Gaussian and Mean Curvature

In[3]:=
```
ClearAll["Global`*"];
```

In[4]:=
```
Block[{range},
 posFn = Compile[{{triangleList, _Integer, 2}, {nvert, _Integer}},
   Position[triangleList, nvert]
   ,
   CompilationTarget → "C", RuntimeAttributes → {Listable}, RuntimeOptions → "Speed",
   Parallelization → True
  ]
]
```

Out[4]= CompiledFunction[Argument count: 2 Argument types: {{_Integer, 2}, _Integer}]

In[5]:=
```
faceNormfunc = Compile[{{tricoords, _Real, 2}},
  Cross[tricoords[[1]] - tricoords[[2]], tricoords[[3]] - tricoords[[2]]],
  CompilationTarget → "C", RuntimeAttributes → {Listable}, Parallelization → True
 ]
```

Out[5]= CompiledFunction[Argument count: 1 Argument types: {{_Real, 2}}]

In[6]:=
```
LaunchKernels[]

Clear[counter];
curvatureEstimatesimplequadric[mesh0_, nextnn_] := Module[{mesh = mesh0},
   (*Coordinates of Vertices on Mesh*)
  mc = MeshCoordinates[mesh];
   (*Number of vertices*)
  nvert = MeshCellCount[mesh, 0];
   (*Number of edges*)
  nedge = MeshCellCount[mesh, 1];
   (*Number of faces*)
  nfaces = MeshCellCount[mesh, 2];
   (*List of Edges,consisting of a list of pairs of vertex numbers*)
  edgeList = MeshCells[mesh, 1][[All, 1]];
   (*List of Triangles consisting of a list of triples of vertex numbers*)
  triangleList = MeshCells[mesh, 2][[All, 1]];
   (*Triangle Areas*)
  areaTriangles = PropertyValue[{mesh, 2}, MeshCellMeasure];
```

```
(*Length of Edges*)
edgeLengths = PropertyValue[{mesh, 1}, MeshCellMeasure];
Echo["Properties calculated"];
(*Positions of vertex i in the edge list (*SLOW*),Note this gives the
 edge index and either 1 or 2 depending on the order inside that edge*)
(*posinEdgeList=Position[edgeList,#]&/@Range[nvert];*)
posinEdgeList = posFn[edgeList, Range[nvert]];
(*Positions of vertex i in the triangle list
(*SLOW*),Note this gives the triangle index and either 1,
2 or 3 depending on the order inside that triangle*)
posinTriangleList = posFn[triangleList, Range[nvert]];
(*Number of nearest neighbours at each vertex*)
nearestneighbourList = Length /@ posinEdgeList;
(*Function that calculates for a given pair of vertex indices from a line i,
j,what triangles in the mesh indices also contain these indices,
output is the triangle index*)trilistfunc[line_] := Intersection[
  posinTriangleList⟦line⟦1⟧⟧⟦All, 1⟧, posinTriangleList⟦line⟦2⟧⟧⟦All, 1⟧];
(*List of triangle indices that are attached to each line,
This means that trianglesAtLines⟦k⟧ will return the indices of the triangles
 containing the line k (If only one index is returned we are on the boundary!)*)
trianglesAtLines = Map[trilistfunc, edgeList];
(*List of indices of edges that are on the boundary*)
boundaryedges = Flatten[Position[Length /@ trianglesAtLines, 1]];
(*List of indices of vertices that are on the boundary*)
boundaryvertices = Flatten[edgeList⟦#⟧ & /@ boundaryedges] // DeleteDuplicates;
(*Function that calculates which vertices in the attached triangles to
 a given edge are opposite to this edge,vertices are given as indices*)
(*Create Function to calculate mixed Voronoi Area (see paper for explanation)*)
trianglecoords = Map[mc⟦#⟧ &, triangleList];
(*faceNormalfunc[tricoords_]:=
 Cross[tricoords⟦1⟧-tricoords⟦2⟧,tricoords⟦3⟧-tricoords⟦2⟧];
facenormals=Map[faceNormalfunc,trianglecoords];*)
facenormals = faceNormfunc[trianglecoords];
temp = posinTriangleList⟦All, All, 1⟧;
mcnewcalc =
 Map[Total[Map[(facenormals⟦#⟧ * areaTriangles⟦#⟧) &, temp⟦#⟧]] &, Range[nvert]];
(*List of normalised normal vectors at each vertex*)
(*Nvectors=Map[(mcnewcalc⟦#⟧/Norm[mcnewcalc⟦#⟧])&,Range[nvert]];*)
Nvectors = mcnewcalc / (Norm /@ mcnewcalc);
(*Function to give the vertex indices of all the nearest
 neighbours j attached to vertex i by edges ij*)nneighbindexes[i_] :=
 Cases[Flatten[Map[edgeList⟦#⟧ &, posinEdgeList⟦i⟧⟦All, 1⟧]], Except[i]];
nextnneighbourindexes[i_] :=
 DeleteDuplicates[Flatten[Map[nneighbindexes[#] &, nneighbindexes[i]]]];
(*List of points to be fitted around vertex i*)
ptstofit[i_] := If[nextnn == 1, Join[{mc⟦i⟧}, Map[mc⟦#⟧ &, nneighbindexes[i]]],
  Join[{mc⟦i⟧}, Map[mc⟦#⟧ &, nextnneighbourindexes[i]]]];
```

```
(*The following calculates on next nearest
 neighbours (need to introduce code though inside this module?)*)
(*ptstofit[i_]:=Join[{mc⟦i⟧},Map[mc⟦#⟧&,nextnneighbourindexes[i]]];*)
(*calculation of points to fit in a rotated coordinate system aligned with
 the estimated normal and translated such that vertex i is at the origin*)
localcoordpointslist[i_] :=
 Map[RotationMatrix[{Nvectors⟦i⟧, {0, 0, 1}}].(# - mc⟦i⟧) &, ptstofit[i]];
lmmodelfit[i_] := LinearModelFit[localcoordpointslist[i], {1, x^2, x y, y^2}, {x, y}];

Echo["model fitting in progress"];
lmmodelfits = Map[(
    counter = #;
    If[MemberQ[boundaryvertices, #], 0, lmmodelfit[#]]) &, Range[nvert]
 ];
(*{gaussc,meanc}=Transpose@Map[
    (counter=#;
      If[MemberQ[boundaryvertices,#],0,
       bestfitparams=lmmodelfits⟦#⟧["BestFitParameters"];
       {(4*bestfitparams⟦2⟧*bestfitparams⟦4⟧-bestfitparams⟦3⟧^2),
        bestfitparams⟦2⟧+bestfitparams⟦4⟧}])&,Range[nvert]
   ];*)
bestfitparams = lmmodelfits⟦All, 1, 2⟧;
gaussc = 4 * bestfitparams⟦All, 2⟧ * bestfitparams⟦All, 4⟧ - bestfitparams⟦All, 3⟧^2;
meanc = bestfitparams⟦All, 2⟧ + bestfitparams⟦All, 4⟧;
Echo["Mean and Gaussian Curvature computed"];

(*eigenveccalcs=Map[If[MemberQ[boundaryvertices,#],0,
    Eigenvectors[{{2*lmmodelfits⟦#⟧["BestFitParameters"]⟦2⟧,lmmodelfits⟦#⟧[
          "BestFitParameters"]⟦3⟧},{lmmodelfits⟦#⟧["BestFitParameters"]⟦3⟧,
       2*lmmodelfits⟦#⟧["BestFitParameters"]⟦4⟧}}]]&,Range[nvert]];
ev1=Map[If[MemberQ[boundaryvertices,#],{0,0,0},
    RotationMatrix[{{0,0,1},Nvectors⟦#⟧}].(eigenveccalcs⟦#⟧⟦1,1⟧*{1,0,0}+
       eigenveccalcs⟦#⟧⟦1,2⟧*{0,1,0})]&,Range[nvert]];
ev2=Map[If[MemberQ[boundaryvertices,#],{0,0,0},
    RotationMatrix[{{0,0,1},Nvectors⟦#⟧}].(eigenveccalcs⟦#⟧⟦2,1⟧*{1,0,0}+
       eigenveccalcs⟦#⟧⟦2,2⟧*{0,1,0})]&,Range[nvert]];
(*Perhaps do this in the eigensystem to speed up*)
evals=Map[If[MemberQ[boundaryvertices,#],{0,0},
    Eigenvalues[{{2*lmmodelfits⟦#⟧["BestFitParameters"]⟦2⟧,lmmodelfits⟦#⟧[
          "BestFitParameters"]⟦3⟧},{lmmodelfits⟦#⟧["BestFitParameters"]⟦3⟧,
       2*lmmodelfits⟦#⟧["BestFitParameters"]⟦4⟧}}]]&,Range[nvert]];*)

eM = Flatten[k /. {Quiet@NSolve[1 / (1 + Exp[- k Min[meanc]]) == 10^-6, k],
     Quiet@NSolve[1 / (1 + Exp[- k Max[meanc]]) == 0.9999999, k]}] // Mean;
eG = Flatten[k /. {Quiet@NSolve[1 / (1 + Exp[- k Min[gaussc]]) == 10^-6, k],
     Quiet@NSolve[1 / (1 + Exp[- k Max[gaussc]]) == 0.9999999, k]}] // Mean;
```

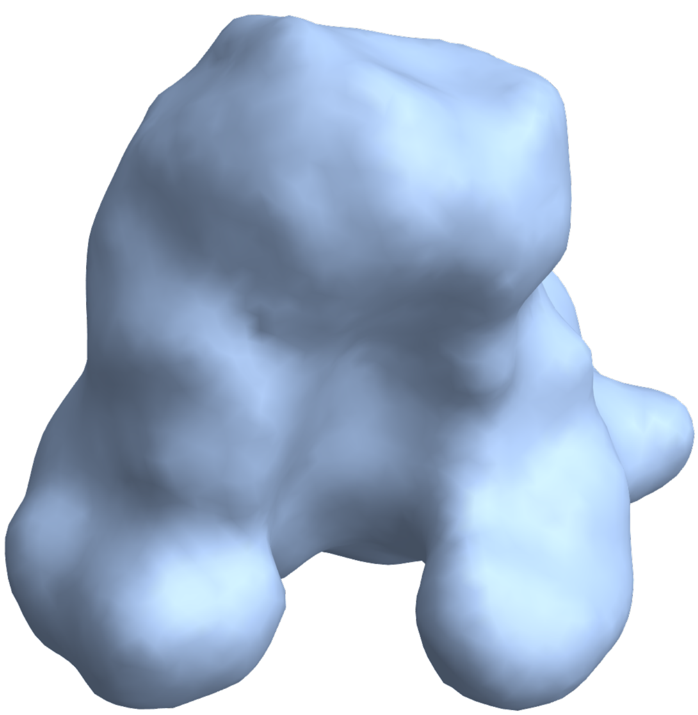
```
Print@Grid@List[Graphics3D[GraphicsComplex[mc, {EdgeForm[], Polygon[triangleList]},
      VertexColors → Map[ColorData[#], LogisticSigmoid[eM meanc]]], Boxed → False,
     Lighting → "Neutral", PlotLabel → "estimated mean", ImageSize → Small] & /@
   {"TemperatureMap", "GreenPinkTones"}];

Print@Grid@List[Graphics3D[GraphicsComplex[mc, {EdgeForm[], Polygon[triangleList]},
      VertexColors → Map[ColorData[#], LogisticSigmoid[eG gaussc]]],
     Boxed → False, Lighting → "Neutral", PlotLabel → "estimated gaussian",
     ImageSize → Small] & /@ {"TemperatureMap", "GreenPinkTones"}];

{nvert, mc, triangleList, meanc, gaussc, ev1, ev2, evals}]
```

Out[6]=
```
{KernelObject[ Local KernelID: 1 ], KernelObject[ Local KernelID: 2 ],
 KernelObject[ Local KernelID: 3 ], KernelObject[ Local KernelID: 4 ],
 KernelObject[ Local KernelID: 5 ], KernelObject[ Local KernelID: 6 ]}
```

```
(*
one can import the mesh using Import[filename]
*)
```

In[9]:=
```
mesh = Import["C:\\Users\\hashmial\\Downloads\\For Ali\\For Ali\\Cell3.ply"]
```

Out[9]=



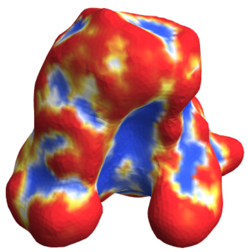
In[10]:=
```
Monitor[curvatureEstimatesimplequadric[mesh, 2], counter];
```

» Properties calculated

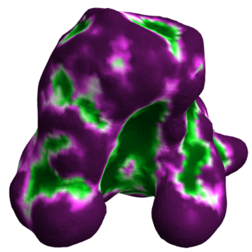
» model fitting in progress

» Mean and Gaussian Curvature computed

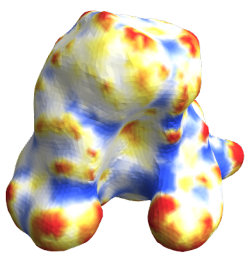
estimated mean



estimated mean



estimated gaussian



estimated gaussian