Robespierre: a Slicing Utility

Matteo Formenti
Unimib

June 19, 2020

Abstract

Analisi del processo di slicing di modelli 3D per utlilizzo con macchine di additive manufacturing

Introduzione

Il processo di realizzazione di oggetti fisici tramite macchine di additive manufacturing prevede uno step di trasformazione dei modelli 3D in dati interpretabili dalla macchina. Le macchine in questione lavorano solitamente con un concetto di layer, creando quindi l'oggetto in vari strati. In particolare le stampanti 3D *FDM* creano oggeti depositando linee di materiale plastico seguendo una traiettoria precisa, la creazione di questi path è un processo estreamente complesso, che prende in considerazione le capacità mecchaniche/fisiche della macchina, le proprietà meccaniche del pezzo da creare come peso o resistenza, il materiale di stampa e altri svariati parametri che non verranno considerati in questa relazione. L'obiettivo di questa analisi riguarda il primo passo di questo processo: convertire il modello solido in una serie di *strati* da cui verrà generato il `GCODE` interpretabile dalla macchina.

1 Formato STL

Il formato STL (STereoLitography) descrive unicamente la geometria del modello, approssimando le superfici solide in triangoli. I file STL non descrivono altre proprietà dei modelli come texture, materiali o altri metadata, in quanto lo scopo princiaple è la rappresentazione della forma per poter poi realizzare fisicamente l'oggetto, è infatti inventato da 3D Systems per essere utilizzato in un programma CAM. Esistono due tipi di file STL: binari e ASCII. I primi sono ampiamente più uitilizzati in quanto più compatti, ma per scopi didattici ho scelto di utilizzare lo standard ASCII che permette di comprendere il file direttamente leggendolo. La forma di un file STL ASCII è la seguente

```
1 solid Mesh
2   facet normal 0.000000 1.000000 0.000000
3     outer loop
4       vertex 5.000000 5.000000 0.000000
5       vertex -5.000000 5.000000 0.000000
6       vertex -5.000000 5.000000 10.000000
7     endloop
8   endfacet
9 endsolid Mesh
```

Listing 1: Esempio di file STL contenente un unico trinagolo

La seconda riga del file mostra la direzione in cui è rivolto il triangolo, mentre le righe 4, 5 e 6 indicano le coordinate sugli assi x, y e z dei tre vertici del triangolo.

2 Creazione del modello

Esistono una miriade di software di modellazione 3D, per vedere le differenze tra questi ho scelto due programmi disponibili su Ubuntu: Blender e Freecad. Sebbene sia possibile creare gli stessi modelli con entrambi i software, questi hanno una differenza fondamentale: Freecad è un modellatore parametrico, mentre Blender è un programma di modellazione diretta. Storicamente tutti i software CAD seguono un approccio parametrico, ovvero che il modello viene creato aggiungendo feature 2D e poi rendendole 3D, questo permette di modificare una quota all'inizio del progetto senza dover ricreare tutti i passi successivi, di contro questi devono essere correttamente vincolati in modo da dare indicazioni precise al software. Al contrario blender segue un approccio what-you-see-is-what-you-get, ovvero che ogni faccia, vertice o lato del modello è modificabile a prescindere dal momento in cui viene creata[1] La mia esperienza è limitata a software parametrici, ma per capire veramente le differenze ho realizzato lo stesso modello con entrambi gli strumenti. La realizzazione in freecad è stata la più semplice, il modello consiste di un parallelepipedo con due cut, uno passante e uno di 4mm, quindi ho applicato un fillet su due lati per ottenere l'effetto stondato. Per quanto riguarda blender mi sono trovato ad usarlo come se fosse un programma parametrico, ovvero non ho sfruttato gli strumenti di modellazione principali ma ho utilizzato gli operatori binari di differenza e unione per tagliare i componenti. Il problema che mi ha portato a seguire quasto approccio al posto di quello più "classico" di blender è che non è facile gestire le quote dei vari loop, quindi risulta difficile avere una precisione sufficiente nelle varie features.

[1] https://www.engineering.com/DesignSoftware/DesignSoftwareArticles/ArticleID/16587/Whats-the-Difference-Between-Parametric-and-Direct-Modeling.aspx

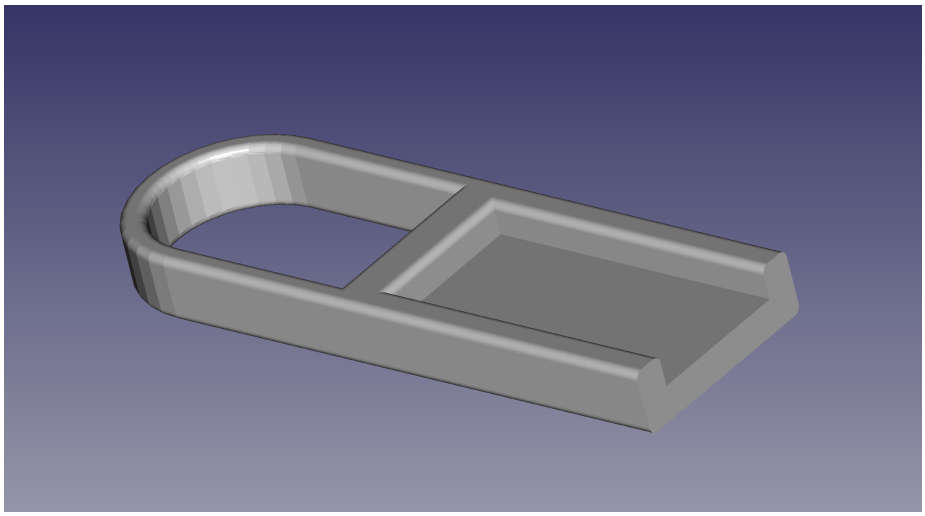

Figure 1: Render da Freecad del modello realizzato

3 Slicer

Lo slicer è realizzato in Java consiste di un modulo di importazione e parsing, un modulo di slicing e un modulo di esportazione sotto forma di immagine.

Importazione e parsing

La prima fase riguarda l'importazione e parsing del file STL, questa fase è stata progettata per organizzare il file ricevuto in oggetti Java, in modo da avere a disposizione una serie di strumenti "geometrici" all'interno della classe. In particolare il parser legge il file STL riga per riga, crea degli oggetti di tipo `Point` per ogni vertex del file e raggruppa i tre vertex del facet in un oggetto `Triangle`. L'oggetto triangle in fase di costruzione calcola il bounding box dei tre punti, quindi questi dati vengono utilizzati per calcolare il bounding box del modello completo. Tramite il bounding box vengono calcolate le dimensioni dell'immagine di output e il numero di layer del modello.

3.1 Tipi di triangoli

Per effettuare lo slicing di un modello, il processo consiste nel far scorrere un piano parallelo al piano XY (quindi ortogonale all'asse Z) e calcolare tutte le intersezioni tra il piano di taglio e i triangoli del modello. Ci sono 6 possibili casi: Partendo da sinistra, i casi sono

1. Due vertici appartengono all'asse di taglio
2. L'asse di taglio passa attraverso un vertice e un punto di un segmento

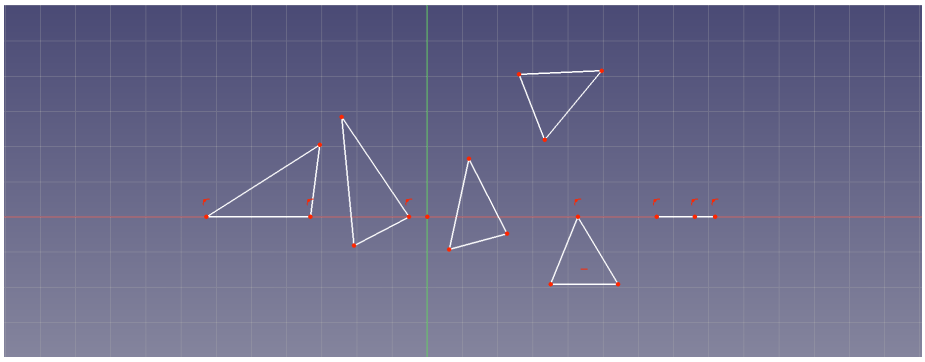

Figure 2: Sei tipi di intersezione

3. L'asse di taglio passa attraveso due punti su due segmenti differenti
4. L'asse di taglio non interseca il triangolo
5. Solo un vertice tocca l'asse di taglio
6. Il triangolo è interamente contenuto sull'asse di taglio

I casi in realtà sono riconducibili a due casi base, innanzitutto il caso 4 viene escluso in quanto l'algoritmo ignora i triangoli che non intersecano l'asse nel punto z_i, quindi se il vettore normale del triangolo corrisponde al vettore normale del piano (caso 6), tutti i segmenti del triangolo vengono inseriti nel layer. Rimangono i casi da 1, 2, 3 e 5, che però possono essere racchiusi in un unica formula. Infatti basta calcolare la formula della retta passante per ogni lato del triangolo, calcolare il punto di intersezione con il piano di taglio e infine verificare che questo sia all'interno del segmento delimitato dai due vertici del triangolo che generano tale retta. Ovviamente è da tenere in considerazione che i lati paralleli all'asse di taglio non avranno soluzione.

3.2 Calcolo dei layer

Avendo formulato le condizioni di intersezione e le formule per trovare i punti di intersezione, è sufficiente iterare un ciclo che "muove" l'asse di taglio e memorizza per ogni valore delle z, i segmenti che fanno parte di quel layer. Sarà possibile però che un trinagolo risulti intersecare il piano in più di 2 punti, questo succede perchè più edge hanno un intersezione con il piano (esempio caso 2), ma che in realtà risulta un solo punto distinto, possiamo quindi affermare che ogni triangolo può intersecare il piano di taglio in solamente uno o due punti.

```
1 public Point intersectionPoint(double z) {
2     if (p1.getZ() == p2.getZ())
3         return null;
```

```
4       Point p = new Point(
5           p2.getX() - p1.getX(),
6           p2.getY() - p1.getY(),
7           p2.getZ() - p1.getZ());
8       double t = (z - this.p1.getZ()) / p.getZ();
9       double int_x = this.p1.getX() + (t * p.getX());
10      double int_y = this.p1.getY() + (t * p.getY());
11      return (
12          int_x <= Math.max(p1.getX(), p2.getX()) &&
13          int_x >= Math.min(p1.getX(), p2.getX()) &&
14          int_y <= Math.max(p1.getY(), p2.getY()) &&
15          int_y >= Math.min(p1.getY(), p2.getY()))
16          ? new Point(int_x, int_y, z) : null;
17  }
```

Listing 2: Calcolo intersezione tra retta e piano z

3.3 Creazione delle immagini di output

L'ultimo passo consiste nel mostrare il risultato dello slicing, per fare questo la soluzione migliore consiste nel generare un'immagine per ogni layer, la libreria `java.awt.image.BufferedImage` permette di disegnare direttamente delle rette sull'immagine. Le coordinate reali dei vertici vengono mappate sui pixel dell'immagine creando una rappresentazione del layer. L'output è in formato BMP per evitare di perdere dettagli a causa della compressione JPEG.

```
1   BufferedImage layer_image = new BufferedImage(
2       (int) Math.ceil(
3           (Math.abs(minX) + Math.abs(maxX) + 10)
4           * output_scaling),
5       (int) Math.ceil(
6           (Math.abs(minY) + Math.abs(maxY) + 10)
7           * output_scaling),
8       BufferedImage.TYPE_INT_RGB);
9   for (Segment s : layer) {
10      layer_image.getGraphics().drawLine(
11          (int) (s.getP1().getX() + Math.abs(minX) + 5)
12           * output_scaling,
13          (int) (s.getP1().getY() + Math.abs(minY) + 5)
14           * output_scaling,
15          (int) (s.getP2().getX() + Math.abs(minX) + 5)
16           * output_scaling,
17          (int) (s.getP2().getY() + Math.abs(minY) + 5)
18           * output_scaling);
19  }
```

Listing 3: Inserimento dei segmenti nell'immagine

3.4 Risultati

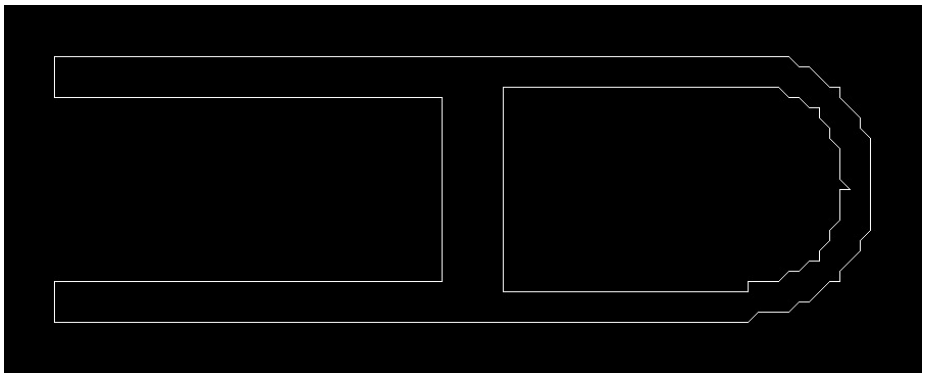

Figure 3: Layer 8 dell'oggetto, layer da 0.5mm

Il risultato è ottimo nelle linee rette, tuttavia si notano evidenti problemi non di poca importanza nei segmenti curvi, dopo svariate ore di analisi non sono riuscito a trovare una risposta a questo problema in quanto debuggare un codice così semplice ma con così tanti dati risulta estremamente lungo e difficile. Il codice è in grado di scalare su più thread, creano un thread per ogni layer (specialmente utile in caso di altezze di layer molto piccole). Mi ritengo soddisfatto del risultato in quanto ho avuto modo di entrare all'interno della scatola nera che fino ad ora erano per me gli slicer, ho avuto modo di applicare le conoscenze di algebra lineare e di informatica grafica e sento di aver capito più a fondo il concetto di modello 3D e manipolazione di modelli 3D.