module - computing Volume ∇

In[1]:=
```
DumpGet["C:\\Users\\aliha\\Desktop\\wolfram-vertex-3D\\PREVIOUS
    CODE - slow heuns\\meshGen_noise.mx"];
```

In[2]:=
```
yLim[[1]] = 0.;
edges = SetPrecision[edges, 10];
faceListCoords = SetPrecision[faceListCoords, 10];
(*convert faceListCoords into an association*)
indToPtsAssoc = SetPrecision[indToPtsAssoc, 10];
ptsToIndAssoc = KeyMap[SetPrecision[#, 10] &, ptsToIndAssoc];
xLim = SetPrecision[xLim, 10];
yLim = SetPrecision[yLim, 10];
faceListCoords = Map[Lookup[indToPtsAssoc, #] &, cellVertexGrouping, {2}];
```

In[10]:=
```
With[{xlim1 = xLim[[1]], xlim2 = xLim[[2]], ylim1 = yLim[[1]], ylim2 = yLim[[2]]},
  periodicRules = Dispatch[{
     {x_ /; x ≥ xlim2, y_ /; y ≤ ylim1, z_} :> SetPrecision[{x - xlim2, y + ylim2, z}, 10],
     {x_ /; x ≥ xlim2, y_ /; ylim1 < y < ylim2, z_} :> SetPrecision[{x - xlim2, y, z}, 10],
     {x_ /; xlim1 < x < xlim2, y_ /; y ≤ ylim1, z_} :> SetPrecision[{x, y + ylim2, z}, 10],
     {x_ /; x < 0., y_ /; y ≤ ylim1, z_} :> SetPrecision[{x + xlim2, y + ylim2, z}, 10],
     {x_ /; x < 0., y_ /; ylim1 < y < ylim2, z_} :> SetPrecision[{x + xlim2, y, z}, 10],
     {x_ /; x < 0., y_ /; y > ylim2, z_} :> SetPrecision[{x + xlim2, y - ylim2, z}, 10],
     {x_ /; 0. < x < xlim2, y_ /; y > ylim2, z_} :> SetPrecision[{x, y - ylim2, z}, 10],
     {x_ /; x > xlim2, y_ /; y ≥ ylim2, z_} :> SetPrecision[{x - xlim2, y - ylim2, z}, 10]}];
  transformRules = Dispatch[{
     {x_ /; x ≥ xlim2, y_ /; y ≤ ylim1, _} :> SetPrecision[{-xlim2, ylim2, 0}, 10],
     {x_ /; x ≥ xlim2, y_ /; ylim1 < y < ylim2, _} :> SetPrecision[{-xlim2, 0, 0}, 10],
     {x_ /; xlim1 < x < xlim2, y_ /; y ≤ ylim1, _} :> SetPrecision[{0, ylim2, 0}, 10],
     {x_ /; x < 0, y_ /; y ≤ ylim1, _} :> SetPrecision[{xlim2, ylim2, 0}, 10],
     {x_ /; x < 0, y_ /; ylim1 < y < ylim2, _} :> SetPrecision[{xlim2, 0, 0}, 10],
     {x_ /; x < 0, y_ /; y > ylim2, _} :> SetPrecision[{xlim2, -ylim2, 0}, 10],
     {x_ /; 0 < x < xlim2, y_ /; y > ylim2, _} :> SetPrecision[{0, -ylim2, 0}, 10],
     {x_ /; x > xlim2, y_ /; y ≥ ylim2, _} :> SetPrecision[{-xlim2, -ylim2, 0}, 10],
     {___Real} :> SetPrecision[{0, 0, 0}, 10]}];
 ];
```

In[11]:=
```
origcellOrient = <|MapIndexed[First[#2] → #1 &, faceListCoords]|>;
boundaryCells = With[{ylim1 = yLim[[1]], ylim2 = yLim[[2]], xlim2 = xLim[[2]]},
   Union[First /@ Position[origcellOrient,
       {x_ /; x ≥ xlim2, __} | {x_ /; x < 0, __} |
        {_, y_ /; y > ylim2, _} | {_, y_ /; y ≤ ylim1, _}] /. Key[x_] :> x]
  ];
wrappedMat = AssociationThread[
   Keys[cellVertexGrouping] → Map[Lookup[indToPtsAssoc, #] /. periodicRules &,
     Lookup[cellVertexGrouping, Keys[cellVertexGrouping]], {2}]];
```

In[14]:=
```
meanTri = Compile[{{faces, _Real, 2}},
  Mean@faces,
  CompilationTarget → "C", RuntimeAttributes → {Listable},
  Parallelization → True
 ]
```

Out[14]= CompiledFunction[Argument count: 1 Argument types: {{_Real, 2}}]

In[15]:=
```
Clear[triNormal];
triNormal = Compile[{{ls, _Real, 2}},
  Block[{res},
   res = Partition[ls, 2, 1];
   Cross[res[[1, 1]] - res[[1, 2]], res[[2, 1]] - res[[2, 2]]]
  ], CompilationTarget → "C", RuntimeAttributes → {Listable}
 ]
```

Out[16]= CompiledFunction[Argument count: 1 Argument types: {{_Real, 2}}]

In[17]:=
```
Clear[meanFaces, triangulateToMesh];
meanFaces = Compile[{{faces, _Real, 2}},
  Block[{facepart, edgelen, mean},
   facepart = Partition[faces, 2, 1];
   AppendTo[facepart, {facepart[[-1, -1]], faces[[1]]}];
   edgelen = Table[Norm[SetPrecision[First@i - Last@i, 10]], {i, facepart}];
   mean = Total[edgelen * (Mean /@ facepart)] / Total[edgelen];
   mean],
  RuntimeAttributes → {Listable}, CompilationTarget → "C",
  CompilationOptions → {"InlineExternalDefinitions" → True}
 ]

triangulateToMesh[faces_] := Block[{mf, partfaces},
   mf = SetPrecision[meanFaces@faces, 10];
   partfaces = Partition[#, 2, 1, 1] & /@ faces;
   MapThread[
    If[Length[#] ≠ 3,
      Function[x, Join[x, {#2}]] /@ #1,
      {#[[All, 1]]}
     ] &, {partfaces, mf}]
  ];
```

Out[18]= CompiledFunction[Argument count: 1 Argument types: {{_Real, 2}}]

In[20]:=
```
Clear@cellCentroids;
cellCentroids[polyhedCentAssoc_, keystopo_, shiftvec_] :=
  Block[{assoc = <||>, regcent, counter},
   AssociationThread[Keys@keystopo →
     KeyValueMap[
      Function[{key, cellassoc},
       If[KeyFreeQ[shiftvec, key],
        Lookup[polyhedCentAssoc, cellassoc],
        If[KeyFreeQ[shiftvec[key], #],
           regcent = polyhedCentAssoc[#],
           regcent = polyhedCentAssoc[#] + shiftvec[key][#];
           regcent
          ] & /@ cellassoc
       ]
      ], keystopo]
   ]
  ];
```

In[22]:=
```
𝒟 = Rectangle[{First@xLim, First@yLim}, {Last@xLim, Last@yLim}];
```

In[23]:=
```
getLocalTopology[ptsToIndAssoc_, indToPtsAssoc_, vertexToCell_,
    cellVertexGrouping_, wrappedMat_, faceListCoords_][vertices_] :=
  Module[{localtopology = <||>, wrappedcellList = {}, vertcellconns,
    localcellunion, vertInBounds, v, wrappedcellpos, vertcs = vertices,
    transVector, wrappedcellCoords, wrappedcells, vertOutofBounds,
    shiftedPt, transvecList = {}, $faceListCoords = Values@faceListCoords,
    vertexQ},
   vertexQ = MatchQ[vertices, {__?NumberQ}];
   If[vertexQ,
    vertcellconns =
     AssociationThread[{#} , {vertexToCell[ptsToIndAssoc[#]]}] &@vertices;
    vertcs = {vertices};
    localcellunion = Flatten[Values@vertcellconns],
    (* this will yield vertex → cell indices connected in the local mesh *)
    vertcellconns =
     AssociationThread[# , Lookup[vertexToCell, Lookup[ptsToIndAssoc, #]]] &@vertices;
    localcellunion = Union@Flatten[Values@vertcellconns];
   ];
   (* condition to be an internal
    edge: both vertices should have 3 or more neighbours *)
   (*Print["All topology known"];*)
   (* the cells in the local mesh define the entire network topology →
    no wrapping required *)
   (* else cells need to be wrapped because other cells are
     connected to the vertices → periodic boundary conditions *)
   With[{vert = #},
      If[((𝒟~RegionMember~Most[vert]) &&
          ! (vert[[1]] == xLim[[2]] || vert[[2]] == yLim[[2]])),
        (* the vertex has less than 3 neighbouring cells but
```

```
 the vertex is within bounds *)
(*Print["vertex inside bounds with fewer than 3 cells"];*)
v = vertInBounds = vert;
(* find cell indices that are attached to the vertex in wrappedMat *)
wrappedcellpos = DeleteDuplicatesBy[
  Cases[Position[wrappedMat, x_ /; SameQ[x, v], {3}],
   {Key[p : Except[Alternatives @@
        Join[localcellunion, Flatten@wrappedcellList]]], y__} :> {p, y}],
  First];
(*wrappedcellpos = wrappedcellpos/.
   {Alternatives@@Flatten[wrappedcellList],__} :> Sequence[];*)
(* if a wrapped cell has not been considered earlier (i.e. is new)
 then we translate it to the position of the vertex *)
If[wrappedcellpos ≠ {},
 If[vertexQ,
  transVector = SetPrecision[(v - Extract[$faceListCoords,
        Replace[#, {p_, q__} :> {Key[p], q}, {1}]]) & /@ wrappedcellpos, 10],
   (*the main function is enquiring an edge and not a vertex*)
  transVector =
   SetPrecision[(v - Extract[$faceListCoords, #]) & /@ wrappedcellpos, 10]
 ];
 wrappedcellCoords = MapThread[#1 →
     Map[Function[x, SetPrecision[x + #2, 10]], $faceListCoords[[#1]], {2}] &,
   {First /@ wrappedcellpos, transVector}];
 wrappedcells = Keys@wrappedcellCoords;
 AppendTo[wrappedcellList, Flatten@wrappedcells];
 AppendTo[transvecList, transVector];
 AppendTo[localtopology, wrappedcellCoords];
 (*local topology here only has wrapped cell *)
],
(*Print["vertex out of bounds"];*)
(* else vertex is out of bounds *)
vertOutofBounds = vert;
(* translate the vertex back into mesh *)
transVector = vertOutofBounds /. transformRules;
shiftedPt = SetPrecision[vertOutofBounds + transVector, 10];
(* find which cells the vertex is a part of in the wrapped matrix *)
wrappedcells = Complement[
  Union@Cases[Position[wrappedMat, x_ /; SameQ[x, shiftedPt], {3}],
     x_Key :> Sequence @@ x, {2}] /. Alternatives @@ localcellunion → Sequence[],
  Flatten@wrappedcellList];
(*forming local topology now that we know the wrapped cells *)
If[wrappedcells ≠ {},
 AppendTo[wrappedcellList, Flatten@wrappedcells];
 wrappedcellCoords = AssociationThread[wrappedcells,
   Map[Lookup[indToPtsAssoc, #] &, cellVertexGrouping[#] & /@ wrappedcells, {2}]
  ];
 With[{opt = (vertOutofBounds /. periodicRules)},
  Block[{pos, vertref, transvec},
    Do[
```

```
            With[{cellcoords = wrappedcellCoords[cell]},
             pos = FirstPosition[cellcoords /. periodicRules, opt];
             vertref = Extract[cellcoords, pos];
             transvec = SetPrecision[vertOutofBounds - vertref, 10];
             AppendTo[transvecList, transvec];
             AppendTo[localtopology, cell →
               Map[SetPrecision[# + transvec, 10] &, cellcoords, {2}]];
            ], {cell, wrappedcells}]
          ];
        ];
       ];
      ];
    ] & /@ vertcs;
  If[localcellunion ≠ {},
   AppendTo[localtopology,
    Thread[localcellunion →
      Map[Lookup[indToPtsAssoc, #] &, cellVertexGrouping /@ localcellunion, {2}]]
   ]
  ];
  transvecList = Which[
    MatchQ[transvecList, {{{__?NumberQ}}}], First[transvecList],
    MatchQ[transvecList, {{__?NumberQ} ..}], transvecList,
    True, transvecList //. {x___, {p : {__?NumberQ} ..}, y___} :→ {x, p, y}
   ];
  {localtopology, Flatten@wrappedcellList, transvecList}
 ];
```

Launch Kernels

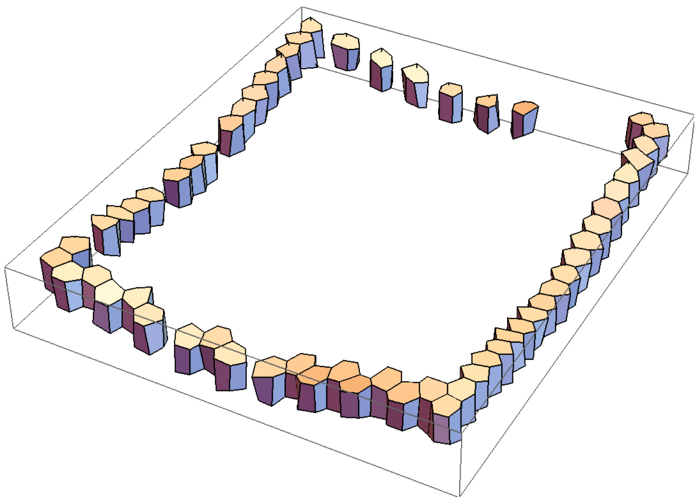
In[24]:=
```
LaunchKernels[]
```

Out[24]= {KernelObject[Name: local KernelID: 1], KernelObject[Name: local KernelID: 2], KernelObject[Name: local KernelID: 3], KernelObject[Name: local KernelID: 4]}

prerequisite run

In[25]:=
```
Graphics3D[Polygon /@ (faceListCoords /@ boundaryCells)]
```

Out[25]=



In[26]:=
```
(*missing boundary cells need to be found *)
```

In[27]:=
```
bcells = KeyTake[faceListCoords, boundaryCells];
```

In[28]:=
```
Length@boundaryCells
```

Out[28]= 60

In[29]:=
```
keyLs = Union@(Flatten@Lookup[vertexToCell,
          Lookup[ptsToIndAssoc,
           With[{ylim1 = yLim[[1]], ylim2 = yLim[[2]], xlim2 = xLim[[2]]},
             DeleteDuplicates@Cases[bcells,
                {x_ /; x ≥ xlim2, __} | {x_ /; x < 0, __} |
                 {_, y_ /; y > ylim2, _} | {_, y_ /; y ≤ ylim1, _}, {3}]
            ] /. periodicRules
          ]
        ] ~Join~ boundaryCells);
```

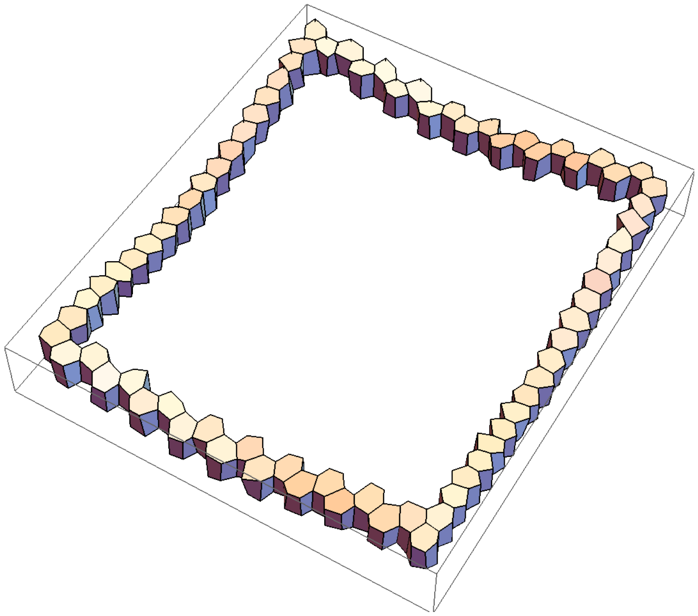
In[30]:=
```
Length[keyLs] - Length[boundaryCells]
```

Out[30]= 16

In[31]:=
```
border = faceListCoords /@ keyLs;
```

In[32]:=
```
Graphics3D[{Polygon /@ border}, ImageSize → Medium]
```

Out[32]=



In[33]:=
```
wrappedMatC = KeyTake[wrappedMat, keyLs];
```

In[34]:=
```
vertKeys = Keys@indToPtsAssoc;
```

In[35]:=
```
(
  topo = <|
    # → (getLocalTopology[ptsToIndAssoc, indToPtsAssoc, vertexToCell, cellVertexGrouping,
          wrappedMatC, faceListCoords][indToPtsAssoc[#]] // First) & /@ vertKeys
    |>;
 ) // AbsoluteTiming
```

Out[35]= {3.76305, Null}

Growing/Static cells

randomly select cells in the mesh to grow

In[36]:=
```
cellIds = Keys@cellVertexGrouping;
```

In[37]:=
```
fractionPopulation = 0.07;
growingcellIndices = RandomSample[cellIds, Round[fractionPopulation Length@cellIds]]
```

Out[38]= {266, 152, 71, 103, 101, 73, 108, 373, 215, 126, 94, 140, 385,
 95, 28, 382, 65, 290, 187, 304, 251, 248, 84, 116, 176, 313, 26, 115}

In[39]:=
```
nongrowingCellIndices = cellIds ~ Complement ~ growingcellIndices;
```

finding triangles connected to a vertex

In[40]:=
```
pointind = 1167;
```

In[41]:=
```
point = indToPtsAssoc@pointind;
```

In[42]:=
```
triangulatedCells = triangulateToMesh /@ faceListCoords;
polyhedraAssoc = Polyhedron@Flatten[#, 1] & /@ triangulatedCells;
```

In[44]:=
```
(trimesh = Map[triangulateToMesh, topo, {2}]); // AbsoluteTiming
```

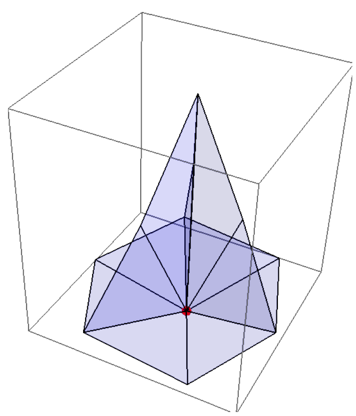
Out[44]=
```
{1.82182, Null}
```

In[45]:=
```
examplevertToTri =
   GroupBy[Flatten[Values@trimesh[#], 2], MemberQ[indToPtsAssoc[#]]][True] &[
    1]; // AbsoluteTiming
```

Out[45]=
```
{0.0003332, Null}
```

In[46]:=
```
(examplevertToTri =
    GroupBy[Flatten[Values@trimesh[#], 2], MemberQ[indToPtsAssoc[#]]][True];
   Graphics3D[{{Opacity[0.15], Blue, Triangle /@ examplevertToTri},
     Red, PointSize[0.03], Point@indToPtsAssoc[#]},
    ImageSize → Small]
  ) &[RandomInteger[Max@Keys@indToPtsAssoc]]
```

Out[46]=

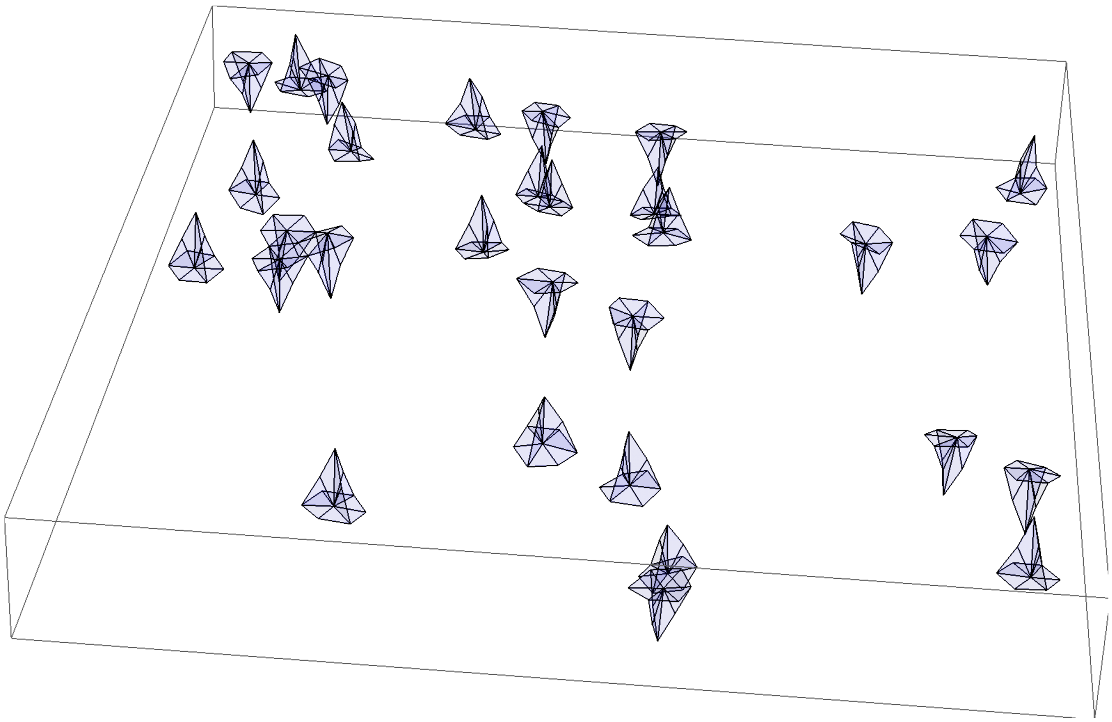


In[47]:=
```
(associatedtri = With[{ItoPA = indToPtsAssoc, tmesh = trimesh},
     AssociationThread[vertKeys, Function[vert, <|GroupBy[
            Flatten[#, 1], MemberQ[ItoPA[vert]]
          ][True] & /@ tmesh[vert]|>] /@ vertKeys]
    ];
 ) // AbsoluteTiming
```

Out[47]=
```
{0.659221, Null}
```

In[48]:=
```
SeedRandom[3];
Graphics3D[{Opacity[0.1], Blue, Triangle /@
   Flatten[Values@Values@RandomSample[associatedtri, 30], 2]}, ImageSize → Large]
```

Out[49]=



In[50]:=
```
(centTri = <|# → meanTri[Values[associatedtri@#]] & /@ Keys@indToPtsAssoc|>;) //
 AbsoluteTiming
```

Out[50]= {0.372214, Null}

In[51]:=
```
centTri = SetPrecision[#, 10] & /@ centTri;
```

In[52]:=
```
(normals = Map[SetPrecision[#, 10] &, triNormal@Values@# & /@ associatedtri]); //
 AbsoluteTiming
```

Out[52]= {0.516917, Null}

In[53]:=
```
(normNormals = Map[Normalize, normals, {3}];) // AbsoluteTiming
```

Out[53]= {0.105552, Null}

In[54]:=
```
(triangulatedmesh = triangulateToMesh /@ faceListCoords); // AbsoluteTiming
(polyhedra = Polyhedron@* (Flatten[#, 1] &) /@ triangulatedmesh;) // AbsoluteTiming
```

Out[54]= {0.175875, Null}

Out[55]= {0.001163, Null}

In[56]:=
```
(polyhedcent = RegionCentroid /@ polyhedra); // AbsoluteTiming
```

Out[56]= {4.1829, Null}

In[57]:=
```
(
  topoF = <|
    # → (getLocalTopology[ptsToIndAssoc, indToPtsAssoc, vertexToCell, cellVertexGrouping,
          wrappedMatC, faceListCoords][indToPtsAssoc[#]]) & /@ vertKeys
    |>;
) // AbsoluteTiming
```

Out[57]= {3.40155, Null}

In[58]:=
```
(keyslocaltopoF = Keys@*First /@ topoF); // AbsoluteTiming
```

Out[58]= {0.003706, Null}

In[59]:=
```
(shiftVecAssoc = Association /@ Map[Apply[Rule],
      Thread /@ Select[(#[[2 ;; 3]]) & /@ topoF, # ≠ {{}, {}} &], {2}]); // AbsoluteTiming
```

Out[59]= {0.0062151, Null}

In[60]:=
```
(cellcentroids = cellCentroids[polyhedcent, keyslocaltopoF, shiftVecAssoc]);
```

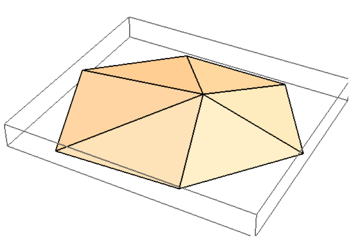
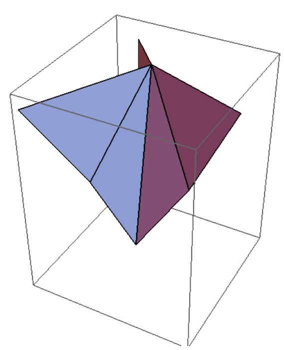
In[61]:=
```
(signednormals = AssociationThread[Keys@indToPtsAssoc,
     Map[
      MapThread[
        #2 Sign@MapThread[Function[{x, y}, (y - #1).x], {#2, #3}] &,
        {cellcentroids[#], normNormals[#], centTri[#]}] &, Keys@indToPtsAssoc]
     ]
  ); // AbsoluteTiming
```

Out[61]= {0.207126, Null}

make sets of open/closed triangles

In[62]:=
```
opencloseTri = Flatten[Values@#, 1] & /@ associatedtri;
```

In[63]:=
```
Graphics3D /@ Map[Triangle,
  GroupBy[GatherBy[opencloseTri[1], Intersection], Length, Flatten[#, 1] &], {2}]
```

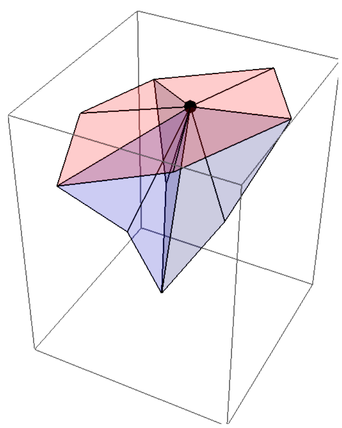
Out[63]= ⟨| 1 → , 2 →  |⟩

In[64]:=
```
triDistAssoc = Block[{trianglemembers},
   Map[
    (trianglemembers = #;
      GroupBy[GatherBy[trianglemembers, Intersection], Length, Flatten[#, 1] &]) &,
    opencloseTri]
   ];
```

In[65]:=
```
{opentri, closedtri} = {triDistAssoc[pointind][1], triDistAssoc[pointind][2]};
```

In[66]:=
```
Graphics3D[{{Opacity[0.2], Red, Map[Triangle][opentri], Blue, Map[Triangle][closedtri]},
  {Black, PointSize[0.04], Point@indToPtsAssoc[pointind]}}, ImageSize → Small]
```

Out[66]=



associate normals with triangles

In[67]:=
```
vertTriNormalpairings = <|
   # → <|Thread[Flatten[Values@associatedtri[#], 1] → Flatten[signednormals@#, 1]]|> & /@
    vertKeys|>;
```

To associate the open/closed triangles with their respective normals we simply need to perform a lookup in the association for (vertex1,vertex2,vertex3) - a triangle face - and its normal.

In[68]:=
```
normalsO = Lookup[vertTriNormalpairings[pointind], opentri];
```

In[69]:=
```
normalsC = Lookup[vertTriNormalpairings[pointind], closedtri];
```

In[70]:=
```
normalLs = normalsO~Join~normalsC;
```

volume gradient F[x]

gradient of volume is computed as: $1/3\,\Sigma\, A_{\Delta}\vec{N}$

In[71]:=
```
volumeGradient[point_, opentri_, closedtri_, normalLs_, cellids_,
   localtopology_, polyhedraAssoc_, growingCellIds_] :=
  Reap@Block[{topo, topology, normalassoc, gradV, gradVCont,
     triangulatedCellsSel, polyhedraSel, volume, growingIndkeys},
    triangulatedCellsSel = triangulateToMesh /@ localtopology;
    polyhedraSel = Lookup[polyhedraAssoc, cellids];
    topo =
     (Cases[#, x_ /; MemberQ[x, point]] &) @* (Flatten[#, 1] &) /@ triangulatedCellsSel;
    Sow[topo];
    normalassoc = AssociationThread[opentri~Join~closedtri, normalLs];
    gradV = Table[topology = topo[cell];
      (1.0 / 3.0)
       Total[Map[Area[Triangle[#]] * normalassoc[#] &, topology]], {cell, cellids}];
    volume = AssociationThread[cellids → ConstantArray[Vo, Length@cellids]];
    growingIndkeys =
     Replace[Intersection[cellids, growingCellIds], k_Integer :→ {Key[k]}, {1}];
    volume = If[growingIndkeys ≠ {},
      Values@MapAt[(1 + Vgrowth time) # &, volume, growingIndkeys],
      Values@volume
     ];
    gradVCont = kcv Total[(Volume[polyhedraSel] / volume - 1) gradV]
   ];
```

In[72]:=
```
{cellids, localtopology} = Through[{Keys, Identity}[#]] &[First@topoF[pointind]];
```

In[73]:=
```
{res, pyramids} = volumeGradient[point, opentri, closedtri,
   normalLs, cellids, localtopology, polyhedraAssoc, growingcellIndices];
```

In[74]:=
```
Column@Through[{Length, Identity}[#]] &[res]
```

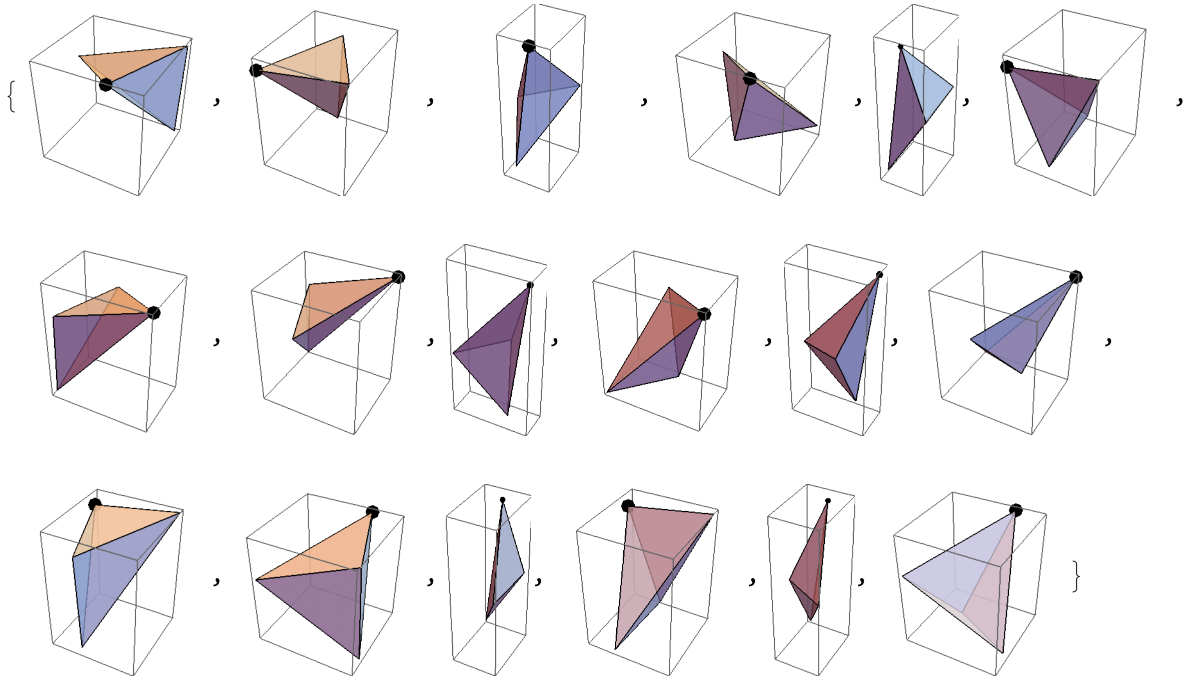
Out[74]=

3

$$\left\{k_{cv}\left(0.11384\left(-1+\frac{0.506575}{V_o}\right)-0.0956144\left(-1+\frac{0.588289}{V_o}\right)-0.0668358\left(-1+\frac{0.624122}{V_o}\right)\right),\right.$$
$$k_{cv}\left(-0.000830056\left(-1+\frac{0.506575}{V_o}\right)+0.11885\left(-1+\frac{0.588289}{V_o}\right)-0.123506\left(-1+\frac{0.624122}{V_o}\right)\right),$$
$$\left.k_{cv}\left(0.0482789\left(-1+\frac{0.506575}{V_o}\right)+0.0784126\left(-1+\frac{0.588289}{V_o}\right)+0.0646893\left(-1+\frac{0.624122}{V_o}\right)\right)\right\}$$

In[100]:=
```
plt = Graphics3D[{{Opacity[0.7], #}, {Black, PointSize[0.075], Point@point}},
    ImageSize → Tiny] & /@ Flatten@MapThread[Function[x, Tetrahedron@Join[{#}, x]] /@ #2 &,
    {cellcentroids[pointind], Values@pyramids[[1, 1]]}]
```

Out[100]=



In[101]:=
```
Show[plt, Graphics3D[{Blue, PointSize[0.04], Point@cellcentroids[pointind]}],
 ImageSize → Small]
```

Out[101]=