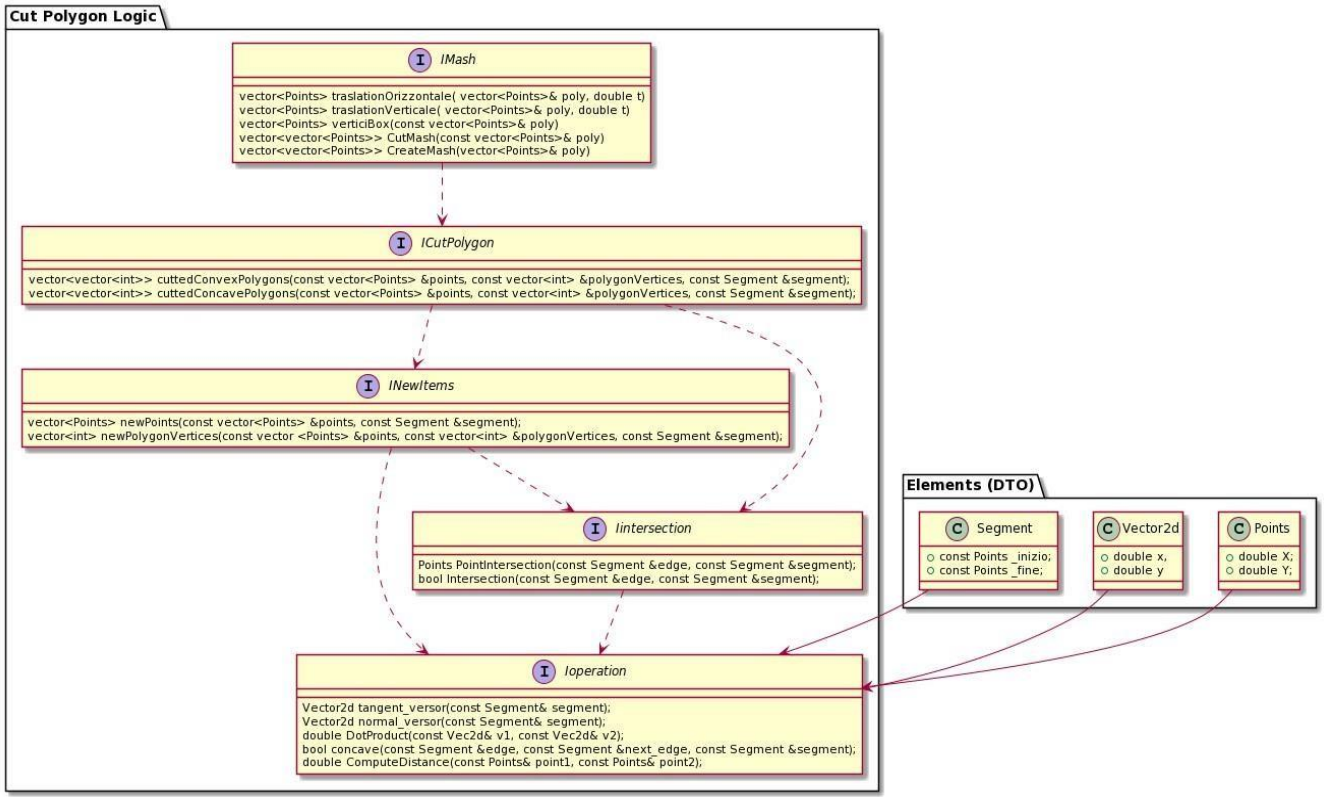
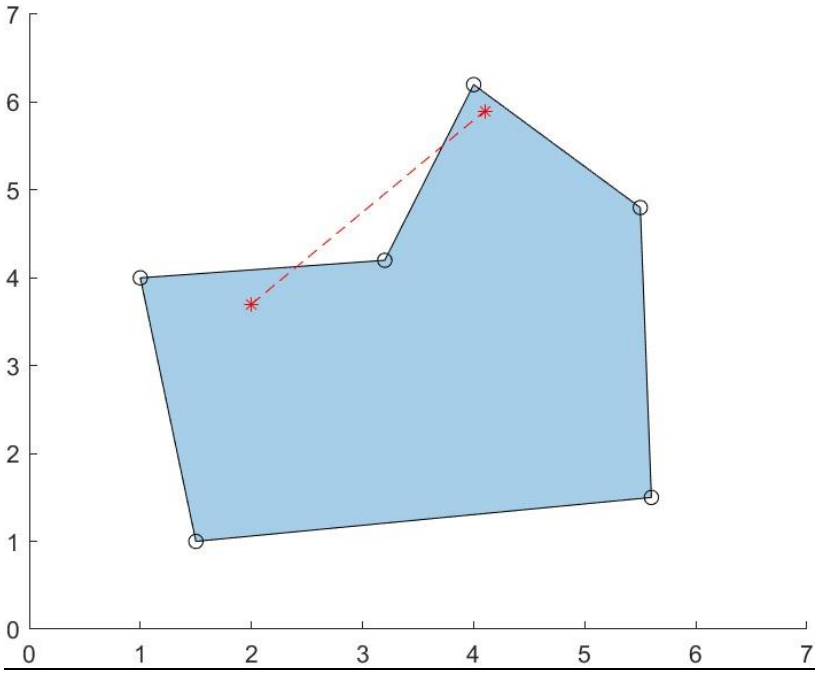
Presentazione Progetto PCS 2020/2021.

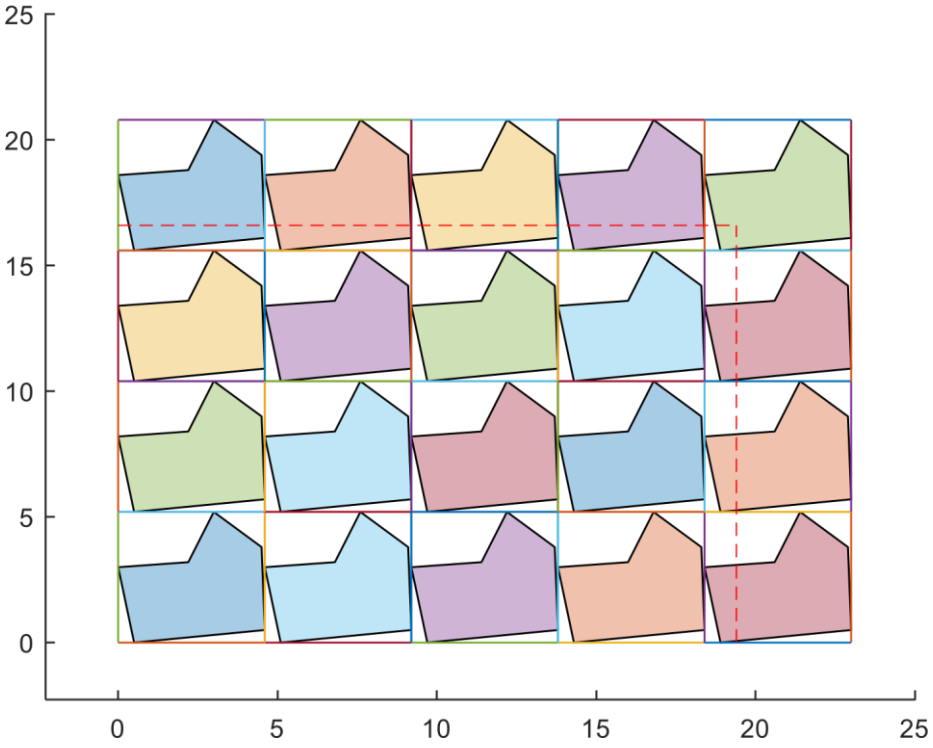
UML



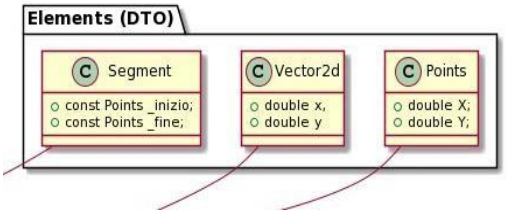
<u>Prima parte del progetto:</u> taglio di poligoni concavi e convessi.



<u>Seconda parte del progetto:</u> creazione di una mesh, dato un dominio rettangolare.



DTO (Data Transfer Objects)



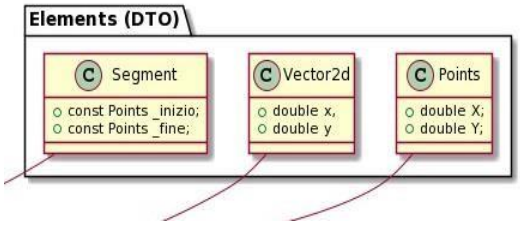
Implementazione in Python

```python
class Points:
    def __init__(self, x: float, y: float):
        self.X = x
        self.Y = y
```

```python
import src.points_class as PointsLibrary

class Segment:
    def __init__(self, inizio: PointsLibrary.Points, fine: PointsLibrary.Points):
        self.Inizio = inizio
        self.Fine = fine
```

```python
class Vector2d:
    def __init__(self, x: float, y: float):
        self.X = x
        self.Y = y
```

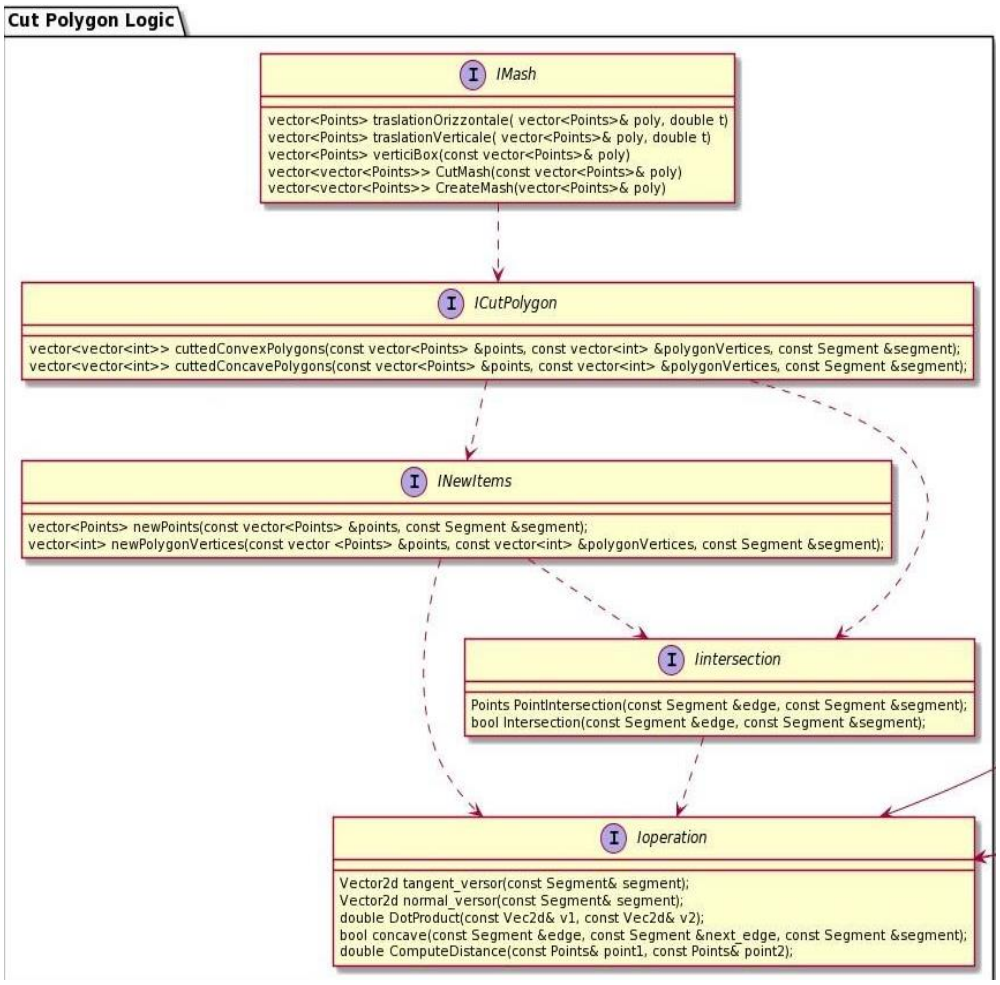
DTO (Data Transfer Objects)



Implementazione in C++

```
class Points{
public:
  double X;
  double Y;
  Points(const double& x,
             const double& y) { X = x, Y = y ;}
  Points();
};

class Segment {
public:
    const Points _inizio;
    const Points _fine;

    Segment(const Points& inizio,
            const Points& fine) : _inizio(inizio), _fine(fine) {};
    Segment();
};

class Vec2d {
    public:
      double _x;
      double _y;

      Vec2d(const double& x, const double& y) {_x = x; _y = y;};
      Vec2d();
};
```

Cut Polygon Logic



Cut Polygon Logic: **class Operation**.

Dichiarazione della classe in Geometry.hpp (C++).

```cpp
class Operation : public IOperation{

private:

    Points _point;
    Segment _segment;
    Vec2d _vector;

public:

    Operation(const Points& point, const Segment& segment, const Vec2d& vector)
        : _point(point), _segment(segment), _vector(vector) {}
    Operation();

    double ComputeDistance(const Points& point1, const Points& point2) const;
    double Area(const vector<Points>& points) const;
    double DotProduct(const Vec2d& v1, const Vec2d& v2) const;
    const Vec2d tangent_versor(const Segment& segment) const;
    const Vec2d normal_versor(const Segment& segment) const;
    bool concave(const Segment &edge, const Segment &next_edge, const Segment &segment) const;

};
```

Python: interfaccia Ioperation con la dichiarazione dei metodi virtuali, implementati nella classe Operation.

```python
class IOperation:

    # @abstractmethod
    def ComputeDistance(self, point1: PointsLibrary.Points, point2: PointsLibrary.Points) -> float:
        pass

    # @abstractmethod
    def TangentVersor(self, segment: SegmentLibrary.Segment) -> VectorLibrary.Vector2d:
        pass

    # @abstractmethod
    def NormalVersor(self, segment: SegmentLibrary.Segment) -> VectorLibrary.Vector2d:
        pass

    # @abstractmethod
    def Concave(self, edge: SegmentLibrary.Segment, next_edge: SegmentLibrary.Segment, segment: SegmentLibrary.Segment) -> bool:
        pass

    # @abstractmethod
    def DotProduct(self, v1: VectorLibrary.Vector2d, v2: VectorLibrary.Vector2d) -> float:
        pass
```

Di fondamentale importanza è il metodo che implementa il calcolo dell'area dei poligoni.

Formula dell'area di Gauss implementata nel metodo Area.

$$A = \frac{1}{2}|x_1y_2 + x_2y_3 + \ldots + x_{n-1}y_n + x_ny_1 - x_2y_1 - x_3y_2 - \ldots - x_ny_{n-1} - x_1y_n|,$$

Implementazione in C++

```
double Operation::Area(const vector<Points> &points) const
{
    double sum = 0;
    for (unsigned int i = 0; i < points.size(); i++){
        if (i != points.size() - 1)
            sum += points[i].X * points[i+1].Y - points[i + 1].X * points[i].Y; //formula di Gauss
        if (i == points.size() - 1)
            sum = sum + points[i].X * points[0].Y - points[0].X * points[i].Y;
    }

    double area = 0.5 * abs(sum);
    return area;
}
```

Cut Polygon Logic: **class Intersection.**

Dichiarazione in Python.

```python
class IIntersector:

    def PointIntersection(self, edge: SegmentLibrary.Segment, segment: SegmentLibrary.Segment) -> PointsLibrary.Points:
        pass

    def Intersection(self, edge: SegmentLibrary.Segment, segment: SegmentLibrary.Segment) -> bool:
        pass
```

Overview di una parte specifica comune ai due metodi.

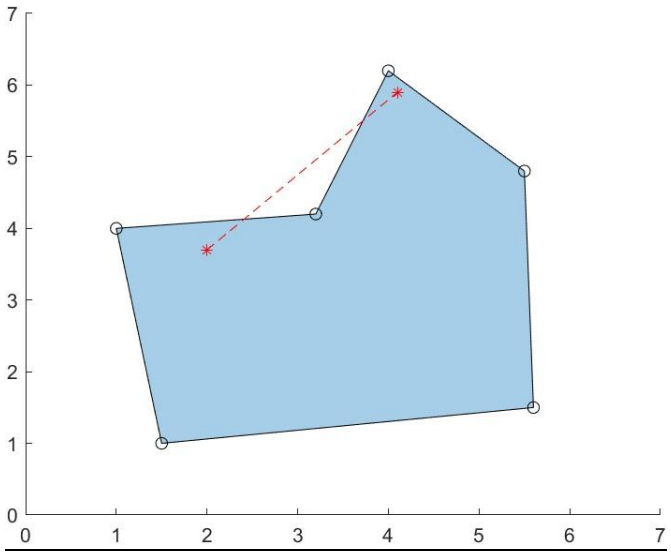
```
mtx_tang <<  ver2._x, -ver._x,
             ver2._y , -ver._y;
mtx_tang_inversa << -ver._y, ver._x,
                -ver2._y , ver2._x;
double det = mtx_tang.determinant();

double check = TOLLERANZA_PARALLELISMO * TOLLERANZA_PARALLELISMO
               * mtx_tang.col(0).squaredNorm() *  mtx_tang.col(1).squaredNorm();
               //utilizzando il prodotto esterno, ||t||^2 <= toll^2 * ||w||^2 * ||v||^2

if(det * det >= check)  ///ovvero, calcoliamo il punto di intersezione solo quando i due segmenti
                        ///non sono paralleli,ntroducendo tale controllo.

{
  Eigen::Vector2d b;                  //A*x = b => matx_tang_inversa * s = b
  b << segment._inizio.X - edge._inizio.X,
     segment._inizio.Y - edge._inizio.Y ;
  Eigen::Vector2d s = Eigen::Vector2d::Zero();
  s = mtx_tang_inversa * b;
  s /= det;        //soluzione sistema divisa per determinante mtx di partenza = coordinata curvilinea
```

Cut Polygon Logic: **class NewItems**

Dichiarazione in Geometry.hpp (C++)

```
class NewItems : public INewItems{
private:

    const IIntersector& _intersector;
    const IOperation& _operation;

public:

    NewItems(const IIntersector& intersector, const IOperation& operation)
        : _intersector(intersector), _operation(operation){}
    NewItems();

    const vector<Points> newPoints(const vector<Points> &points, const Segment &segment) const;
    const vector<int> newPolygonVertices(const vector <Points> &points, const vector<int> &polygonVertices,
                                         const Segment &segment) const;

};
```

NewPoints e newPolygonVertices: output.



Punti: {(1.5,1), (5.6,1.5), (5.5,4.8), (4,6.2), (3.2,4.2), (1,4), (4.2,6), (3.7,5.5), (2.4,4.1), (1.2,2.9), (4.1,5.9), (2,3.7)}

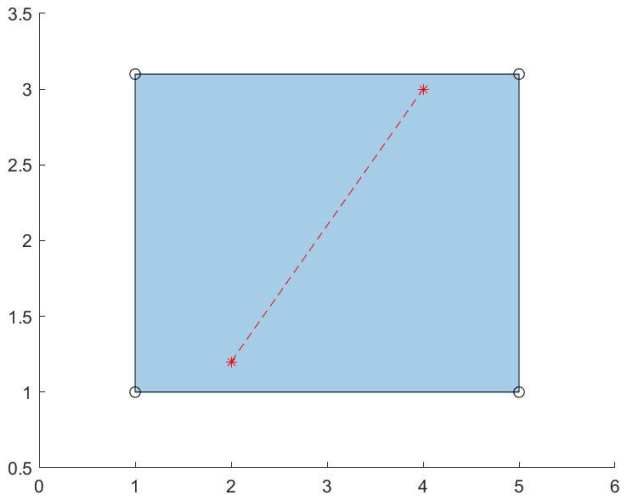
Vertici: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Cut Polygon Logic**: class CutPolygon**

Dichiarazione dell'interfaccia IcutPolygon in Python

```
class ICutPolygon:

    def CuttedConvexPolygons(self, points: [], polygon_vertices: [], segment: SegmentLibrary.Segment) -> [[]]:
        pass

    def CuttedConcavePolygons(self, points: [], polygon_vertices: [], segment: SegmentLibrary.Segment) -> [[]]:
        pass
```

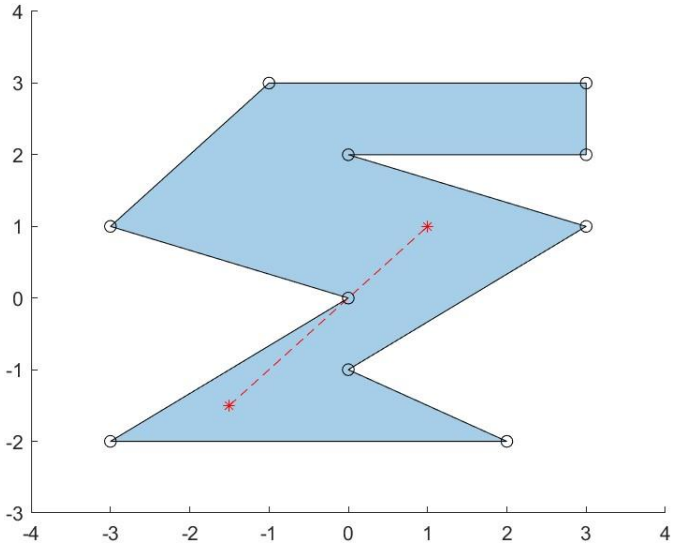
CuttedConvexPolygons: overview



Parte di codice riguardante la "creazione" del secondo poligono

```
# CREO SECONDO POLIGONO

# se intersezione non è un vertice
if new_polygonvertices[len(polygon_vertices)] >= len(polygon_vertices):
    polygon2.append(new_polygonvertices[len(polygon_vertices)])  # aggiungo prima intersezione

for i in range(lato_intersezione[0], lato_intersezione[1]):
    polygon2.append(new_polygonvertices[i + 1])  # aggiungo punto finale lato

polygon2.append(new_polygonvertices[len(polygon_vertices) + 1])  # aggiungo seconda intersezione
```

CuttedConcavePolygons: overview



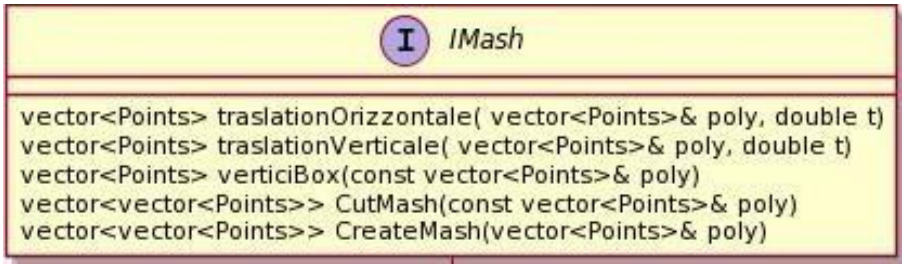
Parte del codice della funzione CuttedConcavePolygons.

```
if fineCiclo == 0:
    # Creo il poligono contenente lo zero.
    polygon.append(new_polygonVertices[len(points)])

    # numPunti: conto numeri punti intersezione tra i segmenti con intersezione 10 e segmento con intersezi
    for var in range(len(points), len(points) + 2):
        if new_points[var + 1].X > new_points[len(points)].X:  #
            numPunti = numPunti + 1

    if segment.Fine.Y > segment.Inizio.Y:
        if (new_points[len(new_points) - 2].X < new_points[len(points)].X and
                new_points[len(new_points) - 2].X > new_points[len(points) + numPunti + 1].X and
                new_points[len(new_points) - 2].Y > new_points[len(points) + numPunti + 1].Y and
                new_points[len(new_points) - 2].Y < new_points[len(points)].Y):
            polygon.append(new_polygonVertices[len(new_polygonVertices) - 2])

        if (new_points[len(new_points) - 1].X < new_points[len(points)].X and
                new_points[len(new_points) - 1].X > new_points[len(points) + numPunti + 1].X and
                new_points[len(new_points) - 1].Y > new_points[len(points) + 1 + numPunti].Y and
                new_points[len(new_points) - 1].Y < new_points[len(points)].Y):
            polygon.append(new_polygonVertices[len(new_polygonVertices) - 1])

    polygon.append(new_polygonVertices[len(points) + 1 + numPunti])
```

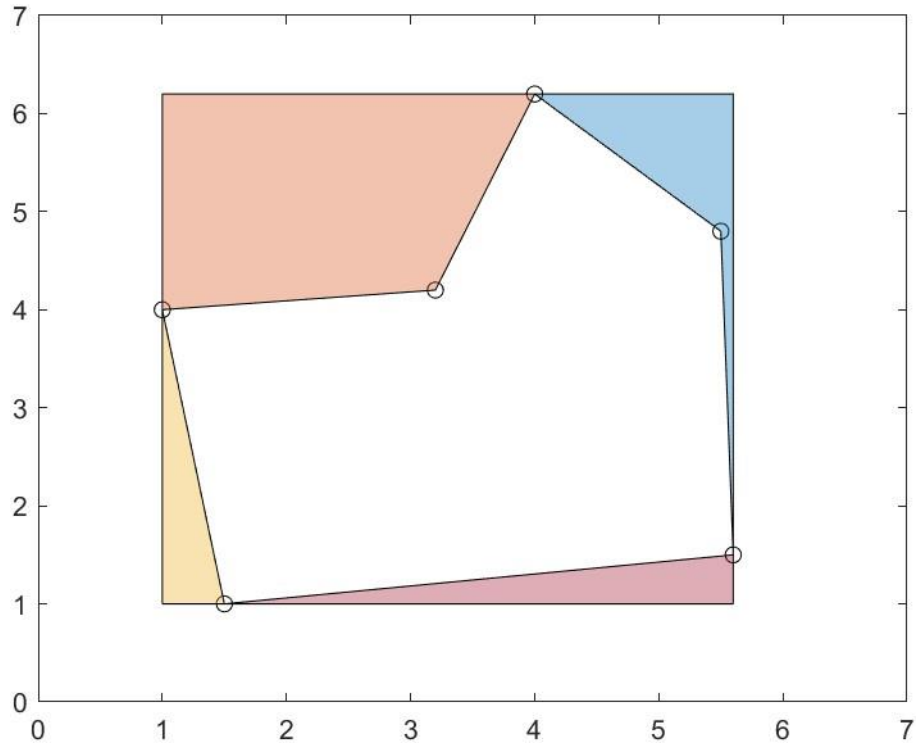
Output ottenuto nel caso di questo specifico poligono.

```
I punti, vertici e poligoni ottenuti dal secondo poligono concavo sono:
 Punti: {(2,-2), (0,-1), (3,1), (0,2), (3,2), (3,3), (-1,3), (-3,1), (0,0), (-3,-2),
(1.5,1.5), (2,2), (3,3), (0,0), (-2,-2), (1,1), (-1.5,-1.5)}
 Vertici: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 5, 8, 12, 13, 14}
 - {3, 11, 5, 6, 7, 8, 13, 10}
 - {11, 4, 5}
 - {8, 9, 12, 14}
 - {10, 13, 8, 14, 12, 0, 1, 2}
```

Seconda parte: classe Mash



CutMash: overview



Controllo sull'area totale del Reference Element

```
///test sull'area.

double areaReferenceElement = 0;

for (unsigned int i = 0; i < polygon_mash.size() - 1; i++)
{
    areaReferenceElement = _method.Area(polygon_mash[i]) + areaReferenceElement;
}

EXPECT_TRUE(abs(areaReferenceElement - areaBoundingBox) < TOLLERANZA_AREA*areaBoundingBox);
```

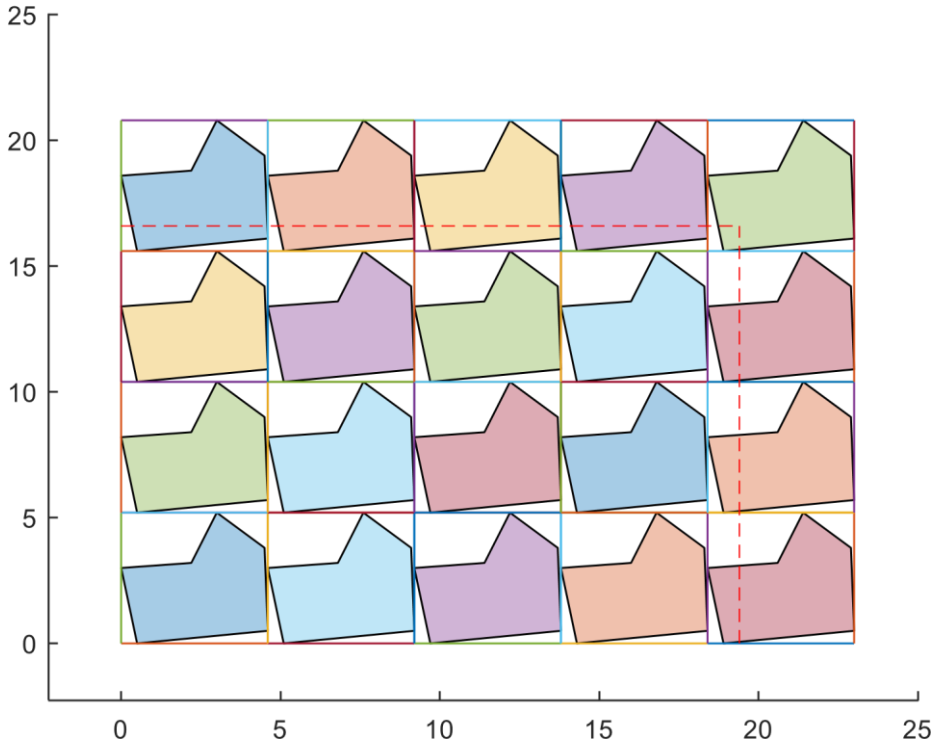
Parte del codice della funzione

```
vector<PolygonLibrary::Points> intersection;
vector<PolygonLibrary::Points> vertici_box;

vertici_box = _mash.verticiBox(poly);

for (unsigned int i = 0; i < poly.size(); i++){

    //vettore dei punti di intersezione tra box e poly
    if (poly[i].X == x_min || poly[i].X == x_max || poly[i].Y == y_min || poly[i].Y == y_max){
        intersection.push_back(poly[i]);
        intersection[i].X = poly[i].X;
        intersection[i].Y = poly[i].Y;

        //in vertici_intersection salviamo la numerazione intera dei vertici del poligono
        //che intersecano il bounding box.
        vertici_intersection.push_back(i);

    }
}

//scorro sulle intersezioni trovate
for (unsigned int j = 0; j < intersection.size(); j++){
```

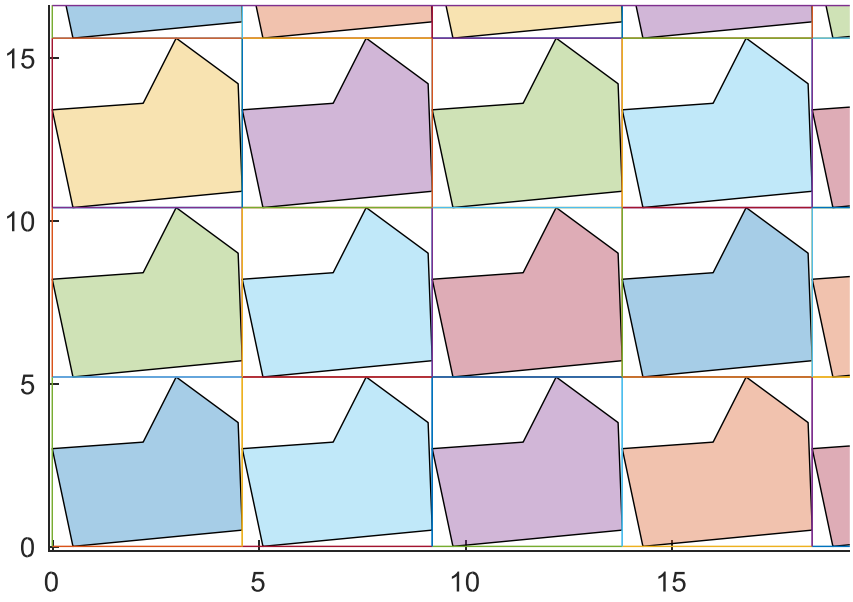
CreateMash: overview

Costruzione del dominio rettangolare e definizione di altre variabili.

```
vector<Points> box = _mash.verticiBox(poly); // creo la box del poligono.

demain.push_back(PolygonLibrary::Points(0.0,0.0));
demain.push_back(PolygonLibrary::Points(4*(box[1].X - box[0].X) + 1,0.0));
demain.push_back(PolygonLibrary::Points(4*(box[1].X - box[0].X) + 1, 3*(box[2].Y - box[1].Y) + 1));
demain.push_back(PolygonLibrary::Points(0.0, 3*(box[2].Y - box[1].Y) + 1));

//calcolo quante volte nella base
double base_demain = demain[1].X - demain[0].X;
double base_box = box[1].X - box[0].X;

ripetizioni_base = int(base_demain/base_box);
resto_base = base_demain - ripetizioni_base * base_box ;

//calcolo quante volte nell'altezza
altezza_demain = demain[2].Y - demain[1].Y;
altezza_box = box[2].Y - box[1].Y;

double areaEsattaMash = base_demain*altezza_demain;

ripetizioni_altezza = int(altezza_demain/altezza_box);
resto_altezza = altezza_demain - ripetizioni_altezza * altezza_box;
```



Mesh finale ottenuta



Controllo sull'area della mesh ottenuta

```
///Confrontiamo infine l'area trovata con quella totale del dominio.

if(abs(areaMash - areaEsattaMash) > TOLLERANZA_AREE*areaMash)
{
    std::cout<<areaMash - areaEsattaMash;
    throw runtime_error("L'area TOTALE della mash calcolata non e' corretta");
}
```