SlabCutOpt Algorithm

S. Bonduà, S. Focaccia, M. Elkarmoty

Version 1.0 – 2024

Summary

1 Software description

The **SlabCutOpt** software is coded in C++. It generates a hypothetical 3D cutting grid in the specified domain. The 3D cutting grid is then rotated (3D rotation) and translated (in the 3 Cartesian direction) several times. For each generated grid each block, the software tests if it intersects a fracture. The intersection of the block with a fracture (or boundaries) is performed using a segment/triangle intersection algorithm[1]. So for each block, that is composed of 12 edge segments, the algorithm checks for intersections between the edges of the block and the triangles composing a fracture. The use of triangles for fracture representation allows the approximation of using planes for representing the fractures, as well as the possibility of representing the fracture as discrete surfaces, with holes and complicated shapes. In the output file, a report of each cutting grid stores the geometrical parameters used to generate the blocks (rotation and translation), the number of the blocks inside the domain, the number of not intersected blocks. Optionally the algorithm can operate in 2D, avoiding testing for the z dimension. The user can define the space of solutions to be examined, by defining in the ASCII input file the parameter of rotation and translation, the block dimensions and output options.

2 SlabCutOpt Input files

The input data of the SlabCutOptsoftware is composed of several files:

- SlabCutOpt.par: it is the ASCII file containing the input parameters of the model. It must contain: the geometrical parameter of the domain to be investigated, the block dimensions, the orientation and translation parameters and the options about the operating mode, the output mode, etc. See Appendix A for further details;
- PLY_FileList.dat: it is an ASCII file containing the ply file list of the fracture set. See Appendix B for further details;
- Ply files: each fracture or fracture set has to be stored in ply file format. See Appendix C for details;
- borders.dat (optional): it is an ASCII file containing the boundary of the domain, as a set of 2D points in anticlockwise order. Note that this function operates only in 2D, by defining a closed shape with vertical walls. See Appendix D for details.

3 SlabCutOpt output files

The standard outputs of SlabCutOpt are ASCII files with information about the cutting grid results run.

Specifically, the outputs of the SlabCutOpt software are:

- Results.log file. It is an ASCII file, where each line contains: the orientation of the cutting grid, the translation used, the number of blocks inside the domain, the number of blocks with non-intersection (recoverable blocks). See Appendix E for additional details.
- blocks vtu files: for each generated cutting grid, a vtu file is written to allow 3D visualization. A rocktype color is assigned to each block to allow for an easy identification of blocks inside the domain, intersected blocks and no-intersected blocks. See Appendix F for details.
- Bounds.ply: it is the PLY file of the model input domain.

4 APPENDICES

In the following appendices the input/output files of SlabCutOpt are briefly presented and described. For each file, an excerpt of the file of the case study is shown.

4.1 Appendix A – Parameter File 3D_FracturedCuttingBlocksOptimization.par

The 3D_FracturedCuttingBlocksOptimization.par contains directive of working mode, bounding domain, output options of 3D_FracturedCuttingBlocksOptimization. Keywords are briefly resumed in the given example file.

```
#SlabCutOpt Parameter file
#
# Block Domain definition
x_max=+0.9 #
x_min=+0.0 #
y_max=+2.0 #
y_min=+0.0 #
z_max=+1.1 #
z_min=+0.0 #
#Cutting block dimensions
dim_block_x=0.41 # Distance between cutting planes in x direction
dim_block_y=0.21 # Distance between cutting planes in y direction
dim_block_z=0.04 # Distance between cutting planes in z direction
cut_saw_thickness=0.01 # Saw thickness
#Space of solutions
#angles (radiant)
tetha_step=1.570796327 #
tetha_max=1.570796328 #
phi_step=1.570796327 #
phi_max=1.570796328 #
csi_step=1.570796327 #
csi_max=1.570796328 #
#displacement(meters)
dx_max=.20
dy_max=.20
dz_max=.20
dx_step=0.05 #
dy_step=0.05 #
dz_step=0.05 #
#Options
read_bound=0 #
rotation_method=1 #0: Euler rotation; 1: Cartesian axis rotation
BiDimensional=0 #
write_vtu=1 #0: no vtu file in output;#1: for each solution, a block.vtu file is generated
read_PLY_FileList=1 #0: Read the fracture set from "test_fractures.ply"; 1 read the PLY_FileList.dat for using several fracture file.
end=end
```

4.2 Appendix B – PLY_FileList.dat input file.

The PLY_FileList.dat input file must contain the list of the ply file where the fracture sets are stored. No header is allowed.

```
block_fracturepink_f.ply
```

block_fracturepink_g.ply

block_fracturepink_k.ply

brown_frac.ply

extended_void_blue.ply

purbale_frac.ply

red_frac.ply

single_void.ply

terquaze_frac.ply

yellow_frac.ply

4.3 Appendix C – PLY file example

The PLY file used by the SlabCutOptsoftware is a fixed simplified version of the more general and flexible PLY file format. Each element must have 3 vertexes (a triangle in the 3D space). In italic the fixed characters, in bold the geometrical parameters for fracture definition. After the header, a number of file specified in [*element vertex*] indicates the vertex coordinates, then the following [*element face*] lines define the number of vertex [*3*] and the vertexes index of the triangle. Note that in the more general PLY file format, the number of vertexes can be higher for polygonal elements.

```
ply
format ascii 1.0
comment example of a fracture represented by 2 triangles
element vertex 4
property float x
property float y
property float z
element face 2
property list uchar int vertex_index
end_header
0.000 2.000 0.553
0.450 2.000 0.553
0.000 0.000 0.583
0.450 0.000 0.599
3 0 1 2
3 1 2 3
```

4.4 Appendix D – border file description

The 2D quarries domain is not, in general, a rectangular shape. For this reason the user can define an irregular shape domain, giving the 2D coordinates of the boundaries. The file must contain in the first line the number of vertexes of the polyline and in the following lines the coordinate of each vertex, in anticlockwise order. The last vertex will close the polyline with the segment connected to the first vertex.

```
4
 25397.77196 5007.604247
 31070.80449 5007.604247
 31070.80449 10723.88741
```

25397.77196 10723.88741

4.5 Appendix E – Result file description

The result file is an ASCII file containing the output information about the cutting grid optimization.

In the first line the parameter used as input are summarized.

The following lines contain information about rotation and translation, displacement and number of blocks inside the domain, number of non-intersected blocks.

```
Optimization results:
tetha_step=1.570796
phi_step=1.570796
phi_max=1.570796
dx_step=0.050000
dy_step=0.050000
dz_step=0.050000
dim_block_x=0.400000
dim_block_y=0.200000
dim_block_z=0.020000
n_triangles=0.000000
read_bound=0
BiDimensional=0
write_vtu=1
read_PLY_FileList=1

N 0 Tetha 0.000000 Phi 0.000000 Csi 0.000000 dx -0.100000 dy -0.100000 dz -0.100000 n_block_inside=486
n_block_no_intersect=375
N 1 Tetha 0.000000 Phi 0.000000 Csi 0.000000 dx -0.100000 dy -0.100000 dz -0.050000 n_block_inside=495
n_block_no_intersect=384
N 2 Tetha 0.000000 Phi 0.000000 Csi 0.000000 dx -0.100000 dy -0.100000 dz 0.000000 n_block_inside=486
n_block_no_intersect=375
……
```

4.6 Appendix F – blocks.vtu file description

The SlabCutOptsoftware writes a set of vtu files for visualization. The vtu file format can be read by using Paraview. For an exhaustive description of the vtu file format see **www.paraview.com** web pages.

The rocktype values used for coding blocks are explained in Table 1:

Rocktype value	Description
-2	Blocks outside of the domain
1	No-intersected block
2	Intersected blocks

Table 1: Rocktype code description

```
<?xml version="1.0"?>
<VTKFile type="UnstructuredGrid" version="0.1" byte_order="LittleEndian"
compressor="vtkZLibDataCompressor">
  <UnstructuredGrid>
    <Piece NumberOfPoints="30464" NumberOfCells="3808">
      <PointData>
      </PointData>
      <CellData Scalars="myTYPES">
        <DataArray type="Float32" Name="RockTypes" format="ascii"
RangeMin="0" RangeMax="2">
        -2 -2 -2 -2
...
         </DataArray>
      </CellData>
      <Points>
        <DataArray type="Float32" Name="Points" NumberOfComponents="3"
format="ascii" RangeMin="0" RangeMax="3.483490">
          -0.445000 -0.495000 -0.225000
...
</DataArray>
      </Points>
      <Cells>
        <DataArray type="Int32" Name="connectivity" format="ascii"
RangeMin="0" RangeMax="30464">
          0 1 2 3 4 5
...
<DataArray type="Int32" Name="offsets" format="ascii" RangeMin="8"
RangeMax="30464">
           8 16 24 32 40 48
...
        </DataArray>
        <DataArray type="UInt8" Name="types" format="ascii" RangeMin="42"
RangeMax="42">
          42 42 42 42 42 42
...
       </DataArray>
        <DataArray type="Int32" Name="faces" format="ascii" RangeMin="0"
RangeMax="30463">
          6
          4 0 3 2 1
          4 2 6 5 1
          4 4 5 6 7
          4 0 4 7 3
          4 2 3 7 6
          4 0 1 5 4
…
        </DataArray>
        <DataArray type="Int32" Name="faceoffsets" format="ascii"
RangeMin="31" RangeMax="118048">
          31 62 93 124 155 186
...
        </DataArray>
      </Cells>
    </Piece>
```

```
  </UnstructuredGrid>
</VTKFile>
```

Vtu file example

1. Guigue P, Devillers O. Fast and Robust Triangle-Triangle Overlap Test Using Orientation Predicates. *Journal of Graphics Tools.* 2003;8(1):25-32.