Relazione LAR Splitting 2D

Studenti: Caponi Marco, Ceneda Gianluca

Prime analisi, test e possibili ottimizzazioni sul progetto LAR SPLITTING 2D con l'utilizzo della seguente repository:

- Main repository: https://github.com/MarcoCap13/LAR-SPLITTING-2D-5.b-
  - https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/src/refactoring.jl
  - https://github.com/cvdlab/Lar.jl/blob/master/src/fragface.jl

Obiettivi:

- Studio del progetto **LAR SPLITTING 2D** e di tutte le funzioni e strutture dati utilizzate.
- Descrizione di ogni task individuata, tipo e significato di ogni parametro ed eventuale valore di ritorno.
- Suddivisione delle tipologie di funzioni e creazione di grafi delle dipendenze.
- Individuare eventuali problemi riscontrati durante lo studio preliminare del codice.

Analisi Preliminare

Lo studio del nostro progetto si basa sull'ottimizzazione delle funzioni principali della classe **refactoring** e **fragface** tra cui: *spaceindex*, *pointInPolygonClassification* e *fragmentLines*. Nello specifico, ci occupiamo della generazione di una proiezione 2D del piano euclideo di ciascuna faccia, costituita da un insieme di forme solide 3D o da un insieme di primitive 1D (es: linee, poligoni, cerchi etc.). * **Input:** In input abbiamo o una struttura *LAR*, da trasformare in un insieme di poligoni sia 2D che 1D oppure un array formato dagli stessi elementi facente parte dello stesso sistema di coordinate. * **Output:** In output invece abbiamo una collezione di elementi 2D inserite in due matrici diagonali.

Riassumendo, in questa sezione affronteremo lo studio preliminare delle funzioni principali della classe *refactoring* e *fragface*:

Analisi input e output delle funzioni della classe Refactoring:

- **CrossingTest:** è una funzione di supporto per la funzione primaria *pointInPolygonClassification*. Aggiorna il count a seconda dello stato identificato come 'new' oppure 'old', incrementiamo nello specifico di 0.5
- **setTile:** è una funzione che imposta il *tileCode* della bounding box 2D [b1, b2, b3, b4] includendo i "point" delle coordinate 2D x, y. A seconda della posizione della variabile 'point', tileCode ha un range di 0:15 e usa l'operatore di bit. Successivamente testeremo il codice di TileCode

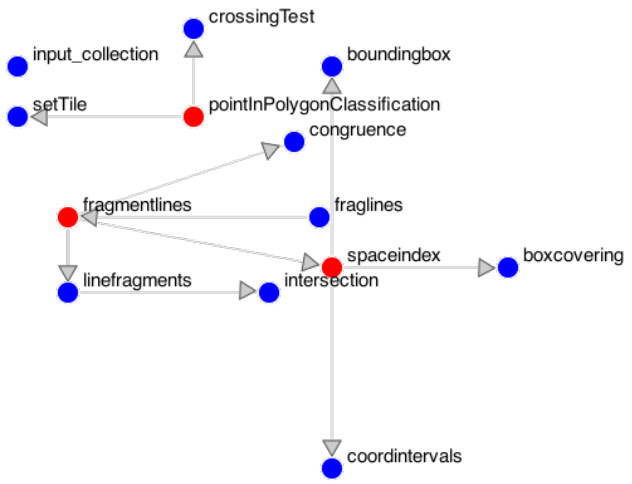

Figure 1: Grafo delle dipendenze della classe Refactoring

riguardante i bordi di un poligono 2D per determinare se i punti della variabile 'point' sono interni, esterni o sono situati sul confine del poligono.

- **pointInPolygonClassification:** questa funzione è fondamentale per identificare se i punti del poligono sono *interni*, *esterni* o sono di *frontiera* (ovvero sul bordo del poligono)
- **input_collection:** seleziona una 'faccia' e costruisce una collezione di dimensione (d-1). L'output è un input ammissibile per gli algoritmi inerenti alla pipeline 2D/3D
- **bounding box:** prende in input un vertice di tipo *Lar.point* (in altre parole una matrice MxN dove M è la dimensione dello spazio in analisi e N è il numero di vertici). La funzione restituisce in output due array che indicano gli estremi del bounding box.
- **Coordintervals:** funzione che prende in input una matrice (ovvero i bounding box) e un intero che serve a specificare su quale coordinata si vuole lavorare restituendo in output una lista *boxdict* ordinata.
- **Boxcovering:** prende in input una matrice (ovvero i bounding box), un intero che indica su quale coordinata si sta lavorando e un '*intervalTrees*' restituendo una matrice che contiene tutte le intersezioni tra i bounding box.
- **SpaceIndex:** funzione che prende in input una tupla costituita da una matrice che contiene i punti del modello e da una matrice che contiene le scomposizioni dello spazio geometrico (formato da vertici, lati e facce). Restituisce una matrice covers[k] dove l'elemento *k-esimo* rappresenta quali intersezioni ha il bounding box (k-esimo) con gli altri bounding box.
- **Intersection:** funzione che interseca due segmenti nel piano 2D, calcolando i due parametri di linea del punto di intersezione.
- **Linefragment:** Calcola le sequenze dei parametri ordinati frammentando l'input. Inoltre, i parametri di bordo (0 e 1) sono inclusi nel valore di ritorno dell'output. Il parametro 'Sigma' identifica un indice che fornisce un sottoinsieme di linee il cui contenuto interseca il 'box' di ciascuna linea di input (identificata dal parametro "EV")
- **Fragmentlines:** prende in input il modello e anche grazie a spaceindex calcola e restituisce vertici e spigoli di quest'ultimo.
- **FragLines:** Chiedi al professore riga 485
- **Congruence:** funzione che prende in ingresso un modello di Lar, restituendo una funzione di base denominata *hcat* che concatena due array lungo due dimensioni.

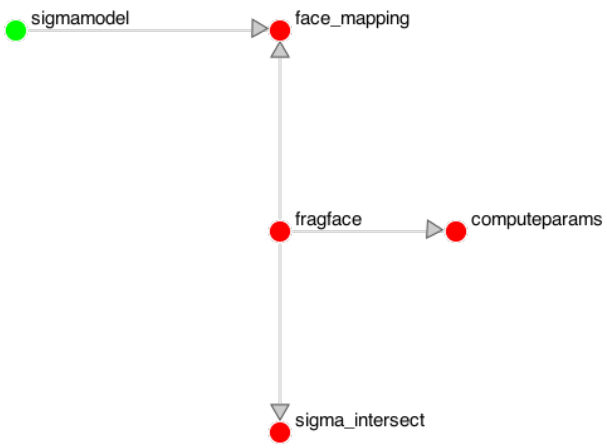

Figure 2: Grafo delle dipendenze della classe Fragface

Analisi input e output delle funzioni della classe Fragface:

- **facemapping:** funzione che calcola la matrice denominata *mapping* per trasformare la variabile 'sigma' e le facce da *sp_idx*[sigma] a z=0. La funzione restituisce una matrice denominata *mapping*.
- **sigmaModel:** funzione che lavora in parallelo con *facemapping* con la differenza che restituisce una serie di valori mappati lungo il piano z=0.
- **sigma_interect:** è una funzione che ci permette di intersecare il piano sigma z= 0 con i bordi usando la trasformata denominata '*bigpi*'. Questa funzione restituisce la lunghezza della variabile *sigma_lines* e l'intersezione delle linee memorizzate nella variabile *linestore*.
- **Computeparams:** funzione che prende in ingresso *linestore* e *linenum* precedentemente calcolate dalla funzione *sigma_intersect* e restituisce un array con tutte le coppie intersecate.
- **Fragface:** funzione che prende in ingresso dei parametri che permettono di modificare le matrici calcolate dalle precedenti funzioni restituendo variabili inerenti alla funzione LAR.