MESH GENERATION FOR RADIOACTIVE WASTE MANAGEMENT TASKS

Butov R. A., Drobyishevsky N. I., Moiseenko E. V., Tokarev Yu. N.

Nuclear Safety Institute of the Russian Academy of Sciences, Moscow, Russia

Article received on February 07, 2021

The article describes an approach to mesh generation allowing to address radioactive waste management tasks. The approach is based on object representation as a set of simple geometric shapes (primitives). Mesh characteristics (type, size, number of elements) are controlled at a primitive level. The program allows to combine primitives using boolean operations to describe complex objects (for example, intrusions in host rock and their intersections with excavations). To describe nested objects (for example, a radioactive waste container in a borehole or a borehole in a host rock), the program implements a hierarchy of primitives. This approach can be used to generate meshes of objects with regular geometry and/or layered structures. The program is written in Python language. The mesh is created by editing the input files in JSON format.

Keywords: *radioactive waste, computer simulation, mesh generation, deep geological repository, host rock.*

Introduction

Radioactive waste management provides for waste storage in purpose-designed containers at almost all stages of the process: from waste collection to its removal from the biosphere. Despite its diversity, such containers nevertheless have quite universal shape: it can be axisymmetric cylindrical, rectangular or a combination of these shapes (Figure 1).

This relates both to the shape of the tanks themselves and the shape of relevant storage facilities (Figure 2) with the exception of some cases, such as, for example, liquid radioactive waste injection into geological formations.

Such versatility of forms subsequently resulted in a similarly universal approach to the generation of computational meshes discussed in this article.

The basic concept

Computational mesh is generated based on a set of simple elements — primitives. Each primitive is a hexagon with its faces having different shapes (Figure 3). By combining primitives and changing the shape of their faces, one can create various objects from simple items (cube, cylinder, ball) to complex multi-layer engineered structures (containers, wells, tunnels).

Of course, based on this approach one cannot accurately reproduce the shape of a real object with all its small details and inhomogeneities. However, for calculation purposes, all objects are commonly reduced to relatively simple elements, which, in turn, can be recreated using this method. Meshes made of primitives have an important advantage suggesting easy and reliable control over their quality — fine

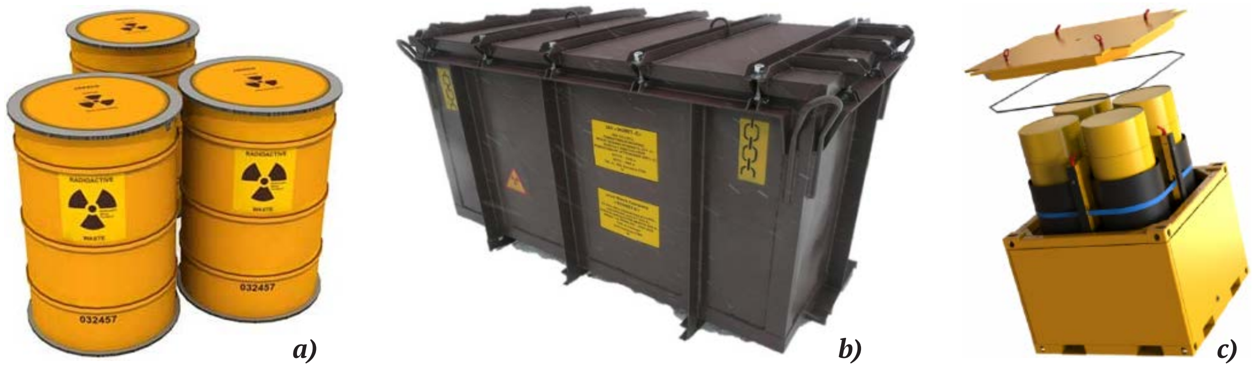

Figure 1. Forms of containers for radioactive waste: a) cylindrical, b) rectangular, c) mixed

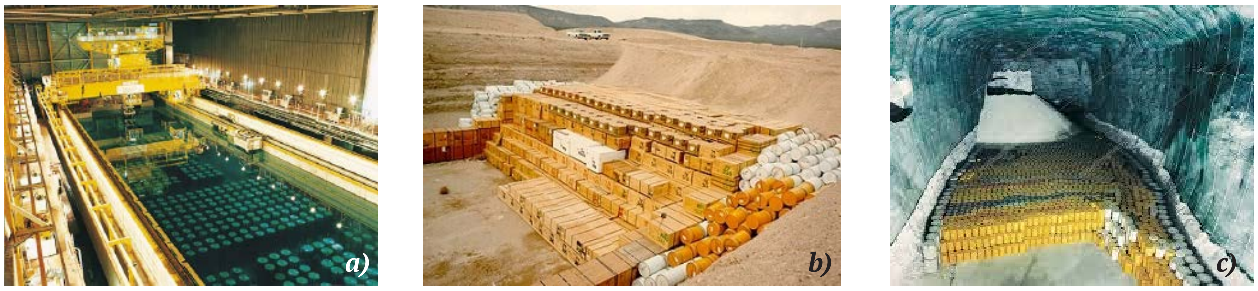

Figure 2. RW storage facilities: a) spent fuel pool, b) near-surface repository, c) deep disposal facility

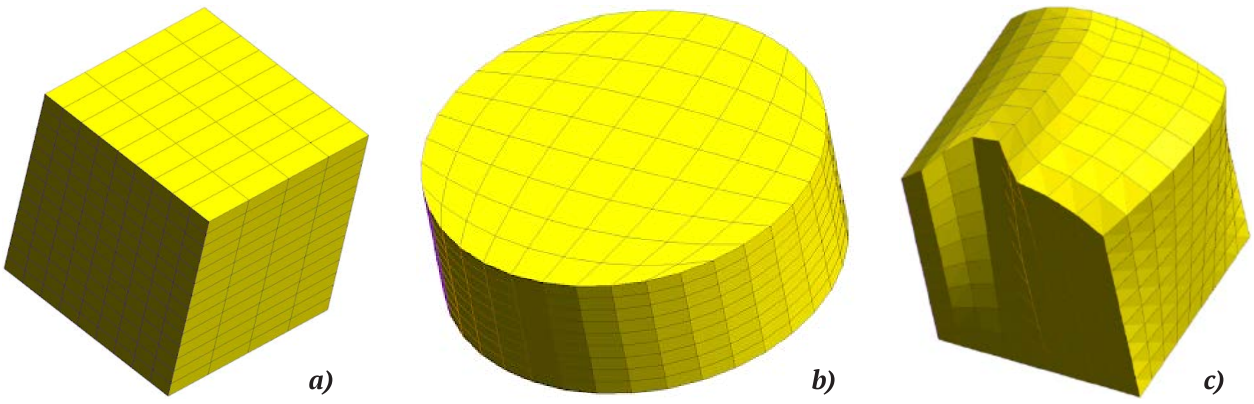

Figure 3. Primitives: a) cube, b) cylinder, c) complex shape

tuning. From a topological perspective, each primitive, despite the free shape of its faces, is a hexagon, thus, the quality of the mesh can be managed over three directions: length, width and height. If volumes of arbitrary shapes are applied, the size control is implemented immediately in all directions, which appears to be quite inconvenient, for example: when thin cylindrical shapes are simulated and it's necessary to improve the mesh quality in the wall radially, the cell size also decreases vertically resulting in an increased number of grid elements and, consequently, increased computation time and resources.

Primitives are put together into complexes usually reflecting some physical object or form (Figure 4). Such complexes, firstly, ease mesh operations since they can be used to describe it as a composition of separate large blocks. For example, in case of tasks associated with RW storage facilities, when the entire mesh actually involves many copies of the same object (for example, RW container), it's considered sufficient to reproduce one container in the form of a complex and then to expand it to the entire storage facility.

Secondly, under this approach complexes, once created under other tasks, can be used further. For example, one can use once created containers when a new storage layout is considered, or, conversely, use an old layout, but with a new type of containers. If we draw an analogy between a computational mesh and a city, then the complexes can be considered as houses and the primitives as the building blocks or bricks.

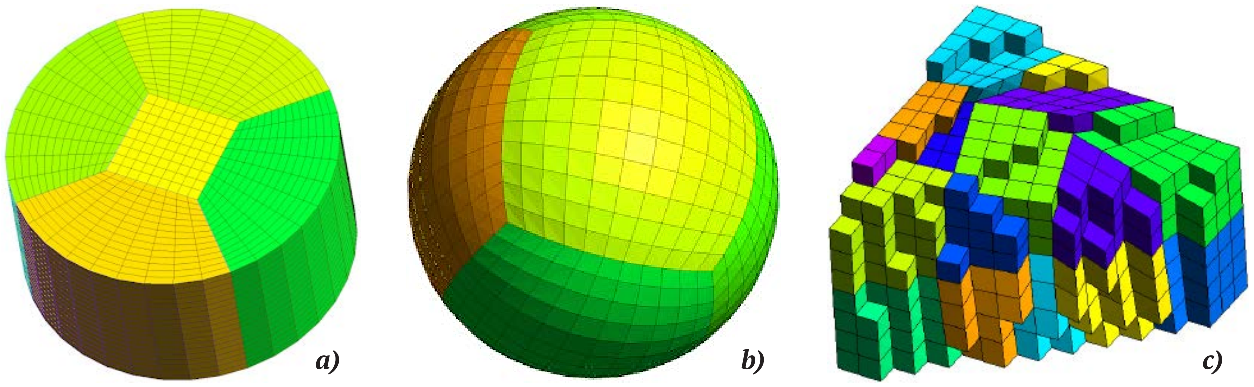

Figure 4. Complexes: a) a cylinder made of 5 primitives, b) a ball made of 7 primitives, c) cross-section of a complex shape (27 primitives)

Figure 5 presents the hierarchy of complexes. It should be emphasized that all primitives are elements of complexes that for illustration purposes were shown as a red rectangle in the diagram.

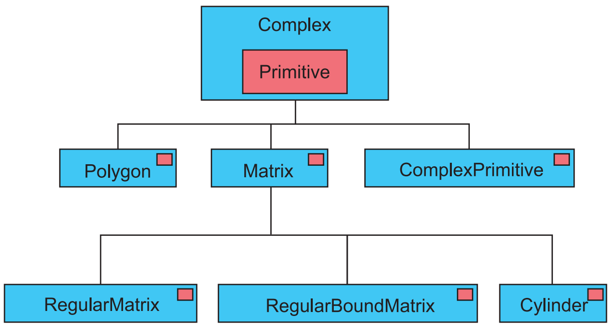

Figure 5. Hierarchy of complexes

Matrix complex is seen as a basic one. It is a three-dimensional mesh used to create computational meshes of objects with a regular structure: in the cells of this mesh one can set various materials and boundary conditions, deformations, primitives and other complexes. Since other complexes can be placed into the cells, one can generate complex nested recursive computational meshes, for example, other "Matrices" can be emplaced into each cell of the "Matrix" and etc.

Polygon complex is used to generate meshes based on a series of polygons and is applied to import some geometry from other software.

ComplexPrimitive is often required to create a complex composed of one primitive.

RegularMatrix and RegularBoundMatrix are used to create symmetrical repeating geometries such as vaults or tube bundles.

The Cylinder is used to construct multilayer cylindrical geometries (containers, drums, etc.); it can be also used to alternate cylindrical geometry with the rectangular one, for example, to accommodate a container within a cubic block (as seen in Figure 1c).

All meshes are generated using ComplexFactory allowing to standardize the process of mesh generation regardless of what complexes and primitives it is made of. An input file in json format is used as an input to the "factory". At its output, the "factory" generates a computational mesh file. Since the input files may be interconnected with other input files, the generation process will be as the one shown in Figure 6.

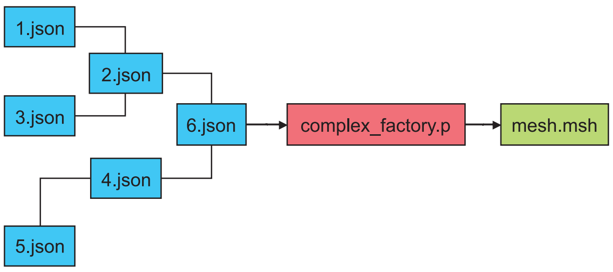

Figure 6. Mesh generation process

The main input file 6.json with possible interconnections with other input files 1-5.json at the output is fed to the input of the executable script "factory" complex_factory.py, then the mesh.msh mesh is generated by the factory.

The gmsh package [1] is used to generate computational mesh. However, even though the considered approach was based on the package capabilities, it can be basically transferred to another generator if its capabilities are similar to gmsh. To date, gmsh is de jure the most common (cited) package used by the scientific community to generate meshes (4,906 citations as of January 2021 with an average of 500 citations per year). It has a broad functionality, providing integration with the OpenCASCADE CAD system [2] and API based on languages commonly used today (C, C++, Python and Julia) and is also evolving quite dynamically. The code itself uses libraries for mesh operations: Netgen [2] and Mmg3d [3].

There are two meshing kernels available in gmsh: geo and occ. The geo kernel is a standard software core providing all functions required for mesh generation. The occ kernel is a kernel extension supplemented by OpenCascade CAD package capabilities [4]. Its main advantage is seen in its ability to use Boolean operations with volumes (Constructive solid geometry, CSG) allowing to create complex shapes from a combination of simple ones by applying logical operations.

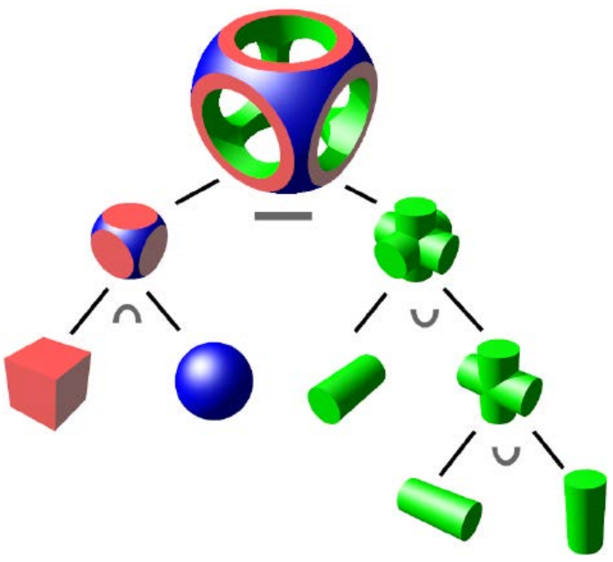

Figure 7. Boolean volume operations

Figure 7 shows the process of a complex shape formation based on a cube, a ball and cylinders using the Boolean operations of intersection, union and subtraction. Thus, for example, when the meshes are generated, intersections of RW emplacement wells with rock inhomogeneities (intrusions, crushing zones, etc.) are reproduced under the process of geological RW repository modelling.

However, this algorithm has certain disadvantages. Due to a large number of items inside the well (containers, engineered safety barriers (EBS)), the running speed of the algorithm sharply decreases since intersection operations are required for each item. After this operation is completed for one well, the number of items increases, which slows down the intersecting operation for the next well, etc. Another issue is associated with possible formation of small volumes resulting in the thickening of the computational mesh and, accordingly, a sharp increase in its size. In practice, the algorithm runs either for a limited number of wells (1-4) or for wells without an internal structure, i.e., consisting of a homogeneous material. However, its versatility (with the intersections that can be reproduced for any volumes) makes it quite helpful, for example, when it comes to intersections with volumes built in third-party software without additional preprocessing and analysis.

DDFRW modeling

Below considered is a mesh generation process for a deep RW disposal facility (DDFRW) [5], which consists of a number of wells with RW and EBS confined in them. Wells are allocated in the bedrock in an orderly manner accounting for two zones according to the corresponding RW activity level.

Three complexes are used to generate the computational mesh: Cylinder, Matrix and RegularMatrix. Cylinder is used to model wells with their axisymmetric shape, RegularMatrix is used to reproduce two zones with ordered wells (Cylinder), Matrix is used to accommodate two zones with wells (RegularBoundMatrix) in the bedrock.

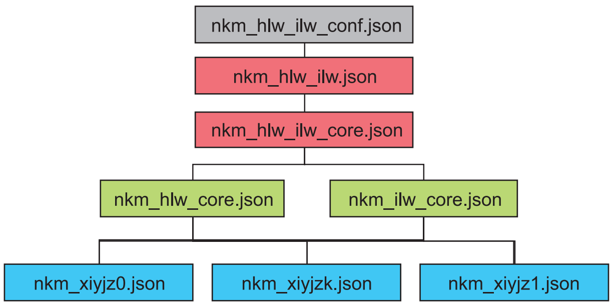

Figure 8. Structure of DDFRW input files: red – Matrix complex, green – RegularBoundMatrix, blue – Cylinder, gray – configuration file

Figure 8 presents the structure of the input files used to build the complexes. The files **nkm_ hlw_ core.json** and **nkm_ilw_core.json** are assembled from three types of input files reflecting the base of the well (**nkm_xiyjz0.json**), its top (**nkm_xiyjz1. json**) and the intermediate part with RW containers (**nkm_xiyjzk.json**), the number of which can vary, i. e. any number of RW containers can be set. In the input file **nkm_hlw_ilw_core.json**, zones with wells are oriented relative to each other. In the **nkm_hlw_ilw.json** file, zones are accommodated within the bedrock. Additionally, the upper input file of the hierarchy nkm_hlw_ilw_conf.json can be set to a configuration file with a customized complex "factory".

Figure 9 presents a computational mesh structured from tetrahedral elements. To create a hexahedral and/or unstructured mesh, one should introduce relevant changes to the root configuration file **nkm_hlw_ilw_conf.json**. In case of changes introduced to repository layout, changes are introduced to the zone files: **nkm_hlw_core.json**

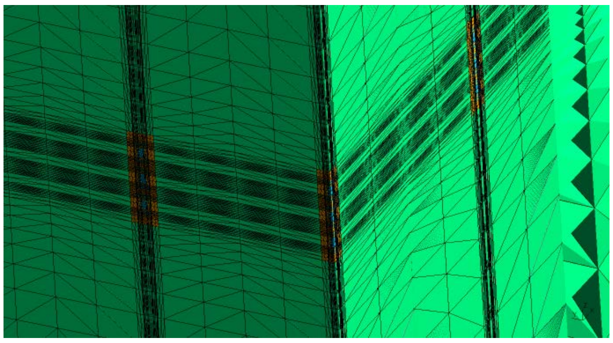

Figure 9. Cross-section of a generated mesh

and **nkm_ilw_core.json**. If changes are associated with wells and/or RW containers, changes are introduced to the files: **nkm_xiyjz0.json**, **nkm_xiyjz1.json** and **nkm_xiyjzk.json**.

This approach allows to change parameters of a computational mesh through the input file in an automatic mode, for example, in case of multivariate calculations implemented for an object when searching for its optimal layout. It also allows to import some part of a computational mesh, for example, one zone of the mesh, to another project by copying the input file associated with this zone and the input files associated with the wells. This enables effective application of already available solutions under new projects. Of course, there will be some difficulties in reconciling the parameters of old and new models, but the internal parameters reflecting the geometry of each zone remain the same.

Examples

Presented below are some examples of meshes generated for RW management calculations using the approach presented in the article.

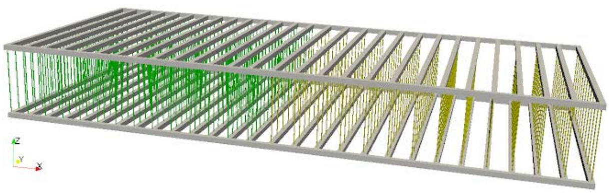

Figure 10. Possible layout of the underground DDFRW section

Figure 10 presents a DDFRW layout with relevant excavations: the wells are plotted as a structured mesh, whereas the excavations and the bedrock (not shown in the figure, fills the entire space surrounding the object) are unstructured.

Unstructured areas with poorly controlled mesh quality are generated when it appears impossible to combine several structured geometries or because of a high cost of such a combination. For example, tunnels are axisymmetric or rectangular in their cross-section, which can be reproduced as a structured one. Vertical wells have the same shape, but a combination of these objects is no longer an ordered shape and requires an unstructured insert, which complicates mesh generation making the algorithm less reliable and less controllable by the user.

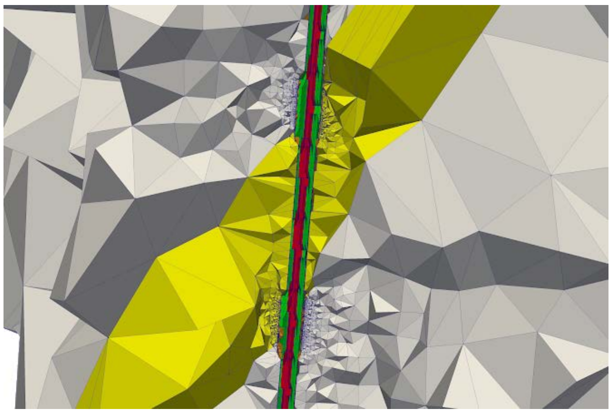

Figure 11. An example of Boolean operations: intersection of an intrusion (yellow) in a rock (gray) with an EBS-fitted well (red, green)

In case of curved geometry, such as a rock mass intrusion (Figure 11), combination of objects becomes even more difficult. Well-intrusion intersection forms a curved surface and it's considered almost impossible to set this surface in a manual way. However, even if it appears possible, there is a very limited set of parameters allowing its control.

In this case, OpenCASCA DE kernel with boolean operations on volumes is used. In fact, each volume is viewed as a primitive: in the computational mesh, it intersects any other forming new surfaces at the intersections usually having a curved shape. After that, a disordered computational mesh is created in them, the size of which is controlled indirectly, for example, along the shortest face in the volume. This greatly complicates the control over the computational mesh generation due to possible occurrence of very small volumes with a small edge length and, accordingly, a severe thickening of the computational mesh in the intersection region, which is clearly seen in the figure. With an increase in the number of intrusions and/or wells, potential occurrence of such volumes increases due to which it either becomes impossible to generate a computational mesh being adequate in its size or the available computer RAM is exceeded. Also, an increase in the number of objects leads to a significant increase in the computational complexity of the problem, which is quite difficult to estimate since after each Boolean operation, the number of objects for new operations increases.

As practice shows, this approach enables the generation of computational meshs for a limited number of wells and intrusions (up to 10) or several intrusions and hundreds of wells, nevertheless, having a primitive homogeneous environment inside.

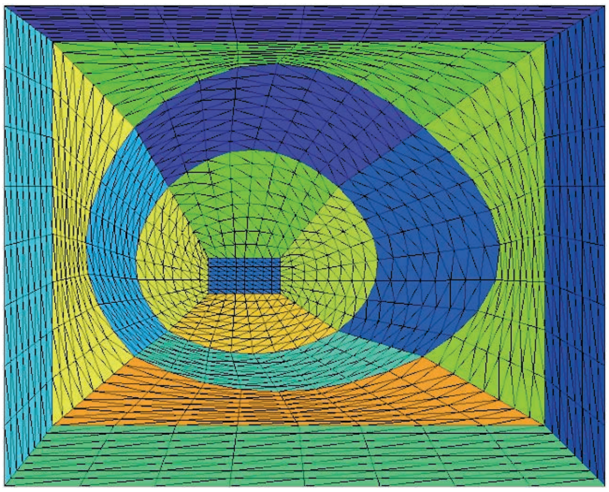
Figure 12. An example of a multilayer curved geometry based on the Matrix complex

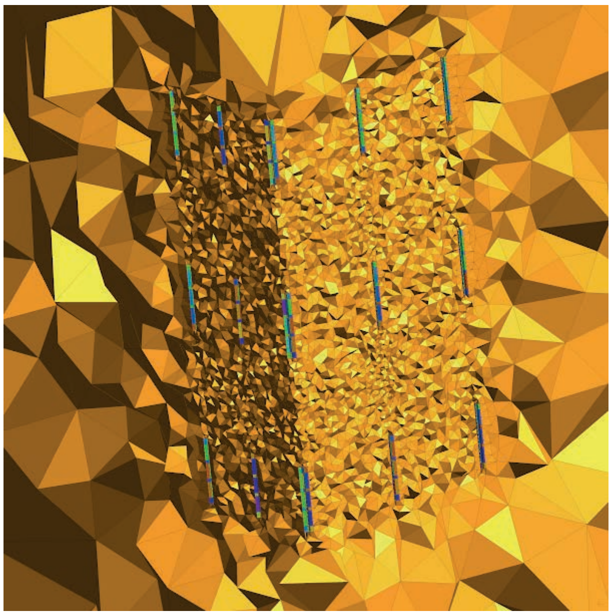
Figure 13. DDF RW wells (part)

Figure 12 shows that it's possible to generate non-axisymmetric sections for modeling purposes, for example, to simulate the imperfection in the shapes of a container or a well, provided that the computational mesh remains structured, which increases the reliability and quality control. Figure 13 provides an example of a DDFRW layout design. The wells are reproduced as a structured computational mesh, whereas the bedrock is unstructured. This results in almost uncontrollable thickening of the mesh in the area between the wells, nevertheless, allowing to accommodate the wells with different types of containers.

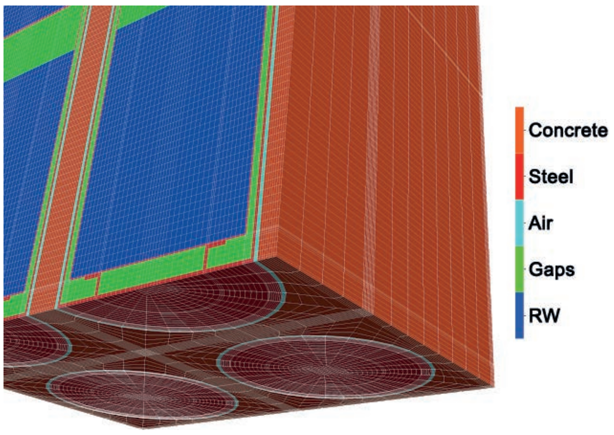

Figure 14. RW containers accommodated in a storage facility with forced cooling

Figure 14 provides an example of a cell with RW containers in a surface storage facility under forced air cooling. The cell section contains both circular and rectangular sections, while the computational mesh remains structured. Since it's possible to copy already created geometries, one can reproduce blocks of cells under an arbitrary layout.

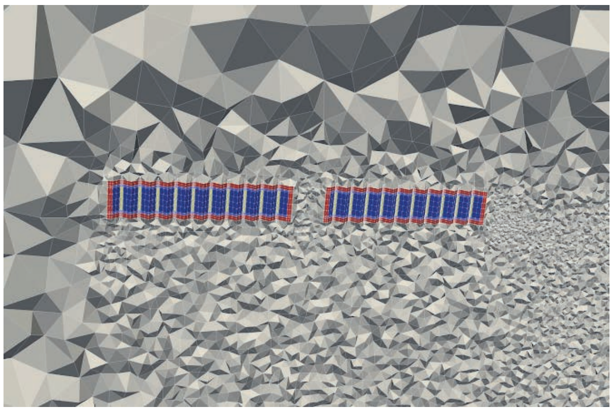
Figure 15. DDFRW (trench concept, part)

Figure 15 presents wells assuming a trench layout with a structured computational mesh inside of the trenches.

Figure 16 provides an example of geometry import from the Micromine geological package [6]. The imported geometry (orange) represents rock intrusions that intersect the envisaged wells (turquoise). The OpenCASCADE kernel with Boolean intersection operations was used to process the intersections.

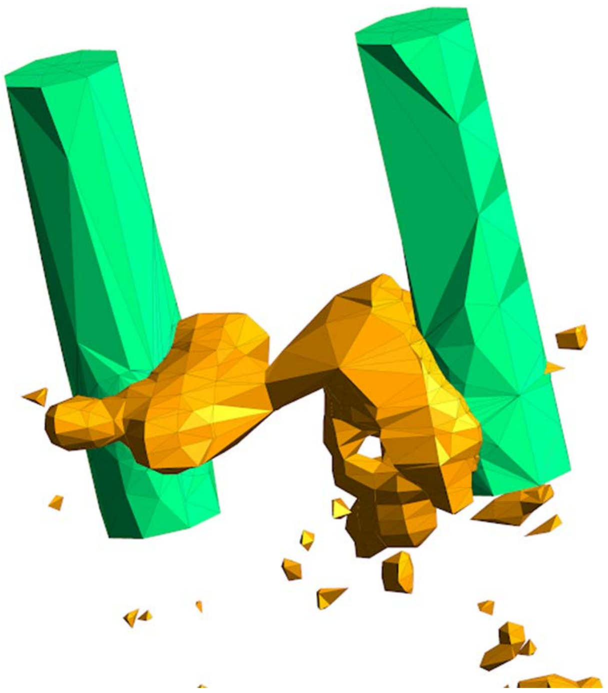

Figure 16. An example of Boolean operations applied between wells (turquoise) and intrusions in the bedrock (orange). Intrusions are imported from real geological Micromine data using the Polygon complex

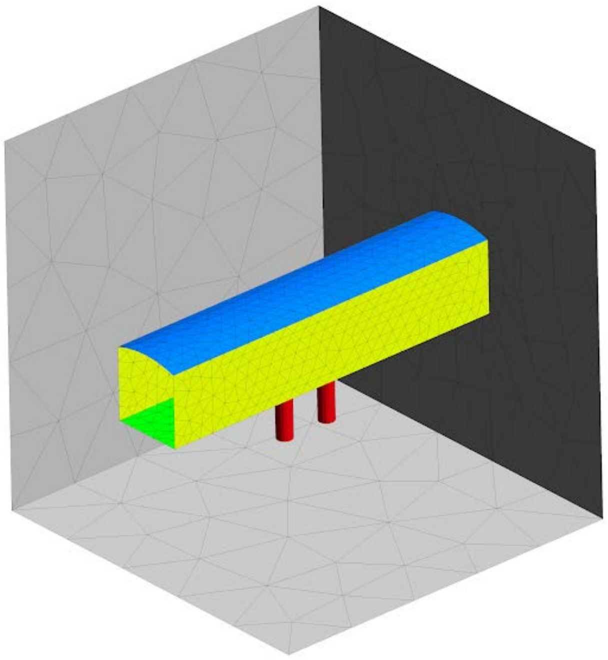

Figure 17. Computational mesh used to simulate a URL experiment: gray — bedrock, yellow — tunnel walls, green — tunnel floor, blue — tunnel ceiling, red — wells with mockup RW. Boolean operations were used for its development

The OpenCASCADE kernel was also used to model a thermomechanical URL experiment (Figure 17) [7]. Tunnel and well volumes were subtracted from the rock volume (gray). This computational mesh can be generated directly as well with all its volumes and surfaces, including intersections, being specified explicitly. However, in practice this would be a much more labor-intensive task than the automated calculations with Boolean operations applied on elementary easily set forms.

It is important that the system of boolean operations is built into the general concept of the approach with only one parameter in the upper configuration file required to be changed to allow its inclusion. For example, if we want to add an intersection with another mesh generated without Boolean operations, we don’t need to introduce any changes to their input files: all intersections will be calculated automatically.

Conclusion

The approach described allows to generate computational meshes to address RW management modeling problems. Meshing is done by editing the input files.

This approach enabled a three-dimensional calculation of DDFRW thermal mode with varying degrees of detail: for most wells, a simplified model was chosen with a detailed one build for several wells. Thus, assessment both of large-scale processes such as heating of the surrounding rock and small-scale ones (heating of container walls) can be combined under a single calculation.

For the moment, the main focus areas requiring further advancement can be summarized as follows:

1) accumulation of an input file database to address some new problems;

2) addressing the problems requiring Boolean operations with volumes (for example, intersection of bedrock intrusions and excavations);

3) import of geometries from third-party software (for example, import of real bedrock forms from purpose-developed software);

4) optimization and support of the computer code, as well as addition of new capabilities.

Despite the fact that the software has been developed and tailored to address RW management problems, this approach can be applied in addressing other problems with similar geometries, for example, such as:

1) wires, boxes, pipelines of various configurations;

2) objects with a repeated structure: foundations, tanks, warehouses, multi-storied residential buildings, supercomputers;

3) multilayer composite materials.

References

1. Christophe G., Remacle J. F. Gmsh: A 3-D finite element mesh generator with built-in pre- and

post-processing facilities. I*nternational journal for numerical methods in engineering,* 2009, vol. 79, no. 11, pp. 1309—1331. DOI: 10.1002/nme.2579.

2. Netgen Software [Electronic source]. — URL: https://ngsolve.org/ (accessed on 14.01.2021).
3. mmg3d Software [Electronic source]. — URL: https://www.mmgtools.org/ (accessed on 14.01.2021).
4. OpenCASCADE Software [Electronic source]. — URL: https://www.opencascade.com/ (accessed on 14.01.2021).
5. Drobyshevskiy N. I., Moiseenko E. V., Butov R. A., Tokarev Yu. N. Trekhmernoe chislennoe modelirovanie teplovogo sostoyaniya punkta glubinnogo zahoroneniya radioaktivnyh othodov v Nizhnekanskom massive gornyh porod [Three-dimensional numerical modelling of the thermal state of the deep radioactive waste disposal facility in the Nizhnekansk granitoid massif]. *Radioaktivnye otkhody — Radioactive Waste,* 2017, no. 1, pp. 66—75.
6. Micromine Software [Electronic source]. — URL: https://www.micromine.com/ (accessed on 14.01.2021).
7. Moiseenko E. V., Drobyshevskiy N. I., Butov R. A., Tokarev Yu. N. Kontseptsiya provedeniya krupnomasshtabnykh termomekhanicheskikh ehksperimentov v PIL v Nizhnekanskom massive gornykh porod [Concept of Large-Scale Thermomechanical Url Experiments in the Nizhnekanskiy Rock Massif]. *Radioaktivnye otkhody — Radioactive Waste,* 2020, no. 3, pp. 101—111.

Information about the authors

Butov Roman Aleksandrovich, Engineer, Nuclear Safety Institute of the Russian Academy of Sciences (52, Bolshaya Tulskaya st., Moscow, 115191, Russia), e-mail: bra@ibrae.ac.ru.

Drobyishevsky Nikolay Ivanovich, PhD, Senior Researcher, Nuclear Safety Institute of the Russian Academy of Sciences (52, Bolshaya Tulskaya st., Moscow, 115191, Russia), e-mail: dni@ ibrae.ac.ru.

Moiseenko Evgeny Viktorovich, PhD, Senior Researcher, Nuclear Safety Institute of the Russian Academy of Sciences (52, Bolshaya Tulskaya st., Moscow, 115191, Russia), e-mail: moi@ibrae.ac.ru.

Tokarev Yuriy Nikolaevich, PhD, Senior Researcher, Nuclear Safety Institute of the Russian Academy of Sciences (52, Bolshaya Tulskaya st., Moscow, 115191, Russia), e-mail: ytokarev@ya.ru.

Bibliographic description

Butov R. A., Drobyishevsky N. I., Moiseenko E. V, Tokarev Y. N. Mesh generation for radioactive waste management tasks. *Radioactive waste,* 2021, no. 1 (14), pp. xx—xx. DOI: 10.25283/2587-9707-2021-1-xx-xx. (In Russian).