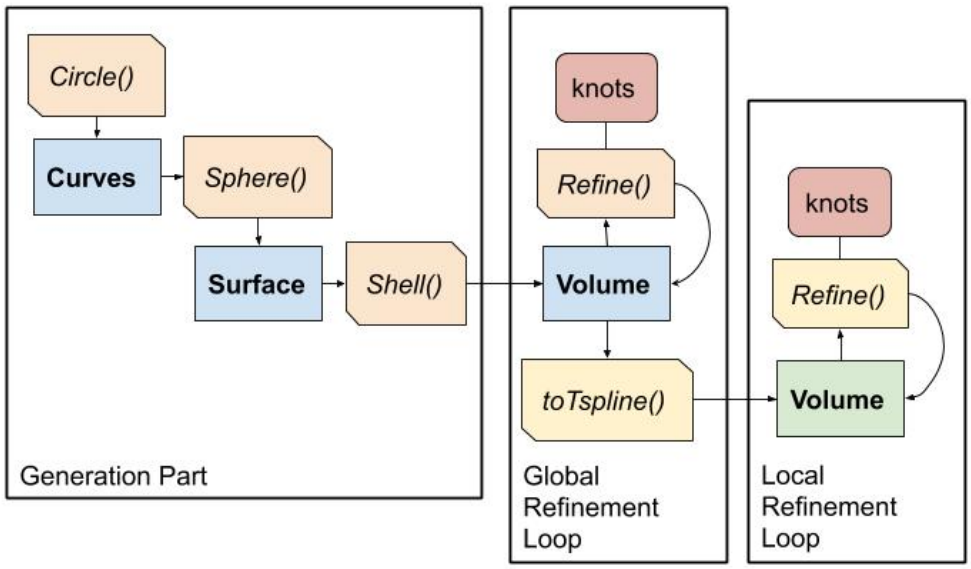

Figure 11: Implementation Workflow

4.3 Validation & Tests

Validation routines have been written for all generators, as a means for accessing the quality of the spline. The process involves evaluating a spline entity and then comparing the result with a algebraic parametric representation of the given entity. In this way, we accomplish both validation of mesh quality and also provide a base for implementing tests for the tool.

5 Evaluation & Results

5.1 Generated and Refined meshes

In the current implementation, the tool can generate *NURBS* and *T-Spline* volumes and apply refinement in the global and/or local level. The location and level or refinement is left for the user to decide. A bit of background in *NURBS* is required in order to choose the right knots to insert. The figures below show knot insertion in the up-down parametric direction.

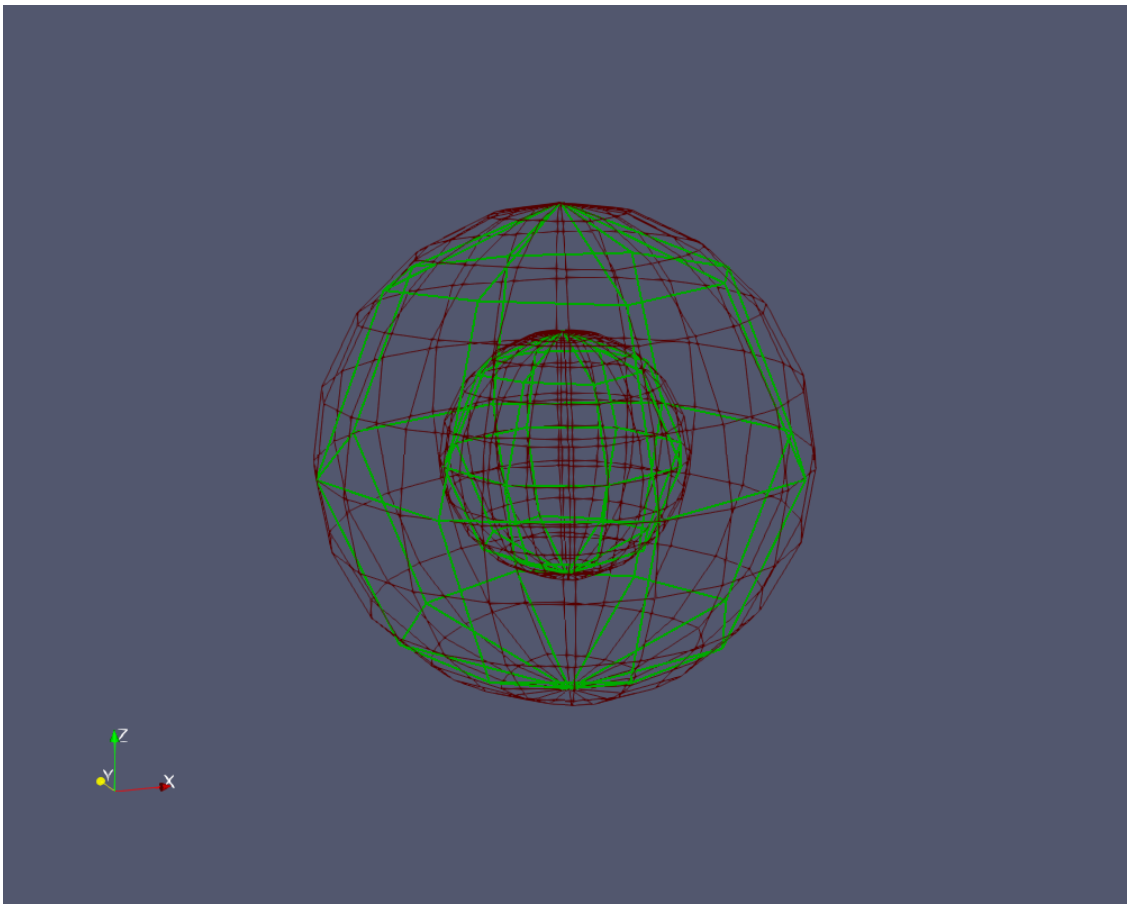

Figure 12: Green: coarse *control net* in parametric space. Red: Evaluated volume.

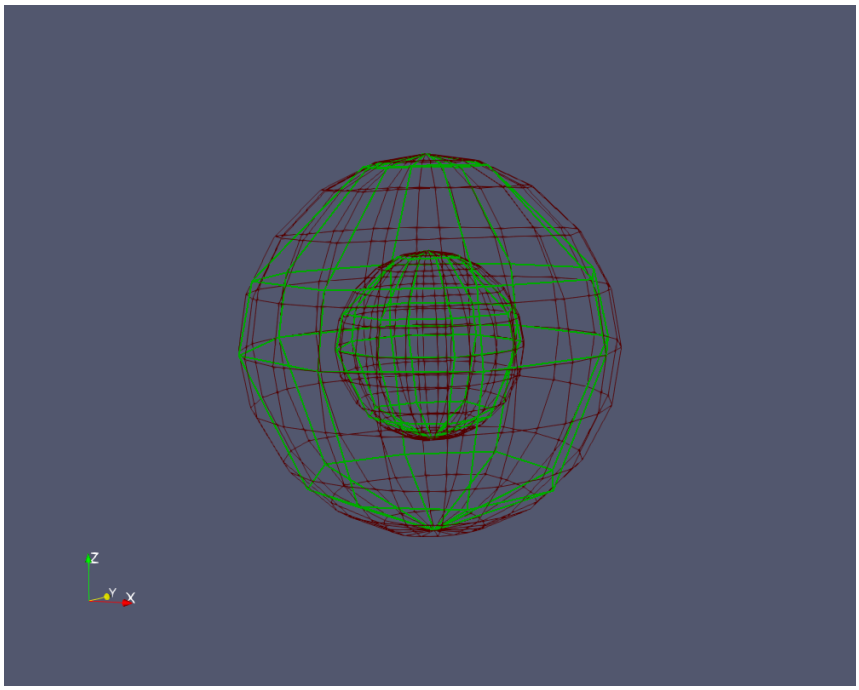

Figure 13: The same *control net* but with knot $\{0.25\}$ inserted. Notice that the distribution of the elemenets is no longer uniform.

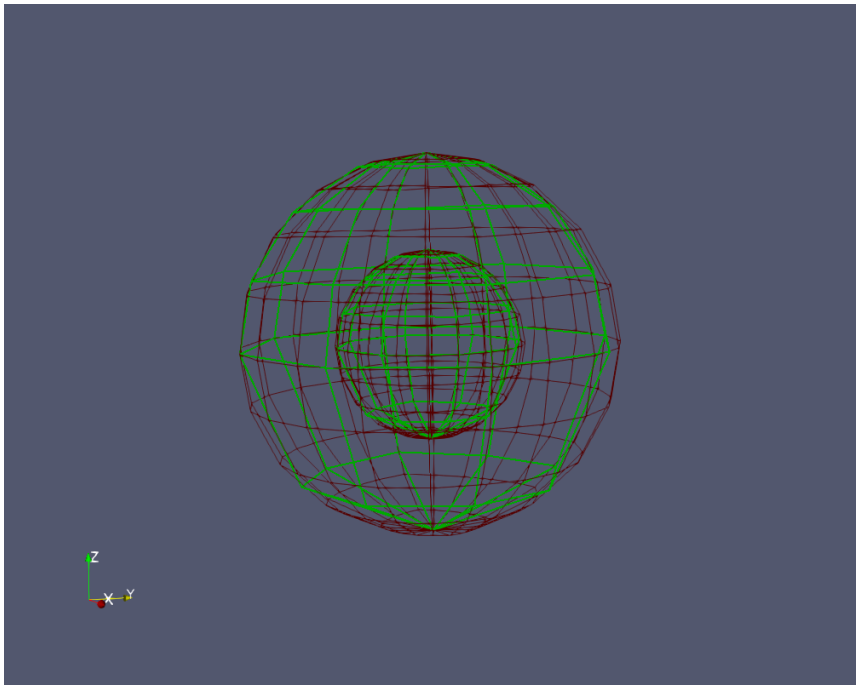

Figure 14: By adding another knot $\{0.25\}$, we complete the required mutiplicity to achieve uniform distribution of elements in the north hemisphere.

Full uniform global refinement in the up-down and left-right parametric directions are displayed in the following figures:

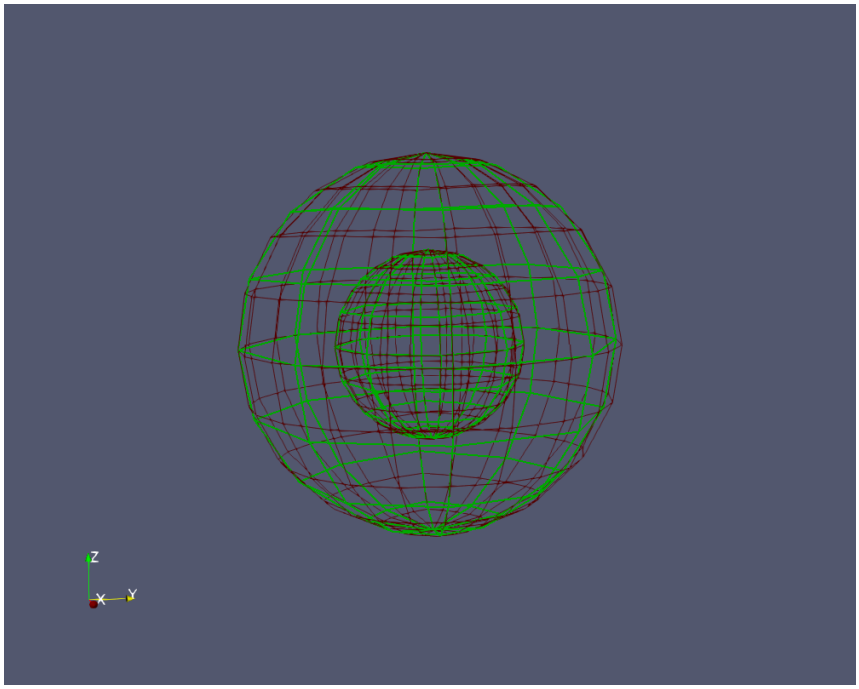

Figure 15: Uniform global refinement for lateral parametric direction.

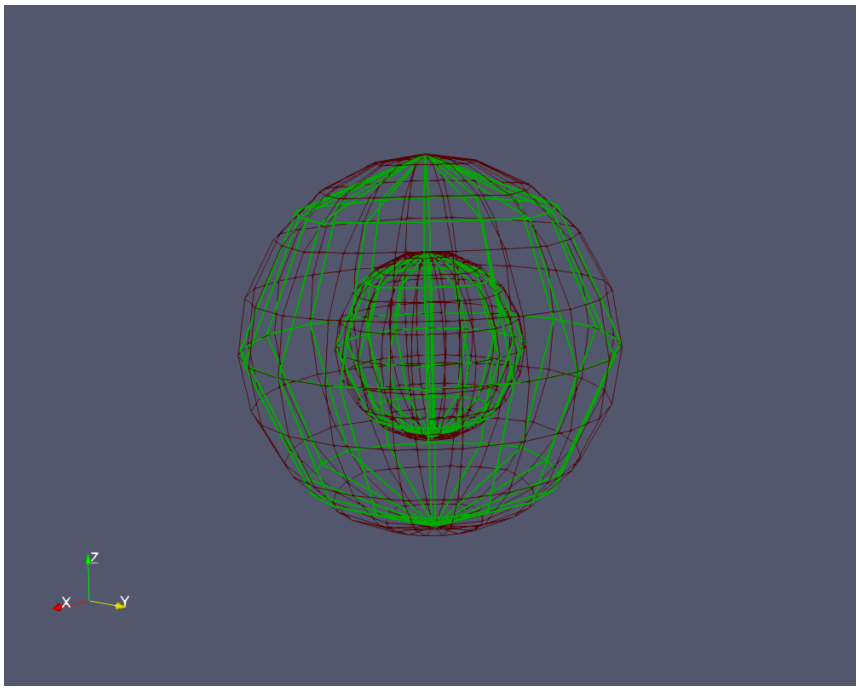

Figure 16: Uniform global refinement for longitudinal parametric direction.

For the global case, refinement in different parametric directions can be combined to increase the resolution in specific parts of the domain. As a first example, we can refine the area around the equator of the shell, and also refine one quadrant of the shell:

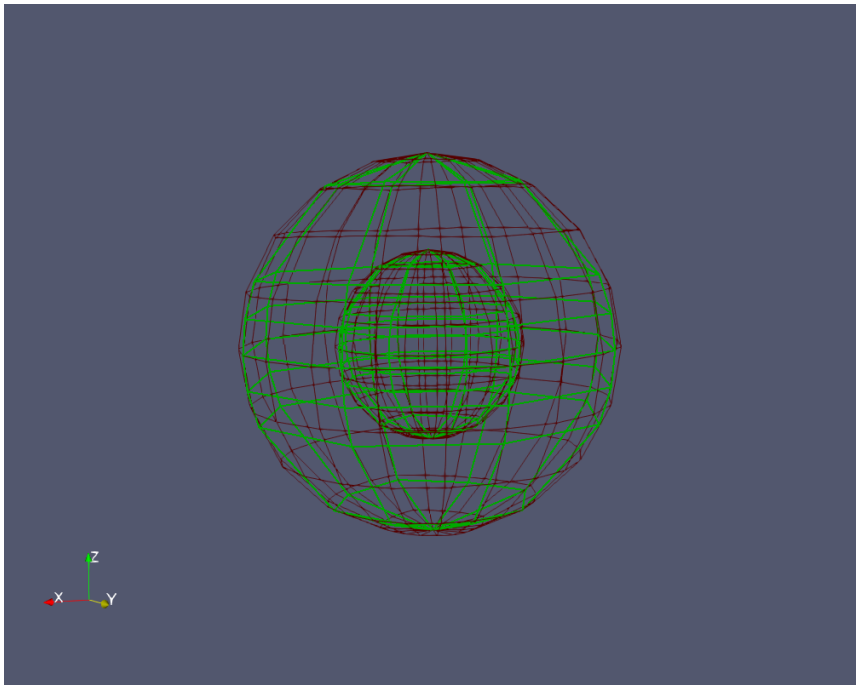

Figure 17: Global refinement around the equator.

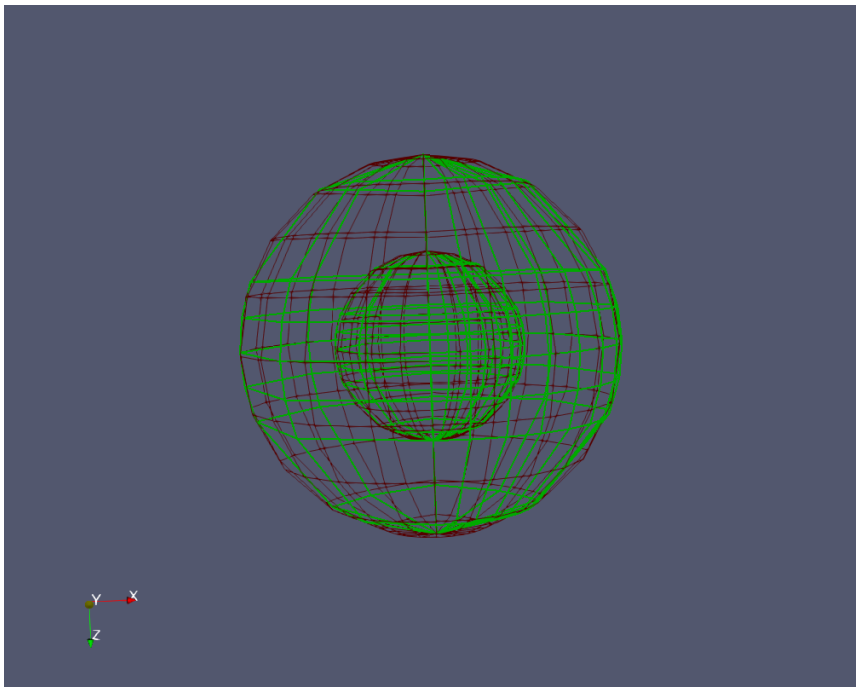

Figure 18: Refined equator together with a shell quadrant.

The combination of different global refinements become clear when we take advantage of the parametrization to limit it to a specified region. An example would be to refine one of the poles (with and without the quadrant):

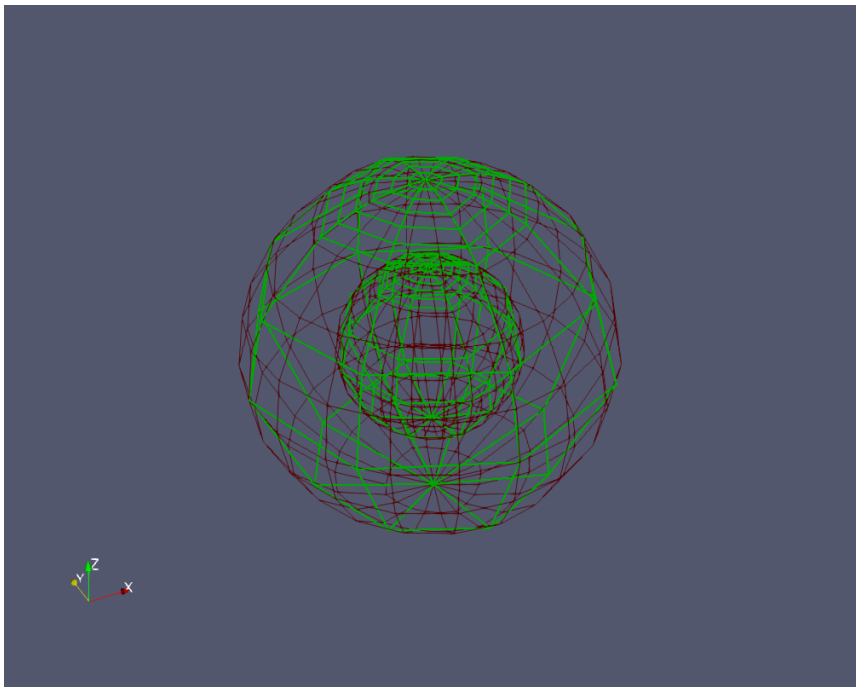

Figure 19: Global refinement around the north pole.

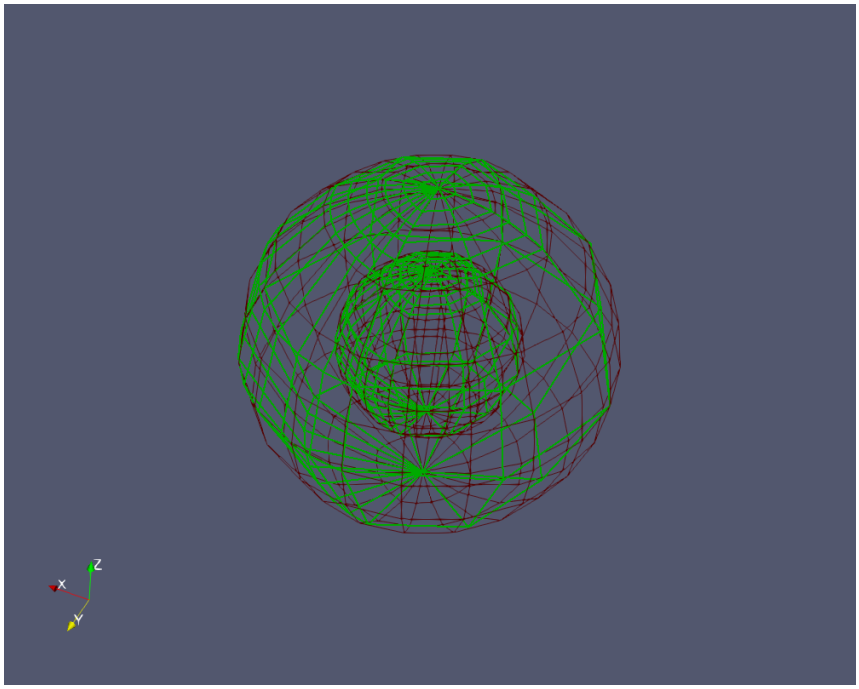

Figure 20: Refined north pole together with a shell quadrant.

When it comes to local refinement (and especially S-Splines) showcasing local refinement in the physical space gets. For the global case it is possible to evaluate the volume with the *anchor* knot values and get a parametric representation of the domain in the physical space, but with T and *S-Splines* the same process gets more complicated (as knot lines are by definition not continuous).

To display local refinement, we can make use of the *T-mesh* in the index space in order to also make clear where we add new knots and control points. First of all, let us generate a *T-mesh* for the *NURBS* shell volume before any refinement.

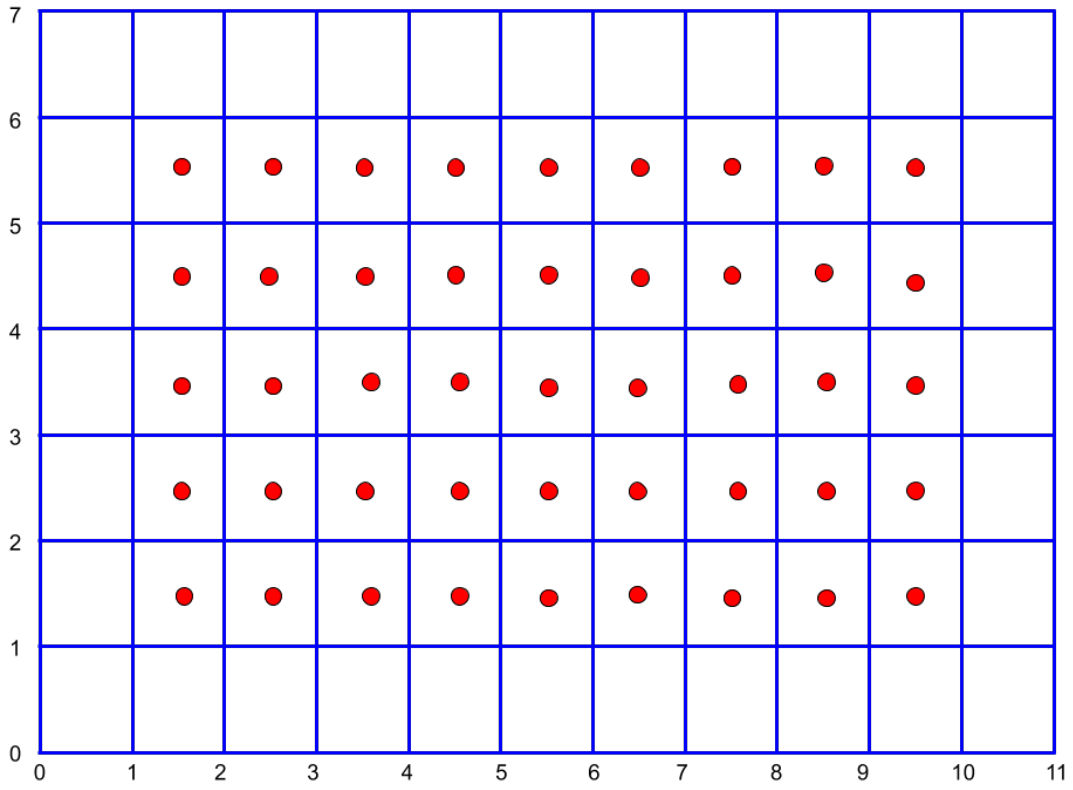

Figure 21: A cross section of a/T-mesh/ for the unrefined *NURBS* volume. Lines represent *knot values* and the red circles are the *control points* in parameter space. Recall from Section 2.3.1 that in the case of spline of degree 2, the points are located in the faces created by the intersection of knot lines.

We can add control points by local refinement and split the faces into four pieces. Let's do that for faces with index values.

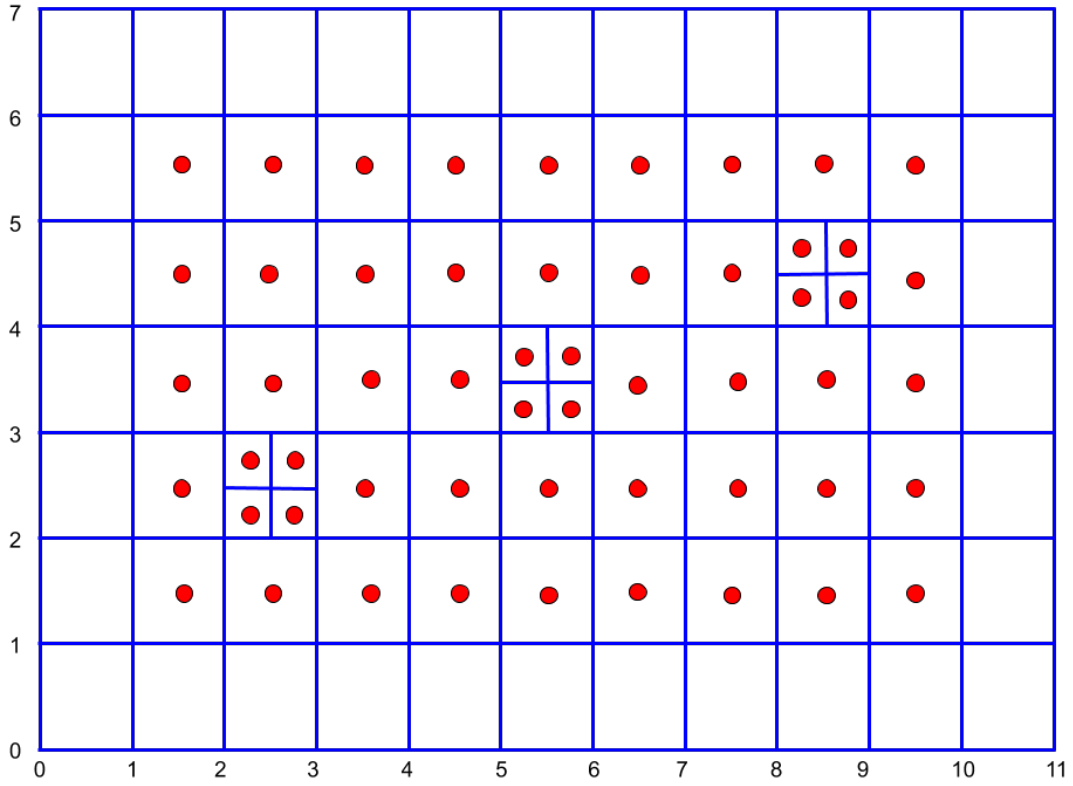

Figure 22: Faces in indexes: [2.5, 2.5], [5.5, 3.5] and [8.5, 4.5] were divided into 4 faces.

Finally, we can also apply global refinement before applying local refinement. Let's revisit an example we examined before in the refined north pole and a small quadrant of the shell. Although the index space is different for this case, we can alter choose not to place every *anchor* at equal distance from one another, so as to better show the combination of global and local refinement.

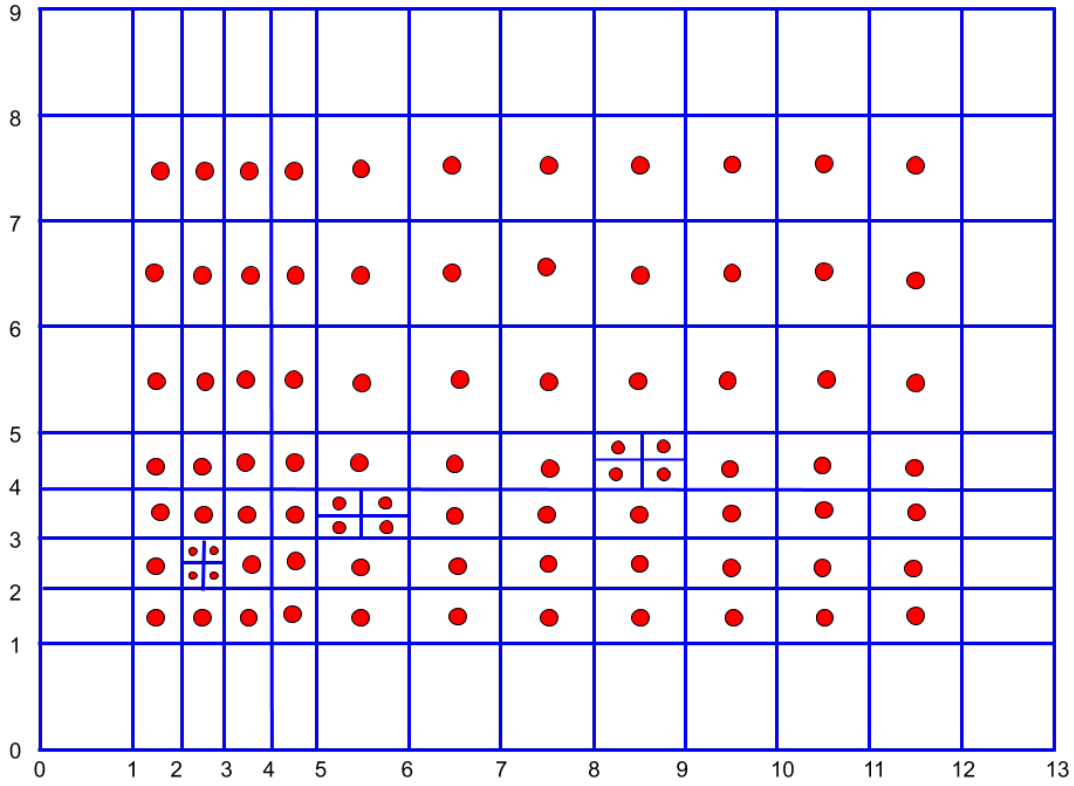

Figure 23: The same faces are chosen to be divided into fours. Notice that the numbering in the axis is different, as there are more *anchors* created by the global refinement.

5.2 Discussion

This tool provides a rigorous solution for generating volumetric spherical domains, something that is missing from most publications on *NURBS* and *T-Splines* (Piegl and Tiller 1997; Sederberg et al. 2004; Perduta and Putanowicz 2019) and even software (Bingol and Krishnamurthy 2019; Rhino). The backbone of any solid Spline implementation is established however and can thus be easily expanded (due to the clear OO design) to include other geometries and operations that can tackle a range of applications.

S-Splines although simple and powerfull, are still a fairly new derivative of *T-Splines* and thus their usage is limited to IGA and CAD where the spline basis functions are used exclusively. Nevertheless, the changes made to transition from *T* to *S-Spline* is trivial, and thus, any other *T-Spline* refinement method can also be coded into the tool.

6 Appendix A

Bellow are tables for the point data from the local refinement and local with global refinement case. Note that the parametrization in the thickness (as well as their local knot vectors) are omitted for clarity.

x	y	z	anchorx	anchory	anchorz	beta	local knot
0	0	1	1.5	1.5	1	1	[0 0 0 0.25][0 0 0 0.5]
0	0	1	2.5	1.5	1	1	[0 0 0.25 0.25][0 0 0 0.5]
0	0	1	3.5	1.5	1	1	[0 0.25 0.25 0.5][0 0 0 0.5]
0	0	1	4.5	1.5	1	1	[0.25 0.25 0.5 0.5][0 0 0 0.5]
0	0	1	5.5	1.5	1	1	[0.25 0.5 0.5 0.75][0 0 0 0.5]
0	0	1	6.5	1.5	1	1	[0.5 0.5 0.75 0.75][0 0 0 0.5]
0	0	1	7.5	1.5	1	1	[0.5 0.75 0.75 1][0 0 0 0.5]
0	0	1	8.5	1.5	1	1	[0.75 0.75 1 1][0 0 0 0.5]
0	0	1	9.5	1.5	1	1	[0.75 1 1 1][0 0 0 0.5]
0.707106	0	0.707107	1.5	2.5	1	1	[0 0 0 0.25][0 0 0.5 0.5]
0.500001	0.5	0.5	2.5	2.5	1	1	[0 0 0.25 0.25][0 0 0.5 0.5]
0	0.707107	0.707107	3	3	1	0.125	[0 0.25 0.125 0.25][0 0 0.125 0.5]
0	0.707107	0.707107	3	2.5	1	0.375	[0 0.25 0.125 0.25][0 0.125 0.5 0.5]
0	0.707107	0.707107	3.5	2.5	1	1.5	[0.25 0.125 0.25 0.5][0 0 0.5 0.5]
-0.500001	0.5	0.5	4.5	2.5	1	1	[0.25 0.25 0.5 0.5][0 0 0.5 0.5]
-0.707107	0	0.707107	5.5	2.5	1	1	[0.25 0.5 0.5 0.75][0 0 0.5 0.5]
-0.5	-0.5	0.5	6.5	2.5	1	1	[0.5 0.5 0.75 0.75][0 0 0.5 0.5]
0	-0.707107	0.707107	7.5	2.5	1	1	[0.5 0.75 0.75 1][0 0 0.5 0.5]
0.5	-0.5	0.5	8.5	2.5	1	1	[0.75 0.75 1 1][0 0 0.5 0.5]
0.707107	0	0.707107	9.5	2.5	1	1	[0.75 1 1 1][0 0 0.5 0.5]
1	0	0	1.5	3.5	1	1	[0 0 0 0.25][0 0.5 0.5 1]
0.707107	0.707107	0	2.5	3.5	1	1	[0 0 0.25 0.25][0 0.5 0.5 1]
0	1	0	3.5	3	1	0.25	[0 0.25 0.25 0.5][0 0.5 0.125 0.5]
0	1	0	3.5	3.5	1	1.75	[0 0.25 0.25 0.5][0.5 0.125 0.5 1]
-0.707107	0.707107	0	4.5	3.5	1	1	[0.25 0.25 0.5 0.5][0 0.5 0.5 1]
-1	0	0	5.5	3.5	1	1	[0.25 0.5 0.5 0.75][0 0.5 0.5 1]
-0.707106	-0.707107	0	6	4	1	0.625	[0.5 0.5 0.625 0.75][0 0.5 0.625 0.5]
-0.707106	-0.707107	0	6	3.5	1	0.375	[0.5 0.5 0.625 0.75][0.5 0.625 0.5 1]
-0.707106	-0.707107	0	6.5	3.5	1	0.5	[0.5 0.625 0.75 0.75][0 0.5 0.5 1]
0	-1	0	7.5	3.5	1	1	[0.5 0.75 0.75 1][0 0.5 0.5 1]
0.707106	-0.707107	0	8.5	3.5	1	1	[0.75 0.75 1 1][0 0.5 0.5 1]
1	0	0	9.5	3.5	1	1	[0.75 1 1 1][0 0.5 0.5 1]
0.707107	0	-0.707107	1.5	4.5	1	1	[0 0 0 0.25][0.5 0.5 1 1]
0.500001	0.5	-0.5	2.5	4.5	1	1	[0 0 0.25 0.25][0.5 0.5 1 1]
0	0.707107	-0.707107	3.5	4.5	1	1	[0 0.25 0.25 0.5][0.5 0.5 1 1]
-0.500001	0.5	-0.5	4.5	4.5	1	1	[0.25 0.25 0.5 0.5][0.5 0.5 1 1]
-0.707107	0	-0.707107	5.5	4.5	1	1	[0.25 0.5 0.5 0.75][0.5 0.5 1 1]
-0.5	-0.5	-0.5	6.5	4	1	0.25	[0.5 0.5 0.75 0.75][0.5 0.5 0.625 1]
-0.5	-0.5	-0.5	6.5	4.5	1	0.75	[0.5 0.5 0.75 0.75][0.5 0.625 1 1]
0	-0.707107	-0.707107	7.5	4.5	1	1	[0.5 0.75 0.75 1][0.5 0.5 1 1]
0.5	-0.5	-0.5	8	4	1	0.25	[0.75 0.75 0.875 1][0.5 0.5 0.75 1]

0.5	-0.5	-0.5	8.5	4	1	0.25	[0.75 0.875 1 1][0.5 0.5 0.75 1]
0.5	-0.5	-0.5	8	4.5	1	0.25	[0.75 0.75 0.875 1][0.5 0.75 1 1]
0.5	-0.5	-0.5	8.5	4.5	1	0.25	[0.75 0.875 1 1][0.5 0.75 1 1]
0.707107	0	-0.707107	9.5	4.5	1	1	[0.75 1 1 1][0.5 0.5 1 1]
0	0	-1	1.5	5.5	1	1	[0 0 0 0.25][0.5 1 1 1]
0	0	-1	2.5	5.5	1	1	[0 0 0.25 0.25][0.5 1 1 1]
0	0	-1	3.5	5.5	1	1	[0 0.25 0.25 0.5][0.5 1 1 1]
0	0	-1	4.5	5.5	1	1	[0.25 0.25 0.5 0.5][0.5 1 1 1]
0	0	-1	5.5	5.5	1	1	[0.25 0.5 0.5 0.75][0.5 1 1 1]
0	0	-1	6.5	5.5	1	1	[0.5 0.5 0.75 0.75][0.5 1 1 1]
0	0	-1	7.5	5.5	1	1	[0.5 0.75 0.75 1][0.5 1 1 1]
0	0	-1	8.5	5.5	1	1	[0.75 0.75 1 1][0.5 1 1 1]
0	0	-1	9.5	5.5	1	1	[0.75 1 1 1][0.5 1 1 1]

And next are the set of points for combination of local with global refinement.

x	y	z	anchorx	anchory	anchorz	beta	local knot
0	0	1	1.5	1.5	1	1	[0 0 0 0.125][0 0 0 0.25]
0	0	1	2.5	1.5	1	1	[0 0 0.125 0.125][0 0 0 0.25]
0	0	1	3.5	1.5	1	1	[0 0.125 0.125 0.25][0 0 0 0.25]
0	0	1	4.5	1.5	1	1	[0.125 0.125 0.25 0.25][0 0 0 0.25]
0	0	1	5.5	1.5	1	1	[0.125 0.25 0.25 0.5][0 0 0 0.25]
0	0	1	6.5	1.5	1	1	[0.25 0.25 0.5 0.5][0 0 0 0.25]
0	0	1	7.5	1.5	1	1	[0.25 0.5 0.5 0.75][0 0 0 0.25]
0	0	1	8.5	1.5	1	1	[0.5 0.5 0.75 0.75][0 0 0 0.25]
0	0	1	9.5	1.5	1	1	[0.5 0.75 0.75 1][0 0 0 0.25]
0	0	1	10.5	1.5	1	1	[0.75 0.75 1 1][0 0 0 0.25]
0	0	1	11.5	1.5	1	1	[0.75 1 1 1][0 0 0 0.25]
0.353553	0	0.853553	1.5	2.5	1	1	[0 0 0 0.125][0 0 0.25 0.25]
0.301777	0.125	0.801777	2.5	2.5	1	1	[0 0 0.125 0.125][0 0 0.25 0.25]
0.213388	0.213388	0.801777	3	3	1	0.125	[0 0.125 0.0625 0.125][0 0 0.0625 0.25
0.213388	0.213388	0.801777	3	2.5	1	0.375	[0 0.125 0.0625 0.125][0 0.0625 0.25 0.
0.213388	0.213388	0.801777	3.5	2.5	1	1.5	[0.125 0.0625 0.125 0.25][0 0 0.25 0.25
0.125	0.301777	0.801777	4.5	2.5	1	1	[0.125 0.125 0.25 0.25][0 0 0.25 0.25]
0	0.353553	0.853553	5.5	2.5	1	1	[0.125 0.25 0.25 0.5][0 0 0.25 0.25]
-0.25	0.25	0.75	6.5	2.5	1	1	[0.25 0.25 0.5 0.5][0 0 0.25 0.25]
-0.353553	0	0.853553	7.5	2.5	1	1	[0.25 0.5 0.5 0.75][0 0 0.25 0.25]
-0.25	-0.25	0.75	8.5	2.5	1	1	[0.5 0.5 0.75 0.75][0 0 0.25 0.25]
0	-0.353553	0.853553	9.5	2.5	1	1	[0.5 0.75 0.75 1][0 0 0.25 0.25]
0.25	-0.25	0.75	10.5	2.5	1	1	[0.75 0.75 1 1][0 0 0.25 0.25]
0.353553	0	0.853553	11.5	2.5	1	1	[0.75 1 1 1][0 0 0.25 0.25]
0.603553	0	0.603553	1.5	3.5	1	1	[0 0 0 0.125][0 0.25 0.25 0.5]
0.515165	0.213388	0.551777	2.5	3.5	1	1	[0 0 0.125 0.125][0 0.25 0.25 0.5]
0.364277	0.364277	0.551777	3.5	3	1	0.25	[0 0.125 0.125 0.25][0 0.25 0.0625 0.25
0.364277	0.364277	0.551777	3.5	3.5	1	1.75	[0 0.125 0.125 0.25][0.25 0.0625 0.25 0
0.213389	0.515165	0.551777	4.5	3.5	1	1	[0.125 0.125 0.25 0.25][0 0.25 0.25 0.5
0	0.603553	0.603553	5.5	3.5	1	1	[0.125 0.25 0.25 0.5][0 0.25 0.25 0.5]
-0.426777	0.426777	0.5	6	4	1	0.75	[0.25 0.25 0.375 0.5][0 0.25 0.375 0.25

-0.426777	0.426777	0.5	6	3.5	1	0.25	[0.25 0.25 0.375 0.5][0.25 0.375 0.25 0
-0.426777	0.426777	0.5	6.5	3.5	1	0.5	[0.25 0.375 0.5 0.5][0 0.25 0.25 0.5]
-0.603553	0	0.603553	7.5	3.5	1	1	[0.25 0.5 0.5 0.75][0 0.25 0.25 0.5]
-0.426776	-0.426777	0.5	8.5	3.5	1	1	[0.5 0.5 0.75 0.75][0 0.25 0.25 0.5]
0	-0.603553	0.603553	9.5	3.5	1	1	[0.5 0.75 0.75 1][0 0.25 0.25 0.5]
0.426776	-0.426777	0.5	10.5	3.5	1	1	[0.75 0.75 1 1][0 0.25 0.25 0.5]
0.603553	0	0.603553	11.5	3.5	1	1	[0.75 1 1 1][0 0.25 0.25 0.5]
0.853553	0	0.353553	1.5	4.5	1	1	[0 0 0 0.125][0.25 0.25 0.5 0.5]
0.728554	0.301777	0.301777	2.5	4.5	1	1	[0 0 0.125 0.125][0.25 0.25 0.5 0.5]
0.515165	0.515165	0.301777	3.5	4.5	1	1	[0 0.125 0.125 0.25][0.25 0.25 0.5 0.5]
0.301777	0.728553	0.301777	4.5	4.5	1	1	[0.125 0.125 0.25 0.25][0.25 0.25 0.5 0
0	0.853553	0.353553	5.5	4.5	1	1	[0.125 0.25 0.25 0.5][0.25 0.25 0.5 0.5
-0.603554	0.603553	0.25	6.5	4	1	0.5	[0.25 0.25 0.5 0.5][0.25 0.25 0.375 0.5
-0.603554	0.603553	0.25	6.5	4.5	1	0.5	[0.25 0.25 0.5 0.5][0.25 0.375 0.5 0.5]
-0.853553	0	0.353553	7.5	4.5	1	1	[0.25 0.5 0.5 0.75][0.25 0.25 0.5 0.5]
-0.603553	-0.603553	0.25	8	4	1	0.25	[0.5 0.5 0.625 0.75][0.25 0.25 0.375 0.5
-0.603553	-0.603553	0.25	8.5	4	1	0.25	[0.5 0.625 0.75 0.75][0.25 0.25 0.375 0
-0.603553	-0.603553	0.25	8	4.5	1	0.25	[0.5 0.5 0.625 0.75][0.25 0.375 0.5 0.5
-0.603553	-0.603553	0.25	8.5	4.5	1	0.25	[0.5 0.625 0.75 0.75][0.25 0.375 0.5 0.5
0	-0.853553	0.353553	9.5	4.5	1	1	[0.5 0.75 0.75 1][0.25 0.25 0.5 0.5]
0.603553	-0.603553	0.25	10.5	4.5	1	1	[0.75 0.75 1 1][0.25 0.25 0.5 0.5]
0.853553	0	0.353553	11.5	4.5	1	1	[0.75 1 1 1][0.25 0.25 0.5 0.5]
1	0	0	1.5	5.5	1	1	[0 0 0 0.125][0.25 0.5 0.5 1]
0.853554	0.353553	0	2.5	5.5	1	1	[0 0 0.125 0.125][0.25 0.5 0.5 1]
0.603554	0.603553	0	3.5	5.5	1	1	[0 0.125 0.125 0.25][0.25 0.5 0.5 1]
0.353554	0.853553	0	4.5	5.5	1	1	[0.125 0.125 0.25 0.25][0.25 0.5 0.5 1]
0	1	0	5.5	5.5	1	1	[0.125 0.25 0.25 0.5][0.25 0.5 0.5 1]
-0.707107	0.707107	0	6.5	5.5	1	1	[0.25 0.25 0.5 0.5][0.25 0.5 0.5 1]
-1	0	0	7.5	5.5	1	1	[0.25 0.5 0.5 0.75][0.25 0.5 0.5 1]
-0.707106	-0.707107	0	8.5	5.5	1	1	[0.5 0.5 0.75 0.75][0.25 0.5 0.5 1]
0	-1	0	9.5	5.5	1	1	[0.5 0.75 0.75 1][0.25 0.5 0.5 1]
0.707106	-0.707107	0	10.5	5.5	1	1	[0.75 0.75 1 1][0.25 0.5 0.5 1]
1	0	0	11.5	5.5	1	1	[0.75 1 1 1][0.25 0.5 0.5 1]
0.707107	0	-0.707107	1.5	6.5	1	1	[0 0 0 0.125][0.5 0.5 1 1]
0.603554	0.25	-0.603553	2.5	6.5	1	1	[0 0 0.125 0.125][0.5 0.5 1 1]
0.426777	0.426777	-0.603553	3.5	6.5	1	1	[0 0.125 0.125 0.25][0.5 0.5 1 1]
0.25	0.603553	-0.603553	4.5	6.5	1	1	[0.125 0.125 0.25 0.25][0.5 0.5 1 1]
0	0.707107	-0.707107	5.5	6.5	1	1	[0.125 0.25 0.25 0.5][0.5 0.5 1 1]
-0.500001	0.5	-0.5	6.5	6.5	1	1	[0.25 0.25 0.5 0.5][0.5 0.5 1 1]
-0.707107	0	-0.707107	7.5	6.5	1	1	[0.25 0.5 0.5 0.75][0.5 0.5 1 1]
-0.5	-0.5	-0.5	8.5	6.5	1	1	[0.5 0.5 0.75 0.75][0.5 0.5 1 1]
0	-0.707107	-0.707107	9.5	6.5	1	1	[0.5 0.75 0.75 1][0.5 0.5 1 1]
0.5	-0.5	-0.5	10.5	6.5	1	1	[0.75 0.75 1 1][0.5 0.5 1 1]
0.707107	0	-0.707107	11.5	6.5	1	1	[0.75 1 1 1][0.5 0.5 1 1]
0	0	-1	1.5	7.5	1	1	[0 0 0 0.125][0.5 1 1 1]
0	0	-1	2.5	7.5	1	1	[0 0 0.125 0.125][0.5 1 1 1]
0	0	-1	3.5	7.5	1	1	[0 0.125 0.125 0.25][0.5 1 1 1]
0	0	-1	4.5	7.5	1	1	[0.125 0.125 0.25 0.25][0.5 1 1 1]

0	0	-1	5.5	7.5	1	1	[0.125 0.25 0.25 0.5][0.5 1 1 1]
0	0	-1	6.5	7.5	1	1	[0.25 0.25 0.5 0.5][0.5 1 1 1]
0	0	-1	7.5	7.5	1	1	[0.25 0.5 0.5 0.75][0.5 1 1 1]
0	0	-1	8.5	7.5	1	1	[0.5 0.5 0.75 0.75][0.5 1 1 1]
0	0	-1	9.5	7.5	1	1	[0.5 0.75 0.75 1][0.5 1 1 1]
0	0	-1	10.5	7.5	1	1	[0.75 0.75 1 1][0.5 1 1 1]
0	0	-1	11.5	7.5	1	1	[0.75 1 1 1][0.5 1 1 1]

References

Aydin, Atilla (2006). "Failure modes of the lineaments on Jupiter's moon, Europa: Implications for the evolution of its icy crust". In: *Journal of Structural Geology* 28.12, pp. 2222–2236. ISSN: 01918141. DOI: 10.1016/j.jsg.2006.08.003.

Bazilevs, Y. et al. (2010). "Isogeometric analysis using T-splines". In: *Computer Methods in Applied Mechanics and Engineering* 199.5-8, pp. 229–263. ISSN: 00457825. DOI: 10.1016/j.cma.2009.02.036. URL: http://dx.doi.org/10.1016/j.cma.2009.02.036.

Billings, Sandra E. and Simon A. Kattenhorn (2005). "The great thickness debate: Ice shell thickness models for Europa and comparisons with estimates based on flexure at ridges". In: *Icarus* 177.2, pp. 397–412. ISSN: 00191035. DOI: 10.1016/j.icarus.2005.03.013.

Bingol, Onur Rauf and Adarsh Krishnamurthy (2019). "NURBS-Python: An open-source object-oriented NURBS modeling framework in Python". In: *SoftwareX* 9, pp. 85–94. ISSN: 23527110. DOI: 10.1016/j.softx.2018.12.005. URL: https://doi.org/10.1016/j.softx.2018.12.005.

Bray, Veronica J. et al. (2014). "Hydrocode simulation of ganymede and europa cratering trends - how thick is europa's crust?" In: *Icarus* 231, pp. 394–406. ISSN: 00191035. DOI: 10.1016/j.icarus.2013.12.009. URL: http://dx.doi.org/10.1016/j.icarus.2013.12.009.

Craft, Kathleen L. et al. (2016). "Fracturing and flow: Investigations on the formation of shallow water sills on Europa". In: *Icarus* 274, pp. 297–313. ISSN: 10902643. DOI: 10.1016/j.icarus.2016.01.023.

Dombard, Andrew J. et al. (2013). "Flanking fractures and the formation of double ridges on Europa". In: *Icarus* 223.1, pp. 74–81. ISSN: 00191035. DOI: 10.1016/j.icarus.2012.11.021. URL: http://dx.doi.org/10.1016/j.icarus.2012.11.021.

Figueredo, Patricio H. and Ronald Greeley (2004). "Resurfacing history of Europa from pole-to-pole geological mapping". In: *Icarus* 167.2, pp. 287–312. ISSN: 00191035. DOI: 10.1016/j.icarus.2003.09.016.

Goldreich, Peter M. and J. L. Mitchell (2010). "Elastic ice shells of synchronous moons: Implications for cracks on Europa and non-synchronous rotation of Titan". In: *Icarus* 209.2, pp. 631–638. ISSN: 00191035. DOI: 10.1016/j.icarus.2010.04.013. URL: http://dx.doi.org/10.1016/j.icarus.2010.04.013.

Golombek, M. P. and W. B. Banerdt (1990). "Constraints on the subsurface structure of Europa". In: *Icarus* 83.2, pp. 441–452. ISSN: 10902643. DOI: 10.1016/0019-1035(90)90078-N.

Greeley, Ronald et al. (2000). "Geologic mapping of Europa". In: *Journal of Geophysical Research E: Planets* 105.E9, pp. 22559–22578. ISSN: 01480227. DOI: 10.1029/1999JE001173.

Gu, Jinliang et al. (2015). "An isogeometric BEM using PB-spline for 3-D linear elasticity problem". In: *Engineering Analysis with Boundary Elements* 56, pp. 154–161. ISSN: 09557997. DOI: 10.1016/j.enganabound.2015.02.013. URL: http://dx.doi.org/10.1016/j.enganabound.2015.02.013.

Han, Lijie and Adam P. Showman (2010). "Coupled convection and tidal dissipation in Europa's ice shell". In: *Icarus* 207.2, pp. 834–844. ISSN: 00191035. DOI: 10.1016/j.icarus.2009.12.028. URL: http://dx.doi.org/10.1016/j.icarus.2009.12.028.

Helfenstein, Paul and E. M. Parmentier (1985). "Patterns of fracture and tidal stresses due to nonsynchronous rotation: Implications for fracturing on Europa". In: *Icarus* 61.2, pp. 175–184. ISSN: 10902643. DOI: 10.1016/0019-1035(85)90099-5.

Hughes, T. J.R., J. A. Cottrell, and Y. Bazilevs (2005). "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement". In: *Computer Methods in Applied Mechanics and Engineering* 194.39-41, pp. 4135–4195. ISSN: 00457825. DOI: 10.1016/j.cma.2004.10.008.

Johnston, Stephanie A. and Laurent G.J. Montési (2014). "Formation of ridges on Europa above crystallizing water bodies inside the ice shell". In: *Icarus* 237, pp. 190–201. ISSN: 10902643. DOI: 10.1016/j.icarus.2014.04.026. URL: http://dx.doi.org/10.1016/j.icarus.2014.04.026.

Kattenhorn, Simon A. (2002). "Nonsynchronous rotation evidence and fracture history in the bright plains region, Europa". In: *Icarus* 157.2, pp. 490–506. ISSN: 00191035. DOI: 10.1006/icar.2002.6825.

Kattenhorn, Simon A. and Scott T. Marshall (2006). "Fault-induced perturbed stress fields and associated tensile and compressive deformation at fault tips in the ice shell of Europa: implications for fault mechanics". In: *Journal of Structural Geology* 28.12, pp. 2204–2221. ISSN: 01918141. DOI: 10.1016/j.jsg.2005.11.010.

Leonard, E. J., R. T. Pappalardo, and A. Yin (2018). "Analysis of very-high-resolution Galileo images and implications for resurfacing mechanisms on Europa". In: *Icarus* 312, pp. 100–120. ISSN: 10902643. DOI: 10.1016/j.icarus.2018.04.016. URL: https://doi.org/10.1016/j.icarus.2018.04.016.

Leonard, E. J., A. Yin, and R. T. Pappalardo (2020). "Ridged plains on Europa reveal a compressive past". In: *Icarus* 343.January, p. 113709. ISSN: 10902643. DOI: 10.1016/j.icarus.2020.113709. URL: https://doi.org/10.1016/j.icarus.2020.113709.

Li, Xin and Thomas W. Sederberg (2019). "S-splines: A simple surface solution for IGA and CAD". In: *Computer Methods in Applied Mechanics and Engineering* 350, pp. 664–678. ISSN: 00457825. DOI: 10.1016/j.cma.2019.03.035. URL: https://doi.org/10.1016/j.cma.2019.03.035.

Marshall, Scott T. and Simon A. Kattenhorn (2005). "A revised model for cycloid growth mechanics on Europa: Evidence from surface morphologies and geometries". In: *Icarus* 177.2, pp. 341–366. ISSN: 00191035. DOI: 10.1016/j.icarus.2005.02.022.

Nimmo, F. and R. T. Pappalardo (2016). *Ocean worlds in the outer solar system*. DOI: 10.1002/2016JE005081.

Perduta, Anna and Roman Putanowicz (2019). "Tools and techniques for building models for isogeometric analysis". In: *Advances in Engineering Software* 127.May 2018, pp. 70–81. ISSN: 18735339. DOI: 10.1016/j.advengsoft.2018.10.008. URL: https://doi.org/10.1016/j.advengsoft.2018.10.008.

Piegl, Les and Wayne Tiller (1997). *Monographs in Visual Communication*. ISBN: 9783540615453.

Preblich, Brandon et al. (2007). "Tidally driven strike-slip displacement on Europa: Viscoelastic modeling". In: *Planetary and Space Science* 55.10, pp. 1225–1245. ISSN: 00320633. DOI: 10.1016/j.pss.2007.01.018.

Sederberg, Thomas W. et al. (2004). "T-spline simplification and local refinement". In: *ACM Transactions on Graphics* 23.3, pp. 276–283. ISSN: 07300301. DOI: 10.1145/1015706.1015715.

Skjetne, Helle L. et al. (2020). "Morphological comparison of blocks in chaos terrains on Pluto, Europa, and Mars". In: *Icarus*, p. 113866. ISSN: 00191035. DOI: 10.1016/j.icarus.2020.113866. URL: https://doi.org/10.1016/j.icarus.2020.113866.

Vance, Steven D. et al. (2018). "Geophysical Investigations of Habitability in Ice-Covered Ocean Worlds". In: *Journal of Geophysical Research: Planets* 123.1, pp. 180–205. ISSN: 21699100. DOI: 10.1002/2017JE005341.

Weller, Matthew B., Lukas Fuchs, et al. (2019). "Convection in Thin Shells of Icy Satellites: Effects of Latitudinal Surface Temperature Variations". In: *Journal of Geophysical Research: Planets* 124.8, pp. 2029–2053. ISSN: 21699100. DOI: 10.1029/2018JE005799.

Weller, Matthew B. and Adrian Lenardic (2018). "On the evolution of terrestrial planets: Bistability, stochastic effects, and the non-uniqueness of tectonic states". In: *Geoscience Frontiers* 9.1, pp. 91–102. ISSN: 16749871. DOI: 10.1016/j.gsf.2017.03.001. URL: https://doi.org/10.1016/j.gsf.2017.03.001.