Progetto di programmazione avanzata e parallela

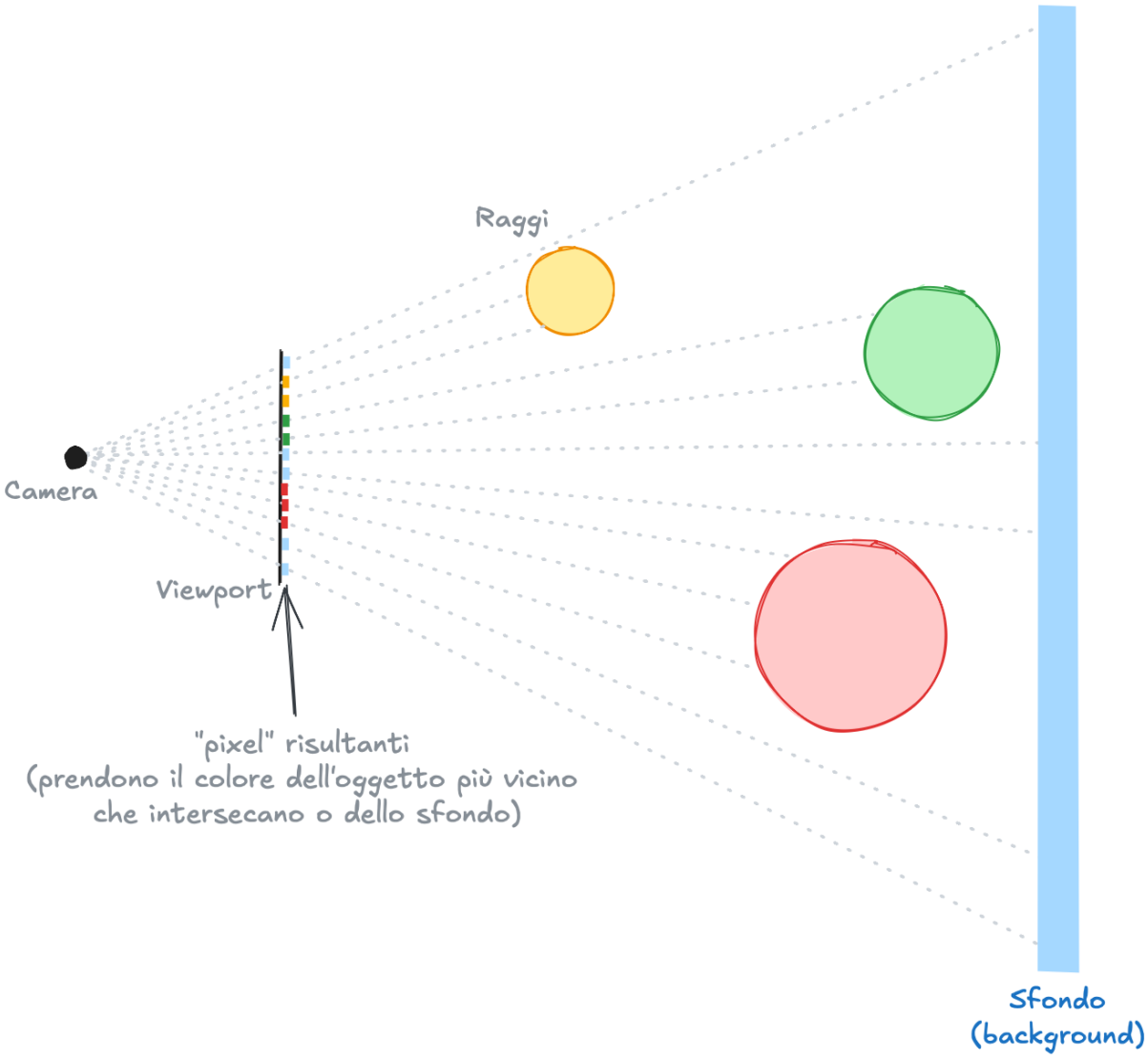
Parte 1 di 2 - Linguaggio C

Lo scopo di questo progetto è la realizzazione di un programma in grado di effettuare il rendering tramite tecniche di raytracing di un insieme di sfere in cui l'utente possa specificare posizione, raggio e colore.

Come funziona il raytracing

L'idea del ray tracing è quella di avere una serie di "raggi" (almeno uno per pixel) che sono emessi da una camera attraverso un "viewport" (pensatelo come l'obiettivo della camera) e che vanno a intersecare diversi oggetti in una scena. Il colore dell'oggetto che viene intersecato corrisponderà poi al colore del pixel a cui è associato il raggio.

Nel caso bidimensionale possiamo vederne un esempio nell'immagine seguente:



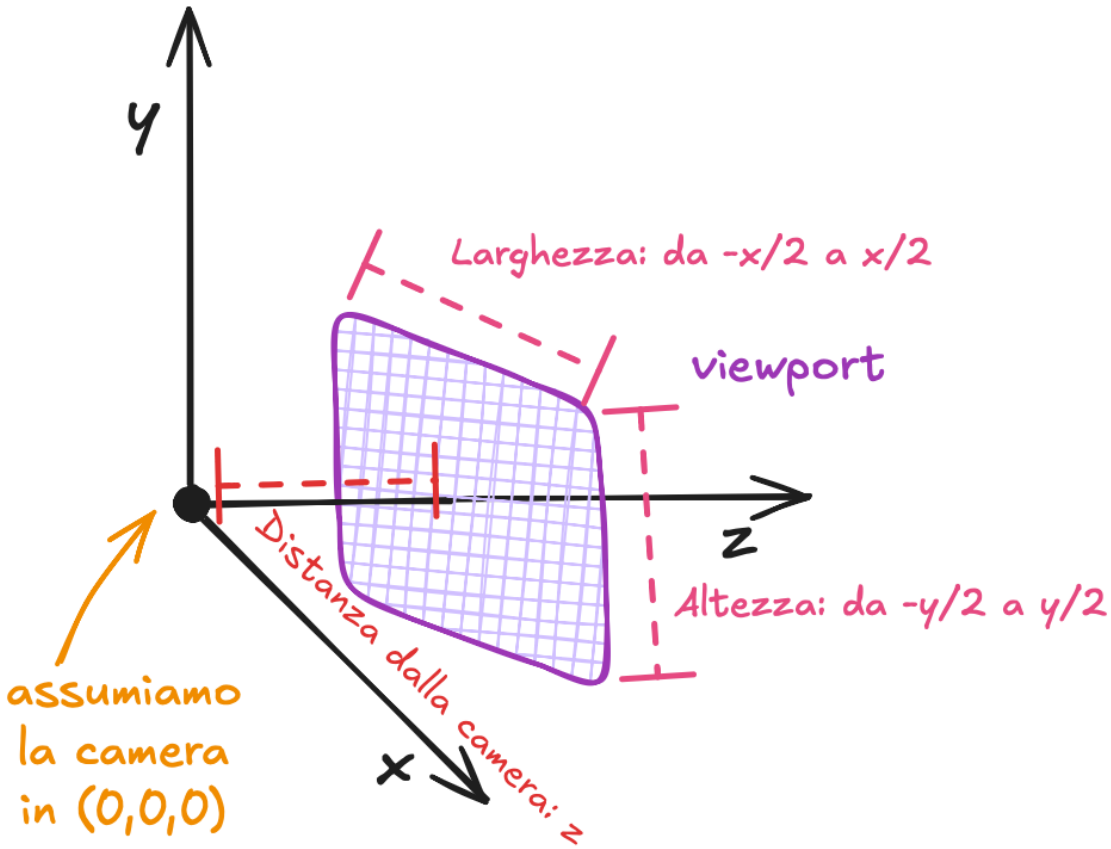
Notate come sia generato un raggio per ogni "pixel" (in questo esempio "l'immagine" risultante è di una singola dimensione) uniformemente distribuiti nel viewport. Ogni raggio viene rappresentato da un vettore di lunghezza unitaria la cui direzione è definita in modo univoco da due punti: la camera e un punto del viewport (corrispondente al pixel che si vuole "colorare"). Il vettore viene poi prolungato e si verifiche se interseca almeno un oggetto della scena (nel caso ne intersechi più di uno è necessario scegliere il più vicino). In quel caso il colore del pixel corrisponde a quello dell'oggetto intersecato. Se non vi sono oggetti intersecati allora il colore assunto è il colore di sfondo.

Abbiamo quindi diversi componenti da descrivere:

1. La posizione della camera
2. La posizione del viewport
3. Gli oggetti della scena
4. Come passare da viewport a pixel dell'immagine
5. Come generare i raggi
6. Come verificare che oggetti sono intersecati dai raggi

Questi sono descritti come segue:

1. La posizione $C = (x, y, z)$ della camera sarà assunta, per semplicità, essere sempre $(0, 0, 0)$. Notate come questa non sia una perdita di generalità in quando è sempre possibile cambiare le coordinate affinché C sia considerata l'origine degli assi.
2. Il view port è da considerarsi un rettangolo che, per semplicità, sarà sempre considerato perpendicolare all'asse z e tale per cui l'asse z passi esattamente nel centro, come dalla seguente figura:



3. Per semplicità gli oggetti della scena saranno considerati solamente come sfere, definite da un centro $C_s = (x_s, y_s, z_s)$ e un raggio r. Si ricorda che la sfera è l'insieme dei punti (x, y, z) tali per cui $(x_s - x)^2 + (y_s - y)^2 + (z_s - z)^2 \leq r^2$.
4. Si supponga di avere un viewport di dimensioni 1.77×1 (corrispondente al normale formato 16:9) a distanza 1 dalla camera. In questo caso se l'immagine da generare è di 1920×1080 pixel allora sarà necessario avere un raggio (rappresentato da un vettore) per ciascuno dei pixel. Ognuno di questi vettori sarà della forma $V = (\frac{1.77}{1920-1}i - 1.77/2, \frac{1}{1080-1} - 1/2j, 1)$, $i \in [0, 1920)$ e $j \in [0, 1080)$. Notate che i valori $1.77/2$ e $1/2$ sono necessari per centrare il viewport in modo che l'asse z passi per il suo centro. Il vettore V deve poi essere normalizzato rendendolo di lunghezza unitaria.
5. Un vettore V di lunghezza unitaria indica solo la direzione del raggio, il raggio è da considerarsi la retta denotata dal vettore V, quindi tV per $t \in \mathbb{R}$ (anche se solo la parte positiva sarà considerata, dato che il raggio non deve propagarsi "dietro" la camera).
6. Per stabilire se un raggio Vt interseca una sfera di centro C_s e raggio r è necessario considerare se $(Vt - C_s) \cdot (Vt - Cs) \leq r^2$ (con $\cdot$ rappresentante il prodotto interno), quindi se $(V \cdot V)t^2 - 2C_s \cdot Vt + C_s \cdot C_s \leq r^2$, con t come incognita. Notate che serve solo verificare se $(V \cdot V)t^2 - 2C_s \cdot Vt + C_s \cdot C_s = r^2$. Se ha zero soluzioni allora il raggio non interseca la sfera. Si definiscano $a = V \cdot V$, $b = -2(C_s \cdot V)$, $c = C_s \cdot C_s - r^2$. Si ha quindi che se $b^2 - 4ac < 0$ non ci sono soluzioni (il raggio non interseca), se $b^2 - 4ac = 0$ allora l'intersezione è a distanza $\left|\frac{-b}{2a}\right|$, se $b^2 - 4ac > 0$ allora ci sono due intersezioni con la superficie della sfera a distanza $\left|\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}\right|$. Notate che se vi sono più intersezioni con più

sfere il colore da considerare è solo quello dell'intersezione più vicina.

Formato PPM

Il formato PPM è un formato estremamente semplificato per la gestione delle immagini a colori. In particolare, esistono due possibili varianti di questo formato, una in cui i dati sono rappresenti in formato testuale e una in cui i dati sono rappresentati in formato binario. Per il progetto è richiesto l'utilizzo di questo secondo formato.

La struttura generale di questo formato è la seguente:

```
P6
width height
255
[binary data]
```

dove `P6` indica il formato, `width` e `height` sono la larghezza e l'altezza dell'immagine in pixel e `255` il valore massimo di intensità dei singoli canali di colore (in questo caso essendo `255` è richiesto un byte per ogni canale di colore). Notate come le prime tre righe del file siano in formato testuale e i dati di colore dei singoli pixel inizino solo dalla quarta riga. Questi dati sono rappresentati da una sequenza di valori di 24 bit divisi in 8 bit per l'intensità di ognuno dei tre canali colore (rosso, verde e blu in questo ordine) per ognuno dei $\texttt{width} \times \texttt{height}$ pixel. I pixel sono rappresentati a partire dall'angolo in alto a sinistra dell'immagine una riga alla volta dall'alto verso il basso. All'interno delle singole righe i pixel sono ordinati da sinistra verso destra.

Formato di definizione della scena

Per definire gli oggetti della scena viene definito un file in un formato specifico che vada a definire le caratteristiche globali della scena e tutti gli oggetti (solo sfere) presenti nella scena stessa.

Un esempio di file di definizione della scena è il seguente:

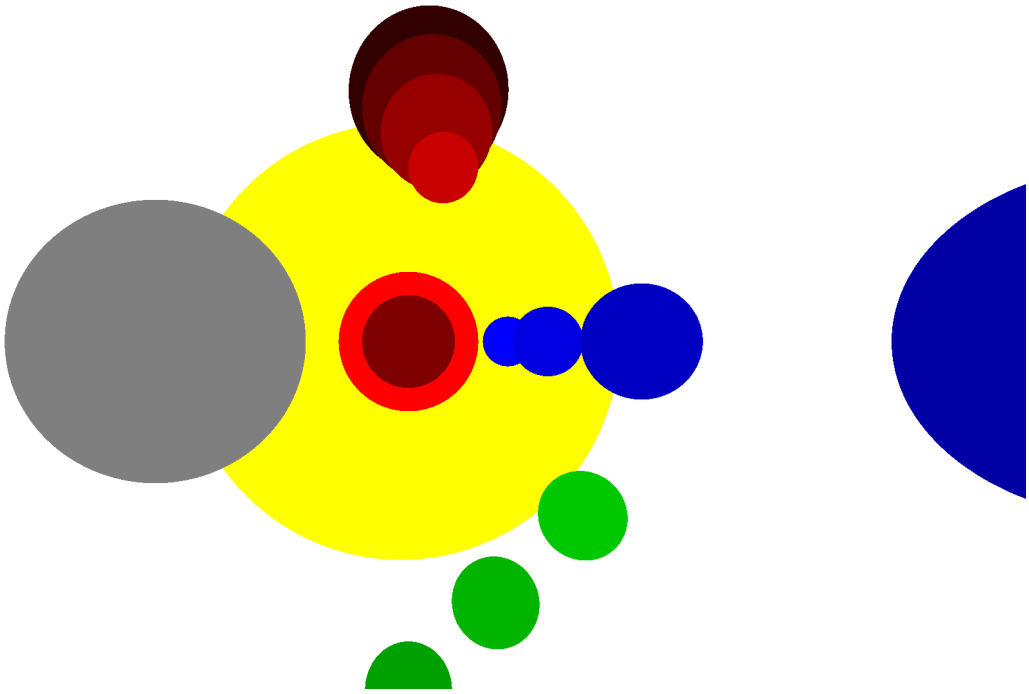
```
VP  1.777 1 1
BG 255 255 255
OBJ_N 16
S 0.2 1 4 0.2 200 0 0
S 0.2 1.5 5 0.4 150 0 0
S 0.2 2 6 0.6 100 0 0
S 0.2 2.5 7 0.8 50 0 0
S 0 0 3 0.2 127 0 0
S 0 0 10 1 255 0 0
S 2 -2 8 0.5 0 200 0
S 1 -3 8 0.5 0 180 0
S 0 -4 8 0.5 0 160 0
S -1 -5 8 0.5 0 140 0
S 1 0 7 0.25 0 0 255
S 1 0 5 0.25 0 0 225
S 1 0 3 0.25 0 0 195
S 1 0 1 0.25 0 0 165
S -35 0 100 20 127 127 127
S -100 0 10000 3000 255 255 0
```

In questo file possiamo riconoscere i seguenti componenti:

1. `VP float float float` indica il *viewport*, in cui le dimensioni sono nell'ordine x, y e z. In questo caso il viewport sarà dato da un rettangolo in i vertici sono in posizione $(x/2, y/2, z)$, $(-x/2, y/2, z)$, $(x/2, -y/2, z)$ e $(-x/2, -y/2, z)$.
2. `BG u_int8_t u_int8_t u_int8_t` indica il colore di background della scena.
3. `OBJ_N int` indica il numero di oggetti che andranno a seguire. Nel ray tracer del progetto questi oggetti sono solamente sfere
4. `S float float float float u_int8_t u_int8_t u_int8_t` definisce una sfera in cui:
    - I primi tre valori `float` indicano le coordinate (x, y, z) del centro della sfera;
    - La quarta coordinata `float` indica il raggio della sfera;
    - I tre valori `u_int8_t` indicano il colore in formato RGB della sfera.

Nel caso il file non sia ben formato il programma può terminare indicando con un messaggio di errore la non correttezza del file di definizione della scena.

Il risultato del rendering della scena indicata nel file di esempio è il seguente:



Interfaccia da linea di testo del programma

Il programma deve accettare come argomenti da linea di comando esattamente in questo ordine:

1. una stringa che indica il nome del file della scena (e.g., `test.txt`);
2. una stringa che il nome del file immagine in cui salvare il risultato del ray tracing (e.g., `test.ppm`);
3. Due valori interi positivi rappresentanti la *larghezza* e l'*altezza* in pixel dell'immagine da generare.

Il programma dovrà leggere il file di definizione della scena e generare la corrispondente immagine delle dimensioni indicate da linea di comando. Nel caso vi siano errori (e.g., file non leggibile, non nel formato corretto, file immagine non scrivibile) il programma potrà terminare indicando la presenza di un errore. Si ricorda che il programma deve comunque terminare in modo "ordinato", non semplicemente terminare a causa, per esempio, di un segmentation fault.

È consentito ma non richiesto fornire informazioni diagnostiche durante l'esecuzione del programma.

Assunzioni e suggerimenti

È possibile assumere le seguenti semplificazioni:

- La camera e il viewport sono esterni a ogni sfera e non vi sono sfere dietro la camera (altrimenti sarebbe necessario verificare che la sfera con intersezione più vicina non sia dietro la camera).
- Non vi sono sfere interamente coincidenti (cosa che renderebbe non chiaro quale colore assumere).

Si suggeriscono i seguenti approcci:

- Scrivere delle funzioni e dei tipi per lavorare con vettori di 3 elementi di valori float, definendo adeguate struct e typedef.
- Si mantenga una struttura dati per iterare facilmente su tutti gli oggetti di una scena. Notate che una volta letti questi saranno sempre in sola lettura.
- È consentito e consigliato l'utilizzo di `__attribute__((packed))` per avere una rappresentazione dei colori che occupi esattamente 24 bit (3 byte).

Requisiti

- Il programma deve essere strutturato in più file, almeno un file `main.c`, due file `scene.h` e `scene.c` per la gestione della scena e della generazione del colore dei singoli pixel e due file `ppm.h` e `ppm.c` per il salvataggio dell'immagine nel formato corretto.
- Il programma deve poter essere compilato tramite un makefile.
- La generazione dei pixel dell'immagine deve essere eseguita in parallelo usando OpenMP.
- Il salvataggio dell'immagine deve usare `mmap`. Per la lettura del file di scena si consiglia invece un approccio con `fscanf`.
- Come tipo di dato floating point si utilizzi `float`.
- Il programma deve correttamente rilasciare le risorse allocate.
- In caso di errore (e.g., file non apribile) il programma deve segnalare l'errore e uscire normalmente con un codice di errore diverso da 0.
- Ogni funzione rilevante (i.e., non di 2-3 righe) deve essere adeguatamente commentata spiegando che compito svolge, che input richiede e che output genera.
- Ogni file .c e .h deve contenere all'inizio come commento il proprio nome, cognome e numero di matricola.

Note importanti

- Questo progetto è relativo solamente alla parte in C del corso. Per sostenere la parte di progetto del corso è necessario consegnare sia la parte in C (parte 1) che la parte in Python (parte 2).
- Sebbene appaia ovvio, quando viene chiesto di inserire Nome, Cognome e Matricola, dovete inserire il vostro nome, cognome e numero di matricola a non, letteralmente, il testo "Nome", "Cognome" e "Matricola".
- Le istruzioni di consegna saranno fornite con la parte 2 del progetto.