Università di Pisa

Dipartimento di Informatica
Corso di Laurea Triennale in Informatica

Corso a Libera Scelta - 6 CFU

Computer Graphics

Professore:
Prof.

Autore:
Filippo Ghirardini

Anno Accademico 2023/2024

Contents

Computer Graphics

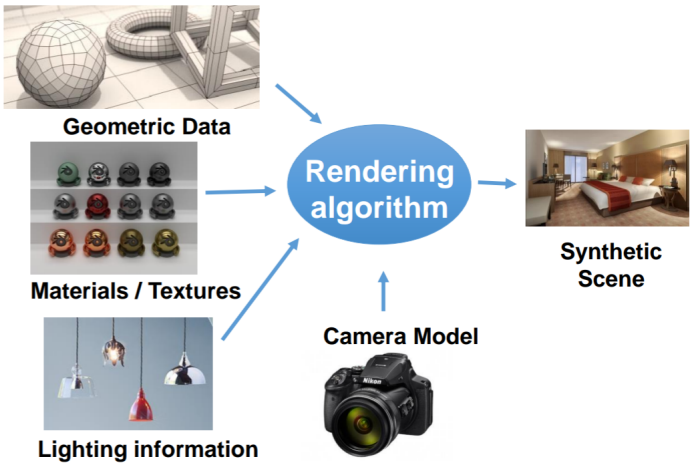
Realizzato da: Ghirardini Filippo

A.A. 2023-2024

---

1 Paradigmi di renderizzazione

Un **algoritmo di rendering** è una serie di passi che trasforma la descrizione digitale di una scena e di quattro parametri in un'**immagine raster**.



1.1 Ray Tracing

L'idea alla base è quella di "sparare" un **raggio** dal punto di partenza e controllare se ha colpito qualche oggetto.

```
for each pixel p;
   make a ray r (viewpoint to p)
   for each primitive o in scene:
      find intersect(r,o)
   keep the closest intersection o_j
   find color of o_j at p
```

Non tutti gli oggetti della scena però saranno illuminati, quindi quando un raggio interseca il punto della scena si fa partire un altro raggio che va verso la fonte di luce.
Se quest'ultimo incontra un oggetto vuol dire che l'oggetto è in ombra e non è raggiunto dalla luce.
C'è da tenere conto che la luce rimbalza un certo numero di volte, tramite il **reflection ray**. Ovviamente questo costa risorse, più si fa rimbalzare la luce e più l'immagine è realistica e costosa.
C'è poi la **rifrazione** di un oggetto che consiste sempre nel far partire altri raggi una volta che uno raggiunge un oggetto (ad esempio una bottiglia d'acqua).

1.1.1 Costo

Dipende da quanti rimbalzi (N) facciamo e da quante intersezioni con gli oggetti abbiamo:

$$RTCost(r) = N \sum_{\forall o \in S} Int(r, o) \tag{1}$$

Si noti che $Int(r, o)$ rappresenta il costo dell'intersezione del raggio r con l'oggetto o.

1.1.2 Primitive

Tutto ciò che riesco facilmente ad intersecare con raggi:

- Triangoli, quadrilateri, etc...
- Superfici implicite: sfera, geometria solida costruttiva, etc...

1.2 Rasterizzazione

Nella rasterizzazione (*Transform & Lighting*) proietto le primitive della scena sul mio schermo, ovvero prendo ogni **vertice** di ogni primitiva, lo proietto verso il viewpoint e vedo dove interseca la mia finestra. La linea che segna è il **proiettore**. A partire dalla proiezione dei soli vertici saprò quali pixel selezionare.
Il vantaggio principale è che è sufficiente proiettare pochi vertici per rasterizzare gran parte dello schermo.

```
for each primitive t:
   find where t falls on screen
   rasterize the 2D shape
   for each produced pixel p:
      find the color for t
      color p with it
```

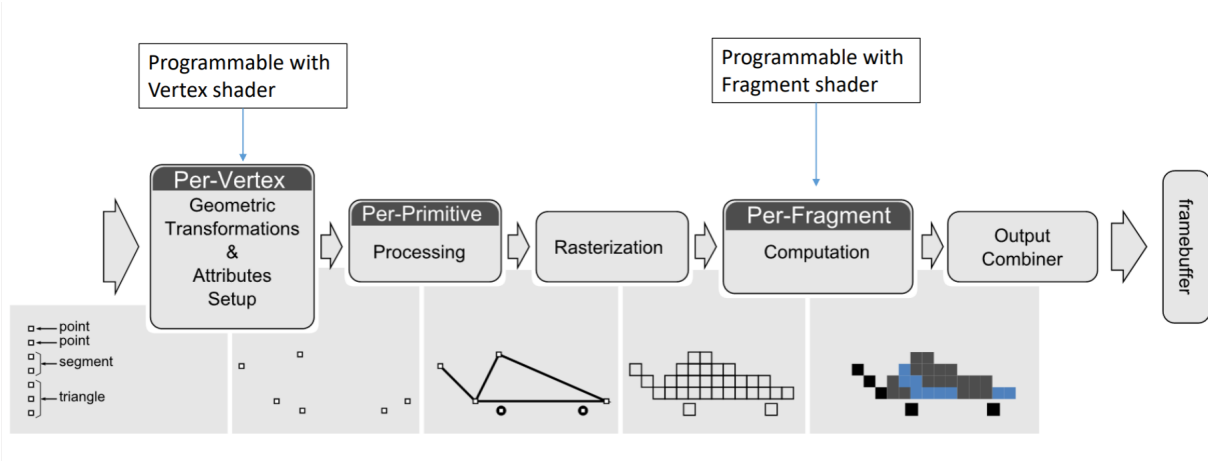
1.2.1 Primitive

Tutto ciò che so proiettare da 3 dimensioni a 2 e rasterizzare:

- Punto
- Segmento
- Triangolo

1.2.2 Pipeline

1. Identifico le proiezioni delle primitive sullo schermo
2. Trovo l'area da rasterizzare
3. La computo



1.2.3 Punto

1.2.4 Linea

Conoscendo i due vertici trovare tutti i pixel che la coinvolgono. Utilizzando l'equazione della rette si applica il seguente codice:

```
RasterizeH(x0,y0,x1,y1){
   m = (y1-y0)/(x1-x0)
   x=x0;
   y=y0;
   do{
      pixel(x,y)=0
   }
}
```

1.2.5 Triangoli

Dati i tre vertici, se i lati passano per il centro del pixel allora questo viene illuminato. Questo ci garantisce che non ci sia ambiguità in caso di triangoli adiacenti, poiché non "litigheranno" per lo stesso pixel.

1.2.6 Costi

È composto dal costo della trasformazione dei vertici e il costo di rasterizzare la primitiva p in proporzione alla sua dimensione sullo schermo:

(2)

1.3 Confronto

Ancora ad oggi il metodo della *rasterizzazione* è il più popolare.
I vantaggi del *ray-tracing* sono i seguenti:

- Algoritmo più semplice concettualmente
- Ottimo per effetti grafici complessi di **alta qualità**

mentre quelli della *rasterizzazione*:

- Complessità più controllabile in quanto non serve tutta la scena in ogni momento della renderizzazione: ogni primitiva lavora per conto suo ed è quindi più facilmente parallelizzabile
- Funziona meglio con i dati dinamici (oggetti che si muovono sulla scena)
- Più **controllabile** e **veloce**

Uno dei motivi principali per cui ancora oggi la rasterizzazione è più popolare è per come affronta il problema dell'**Hidden Surface Removal**.

1.3.1 Combinazione

La soluzione vincente è quella di usare entrambi gli approcci: la **rasterizzazione** per disegnare gli oggetti della scena e il **ray-tracing** per elaborare luci ed ombre.