Progetto n. 2

Convessizzazione di un poliedro

Spiegazione dell'algoritmo

Funzione `TestPolyhedron` (in `geometryUnitTest`)

Abbiamo modificato questa funzione in modo che il programma crei una copia del poliedro iniziale e lo dia in pasto ciclicamente a `IsPolyhedronConvex` e a `ConvexPolyhedron` fin quando `IsPolyhedronConvex` non ritorna `true`. A questo punto si applica `IsPolyhedronConvex` a tutti i poliedri della lista `convexPolyhedra` per verificare che i poliedri ottenuti siano effettivamente tutti convessi

Funzione `IsPolyhedronConvex`

La funzione prende in input un poliedro, una tolleranza ed un vettore di puntatori ad `int` vuoto.
La funzione cicla sulle facce, prende tramite gli appositi getter il `PlaneNormal` ed il `PlaneTranslation`. Per ogni faccia si cicla quindi sui vertici, controllando che questi appartengano (tenendo conto della tolleranza) alla faccia considerata; ogni volta che questa condizione viene soddisfatta, un contatore "`uguali`" inizializzato a 0 viene incrementato. Se questa condizione non è soddisfatta si controlla se il punto si trovi da un lato o dall'altro rispetto al piano individuato dalla faccia in esame. Se il punto si trova da un lato considerato convenzionalmente positivo e una variabile "`occorrenze`" viene incrementata, altrimenti viene decrementata. Se si verifica la condizione "`NumberOfVertices == uguali + abs(occorrenze)`" allora vuol dire che tutti i punti sono stati trovati da uno stesso lato rispetto alla faccia e non ci sono problemi. In caso contrario, ad un flag inizializzato ad inizio funzione a `true` viene assegnato il valore `false`: il valore di questo flag sarà proprio quello ritornato dalla funzione. Inoltre, l'indice della faccia in cui viene rilevato il problema di concavità viene salvato all'interno del vettore di puntatori a `int` preso in input in modo da poter essere utilizzato al di fuori di questa funzione: verrà utilizzato, infatti, nella funzione `ConcaveToConvex` come indicazione sul dove effettuare i tagli.

Abbiamo lasciato nel codice alcune istruzioni commentate che possono risultare utili per la visualizzazione degli output e verificare il corretto funzionamento della funzione una volta scommentati.

Funzione `ConcaveToConvex`

La funzione prende in input un poliedro, la lista dei poliedri tagliati, una tolleranza e un vettore di puntatori a `int`.
Sfruttando il vettore di puntatori ad interi riempito nella funzione `IsPolyhedronConvex`, scorriamo solo le facce del poliedro rilevate come "problematiche" con due cicli `for`, in modo che ogni faccia venga comparata con tutte le altre. Per evitare che uno stesso confronto venga fatto due volte, abbiamo pensato a un sistema di eliminazione degli elementi del vettore di puntatori a interi dopo opportune verifiche, in modo che, ad esempio, avendo come facce problematiche quelle di indici 2, 4, 5, 7, avremo la seguente successione di controlli:
2 - 4
2 - 5
2 - 7
4 - 5
4 - 7
5 - 7
Per ogni coppia cerchiamo un segmento in comune e se lo troviamo consideriamo quel segmento come facente parte della faccia di taglio, ossia della nuova faccia che farà da separatore dei due nuovi poliedri. Cerchiamo quindi un vertice qualunque del poliedro con la condizione che non appartenga a nessuna delle due facce in esame e creiamo così la faccia triangolare definita dal suddetto vertice e dal segmento trovato precedentemente; ottenuta questa faccia, si interseca il piano che la contiene con il poliedro.
Nel pratico, si cercano i segmenti che intersecano il suddetto piano analizzando la posizione dei vertici di origine e fine del segmento in esame. In particolare assegniamo un indice -1 al vertice se si trova da un lato rispetto al piano oppure +1 se si trova dall'altro oppure 0 se esso appartiene al piano.
Fatto ciò, sapremo che se il prodotto degli indici dei due vertici di inizio e fine dello stesso segmento è minore o uguale a 0 allora c'è intersezione, eventualmente in uno dei due capi del segmento: ciò significa che questa intersezione andrà calcolata e sarà un vertice della faccia di taglio. Inoltre, se tale prodotto risulterà essere anche strettamente minore di 0, vorrà dire che l'intersezione avviene in un punto intermedio del segmento e andrà dunque tagliato. Otterremo quindi due nuovi segmenti che avranno uno inizio nell'origine del segmento di partenza e fine nell'intersezione appena calcolata e l'altro avrà inizio nell'intersezione e fine nell'altro estremo del segmento originale. Ora i due segmenti vengono aggiunti al poliedro mentre quello iniziale viene eliminato, quindi sovrascriviamo la lista dei vertici della faccia di taglio. Collegando tra di loro i vertici appena ottenuti, imponendo la condizione che appartengano alla medesima faccia del poliedro iniziale otteniamo i segmenti della faccia di taglio. Anche in questo caso modifichiamo la lista.
Abbiamo implementato i metodi "`DeleteSegments`" , "`DeleteVertices`" e "`RemoveSegment`" per non caricare eccessivamente il codice.

Successivamente dichiariamo due nuovi poliedri vuoti ed iniziamo a riempirli.
In maniera analoga a quanto fatto per la funzione `IsPolyhedronConvex` controlliamo sul poliedro concavo quali vertici sono da un lato e quali dall'altro rispetto alla faccia di taglio. Aggiungeremo quelli che stanno da un lato al primo e quelli dall'altro lato al secondo. Aggiungiamo inoltre i vertici appartenenti alla faccia di taglio ad entrambi. Procediamo poi con i segmenti: tra tutti i segmenti del poliedro concavo, assegniamo ad ognuno dei due poliedri quelli che collegano i vertici che lo costituiscono. Aggiungiamo inoltre i segmenti appartenenti alla faccia di taglio ad entrambi*.
Per l'aggiunta delle facce procediamo in maniera simile a quanto fatto per i segmenti. Sfruttiamo l'assioma per cui per tre punti passa un solo piano e in questo modo cerchiamo terne di vertici che appartengono sia al `convexPolyhedron` sia a quello di partenza e le aggiungiamo quindi al primo di questi.
All'operazione appena descritta facciamo precedere un controllo: abbiamo sfruttato il ciclo per l'aggiunta dei segmenti per verificare se i segmenti della faccia di taglio hanno un corrispettivo tra i segmenti già esistenti nel poliedro. Se questa condizione non è rispettata, significa che il segmento in questione giace su una faccia che andrà quindi tagliata lungo di esso. Prima dell'aggiunta delle facce verrà dunque creata una nuova faccia che sarà il risultato del taglio sopra citato e verrà aggiunta al nuovo poliedro). Inoltre imponiamo che nell'aggiunta "regolare" delle facce venga saltata la faccia che ha subito il taglio dal momento che è appena stata aggiunta e risulterebbe duplicata.
Procediamo con le medesime operazioni anche per il secondo poliedro ottenuto.

Infine chiamiamo sui due poliedri la funzione `IsPolyhedronConvex` per identificare quali dei due risultino convessi e quali no. Per ognuno, se il controllo fallisce (poliedro non convesso e quindi concavo), questo viene eliminato dalla lista dei `convexPolyhedra` e dato in input alla funzione `ConcaveToConvex` affinché venga tagliato ulteriormente.

N.B.: Notare che la funzione `ConcaveToConvex` viene chiamata in maniera ricorsiva. Nel caso dell'ultima chiamata di questa funzione entrambi i poliedri saranno convessi e quindi nessun elemento della lista sarà eliminato.

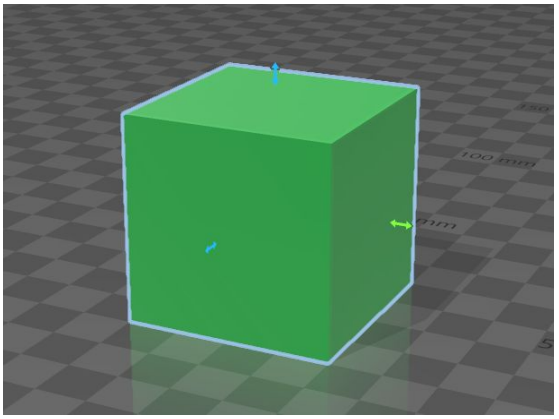
*Per verificare l'uguaglianza tra due punti nello spazio abbiamo ritenuto inadeguato l'utilizzo dell'operatore "=" in quanto non tiene conto della tolleranza. Abbiamo quindi implementato il metodo `IsEqual` che verifica se il punto dato in input si trova all'interno di una sfera di raggio tolleranza e centrata nel punto a cui è applicato il metodo.

Test eseguiti

Abbiamo predisposto all'interno dei file `"geometryUnitTest.cpp"` e `"geometryUnitTest.hpp"` le funzioni `"TestZero", "TestOne", "TestTwo", "TestThree", "TestFour"` che richiamano rispettivamente `"CreatePolyhedronTestZero", "CreatePolyhedronTestOne", "CreatePolyhedronTestTwo", "CreatePolyhedronTestThree", "CreatePolyhedronTestFour"` e ognuna esegue il test sul rispettivo poliedro creato. Li analizziamo singolarmente di seguito.

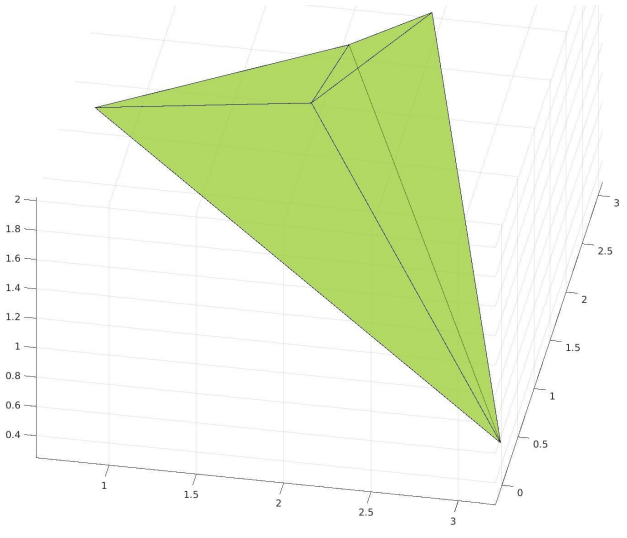
TestZero

Questo test è chiamato test 0 perché di fatto il programma non esegue alcun taglio.
Il poliedro è un semplice cubo e questo test è stato pensato unicamente per verificare che il programma riesca a riconoscere quando il poliedro è già convesso e non esegua dunque alcuna altra operazione inutile.

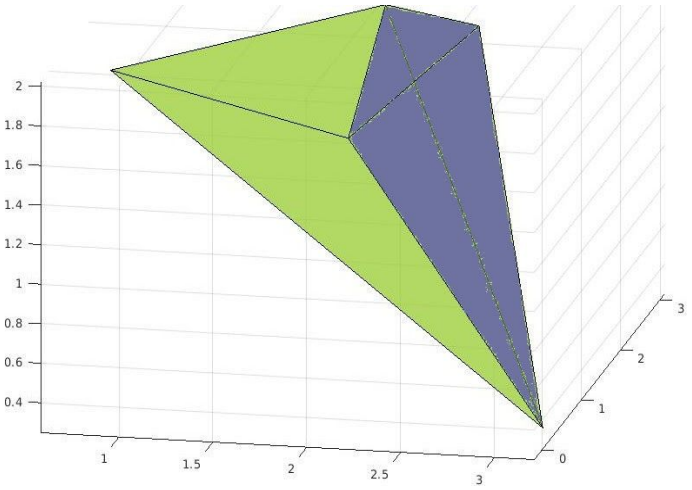


TestOne

Questo test lavora sul poliedro di default che ci è stato dato ad inizio progetto. E' il test più semplice: un poliedro con solo facce triangolari in cui è sufficiente effettuare un solo taglio, che avviene peraltro su una faccia costruibile utilizzando solo vertici e segmenti già esistenti.
Il programma individua le due facce superiori come facce "problematiche" e quindi il segmento in comune come punto di partenza per generare la faccia di taglio. Il programma cerca infatti un vertice che non appartiene a nessuna delle due facce problematiche. In questo caso l'unica possibilità è il vertice in basso. A questo punto la faccia di taglio viene costruita con i tre vertici ottenuti collegati tra di loro.
N.B.: per questo test è sufficiente questa faccia triangolare. La generalizzazione della faccia di taglio avviene comunque ma restituisce lo stesso risultato.

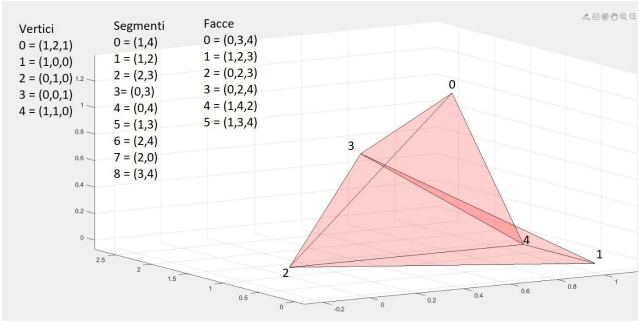


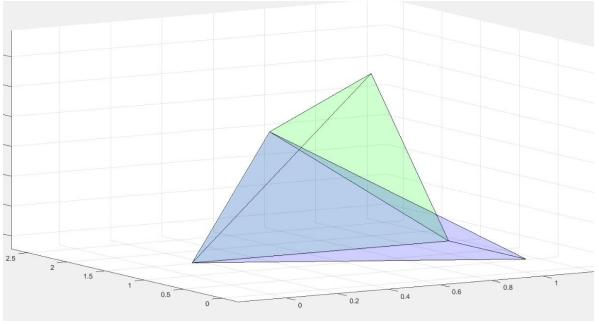
Arrivati a questo punto viene allocata memoria per due poliedri che vengono inseriti nella lista "`convexPolyhedra`". Segue una verifica di quali vertici, segmenti, facce devono appartenere rispettivamente ai due poliedri che stiamo creando e li aggiungiamo. In questa fase non sorgono particolari problemi data la semplicità della figura.
Infine viene richiamata la funzione "`IsPolyhedronConvex`" che verifica la convessità dei poliedri ottenuti e ritorna un successo.



TestTwo

Questo test lavora su un poliedro con caratteristiche del tutto analoghe a quelle del `TestOne`, quindi ha solo facce triangolari ed è sufficiente un solo taglio. Allora anche qui il taglio avviene su una faccia costruibile utilizzando solo vertici e segmenti già esistenti nel poliedro.
Questo test infatti ed è stato pensato per provare che il corretto funzionamento del test precedente non sia un caso fortuito.

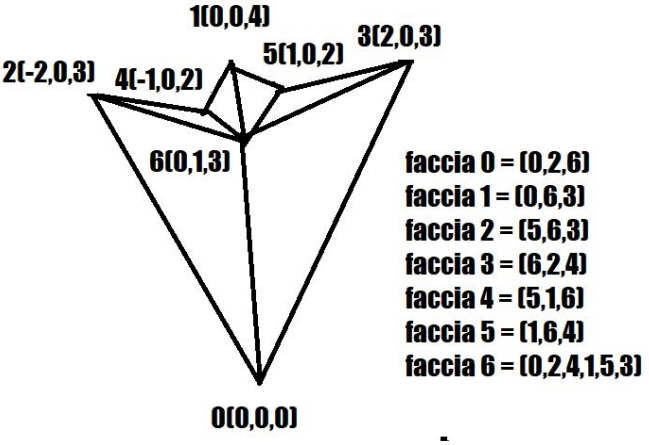




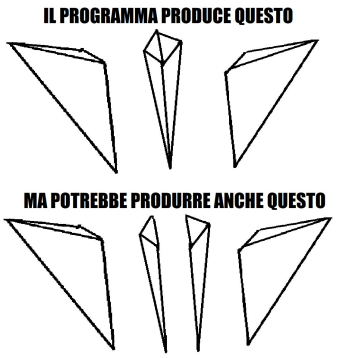
TestThree

Questo test lavora su un poliedro con tutte le facce triangolari tranne una, creata volutamente concava in modo da poterci concentrare sui problemi che essa può dare senza dover perdere tempo a capire ogni volta quale faccia non triangolare dia quali problemi.

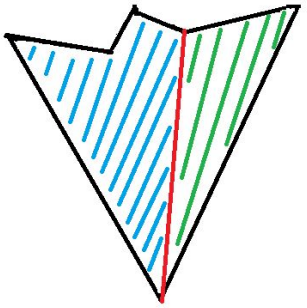
Segmenti
0-> (0,6)
1-> (0,2)
2-> (0,3)
3-> (2,6)
4-> (6,3)
5->(2,4)
6->(5,3)
7-> (4,6)
8-> (5,6)
9-> (1,6)
10-> (4,1)
11-> (5,1)



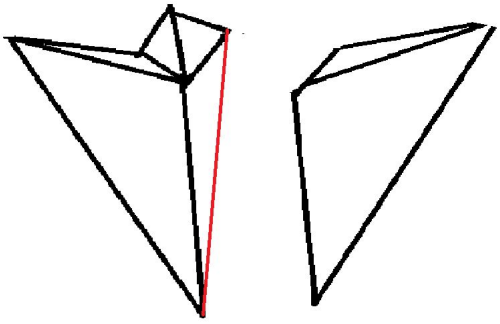
In questo test inoltre, a differenza dei precedenti, sono necessari e sufficienti due tagli, tuttavia una numerazione differente delle facce potrebbe portare il programma a farne tre. Questo ci ha posto di fronte ad un problema che, in seguito ad una consulenza, abbiamo ritenuto essere trascurabile, dal momento che l'algoritmo riesce sempre ad ottenere una soluzione al problema, anche se non sempre quella col minor numero di operazioni. Possiamo osservare che i tagli avvengono su facce triangolari costruibili utilizzando solo vertici già esistenti ma richiedono la costruzione di segmenti nuovi.



Queste differenze dai test precedenti risultano essere sostanziali tanto nell'esecuzione di più di un taglio quanto nel processo obbligatorio che va a tagliare una faccia. Questo implica la creazione di due nuove facce (risultato del taglio della faccia) da aggiungere ai due poliedri risultanti. Successivamente occorre accertarsi che la faccia tagliata nella sua forma originale non venga inserita. Per fare ciò cerchiamo i segmenti della faccia di taglio tra i segmenti del poliedro convesso e se uno (o più) di questi non viene trovato vuol dire che quel segmento taglia la faccia su cui giace. A questo punto quella faccia viene tagliata in due lungo il segmento trovato, assegnate le due parti ai due diversi poliedri e contrassegnata affinché non venga successivamente aggiunta assieme alle altre facce.



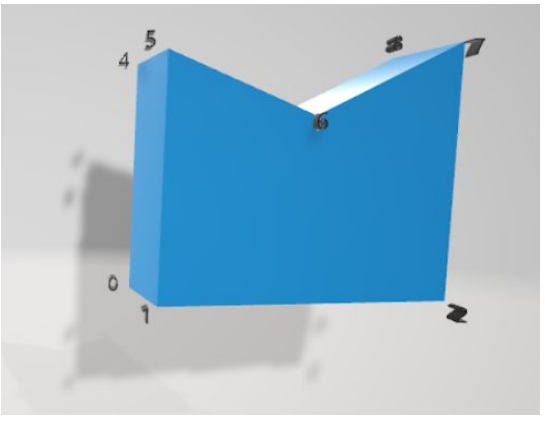
Per quanto riguarda il taglio multiplo, abbiamo deciso di utilizzare la funzione `ConcaveToConvex` in maniera ricorsiva. Anche qui infatti abbiamo inserito una differenza rispetto ai casi precedenti: solo uno dei due poliedri passa il test di convessità, l'altro viene dato nuovamente in pasto alla funzione affinché venga tagliato ulteriormente, dopo averlo eliminato dalla lista dei poliedri convessi dal momento che è risultato concavo. All'interno di `ConcaveToConvex` vengono memorizzati quindi due nuovi poliedri e nel momento in cui entrambi passeranno il test di convessità, il programma aggiungerà quei due.



Per sicurezza abbiamo deciso di eseguire a posteriori, dopo che il programma ha finito di fare tutti i tagli, un controllo che tutti i poliedri ottenuti siano effettivamente convessi.

TestFour

Questo test lavora su un poliedro con sole facce non triangolari in cui è sufficiente un solo taglio. L'importante differenza che porta questo test rispetto ai precedenti è la necessità di generalizzare la faccia di taglio. Non basta più una faccia di taglio triangolare ma ne serve una rettangolare. Decidiamo quindi di calcolare ugualmente la faccia di taglio triangolare (insufficiente) e sfruttarla per calcolare l'equazione del piano che la contiene. A questo punto (concettualmente parlando) intersechiamo l'intero poliedro con il piano trovato identificando la vera faccia di taglio.



Esecuzione

```
>> Polyhedron tests
>> ------------------------------------------------------------------------

>> 1 - TestZero: IsPolyhedronConvex SUCCESS

>> ------------------------------------------------------------------------
```

Il poliedro è convesso, per cui tale test ritorna successo e termina.

```
>> 2 - TestOne: IsPolyhedronConvex FAILED
     - Il poliedro Þ concavo, procedo con il taglio
>> 3 - TestOne: ConcaveToConvexPolyhedron SUCCESS
     - Controllo che tutti i poliedri ottenuti siano effettivamente convessi
>> 4 - TestOne: IsPolyhedronConvex 1 SUCCESS
>> 5 - TestOne: IsPolyhedronConvex 2 SUCCESS

>> ------------------------------------------------------------------------
```

Il poliedro è concavo, per cui tale test ritorna fallimento, segnale che indica la necessità di eseguire un taglio. Il taglio viene eseguito senza intoppi e procede quindi con il controllo del risultato. I poliedri ottenuti sono entrambi convessi, per cui entrambi ritornano successo e termina.

```
>> 6 - TestTwo: IsPolyhedronConvex FAILED
     - Il poliedro Þ concavo, procedo con il taglio
>> 7 - TestTwo: ConcaveToConvexPolyhedron SUCCESS
     - Controllo che tutti i poliedri ottenuti siano effettivamente convessi
>> 8 - TestTwo: IsPolyhedronConvex 1 SUCCESS
>> 9 - TestTwo: IsPolyhedronConvex 2 SUCCESS

>> ------------------------------------------------------------------------
```

Il poliedro è concavo, per cui tale test ritorna fallimento, segnale che indica la necessità di eseguire un taglio. Il taglio viene eseguito senza intoppi e procede quindi con il controllo del risultato. I poliedri ottenuti sono entrambi convessi, per cui entrambi ritornano successo e termina.

```
>> 10 - TestThree: IsPolyhedronConvex FAILED
      - Il poliedro Þ concavo, procedo con il taglio
>> 11 - TestThree: ConcaveToConvexPolyhedron SUCCESS
      - Controllo che tutti i poliedri ottenuti siano effettivamente convessi
>> 12 - TestThree: IsPolyhedronConvex 1 SUCCESS
>> 13 - TestThree: IsPolyhedronConvex 2 SUCCESS
>> 14 - TestThree: IsPolyhedronConvex 3 SUCCESS

>> ----------------------------------------------------------------------
```

Il poliedro è concavo, per cui tale test ritorna fallimento, segnale che indica la necessità di eseguire un taglio. Il taglio viene eseguito senza intoppi e procede quindi con il controllo del risultato. I poliedri ottenuti sono tutti e tre convessi, per cui ritornano successo e termina.

```
>> 15 - TestFour: IsPolyhedronConvex FAILED
      - Il poliedro Þ concavo, procedo con il taglio
>> 16 - TestFour: ConcaveToConvexPolyhedron SUCCESS
      - Controllo che tutti i poliedri ottenuti siano effettivamente convessi
>> 17 - TestFour: IsPolyhedronConvex 1 SUCCESS
>> 18 - TestFour: IsPolyhedronConvex 2 SUCCESS

>> ----------------------------------------------------------------------
```

Il poliedro è concavo, per cui tale test ritorna fallimento, segnale che indica la necessità di eseguire un taglio. Il taglio viene eseguito senza intoppi e procede quindi con il controllo del risultato. I poliedri ottenuti sono entrambi convessi, per cui entrambi ritornano successo e termina.

```
>> **********************************************************************
>> Total Tests done: 18, Number Success Tests: 14, Number Error Tests: 4
>> **********************************************************************
```

Vengono eseguiti in totale tra i 5 poliedri 18 test di cui 4 ritornano un fallimento, evidenziando come 4 dei 5 poliedri siano concavi.

Analisi di complessità

Riteniamo costruttivo analizzare la complessità computazionale delle operazioni principali che esegue il programma, ovvero di quelle che avvengono per la maggior parte dei poliedri che si analizzano. Parliamo quindi del controllo che un poliedro sia convesso e della costruzione dei nuovi poliedri quando risulta concavo, tralasciando invece operazioni singole e controlli. Trascureremo anche alcuni costrutti `if` sia nella condizione sia nel contenuto poiché poco rilevanti per questo scopo.
In particolare vogliamo analizzare le operazioni di:

- ➢ verifica della convessità del poliedro;
- ➢ costruzione della faccia di taglio;
- ➢ assegnazione dei vertici;
- ➢ assegnazione dei segmenti;
- ➢ assegnazione delle facce;

N.B. gli intervalli di numeri che riporteremo accanto al titolo di ogni sezione indicano le righe di codice contenute nel file `ConcaveToConvex.cpp`

Verifica della convessità del poliedro (rr. 71-104)

Questa è la prima operazione che viene eseguita. La complessità finale dipende solo dalle dimensioni del poliedro che viene preso in input.

Utilizzeremo le seguenti notazioni:
F= n di facce;
V= n di vertici;
Q= costo di verifica se è convesso;
C= costo dell'operazione di controllo se maggiore/uguale/minore;

```
for {sulle facce
	for {sui vertici
		if {uguali
		}
		else{
			if {maggiore
			}
			else{minore
			}
		}
	}
	if{controllo se è convesso
	}
}
```

Costo comp.: F*(V +V*C^2+Q) asintoticamente parliamo di una complessità O(F*V)

Costruzione della faccia di taglio (rr. 152-308)

Per quest'analisi, dividiamo il processo di costruzione della faccia di taglio in due step: la creazione della faccia triangolare, e in seguito la sua generalizzazione che parte proprio dall'utilizzo dell'equazione del suo piano per la ricerca di nuovi vertici e segmenti da aggiungere.

Creazione della faccia triangolare (rr. 152-171)

Utilizzeremo le seguenti notazioni:
V= n. vertici poliedro
V1= n. vertici faccia problemi 1
V2= n. vertici faccia problemi 2
A= operazione di assegnazione

```
for{vertici poliedro
	for{vertici faccia problemi 1
		if {
			A
		}
	}
	for{vertici faccia problemi 2
		if {
			A
		}
	}
}
//Seguono operazioni di costo costante
```

Costo comp.: V + 2V(V1*A + V2*A) +C asintoticamente O(V*(V1+V2))

Generalizzazione della faccia di taglio (rr. 195-308)

Qui separiamo per semplicità di visualizzazione il calcolo dei vertici e dei segmenti.

Vertici (rr. 195-262)
Utilizzeremo le seguenti notazioni:

A = operazione di assegnazione. Nelle righe da 197 a 226, data la semplicità delle operazioni e soprattutto la costanza del costo delle stesse, possiamo semplificare la struttura di quella parte di codice raggruppando tutto sotto un'unica operazione di assegnazione. In base alla sua complessità asintotica la tratteremo come un O(1)
S = n. di segmenti del poliedro
Vf = n. di vertici della f di taglio
T = operazione taglio segmenti

```
for {segmenti
	A
	if {segmento intersecato
		for{vertici della f di taglio
			if{già aggiunto
			}
		}
		if{non aggiunto
			aggiungi
		}
		if{segmento intersecato in senso proprio
			taglialo
		}
	}
}
```

Costo comp.: S + S(A+Vf+T) Asintoticamente O(S*Vf)

Segmenti (rr. 268-308)
Utilizzeremo le seguenti notazioni:
S = n. di segmenti del poliedro
Vf = n. di vertici della faccia di taglio
F = n. di facce del poliedro

```
for{facce
	for{vertici della f di taglio
		if{appartiene alla f di taglio
			for{vertici della f di taglio	//sarebbe stato meglio farli in sequenza
				if{appartiene alla f di taglio
					creo il segmento
					for{segmenti della f di taglio
						if{
						}
				}
				if{non aggiunto
					aggiungo il segmento
				}
			}
		}
	}
}
```

Costo comp.: F + F(Vf+Vf(Vf+Vf*S)) asintoticamente O(F*(Vf^2)*S)
Se avessimo posto i cicli `for` per la selezione dei vertici in sequenza avremmo avuto una complessità asintotica O(F*Vf*S)

In conclusione, se chiamiamo CT il costo della costruzione della faccia triangolare e GF la generalizzazione della faccia di taglio, composta dalle operazioni AV (aggiunta vertici) e AS (aggiunta segmenti) avremo:
CT + GF = CT + (AV + AS) = V + 2V(V+V*A) + (S+S(A+Vf+T) + F + F(Vf+Vf(Vf+Vf*S)))
asintoticamente O(V^2) + O(S*Vf) + O(F*(Vf^2)*S)

Assegnazione dei vertici (rr. 331-350)

L'assegnazione dei vertici viene eseguita in maniera molto semplice scorrendo i vertici del poliedro iniziale e selezionando quelli che rispettano una certa condizione di appartenenza. Questo garantisce anche un costo molto basso.

Utilizzeremo le seguenti notazioni:
V=vertici
A=operazione di assegnazione

```
for{vertici
	if{uguale
	A
	A
	}
	else{
		if{maggiore
		A
		}
		else{minore
		A
		}
	}
}
```

Costo comp.: V + V*A asintoticamente O(V)

Assegnazione dei segmenti (rr. 357-383)

L'operazione di assegnazione dei segmenti qui riportata viene eseguita due volte dal programma, una per poliedro tagliato.
Anche in questo caso, analizzando la complessità, ci siamo resi conto che alcuni cicli `for` che utilizzati in questa parte di codice potevano essere riscritti in sequenza anziché annidati per ottenere un risparmio a livello di costo computazionale.

Utilizzeremo le seguenti notazioni:

Sf=n segmenti f di taglio
S=n di segmenti
V=n di vertici
A=operazione di assegnazione

```
for{segmenti f di taglio
       A
}
for{segmenti
       for{vertici
              for{vertici                          //poteva essere fatto in sequenza
              }
              if{
                     for{segmenti f di taglio
                            if{
                            }
                     }
                     if{
                            A
                     }
              }
       }
}
```

Costo comp.: 2*((Sf + Sf*A) + S + S(V+V(V+V(Sf+Sf +A)))), asintoticamente O(Sf)+O(S*(V^2)*Sf)
Se avessimo posto i cicli `for` per la selezione dei vertici in sequenza avremmo avuto una complessità asintotica O(Sf)+O(S*V*Sf)

Assegnazione delle facce (rr. 423-494)

L'operazione di assegnazione delle facce qui riportata viene eseguita due volte dal programma, una per poliedro tagliato.
Dividiamo l'operazione in due step: la creazione e assegnazione di eventuali facce tagliate (questo viene eseguito raramente o comunque con pochi cicli del `for` più esterno), e l'assegnazione delle facce che esistono nel poliedro di partenza e sono presenti anche in quello tagliato.

Creazione eventuali facce tagliate (rr. 423-466)
Nel caso si presenti la necessità di tagliare una faccia del poliedro, viene effettuato questo procedimento e la faccia tagliata viene aggiunta al nuovo poliedro. Alla fine di queste operazioni viene aggiunta anche la faccia di taglio, operazione qui riportata per la sua rilevanza a livello logico più che computazionale.
Utilizzeremo le seguenti notazioni:

C = Creazione di una nuova faccia
Ac = operazione di assegnazione alla nuova faccia
A = operazione di assegnazione della faccia
Sf= n. segmenti faccia di taglio
Vc = n. vertici faccia corrente
Sc = n. segmenti faccia di taglio
Vp = n. vertici poliedro nuovo
Sp = n. segmenti poliedro nuovo
F = n. facce poliedro

```
for{segmenti f di taglio
	if{
		C
		for{facce poliedro
			for{vertici faccia corrente poliedro
				if{
					for{vertici faccia corrente poliedro
						if{
							for{vertici faccia corrente poliedro
								for{vertici poliedro nuovo
									if{
										Ac
									}
								}
							}
							Ac
							for{segm faccia corrente
								for{segm nuovo poliedro
									if{
										Ac
									}
								}
							}
						}
					}
				}
			}
		}
		A
	}
}
A
```

Costo comp.: Sf+Sf(C+F+F(Vc+Vc(Vc+Vc((Vc+Vc(Vp+Vp*A)*Ac+Sc+Sc(Sp+Sp*Ac))))*A)+A
asintoticamente O(Sf*F*Vc^3(Vp+Sc*Sp))
Se avessimo posto i cicli `for` per la selezione dei vertici della faccia corrente in sequenza avremmo avuto una complessità asintotica O(Sf*F*Vc(Vp+Sc*Sp))

Assegnazione delle facce (rr. 468-494)
Ora vengono effettuati diversi cicli annidati che servono ad identificare quali facce del poliedro originale devono essere aggiunte al nuovo poliedro, cercandole con un confronto diretto tra i vertici dei due poliedri in questione. Anche in questo caso ci sarebbe stato un risparmio a livello di costo con l'utilizzo dei cicli posti in sequenza.

Utilizzeremo le seguenti notazioni:

A=operazione di assegnazione della faccia
Vc=n vertici faccia corrente
Vp=n vertici poliedro nuovo
F= n facce poliedro

```
for{facce del poliedro
	for{segmenti faccia di taglio
		if{
		}
	}
	if{
		for{vertici nuovo poliedro		//poteva essere fatto in sequenza
			for{vertici nuovo poliedro
				for{vertici nuovo poliedro
					for{vertici faccia corrente
						if{i vertici sono uguali
							if{
								A
							}
						}
					}
				}
			}
		}
	}
}
```

Costo comp.: F+F(F+Vp+Vp(Vp+Vp(Vp+Vp((Vc+Vc*A)))), asintoticamente O(F*(Vp^3)*Vc) se avessimo posto i cicli `for` per la selezione dei vertici del poliedro in sequenza avremmo avuto una complessità asintotica O(F*Vp*Vc)

In conclusione, indicati con CF il costo della costruzione delle facce tagliate e con AF il costo dell'assegnazione delle facce al nuovo poliedro, avremo:
CF +AF = Sf +Sf(C+F+F(Vc+Vc(Vc+Vc((Vc+Vc(Vp+Vp*A)*Ac+Sc+Sc(Sp+Sp*Ac))))*A)+A+
+F+F(F+Vp+Vp(Vp+Vp(Vp+Vp((Vc+Vc*A))))
asintoticamente O(Sf*F*Vc^3(Vp+Sc*Sp)+F*(Vp^3)*Vc)

N.B.: durante la trattazione alcuni costi computazionali scritti nella loro forma asintotica sono volutamente riportati con delle operazioni di addizione non svolte al loro interno, in quanto queste potrebbero comportare delle variazioni non trascurabili a livello di complessità a seconda del poliedro con cui il programma sta lavorando.