

Grid creation and visualization in PDELib.jl

This notebook shows how to perform grid creation and visualization with the assistance of the packages [ExtendableGrids.jl](#) and [SimplexGridFactory.jl](#) which are part of the [PDELib.jl](#) meta package.

Visualization in this notebook is done using the [GridVisualize.jl](#) package.

PlutoVista

```

• begin
•   using SimplexGridFactory
•   using ExtendableGrids
•   using Triangulate
•   using TetGen
•   using GridVisualize
•     using PlutoVista
•   using PlutoUI
•     isdefined(Main, :PlutoRunner) ? default_plotter!(PlutoVista) :
• default_plotter!(nothing)
• end

```

1D grids

1D grids are created just from arrays of monotonically increasing coordinates using the [simplexgrid](#) method.

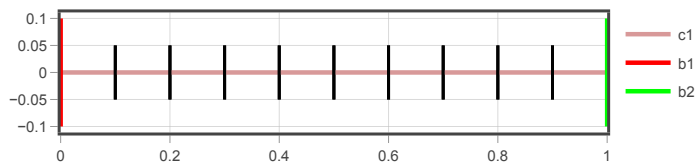
`X1 = 0.0:0.1:1.0`

```
• X1=range(0,1;length=11)
```

```
ExtendableGrids.ExtendableGrid{Float64, Int32};
dim: 1 nodes: 11 cells: 10 bfaces: 2
```

```
• g1=simplexgrid(X1); @show g1
```

We can plot a grid with a method from [GridVisualize.jl](#)



```
• gridplot(g1; resolution=(500,150), legend=:rt)
```

We see some additional information:

- `cellregion`: each grid cell (interval, triangle, tetrahedron) as an integer region marker attached
- `bfaceregion`: boundary faces (points, lines, triangles) have an interger boundary region marker attached

We can also have a look into the grid structure:

```
Dict{Coordinates => 1x11 Matrix{Float64}:
  0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0}, CellGeometrie
```

```
• g1.components
```

Components can be accessed via `[]`. In fact the keys in the dictionary of components are **types**.

```
1x11 Matrix{Float64}:
 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
• g1[Coordinates]
```

```
2x10 Matrix{Int32}:
 1 2 3 4 5 6 7 8 9 10
 2 3 4 5 6 7 8 9 10 11
```

```
• g1[CellNodes]
```

Modifying region markers

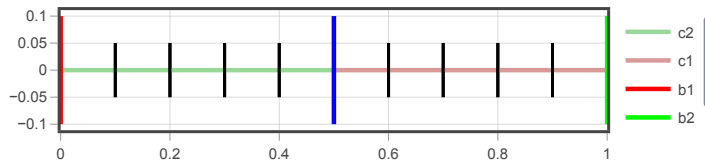
The `simplexgrid` method provides a default distribution of markers, but we would like to be able to change them. This can be done by putting masks on cells or faces (points in 1D):

```
g2 = ExtendableGrids.ExtendableGrid{Float64, Int32};
    dim: 1 nodes: 11 cells: 10 bfaces: 2
```

```
• g2=deepcopy(g1)
```

```
• cellmask!(g2, [0.0], [0.5], 2);
```

```
• bfacemask!(g2, [0.5],[0.5], 3);
```



```
• gridplot(g2; resolution=(500, 150), legend=:rt)
```

Creating locally refined grids

For this purpose, we just need to create arrays with the corresponding coordinate values. This can be done programmatically.

Two support methods are provided for this purpose.

```
0.1
```

```
• hmin=0.01 ; hmax=0.1
```

The `geomspace` method creates an array such that the smallest interval size is `hmin` and the largest interval size is not larger but close to `hmax`, and the interval sizes constitute a geometric sequence.

```
X2L =
```

```
[0.0, 0.0931551, 0.170501, 0.234722, 0.288044, 0.332316, 0.369076, 0.399597, 0.424939, 0.4
```

```
• X2L=geomspace(0,0.5,hmax,hmin)
```

```
DX2 =
```

```
[0.0931551, 0.0773463, 0.0642203, 0.0533218, 0.0442729, 0.0367596, 0.0305213, 0.0253417, 0
```

```
• DX2=X2L[2:end]-X2L[1:end-1]
```

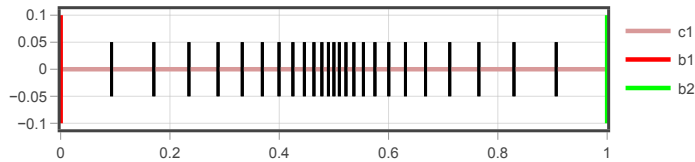
```
[1.20439, 1.20439, 1.20439, 1.20439, 1.20439, 1.20439, 1.20439, 1.20439, 1.20439, 1.20439,
```

```
• DX2[1:end-1]./DX2[2:end]
```

```
X2R =
 [0.5, 0.51, 0.522044, 0.536549, 0.55402, 0.575061, 0.600403, 0.630924, 0.667684, 0.711956,
 • X2R=geospace(0.5,1,hmin,hmax)
```

We can glue these arrays together and create a grid from them:

```
X2 =
 [0.0, 0.0931551, 0.170501, 0.234722, 0.288044, 0.332316, 0.369076, 0.399597, 0.424939, 0.4
 • X2=glue(X2L,X2R)
```



```
• gridplot(simplexgrid(X2); resolution=(500,150),legend=:rt)
```

Plotting functions

We assume that functions can be represented by their node values and plotted via their piecewise linear interpolants. E.g. they could come from some simulation.

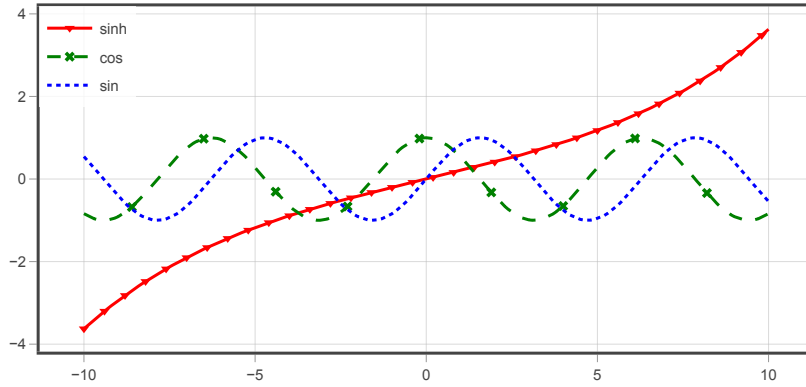
```
g1d2 = ExtendableGrids.ExtendableGrid{Float64, Int32};
 dim: 1 nodes: 201 cells: 200 bfaces: 2
```

```
• g1d2=simplexgrid(range(-10,10,length=201))
```

```
fsin =
 [0.544021, 0.457536, 0.366479, 0.271761, 0.174327, 0.0751511, -0.0247754, -0.124454, -0.22
 • fsin=map(sin,g1d2)
```

```
fcos =
 [-0.839072, -0.889191, -0.930426, -0.962365, -0.984688, -0.997172, -0.999693, -0.992225, -
 • fcos=map(cos,g1d2)
```

```
fsinh =
 [-3.62686, -3.55234, -3.47923, -3.40752, -3.33718, -3.26816, -3.20046, -3.13403, -3.06886,
 • fsinh=map(x->sinh(0.2*x), g1d2)
```



```

• let
•   vis=GridVisualizer(;resolution=(600,300),legend=:lt)
•
•   scalarplot!(vis, g1d2, fsinh, label="sinh", markershape=:dtriangle,
•               color=:red,markevery=5,clear=false)
•
•   scalarplot!(vis, g1d2, fcos, label="cos", markershape=:xcross, color=:green,
•               linestyle=:dash, clear=false,markevery=20)
•
•   scalarplot!(vis, g1d2, fsin, label="sin", markershape=:none, color=:blue,
•               linestyle=:dot, clear=false, markevery=20)
•
•   reveal(vis)
• end

```

2D grids

Tensor product grids

For 2D tensor product grids, we can again use the `simplexgrid` method and apply the mask methods for modifying cell and boundary region markers.

```

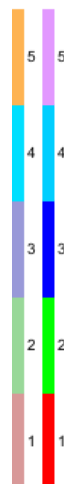
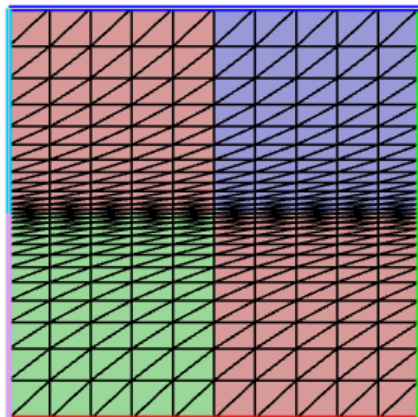
ExtendableGrids.ExtendableGrid{Float64, Int32};
dim: 2 nodes: 297 cells: 520 bfaces: 72

```

```

• begin
•   g2d1=simplexgrid(X1,X2)
•   cellmask!(g2d1, [0.0,0.0], [0.5, 0.5], 2)
•   cellmask!(g2d1, [0.5,0.5], [1.0, 1.0], 3)
•   bfacemask!(g2d1, [0.0, 0.0], [0.0, 0.5],5)
• end

```



```

• gridplot(g2d1,resolution=(600,400),linewidth=0.5,legend=:lt)

```

To interact with the plot, you can use the mouse wheel or double touch to zoom, "shift-mouse-left" to pan, and "alt-mouse-left" or "ctrl-mouse-left" to reset.

We can also have a look into the components of a 2D grid:

```
Dict{FaceCells => 2x816 Matrix{Int32}:
  1  1  1  2  2  3  3  3  4  5  5  5  6  ...  515  516  517  517  518
  0  4  2  21  0  0  6  4  23  0  8  6  25  ...  516  0  520  518  0
```

```
◀ | ▶
• g2d1.components
```

Unstructured grids

For the triangulation of unstructured grids, we use the mesh generator Triangle via the [Triangulate.jl](#) and [SimplexGridFactory.jl](#) packages.

The later package exports the `SimplexGridBuilder` which shall help to simplify the creation of the input for `Triangulate`.

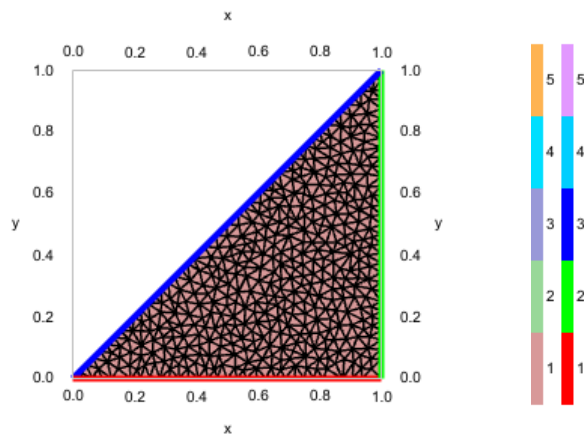
```
builder2 =
  SimplexGridBuilder(Triangulate, 3, 1, 1.0, 1.0e-12, [1, 2, 3], [[1, 2], [2, 3], [3, 1]], Bi
```

```
◀ | ▶
• builder2=let
•   b=SimplexGridBuilder(Generator=Triangulate)
•   p1=point!(b,0,0)
•   p2=point!(b,1,0)
•   p3=point!(b,1,1)
•
•   # Specify outer boundary
•   facetregion!(b,1)
•   facet!(b,p1,p2)
•   facetregion!(b,2)
•   facet!(b,p2,p3)
•   facetregion!(b,3)
•   facet!(b,p3,p1)
•
•   cellregion!(b,1)
•   regionpoint!(b,0.75,0.25)
•
•   options!(b,maxvolume=0.01)
•   b
• end
```

We can plot the current state of the builder:

```
grid2d2 = ExtendableGrids.ExtendableGrid{Float64, Int32};
          dim: 2 nodes: 449 cells: 793 bfaces: 103
```

```
• grid2d2=simplexgrid(builder2;maxvolume=0.001)
```



```
• gridplot(grid2d2, resolution=(400,300),linewidth=0.5)
```

More complicated grids

More complicated grids include:

- local refinement
- interior boundaries
- different region markers
- holes

The particular way to describe these things is due to Jonathan Shewchuk and his mesh generator [Triangle](#) via its Julia wrapper package [Triangulate.jl](#).

Local refinement

```
refinement_center = [0.8, 0.2]
• refinement_center=[0.8,0.2]
```

For local refinement, we define a function, which is able to tell if a triangle is to be refined ("unsuitable") or can be kept as it is.

The function measures the distance between the refinement center and the triangle barycenter. We require that the area increases with the distance from the refinement center.

```
• function unsuitable(x1,y1,x2,y2,x3,y3,area)
•     bary_x=(x1+x2+x3)/3.0
•     bary_y=(y1+y2+y3)/3.0
•     dx=bary_x-refinement_center[1]
•     dy=bary_y-refinement_center[2]
•     qdist=dx^2+dy^2
•     area>0.1*max(1.0e-2,qdist)
• end;
```

Interior boundaries

Interior boundaries are described in a similar as exterior ones - just by facets connecting points.

Subregions

Subregions are defined as regions surrounded by interior boundaries. By placing a "region point" into such a region and specifying a "region number", we can set the cell region marker for all triangles created in the subregion.

Holes

Holes are defined in a similar way as subregions, but a "hole point" is placed into the place which shall become the hole.

```

• builder3=let
•   b=SimplexGridBuilder(Generator=Triangulate;tol=1.0e-10)
•
•   # Specify points
•   p1=point!(b,0,0)
•   p2=point!(b,1,0)
•   p3=point!(b,1,1)
•   p4=point!(b,0,0.7)
•
•   # Specify outer boundary
•   facetregion!(b,1)
•   facet!(b,p1,p2)
•   facetregion!(b,2)
•   facet!(b,p2,p3)
•   facetregion!(b,3)
•   facet!(b,p3,p4)
•   facetregion!(b,4)
•   facet!(b,p1,p4)
•
•   # Activate unsuitable callback
•   options!(b,unsuitable=unsuitable)
•
•   # Specify interior boundary
•   facetregion!(b,5)
•   facet!(b,p1,p3)
•
•   # Coarse elements in upper left region #1
•   cellregion!(b,1)
•   maxvolume!(b,0.1)
•   regionpoint!(b,0.1,0.5)
•
•   # Fine elements in lower right region #2
•   cellregion!(b,2)
•   maxvolume!(b,0.01)
•   regionpoint!(b,0.9,0.5)
•
•   # Hole
•   hp1=point!(b,0.4,0.1)
•   hp2=point!(b,0.6,0.1)
•   hp3=point!(b,0.5,0.3)
•   holepoint!(b,0.5,0.2)
•   facetregion!(b,6)
•   facet!(b,hp1,hp2)
•   facet!(b,hp2,hp3)
•   facet!(b,hp3,hp1)
•
•   b
• end;

```

Create a simplex grid from the builder

```

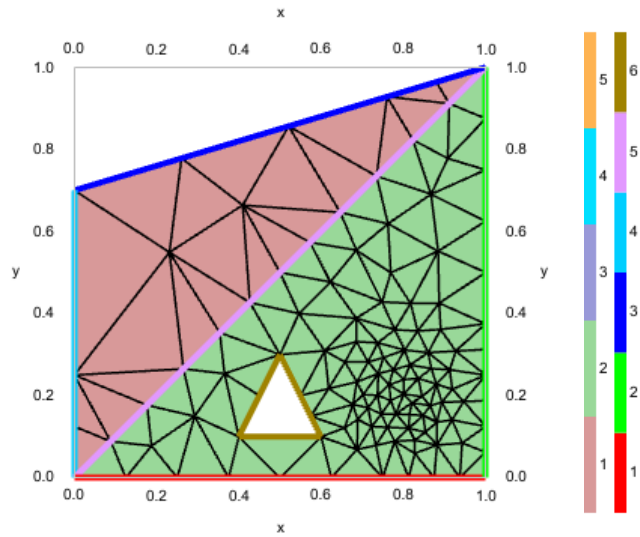
grid2d3 = ExtendableGrids.ExtendableGrid{Float64, Int32};
          dim: 2 nodes: 117 cells: 199 bfaces: 47

```

```

• grid2d3=simplexgrid(builder3)

```



```
• gridplot(grid2d3, legend=:lt, resolution=(400,400))
```

Plotting of functions

Functions defined on the nodes of a triangular grid can be seen as piecewise linear functions from the P_1 finite element space defined by the triangulation.

`fsin2 =`

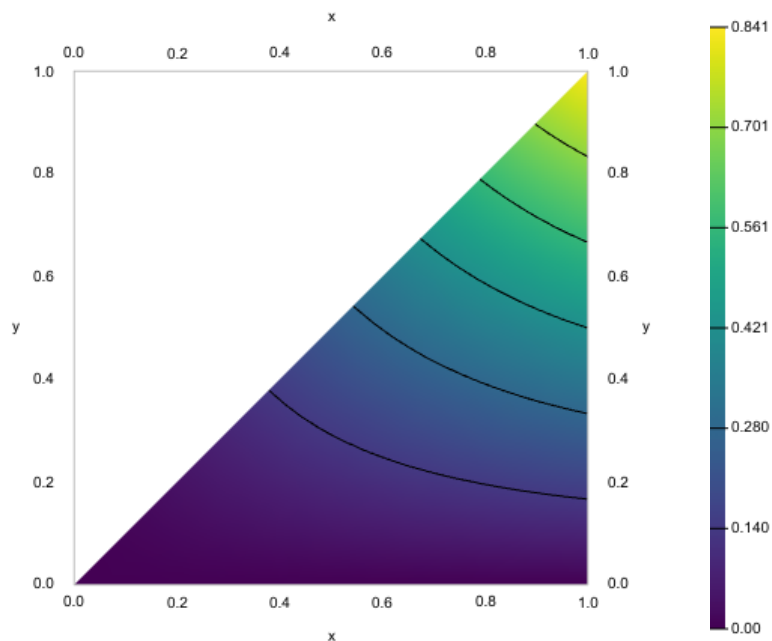
```
[0.0, 0.0, 0.841471, 0.38939, 0.420735, 0.631103, 0.603703, 0.467345, 0.720622, 0.736287, 0.6
```

```
• fsin2=map((x,y)-> sin(x)*y, grid2d2)
```

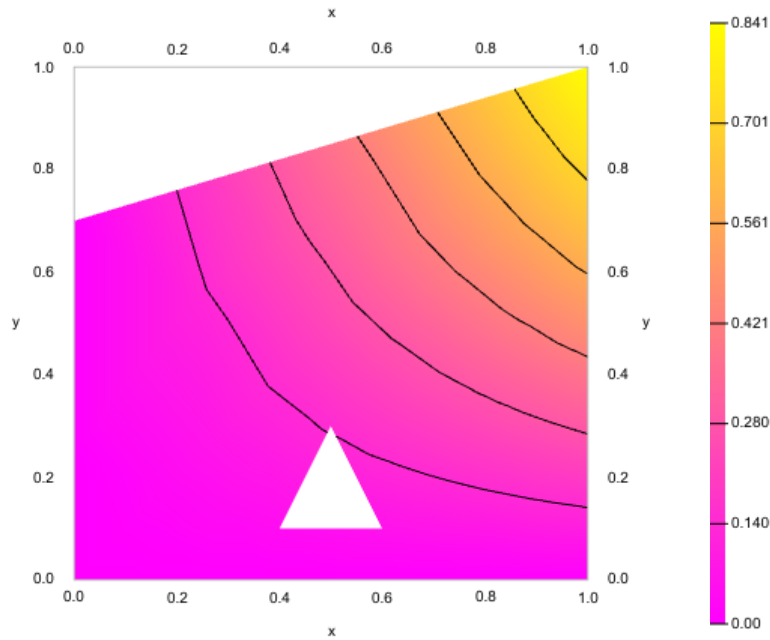
`fsin3 =`

```
[0.0, 0.0, 0.841471, 0.0, 0.0399334, 0.0599, 0.14776, 0.0, 0.122412, 0.0, 0.0, 0.38939, 0.0
```

```
• fsin3=map((x,y)-> sin(y)*x, grid2d3)
```



```
• scalarplot(grid2d2, fsin2, label="grid2d2")
```

```
• scalarplot(grid2d3, fsin3, label="grid2d3", colormap=:spring, isolines=10)
```

3D Grids

Tensor product grids

Please note that "masking" is not yet implemented. Furthermore, PyPlot visualization is slow, with GLMakie it is way faster.

```
X3 = 0.0:1.01:10.1
```

```
• X3=range(0.,10.1,length=11)
```

```
grid3d1 = ExtendableGrids.ExtendableGrid{Float64, Int32};
          dim: 3 nodes: 1331 cells: 6000 bfaces: 1200
```

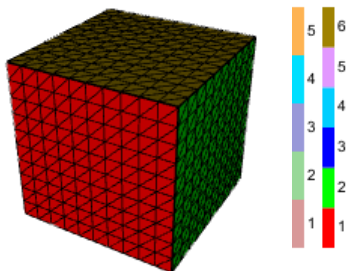
```
• grid3d1=simplexgrid(X3,X3,X3)
```

```
func3 =
```

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
```

```
• func3=map((x,y,z)-> sin(x/2)*cos(y/2)*z/10,grid3d1)
```

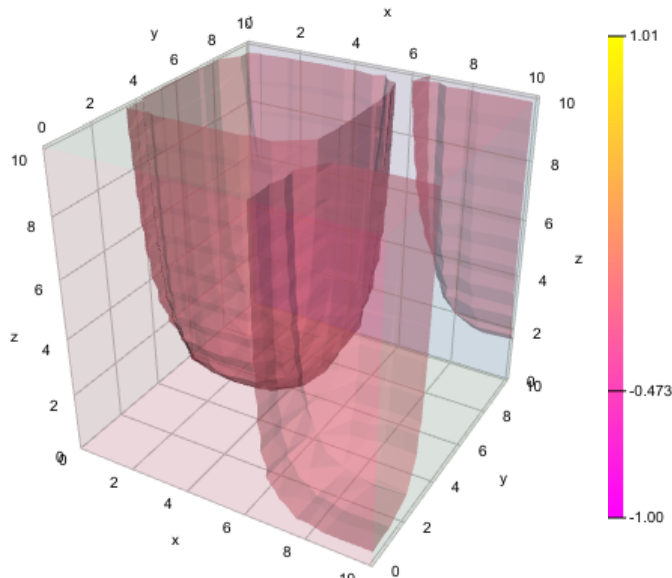
```
p3dg =
```



```
• p3dg=GridVisualizer(dim=3,resolution=(200,200))
```

```
• gridplot!(p3dg,grid3d1,zplanes=[zplane],yplanes=[yplane], xplanes=[xplane],
            resolution=(200,200),show=true)
```

p3ds =



```
• p3ds=GridVisualizer(dim=3,resolution=(400,400))
```

f= -0.473054741596316

x= 10.1

y= 10.1

z= 10.1

```
• md """
• f=$(@bind flevel
  Slider(range(extrema(func3)...,length=20),default=mean(func3),show_value=true))
•
• x=$(@bind xplane Slider(X3[1]:0.1:X3[end],default=X3[end],show_value=true))
•
• y=$(@bind yplane Slider(X3[1]:0.1:X3[end],default=X3[end],show_value=true))
•
• z=$(@bind zplane Slider(X3[1]:0.1:X3[end],default=X3[end],show_value=true))
•
• """
```

mean (generic function with 1 method)

```
• mean(x)=sum(x)/length(x)
```

Unstructured grids

The SimplexGridBuilder API supports creation of three-dimensional grids in way very similar to the 2D case. Just define points with three coordinates and planar (!) facets with at least three points to describe the geometry.

The backend for mesh generation in this case is the [TetGen](#) mesh generator by Hang Si from WIAS Berlin and its Julia wrapper [TetGen.jl](#).

```

• builder3d=let
•
•     b=SimplexGridBuilder(Generator=TetGen)
•
•     p1=point!(b,0,0,0)
•     p2=point!(b,1,0,0)
•     p3=point!(b,1,1,0)
•     p4=point!(b,0,1,0)
•     p5=point!(b,0,0,1)
•     p6=point!(b,1,0,1)
•     p7=point!(b,1,1,1)
•     p8=point!(b,0,1,1)
•
•     facetregion!(b,1)
•     facet!(b,p1 ,p2 ,p3 ,p4)
•     facetregion!(b,2)
•     facet!(b,p5 ,p6 ,p7 ,p8)
•     facetregion!(b,3)
•     facet!(b,p1 ,p2 ,p6 ,p5)
•     facetregion!(b,4)
•     facet!(b,p2 ,p3 ,p7 ,p6)
•     facetregion!(b,5)
•     facet!(b,p3 ,p4 ,p8 ,p7)
•     facetregion!(b,6)
•     facet!(b,p4 ,p1 ,p5 ,p8)
•
•
•     hp1=point!(b,0.4,0.4,0.4)
•     hp2=point!(b,0.6,0.4,0.4)
•     hp3=point!(b,0.6,0.6,0.4)
•     hp4=point!(b,0.4,0.6,0.4)
•     hp5=point!(b,0.4,0.4,0.6)
•     hp6=point!(b,0.6,0.4,0.6)
•     hp7=point!(b,0.6,0.6,0.6)
•     hp8=point!(b,0.4,0.6,0.6)
•
•     facetregion!(b,7)
•     facet!(b,hp1 ,hp2 ,hp3 ,hp4)
•     facet!(b,hp5 ,hp6 ,hp7 ,hp8)
•     facet!(b,hp1 ,hp2 ,hp6 ,hp5)
•     facet!(b,hp2 ,hp3 ,hp7 ,hp6)
•     facet!(b,hp3 ,hp4 ,hp8 ,hp7)
•     facet!(b,hp4 ,hp1 ,hp5 ,hp8)
•     holepoint!(b, 0.5,0.5,0.5)
•
•     b
•
• end;

```

```

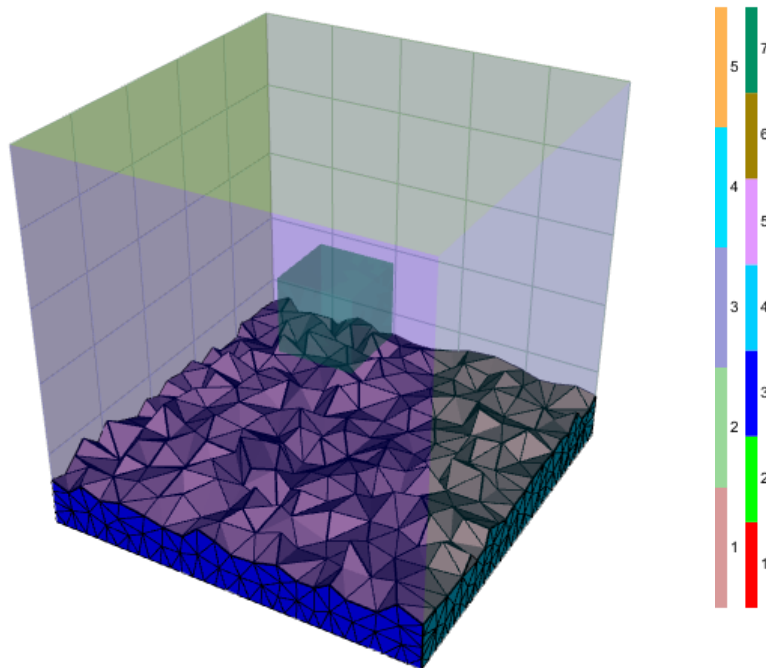
grid3d2 = ExtendableGrids.ExtendableGrid{Float64, Int32};
          dim: 3 nodes: 4650 cells: 21311 bfaces: 4890

```

```

• grid3d2=simplexgrid(builder3d,maxvolume=0.0001)

```



```
• gridplot(grid3d2, zplane=0.1, azim=20, elev=20, linewidth=0.5, outlinealpha=0.3)
```

Table of Contents

Grid creation and visualization in PDELib.jl

1D grids

- Modifying region markers
- Creating locally refined grids
- Plotting functions

2D grids

- Tensor product grids
- Unstructured grids
- More complicated grids
 - Local refinement
 - Interior boundaries
 - Subregions
 - Holes
- Plotting of functions

3D Grids

- Tensor product grids
- Unstructured grids

```
• TableOfContents(depth=4)
```

```
begin
  using Pkg
  Pkg.activate(mktempdir())
  Pkg.add(["PlutoUI", "Revise", "Triangulate", "TetGen"])
  using Revise
  Pkg.develop(["ExtendableGrids", "SimplexGridFactory",
              "GridVisualize", "PlutoVista"])
end
```

```
• md"""
•   begin
•     using Pkg
•     Pkg.activate(mktempdir())
•     Pkg.add(["PlutoUI", "Revise", "Triangulate", "TetGen"])
•     using Revise
•     Pkg.develop(["ExtendableGrids", "SimplexGridFactory",
•                 "GridVisualize", "PlutoVista"])
•   end
• """
```