Report 2048465-TN-01-1: Surface mesh generation

David Moxey, University of Exeter
Chris Cantwell & Spencer Sherwin, Imperial College London

12th May 2021

Contents

1 Executive summary

This report focuses on the initial phase of work undertaken by the University of Exeter and Imperial College London to investigate challenges for generation of high-order meshes for the NEPTUNE project. In this first report that comprises deliverable 1.3 of our workplan, we assess surface mesh quality attained during the high-order mesh generation process, with a particular focus on the maximum deflection angle of surface normal from the underlying CAD representation.

1.1 Changelog

- 12/5/21: Initial version.

2 Introduction

Although the use of high-order finite element methods, such as the spectral/*hp* element method [1], have continued to gain popularity in recent years owing to their favourable numerical and computational efficiency properties, it is well known that a significant challenge in the deployment of such methods in broader industrial and scientific environments is handling the complex geometries that are typically required within these settings: in other words, in order to run a high-order simulation, one first requires a *high-order mesh*. Such meshes must not only accurately represent the geometry of interest, by curving straight-sided elements to align them with the underlying boundary, but also consider physics-specific challenges, such as the use of boundary layers in fluid dynamics to resolve high-shear regions.

Within NEPTUNE, it is clear that a key challenge to overcome is not only the development of solvers to accurately simulate challenging equations of state, but also to handle challenging geometries which further must accommodate features such as magnetic field lines that impinge on surfaces at shallow angles of attack of order 2°. From this perspective, it is not only necessary to ensure accruate surface alignment, but also to investigate the accuracy of face-interior surface normals from high-order surface meshes.

In this report, we will investigate the quality of surface mesh generation from the open-source high-order mesh generator *NekMesh*, which is a pre-processing tool based on the spectral/*hp* element framework *Nektar++* [2, 3]. In particular, we will focus on assessment from the perspective of two metrics: the deflection of surface normals from the 'exact' normals as specified by the underlying CAD surface; and the maximum distance of the mesh from the CAD surface evaluated at each solution point. In this manner, we can identify areas for potential improvement with regards to these metrics, which are discussed in section 5.

The rest of this report is structured as follows. A brief introduction and overview of the *a posteriori* high-order surface mesh generation process is discussed in section 3. In section 4, we consider two simple example geometries to assess surface mesh quality for known-good geometries. Finally, section 5 draws some brief conclusions and areas for future work.

3 High-order surface mesh generation

In this section, we briefly outline the various technologies and algorithms used to generate a high-order mesh within *NekMesh*. Starting with a CAD representation of the geometry, we outline how a linear surface mesh is first constructed from a mesh sizing distribution, and then how additional degrees of freedom are introduced so as to elevate the mesh to higher orders. A complete description, as this pertains to *NekMesh*, can be found in reference [4].

3.1 CAD representation

Typically the geometry of interest is modelled using a computer-aided design (CAD) system and represented by a BRep (boundary representation). A BRep is a top-down definition of the computational domain, where it is viewed as a volume of three-dimensional space interior to a boundary that can be interpreted as a closed shell or curved polyhedron. Its faces are regions defined on surfaces interior to curved polygons also lying on those surfaces. These polygons are composed by a set of edges, which in turn are segments of a curve, which themselves are bounded by points. The points are represented by their 3D Cartesian coordinates. Curves and surfaces are usually defined in CAD systems through a parametric representation in the form of non-uniform rational B-splines (NURBS). These representations provide a mapping between

parametric coordinates in a lower dimensional space to the 3D space, so that for example a curve is represented as a map between a parametric coordinate $t \in [t_{\min}, t_{\max}]$ and its Cartesian coordinate $\boldsymbol{c}(t)$, or a surface between $(u, v) \in [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ and coordinate $\boldsymbol{r}(u, v)$.

The handling and computational implementation of such representations is highly complex in practice, and one of the main challenges for mesh generation is ensuring a watertight BRep that is suitable for meshing. Due to this complexity, the starting point for *NekMesh* is the assumption of such a watertight BRep; furthermore, access to points, curves, surfaces and their various properties (such as calculating derivatives, normals or projections) is accessed via a suitable external CAD engine. Although several commercial CAD engines are available, in order to broaden access to high-order mesh generation routines, *NekMesh* utilises the open-source CAD framework OpenCASCADE [5]. In this report we limit ourselves to the use of OpenCASCADE; however, it is notable that *NekMesh* accesses CAD engines through a lightweight wrapper, which means that other engines can be incorporated in an API-independent manner.

3.2 Mesh sizing specification

Before any mesh construction can commence, a prerequisite is to specify a mesh sizing distribution that indicates the desired density of elements within the curves and surfaces of the BRep. In *NekMesh*, the goal is to do this in a manner that is as automatic as possible; i.e. using as few parameters as possible. We therefore adopt a curvature-based refinement approach, in which the underlying assumption is that regions of higher curvature generally require a smaller element size in order to resolve them. We briefly outline this approach here, but further details can be found in [6].

The radius of curvature, R, at a point on the surface is the radius of a circle that has "optimal" osculating contact with the surface. A measure of how well a mesh of size δ represents the surface will be given by the ratio $\epsilon = c/R$, where c is the maximum distance between the mesh element and the surface. If we approximate the surface by its osculating circle we can relate the mesh sizing, the radius of curvature, and the ratio ϵ through

$$\delta = 2R\sqrt{\epsilon(2-\epsilon)}, \tag{1}$$

where now ϵ can be interpreted as a user-defined parameter that controls the mesh resolution with respect to surface curvature. For instance, decreasing ϵ will increase the number of elements specified for a given curvature. In practice, the radius of curvature is calculated as $R = \min\left\{{k_1}^{-1}, {k_2}^{-1}\right\}$ where k_1 and k_2 are the principal curvatures obtained from the metric and curvature tensors (or first and second fundamental forms) of the parametric form of the CAD surfaces. This leads to an isotropic mesh specification since the directionality of the curvature is not considered, but extension to anisotropic meshes is possible.

Since the radius of curvature has the range $R \in [0, \infty)$, where $R = \infty$ corresponds to a flat surface, we must set bounds on δ, so that $\delta \in [\delta_{\min}, \delta_{\max}]$. The parameter $\delta_{\min}$ limits the density of elements in regions of high curvature, while $\delta_{\max}$ is the global maximum element size. The three constants $\delta_{\min}$, $\delta_{\max}$ and ϵ represent the only user-specified parameters required for the automatic mesh specification process. To specify a spatial distribution of mesh sizing, the three-dimensional domain is spatially subdivided and smoothed from the BRep through the use of an octree structure.

3.3 Surface mesh generation

The surface mesh is generated via a bottom-up approach, wherein mesh vertices are fixed to the vertices of the CAD (0D); curves are then meshed in their 1D parameter space using the

bounding vertices; and the the 2D parameter spaces of the surfaces are then meshed, using the curve meshes as boundaries. Should a volume mesh be desired, the resulting interior volume is constructed using the surface mesh as its exterior constraint.

So far, none of the techniques discussed above are related to high-order meshes in particular: indeed bottom-up mesh generation is typically adopted in standard linear mesh generation frameworks. However, when elevating a linear to high-order mesh, maintaining the connection between mesh vertices and their corresponding parametric coordinates within CAD vertices, curves and surfaces is critical in ensuring that optimal node locations can be found for the new degrees of freedom needed to construct high-order elements. A key motivation for the development of *NekMesh* is that when using external open-source or commerical linear mesh generation technology, such a connection is typically lost in the resultant surface mesh. Use of this bottom-up approach therefore ensures that the CAD information associated with all the vertices, edges and faces in the surface mesh is kept and can be utilised easily in the high-order stages of the meshing pipeline.

In the remainder of this section we discuss the (linear) discretisation of curves and surfaces, and then outline the high-order mesh generation process.

3.3.1 Curve discretisation

The discretisation of a CAD curves $\boldsymbol{c}(t)$ into linear segments approximately compliant with the spacing specified by the mesh control system is achieved through a *distribution function*. If the mesh spacing at a point $\mathbf{c}(t)$ is δ, then the number of subdivisions generated in a small interval of length ds in its neighbourhood, so δ is approximately constant, is ds/δ. Further, if ds corresponds to the arc length of the curve, the total number of subdivisions for an interval $t_{\min} \leq t \leq t_k$ will be

$$I(t_k) = \int_{t_{\min}}^{t_k} \frac{1}{\delta(t)} \left\| \frac{d\mathbf{c}}{dt} \right\| dt \tag{2}$$

The positions $t_k; k = 1, \ldots, N_e - 1$ of the internal nodes to be created are the solutions of the equation

$$\phi(t_k) = N_e \frac{I(t_k)}{I(t_{\max})} = k \quad ; \qquad k = 1, \ldots, N_e - 1 \tag{3}$$

where $\phi(t)$ is the *distribution function* and N_e denotes the number of sides generated on the curve. Its value is chosen to be the nearest integer value to $I(t_{\max})$. The solution of equation (3) requires first to approximate the distribution of $\delta(t)$ in equation (2) by sampling the mesh spacing along the curve. The samples are chosen to be equally spaced along the curve at a distance smaller than $\delta_{\min}$. The integral is then evaluated using a suitable quadrature and the solution to equation (3) is obtained via Newton iteration. More details of the method are given in reference [7].

3.3.2 Surface discretisation

Meshing the surfaces in the bottom-up meshing process requires the meshing of a 2D domain, the parametric space of the surface, which is bound by the mesh vertices and edges as defined by the curve discretisations. *NekMesh* utilises an adapted version of the code *Triangle*, a 2D Delaunay mesh generation library [8], to deal with the specific aspects of surface mesh generation. The reason for choosing the Delaunay method is that it guarantees a unique triangulation for a given set of points, provided the circumcircle passing through the three vertices of any triangle in the mesh does not contain another point in the mesh, with the co-circular case requiring special handling to account for floating-point arithmetic. Further, the

implementation via robust geometric predicates [9] of the circumcircle test, the incremental point insertion and boundary sides recovery algorithms in *Triangle* ensures that the mesh is valid at each stage of the generation process.

The main drawback of the Delaunay approach for the purposes of surface mesh generation is that it tends to create a high quality, isotropic triangulation in the 2D space. Therefore the 3D mesh will inherit the distortion of the parameter space and elements defined in the 2D parameter space may become stretched and distorted in 3D space. This effect depends on how the surface is parameterised; approaches to mitigate this can be found in [4].

The method used for generating meshes on individual surfaces throughout this work proceeds as follows. Beginning with the discretisation of the CAD curves which represent the boundary of the surface, a set of mesh nodes and edges are determined. *Triangle* is used to obtain an initial triangulation of these boundary points in the parameter space. New mesh points are then determined by looking over the triangles of the mesh in 3D space. If any side of the triangle is greater in length than $\sqrt{2}\delta(\mathbf{x})$, where $\mathbf{x}$ is taken to be the midpoint of the side, a new point is defined at the centre of the triangle. The set of points, interior and boundary, are then re-meshed. The process is repeated until all the sides in the triangulation conform to the mesh spacing specification. In order to enhance the quality of the resultant mesh, various local refinement strategies are employed: notably, the use of edge swapping dependent on optimal valance and minimum angle, and mesh smoothing, which modifies the positions of the interior nodes without changing the connectivity of the grid.

3.3.3 High-order surface mesh generation

A high-order element Ω^e of a surface triangulation of order P is represented through a polynomial mapping $\boldsymbol{\chi}^e : \hat{\Omega} \to \Omega^e$, where $\hat{\Omega} = \{(\xi_1, \xi_2) \mid \xi_1, \xi_2 \in [-1, 1], \xi_1 + \xi_2 \leq 0\}$ is a reference simplex and $\boldsymbol{\xi} = (\xi_1, \xi_2)$ denotes a coordinate in $\hat{\Omega}$. Furthermore, this mapping is typically written using collocated Lagrange interpolants $\boldsymbol{\ell}_n(\boldsymbol{\xi})$ that depend on a choice of $N = (P+1)(P+2)/2$ cubature points $\{\boldsymbol{\xi}_n \mid \boldsymbol{\xi}_n \in \hat{\Omega}, 1 \leq n \leq N\}$, such as the α-optimised points of Hesthaven or a distribution of Fekete points, both discussed in [10]. In this manner, world-space coordinates $\boldsymbol{x}$ may be written through the expansion

$$\boldsymbol{x} = \boldsymbol{\chi}^e(\boldsymbol{\xi}) = \sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{\ell}_n(\boldsymbol{\xi}).$$

The challenge therefore is to determine a set of spatial coordinates $\boldsymbol{x}_n$ which correspond to each cubature point $\boldsymbol{\xi}_n$, so that 'high quality' surface meshes are defined, using a definition of quality that aligns with the problem of interest. It is well known that inaccuracies in the representation of the geometric boundary have a significant impact on the flow solution in high-order simulations. These inaccuracies include highly distorted surface elements, mesh nodes being a significant distance from the true CAD surface, and under-representation of the geometric curvature due to using an insufficiently polynomial order with too large a element. In the problems of interest to NEPTUNE, we may also add to this list that surface normals that result from the high-order element, i.e.

$$\boldsymbol{n}(\boldsymbol{\xi}) = \frac{\dfrac{\partial \boldsymbol{\chi}^e}{\partial \xi_1} \times \dfrac{\partial \boldsymbol{\chi}^e}{\partial \xi_2}}{\left\| \dfrac{\partial \boldsymbol{\chi}^e}{\partial \xi_1} \times \dfrac{\partial \boldsymbol{\chi}^e}{\partial \xi_2} \right\|} \tag{4}$$

should be closer than 0.1° to those normals observed using the CAD engine.

If the vertex locations of the linear surface mesh are taken to be fixed, producing a high-order surface can be accomplished simply by using an affine mapping of the triangle in the 2D parameter plane to the reference triangle of a high-order element. This can then be used to locate the new high-order nodes in the parameter space, which are then projected into 3D using the CAD engine. However, this means that the high-order triangles will inherit the distortion of the CAD surface, lowering the quality of the mesh and in some cases causing invalid elements.

Curve discretization. The rest of this section presents a method to take the high-order surface mesh made using the affine mapping approach and optimise the location of the high-order nodes to reduce CAD induced distortion. This is done by modelling the mesh entities as spring networks and minimising the spring energy, in a similar approach to the work of [11]. This is expressed mathematically as finding

$$\min_{\boldsymbol{x}_n} f = \min \sum_s \frac{\|\boldsymbol{x}^1 - \boldsymbol{x}^2\|^2}{w_s}, \tag{5}$$

which states that f, the spring energy, is the sum over all the springs in the system, where $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ are the 3D locations of the nodes at the ends of the springs and w_s is the inverse of the spring stiffness, which is calculated as a function of the nodal distribution being targeted. Because the linear mesh vertices are held fixed during this procedure, the problem can be reduced to an entity-by-entity approach. First optimising mesh edges that lie on curves; then edges that lie on surfaces; and finally interior triangle faces that lie on CAD surfaces. In the first case (edges on CAD curves), the problem is a 1D optimisation of spring system in the curve's parameter space t.

$$f = \sum_{i=1}^{P} \frac{\|\boldsymbol{c}(t_{i+1}) - \boldsymbol{c}(t_i)\|^2}{w_i}, \tag{6}$$

Where i is one of the $P+1$ nodes along the high-order edge. Here, P is the polynomial order of the mesh being created and $w_i = z_{i+1} - z_i$, where z_i is the i-th entry in the distribution of nodal points in the where $-1 \leq z \leq 1$. The initial values of t are obtained from the linear 1D mapping

$$t_i = t_1 \left(\frac{1 - z_i}{2}\right) + t_{P+1} \left(\frac{1 + z_i}{2}\right) \quad i = 1, ..., P+1. \tag{7}$$

where t_1 and t_{P+1} are the parametric coordinates of the end nodes of the edge, which are the vertices in the linear mesh and are considered to be fixed.

Surface discretization. Performing the optimisation of the edges which lie on the CAD surfaces follows a similar procedure but is formulated in the 2D parameter plane, i.e

$$f = \sum_{i=1}^{P} \frac{\|\boldsymbol{r}(u_{i+1}, v_{i+1}) - \boldsymbol{r}(u_i, v_i)\|^2}{w_i}. \tag{8}$$

This procedure reduces the distortion found in the high-order edges by minimising the length of the edge; that is, the optimised high-order edge will lie approximately on the geodesic between the two end points on the surface.

The procedure for optimising the location of face interior nodes requires a slightly alternative approach. The system is considered as a set of freely movable nodes, consisting of those nodes

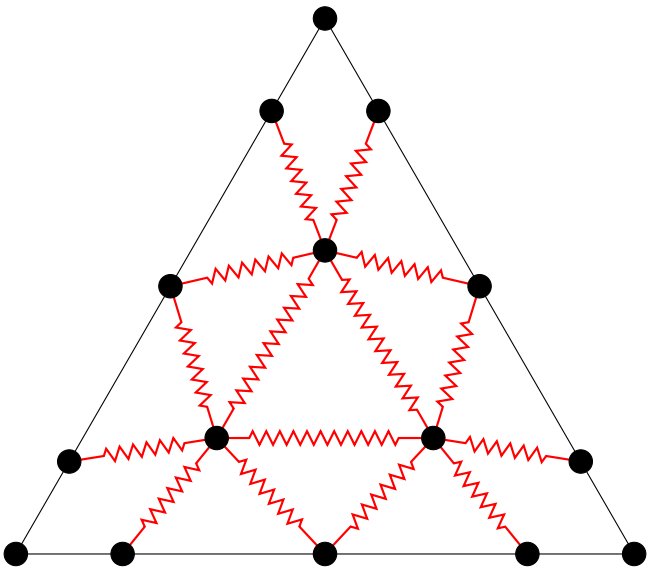


Figure 1: Distribution of spring-node system for face-interior nodes within a high-order $P = 4$ Fekete distribution.

lying on the interior of the triangle, and a set of fixed nodes which lie on the edges. Each of the free nodes is connected to a system of six surrounding nodes by springs, and this is the system which is minimised. In a triangle of order P, there are $(P-2)(P-1)/2$ interior nodes. The function f is

$$f = \sum_{i=1}^{(P-2)(P-1)/2} \sum_{s}^{6} \frac{\|\boldsymbol{r}(u_i, v_i) - \boldsymbol{r}(u_s, v_s)\|^2}{w_s}, \tag{9}$$

where w_s is calculated as the distance between the two nodes in a reference equilateral triangle, shown in figure 1 along with the connectivity of the springs. The choice of a six spring system means that the method is applicable to any point distribution at any order. For example, figure 1 shows a $P = 4$ triangle with a Gauss-Lobatto-Legendre distribution along the edges and a triangular Fekete distribution for the face interior points.

The optimisation of the energy itself must be performed numerically. A number of optimisation algorithms are suitable, since the function readily admits analytic derivatives. However, since each CAD entity is typically defined using a bounded parametric space, this imposes some limits in terms of constrained optimisation. In practice therefore, we have found that a bounded version of the BFGS algorithm [12] is well-suited to this problem and produces optimal locations after only a few iterations.

4 Test cases

Since surface normal representation is a key quality, a notable omission in the high-order generation process above is that the functional f defined in equation (5), which is used to optimise for node locations, is designed to only target geodesics of the underlying surface: no particular emphasis is given in the optimisation to other metrics, such as normal deflection. Therefore there remains a question of whether this approach is well-suited for the specific NEPTUNE applications.

In this section, we attempt to investigate this through the use of two simple geometries. The first is a straightforward sphere, for which we would expect excellent agreement with the exact

and CAD normals. The second geometry is a more complex box section, which comprises several cylindrical sections arranged in a cuboid structure with weldments at each corner. This provides a more challenging test, although still being relatively simple. Moreover, each case contains only curved surfaces, since preliminary investigation showed that planar surfaces clearly present far less of a challenge from this perspective. Each problem is then be examined at a variety of mesh densities and polynomial orders. However, before presenting the results, we first take the opportunity to discuss the implementation of a new module within *NekMesh* undertaken during this work package to calculate the desired metrics.

4.1 Methodology and *NekMesh* code

In *NekMesh*, we adopt a 'pipeline' framework, whereby either CAD or the resulting linear/high-order meshes are translated through a series of modules. Each module has a specific purpose: be that to input (e.g. read an existing mesh, generate a surface mesh from CAD, etc); process a mesh (e.g. elevate a linear mesh to high order) or to write the mesh to a number of known formats. Programattically, a module is represented as an abstract base class `Module`, and the `Mesh` object is passed to a virtual `Process` function that allows the module to run its task. For this deliverable a new `CADMetrics` module was designed to compute the desired metrics by evaluating surface normals of high-order elements and comparing their deflection to those given by the CAD engine. As a secondary assessment, we also consider the maximum displacement from the CAD surface.

The first task is to determine the order of the incoming mesh, and from this obtain the distribution of cubature points $\boldsymbol{\xi}_n$ within the reference element $\hat{\Omega}$.

`Process` function inside `CADMetrics` module.

```
// Assume all elements have homogeneous order and are triangles embedded in
// 3D space.
auto elmt0 = m_mesh->m_element[2][0];
int order = elmt0->GetEdge(0)->m_edgeNodes.size() + 1;
int npts = (order + 1) * (order + 2) / 2;

// Get points type (e.g. Fekete, electrostatic)
LibUtilities::PointsType pt = elmt0->GetCurveType();

// Get the points themselves from the points manager.
LibUtilities::PointsSharedPtr pts = LibUtilities::PointsManager()[
    LibUtilities::PointsKey(order + 1, pt)];

// Get positions of cubature points in standard element.
Array<OneD, NekDouble> r(npts), s(npts);
pts->GetPoints(r, s);
```

Ultimately, we will require the computation of the surface normal: this necessitates the computation of the derivatives of the mapping $\boldsymbol{\chi}^e$ for each element Ω^e. This can be achieved through the computation of the derivative matrices $\boldsymbol{D}_0$ and $\boldsymbol{D}_1$, which when multiplied by the coordinate vector $\{\boldsymbol{x}_n\}_{n=1}^N$ will yield partial derivatives with respect to ξ_1 and ξ_2, accordingly. Given the locations of the nodal points, we can use the broader *Nektar++* library to construct a `NodalUtilTriangle` object, which computes these matrices for us.

Calculating derivative matrices.

```
// Build derivative matrices from Vandermonde matrix using
// the NodalUtilTriangle class.
std::unique_ptr<LibUtilities::NodalUtilTriangle> triutil =
    std::make_unique<LibUtilities::NodalUtilTriangle>(order, r, s);

// Compute derivative matrices for each coordinate direction.
auto D0 = triutil->GetDerivMatrix(0);
auto D1 = triutil->GetDerivMatrix(1);
```

Then we simply loop over each element and call a utility member function, `ElementStats`, which will compute the desired statistics for this element.

Computing elemental statistics.

```
// Loop over all 2D surface elements, compute normal deflection at each
// quadrature point and surface distance at each point, then place into
// angs and dists vectors accordingly.
std::vector<NekDouble> angs, dists;
int cnt = 0;
for (auto &elmt : m_mesh->m_element[2])
{
    ElementStats(elmt, D0, D1, angs, dists);
    ++cnt;
}
```

Inside `ElementStats`, we first obtain the list of coordinates $\{\boldsymbol{x}_n\}_{n=1}^N$ that has been generated by the surface meshing process outlined in section 3.3.2, and then multiply by the derivative matrices to obtain the desired vectors for the normal calculation.

Compute derivatives.

```
// Get a vector of element coordinates.
std::vector<NodeSharedPtr> nodes;
elmt->GetCurvedNodes(nodes);
int npts = nodes.size();

// Construct coordinate vectors from this list.
NekVector<NekDouble> xc(npts), yc(npts), zc(npts);
for (int i = 0; i < npts; ++i)
{
    xc[i] = nodes[i]->m_x;
    yc[i] = nodes[i]->m_y;
    zc[i] = nodes[i]->m_z;
}

// Compute derivatives.
NekVector<NekDouble> xd0 = (*D0) * xc, xd1 = (*D1) * xc;
NekVector<NekDouble> yd0 = (*D0) * yc, yd1 = (*D1) * yc;
```

```
NekVector<NekDouble> zd0 = (*D0) * zc, zd1 = (*D1) * zc;
```

Looping over each point, we then compute the surface normal as given in equation (4).

Computing surface normals.

```
// Compute surface normal from curl at each point.
NekDouble N0 = yd0[i] * zd1[i] - zd0[i] * yd1[i];
NekDouble N1 = zd0[i] * xd1[i] - xd0[i] * zd1[i];
NekDouble N2 = xd0[i] * yd1[i] - yd0[i] * xd1[i];
NekDouble norm_abs = sqrt(N0 * N0 + N1 * N1 + N2 * N2);
N0 /= norm_abs;
N1 /= norm_abs;
N2 /= norm_abs;
```

Finally, we can then loop over each CAD surface that a node is attached to, and query the CAD surface at the parametric coordinates stored for that node. We note that a node might well be attached to a number of surfaces if it happens to be on a curve that connects surfaces, or indeed a vertex that connects multiple curves; we therefore assume the evaluation that gives us the lowest value of the normal deflection is the 'best', although in theory for smooth surfaces these differences should be minimal.

Querying the CAD engine and computing desired metrics.

```
std::vector<NekDouble> tmpangs, tmpdists;
for (auto &surfpair : nodes[i]->CADSurfList)
{
    // CAD surface object.
    CADEngine::CADSurf surf = surfpair.second.first.lock();

    // Location of (u, v) within parametric space.
    auto uv = surfpair.second.second;

    // Calculate CAD normal and position.
    auto cadN = surf->N(uv), cadP = surf->P(uv);

    if (ang != ang)
    {
        // Occasionally CAD engine fails to evaluate normal...
        continue;
    }

    // Calculate distance.
    tmpdists.push_back(
        sqrt(
            (cadP[0] - xc[i]) * (cadP[0] - xc[i]) +
            (cadP[1] - yc[i]) * (cadP[1] - yc[i]) +
            (cadP[2] - zc[i]) * (cadP[2] - zc[i])
            ));
```

```
    // Calculate normal deflection angle.
    NekDouble ang = std::acos(
        N0 * cadN[0] + N1 * cadN[1] + N2 * cadN[2]);

    std::vector<NekDouble> tmpangs2 = {
        std::abs(ang), std::abs(M_PI-ang), std::abs(ang-M_PI)
    };

    // Check also pi - angle, in case normal is reversed in orientation.
    tmpangs.push_back(*std::min_element(tmpangs2.begin(), tmpangs2.end()));
}

// Store lowest answers.
dists.push_back(*std::min_element(tmpdists.begin(), tmpdists.end()));
angs.push_back(*std::min_element(tmpangs.begin(), tmpangs.end()));
```

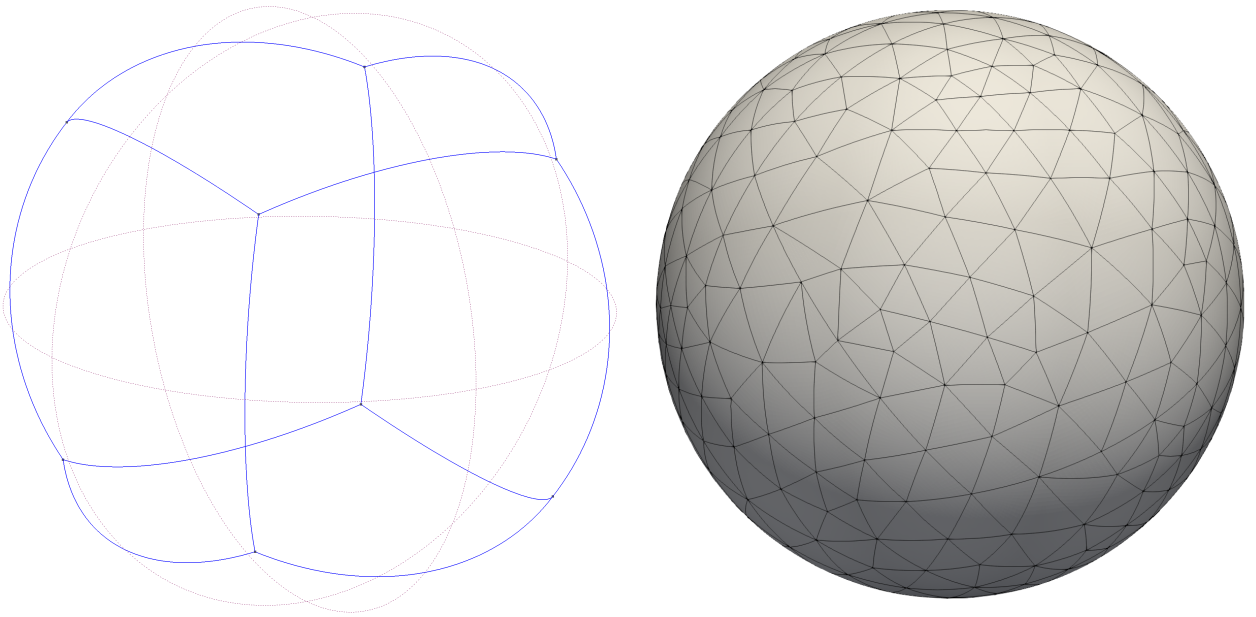
The module concludes by simply printing out the surface area and the desired statistics.

Compute and print desired statistics.

```
// Compute total surface area
int nCadSurf = m_mesh->m_cad->GetNumSurf();
NekDouble totarea = 0.0;
for (int i = 0; i < nCadSurf; ++i)
{
    totarea += m_mesh->m_cad->GetSurf(i+1)->Area();
}

m_log(INFO) << "Surface area: " << totarea << std::endl;

// Compute maximum and average normal angle deflection.
NekDouble avgang =
    std::accumulate(angs.begin(), angs.end(), 0.0) / angs.size();

m_log(INFO) << "Max. angle  : "
            << rad2deg(*std::max_element(angs.begin(), angs.end()))
            << std::endl;
m_log(INFO) << "Avg. angle  : " << rad2deg(avgang) << std::endl;

// Compute maximum distance to CAD surface.
m_log(INFO) << "Max. dist.  : "
            << *std::max_element(dists.begin(), dists.end()) << std::endl;
```

4.2 Spherical geometry

The first test case is a simple sphere of diameter 1 with corresponding surface area of π, shown in figure 2. The CAD is comprised of six equally sized tensor-product patches of NURBS surfaces. Each edge is a great circle of the sphere, and thus this yields an exact representation



(a) Sphere defined using 6 NURBS surfaces.

(b) Resultant high-order mesh at $P = 4$.

Figure 2: Simple sphere surface in (a) BRep form, where dashed lines show surfaces and solid lines show curves; (b) meshed form, after high-order mesh generation at polynomial order $P = 4$ with $\delta = 0.1$.

of the sphere as well as removes effects of surface distortion owing to the use of e.g. degenerate patches.

For this series of simulations, we prescribe $\delta_{\min} = \delta_{\max} = \epsilon =: \delta$ with a fixed value of $\delta = 0.1$, 0.05, 0.01 and 0.005, so that all meshes are uniform in element size. This yields mesh densities of between 814 and 333,714 elements, or between 259 and 106,224 elements per surface area unit. The resultant statistics for this case can be found in table 1. Broadly, this case is rather as one would expect to see; in virtually all cases, the maximum normal deflection angle $\theta_{\max}$ is well below the required 0.1° in virtually all cases, as is the average θ_{avg}; furthermore, as the element or nodal point density is increased, either as the curvature bound δ is reduced or the polynomial order increased, there is a broad reduction in all metrics, as expected. Uniformly the surface distance is near machine tolerance.

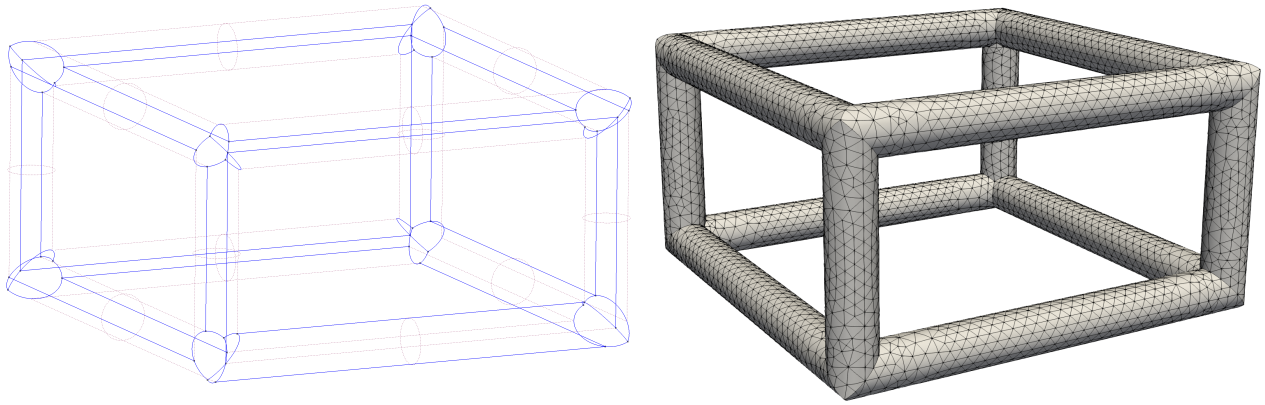
4.3 Cuboid box section

In this section, we evaluate normals for a cuboid box section, the BRep of which can be depicted visually in figure 3(a). This geometry contains more complexity, in that it comprises 12 cylindrical sections arranged in a cuboid section, which are connected with weldments at each corner. Consequently the BRep contains 24 faces and 82 curves. As in the previous section, we record the same statistics but now utilising a slightly different choice of δ to allow for the increased level of curvature in this model; we select $\delta = 0.007$, 0.003 and 0.0008 respectively. A high-order mesh at $P = 5$ is visualised in figure 3(b).

The presented results shown in table 2 reveal much of the same trend as we see in the sphere geometry, although broadly the maximum and average angles of deflection are larger. Very curiously however, at $P = 4$ and $P = 5$, there is a large deflection noted at the finest resolution, virtually at a right angle to the desired normal. Clearly the presence of such a normal in reality would render the mesh virtually unusable. To investigate this further, the location of points

P	δ	$\theta_{\max}$ [°]	θ_{avg} [°]	$d_{\max}$	N_{elmt}
2	0.1	$5.82 \cdot 10^{-1}$	$1.21 \cdot 10^{-1}$	$1.35 \cdot 10^{-15}$	814
2	0.05	$1.18 \cdot 10^{-1}$	$2.73 \cdot 10^{-2}$	$1.45 \cdot 10^{-15}$	3,414
2	0.01	$6.75 \cdot 10^{-3}$	$1.15 \cdot 10^{-3}$	$1.52 \cdot 10^{-15}$	82,384
2	0.005	$1.97 \cdot 10^{-3}$	$2.75 \cdot 10^{-4}$	$1.61 \cdot 10^{-15}$	333,714
3	0.1	$6.54 \cdot 10^{-2}$	$9.91 \cdot 10^{-3}$	$1.57 \cdot 10^{-15}$	814
3	0.05	$8.26 \cdot 10^{-3}$	$1.17 \cdot 10^{-3}$	$1.57 \cdot 10^{-15}$	3,414
3	0.01	$9.02 \cdot 10^{-5}$	$1.03 \cdot 10^{-5}$	$1.64 \cdot 10^{-15}$	82,384
3	0.005	$1.29 \cdot 10^{-5}$	$1.27 \cdot 10^{-6}$	$1.61 \cdot 10^{-15}$	333,714
4	0.1	$3.69 \cdot 10^{-3}$	$3.08 \cdot 10^{-4}$	$1.44 \cdot 10^{-15}$	814
4	0.05	$1.86 \cdot 10^{-4}$	$1.48 \cdot 10^{-5}$	$1.58 \cdot 10^{-15}$	3,414
4	0.01	$1.71 \cdot 10^{-6}$	$2.68 \cdot 10^{-7}$	$1.54 \cdot 10^{-15}$	82,384
4	0.005	$1.91 \cdot 10^{-6}$	$2.67 \cdot 10^{-7}$	$1.64 \cdot 10^{-15}$	333,714

Table 1: Mesh generation statistics of maximum normal deflection angle $\theta_{\max}$, average normal deflection angle θ_{avg}, maximum surface distance $d_{\max}$ and number of elements N_{elmt} for the sphere geometry, given an input polynomial order P and curvature bound δ.



(a) BRep of box section.

(b) Resultant high-order mesh at $P = 4$.

Figure 3: Box section surface in (a) BRep form, where dashed lines show surfaces and solid lines show curves; (b) meshed form, after high-order mesh generation at polynomial order $P = 5$ with $\delta = 0.003$.

P	δ	$\theta_{\max}$ [°]	θ_{avg} [°]	$d_{\max}$	N_{elmt}
2	0.007	$5{\cdot}10^{1}$	$1.79 \cdot 10^{0}$	$1.46 \cdot 10^{-16}$	1,676
2	0.003	$3.03 \cdot 10^{0}$	$8.76 \cdot 10^{-2}$	$1.46 \cdot 10^{-16}$	10,696
2	0.0008	$1.4 \cdot 10^{-1}$	$2.34 \cdot 10^{-3}$	$1.53 \cdot 10^{-16}$	129,734
3	0.007	$3.16 \cdot 10^{1}$	$7.57 \cdot 10^{-1}$	$1.46 \cdot 10^{-16}$	1,676
3	0.003	$1.12 \cdot 10^{0}$	$2.98 \cdot 10^{-2}$	$1.46 \cdot 10^{-16}$	10,696
3	0.0008	$4.34 \cdot 10^{-2}$	$7.55 \cdot 10^{-4}$	$1.72 \cdot 10^{-16}$	129,734
4	0.007	$2.12 \cdot 10^{1}$	$9.98 \cdot 10^{-2}$	$1.46 \cdot 10^{-16}$	1,676
4	0.003	$1.32 \cdot 10^{-1}$	$4.33 \cdot 10^{-4}$	$1.53 \cdot 10^{-16}$	10,696
4	0.0008	$9{\cdot}10^{1}$	$3.15 \cdot 10^{-3}$	$1.86 \cdot 10^{-16}$	129,734
5	0.007	$8.68 \cdot 10^{0}$	$5.35 \cdot 10^{-2}$	$1.46 \cdot 10^{-16}$	1,676
5	0.003	$4.94 \cdot 10^{-2}$	$1.83 \cdot 10^{-4}$	$1.62 \cdot 10^{-16}$	10,696
5	0.0008	$9{\cdot}10^{1}$	$3.84 \cdot 10^{-3}$	$1.87 \cdot 10^{-16}$	129,734

Table 2: Mesh generation statistics of maximum normal deflection angle $\theta_{\max}$, average normal deflection angle θ_{avg}, maximum surface distance $d_{\max}$ and number of elements N_{elmt} for the box geometry, given an input polynomial order P and curvature bound δ.

with a deflection angle above 1 radian is visualised in figure 4, where it is evident that these occur on or very close to the weldment boundaries. It is therefore perhaps more probable in this instance, since the high-order elements themselves seem well-aligned with the surface, that the CAD engine itself is generating spurious normals close to the edge of the parametrisation. Indeed on the whole, average deflections remain very small throughout. We do additionally note that the mesh in this figure is not the finest level mesh, but is used purely for visualisation purposes.

5 Conclusions and future directions

This report has outlined the methods and techniques used for high-order surface mesh generation within the *NekMesh* mesh generation package, with a particular focus on the maximum deflection angle of surface normals from the CAD engine. Broadly, it has been found that the existing methodology, at least for simple geometries, is capable of generating high quality surface meshes. Due to time constraints in the initial phase of this project, it was not possible to consider far more complex geometries due to time required for CAD cleanup and watertight geometry testing; however this is a clear area for future study, and this work outlines at least the potential for this methodology.

There are a clear number of routes for future work in this area. Firstly, the abnormalities seen in the box geometry in the previous section should be validated using an alternative CAD engine. *NekMesh* has a link to the commercial CAD engine CADfix, which could be used as an alternative testbed for the geometry. Additionally, it is clear that if the requirement is to guarantee surface normal representation below a user-supplied threshold, then a route towards enabling this would be to incorporate additional terms into the minimisation process defined in equation (5). Clearly a balance between surface geometry representation (i.e. the ability to track geodesics) together with normal representation must be struck. However, the addition of an extra term that gives high functional values to configurations having large deflections from the CAD normal should lead to the desired effect. However we do note that in this case, computing derivatives may be more challenging, in which case non-gradient based optimisation

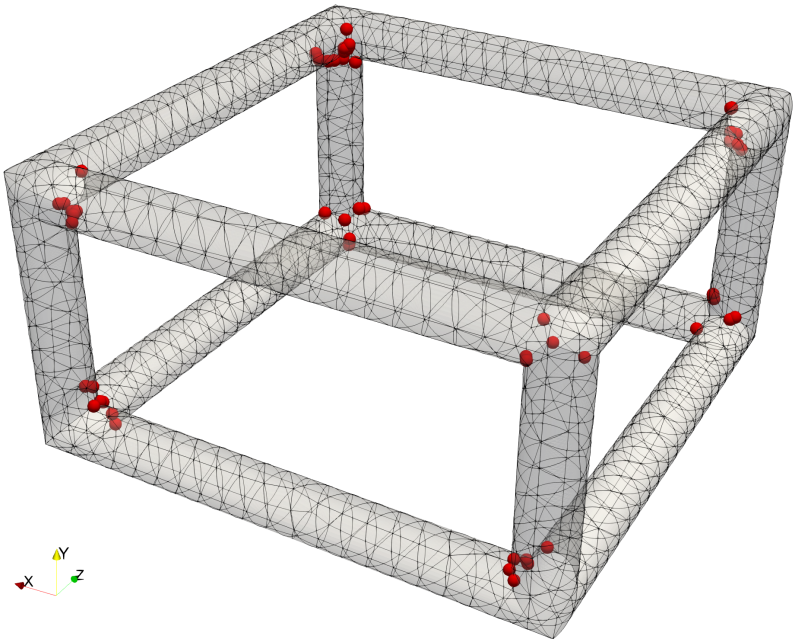


Figure 4: Visualisation of location of high deflection points with reported normals above 1 radian.

techniques could be appropriate.

References

[1] George Karniadakis and Spencer Sherwin. *Spectral/Hp Element Methods for Computational Fluid Dynamics*. Oxford University Press, 2013.

[2] David Moxey, Chris D. Cantwell, Yan Bao, Andrea Cassinelli, Giacomo Castiglioni, Sehun Chun, Emilia Juda, Ehsan Kazemi, Kilian Lackhove, Julian Marcon, Gianmarco Mengaldo, Douglas Serson, Michael Turner, Hui Xu, Joaquim Peiró, Robert M. Kirby, and Spencer J. Sherwin. Nektar++: Enhancing the capability and application of high-fidelity spectral/hp element methods. *Computer Physics Communications*, 249:107110, April 2020.

[3] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. de Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. M. Kirby, and S. J. Sherwin. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications*, 192:205–219, 2015.

[4] Michael Turner. *High-Order Mesh Generation for CFD Solvers*. PhD Thesis, Imperial College London, 2017.

[5] Open Cascade - software development company. https://www.opencascade.com/.

[6] M. Turner, D. Moxey, and J. Peiró. Automatic mesh sizing specification of complex three dimensional domains using an octree structure. In *24th International Meshing Roundtable*, 2015.

[7] J. Peiró. Surface grid generation. *Handbook of grid generation*, pages 19–1, 1999.

[8] Jonathan Richard Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Workshop on Applied Computational Geometry*, pages 203–222. Springer, 1996.

[9] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3):305–363, 1997.

[10] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer Science & Business Media, 2007.

[11] S. J. Sherwin and J. Peiró. Mesh generation in curvilinear domains using high-order elements. *International Journal for Numerical Methods in Engineering*, 53(1):207–223, 2002.

[12] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.