

Triangulate3D

For commands referencing other waves in a graph: **ImageNameList**, **ImageNameToWaveRef**, **ContourNameList**, and **ContourNameToWaveRef**.

Triangulate3D

Triangulate3D [/OUT=*format*] *srcWave*

The Triangulate3D operation creates a Delaunay “triangulation” of a 3D scatter wave. The output is a list of tetrahedra that completely span the convex volume defined by *srcWave*. Triangulate3D can also generate the triangulation needed for performing 3D interpolation for the same domain. Normally *srcWave* is a triplet wave (a 2D wave of 3 columns), but can use any 2D wave that has more than 3 columns (the operation ignores all but the first 3 columns).

Flags

| | |
|---------------------|--|
| /OUT= <i>format</i> | Specifies how to save the output triangulation data. <i>format</i> =1: Default; saves the triangulation result in the wave M_3DVertexList, which contains in each row, indices to rows in <i>srcWave</i> that describe the X, Y, Z coordinates of a single tetrahedral vertex. Each tetrahedron is described by one row in M_3DVertexList. <i>format</i> =2: Saves the triangulation result in the wave M_TetraPath, which is a triplet path wave describing the tetrahedra edges. For each tetrahedron, there are four rows (triangles) separated by row of NaNs. The total number of rows in M_TetraPath is 20 times the number of tetrahedra in the triangulation. <i>format</i> =4: Saves a wave containing internal diagnostic information generated during the triangulation process. |
| /VOL | Computes the volume of the full convex hull by summing the volumes of the tetrahedra generated in the triangulation. The result is stored in the variable V_value. This flag requires Igor Pro 7.00 or later. |

Details

Triangulate3D implements Watson’s algorithm for tetrahedralization of a set of points in three dimensions. It starts by creating a very large tetrahedron which inscribes all the data points followed by a sequential insertion of one datum at a time. With each new datum the algorithm finds the tetrahedron in which the datum falls. It then proceeds to subdivide the tetrahedron so that the datum becomes a vertex of new tetrahedra.

The algorithm suffers from two known problems. First, it may, due to numerical instabilities, result in tetrahedra that are too thin. You can get around this problem by introducing a slight random perturbation in the input wave. For example:

```
srcWave+=enoise(amp)
```

Here amp is chosen so that it is much smaller than the smallest cartesian distance between two input points.

The second problem has to do with memory allocations which may exhaust available memory for some pathological spatial distributions of data points. The operation reports both problems in the history area.

Examples

```
Make/O/N=(10,3) ddd=gnoise(5)      // create random 10 points in space
Triangulate3d/out=2 ddd

// now display the triangulation in Gizmo:
Window Gizmo() : GizmoPlot
    PauseUpdate; Silent 1
    if(exists("NewGizmo")!=4)
        DoAlert 0, "Gizmo XOP must be installed"
        return
    endif
    NewGizmo/N=Gizmo0 /W=(309,44,642,373)
    ModifyGizmo startRecMacro
    ModifyGizmo scalingMode=2
    AppendToGizmo Scatter=root:ddd,name=scatter0
    ModifyGizmo ModifyObject=scatter0 property={ scatterColorType,0}
    ModifyGizmo ModifyObject=scatter0 property={ Shape,2}
    ModifyGizmo ModifyObject=scatter0 property={ size,0.2}
```