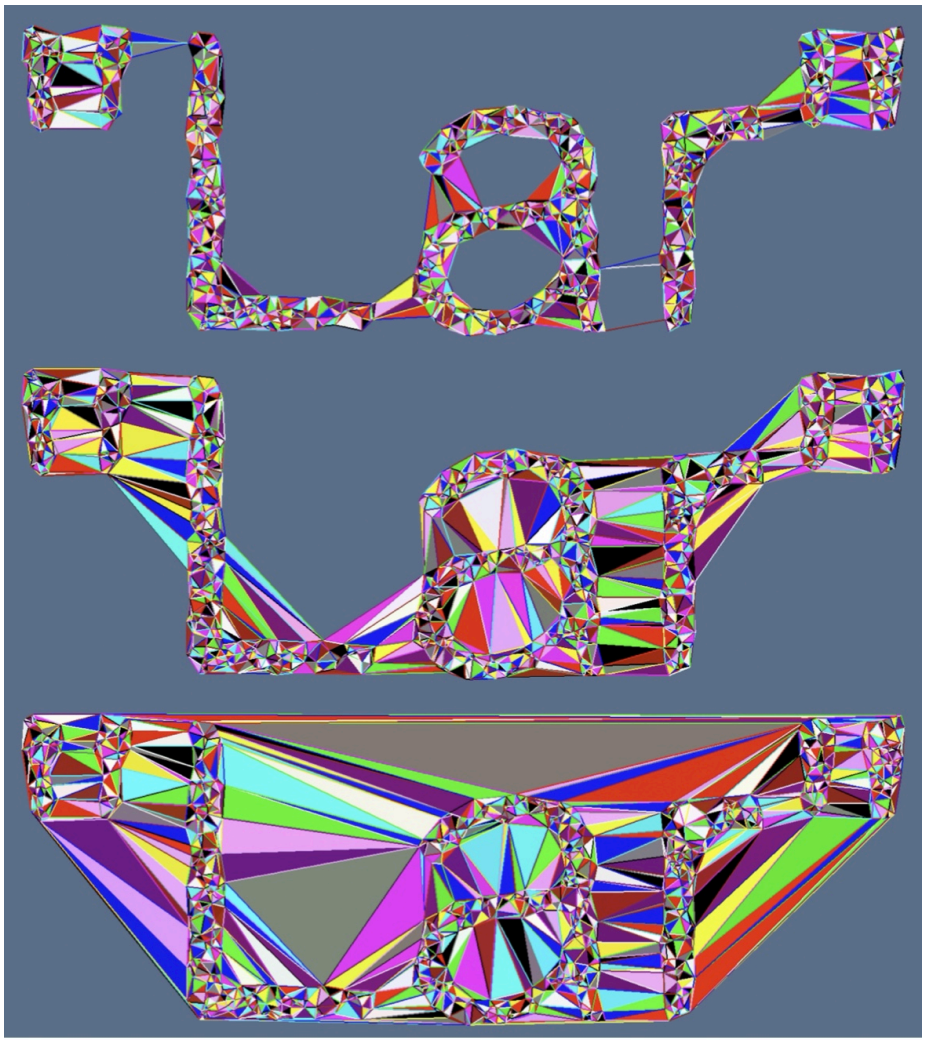
Progetto Alphashapes



link al progetto https://github.com/luigibvl/AlphaShapes.jl

Alphashapes functions

```
function alphaFilter(V::Lar.Points,
                     DT = Array{Int64,1}[];
                     digits=64)::DataStructures.SortedDict{}
```

Restituisce una raccolta ordinata di coppie (caratteristica alfa, complessi). Questo metodo valuta il filtro per un gruppo di punti V su un piano. Se Delaunay Triangulation DT non è specificato, viene valutato tramite AlphaStructures.delaunayTriangulation().

Tasks:

- `function size(V::Lar.Points,n::Int64)::Int64`
- `function updateFiltration(filtration::DataStructures.SortedDict{Array{Int64,1}, Float64})`
- `function delaunayTriangulation(V::Lar.Points)::Lar.Cells`
- `function processuppersimplex(V::Lar.Points,up_simplex::Array{Int64,1}, filtration::DataStructures.SortedDict{}; digits=64)`

```
function processuppersimplex(V::Lar.Points,up_simplex::Array{Int64,1},
                             filtration::DataStructures.SortedDict{};digits=64)
```

Elabora il simplesso superiore.

Tasks:

- `function findRadius(P::Lar.Points, center=false; digits=64) ::Union{Float64, Tuple{Float64, Array{Float64,1}}}`
- `function createNewSimplex(up_simplex::Array{Int64,1}, d::Int64)::Array{Array{Int64,1}}`
- `function processlowsimplex(V::Lar.Points,up_simplex::Array{Int64,1}, filtration::DataStructures.SortedDict{}; digits=64)`

```
function processlowsimplex(V::Lar.Points,
                           up_simplex::Array{Int64,1},
                           lowsimplex::Array{Int64,1},
                           filtration::DataStructures.SortedDict{};
                           digits=64)
```

Elabora il simplesso inferiore conoscendo quello superiore.

Tasks:

- `function findRadius(P::Lar.Points, center=false; digits=64)`
  `::Union{Float64, Tuple{Float64, Array{Float64,1}}}`
- `function createPoint(V::Lar.Points, up_simplex::Array{Int64,1},`
  `lowsimplex::Array{Int64,1})::Array{Float64,2}`
- `function vertexInCircumball(P::Lar.Points, _char::Float64,`
  `point::Array{Float64,2})::Bool`
- `function updateLowSimplex(d::Int,V::Lar.Points,up_simplex::Array{Int64,1},`
  `lowsimplex::Array{Int64,1},`
  `filtration::DataStructures.SortedDict{};`
  `digits=64)`
  - `function Combinations(lowsimplex::Array{Int64, 1},d::Int)`
  - `function processlowsimplex(V::Lar.Points,up_simplex::Array{Int64,1},`
    `lowsimplex::Array{Int64,1},`
    `filtration::DataStructures.SortedDict{};`
    `digits=64)`

`function alphaSimplex(V::Lar.Points,`
`filtration::DataStructures.SortedDict{},`
`_threshold::Float64)::Array{Lar.Cells,1}`

Restituisce la raccolta di tutti i d-simplessi, per d [0, dimensione], con la caratteristica minore rispetto al valore _threshold.

Tasks:

- `function size(V::Lar.Points)::Tuple{Int64}`
- `function createSimplexCollection(V::Lar.Points,dim::Int64)`
  `::Array{Array{Int64,1},1}`
- `function updateSimplexCollection(filtration::DataStructures.SortedDict{},`
  `_threshold::Float64,`
  `simplexCollection::Array{Int64,1})`
- `function sort!(simplexCollection::Array{Int64,1})`

`function delaunayTriangulation(points::Lar.Points)::Lar.Cells`

Restituisce i simplessi di livello più alto della triangolazione di Delaunay.

Tasks:

- `function size(V::Lar.Points,n::Int64)::Int64`

  i task vcat, sortprem e upper_simplex saranno eseguiti se size=1 altrimenti sarà eseguito il task delaunayMATLAB

- `function vcat(V::Lar.Points)::Array{Int64,1}`
- `function sortprem(vertices::Array{Int64,1})::Array{Int64,1}`
- `function upper_simplex(p::Array{Int64,1})::Array{Int64,2}`
- `function delaunayMATLAB(V::Lar.Points)::Array{Int64, 1}`

`function delaunayMATLAB(V::Lar.Points)::Array{Int64, 2}`

Interfaccia MATLAB per la Delaunay Triangulation eseguendo le opportune conversioni dei LAR points

Tasks:

- `function size(V::Lar.Points,n::Int64)::Int64`
- `function convert(V'::Array{Int64,1})::Lar.Points`
- `function mat(w:Lar.Points)`
- `function convert(Array{Int64,2},DT)::Array{Int64, 2}`
- `function getDelaunayTriangulation(DT::Array{Int64,2})::Array{Int64, 2}`

```
function delaunayWall(P::Lar.Points, ax = 1, Pblack = Float64[],
                        AFL = Array{Int64,1}[],
                        tetraDict = DataStructures.Dict{Array{Int64,1},
                        Array{Float64,1}}();DEBUG = false)::Lar.Cells
```

Restituisce la triangolazione Delaunay del gruppo di punti P tramite l'algoritmo Delaunay Wall. L'argomento opzionale ax specifica su quale asse costruiremo Wall. L'argomento opzionale AFL viene utilizzato nella chiamata ricorsiva. Se l'argomento della parola chiave DEBUG è impostato su true, viene mostrata tutta la procedura.

Tasks:

- `function findMedian(P::Lar.Points, ax::Int64)::Float64`
- `function checkPblack(Pblack::Float64[])::Lar.Points`

  se AFL is empty do firstSimplex and its subtasks
- `function firstSimplex(AFL::Array{Int64,1})::Array{Array{Int64,1},1}`
  - `function firstDeWallSimplex(P::Lar.Points,ax::Int64, off::Float64;DEBUG = false)::Array{Int64,1}`
  - `function simplexFaces( ::Array{Int64,1})::Array{Array{Int64,1},1}`

  - `function updateTetraDict!(P::Lar.Points,`
    `tetraDictDataStructures.Dict{Array{Int64,1},`
    `Array{Float64,1}}(),`
    `AFL::Array{Int64,1}, ::Array{Int64,1})::Nothing`

- `function updateAFL!(P::Lar.Points,new ::Array{Array{Int64,1},1},`
  `AFL ::Array{Array{Int64,1},1},`
  `AFLplus::Array{Array{Int64,1},1},`
  `AFLminus::Array{Array{Int64,1},1},`
  `ax::Int64, off::Float64;`
  `DEBUG =false)::Bool`

  while (AFL) non è vuoto do buildSimplexWall and its subtasks

- `function buildSimplexWall(AFL ::Array{Int64,1})`
  - `function findWallSimplex(P::Lar.Points,face::Array{Int64,1},`
    `åoppoint::Array{Float64,1},`
    `blackidx = size(P, 2);DEBUG = false)`
    `::Union{Array{Int64,1},Nothing})`

    se simplex esiste e non appartiene alla Delaunay Triangulation do task simplexFaces, updateTetraDict! e updateAFL!

  - `function simplexFaces( ::Array{Int64,1})::Array{Array{Int64,1},1}`
  - `function updateTetraDict!(P::Lar.Points,`
    `tetraDictDataStructures.Dict{Array{Int64,1},`
    `Array{Float64,1}}(),AFL::Array{Int64,1},`
    `::Array{Int64,1})::Nothing`
  - `function updateAFL!(P::Lar.Points,new ::Array{Array{Int64,1},1},`
    `AFL ::Array{Array{Int64,1},1},`
    `AFLplus::Array{Array{Int64,1},1},`
    `AFLminus::Array{Array{Int64,1},1},`
    `ax::Int64, off::Float64;`
    `DEBUG = false)::Bool`

  se AFLminus non è vuoto

- `function union!(AFLminus::Array{Int64,1}, DT::Array{Int64,1}[],`
  `recursiveDelaunayWall)`
  - `function recursiveDelaunayWall(AFLminus::Array{Int64,1},P::Lar.Points,`
    `Pblack::Array{Float64},`
    `tetraDict::DataStructures.Dict{Array{Int64,1},`
    `Array{Float64,1}},`
    `AFL::Array{Array{Int64,1},1},`
    `ax::Int64,off::Float64,`
    `positive::Bool;DEBUG = false)`

  se AFLplus non è vuoto

- `function union!(AFLminus::Array{Int64,1}, DT::Array{Int64,1}[], recursiveDelaunayWall)`
  - `function recursiveDelaunayWall(AFLplus::Array{Int64,1}, P::Lar.Points,Pblack::Array{Float64}, tetraDict::DataStructures.Dict{Array{Int64,1}, Array{Float64,1}}, AFL::Array{Array{Int64,1},1}, ax::Int64,off::Float64, positive::Bool;DEBUG = false)`

```
function findWallSimplex(P::Lar.Points,
                         face::Array{Int64,1},
                         oppoint::Array{Float64,1},
                         blackidx = size(P, 2);
                         DEBUG = false)
                         ::Union{Array{Int64,1}, Nothing}
```

Restituisce la costruzione del simplesso ‘ ’ con una faccia e un punto da P[:, 1: blackidx] tale che si trova nel semipiano opposto del punto opposto. Se un tale simplesso non esiste, non viene restituito nulla. Se blackidx non è specificato tutti i punti P sono considerati validi. Se l'argomento della parola chiave DEBUG è impostato su true, viene mostrata tutta la procedura.

Tasks:

- `function sort(face::Array{Int64,2})::Array{Int64,2}`
- `function oppositeHalfSpacePoints(P::Lar.Points,face::Array{Float64,2}, point::Array{Float64,1})::Array{Int64,1}`
- `function findClosestPoint(Psimplex::Lar.Points, P::Lar.Points; metric = "circumcenter") ::Union{Int64, Nothing}`
- `function findRadius(P::Lar.Points, center=true; digits=64) ::Union{Float64, Tuple{Float64, Array{Float64,1}}}`
- `function simplex_correctness(P::Lar.Points)::Nothing`

  se Lar.norm = true return nothing
  - `function Lar.norm(center::Array{Float64,1},P::Lar.Points,radius::Float64) ::Float64`

```
function firstDeWallSimplex(P::Lar.Points,
                            ax::Int64,
                            off::Float64;
                            DEBUG = false)::Array{Int64,1}
```

Restituisce l'array degli indici dei punti appartenenti a P che formano il primo tetraedro costruito attraverso the Wall specificando l'asse ax ed il termine 'off'. Se l'argomento della parola chiave DEBUG è impostato su true, viene mostrata tutta la procedura.

Tasks:

- `function size(V::Lar.Points,n::Int64)::Int64`
- `function size(V::Lar.Points,n::Int64)::Int64`
- `function findallPselection(P::Lar.Points)::Array{Int64,1}`
- `function findmax(P::Lar.Points)::Array{Int64,1}`
- `function findallPselection(P::Lar.Points)::Array{Int64,1}`

  per tutti i valori della dimensione dim, do findClosestPoint e updatePselection
- `function findClosestPoint(P::Lar.Points,face::Array{Int64,1},`
  `oppoint::Array{Float64,1})`
  `::Union{Int64, Nothing}`
- `function updatePselection(n::Array{Int64,1})::Array{Int64,1}`
- `function findRadius(Psimplex::Lar.Points, center=false; digits=64)`
  `::Union{Float64, Tuple{Float64, Array{Float64,1}}}`

```
function recursiveDelaunayWall(P::Lar.Points,
        Pblack::Array{Float64},
        tetraDict::DataStructures.Dict{Array{Int64,1},Array{Float64,1}},
        AFL::Array{Array{Int64,1},1},
        ax::Int64,
        off::Float64,
        positive::Bool;
        DEBUG = false)::Lar.Cells
```

Funzione di utilità che prepara la fase 'divide' per Delaunay Wall. Restituisce la triangolazione di Delaunay per il sottospazio positivo o negativo di P determinato dall'iperpiano dall'asse normale e dal termine 'off'. Se l'argomento della parola chiave DEBUG è impostato su true, viene mostrata tutta la procedura.

Tasks:

- `function size(V::Lar.Points)::Tuple{Int64}`
- `function findallPsubset(P::Lar.Points)::Array{Int64,1}`
- `function setdiffBlacklist(tetraDict::DataStructures.Dict{Array{Int64,1},`
  `Array{Float64,1}},`
  `Psubset::Array{Int64,1})::Array{Int64,1}`

da parallelizzare il quarto e quinto parametro di DTdelaunayWall poichè sono delle funzioni findall

- `function DTdelaunayWall(P::Lar.Points,ax = 1,Pblack = Float64[], AFL = Array{Int64,1}[], tetraDict = DataStructures.Dict{Array{Int64,1}, Array{Float64,1}}();DEBUG = false)::Lar.Cells`
- `function updatePsubset(DT::Lar.Cells,Psubset::Array{Int64,1})::Lar.Cells`

```
function updateAFL!(P::Lar.Points,
                    new ::Array{Array{Int64,1},1},
                    AFL ::Array{Array{Int64,1},1},
                    AFLplus::Array{Array{Int64,1},1},
                    AFLminus::Array{Array{Int64,1},1},
                    ax::Int64, off::Float64;
                    DEBUG = false)::Bool
```

Modificare le liste di facce `AFL*` aggiungendo ad essi le facce all'interno di quella nuova(new) (che si riferisce ai punti `P`) secondo la loro posizione rispetto all'asse definito dalla normale direzione `ax` e il termine contante "off". La funzione restituisce un valore Bool che indica se l'operazione è stata eseguita correttamente. Se l'argomento della parola chiave "DEBUG" è impostato su true, viene mostrata tutta la procedura.

Tasks:

`per ogni elemento della lista delle facce`

- `function planarIntersection(P::Lar.Points,new ::Array{Array{Int64,1},1}, ax::Int64, off::Float64;)::Int64`

  update based on intersection

- `function updateList!(list, element)::Bool`

```
function updatelist!(list, element)::Bool
```

Se l'elemento è nella lista, viene rimosso (restituisce `false`). Se l'elemento non è nella lista, viene aggiunto (restituisce `true`).

```
function updateTetraDict!(P::Lar.Points,
                          tetraDict::DataStructures.Dict{Array{Int64,1},
                          Array{Float64,1}},
                          AFL::Array{Array{Int64,1},1},
                           ::Array{Int64,1})::Nothing
```

Aggiorna il contenuto di `tetraDict` aggiungendo i punti esterni delle facce di nel dizionario.

Tasks:

`per ogni elemento della lista delle facce`

- `function setdiff( ::Array{Int64,1},AFL::Array{Array{Int64,1},1})`
  `::Array{Int64,1}`

```
function findCenter(P::Lar.Points)::Array{Float64,1}
```

Valuta il circumcentro dei punti `P`. Se i punti si trovano su un circumcerchio `d-1`, la funzione non è in grado di eseguire la valutazione e quindi restituisce un array `NaN`.

Tasks:

- `function size(V::Lar.Points)::Tuple{Int64}`

  n = numero di punti dim = R2 o R3

  se n = 3 e dim = 2 do task DenomDef e DeterDef

- `function DenomDef(P::Lar.Points)::Float64`
- `function DeterDef(P::Lar.Points)::Float64`

  se n=3 e dim=3 do task DenomDefForThirdDimension e DeterDefForThirdDimension

- `function DenomDefForThirdDimension(P::Lar.Points)::Float64`
- `function DeterDefForThirdDimension(P::Lar.Points)::Float64`

  se n=4 do task Det, sum ,DX , DY e DZ

- `function Det(P::Lar.Points)::Float64`
- `function sum(P::Lar.Points)sq::Float64`
- `function DX(sq::float64,P::Lar.Points)::Float64`
- `function DY(sq::float64,P::Lar.Points)::Float64`
- `function DZ(sq::float64,P::Lar.Points)::Float64`

```
function findClosestPoint(Psimplex::Lar.Points, P::Lar.Points;
                          metric = "circumcenter")::Union{Int64, Nothing}
```

Restituisce l'indice del punto in `P` più vicino ai punti `Psimplex`, in base alla parola chiave `metric`. Le scelte possibili sono:

- `circumcenter`: (predefinito) restituisce il punto che minimizza il circumradius
- `dd`: come `circumcenter` ma il circumradius è considerato negativo se il circumcenter è opposto al nuovo punto rispetto a `Psimplex`

Tasks:

- `function size(V::Lar.Points,n::Int64)::Int64`

  per ogni colonna in P do task findRadius e oppositeHalfSpacePoints
- `function findRadius(P::Lar.Points, center=false; digits=64)`
  `::Union{Float64, Tuple{Float64, Array{Float64,1}}}`
- `function oppositeHalfSpacePoints(P::Lar.Points,face::Array{Float64,2},`
  `point::Array{Float64,1})::Array{Int64,1}`
- `function findmin(radList::SharedArray{Float64})::Union{Float64,Int64}`

```
function findMedian(P::Lar.Points, ax::Int64)::Float64
```

Restituisce la mediana dei punti `P` sull'asse `ax`

Task:

- `function sort(face::Array{Int64,2})::Array{Int64,2}`

```
function findRadius(P::Lar.Points, center=false; digits=64)
                    ::Union{Float64, Tuple{Float64, Array{Float64,1}}}
```

Restituisce il valore del raggio circumcerchio dei punti dati. Se la funzione findCenter non è in grado di determinare il circumcenter, la funzione restituisce `Inf`. Se l'argomento opzionale `center` è impostato a `true`, la funzione restituisce anche le coordinate cartesiane del circumcenter.

Tasks:

- `function findCenter(P::Lar.Points)::Array{Float64,1}`

se center !=NAN then per ogni punto in P do task roundFindmin

- `function roundFindmin(P::Lar.Points,c::Array{float64,1})::Union{Int64,Int64}`

```
function matrixPerturbation(M::Array{Float64,2};
                            atol=1e-10, row = [0],
                            col = [0])::Array{Float64,2}
```

Restituisce la matrice `M` con una perturbazione, che può essere sia negativa che positiva, su ogni valore determinato dalle i-esime righe e dalle j-esime colonne. Se 'row'/'col' sono impostati a '[0]' (o non specificati), tutte le righe/colonne sono perturbate.

Tasks:

`per ogni elemento della riga della matrice do rowPerturbation`

- `function rowPerturbation(M::Array{Float64,2})::Array{Float64,1}`

  per ogni elemento della colonna della matrice do colPerturbation

- `function colPerturbation(M::Array{Float64,2})::Array{Float64,1}`

```
function oppositeHalfSpacePoints(P::Lar.Points,
                                 face::Array{Float64,2},
                                 point::Array{Float64,1})::Array{Int64,1}
```

Restituisce la lista indice dei punti `P` situati nel semispazio definito dalle facce che non contengono il punto.

Tasks:

- `function size(V::Lar.Points)::Tuple{Int64}`
- `function size(V::Lar.Points,n::Int64)::Int64`

  se dimP=1 do oppositeCalculationForOneDimension

- `function oppositeCalculationForOneDimension(P::Lar.Points,threshold::Float64,`
  `major::Bool)::Array{Float64,1}`

  se dimP=2 e check dimensions do oppositeCalculationForTwoDimension-IfCheck else do oppositeCalculationForTwoDimensionIfNotCheck

- `function oppositeCalculationForTwoDimensionIfCheck(n::Int64,m::Float64, P::Lar.Points, q::Float64,side::Int64) ::Array{Float64,1}`
- `function oppositeCalculationForTwoDimensionIfNotCheck(n::Int64,P::Lar.Points, q::Float64, side::Int64) ::Array{Float64,1}`

  se dimP=3 do oppositeCalculationForThirdDimension (vanno anche parallelizzate le funzioni Lar.dot e Lar.cross)
- `function oppositeCalculationForThirdDimension(axis::Array{Int64,1}, P::Lar.Points, off::Array{Int64,1}, major::Bool) ::Array{Float64,1}`
- `function PIndexList(P::Lar.Points,face::Array{Float64,2})::Array{Int64,1}`

```
function planarIntersection(P::Lar.Points,
                            face::Array{Int64,1},
                            axis::Int64,
                            off::Float64)::Int64
```

Calcola la posizione di `face` rispetto all'iperpiano , definito da `axis` e dal termine 'off'. Ritorna:

- `0` se `f` interseca internamente (non solo il confine)
- `+1` se `f` è completamente contenuto nel semispazio positivo di
- `-1` se `f` è completamente contenuto nel semispazio negativo di

Tasks:

- `function getPosition(P::Lar.Points,face::Array{Int64,1}, axis::Int64,off::Float64)::Array{Int64,2}`
- `function sumPointsOnAxis(P::Lar.Points,face::Array{Int64,1}, axis::Int64,off::Float64)::Array{Int64,2}`

  if return of sumPointsOnAxis != length(return of getPosition) do sumReslultOfGetPosition
- `function sumReslultOfGetPosition(a::Array{Int64,2})::Int64`

```
function simplexFaces( ::Array{Int64,1})::Array{Array{Int64,1},1}
```

Restituisce le facce del simplesso .

Tasks:

- `function collectCombinations( ::Array{Int64,1},d::Int64)::Array{Array{Int64,1},1}`
- `function sort!(toSort::Array{Array{Int64,1},1})`

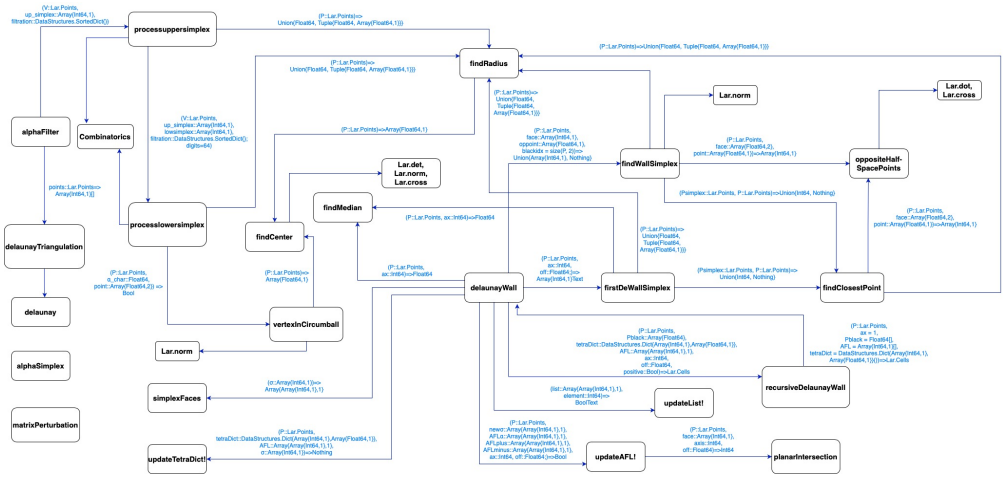
```
function vertexInCircumball(P::Lar.Points,
                            _char::Float64,
                           point::Array{Float64,2})::Bool
```

Determina se un punto è interno alla circumcerchio determinata dai punti `P` e dal raggio `_char`.

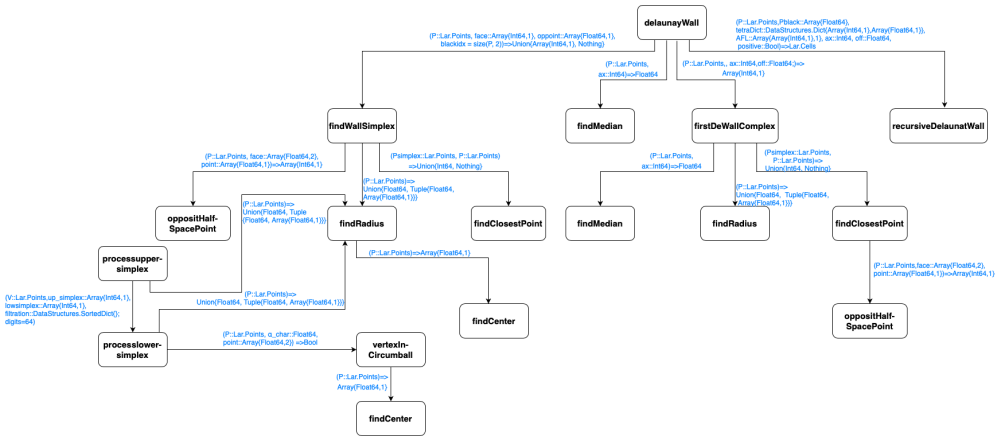
Tasks:

- `function findCenter(P::Lar.Points)::Array{Float64,1}`
- `function Lar.norm(f::Float64)::Float64`

Grafo delle dipendenze



Ristrutturazione del grafo delle dipendenze



Grafo ristrutturato dividendo le funzioni in task