H1 Immagini Digitali

H2 Introduzione

Per visualizzare un modello 3D di qualsiasi tipo a schermo dovremo *renderizzarlo*, cioè fare ***rendering***.

> Per **rendering** intendiamo il processo che permette la visualizzazione di modelli 3D a schermo.

Possiamo quindi immaginare il rendering come il processo che converte il modello 3D in un'**immagine digitale** visualizzabile a schermo.

Come possiamo definire in modo "formale" un'immagine, in modo da poterla rappresentare digitalmente? Una definizione semplice e efficace è la seguente:

> Un'immagine è un assegnamento di un colore ad ogni punto di una regione piana, cioè una funzione
>
> $$f : (x, y) \mapsto \text{colore} \tag{1}$$

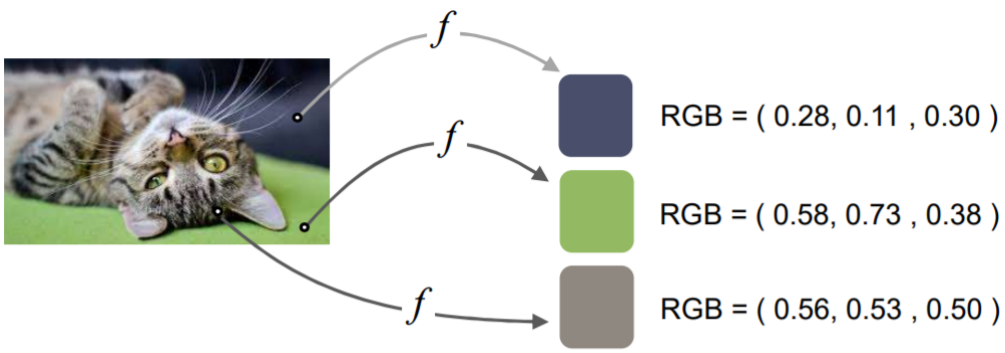
Principalmente, ci sono due modi diversi di eseguire questo mapping, e quindi due tipi di rappresentazione di un'immagine:

- Rappresentazioni vettoriali
- Rappresentazioni rasterizzate

Prima di descriverle, vediamo innanzitutto come fare a rappresentare digitalmente il colore.

H3 Rappresentare digitalmente il colore

Il colore viene rappresentato secondo la **teoria del tristimolo** come una tripletta di colori primari, che sovrapponendosi "creano" un nuovo colore (*sintesi additiva del colore*). I tre colori sono rosso, verde, blu; cioè la tripla RGB. Ciascun colore primario avrà un valore di intensità variabile dal minimo al massimo, che ci permette di variare il colore risultante sullo schermo.



H3 Rappresentazioni vettoriali

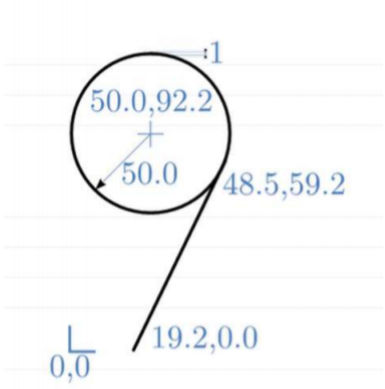
Le immagini sono rappresentate come set di primitive, le quali possono essere:

- curve parametriche
- poligoni, cerchi, ...
- testo

Ad ogni primitiva è associabile un colore, andando quindi a definire il mapping (1).

Un esempio giocattolo di un'immagine vettoriale è il seguente:

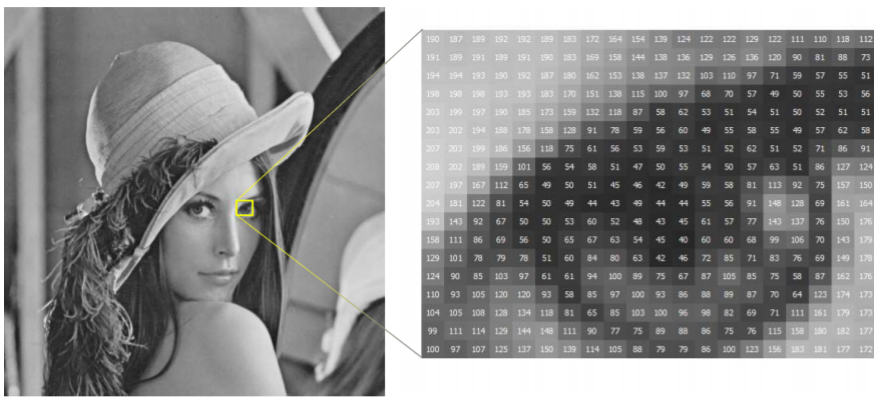
```
NUMBER_OF_PRIMITIVES 2

CIRCLE
center 50.0, 92.2
radius 50.0
fill_color 1.0, 1.0, 1.0
line_color 0.0, 0.0, 0.0
line_thickness 1pt

SEGMENT
endpoint_one 19.2, 0.0
endpoint_two 48.5, 59.2
line_color 0.0, 0.0, 0.0
line_thickness 1pt
```



Come si può notare, la memoria usata è solo quella relativa alla descrizione matematica delle primitive.

H3 Rappresentazioni rasterizzate

Le immagini sono rappresentate come griglie regolari 2D, e ciascuna cella è un ***pixel***. Ad ogni pixel viene associato un colore, andando quindi a definire il mapping (1). Ad esempio in un'immagine in scala di grigi avremo:



La dimensione della griglia, cioè il numero di pixel, definisce la *risoluzione* dell'immagine. Le immagini raster sono *resolution-dependent*:

Ogni pixel ha k canali, dove k è il numero di colori primari usati per la rappresentazione del colore (un'immagine a scala di grigi ha $k = 1$, una RGB ha $k = 3$). Ogni canale è rappresentato da un dato numero di bit, quindi maggiore è il numero di bit per canale, più "sfumature" potrà avere il colore di quel canale. Solitamente, per le immagini RGB si usano 8 bit per canale (*"true color"*), quindi l'intensità dei colori primari ha $2^8 = 256$ valori possibili. Da questa caratteristiche viene determinata un'ulteriore proprietà delle immagini raster:

> **Image depth**: bit totali per un pixel

Ad esempio, nel caso delle immagini RGB con 8 bit per canale avremo
$\text{image depth} = 3 \times 8 = 24\text{bit}$
È possibile introdurre anche il concetto di **Dynamic Range (DR)**:

> **Dynamic Range** è il rapporto fra la luminosità del punto più luminoso e quella del punto più buio

Un Dynamic Range elevato (HDR) richiede un elevato numero di bit per canale, quindi un'elevata image depth.

La memoria richiesta per un'immagine rasterizzata è quindi calcolabile nel seguente modo:

$$resX \times resY \times imageDepth$$

Ad esempio per un'immagine true color con risoluzione 1080×1920 (1080p) peserà

$$1080 \times 1920 \times 24 = 49.766.400\text{bit} = 6.220.800\text{byte} \approx 6\text{MB} \quad (2)$$

Come si può notare dall'esempio, la compressione è spesso necessaria, e alcuni formati la prevedono (JPEG) e altri no (RAW).

H3 Immagini vettoriali vs. immagini raster

Immagini vettoriali

- ✔ Sono indipendenti dalla risoluzione del monitor
- ✔ Adatte per loghi, design e immagini artificiali

✔ Usano poca memoria

✔ Non presentano *aliasing* se ingrandite

✘ Difficili da renderizzare direttamente, vengono quindi rasterizzate

Immagini raster

✔ Adatte per immagini naturali (fotografie)

✔ Facili da renderizzare direttamente dal monitor

✘ Sono dipendenti dalla risoluzione del monitor

✘ Presentano *aliasing* se ingrandite

H3 Hardware display

Per capire quali dati dobbiamo produrre per renderizzare qualcosa a schermo, dobbiamo avere un'idea di come funzioni un display moderno. Storicamente vi sono stati due tipi importanti di display:

- Display vettoriali
- Display raster

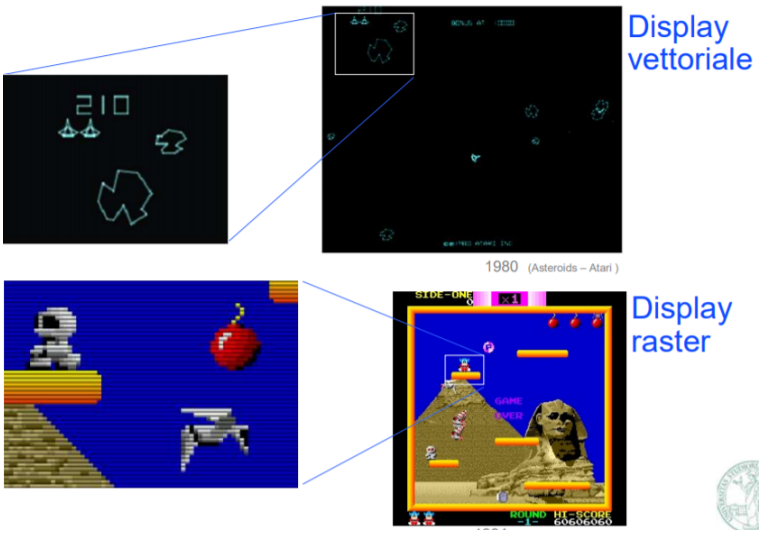
In entrambi i casi assumeremo una tecnologia a tubo catodico

H4 Display vettoriali

Si tratta idealmente di un fascio di elettroni che si muove liberamente e traccia linee sul monitor. Il rendering in questo caso dovrà produrre il cammino che il fascio dovrà percorrere per tracciare le immagini. Inoltre il fascio avrà sempre la stessa intensità.

H4 Display raster

Lo schermo è diviso in pixel fisici che vengono illuminati dal fascio di elettroni, che questa volta si muoverà in modo fisso, riga per riga (tipicamente dall'angolo in alto a sinistra) un certo numero di volte al secondo (**refresh rate** o **frame rate**). Il fascio avrà intensità variabile, e quindi il rendering dovrà produrre i valori di intensità per ciascun pixel, ovvero dovrà produrre un'immagine rasterizzata.



I display moderni, anche se non più con tecnologia a tubo catodico, sono display raster.

I pixel fisici del display raster sono associati ad un buffer specifico in memoria, detto ***screen buffer*** o ***frame buffer***. I pixel fisici `fps` volte al secondo vengono illuminati a intensità variabile, secondo il valore attuale dello screen buffer. Visualizzare qualcosa a schermo implica il modificare il valore dello screen buffer. Quanti byte bisogna produrre per unità di tempo per riempire lo screen buffer? Questa metrica è il **fill rate** e viene calcolata nel seguente modo:

$$\text{fill rate} = res \times depth \times fps$$

Ad esempio, prendendo l'immagine dell'esempio (2) e una frame rate pari a 60, avremo:

$$6.220.800 \times 60 = 373.248.000 \text{byte/s} \approx 373 \text{MB/s}$$

La quantità di dati da produrre al secondo è elevata, ma fortunatamente il processo è altamente parallelizzabile → **GPU**.