VISUALIZATION OF STRESS TENSOR FIELD USING CUTTING PLANE TECHNIQUE (IMPLEMENTED IN VISUALIZATION TOOLKIT (VTK))

Sugeng Waluyo

Balai Besar Teknologi Kekuatan Struktur (B2TKS) Badan Pengkajian dan Penerapan Teknologi (BPPT)
Kawasan Puspiptek Serpong Gd. 220 Telepon (021) 7560562 ekst. 1063
E-mail : sugeng_waluyo@webmail.bppt.go.id, sugeng_walj@yahoo.com

Abstrak

One of the most challenging tasks in scientific visualization is how to visualize 3-D stress tensor field in intuitive and uncluttered images. Many techniques e.g., glyphing and hyperstreamline, have been developed to visualize the stress tensor field, however, they can be used only for specific applications. Within this work, another technique called cutting plane will be used and implemented in Visualization Toolkit (VTK). The idea behind development of tensor visualization using cutting plane comes from the functionality of vtkCutter class in VTK library which cuts a 3-D object by reducing a 3-D cell to a cut surface. Then, by using data set attribute at each point on the cut surface, i.e., stress tensor, and manipulating it to obtain a traction vector on the cut surface, normal stress and shear stress with respect to cut surface should be easily computed.

Keywords: scientific visualization, stress tensor, cutting plane, VTK

INTRODUCTION

In general, visualization is understood as a method to visually represent information content within a set of large scale multidimensional data. It offers capability to see the unseen information inside a data set. For science and engineering purpose, this is known as scientific visualizations. Recently, scientific visualizations are already revolutioning the way of analysis and design because they are cheaper and more understandable than an experiment in many cases.

Meanwhile, a trend in computational solid mechanics is that simulations are more entirely performed with volume-oriented continuum models. For finite element method (FEM), it means that the structure are not longer described by dimensionally reduced system, like beam or plate models, however, they are discretized by three dimension (3-D) element, e.g., hexahedral and tetrahedral elements.

One of the most challenging tasks in scientific visualization is how to visualize 3-D stress tensor field in intuitive and uncluttered images. Many techniques e.g., glyphing and hyperstreamline, have been developed to visualize the stress tensor field, however, they can be used only for specific applications. Within this project, another technique called *cutting plane* will be used and implemented in Visualization Toolkit (VTK), i.e., an open source, freely available software system for 3-D computer graphics, image processing, and visualization based on C++ class library.

The idea behind development of tensor visualization using *cutting plane* comes from the functionality of *vtkCutter* class in VTK library which cuts a 3-D object by reducing a 3-D cell to a cut surface (see Fig.1). Then, by simply using data set attribute at each point on the cut surface, i.e., stress tensor, and manipulating them to obtain a traction vector on the cut surface, normal stress and shear stress with respect to cut surface should be computed easily (see Fig.2).

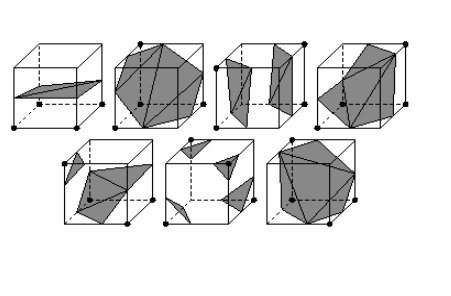


Figure 1. The cutting processes using marching cube show several possibilities to obtain cut surfaces from 3-D objects numerically

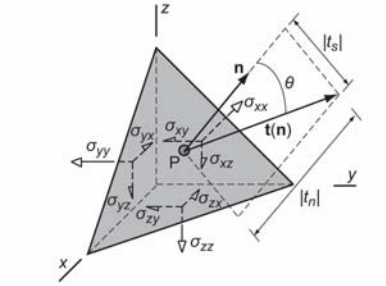


Figure 2. Diagram of traction vector $\mathbf{t}(\mathbf{n})$ and stress state σ_{ij} at point P which is located on the specific surface. Here, subscripts *i* and *j* are defined by x, y, and z

VISUALIZATION OF STRESS TENSOR FIELD

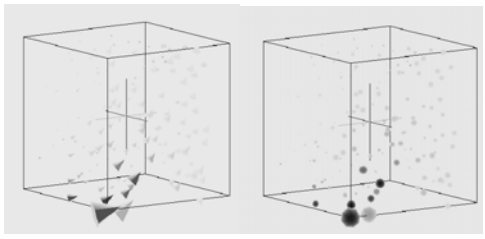
In solid mechanics, a stress tensor is commonly described as a matrix

$$\boldsymbol{\sigma} = \sigma_{ij} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix} \quad (1)$$

where subscripts *x*, *y*, and *z* are oriented based on user defined coordinate system (see Fig. 2) and subscript *i* and *j* are tensor components. Using this matrix, one can obtain the invariants, eigenvalues, and eigenvectors of it. These are properties of the stress tensor in (1) which are invariant under a change of coordinate system.

Nowadays, at least two methods are commonly used to visualize tensor, i.e., glyph and hyperstreamline. Glyphing is a visualization technique which represents data by using geometric symbols like oriented cones or spheres to show principle stresses in 3-D (see Figs. 3a and 3b). Here, for each tensor matrix, the eigenvalues and associated eigenvectors are sorted to determine the major, medium, and minor eigenvalues and eigenvectors. In case of 2nd order tensor, those values will be three axes of ellipsoid. The concept of hyperstreamline is quite similar to glyphing. While glyphing visualizes the eigenvalues and eigenvectors at discrete points, hyperstreamline connects all of them through points and visualizes them similar to conventional streamline used to describe vector field (see Fig. 4).

Either glyphing or hyperstreamline technique was developed in order to visualize general 2nd order tensor field, e.g., magnetic field, gravity field and stress in fluid, which is not commonly used in solid mechanics. Solid mechanics community usually work with stress distribution over a length or an area and the corresponding force on that length or area. They normally require information of stress distribution only for specific locations of a model not for overall model. Therefore, in this work, a new approach called cutting plane technique is introduced as following.



(a) (b)

Figure 3. Glyphing technique using (a) sphere and (b) cone geometric symbol representation

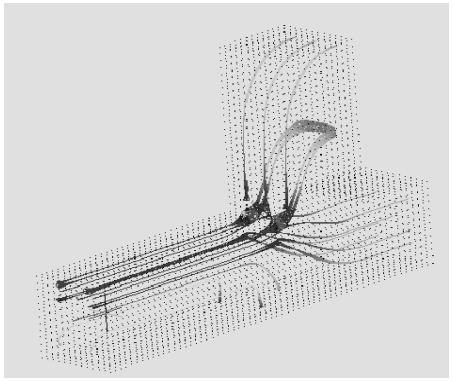


Figure 4. Hyperstreamline

Assuming that a solid model is built by a volume inside boundary surfaces, the cutting plane will cut the model along the volume using implicit function

$$F(x, y, z) = A \quad (2)$$

where A is a constant determining the cutting position with respect to 3-D volume as shown in Figure 5 as parallel lines. Numerical implementation of analytical approach in (2) can be done by approximating the analytical volume in Figure 5 with volume of small, but finite, cubes.

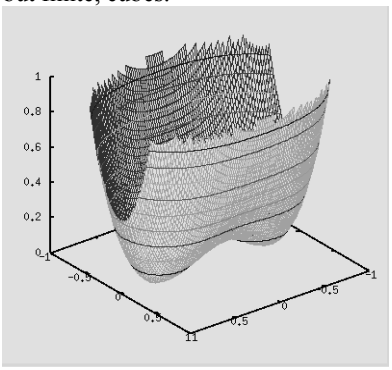


Figure 5. Implicit function $F(x, y, z) = A$ for arbitrary value of A is shown by parallel lines

Then, by checking the possibility of the cutting planes cut the single cube as shown in Figure 1, one can built 2-D planes through a 3-D volume or a solid model.

ISBN : 978-979-3980-15-7
Yogyakarta, 22 November 2008

IMPLEMENTATION OF THE CUTTING PLANE TECHNIQUE IN VTK

VTK is an open source, freely available software system for 3-D computer graphics, image processing, and visualization used by thousands of researchers and developers around the world. VTK consists of C++ class library, and several interpreted interface layers including Tcl/Tk, Java, and Python.
Based on guidances and rules on how to extend VTK classes efficiently, this work will be done only in Microsoft Visual C++ 6.0 as programming language. The advantage of using C++ as development language will typically result in smaller, faster, and more easily deployed applications than most other language. A development using C++ also has an advantage that one does not need to compile any additional support for Tcl, Java, or Python.

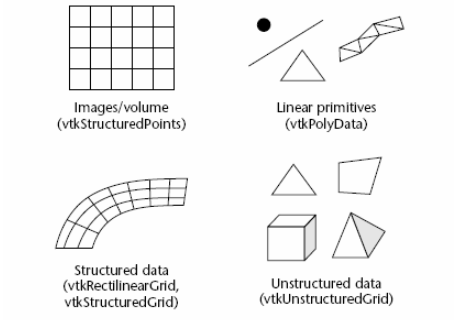


Figure 6. Type of data set supported by VTK (see VTK User's Guide)

Besides the cube geometry, VTK supports several geometry data which is commonly used in visualization (see Fig. 6) as well. Except a single grid data, each of data consists of a network of lines which connects two grid points. In practice, programmers put data attribute information, e.g., coordinate, velocity, temperature, and stresses in the grid points.

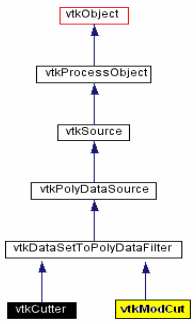


Figure 7. Hierarchy of the vtkModCut (see VTK User's Guide)

Using interpolation, the value of data attribute between grid points can be approximated. This idea is used for transferring data attribute from its original location onto cutting plane.

To start with, a new class library is proposed which represents the cutting plane. This class is called *vtkModCut* and derived directly from *vtkDataSetToPolyDataFilter class* which has capability to change the data set to polygonal data. The *vtkDataSetToPolyDataFilter* class has parent classes with their own functionality (see Fig. 7). It is obvious from Figure 7 that *vtkModCut* is independent from existing cutting class in VTK called *vtkCutter* though it is a modification of *vtkCutter*.
Furthermore, a stress tensor can be represented onto a surface by a traction vector $\mathbf{t(n)}$ simply by multiplication of stress tensor matrix σ_{ij} and normal vector $\mathbf{n}$ of the surface as

$$T_i = \sigma_{ij} n_i . \qquad (3)$$

Hence, the normal stress vector $\boldsymbol{\sigma}$ on the surface is computed by dot product of $\mathbf{t(n)}$ and $\mathbf{n}$ (see Fig. 8). Then, the shear stress vector $\boldsymbol{\tau}$ is normally computed by simply projecting of the traction vector onto the surface. However, those approach are difficult to be implemented here for arbitrary surface form because, in VTK, tensor is very often associated as a data attribute of point (see Fig. 8) while normal vector is a property of a surface.

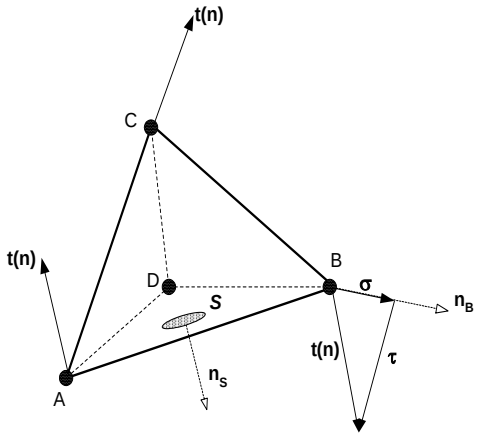


Figure 8. In VTK, a normal vector is defined directly at a surface e.g., $\mathbf{n_S}$ at surface S, and tensor attributes is a property of point or grid, e.g., point or grid B

There are two methods which are normally used to solve the problem above. First, a normal vector at a point, e.g., shown as $\mathbf{n_B}$ in Figure 8, is defined as average of normal vectors of cells around that point. Second, cell data attribute, i.e., stress tensor is defined as average values of stress tensor at points around the cells. At the first glance, the second method is easier because, in VTK, there is a function which returns number of point belongs to a cell. Unfortunately, this method leads to bad visualization result, i.e., uniform colour in a cell, when direct colouring method is applied during rendering because VTK considers only

interpolation of colour among points not cells. Whereas, the first method, which is used in this work, should be a better choice as long as one knows how many cells belong to a point.

Instead of walking through points to find their associated cells, Figure 10 shows an algorithm which walks through cells and collects normal vector of the cells which shares exactly the same points. Thus, the normal vector at the point is computed by summation of normal vector at the point collected from each appropriate cells divided by total number of cells associated with that point which is indicated by the counter (see Fig. 10). For example, assuming a point with ID=1 belongs to four cells around it. Then, normal vector at that point is given by summation of each component of normal vectors of that four cells divided by four. The algorithm has been implemented in *vtkModCut* class

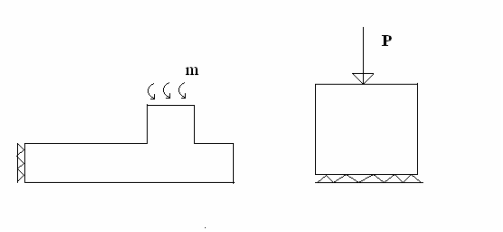
```
for (cellId 0 to number of cell)
{
    1. Get Point ID for each cell
    2. Compute normal vector of cell
    3. Associate normal to each point
    4. Sum up normal for each point
       with same point ID
    5. Increase counter for
       appropriate point ID
}

for (pointId 0 to number of point)
{
    For each point compute
    normal by division of sum of
    normal vector and counter
}
```

Figure 10. Procedure to obtain normal vector at a point (see VTK User's Guide book for explanation of the parameters).

RESULTS

Two examples are presented here to give an illustration how the *vtkModCut* works (see Fig. 11). However, those of examples are intended to show realistic results of visualization only. Additional testing procedures should be conducted carefully to give a guarantee that the value of normal stress can be correctly proved in the sense of solid mechanics point of view.

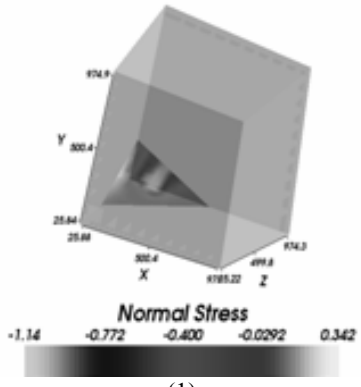
The first example is a cube loaded by a point load as shown in Figure 11. The load is located in the middle of top surface while the opposite surface is fixed. Finite element modelling and simulation had been done using 3-D solid element in separate finite element software. Finally, the stress values are modified and transferred to VTK standard input format. The second example is a T-bar with distributed moment applied at top most of the bar while left end side is fixed (see Fig. 11). Using the same procedure as the cube example, the result for different cutting plane position, can be seen in Figure 13.



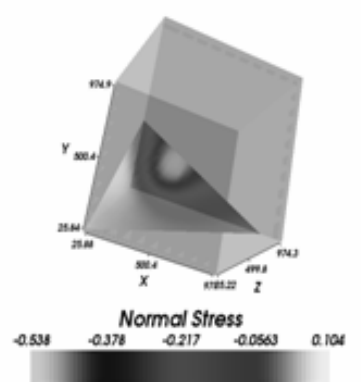
(a) (b)

Figure 11. Two examples of implementation of the cutting plane technique in VTK i.e., (a) a cube with point load and (b) a T-bar with distributed moment.

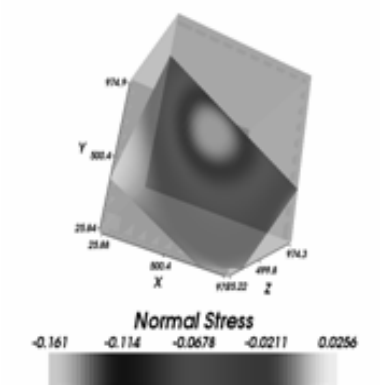
Results from the first example are shown in Figure 12 with different positions of the cutting plane. Here, using the cutting plane, one can observe distribution of normal stress at any distances from loaded location which seems realistic. As the distance is closer to the location of point load, the influence of load is more intense which is commonly produced by solid approach in a point load simulation. Furthermore, in second example, a unique distribution of normal stress due to bending can be seen at bottommost of the cutting plane (see Fig. 13). A distribution from negative or compressive stress state to positive or tensile stress state has been visualized nicely.



(1)

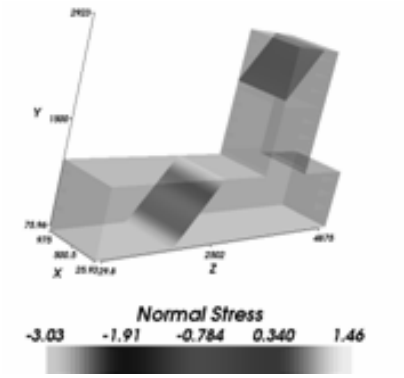


(2)

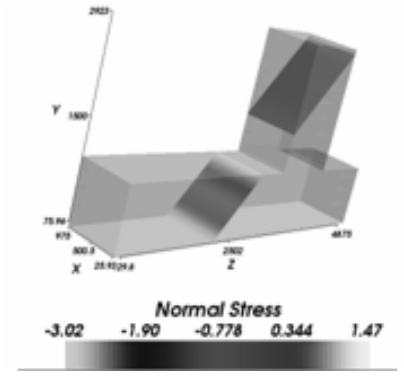


(3)

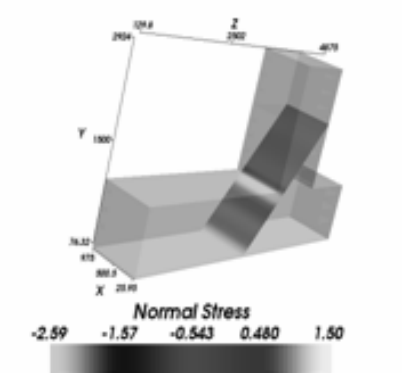
Figure 12. Different location of cutting plane inside a cube with a point load, i.e., positions (1), (2), and (3), shows the concentration of normal stress



(1)



(2)



(3)

Figure 13. A unique distribution of normal stress along thickness of the left side of bar is shown here with different positions of the cutting plane, i.e., positions (1), (2), and (3)

CONCLUDING REMARKS

The cutting plane technique has been implemented in VTK successfully. The implementation is represented in a C++ class called *vtkModCut.* Two examples are given here which produce quite nicely visualization results, i.e., concentration of stress due to a point load and a unique stress distribution by bending.

REFERENCES

[1] The VTK User's Guide Version 4.4, Kitware, Inc. 2004.
[2] Hesselink, L., Levy, Y., and Lavin, Y. (1997). The Topology of Symmetric, Second-Order 3D Tensor Fields. *IEEE Trans. on Visualization and Computer Graphics, Vol. 3, No. 1, January-March*
[3] http:// www.vtk.org
[4] Waluyo, S. (2007). Advance Visualization of Continuum Model Using Visualization Toolkit (VTK) . *Software Lab. Report, TU München, Germany (supervised by Andreas Niggl and Holger Heidkamp)*.