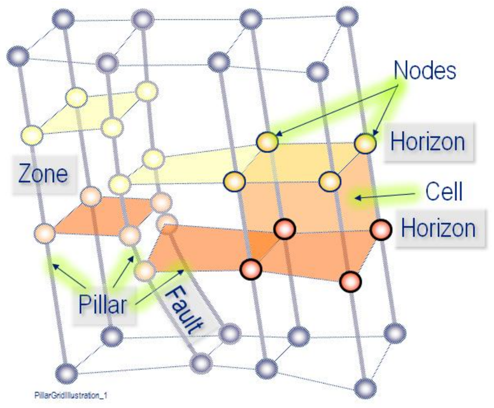
Efficiently Converting a geobody defined in a regular 3D grid to a irregular model 3D grid

A frequently performed task in seismic geobody interpretation is to upscale a geobody (defined by segmenting certain seismic areas) to a 3D model. A geobody is defined in a regular grid where a certain value in the grid correspond to a patch of the geobody. A IxJxK grid with a certain resolution (often the same as the seismic post-stack resolution) overlapping with the seismic data being analyzed is defined. In this grid, cells corresponding to known or predicted geological properties can be labeled. For instance, cells predicted or corresponding to sand can be value 1, shale as 2, carbonate as 3 etc. The value 0 is usually mapped to a transparent undefined property.



A 3D Grid

A 3D model is defined as a irregular grid where cells are defined between 8 corners. Size and shape of a cell can vary in each direction and there are no guarantee that any of corners in a cell shares the same location in x, y or z coordinates.

In addition, the number of cells and cell resolution is can be very different between the 3D model and the geobody. One geobody cell could for instance contain many model cells and vice versa.

The task is to efficiently map a geobody with its' regular voxels into a model with irregular cells. implemented methods can do the following (although not very efficiently):

1. For each voxel, check if the voxel center is inside a gid cell.

or

2. For each grid cell, check if it is inside a geobody voxel .

In addition, an option to check if each of the corners of the grid cell is inside a voxel slightly redeems the situation where the regular geobody grid differs a lot from the 3D model grid.

While these two methods gives the user the possibility to slightly control the error when geobody voxels and grid cells have different sizes, the results are often poor and time consuming. The optimal algorithm for this problem would be to check for each cell in the model if the intersection between the cells volume and each grid volume is > 0, and if so include the cell in the result. The size of most 3D model and geobody grids, and the limitation of computer memory has so far prevented this algorithm from being possible, but a possibility is to first map the model and geobody into a coarse grid and check for intersection and then focus on smaller and smaller pieces until one reach single cells.

Input to the algorithm is a Ocean Petrel Grid Class (Perhaps we need to wrap this for external use) and a geobody with byte[,,] where each voxel has a location (x,y,z) point in space and every voxel have equal size.

Another important thing that should be mentioned is that sub-cubing is used to achieve scalability. This means that when the seismic grid (inherited by the geobody) is very big it is cached in memory by sub-grids. The same applies for big 3D models. This should not introduce any limitations to the algorithm except that cells far apart in the 3D model or geobody should not be frequently accessed.

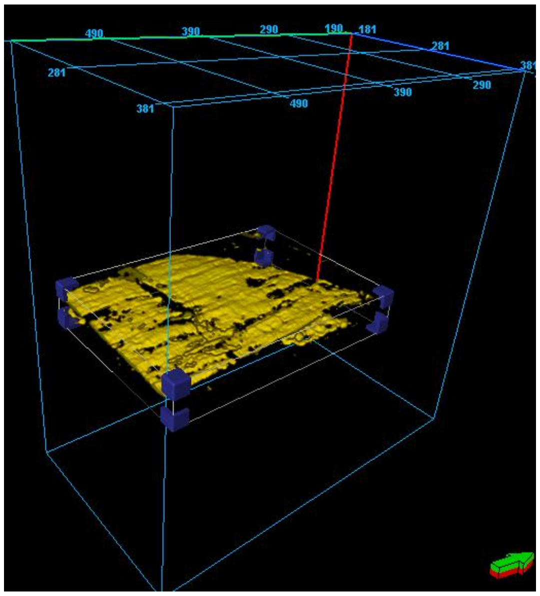


Figure 1 Geobody to be segmented

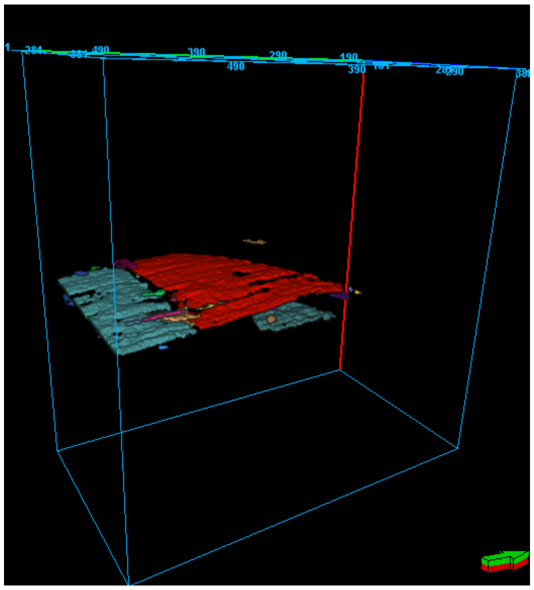


Figure 2 Segmented geobody (multiple patches)

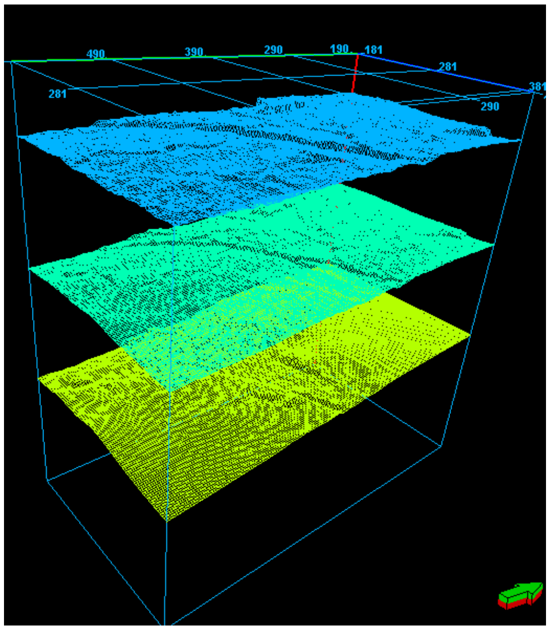


Figure 3 3D model defined between upper and lower horizon

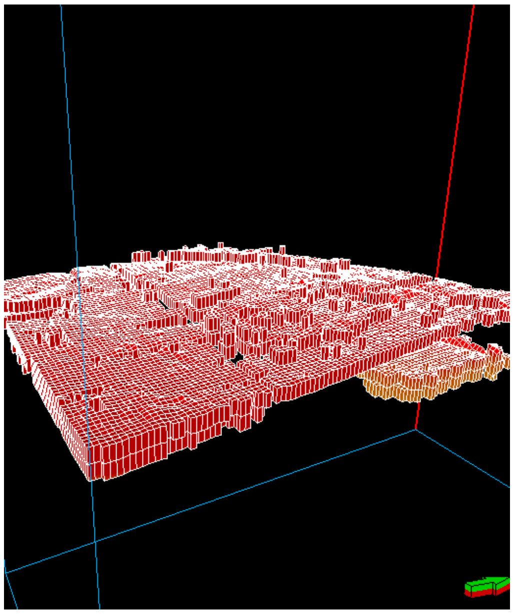


Figure 4 Segmented geobody upscaled to model

Programming interface

What we have for a geobody grid is typically:

- Grid location (center) (X,Y,Z)
- Grid orientation(4x4 Matrix)
- Grid Extent (X,Y,Z)
- number of cells (I,J,K)

What we have for a 3D model is:

- GetCellCenter (I,J,K)
- GetCellAtPoint (X,Y,Z)
- GetPointAtCell(I,J,K)
- GetCellCorners (CellCorner[])
- NumCellsIJK (I,J,K)
- ++