Simplifying high-order mesh generation for computational scientists

Jeremy Cohen
Department of Computing
Imperial College London
South Kensington Campus
London, SW7 2AZ, UK
Email: jeremy.cohen@imperial.ac.uk

Julian Marcon, Michael Turner, Chris Cantwell,
Spencer Sherwin, Joaquim Peiró
Department of Aeronautics
Imperial College London
South Kensington Campus
London, SW7 2AZ, UK

David Moxey
College of Engineering,
Mathematics and Physical Sciences
University of Exeter
North Park Road
Exeter, EX4 4QF, UK

***Abstract*—Computational modelling is now tightly integrated into many fields of research in science and industry. Computational fluid dynamics software, for example, gives engineers the ability to model fluid flow around complex geometries defined in Computer-Aided Design (CAD) packages, without the expense of constructing large wind tunnel experiments. However, such modelling requires translation from an initial CAD geometry to a *mesh* of many small elements that modelling software uses to represent the approximate solution in the numerical method. Generating sufficiently high-quality meshes for simulation is a time-consuming, iterative and error-prone process that is often complicated by the need to interact with multiple command-line tools to generate and visualise the mesh data. In this paper we describe our approach to overcoming this complexity through the addition of a *meshing console* to Nekkloud, a science gateway for simplifying access to the functionality of the Nektar++ spectral/*hp* element framework. The meshing console makes use of the NekMesh tool in Nektar++ to help reduce the complexity of the mesh generation process. It offers a web-based interface for specifying parameters, undertaking meshing and visualising results. The meshing console enables Nekkloud to offer support for a full, end-to-end simulation pipeline from initial CAD geometry to simulation results.**

***Keywords*—science gateway, high-order, mesh generation, finite element method, reproducibility**

I. Introduction

Scientists and engineers have been making use of large-scale computation for many years to support modelling, simulation and analysis of complex physical processes. As computational infrastructure has become ubiquitous and more easily accessible, simulations at larger scales and higher resolution have become something that domain scientists expect to be able to undertake themselves, without having to rely on direct support from computer scientists or those with specialist technical training. Science gateways have emerged as a means of providing user-friendly interfaces to complex scientific tools and workflows on a variety of infrastructure, initially focusing on platforms such as computational grids [1]. Modern science gateways are just as likely to target cloud and cluster infrastructure, standalone resources, or to abstract away the underlying computational environment from the user entirely, offering the possibility of transparent access to advanced research infrastructure, as discussed in [2]. Science gateways can offer huge usability benefits [3] to end-users wanting to undertake challenging scientific processes and can provide an array of scientific community benefits to their target domains. These include support for reproducibility of processes and results, through the sorts of approaches described in [4], including sharing and provenance of configurations.

One of the most challenging (and frequently overlooked) aspects of modelling complex geometries, which arise from Computer-Aided Design (CAD) packages, is the translation of a CAD geometry into a *mesh* used to represent the geometry in the numerical method [5]. Generating meshes that are sufficiently high-quality for simulation is well-known to be a time-consuming, iterative and error-prone process that requires detailed technical knowledge to address issues that may be encountered as the mesh is constructed. It is frequently made more complex by the need to interact with multiple command-line tools to generate and visualise the mesh data. Ultimately, the process poses a significant technical challenge for entry-level users and a time drain for more experienced users. In many cases, this preprocessing phase can consume much of the overall time required to prepare and undertake a simulation.

We present an extension to an existing science gateway, Nekkloud [6], to provide a “meshing console” offering new functionality to support the generation of high-order mesh structures used in computational simulations in areas such as fluid and airflow modelling. This work focuses on enhancing the user experience of an iterative scientific workflow by providing a simple gateway-style interface that hides underlying complexity, while providing an improved means of managing, storing and sharing task configurations in order to capture domain-specific knowledge. This in turn supports the storage of a provenance trail enabling scientists to understand how a given output was achieved and to reproduce it in future.

The target software application for the meshing console is NekMesh [7], [8], a high-order mesh generator that is part of the open-source Nektar++ [9], [10] framework for undertaking high-order spectral/*hp* element analysis, a process similar in its approach to the finite element method. Nektar++ provides a set of libraries and a group of solvers that can be used to simulate problems in areas including automotive, aeronautical and mechanical engineering, and biomedical applications. The Nekkloud web application, which the meshing console is part of, was developed to address some of the challenges of working with Nektar++, offering a straightforward and flexible web-based interface for specifying, running and monitoring simulation jobs. Integration of meshing support into Nekkloud via addition of the meshing console will open up the specialist capabilities of NekMesh to users in a range of engineering and science domains. The console is designed to support the complex iterative workflow required to generate high-quality meshes. The resulting mesh data can be used within Nektar++ jobs, supporting a complete computational pipeline from an initial CAD geometry through to visualised simulation results.

In Section II we provide more detail of the mesh generation process, its challenges, and the motivation for the development of the meshing console. Section III describes the meshing console architecture and an overview of its implementation within Nekkloud. In section IV we look at the end-user experience of using the meshing console and the user community benefits the system can offer, with conclusions in Section V.

II. Mesh generation

The process of undertaking a simulation over a two- or three-dimensional model begins with a CAD geometry that will have been developed in one of a number of different CAD packages. When undertaking finite element analysis on such a model, it is necessary to break the domain up into a series of smaller units over which governing partial differential equations can be integrated numerically. This is the process of mesh generation which aims to provide a representation of the initial domain that is as geometrically accurate as possible.

Historically finite element/volume-based solvers relied on the generation of linear meshes for the solution of governing equations. Recent developments in high-order methods, which provide exponential error decay rates for sufficiently smooth solutions, increase data locality and improve cache efficiency. This makes them highly suitable for modern High-Performance Computing (HPC) architectures where limiting data movement is becoming increasingly important. One class of such high-order methods is the spectral/*hp* element method, used in the Nektar++ framework among others, which provides lower dispersion and diffusion errors than low-order methods. High-order methods generally use larger elements and are more dependent on a geometrically-accurate discretisation of the physical domain that linear meshes are not able to provide. They must use high-order *curvilinear* meshes which provide an enhanced sub-element level of geometrical accuracy. The generation of potentially highly curved elements however introduces additional challenges, the main one being the validity of elements, primarily that of avoiding self-intersection within elements. These difficulties make curvilinear mesh generation non-trivial and require high-order tools.

NekMesh undertakes the process of high-order mesh generation and manipulation for use in high-order solvers. It accepts a CAD geometry as input, with support for various input file formats. The meshing functionality makes use of the OPEN CASCADE [11]/Open CASCADE Community Edition [12] software to support the mesh generation process.

A. The user perspective

Beyond the technical aspects of the mesh generation process, there is the question of the end user workflow followed to obtain a satisfactory mesh structure using NekMesh. From this perspective, mesh generation is typically an iterative and time-consuming process. It is not uncommon for it to require the majority of the work undertaken for an end-to-end analysis of a given test case.

The user selects an initial set of mesh parameters that they believe to be acceptable for their desired mesh structure. These parameters are placed in a configuration file that NekMesh reads. The user then runs NekMesh to generate the mesh. The resulting mesh must then be converted into a format that can be visualised, generally VTK format. The user inspects the visualised mesh to check the quality of the generated structure. They may then need to tweak values from their initial input parameter set and re-run the meshing process to correct or improve any issues that they have spotted on visual inspection of the mesh. This process continues until the user is satisfied with a mesh they deem suitable for simulation. After simulation, they may realise that further improvement of the mesh is necessary, in which case they will need to tweak meshing parameter values and re-mesh again. The loop continues until the user obtains a converged mesh, i.e. simulation results do not change with further refinement.

The aim of the Nekkloud meshing console has been to automate this process as far as possible, while providing a straightforward and user friendly interface for the aspects of the process that require manual intervention.

III. Meshing console architecture and implementation

The Nekkloud [6] web application is a science gateway that was originally developed to support running parallel Nektar++ computations on cloud computing infrastructure. Since development of the initial prototype, some years ago, Nekkloud has been extended to include the ability to run jobs on cluster and standalone server platforms and to integrate the TemPSS (Templates and Profiles for Scientific Software) [13] tool that allows configuration of application input parameters via a structured, tree-style, visual interface. Nekkloud builds on top of lower-level tools offering flexibility for future development but with more technical complexity than would be required with a modern gateway framework. With the various mature tools now available to aid the development of science gateways, opportunities include building on top of the service layer of a framework such as Apache Airavata [14], or developing a portlet-style interface that integrates with a framework such as the Catania Science Gateway Framework [15]. The Nekkloud web application makes use of a custom simulation job deployment library that leverages SAGA-Python [16], a Python library for interacting with a range of job submission middleware and the web application is developed in Python using the Django web framework.

In this section we describe the architecture and implementation of the Nekkloud "*meshing console*", an additional set of functionality added to Nekkloud's web-based user interface that enables users to undertake mesh generation using NekMesh. The meshing console enables a user to upload their CAD geometry, iterate over the meshing, visualisation and optimisation loop described in Section II-A, and then obtain the resulting meshed geometry as a file in Nektar++ XML format. The resulting file can be downloaded by the user for use outside of the Nekkloud environment, or it can be used as input directly into a new Nektar++ computation within the Nekkloud environment. Figure 1 shows the workflow for a user working with the Nekkloud meshing console.

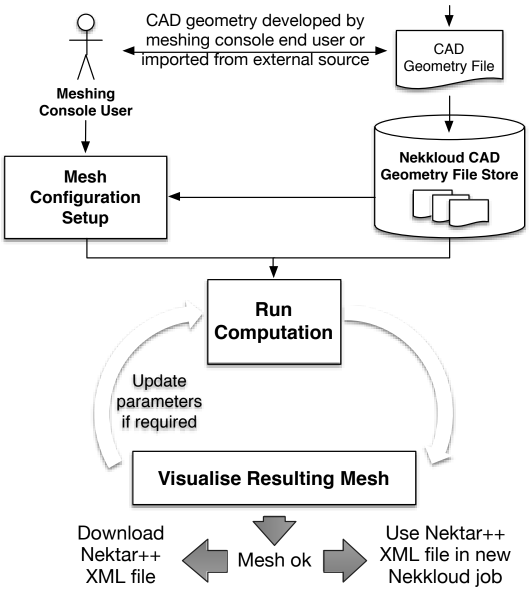
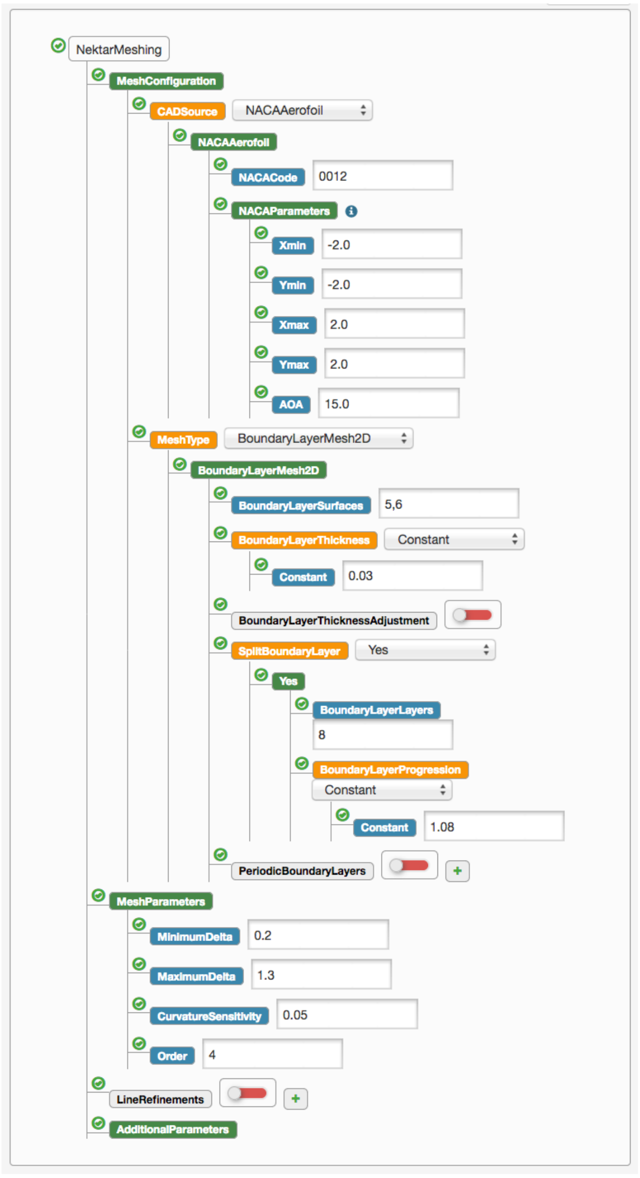

CAD geometry developed by meshing console end user or imported from external source
Meshing Console User
CAD Geometry File
Nekkloud CAD Geometry File Store
Mesh Configuration Setup
Run Computation
Update parameters if required
Visualise Resulting Mesh
Download Nektar++ XML file
Mesh ok
Use Nektar++ XML file in new Nekkloud job

Fig. 1. Example of the workflow followed to undertake the meshing process.

The processing status of a meshing task within the meshing console can be monitored using the task's job ID. Regular Nekkloud simulation jobs and meshing jobs are differentiated by job IDs beginning with either "`job-`" or "`mesh-job-`".

The NekMesh tool requires an input Mesh Configuration File (MCF). An MCF typically includes a list of interlinked parameters defining the properties to use for mesh generation. This includes core parameters required for all computations, such as the source geometry or the dimensionality of the mesh, but also sub-parameters that are only required in certain cases, triggered by other parameters. We leverage TemPSS, which is integrated into Nekkloud, to simplify MCF creation. A TemPSS template for the MCF parameters has been developed (see Figure 2). TemPSS builds on [17], a project based on the Bootstrap front-end framework, for displaying templates. When a user enters the required parameters into the MCF template in the web-based meshing console interface, TemPSS converts the parameters into a complete MCF file that is passed to the NekMesh tool. This workflow simplifies the process of creating/editing MCFs and encapsulates details of parameter names, combinations and applicability within the TemPSS template removing the need for users to acquire this knowledge before they can generate meshes. Users will generally want to create a new MCF template when undertaking a new meshing task and then edit this slightly for each iteration of the meshing process. The MCF XML format can be somewhat unintuitive to inexperienced users and the use of a template and automated MCF generation helps to ensure the specification of sensible values and correct formatting of the MCF. This can also assist more experienced users (e.g. by helping to avoid unintentional formatting or typing errors). We therefore consider the MCF generation to be an important element in linking together and supporting the complete mesh generation process.



- NektarMeshing
  - MeshConfiguration
    - CADSource: NACAAerofoil
      - NACAAerofoil
        - NACACode: 0012
        - NACAParameters
          - Xmin: -2.0
          - Ymin: -2.0
          - Xmax: 2.0
          - Ymax: 2.0
          - AOA: 15.0
    - MeshType: BoundaryLayerMesh2D
      - BoundaryLayerMesh2D
        - BoundaryLayerSurfaces: 5,6
        - BoundaryLayerThickness: Constant
          - Constant: 0.03
        - BoundaryLayerThicknessAdjustment
        - SplitBoundaryLayer: Yes
          - Yes
            - BoundaryLayerLayers: 8
            - BoundaryLayerProgression: Constant
              - Constant: 1.08
        - PeriodicBoundaryLayers
  - MeshParameters
    - MinimumDelta: 0.2
    - MaximumDelta: 1.3
    - CurvatureSensitivity: 0.05
    - Order: 4
  - LineRefinements
  - AdditionalParameters

Fig. 2. A completed, valid TemPSS template tree for an example meshing job to generate a NACA wing mesh. Some unused parameters are hidden.

To avoid having to upload input CAD geometry data multiple times, uploaded geometry files are stored in the meshing console for reuse, for example if several iterations of the meshing process are required to achieve the desired result.

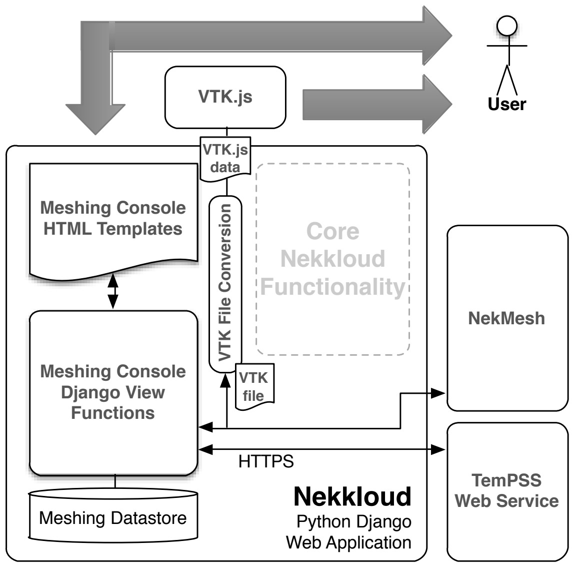
When a user has uploaded their geometry file (or selected a previously uploaded file), and completed the MCF template (see Figure 2), the meshing process can be undertaken. The green tick beside the tree's root node shows that the tree is complete and valid. Nekkloud passes the completed MCF tree to the TemPSS service which applies a transform to generate a NekMesh MCF input file. TemPSS returns a URL to the generated file which is downloaded to a temporary location on the Nekkloud server for use by NekMesh. A processing job is set up to run the NekMesh tool, passing the path to the MCF file (including the input geometry location) as input.

At present, Nekkloud runs NekMesh jobs locally on the Nekkloud server, a multi-core system. Most of the meshing code is not currently parallelised but the use of a multi-core server means that multiple meshing jobs can be run concurrently. It would, however, be possible in future to use Nekkloud's functionality for deploying large-scale computations to remote processing platforms to run NekMesh jobs.

The meshing console also provides the ability to view a generated mesh. This requires visualisation of the mesh data for which there are two core approaches: server-side or client-side. The most suitable option is dependent on the size and type of a mesh and its associated mesh data. Larger, higher-resolution meshes will result in larger file sizes. In the case of a three-dimensional mesh, the mesh file can contain either full, internal mesh structure data or only surface mesh data. Surface mesh files are significantly smaller due to the vastly reduced amount of data required when omitting the internal mesh structure for a three-dimensional model.

- **Client-side visualisation:** Places the processing requirements for visualising a mesh structure onto a local user's system. This in turn means that the resulting performance and reliability of the visualisation process is, to some extent, reliant on the hardware of a given user's system.
- **Server-side visualisation:** Uses server-side hardware, which may include specialist GPU hardware, to undertake the visualisation, streaming the resulting rendered image to the end-user's system. User interaction with a visualised model results in messages being sent from the client to the server which then enacts the changes and streams the updated display back to the user. This approach removes the need to transfer large mesh data files to an end-user's system but network latency can be an issue and there needs to be sufficient server-side capacity to handle the expected user numbers and load.

At present, the meshing console makes use of client-side visualisation via the VTK.js library [18], a JavaScript implementation of the widely used Visualisation Toolkit. VTK.js requires data to be provided in a specific format consisting of one or more gzip compressed data slices and a metadata file that acts as a directory for the data files. A tool is provided to convert a regular VTK grid file into the VTK.js file format. To visualise a generated Nektar++ XML mesh file it is therefore necessary to first convert the Nektar++ file into VTK format. This can be done using Nektar++'s *FieldConvert* tool, however, this generates a full volume mesh including internal mesh elements. The approach taken was to use NekMesh's built-in file conversion capabilities to generate a much smaller surface mesh file. The VTK.js tool *vtkDataConverter* is run on the VTK file to generate data suitable for visualisation with VTK.js. This data is placed at a temporary location from where it can be accessed via HTTP by the user's browser. When downloading the data for display, the VTK.js library requests the metadata file for a mesh, reads this and then makes HTTP requests for the associated data files in order to obtain all necessary data to display the mesh. Since visualisation is currently undertaken on linear surface meshes (even in 3D), the data sets are quite small, even for larger 3D problems. This makes client-side visualisation practical and with a good quality network connection, a user can expect to wait no more than a few seconds for the server-side file conversion and download of the data to the client to complete. Figure 3 shows the high-level structure of the meshing console architecture.



VTK.js
User
VTK.js data
Meshing Console HTML Templates
VTK File Conversion
Core Nekkloud Functionality
NekMesh
Meshing Console Django View Functions
VTK file
HTTPS
TemPSS Web Service
Meshing Datastore
Nekkloud
Python Django Web Application

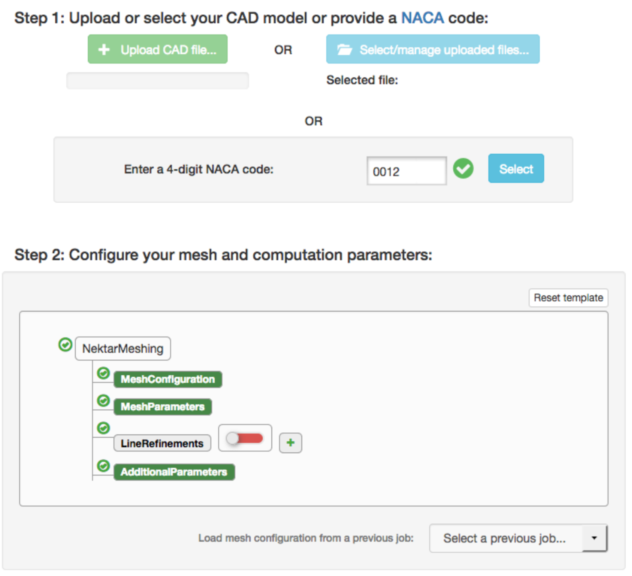
Fig. 3. Architecture of the meshing console within the wider Nekkloud application structure.

IV. User experience and community support

A. Using the meshing console

From a user perspective, the meshing console offers a straightforward interface that enables a user to go from a CAD geometry to a complete Nektar++ XML mesh file without the need to manually edit configuration files or directly run command-line tools. Users log in to the regular Nekkloud web interface and select the "Meshing" option from the menu. This opens the meshing console view through which they can generate meshes. The interface sets out the meshing process as a series of 4 steps. Steps 1 and 2, shown in Figure 4, cover the initial configuration process for the desired meshing task.

In step 1, the user has the choice of uploading a new CAD geometry file or selecting one that was previously uploaded. Clicking the "Select/manage uploaded files" option opens a panel that shows previously uploaded geometry files belonging to the current user. A further option, instead of uploading a geometry file, is to enter a NACA code. A NACA code is a 4-digit code describing one of 78 airfoil structures. The numbers in the 4-digit code describe various properties of the airfoil and an airfoil geometry can be generated by plugging these values into a specific algorithm. The NekMesh tool is able to accept a NACA code as input, in place of a CAD geometry file.


Step 1: Upload or select your CAD model or provide a NACA code:

Upload CAD file... OR Select/manage uploaded files...

Selected file:

OR

Enter a 4-digit NACA code: 0012 Select

Step 2: Configure your mesh and computation parameters:

Reset template

NektarMeshing
- MeshConfiguration
- MeshParameters
- LineRefinements
- AdditionalParameters

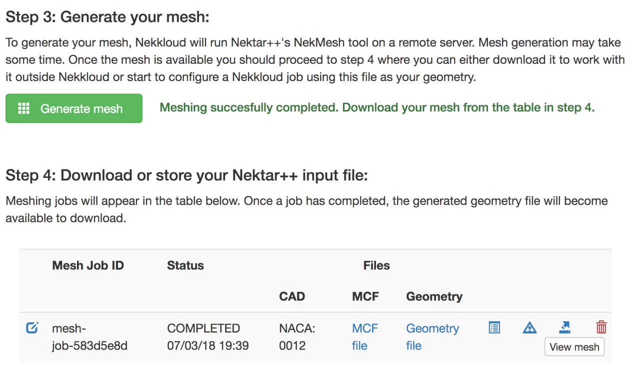
Load mesh configuration from a previous job: Select a previous job...

Fig. 4. Nekkloud screenshot showing the section of the meshing console user interface covering mesh generation steps 1 and 2.

Step 2 covers the setup of the meshing process by specifying the mesh configuration parameters. This is done by populating the TemPSS template tree. The tree shown in Figure 4 has its nodes contracted but clicking on a node expands it to show the sub-parameters (see Figure 2). Some parameters differ when using either a NACA code or a regular CAD geometry file as input. Under the `MeshingConfiguration` node, a `CADSource` element is present and this allows selection between a NACA airfoil code or an uploaded CAD file. This selection, in turn, shows any additional values that are specific to the type of input data. The drop down select box, at the bottom of the panel, labelled "*Load mesh configuration from a previous job*" lists all the current user's previous jobs. If a user is running another iteration of a meshing process for a previously used geometry, this option can be used to populate the configuration with the previous job's values and one or more previously selected values can be modified as required.

After completing steps 1 and 2, the user can request generation of the mesh in step 3 (see Figure 5). A status message alongside the "*Generate mesh*" button shows when a mesh is being generated and whether or not generation was successful. For the type of studies being undertaken by Nektar++ users, processing times are generally of the order of minutes.

Once the mesh generation process completes, an entry will appear in the table in step 4. If the meshing task was successful, the user can download the resulting Nektar++ XML


Step 3: Generate your mesh:

To generate your mesh, Nekkloud will run Nektar++'s NekMesh tool on a remote server. Mesh generation may take some time. Once the mesh is available you should proceed to step 4 where you can either download it to work with it outside Nekkloud or start to configure a Nekkloud job using this file as your geometry.

Generate mesh — Meshing succesfully completed. Download your mesh from the table in step 4.

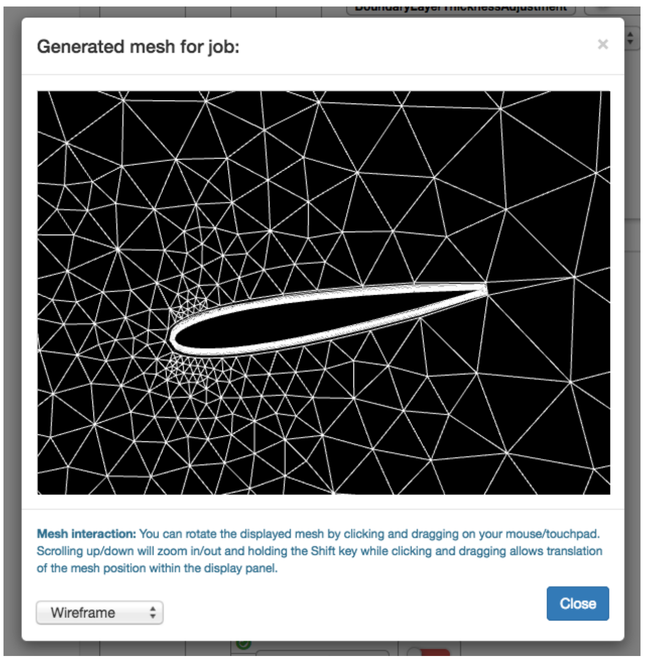
Step 4: Download or store your Nektar++ input file:

Meshing jobs will appear in the table below. Once a job has completed, the generated geometry file will become available to download.

Mesh Job ID	Status	Files			
		CAD	MCF	Geometry	
mesh-job-583d5e8d	COMPLETED 07/03/18 19:39	NACA: 0012	MCF file	Geometry file	View mesh

Fig. 5. Nekkloud screenshot showing the section of the meshing console user interface covering mesh generation steps 3 and 4.

mesh geometry file. The generated MCF file, based on the content entered into the TemPSS tree in stage 2 can also be accessed. The standard output and standard error streams are captured during the running of each NekMesh job and made available to the user to view/download. Before using the mesh in a simulation job, the user will want to undertake a visual check to see if the generated mesh is acceptable. Clicking the "*View mesh*" button undertakes the conversion of the generated Nektar++ XML mesh surface geometry file into VTK.js data for visualisation within the browser. The mesh is then displayed in a panel (see Figure 6) that allows the user to interact with the displayed mesh in two or three dimensions.


Generated mesh for job:

Mesh interaction: You can rotate the displayed mesh by clicking and dragging on your mouse/touchpad. Scrolling up/down will zoom in/out and holding the Shift key while clicking and dragging allows translation of the mesh position within the display panel.

Wireframe

Close

Fig. 6. Screenshot showing the meshing console visualisation window that allows a user to view and manipulate a mesh.

B. User community benefits

We consider that the meshing console significantly reduces the barriers to entry for users wanting to undertake mesh generation using open source tooling within the Nektar++ ecosystem. While the system is currently at a prototype stage and we do not have detailed feedback from a wide community of users, early feedback from initial test usage suggests that the system's benefits are immediately obvious and it is clear how the web-based interface offers a smoother route to generating a Nektar++ mesh, even before the ability to import the mesh straight into a new Nekkloud job is considered. As the tool is made available to a wider user base and more detailed feedback is obtained, we intend to undertake an iterative process of optimising the tool to address issues raised by users.

More broadly, the meshing console offers various potential longer-term benefits that are often key properties of science gateway-style environments. These include support for reproducibility of scientific processes and storage of a provenance trail that can be used to understand how a given state was reached and why. When a mesh is generated, the input geometry and MCF are stored (enabling re-creation of the populated template tree) alongside the mesh. This is particularly valuable for an iterative process such as mesh generation, allowing the user to build on a previous state when undertaking several iterations within a meshing process. This avoids the need to manually manage multiple input configurations, which can be error-prone, and stores each iteration of the process as a new data point. At the end of the process it is possible to see and retain details of the stages that the user went through which can simplify future jobs, make errors easier to detect and allow a given state to be straightforwardly reproduced in future.

V. Conclusions and Further Work

The Nekkloud Meshing Console provides a valuable additional element to an existing science gateway. It demonstrates the power of a user-facing web-based interface to complex scientific processes and shows how end-users, who may lack low-level computing knowledge, can benefit from the capabilities of scientific software. For the target scientific community, the meshing console represents a major usability enhancement for accessing a leading-edge tool that can otherwise be used only via a command-line. The meshing console is also considered useful to support teaching, enabling students to easily explore the effect of parameter changes on generated meshes.

We plan to undertake various pieces of further work. These include user interface enhancements to better group jobs from individual users that represent multiple iterations of a given meshing process and a plan to offer more flexibility by making job data selectively visible to different user groups, supporting skills development for new users and simplifying collaboration. We will be undertaking updates to address user feedback that is received as the user base grows and it is likely that future large-scale meshing jobs or an increase in user numbers will require a more advanced computation framework. This could be serviced either via Nekkloud's existing computation layer, or via a more traditional task-queuing system with a scalable base of worker compute nodes that could be hosted on a cloud platform. We also plan to further investigate the use of server-side visualisation for large meshes.

Acknowledgment

The authors would like to acknowledge the Nektar++ team, based at both Imperial College London and the University of Utah. JC acknowledges support from the UK Engineering and Physical Sciences Research Council (EPSRC) under grants EP/K503733/1, EP/L000407/1 and EP/R025460/1. JM's work, which has been applied in this project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 675008. We also acknowledge Prof. John Darlington, Imperial College London, who led the projects under which Nekkloud and TemPSS were originally developed.

References

[1] N. Wilkins-Diehr, "Special issue: Science gateways–common community interfaces to grid resources," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 6, pp. 743–749, 2007. [Online]. Available: \url{http://dx.doi.org/10.1002/cpe.1098}

[2] S. Gesing, J. Krüger, R. Grunzke, S. Herres-Pawlis, and A. Hoffmann, "Using science gateways for bridging the differences between research infrastructures," *Journal of Grid Computing*, vol. 14, no. 4, pp. 545–557, Dec 2016. [Online]. Available: \url{https://doi.org/10.1007/s10723-016-9385-8}

[3] S. Gesing et al., "Science gateways - leveraging modeling and simulations in hpc infrastructures via increased usability," in *International Conference on High Performance Computing Simulation (HPCS) 2015*, July 2015, pp. 19–26.

[4] V. Stodden at al., "Enhancing reproducibility for computational methods," *Science*, vol. 354, no. 6317, pp. 1240–1241, 2016. [Online]. Available: http://science.sciencemag.org/content/354/6317/1240

[5] J. Thompson, B. Soni, and N. Weatherill, *Handbook of Grid Generation*. CRC-Press, 1999.

[6] J. Cohen, D. Moxey, C. Cantwell, P. Burovskiy, J. Darlington, and S. J. Sherwin, "Nekkloud: A software environment for high-order finite element analysis on clusters and clouds," in *IEEE International Conference on Cluster Computing*. IEEE, 2013, Poster paper. DOI: http://dx.doi.org/10.1109/CLUSTER.2013.6702616.

[7] M.Turner, D. Moxey, S. Sherwin, and J.Peiró, "Automatic generation of 3D unstructured high-order curvilinear meshes," in *VII European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2016)*, 2016, pp. 428 – 443.

[8] Nektar++ User Guide, Chapter 4: NekMesh. http://doc.nektar.info/userguide/latest/user-guidech5.html. [Accessed 13th Mar 2018].

[9] C.D. Cantwell et al., "Nektar++: An open-source spectral/hp element framework," *Computer Physics Communications*, vol. 192, no. Supplement C, pp. 205 – 219, 2015.

[10] Nektar++ – Spectral/hp Element Framework. http://www.nektar.info. [Accessed 9th Mar 2018].

[11] OPEN CASCADE. https://www.opencascade.com. [Accessed 13th Mar 2018].

[12] Open CASCADE Community Edition. https://github.com/tpaviot/oce. [Accessed 13th Mar 2018].

[13] GitHub – TemPSS: A template and profile manager and editor for handling application inputs for HPC software. https://github.com/london-escience/tempss. [Accessed 9th Mar 2018].

[14] M. Pierce et al., "Apache Airavata: Design and Directions of a Science Gateway Framework," in *6th International Workshop on Science Gateways (IWSG2014)*, June 2014, pp. 48–54.

[15] Catania Science Gateway. http://www.catania-science-gateways.it/. [Accessed 13th Mar 2018].

[16] A. Merzky, O. Weidner, and S. Jha, "SAGA: A standardized access layer to heterogeneous Distributed Computing Infrastructure," *SoftwareX*, vol. 1-2, pp. 3–8, 2015.

[17] Bootstrap-Themed Tree Widget Documentation. http://jhfrench.github.io/bootstrap-tree/docs/example.html. [Accessed 30th Apr 2018].

[18] Kitware, Inc. VTK.js – The Visualization Toolkit for JavaScript. https://github.com/Kitware/vtk-js. [Accessed 15th Mar 2018].