Programmazione e Calcolo Scientifico

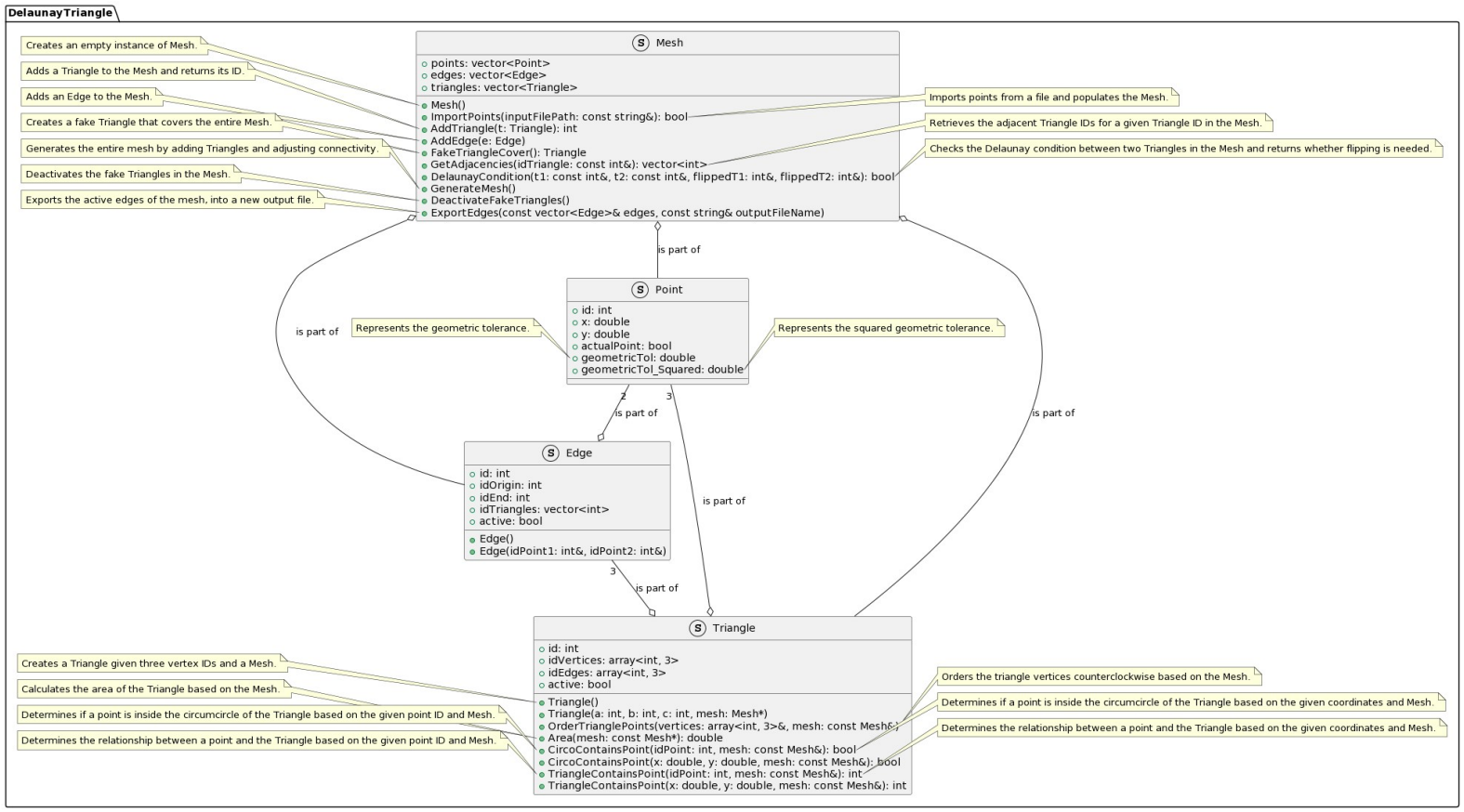
Progetto - Delaunay

Documentazione

Annalisa Belloni s281526
Samuele Bocco s283197
Sara Bonino s282836

[Link](#) al repository di GitHub del progetto.

Diagramma UML



Namespace DelaunayTriangle

Il namespace `DelaunayTriangle` contiene le seguenti classi:

Struct `Point`

```
struct Point
{
    int id;
    double x, y;
    bool actualPoint = true;  // false se il punto è fittizio, true altrimenti

    static constexpr double geometricTol = 1.0e-12;
    static constexpr double geometricTol_Squared = max_tolerance(Point::geometricTol * Point::geometricTol,
                                                   numeric_limits<double>::epsilon());
};
```

La struttura `Point` rappresenta un punto nel piano cartesiano. Ha i seguenti membri:

- `id`: l'identificatore univoco del punto.
- `x`: la coordinata x del punto.
- `y`: la coordinata y del punto.
- `actualPoint`: un flag che indica se il punto è un punto reale (true) o fittizio (false).
- `geometricTol`: una costante che rappresenta la tolleranza geometrica utilizzata per confrontare valori double.
- `geometricTol_Squared`: il quadrato della tolleranza geometrica, calcolato come il massimo tra `geometricTol * geometricTol` e `numeric_limits<double>::epsilon()`.

Struct `Edge`

```
struct Edge
{
    int id;
    int idOrigin, idEnd;
    vector<int> idTriangles;
    bool active = false;
    Edge() = default;
    Edge(int& idPoint1, int& idPoint2);
};
```

La struttura `Edge` rappresenta un lato. Ha i seguenti membri:

- `id`: l'identificatore univoco del lato.
- `idOrigin`: l'identificatore del punto di origine del lato.
- `idEnd`: l'identificatore del punto di arrivo del lato.
- `idTriangles`: un vettore contenente gli identificatori dei triangoli che il lato contribuisce a comporre.
- `active`: un flag che indica se il lato è attivo (true) o inattivo (false).

Costruttore `Edge`

```
Edge::Edge(int& idPoint1, int& idPoint2)
```

Input:

- `int& idPoint1`, `int& idPoint2`: gli ID dei punti che costituiscono l'edge.

Descrizione:
Il costruttore `Edge` crea un oggetto `Edge` con i punti di origine e arrivo specificati dagli identificatori `idPoint1` e `idPoint2`.

Struct Triangle

```
struct Triangle
{
    int id;
    array<int, 3> idVertices;
    array<int, 3> idEdges;
    bool active = false;
    Triangle() = default;
    Triangle(int a, int b, int c, Mesh* mesh);
    void OrderTrianglePoints(array<int, 3>& vertices, const Mesh& mesh);
    double Area(const Mesh* mesh) const;
    bool CircoContainsPoint(const int idPoint, const Mesh& mesh) const;
    bool CircoContainsPoint(const double x, const double y, const Mesh& mesh) const;
    int TriangleContainsPoint(const int idPoint, const Mesh& mesh) const;
    int TriangleContainsPoint(const double x, const double y, const Mesh& mesh) const;
};`
```

La struttura `Triangle` rappresenta un triangolo. Ha i seguenti membri:

- `id`: l'identificatore univoco del triangolo.
- `idVertices`: un array di 3 interi che rappresentano gli identificatori dei punti che formano i vertici del triangolo.
- `idEdges`: un array di 3 interi che rappresentano gli identificatori dei lati che compongono il triangolo.
- `active`: un flag che indica se il triangolo è attivo (true) o inattivo (false).

Costruttore `Triangle(int a, int b, int c, Mesh* mesh)`

```
Triangle(int a, int b, int c, Mesh* mesh)
```

Input:

- `int a`, `int b`, `int c`: gli ID dei punti che costituiscono il triangolo.
- `Mesh* mesh`: un puntatore all'oggetto `Mesh` che contiene i punti.

Descrizione:
Costruisce un oggetto `Triangle` con i vertici specificati dagli ID `a`, `b` e `c`. Il parametro `mesh` rappresenta la mesh a cui il triangolo appartiene.

Metodo `OrderTrianglePoints(array<int, 3>& vertices, const Mesh& mesh)`

```
void OrderTrianglePoints(array<int, 3>& vertices, const Mesh& mesh)
```

Input:

- Input: `array<int ,3>& vertices` : array di ID dei vertici del triangolo
- `const Mesh& mesh`: un riferimento costante all'oggetto `Mesh` che contiene i punti.

Descrizione:
La funzione `OrderTrianglePoints` ordina i vertici del triangolo in senso antiorario. Questo metodo utilizza il *prodotto vettoriale* per determinare l'orientamento dei vertici. I parametri `vertices` rappresentano gli ID dei vertici del triangolo e `mesh` rappresenta la mesh a cui il triangolo appartiene.

Metodo `Area(const Mesh* mesh) const`

```
double Area(const Mesh* mesh) const
```

Input:

- `const Mesh* mesh`: un puntatore costante all'oggetto `Mesh` che contiene i punti.

Output:

- `double`: l'area del triangolo.

Descrizione:
Calcola l'area del triangolo utilizzando il prodotto vettoriale.

Metodo `CircoContainsPoint(const int idPoint, const Mesh& mesh) const`

```
bool CircoContainsPoint(const int idPoint, const Mesh& mesh) const
```

Input:

- Input: `const int idPoint` : ID del punto da verificare
- `const Mesh& mesh` : un riferimento costante all'oggetto `Mesh` che contiene i punti.

Output:

- `bool`: `true` se il punto è interno alla circonferenza, `false` altrimenti.

Descrizione:
Determina se il punto specificato dall'ID `idPoint` è interno o esterno alla circonferenza circoscritta al triangolo.

Metodo `CircoContainsPoint(const double x, const double y, const Mesh& mesh) const`

```
bool CircoContainsPoint(const double x, const double y, const Mesh& mesh) const
```

Input:

- `const double x`, `const double y`: coordinate del punto da verificare.
- `const Mesh& mesh`: un riferimento costante all'oggetto `Mesh` che contiene i punti.

Output:

- `bool`: `true` se il punto è interno alla circonferenza, `false` altrimenti.

Descrizione:
Determina se il punto specificato dalle coordinate `(x, y)` è interno o esterno alla circonferenza circoscritta al triangolo.

Metodo `TriangleContainsPoint(const int idPoint, const Mesh& mesh) const`

```
int TriangleContainsPoint(const int idPoint, const Mesh& mesh) const
```

Input:

- `const int idPoint` : ID del punto da verificare
- `const Mesh& mesh` : un riferimento costante all'oggetto `Mesh` che contiene i punti.

Output:

- `int`: `0` se il punto è esterno, `1` se è interno, `2` se è di bordo.

Descrizione:
Determina se il punto specificato dall'ID `idPoint` è interno, di bordo o esterno al triangolo.

Metodo `TriangleContainsPoint(const double x, const double y, const Mesh& mesh) const`

```
int TriangleContainsPoint(const double x, const double y, const Mesh& mesh) const
```

Input:

- `const double x`, `const double y`: coordinate del punto da verificare.
- `const Mesh& mesh`: un riferimento costante all'oggetto `Mesh` che contiene i punti.

Output:

- `int`: `0` se il punto è esterno, `1` se è interno, `2` se è di bordo.

Descrizione:
Determina se il punto specificato dalle coordinate `(x, y)` è interno, di bordo o esterno al triangolo. Restituisce `0` se il punto è esterno, `1` se è interno o `2` se è di bordo. Il parametro `mesh` rappresenta la mesh a cui il triangolo appartiene.

Struct `Mesh`

```
struct Mesh
{
   vector<Point> points;
   vector<Edge> edges;
   vector<Triangle> triangles;
            bool ImportPoints(const string& inputFilePath);
            int AddTriangle(Triangle& t);
            void AddEdge(Edge& e);
            Triangle FakeTriangleCover();
            vector<int> GetAdjacencies(const int& idTriangle);
   bool DelaunayCondition(const int& t1, const int& t2, int& flippedT1, int& flippedT2);
   void GenerateMesh();
   void DeactivateFakeTriangles();
   void ExportEdges(const vector<Edge>& edges, const string& outputFileName);
};
```

La classe `Mesh` rappresenta una mesh geometrica composta da punti, triangoli e lati. Ha i seguenti membri:

- `points`: un vettore di oggetti `Point` che rappresentano i punti della mesh.
- `edges`: un vettore di oggetti `Edge` che rappresentano i lati della mesh.
- `triangles`: un vettore di oggetti `Triangle` che rappresentano i triangoli della mesh.

Metodo `Mesh::ImportPoints`

```
bool Mesh::ImportPoints(const string& inputFilePath)
```

Input:

- `inputFilePath`: una stringa rappresentante il percorso del file contenente i punti della mesh.

Output:

- `bool`: valore booleano che indica se l'importazione ha avuto successo o meno.

Descrizione:
La funzione `ImportPoints` apre il file specificato e legge i punti in esso contenuti. Ogni riga del file rappresenta un punto e contiene l'identificatore del punto, la coordinata x e la coordinata y. I punti vengono aggiunti al vettore `points` della mesh. Se l'apertura del file fallisce, viene restituito `false`. Altrimenti, viene restituito `true`, dopo aver chiuso il file.

Metodo `Mesh::AddTriangle`

```
int Mesh::AddTriangle(Triangle& t)
```

Input:

- `t`: un riferimento a un oggetto `Triangle` che rappresenta il triangolo da aggiungere alla mesh.

Output:

- `int`: l'identificatore del triangolo aggiunto.

Descrizione:
La funzione assegna un identificatore univoco al triangolo specificato e lo aggiunge al vettore `triangles` della mesh. Inoltre, per ciascuno dei tre lati del triangolo, verifica se il lato esiste già nella mesh o se deve essere creato. Se il lato esiste, aggiunge l'identificatore del triangolo corrente al vettore `idTriangles` del lato e assegna l'identificatore del lato al vettore `idEdges` del triangolo aggiunto. Se il lato non esiste, crea un nuovo oggetto `Edge` e lo aggiunge al vettore `edges` della mesh. Aggiunge poi l'identificatore del triangolo corrente al vettore `idTriangles` del nuovo lato e assegna l'identificatore del nuovo lato al vettore `idEdges` del triangolo aggiunto. Infine, imposta il flag `active` del triangolo a `true` per indicare che è attivo.

Metodo `Mesh::AddEdge`

```
void Mesh::AddEdge(Edge& e)
```

Input:

- `e`: un riferimento a un oggetto `Edge` che rappresenta il lato da aggiungere alla mesh.

Descrizione:
La funzione `AddEdge` assegna un identificatore univoco al lato specificato e lo aggiunge al vettore `edges` della mesh. Il flag `active` del lato viene impostato a `true` per indicare che è attivo.

Metodo `FakeTriangleCover()`

```
Triangle TriangleMesh::FakeTriangleCover()
```

Output:

- `Triangle`: il triangolo coprente.

Descrizione:
Questo metodo restituisce un triangolo che "copre" l'intero set di punti. Il triangolo coprente è ottenuto attraverso un procedimento che coinvolge il calcolo delle coordinate massime e minime dei punti del dataset. Per ogni vertice del suddetto triangolo, viene verificato se i punti del triangolo coprente corrispondono o meno a punti reali del dataset. Se uno o più vertici del triangolo coprente sono fittizi, allora vengono creati nuovi punti (corrispondenti ai vertici mancanti) e successivamente aggiunti al vector `points` della `mesh`.

Metodo `GetAdjacencies(const int& idSource)`

```
vector<int> TriangleMesh::GetAdjacencies(const int& idSource)
```

Input:

- `const int& idSource`: l'ID del triangolo di cui si vogliono ottenere gli adiacenti.

Output:

- `vector<int>`: un vettore contenente gli ID dei triangoli adiacenti al triangolo in questione.

Descrizione:
Viene inizializzato un vettore vuoto `adjacentIds` che conterrà gli ID dei triangoli adiacenti.
Per ogni lato del triangolo di origine, si cercano i triangoli che condividono quell'edge. Gli ID di questi triangoli vengono quindi aggiunti ad `adjacentIds`, escludendo il triangolo d'origine e quelli non attivi.

Metodo `DelaunayCondition`

```
bool Mesh::DelaunayCondition(const int& t1, const int& t2, int& flippedT1, int& flippedT2)
```

Input:

- `const int& t1`: l'ID del primo triangolo, su cui verificare l'ipotesi di Delaunay.
- `const int& t2`: l'ID del secondo triangolo, su cui verificare l'ipotesi di Delaunay.
- `int& flippedT1`: conterrà l'ID del primo triangolo "flippato", in caso di flip.
- `int& flippedT2`: conterrà l'ID del secondo triangolo "flippato", in caso di flip.

Output:

- `bool`: `true` se l'operazione di flip è stata eseguita, altrimenti `false`.

Descrizione:
Questo metodo verifica la condizione di Delaunay tra due triangoli `t1` e `t2`. Se la condizione non è soddisfatta, esegue l'operazione di flip, creando due nuovi triangoli e disattivando i triangoli di

origine e l'edge in comune. Infine restituisce `true` se viene eseguita l'operazione di flip, altrimenti `false`.

Metodo `GenerateMesh()`

```
void Mesh::GenerateMesh()
```

Descrizione:
Per ogni punto del dataset, viene avviato un ciclo for sui triangoli esistenti nella mesh che si interrompe non appena si individua il triangolo che contiene il punto corrente, escludendo i triangoli non attivi. Si procede dunque alla generazione di tre nuovi triangoli (`t1`, `t2` e `t3`), ottenuti collegando il punto corrente con i tre vertici del triangolo trovato, che invece viene disattivato. Se un triangolo è degenere, viene disattivato insieme agli eventuali lati corrispondenti. Altrimenti, viene aggiunto alla mesh.
Successivamente si procede alla verifica della condizione di Delaunay: si avvia un ciclo while finché ci sono triangoli da controllare. Per ciascun triangolo, si ottengono gli ID dei suoi adiacenti e si verifica l'ipotesi di Delaunay tra il triangolo corrente e ciascuno dei suoi adiacenti. Se la condizione non è soddisfatta, viene eseguito un flip per correggere la mesh e i nuovi triangoli "flippati" vengono aggiunti al vector `trianglesToCheckIds`.
Inoltre la funzione `GenerateMesh()` prevede una gestione apposita del problema, per i punti che giacciono sul bordo dei triangoli che li contengono.

Metodo `DeactivateFakeTriangles()`

```
void Mesh::DeactivateFakeTriangles()
```

Descrizione:
Questo metodo disattiva i triangoli e i lati associati ai punti fittizi presenti nella mesh. In particolare, si itera su tutti i lati presenti nella mesh e, per ogni lato attivo, si verifica se sia collegato ad almeno un punto fittizio. Se sì, tale lato viene disattivato, insieme a tutti i triangoli che contribuisce a definire.

Metodo `ExportEdges()`

```
void Mesh::ExportEdges(const vector<Edge>& edges, const string& outputFileName)
```

Input:

- `vector<Edge> edges`: vettore di oggetti `Edge` che rappresentano i segmenti della mesh.
- `string outputFileName`: nome del file in cui esportare i bordi.

Descrizione:
Questo metodo esporta in un file specificato i segmenti della mesh, contenuti nel vector `edges`, solo nel caso in cui il loro attributo `active` sia true.
Se il file specificato non esiste viene creato, altrimenti il file già esistente viene sovrascritto. Il formato del file di output è il seguente: `Id xOrigin yOrigin xEnd yEnd`. Infine, dopo aver riportato i dati richiesti, il metodo chiude il file.

Funzioni esterne

`Distance`

```
double Distance(const Point& p1, const Point& p2)
```

Input:

- `Point p1`: Il primo punto.
- `Point p2`: Il secondo punto.

Output:
La distanza euclidea tra `p1` e `p2`.

Descrizione:
Questa funzione calcola la distanza euclidea tra due punti nel piano.

`CalculateAngle`

```
double CalculateAngle(const Point& p1, const Point& p2, const Point& p3)
```

Input:

- `p1` (tipo: `Point`): Il primo punto.
- `p2` (tipo: `Point`): Il secondo punto.
- `p3` (tipo: `Point`): Il terzo punto.

Output:
L'angolo in radianti formato dai tre punti.

Descrizione:
La funzione `CalculateAngle` calcola l'angolo tra i punti `p1`, `p2` e `p3` nel piano utilizzando il teorema del coseno:
`acos((d2 * d2 + d3 * d3 - d1 * d1) / (2 * d2 * d3))`, dove `d1`, `d2` e `d3` sono le distanze tra i punti, calcolate con la funzione `Distance`.