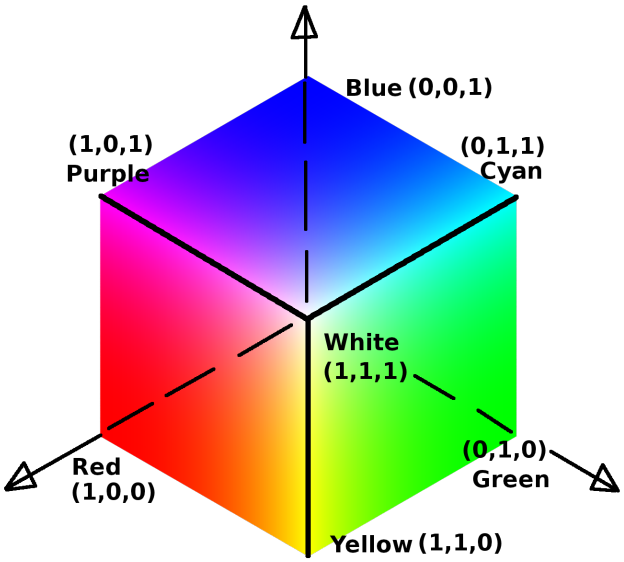
Appunti per POV-Ray - II

Area progetto *Longitudine*, 5D Liceo Torricelli

anno scolastico 2012-13

1 Colori

Tutti i colori vengono ottenuti mescolando rosso (r), verde (g) e blu (b) in quantità diverse: la quantità varia da 0 a 1. Se il valore dato è fuori dall'intervallo $[0, 1]$, viene considerato come l'estremo più vicino (invece di 2 viene considerato 1). I colori principali (quelli saturi) sono raffigurati in figura e si trovano sulle tre facce del cubo lontane dall'origine. Il nero è `<0,0,0>` (nell'origine), il bianco `<1,1,1>`, mentre tra i due, sulla diagonale principale, ci sono le tonalità di grigio.



2 Evitare le ripetizioni

2.1 Definire oggetti

In uno degli esercizi chiedevo di disegnare un orologio, in particolare 12 tacchette uguali, ma in posizioni differenti. Invece di disegnare 12 volte lo stesso oggetto,

possiamo definirne uno con le caratteristiche desiderate, dargli un nome e poi richiamare il nome. La definizione di un oggetto si fa con la parola chiave `#declare` seguita dal nome scelto per l'oggetto, dal segno `=` e dalla definizione dell'oggetto (sfera, scatola, unione di oggetti, ecc.). Finisce con un punto e virgola. Ecco un esempio

```
#declare MiaScatola = box { // definisco l'oggetto con un nome
  0, 1
  texture {
    pigment { color Red }
  }
};                          // chiudo la definizione con ;

object {                    // richiamo un oggetto, e precisamente
  MiaScatola                // quello che ha questo nome
}
```

Nel caso sopra non ha molto senso usare `#declare`, ma se dovessi disegnare dieci scatole uguali, sarebbe una bella scorciatoia. Soprattutto se poi decido di cambiare il colore o le dimensioni di tutte le scatole: dovrò modificare una riga sola invece di dieci. `#declare` può essere vantaggioso anche nel caso di oggetti complessi da ripetere più volte: invece di una scatola potrei mettere per esempio un'unione di oggetti.

Si possono applicare tutte le trasformazioni che conosci già, sia nella definizione dell'oggetto, sia quando esso viene richiamato. Ad esempio:

```
#declare MiaScatola = box {
  0, 1
  texture {
    pigment { color Red }
  }
  scale 2        // trasformazioni qui...
};

object {
  MiaScatola
  scale 2        // ...e qui si sommano!
  translate y
}
```

Si possono dichiarare anche oggetti che non hanno colore, e dare un colore diverso a ogni esemplare che si disegna. Ad esempio:

```
#declare MiaScatola = box {  // definisco l'oggetto
  0, 1
};

object {                   // richiamo l'oggetto
```

```
  MiaScatola
  texture {
    pigment { color Red }  // e lo coloro di rosso
  }
}

object {                   // richiamo di nuovo lo stesso oggetto
  MiaScatola               // ma cambio:
  texture {                //            colore
    pigment { color Blue }
  }
  translate 2*x            //            e posizione
}
```

Esercizio 1. *Disegna un pezzo di muro formato da quattro strati di quattro mattoni l'uno.*
Suggerimento: puoi richiamare un oggetto già definito (un mattone) all'interno di un'altra definizione (uno strato).

Esercizio 2. *Riproduci uno dei pavimenti della figura*

Esercizio 3. *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.*
Suggerimento: fai in modo che il centro dell'orologio sia nell'origine; definisci un oggetto `Tacchetta` *che segni le 12 e creane 12 esemplari, che ruoterai ognuno di un angolo diverso...*

2.2 Cicli *#while*

Riguardiamo l'esempio del muro: un modo per disegnarlo consiste nel dichiarare un oggetto `Mattone`, che sarà un parallelepipedo rosso, e un oggetto `Strato`,

l'unione di quattro mattoni distanziati di poco lungo l'asse `x`. Se ora volessimo allungare gli strati, in modo che ognuno contenga sei mattoni, dovremmo aggiungere altri due oggetti nell'unione. Dato che i mattoni sono distanziati regolarmente, posso usare un'altra scorciatoia: usare un contatore e disegnare tanti mattoni finchè il contatore non arriva al valore scelto (nel nostro caso sei). Ad esempio il codice seguente disegna sei mattoni (vedi il file `esempio5.pov`).

```
#declare Mattone = box {   // definisco l'oggetto
  0, <1, 0.5, 0.5>
  texture {
    pigment { color Red }
  }
};

#declare i = 0;          // definisco e inizializzo il contatore
#while(i < 6)            // ciclo: ripeti la stessa operazione
                         // finchè la condizione è soddisfatta
  object {
    Mattone
    translate i*1.1*x    // posso usare il contatore nel ciclo
                         // ha un valore diverso a ogni passaggio
  }
  #declare i=i+1;        // aumento il valore del contatore
#end                     // chiudo il ciclo
```

Nota: ricordati sempre di incrementare il contatore. Se lo dimentichi, può succedere che il programma non termini (allora usa `Ctrl+c` nel terminale).

Esercizio 4. *Modifica il programma precedente, definendo uno* `Strato` *come unione dei sei mattoni e disegnando quattro strati sovrapposti, con i mattoni sfasati come in un vero muro.*
Modifica il programma in modo che il muro sia composto da sei strati con dieci mattoni l'uno.

Esercizio 5. *E se ora volessi disegnare una stanza, cioè uno spazio a base quadrata circondato da quattro muri come sopra?*
Prova anche a muovere la `camera` *o ruotare la stanza, per verificare che la costruzione sia corretta anche "dietro".*

Esercizio 6. *Riprendi il codice originale e modificalo in modo che i mattoni siano sempre più piccoli quanto più si trovano a destra.*

Esercizio 7. *Cosa farà il seguente programma?*

```
#include "colors.inc"

camera {
```

```
  location <0, 0, -7>
  look_at <0, 0, 0>
}

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

#declare MiaSfera = sphere {
  0, 1
};

#declare i=0;
#while(i<10)
  object {
    MiaSfera
    texture {
      pigment { color rgb <0,0,0.1>*i }
    }
    translate <2*cos(pi/5*i), 2*sin(pi/5*i), 0>
  }
  #declare i=i+1;
#end
```

Modifica il programma in modo che i colori delle sfere cambino dal rosso al giallo, passando per l'arancione e in modo che le sfere siano distribuite su una semicirconferenza.

Esercizio 8. *Che cosa farà questo programma? Suggerimento: ci sono due errori. Correggili prima di compilare e ricordati di inserire le istruzioni mancanti.*

```
#declare MiaSfera = sphere {
  0, 1
  texture {
    pigment { color rgb <1,0,0>*(1-i/10) }
  }
};

#declare i=0;
#while(i<10)
  object {
    MiaSfera
    translate i
  }
#end
```

I cicli possono essere uno dentro l'altro, come nel file `esempio6.pov`.

Esercizio 9. *Modifica il file* `esempio6.pov` *in modo da costruire un cubo fatto di 27 sfere.*

2.3 Cicli *#if*

Oltre a `#while`, c'è un'altra istruzione condizionale: `#if`. Funziona allo stesso modo, ma è più semplice, perché non richiede contatori:

```
#if(CONDIZIONE)
  ISTRUZIONI da eseguire se vale la CONDIZIONE
#end
```

oppure

```
#if(CONDIZIONE)
  ISTRUZIONI da eseguire se vale la CONDIZIONE
#else
  ISTRUZIONI da eseguire se NON vale la CONDIZIONE
#end
```

Condizioni e cicli possono essere uno dentro l'altro. Ad esempio

```
#declare i=0;
#while(i < 10)
  sphere { 0, .5
    #if(mod(i,2) = 0)                    // se i diviso 2 dà resto 0 cioè
                                         // se i è pari
      texture { pigment { color Red } }
    #else                                // se i è dispari
      texture { pigment { color Blue } }
    #end
    translate i*y
  }
  #declare i=i+1;
#end
```

Esercizio 10. *Modifica il programma precedente in modo che le sfere siano di tre colori: rossa, blu, gialla, rossa, blu, ecc.*
Suggerimento: usa due `if` *uno dentro l'altro.*

Se ci sono più di due alternative, si può usare la parola chiave `#switch`, che funziona in questo modo:

```
#declare i=0;
#while(i < 10)
  sphere { 0, .5
    #switch(mod(i,5))                    // se i diviso 5 dà resto:
      #case(0)                           // resto 0
        texture { pigment { color Red } }
      #break
      #case(0)                           // resto 1
        texture { pigment { color Blue } }
```

```
      #break
      #case(0)                          // resto 2
        texture { pigment { color Yellow } }
      #break
      #else                             // in tutti gli altri casi
        texture { pigment { color Green } }
      #break
    #end
    translate i*y
  }
  #declare i=i+1;
#end
```

2.4 Le *#macro*

Si possono definire funzioni con parametri, che restituiscono un valore. Ad esempio

```
#macro Quadrato(a)  // Quadrato è il nome della macro
                    // a è il nome del parametro
  pow(a,2)          // restituisce questo valore
#end

sphere {
  0, Quadrato(0.5)  // chiamo la macro
}
```

Le macro possono avere un qualsiasi numero di parametri (anche zero). Possono non avere output.

```
#declare Valore = 0;

#macro Somma(a,b)           // Somma è il nome della macro
                            // a, b sono i nomi dei parametri
  #declare Valore = a+b;    // non restituisce alcun valore
#end

Somma(0.5, 1)               // chiamo la macro

sphere {
  0, Valore
}
```

Un esempio meno stupido:

```
#macro Calcoli(Numero)          // Calcoli è il nome della macro
                                // Numero è il parametro passato
  #local Result = 1;            // definisco una variabile
```

```
  #local Ind = 2;               // definisco una variabile contatore
  #while(Ind <= Numero)         // ciclo
    #local Result = Result*Ind; // aggiorno valore
    #local Ind = Ind+1;         // aggiorno contatore
  #end
  Result                        // la macro ritorna questo valore
#end                            // fine della macro

#declare Valore = Calcoli(5);   // chiamo la macro e calcolo

sphere {
  0, Valore
}

// #declare Valore = Result;    // errore, Result era locale
                                // ed è stato cancellato
```

`#local` è simile a `#declare`; l'unica differenza è che una variabile definita tramite `#local` all'interno di una macro viene cancellata quando la macro termina. Per questo nel codice precedente la riga commentata non è lecita.

Esercizio 11. *Cosa fa la macro* `Calcoli`*? Cosa fa il programma?*

2.5 Di nuovo *#declare*

La parola chiave `declare` può essere usata anche per definire delle trasformazioni o delle `texture` o altro.

```
#declare MiaScatola = box {               // dichiaro oggetto
  0, 1
};

#declare MioColore = texture {            // dichiaro texture
  pigment { color Red }
};

#declare MiaTrasformazione = transform {  // dichiaro trasformazione
  rotate 30*y
  scale 3
};

object {
  MiaScatola
  texture { MioColore }
  transform MiaTrasformazione
  translate 2*x
}
```

```
object {
  MiaScatola
  texture { MioColore }
  translate -2*x
}
```

Si possono usare macro e definizioni nell'ordine voluto.

```
#macro RuotaEScala(angolo,scala)
  rotate angolo*y
  scale scala
#end

#declare MiaScatola = box {
  0, 1
  texture {
    pigment { color Red }
  }
};

#declare MiaTrasformazione = transform {
  RuotaEScala(60,2)
};

object {
  MiaScatola
  RuotaEScala(30,3)
  translate 2*x
}

object {
  MiaScatola
  transform MiaTrasformazione
  translate -2*x
}
```

Esercizio 12. *Disegna una circonferenza composta da 20 piccole sfere equidistanti.*
Disegna un'elica composta da 20 piccole sfere. Suggerimento: un'elica è come una circonferenza, con la terza coordinata crescente.

Esercizio 13. *Disegna una piramide a base pentagonale. Suggerimento: disegna una scatola. Definisci un piano obliquo P_0; ottieni i piani P_1, P_2, P_3 e P_4 ruotando P_0 della frazione di angolo giro voluta; taglia la scatola con tutti i piani.*

Esercizio 14. *Modifica il programma dell'esercizio precedente in modo da poter scegliere quanti lati ha la base. Suggerimento: disegna una scatola. Definisci un piano obliquo P_0; con il ciclo* `while` *fai successive sottrazioni dalla scatola, ruotando ogni volta il piano dell'angolo necessario.*

3 File *.ini* e animazioni

La parola chiave `clock` indica una variabile "tempo", che viene usata per creare animazioni, definendo movimenti dipendenti dal tempo. Per ogni istante (una sottounità di tempo) viene generata un'immagine. Guardando le immagini (dette frames) in sequenza si vede l'animazione, proprio come nei libri animati. Con circa 25 frames al secondo, non percepiamo più le immagini staccate, ma il movimento ci sembra continuo.

Il valore del `clock` viene gestito automaticamente dal compilatore, ma la sua inizializzazione deve essere fatta in un file particolare, con estensione `ini`.

Quindi per creare animazioni servono

- un file `pov` che descrive la scena. In esso gli oggetti e i movimenti possono dipendere dalla variabile `clock`.
- un file `ini`, che contiene solo istruzioni per il compilatore: quale file `pov` leggere, che nome dare ai file di output, quanti file di output generare, che valori assume il tempo, ecc.

Ecco un esempio: il file `esempio7.pov` contiene degli oggetti con caratteristiche dipendenti da `clock`.

```
#include "colors.inc"

camera {
  location <1, 1.5, -3>
  look_at <0.5, 1, 0>
}

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

box {
  <0,0,0>, <1,1,1>
  texture { pigment { color Yellow } }
  translate clock*x                    // dipende dal tempo
}

box {
  <0,0,0>, <1,1,1>
  texture { pigment {
      color rgb <1-clock, 0, clock>    // dipende dal tempo
```

```
    }
  }
  translate -x
}
```

Se si compila questo file, il `clock` viene interpretato con valore `0` e il file generato si chiama `esempio7.png`.

Ora guardiamo il file `esempio7.ini`.

```
; nei file ini i commenti si fanno così

Input_File_Name = esempio7.pov  ; nome file di input
Output_File_Name = cubi         ; nome file di output
Width = 800                     ; larghezza immagine output
Height = 600                    ; altezza immagine output

Initial_Clock = 0               ; clock parte da questo valore
Final_Clock = 1                 ; e arriva fino a questo

Initial_Frame = 0               ; prima immagine generata
Final_Frame = 10                ; ultima immagine generata
```

Esso richiama il file `esempio7.pov` e genera undici file di output con nome `cubi`, il numero sequenziale e l'estensione `png`. Il tempo varia da 0 a 1, i frames da 0 a 10, quindi vengono generati

- un file `cubi00.png` in cui il `clock` vale `0`
- un file `cubi01.png` in cui il `clock` vale `0.1`
- un file `cubi02.png` in cui il `clock` vale `0.2`
- ...
- un file `cubi10.png` in cui il `clock` vale `1`

Altre opzioni che possono essere specificate nel file `ini`.

```
Subset_Start_Frame = 0    ; genera solo questi frames
Subset_End_Frame = 1      ; (tra Initial_Frame e Final_Frame)

All_Console = false       ; non mandare i messaggi sul terminale
All_File = true           ; scrivi tutti i messaggi su un file

Output_File_Type = N      ; tipo di output (png)
-D                        ; non far vedere immagine mentre compila
```

Esercizio 15. *Genera 50 frames usando lo stesso intervallo di tempo.*
Fai variare il tempo da 0.5 a 2.
Compila solo i frames dal 20 al 30.
Fai diventare il primo cubo trasparente man mano che si sposta a destra.

Esercizio 16. *Riprendi l'orologio che hai disegnato. Crea dei frames (almeno 60) in cui le lancette girano in modo realistico dalle ore 1 alle ore 2.*

4 Testo

Il testo si inserisce così

```
text {
  ttf "timrom.ttf" "POV-RAY" 1, <0,0,0>
  pigment { Red }
}
```

L'oggetto si chiama `text`, `ttf` è il tipo di carattere (questa parola chiave deve esserci sempre), la parte seguente tra virgolette è il nome del file del carattere da usare, la seconda stringa tra virgolette è il testo da scrivere, il numero `1` è lo spessore della scritta (che è tridimensionale), il vettore indica lo spazio tra le lettere nelle tre direzioni.

Alcuni file con i caratteri si trovano nella cartella dov'è installato povray: `povray-3.6/include/`. Altri possono essere scaricati gratuitamente da internet. Per usarli basta metterli nella stessa cartella dove si trova il file `pov`.

Trovi un esempio nel file `esempio8.pov`.

Esercizio 17. *Scrivi un file con la scritta "Liceo Torricelli" che sporge da un parallelepipedo.*
Come faresti per incidere la scritta nel solido?