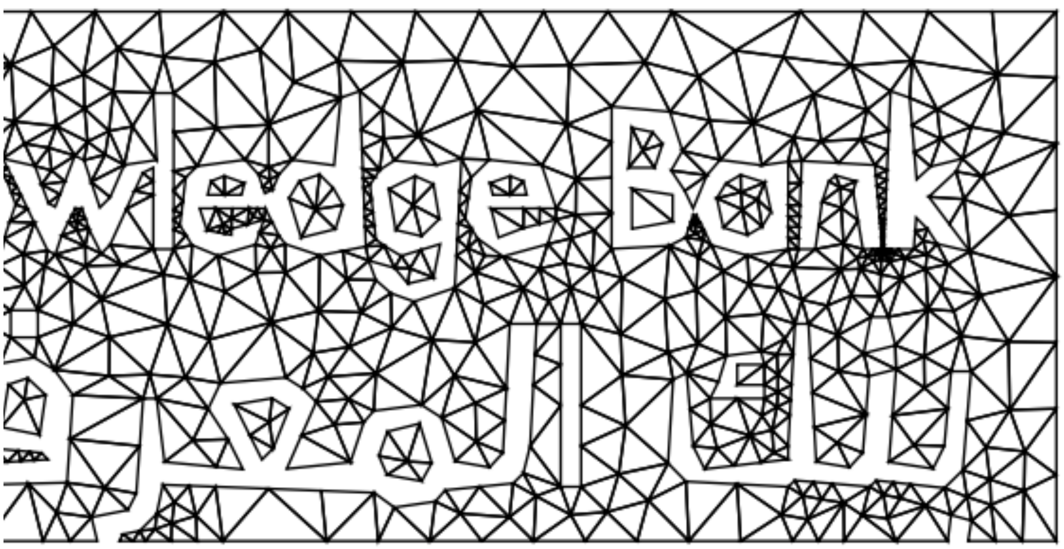
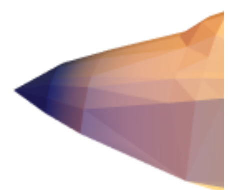
WOLFRAM U

Finite Element Programming with the Wolfram Language
Part: 2





M. K. AbdElrahman

Wolfram Research

Last Time

Introduction

Finite Element Data within NDSolve

Passing Finite Element Options to NDSolve

A Workflow Overview

The Partial Differential Equation Problem Setup

Stationary PDEs

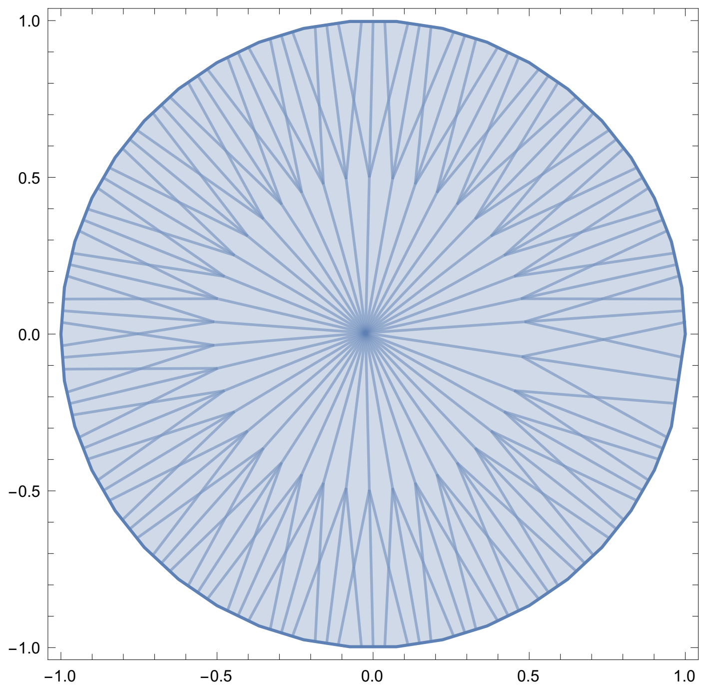
Last Time

In[24]:=
```
Needs["NDSolve`FEM`"]
```

Define Region:

In[38]:=
```
Ω= Disk[];
Show[RegionPlot [Ω]]
```

Out[39]=



Define PDE:

In[27]:=
```
Epsilon[x_,y_]:= 1+x^2+y^2
pde = PoissonPDEComponent [{u[x,y],{x,y}},<|"PoissonSourceTerm "→Epsilon[x,y]|>]==0
ΓD= {DirichletCondition [u[x,y]==0.0,True]};
```

Out[28]= $-1-x^2-y^2+\nabla_{\{x,y\}}\cdot(\{\{1,0\},\{0,1\}\}.\nabla_{\{x,y\}}u[x,y]) == 0$

ProcessEquation:

In[30]:=
```
{dpde,dbc,vd,sd,mdata}=ProcessPDEEquations [{pde,ΓD},u,{x,y}∈Ω];
```

Discretize PDE

In[40]:=
```
{load,stiffness,damping,mass}=dpde["All"];
```

Discretize the boundary conditions.

In[41]:=
```
DeployBoundaryConditions [{load ,stiffness },dbc];
```

Linear Solve

In[43]:=
```
solution = LinearSolve [stiffness ,load];
```

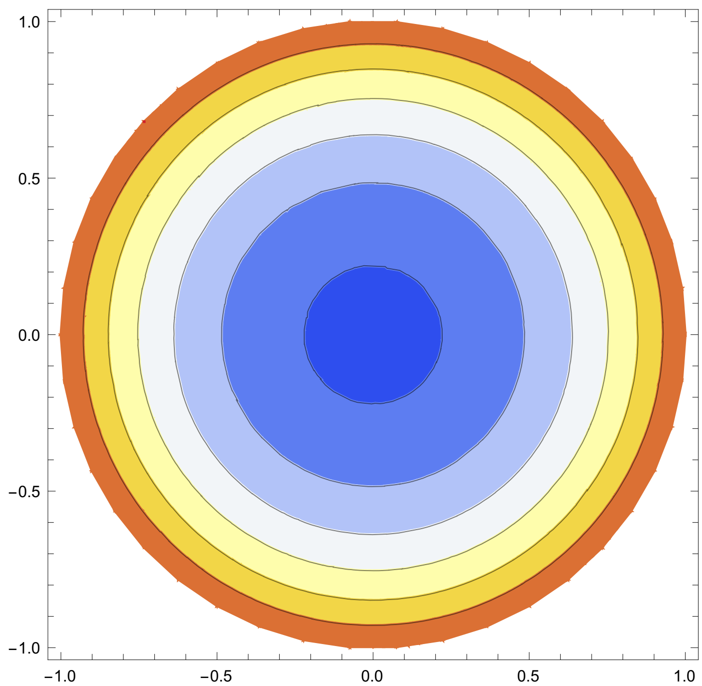
Save Solution

In[44]:=
```
NDSolve`SetSolutionDataComponent [sd,"DependentVariables ",Flatten [solution ]];
```

Create an InterpolatingFunction object.

In[45]:=
```
mesh = mdata["ElementMesh "];
ifun=ElementMeshInterpolation [mesh, solution ];
ContourPlot [ifun[x,y],{x,y}∈Ω,PlotRange →All,ColorFunction →"TemperatureMap "]
```

Out[47]=



Outline

Transient PDEs

The first example is a heat equation in 1D. Consider the following model PDE

$$\frac{\partial}{\partial t} u + \nabla \cdot (- \nabla u) = 1$$

in a spatial region from 0 to 1 and a time domain from 0 to 1. Boundary and initial conditions are 0 everywhere:

In[48]:=
```
<< NDSolve`FEM`
```

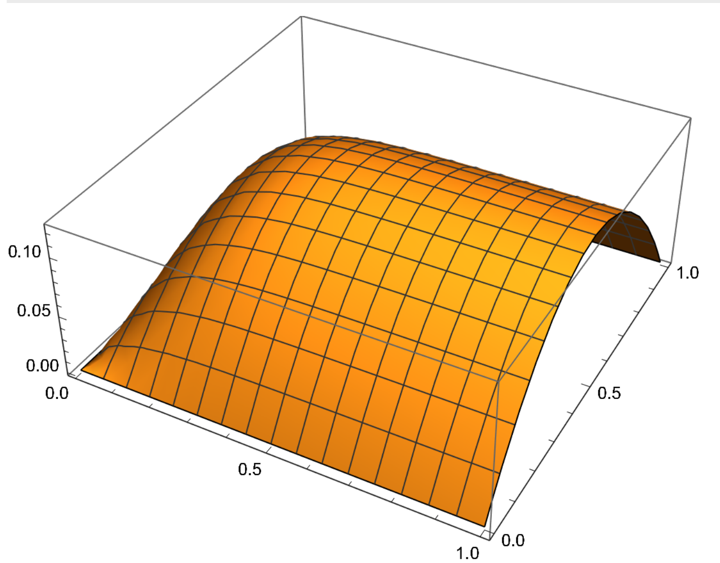
In[50]:=
```
ufun=NDSolveValue [{D[u[t,x],{t,1}]-D[u[t,x],{x,2}]==1,u[0,x]==0,DirichletCondition [u[t,x]==0,
True]},u,{t,0,1},{x,0,1},
Method →{"PDEDiscretization "→{"MethodOfLines ","SpatialDiscretization "->{"FiniteElement "}}}]
```

Out[50]=
```
InterpolatingFunction [ Domain : {{0., 1.}, {0., 1.}}  Output : scalar ]
```

Plot the numerical solution:

In[51]:=
```
Plot3D [ufun[t,x],{t,0,1},{x,0,1}]
```

Out[51]=



First, the variable data is created and populated:

In[54]:=
```
vd=NDSolve`VariableData [{"DependentVariables "→{u},"Space "→{x},"Time "→t}]
```

Out[54]=
```
{t, {x}, {u}, {}, {}, {}, {}, {}}
```

Specify a **NumericalRegion**:

In[56]:=
```
nr=ToNumericalRegion [FullRegion [1],{{0,1}}]
```

Out[56]= NumericalRegion [FullRegion [1], {{0, 1}}]

Create the solution data with the **"Space"** and **"Time"** components set:

In[57]:=
```
sd=NDSolve`SolutionData [{"Space"→nr,"Time"→0.}]
```

Out[57]= {0., NumericalRegion [FullRegion [1], {{0, 1}}], {}, {}, {}, {}, {}, {}}

Initialize the partial differential equation coefficients:

In[58]:=
```
initCoeffs =InitializePDECoefficients [vd,sd,"DiffusionCoefficients "→
{{-IdentityMatrix [1]}},"LoadCoefficients "→{{1}},"DampingCoefficients "->{{1}}]
```

Out[58]= PDECoefficientData [<1,1>]

Initialize the boundary conditions:

In[59]:=
```
initBCs =InitializeBoundaryConditions [vd,sd,{{DirichletCondition [u[t,x]==0,True]}}]
```

Out[59]= BoundaryConditionData [<1,1>]

Initialize the finite element data with the variable and solution data:

In[60]:=
```
methodData =InitializePDEMethodData [vd,sd]
```

Out[60]= FEMMethodData [<41,{2},4>]

Extract the **ElementMesh** from the NumericalRegion:

In[61]:=
```
mesh=nr["ElementMesh "]
```

Out[61]= ElementMesh [{{0., 1.}}, {LineElement [<20>]}]

Compute the discretized partial differential equation:

In[62]:=
```
discretePDE =DiscretizePDE [initCoeffs ,methodData ,sd]
```

Out[62]= DiscretizedPDEData [<41>]

Discretize the initialized boundary conditions:

In[63]:=
```
discreteBCs =DiscretizeBoundaryConditions [initBCs ,methodData ,sd]
```

Out[63]= DiscretizedBoundaryConditionData [<41>]

Extract all system matrices:

In[64]:=
```
{load,stiffness ,damping ,mass}=discretePDE ["SystemMatrices "];
```

Deploy the boundary conditions in place:

In[65]:=

```
DeployBoundaryConditions [{load ,stiffness ,damping },discreteBCs ]
```

Out[65]=

```
DeployedBoundaryConditionData [<Insert>]
```

Set up initial conditions based on the boundary conditions:

In[68]:=

```
init=Table[{0.},{methodData ["DegreesOfFreedom "]}];
init[[discreteBCs ["DirichletRows "]]]=discreteBCs ["DirichletValues "];
```

Time-integrate the system of equations with **NDSolve**:

In[70]:=

```
tufun =NDSolveValue [{damping .u'[ t]+stiffness .u[ t]==load ,u[0]==init},u,{t,0,1},
Method →{"TimeIntegration "→"IDA"},AccuracyGoal →$MachinePrecision /4,PrecisionGoal →$MachinePr
```

Out[70]=

```
InterpolatingFunction [ Domain : {{0., 1.}} Output dimensions : {41, 1} ]
```

Set up a function that, given a time t, constructs and memorizes an interpolating function:

In[71]:=

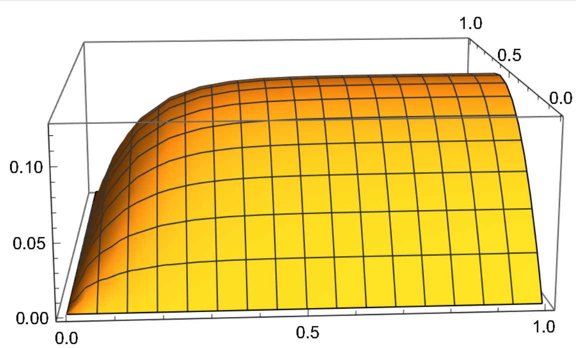
```
ClearAll [fun]
fun[t_?NumericQ ]:=fun[t]=ElementMeshInterpolation [{mesh}, tufun [t]]
```

Visualize the difference between the automatic and manual solutions:

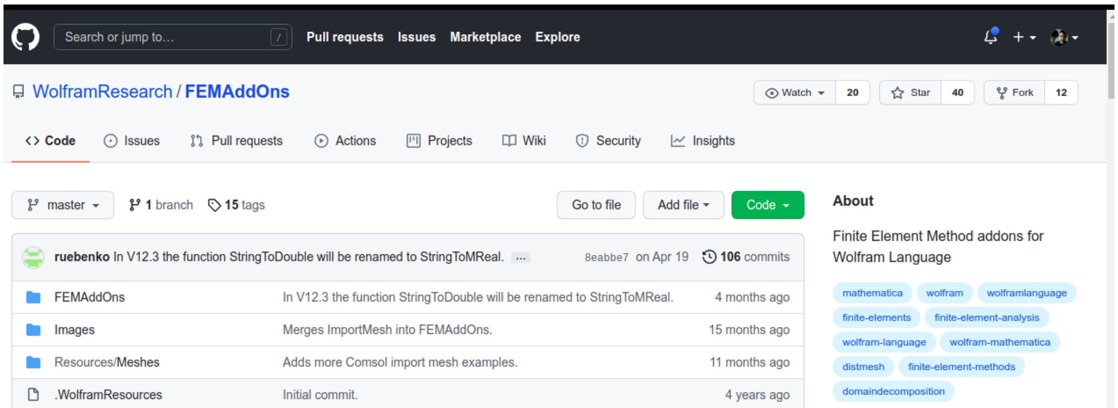
In[74]:=

```
Plot3D [fun[t][x],{t,0,1},{x,0,1}]
```

Out[74]=



Finite Element Method addons for Wolfram Language



The easiest way to install or update the FEMAddOns is to evaluate the following:

In[ ]:=
```
ResourceFunction ["FEMAddOnsInstall "][]
```

Out[ ]= PacletObject[Name : FEMAddOns Version : 1.4.5]

In[11]:=
```
PacletInstall ["/home/mk/Downloads /FEMAddOns -1.4.5.paclet "]
```

Out[11]= PacletObject[Name : FEMAddOns Version : 1.4.5]

In[10]:=
```
PacletFind ["FEMAddOns "]
```

Out[10]= {}

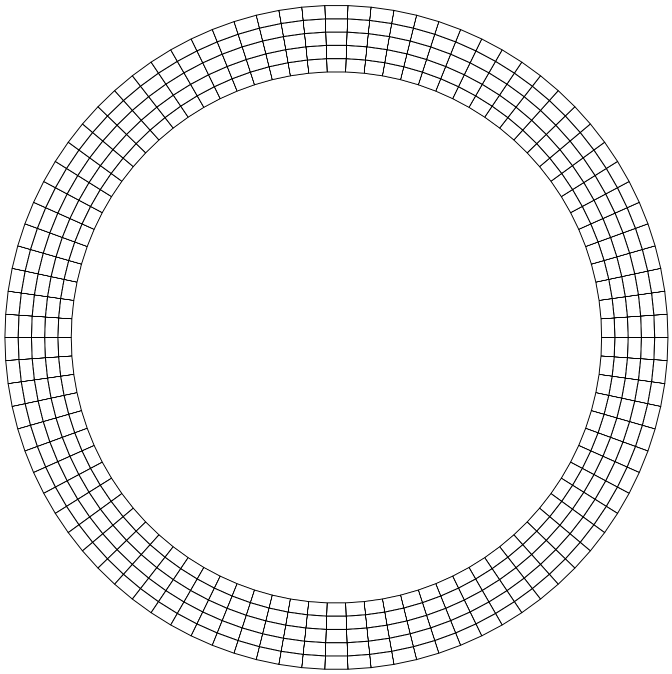
In[13]:=
```
PacletUninstall ["FEMAddOns "]
```

For example generate structured meshes with StructuredMesh:

In[14]:=
```
Needs["FEMAddOns` "]
```

In[15]:=
```
raster = Table[#, {fi, 0, 2 Pi, 2.0 Pi/360}] & /@ {{Cos[fi], Sin[fi]}, 0.8*{Cos[fi], Sin[fi]}};
mesh = StructuredMesh[raster, {90, 5}];
mesh["Wireframe"]
```

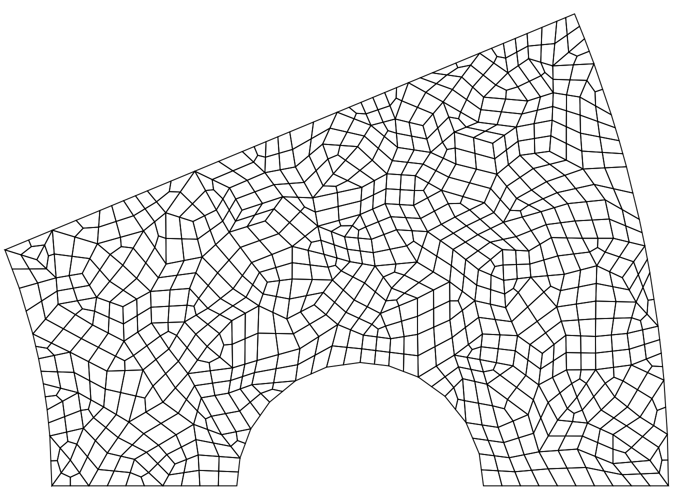
Out[17]=



With ToQuadMesh convert triangle meshes into quadrilateral meshes:

In[18]:=
```
region = ImplicitRegion[And @@ (# <= 0 & /@ {-y, 1/25 - (-3/2 + x)^2 - y^2,
    1 - x^2 - y^2, -4 + x^2 + y^2, y - x*Tan[Pi/8]}), {x, y}];
ToQuadMesh[ToElementMesh[region]]["Wireframe"]
```

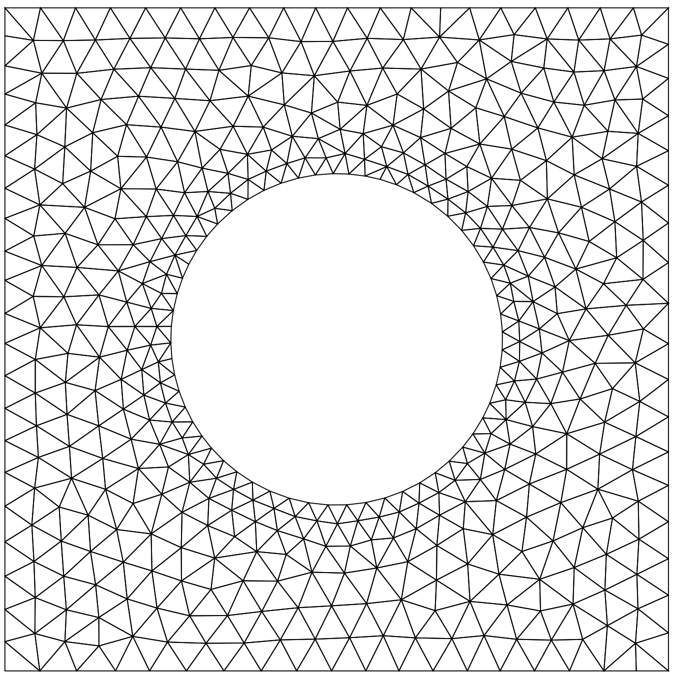
Out[19]=



Use the DistMesh mesh generator to create smooth meshes:

In[20]:=
```
mesh = DistMesh[RegionDifference [Rectangle [{-1, -1}, {1, 1}], Disk[{0, 0}, 1/2]],
   "DistMeshRefinementFunction " ->
    Function[{x, y}, Min[4*Sqrt[Plus @@ ({x, y}^2)] - 1, 2]],
   "MaxCellMeasure " -> {"Length" -> 0.05},
   "IncludePoints " -> {{-1, -1}, {-1, 1}, {1, -1}, {1, 1}}];
mesh["Wireframe "]
```

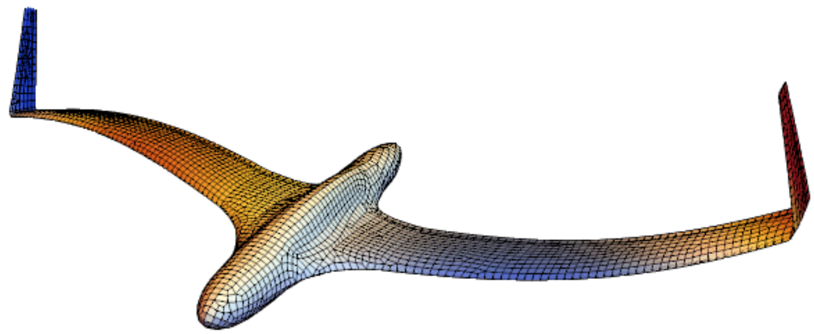
Out[21]=



With ImportMesh load meshes from Abaqus, Comsol, Elfen and Gmsh

In[22]:=
```
mesh = ImportMesh[ "filePath ", "mesh.mphtxt "];
mesh["Wireframe "]
```

Out[23]= ImportMesh[filePath , mesh.mphtxt][Wireframe]



Element Mesh Generation

Passing an ElementMesh to NDSolve

Set up a region:

In[75]:=
```
Ω=ImplicitRegion [True ,{{x,0,2},{y,0,1}}]
```

Out[75]=
```
ImplicitRegion [0 ≤ x ≤ 2 && 0 ≤ y ≤ 1, {x, y}]
```

Set up a PDE operator:

In[76]:=
```
op=-Laplacian [u[x,y],{x,y}]-20
```

Out[76]= $-20 - u^{(0,2)}[x, y] - u^{(2,0)}[x, y]$

Specify boundary conditions:

In[77]:=
```
Γ=DirichletCondition [u[x,y]==0, x==0||x==2]
```

Out[77]=
```
DirichletCondition [u[x, y] == 0, x == 0 || x == 2]
```

Solve the PDE:

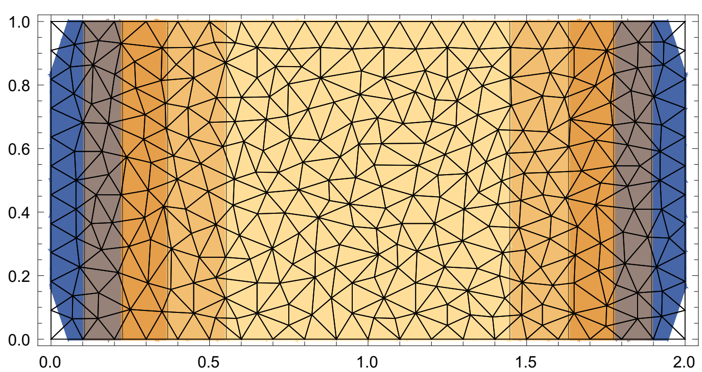
In[78]:=
```
ufun=NDSolveValue [{op==0,Γ},u,{x,y}∈Ω]
```

Out[78]=
```
InterpolatingFunction [ Domain : {{0., 2.}, {0., 1.}}  Output : scalar ]
```

Plot a contour plot of the solution with the element mesh from the interpolation function on top:

In[79]:=
```
Show[
ContourPlot [ufun[x,y],{x,y}∈Ω,AspectRatio →Automatic ],
ufun["ElementMesh "]["Wireframe "]]
```

Out[79]=



Extract the ElementMesh from an interpolating function:

```
ufun["ElementMesh "]
```

Instead of specifying an implicit parametric region, it is also possible to specify an explicit Elemen - tMesh. This can be done by using **ToElementMesh**:

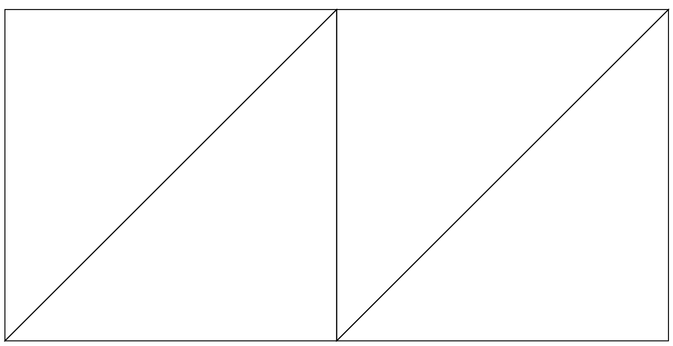
In[80]:=
```
mesh=ToElementMesh ["Coordinates "→{{0.,0.},{1.,0.},{2.,0.},{2.,1.},{1.,1.},{0.,1.}},
"MeshElements "→{TriangleElement [{{1,2,5},{5,6,1},{2,3,4},{4,5,2}}]}]
```

Out[80]=
```
ElementMesh [{{0., 2.}, {0., 1.}}, {TriangleElement [<4>]}]
```

Show a wireframe of the element mesh:

In[81]:=
```
mesh["Wireframe "]
```

Out[81]=



Next, the same PDE is solved, this time with only the explicit mesh defined:

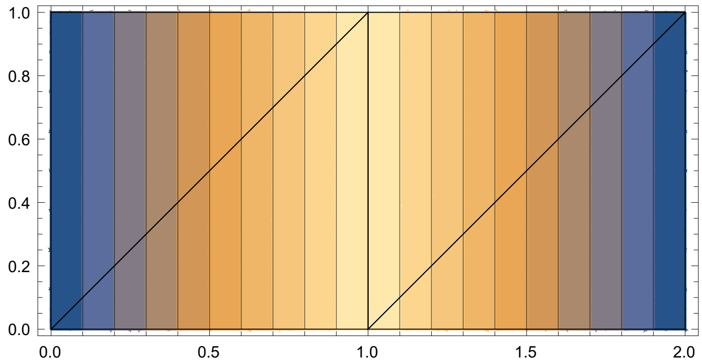
In[83]:=
```
ufun=NDSolveValue [{op==0,Γ},u[x,y],{x,y}∈mesh]
```

Out[83]=
```
InterpolatingFunction [ Domain : {{0., 2.}, {0., 1.}}  Output : scalar ][x, y]
```

This makes a contour plot of the solution and plots the element mesh:

In[∘]:=
```
Show[
ContourPlot[ufun,{x,y}∈mesh,AspectRatio→Automatic],
mesh["Wireframe"]]
```

Out[∘]=



Element Mesh Generation

Passing Options for the ElementMesh Creation to NDSolve via MeshOptions

Solve the PDE with options given to influence the element mesh generation:

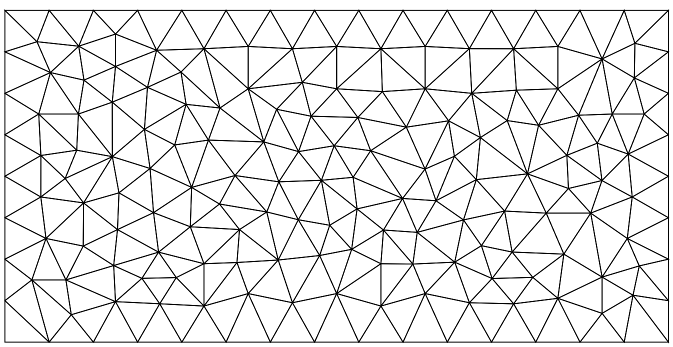
In[84]:=
```
ufun=NDSolveValue [{op==0,Γ},u,{x,y}∈Ω,
Method→{"FiniteElement ","MeshOptions "→{MaxCellMeasure →0.01}}]
```

Out[84]= InterpolatingFunction [Domain : {{0., 2.}, {0., 1.}} Output : scalar]

As an alternative, the element mesh can be generated prior to the simulation and given to NDSolve:

In[89]:=
```
mesh=ToElementMesh [Ω,MaxCellMeasure →0.01];
mesh["Wireframe "]
```

Out[90]=



In[91]:=
```
ufun=NDSolveValue [{op==0,Γ},u,{x,y}∈mesh]
```

Out[91]= InterpolatingFunction [Domain : {{0., 2.}, {0., 1.}} Output : scalar]

Solve the time-dependent PDE with options given to influence the element mesh generation:

```
ufun=NDSolveValue [{D[u[t,x,y],t]-Laplacian [u[t,x,y],{x,y}]-20==0,DirichletCondition [u[t,x,y]==0, x
Method →{"PDEDiscretization "→{Automatic ,
"SpatialDiscretization "→{"FiniteElement ","MeshOptions "→{MaxCellMeasure →0.01}}}}]
```

Out[∘]= InterpolatingFunction [Domain : {{0., 1.}, {0., 2.}, {0., 1.}} Output : scalar]

Data not in notebook ; Store now »

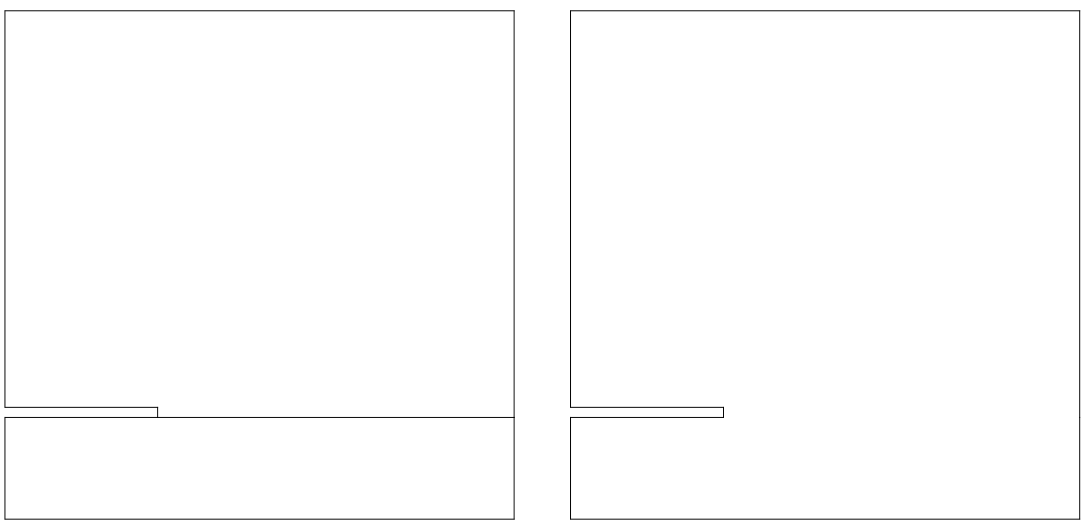
Element Mesh Generation

Element Meshes with Subregions

It is common for a PDE to interact with a region that is made up of multiple materials. The solutions of PDEs will be of a higher quality if the mesh elements do not cross the internal boundaries. To illustrate this, a PDE with a variable diffusion coefficient is reconsidered and solved over two regions (see Solving Partial Differential Equations with Finite Elements). One region respects the internal boundary, while the other does not:

In[92]:=
```
sh=0.2;sh2=0.02;sw=0.3;
coordinates ={{0.,0.},{1.,0.},{1.,sh},{1.,1.},{0.,1.}, {0.,sh+sh2}, {sw,sh+sh2},{sw,sh},{0.,sh}};
el1=LineElement [{{1,2},{2,3},{3,4},{4,5},{5,6},{6,7},{7,8},{8,9},{9,1}}];
bMesh1 =ToBoundaryMesh ["Coordinates "→coordinates , "BoundaryElements "→{el1,LineElement [{{3,8}}
bMesh2 =ToBoundaryMesh ["Coordinates "→coordinates , "BoundaryElements "→{el1}];
GraphicsRow [{bMesh1 ["Wireframe "],bMesh2 ["Wireframe "]}]
```

Out[97]=



The diffusion coefficient has a jump discontinuity at $y = 0.2$. Set up a diffusion coefficient that is space dependent:

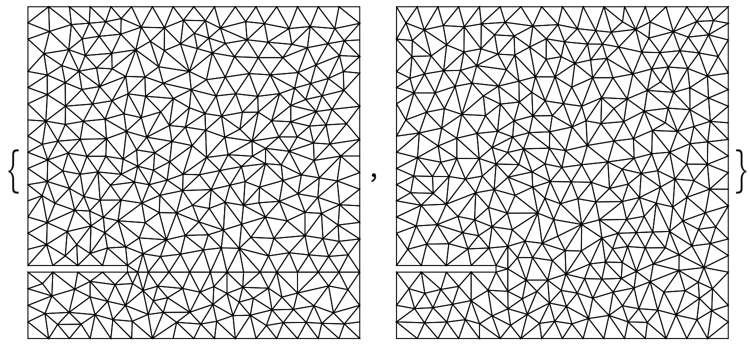
In[98]:=
```
ϵr=If[y≤sh,{{11.7,0.},{0.,11.7}},{{1.,0},{0.,1.}}]
```

Out[98]=
```
If[y ≤ 0.2, {{11.7, 0.}, {0., 11.7}}, {{1., 0}, {0., 1.}}]
```

Create and visualize the element meshes:

In[99]:=
```
mesh1 =ToElementMesh [bMesh1 ];
mesh2 =ToElementMesh [bMesh2 ];
{mesh1 ["Wireframe "],mesh2 ["Wireframe "]}
```

Out[101]=



Specify a PDE operator and boundary conditions:

In[102]:=
```
op=Inactive [Div][-ϵr.Inactive [Grad][u[x,y],{x,y}],{x,y}]-10^-8./8.86*^-12 ;
ΓD={DirichletCondition [u[x,y]==0,x==1||y==1||y==0],
 DirichletCondition [u[x,y]==10^3,0<x≤sw&&sh≤y≤sh+sh2]};
```

Solve the equation over each mesh:

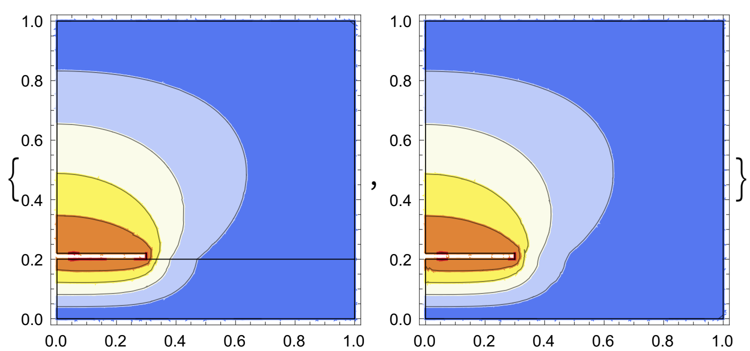
In[104]:=
```
ufun1 =NDSolveValue [{op==0,ΓD},u,{x,y}∈mesh1 ];
```

In[105]:=
```
ufun2 =NDSolveValue [{op==0,ΓD},u,{x,y}∈mesh2 ];
```

Visualize the solution:

In[106]:=
```
{
Show[ContourPlot [ufun1 [x,y],{x,y}∈mesh1 ,ColorFunction →"TemperatureMap ",AspectRatio →Automatic
bMesh1 ["Wireframe "]],
Show[ContourPlot [ufun2 [x,y],{x,y}∈mesh2 ,ColorFunction →"TemperatureMap ",AspectRatio →Automatic
bMesh2 ["Wireframe "]]
}
```

Out[106]=

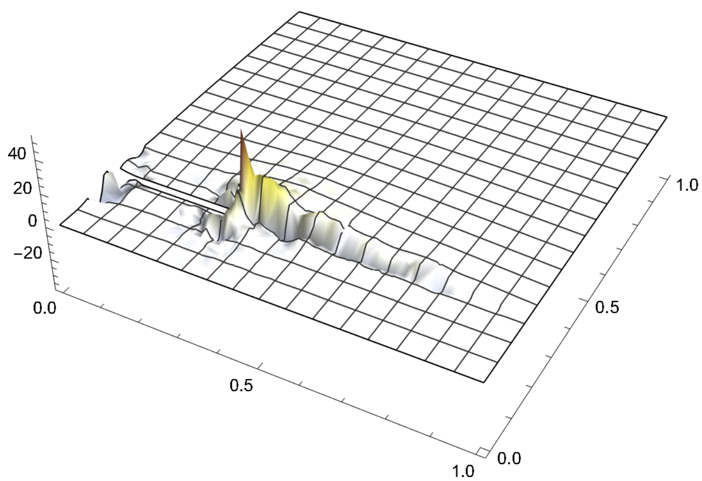


Visualize the difference between the two solutions:

In[107]:=

```
Plot3D[ufun1[x,y]-ufun2[x,y],{x,y}∈mesh1,ColorFunction →"TemperatureMap",PlotRange →All,Boxed→
```

Out[107]=



The next example shows a circular region with a subregion and holes inside the subregion:

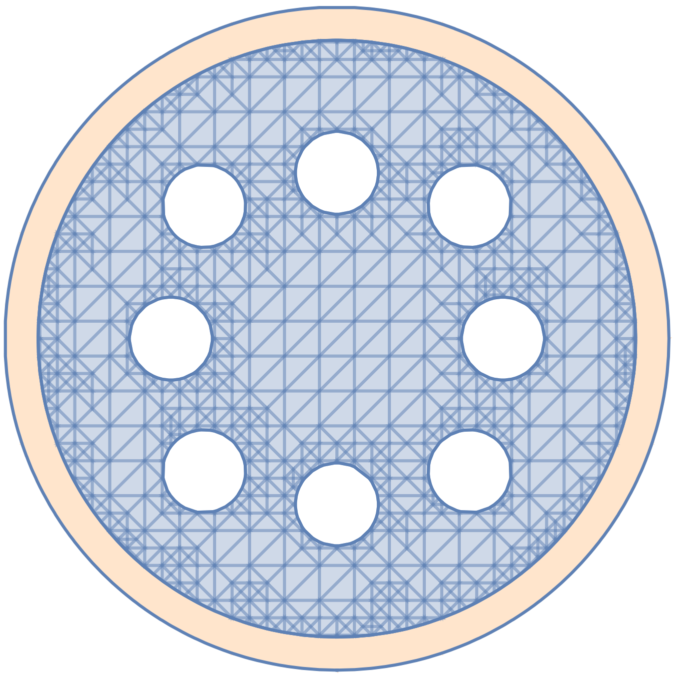
In[108]:=

```
annulus[x_,y_]:=(9/10)^2≤x^2+y^2≤1^2
holes[{x0_,y0_},r_]:=((x+x0)^2+(y+y0)^2≤(r)^2)
crds={{-1/2,0},{1/2,0},{0,-1/2},{0,1/2},{2/5,2/5},{-2/5,-2/5},{2/5,-2/5},{-2/5,2/5}};
sd=Or@@(holes[#,1/8]&/@crds);
```

Display the region:

In[112]:=
```
Show[
RegionPlot[annulus[x,y],{x,-1,1},{y,-1,1},PlotStyle→LightOrange],
RegionPlot[x^2+y^2<(9/10)^2&&!sd,{x,-1,1},{y,-1,1}]
,Frame→False]
```

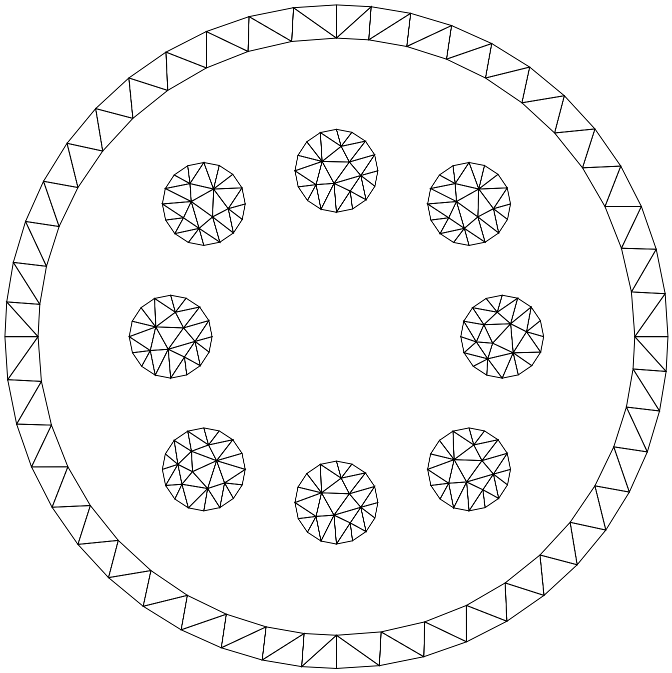
Out[112]=



Specify the region as an implicit region and create an element mesh:

In[115]:=
```
Ω2=ImplicitRegion [Or[annulus [x,y],sd],{x,y}];
ToElementMesh [Ω2]["Wireframe "]
```

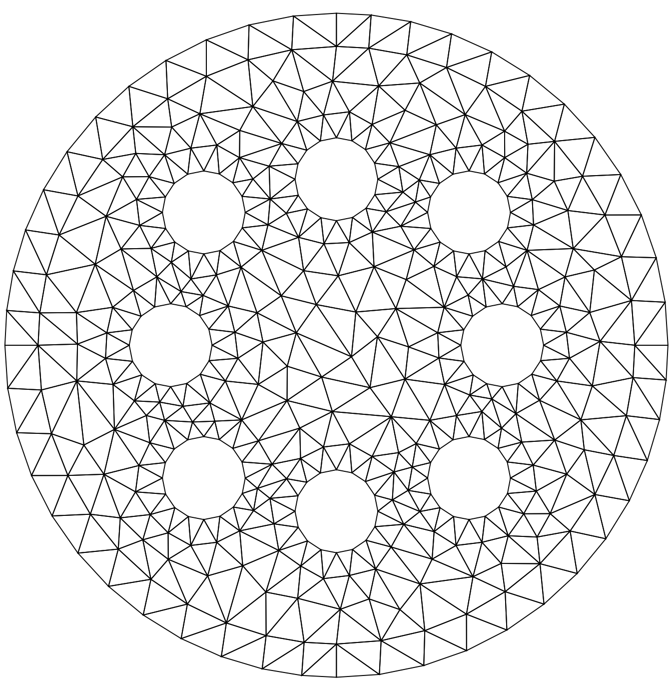
Out[116]=



In the next step, what is a region hole and what is not is inverted in the subregion by explicitly specify - ing the region holes:

In[117]:=
```
ToElementMesh [Ω2,"RegionHoles "→crds]["Wireframe "]
```

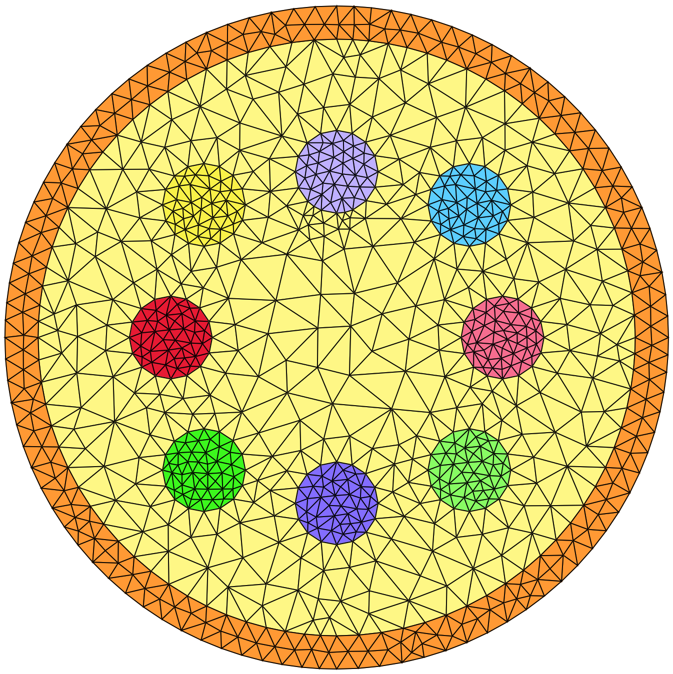
Out[117]=



In[118]:=

```
mesh=ToElementMesh [Ω2 ,"RegionHoles "→None ,"RegionMarker "→Join[
 MapThread [{#1 ,#2 ,0.001 }&,{crds ,Range [Length [crds ]]}],{{{0 ,0},Length [crds ]+1 ,0.01 },{{19/20 ,0},Lengt
temp =Most [Range [0 ,1 ,1/(Length [crds ]+2)]];
colors =ColorData ["BrightBands "][#]&/@ temp ;
mesh["Wireframe "["MeshElementStyle  "→FaceForm /@ colors ]]
```

Out[121]=



Element Mesh Visualization

An ElementMesh is typically created with either **ToBoundaryMesh** or **ToElementMesh**:

In[122]:=
```
Needs["NDSolve`FEM`"]
```

Create a triangle element mesh with six triangle elements and markers:

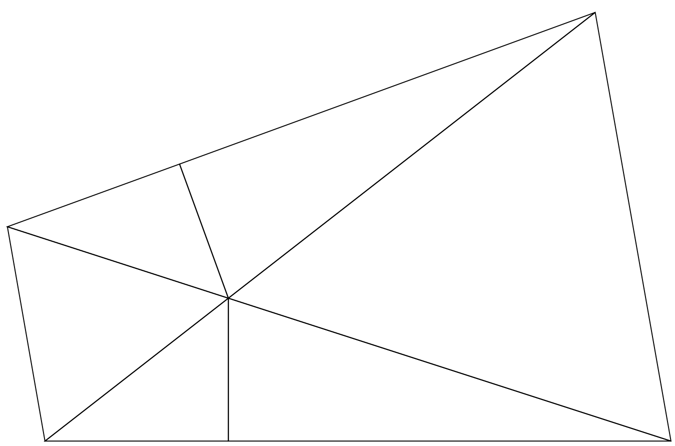
In[123]:=
```
mesh=ToElementMesh["Coordinates"->{{1.293,0.228},{1.,0.},{0.94,0.342},{1.293,0.},{1.215,0.442},
"MeshElements"→ {TriangleElement[{{1,3,2},{1,2,4},{1,4,6},{1,6,7},{1,7,5},{1,5,3}},{66,66,66,44,
```

Out[123]= ElementMesh[{{0.94, 2.}, {0., 0.684}}, {TriangleElement[<6>]}]

Visualize the element mesh wireframe:

In[124]:=
```
mesh["Wireframe"]
```

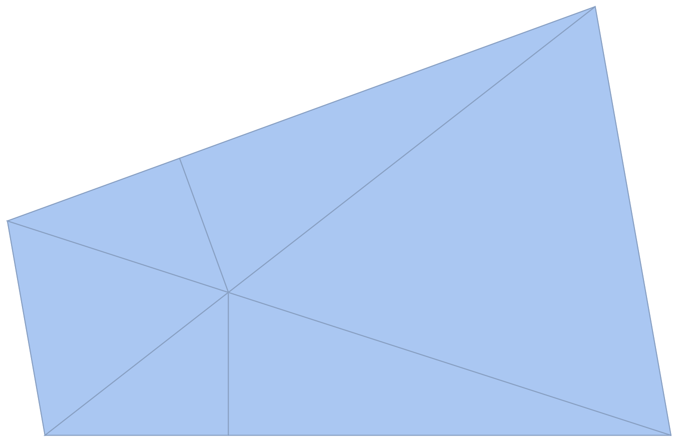
Out[124]=



Convert to a **MeshRegion**:

In[125]:=
```
MeshRegion[mesh]
```

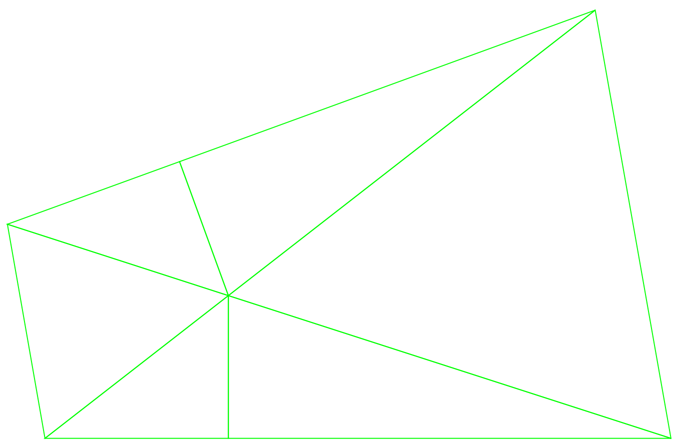
Out[125]=



Visualize the element mesh wireframe in green:

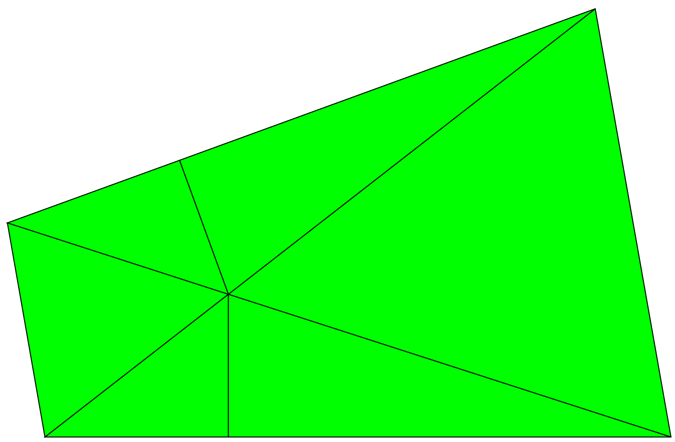
In[126]:=
```
mesh["Wireframe"["MeshElementStyle"→EdgeForm[Green]]]
```

Out[126]=



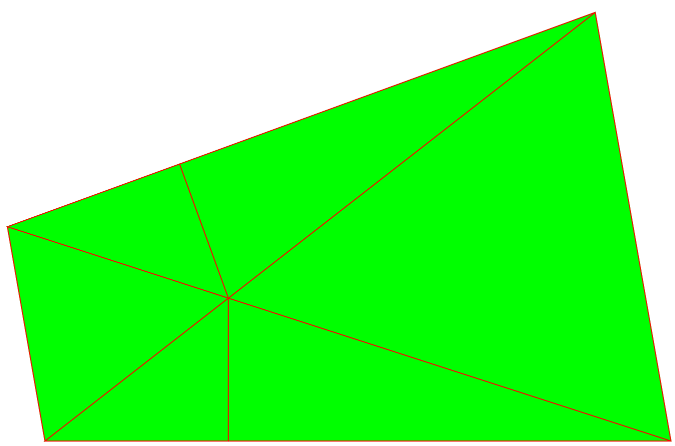
Visualize the element mesh in green:

In[ ]:=
```
mesh["Wireframe"["MeshElementStyle"→FaceForm[Green]]]
```

Out[ ]=


Visualize the element mesh in green and the faces in red:

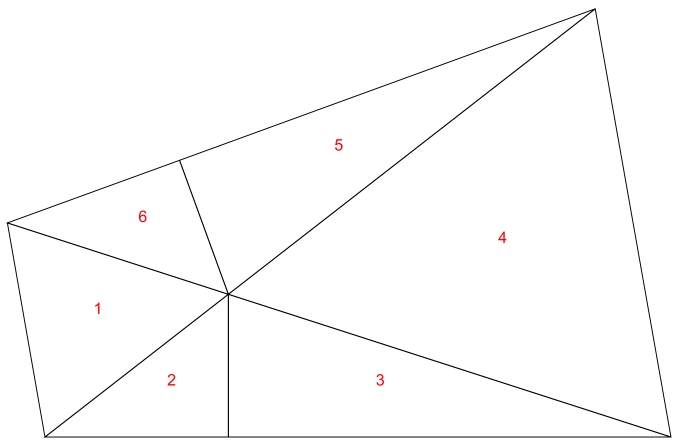
In[ ]:=
```
mesh["Wireframe"["MeshElementStyle"→Directive[FaceForm[Green],EdgeForm[Red]]]]
```

Out[ ]=


Visualize the element mesh wireframe with the element identification numbers in red:

In[128]:=
```
mesh["Wireframe "["MeshElementIDStyle  "→Red]]
```

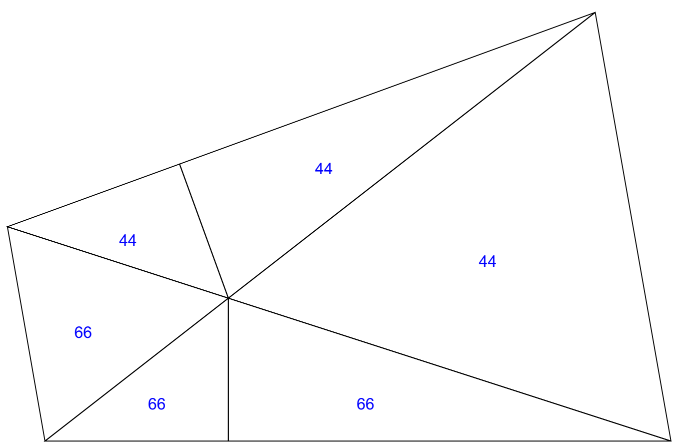
Out[128]=



Visualize the element mesh wireframe with the element markers in blue:

In[∘]:=
```
mesh["Wireframe "["MeshElementMarkerStyle  "→Blue]]
```

Out[∘]=



Find the positions of elements that have a quality less than 0.9:

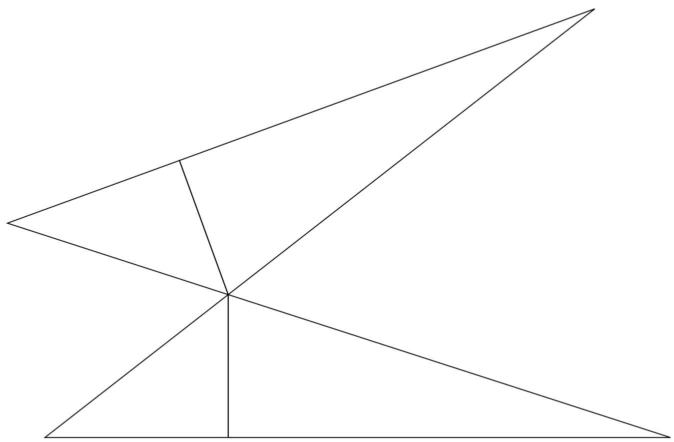
In[129]:=
```
pos=Position[mesh["Quality "],_?(#≤0.9&)]
```

Out[129]= {{1, 2}, {1, 3}, {1, 5}, {1, 6}}

Visualize just those mesh elements as a wireframe:

In[130]:=
```
mesh["Wireframe"[pos]]
```

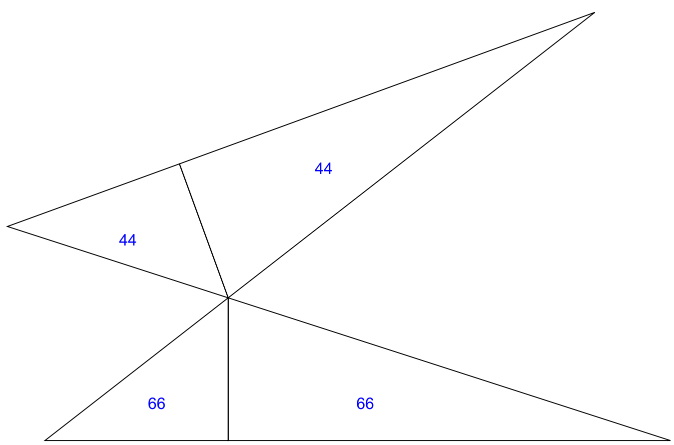
Out[130]=



Additionally, highlight the element markers in blue:

In[131]:=
```
mesh["Wireframe"[pos,"MeshElementMarkerStyle"→Blue]]
```

Out[131]=



Find the positions of elements that have a certain marker:

In[ ]:=
```
pos=Position[ElementMarkers[mesh["MeshElements"]],44]
```

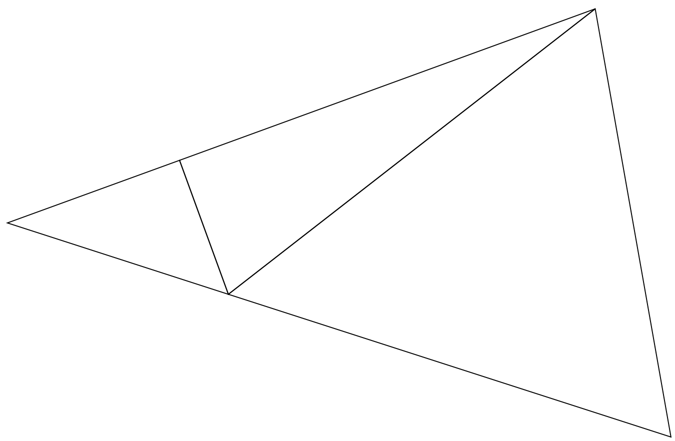
Out[ ]= {{1, 4}, {1, 5}, {1, 6}}

Visualize just those mesh elements as a wireframe:

```
mesh["Wireframe"[pos]]
```

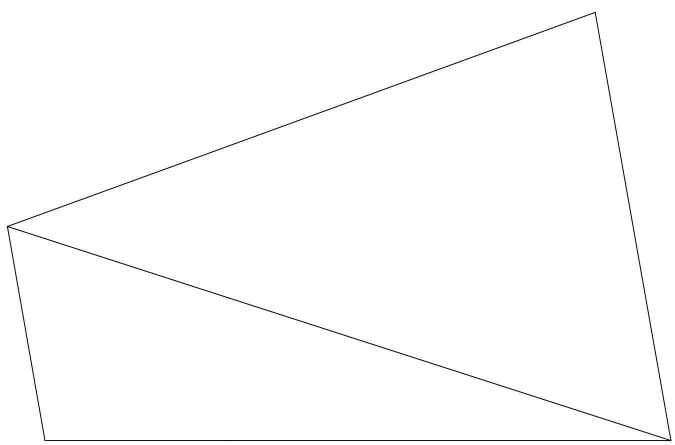
Directly visualize mesh elements that contain markers as a wireframe:

In[ ]:=
```
mesh["Wireframe "[ElementMarker ==44]]
```

Out[ ]=


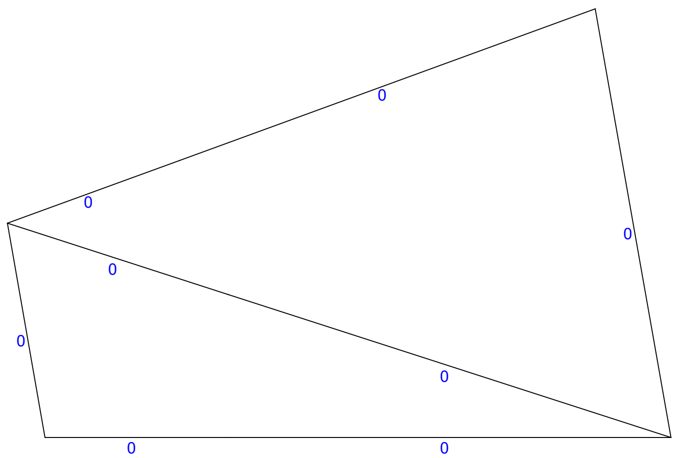
Visualize the boundary element mesh wireframe:

In[ ]:=
```
mesh["Wireframe "["MeshElement "→"BoundaryElements "]]
```

Out[ ]=


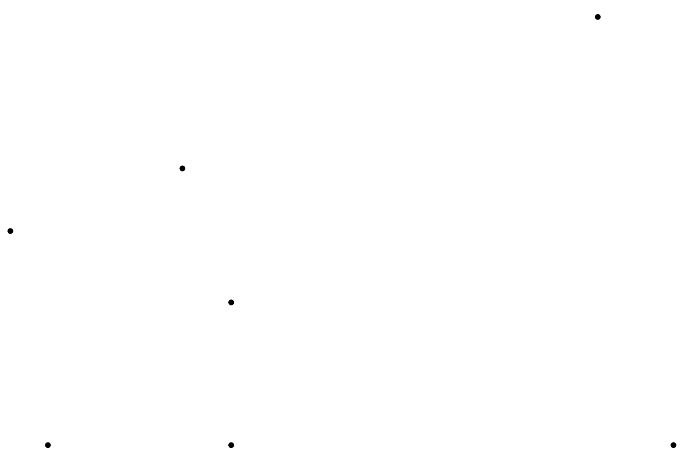
Visualize the boundary element mesh wireframe with the element markers in blue:

In[ ]:=
```
mesh["Wireframe"["MeshElement"→"BoundaryElements","MeshElementMarkerStyle"→Blue]]
```

Out[ ]=


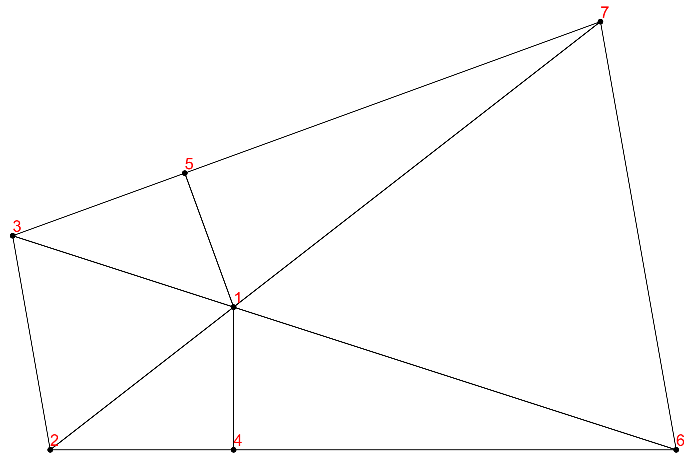
Visualize the point element mesh wireframe:

In[ ]:=
```
mesh["Wireframe"["MeshElement"→"PointElements"]]
```

Out[ ]=


Visualize a combination of different aspects of an element mesh:

```
In[ ]:= Show[mesh["Wireframe "],mesh["Wireframe "["MeshElement "→"PointElements ","MeshElementIDStyle "→
```

Out[ ]=


Inspect the boundary and point elements:

```
In[ ]:= mesh["BoundaryElements "]
```

```
Out[ ]= {LineElement [{{3, 2}, {1, 3}, {2, 4}, {4, 6}, {6, 1}, {6, 7}, {7, 5}, {5, 3}}]}
```

```
In[ ]:= mesh["PointElements "]
```

```
Out[ ]= {PointElement [{{1}, {2}, {3}, {4}, {5}, {6}, {7}}]}
```

Converting Images to Meshes

In[133]:=
```
ekb = Egyptian Knowledge Bank
      بنك المعرفة المصري ;
```

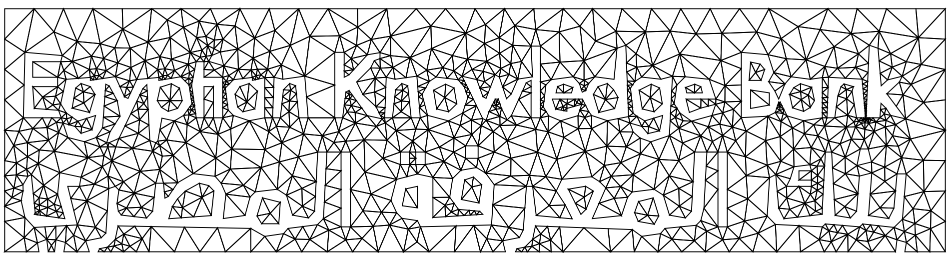
In[ ]:=
```
Needs["NDSolve`FEM` "]
```

In[134]:=
```
mesh = ToElementMesh [ImageMesh [ekb]]
```

Out[134]=
```
ElementMesh [{{0. , 302.}, {0. , 78.}}, {TriangleElement [<1859>]}]
```

In[138]:=
```
mesh["Wireframe "]
```

Out[138]=



In[135]:=
```
op=-Laplacian [u[x,y],{x,y}]-20
```

Out[135]= $-20 - u^{(0,2)}[x, y] - u^{(2,0)}[x, y]$

In[136]:=
```
Γ=DirichletCondition  [u[x,y]==0,  True]
```

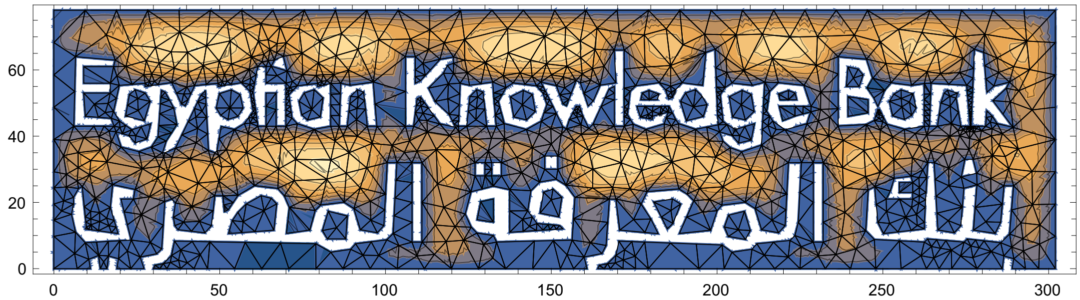
Out[136]=
```
DirichletCondition [u[x, y] == 0, True]
```

In[139]:=
```
ufun=NDSolveValue [{op==0,Γ},u,{x,y}∈mesh]
```

Out[139]=
```
InterpolatingFunction [ Domain : {{0., 302.}, {0., 78.}}
                        Output : scalar ]
```

In[140]:=
```
Show[
ContourPlot [ufun[x,y],{x,y}∊mesh ,AspectRatio →Automatic ],
ufun["ElementMesh "]["Wireframe "]]
```

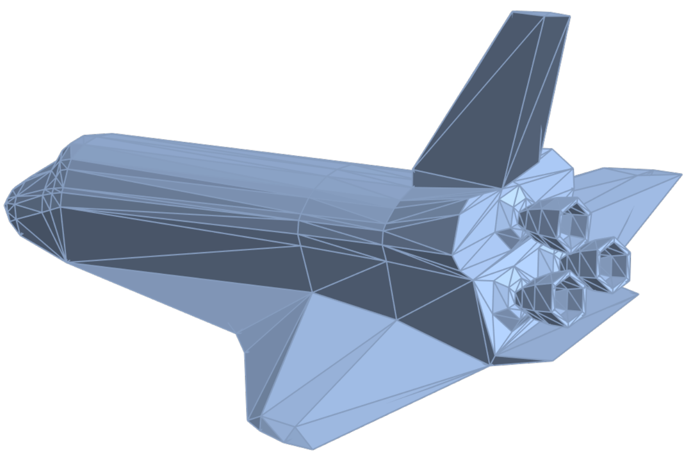
Out[140]=



Three Dimensional Meshes

In[141]:=
```
mr = BoundaryDiscretizeGraphics [ExampleData [{"Geometry3D ", "SpaceShuttle "}]]
```

Out[141]=



In[142]:=
```
uif = NDSolveValue [{Inactive [Laplacian ][u[x, y, z], {x, y, z}] == 0,
    DirichletCondition [u[x, y, z] == 1, z ≤ -1.3],
    DirichletCondition [u[x, y, z] == 0, x ≤ -7.]}, u, {x, y, z} ∈ mr];
```

In[143]:=
```
Needs["NDSolve`FEM`"]
ElementMeshSurfacePlot3D [uif, Boxed → False, ViewPoint → {0, -4, 2}]
```

Out[144]=