Appunti per POV-Ray

Area progetto *Longitudine*, 5D Liceo Torricelli

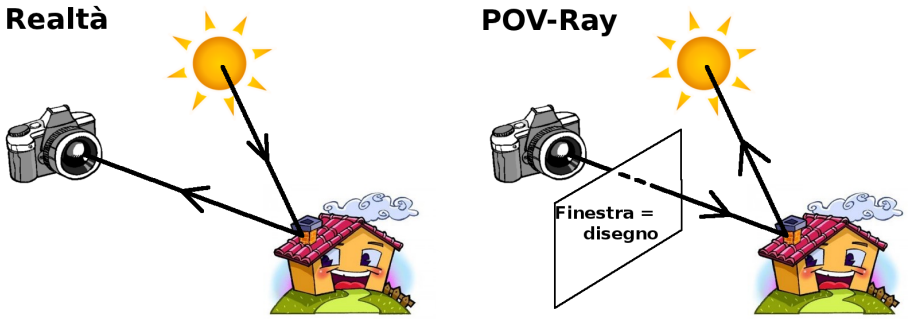
anno scolastico 2012-13

1 Introduzione

POV-Ray (Persistence of Vision Ray-Tracer) crea scene tridimensionali con una tecnica di rendering chiamata ray-tracing. Legge in un file le descrizioni di oggetti e luci della scena e genera un'immagine dal punto di vista della macchina fotografica specificata.

Il ray-tracing simula il percorso di raggi di luce; rispetto alla realtà però, i raggi viaggiano al contrario: nella realtà la luce viene emessa da una sorgente e illumina gli oggetti. I raggi riflessi dagli oggetti arrivano ai nostri occhi (o alla macchina fotografica). La maggior parte dei raggi, però, non colpisce il nostro occhio, quindi una scena renderizzata così calcolerebbe inutilmente molti raggi e necessiterebbe di un lunghissimo tempo.

Invece nel ray-tracing i raggi partono dalla macchina fotografica e sono diretti verso la scena. Essi passano tutti per una "finestra" che corrisponde all'immagine finale. Vengono tracciati uno o più raggi per ogni pixel nell'immagine finale. Il colore di ogni pixel viene modificato in modi diversi a seconda degli oggetti che incontrati dai raggi e delle proprietà specificate (colore, trasparenza, materiale, ecc.). Le modifiche sono calcolate tramite dei raggi che dall'oggetto sono tracciati verso le sorgenti di luce.



Il linguaggio per descrivere le scene è chiamato proprio "scene description language". I comandi vengono inseriti in un file di testo, che viene poi interpretato da POV-Ray per generare l'immagine.

Alcuni link utili:

- `www.povray.org` è il sito ufficiale, vi si trovano una galleria di immagini, un tutorial, un forum, le istruzioni per scaricare e installare POV-Ray
- `www.povray.org/documentation/` contiene il tutorial ufficiale in html
- `www.povray.org/ftp/pub/povray/Official/Documentation/povdoc-3.6.1-a4-pdf.zip` è il link diretto al tutorial in pdf (compresso)
- `www.f-lohmueller.de/pov_tut/pov__ita.htm` è un altro buon tutorial in inglese, tedesco, e anche in italiano con qualche errore di ortografia.

2 Primo esempio

Per definire oggetti nello spazio servono tre coordinate. Il sistema di coordinate di POV-Ray è formato dall'asse x che punta a destra, dall'asse y che punta in alto e dall'asse z che punta verso l'interno dello schermo (mano sinistra).

Apriamo il file `esempio1.pov`.

```
// questo è un commento, non sarà letto dal compilatore
/* questo è
un commento
su più righe */

#include "colors.inc" // questi files contengono
#include "stones.inc" // elementi predefiniti

camera {              // macchina fotografica:
  location <0, 2, -3> // posizione
  look_at <0, 1, 2>   // direzione verso cui è rivolta
}

sphere {              // oggetto: nome
  <0, 1, 2>, 1.5      //          dati per costruirlo
  texture {           //          caratteristiche
    pigment { color Yellow }
  }
}

light_source {        // luce:
  <2, 4, -3>          //       posizione
  color White         //       colore
}
background {          // sfondo
  color Red
}
```

I punti sono descritti con tre coordinate (vettori). Si usano vettori a 2, 3, 4, 5 componenti (noi useremo quasi esclusivamente quelli a 3 componenti).

I numeri possono essere scritti in modi diversi, con il punto come separatore decimale: `1` è come `1.` e `1.0`. La scrittura `.0` significa `0.0`. Il numero `0.5` è uguale a `1/2` ecc.

Per aiutare la lettura si usano alcuni accorgimenti:

- indentazione e colorazione delle parole chiave (automatica!)
- usare tanto spazio e nuove righe quando serve: le righe vuote e gli spazi non hanno significato per il compilatore
- usare i commenti: dopo qualche giorno non ci ricordiamo più cosa abbiamo scritto
- scrivere file ordinati: tutti gli oggetti vicini, tutte le luci vicine, ecc.

L'ordine delle istruzioni in generale non è importante, a parte gli `#include`, che vanno messi prima del punto in cui vengono richiamati gli oggetti definiti al loro interno (per non sbagliarsi, metterli sempre in cima!).

Gli oggetti sono descritti nel modo più rapido possibile:

- `sphere`: sfera, un punto per il centro e un numero reale per il raggio.
- `box` (scatola): parallelepipedo, due punti che identificano due vertici opposti.
- `cylinder`: cilindro, un punto per il centro di una base, un punto per il centro dell'altra base e un numero per il raggio.
- `cone`: cono o tronco di cono, un punto per il centro di una base, un numero per il raggio di quella base, un punto per il centro della seconda base, un numero per il raggio della seconda base.
- `plane`: piano (2d), un vettore ortogonale al piano e un numero che indica la distanza dall'origine lungo il vettore.
- `polygon`: poligono (2d), il numero di vertici, le coordinate dei vertici (su un piano). Le coordinate possono essere espresse come vettori 2d, in questo caso la coordinata z è interpretata come 0.
- `disc`: disco (2d), un punto per il centro, un vettore ortogonale, un numero per il raggio.

Esercizio 1. *Disegna su carta (anche approssimativamente) alcuni dei seguenti oggetti e verifica che i dati specificati li definiscono univocamente.*

```
sphere {
  <1, 0, 1>, 1
}
```

```
box {
  <-1, -1, -1>, <1, 1, 1>
}

cylinder {
  <0, -1, 0>, <0, 1, 0>, 4
}

cone {
  <0, 0, 0>, 2, <0, 1, 1>, 0
}

cone {
  <0, 0, 0>, 2, <0, 1, 1>, 4
}

plane {
  <1, 1, 0>, 2
}

plane {
  <1, 1, 0>, -2
}

polygon {
  4,
  <0, 0>, <1, 0>, <1, 1>, <0, 1>
}

polygon {
  4,
  <0, 0, 0>, <0, 1, 0>, <0, 1, 2>, <0, 0, 2>
}

disc {
  <1, 1, 1>, <0, 0, 1>, 2
}
```

Le proprietà degli oggetti (più precisamente, delle superfici) sono descritte nella sezione che inizia con la parola chiave `texture`.

I principali colori sono definiti (in inglese) nel file `colors.inc`, ma si possono definire colori a piacere, usando la codifica `rgb`, cioè red, green, blu: si indica quali quantità (tra 0 e 1) di ogni colore fondamentale vanno usate. Ad esempio:

```
  pigment { color rgb <1.0, 0.6, 0.6> } // rosa
```

Le luci (in ogni scena deve esserci almeno una luce, ma possono essercene anche di più!) sono invisibili ed emettono luce: le ombre sono calcolate in base alla posizione della luce e a dove essa punta.

Esercizio 2. *Modifica il file* `esempio1.pov` *in modo che lo sfondo sia grigio scuro e la sfera sia gialla. Definisci tu i colori tramite la parola chiave* `rgb`.
Sposta la luce in modo che la sfera sia illuminata dal basso.
Cosa succede se la luce è esattamente dietro la sfera?
Sposta la sfera in modo che si trovi nell'angolo in alto a destra, più distante dall'obiettivo.

3 Le trasformazioni

Nel file `esempio2.pov` troviamo l'oggetto

```
box {
  <-1, -0.7, -1>,   // un angolo
  < 1, 0.3, 3>      // l'angolo opposto
  texture {
    pigment { color rgbt <1.0, 1.0, 0.0, 0.6> } // 4 coordinate!
  }
  scale 0.5         // scala in tutte le direzioni
  translate 1.35*y  // trasla verso l'alto
  rotate 45*y       // ruota attorno all'asse y di 45 gradi
}
```

Il vettore dei colori ha la parola chiave `rgbt` e ha quattro coordinate: la quarta, `t`, indica la trasparenza. Un oggetto con `t` pari a `0` è completamente opaco, con `t` pari a `1` è completamente trasparente (provare per credere!).

Le tre righe dopo la fine della `texture` indicano le trasformazioni da applicare all'oggetto: `scale` (scala), `translate` (traslazione) e `rotate` (rotazione). Tutte prendono in input un vettore di tre componenti (che può essere specificato in modi diversi).

- `scale 0.5` significa: scala l'oggetto della quantità `<0.5, 0.5, 0.5>` nelle tre direzioni principali.
  Si può scalare di quantità diverse in ogni direzione, ad esempio `<0.25, 1, 0.5>`.
- `translate 1.35*y` significa: trasla l'oggetto secondo il vettore `<0, 1.35, 0>`. Infatti i tre vettori `<1,0,0>`, `<0,1,0>`, `<0,0,1>` si chiamano anche rispettivamente `x`, `y`, `z`. Quindi `0.35*y` è come `0.35*<0,1,0>` e come `<0,0.35,0>`.
- `rotate 45*y` significa: ruota l'oggetto di 45 gradi attorno all'asse `<0,1,0>`, cioè alla retta definita da questo vettore, cioè all'asse `y`. La direzione di rotazione viene definita con la regola della mano sinistra: si mette il pollice

nella direzione dell'asse puntandolo verso la parte positiva e la direzione di rotazione è data dalla direzione in cui si piegano le altre quattro dita.

Queste tre trasformazioni possono essere fatte in qualsiasi ordine, ma attenzione: le stesse trasformazioni in ordine diverso possono dare effetti molto diversi!

Esercizio 3. *I vettori possono essere sommati, moltiplicati per uno scalare, ecc. Ad esempio è lecita la scrittura*

```
translate 1.35*y + 5*<1,0,1> - 2
```

Esegui la somma.

Esercizio 4. *Scrivi un file che disegni i tre vettori* `x`, `y`, `z` *con tre colori diversi (rispettivamente rosso, giallo e blu). Suggerimento: disegna ogni vettore come un segmento "ciccione".*

Nello spazio conosciamo un altro tipo di trasformazione, la riflessione. Per questa non c'è una parola chiave, ma viene usata la scala con valori negativi. Ad esempio per riflettere un oggetto rispetto al piano orizzontale $y = 0$ possiamo scrivere

```
scale <1, -1, 1>
```

Le coordinate `x` e `z` non verranno toccate da questa trasformazione (sono scalate di 1), mentre la `y` sarà invertita.

Esercizio 5. *Cosa fa la trasformazione* `scale <1, -1, -1>`*? E la trasformazione* `scale <-1, -1, -1>`*?*

Esercizio 6. *Prova a calcolare cosa succede a un cubo di lato 1 con un vertice nell'origine e uno in* `<1, 1, 1>` *se gli viene applicata la trasformazione*

```
scale 2
translate <5, 6, 7>
```

e se invece la trasformazione applicata è

```
translate <5, 6, 7>
scale 2
```

Suggerimento: calcola dove vanno a finire i due vertici opposti specificati. Il cubo mantiene le stesse proporzioni nei due casi?
Cosa succede nai due casi se la scala è `-2`*?*

Esercizio 7. *Disegna su un foglio l'oggetto*

```
cylinder {
  <0,-1,0>, y, 0.5
  scale 2
  translate <2, 1, 3>
}
```

Verifica compilando un file in cui hai disegnato anche gli assi cartesiani. Ruota il cilindro in modo che rimanga al suo posto, ma le basi siano parallele al piano `xy` *(cioè l'asse del cilindro sia l'asse z).*

Esercizio 8. *Trova una trasformazione inversa di*

```
scale 2
translate <2, 1, 3>
```

che sia fatta in questo modo (con questo ordine!):

```
scale ...
translate ...
```

Verifica il risultato. Cosa succede se scambio l'ordine delle trasformazioni?

Esercizio 9. *(lungo) Disegna un orologio: deve avere un quadrante rotondo bianco, le tacchette delle ore (della forma preferita, ma tutte uguali) e due lancette che segnino le 16.30.*

4 CSG - costruzioni in geometria solida

La gamma di figure di base è molto limitata. Per creare nuovi oggetti si possono usare le operazioni insiemistiche tra due o più oggetti:

- `union`: unione
- `intersection`: intersezione
- `difference`: differenza (attenzione all'ordine!)

Il file `esempio3.pov` contiene il seguente codice.

```
#include "colors.inc"

camera {
  location <0, 0, -5>
  look_at <0, 0, 0>
}

light_source { <5, 10, -10> color White }
background { color Black }

union { // intersection, difference
  cylinder {
    <0, -1.5, 0>, <0, 1.5, 0>, 1
    pigment { color Red }
  }
  sphere {
```

```
    <0, 0, 0>, 1.3
    pigment { color Blue }
  }
}
```

Esercizio 10. *Cosa ti aspetti di vedere compilando il file?*
Cambia la parola chiave `union` *prima con* `intersection` *e poi con* `difference`*.*
Cosa ti aspetti?

Nel file `esempio4.pov` trovi il codice per disegnare una semplice casa.

```
#include "colors.inc"

camera {
  location <0, 2, -5>
  look_at <0, 1, 2>
}

light_source { <5, 10, -10> color White }
background { color Black }

plane {
  <0,1,0>, 0
  pigment { color rgb <0.1, 0.9, 0.3> }
}

union {
  cylinder {
    <0, 1, 0>, <0, 1.5, 0>, 1
    pigment { color rgb <0.9, 0.9, 0.3> }
  }
  cone {
    <0, 1.5, 0>, 1.3,
    <0, 2.5, 0>, 0
    pigment { color rgb <0.9, 0.6, 0.3> }
  }
  difference {
    box {
      <-1, 0, -1>, <1, 1, 1>
    }
    box {
      <-.2, 0, -1.1>, <.2, .6, 1.1>
    }
    pigment { color rgb <0.9, 0.9, 0.3> }  // colore delle scatole
  }
  rotate 20*y
}
```

La casetta è l'unione tra un cilindro, un cono e un terzo oggetto che a sua volta è una scatola da cui è stata tolta un'altra scatola. Le caratteristiche (come il colore) e le trasformazioni (come la rotazione) possono essere applicate a ogni singolo oggetto o al gruppo di oggetti. Se una stessa caratteristica è indicata più volte, viene interpretata solo la prima. Ad esempio, se nel primo `box` viene specificato `pigment { color Red }`, allora il `pigment` della differenza tra i box si applicherà solo alla seconda scatola (prova!). Le trasformazioni invece vengono interpretate tutte.

Esercizio 11. *Modifica il file nei seguenti modi e nota cosa succede (prima fai un'ipotesi e poi verificala)*

1. *modifica il cilindro in modo che le due basi stiano sui piani* $y = 0$ *e* $y = 2$
2. *metti il colore delle scatole solo alla prima*
3. *taglia la seconda scatola esattamente dove finisce la prima nella coordinata* z
4. *crea una finestra circolare che trapassi la casetta in tutta profondità appena sotto il tetto.*

Esercizio 12. *Riprendi il file dove disegni i tre vettori* `x`, `y`, `z` *con tre colori diversi (rispettivamente rosso, giallo e blu): disegna ogni vettore come una freccia (unione di cilindro e cono).*
Ora togli dalle tre frecce una sfera di raggio 0.25 con centro nell'origine. Cosa ti aspetti?
Che cosa succede se invece si interseca questa sfera con l'unione delle tre frecce?

Esercizio 13. *Nella CSG un piano viene trattato come un semispazio.*
Modifica il codice di `esempio3.pov` *sostituendo la sfera con un piano passante per l'origine.*
Ora, se v *è il vettore ortogonale al piano che hai scelto, fai la stessa operazione con il piano passante per l'origine individuato dal vettore* $-v$*. Che cosa ti aspetti?*

Esercizio 14. *Operazioni con più di due oggetti.*
Modifica il codice di `esempio3.pov` *aggiungendo un terzo oggetto: una scatola di colore giallo con due vertici opposti in* `<-2,0,-1>` *e* `<2,1.25,1>`*. Prova a vedere cosa succede facendo unione, intersezione e differenza di questi tre oggetti. Cosa cambia se si cambia l'ordine dei tre oggetti?*

Esercizio 15. *Disegna una piramide a base quadrata. Suggerimento: disegna una scatola e tagliala con dei piani obliqui.*

Esercizio 16. *(difficile?) Un problema geometrico!*
Disegna un cubo e taglialo con un piano in vari modi: riesci ad ottenere una sezione quadrata? una rettangolare ma non quadrata? una triangolare? una esagonale?
Per vedere meglio la sezione, disegna il cubo in giallo e la sezione in rosso.