Corso di Perfezionamento 2007

Laboratorio: Open Source

POV – RAY: INTRODUZIONE ALL'USO E PROPOSTE DIDATTICHE.

Relazione finale di Giulio Vissani.

1. Introduzione

Nel corso della lezione del prof. Domenico Luminati su esempi di utilizzo di software libero nella scuola si è fatto cenno ad un programma per la visualizzazione di oggetti tridimensionali. La qualità delle immagini mostrate era sorprendente, mentre, a detta del relatore, lo sforzo necessario per realizzarle modesto.

A volte ho sentito la necessità a scopo didattico di avere illustrazioni per lo meno comprensibili di oggetti nello spazio. Personalmente ho molte difficoltà a realizzarle a mano, così ho trovato interessanti le possibilità offerte da questa applicazione ed ho sfruttato il laboratorio del corso di perfezionamento per sperimentarle direttamente. In effetti l'uso del programma, possedendo un minimo di familiarità con la geometria analitica ed un linguaggio per computer, non presenta particolari difficoltà e le soddisfazioni ripagano del tempo dedicato ad esso. Nella speranza di facilitare il percorso a chi volesse tentare lo stesso esperimento ho cercato di dare a questa relazione il carattere di un tutorial allegando in appendice i listati commentati di un paio di prove fatte.

Utilizzando POV-Ray ho avuto l'impressione che le sue potenzialità potessero andare oltre a quelle di strumento di aiuto all'insegnante nella preparazione di illustrazioni. Penso infatti che questo programma si presti ad essere proposto direttamente agli studenti nel laboratorio di informatica. Esso è infatti corredato degli elementi essenziali di un linguaggio di programmazione e la possibilità di ottenere immagini di qualità potrebbe fornire una motivazione all'apprendimento. Si potrebbero ideare attività sia direttamente legate alla programmazione che ad una sorta di laboratorio di geometria nello spazio. Dal punto di vista didattico anche l'interfaccia testuale e la relativa povertà di elementi di base, primitive grafiche, disponibili per modellare gli oggetti possono essere vantaggiose. Chi lavora con POV-Ray non si limita a puntare e cliccare. È costretto a riflettere su quanto sta facendo e, per arrivare a forme non elementari, deve progettare una scomposizione in termini delle primitive disponibili ed indicare la collocazione spaziale delle varie parti. Ho provato a verificare direttamente questa idea ed al resoconto della microsperimentazione condotta con una classe III di liceo scientifico PNI è dedicato l'ultimo paragrafo.

2. Cos'è POV Ray e dove reperirlo

L'applicativo *Persistence of Vision Ray-Tracer*, o più brevemente POV–Ray, è il frutto del lavoro di un gruppo di appassionati, *POV Team*, e può essere liberamente prelevato, nel rispetto delle condizioni per l'utilizzo che lo accompagnano, dal sito ufficiale del team:

www.povray.org

Si tratta di un sito molto ricco di materiali che vanno dalla galleria di immagini realizzate con POV–Ray, ai link verso altri siti di appassionati, alle raccolte di esempi e tutorial.

Il programma, di cui è disponibile il sorgente, si trova già compilato per diverse piattaforme. Personalmente ho utilizzato la versione 3.6 per Windows che, una volta scaricata, provvede automaticamente all'istallazione. Non esiste localizzazione in italiano, ma non ho incontrato particolari difficoltà nel seguire le indicazioni in lingua inglese. Per muovere i primi passi si può seguire il tutorial che si trova all'interno dell'aiuto in linea al quale si accede con un clic sul bottone contrassegnato da un punto interrogativo verde o dall'apposita voce di menu.

Un breve cenno sui principi di funzionamento di questi programmi in grado di creare immagini tridimensionali di qualità fotografica utilizzando la tecnica del ray-tracing. Immaginiamo per un momento che lo schermo del computer sia una finestra al di là della quale si trovano gli oggetti da rappresentare. Dividiamo lo schermo in una griglia e, per ciascuna maglia (pixel), consideriamo un raggio che parte da un punto P, posizione dell'osservatore, attraversa il centro della maglia e prosegue finché non incontra la superficie di uno degli oggetti o lo sfondo. A seconda delle caratteristiche del materiale dell'oggetto il raggio potrà essere solo riflesso o in parte riflesso ed in parte trasmesso. Si seguono questi nuovi raggi secondari annotando le successive riflessioni/trasmissioni che subiscono per un numero prefissato di volte o finché non superano una certa distanza preassegnata. Una volta terminato il processo si assegna al pixel un colore calcolato in base alle vicende subite dal raggio, alle leggi dell'ottica ed alle condizioni di illuminazione.
L'approccio utilizzato nel ray-tracing è dunque quello di seguire a ritroso il percorso dei raggi luminosi che a partire dalle superfici illuminate degli oggetti giungono fino all'osservatore. Il processo di resa grafica dell'immagine non è affatto veloce, ma la qualità che si ottiene è elevata ed include riflessioni, ombreggiature, prospettiva, effetti atmosferici ed molto altro ancora.

3. Generare la prima immagine.

La costruzione di un'immagine richiede due fasi. Nella prima, utilizzando il linguaggio di POV-Ray, in un file di testo si forniscono le informazioni relative alla descrizione e collocazione sia degli oggetti che devono comparire che delle sorgenti di illuminazione. Occorre anche includere le caratteristiche della macchina fotografica virtuale (posizione, orientamento, ottica…) con la quale si riprende la scena. Per la scrittura si può utilizzare un editor qualsiasi, ma quello interno alla versione Windows presenta molte funzioni d'aiuto alla redazione, quali ad esempio l'indentazione delle righe e l'evidenziazione in colori diversi di parole chiave del linguaggio, dei valori numerici, dei commenti. Inoltre il riutilizzo di parti di listati mediante copia ed incolla è agevolato dalla possibilità di aprire contemporaneamente più documenti. Questi vengono sistemati in schede contrassegnate in alto da un'etichetta con il nome del file. Per accedere ad una scheda basta fare clic sull'etichetta.

Nella seconda fase della costruzione si lancia il programma di resa grafica (rendering) che utilizza le informazioni fornite nel file di testo per generare automaticamente un'immagine della scena simile a quella che si otterrebbe scattando una foto ad oggetti reali.

Penso che una delle vie migliori per apprendere un nuovo linguaggio di programmazione sia quella di studiare dei listati prodotti da altri. POV–Ray facilita il compito in due modi. Durante l'istallazione crea una sottocartella denominata scenes nella quale si trovano esempi di tutti i livelli. Inoltre, una volta avviato il programma ed aperta una pagina nuova nell'editor, è possibile inserire tutto il codice che serve per creare un'immagine elementare semplicemente selezionando la voce di menu:

Insert → Scene Templates → Basic Scene

Trovo comoda la possibilità di avere a disposizione uno scheletro di programma da cui partire e l'ho sfruttata per tutte le prove fatte. Anche la possibilità di avere evidenziate in colore le parole chiave

del linguaggio è utile per evitare errori durante l'immissione del codice. Per abilitare questa opzione occorre salvare il file indicando come estensione ".pov", aprire una finestra di dialogo con

Editor → Editor Preferences → Language/Tabs

e scegliere POV-Ray nella casella a discesa Language.

Per entrare nei dettagli del linguaggio utilizziamo un esempio concreto. Quella che segue è una versione ridotta all'osso del listato inserito da POV-Ray come Basic Scene. Ho eliminato i commenti dell'intestazione iniziale ed alcune istruzioni per il momento superflue.

```
#include "colors.inc"

global_settings { assumed_gamma 1.0 }

// ----------------------------------------

camera {
  location  <0.0, 0.5, -4.0>
  look_at   <0.0, 0.0,  0.0>
}

sky_sphere {
  pigment {
    gradient y
    color_map {
      [0.0 rgb <0.6,0.7,1.0>]
      [0.7 rgb <0.0,0.1,0.8>]
    }
  }
}

light_source {
  <0, 0, 0>             // light's position (translated below)
  color rgb <1, 1, 1>   // light's color
  translate <-30, 30, -30>
}

// ----------------------------------------

plane {
  y, -1
  pigment { color rgb <0.7,0.5,0.3> }
}

sphere {
  0.0, 1
  pigment { color rgb <1.0,0.0,0.0> }
}
```

3.1 La fotocamera

Passiamo in rassegna il significato delle diverse istruzioni, anche se probabilmente il significato di gran parte di esse è facilmente intuibile. Tralasceremo di illustrare `global_settings` (per la quale rinviamo al manuale in linea) limitandoci a dire che si tratta di un'istruzione necessaria a garantire l'uniformità di prestazioni su diverse piattaforme.

```
camera {
  location  <0.0, 0.5, -4.0>
  look_at   <0.0, 0.0,  0.0>
```

```
}
```

All'interno dell'istruzione `camera{…}` vengono specificate le caratteristiche della fotocamera virtuale a partire da quelle essenziali: posizione e direzione di puntamento. Notare che la posizione è preceduta dalla parola chiave `location` mentre il punto verso il quale è rivolto l'obiettivo, ovvero il centro di attenzione dell'immagine, è preceduto da `look_at`. Il tutto è racchiuso in parentesi graffe. Ricordiamo a questo proposito che se il computer è equipaggiato con una tastiera italiana le parentesi graffe si ottengono con le combinazioni di tasti:

<Maiusc> + <Alt Gr> + < [> per avere {
<Maiusc> + <Alt Gr> + <] > per avere }

Per indicare le posizioni degli oggetti nel mondo di POV-Ray si utilizza il seguente sistema di coordinate cartesiane 3D:

- asse x: direzione orizzontale, orientato verso destra
- asse y: direzione verticale, orientato verso l'alto
- asse z: direzione ortogonale allo schermo del monitor, orientato verso l'interno

Notare che, a differenza dei sistemi comunemente usati in matematica e fisica, si tratta di un sistema sinistrorso: l'asse x ha la direzione del pollice della mano sinistra, l'asse y quella dell'indice, l'asse z quella del medio. Occorre tenerlo presente anche per determinare il senso positivo delle rotazioni. Immaginando di afferrare con la sinistra l'asse di rotazione in modo che il pollice punti nella direzione positiva dell'asse stesso si avrà che la direzione di rotazione positiva è quella seguita dalle dita che si chiudono intorno all'asse.

Quando si specificano le coordinate di un punto occorre racchiuderle tra parentesi angolose separandole con virgole. Ad esempio: `<0.5, -3, 2.1>`. La stessa notazione si usa per indicare un vettore. POV-Ray accetta che un vettore venga specificato in termini dei versori degli assi:

```
<a, b, c> = a*x + b*y + c*z.
```

All'interno dell'istruzione `camera{…}` possono essere indicate anche altre caratteristiche elencate nel manuale in linea. Qui ne indichiamo solo due. La prima riguarda l'apertura angolare dell'obiettivo che può essere specificata mediante la parola chiave `angle` seguita dal valore in gradi. Per ridurre le deformazioni prospettiche può essere utile allontanare il punto di ripresa dal soggetto e stringere l'angolo di ripresa. La seconda caratteristica è il tipo di proiezione utilizzata nella costruzione dell'immagine. Il tipo predefinito è quello prospettico, ma si può avere una proiezione ortogonale inserendo la parola chiave `orthographic` al primo posto nelle graffe.

3.2 Luci e colori

Non è possibile vedere alcun oggetto se questo non viene illuminato e l'istruzione seguente serve appunto per introdurre una sorgente di illuminazione:

```
light_source {
  <0, 0, 0>             // posizione illuminazione (traslata sotto)
  color rgb <1, 1, 1>   // colore dell'illuminazione
  translate <-30, 30, -30>
}
```

Il primo vettore in `light_source` specifica la collocazione della sorgente che può essere pensata come un punto invisibile che emette luce del colore indicato nella riga successiva.

Per specificare il colore si fa direttamente riferimento alle modalità di funzionamento dei monitor.

Questi infatti generano l'intera gamma dei colori di un'immagine miscelando in modo opportuno tre colori di base: rosso, verde e blu (in inglese: red, green, blue). Le tre componenti del vettore che compare dopo `color rgb`, ciascuna delle quali può assumere un valore compreso nell'intervallo [0, 1], indicano la frazione dei tre fondamentali da usare per ottenere la tonalità voluta. Quella che compare nell'esempio è una luce bianca, visto che per ciascuna componente viene usato l'intero. Per una luce rossa avremmo usato solo la prima componente, <1, 0, 0>, per una gialla una miscela in parti uguali di rosso e verde, <1, 1, 0>, per una arancione la quantità di verde deve essere ridotta, <1, 0.5, 0>.

Questa scomposizione dei colori nei tre componenti fondamentali rgb viene usata in tutti i programmi di grafica al computer, a partire dai più elementari quali ad esempio Paint, fornito a corredo di Windows. Se occorre sapere a quale terna corrisponde una certa tinta basta aprire la finestra di dialogo per la scelta dei colori presente in tali programmi, selezionare visivamente il colore voluto ed annotarsi i tre valori. Può essere necessaria un'operazione di normalizzazione perché in molti casi i valori di ciascuna componente variano su intervalli diversi da quello adottato da POV-Ray. Ad esempio, Paint utilizza l'intervallo 0-255, pertanto un verde acqua (0, 244, 220) dovrà essere indicato <0, 244/255, 220/255>.

Esiste anche un'altra possibilità per specificare i colori ed è strettamente legata all'istruzione `#include` che compare nel listato. Questo comando permette l'importazione e l'utilizzo del contenuto di un file con estensione `.inc` che solitamente contiene elenchi di definizioni. Questi file si trovano in una cartella denominata include a sua volta collocata nella cartella contenente POV-Ray. In particolare con `colors.inc` si carica un elenco di colori predefiniti ed è possibile far riferimento ad essi con dei nomi (inglesi) al posto delle terne rgb. Ecco un esempio:

```
pigment { color Yellow }.
```

Per scorrere la lista dei nomi disponibili il file può essere aperto, ed eventualmente modificato, all'interno dell'editor stesso

L'ultima istruzione che compare nella definizione della sorgente luminosa, `translate`, serve a spostarla dalla posizione in cui è stata creata (l'origine) ad un punto posto 30 unità più a sinistra ed in alto ed altrettante verso l'esterno dello schermo (ovvero dalla parte di chi osserva il monitor). Quella di creare inizialmente un oggetto in corrispondenza dell'origine e spostarlo altrove solo dopo averne specificato tutti gli attributi è una pratica che si incontra spesso scorrendo i listati.

La sorgente utilizzata è di tipo puntiforme, la più semplice tra quelle messe a disposizione da POV-Ray. È possibile inserire più sorgenti nella stessa scena ottenendo illuminazioni più realistiche, ma il tempo per la resa grafica cresce sensibilmente. Conviene allora costruire gli oggetti con una sola fonte luminosa riservandosi di aggiungere le altre solo una volta che siano stati messi a punto tutti gli oggetti.

Le scritte precedute dalla doppia barra // sono semplicemente dei commenti che chiariscono lo scopo delle istruzioni ma vengono completamente ignorati durante l'esecuzione del programma.

3.3 Oggetti

Una volta specificate luci e fotocamera si passa alla descrizione dei singoli oggetti che andranno a popolare la scena. Nel nostro esempio compaiono una sfera, un piano d'appoggio ed un fondale che simula il cielo. Si tratta di tre forme geometriche, o meglio primitive grafiche, che sono parte integrante del linguaggio e non devono pertanto essere definite dal programmatore. Occorre soltanto indicare al programma gli attributi relativi a dimensioni, posizione e descrizione del

materiale di cui è fatto un dato oggetto.

Per fissare la disposizione del piano si specifica il suo vettore normale e di quanto dovrà essere mosso a partire dall'origine lungo la normale stessa.

```
plane {
  y, -1
  pigment { color rgb <0.7,0.5,0.3> }
}
```

Il piano del nostro esempio ha come normale il versore y dell'asse verticale (l'uso di y è una scorciatoia consentita per evitare di scrivere per esteso `<0, 1, 0>`), ed è spostato di una unità verso il basso.

Le caratteristiche della sfera vengono invece descritte attraverso un vettore che indica il centro ed un valore per il raggio.

```
sphere {
  0.0, 1
  pigment { color rgb <1.0,0.0,0.0> }
}
```

Notare l'uso di un'altra scorciatoia per indicare il centro della sfera. Quando POV-Ray trova un numero al posto di un vettore non indica un errore, ma "promuove" il numero a vettore considerando tutte le componenti uguali al numero stesso. Così se indichiamo `0.0` come posizione del centro verrà considerato il punto `<0.0, 0.0, 0.0>` mentre indicare 2.1 equivale a `<2.1, 2.1, 2.1>`.

Per quanto riguarda la dichiarazione delle caratteristiche del materiale, nell'esempio ci si riduce all'essenziale: il colore. In realtà le possibilità offerte sono enormi, ma non sempre immediatamente padroneggiabili. Si possono specificare proprietà della superficie dell'oggetto che vanno dal grado di levigatezza, alla riflettività, durezza, capacità di diffondere la luce incidente … Fortunatamente si ottengono immagini discrete anche dopo aver letto solo le prime parti dei paragrafi dell'aiuto in linea ad esse dedicate.

Con `sky_sphere` si definisce lo sfondo dell'immagine. Per questo oggetto si ha un uso più raffinato del colore in quanto al posto di una tinta uniforme viene definita una sfumatura che parte da un celeste chiaro per arrivare, muovendosi lungo l'asse verticale, ad un azzurro carico.

3.4 Resa grafica dell'immagine

Una volta terminata l'immissione del testo resta solo da lanciare l'esecuzione della resa grafica. Per far questo basta fare clic sul bottone Run (caratterizzato dall'icona di un omino in corsa) presente sulla barra dell'editor. Volendo si possono selezionare le dimensioni dell'immagine da produrre nella finestra a scorrimento posta in alto a sinistra, immediatamente sotto la barra dei bottoni.

Per evitare tempi di elaborazione troppo lunghi conviene, almeno nella fase di progetto e sviluppo iniziale, scegliere dimensioni contenute, tipo (512x384, No AA). La dicitura No AA , ovvero no antialias, sta ad indicare che durante l'elaborazione dell'immagine non dovranno essere adottati particolari accorgimenti per correggere imperfezioni quali le scalettature sui bordi obliqui degli oggetti. Quando la descrizione della scena è completa ed i test hanno dato esito positivo si effettua il

rendering definitivo dell'immagine con dimensioni e livelli di antialias adeguati.

L'immagine prodotta viene automaticamente salvata in un file formato bitmap (estensione .bmp) avente lo stesso nome del file .pov contenete il testo con la descrizione della scena. Volendo si possono scegliere altri formati per l'immagine.

Non sempre tutto funziona al primo colpo. Un errore di battitura o una parentesi dimenticata sono sempre in agguato per impedire un funzionamento corretto del programma. In questi casi POV-Ray arresta l'elaborazione e segnala nella scheda etichettata Messages la posizione dell'errore unitamente ad alcune informazioni per facilitarne l'individuazione. Per evitare spiacevoli sorprese non è consigliabile aspettare di aver digitato l'intera descrizione della scena prima di eseguire il rendering. Conviene lanciare il programma dopo l'aggiunta di ogni nuovo oggetto, anche per rendersi conto di posizione ed illuminazione. Lavorando in bassa risoluzione la resa grafica è veloce ed evita le perdite di tempo ben più consistenti che si hanno cercando di rimediare ad errori nascosti in lunghi file. Per facilitare la fase di sviluppo POV-Ray permette anche l'inserimento di istruzioni per la stampa di messaggi durante l'elaborazione dell'immagine mediante le direttive: #warning, #error, #debug.

4. Oggetti elementari

I mattoni di base con i quali costruire gran parte degli oggetti che compaiono nelle immagini create con POV-Ray sono i solidi classici della geometria dello spazio: sfera, piano, parallelepipedo, tronco di cono, cilindro, toro. Si tratta di una scelta differente rispetto a quella adottata in gran parte dei programmi di modellazione tridimensionale. In questi solitamente i volumi vengono definiti in termini di reticolati di poligoni, solitamente triangoli o rettangoli.

La possibilità di usare reti di triangoli, o `mesh`, esiste anche in POV-Ray, ma non viene enfatizzata particolarmente. Il motivo è semplice: per descrivere un singolo oggetto possono servire centinaia o addirittura migliaia di poligoni ed è impensabile codificarne a mano le posizioni dei singoli vertici. Personalmente ho trovato interessante l'uso delle `mesh` per rappresentare superfici matematiche nello spazio. In questo caso infatti, scrivendo un piccolo frammento di codice racchiuso in un ciclo, è possibile delegare il calcolo delle coordinate dei vertici e la costruzione della rete al programma stesso.

Abbiamo già visto nell'esempio precedente come introdurre piani e sfere nella descrizione dell'immagine da realizzare. Esaminiamo ora per sommi capi come utilizzare le altre primitive.

Un parallelepipedo viene descritto indicando le coordinate di due vertici posti alle estremità opposte di una diagonale. Generalmente, ma non è obbligatorio, i valori delle coordinate del primo vertice sono minori di quelli del secondo. Si assume inoltre che le facce del parallelepipedo siano parallele ai piani coordinati.

```
box {
    <x1, y1, z1>,        // Angolo vicino in basso a sinistra
    <x2, y2, z2>         // Angolo lontano in alto a destra
    pigment {...}        // Descrizione del materiale
}
```

Il cono (o meglio il tronco di cono) viene descritto indicando le posizioni dei centri e valori dei

raggi dei cerchi che chiudono ciascuna delle due estremità. Per avere un semicono uno dei due raggi dovrà essere uguale a zero. Per avere un cono cavo occorre aggiungere la parola chiave `open` dopo il secondo raggio.

```
cone {
    < x1, y1, z1>, r1     // Centro e raggio di un'estremità
    < x2, y2, z2>, r2     // Centro e raggio dell'altra estremità
    pigment {...}         // Descrizione del materiale
  }
```

Per avere un cilindro dovremo specificare i centri delle due basi ed il raggio. Anche in questo caso si può ottenere una figura cava aggiungendo `open` dopo il valore del raggio:

```
cylinder {
    < x1, y1, z1>,        // Centro di una delle due basi
    < x2, y2, z2>,        // Centro dell'altra base
    r                     // Raggio
    pigment {...}         // Descrizione del materiale
}
```

Per comprendere il significato dei parametri che compaiono in un toro basta pensare ad esso come al volume spazzato da un cerchio di raggio r quando il suo centro viene mosso lungo una circonferenza di raggio R centrata nell'origine e giacente sul piano xz ortogonale a quello del cerchio.

```
torus {
      R,          // Raggio circonferenza
      r,          // Raggio cerchio
      pigment {...}
}
```

5. Trasformazioni

Gli oggetti appena elencati possono essere modificati specificando all'interno delle graffe che contengono i loro attributi anche delle trasformazioni. Le principali sono: traslazione, cambiamento di scala e rotazione. Trasformazioni più complesse si ottengono specificando gli elementi di una matrice.

Per spostare un oggetto si utilizza la parola chiave `translate` seguita dall'indicazione di un vettore che specifica lo spostamento. Gli spostamenti sono sempre relativi alla posizione corrente dell'oggetto. Ecco un esempio:

```
sphere { <10, 10, 10>, 1
  pigment {...}
  translate <-5, 2, 1>  //dopo lo spostamento il centro si troverà in <5,12,11>
}
```

Le dimensioni di un oggetto possono essere modificate con un cambiamento di scala indicato con la parola chiave `scale` seguita da un vettore che ha come componenti i valori dei cambiamenti da eseguire in corrispondenza di ciascun asse coordinato. La possibilità di specificare valori diversi per i tre assi permette la deformazione di un oggetto. Ad esempio vediamo come ottenere un'ellissoide a partire da una sfera:

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

```
  scale <2,1,0.5>
}
```

Per cambiare l'orientamento di un oggetto si deve aggiungere la parola chiave `rotate` seguita da un vettore. Le tre componenti del vettore indicano i gradi di rotazione intorno a ciascuno degli assi. Tenere presente che le rotazioni vengono fatte intorno agli assi coordinati, pertanto se un oggetto si trova ad una certa distanza dall'asse di rotazione subirà oltre alla rotazione anche un moto orbitale. In altre parole il risultato che si ottiene prima ruotando un oggetto è poi traslandolo non è uguale a quello ottenuto prima traslando e poi ruotando.

L'esempio indica come ottenere un parallelepipedo con facce non parallele ai piani coordinati:

```
box {
    <-1, 0,   -1>,  // Angolo vicino in basso a sinistra
    < 1, 0.5,  3>   // Angolo lontano in alto a destra
    pigment {...}
    rotate <30, 0, 10>
}
```

Nei listati di esempi a corredo del programma compare spesso una notazione compatta per indicare rotazioni intorno ad uno solo degli assi coordinati che utilizza i versori `x, y` e `z`. Una rotazione di 30 gradi intorno all'asse y può allora essere indicata con `rotate 30*y` al posto di `rotate <0, 30, 0>`.

6. Oggetti più complessi

Se la gamma di oggetti rappresentabili fosse ristretta a quanto descritto finora probabilmente l'interesse per POV-Ray si esaurirebbe abbastanza presto. In realtà le forme realizzabili sono praticamente infinite grazie alla CSG, o Contructive Solid Geometry. I solidi di base possono essere combinati tra di loro per creare nuovi oggetti con operazioni simili a quelle dell'insiemistica. Due o più forme possono essere intersecate tra loro, unite, fuse, o sottratte. Non ci sono limitazioni né al numero di operazioni compiute su un oggetto, né a quello degli oggetti coinvolti.

`union{…}`	L'unione viene utilizzata per raggruppare insieme due o più oggetti. In questo modo è possibile applicare degli attributi (ad es. il colore) o una trasformazione (traslazione, rotazione…) a tutti gli oggetti del gruppo con un solo comando
`intersection{…}`	L'intersezione crea un nuovo oggetto formato dalle regioni comuni degli oggetti che vengono intersecati.
`difference{…}`	Con questa operazione vengono sottratte all'oggetto di partenza tutte le regioni che ha in comune con gli altri oggetti combinati con esso.
`merge{…}`	La fusione provvede, come l'unione, al raggruppamento di oggetti ed in aggiunta elimina le superfici interne. Risulta utile nella costruzione di oggetti trasparenti.

Inoltre POV-Ray dispone degli elementi essenziali di un linguaggio di programmazione grazie al quale creare algoritmi per definire nuovi oggetti, assemblarli in strutture più complesse e disporli nello spazio. Supponiamo ad esempio di dover rappresentare un vettore unitario. Per avere un punto di partenza lanciamo POV-Ray, creiamo un nuovo documento, inseriamo il listato della scena base e cancelliamo le righe finali relative alla sfera. Dobbiamo ora comunicare a POV-Ray una descrizione del nuovo oggetto che vogliamo costruire. Viste le primitive di cui disponiamo rappresenteremo il vettore come unione di un cilindro, corpo del vettore, ed un cono, la punta.

Digitiamo la seguente direttiva dal significato facilmente intuibile:

```
#declare versore = union{
   cylinder{<0,0,0>,<1,0,0>,0.03}
   cone{<1,0,0>,0.07,<1.2,0,0>,0}
   pigment{color rgb <1, 0, 0>}
}
```

L'oggetto denominato "versore" che abbiamo appena definito può essere utilizzato come uno qualsiasi di quelli predefiniti nel linguaggio. Unica accortezza da usare al momento in cui si vuole inserire un elemento che si è definito in precedenza è quella di inserire il nome che identifica l'oggetto unitamente ad eventuali trasformazioni da applicare all'interno di `object{…}`. Così, se ad esempio, vogliamo che il versore sia applicato nel punto (1; 2; 3) lungo una direzione che forma un angolo di 30° rispetto all'asse x scriveremo:

```
object{
      versore
      rotate 30*z
      translate <1,2,3>
}
```

Gli oggetti che abbiamo definito possono essere usati anche all'interno di altre definizioni. Ecco come assemblare i tre assi di un sistema di riferimento usando tre copie opportunamente orientate del nostro versore:

```
#declare assi = union{
   object{versore}                  //asse x
   object{versore rotate 90*z}      //asse y
   object{versore rotate -90*y}     //asse z
}
```

Il nostro versore ha lunghezza fissa, ma a volte si potrebbe aver bisogno di vettori con ampiezza regolabile a piacere. Una soluzione al problema potrebbe essere quella di moltiplicare per un fattore di scala, ma in questo modo si deforma la punta con risultati poco soddisfacenti. Meglio allora scrivere una macro, ovvero una sorta di funzione che provveda a costruire un vettore in base alla lunghezza assegnata. Anche in questo caso la sintassi usata in POV-Ray è molto semplice. Si comincia con la direttiva `#macro` seguita dal nome con il quale etichettiamo la funzione e da eventuali parametri da passare al momento della chiamata (nel nostro caso la lunghezza del vettore). Si inserisce poi la sequenza delle operazioni che dovranno essere eseguite in risposta alla chiamata e si termina con la direttiva `#end`.

```
#macro vettore(modulo)
     union{
           cylinder{<0,0,0>,<modulo,0,0>,0.03}
           cone{<modulo,0,0>,0.07,<modulo + 0.2,0,0>,0}
           pigment{color rgb <0, 1, 0>}
         }
#end
```

Aumentiamo le nostre richieste: vogliamo una macro che visualizzi un vettore applicato in un punto assegnato.

```
#macro vett_in_P(vett,punto)
     #declare inizio = punto;
     #declare fine = punto + vett;
     #declare apice = fine + 0.2*vnormalize(vett);
     union{
```

```
          cylinder{inizio,fine,0.03}
          cone{fine,0.07,apice,0}
          pigment{color rgb <0, 0, 1>}
     }
#end
```

All'interno della macro vengono definite tre variabili contenenti rispettivamente la posizione del primo estremo del corpo del vettore, del secondo e dell'apice della punta. Per quest'ultima viene utilizzata una funzione predefinita in POV-Ray, `vnormalize()`, per normalizzare il vettore. L'elenco completo delle funzioni predefinite per operare sia su numeri che su vettori è piuttosto esteso e si trova nell'aiuto in linea. Vale la pena di notare che nel dichiarare nuove variabili non occorre specificarne il tipo come avviene in molti linguaggi di programmazione. Inoltre si è potuto combinare tra loro vettori come se si trattasse di numeri, senza dover ricorrere a codice aggiuntivo. Due aspetti apprezzabili per chi, come me, non è particolarmente esperto in informatica e non ha molto tempo da dedicare allo sviluppo di programmi.

La macro mostrata va intesa come un punto di partenza da sviluppare ulteriormente, ad esempio per quanto riguarda il controllo dati in ingresso. Se inavvertitamente viene passato un vettore nullo l'istruzione di normalizzazione provoca il blocco dell'esecuzione. Tuttavia, con un po' di attenzione, si può utilizzare per visualizzare senza sforzo i vettori velocità o accelerazione nel moto di una particella o quelli di un campo.

Come esempio conclusivo supponiamo di voler realizzare un'illustrazione che mostri il vettore velocità e le sue componenti in corrispondenza di alcuni istanti durante il moto parabolico di una particella. Utilizziamo la macro ed un ciclo "while" grazie al quale automatizzare il calcolo. Il valore dell'accelerazione ed il fattore di scala per la velocità sono stati scelti in funzione delle dimensioni della finestra di visualizzazione. Si possono mostrare (o eliminare dall'immagine) le componenti eliminando (o ripristinando) le doppie barre che denotano i commenti. Da sottolineare la possibilità di utilizzare il prodotto scalare tra vettore e versori degli assi per ottenere le componenti.

```
// condizioni iniziali

#declare r0 = <-1.5,-0.5,0>;
#declare v0 = <2,3,0>;
#declare g = <0, -5, 0>;

#declare t0 = 0;
#declare t1 = 1;
#declare dt = 0.25;

// calcolo dei vettori

#while  (t0<=t1)
     #declare r = r0 + t0*v0 + 0.5*t0*t0*g;
     #declare v1 = v0 + t0*g;
     object{vett_in_P(0.25*v1,r)}
     //object{vett_in_P(0.25*(v1.x)*x,r)}
     //object{vett_in_P(0.25*(v1.y)*y,r)}
     #declare t0 = t0 + dt;
#end
```

7. Animazioni

Trovo che la possibilità di realizzare filmati sia uno degli aspetti più intriganti di POV-Ray. L'idea di fondo non è diversa da quella del cinema di animazione: creare una sequenza di immagini fisse che poi dovranno essere riprodotte in rapida successione per dare l'illusione di movimento.

In pratica si devono affrontare due passaggi. Prima nel file di testo, che indicheremo nome_file.pov, contenente la descrizione della scena si parametrizzano gli oggetti da animare in funzione del tempo, ovvero della variabile `clock`. Poi si crea un secondo file, nome_file.ini, nel quale si indicano al programma le caratteristiche principali del filmato, tra le quali: nome del file contenente la descrizione degli oggetti, valori iniziali e finali della variabile `clock`, numeri che devono contrassegnare rispettivamente la prima e l'ultima immagine, dimensioni delle immagini e formato. Per risparmiare fatica ed evitare nel contempo di dimenticare qualcosa conviene anche in questo caso utilizzare un file .ini tra quelli già presenti limitandosi a modificare i valori dei diversi parametri prima di salvarlo con altro nome nella stessa cartella contenente il file nome_file.pov.

Una volta pronto nome_file.ini non resta altro da fare che lanciare il programma senza uscire della finestra dell'editor contenente il file .ini. Tutte le immagini richieste verranno automaticamente create, numerate in sequenza, denominate con nomi tipo: nome_filenn.bmp, (dove nn = 00, 01, 02, …) e salvate nella stessa cartella di nome_file.pov. La creazione di tutti i fotogrammi può essere un processo lungo visto che per avere un movimento fluido sono consigliate 20 immagini al secondo.

Sfortunatamente POV-Ray non offre al suo interno la possibilità di creare il filmato vero e proprio a partire dalle immagini ed occorre utilizzare altri programmi. Suggerimenti in merito si trovano sul sito *www.povray.org*.
Personalmente ho utilizzato un programma che avevo già sperimentato in altre occasioni: *pjbmp2avi.exe*. È un programma freeware, facilmente rintracciabile con un motore di ricerca, sviluppato da Paul Cador Roberts diversi anni fa. Dotato soltanto delle funzionalità essenziali è semplice da usare ed in grado di svolgere il suo compito senza problemi. Per costruire un filmato basta copiare *pjbmp2avi.exe* nella cartella contenente le immagini (appena 50 K di eseguibile, nessuna installazione) e lanciarlo. Nella finestra che si apre si indica il nome che contraddistingue la serie di immagini (bastano le prime lettere), il tipo di immagini (.bmp o .tga), il nome che dovrà avere il filmato ed il numero di fotogrammi al secondo, poi si clicca su *create*. Si apre una seconda finestra con una casella a discesa nella quale scegliere il programma che si occuperà della compressione (codec). I nomi della lista variano da computer a computer, basterà fare alcune prove per decidere qual è il più adatto. Tenere presente che scegliendo "fotogrammi non compressi" si rischia di avere un filmato di diverse centinaia di MByte. Scelto il codec ha inizio la creazione del filmato che solitamente è abbastanza veloce.

Il filmato prodotto può essere visto con un riproduttore quale *Real Player* o *Windows Media Player*.

8. Piccolo esperimento in classe

Per sondare le reazioni degli studenti di una III liceo scientifico PNI di fronte al programma ho dedicato due ore di laboratorio di informatica all'attività descritta nel seguito. Dato che in classe si stavano trattando le coniche, la proposta è stata quella di fare alcuni esperimenti con le sezioni di un cono.

Una volta ricevute le informazioni di base riassunte in una presentazione elettronica i ragazzi hanno cominciato le loro sperimentazioni partendo dalla Basic Scene discussa in precedenza. È stato

chiesto loro di cercare di capire il senso di alcune istruzioni, ad esempio dove vengono specificati raggio e posizione della sfera, e di apportare delle modifiche per studiarne l'effetto.

Indicato poi come sostituire la sfera con un cono, si è chiesto agli studenti di trovare i valori corretti di altezza e raggio per avere un angolo al vertice assegnato. In fine, dopo aver illustrato come introdurre un piano, orientarlo ed utilizzarlo per sezionare il cono, i ragazzi sono stati lasciati liberi di sperimentare con le sezioni. Il file di testo con la descrizione della scena utilizzato per le prove è riportato in appendice.

La reazione alla comparsa della prima immagine prodotta da POV-Ray è stata per tutti improntata alla sorpresa. La qualità dell'immagine li ha colpiti e molti sono stati i commenti favorevoli, specialmente sulla presenza delle ombreggiature che contribuisce molto al realismo.
La maggior parte della classe ha affrontato con interesse anche la proposta di sperimentazione. Ad onor del vero va registrato che gli sforzi erano più orientati al cambiare il colore degli oggetti o la loro orientazione che ad esplorare le diverse sezioni operabili con il piano. Tuttavia questo atteggiamento si può imputare probabilmente alla curiosità suscitata dal primo incontro con un'applicazione di questo tipo.

Per avere più chiaramente il polso della situazione ho chiesto successivamente ai ragazzi di mettere brevemente per scritto le loro impressione. I giudizi sono stati per lo più positivi, tipo:

> "Credo sia un programma molto semplice da usare, infatti i comandi non sono "incomprensibili" e sono brevi in confronto a quanto realmente ci voglia per creare un oggetto 3D sul computer. La mia impressione è positiva."

> "Secondo me cattura più l'attenzione rispetto a Mathematica per le immagini tridimensionali che si vanno a trovare e magari lo sentiamo più vicino proprio perché le immagini si avvicinano alla realtà. "

Non sono mancati, sia pure in misura molto minore, giudizi negativi:

> "E' un programma abbastanza difficile. Noi abbiamo fatto il cono sezionato da un piano, ma per farlo ci sono voluti molti comandi, quindi secondo me è un po' complicato."

> "E' un programma difficile perché devi specificare tutto; ci sono un sacco di particolari (piano, colori, sfondo …) e richiederebbe, secondo me, uno studio prima delle regole di base, e poi un'applicazione. "

Tuttavia è incoraggiante il fatto che, anche tra chi si è espresso negativamente, quasi tutti abbiano giudicato favorevolmente la proposta di ripetere in futuro ed in modo più sistematico esperienze analoghe.

9. Bibliografia

D. Hearn, M. P. Baker *Computer Graphics* Pearson Education International

Guida in linea di POV – Ray

Appendice

In appendice sono riportati due listati commentati come esempi dei possibili usi di POV-Ray.
Il primo è quello effettivamente utilizzato insieme ai ragazzi nella prova in laboratorio.
Il secondo vuole rappresentare una produzione di materiale didattico. Anche in questo caso si tratta delle sezioni del cono, ma la presentazione dell'oggetto è maggiormente curata e l'angolazione del piano secante viene variata nel tempo per produrre una un'animazione.

Esempio 1: Sezione di un cono

Variando l'angolo di inclinazione del piano secante si ottengono le diverse sezioni coniche.

```
// Persistence of Vision Ray Tracer Scene Description File
// File: lab01.pov

#version 3.6;
#include "colors.inc"

global_settings {  assumed_gamma 1.0 }

camera {
  location  <0.0, 0.5, -4.0>
  look_at   <0.0, 0.0,  0.0>
}

sky_sphere {
  pigment {
    gradient y
    color_map {
      [0.0 rgb <0.6,0.7,1.0>]
      [0.7 rgb <0.0,0.1,0.8>]
    }
  }
}

light_source {  <-30, 30, -30> color rgb <1, 1, 1> }

plane { y, -1
  pigment { color rgb <0.7,0.5,0.3> }
}

#declare alfa = 60;     // Inclinazione del piano secante

// Al cono si sottrae il semispazio "interno" al piano inclinato. Per
// convenzione si tratta del semispazio posto dalla parte opposta
// a quella indicata dalle normale al piano.
difference{
        cone{-y,1,y,0 pigment{DarkGreen}}
        plane{-y,0 pigment{Orange}       //Si crea il piano y = 0…
            rotate alfa*z}               //… poi si ruota

        rotate -60*y         //rotazione dell'oggetto per migliorare la vista
}
```

Esempio 2: Sezioni animate

Viene rappresentato un cono di vetro sezionato da un piano, anch'esso di vetro. La sezione all'interno è evidenziata in colore. Per ottenere questo effetto la sezione colorata viene prodotta separatamente come intersezione tra un secondo cono leggermente più piccolo ed un sottile parallelepipedo. La sezione viene poi reinserita all'interno del primo cono.

```
// Persistence of Vision Ray Tracer Scene Description File
// File: coniche_perf.pov

#version 3.6;
global_settings { assumed_gamma 1.6 }

// Importazione librerie con definizione colori e materiali
#include "colors.inc"
#include "stones1.inc"

// Camera e Luci ====================================
camera {
        location <0,0.5,-7.2>
        direction <0,0,2.5>
        up <0,1,0>
        right<1.33333,0,0>
        look_at<0,0,-1>
}

object {
        light_source{ <3,4,-3>
                color rgb <1,1,1> }}

object {
        light_source{ <-2.5,6,-3>
                color rgb <0.25, 0.25, 0.25> }}

// Sfondo ===========================================
sky_sphere{pigment{color SkyBlue}}

// ## Dichiarazioni di variabili, oggetti e materiali ##

// Variabili per il controllo dell'angolo di sezionamento, dipendente dal tempo,
// e di orientazione dell'intera figura. Queste variabili vengono usate nelle
// successive dichiarazioni degli oggetti e devono pertanto precederle.
#declare Angolo=<0,0,80*clock>;
#declare Rotazione = <0,-60,0>;

// Materiale: vetro (copiato da un file .ini) =======
#declare T_Old_Glass=
texture {
    finish {
            ambient 0.1 diffuse 0.1
            reflection .1 specular 1
            roughness 0.001
    }
    pigment { color rgbf <0.8, 0.9, 0.85, 0.85> }
}

// Oggetto: cono. La base: raggio 1, poggia su y = -1; vertice: (0,1,0)
#declare Cono = object {
        cone {
                <0,1.0,0>,0.0
                <0,-1.0,0>,1
```

```
                texture {
                        pigment {color rgbf <1, 1, 1, 0.7>}
                        finish {phong 1}
                }
        }
     }

// Oggetto: Sezione conica. Viene descritta come intersezione tra un sottile
// parallelepipedo opportunamente ruotato ed un cono di dimensioni ridotte del
// 5%. La riduzione assicura che la sezione non esca dal cono sezionato.
 #declare Sezione = object {
                intersection{
                        box {<-1.4, -0.0075, -1>,<1.0, 0.0051, 1>
                                rotate Angolo
                                translate <0,0,0>
                        }
                        cone {
                                <0,0.95,0>,0.0
                                <0,-1,0>,0.95
                        }
                }
                texture {pigment {color rgb <1, 0.5, 0>}
                finish {ambient 0.3}
                }}

//Oggetto: piano secante ruotato dell'angolo definito sopra
#declare Piano = object {
                        box {<-1.4, -0.005, -1.0>,<1.0, 0.005, 1.0>
                                rotate Angolo
                                   }
                        pigment {color rgbf <0.2, 0.9, 1, 0.8>}
                        texture{T_Old_Glass}
                        interior{ior 1.5}}

//Oggetto: supporto. Per smussare i bordi sono stati aggiunti dei cilindri
// e delle sfere intorno al parallelepipedo.
#declare Supporto = union {
        box {<-1.3, -1.25, -1.3>,<1.3, -1.0, 1.3>}
        cylinder {<1.3, -1.125, -1.3>,<1.3, -1.125, 1.3>,.125}
        cylinder {<-1.3, -1.125, -1.3>,<-1.3, -1.125, 1.3>,.125}
        cylinder {<-1.3, -1.125, 1.3>,<1.3, -1.125, 1.3>,.125}
        cylinder {<-1.3, -1.125, -1.3>,<1.3, -1.125, -1.3>,.125}
        sphere{<1.3, -1.125, 1.3>,.125}
        sphere{<-1.3, -1.125, 1.3>,.125}
        sphere{<-1.3, -1.125, -1.3>,.125}
        sphere{<1.3, -1.125, -1.3>,.125}
        texture {T_Stone12}
}

// Ora gli oggetti vengono messi in scena. La rotazione migliora la vista
// che si ottiene
object {
       union{
             object {Cono}
             object {Supporto}
             object {Sezione}
             object {Piano}
       }
       rotate Rotazione
}
```

Quello che segue è il file .ini necessario per produrre tutti i fotogrammi dell'animazione.

Supponendo una riproduzione di 15 fotogrammi al secondo con le 120 immagini prodotte si ottiene un'animazione di 8 sec.

```
; Persistence Of Vision raytracer version 3.6.

Antialias=on

Antialias_Threshold=0.1
Antialias_Depth=2
Test_Abort_Count=10
Input_File_Name=coniche_perf.POV

Initial_Frame=1
Final_Frame= 120
Initial_Clock=0
Final_Clock=1

Cyclic_Animation=off
Pause_when_Done=off
Output_File_Type=T

Width=512
Height=384
```