

Mesher Trona Pinnacles Spire

“P006”

This notebook processes the site pointcloud to come up with a mesh of hexahedrons, thus converting the pointcloud in a 3D solid geometry.

© Joaquin Garcia-Suarez (2021)

Load data

The original pointcloud was pre-filtered to remove most of the points that do not belong to the spire. **dirName** must be the name of the directory where the file “selected_points _PN06.csv” is located. This file is provided in the release zip.

```
(*dirName=;
SetDirectory[dirName]*)
```

```
In[3]:= data = Import["selected_points_PN06.csv"];
```

```
center = (Take[#, Ordering[Last /@ #, -1]] &@data) [[1]];
```

Keep only data around the pinnacle:

```
In[5]:= data2 = Select[data, (Abs[#[[1]] - center[[1]]] \leq 1.5 && Abs[#[[2]] - center[[2]]] \leq 1.5) &];
```

Center points:

```
In[6]:= auxTab = Transpose[Transpose[data2] - Mean[data2]];
```

Remove extra points (below supporting points and detached ones) and consolidate

Assign the length of the mesh element (**L**):

```
In[7]:= L = 0.1;
```

Consolidate points at heights that are multiple of L:

```
In[8]:= orderedList2 = auxTab;
auxRoundZ = Round[#[[3]], L] & /@ auxTab;
(*auxRoundZ=Floor[auxTab[[#]][[3]]&/@Range[Length[auxTab]]*(1/\[DeltaZ])]*\[DeltaZ];*)
Do[
  orderedList2[[jj]][[3]] = auxRoundZ[[jj]],
  {jj, 1, Length[auxTab]}];
```

Supporting plane:

```
In[11]:= hsp = 0.;
SupportingPlane = InfinitePlane[{{0., 0., hsp}, {0., 1., hsp}, {1., 0., hsp}}];

In[12]:= auxTab2 = auxTab;
auxTab = Select[orderedList2, (#[[3]] >= hsp) &];

Remove surrounding rocks:

In[14]:= ymax = 0.85; (*position to cut chaff*)
auxTab3 = auxTab;
auxTab = Select[auxTab3, (#[[2]] <= ymax) &];

Visualize the data we will work with henceforth:

In[17]:= auxTab4 = Reverse@GatherBy[auxTab, Last];

In[19]:= Graphics3D[{
Yellow, Point[auxTab], PointSize -> Medium}], Boxed -> False]
```

Out[19]=



Interpolate data at each level of the pinnacle

Transform the points from Cartesian to Cylindrical.

```
In[20]:= auxCil = SortBy[CoordinateTransform["Cartesian" -> "Cylindrical", auxTab], First];
```

The data we need to use is already available in **auxTab**, consolidated at every level (for z=constant) and expressed in cylindrical coordinates in **auxCil**, **auxCil2** is grouped at height level:

```
In[21]:= auxCil2 = GatherBy[SortBy[auxCil, Last], Last];
```

Number of levels:

In[22]:= Length@auxCil2

Out[22]= 31

Loop to generate interpolators:

The following loop generates all the information about the cross-sections that we will later need to fit the mesh to them. The way that is done is by:

- (1) building a spline through the points in each cross-sections,
- (2) smooth out this spline by sampling points out and then take a moving average (window average),
- (3) build an interpolant out of those points.

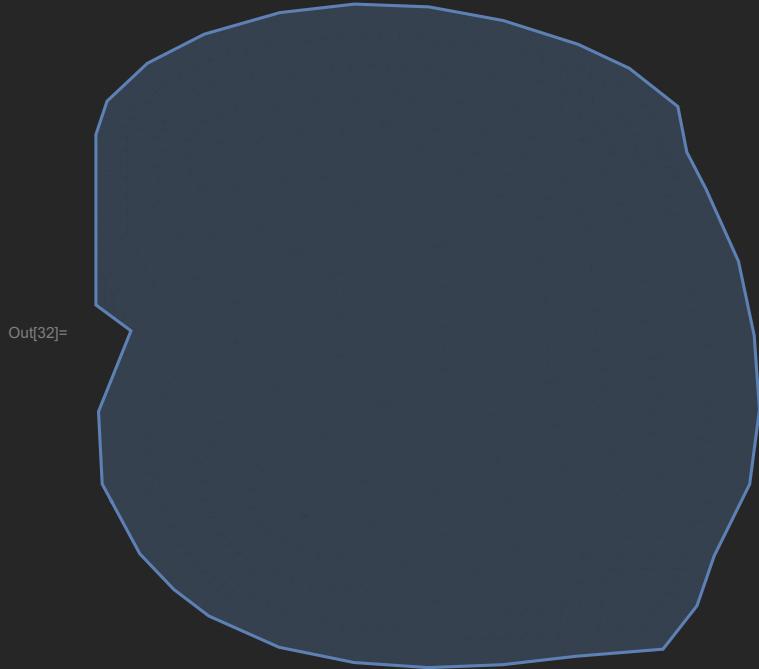
Including lists to save the interpolators:

```
n = 3; (* B-spline degree *)
funList = Array[, Length[auxCil2]];
regionList = Array[, Length[auxCil2]];
interpList = Array[, Length[auxCil2]];
auxCartList = Array[, Length[auxCil2]];
Do[
  (*Order cross-sections according to angle*)
  auxR = SortBy[auxCil2[[ii]][[All, {2, 1}]], First];
  (*Create interpolation*)
  fun = BSplineFunction[auxR, SplineDegree → n, SplineClosed → True];
  funList[[ii]] = fun;
  (*Create the implicit region*)
  (*data: spaced evenly, 1000 points*)
  data = fun[#] & /@ Subdivide[0, 0.99, 1000];
  (*smooth cross-sections: every 50 points*)
  filtAng = MovingAverage[#[[1]] & /@ data, 50];
  (*Close the cross-section*)
  filtAng = AppendTo[filtAng, N[π]]; filtAng = Prepend[filtAng, -N[π]];
  filtR = MovingAverage[#[[2]] & /@ data, 50];
  (*Close the cross-section*)
  filtR = AppendTo[filtR, filtR[[1]]]; filtR = Prepend[filtR, filtR[[1]]];
  interp = Interpolation[Transpose@{filtAng, filtR}, InterpolationOrder → 1];
  (*Save region and interpolant*)
  regionList[[ii]] = ImplicitRegion[x^2 + y^2 < interp[ArcTan[x, y]]^2, {x, y}];
  interpList[[ii]] = interp;
  (*Save point lists to find nearest points later*)
  auxCartList[[ii]] = CoordinateTransform["Polar" → "Cartesian",
    Flatten[Transpose@{{interp[#], #} & /@ Subdivide[-π * 0.999, π * 0.999, 500]}, 1]];
  , {ii, 1, Length[auxCil2]}] // Timing
```

Out[28]= {0.46875, Null}

The following extracts the basal section geometry and prepares to be used later to generate the stress figures (see notebook called "stresses_base_spire.nb").

```
In[31]:= regionOneScaled =
  ImplicitRegion[ \left( \frac{x}{10^0} \right)^2 + \left( \frac{y}{10^0} \right)^2 < interpList[[1]][ArcTan[\frac{x}{10^0}, \frac{y}{10^0}]]^2, {x, y}];
RegionPlot[regionOneScaled, Frame -> False]
```



```
In[32]:= SystemOpen[DirectoryName[AbsoluteFileName["regionOneScaled.mx"]]]
```

Visualize the interpolation

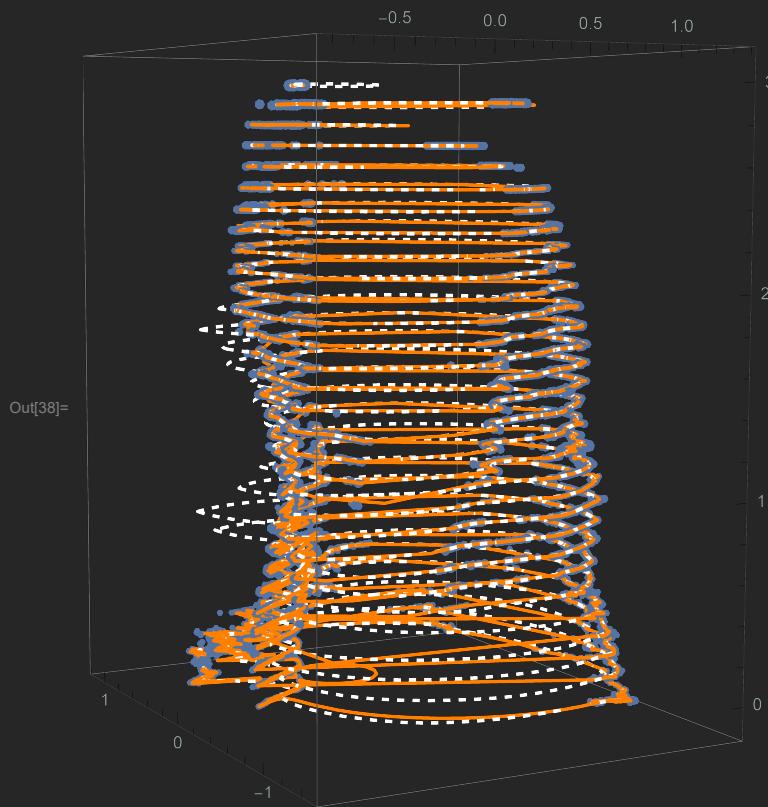
```
In[33]:= auxTab4 = Reverse@GatherBy[auxTab, Last];
orderedList3 = GatherBy[auxTab, Last];
```

Create list for figures and fill them up:

```
In[35]:= auxFigures = Table[ , {Length[auxCil2]}];
Do[
  auxp1 = ParametricPlot3D[
    {funList[[jj]][t][[2]] * Cos[funList[[jj]][t][[1]]], funList[[jj]][t][[2]] *
     Sin[funList[[jj]][t][[1]]], hsp + L * (jj - 1)}, {t, 0, 1}, PlotStyle -> Orange];
  auxp2 = ListPointPlot3D[orderedList3[[jj]], AspectRatio -> Automatic,
    ColorFunction -> Automatic];
  auxp3 = ParametricPlot3D[{interpList[[jj]][t] * Cos[t], interpList[[jj]][t] * Sin[t],
    hsp + L * (jj - 1)}, {t, -\pi, \pi}, PlotStyle -> {White, Dashed}];
  auxFigures[[jj]] = Show[auxp1, auxp2, auxp3]
  , {jj, 1, Length[auxCil2]}] // Timing
```

```
Out[36]= {1.125, Null}
```

```
In[38]:= Show[#, ImageSize -> Medium, PlotRange -> All] &@auxFigures
```

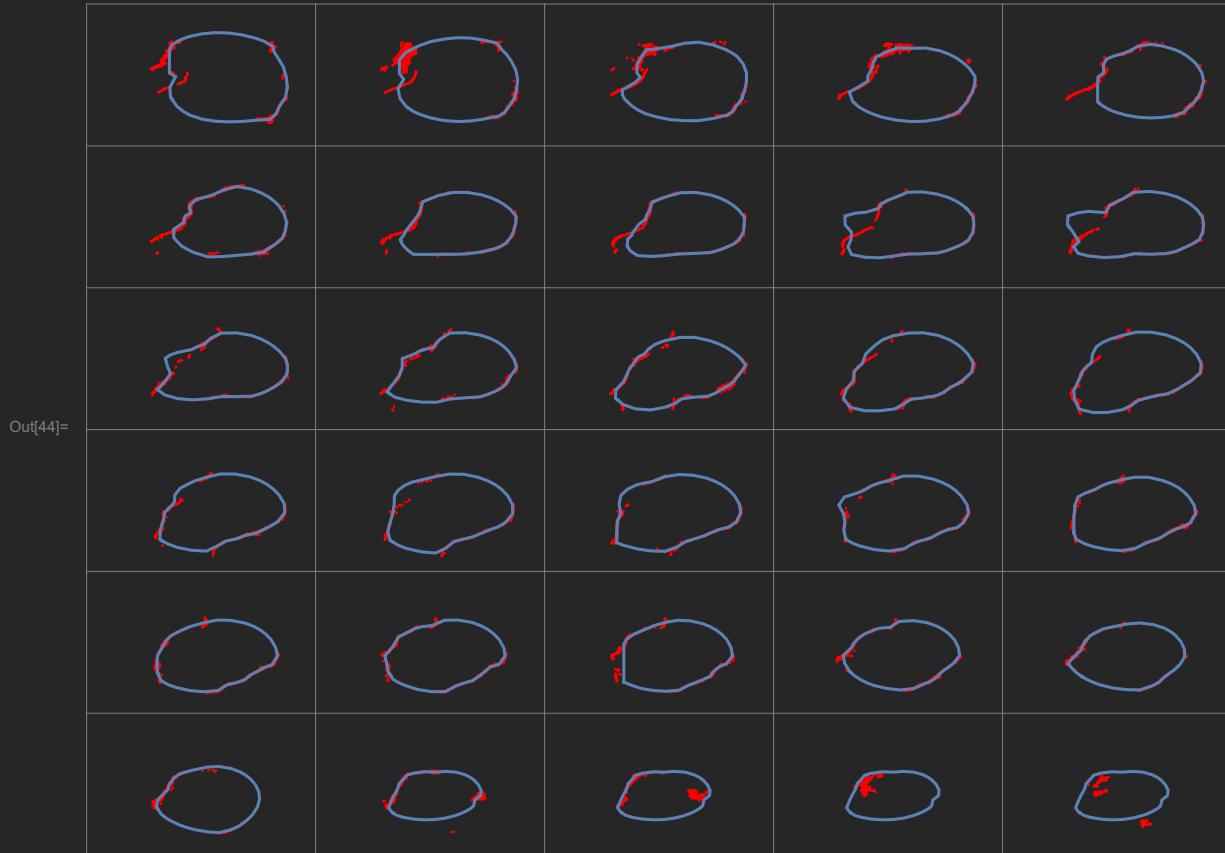


Fix bad interpolation:

```
In[39]:= regionList[[-4]] = regionList[[-5]];
regionList[[-3]] = regionList[[-5]];
regionList[[-2]] = regionList[[-5]];
```

Visualize sections that we have interpolated:

```
In[42]:= auxp4 = Table[ , {Length@regionList}] ;
Do[
  auxp4[[jj]] =
  Show[ListPlot[auxTab4[[jj]][[All, {1, 2}]], PlotStyle -> Red, Axes -> False],
  RegionPlot[regionList[[jj]],
  PlotStyle -> None,
  Frame -> False,
  Epilog -> Inset["z =" <> ToString[(jj - 1) * L], {0.5, 05}],
  PlotRange -> {{-1.5, 1.5}, {-1.5, 1.5}}]
, {jj, 1, Length@regionList}
];
toExport = GraphicsGrid[ArrayReshape[auxp4[[1 ;; -1]], {6, 5}],
Frame -> All, FrameStyle -> Directive[Gray]]
```



Generate the mesh of the pinnacle

Point Grid

Get dimensions encompassing the body:

```
In[45]:= xMin = Min[auxTab[[All, {1}]]] - L;
xMax = Max[auxTab[[All, {1}]]] + L;
yMin = Min[auxTab[[All, {2}]]] - L;
yMax = Max[auxTab[[All, {2}]]] + L;
zMin = Min[auxTab[[All, {3}]]] - L;
zMax = Max[auxTab[[All, {3}]]] + L;
```

How many units?

```
In[51]:= xUnits = Ceiling[(xMax - xMin) / L]
```

```
Out[51]= 23
```

```
In[52]:= yUnits = Ceiling[(yMax - yMin) / L]
```

```
Out[52]= 25
```

```
In[53]:= zUnits = Ceiling[(zMax - hsp) / L]
```

```
Out[53]= 31
```

Create a basic array of points:

```
In[55]:= points2D =
  Flatten[Table[{xMin + (ii - 1) * L, yMin + (jj - 1) * L}, {ii, 1, xUnits}, {jj, 1, yUnits}], 1];
```

```
In[56]:= Npoints2D = Length[points2D];
```

Create the structures (Associations) needed to gather the node information

The elements in this lists are “associations”, which act as structures in C++: there two keys associated to values, “ID” is just the node ID and “XYZ” are just the coordinates (the Z coordinate) for the first level is just the Z coordinate of the supporting plane.

nodelist contains the first base level

```
In[57]:= nodesList = <|"ID" → #[[1]], "XYZ" → {#[[2]][[1]], #[[2]][[2]], hsp}|> & @
  Transpose@{Range[Length[points2D]], points2D};
```

pointlist will contain the upper nodes too

```
In[58]:= pointList = Table[#, {zUnits}];
```

```
In[59]:= pointList[[1]] = nodesList;
```

The following loop fills the rest of nodes:

```
In[60]:= Do[
  (*node counter*)
  nCount = (jj - 1) * xUnits * yUnits;
  (*Create auxiliary*)
  aux = nodesList;
  (*Update height*)
  aux[[All, 2, 3]] = # &@Table[hsp + (jj - 1) * L, {Length[nodesList]}];
  (*Update ID*)
  aux[[All, 1]] = # &@Table[nCount + jj, {jj, 1, Length[nodesList]}];
  (*Assign in the list*)
  pointList[[jj]] = aux;
  , {jj, 2, zUnits}]
```

From-ID-to-coordinates Function

This function maps the corresponding node ID (what is saved in the node connectivity key of each element) to its coordinates:

```
In[61]:= f[i_] := Select[Flatten[pointList], #[["ID"]] == i &][[1]][["XYZ"]]
```

Create scaffold of cubes:

Select the node IDs to pick:

```
In[62]:= nToPick = Delete[Range[yUnits * (xUnits - 1)], Map[## &, yUnits * Range[xUnits - 1]]];
```

The following list of associations (**cubelist**) contains all the elements of the base (where we do not have to filter)

```
In[63]:= cubeList = <|"elementID" → #[[1]], "Nodes" → {#[[2]], (*1*)
  #[[2]] + 1, (*2*)
  #[[2]] + yUnits + 1, (*3*)
  #[[2]] + yUnits, (*4*)
  #[[2]] + Npoints2D, (*5*)
  #[[2]] + Npoints2D + 1, (*6*)
  #[[2]] + Npoints2D + yUnits + 1, (*7*)
  #[[2]] + Npoints2D + yUnits}|> & /@
  Transpose@{Range[Length[nToPick]], nToPick};
```

Convert IDs to coordinates, using the function **f**: (this mapping is done in parallel rather quickly)

```
(*CloseKernels[];
LaunchKernels[2]*)
```

```
In[64]:= newCube = cubeList[[1]]
```

```
Out[64]= <| elementID → 1, Nodes → {1, 2, 27, 26, 576, 577, 602, 601} |>
```

```
In[65]:= level = 1;
newCube["Nodes"] = Map[## + (level - 1) * Npoints2D &, cubeList[[1]][["Nodes"]]]
```

```
Out[66]= {1, 2, 27, 26, 576, 577, 602, 601}
```

THE ACTUAL LOOP:

If necessary, in order to avoid running this loop again, load the data of the meshes at this point:

```
(*elementList=Import["PN06_elementList.mx"];
borderEleList=Import["PN06_border.mx"]*)
```

If desired, run the loop (w/o parallelization, it should about 60 mins when L=0.1.)

```
In[]:= borderEleList = Table[{}, {zUnits - 1}];
elementList = Table[{}, {zUnits - 1}];
Do[(*first, for each level*)
Print["Level=", level];
(*START DO LOOP*)
Do[(*second, in each level, for each element*)
(*Create new element to be checked*)
(*jj indexes the "base" element*)
newCube = cubeList[[jj]];
newCube["Nodes"] = Map[# + (level - 1) * Npoints2D &, cubeList[[jj]][["Nodes"]]];
(*Check elements*)
addElementQ = 1;
addEleBorderQ = 0;
(*by default, add; the loop will change this value to 0 if there are nodes outside*)
Which[
(*0) Check if {0,0} is any of the nodes, if so, take automatically*)
MemberQ[#[[1]] + #[[2]] & /@ Map[f, newCube["Nodes"]][[1 ;; 4]], 0.],
True, (*just take them*)
(*-----*)
(*1) Check 4 nodes within the region*)
RegionMember[regionList[[level]], {#[[1]], #[[2]]}] & /@
Map[f, newCube["Nodes"]][[1 ;; 4]] == {True, True, True, True},
True(*Inner ring, take*),
(*-----*)
(*2) At least one element in the region, move to border list*)
Length[Cases[RegionMember[regionList[[level]], {#[[1]], #[[2]]}] & /@
Map[f, newCube["Nodes"]][[1 ;; 4]], True]] > 0,
addElementQ = 0; addEleBorderQ = 1,
(*-----*)
(*3) All nodes out*)
True, addElementQ = 0
];
Which[
(*If the inner element is to be included*)
addElementQ == 1,
(*Update ID*)
newCube["elementID"] =
Length[Flatten[elementList]] + Length[Flatten[borderEleList]] + 1;
(*+Length[cubeList]*)]
```

```
(*Append new cube*)
elementList[[level]] = AppendTo[elementList[[level]], newCube];
,
addEleBorderQ == 1,
newCube["elementID"] =
Length[Flatten[elementList]] + Length[Flatten[borderEleList]] + 1;
(*+Length[cubeList]*)
borderEleList[[level]] = AppendTo[borderEleList[[level]], newCube];
,
(*It is inside, then add and keep: first add ID and then include conveniently*)
True,
],
{jj, 1, Length[cubeList]}]
, {level, 1, zUnits - 1}];
```

The following maps the node IDs to coordinates, which are later used to generate the plots:

```
In[1]:= coordList2 = Parallelize[Map[f, Flatten[elementList][[#]][["Nodes"]]] & /@
Range[Length[Flatten[elementList]]]]; // AbsoluteTiming
Out[1]= {57.5097, Null}

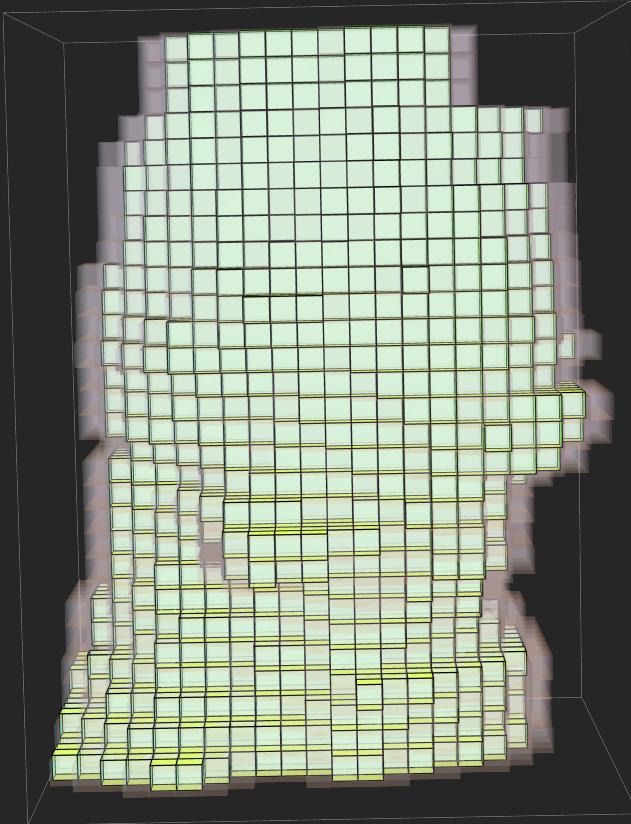
(*LaunchKernels[2];*)
coordList2 = Parallelize[Map[f, Flatten[elementList][[#]][["Nodes"]]] & /@
Range[Length[Flatten[elementList]]]];
CloseKernels[];

(*LaunchKernels[2];*)
coordFront = Parallelize[
Map[f, Flatten[borderEleList][[#]][["Nodes"]]]
& /@ Range[10]
]; // Timing

In[2]:= coordFront = Map[f, Flatten[borderEleList][[#]][["Nodes"]]] & /@ Range[10]; // Timing
Out[2]= {1.1875, Null}
```

Prepare cubic elements and display them (green = bulk elements, shaded pink = frontier)

```
In[6]:= plotcube3 =
  Graphics3D[{EdgeForm[None], Opacity[0.15], Hexahedron[##] & /@ coordFront[[1 ;; -53]]}];
plotcube2 = Graphics3D[{Glow[Green], Opacity[0.85],
  Hexahedron[##] & /@ coordList2[[1 ;; -100]]}];
Show[plotcube2, plotcube3]
```



Correct frontier nodes

Finally, the following loop takes or discards the frontier nodes, and moves the nodes accordingly in order to fit them to the boundary.

```
In[7]:= Do[(*For each level*)
  Print["Level=", level];
  auxInt = 0.5 * auxCartList[[level + 1]] + 0.5 * auxCartList[[level]];
  (*auxiliary samples points*);
  (*-----*)
  Do[(*For each element on the frontier*)
    eleToCheck = borderEleList[[level]][[jj]][[2]];
    (*Check which nodes are Inside*)
```

```

posOut = Flatten[Position[RegionMember[regionList[[level]], {#[[1]], #[[2]]}]] & @
    Map[f, eleToCheck[[1 ;; 4]], False]];
(*This vector contains the position of the points that are inside*)
(*-----*)
If[Length[posOut] == 1, (*1 node out,
    move the outer one and add the new element to the list*)
(*-----*)
nodeIDsToCheck = eleToCheck[[posOut[[1]]]];
pointList[[Ceiling[ $\frac{\text{nodeIDsToCheck}}{\text{Npoints2D}}$ ]]][[nodeIDsToCheck -
    Npoints2D
    Npoints2D * Floor[ $\frac{\text{nodeIDsToCheck}}{\text{Npoints2D}}$ ]]][["XYZ"]][[{1, 2}]] = First@Nearest[
    auxCartList[[level]], Flatten[{#[[1]], #[[2]]}] & @ Map[f, {nodeIDsToCheck}]]];
(*the previous chunk updates the node position,
the following lines update the list adding the new border elements*)
(*Update list of nodes in the elements (done in this way takes no time) *)
presentNodes = DeleteDuplicates[Flatten[Lookup[Flatten[elementList], "Nodes"]]];
(*Compare nodes of the element, if there are some that are not in the list,
project them to the nearest intermediate*)
If[Length@Flatten[Position[MemberQ[presentNodes, #] & @ eleToCheck, False]] > 0,
    (*if there are elements out of the list*)
    posMove = Flatten[Position[MemberQ[presentNodes, #] & @ eleToCheck, False]];
    (*position of the elements to move*)
    Do[
        nodeIDsToCheck = eleToCheck[[posMove[[nn]]]];
        (*update first two components*)
        pointList[[Ceiling[ $\frac{\text{nodeIDsToCheck}}{\text{Npoints2D}}$ ]]][[
            nodeIDsToCheck - Npoints2D * Floor[ $\frac{\text{nodeIDsToCheck}}{\text{Npoints2D}}$ ]]][["XYZ"]][[{1, 2}]] =
            First@Nearest[auxInt, Flatten[{#[[1]], #[[2]]}] & @ Map[f, {nodeIDsToCheck}]]];
        (*update height*)
        pointList[[Ceiling[ $\frac{\text{nodeIDsToCheck}}{\text{Npoints2D}}$ ]]][[nodeIDsToCheck -
            Npoints2D
            Npoints2D * Floor[ $\frac{\text{nodeIDsToCheck}}{\text{Npoints2D}}$ ]]][["XYZ"]][[{3}]] = hsp + level * L -  $\frac{L}{2}$ ;
        , {nn, 1, Length[posMove]}]
    , True];
    elementList[[level]] =
        AppendTo[elementList[[level]], borderEleList[[level]][[jj]]];
    (*-----*)
    (*else, there are two or more nodes out, then move the inner nodes to the border*)
    posIn = DeleteCases[{1, 2, 3, 4}, Alternatives @@ posOut];
    Do[
        nodeIDsToCheck = eleToCheck[[posIn[[nn]]]];
        pointList[[Ceiling[ $\frac{\text{nodeIDsToCheck}}{\text{Npoints2D}}$ ]]][[nodeIDsToCheck -

```

```

Npoints2D * Floor[ $\frac{\text{nodeIDsToCheck}}{\text{Npoints2D}}$ ]]][["XYZ"]][[{1, 2}]] = First@Nearest[
  auxCartList[[level]], Flatten[{#[[1]], #[[2]]}] & /@ Map[f, {nodeIDsToCheck}]]]
, {nn, 1, Length[posIn]}]
(*-----*)
]
(*-----*)
(*Next element in the level*)
, {jj, 1, Length[borderEleList[[level]]]}];
(*Length[borderEleList[[level]]]*)]
(*-----*)
-----),
{level, 1, Length@elementList}];

```

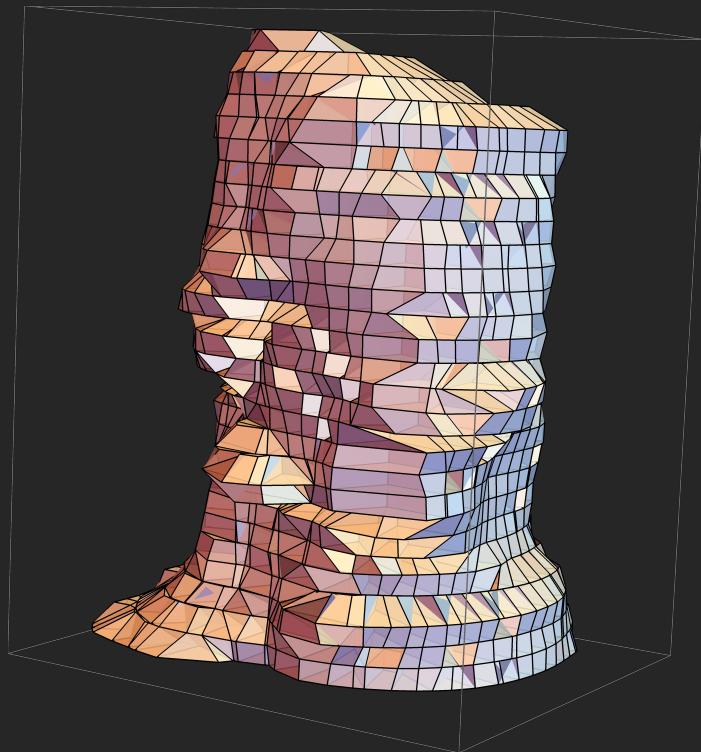
Visualize almost-ready mesh:

```

In[6]:= coordList2 = Parallelize[Map[f, Flatten[elementList] [[#]] ][["Nodes"]]] & /@
  Range[Length[Flatten[elementList[[1 ;; -2]]]]];
plotcube2 = Graphics3D[{Opacity[0.85], Hexahedron[#[{"x", "y", "z"}] & /@ coordList2}]];
Show[plotcube2]

```

Out[6]=



Compare mesh to original pointcloud:

The following is similar to Figure 7 panel (a) in the article.

```
In[6]:= Graphics3D[ {
  {Glow[Green], Opacity[0.95], Hexahedron[#[<]& /@ coordList2[[1 ;; -1]]]},
  {Red, Point[auxTab2], PointSize → Medium} }
, Boxed → False]
```

