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

Independent Research Project
Project Plan

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 Introduction

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 $2989 kg/m^3$ (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.

1.2 Project Description

The goal of this project is to model fracture formation and growth of Europa's entire ice shell. This task is divided into two sub-projects: the first investigating the stresses that spawn and drive the expansion of fracture units and the second exploring the means to model fracture growth on spherical volumetric shell of such scale and resolution; the end result will be be a simulation that resembles the surface lineaments present on Europa.The former part of the project will be tackled by another ACSE student, while the latter will be my undertaking. The final goal of this endeavour, would be to have a set of tools for generating and refining volumetric spherical meshes. Code will be written in Python and C++.

2 Domain Generation

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 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 adptive 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 Vectori

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 & Solids

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 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 $N_{i,p}(v)$, $M_{j,q}(u)$ and $L_{k,r}(h)$ 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 refered 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:

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

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

2.2.5 Global refinement - Knot insertion

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,

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

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. Algorithms will be presented in Section 3 for implementing knot insertion in Volumes.

3 Implementing NURBS routines

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 (NURBS BOOK) 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 (NURBS BOOK), 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 (Nurbs book). 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: $(min(\theta, 45) < d\theta \leq 90°$, 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
```

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. The figures below show generated unit full circle and semi circle curves with their unweighted control points.

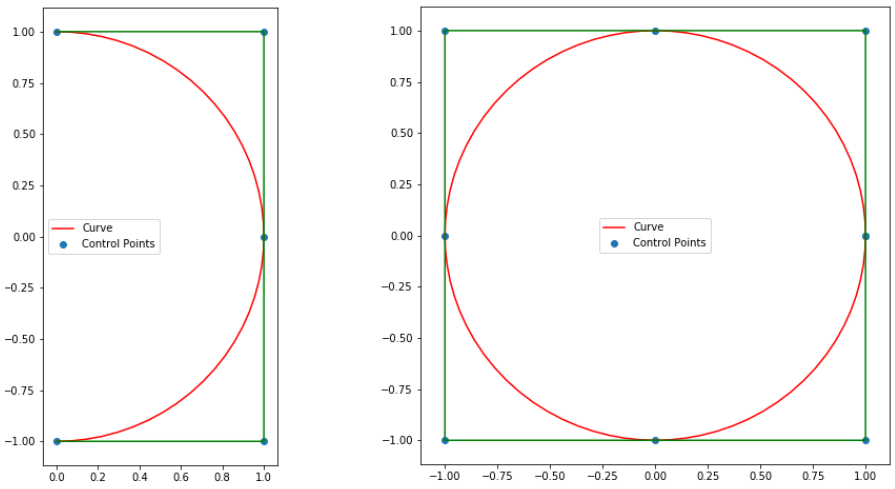

Figure 1: a) *Semi circle* control points with generated curve, b) *Full circle* control points with generated curve.

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
```

The output is the homogeneous *control net* $\boldsymbol{P_{ij}^{w}}$ which together with *knot vectors* Uf and Uh define the NURBS 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
```

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.

3.3 Compute Volume

Now that we have defined the NURBS volume, we need a routine for computing it. Algorithm **VolumePoint()** expands on the algorithms from (NURBS BOOK) 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
```

4 Refinement

4.1 Knot Insertion

4.2 T Splines

4.3 Local Refinement

5 Implementation outline

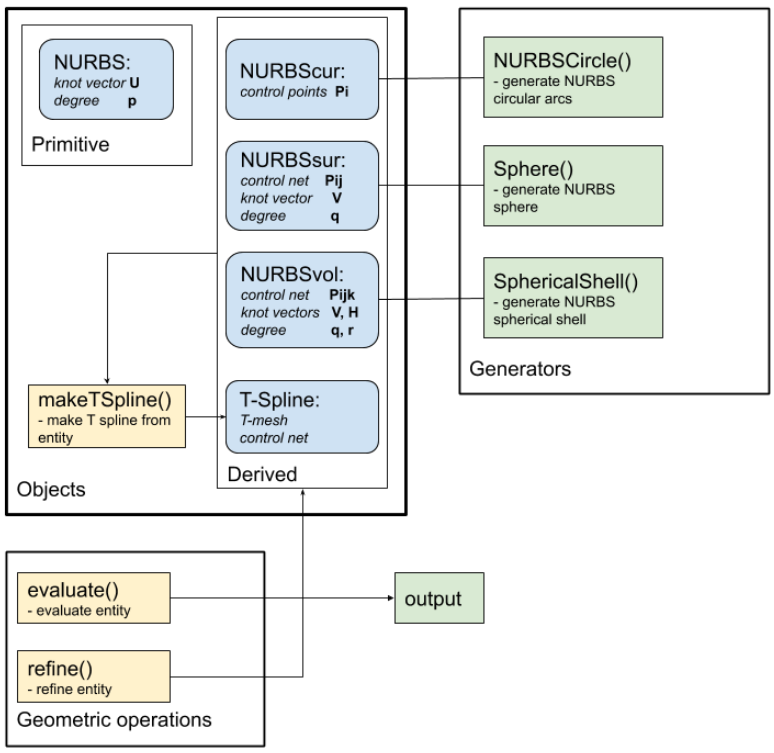

Figure 2: Implementation Design

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.

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.

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.

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.

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.

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.

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.

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.