MVE

Modular Visualisation Environment

User documentation

version 1.04

Note that this is still *BETA* version of manual.
To get the latest copy of this manual visit

http://herakles.zcu.cz/research.php

**Centre of Computer Graphics
and Data Visualisation**

University of West Bohemia
Univerzitní 8, Box 314, 30614 Plzeň
Czech Republic

Centre of Computer Graphics and Data Visualisation

http://herakles.zcu.cz

Staff

Prof. ing. Václav Skala, CSc.
Doc. dr. ing. Ivana Kolingerová
Ing. Jiří Dobrý
Ing. Martin Franc
Ing. Jan Hrádek
Ing. Marek Krejza
Ing. Martin Kuchař
Ing. Pavel Maur
Ing. Ladislav Pešička
Ing. Michal Roušal
Ing. Tomáš Jirka
Ing. Radek Sviták
Martin Čermák
Jindřich Suja

July 25, 2001

1. Table of contents

2. MVE system version 1.3

The editor for designing module schemes in the MVE system will be described in this part of the documentation. We will explain how to set up and control this editor as a new user would whilst designing a new scheme.

2.1. Installation and start

After inserting the MVE CD into CDROM drive, the installation program should start automatically. If not, run the file **autorun.exe** from the root directory on the CD. Follow the instructions of the installation program.

After the execution of *MVE_Editor.exe*, the main application window will appear on the screen with an empty document (scheme), see Figure 2.1.

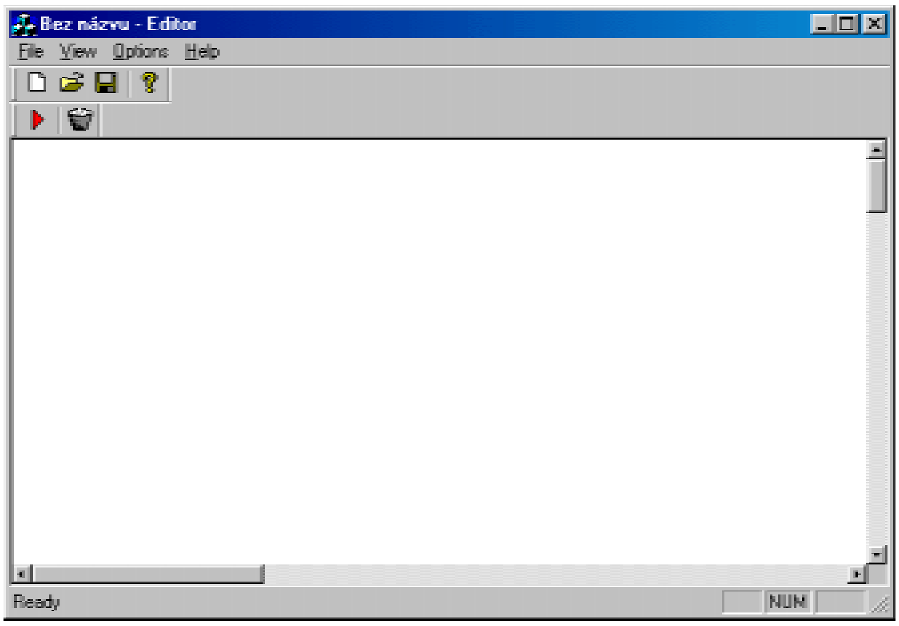


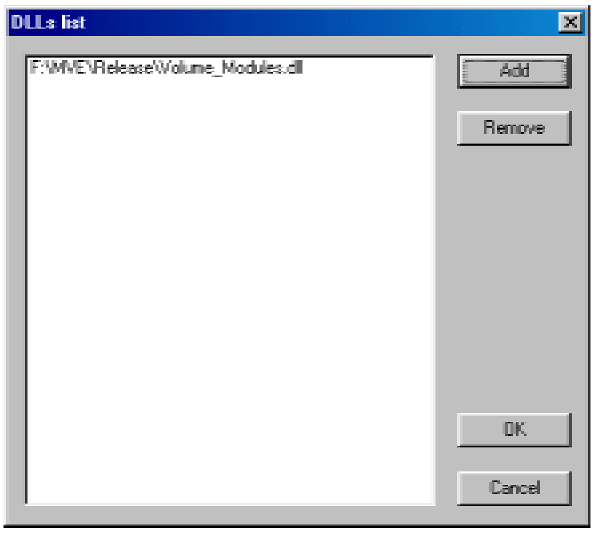
Figure 2.1 : *Empty document*

2.2. How to add new modules into MVE

Before the scheme design process can begin, we need to add some modules into the system; hence we have to add the DLL libraries that contain them. You can register the DLL libraries in two different ways.

1) Copy DLL libraries containing modules into subdirectory *Modules* located in the main MVE directory. If you copy DLL libraries into this directory while MVE is running, then you have to restart MVE.

2) The other way to add modules into MVE is through the menu *Options – Modules* dialog, where you can manually add new (external) modules. This option is aimed at modules which are either non-standard or have to be located in different directories (Figure 2.2).

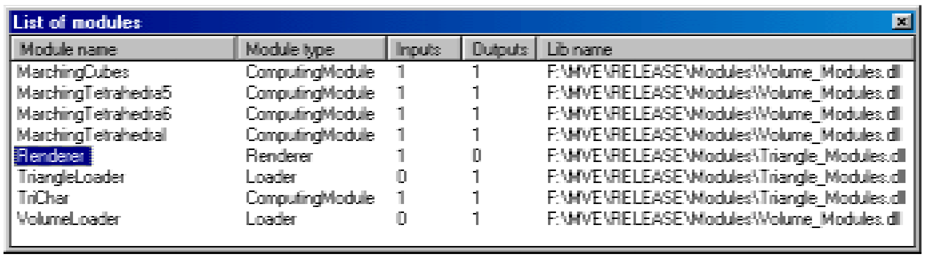


F:\MVE\Release\Volume_Modules.dll

Figure 2.2 : *External modules dialog*

2.3. How to design a schema

It is possible to display all available modules in a modules window (see Figure 2.3) by selecting *Modules* from *View* menu or by pressing *Ctrl–M* hotkey. There is a list of modules with information about the type (loader, renderer, etc.), number of inputs and outputs and the path to the DLL library where said module is located.



Module name	Module type	Inputs	Outputs	Lib name
MarchingCubes	ComputingModule	1	1	F:\MVE\RELEASE\Modules\Volume_Modules.dll
MarchingTetrahedra5	ComputingModule	1	1	F:\MVE\RELEASE\Modules\Volume_Modules.dll
MarchingTetrahedra6	ComputingModule	1	1	F:\MVE\RELEASE\Modules\Volume_Modules.dll
MarchingTetrahedral	ComputingModule	1	1	F:\MVE\RELEASE\Modules\Volume_Modules.dll
Renderer	Renderer	1	0	F:\MVE\RELEASE\Modules\Triangle_Modules.dll
TriangleLoader	Loader	0	1	F:\MVE\RELEASE\Modules\Triangle_Modules.dll
TriChar	ComputingModule	1	1	F:\MVE\RELEASE\Modules\Triangle_Modules.dll
VolumeLoader	Loader	0	1	F:\MVE\RELEASE\Modules\Volume_Modules.dll

Figure 2.3

Now you can easily drag the modules from this list and drop them on the workspace using the left mouse button. Each module is visualized as a special graphic object – *an icon*. The icon, representing the module, contains the module name and all of its inputs and outputs (along with their data types). Inputs and outputs are depicted with small arrows, as shown in Figure 2.5.

On the workspace we can manipulate (drag) modules with the mouse. The mouse cursor changes to show that dragging is possible.

2.4. Designed schemes loading and saving

Schemes designed in the editor can be saved to/loaded from a disk. This is done using the *File – Open*, *File – Save* menu items. The scheme from the editor is saved into a binary file with *.mve* extension. Paths to the DLLs can be relative (DLLs from *Modules* directory) or absolute (DLLs from other directories).

This means that schemes designed only from modules with relative paths can be moved into different directories and they should still work. But the schemes containing modules with absolute paths depend on accessibility of their DLL libraries.

We recommend using modules with relative paths (*Modules* directory) and use the external modules (DLL list) only for debugging new modules.

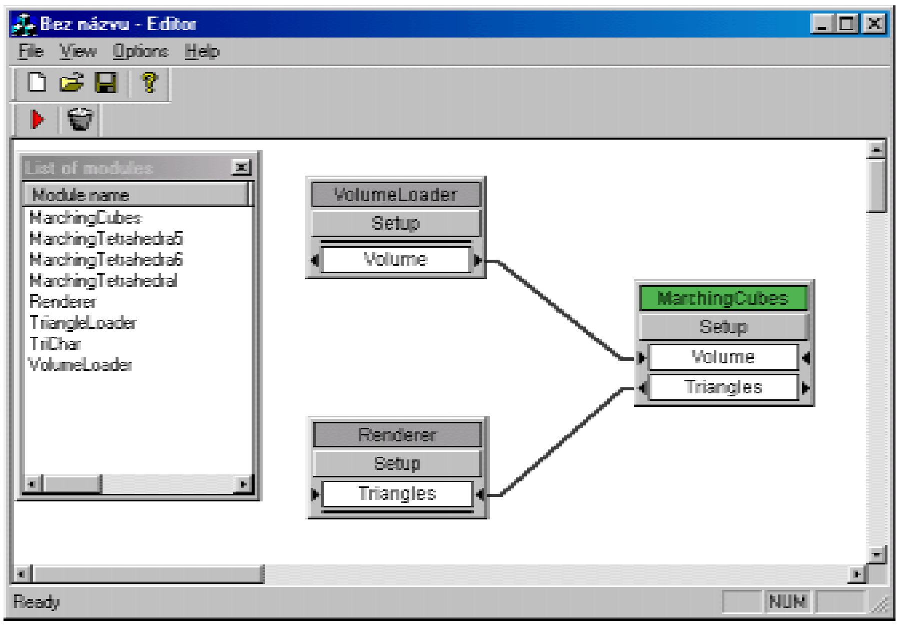


Figure 2.4

2.5. Connecting the modules

The modules on the workspace can be connected together as a user wishes enabling the data flow to be defined. Modules can be connected using a mouse. You need only to drag an input or output of one module to an output or input of other one as is shown on Figure 2.5.

When connecting the modules, data types are immediately checked so the user can connect only input and output of the same data type. When it is possible to connect the modules the line representing the connection is red, otherwise it is grey.

Connections between modules can be easily cancelled by pressing the right mouse button on the destination of the connection (input of destination module) and selecting *disconnect* from the menu.

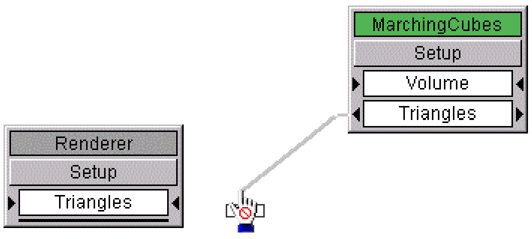
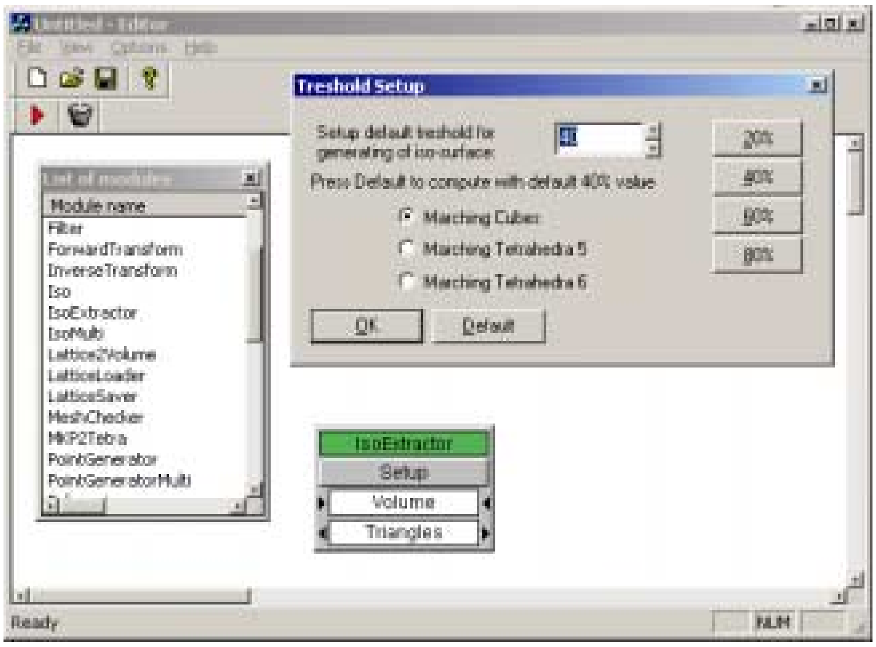


Figure 2.5 : *Connecting the modules*

2.6. User set up of modules

Some modules require a user set up before their execution (for example: the module for isosurface extraction from volume data needs a border value to be set, see Figure 2.6). Set-up data required by such modules can be inputted by pressing the *Set-up* button on the icon representing that module, otherwise default values will be used.



Setup default treshold for generating of iso-surface: 40

Press Default to compute with default 40% value

[✓ Marching Cubes]
Marching Tetrahedra 5
Marching Tetrahedra 6

Figure 2.6

2.7. Module schemes execution

If all of the modules in a scheme are connected and set up then they can be executed. Execution of the whole scheme can be done by pressing the red “run” button on the application toolbar. The names of modules that are currently running are shown on the status line of the editor. Next to the “run” button is the “recycle bin” button, which frees up all useless data (if there are any left after the execution).

The “run” button and the “recycle bin” button :

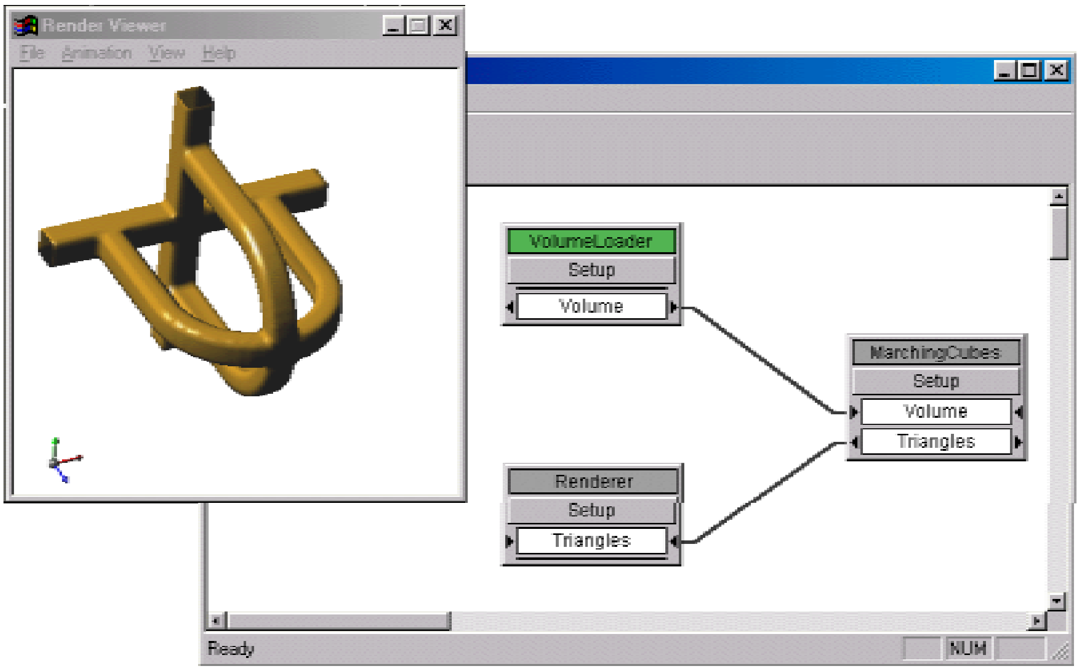
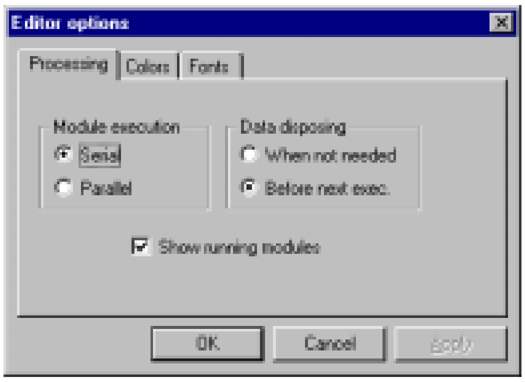


Figure 2.7 : *A simple visualisation*

2.8. Editor and module options

In the editor options a user can set up several runtime parameters. There is a special set-up dialog in the editor, which is activated from the menu *Options – Editor*. You can see these parameters in this dialog (Figure 2.8).



Serial [✓]
Parallel [ ]
When not needed [ ]
Before next exec. [✓]
Show running modules [✓]

Figure 2.8 : *Editor options*

Module execution

• *Serial*	– serial execution of modules in the designed scheme
• *Parallel*	– parallel execution of modules in the designed scheme (if the scheme design allows it). This option is especially recommended for multiprocessor systems.

Data disposal

• *When not needed*	– frees the data when they are not needed
• *Before next execution*	– frees the data before the next module execution
Show running modules	– shows names of running modules on application status line

2.9. Other module functions

Some other functions for modules are available through popup menu. This menu is activated when you press the right mouse button on the module title (see Figure 2.9).

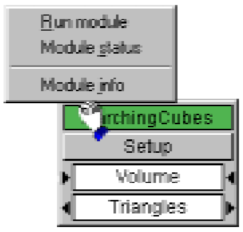


Figure 2.9 : *Module popup menu*

Popup menu items:

Run module	– where possible (i.e. when all data on inputs are ready and output isn't blocked) the module is executed and data are transferred to the next module
Module status	– displays an information window with the module execution time
Module info	– if the function *ModuleName_HELP_FUNC* is implemented in the module, then the module info window will be displayed.

3. Modules overview

3.1. Table of modules with a short description

Module name	Short module description	Inputs	Outputs	Library
PointLoader	Loads a set of points (max. 4D) from an input text file		Points	PointLoader.dll
PointLoaderMulti	Loads a set of points (max. 20D) from an input text file		Points2	PointLoaderMulti.dll
PointGenerator	Generates a set of points with different distribution in 3D space		Points	Tetra.dll
PointGeneratorMulti	Generates a set of points with different distribution in 3D space		Points2	TetraMulti.dll
TriangleLoader	Loads a triangle mesh from a file in SLT or TRI format.		Triangles	Triangle_Modules.dll
PureSTLLoader	Loads a triangle mesh from an STL file		Triangles	MeshModule.dll
MKP2Tetra	Reconstructs evaluated tetrahedral meshes from output data of MKP application.		Tetrahedra	MKP2Tetra.dll
VolumeLoader	Loads volumetric data from a file in CT, MRI or other known volumetric data formats.		Volume	Volume_Modules.dll
LatticeLoader	Loads a BMP picture to Lattice data.		Lattice	Transform.dll
Polyg_editor	Simple module with predefined implicit functions and methods for CSG modelling.		F-rep	Polyg_editor.dll
LatticeSaver	Saves Lattice 2D data to a file as BMP.	Lattice		Transform.dll
Rotation	Visualisation of N-Dimensional data using isolines.	Triangles Points2		Rotation_module.dll
Renderer	Renders an input triangle mesh using OpenGL in a rendering window.	Triangles		Triangle_Modules.dll
SliceViewer	View slices.	Slices		SlicesModules.dll
VolumeSlicer	Plane slicing of volumetric data set.	Volume		VolumeSlicer.dll
Dtlib	Computes a 2D Delaunay triangulation.		Triangles	Dtlib.dll
DT3Dlib	Computes a 3D Delaunay triangulation (tetrahedronization).		Tetrahedra	DT3Dlib.dll
TetraGenerator	Generates tetrahedra from the input data set of points.	Points	Tetrahedra	Tetra.dll
TetraGeneratorMulti	Generates tetrahedra from the input data set of points.	Points2	Tetrahedra2	TetraMulti.dll
CreateMesh	Adds an adjacency information into triangle mesh and computes normal vectors in vertices and triangles.	Triangles	Triangles	MeshModule.dll

MeshChecker	Checks correctness of a triangle mesh (number of vertices, edges, triangles and solid surfaces) using both Euler's and Vertex-to-vertex rule.	Triangles	Triangles	MeshModule.dll
Decimation	Simplifies triangular meshes.	Triangles	Triangles	Decim_module.dll
SliceMaker	Generates orthogonal sets of parallel slices from a triangle mesh.	Triangles	Slices	SlicesModules.dll
SurfReconstr	Reconstructs a surface from orthogonal slices.	Slices	Triangles	SlicesModules.dll
Iso	Extracts isostatic levels from a set of tetrahedral.	Tetrahedra	Triangles	Isosurface.dll
TetraIso	Generates an iso-surface from an input tetrahedral mesh.	Tetrahedra	Triangles	Tetra.dll
TetraIsoMulti	Generates an iso-surface from an input tetrahedral mesh.	Tetrahedra2	Triangles	TetraMulti.dll
Tetra2Triangle	Converts a tetrahedral mesh to a triangular mesh.	Tetrahedra	Triangles	Tetra.dll
Tetra2TriangleMulti	Converts a tetrahedral mesh to a triangular mesh.	Tetrahedra2	Triangles	TetraMulti.dll
DT3Auxlib	Converts tetrahedronization into a set of triangles.	Tetrahedra	Triangles	DT3Dlib.dll
IsoExtractor	Extracts an isosurface (triangle mesh) from volume data using Marching Cubes, Marching Tetrahedra5 or Marching Tetrahedra 6 methods.	Volume	Triangles	Volume_Modules.dll
Volume2Lattice	Converts Volume data to Lattice data.	Volume	Lattice	Transform.dll
Lattice2Volume	Converts Lattice data to Volume data.	Lattice	Volume	Transform.dll
ForwardTransform	Transforms Lattice data to frequency using FFT,DCT or FHT.	Lattice	Frequency	Transform.dll
InverseTransform	Gets Lattice data from an frequency pattern using FFT,DCT or FHT.	Frequency	Lattice	Transform.dll
Filter	Filters frequency data.	Frequency	Frequency	Transform.dll
Polygonizer	Polygonizer module generates the triangle mesh of objects as defined by an implicit function.	F-rep	Triangles	Polygonizer.dll

3.2. Table of modules according to the input and output

Input \ Output	Save	Render	Point	Triangles	Slices	Tetrahedron	Volume	Lattice	Frequency	F-rep
Load / Generate			PointLoader PointGenerator	TriangleLoader PureSTLloader		MKP2Tetra	VolumeLoader	LatticeLoader		Polyg_editor
Point		Rotation[I]		Dtlib		DT3Dlib TetraGenerator				
Triangles		Renderer Rotation[I]		CreateMesh MeshChecker Decimation	Slice-Maker					
Slices		Slice-Viewer		SurfReconstr						
Tetrahedron				Iso TetraIso Tetra2Triangle DT3Auxlib						
Volume		Volume-Slicer		IsoExtractor				Volume2Lattice		
Lattice	Lattice-Saver						Lattice2Volume		Forward-Transform	
Frequency								Inverse-Transform	Filter	
F-rep				Polygonizer						

The modules PointLoader, PointGenerator, TetraGenerator, TetraIso, Tetra2Triangle, Iso each have multidimensional implementation (up to 20D) in modules named *Multi.

Indexes ^{I},^{O} indicates more inputs or outputs of the module.

4. Input/Output modules

4.1. PointLoader & PointLoaderMulti

`PointLoader` is a loader for a set of points. This module has no input, the set of points is an output of the module. In the setup window (see Figure 4.1.) the user can select the name of the file by typing the filename with the path into the text box or by pressing *Browse* button – this action opens the file dialog.

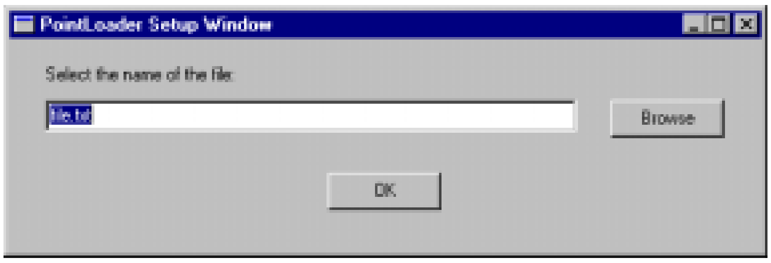


Figure 4.1 : *Set up window of `PointLoader` module*

The points are read from the text file, structured as follows:

`50`	number of points,
`10.0 5.0 8.0…`	x,y,z coordinates of each point; the number of these coordinates can be up to four, or up to twenty for the module `PointLoaderMulti`; coordinates are separated with white spaces.

The number of points must be on the first line, with the expected coordinates of each vertex on the following lines, each vertex on a special line. As yet, blank lines and comments are unsupported.

4.2. PointGenerator & PointGeneratorMulti

`PointGenerator` generates the points in 3D space. Again this module has no input, and the set of points is its output. In the set up window (see Figure 4.2) the user can select the point distribution from uniform, gaussian, points placed in clusters and points placed on lines. It is possible to place a scalar value in each generated point (4th dimension option). The last option is for the number of points to be generated. As yet `PointGeneratorMulti` doesn't allow the usage of more than four dimensions.

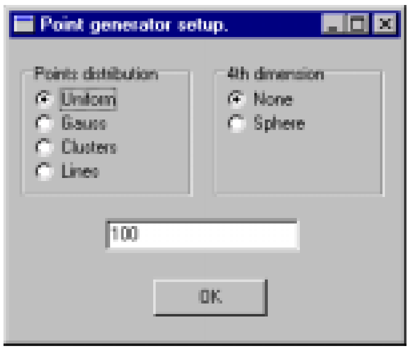


Figure 4.2 : *Setup window of `PointGenerator` module*

4.3. TriangleLoader

This module loads a triangle mesh from a file in TRI or STL format in binary or ascii form. You choose the input file using Open file dialog which will appear after clicking on Set up button.

4.4. PureSTLLoader

This module is used for loading a triangle mesh from an STL file. It reads all triangles in sequence from a file, setting up the necessary values (bounding box, number of vertices and triangles, etc.) and giving a triangle mesh as the output. The required input is simply the name of STL file to be loaded. An input file is chosen by clicking the Setup button and selecting the desired file from a dialog box.

4.5. MKP2Tetra

This module is designed to reconstruct evaluated tetrahedra meshes from the output data of an MKP application (Finite elements method). Reconstruction is based on a certain index net which is then divided into a tetrahedra mesh with conform decomposition.

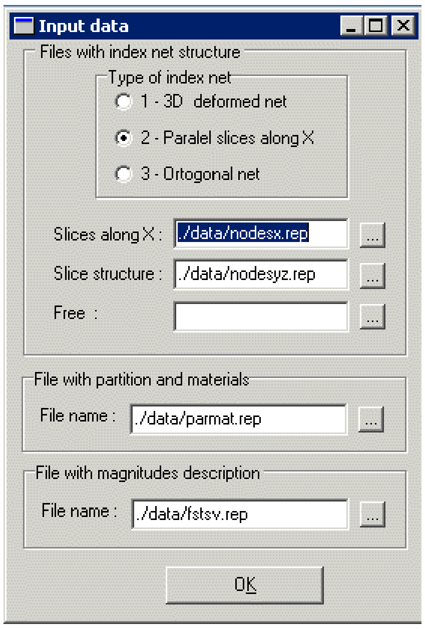
All input data for this module is given in text files of the same format, but they are of three different types. The first one describes the index net with maximum of 1 500 000 vertices in E^3, the second gives us the type of cell decomposition in the index net, mostly to tetrahedra and the type of material the tetrahedron is made of. Finally, the third type of file contains values of measured magnitudes in the referential points of tetrahedra.

The output of the module is an evaluated tetrahedral mesh.

Module was created as a *loader* for isosurface extraction.

Input parameters setting

The process for setting the input parameters is divided between two forms. In the first the user sets up the names of the input files with tetrahedra mesh, magnitudes, partition and materials. Edit boxes are set up with names of the example input files. File browsing is also possible from this form.


Input data

Files with index net structure

Type of index net

1 - 3D deformed net [✓ 2 - Paralel slices along X] 3 - Ortogonal net

Slices along X : ./data/nodesx.rep

Slice structure : ./data/nodesyz.rep

Free :

File with partition and materials

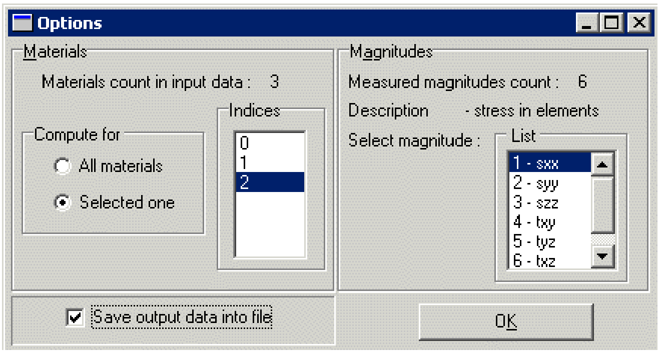
File name : ./data/parmat.rep

File with magnitudes description

File name : ./data/fstsv.rep

Figure 4.3 : *Setting up the input file names and net type*

The other form sets up the method of reconstruction. There is a selection of affected materials and magnitudes used to evaluate the output vertices, along with the option to save the output to file.


Options

Materials
Materials count in input data : 3
Compute for
All materials [✓ Selected one]
Indices
0
1
[✓ 2]

Magnitudes
Measured magnitudes count : 6
Description - stress in elements
Select magnitude :
List
[✓ 1 - sxx]
2 - syy
3 - szz
4 - txy
5 - tyz
6 - txz

[✓ Save output data into file]

Figure 4.4 : *Setting up the data subset and magnitude for reconstruction*

To complete the information about the reconstruction it is necessary to choose at least one item from each of the lists.

4.6. VolumeLoader

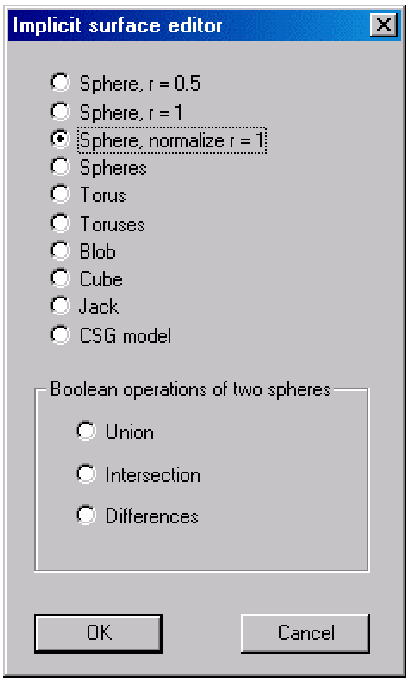
`VolumeLoader` loads volumetric data from a file in CT, MRI or other supported format. You choose the input file using Open file dialog that will appear when the Setup button is clicked.

4.7. LatticeLoader

`LatticeLoader` loads a picture in BMP format. Using the Open file dialog you choose the file to load. The formats of supported BMP files are: Windows 3.0 device independent bitmap in 1,4,8,24 bits, not compressed or RLE encoded.

4.8. Polyg_editor

The `Polyg_editor` is a simple module with certain predefined functions. This module takes no input and its output is the implicit function. The module's Set up dialog is shown in Figure 4.5.


Implicit surface editor

Sphere, r = 0.5
Sphere, r = 1
[✓ Sphere, normalize r = 1]
Spheres
Torus
Toruses
Blob
Cube
Jack
CSG model

Boolean operations of two spheres
Union
Intersection
Differences

Figure 4.5 : *The Setup dialog of Polyg_editor module.*

4.9. LatticeSaver

`LatticeSaver` saves two-dimensional data from Lattice into the file in BMP format. By clicking on the Set up button the output file is chosen. The file will be saved into a 24bit BMP without RLE. If data are not two dimensional, the file will not be saved.

5. Renderers

5.1. Rotation

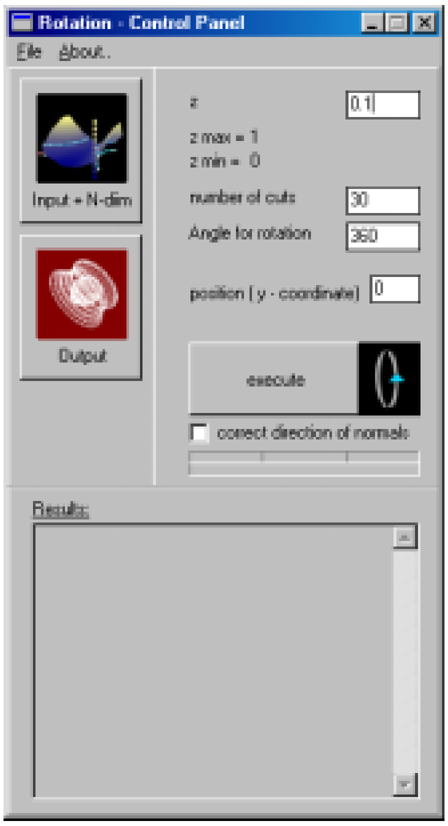
The `rotation` module is a module for visualisation N-dimensional data. This module has two main parts.

The computing part is a conversion from 2D triangle mesh (with values in z-coordinate) to a 3D surface using rotation of the isoline by x-axis. The izolines are searched by third z-coordinate (valuation). The next step is the conjunction of this izolines to the connected sections. The resulting 3D surface is obtained by rotating the connected sections.

The second part visualise the output data (rotation surface), and the points (input N-D data) as glyphs.

This module has two inputs and no outputs. The array of points in both inputs must have the same values in first two dimensions.

Control Panel



z 0.1
z max = 1
z min = 0
number of cuts 30
Angle for rotation 360
position (y - coordinate) 0
☐ correct direction of normals

Figure 5.1 : *Main window –Control Panel of rotation module*

The menu of Control panel (Figure 5.1) provides the following functions:

File	*Exit* - end of module
About	*About* - Module info

Here is explanation of important function in the window:

z	- the z-coordination for computing isoline
number of cuts	- number of sides the rotation surface
Angle for rotation	- maximal angle of rotation
Position (y-coordinate)	- distance of rotation axis from mesh
Execute	- starts the computation
correct direction of normals	- correct the direction of the normals in output
Input + N-dim	- dialog of visualisation of input data, isoline and axis of rotation
Output	- visualise the rotation mesh

Window – Input

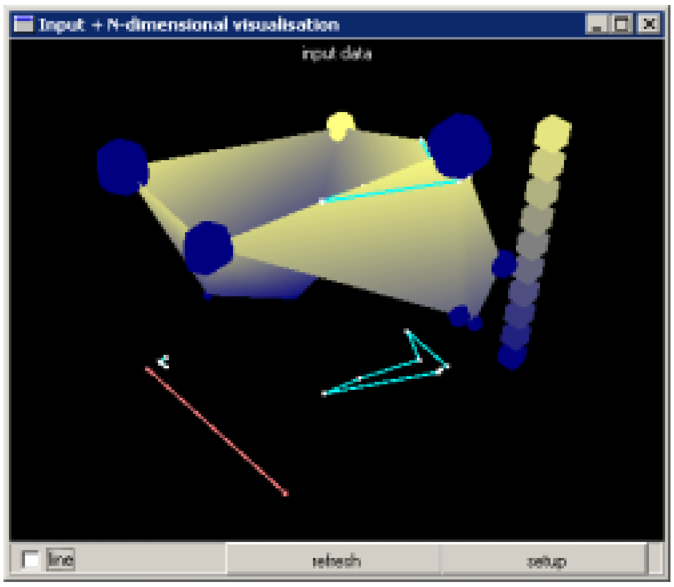


Figure 5.2 : *Input window*

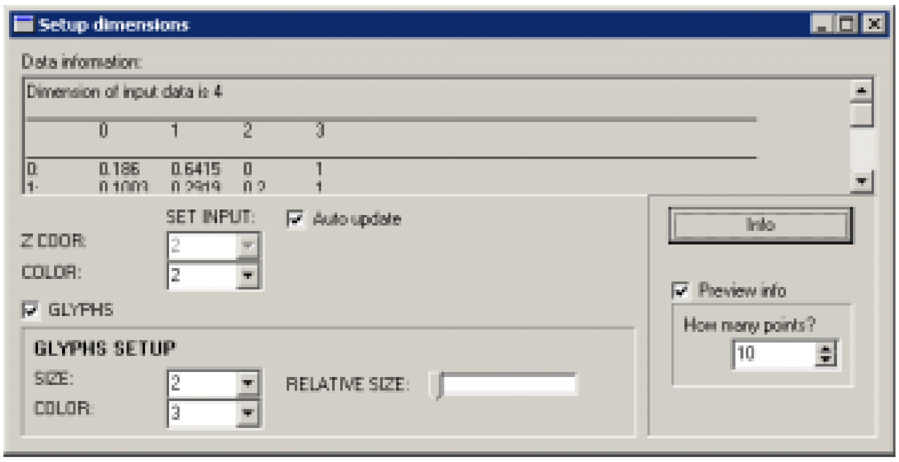
Here is explanation of important function in the Input window (Figure 5.2):

line	- change line/surface view
refresh	- refresh data
set up	- settings of input data

Functions of mouse buttons:

left button	- rotation with objects
right button	- zoom/unzoom scene
middle button	- change view-position in x, y

Window - Setup dimensions



```
Dimension of input data is 4
        0        1        2     3
0:      0.186    0.6415   0     1
1:      0.1003   0.2919   0.2   1
```

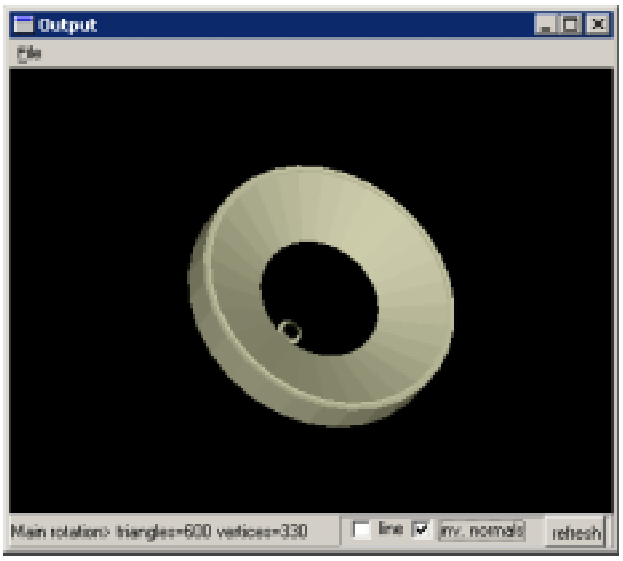
SET INPUT: ☑ Auto update
Z COOR: 2
COLOR: 2
☑ GLYPHS
SIZE: 2
COLOR: 3
☑ Preview info
How many points? 10

Figure 5.3 : *Set up dimensions window*

Here is the explanation of important function in the Set up dimensions window (Figure 5.3)

Info	- writes the information about the input data to the Data information box
Preview info	- set/unset if all of the points or just the selected number of points will be written to the Data information box
How many points	- the number of points for preview information
Auto update	- set/unset automatically refresh of input window
COLOR	- set coordinate for colour of surface
GLYPHS	- enable/disable visualisation of glyphs
SIZE	- set the dimension for the size of glyphs
COLOR	- set the dimension for the colour of glyphs
RELATIVE SIZE	- set the relative size of glyphs

Output window



Main rotation> triangles=600 vertices=330
☐ line ☑ inv. normals

Figure 5.4 : *Output window*

Here is the explanation of the important functions in the Output window (Figure 5.4):

line	- change line/surface view

inv.normals	- invert the direction of the normals
refresh	- refresh the data

Functions of mouse buttons:

left button	- rotate object
right button	- zoom/unzoom scene
middle button	- change view-position in x, y

The menu of this window provides the following functions:

File

Save as – save mesh as `*.stl` file

5.2. Renderer

`Renderer` is used for visualisation of triangle meshes. It has a configuration dialog (under View: (Un)Hide Options menu) where you can set various features of renderer. `Renderer` is capable of saving the displayed mesh in TRI or STL format (menu File:Save as..). You can also save a screenshot of the render window as BMP (menu File:Export BMP).

5.3. SlicesViewer

This views the sets (SLC format) of parallel slices (produced by e.g. `SlicesMaker`). Users select the number, colour and watch orientation of the visible slices to be viewed.

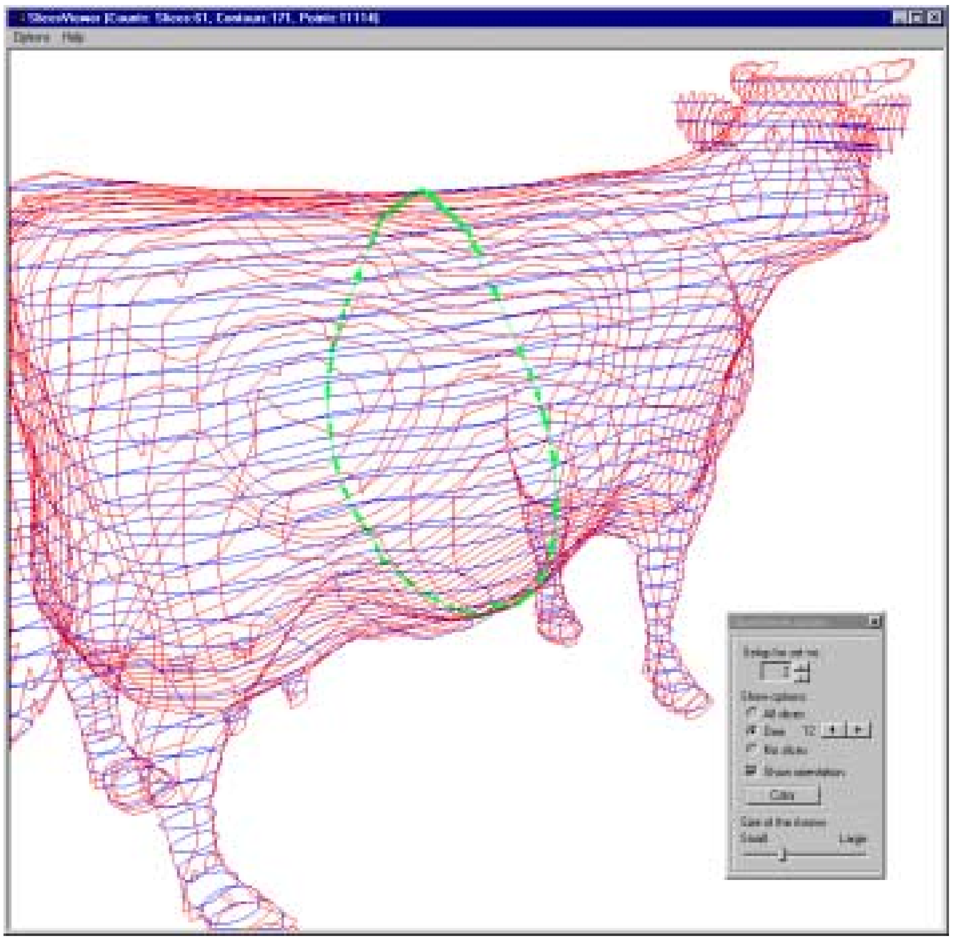


Figure 5.5 : *SlicesViewer*

5.4. VolumeSlicer

`VolumeSlicer` visualizes the intersection between a volumetric data set and a generic plane (which can be moved and rotated). This module uses volume data for its input and doesn't produce an output. Each one of `VolumeSlicer` features is accessible from a toolbar and so it does not require any set-up before being run. The module window is shown in Figure 5.6.

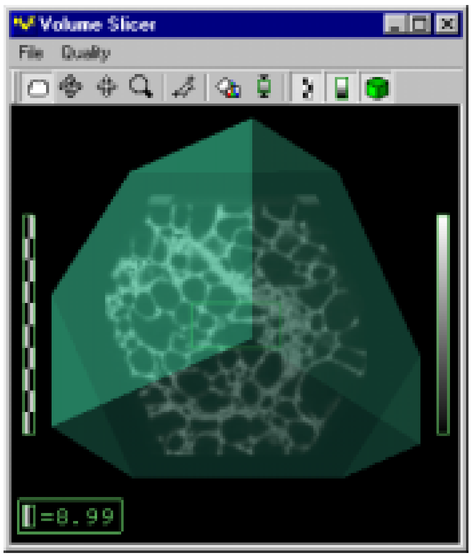


Figure 5.6 : *Module main window*

Features

In each of the following cases, after choosing the tool, move the cursor over image and hold the left mouse button.

Panning - Movement of the mouse moves the image correspondingly.

Rotating - Movement of the mouse moves the image correspondingly

Rotating - By moving mouse up and down the plane is translated forwards and backwards from the current view.

Zoom - By moving mouse up and down the view is zoomed in and out.

Measure - After choosing this tool, move cursor over first point and hold LMB then move cursor over second point and release LMB.

Other icons on toolbar and features:

Colouring - After choosing this tool a new dialog appears (Figure 5.3). Text in upper left corner is informative and is possible to change it by double clicking on it. The check box activates and deactivates colour substitution. The colour bar represents current substitute colour, which can be changed by clicking on it. The graph on the right controls the method of colour mixing. Different types are accessible from popup menu. Cursors in grey gradient represent interval.

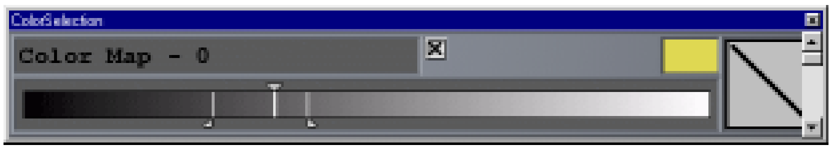
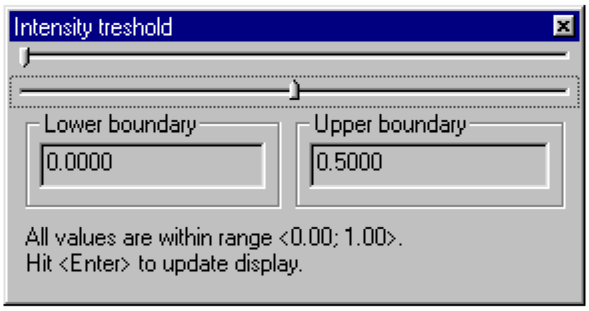


Figure 5.7 : *Coloring dialog*

Intensity clipping - After choosing this tool a new dialog appears (Figure 5.4). The first slider represents the lower clipping value with the second representing the upper clipping value. After editing, values are shown in edit boxes . The <Enter> key should be pressed to confirm these new values.


Intensity treshold

Lower boundary: 0.0000

Upper boundary: 0.5000

All values are within range <0.00; 1.00>.
Hit <Enter> to update display.

Figure 5.8 : *Clipping dialog*

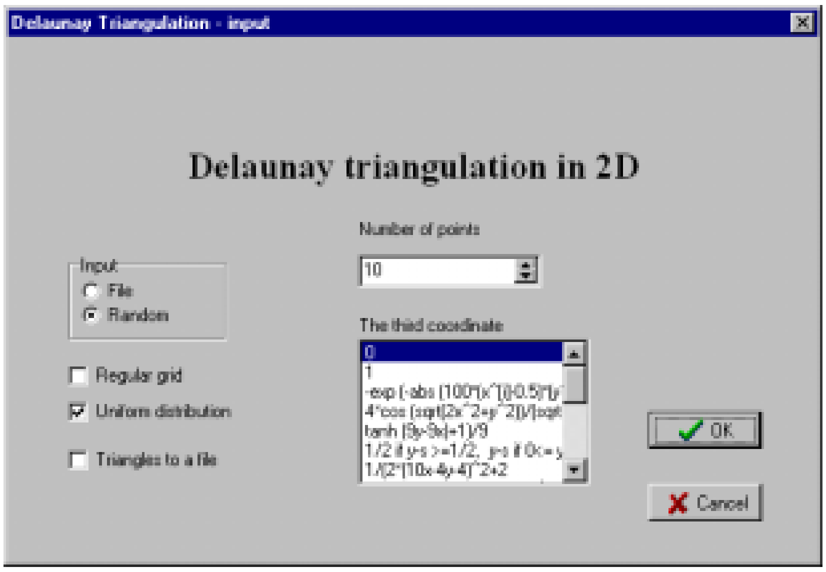
Ruler, pallet, data bounding box - switch the informative tools.

Sampling rate - Sampling rate is accessible from Quality menu.

6. Computing modules

6.1. DTlib

`DTlib` triangulates input points into triangles. It creates a triangular mesh using Delaunay triangulation.


Delaunay Triangulation - input

Delaunay triangulation in 2D

Input: ( ) File [✓ Random]

[ ] Regular grid
[✓] Uniform distribution
[ ] Triangles to a file

Number of points: 10

The third coordinate:
0
1
-exp (-abs (100*(x^(i)-0.5)*(y
4*cos (sqrt(2x^2+y^2))/sqrt
tanh (9y-9x)+1)/9
1/2 if y-s >=1/2, y-s if 0<=y
1/(2*(10x-4y-4)^2+2

Figure 6.1 : *Setting parameters of the DTlib*

The module shows a form (see Figure 6.1) where you can set parameters as follows:
Input:

- **Random** - randomly generated points on regular grid or uniformly distributed points or normally distributed points. The points are generated in 2-dimensional space, and the third coordinate can be set as a function from the list box.

- **File** – points are read from a text file in format:

  n
  $x_0\ y_0\ z_0 1\ \ldots.\ z_0 k$
  $\ldots$
  $x_{n-1}\ y_{n-1}\ z_{n-1}\ \ldots\ z_{n-1} k$

  where `n` is the total number of points (integer number) and $(x_i, y_i, z_i 0, \ldots, z_i k), i=0,1,\ldots,n-1$ are the given input points coordinates in k-dimensional space. Only $(x_i, y_i), i=0,1,\ldots,n-1$ are used for computation. Coordinates are floating point numbers.

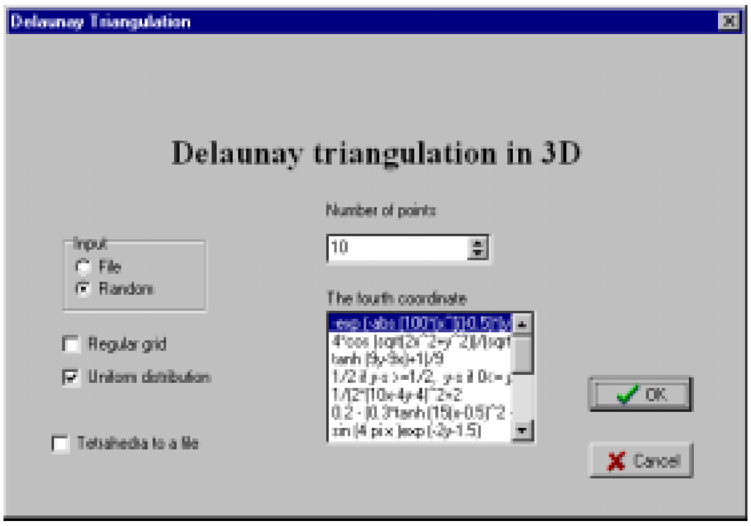
The checkbox **Triangles to a file** enables an output to be saved to file. The file contains a list of point coordinates in the same format as described above. Then the file continues with triangles as follows:

nt
$v_0 0\ \ v_0 1\ \ v_0 2$
$\ldots$
$v_{nt-1} 0\ \ v_{nt-1} 1\ \ v_{nt-1} 2\ \ v$

where `nt` is the total number of triangles (integer number) and $(v_i 0, v_i 1, v_i 2)$, $i=0,1,\ldots,nt-1$ are vertices of the triangle no. `i`.

6.2. DT3Dlib

`DT3Dlib` tetrahedrizes input points into tetrahedra. It creates the tetrahedra using Delaunay tetrahedrization.


Delaunay Triangulation

Delaunay triangulation in 3D

Input: ( ) File [✓ Random]

[ ] Regular grid
[✓] Uniform distribution
[ ] Tetrahedra to a file

Number of points: 10

The fourth coordinate:
- -exp (-abs (100*(x^2)-0.5)*[illegible]
- 4*cos (sqrt(2x^2+y^2))/(sqrt[illegible]
- tanh (9y-9x)+1)/9
- 1/2 if y-x >=1/2, y-x if 0<=[illegible]
- 1/(2*(10x-4y-4)^2+2
- 0.2 - (0.3*tanh (15(x-0.5)^2 [illegible]
- sin (4 pi x)exp (-2y-1.5)

Figure 6.2 : *Setting parameters of the DT3Dlib*

This module shows a form (Figure 6.2) where you can set parameters as follows:
Input:

- **Random** - randomly generates points on either regular grid or a choice between uniformly or normally distributed points. In this case points are generated in 3-dimensions and the fourth coordinate can be set as a function in the list box.

- **File**– points are read from a text file in format:

```
n
x0 y0 z01 .... z0k
...
xn-1 yn-1 zn-1 ... z n-1k
```

  where `n` is the total number of points (integer number) and $(x_i,y_i,z_i0,\ldots,z_ik), i=0,1,\ldots,n-1$ are the given input points' coordinates in k-dimensional space. Only $(x_i,y_i,z_i0), i=0,1,\ldots,n-1$ are used for computation. Coordinates are floating point numbers.

The checkbox Tetrahedra to a file enables the saving of output to file. The first part of the file contains:

```
<new_net>
<vertex_3d>
```

Then follows a list of point coordinates in the same format as described above. The file continues as follows:

```
<tetrahedron_by_vertices>
nt
v00 v01 v02 v03
...
vnt-10  vnt-11  vnt-12  vnt-13
```

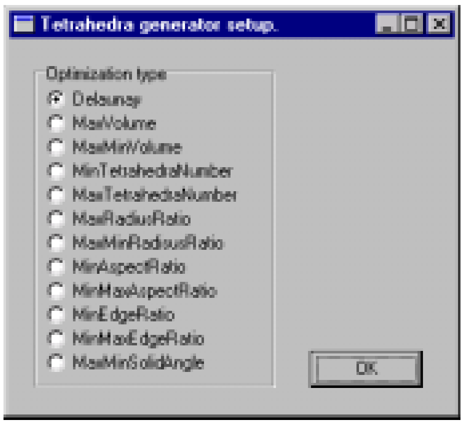
where `nt` is the total number of tetrahedra (integer number) and (v_i0,v_i1,v_i2,v_i3), $i=0,1,\ldots,nt-1$ are vertices of the terahedron no. i.

6.3. TetraGenerator & TetraGeneratorMulti

`TetraGenerator` is a tetrahedra mesh creator and optimiser. The input is a set of points, and the output is a mesh of tetrahedra. In the set up window (see Figure 6.3) it is possible to select a criterion for the process of local optimisation. Meshes created are then locally optimised according to this criterion.

User can select one of these criteria:

- `Delaunay` – criterion of empty circumsphere, it creates a 3D Delaunay triangulation,
- `MaxVolume` – maximizing of tetrahedra volume,
- `MaxMinVolume` – maximizing of minimal tetrahedra volume,
- `MinTetrahedraNumber` – minimizing of number of all tetrahedra,
- `MaxTetrahedraNumber` – maximizing of number of all tetrahedra,
- `MaxRadiusRatio` – maximizing of ratio of 3*radius of inscribed sphere and radius of circumscribed sphere,
- `MaxMinRadiusRatio` – maximizing of minimal ratio of 3*radius of inscribed sphere and radius of circumscribed sphere,
- `MinAspectRatio` – minimizing of ratio of the cirsumsphere radius and the longest edge,
- `MinMaxAspectRatio` – minimizing of maximal ratio of the cirsumsphere radius and the longest edge,
- `MinEdgeRatio` – minimizing of ratio of the longest and the shortest edge,
- `MinMaxEdgeRatio` – minimizing of maximal ratio of the longest and the shortest edge,
- `MaxMinSolidAngle` – maximizing of minimal solid angle.
- `MinMaxSolidAngle` – minimizing of maximal solid angle.

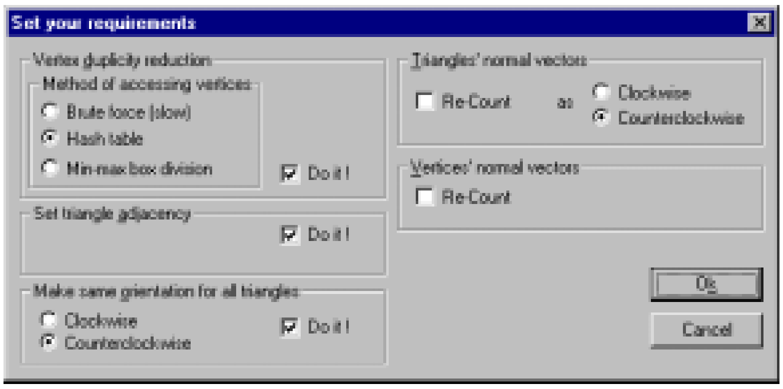

Tetrahedra generator setup.

Optimization type
[✓ Delaunay]
MaxVolume
MaxMinVolume
MinTetrahedraNumber
MaxTetrahedraNumber
MaxRadiusRatio
MaxMinRadiusRatio
MinAspectRatio
MinMaxAspectRatio
MinEdgeRatio
MinMaxEdgeRatio
MaxMinSolidAngle

Figure 6.3 : *Set up window of `TetraGenerator` module*

If the tetrahedral mesh cannot be created (e.g. due to insufficient memory space or problems with numerical precision) an error message box is shown.
The difference between the `TetraGeneratorMulti` and `TetraGenerator` lies solely in the number of supported point dimensions.

6.4. CreateMesh


Set your requirements

Vertex duplicity reduction
Method of accessing vertices: Brute force (slow) [✓ Hash table] Min-max box division
[✓ Do it !]

Set triangle adjacency
[✓ Do it !]

Make same orientation for all triangles
Clockwise [✓ Counterclockwise]
[✓ Do it !]

Triangles' normal vectors
Re-Count as Clockwise [✓ Counterclockwise]

Vertices' normal vectors
Re-Count

Figure 6.4 : *Set-up dialog of module CreateMesh*

This module is mainly used for creating information about the adjacency of triangles within the triangle mesh. The module's functions are:

- Vertex duplicity reduction
- Set triangle adjacency
- Make same orientation for all triangles
- Triangles' normal vectors
- Vertices' normal vectors

If you require the function you have to check its *enable* check-box in the set-up dialog (see Figure 6.4).

Vertex duplicity reduction

It is recommended to reduce the duplicates of vertices where such duplication arises between several triangles.

There are three methods used for searching for duplicate copies of a vertex:

- Brute force – searching the set of vertices one by one until the first duplicate vertex is found ($\mathbf{O(N^2)}$).
- Hash table – divides the set of vertices using a hash function and scans only vertices with the same hash value ($\mathbf{O(N \cdot k_1)}$, $\mathbf{k_1} \ll \mathbf{N}$, where $\mathbf{k_1}$ is the average number of vertices with the same value and is usually of order **1** or **10**).
- Min-max box division – divides the set of vertices using division of an object's bounding box, scanning only the vertices belonging to the same sub-box ($\mathbf{O(N \cdot k_2)}$, $\mathbf{k_2} \ll \mathbf{N}$, usually $\mathbf{k_2} > \mathbf{k_1}$).

Set triangle adjacency

In this function information about triangles adjacency is created (or re-created if it is already existent). For each edge of each triangle another triangle is found (if it exists) containing the same edge (i.e. an edge with the same vertices), e.g., triangles **1** and **2** in Figure 6.5. Indices to adjacent triangles are stored so that triangle's first adjacent triangle is the opposite one to the triangle's first vertex (see Figure 6.5), similarly for the second and third adjacent triangles.

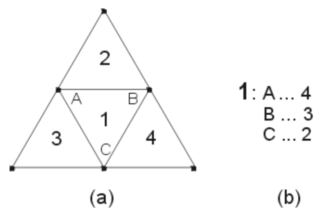


Figure 6.5 : *Adjacency of triangles*
(a) Adjacent triangles for triangle **1**
(b) Data structures for this example

If no such triangle is found, then the index to the adjacent triangle for the current triangle is set to the highest possible value (2^{32}-1 for DWORD). The same situation arises when more than one triangle with the same edge is found (see Figure 6.6). This is because no function is available to determine which two triangles are really adjacent.

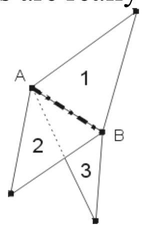


Figure 6.6 : *More than two adjacent triangles*

Make the same orientation for all triangles

Use this function when you need all triangles oriented in the same direction (counter clockwise or clockwise, (see Figure 6.7). The orientation of triangles can be used for computing normal vectors of triangles. Triangles are checked for the same orientation within solid surfaces.

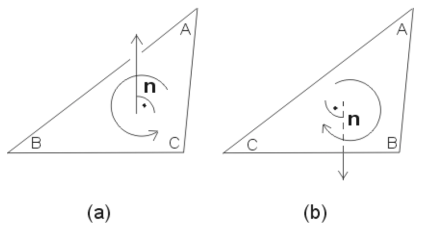


Figure 6.7 : *Triangle orientation*
(a) Counter clockwise (b) Clockwise

Normal vectors of triangles

This function computes normal vectors of all triangles, considering the chosen triangle orientation for front faced surface of triangle (see normal vectors in Figure 6.7).

Normal vectors in vertices

This function computes normal vectors in all vertices of triangle mesh. The normal vector is computed as an average of the normal vectors of triangles sharing the vertex.

6.5. MeshChecker

This module checks the correctness of a triangle mesh. It works with information about triangle adjacency but, if it is not included in data, the function **doesn’t** compute it. This means that this function doesn’t actually change any information or value included in triangle mesh, merely computing the number of vertices, edges, triangles and solid surfaces and checking them under the following rules:

- Euler's rule
- Vertex-to-vertex rule
- Checking of number of vertices and triangles

This module has no set-up dialog. When all checking operations are done the function shows the results in a modal dialog.

Euler's rule

The basic method for checking triangle mesh correctness is Euler's rule $T + V - E = 2 \cdot S$, where T is number of all triangles, V is number of all vertices, E is number of all edges, S is number of all surfaces. If Euler's rule is passed then triangle mesh is correct and contains only surfaces.

Vertex-to-vertex rule

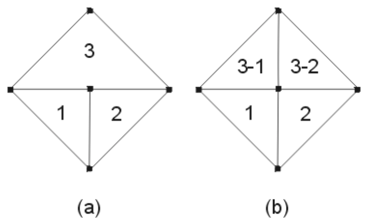


Figure 6.8 : *Vertex-to-vertex rule*
(a) Triangles don't fulfil the rule (b)Correction of the situation

This rule states that each edge must be shared just by two triangles. Each triangle has to have just two common vertices with all its adjacent triangles. When the situation from Figure 6.8a arises the triangle must be divided to two as shown in Figure 6.8b. This module doesn't perform this division.

Checking of number of vertices and triangles

On the basis of validity of Euler's rule we have three additional rules for checking triangle mesh correctness:

- Number of all triangles (T) must be even
- Number of all edges (E) must be multiple of 3
- 2 * Number of all edges (E) must be equal to 3 * Number of all triangles (T)

When these equations are satisfied, then triangle mesh is adjudged correct.

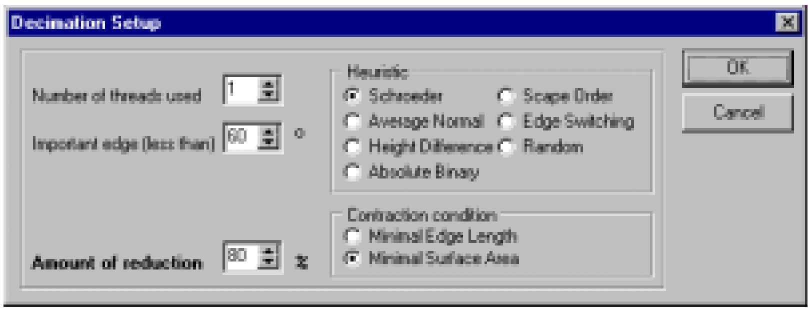
6.6. Decimation

`Decimation` for triangular mesh simplification involves reducing the number of triangles in the mesh. This module has a triangular mesh both input and output. It provides several methods of reduction allowing one to obtain different approximations of the original model at different quality levels(see Settings paragraph).

The `Decimation` module is intended to be used mostly when visualising large and complex models, with thousands or millions of triangles. Such visualisation is very demanding in terms of both memory requirement and graphics hardware performance. The manipulation (rotation, shading type change, etc.) of such complex models is almost impossible on common PCs. It is convenient to simplify the model first (preserving as much important detail as possible) and then visualise it. Results indicate that a 60% reduction in the number of triangles still leaves an acceptable model to work with.

Module set up

The `Decimation` module has plenty of user-defined parameters, with the resultant reduction being dependent on the settings. The set up window (see Figure 6.9) appears by clicking on set up button on the module in MVE.



Decimation Setup

Number of threads used: 1

Important edge (less than): 60 °

Amount of reduction: 80 %

Heuristic: [✓ Schroeder] Scape Order, Average Normal, Edge Switching, Height Difference, Random, Absolute Binary

Contraction condition: Minimal Edge Length, [✓ Minimal Surface Area]

Figure 6.9 : *Decimation module set up window.*

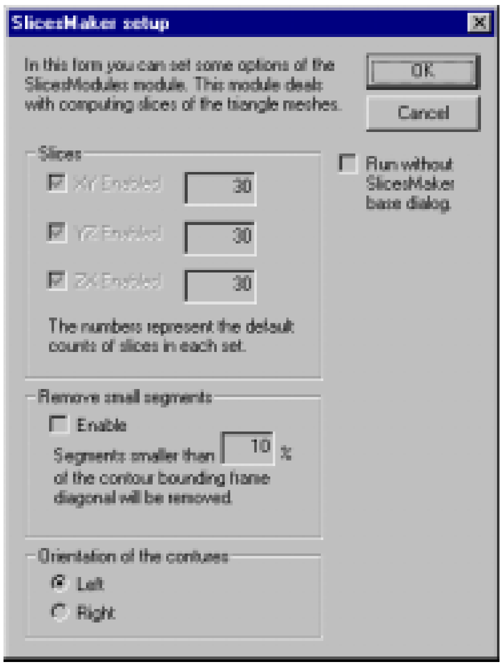
In the set up window you can choose number of threads processing the simplification algorithm. The next parameter is the maximal angle between two neighbouring triangles (dihedral angle), i.e. will the edge shared by these two triangles will be marked as a „sharp“ edge[1]? Probably most important parameter is the amount of resulting reduction. You can choose value between 0% and 100%. The simplification method can be chosen in the *Heuristic* radio group. The edge contraction criterion is selected from the radio group *Contraction Condition.*

6.7. SlicesMaker

`SliceMaker` is a generator of orthogonal sets of parallel slices. Input for this module is a triangle mesh (produces e.g. `TriangleLoader`), and the output of the module are the sets of orthogonal slices (in SLC format).

In the set up window (see Figure 6.10) a user can select number of slices in each set, the minimal size of the edges in the contours, and the orientation of these contours. A user can also decide that `SlicesMaker` will run without its main window (see Figure 6.11)

[1] Vertices on such edges are reduced by different criterions than other (sharp edged are preserved in the model).



SlicesMaker setup

In this form you can set some options of the SlicesModules module. This module deals with computing slices of the triangle meshes.

OK | Cancel

Slices
- [✓] XY Enabled: 30
- [✓] YZ Enabled: 30
- [✓] ZX Enabled: 30

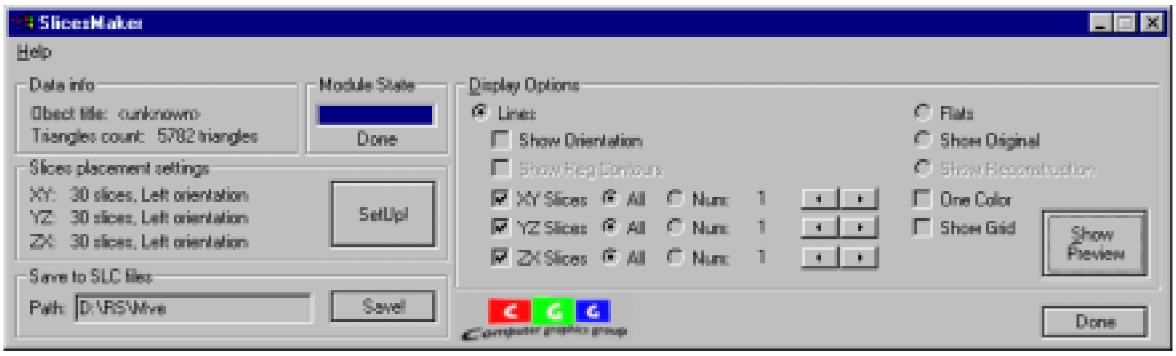
The numbers represent the default counts of slices in each set.

[ ] Run without SlicesMaker base dialog

Remove small segments
- [ ] Enable
- Segments smaller than 10 % of the contour bounding frame diagonal will be removed.

Orientation of the contures
- [✓ Left]
- Right

Figure 6.10 : *Setup window of SlicesMaker module*



SlicesMaker

Help

Data info
- Object title: <unknown>
- Triangles count: 5782 triangles

Module State: Done

Slices placement settings
- XY: 30 slices, Left orientation
- YZ: 30 slices, Left orientation
- ZX: 30 slices, Left orientation

SetUp!

Save to SLC files
- Path: D:\RS\Mve
- Save!

Display Options
- [✓ Lines] | Flats
- [ ] Show Orientation | Show Original
- [ ] Show Reg Contours | Show Reconstruction
- [✓] XY Slices [✓ All] Num: 1 | [ ] One Color
- [✓] YZ Slices [✓ All] Num: 1 | [ ] Show Grid
- [✓] ZX Slices [✓ All] Num: 1

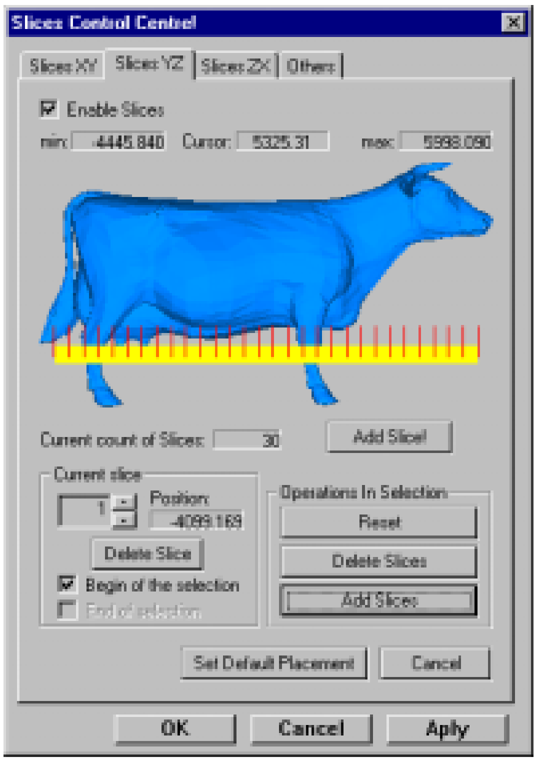
Show Preview

Done

Figure 6.11 : *SlicesMaker base dialog*

In `SlicesMaker` base dialog (Figure 6.11) a user can store slices into the files, preview generated slices (`Show Preview` button) and set up the placement of the slices (`SetUp!` button) in Slices Control Centre! (Figure 6.12).

In Slices Control Centre! user can adjust the placement of the slices in each set. The red lines represent the slices. The yellow bar represents the selection of the slices over which the user can make operations (such as 'add' or 'delete'). The user can chose to add or delete slices on an individual basis.



[✓] Enable Slices
min: -4445.840 Cursor: 5325.31 max: 5998.090
Current count of Slices: 30
Current slice: 1 Position: -4099.169
[✓] Begin of the selection
[ ] End of selection

Figure 6.12 : *Slices Control Centre window*

6.8. SurfReconstr

`SurfReconstr` is a module for reconstruction of the triangle surface from orthogonal sets of parallel slices. Its input are sets of orthogonal slices (produces e.g. `SlicesMaker`), the output is a triangle mesh, which can be seen using the `Renderer` module.

6.9. Iso & IsoMulti

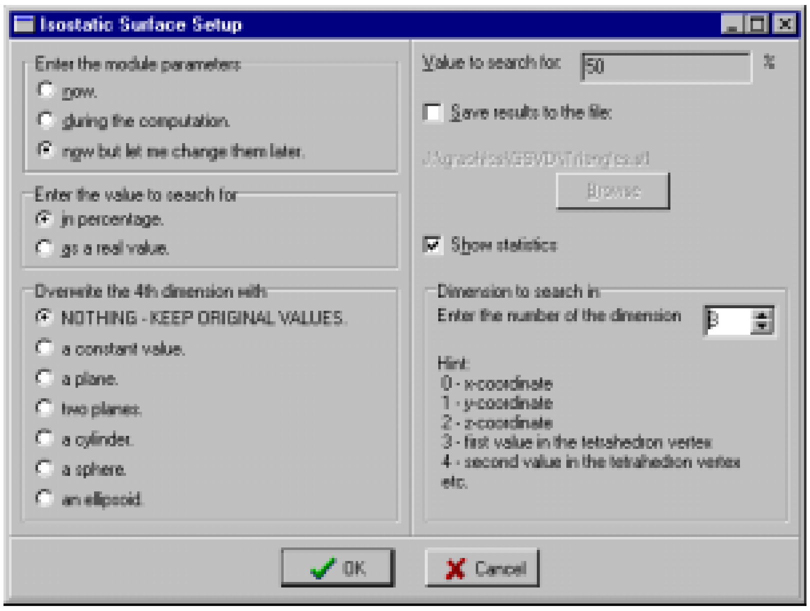
The difference between the modules `Iso` and `IsoMulti` is in number of dimensions they can operate in. The module `Iso` takes input of 4 dimensional points (1 value in each point) whereas the module `IsoMulti` works in 20 dimensions (17 values in each point). The dialogs are nearly the same, differing only that in the multidimensional version the user has to select the dimension for isosurfaces creation. Only the multidimensional version will be described here as the Iso module's functionality is otherwise identical.
In brief, the `IsoMulti` module was designed to extract isostatic levels from a set of tetrahedra. This means that if there is an object consisting of tetrahedral, each of whose vertices is represented by the *x*, *y*, *z* and *v3*,*v4*,...*vn-1* coordinates, then it could be important to find and display the surface where the value of the *vi* parameter is constant. Therefore the input of the `IsoMulti` module is a tetrahedra represented object with specific values in each tetrahedron vertex. On the output there is a triangle mesh approximating the isostatic level of the given value – i.e. the surface where value *vi* is constant. This surface can be displayed in the `Renderer` module or further processed e.g. in the `Decimation` module. The whole triangle mesh may also be saved into a file in the STL format and later loaded again via the `TriangleLoader` module.

Using the `Iso` module within MVE

Since the `IsoMulti` module requires a tetrahedra represented object as its input (the data structure is called `Tetrahedra2`), it may only follow modules that produce a set of tetrahedra with values *v3* through *vn-1* attached to each vertex, that is the `Tetrahedra2` data format, such as `TetraGenerator`. As described above, the module itself produces a triangle mesh, thus being determined to precede modules that accept such a mesh as their input. These are for instance the `Decimation` module or the `Renderer` module.

`IsoMulti` set up

It is now possible to adjust the `IsoMulti` module parameters by clicking the Setup button on the module icon on the MVE main screen before the whole diagram is executed. Although the range of acceptable values cannot be calculated at the moment, users have a chance to insert the value to search for, so that the whole diagram of modules can be run smoothly with no interruption. The value may be entered either directly (in case that the user is familiar with the data he/she works with) or in percentage. Similarly, the number of dimensions, the input data will consist of, is unknown yet. However, the user can choose the dimension to search in. It can be any dimension from the interval *v0* through *vn-1* including *v0*, *v1* and *v2,* which in fact represent the *x*, *y* and *z* coordinates. The only restriction is that, unlike the *v3* through *vn-1* values, these three coordinates may not be overwritten by any function. The set up window is displayed on Figure 6.13.



Isostatic Surface Setup

Enter the module parameters:
- now.
- during the computation.
- [✓ now but let me change them later.]

Enter the value to search for
- [✓ in percentage.]
- as a real value.

Overwrite the 4th dimension with:
- [✓ NOTHING - KEEP ORIGINAL VALUES.]
- a constant value.
- a plane.
- two planes.
- a cylinder.
- a sphere.
- an ellipsoid.

Value to search for: 50 %

Save results to the file: J:\graphics\GEVD\Triangles.st

[✓ Show statistics]

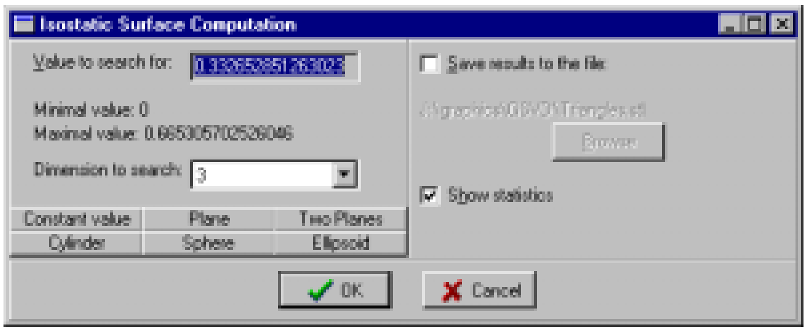
Dimension to search in
Enter the number of the dimension 3

Hint:
0 - x-coordinate
1 - y-coordinate
2 - z-coordinate
3 - first value in the tetrahedron vertex
4 - second value in the tetrahedron vertex
etc.

Figure 6.13 : *The Set up window*

In the *Enter the module parameters* box the user can choose whether to set up the module right *now* or *during the computation* or whether to set it up at the moment yet keeping a chance to change it later. Having checked the appropriate option, the user must choose, whether he/she will *Enter the value to search for in percentage* or directly *as a real value*. For the description of the remaining options on the Set up form, see the Set up window description section below.

If the user decides not to set up the module in advance using the Set up window (i.e. other option than *now* is checked in the first box), the module window (see Figure 3) will appear after the `IsoMulti` module gets focus from the MVE. By that time, the `IsoModule` has already obtained the input data, calculated the minimal and maximal values that can be searched for and displayed these on the form.


Isostatic Surface Computation

Value to search for: 0.332652851263023

Minimal value: 0
Maximal value: 0.665305702526046

Dimension to search: 3

[ ] Save results to the file: [illegible]

[✓] Show statistics

Figure 6.14 : *Adjusting IsoMulti*

Setup window description:

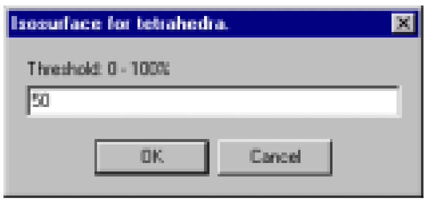
• Value to search for	- This is the value the *Iso* module will search for. It must be from the interval <Minimal value, Maximal value>, otherwise an error message appears.
• Dimension to search	- In this listbox, the user can choose the dimension to search in.
• Constant value	- Pressing this button overwrites the *v* value of all vertices with zero. Consequent search will then result in displaying the whole object.
• Plane	- Pressing this button overwrites the *v* value of each vertex with its *x* coordinate. Consequent search will then result in displaying a plane.
• Two Planes	- Pressing this button overwrites the *v* value of each vertex with the absolute value of its *x* coordinate. Consequent search will then result in displaying one or two planes, according to the location of the original object.
• Cylinder	- Pressing this button overwrites the *v* value of each vertex with its distance from the *z* axis. Consequent search will then result in displaying a part of a cylinder.
• Sphere	- Pressing this button overwrites the *v* value of each vertex with its distance from the center point S [0,0,0]. Consequent search will then result in displaying a part of a sphere.
• Ellipsoid	- Pressing this button overwrites the *v* value of each vertex with its distance from the center point S [0,0,0], where the *z* coordinate is divided by 2. Consequent search will then result in displaying a part of an ellipsoid.

- Save results to file: - Saves the computed triangle mesh into a file using the STL format. The file path and name may be specified by clicking the Browse button

- Show statistics - If this option is checked, the module will display a Statistics window after the computation has finished.

6.10. TetraIso & TetraIsoMulti

`TetraIso` is a module, which implements a simple algorithm of for generating an isosurface from a tetrahedral mesh. The input is a mesh of tetrahedral and the output is a triangle mesh, representing the isosurface. In the set up window (see Figure 6.15) only one parameter is expected: the percentage of the requested isosurface according to the minimal and maximal value in the tetrahedra mesh.
This module was initially created only for checking the functionality of the `TetraGenerator` module. This module will not be improved as it has been superceded by a new module for generating isosurface, developed during the project.
`TetraIsoMulti` version was created only for compatibility with other Multi modules.



Threshold: 0 - 100%
50

Figure 6.15: *Setup window of* `TetraIso` *module*

6.11. Tetra2Triangle & Tetra2TriangleMulti

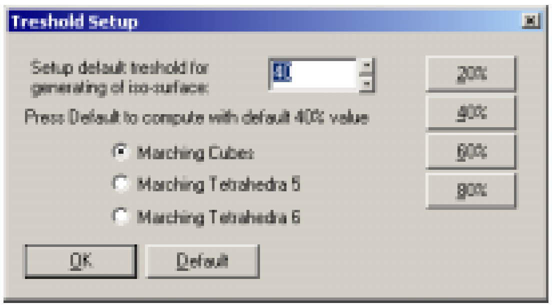
`Tetra2Triangle` is a module for the conversion of a tetrahedral mesh into a triangular mesh. The input is a mesh of tetrahedra and the output consists of a mesh of triangles. This type of conversion is needed only for visualisation of raw tetrahedral mesh. Module requires no set up.
`Tetra2TriangleMulti` version was created only for compatibility with other Multi modules.

6.12. DT3Auxlib

This is an auxiliary module for `DT3Dlib`, converting a tetrahedral mesh into a set of triangles. It doesn't require a set up function, as everything is set up from the module `DT3Dlib`.

6.13. IsoExtractor

`IsoExtractor` is used to compute iso-surfaces (triangle meshes) from the volumetric data. This module uses simple dialog (se Figure 6.16) to set the threshold and the method used. There are three methods implemented : Marching Cubes, Marching Tetrahedra5 and Marching Tetrahedra 6. The most common used method is Marching Cubes. The threshold gives the percentage of the requested isosurface according to the minimal and maximal value in volumetric data.


Treshold Setup

Setup default treshold for generating of iso-surface: 40

Press Default to compute with default 40% value

[✓ Marching Cubes]
Marching Tetrahedra 5
Marching Tetrahedra 6

20% 40% 60% 80%

OK Default

Figure 6.16: *Settings of computing modules (MC, MT5, MT6).*

6.14. Volume2Lattice

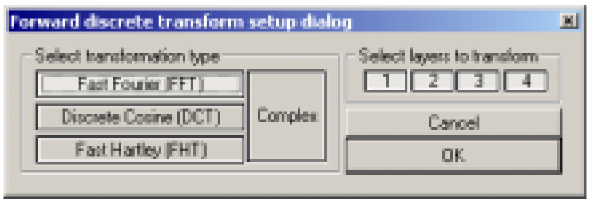
`Volume2Lattice` converts data of type Volume (3 dimensions, 1 layer) to data of type Lattice (1-3 dimensions, 1-4 layers). It doesn't physically change the data.

6.15. Lattice2Volume

`Lattice2Volume` converts data of type Lattice (1-3 dimensions, 1-4 layers) to data of type Volume (3 dimensions, 1 layer). As with `Volume2Lattice,` this conversion makes no change to data. If Lattice data are not three-dimensional and have more (or less) than one layer, then the conversion will not be done.

6.16. ForwardTransform

`ForwardTransform` transforms data of type Lattice to frequency domain (type Freq) using either a fast Fourier transformation, a discrete cosine transformation (DCT2) or a fast Hartley transformation. For module `ForwardTransform` is possible to select which parts will be converted to frequency domain and the type of transformation used. By clicking the SETUP button a dialog (see Figure 6.17) will appear.


Forward discrete transform setup dialog

Select transformation type: Fast Fourier (FFT), Discrete Cosine (DCT), Fast Hartley (FHT), Complex

Select layers to transform: 1 2 3 4

Cancel OK

Figure 6.17 : *Default settings of module ForwardTransform*

The type of transformation can be chosen in the left part of the dialog. It is possible to choose from five types of transformations:

• fast Fourier transform (real)	choose Fast Fourier (FTT), disable Complex
• fast Fourier transform (complex)	choose Fast Fourier (FFT), enable Complex
• discrete cosine transform	choose Discrete Cosine (DCT), disable DCT8
• discrete cosine transform by blocks of size 8	choose Discrete Cosine (DCT), enable DCT8
• fast Hartley transform	choose Fast Hartley (FHT)

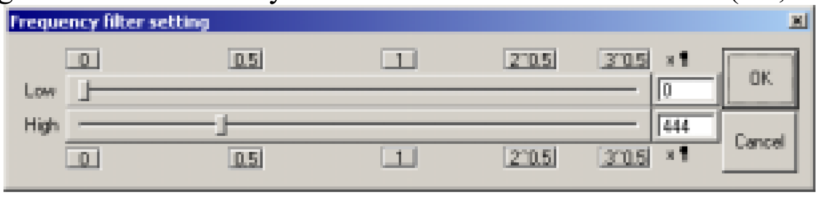
In the right part of the dialog the user can choose which layers to transform.

6.17. InverseTransform

`InverseTransform` transforms data from the frequency domain (type Freq) to the data domain (data of type Lattice) using inverse of FFT, DCT or FHT. The type of transform is taken from the previous type of `ForwardTransform`.

6.18. Filter

`Filter` uses a simple filter to cut high, low or selected frequencies from the frequency domain data (type Freq). Set up is done through a simple dialog window (see Figure 6.18). For module `Filter` an interval (radius) of the remaining frequency spectrum must be set up: slider Start sets up the beginning of this field and slider End defines its conclusion. Buttons above and below the sliders allow setting to specific values. To the right of each slider is the edit field where it is possible to enter an exact value for that slider. Value 1025 corresponds to ¶, as values bigger than 1025 are only for use in multidimensional cases (2D,3D).

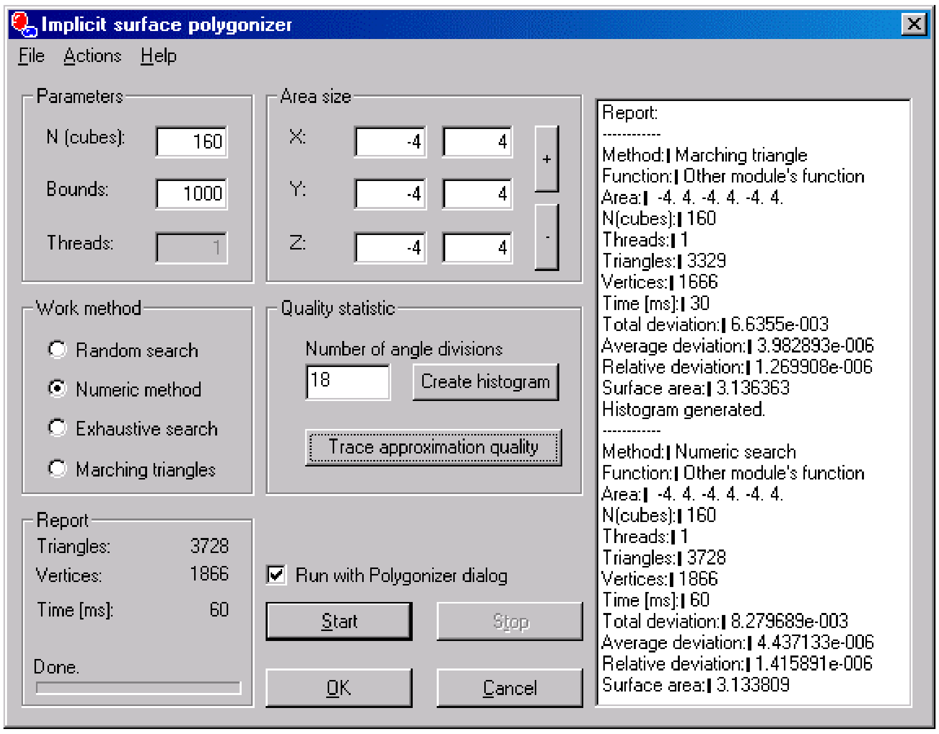


Low: 0
High: 444

Figure 6.18 : *Default settings of module Filter (this is low pass filter (LP))*

6.19. Polygonizer

The `Polygonizer` is one of the MVE modules, which extracts an iso-surface as defined by implicit function. The module input is a pointer to an implicit function and the output is a triangle mesh. The Set up dialog of `Polygonizer` is shown in Figure 6.19.



Parameters
N (cubes): 160
Bounds: 1000
Threads: 1

Area size
X: -4 4
Y: -4 4
Z: -4 4

Work method
( ) Random search
(•) Numeric method
( ) Exhaustive search
( ) Marching triangles

Quality statistic
Number of angle divisions
18

Report
Triangles: 3728
Vertices: 1866
Time [ms]: 60
Done.

[✓] Run with Polygonizer dialog

```
Report:
------------
Method:| Marching triangle
Function:| Other module's function
Area:| -4. 4. -4. 4. -4. 4.
N(cubes):| 160
Threads:| 1
Triangles:| 3329
Vertices:| 1666
Time [ms]:| 30
Total deviation:| 6.6355e-003
Average deviation:| 3.982893e-006
Relative deviation:| 1.269908e-006
Surface area:| 3.136363
Histogram generated.
------------
Method:| Numeric search
Function:| Other module's function
Area:| -4. 4. -4. 4. -4. 4.
N(cubes):| 160
Threads:| 1
Triangles:| 3728
Vertices:| 1866
Time [ms]:| 60
Total deviation:| 8.279689e-003
Average deviation:| 4.437133e-006
Relative deviation:| 1.415891e-006
Surface area:| 3.133809
```

Figure 6.19 : *The Setup dialog of Polygonizer module.*

Controls description

Main menu

- File
  - **Save Triangles** – Saves a triangle net to disk in TRI format.
  - **Save Report** – Saves the *Report* list box to disk in RSL text file.
  - **Save Histogram** – Saves Histogram to disk in HIST text file.
- **Exit** – Closes dialog without saving of setup.
- Actions - **Start** – Starts the computation.
- Help - **About** – The basic information dialog about actual module's version and author.

Buttons description

- Parameters
  - **N (cubes**) – Partitions the defined area (Area size) in all axes (x, y, z).
  - **Bounds** – Bounding box, the maximal number of indexes of cells (cubes) in all axes.
  - **Threads** – The number of working threads in parallel computation (*Exhaustive search* method only).
- **Work** method **– The implemented methods for visualization of implicit surfaces.**
  - **Random search**
  - **Numeric method**
  - **Exhaustive search**
  - **Marching triangles**
- **Report** – Two boxes containing information about the computation (name of method and function, area size and its partitioning in axes, number of threads, number of triangles and vertices, time of computation).
- **Area size** – Size of area in 3D.
- ***Quality statistic***
  - **Number of angle divisions** - *(π/n).*
  - **Create histogram** – *Creates a histogram showing the density of the division of the angle.*
  - **Trace approximation quality** – *The approximate quality. The results are written to* Report *list box.*
- Run with Polygonizer dialog – Running of module with polygonizer set up dialog. It is useful for measuring the parameters of functions from other modules, or for visualisation of suspended polygonization (when *Stop* button is pressed).
- Start – Starts the computation.
- Stop – Stops the computation.
- OK – Saves the module's set up or sends the triangle net to the next MVE module[*].
- Cancel – Closes the dialog without saving the set up.

[*] Only if the module was run with **Run with polygonizer dialog** checked.

Examples

A.1. Examples of visualisation of points

The points can be visualized using triangles. Here are some examples.

On Figure A.1 there is an example of using the Renderer for visualization of tetrahedra created from a set of 3D points.

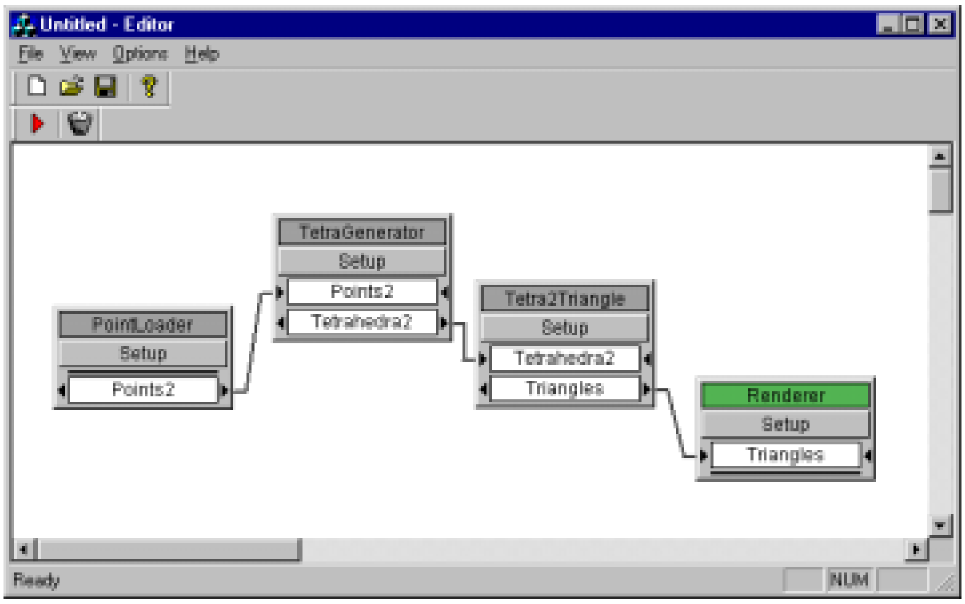


Figure A.1 *Visualization of points (tetrahedra)*

Example of visualization of N-dimensional (more that 3D) points using Renderer is on Figure A.2. Points are loaded, tetrahedronized and then converted to the iso surface (triangle mesh) for rendering.

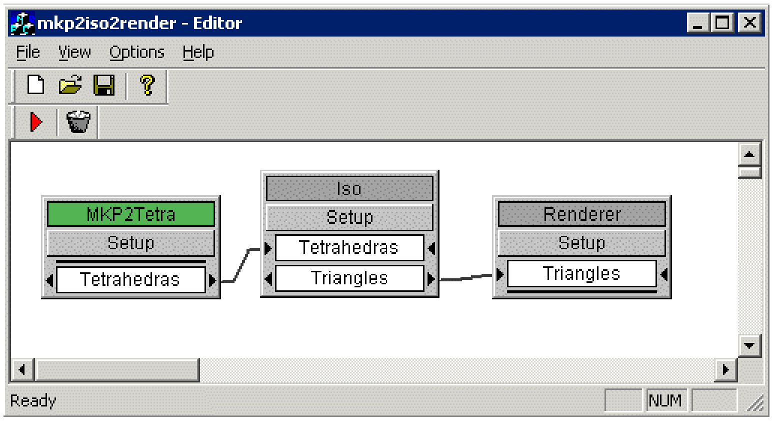


Figure A.2. : *Example of using the MKP2Tetra mod*ule

Figures A.3. and A.4 show how Delaunay triangulation can be used in MVE. On Figure A.3 there is a 2D case: triangulation is done only in two dimensions, but each vertex can have height (value). Figure A.4 shows a scheme for visualizing tertrahedra using a triangle mesh (iso surface).

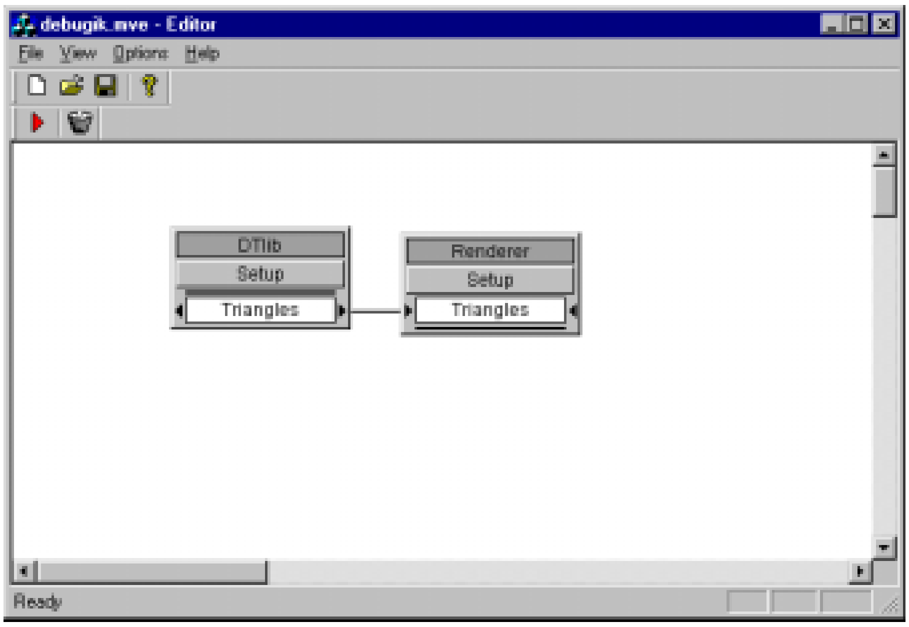


Figure A.3 : *2D Delaunay triangularization*

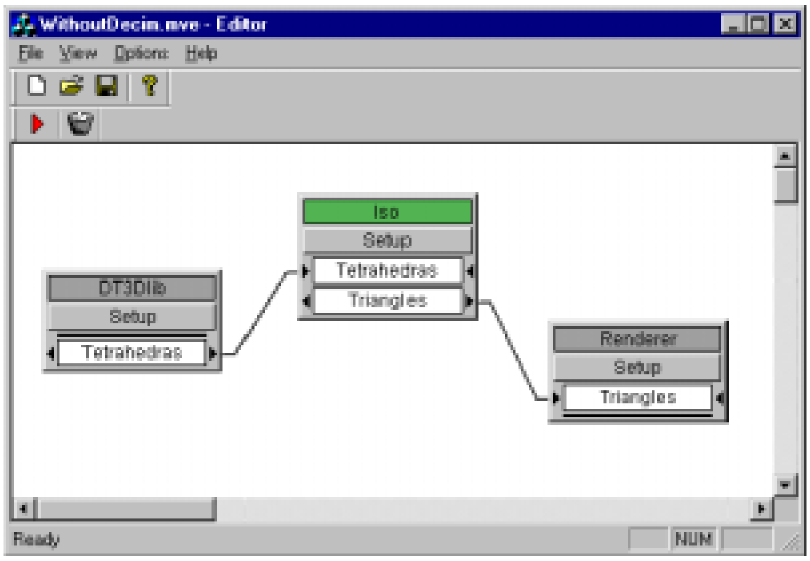


Figure A.4 : *3D Delaunay tetrahedronization*

On Figure A.5 is a visualization of the rotated isoline (making a 3D triangular mesh). The isoline is extracted from triangularized 2D triangular mesh using other dimensions.

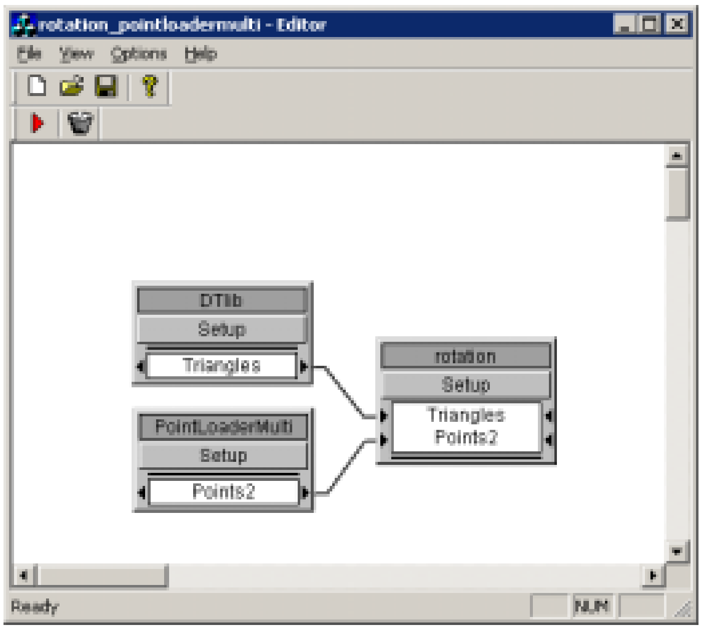


Figure A.5 : *Connection of modules*

A.2. Examples of visualisation of triangles

Visualisation of triangles is the most commonly used method. Triangles can be directly visualised using `Renderer`.

The usual way to visualise triangles is displayed on Figure A.6. Note the decimation module, used to reduce the number of triangles.

Figure A.7 depicts a scheme for adding information about the mesh to the triangles, such as adjacency, normal vectors etc. This information can be used by other modules.

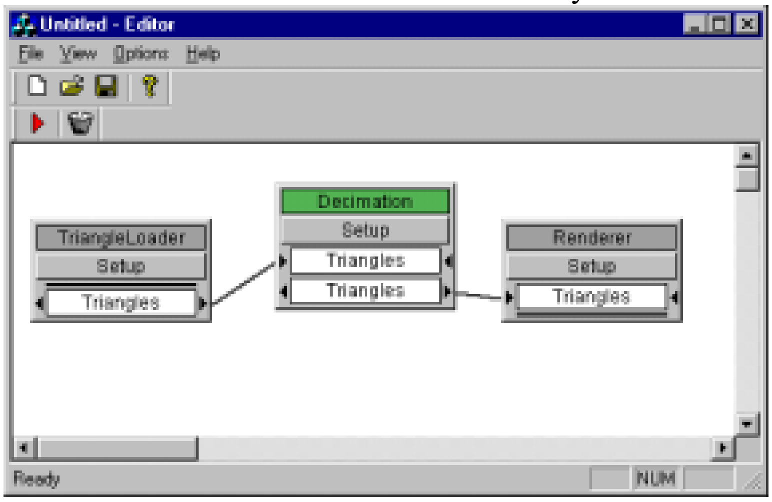


Figure A.6 : *Visualisation of triangles with decimation*

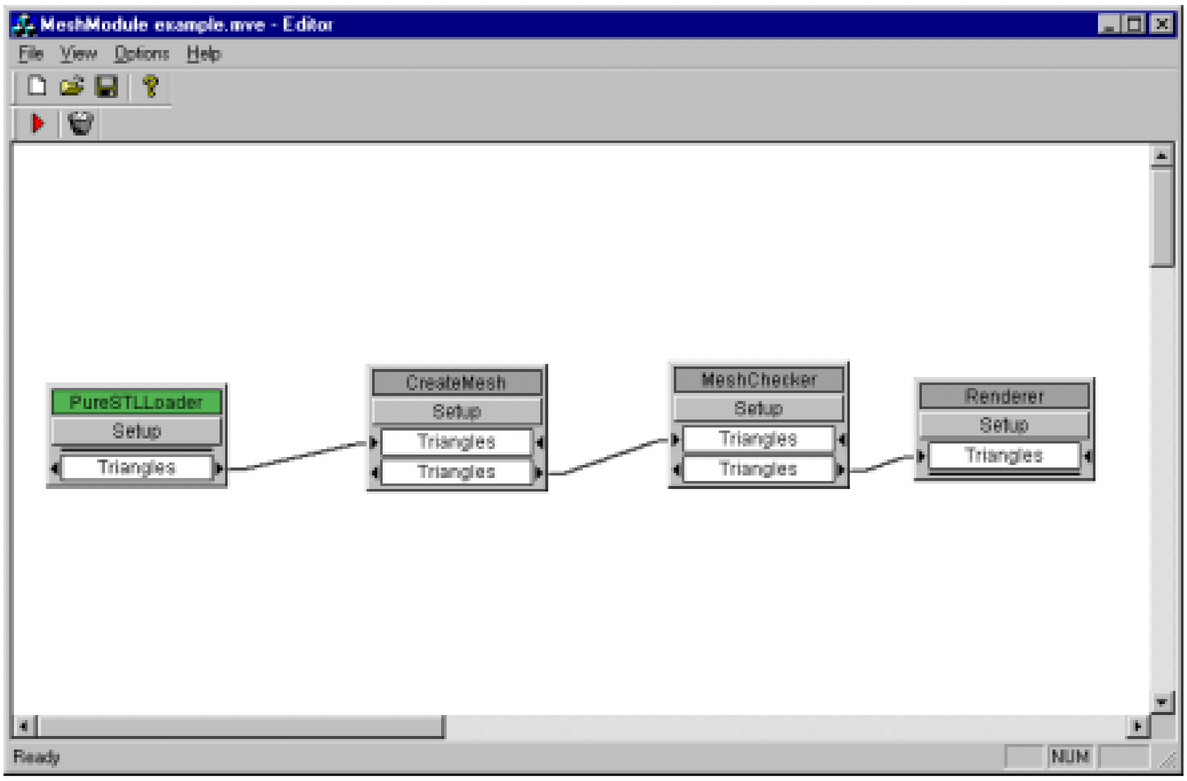


Figure A.7 : *Adding useful information into triangles*

A.3. Visualisation and operations over volumetric data

Volumetric data (usually medical or mechanical CT, MRI) consists of objects, which can be visualised in various ways.

The first way is the extraction of isosurfaces (triangles). The scheme for this type of visualisation is shown on Figure A.8.

Figure A.9 shows a scheme using discrete transformations on volumetric data.

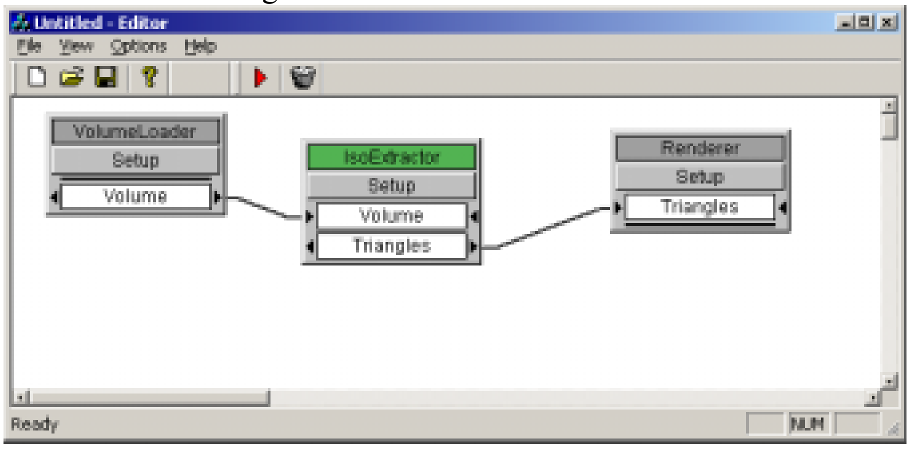


Figure A.8: *Visualisation of volumetric data using isoextraction*

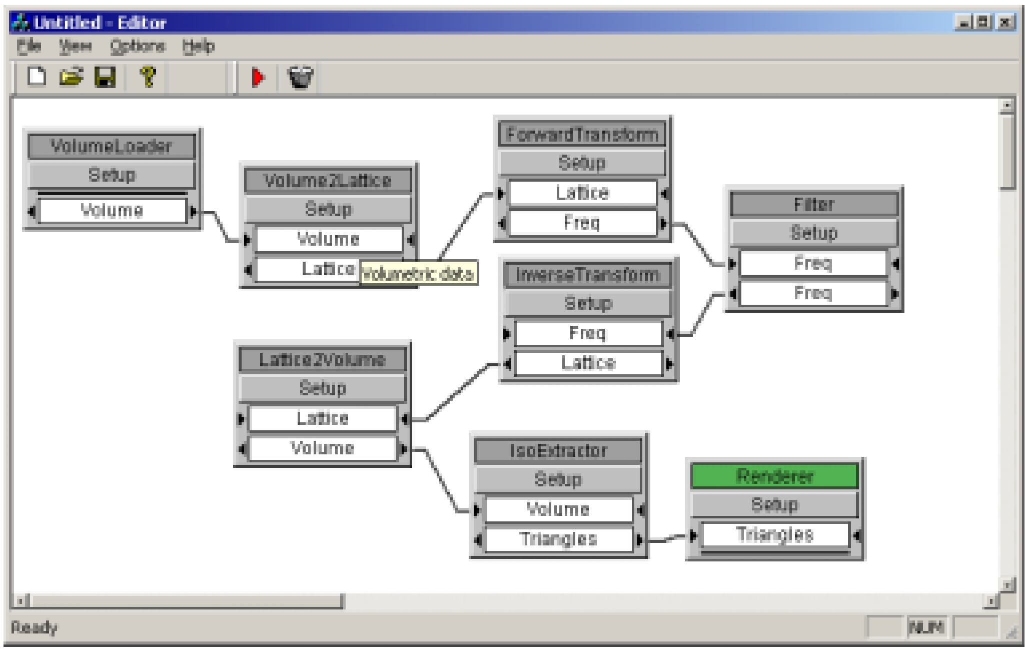


Figure A.9 : *Visualisation of volumetric data with discrete transformation*

A.4. Examples of VolumeSlicer outputs

The scheme below represents the most common configuration of modules for the slicing of a volumetric data set. The first module loads volumetric data from a file. Its output is connected with the input of the `VolumeSlicer` module. The `VolumeSlicer` is a renderer class module so it's always at the end of the evaluation chain.

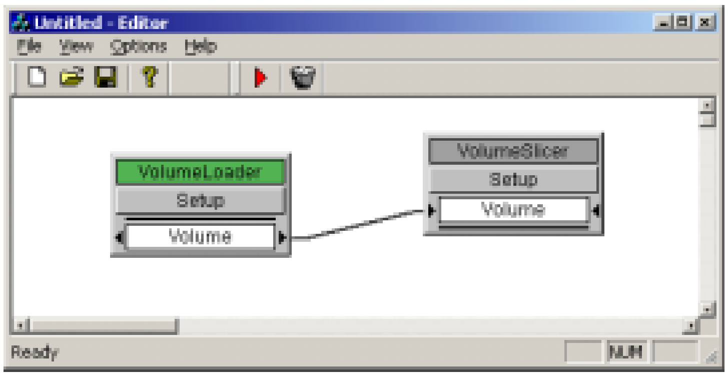


Figure A.10 : *Scheme of Slicing of volumetric data set.*

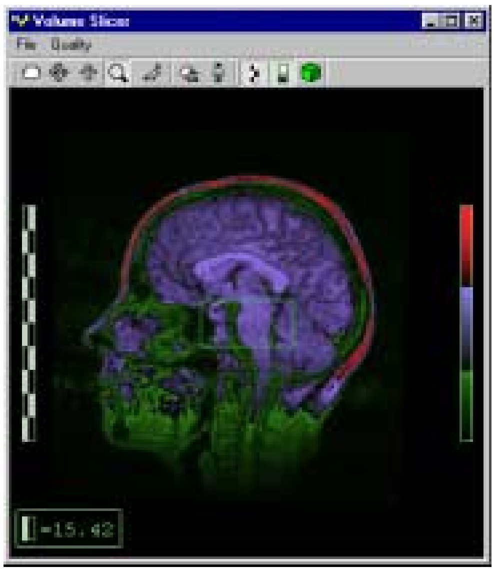


Figure A.11 : *Usage of coloring in volume slicing*

A.5. Visualisation of slices

Presently there is only one way to get these slices-from a triangle mesh (and also you can get triangle mesh from slices). Slices can be displayed separately. Figure A.12 shows such a scheme.

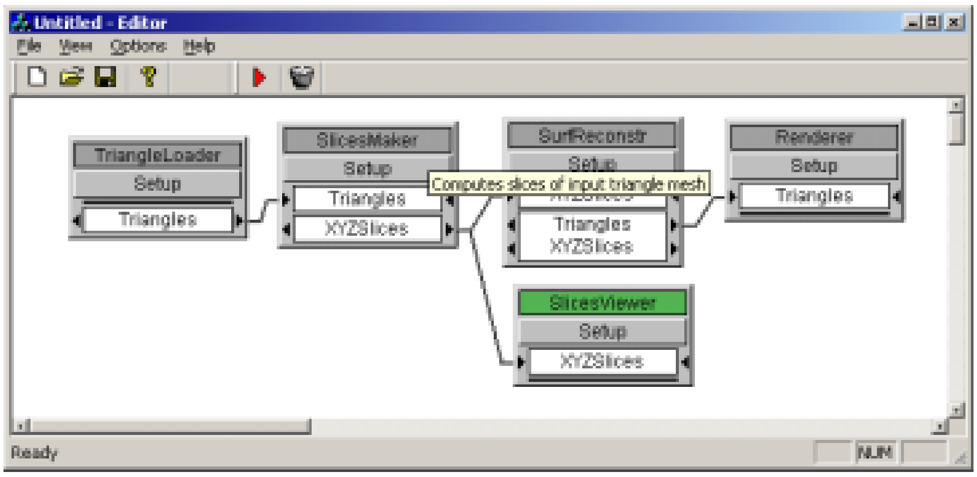


Figure A.12 : *Visualisation of slices*

A.6. Examples of implicit visualization

In the basic scheme (Figure A.13) there are three collaborating MVE modules. The `Polyg_editor` is an implicit function source, `Polygonizer` is the computing module and the `Renderer` module is used for triangle mesh visualization.

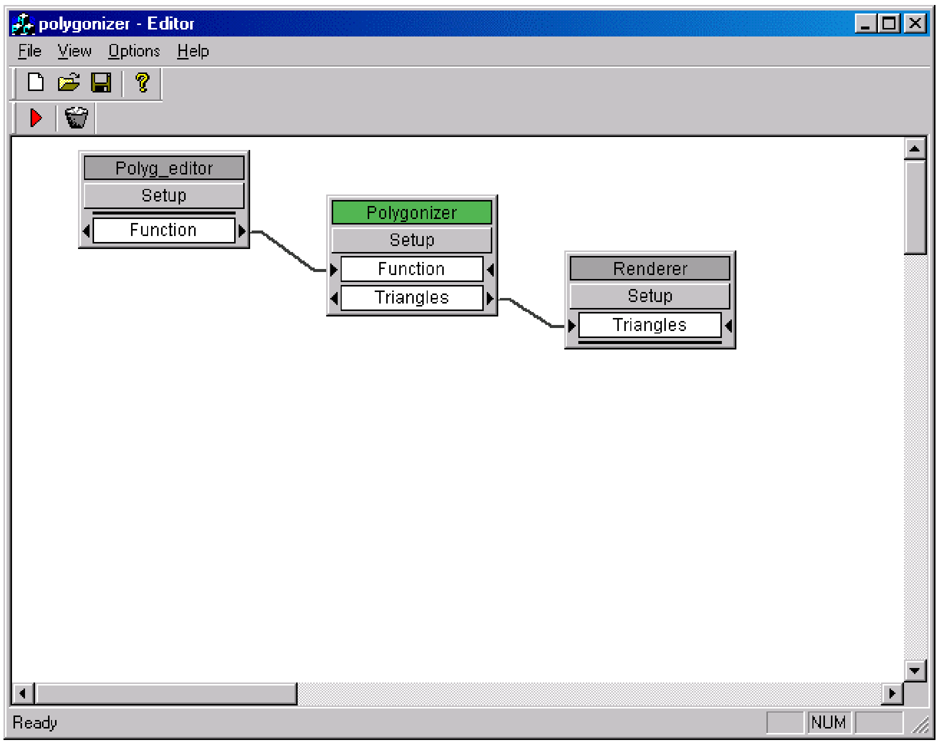


Figure A.13 : *The basic polygonizer scheme*

List of example data

Type		Name	Size	Courtesy
Tri		Bell.tri	28,776kB	
		bone.tri	9,052kB	Cyberware
		bunny.tri	9,467kB	Gatech
		cow.tri	682kB	Avalon
		hand.tri	87,772kB	Gatech
Stl	Bin	Dino50.stl	2,750kB	Cyberware
		teapot.stl	7,794kB	
		teeth25.stl	2,849kB	Cyberware
Stl	Ascii	Baba.stl	9,287kB	Cyberware
		isis25.stl	20,381kB	Cyberware
Vol	1byte	bentum.vol	16,385kB	
		cthead.vol	7,233kB	University of North Carolina
		ctmayo.vol	2,049kB	
		engine.vol	7,041kB	
		hplogo.vol	2,213kB	
		syn_64.vol	257kB	
Vol	2byte	Cthead.vol	14,464kB	University of North Carolina

Cyberware	: http://www.cyberware.com
Gatech	: http://www.cc.gatech.edu/projects/large_models/index.html