Linguaggi di Realtà Virtuale

Introduzione

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Osvaldo Gervasi

osvaldo@unipg.it

Argomenti del corso

- Introduzione alla Realtà Virtuale
- Linguaggi di realtà virtuale:
  - VRML
  - X3D

Le interfaccie grafiche

- Analisi di dati complessi
  - Identificazione di strutture
  - Percezione di fenomeni fisici
  - Aderenza a modelli teorici
  - Rappresentazioni grafiche vs. dati in forma grezza
  - Interazione di tipo *what-if*

Le interfaccie grafiche (i)

- Interfaccie operative semplici e potenti
  - Astrazione dalle modalità specifiche di utilizzo delle procedure
  - Uso di strumenti intuitivi quali
    - Bottoni
    - Icone
    - Finestre

Uno sguardo al passato...

- L'evoluzione delle interfaccie grafiche origina da
  - Evoluzione della tecnologia elettronica
  - Evoluzione dei linguaggi di programmazione
    - Astrazione dei dati
    - Indipendenza dall'architettura del computer

Verso la programmazione ad oggetti...

- Programmazione strutturata: *sequenza*, *selezione*, *ripetizione*
- Astrazione sul controllo: modularità e indipendenza dei *blocchi*
- Information hiding
- Librerie di procedure (moduli pronti all'uso)

Linguaggi orientati agli oggetti: la classe

- L'orientamento agli oggetti consiste nell'applicazione di 3 concetti fondamentali
  - **Incapsulamento**: le **classi** sono composte da campi procedura detti **metodi**, che definiscono il comportamento della classe. Per ogni classe è possibile creare un oggetto che è chiamato **istanza** della classe stessa.

Linguaggi orientati agli oggetti

- **Ereditarietà**:da una **superclasse** si definiscono *classi derivate*, specificando solo le differenze. E' prevista anche l'eredità multipla, in cui una singola classe deriva-ta dipende da più classi di partenza.
- **Polimorfismo**: l'esecuzione di uno stesso metodo in diverse sottoclassi derivate dalla stessa superclasse, causa diverse risposte automatiche dalle diverse classi.

Programmazione per eventi

- La programmazione ad oggetti porta ad uno sconvolgimento, rispetto alla programmazione sequenziale.
- La chiamata alla procedura viene sostituita dall'invio di un **messaggio** ad un oggetto, il quale risponderà con un proprio metodo.
- Questa modalità operativa è chiamata **programmazione per eventi** ed è alla base delle moderne interfacce grafiche.

Dalla linea di comando...

- Dagli albori dell'informatica, molte interfacce utente si sono basati su un approccio basato sul carattere:
  - DOS
  - Unix
- Il primo ambiente grafico fu sviluppato agli inizi degli anni '70 da Alan Kay presso Xerox Corp: **SmallTalk** basato sul mouse e il trackball

...all'interfaccia ad oggetti

- Kay viene assunto da Apple per dare inizio alla produzione di un rivoluzionario tipo di computer: il MacIntosh Apple in cui la linea di comando viene sostituito dalla metafora della scrivania (desktop)e degli oggetti (icone)
- I costi ne limitano la diffusione, poiché risulta un sistema più complesso sia come software che come hardware.

Interfaccia ad oggetti

- *Presentation Manager* e OS/2 (fine anni '80) di IBM hanno avuto una discreta importanza
- *Microsoft Windows* (1983) ha progressiva-mente catturato la gran parte del mercato Personal Computer
- *X-Windows* (1987, MIT) rappresenta l'ambiente a finestre dei sistemi Unix, oggi diventato popolarissimo grazie alla diffusione del sistema operativo Linux

Il software Freeware

- La comunità degli utenti Internet è da sempre estemamente prolifica
- Il piacere di comparire come autore di un prodotto molto diffuso ha spinto molti ad offrire gratuitamente il proprio prodotto
- Opposizione alla grandi industrie software (ad es.: Microsoft) che cercano di imporre a fini commerciali una filosofia aziendale al grande pubblico
- Rigore assoluto nella qualità del codice scritto ed aderenza stretta ai concetti di riuso e portabilità
- Il caso Linux

Java: il linguaggio ad oggetti più potente ed attuale

- Java è un linguaggio orientato agli oggetti sviluppato da Sun Microsystem, ma reso di pubblico dominio, la cui filosofia si sintetizza in: **write once, run everywhere**
- Java è attualmente il linguaggio più popolare tra gli sviluppatori di applicazioni Internet, date le sue caratteristiche di portabilità, sicurezza, fruibilità in rete
- E' lo strumento più potente per
  - scrivere applicazioni software che girano in qualsiasi computer
  - uniformare le caratteristiche delle applicazioni

Java

- Java esegue una compilazione del codice sorgente per produrre un codice denominato bytecode, in grado di girare in una Java Virtual Machine (JavaVM)e garantendo la portabilità del codice.
- I problemi di portabilità vengono confinati nella disponibilità della JavaVM nei vari ambienti.
- I problemi di lentezza sono destinati ad alleviarsi col progredire della tecnologia (compilatori just-in-time, JIT)

Java (i)

- Le applicazioni Web sono state pesantemente modificate con l'introduzione di Java
- L'accesso a database e la generazione dinamica di informazioni è più potente ed efficiente
- E' stato possibile creare degli strumenti di gestione più efficienti che facilitano la gestione di siti Web complessi anche ad utenti con scarse competenze specifiche
- E' sempre in aumento il numero di strumenti e di librerie di funzioni che aiutano lo sviluppo di nuove applicazioni
- Grosse aziende mondiali (ad es.: IBM) puntano ad una massiva diffusione della tecnologia per indurre l'utente ad acquistare servizi a valore aggiunto

Strumenti di produttività

- Nell'ambiente grafico a finestre sono disponibili una serie di strumenti integrati per
  - Word processing e composizione di testi
  - Fogli elettronici
  - Grafica
  - Data Base
  - Elaborazione di immagini
  - Navigazione in Internet
  - Produzione di pagine Web
  - Posta elettronica e servizi Internet

Strumenti di produttività (i)

- L'utente è in grado di archiviare e catalogare il proprio materiale in modo intuitivo
- Il riuso del materiale elettronico è molto spinto
- L'ottimizzazione aumenta se si adottano dei *template* per la gestione delle diverse situazioni
- L'uso di grafica e immagini arricchisce enormemente i documenti prodotti

… però in molti contesti tutto ciò è insufficiente

…

diventa indispensabile la terza dimensione
e l'esplorazione più profonda dei sensi dell'uomo

Introduzione alla VR

- La Realtà Virtuale (VR) consiste nella simulazione di alcuni aspetti del mondo reale.
- Una parola chiave nella definizione della VR è l interattività: la capacità che il mondo virtuale ha di modificarsi in seguito alle azioni che compie il visitatore, espressione del mondo reale.
- Simulazione e modellazione sono tra le aree principali della VR, che trova nell intrattenimento e nella formazione le applicazioni prevalenti.

Introduzione alla VR (i)

- La Realtà Virtuale è uno dei settori più caldi nella ricerca e lo sviluppo dell'industria dei computer moderni.
- Ha applicazioni dal settore delle immagini mediche, al disegno di interni, alle videoconferenze intercontinentali, all'esplorazione di mondi futuri
- Ci sono molti settori nei quali VR può essere utilizzata, ma serve soprattutto a trovare nuovi modi per far lavorare insieme uomo e computer

Introduzione alla VR (ii)

Jaron Lanier

- Il termine Virtual Reality è stato coniato da Jaron Lanier fondatore di VPL Research nel 1989.
- Altri termini collegati furono:
  - Artificial Reality (Myron Krueger, anni 70)
  - Cyberspace (William Gibson, 1984)
  - Virtual Worlds e Virtual Environments (anni 90)

Hamit, Francis: Virtual Reality and the Exploration of Cyberspace Sams Publishing, 1993

Anderson, Time: The Virtual Reality Casebook Van Nostrand Reinhold Publishing, 1994

Introduzione alla VR (iii)

- Il termine Virtual Reality è oggi usato in diversi modi, spesso in modo confuso e errato.
- L accezione originaria faceva riferimento al concetto di Immersive Virtual Reality.
- Nell immersive VR, con l ausilio di dispositivi esterni, l utente diventa immerso in un mondo tridimensionale artificiale generato dal computer.

Introduzione alla VR (iv)

- La Realtà Virtuale è pensata come una nuova tecnologia, ma il suo sviluppo risale a oltre 40 anni, grazie all intuizione diMorton Heilig, un uomo che non era né scienziato né ingegnere. Era un cineasta che voleva utilizzare a pieno il campo visivo dello spettatore:

I became very excited. I thought, "Why stop at a picture that fills only 18 percent of the spectator's visual field, and a two-dimensional picture at that? Why not make it a three-dimensional image that fills 100 percent of the spectator's visual field, accompanied by stereophonic sound? If we're going to step through the window into another world, why not go the whole way?"

Cinematografia

- La Realtà Virtuale è stata influenzata anche da tecniche cinematografiche come il cinema stereoscopico o 3-D e tecniche ad ampio schermo realizzate fin dagli anni 50.
- Il Cinerama, la più nota di queste tecnologie, espande la percezione visiva, riempiendo una porzione più ampia della visuale dello spettatore. La teoria della immersione visuale è importante per la VR

Sensorama

- Morton Heiling riteneva che il futuro del cinema fosse nella creazione di film che esplorassero i sensi umani. Disegnò gli elementi che erano necessari a creare la sensazione di illusione totale (sensori del cervello e dei movimenti del corpo). Chiamò il prodotto teatro dell esperienza.
- Per mancanza di fondi la sua idea non andò oltre al prototipo di un dispositivo che chiamò Sensorama, creato nel 1962.

Sensorama (i)

- Sensorama simulava l esperienza dei sensi legata alla guida di una motocicletta, combinando filmati 3-D (mediante tre videocamere 35mm), suoni stereo, vento e persino aromi

  Facendo presa sul manubrio di una moto con sedile e casco creati ad hoc, l utente potevaviaggiare tra diverse scene, ad es. rappresentanti la California e le strade di Brooklin.

  Delle piccole griglie vicino al naso ed alle orecchie emettevano della brezza e degli aromi

  L idea di un mezzo che combinasse esperienze multisensoriali artificiali divenne realtà

Sketchpad

- Un ramo della ricerca dell'intelligenza artificiale (AI) esplorò la possibilità di creare interfacce migliori tra uomo e macchina.
- Ivan Sutherland presentò una tesi Ph.D. su questo argomento nei primi anni '60, riprendendo l'idea di Heiling e dimostrando modi innovativi di interazione.
- Creò Sketchpad, un programma che usava la tecnologia del computer per creare immagini da idee astratte e che diede vita al Computer Aided Design (CAD).
- Usando un dispositivo a forma di penna, il computer poteva creare immagini sofisticate sullo schermo ed interagire con l'utente.
- Sutherland inventò il concetto di acceleratore grafico, che avrebbe modificato la visualizzazione futura

Simulazione di volo aereo

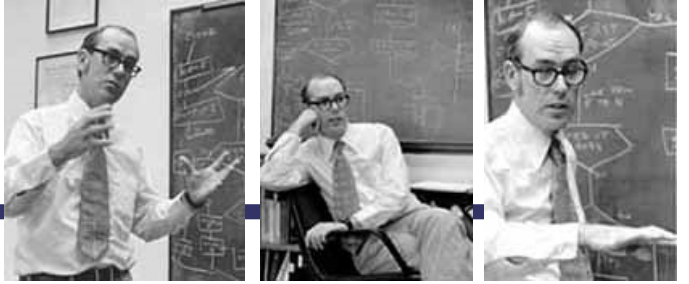
- I militari intuirono la portata di queste idee e negli anni 70 costruirono vari tipi di elmetti che simulassero la visione durante il volo aereo.

- Anche la NASA esplorò nuovi dispositivi in grado di simulare il volo nello spazio e l atterraggio sulla luna.

- I primi prototipi di simulatori di volo da parte dell industria aerea e l aviazione militare americana risalgono alla II Guerra Mondiale: i giovani piloti si esercitavano manovrando dei comandi che simulavano il volo mediante il controllo di piattaforme mobili e ruotanti

Ivan Sutherland

Ivan Sutherland sulla sua BMW K100



Ivan Sutherland durante una lezione al California Institute of Thechnology nel 1976

- E' ritenuto l'inventore della Computer Graphics e della VR
- E' stato fondatore della Evans & Sutherland, una delle più prestigiose fabbriche di stazioni grafiche ad altissime prestazioni.

"A display connected to a digital computer gives us a chance to gain familiarty with concepts not realizable in the physical world. It is a looking glass into a mathematical wonderland."

"I just need to figure out how things work."

"It's very satisfying to take a problem we thought difficult and find a simple solution. The best solutions are always simple."

H M D

- Sutherland si dedicò allo sviluppo di tecnologie che consentissero agli utenti di esplorare un mondo di immagini generate col computer.
- Nel 1965, con fondi DARPA e NASA-AMES, ideò un dispositivo chiamato *Head Mounted Display* (HMD), che trasportava l utente in un mondo 3-D limitando il contatto visuale con le immagini mostrate da piccoli schermi di computer montati nelle lenti di un binocolo.

H M D (i)

- E divenuta una pietra miliare della VR e la prima applicazione commerciale HMD fu introdotta da VPL nel 1989 (EyePhone system)

Altri progetti

- Nei primi anni 70 alcuni progetti contribuirono a creare la VR moderna:
- Aspen Movie Map: (Andrew Lippman, Michael Naimark e Scott Fisher al MIT) mostrava immagini video di Aspen, nel Colorado. I visitatori potevano navigare nel mondo e indicare le loro scelte in uno schermo tattile
- Videoplace: uno dei molti ambienti artisitici sperimentali realizzati dallo studente Myron Krueger, usava i computer per creare quella che lui chiamava Artificial Reality, consentendo ai visitatori di interagire con la grafica generata dal computer e le immagini proiettate

Il primo sistema VR

- Nella metà degli anni 80, le diverse tecnologie usate nella VR diedero vita al primo vero sistema VR: un ambiente affidabile di formazione di astronauti per le missioni spaziali:

  Scott Fisher, che continuò lo studio dei mondi virtuali dopo l Aspen Movie Map

  Stephen Ellis (UCB) che aveva studiato le interazioni dell essere umano con l ambiente

  Michael McGreevy, uno studente UCB che si interessava di display per l immersione nei mondi virtuali

  Warren Robinett, un programmatore con esperienza in software educativo alla Atari

Virtual Interface Environment Workstation

- E il frutto del loro lavoro congiunto e del contributo di altri ricercatori e centri di ricerca.
- E il primo sistemache combina i vari elementi standard della VR, come l animazione video, i suoni 3-D, riconoscimento e sintesi vocale e head-mounted display.
- Un guanto digitale, ideato per suonare la chitarra, completava il sistema

L'evoluzione della VR

- Da questo momento in poi lo sviluppo della VR è stato impressionante: dalla realizzazione di parchi tematici virtuali alla realizzazione dell arredamento delle stanze.
- Sono stati di notevole aiuto gli strumenti realizzati da Jaron Lanier, il cui linguaggio di programmazione guidava il primo guanto digitale alla NASA.
- La sua azienda, VPL(1989), è stata la prima a focalizzare la propria attività sullo sviluppo di dispositivi per la nascente industria VR

L'evoluzione della VR (i)

- Molti dispositivi di visualizzazione assomigliano ad elmetti, nei quali vengono opportunamente visualizzate le immagini attraverso complessi sistemi che combinano diversi piccoli schermi.
- I guanti digitali sono dotati di sensori (trackers) collegati alle parti della mano e delle dita dell utente per misurare le posizioni dell arto.
- Essi convertono i movimenti in coordinate, che vengono trasmesse al computer, in modo che il movimento venga intercettato e gestito nel modello del mondo virtuale.

L'evoluzione della VR (ii)

- E comunque possibile sperimentare i mondi virtuali anche senza dispositivi di immersione.
- Nuove tecnologie come The PHANToM, sviluppata al MIT Artificial Intelligence Lab, creano l illusione di toccare gli oggetti virtuali.
- Sistemi proiettati, spesso usati nei musei e nelle applicazioni mediche, prendono l immagine del movimento dell utente e la proiettano su larghi schermi con altre immagini.
- La simulazione VR, usata nei giochi e nei parchi VR, usa una combinazione di video monitors e capsule o piattaforme mobili per creare esperienze virtuali

L'evoluzione della VR (iii)

- Il Magic Edge era un bar-ristorante a Mountain View, CA, che offriva simulatori di volo in VR, consentendo al pubblico di effettuare combattimenti in volo in un ambiente virtuale molto sofisticato.
- Era anche possibile esplorare mondi virtuali attraverso dei desktop computer. Anche se i costi di realizzazione delle apparecchiature, limitano la possibilità di creare ambienti VR profondamente coinvolgenti.
- Nel 1984 William Gibson ha coniato il termine cyberspace, che descrive un mondo futuro creato dalla interconnessione in rete di diversi sistemi e ambienti VR.

L'evoluzione della VR (iv)

- Con l aumento della potenza dei sistemi e l abbassamento dei costi, il cyberspazio diviene sempre più realtà.
- La miniaturizzazione dei componenti è una tecnologia così sofisticata oggi, che è stato possibile introdurre il concetto di computer indossato.
- I dispositivi di visualizzazione e elaborazione dei segnali esterni, aprono nuove frontiere verso usi sociali di queste tecnologie, orientati a non vedenti e disabili in generale.

VR devices: HMD

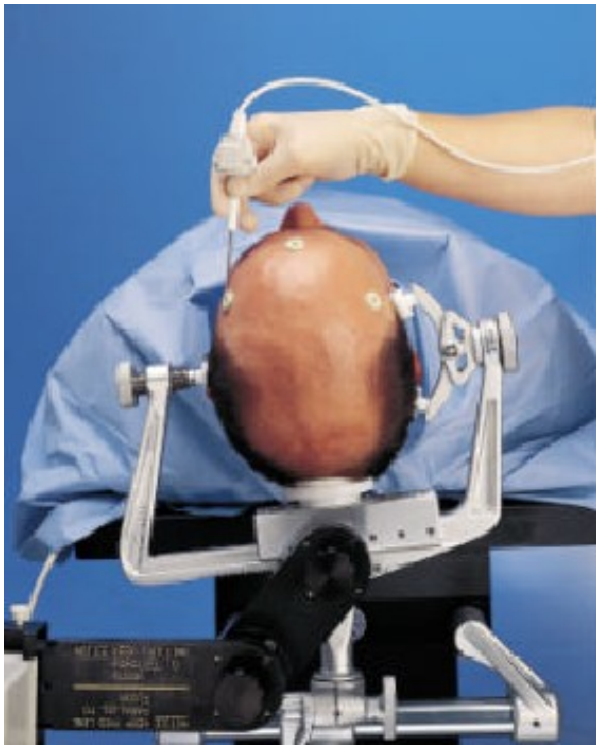
- Un apparato HMD ospita due schermi miniaturizzati ed un sistema ottico che canalizza le immagini dallo schermo agli occhi, presentando una visione stereo del mondo virtuale.
- Un tracciatore del movimento della testa dell'utente segnala al computer come aggiustare la presentazione della scena alla vista corrente.
- Ne risulta per l'utente la possibilità di guardarsi intorno e di camminare attraverso il clima avvolgente dell'ambiente virtuale.
- E' però scomodo ed intrusivo

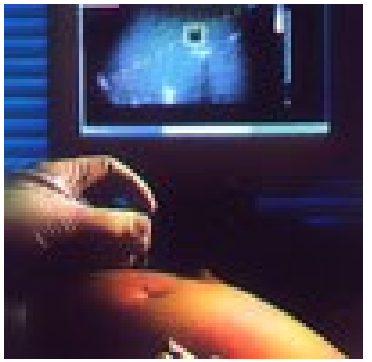
Input devices

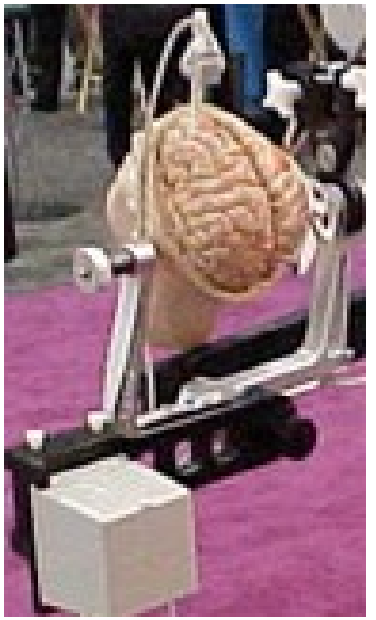
- Ci sono una varietà di dispoditivi di input che consentono all'utente di navigare attraverso un ambiente virtuale e di interagire con oggetti virtuali:
  - guanti digitali

Motion tracker

- La funzione del motion tracker è quella di intercettare il movimento dell'utente nello spazio, fornedo al programma applicativo le coordinate spaziali rivelate. Molte applicazioni in campo medico sono state sviluppate sulla base di questi dispositivi







Motion tracker (i)

- Full body motion trackers:

VR devices: BOOM

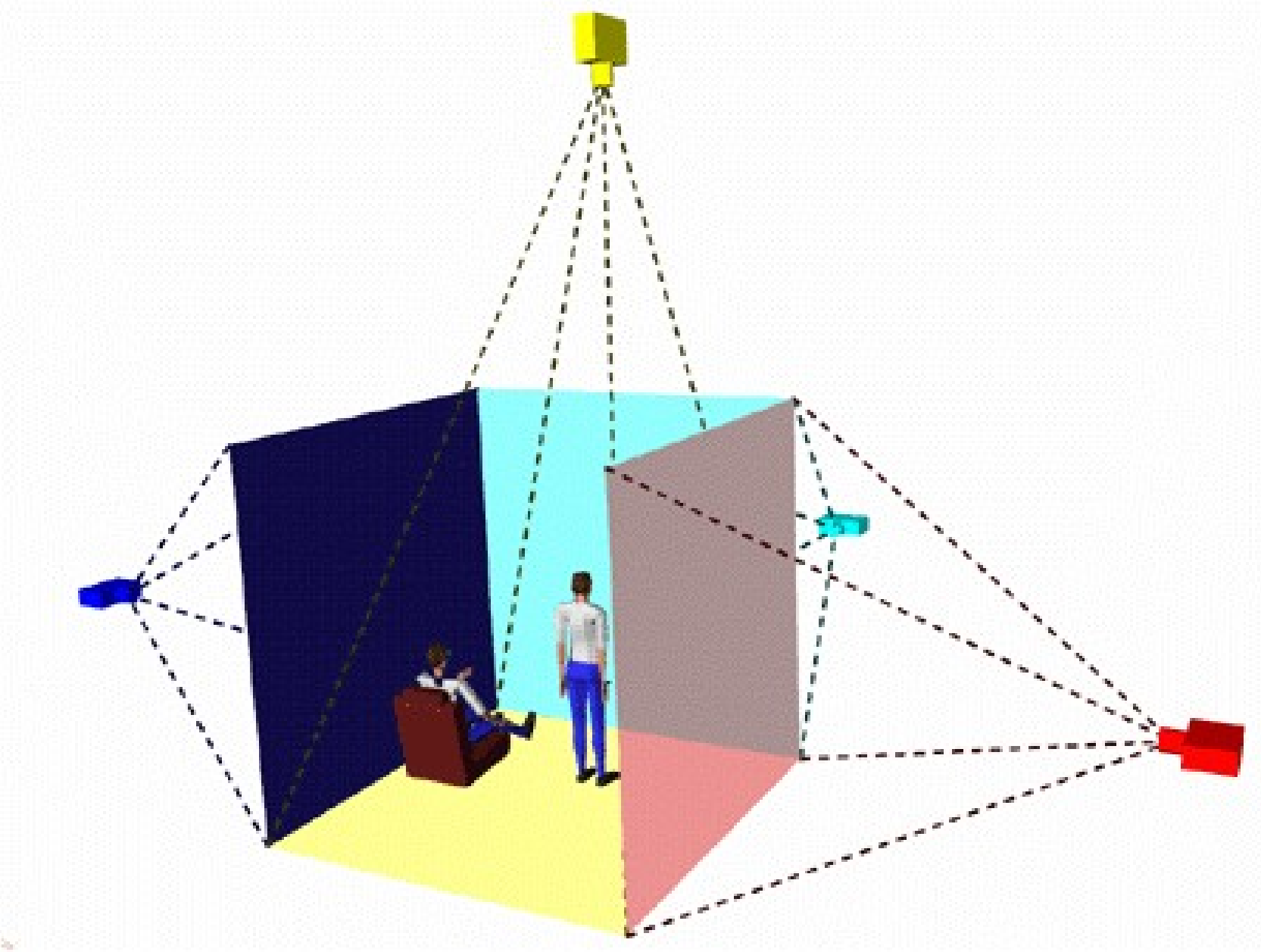
Binocular Omni Orientation Motion

BOOM

- Il BOOM, prodotto da Fakespace, è un dispositivo di visualizzazione stereoscopica.
- Schermo e sistema ottico sono alloggiati in una scatola collegata ad un braccio meccanico con collegamenti multipli.
- L'utente guarda nella scatola attraverso due fori, vede il mondo virtuale, e può manovrare entro lo spazio utile la scatola per esplorare il mondo virtuale
- I movimenti della testa sono monitorati da sensori posti nel braccio che collega la scatola al computer

VR devices: CAVE

Cave Automatic Virtual Environment



CAVE

- Il sistema CAVE è stato sviluppato a Chicago (University of Illinois) e fornisce l'illusione dell'immersione proiettando immagini stereo nelle pareti e nel pavimento di una stanza a forma di cubo
- Diverse persone, le quali indossano dei leggeri occhiali stereoscopici possono entrare e muoversi liberamente dentro CAVE
- Un sistema di rilevamento dei movimenti della testa aggiusta continuamente la proiezione stereografica con la posizione attuale dell'osservatore principale

Immersive VR

- La visione fornisce una interfaccia naturale per la navigazione nello spazio 3-D e consente di guardarsi intorno, camminare e volare negli ambienti virtuali.
- La vista stereoscopica aumenta le percezione della profondità ed il senso dello spazio.
- Il mondo virtuale viene presentato nelle dimensioni corrette e si relaziona con le dimensioni umane.
- Le interazioni realistiche con gli oggetti virtuali mediante l'utilizzo di guanti digitali e dispositivi analoghi, consente il controllo, la manipolazione e la gestione di mondi virtuali
- Può essere potenziata l'illusione di essere immersi completamente in un mondo artificiale mediante l'uso di tecnologie uditive e altre tecnologie non visuali.
- La condivisione di mondi virtuali è resa possibile mediante l'uso di applicazioni condivise in rete

Shared Virtual Environments

University of Michigan
Virtual Reality Laboratory
at the College of Engineering

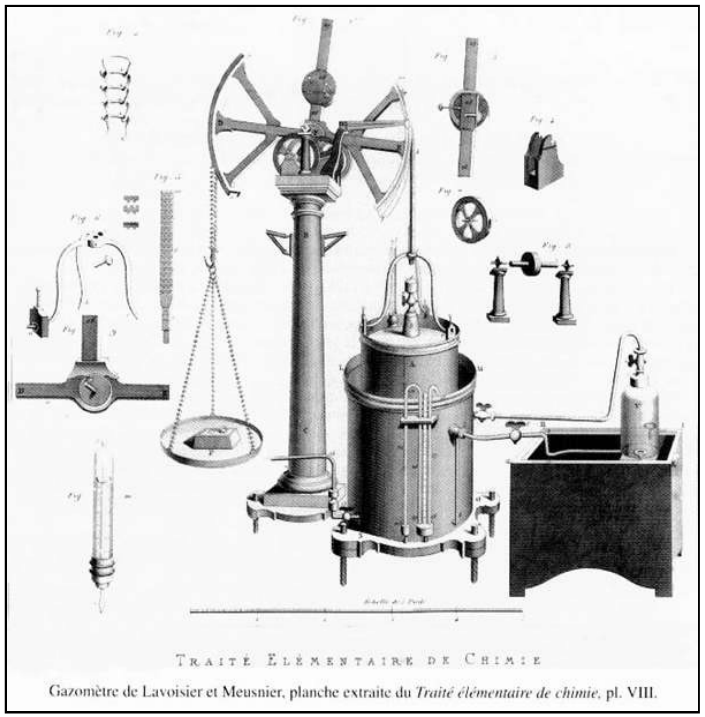
Non immersive VR

- Oggi il termine VR è utilizzato anche per applicazioni che non sono completamente *immersive*.
- I confini sono confusi tra i due modi di operare, ciò che è certo è che tutte le possibili variazioni di VR saranno sempre più importanti in futuro.
- Ciò comprende anche un tipo di navigazione che viene realizzata mediante mouse su uno schermo grafico, visione stereo dal monitor mediante occhiali stereo, stereo projection systems, etc
- Il software Apple Quicktime VR, per esempio usa delle fotografie per costruire mondi virtuali 3-D e fornisce la possibilità di camminare dentro e guardarsi intorno, usando uno schermo grafico

Non interactive VR: Quicktime VR

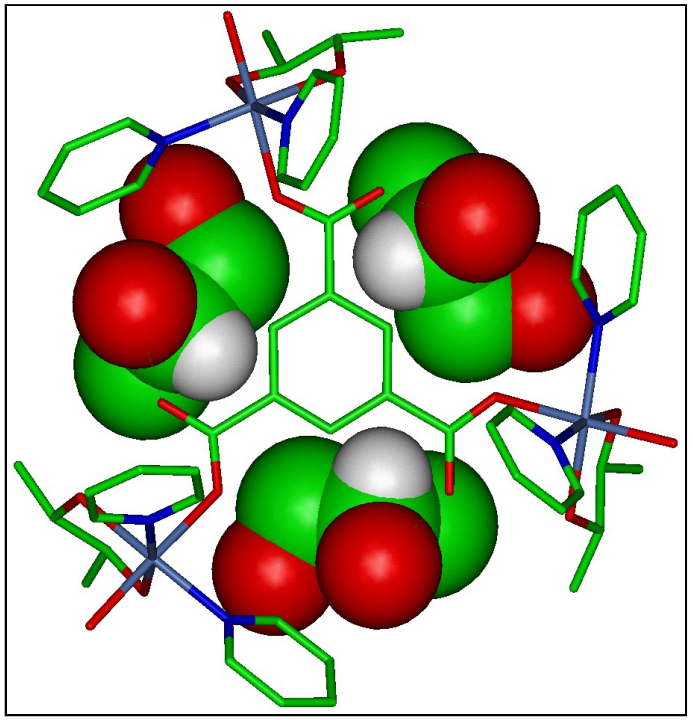
From representation to simulation

Virtual Reality is just the last step in the process of providing simulations of natural processes



Gazomètre de Lavoisier et Meusnier, planche extraite du *Traité élémentaire de chimie*, pl. VIII.

Illustration:
Graphical representation of apparatuses and procedures



Visualization:
Graphical representations of data and concepts

Simulations:
Realistic interactive 3D time dependent model representations

La realtà virtuale a supporto della scienza

- Le tecniche di Realtà Virtuale possono fornire ai ricercatori degli strumenti di investigazione ed analisi innovativi
- Una volta che lo scienziato ha definito il suo modello che descrive il suo particolare esperimento, applicando tale modello in forma predittiva può studiare nuovi effetti e nuovi comportamenti
- In questo senso disporre di strumenti di analisi innovativa porta a studiare più in dettaglio i fenomeni

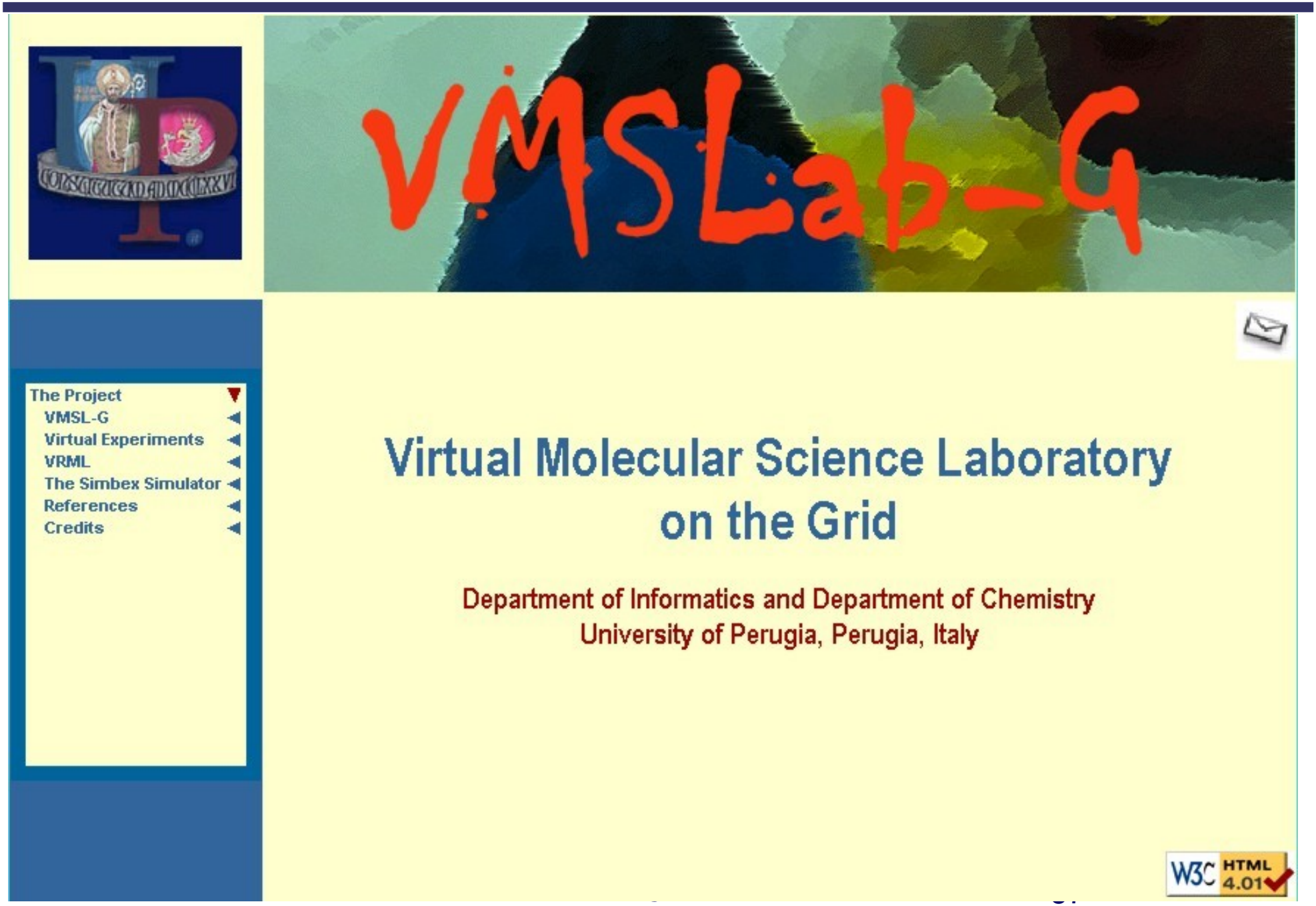
Esempio nel campo delle scienze molecolari

- Molecular Virtual Reality: le tecniche di Realtà Virtuale Molecolare vengono applicate al mondo delle reazioni elementari (scala dei nanometri)
- I motori computazionali usati per le simulazioni possono essere modificati ed integrati per fornire informazioni innovative su
  - comportamento dei sistemi
  - modo con cui le grandezze di sistema variano nel corso delle interazioni

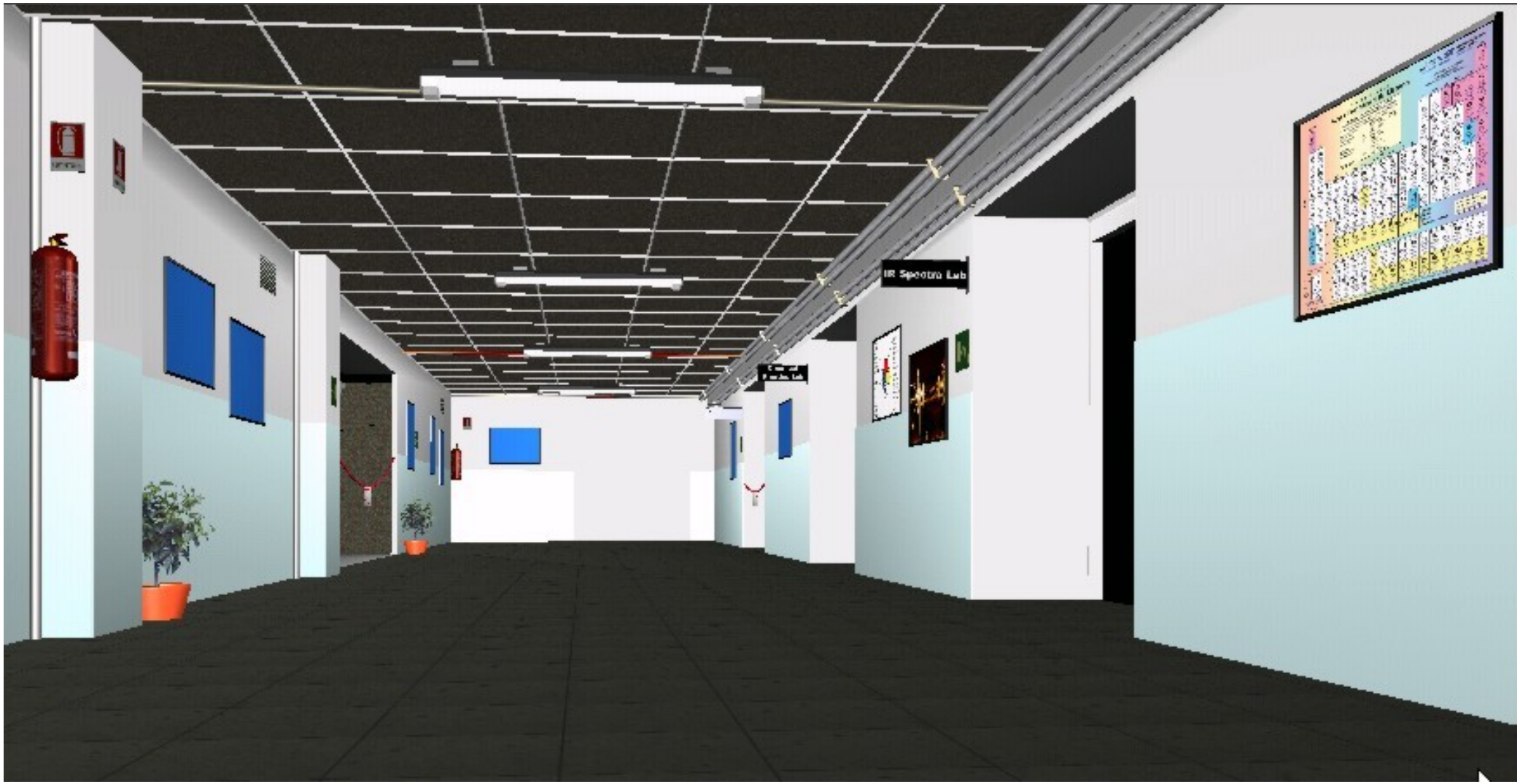
Realtà Virtuale ed e-learning

- La realtà virtule è un potente strumento per e-learning, in particolare per implementare laboratori virtuali per le attività di learning e training (pre-lab experience)
- L'adozione di queste tecnologie comporta lo sviluppo di nuove metodologie didattiche
- LA VR può essere applicata a:
  - Meter level (Human Virtual Reality, HVR)
  - Nanometer level (Molecular Virtual Reality, MVR)
- Le tecnologie di Semantic Web pssono fornire un preziosissimo strumento per seguire lo studente durante il suo percorso formativo.

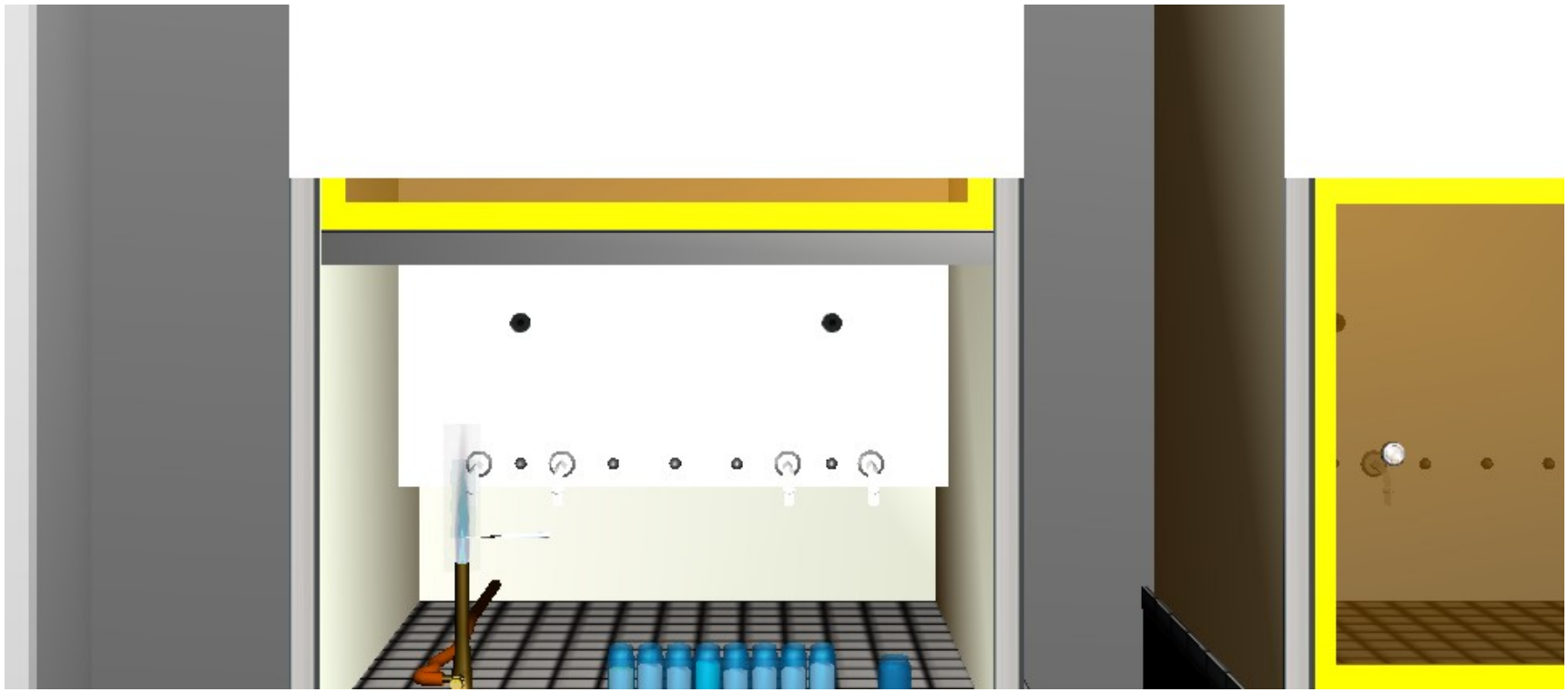
VMSLab-G: un laboratorio virtuale nel web



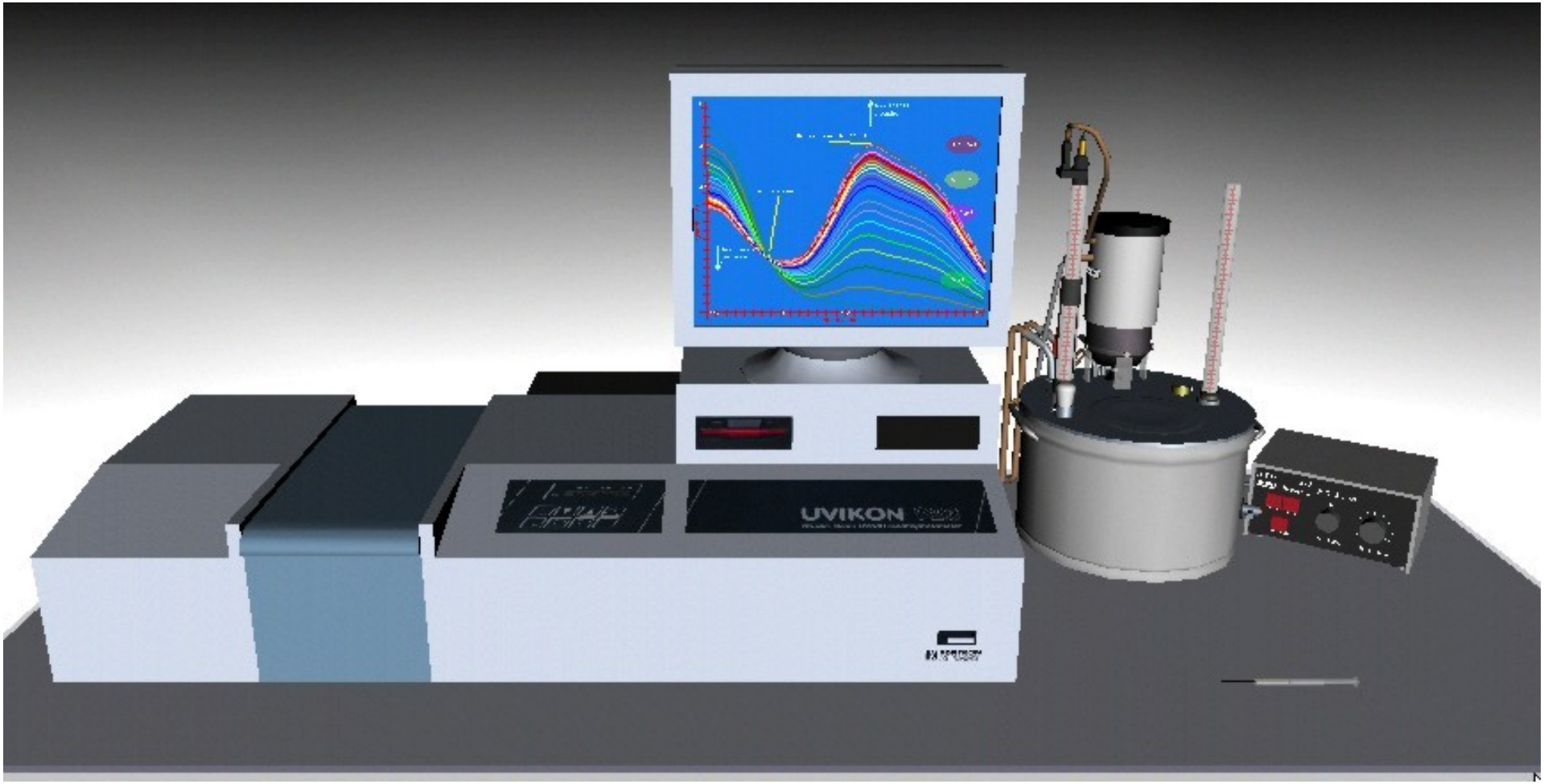
HVR: Modello astratto di un laboratorio reale



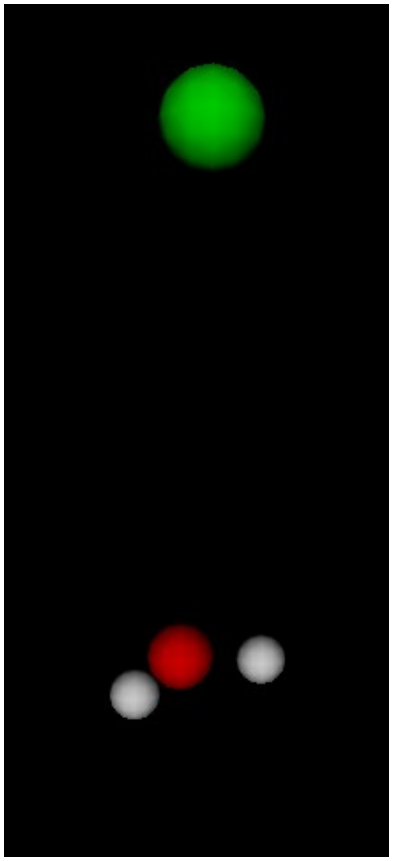
Spectroscopia atomica

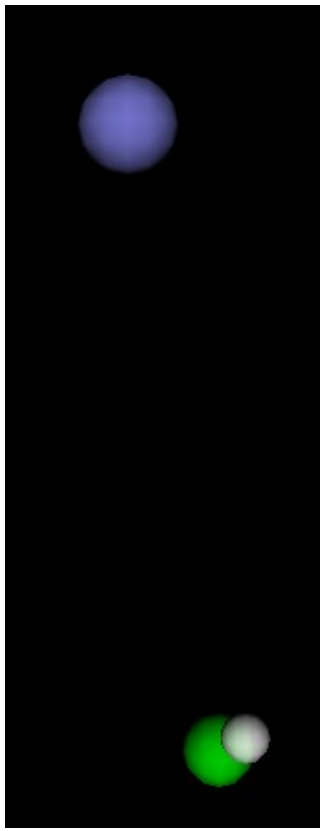


UV-Vis spectroscopy



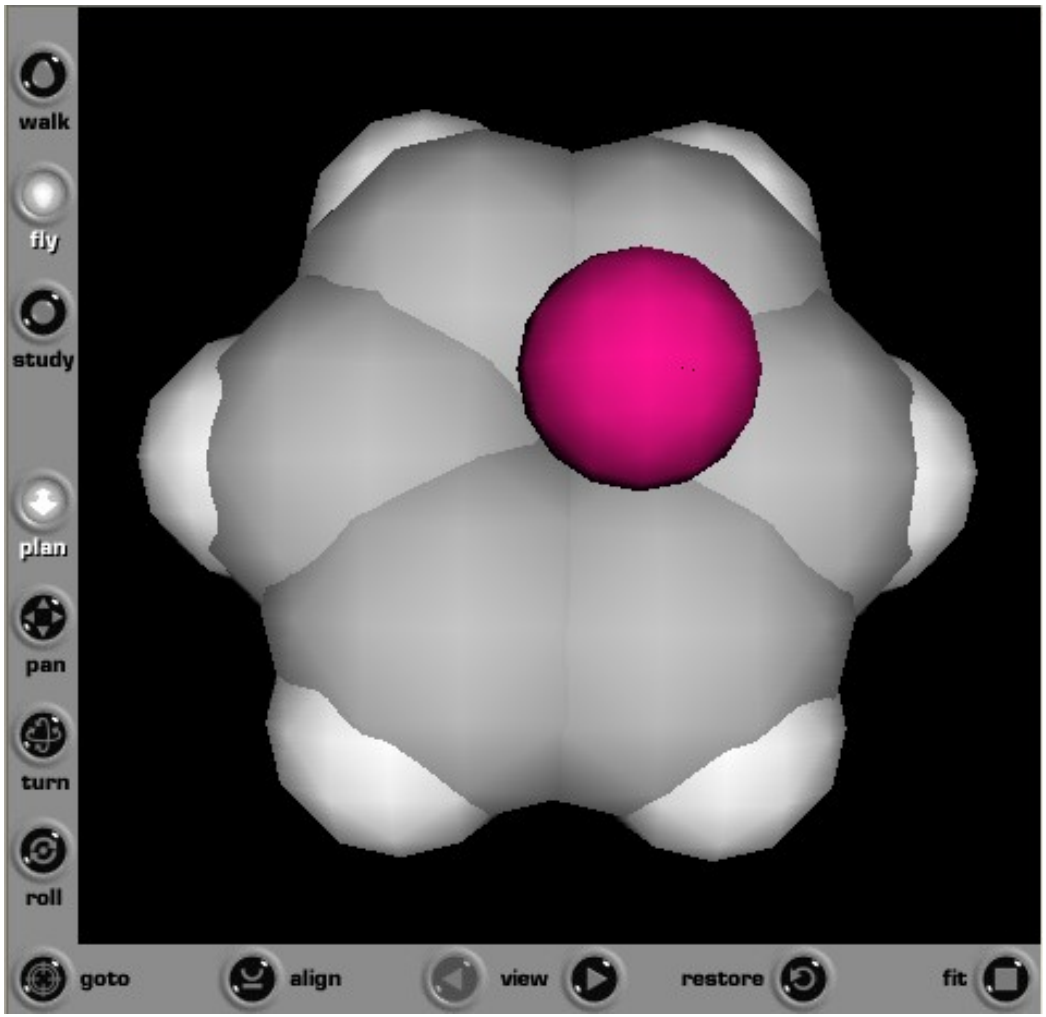
MVR per reazioni chimiche





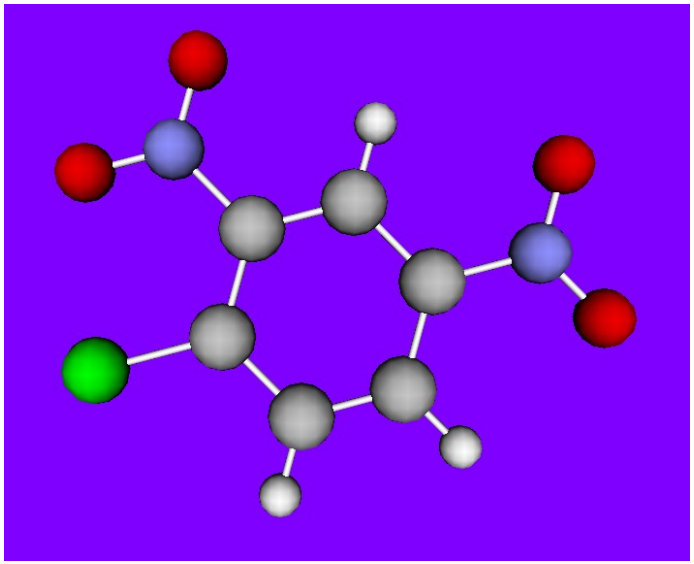
Simulazione di una reazione in fase gassosa di interesse nella chimica dell'atmosfera basata su calcoli di molecular dynamics (in ambiente Grid)

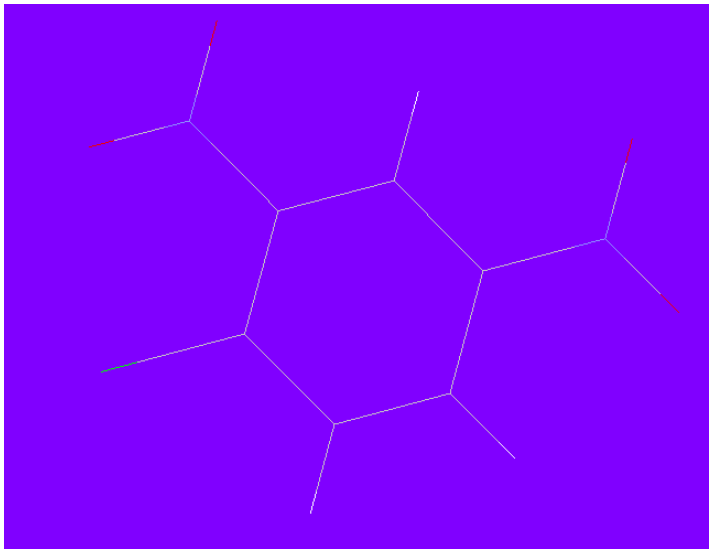
Rappresentazione della dinamica di sistemi

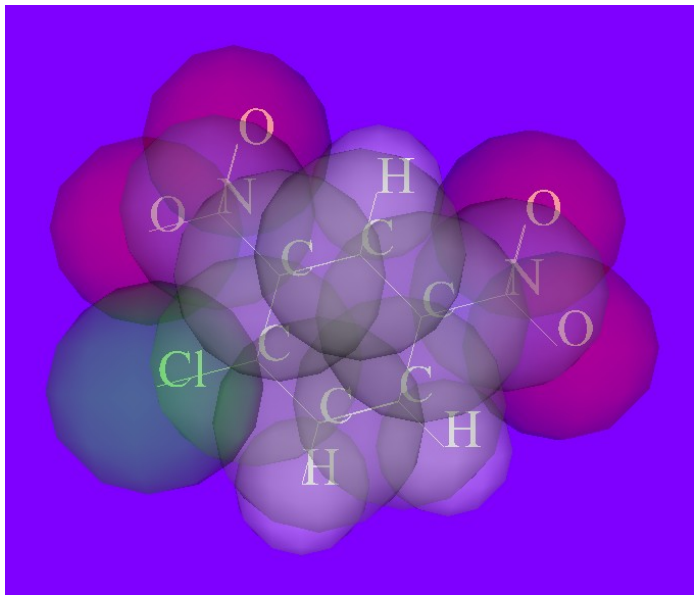


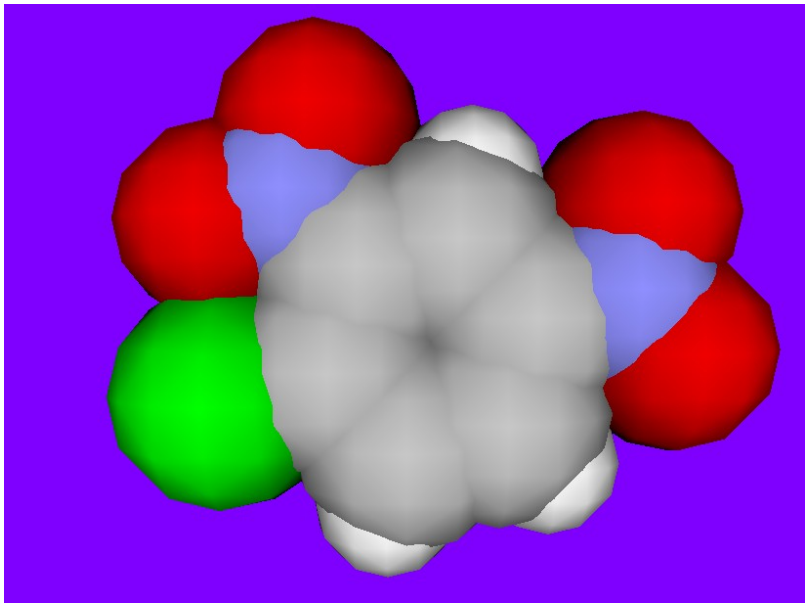
Molecular dynamics simulation
di benzene-atomi di gas nobile

Rappresentazione di molecule



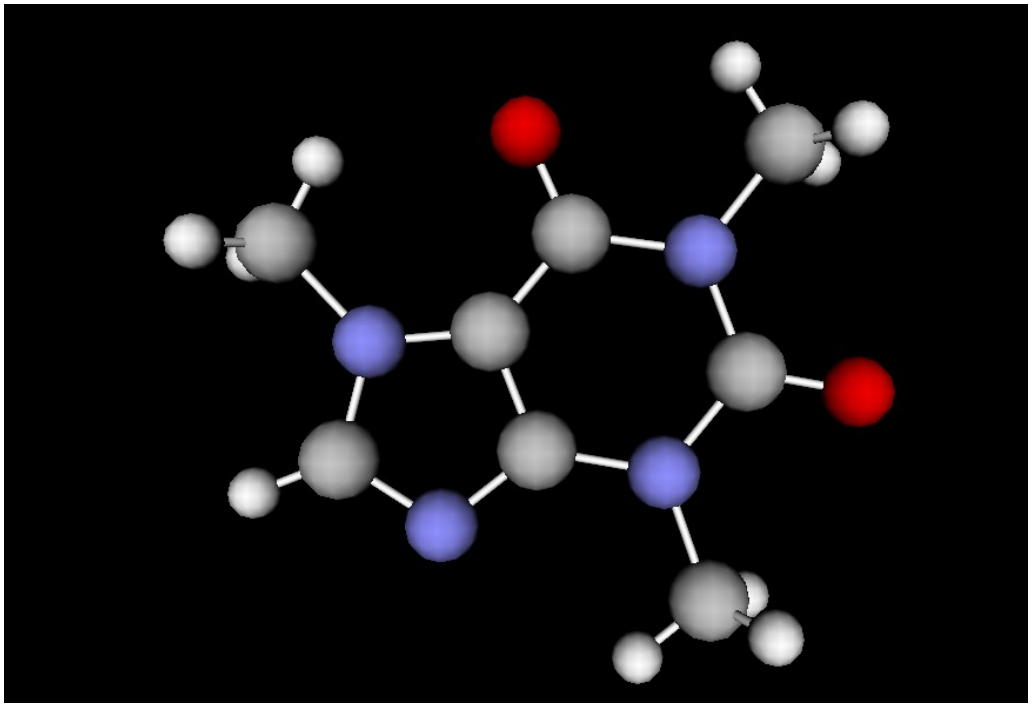






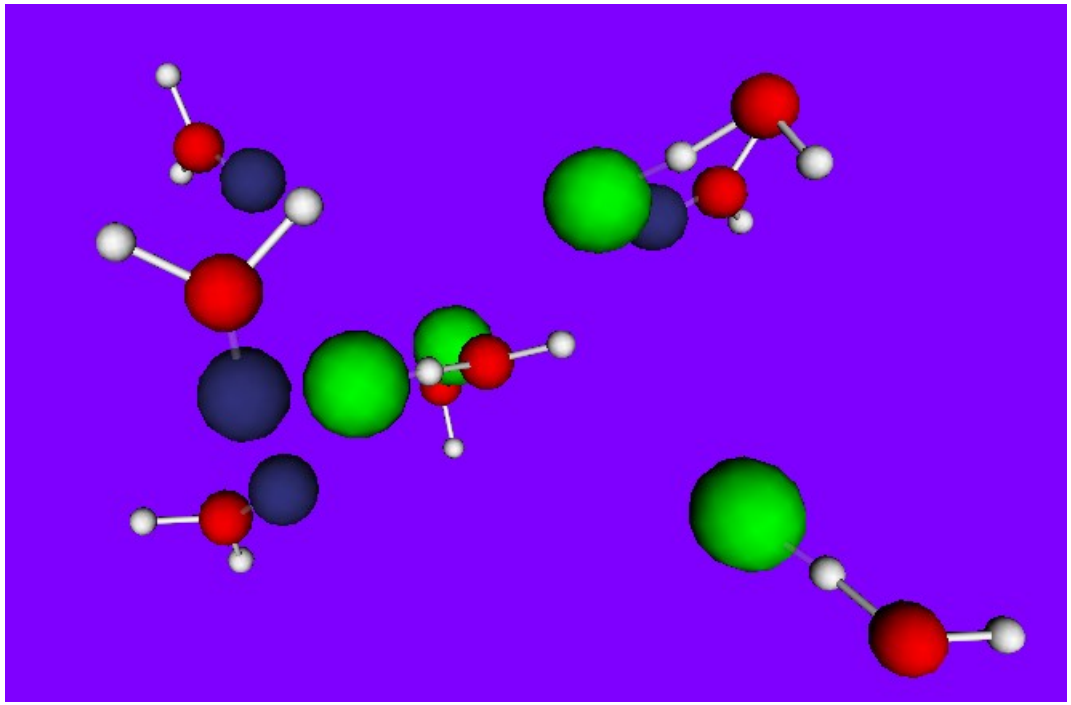
Rappresentazione di formati mol2, pdb … (ball & sticks, wire frame, space filling etc)

Perché il thè verde è meno dannoso per la salute di thè e caffè?



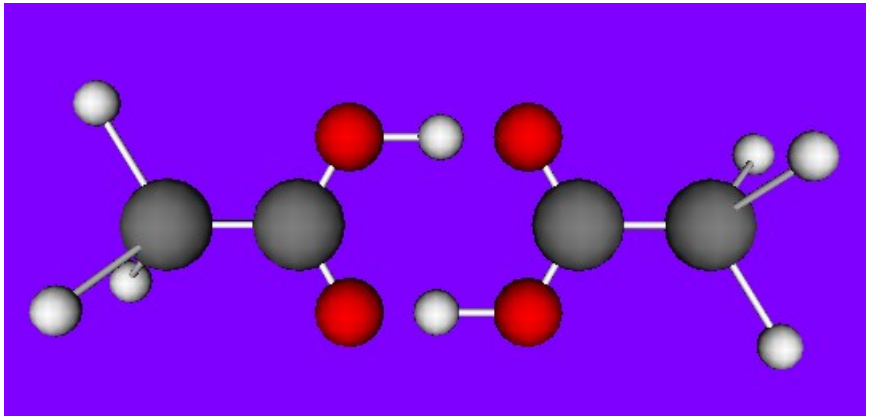
Cliccando sulla struttura è possibile passare da Coffein a Theophylline ed a Theobromine

Solubilizzazione (simulated dynamics)



Attacco di un singolo cristallo NaCl da parte dell'acqua

Dimero dell'Acido Acetico (simulated)



Docking mediante formazione di legami idrogeno

Virtual Reality Modeling Language

VRML

Introduzione

VRML

- Dopo la nascita del Web, uno dei fenomeni più affascinanti è stato lo sviluppo del Virtual Reality Modeling Language (VRML), che viene pronunciato Vermal.
- VRML è un linguaggio usato per descrivere simulazioni interattive con più partecipanti (mondi virtuali) distribuite nella rete Internet ed interconnesse mediante hyperlink attraverso il World Wide Web
- VRML trasforma il concetto di *home-page* in *home-space*
- La vista dei modelli VRML è di tipo *not fully immersive,* attuandosi mediante schermi grafici e mouse

VRML (i)

- La sintassi e la struttura dei dati del VRML forniscono uno strumento eccellente per la modellizzazione di mondi 3D interattivi, che possono essere inseriti in sistemi di visualizzazione *fully immersive*
- L'evoluzione dello standard VRML è stata la seguente: versione 1.0, versione 2.0 ed infine la versione corrente, è divenuta standard ISO/IEC con il nome VRML97
- I mondi possono contenere oggetti che hanno hyperlinks verso altri mondi, documenti HTML, o altri tipi MIME validi.
- Quando un utente seleziona un oggetto con un hyperlink, viene attivato l'appropriato visualizzatore per l'oggetto MIME in questione.

VRML(ii)

- La più interessante proprietà di VRML è quella di consentire la creazione in Internet di mondi dinamici e ambienti virtuali altamente interattivi. In particolare viene fornita la possibilità di:
  - Animare gli oggetti nei propri mondi, facendoli muovere
  - Suonare brani musicali e proiettare video nei propri mondi
  - Consentire agli utenti di interagire con i propri mondi
  - Controllare e potenziare i mondi mediante script, programmi che creiamo per agire nei nostri mondi

MIME Content types

- Multipurpose Internet Mail Extensions (MIME) è uno standard software che definisce (RFC 1521) un semplice meccanismo per descrivere il contenuto di un file trasmesso in rete Internet
- Ogni browser Web capisce i MIME Content Types e li usa per rappresentare l'informazione nella finestra del browser
- La definizione avviene attraverso due campi separati da `/`. Il primo descrive il tipo generale, il secondo il sottotipo e serve a specificare l'esatto formato.

  VRML usa **`x-world/x-vrml`**

  In futuro sarà **`model/vrml`**

Helper apps / Plug-ins

- Quando un browser Web segue un iperlink, scarica il file dal Web e verifica il MIME Content Type.
- Per visualizzare oggetti VRML, il browser può passare l'informazione ad un helper application, cioè un programma in grado di interpretare contenuto e formato del file, o un plug-in, programma che consente di vedere contenuti non-HTML nella finestra del browser.
- Uno dei prodotti più popolari è Cosmo Player disponibile come VRML Browser per IE o come Plug-in per Netscape

URL

- Uniform Resource Locator è definito negli RFC 1736 e 1808.
- Un URL Web è composto da tre campi:
  - Il nome del protocollo di comunicazione necessario per acccedere al file
  - Il nome del computer, o host, in Internet
  - Il percorso da seguire per scaricare il file dall'host

  Es: `http://www.web3d.org/vrml`

  il deposito VRML del Consorzio Web3D e

  nostro sito di riferimento

VRML 2.0

- Le caratteristiche del VRML 2.0 sono pubblicate in un documento di specifica pubblico, disponibile in rete, anche nel repository VRML:

  `http://www.web3d.org/vrml/spec.htm`

- Dalla stessa pagina sono consultabili le specifiche delle precedenti versioni di VRML (VRML 1.0, VRML97, X3D, etc)
- VRML 2.0 è divenuto uno standard dell'International Standards Organization (ISO): ISO/IEC14772
- Recentemente è stata rilasciata la specifica del nuovo standard: X3D

Il file VRML

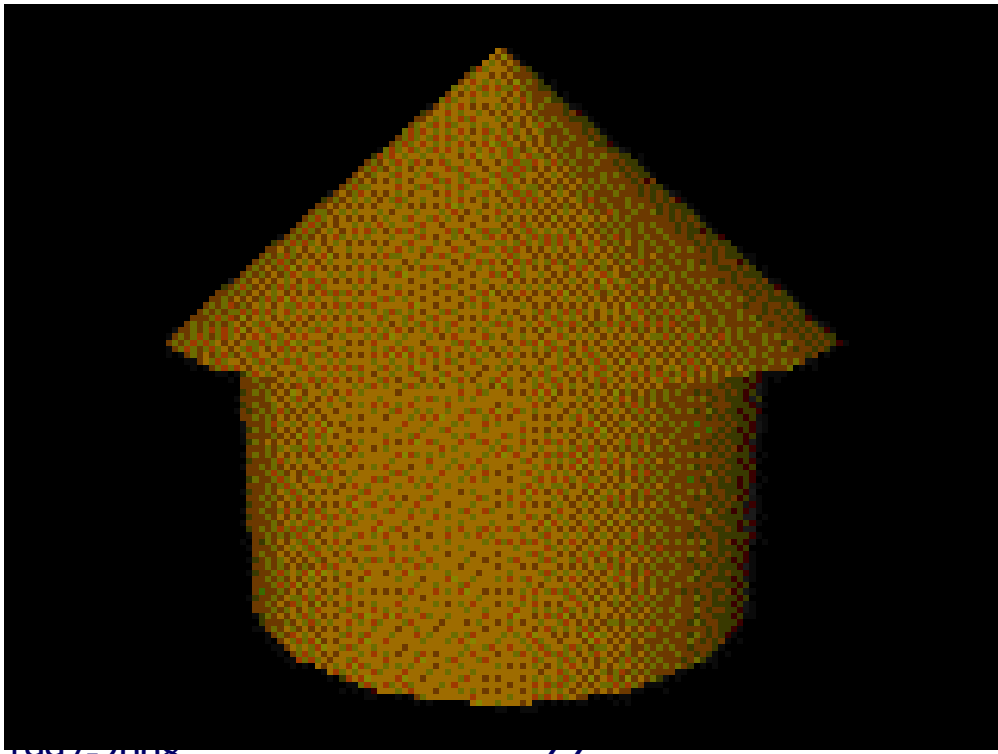
- Il file VRML è la descrizione in testo del mondo VRML
- Può essere creato con qualsiasi Editor o Word Processor
- Può essere creato anche da strumenti di programmazione visuale 3D o da programmi che salvano in linguaggio VRML oggetti e mondi creati con altri strumenti software
- I file VRML terminano con `.wrl` (dot world)
- Quando il browser legge il file `.wrl`, costruisce il mondo ivi descritto; quando ci si muove nel mondo, il browser disegna il mondo

Parti di un file VRML

- Un file VRML può contenere quattro tipi principali di componenti:
  - Header VRML
  - Prototypes
  - Shapes, interpolators, sensors, and scripts
  - Routes
- L'unica componente obbligatoria è il VRML Header
- Inoltre possono essere presenti:
  - Comments
  - Nodes
  - Fields e Field Values
  - Defined node names
  - Used node names

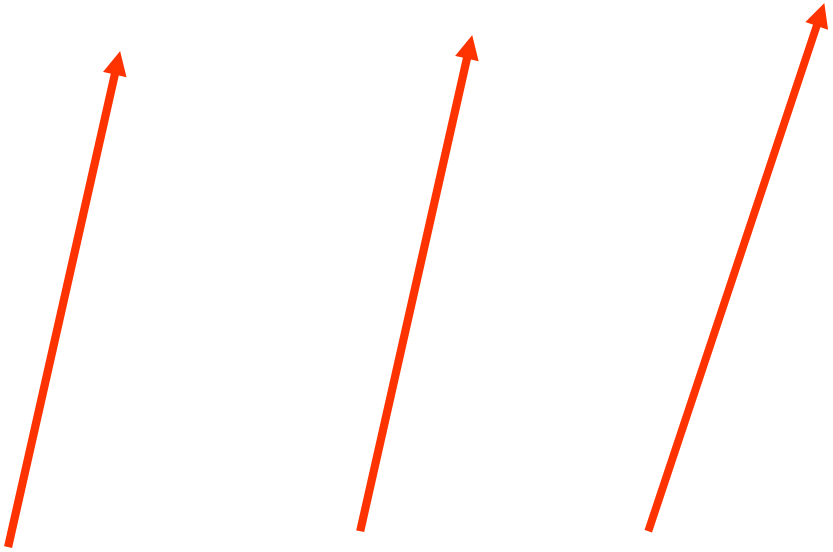
Esempio

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
# A brown hut
Group {
    children [
    # Draw the hut walls
        Shape {
            appearance DEF Brown Appearance {
                material Material {
                    diffuseColor 0.6 0.4 0.0
                }
            }
            geometry Cylinder {
                height 2.0
                radius 2.0
            }
        },
    # Draw the hut roof
        Transform {
            translation 0.0 2.0 0.0
            children Shape {
                appearance USE Brown
                geometry Cone {
                    height 2.0
                    bottomRadius 2.5
                }
            }
        }
    ]
}
```



The VRML header

```
#VRML V2.0 utf8
```



VRML

versione 2.0

The UTF8 Character Set

UCS Transform Format,

(UCS: Universal Character Set)

ISO 10646-1:1993

I Nodi

- Un file VRML contiene dei Nodi che descrivono aspetto e comportamento del mondo virtuale.
- Questi sono le piccole pietre miliari di VRML.
- I nodi descrivono aspetto, colore, luce, punti di vista, come posizionare ed orientare shapes, animation, timers, sensors, interpolators.
- Un nodo contiene:
  - Tipo di nodo (richiesto)
  - Una coppia di parentesi graffe (richiesto)
  - Un certo numero di campi entro le parentesi (opzionali) ed i loro valori che definiscono gli attributi del nodo

I Nodi (i)

Il cilindro visto nella precedente figura è definito da:

```
Cylinder {
        height 2.0
        radius 2.0
}
```

Raggruppano tutte le informazioni all interno del nodo

Queste sono unità VRML arbitrarie

Rappresentano una singola entità nel nostro mondo e definiscono le proprietà del nodo. L ordine è irrilevante. Ogni proprietà ha associato un valore di default (ad es. **`radius`** ha default 1.0)

I Nodi (ii)

- Ogni *field* ha associato un *field value* che definisce attributi quali colore, dimensioni o posizione
- Ogni valore è di un certo *field type* che descrive il tipo di valore consentito in quel *field*. Essi hanno nomi come SFColor e SFImage
- Molti field types sono presenti in due varietà: *single-value types* e *multiple-value types*.
- *single-value types* sono un singolo valore, come un singolo colore o un singolo numero, ed il nome inizia con SF
- *multiple-value types* possono essere più valori , come una lista di colori e di numeri ed hanno nomi che iniziano per MF. La lista va racchiusa tra parentesi quadre e gli elementi possono essere separati da virgola (opzionale)

Field Value Types summary

SFBool	valori Boolean o Logical, possono essere **TRUE** o **FALSE**. Tipicamente usati per attivare o disattivare una proprietà o una forma
SFColor/	Un gruppo di tre valori floating-point. Descrive la quantità di rosso, verde e blu che devono essere mescolati per formare il colore desiderato.
MFColor	Usato in genere per selezionare il colore di una forma o di una luce.
SFFloat/	Valori floating-point.Grandi o piccoli, valori positivi o negativi, con la virgola e campi decimali. Es: 88.5, 3.1415, -489.398
MFFloat	Usati, per esempio, per definire altezza e raggio di un cilindro
SFImage	Lista di valori che descrivono i colori di una immagine digitale. Usata per dare struttura alla superficie di forme colorate
SFInt32/	valore intero a 32 bit. Valori grandi o piccoli, positivi o negativi, senza punto decimale.Es: 42, 182, -37.
MFInt32	Usato per indicare una selezione tra diverse possibilità discrete
SFNode/	Un nodo VRML. Es: **Cylinder**.
MFNode	Tipicamente usato per indicare un nodo proprietà che controlla come un nodo forma disegnerà la forma stessa.
SFRotation/	Un gruppo di 4 valori floating-point. I primi tre valori definiscono un asse di rotazione. L'ultimo valore indica un angolo di rotazione in radianti
MFRotatic	Tipicamente utilizzato per indicare come orientare una forma
SFString/	Una lista di caratteri racchiusi tra apici.
MFString	Tipicamente usata per specificare il nome di una scelta tra diverse opzioni
SFTime	Un valore floating-point. Fornisce il tempo assoluto del mondo reale, misurato in secondi dalla mezzanotte del 1 gennaio 1970 (GMT).
	Tipicamente usato per determinare quando attivare o fermare un'animazione
SFVec2f/	vettore 2-D floating-point. Un gruppo di due valor floating-point
MFVec2f	Tipicamente usata per specificare una posizione 2-D
SFVec3f/	vettore 3-D floating-point. Un gruppo di tre valor floating-point
MFVec3f	Tipicamente usata per specificare una posizione 3-D

Nomi dei nodi

- Si possono definire dei nomi per ciascun nodo nel nostro mondo. I nomi possono essere una qualsiasi sequenza di lettere, numeri e il carattere `_`.
- Una volta che un nodo ha un nome, questo può essere riutilizzato successivamente nel file.
- Se definisco un nodo **`my_chair`** per un nodo o un gruppo di nodi che rappresentano una sedia e voglio mettere quattro sedie intorno al tavolo, si riusa la stessa forma **`my_chair`** tre volte, senza ripeterne la descrizione
- Il nodo con il nome definito è chiamato *original*, ed ogni volta che viene riusato viene detta *istanza.*

Nomi dei nodi (i)

- Se si modifica la definizione della forma originale, anche le istanze vengono immediatamente aggiornate
- Ciò consente di modificare rapidamente i nostri mondi, modificando lo stile di una sedia, di una finestra, di una porta e di ogni altra forma alla quale sia stato dato un nome.
- Un file VRML può contenere un numero qualsiasi di nodi ai quali sia stato assegnato un nome. Ovviamente non si possono definire due forme con lo stesso nome

SINTASSI:

```
DEF node-name node-type {...}
```

Definizione dei Nomi di Nodo

- I nomi sono case-sensitive
- non possono iniziare con un numero
- sono inoltre proibiti i seguenti nomi:

```
DEF         EXTERNPROTO   FALSE          IS      NULL
PROTO       ROUTE         TO             TRUE    USE
eventIn     eventOut      exposedField   field
```

- Un node type è un tipo particolare di nodo, come il nodo **`Cylinder`**. Per definire un nome per il nodo Cylinder usando **`DEF`**, si ha:

Tipo del Nodo che viene definito

```
DEF my_cylinder Cylinder { … }
```

Nome del nodo

Uso dei Nomi di Nodo

- Una volta definito un nome per un nodo, lo si può riusare diverse volte all'interno dello stesso file, precedendo il nome del nodo dalla parola **`USE:`**

  `USE` ***`node-name`***

- Si può usare un nodo ovunque nel file, dove può essere specificato
- Tutte le istanze condividono la stessa definizione del nodo, pertanto se si cambia il nodo originale, vengono aggiornate anche tutte le istanze

Costruzione di Forme

- Come in un progetto di un edificio l'architetto specifica i vari blocchi della costruzione - materiali usati, aspetto della costruzione, etc - in VRML gli elementi base della costruzione di un mondo sono le forme (shapes) che vengono descritte dai nodi, dai campi che li definiscono e dai valori associati.
- Una forma VRML ha una forma, o geometria, che definisce la sua struttura 3-D; ha una apparenza basata sul materiale, un colore, e la struttura (texture) della superficie (es: legno o mattone)
- In VRML questi attributi della forma - geometria e apparenza - sono specificati da valori dei campi di un nodo **`Shape`**

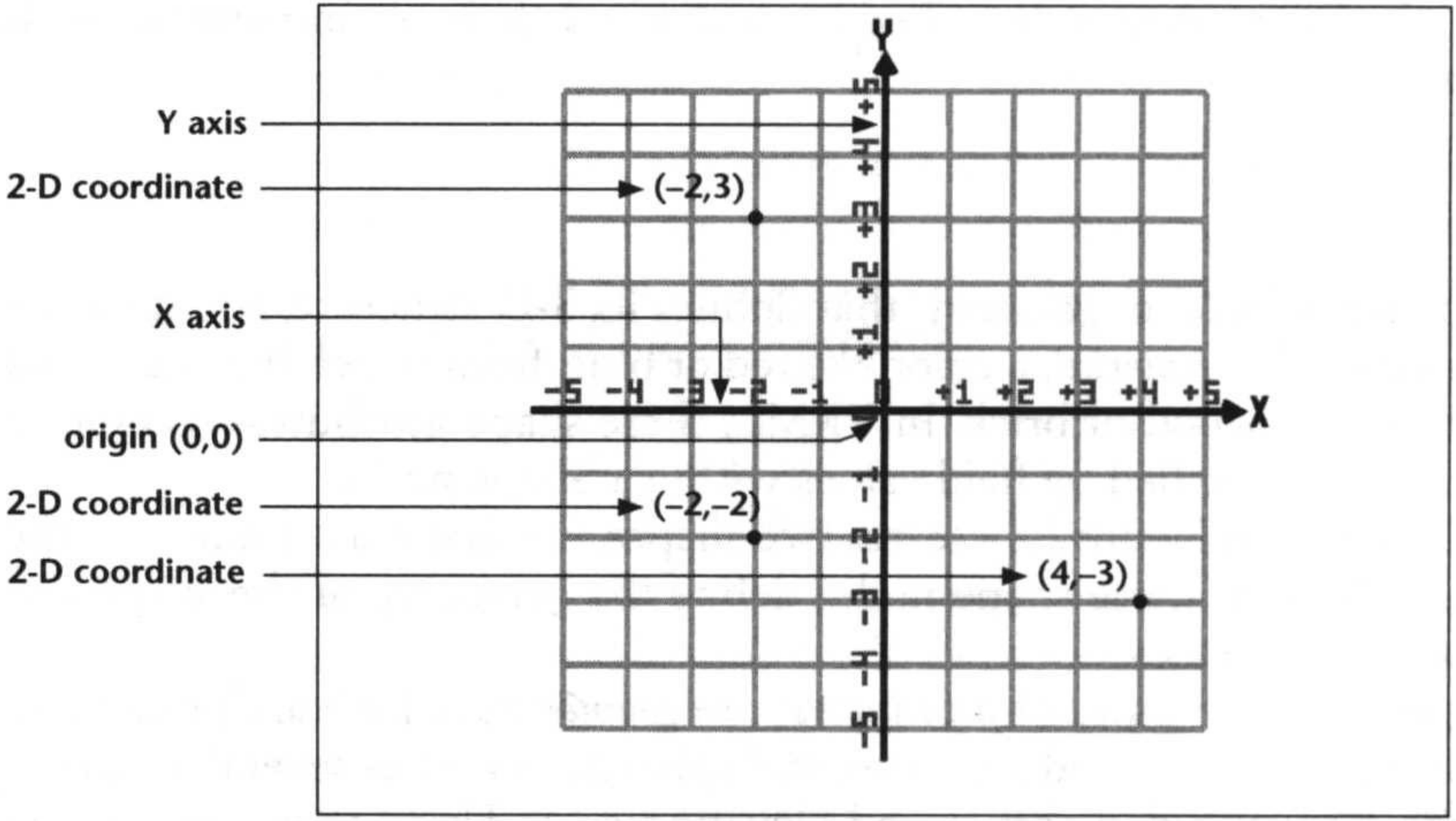
Descrizione di Forme

- L'esempio precedente crea due shapes, la prima è un cilindro, la seconda un cono; entrambi i nodi shape definiscono le forme corrispondenti e condividono una apparenza comune
- VRML supporta diversi tipi di *primitive shape geometries*, che sono predefinite nel linguaggio (scatole, cilindri, coni e sfere) e *advanced shape geometries* (extruded shapes, elevation grids)
- Usando queste forme si possono creare forme più complesse, le quali possono essere usate per dar luogo a forme ulteriormente più complesse, etc.

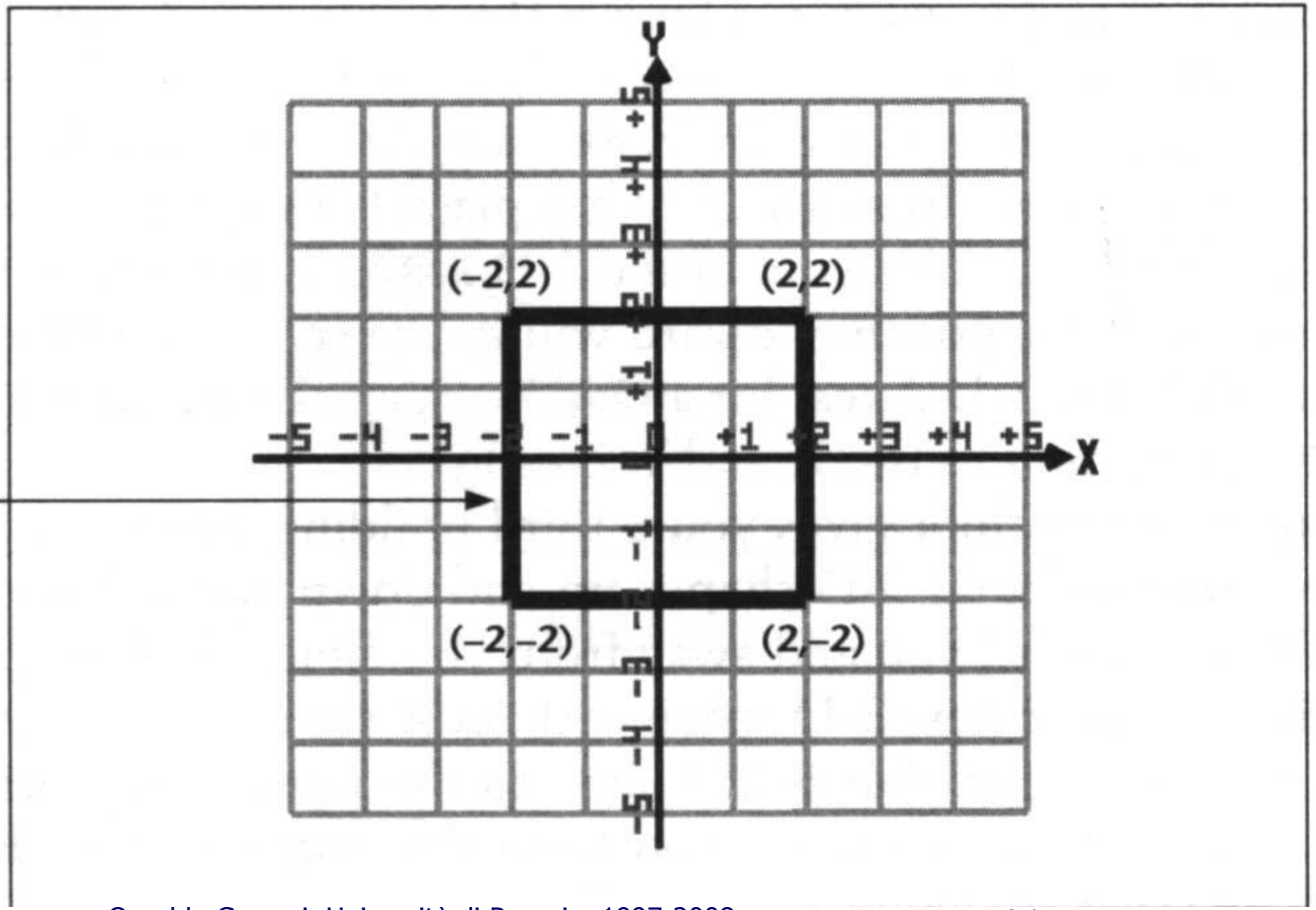
Raggruppamento di Forme

- Le forme possono essere raggruppate per creare forme più complesse.
- Nella figura vista il cono e il cilindro sono stati raggruppati usando il nodo **`Group`**, per creare una forma a cappello.
- Il nodo che raggruppa insieme le forme del gruppo, è chiamato *padre*.
- Le forme che costituiscono il gruppo, sono chiamate *figli* del gruppo.
- Un gruppo può avere un numero qualsiasi di figli
- Un gruppo può avere altri gruppi come figli
- Un gruppo parte di un altro gruppo è detto *nested* nel gruppo più grande

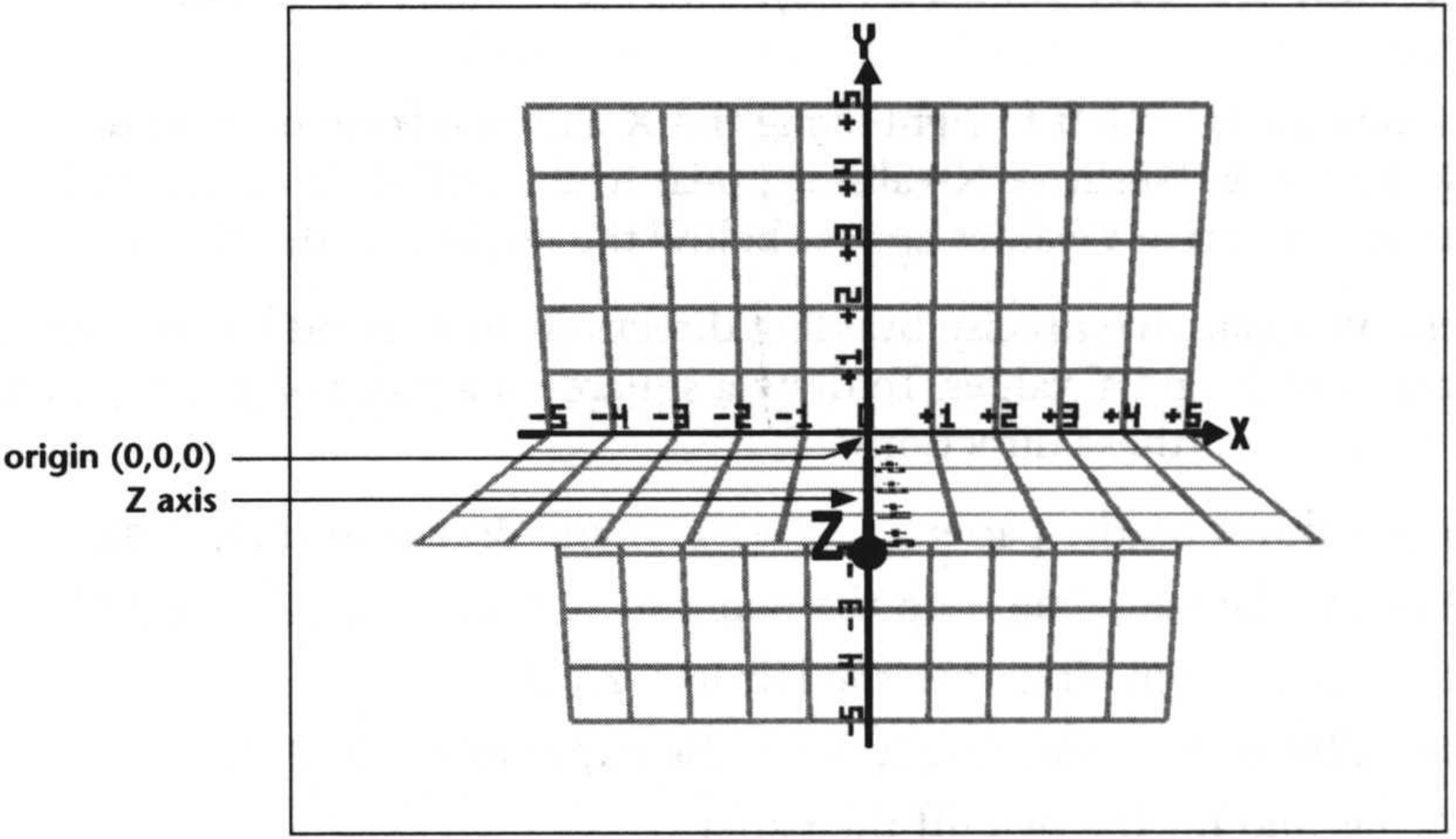
VRML space


Y axis
2-D coordinate
X axis
origin (0,0)
2-D coordinate
2-D coordinate
(–2,3)
(–2,–2)
(4,–3)
Y
X
–5 –4 –3 –2 –1 +1 +2 +3 +4 +5
+5 +4 +3 +2 +1 –1 –2 –3 –4 –5

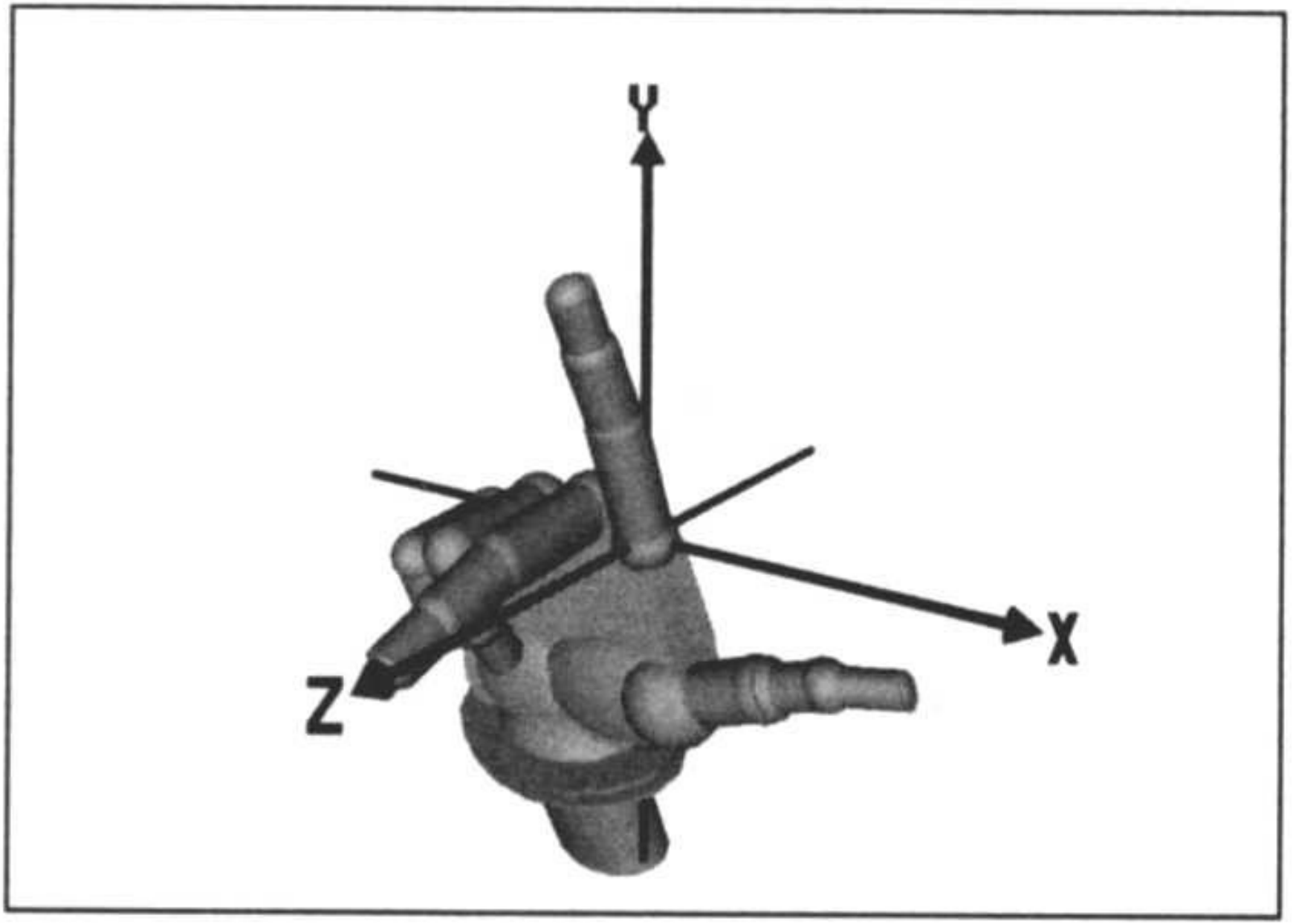
Quadrato in 2-D



Spazio 3-D

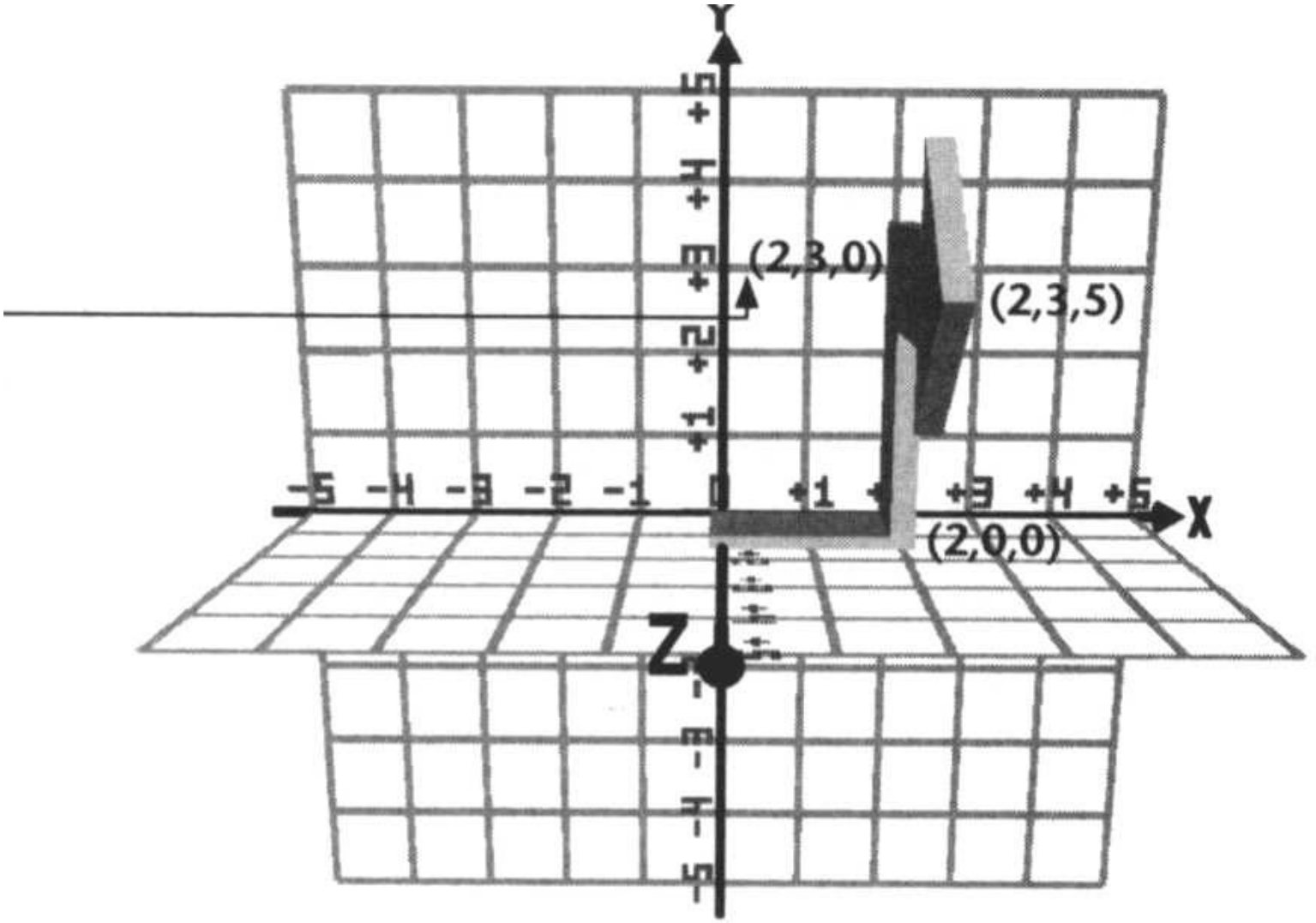


La regola della mano destra



coordinata 3-D

Coordinata 3-D



Virtual Reality Modeling Language

VRML

Forme Predefinite

Eventi e Routes

- Il file VRML contiene le istruzioni per la costruzione del nostro mondo: per renderlo dinamico occorrono delle istruzioni che indichino i legami (VRML wiring) tra vari nodi, definendo dei percorsi che leghino degli eventi ai nodi interessati
- Ciò al fine di definire come per esempio, cliccando una forma possa accendersi una luce, attivare un suono o accendere una macchina
- VRML wiring comporta:
  - una coppia di nodi da collegare insieme
  - un wiring route, o path, tra i due nodi, che in genere è posta alla fine del file:

```
ROUTE NodeName.eventOutName TO NodeName.eventInName
```

Il nome deve essere definito prima dell'istruzione ROUTE

Eventi e Routes (i)

- Una volta stabilito il path, il primo nodo può inviare un messaggio al secondo usando questo path
- Tale messaggio è chiamato un evento, e contiene un valore. Valori tipici sono valori floating-point, valori di tipo colore o coordinata 3-D.
- Quando un nodo riceve un evento può reagire con un'azione (accendendo una luce, riproducendo un suono, attivando un'animazione, etc), in funzione delle proprietà del nodo stesso
- Collegando diversi nodi tra loro si possono realizzare dei complessi circuiti attraverso i quali possono essere istradati degli eventi per rendere il mondo dinamico

Nodi Input e Output

- Diversi nodi possono essere collegati in circuito
- Ciascun nodo ha un connettore (jack) di input e/o uno di output, ai quali possono essere collegati i fili del circuito
- I nodi che creano luci nel nostro mondo hanno dei connettori di input che accendono e spengono la luce
- Collegandoli si possono creare dei circuiti per l'accensione e lo spegnimento automatico delle luci
- Possono esserci diversi tipi di connettori in input e output. Per es: una luce ha connettori anche per cambiare l'intensità, il colore o altre proprietà della luce

Nodi Input e Output (i)

- Un connettore di input di un nodo è chiamato **`eventIn`** e riceve eventi quando è connesso ad una route e gli viene inviato un'evento
- Un connettore di output di un nodo è chiamato **`eventOut`** ed invia eventi quando è connesso ad una route
- Consideriamo un sottoinsieme della definizione del nodo **`Collision`**:

Nodi Input e Output (ii)

- Il nodo ha due input jack (**addChildren** e **removeChildren**) e un output jack (**collideTime**)
- Il nodo ha due field: **children** e **proxy**
- Il campo è un exposedField. Un exposedField ha due jack associati di default: un input jack che serve per definirne il valore ed un output jack che serve a inviare il valore del campo ogni volta che il campo cambia.
- L'input jack implicito si chiama sempre **set_XXX**, dove **XXX** è il nome dell'exposedField.
- Analogamente l'output jack implicito si chiama sempre **XXX_changed**
- Per l'exposedField **children** del nodo **Collision** l'input jack implicito si chiama **set_children** e l'output jack implicito **children_changed**

Node Input and Output (iii)

- Quando si costruiscono le route, si specificano i nomi dei dei nodi **`eventIn`** e **`eventOut`**, per indicare dove si connettono le route
- Per il nodo **`Collision`** si può creare una route che connette il nodo **`eventIn`** (**`addChildren`** e **`removeChildren`**) al nodo **`eventOut`** (**`collideTime`**)
- Per un exposedField si può attivare una route per gli eventIn ed eventOut impliciti (**`set_children`** e **`children_changed`**)
- Si può creare una route per lo stesso exposedField (children): il VRML browser utilizzerà automaticamente Node Input and Output (ii) **`eventIn`** implicito (se route invia un evento al nodo) o **`eventOut`** (se il nodo è il ricevente di eventi sulla route)

Costruzione e Raggruppamento di Forme Predefinite

- Una forma VRML ha un'apparenza e una geometria definita mediante un nodo `Shape`.
- L'apparenza della forma è definita dai nodi `Appearance` e `Material`.
- Il VRML fornisce molti nodi relativi a geometrie primitive che possono essere usati con il nodo `Shape` per costruire forme primitive.
- Le forme primitive sono sempre costruite centrate all'origine. Vedremo più avanti come spostare questa origine per costruire forme ovunque nel mondo
- Le primitive generano solidi pieni

Nodo Shape

```
Shape {
        appearance   NULL      # exposedField      SFNode
        geometry     NULL      # exposedField      SFNode
      }
```

- Tutte le forme sono costruite in VRML usando il nodo **Shape**
- Il valore del campo **geometry** specifica un nodo che definisce la forma 3-D, o geometria, della forma. Valori tipici di questo campo comprendono i nodi delle geometrie primitive **Cone**, **Cylinder** e **Sphere**
- Il valore di default **NULL** indica assenza di geometria

Nodo Shape (i)

- Il valore dell'exposedField `geometry` può essere cambiato istradando un evento sull' exposedField implicito `set_geometry` **`eventIn`**.
- Quando l'evento è ricevuto, viene definito il campo `geometry` e la nuova geometria viene propagata all'exposedField implicito `geometry_changed` **(`eventOut`)**
- Il valore del campo `appearance` specifica un nodo che definisce l'apparenza della forma, incluso il colore e la struttura della superficie

Nodo Shape (ii)

- Il campo **appearance** include per default il nodo **Appearance** ed il valore di default **NULL**, corrispondenti a un aspetto luminescente bianco
- Il valore dell'exposedField **appearance** può essere cambiato istradando un evento sull' exposedField implicito **set_appearance (eventIn).**
- Quando l'evento è ricevuto, viene definito il campo **appearance** e la nuova apparenza viene propagata all'exposedField implicito **appearance _changed (eventOut)**

Nodo Appearance

```
Appearance {
      material          NULL     # exposedField     SFNode
      texture           NULL     # exposedField     SFNode
      textureTransform  NULL     # exposedField     SFNode
       }
```

- Il nodo **Appearance** specifica gli attributi di apparenza e può essere usato come valore del campo **appearance** in un nodo **Shape**
- Il valore del campo **material** specifica un nodo che definisce gli attributi materiali dell'apparenza. Valore tipico di questo campo è il nodo **Material**
- Il valore di default **NULL** indica un materiale luminoso bianco
- Vedremo in dettaglio più avanti i campi **texture** e **textureTransform**

Nodo Material

```
Material {
        ambientIntensity  0.2          # exposedField SFFloat
        diffuseColor      0.8 0.8 0.8  # exposedField SFColor
        emissiveColor     0.0 0.0 0.0  # exposedField SFColor
        shininess         0.2          # exposedField SFFloat
        specularColor     0.0 0.0 0.0  # exposedField SFColor
        transparency      0.0          # exposedField SFFloat
         }
```

- Il nodo **Material** specifica gli attributi materiali e può essere usato come valore del campo **material** in un nodo **Appearance**
- Il valore di default del nodo **Material** crea forme bianche sfumate
- Torneremo con maggior dettaglio più avanti sul nodo **Material**

Nodo Box

```
Box {
      size        2.0 2.0 2.0  # Field SFVec3f
       }
```

- Il nodo **Box** crea primitive geometriche a forma di scatola e può essere usato come valore del campo **geometry** nel nodo **Shape**
- Il valore del campo **size** specifica le dimensioni di una scatola rettangolare 3-D centrata nell'origine
- Il primo valore indica la profondità della scatola lungo la direzione X, il secondo l'altezza della scatola nella direzione Y ed il terzo indica la profondità della scatola nella direzione Z. Tutti e tre i valori devono essere maggiori di `0.0`
- Le dimensioni di default sono di una scatola larga `2.0` unità, alta `2.0` unità e profonda `2.0` unità.

Nodo Cone

```
Cone {
      bottomRadius      1.0     # Field SFFloat
      height            2.0     # Field SFFloat
      side              TRUE    # Field SFBool
      bottom            TRUE    # Field SFBool
       }
```

- Il nodo **Cone** crea primitive geometriche di forma conica e può essere usato come valore del campo **geometry** nel nodo **Shape**
- Il valore del campo **bottomRadius** specifica il raggio della base di un cono 3-D centrato nell'origine, con il suo asse coincidente lungo l'asse Y.

Nodo Cone (i)

- Il valore di `bottomRadius` deve essere maggiore di `0.0`
- Per default `bottomRadius` crea un cono con raggio `1.0` unità
- Il valore del campo `height` specifica l'altezza del cono lungo l'asse Y.
- Il valore di `height` deve essere maggiore di `0.0`
- Per default `height` crea un cono alto `2.0` unità, con la base del cono posta `1.0` unità al di sotto dell'origine e la punta del cono `1.0` unità sopra l'origine.
- Il valore del campo `side` specifica se deve essere costruita la superficie obliqua del cono.

Nodo Cone (ii)

- Se il valore è **TRUE**, la superficie viene costruita, se è **FALSE**, no. Il default è **TRUE**.
- Il valore del campo **bottom** specifica se deve essere costruita la superficie di base del cono.
- Se il valore è **TRUE**, la superficie viene costruita, se è **FALSE**, no. Il default è **TRUE**.
- Se entrambe le opzioni **side** e **bottom** sono disattivate si crea un cono invisibile.
- Se si disattiva **bottom** ma non **side** si crea un cono senza base e si può guardare dentro di esso.

Nodo Cylinder

```
Cylinder {
      radius     1.0     # Field SFFloat
      height     2.0     # Field SFFloat
      side       TRUE    # Field SFBool
      top        TRUE    # Field SFBool
      bottom        TRUE    # Field SFBool
       }
```

- Il nodo **Cylinder** crea primitive geometriche di forma cilindrica e può essere usato come valore del campo **geometry** nel nodo **Shape**
- Il valore del campo **radius** specifica il valore del raggio di un cilindro 3-D centrato nell'origine, con il suo asse coincidente lungo l'asse Y.

Nodo Cylinder (i)

- Il valore di `radius` deve essere maggiore di `0.0`
- Per default `radius` crea un cono con raggio `1.0` unità
- Il valore del campo `height` specifica l'altezza del cono lungo l'asse Y.
- Il valore di `height` deve essere maggiore di `0.0`
- Per default `height` crea un cilindro alto `2.0` unità, con la base del cilindro `1.0` unità sotto l'origine e la cima del cilindro `1.0` unità sopra l'origine.
- Il valore del campo `side` specifica se deve essere costruita la superficie laterale del cilindro.

Nodo Cylinder (ii)

- Se il valore è **`TRUE`**, la superficie viene costruita, se è **`FALSE`**, no. Il default è **`TRUE`**.
- Il valore del campo **`bottom`** specifica se deve essere costruita la superficie di base del cilindro.
- Se il valore è **`TRUE`**, la superficie viene costruita, se è **`FALSE`**, no. Il default è **`TRUE`**.
- Se entrambe le opzioni **`side`** e **`bottom`** sono disattivate si crea un cilindro invisibile.
- Se si disattiva **`bottom`** ma non **`side`** si crea un cilindro senza basi e si può guardare dentro di esso.

Nodo Sphere

```
Sphere {
      radius     1.0     # Field SFFloat
            }
```

- Il nodo **Sphere** crea primitive geometriche di forma sferica (una palla o un globo) e può essere usato come valore del campo **geometry** nel nodo **Shape**
- Il valore del campo **radius** specifica il valore del raggio di una sfera 3-D centrata nell'origine.
- Il valore di **radius** deve essere maggiore di `0.0`
- Per default **radius** crea una sfera con raggio `1.0` unità

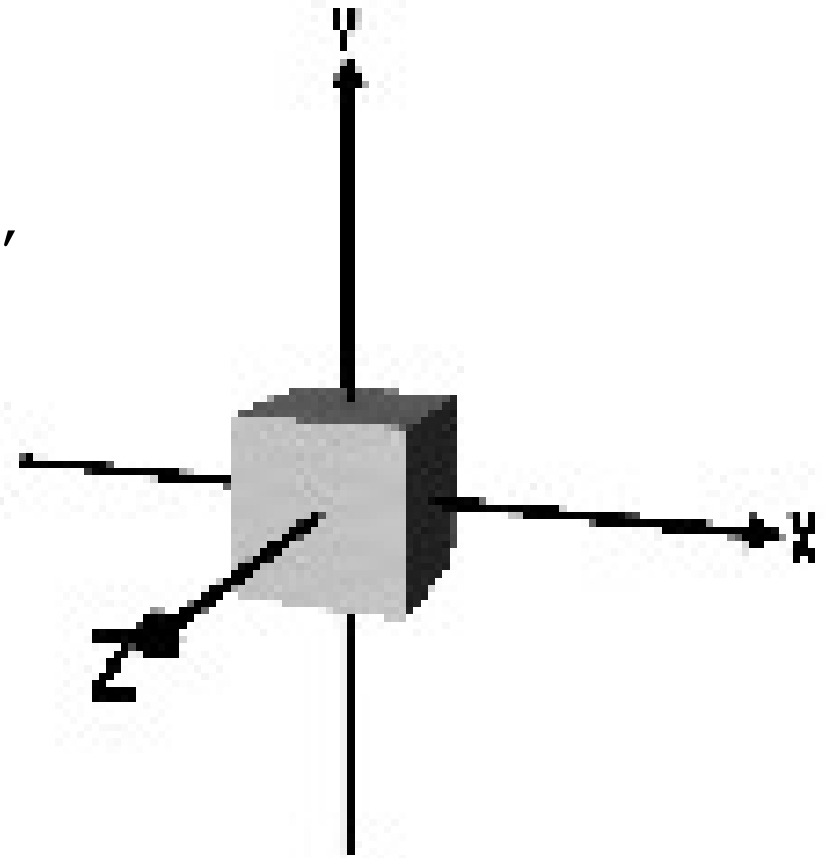
Nodo Group

```
Group {
        children    [ ]                 # exposedField MFNode
        bboxCenter      0.0 0.0 0.0     # field        SFVec3f
        bboxSize     -1.0 -1.0 -1.0     # field        SFVec3f
        addChildren                     # eventIn      MFNode
        removeChildren                  # eventOut     MFNode
                }
```

- I nodi VRML possono essere raggruppati usando il nodo **Group**
- Il valore del campo **children** specifica una lista di nodi che devono essere inclusi nel gruppo.
- Valori tipici di **children** comprendono nodi **Shape** e altri nodi **Group**
- Valore di default di **children** è una lista vuota

Esempi: default box

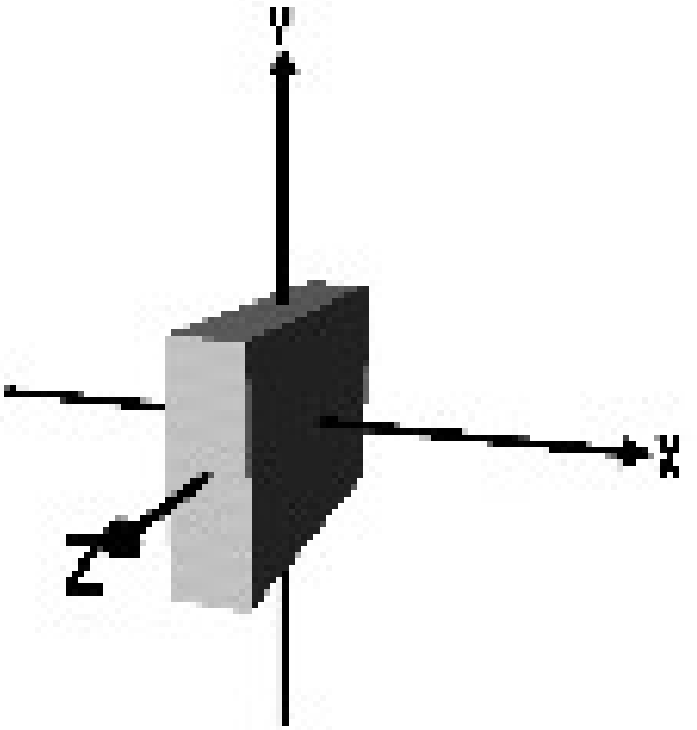
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Box { }
}
```



- scena VRML

Esempi: a box

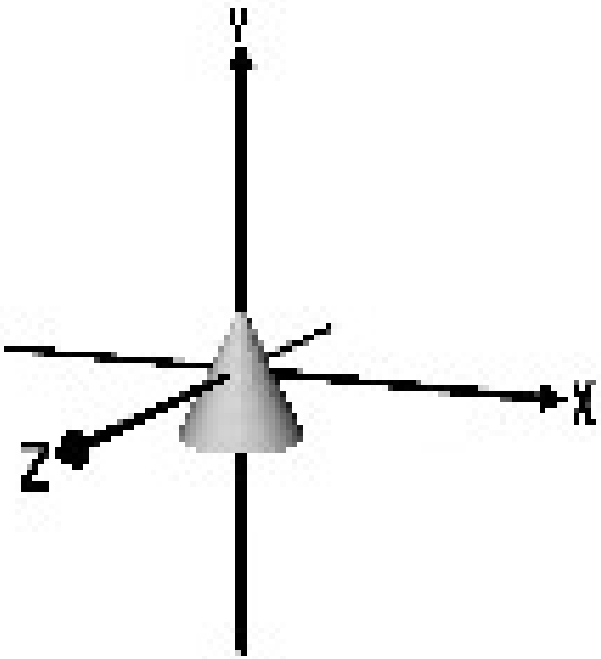
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
   # and John L. Moreland
Shape {
     appearance Appearance {
          material Material { }
     }
     geometry Box {
          size 1.0 3.0 5.0
     }
}
```



- scena VRML

Esempi: default cone

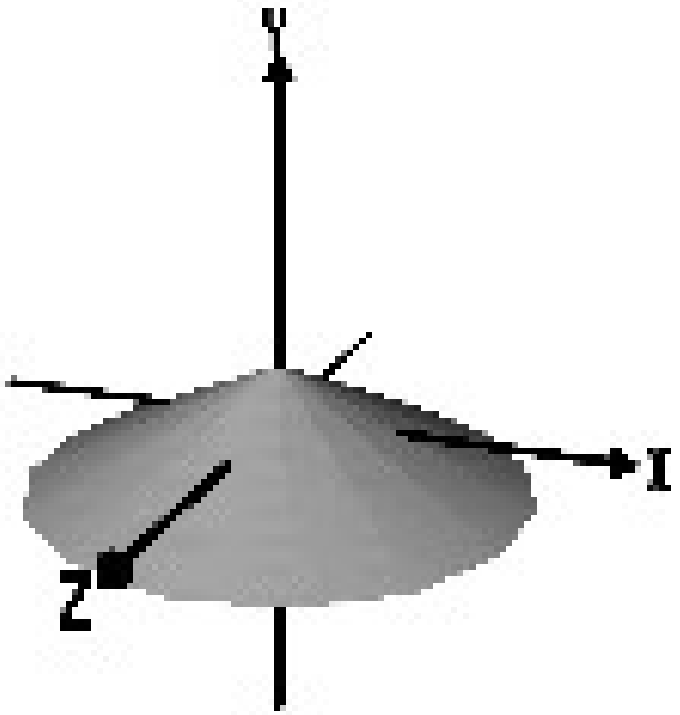
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Cone { }
}
```



- scena VRML

Esempi: a cone

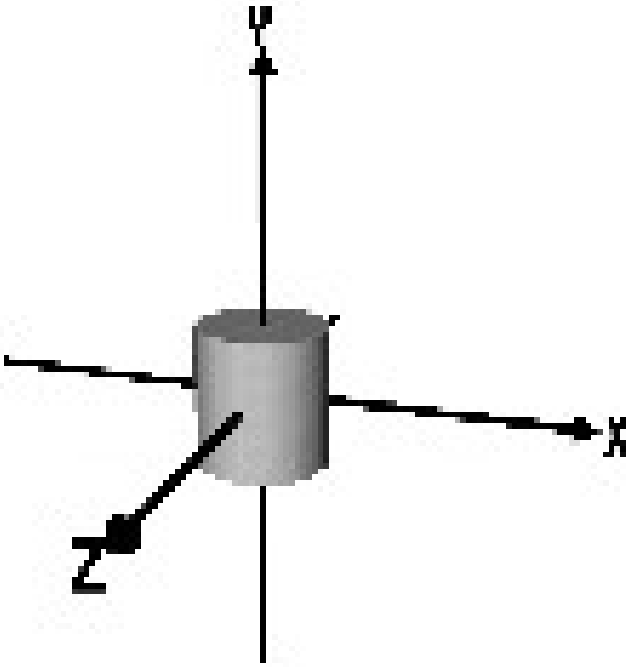
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Cone {
        bottomRadius 3.5
        height 1.5
    }
}
```



- scena VRML

Esempi: default cylinder

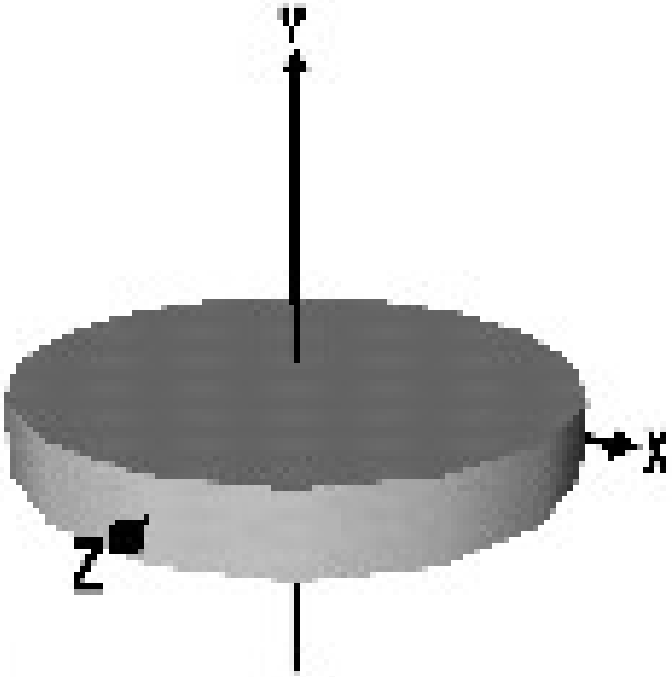
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Cylinder { }
}
```



- scena VRML

Esempi: a cylinder

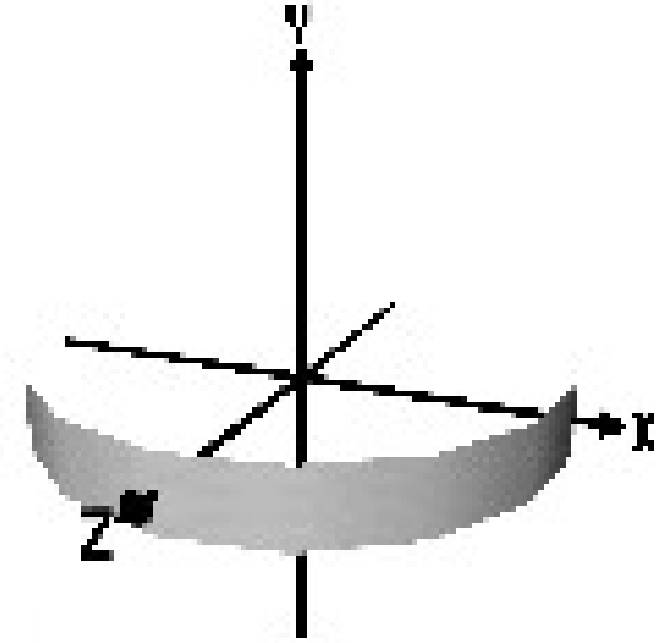
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nade
# and John L. Moreland
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Cylinder {
        radius 4.0
        height 1.0
    }
}
```



- scena VRML

Esempi: un cylinder senza superficie di base superiore e inferiore

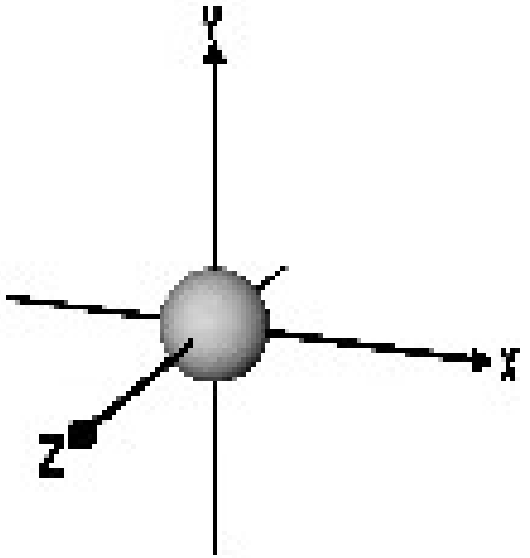
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Cylinder {
        radius 4.0
        height 1.0
        top    FALSE
        bottom FALSE
    }
}
```



- scena VRML

Esempi: default sphere

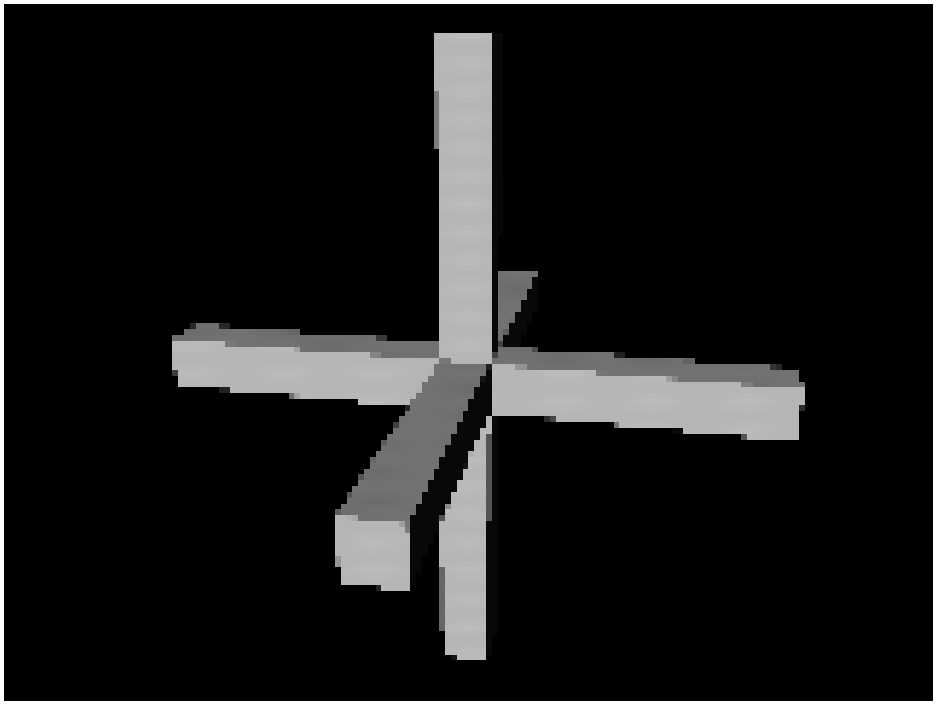
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry Sphere { }
}
```



- scena VRML

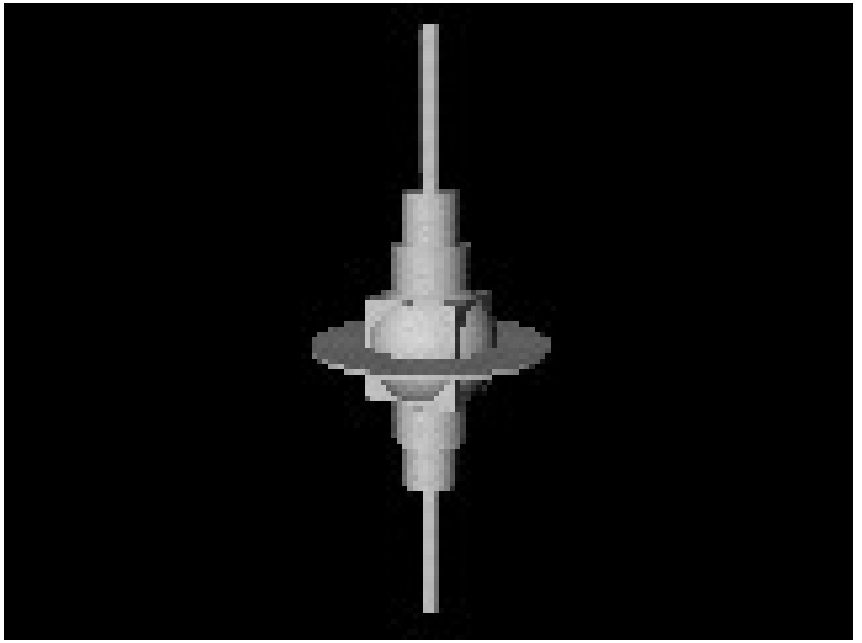
Esempi: Groups

```
#VRML V2.0 utf8
Group {
    children [
        Shape {
            appearance DEF White Appearance {
                material Material { }
            }
            geometry Box {
                size 25.0 2.0 2.0
            }
        }, Shape {
            appearance USE White
            geometry Box {
                size 2.0 25.0 2.0
            }
        },
        Shape {
            appearance USE White
            geometry Box {
                size 2.0 2.0 25.0
            }
        }
    ]
}
```



- scena VRML

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Group {
    children [
        Shape {
            appearance DEF White Appearance {
                material Material { }
            }
            geometry Box {
                size 10.0 10.0 10.0
            }
        },
        Shape {
            appearance USE White
            geometry Sphere {
                radius 7.0
            }
        },
        Shape {
            appearance USE White
            geometry Cylinder {
                radius 12.5
                height 0.5
            }
        },
        Shape {
            appearance USE White
            geometry Cylinder {
                radius 4.0
                height 20.0
            }
        },
        Shape {
            appearance USE White
            geometry Cylinder {
                radius 3.0
                height 30.0
            }
        },
```



```
        Shape {
            appearance USE White
            geometry Cylinder {
                radius 1.0
                height 60.0
            }
        }
    ]
}
```

- Scena VRML

Virtual Reality Modeling Language

VRML

Text Shapes

Text Shapes

- Utilizzando le proprietà relative a testo e font del VRML, si possono aggiungere forme 3-D di testo ai mondi virtuali.
- Per costruire un testo, si usa la il nodo VRML `Text` come valore del campo `geometry` del nodo `Shape`.
- Per ciascun testo si può specificare una stringa di testo e la lunghezza di ciascuna stringa
- Usando il nodo `FontStyle` si può controllare il tipo di font family, lo stile e la dimensione del testo, la spaziatura, il tipo di allineamento, etc

VRML Text

- Una stringa di testo è una sequenza di caratteri che specifica il la geometria di testo da costruire
- Si possono creare linee di testo singole o multiple
- Tutte le linee di testo sono costruite come geometrie 3-D
- In funzione dello stile di font scelto, le linee di testo possono essere costruite da sinistra a destra (come ad es, è scritta la lingua italiana), da destra a sinistra (come è scritto l'arabo), dall'alto verso il basso (come è scritto il cinese) o dal basso verso l'alto.
- Si possono specificare il numero massimo di linee o colonne di testo. Le linee più lunghe sono compresse o riducendo le dimensioni del carattere, o la spaziatura.

Font Styles

- La forma e la posizione dei caratteri costruiti con il nodo Text, dipendono dalle proprietà scelte nel nodo FontStyle. I campi del nodo FontStyle consentono il controllo di:
  - Font family (che definisce la forma dei caratteri usati per il testo)
  - Font style (come bold, italico o normale)
  - Font size del testo
  - Lo spazio tra righe e colonne di testo
  - Il tipo di giustificazione del testo
  - Orientamento del testo: orizzontale o verticale
  - Direzione di flusso del testo: da sx a dx, da dx a sx, da alto a basso, da basso a alto
  - Proprietà di uno specifico linguaggio da usare

Font families

- VRML fornisce tre font familes per le forme di testo:
  - *serif*, simile alla font family **Times Roman**
  - *sans*, simile alla font family **Helvetica**
  - *typewriter*, simile alla font family **Courier**
- Le prime due font family sono proporzionali al carattere (ad es: *i* e *w* occupano spazi differenti), mentre la terza è a spaziatura fissa.
- Tutte le font family possono essere rappresentate a qualsiasi Font size, specificato un unità VRML
- Si può specificare anche il Font style, che può essere plain, **bold**, *italic*.

Character set

- VRML usa il set di caratteri UTF-8 definito dalla standard ISO 10646-1:1993. Questo set di caratteri permette di creare forme con qualsiasi carattere dell'alfabeto italiano, inglese, giapponese, arabo, cirillico, etc.
- Il Character Set indica l'insieme dei caratteri rappresentabili, mentre la font descrive l'aspetto esteriore che un dato carattere assume
- Per ragioni di portabilità dell'applicazione VRML, dei circa 24.000 caratteri definiti da UTF-8, è bene utilizzare solo il sottoinsieme dei caratteri ASCII

Il nodo Text

- Il nodo **Text** crea delle forme di testo e può essere utilizzato come valore per il campo **geometry** nel nodo **Shape**

```
Text {
        string      [ ]       # exposedField MFString
        length      [ ]       # exposedField MFFloat
        maxExtent   0.0       # exposedField SFFloat
        fontStyle   NULL      # exposedField SFNode
    }
```

- Il valore del campo **string** specifica una o più linee di testo da costruire. Ciascuna riga o colonna di testo deve essere racchiusa in apici.
- Per creare più di una riga o colonna di testo, si definisce entro parentesi quadre ciascuna stringa racchiusa tra apici ed eventualmente separata da virgola

Il nodo Text (i)

- Il valore dell'exposedField **`string`** può essere cambiato mediante istradamento di un evento all'exposedField implicito **`set_string`** (**`eventIn`**)
- Quando l'evento è ricevuto, viene definito il campo **`string`** e la nuova stringa viene propagata utilizzando l'exposedField implicito **`string_changed`** (**`eventOut`**)
- Il nodo **`Text`** costruisce le stringhe per default lungo l'asse X, da sinistra a destra. Linee consecutive sono poste in parallelo lungo l'asse Y dalla cima al fondo.
- Questi comportamenti di default possono essere modificati specificando i campi del nodo **`FontStyle`**, che viene specificato come valore del campo **`fontStyle`**.

Il nodo Text: length

- Il valore dell'exposedField `length` specifica la lunghezza desiderata in unità VRML di ciascuna linea di testo
- Per rispettare questo valore, le forme di testo vengono compresse o espanse, modificando le dimensioni o la spaziatura dei caratteri
- Il valore di `0.0` lascia la forma di testo secondo le dimensioni naturali, senza compressione o espansione
- Ciascuna linea di testo può avere la propria lunghezza, includendo diversi valori di lunghezza tra parentesi quadre e separandole eventualmente da virgola. I valori di `length` specificati vengono assegnati univocamente alle corrispondenti linee di testo specificate nel campo `string`

Il nodo Text: length

- Il valore di default di **`length`** è `0.0`.
- Il valore dell'exposedField **`length`** può essere cambiato mediante istradamento di un evento all'exposedField implicito **`set_length`** (**`eventIn`**)
- Quando l'evento è ricevuto, viene definito il campo **`length`** e la nuova stringa viene propagata utilizzando l'exposedField implicito **`length_changed`** (**`eventOut`**)
- Il valore dell'exposedField **`maxExtent`** specifica in unità VRML la lunghezza massima che può assumere una linea o una colonna di testo e deve essere un valore maggiore o uguale a `0.0`.

Il nodo Text: maxExtent

- Le linee più lunghe del valore di **`maxExtent`** vengono compresse entro tale lunghezza, quelle più corte non subiscono alcuna azione.
- Il valore dell'exposedField **`maxExtent`** può essere cambiato mediante istradamento di un evento all'exposedField implicito **`set_maxExtent`** (eventIn)
- Quando l'evento è ricevuto, viene definito il campo **`maxExtent`** e la nuova stringa viene propagata utilizzando l'exposedField implicito **`maxExtent_changed`** (eventOut)

Il nodo Text: fontStyle

- Il nodo **FontStyle** controlla il modo con cui viene realizzata la forma di testo rispetto al default, che vede il primo carattere scritto sull'asse x, in prossimità dell'origine, con la coordinata y=0.0.
- Il valore dell'exposedField **fontStyle** può essere cambiato mediante istradamento di un evento all'exposedField implicito **set_fontStyle** (**eventIn**)
- Quando l'evento è ricevuto, viene definito il campo **fontStyle** e la nuova stringa viene propagata utilizzando l'exposedField implicito **fontStyle_changed** (**eventOut**)
- Un valore tipico del campo **fontStyle** è il nodo **FontStyle**

Il nodo FontStyle

- Il nodo **FontStyle** controlla le caratteristiche che definiscono l'aspetto delle forme di testo create dal nodo **Text**. Il nodo **FontStyle** può essere utilizzato come valore per il campo **fontStyle** nel nodo **Text**

```
FontSTyle {
        family     "SERIF"   # Field SFString
        style      "PLAIN"   # Field SFString
        size       1.0       # Field SFFloat
        spacing    1.0       # Field SFFloat
        justify    "BEGIN"   # Field SFString
        horizontal TRUE      # Field SFBool
        leftToRight TRUE     # Field SFBool
        topToBottom TRUE     # Field SFBool
        language     ""      # Field SFString
    }
```

Il nodo FontStyle(i)

- Il campo family indica il tipo di font utilizzata, tra le tre possibili implementate da VRML:
  - "**SERIF**": font a spaziatura variabile, tipo Times Roman (default)
  - "**SANS**": font a spaziatura variabile, tipo Helvetica
  - "**TYPEWRITER**": font a spaziatura fissa, tipo Courier
- La visualizzazione delle forme di testo avviene però ad opera del browser VRML, pertanto può accadere che la visualizzazione non sia sempre omogenea

Il nodo FontStyle(ii)

- Il campo `style` può assumere i seguenti valori
  - "PLAIN": testo normale (default)
  - "**BOLD**": testo in **grassetto**
  - "*ITALIC*": testo *italico* o *obliquo*
  - "***BOLDITALIC***": testo ***italico*** e in ***grassetto***
- Il valore di defalt del campo `style` è "PLAIN".
- Il valore del campo `size` esprime l'altezza dei caratteri in unità VRML. Il valore di default è `1.0` unità

Il nodo FontStyle(iii)

- Il campo **`spacing`** specifica la spaziatura verticale in unità VRML delle linee nel caso di testo orizzontale, mentre rappresenta la distanza orizzontale nel caso di colonne verticali di testo
- In entrambi i casi linee o colonne consecutive sono costruite alla distanza **`(N-1)x(size x spacing)`** dove **`N`** rappresenta l'n-esima riga.
- Il valore di default di **`spacing`** è **`1.0`**
- Il valore del campo **`horizontal`** indica quando il testo deve essere costruito orizzontalmente (TRUE) o verticalmente (FALSE)

Il nodo FontStyle(iv)

- I valori di `horizontal`, `leftToRight` e `topToBottom` sono utilizzati in combinazione per controllare la posizione orizzontale e verticale del testo
- Il campo `justify` specifica il modo con cui la forma di testo viene posta rispetto agli assi X e Y.
- Esiste una giustificazione maggiore che controlla l'allineamento lungo l'asse principale, mentre ne esiste una minore relativa all'asse secondario (in genere quello verticale). Valori validi per la giustificazione del testo, sono:

  "FIRST", "BEGIN", "MIDDLE", "END"

Il nodo FontStyle(iv)

- Il valore del campo `language` specifica il contesto del linguaggio usato nel campo string del nodo `Text`. In genere questo campo non è specificato ed il suo valore lasciato a quello di default (“”)
- Quando è usato, il valore del campo aiuta a capire la scelta del linguaggio utilizzata nel valore del campo `string`.
- Valori validi di `language` sono basati su specifiche definite in molti standard internazionali, tra cui POSIX e RFC 1766.
- Ogni valore del campo language contiene un *language-code* (es `de`, `jp`, `it`, `en`) eventualmente seguito dal carattere _ e da un *territory-code* (es: `de_CH`, `en_US`, `en_GB`)

Esempi: una forma di testo

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
  appearance Appearance {
       material Material { }
      }
  geometry Text {
       string "Qwerty"
           }
}
```

- scena VRML

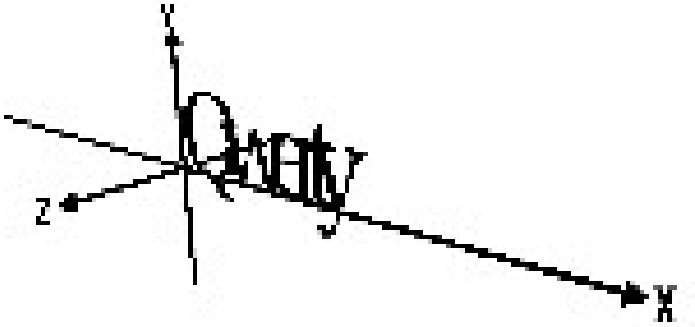
Esempi: una lista di strighe

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
  appearance Appearance {
      material   Material { }
     }
  geometry Text {
     string [ "Qwerty", "0123" ]
     }

}
```

- scena VRML

Esempi: testo compresso

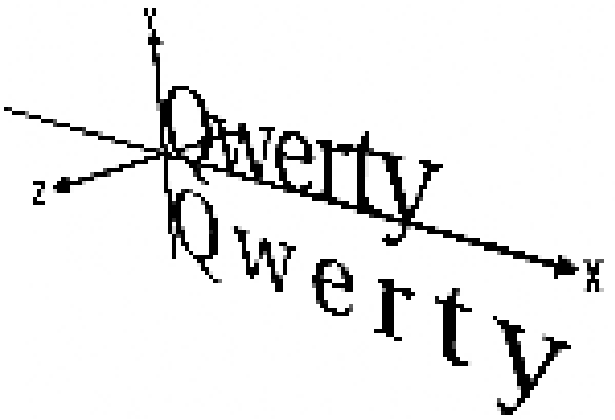
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
  appearance Appearance {
      material Material { }
     }
  geometry Text {
     string "Qwerty"
     length 2.0
     }

}
```



- scena VRML

Esempi: testo espanso

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
  appearance Appearance {
      material Material { }
     }
  geometry Text {
     string [ "Qwerty", "Qwerty" ]
     length [ 3.0, 4.0 ]
     }

}
```



- scena VRML

Esempi: maxExtent

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
  appearance Appearance {
       material Material { }
    }
  geometry Text {
    string [ "Qwerty Uiop", "Asdf" ]
    maxExtent 4.0
    }

}
```

- scena VRML

Esempi: Serif string

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
  appearance Appearance {
       material Material { }
    }
  geometry Text {
    string "Qwerty"
    fontStyle FontStyle {
       family "SERIF"
       style ""
       }
    }
}
```

- scena VRML

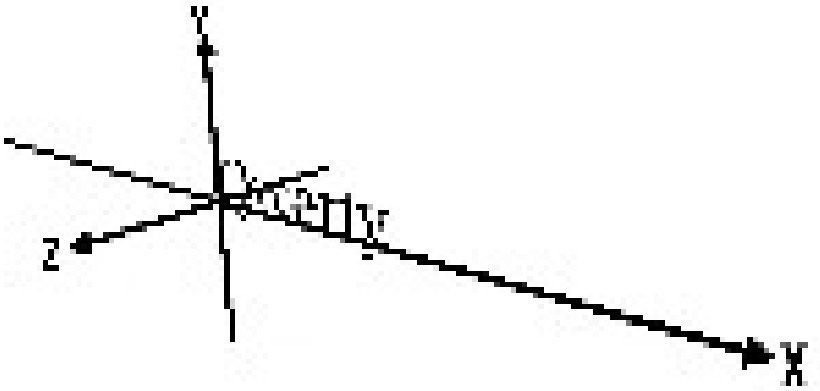
Esempi: Serif Bold string

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
  appearance Appearance {
        material Material { }
     }
  geometry Text {
     string "Qwerty"
     fontStyle FontStyle {
        family "SERIF"
        style ”BOLD"
        }
     }

}
```

- scena VRML

Esempi: dimensioni font

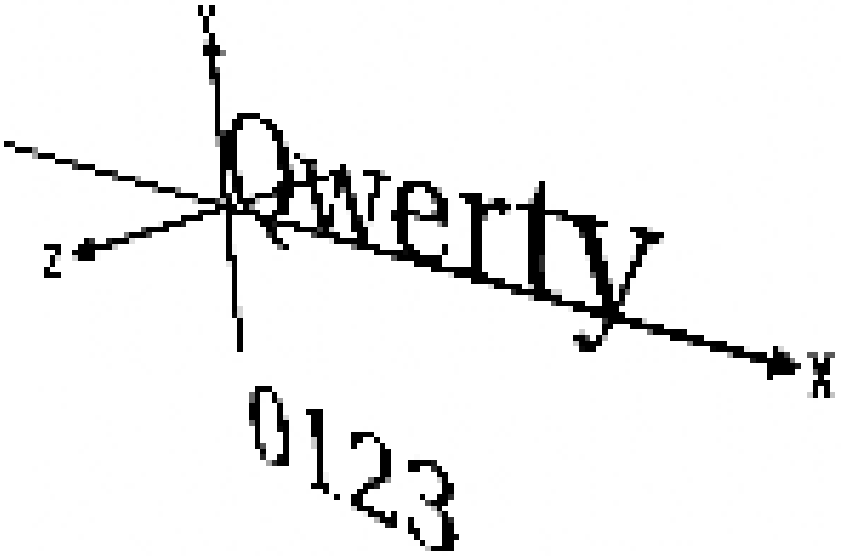
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
   appearance Appearance {
       material Material { }
      }
   geometry Text {
      string "Qwerty"
      fontStyle FontStyle {
         size 0.5
         }
      }
}
```



- scena VRML

Esempi: spaziatura font

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
   appearance Appearance {
      material Material { }
      }
   geometry Text {
      string [ "Qwerty", "0123" ]
      fontStyle FontStyle {
         spacing 2.0
         }
      }
}
```



- scena VRML

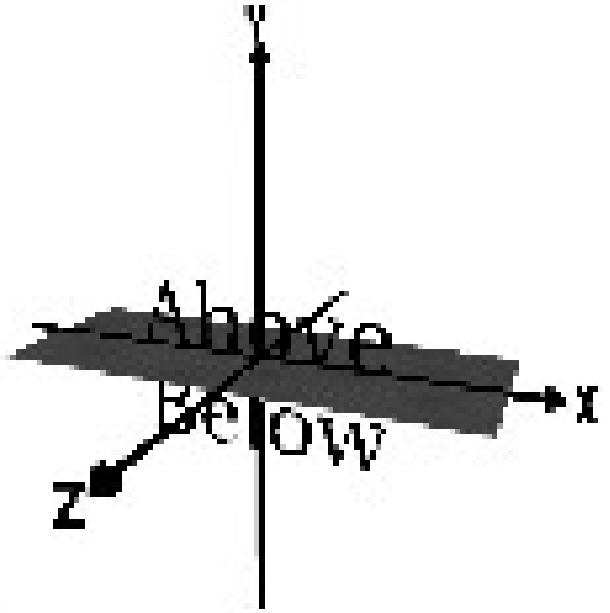
Esempi: Group

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Group {
  children [
      Shape {
      appearance DEF White Appearance {
       material Material { }
          }
      geometry Text {
              string "First"
              fontStyle FontStyle {
              family "SERIF"
              style "ITALIC"
              justify "END"
              size 1.0
                }
              }
          },
      Shape {
         appearance USE White
         geometry Text {
           string "Second"
           fontStyle FontStyle {
                family "SANS"
                style "BOLD"
                justify "BEGIN"
                size 1.0
         }
        }
      }
   ]
}
```

" scena VRML

Esempi: Group (i)

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Group {
  children [
     Shape {
      appearance DEF White Appearance
       material Material { }
           }
      geometry Text {
           string "Above", "Below"
           fontStyle FontStyle {
               justify "MIDDLE"
             }
           }
       },
     Shape {
       appearance USE White
       geometry Box {
          size 5.0 0.01 2.0
       }
     }
   ]
}
```



" scena VRML

Esempi: Group (i)

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Group {
    children [
        Shape {
            appearance DEF White Appearance
   {
                material Material { }
            }
            geometry Text {
                string [ "Over", "Strike" ]
                fontStyle DEF myFontStyle
                   FontStyle {
                       size    6.0
                       family  "TYPEWRITER"
                       justify "MIDDLE"
                   }
            }
        },
        Shape {
            appearance USE White
            geometry Text {
                string [ "----", "------" ]
                fontStyle USE myFontStyle
            }
        }
    ]
}
```

" scena VRML

Virtual Reality Modeling Language

VRML

Posizionamento di Forme

Posizionamento di Forme

- Si possono posizionare delle forme ovunque nel mondo virtuale. Si possono mettere una sopra l'altra, una dentro l'altra o lasciare la forma sospesa
- Ciò si ottiene mediante l'utilizzo del nodo VRML `Transform` e definendo il campo `translation` con la coordinata 3D in coincidenza della quale si vuole spostare la forma o il gruppo di forme.
- Si è visto come il nodo `Shape` sia sempre centrato nell'origine del sistema di coordinate.
- Analogamente il nodo `Text` crea anch'esso delle stringhe di testo centrate nell'origine.

Sistemi di coordinate

- Utilizzando determinati nodi (ad es. `Transform`) viene definito un nuovo sistema di coordinate che fa riferimento a quello corrente.
- Ogni nuovo sistema di coordinate è traslato rispetto all'origine del sistema di coordinate di riferimento
- Quando un nuovo sistema di coordinate è relativo ad un altro, diciamo che il nuovo sistema di coordinate è un sistema di coordinate figlio, nidificato nel sistema di coordinate padre.
- Il vantaggio di questo approccio è che la costruzione del mondo virtuale può avvenire in maniera modulare

Sistemi di coordinate

- La struttura di una lampada è la stessa indipendentemente da dove la lampada è posta sul tavolo,come la struttura del tavolo è la stessa indipendentemente da dove è situato nella stanza.
- Per spostare il tavolo occorre traslare il sistema di coordinate del tavolo. Essendo il sistema di coordinate della lampada, figlio di quello del tavolo, anche la lampada si sposterà di conseguenza.
- Ogni `Shape` viene costruita secondo il suo sistema di coordinate. Queste coordinate della `Shape` possono essere poi raggruppate insieme in un sistema di coordinate padre, il quale può essere raggruppato insieme ad altre forme par dar luogo ad un sistema di coordinate padre ancora più grande, etc.

Sistemi di coordinate (i)

- Questa relazione padre-figlio crea una famiglia di sistemi di coordinate con struttura ad albero.
- La punta dell'albero è rappresentata dal root coordinate system del file VRML. Ogni file VRML ha il proprio root coordinate system.
- Il nodo `Transform` crea un nuovo sistema di coordinate relativo al root coordinate system o a qualsiasi altro sistema di coordinate
- Il root coordinate system è chiamato anche world coordinate system
- L'intero albero di sistemi di coordinate e le forme create nei sistemi di coordinate vengono chiamati scene graph

Il nodo Transform

- Il nodo **Transform** crea un nuovo sistema di coordinate relativo al suo sistema di coordinate padre. Le Forme create come **children** del nodo **Transform** , sono create centrate nell'origine del nuovo sistema di coordinate

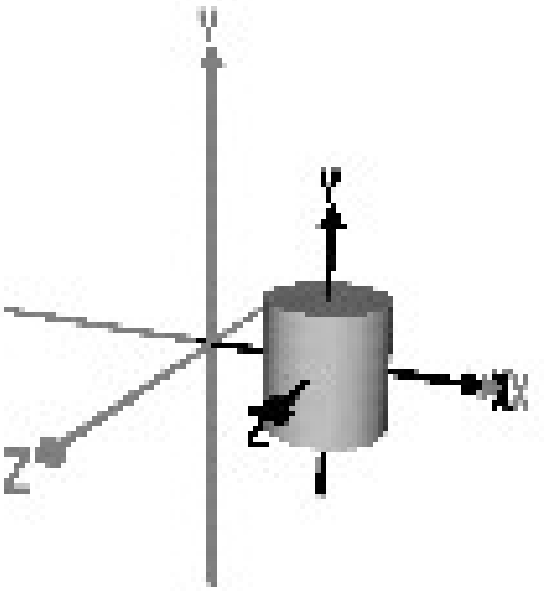
```
Transform {
        children    [ ]                          # exposedField MFNode
        translation 0.0 0.0 0.0                  # exposedField SFVec3f
        rotation  0.0 0.0 1.0 0.0                # exposedField SFRotation
        scale   1.0  1.0 1.0                     # exposedField SFVec3f
        scaleOrientation 0.0 0.0 1.0 0.0         # exposedField SFRotation
        bboxCenter 0.0 0.0 0.0                   # Field SFVec3F
        bboxsize -1.0 -1.0 -1.0                  # Field SFVec3F
        center 0.00 0.0 0.0                      # exposedField SFVec3f
        addChildren                              # eventIn MFNode
        removeChildren                           # eventIn MFNode
}
```

Il nodo Transform: `translation`

- Il valore dell'exposedField **`children`** specifica una lista di nodi figli da includere nel gruppo. Valori tipici di **`children`** sono i nodi **`Shape`**, **`Group`** e **`Transform`** . Il valore di default è una lista vuota
- Il valore dell'exposedField **`translation`** specifica le distanze nelle direzioni X, Y e Z tra il sistema di coordinate padre ed il nuovo sistema di coordinate. I valori possono essere positivi o negativi, il default è 0.0 in tutte le direzioni, che coincide con una traslazione nulla.
- Il valore dell'exposedField **`translation`** può essere cambiato mediante istradamento di un evento all'exposedField implicito **`set_translation`** (**`eventIn`**)
- Quando l'evento è ricevuto, viene definito il campo **`translation`** e la nuova stringa viene propagata utilizzando l'exposedField implicito **`translation _changed`** (**`eventOut`**)

Esempi: traslazione in X

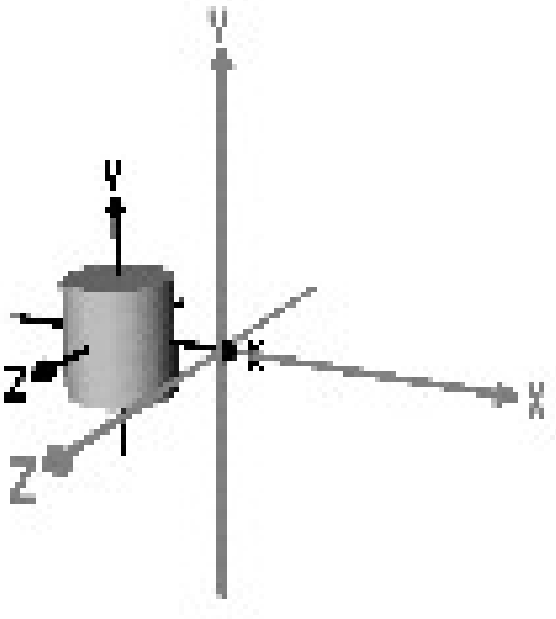
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
   appearance Appearance {
         material Material { }
     }
   geometry Sphere { radius 0.2 }
}
Transform {
  translation 2.0 0.0 0.0
   children [
     Shape { appearance
       Appearance {
         material Material { }
        }
       geometry Cylinder { }
     }
   ]
}
```



- scena VRML

Esempi: traslazione in -X

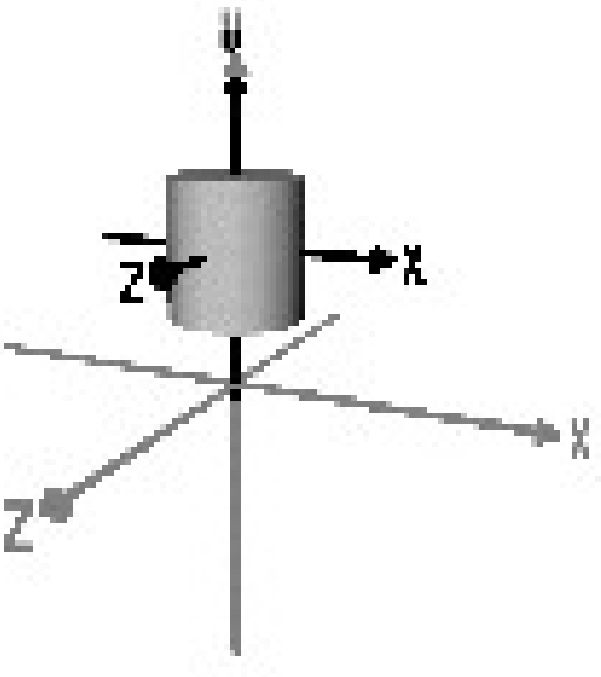
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
    appearance Appearance {
            material Material { }
      }
    geometry Sphere { radius 0.2 }
}
Transform {
   translation -2.0 0.0 0.0
    children [
      Shape { appearance
        Appearance {
          material Material { }
         }
        geometry Cylinder { }
      }
    ]
}
```



- scena VRML

Esempi: traslazione in Y

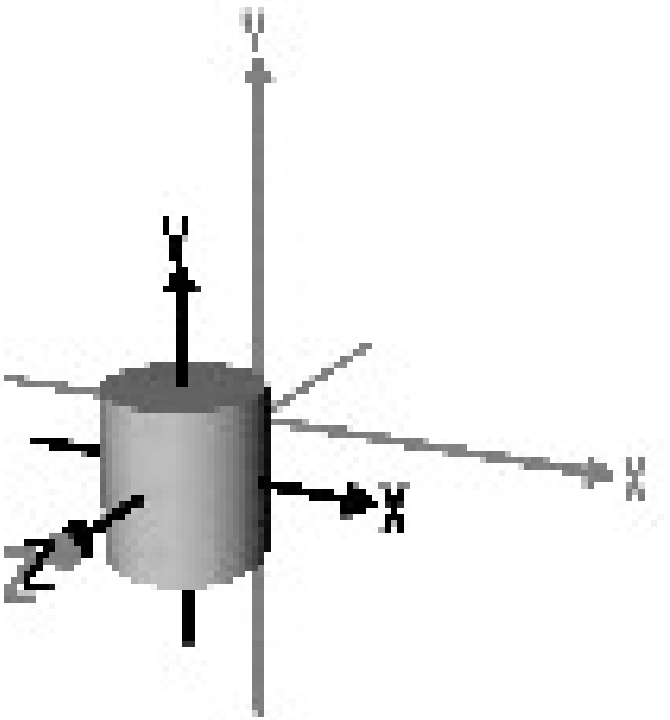
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
    appearance Appearance {
            material Material { }
      }
    geometry Sphere { radius 0.2
}
Transform {
   translation 0.0 2.0 0.0
    children [
      Shape { appearance
        Appearance {
          material Material { }
         }
        geometry Cylinder { }
      }
    ]
}
```



- scena VRML

Esempi: traslazione in Z

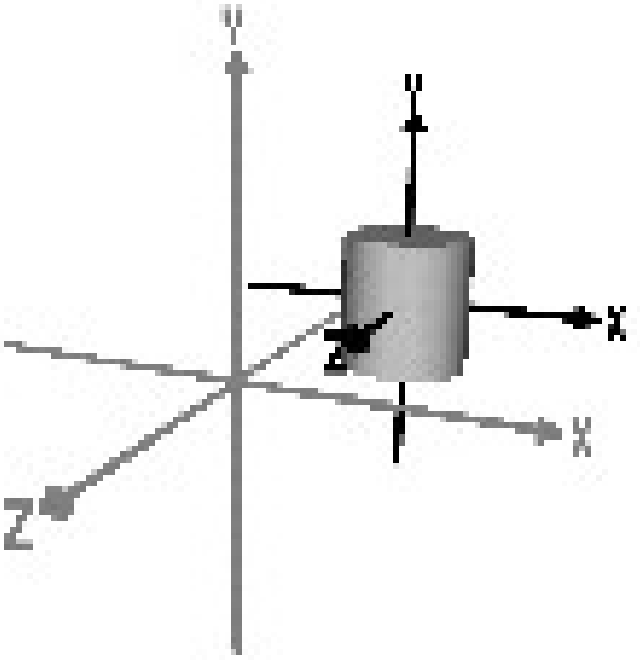
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau
# and John L. Moreland
Shape {
    appearance Appearance {
            material Material { }
      }
    geometry Sphere { radius 0.2 }
}
Transform {
   translation 0.0 0.0 2.0
    children [
      Shape { appearance
        Appearance {
          material Material { }
         }
        geometry Cylinder { }
      }
    ]
}
```



- scena VRML

Esempi: traslazione in X,Y,Z

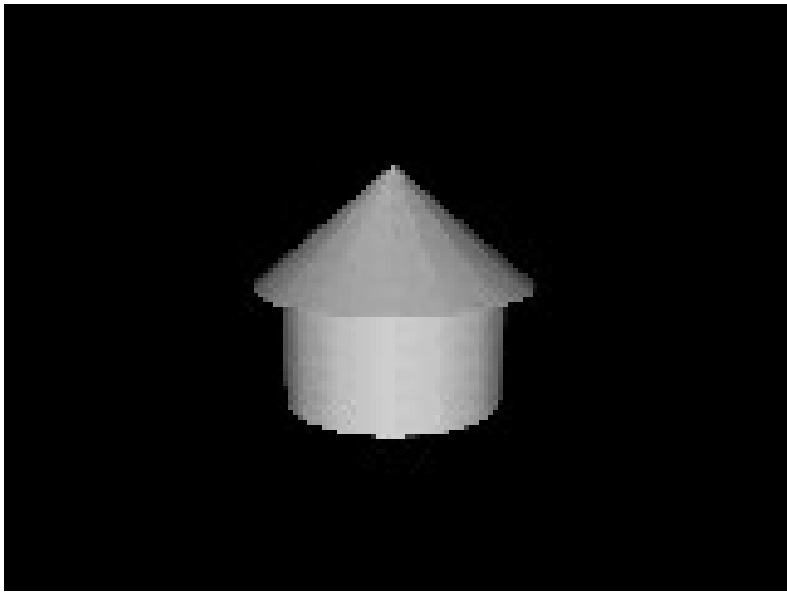
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland
Shape {
    appearance Appearance {
          material Material { }
      }
    geometry Sphere { radius 0.2 }
}
Transform {
   translation 2.0 1.0 -2.0
    children [
      Shape { appearance
        Appearance {
          material Material { }
         }
        geometry Cylinder { }
      }
    ]
}
```



- scena VRML

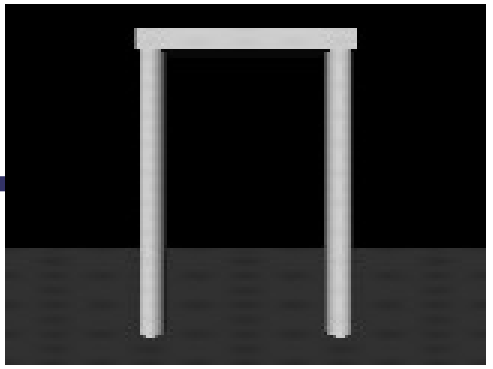
Esempi: capanna

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau,
#  and John L. Moreland
# A gray hut
Group {
    children [
    # Draw the hut walls
    Shape {
         appearance DEF White Appearance {
              material Material { }
         }
         geometry Cylinder {
              height 2.0
              radius 2.0
         }
    },
    # Draw the hut roof
    Transform {
         translation 0.0 2.0 0.0
         children [
              Shape {
                   appearance USE White
                   geometry Cone {
                        height 2.0
                        bottomRadius 2.5
                   }
              }
         ]
    }
    ]
}
```



"scena VRML

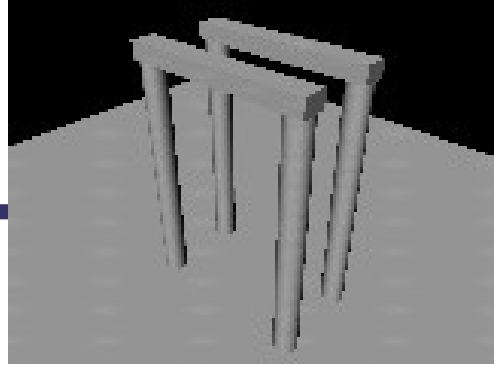
Esempi: arco



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright 1997 By
# Andrea L. Ames, David R. Nadeau,
#   and John L. Moreland
Group {
    children [
    # Ground
    Shape {
         appearance DEF White Appearance {
              material Material { }
         }
         geometry Box {
              size 25.0 0.1 25.0
         }
    },
    # Left Column
    Transform {
         translation -2.0 3.0 0.0
         children Shape {
              appearance USE White
              geometry Cylinder {
                   radius 0.3
                   height 6.0
              }
         }
    },
    # Right Column
    Transform {
         translation 2.0 3.0 0.0
         children Shape {
              appearance USE White
              geometry Cylinder {
                   radius 0.3
                   height 6.0
                                  }
              }
         },
    # Archway span
         Transform {
              translation 0.0 6.05 0.0
              children Shape {
                   appearance USE White
                   geometry Box {
              size 4.6 0.4 0.6
                                  }
              }
         }
    ]
}
```

- scena VRML

Esempi: doppio arco



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright 1997 By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Ground
    Shape {
         appearance DEF White Appearance {
              material Material { }
         }
         geometry Box {
              size 25.0 0.1 25.0
         }
    },
    # First archway
    # Left Column
    DEF LeftColumn Transform {
         translation -2.0 3.0 0.0
         children DEF Column Shape {
              appearance USE White
              geometry Cylinder {
                   radius 0.3
                   height 6.0
              }
         }
    },
    # Right Column
    DEF RightColumn Transform {
         translation 2.0 3.0 0.0
         children USE Column
    },
    # Archway span
         DEF ArchwaySpan Transform {
         translation 0.0 6.05 0.0
         children Shape {
                   appearance USE White
                   geometry Box {
                             size 4.6 0.4 0.6
                             }
                   }
         },
    # Second archway
         Transform {
                   translation 0.0 0.0 -2.0
                   children [
                             USE LeftColumn,
                             USE RightColumn,
                             USE ArchwaySpan
                   ]
         }
    ]
}
```

- scena VRML

Virtual Reality Modeling Language

VRML

Rotazione di forme

Rotating Shapes

- I nodi VRML possono essere ruotati in un sistema di coordinate, rispetto ad un asse di rotazione, mediante il nodo VRML `Transform` ed il campo `rotation`.
- L'asse di rotazione è una linea immaginaria, intorno alla quale il nodo viene ruotato di un certo angolo di rotazione, espresso in radianti.
- L'asse di rotazione può assumere qualsiasi direzione nello spazio VRML. Per disegnare l'asse occorre immaginare la linea che congiunge l'origine del sistema di coordinate con uno specificato punto dello spazio.

Rotating Shapes (i)

- Per calcolare la corrispondenza tra angoli in gradi e angoli in radianti, occorre usare la formula

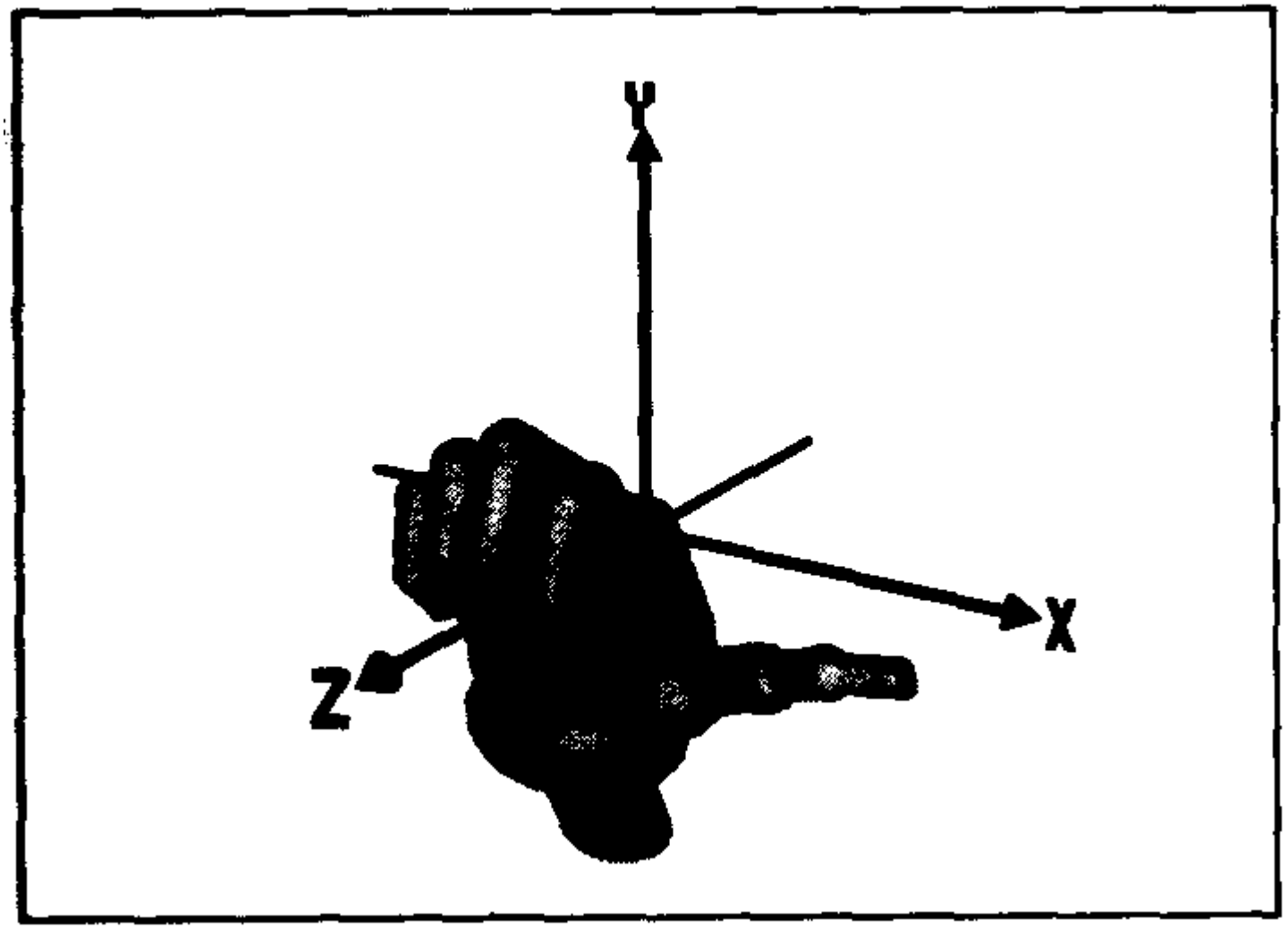
$$rad = gradi / 180 * 3{,}142$$

- L'angolo di rotazione può assumere sia valori positivi che negativi, tenendo conto che in questo caso vale la regola della mano destra per la rotazione

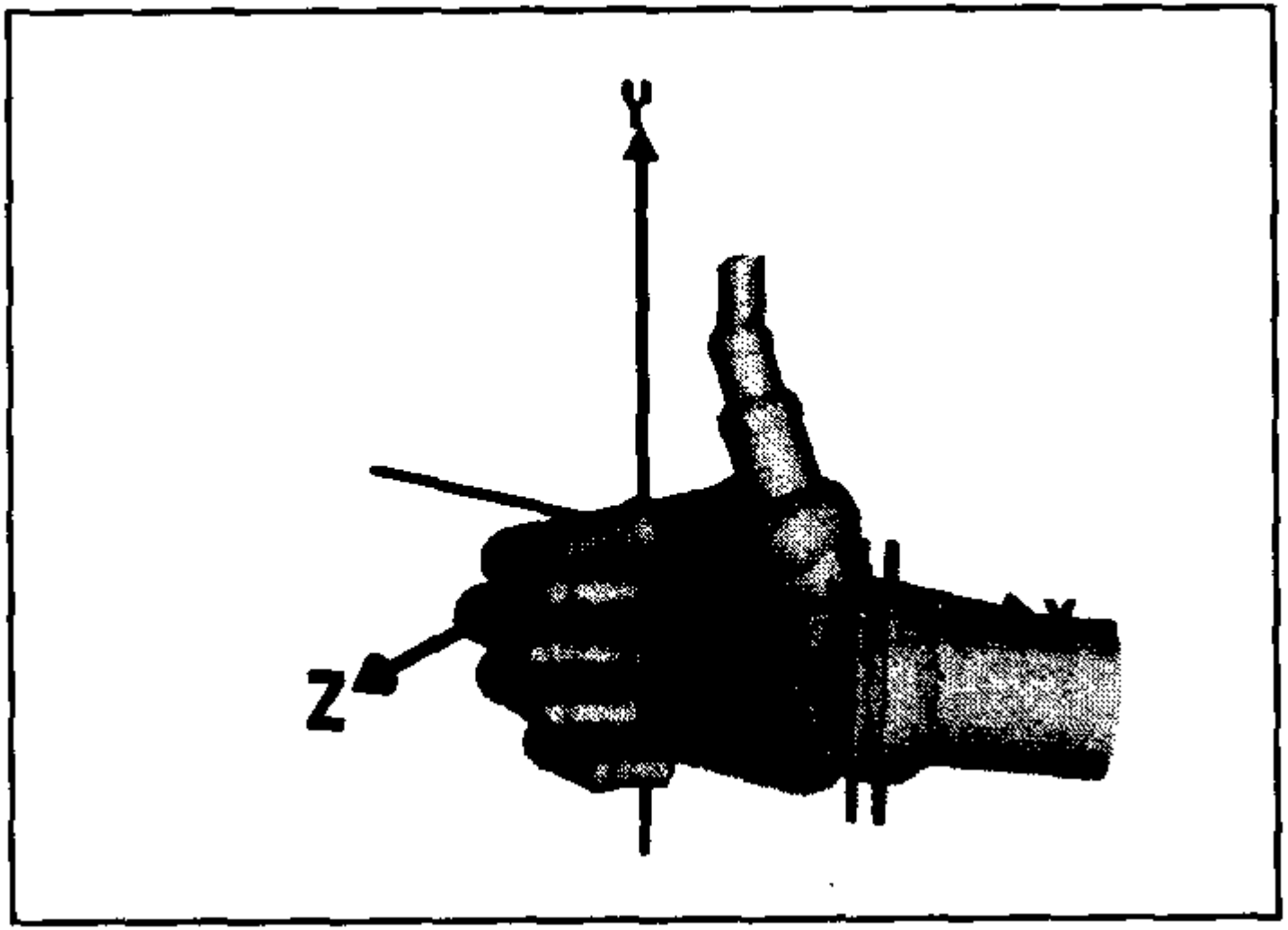
Corrispondenze per gli angoli

Gradi	*Radianti*
10	0.174
20	0.349
30	0.524
45	0.785
60	1.047
90	1.571
120	2.095
150	2.168
180	3.142
210	3.666
240	4.189
270	4.713
300	5.237
330	7.760
360	6.284

Regola della mano destra per la rotazione: asse X



Regola della mano destra per la rotazione: asse Y



Il nodo Transform

- Il nodo **Transform** crea un nuovo sistema di coordinate relativo al suo sistema di coordinate padre. Le Forme create come **children** del nodo **Transform** , sono create centrate nell'origine del nuovo sistema di coordinate

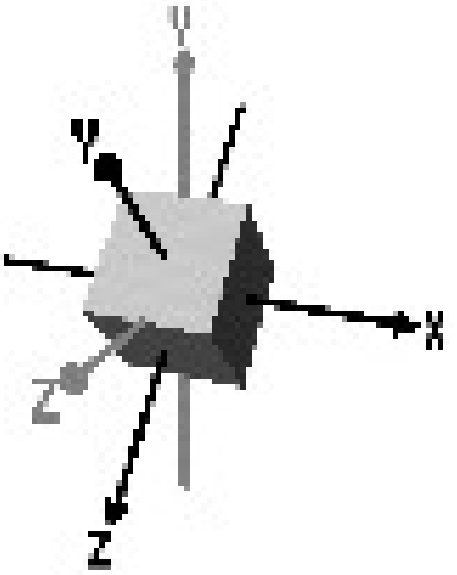
```
Transform {
        children    [ ]                          # exposedField MFNode
        translation 0.0 0.0 0.0                  # exposedField SFVec3f
        rotation  0.0 0.0 1.0 0.0                # exposedField SFRotation
        scale   1.0  1.0 1.0                     # exposedField SFVec3f
        scaleOrientation 0.0 0.0 1.0 0.0         # exposedField SFRotation
        bboxCenter 0.0 0.0 0.0                   # Field SFVec3F
        bboxsize -1.0 -1.0 -1.0                  # Field SFVec3F
        center 0.00 0.0 0.0                      # exposedField SFVec3f
        addChildren                              # eventIn MFNode
        removeChildren                           # eventIn MFNode
}
```

Rotazione + traslazione

- L'operazione di rotazione e traslazione può avvenire simultaneamente, codificando nello stesso nodo **`Transform`** sia i campi **`translation`** che **`rotation`**
- Per default il centro di rotazione coincide con l'origine del sistema di coordinate, è comunque possibile specificare un centro di rotazione diverso, mediante codifica del campo **`center`**
- Il valore dell'exposed-field **`rotation`** può essere cambiato mediante istradamento di un evento all'exposed-field implicito **`set_rotation`** (**`eventIn`**)
- Quando l'evento è ricevuto, viene definito il campo **`rotation`** e la nuova stringa viene propagata utilizzando l'exposed-field implicito **`rotation_changed`** (**`eventOut`**)

Esempi: rotazione di 45° lungo l'asse X

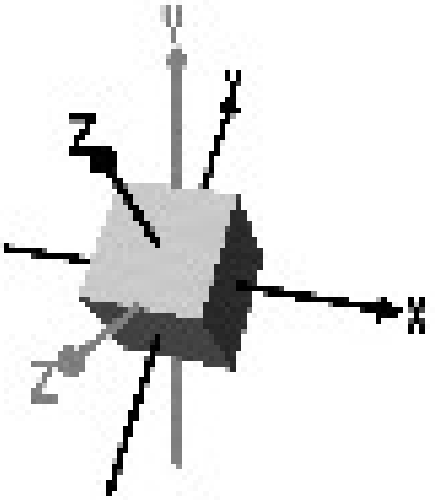
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland

Transform {
 rotation 1.0 0.0 0.0 0.785
   children [
     Shape {
       appearance Appearance {
            material Material { }
               }
       geometry Box { }
     }
   ]
}
```



- see the VRML scene

Esempi: rotazione di -45° lungo l'asse X

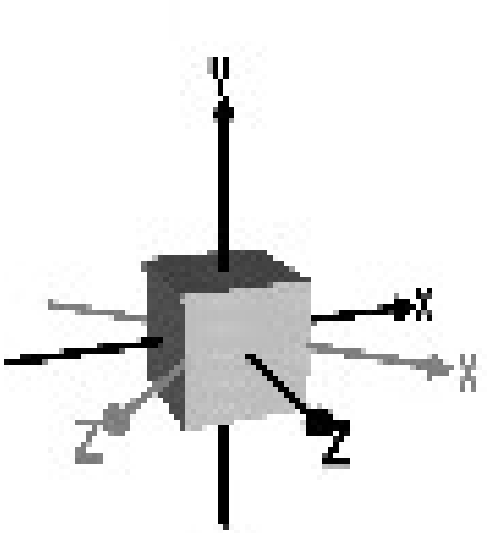
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland

Transform {
 rotation 1.0 0.0 0.0 -0.785
   children [
     Shape {
       appearance Appearance {
            material Material { }
               }
       geometry Box { }
     }
   ]
}
```



- see the VRML scene

Esempi: rotazione di 45° lungo l'asse Y

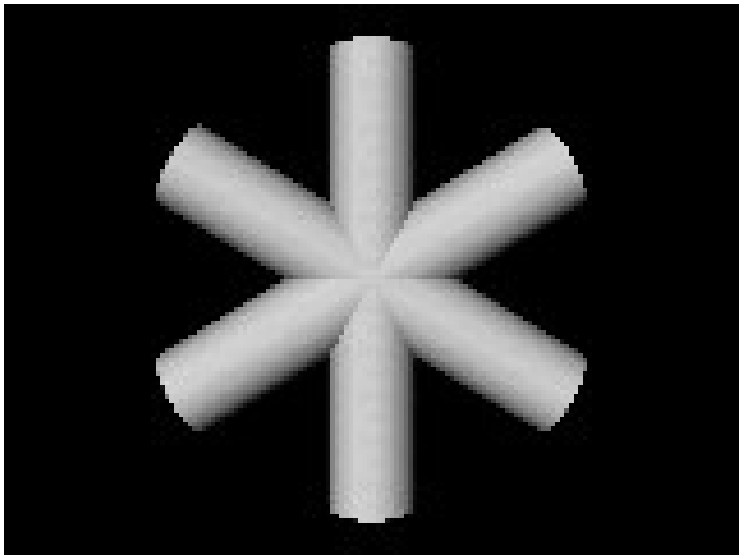
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland

Transform {
 rotation 0.0 1.0 0.0 0.785
   children [
     Shape {
       appearance Appearance {
            material Material { }
               }
       geometry Box { }
     }
   ]
}
```



- see the VRML scene

Esempi: asterisco

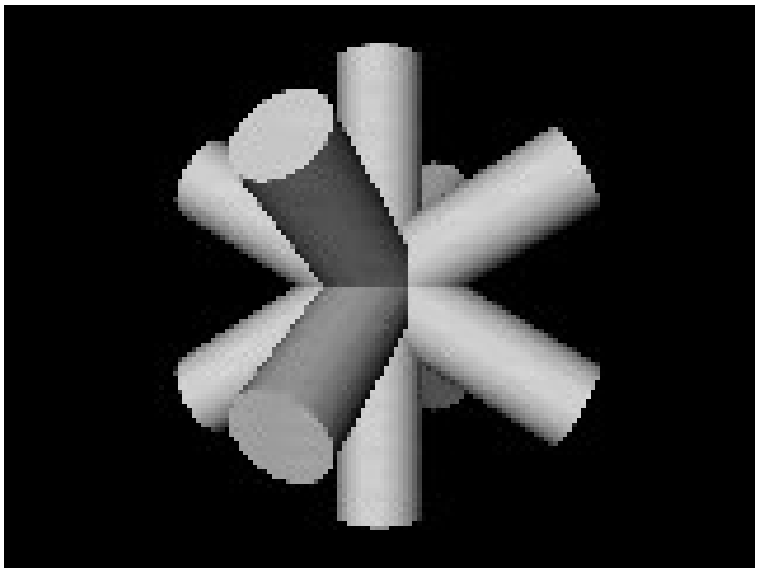
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau, and
   John L. Moreland
Group {
     children [
     # Arm 1
          DEF Arm1 Shape {
               appearance Appearance {
                    material Material { }
               }
               geometry Cylinder {
                    height 1.0
                    radius 0.1
               }
          },
     # Arm 2
          Transform {
               rotation 1.0 0.0 0.0  1.047
               children USE Arm1
          },
     # Arm 3
          Transform {
               rotation 1.0 0.0 0.0  2.094
               children USE Arm1
          }
     ]
}
```



- see the VRML scene

Esempi: asterisco 3D

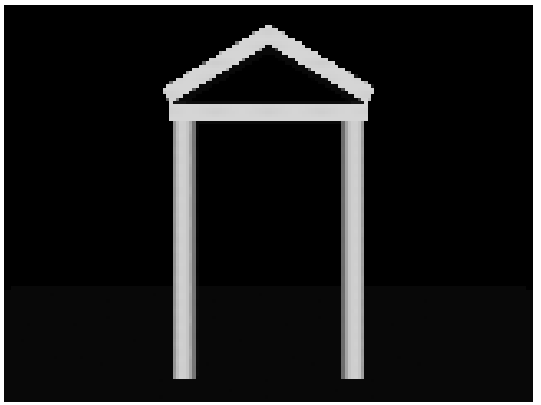
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau, and John L.
   Moreland
Group {
    children [
    # Arm 1
        DEF Arm1 Shape {
            appearance Appearance {
                material Material { }
            }
            geometry Cylinder {
                height 1.0
                radius 0.1
            }
        },
    # Arm 2
        DEF Arm2 Transform {
            rotation 1.0 0.0 0.0  1.047
            children USE Arm1
        },
    # Arm 3
        DEF Arm3 Transform {
            rotation 1.0 0.0 0.0  2.094
            children USE Arm1
        },
    # Arms 4 and 5
        Transform {
            rotation 0.0 1.0 0.0  1.785
            children [
                USE Arm2,
                USE Arm3
            ]
        }
    ]
}
```



see the VRML scene

Esempi: tempio

```
#VRML V2.0 utf8
Group {
   children [
# Ground          Shape {
        appearance DEF White Appearance {
                material Material { }
                                        }
            geometry Box {
                size 25.0 0.1 25.0              }
                    },
# First archway
# Left Column          DEF LeftColumn Transform {
                translation -2.0 3.0 0.0
        children DEF Column Shape {
            appearance USE White
            geometry Cylinder {
                    radius 0.3
                    height 6.0}
                                }
                        },
# Right Column
    DEF RightColumn Transform {
        translation 2.0 3.0 0.0
        children USE Column          },
# Archway span
    DEF ArchwaySpan Transform {
        translation 0.0 6.05 0.0
        children Shape {
            appearance USE White
            geometry Box{
            size 4.6 0.4 0.6
                        }
            }
},
# Left Roof
        DEF LeftRoof Transform {
        translation -1.15 7.12 0.0
         rotation 0.0 0.0 1.0  0.524
```

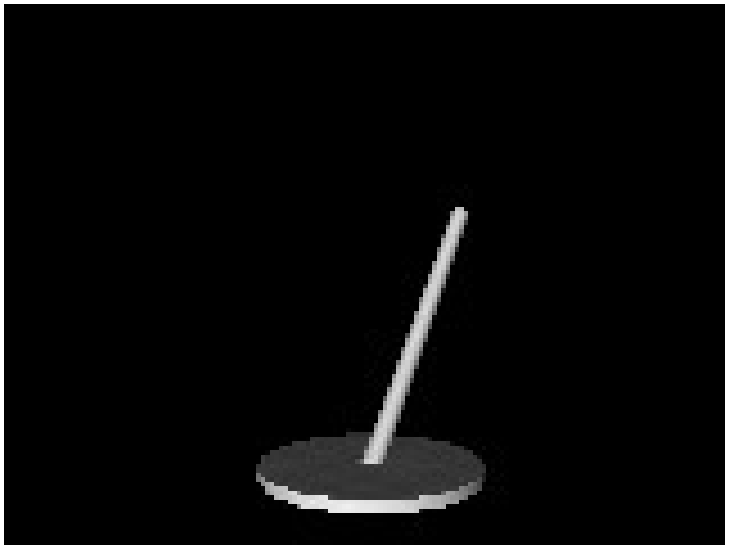


```
        children DEF Roof Shape {
                    appearance USE White
            geometry Box {
                size 2.86 0.4 0.6
                }
            }
    },
# Right Roof
        DEF RightRoof Transform {
            translation 1.15 7.12 0.0
            rotation 0.0 0.0 1.0  -0.524
            children USE Roof          }
        ]
}
```

see the VRML scene

Esempi: base lampada

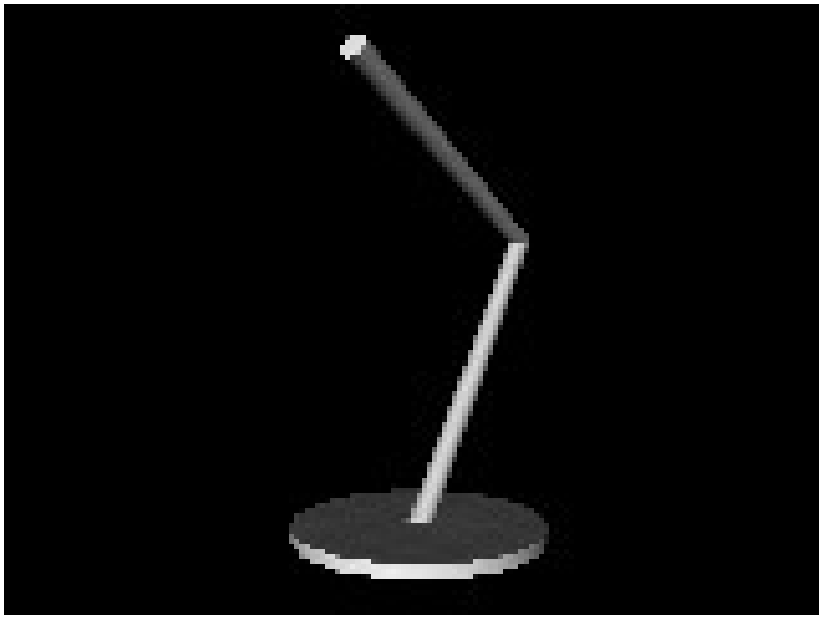
```
#VRML V2.0 utf8
Group{
   children [
# Lamp base
        Shape {
             appearance DEF White Appearance {
                          material Material { }

                 }
             geometry Cylinder {
               radius 0.1
              height 0.01
               }
        },
# Base joint
    Transform {
         translation 0.0 0.15 0.0
             rotation    1.0 0.0 0.0  -0.7
             center      0.0 -0.15 0.0
      children [
# Lower arm
             Shape {
             appearance USE White
                    geometry Cylinder {
                         radius 0.01
                         height 0.3
                       }
                   }
               ]
              }
      ]
}
```



see the VRML scene

Esempi: lampada

```
#VRML V2.0 utf8
Group{
   children [
# Lamp base
        Shape {
            appearance DEF White Appearance {
                        material Material { }
   }
            geometry Cylinder {
              radius 0.1
              height 0.01      }
              },
# Base joint
        Transform {
            translation 0.0 0.15 0.0
            rotation    1.0 0.0 0.0  -0.7
            center      0.0 -0.15 0.0
            children [
# Lower arm
            DEF LampArm Shape {
                  appearance USE White
                  geometry Cylinder {
                     radius 0.01
                     height 0.3      }
                              },
# Lower arm - second arm joint
            Transform {
                translation 0.0 0.3 0.0
                rotation    1.0 0.0 0.0  1.9
                center      0.0 -0.15 0.0
                children [
# Second arm
                USE LampArm ]
                }
            ]
        }
    ]
}
```



see the VRML scene

Virtual Reality Modeling Language

VRML

Scalatura di forme

Scaling Shapes

- Le forme possono avere qualsiasi dimensione in VRML. Si può creare un sistema solare o gli atomi di una molecola. Si possono creare librerie di oggetti e poi combinarli tra loro per creare nuovi mondi, ingrandendo alcune forme e rimpicciolendone altre, per renderle tra loro compatibili nelle dimensioni.
- Mediante il nodo VRML `Transform` ed i campi `scale` e `scaleOrientation` si possono ridimensionare forme o gruppi di forme alle dimensioni che si desidera.
- VRML consente di scalare un sistema di coordinate , aumentando o diminuendo le dimensioni relative ad un sistema di coordinate padre.

Scaling Shapes (i)

- Il fattore di scala è un numero maggiore o uguale 0.0. Fattori di scala tra 0.0 e 1.0 riducono le dimensioni, mentre fattori maggiori di 1.0 le aumentano.
- Il fattore di scala è un fattore moltiplicativo
- Il parametro `scale` è espresso con tre numeri reali, che indicano rispettivamente il fattore di scala nelle direzioni X, Y e Z.
- Se si desidera che l'operazione di scala avvenga lungo un qualsiasi asse, non solo lungo X, Y o Z, va specificato il campo `scaleOrientation` che fornisce un asse ed un angolo di rotazione. Dopo l'operazione di scala il sistema di coordinate non rimane ruotato. Se si desidera che rimanga ruotato, occorre aggiungere un'operazione di *rotate.*

Scaling Shapes (ii)

- Per default il centro da cui ha origine l'operazione di *scale* è l'origine del sistema di coordinate. Si può specificare un altro centro di *scale* ovunque.
- Ciò è utile quando si vuole che una parte della forma rimanga inalterata, mentre il resto venga scalato.
- Utilizzando il campo `center` si specifica una coordinata 3-D come centro di *scale* del nuovo sistema di coordinate. In questo caso il valore di `scale` serve ad indicare il fattore di scala rispetto al centro indicato. Il campo è usato anche come centro di rotazione, per cui le operazioni di scala e rotazione intorno allo stesso centro possono avvenire in contemporanea.

Il nodo Transform

- Il nodo **Transform** crea un nuovo sistema di coordinate relativo al suo sistema di coordinate padre. Le Forme create come **children** del nodo **Transform** , sono create centrate nell'origine del nuovo sistema di coordinate

```
Transform {
        children    [ ]                          # exposedField MFNode
        translation 0.0 0.0 0.0                  # exposedField SFVec3f
        rotation  0.0 0.0 1.0 0.0                # exposedField SFRotation
        scale   1.0  1.0 1.0                     # exposedField SFVec3f
        scaleOrientation 0.0 0.0 1.0 0.0         # exposedField SFRotation
        bboxCenter 0.0 0.0 0.0                   # Field SFVec3F
        bboxsize -1.0 -1.0 -1.0                  # Field SFVec3F
        center 0.00 0.0 0.0                      # exposedField SFVec3f
        addChildren                              # eventIn MFNode
        removeChildren                           # eventIn MFNode
}
```

Il nodo Transform: `scale`

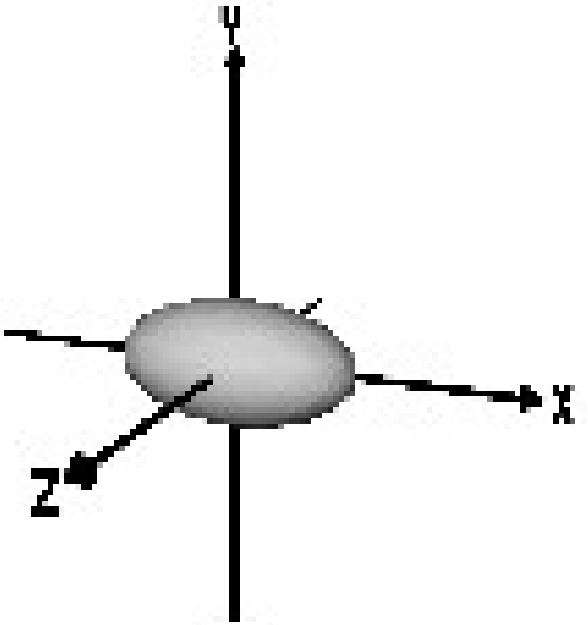
- Il valore di **`scale`** specifica l'entità di scale lungo l'asse X, Y e Z rispettivamente. Il valore di default di 1.0 lascia inalterato il mondo. Un valore pari a 0.0 collassa il mondo in niente.
- Il valore dell'exposed-field **`scale`** può essere cambiato mediante istradamento di un evento all'exposed-field implicito **`set_ scale`** (**`eventIn`**)
- Quando l'evento è ricevuto, viene definito il campo **`scale`** e il nuovo valore viene propagato utilizzando l'exposed-field implicito **`scale _changed`** (**`eventOut`**)
- Il valore di **`scaleOrientation`** specifica un asse immaginario di rotazione (mediante una coordinata 3-D) ed un angolo di rotazione espresso in radianti.

Il nodo Transform: `scaleOrientation`

- Il valore dell'exposed-field **`scaleOrientation`** può essere cambiato mediante istradamento di un evento all'exposed-field implicito **`set_ scaleOrientation`** (**`eventIn`**)
- Quando l'evento è ricevuto, viene definito il campo **`scaleOrientation`** e il nuovo valore viene propagato utilizzando l'exposed-field implicito **`scaleOrientation _changed`** (**`eventOut`**)
- Il valore dell'exposed-field **`center`** può essere cambiato mediante istradamento di un evento all'exposed-field implicito **`set_center`** (**`eventIn`**)
- Quando l'evento è ricevuto, viene definito il campo **`center`** e il nuovo valore viene propagato utilizzando l'exposed-field implicito **`center _changed`** (**`eventOut`**)

Esempi: scaling in X

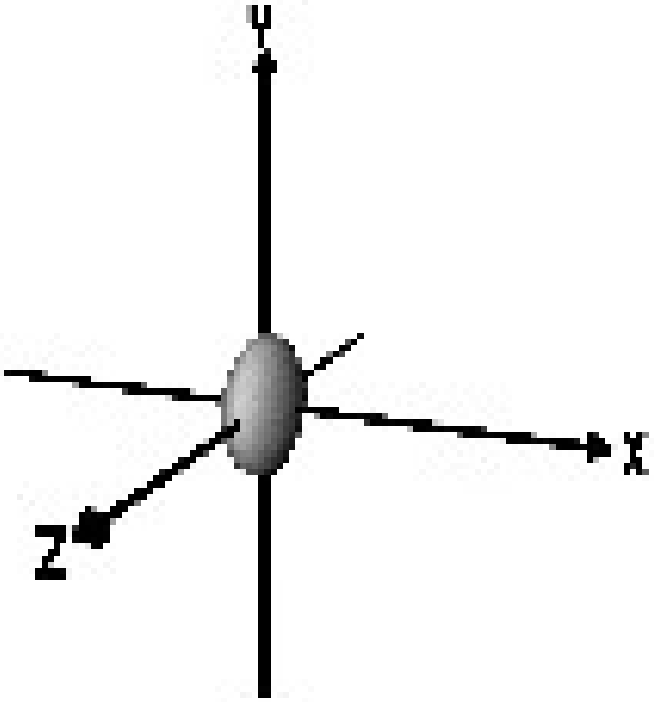
```
#VRML V2.0 utf8
Transform {
   scale 2.0 1.0 1.0
   children [
      Shape {
            appearance Appearance

   material Material{ }}
          geometry Sphere { }
             }
          ]
}
```



- see the VRML scene

Esempi: scaling in X (i)

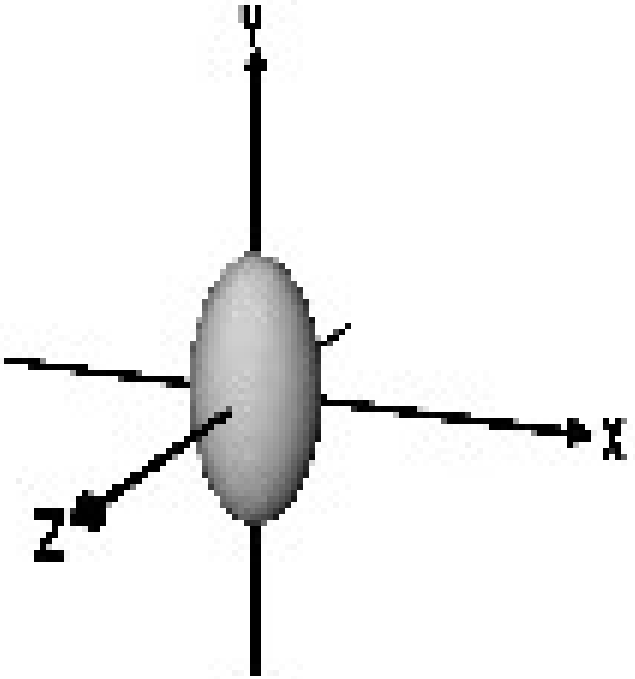
```
#VRML V2.0 utf8
Transform {
   scale 0.5 1.0 1.0
   children [
      Shape {
            appearance Appearanc

   material Material{ }}
          geometry Sphere { }
             }
          ]
}
```



- see the VRML scene

Esempi: scaling in Y

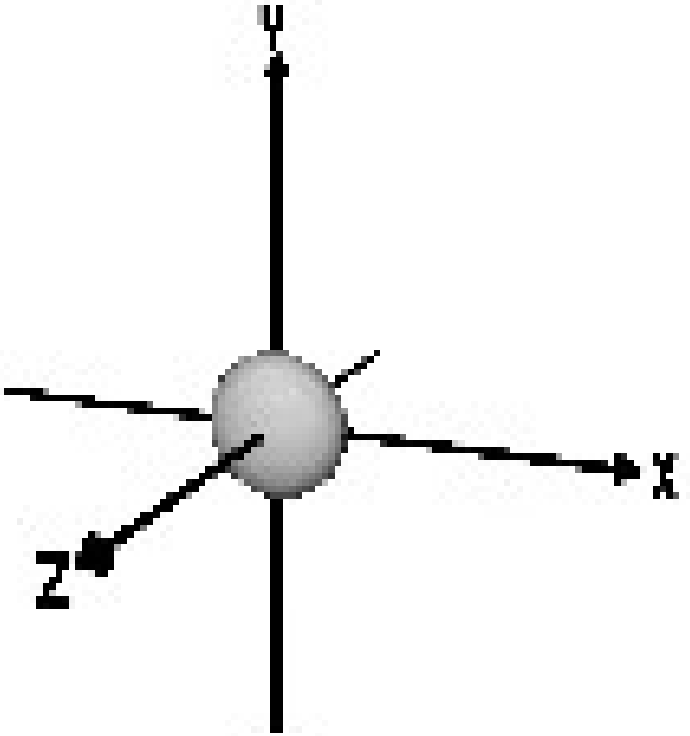
```
#VRML V2.0 utf8
Transform {
   scale 1.0 2.0 1.0
   children [
      Shape {
            appearance Appearanc

   material Material{ }}
          geometry Sphere { }
             }
          ]
}
```



- see the VRML scene

Esempi: scaling in Z

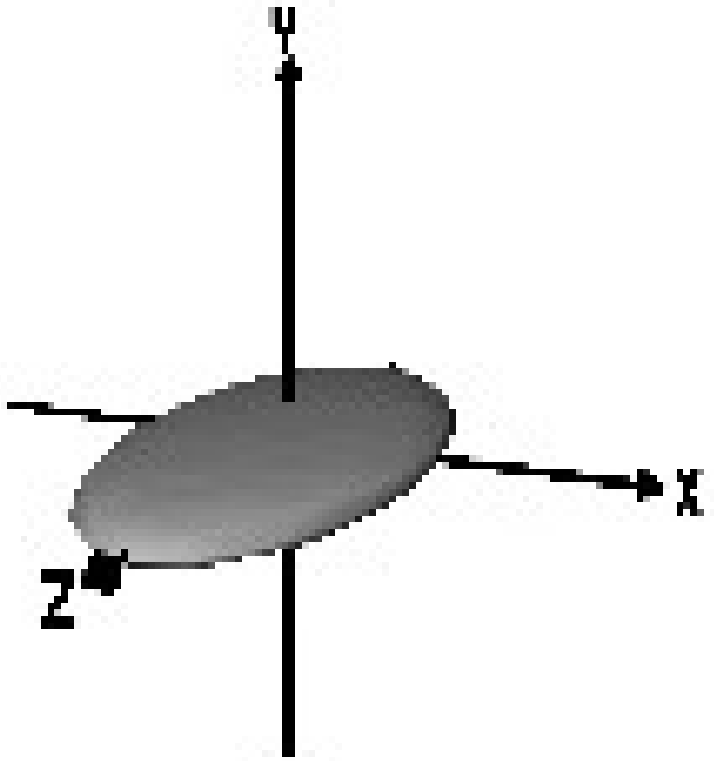
```
#VRML V2.0 utf8
Transform {
   scale 1.0 1.0 0.5
   children [
      Shape {
            appearance Appearance

   material Material{ }}
          geometry Sphere { }
             }
          ]
}
```



- see the VRML scene

Esempi: scaling in X Y Z

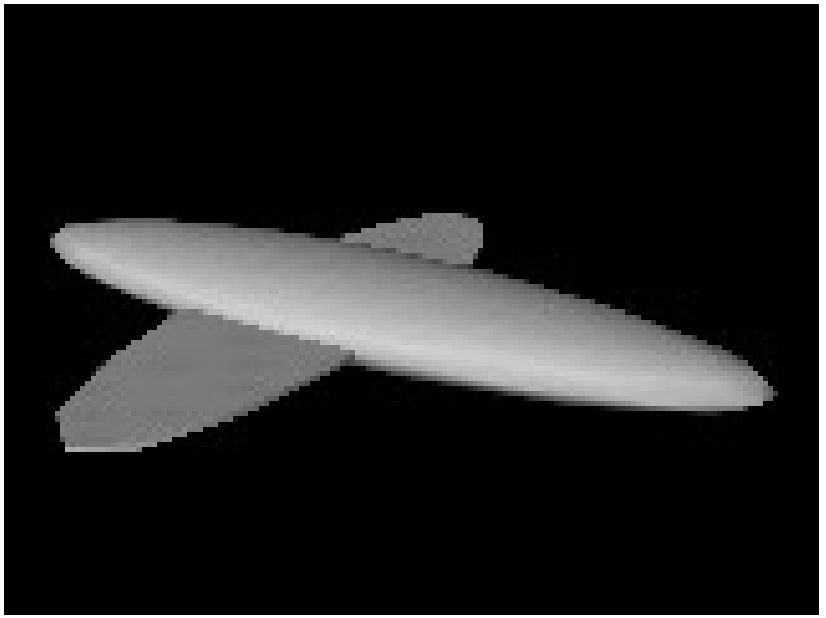
```
#VRML V2.0 utf8
Transform {
   scale 2.0 0.5 4.0
   children [
      Shape {
            appearance Appearanc

   material Material{ }}
          geometry Sphere { }
             }
          ]
}
```



- see the VRML scene

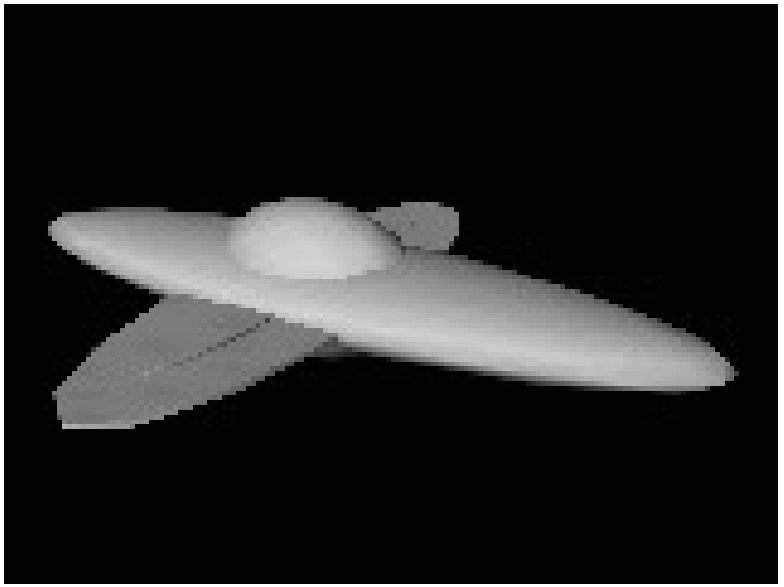
multiple scaled coord. systems

```
#VRML V2.0 utf8
Group {
   children [
# Wing
       Transform {
           scale 0.5 1.0 1.5
              children Shape {
                  appearance DEF White Appearance {
                      material Material { }
                  }
                  geometry Cylinder {
                      radius 1.0
                      height 0.025  }
                }
            },
# Fuselage
       Transform {
              scale 2.0 0.2 0.5
              children Shape {
                  appearance USE White
                  geometry Sphere { }
                }
            }
       ]
}
```



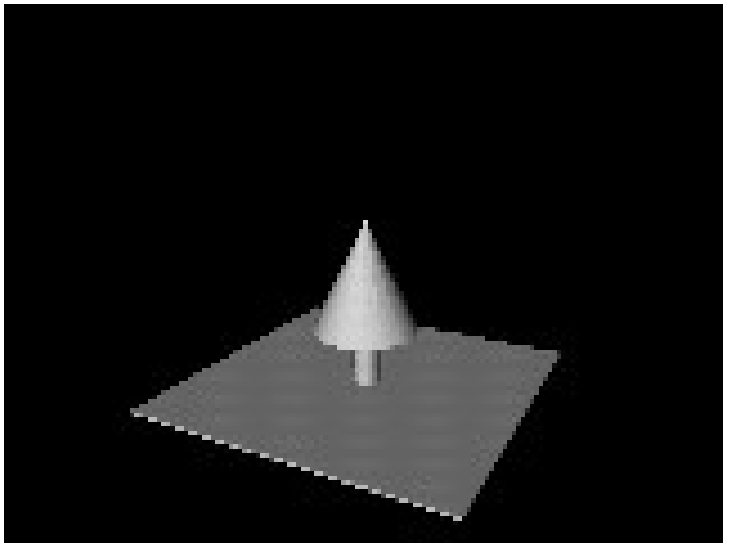
nesting scaled coord. systems

```
#VRML V2.0 utf8
Group {
   children [
# Wing
        DEF Wing Transform {
            scale 0.5 1.0 1.5
              children Shape {
                  appearance DEF White Appearance {
                      material Material { }
                  }
                  geometry Cylinder {
                      radius 1.0
                      height 0.025  }
                }
            },
# Fuselage
        DEF Fuselarge Transform {
              scale 2.0 0.2 0.5
              children Shape {
                  appearance USE White
                  geometry Sphere { }
                }
            },

 # Wing detail and fusalage dome
        Transform {
              scale 0.3 2.0 0.75
              children [
                  USE Wing,
                  USE Fusalage
                   ]
                }
        ]
}
```



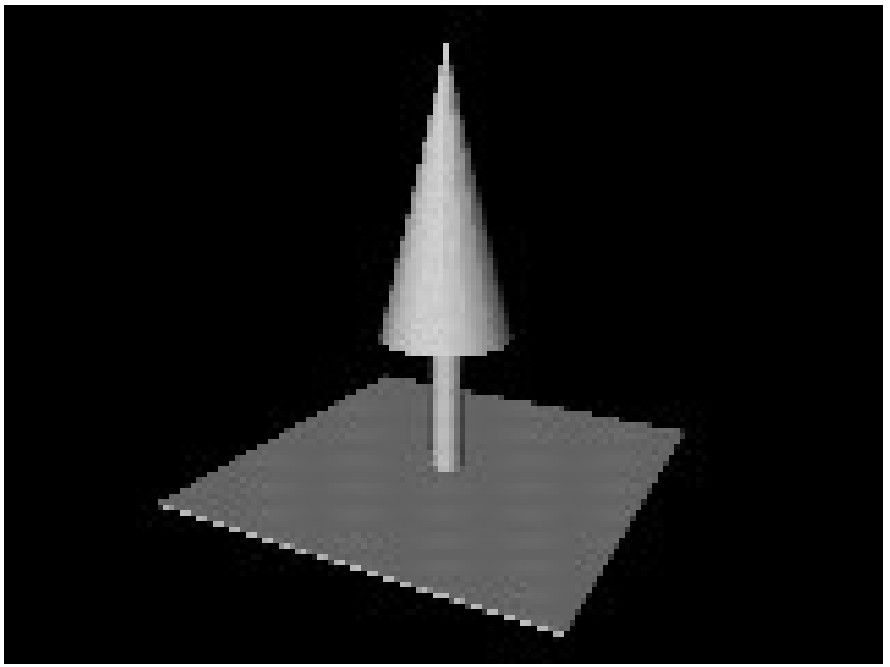
scaling about a center point (a)

```
#VRML V2.0 utf8
Group {
   children [
# Ground
        Shape {
            appearance DEF White Appearance{
                material Material { }       }
            geometry Box {
                 size 12.0 0.1 12.0}
           },
# Tree
    Transform {
            translation 0.0 1.0 0.0
            scale       1.0 1.0 1.0
            center      0.0 -1.0 0.0
            children [
# Trunk
               Shape {
                    appearance USE White
                    geometry Cylinder {
                        radius 0.5
                        height 2.0}
                },
# Branches
                Transform {
                    translation 0.0 3.0 0.0
                    children Shape {
                        appearance USE White
                        geometry Cone {
                            bottomRadius 2.0
                            height 4.0}
                    }
                }
              ]
        }
    ]
}
```



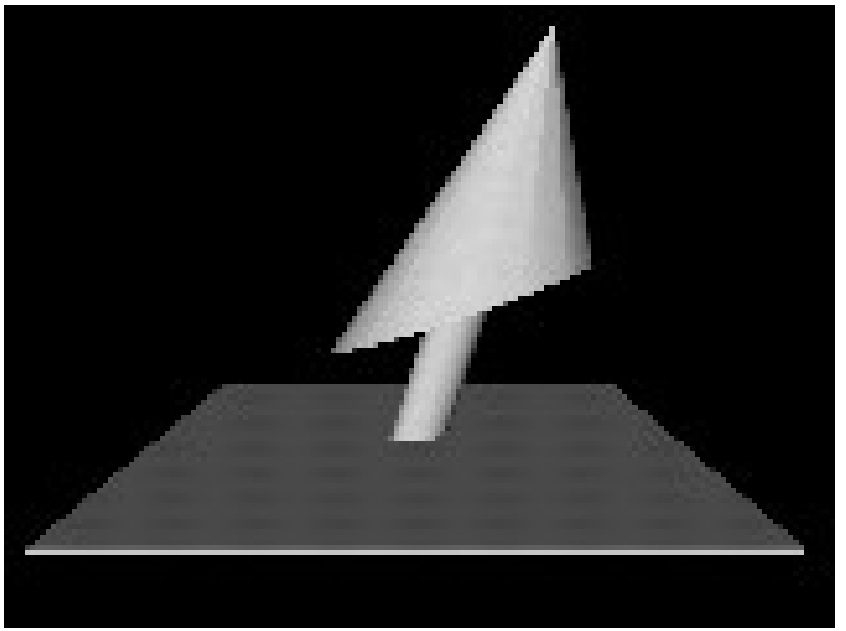
scaling about a center point (b)

```
#VRML V2.0 utf8
Group {
   children [
# Ground
        Shape {
             appearance DEF White Appearance{
                 material Material { }       }
             geometry Box {
                  size 12.0 0.1 12.0}
            },
# Tree
    Transform {
             translation 0.0 1.0 0.0
             scale       1.0 2.0 1.0
             center      0.0 -1.0 0.0
             children [
# Trunk
                Shape {
                     appearance USE White
                     geometry Cylinder {
                         radius 0.5
                         height 2.0}
                 },
# Branches
                 Transform {
                     translation 0.0 3.0 0.0
                     children Shape {
                         appearance USE White
                         geometry Cone {
                             bottomRadius 2.0
                             height 4.0}
                     }
                 }
               ]
        }
    ]
}
```



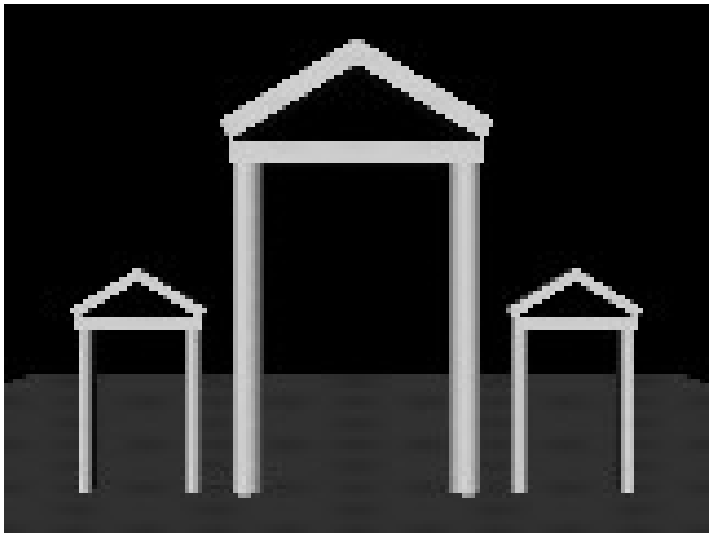
scaling orientation

```
#VRML V2.0 utf8
Group {
   children [
# Ground
        Shape {
             appearance DEF White Appearance{
                 material Material { }       }
             geometry Box {
                  size 12.0 0.1 12.0}
            },
# Tree
    Transform {
             translation 0.0 1.0 0.0
             scale       1.0 2.0 1.0
             scaleOrientation 0.0 0.0 1.0 -0.875
             center      0.0 -1.0 0.0
             children [
# Trunk
                Shape {
                     appearance USE White
                     geometry Cylinder {
                         radius 0.5
                         height 2.0}
                 },
# Branches
                 Transform {
                     translation 0.0 3.0 0.0
                     children Shape {
                         appearance USE White
                         geometry Cone {
                             bottomRadius 2.0
                             height 4.0}
                     }
                 }
               ]
        }
    ]
}
```



Translating and scaling coord. systems

```
#VRML V2.0 utf8
Group {
    children [
# Ground
        Shape {
            appearance DEF White Appearance {
                material Material { }       }
            geometry Box {
                    size 25.0 0.1 25.0 }
        },
# First archway
          DEF Archway Group {
           children [
# Left Column
                DEF LeftColumn Transform {
                    translation -2.0 3.0 0.0
                    children DEF Column Shape {
                        appearance USE White
                        geometry Cylinder {
                            radius 0.3
                            height 6.0}
                    }
                 },
# Right Column
                DEF RightColumn Transform {
                    translation 2.0 3.0 0.0
                    children USE Column
                },
# Archway span
                DEF ArchwaySpan Transform {
                    translation 0.0 6.05 0.0
                    children Shape {
                        appearance USE White
                        geometry Box {
                            size 4.6 0.4 0.6}
                    }
                },
# Left Roof
                DEF LeftRoof Transform {
                    translation -1.15 7.12 0.0
                    rotation 0.0 0.0 1.0  0.524
                    children DEF Roof Shape {
                        appearance USE White
                        geometry Box {
                            size 2.86 0.4 0.6}
                },
 # Right Roof
                DEF RightRoof Transform {
                    translation 1.15 7.12 0.0
                    rotation 0.0 0.0 1.0  -0.524
                    children USE Roof
                }
            ]
        },
```



```
# Left small archway
        Transform {
            translation -4.0 0.0 0.0
            scale 0.5 0.5 0.5
            children USE Archway
        },
# Right small archway
Transform {
            translation 4.0 0.0 0.0
            scale 0.5 0.5 0.5
            children USE Archway
        }
    ]
}
```

Virtual Reality Modeling Language

VRML

Animazioni

Animazioni

- Per aggiungere movimento al mondo, si può animare la posizione, l'orientazione e lo scaling di ogni sistema di coordinate.
- Se il sistema di coordinate si muove, ogni forma costruita in quel sistema di coordinate si muove con esso.
- Animando un sistema di coordinate o un gruppo di forme, si fa in modo che queste forme volino nel nostro mondo, si spostino, ruotino e "scalino" come si desidera.
- Per attivare, fermare e comunque controllare l'animazione, il nodo VRML `TimeSensor` funziona da orologio. Al passare del tempo questo nodo genera eventi che indicano cambiamenti del tempo

Animazioni (i)

- Istradando questi eventi dall'`eventOut` del nodo `TimeSensor` ad altri nodi, si può fare in modo che questi nodi cambino al cambiare del tempo in `TimeSensor`.
- Per fare sì che il sistema di coordinate venga traslato, ruotato, o scalato, si possono istradare gli eventi ai nodi `PositionInterpolator` e `OrientationInterpolator`.
- Ciascuno di questi nodi genera nuove posizioni e valori di rotazione, che vengono inviati attraverso i suoi `eventOut`.
- L'istradamento di questi valori al nodo `Transform` causa la traslazione, rotazione o scalatura del sistema di coordinate di questo nodo al procedere dell'animazione.

Animazioni (ii)

- Il concetto di animazione indica il cambiare di qualcosa al cambiare del tempo.
- Il cambiamento può interessare una modifica della posizione del sistema di coordinate, facendo sì che il sistema di riferimento e le forme generate in esso si spostino da un posto all'altro mano mano che il tempo passa.
- Il cambiamento causato da un'animazione può riguardare anche la posizione, l'orientamento e la "scala" del sistema di coordinate.
- Ogni animazione richiede due elementi:
  - Un clock che controlli l'evoluzione dell'animazione
  - Una descrizione di come qualcosa cambia nel corso dell'animazione

Animazioni (iii)

- Si può creare un clock usando il nodo `TimeSensor`. Il nodo fornisce funzioni per attivare, fermare o controllare quanto velocemente un'animazione si sviluppa.
- Per descrivere i cambiamenti che hanno luogo nel corso dell'animazione si possono usare i nodi VRML `PositionInterpolator` e `OrientationInterpolator`. Ciascuno dei due nodi prende informazioni dal clock per stabilire quali valori assumere per posizione o orientamento, all'interno di una lista fornita dall'utente. Al progredire dell'animazione vengono selezionati e prodotti nuovi valori.
- Per creare un circuito per l'animazione si deve usare la sintassi del costrutto `ROUTE` del VRML per collegare l'output del nodo `TimeSensor` con l'input di `OrientationInterpolator` e `PositionInterpolator`. Si può poi usare un'altra istruzione di `ROUTE` per collegare l'output di un nodo di interpolazione con un campo `eventIn` del nodo `Transform` .

Animazioni (iv)

- Una volta collegato il nodo `TimeSensor` si può attivare il clock del nodo. Gli eventi fluiscono dal nodo `TimeSensor` al nodo di interpolazione e da questi al nodo `Transform`, causando lo spostamento, la rotazione o la scalatura del sistema di coordinate in base all'evoluzione dell'animazione.
- Il nodo `TimeSensor` si comporta come un orologio nel mondo reale. Quando è attivato continua a camminare fino a che non viene spento.
- Mentre `TimeSensor` cammina, produce eventi attraverso il campo `eventOut time`, che indica l'ora attuale.
- Per abilitare il nodo `TimeSensor` a comportarsi come un vero orologio, occorre considerare che il suo è un tempo assoluto misurato in secondi a partire dalla mezzanotte del 1 gennaio 1970 GMT.
- Un nodo può essere fermato o attivato ad una data specifica fornendo il tempo assoluto per i campi `startTime` e `stopTime` del nodo.

Animazioni (v)

- Normalmente i valori dei campi `startTime` e `stopTime` vengono selezionati come output di altri nodi VRML, piuttosto che essere inseriti a mano dal programmatore.
- Quando il nodo `TimeSensor` è attivato, esso produce in output il valore `TRUE` nell' **`eventOut isActive`**. Collegando opportunamente a questo campo altri nodi, possono essere attivate delle animazioni. Analogamente, quando il nodo viene fermato, esso produce in output il valore `FALSE` nell' **`eventOut isActive`**.
- Per attivare un'animazione può essere conveniente utilizzate un tempo frazionario: l'animazione parte al tempo `0.0` e finisce al tempo `1.0`.
- In questo modo l'animazione può avvenire a prescindere dal valore esatto dell'ora attuale e dal come e quando l'animazione ha luogo.
- Il nodo `TimeSensor` produce il tempo frazionario nel suo **`eventOut`** `fraction_changed`. Il nodo all'attivazione produce sempre il tempo frazionario `0.0`

Animazioni (vi)

- Anche l'intervallo di tempo che intercorre tra i tempi frazionari `0.0` e `1.0` è indipendente dal tempo assoluto. Il tempo che intercorre tra i due tempi frazionari viene definito come *cycle interval* e viene specificato attraverso il campo **`cycleinterval`** del nodo **`TimeSensor`**. Maggiore è il valore di **`cycleinterval`**, più lenta risulta l'animazione.
- Si può produrre un loop tra i tempi frazionari `0.0` e `1.0` al fine di produrre delle animazioni infinite, assegnado il valore **`TRUE`** al campo **`loop`** del nodo **`TimeSensor`**.
- Quando si usa un loop, il nodo **`TimeSensor`** produce in output il tempo assoluto, ogni volta che il ciclo ha inizio, attraverso l'**`eventOut`** **`cycleTime`**. Questo tempo assoluto può essere utilizzato per attivate o disattivare altre animazioni o per sincronizzare l'animazione con altre animazioni nel mondo virtuale.

Il nodo TimeSensor

- Il nodo **TimeSensor** crea un orologio che genera eventi che controllano le animazioni.

```
TimeSensor {
        enabled             TRUE            # exposedField SSFBool
        startTime           0.0             # exposedField SFTime
        stopTime            0.0             # exposedField SFTime
        cycleInterval       1.0             # exposedField SFTime
        loop                FALSE           # exposedField SSFBool
        isActive                            # eventOut     SSFBool
        time                                # eventOut     SFTime
        cycleTime                           # eventOut     SFTime
        fraction_changed                    # eventOut     SFFloat
}
```

Il nodo TimeSensor

- Il valore dell'exposed-field **enabled** specifica se il nodo è attivo o no. Se il valore del campo è **TRUE**, il sensore è attivo ed il resto dei campi sono utilizzati per controllare l'output del sensore.
- Se il campo è **FALSE**, il sensore è inattivo e non ci sono campi che vengono generati in output, ad eccezione di quelli generati in risposta del cambiamento di un exposed-field. Il valore di default è **TRUE**.
- Il valore dell'exposed-field **startTime** specifica il tempo al quale, se il sensore è attivo, deve produrre eventi attraverso i suoi **eventOut** . Il suo valore è il solito tempo assoluto a partire dalla mezzanotte del 1 gennaio 1970 GMT. Il valore di default è **0.0** sec.

Il nodo TimeSensor (i)

- Il valore dell'exposed-field **`stopTime`** specifica il tempo al quale cessa di produrre eventi attraverso i suoi **`eventOut`**. Il suo valore è un tempo assoluto ed il default è **`0.0`** sec.
- Il valore dell'exposed-field **`cycleInterval`** specifica la quantità di tempo con la quale il sensore passa dalla frazione di tempo **`0.0`** alla frazione **`1.0`**. Il valore di default è **`1.0`**.
- Il valore dell'exposed-field **`loop`** specifica se il sensore è attivo in ciclo continuo oppure no. Se il valore è **`TRUE`** il sensore emette durante il ciclo frazioni di tempo da **`0.0`** a **`1.0`** . Alla fine del ciclo il valore del tempo ritorna a **`0.0`** ed il ciclo ricomincia. Se il valore è **`FALSE`** il sensore non è in attivo a ciclo continuo: effettua un ciclo e poi cessa l'attività. Il default è **`FALSE`**

Il nodo TimeSensor (ii)

- I campi **startTime**, **stopTime**, **cycleInterval** e **loop** lavorano insieme per controllare l'output del nodo **TimeSensor**. La tabella indica alcuni effetti standard:

loop	startTime stopTime cycleInterval	*effetto*
TRUE	stopTime <= startTime	gira sempre
TRUE	stopTime > startTime	Gira fino a stopTime
FALSE	stopTime <= startTime	Gira per un ciclo (si ferma a startTime+cycleinterval)
FALSE	stopTime => (startTime + cycleinterval) >startTime	Gira per un ciclo (si ferma a startTime+cycleinterval)
FALSE	(startTime+ cycleinterval) > stopTime > startTime	Gira per meno di un ciclo e si ferma a stopTime

Il nodo TimeSensor (iii)

- L'`eventOut` **`isActive`** produce un singolo evento **`TRUE`** quando il nodo diventa attivo e poi continua a produrre eventi. Invia un singolo evento **`FALSE`** quando il nodo diventa inattivo.
- L'`eventOut` **`Time`** produce continuamente un evento espresso in un tempo assoluto fino a che il nodo è attivo.
- L'`eventOut` **`cycleTime`** produce un evento espresso in un tempo assoluto ogni volta che un ciclo ricomincia. Se il campo **`loop`** è **`FALSE`**, esso produce un solo evento per l'unica volta che il ciclo viene eseguito.

Il nodo TimeSensor (iv)

- L'**eventOut** **fraction_changed** produce una frazione di tempo, un numero reale tra `0.0` e `1.0` mano mano che il ciclo progredisce da `0.0` a `1.0`.
- Il valore dell'exposed-field **enabled** può essere cambiato istradando un evento all'exposed-field implicito **set_enabled**. Se viene ricevuto un valore **TRUE** ed il sensore è inattivo, allora il sensore diventa attivo. Se viene ricevuto un valore **FALSE** ed il sensore era attivo, esso diventa inattivo dopo aver prodotto l'insieme finale di valori di output attraverso i suoi **eventOut**. Se un valore **TRUE** viene ricevuto a sensore attivo, o un valore **FALSE** viene ricevuto a sensore inattivo, gli eventi vengono ignorati. Ogni volta che il sensore cambia stato, il nuovo valore viene propagato attraverso l'**eventOut** implicito **enabled_changed**

Il nodo TimeSensor (v)

- Analogamente succede per gli exposed-field **`loop`**, **`startTime`**, **`stopTime`** e **`cycleInterval`** che possono essere modificati istradando degli eventi attraverso gli exposed-field (**`eventIn`**) impliciti.
- Il nodo **`TimeSensor`** non produce forme e non ha effetti visibili nel mondo. Un nodo **`TimeSensor`** può essere inserito in ogni nodo che raggruppa oggetti, ma è indipendente dal sistema di coordinate. In genere un nodo **`TimeSensor`** viene posto alla fine del gruppo più esterno di un file VRML.

Il nodo TimeSensor (vi)

- La frequenza esatta con cui un nodo `TimeSensor` produce eventi dipende dalla potenza del computer in uso, dal carico di lavoro del computer e da quello del browser VRML.
- Il campo `loop` va usato con estrema cautela perché potrebbe produrre una massa di eventi così ampia che il browser VRML non riesce a processare.
- Il browser si troverebbe nella condizione di non avere il tempo di rappresentare gli altri nodi e ridisegnare il mondo virtuale tra un evento e l'altro.

Il nodo PositionInterpolator

- Il nodo **PositionInterpolator** descrive una serie di posizioni chiave disponibili per essere usate in un'animazione.

```
PositionInterpolator {
        key             [ ]         # exposedField MFfloat
        keyValue        [ ]         # exposedField MFVec3f
        set_fraction                # eventIn      SFfloat
        value_changed               # eventOut     SFVec3f
        }
```

Il nodo PositionInterpolator

- Il valore dell'exposed-field **`key`** specifica una lista di frazioni di tempo. In genere questi tempi sono compresi tra `0.0` e `1.0`. Le frazioni di tempo possono essere comunque valori positivi o negativi di ogni grandezza. I tempi devono essere elencati in ordine non decrescente. Il valore di default è una lista vuota.
- Il valore dell'exposed-field **`keyValue`** specifica una lista di posizioni chiave. Ogni posizione chiave è una terna di valori contenenti i valori X, Y e Z di una coordinata 3-D o una distanza relativa a una traslazione. In alcuni casi indicano un fattore di scala o altri gruppi di tre numeri reali. Il valore di default è una lista vuota.

Il nodo PositionInterpolator (i)

- I tempi e le posizioni sono utilizzati congiuntamente, in modo che il primo tempo indichi la prima posizione chiave, il secondo la successiva posizione chiave, e così via. Le liste possono indicare qualsiasi numero di posizioni, ma entrambe devono avere lo stesso numero di valori.
- Quando un nodo **`PositionInterpolator`** riceve un valore di frazione di tempo, esso calcola la posizione sulla base dei valori presenti nella lista delle posizioni chiave e delle relative frazioni di tempo. La nuova posizione calcolata viene propagata attraverso l'**`eventOut`** **`value_changed`**.

Il nodo PositionInterpolator (ii)

- La posizione di output corrispondente ad una determinata frazione di tempo t viene calcolata dal nodo `PositionInterpolator` nel modo seguente:
  - analizza le frazioni di tempo per trovare una coppia di valori tali che $t_1 \leq t \leq t_2$
  - individua le corrispondenti posizioni ai due valori
  - individua la posizione intermedia corrispondente a t mediante interpolazione lineare
- Due posizioni adiacenti potrebbero avere lo stesso valore e causare una discontinuità nel percorso dell'animazione. In questo caso il nodo prende la posizione corrispondente a t_1 per interpolare linearmente tempi inferiori a t_1 e la posizione corrispondente a t_2 quando deve interpolare la posizione di frazioni di tempo maggiori di t_2.

Il nodo PositionInterpolator (iii)

- Le frazioni di tempo sono inviate attraverso l' **eventIn** **set_fraction** creando una route con l' **eventOut** **fraction_changed** di un nodo **TimeSensor**.
- La lista delle posizioni e dei tempi possono essere cambiate inviando dei valori agli **eventIn** impliciti **set_key** e **set_keyValue**, ed i nuovi valori propagati attraverso gli eventsOut **key_changed** e **keyValue_changed**.
- Il nodo **PositionInterpolator** non produce forme e non ha effetti visibili nel mondo. Un nodo **Position-Interpolator** può essere inserito in ogni nodo che rag-gruppa oggetti, ma è indipendente dal sistema di coordi-nate. In genere un nodo **PositionInterpolator** viene posto alla fine del gruppo più esterno di un file VRML.

Il nodo OrientationInterpolator

- Il nodo `OrientationInterpolator` descrive una serie di rotazioni chiave disponibili per essere usate in un'animazione.

```
OrientationInterpolator {
        key                [ ]          # exposedField MFfloat
        keyValue           [ ]          # exposedField MFRotation
        set_fraction                    # eventIn      SFfloat
        value_changed                   # eventOut     SFRotation
        }
```

Il nodo OrientationInterpolator

- Il valore dell'exposed-field **`key`** specifica una lista di frazioni di tempo. In genere questi tempi sono compresi tra `0.0` e `1.0`. Le frazioni di tempo possono essere comunque valori positivi o negativi di ogni grandezza. I tempi devono essere elencati in ordine non decrescente. Il valore di default è una lista vuota.
- Il valore dell'exposed-field **`keyValue`** specifica una lista di rotazioni chiave. Ogni rotazione chiave è una quaterna di valori contenenti i valori X, Y e Z di una coordinata 3-D che identifichi l'asse di rotazione e il quarto campo un angolo in radianti che esprime la rotazione rispetto a tale asse. Il valore di default è una lista vuota.

Il nodo OrientationInterpolator (i)

- I tempi e le rotazioni sono utilizzati congiuntamente, in modo che il primo tempo indichi la prima rotazione chiave, il secondo la successiva rotazione chiave, e così via. Le liste possono indicare qualsiasi numero di rotazioni, ma entrambe devono avere lo stesso numero di valori.
- Quando un nodo **`OrientationInterpolator`** riceve un valore di frazione di tempo, esso calcola la rotazione sulla base dei valori presenti nella lista delle rotazioni chiave e delle relative frazioni di tempo. La nuova rotazione calcolata viene propagata attraverso l'**`eventOut value_changed`**.

Il nodo OrientationInterpolator (ii)

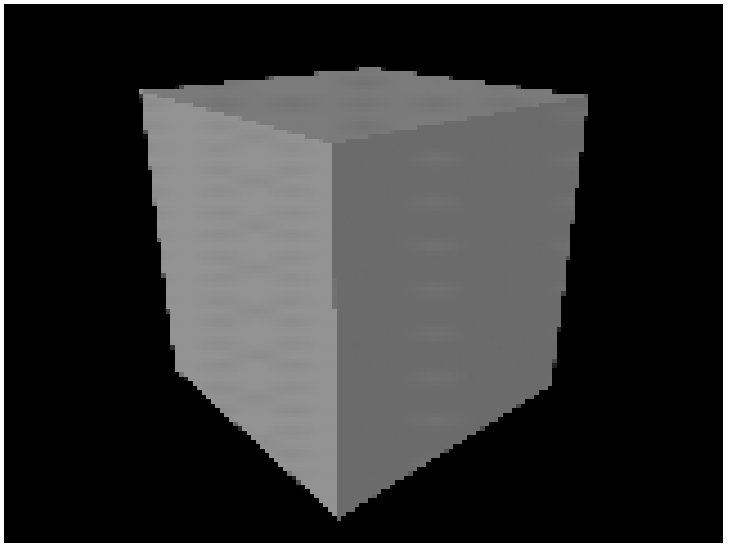
- La posizione di output corrispondente ad una determinata frazione di tempo t viene calcolata dal nodo `OrientationInterpolator` nel modo seguente:
  - analizza le frazioni di tempo per trovare una coppia di valori tali che $t_1 \leq t \leq t_2$
  - individua le corrispondenti rotazioni ai due valori
  - individua la rotazione intermedia corrispondente a t mediante interpolazione lineare
- Due rotazioni adiacenti potrebbero avere lo stesso valore e causare una discontinuità nel percorso dell'animazione. In questo caso il nodo prende la rotazione corrispondente a t_1 per interpolare linearmente tempi inferiori a t_1 e la rotazione corrispondente a t_2 quando deve interpolare la rotazione di frazioni di tempo maggiori di t_2.

Il nodo OrientationInterpolator (iii)

- Le frazioni di tempo sono inviate attraverso l'**eventIn** **set_fraction** creando una route con l'**eventOut** **fraction_changed** di un nodo **TimeSensor**.
- La lista delle rotazioni e dei tempi possono essere cambiate inviando dei valori agli **eventIn** impliciti **set_key** e **set_keyValue**, ed i nuovi valori propagati attraverso gli eventsOut **key_changed** e **keyValue_changed**.
- Il nodo **OrientationInterpolator** non produce forme e non ha effetti visibili nel mondo. Un nodo **OrientationInterpolator** può essere inserito in ogni nodo che raggruppa oggetti, ma è indipendente dal sistema di coordinate. In genere un nodo **OrientationInterpolator** viene posto alla fine del gruppo più esterno di un file VRML.

Animating positions

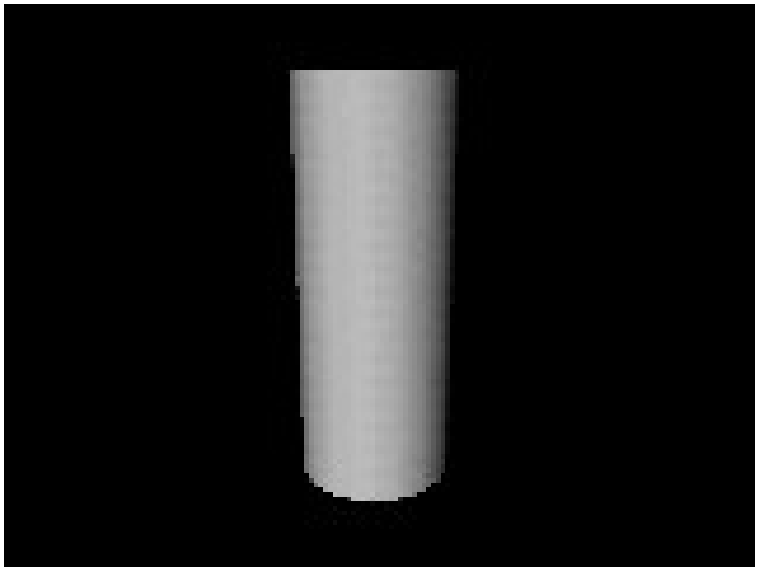
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau,
# and John L. Moreland Group
group {
    children [
    # Moving box
        DEF Cube Transform {
        children Shape {
            appearance Appearance {
               material Material { }
            }
            geometry Box { size 1.0 1.0 1.0 }
        }
      },
    # Animation clock
       DEF Clock TimeSensor {
            cycleInterval 4.0
            loop TRUE
       },
     # Animation path
        DEF CubePath PositionInterpolator {
             key [ 0.00, 0.11, 0.17, 0.22,
                   0.33, 0.44, 0.50, 0.55,
                   0.66, 0.77, 0.83, 0.88,
                   0.99 ]
             keyValue [ 0.0 0.0 0.0, 1.0 1.96 1.0,
                        1.5 2.21 1.5, 2.0 1.96 2.0,
                        3.0 0.0 3.0, 2.0 1.96 3.0,
                        1.5 2.21 3.0, 1.0 1.96 3.0,
                        0.0 0.0 3.0, 0.0 1.96 2.0,
                        0.0 2.21 1.5, 0.0 1.96 1.0,
                        0.0 0.0 0.0 ]
        }
    ]
}
ROUTE Clock.fraction_changed TO
                    CubePath.set_fraction
ROUTE CubePath.value_changed TO
                    Cube.set_translation
```



see the VRML scene

Animating rotations

```
#VRML V2.0 utf8
Group {
   children [
# Rotating cylinder
       DEF Column Transform {
           rotation 0.0 0.0 1.0 0.0
           children Shape {
               appearance Appearance {
                   material Material { }
               }
               geometry Cylinder {
               height 1.0 radius 0.2 }
               }
           },
# Animation clock
       DEF Clock TimeSensor {
           cycleInterval 4.0 loop TRUE },
# Animation path
       DEF ColumnPath OrientationInterpolator {
            key [ 0.0, 0.50, 1.0 ]
           keyValue [ 0.0 0.0 1.0 0.0,
                      0.0 0.0 1.0 3.14,
                      0.0 0.0 1.0 6.28 ]
           }
       ]
}
```

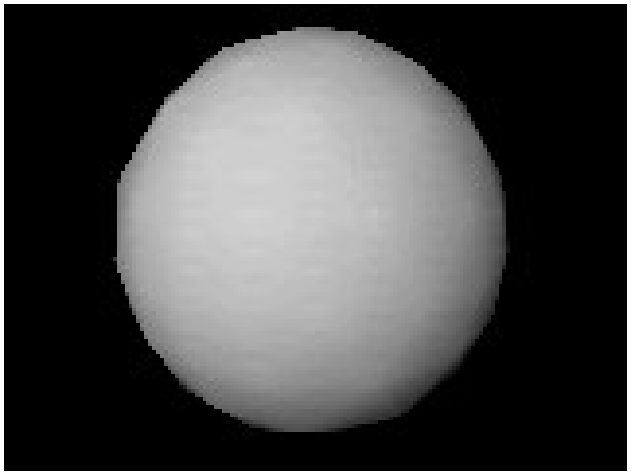


```
ROUTE Clock.fraction_changed TO
          ColumnPath.set_fraction
ROUTE ColumnPath.value_changed TO
          Column.set_rotation
```

see the VRML scene

Animating scale

```
#VRML V2.0 utf8
Group {
    children [
    # Pulsing ball
        DEF Ball Transform {
            children Shape {
                appearance Appearance {
                    material Material { }
                }
                geometry Sphere { }
            }
        },
    # Animation clock
        DEF Clock TimeSensor {
            cycleInterval 2.0
            loop TRUE
        },
    # Animation path
        DEF BallPath PositionInterpolator {
            key [ 0.0, 0.20, 0.65, 1.0 ]
            keyValue [
                1.0 1.0 1.0,
                1.5 1.5 1.5,
                1.1 1.1 1.1,
                1.0 1.0 1.0,
            ]
        }
    ]
}
```

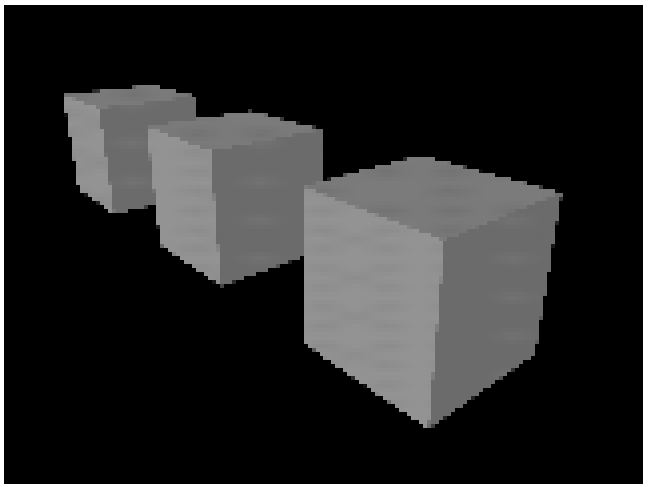


```
ROUTE Clock.fraction_changed TO
        BallPath.set_fraction
ROUTE BallPath.value_changed TO
        Ball.set_scale
```

see the VRML scene

Animating multiple shapes

```
#VRML V20 utf8
Group {
   children [
   # Moving box
      DEF Cube1 Transform {
          children DEF ACube Shape {
              appearance Appearance {
                  material Material { } }
              geometry Box { size 1.0 1.0 1.0 }
              }
          },
      Transform {
          translation -2.0 0.0 0.0
          children DEF Cube2 Transform {
              children USE ACube }
          },
      Transform {
          translation 2.0 0.0 0.0
          children DEF Cube3 Transform {
              children USE ACube }
          },
      # Animation clock
      DEF Clock TimeSensor {
          cycleInterval 4.0 loop TRUE },
      # Animation path
      DEF CubePath PositionInterpolator {
          key [ 0.00, 0.11, 0.17, 0.22,
                0.33, 0.44, 0.50, 0.55,
                0.66, 0.77, 0.83, 0.88, 0.99 ]
          keyValue [ 0.0 0.0 0.0, 1.0 1.96 1.0,
                     1.5 2.21 1.5, 2.0 1.96 2.0,
                     3.0 0.0 3.0, 2.0 1.96 3.0,
                     1.5 2.21 3.0, 1.0 1.96 3.0,
                     0.0 0.0 3.0, 0.0 1.96 2.0,
                     0.0 2.21 1.5, 0.0 1.96 1.0,
                     0.0 0.0 0.0 ]
```

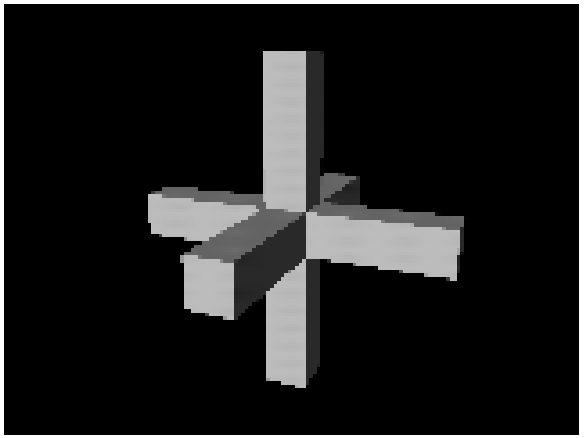


```
          }
      ]
}
ROUTE Clock.fraction_changed TO
CubePath.set_fraction
ROUTE CubePath.value_changed TO
Cube1.set_translation
ROUTE CubePath.value_changed TO
Cube2.set_translation
ROUTE CubePath.value_changed TO
Cube3.set_translation
```

see the VRML scene

Using multiple interpolators

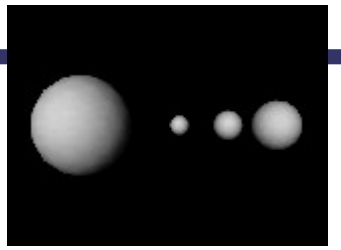
```
#VRML V2.0 utf8
Group {
children [
# Three rotating bars
   DEF Bar1 Transform {
        children Shape {
            appearance DEF White Appearance {
                material Material { } }
            geometry Box { size 1.5 0.2 0.2 }
            } },
   DEF Bar2 Transform {
        children Shape {
            appearance USE White
            geometry Box { size 0.2 1.5 0.2 }
            } },
   DEF Bar3 Transform {
        children Shape {
            appearance USE White
            geometry Box { size 0.2 0.2 1.5 }
            } },
# Animation clock
   DEF Clock TimeSensor {
        cycleInterval 4.0
        loop TRUE },
# Animation paths, one for each bar
   DEF BarPath1 OrientationInterpolator {
        key [ 0.0, 0.50, 1.0 ]
        keyValue [ 0.0 0.0 1.0 0.0,
                   0.0 0.0 1.0 3.14,
                   0.0 0.0 1.0 6.28 ] },
   DEF BarPath2 OrientationInterpolator {
        key [ 0.0, 0.50, 1.0 ]
        keyValue [ 1.0 0.0 0.0 0.0,
                   1.0 0.0 0.0 3.14,
                   1.0 0.0 0.0 6.28 ] },
   DEF BarPath3 OrientationInterpolator {
        key [ 0.0, 0.50, 1.0 ]
        keyValue [ 0.0 1.0 0.0 0.0,
                   0.0 1.0 0.0 3.14,
                   0.0 1.0 0.0 6.28 ] }
   ]
}
```



```
ROUTE Clock.fraction_changed TO BarPath1.set_fraction
ROUTE Clock.fraction_changed TO BarPath2.set_fraction
ROUTE Clock.fraction_changed TO BarPath3.set_fraction
ROUTE BarPath1.value_changed TO Bar1.set_rotation
ROUTE BarPath2.value_changed TO Bar2.set_rotation
ROUTE BarPath3.value_changed TO Bar3.set_rotation
```

see the VRML scene

Using multiple Time Sensors



```
#VRML V2.0 utf8
Group {
children [
# Stationary Sun
   Shape { appearance DEF White Appearance {
       material Material { } }
       geometry Sphere { } },
# Several orbiting planets
   DEF Planet1 Transform {
       translation 2.0 0.0 0.0
       center -2.0 0.0 0.0
       children Shape {
           appearance USE White
           geometry Sphere { radius 0.2 } }
       },
   DEF Planet2 Transform {
       translation 3.0 0.0 0.0
       center -3.0 0.0 0.0
       children Shape {
           appearance USE White
           geometry Sphere { radius 0.3 } }
    },
   DEF Planet3 Transform {
       translation 4.0 0.0 0.0
       center -4.0 0.0 0.0
       children Shape {
           appearance USE White
           geometry Sphere { radius 0.5 } }
    },
# Animation clocks, one per planet
   DEF Clock1 TimeSensor {
           cycleInterval 2.0
           loop TRUE },
   DEF Clock2 TimeSensor {
           cycleInterval 3.5
           loop TRUE },
DEF Clock3 TimeSensor {
           cycleInterval 5.0
           loop TRUE },
```

```
# Animation paths, one per planet
        DEF PlanetPath1 OrientationInterpolator {
                key [ 0.0, 0.50, 1.0 ]
                keyValue [ 0.0 0.0 1.0 0.0,
                           0.0 0.0 1.0 3.14,
                           0.0 0.0 1.0 6.28 ] },
        DEF PlanetPath2 OrientationInterpolator {
                key [ 0.0, 0.50, 1.0 ]
                keyValue [ 0.0 0.0 1.0 0.0,
                           0.0 0.0 1.0 3.14,
                           0.0 0.0 1.0 6.28 ] },
        DEF PlanetPath3 OrientationInterpolator {
                key [ 0.0, 0.50, 1.0 ]
                keyValue [ 0.0 0.0 1.0 0.0,
                           0.0 0.0 1.0 3.14,
                           0.0 0.0 1.0 6.28 ] }
        ]
    }
ROUTE Clock1.fraction_changed TO PlanetPath1.set_fraction
ROUTE Clock2.fraction_changed TO PlanetPath2.set_fraction
ROUTE Clock3.fraction_changed TO PlanetPath3.set_fraction
ROUTE PlanetPath1.value_changed TO Planet1.set_rotation
ROUTE PlanetPath2.value_changed TO Planet2.set_rotation
ROUTE PlanetPath3.value_changed TO Planet3.set_rotation
```

see the VRML scene

Virtual Reality Modeling Language

VRML

Sensibilizzare le forme a seguito di azioni del visitatore

Intercettazione delle azioni

- Per rendere il mondo virtuale interattivo, si può attaccare ad una forma un sensore che intercetti azioni dell'utente con dispositivi di puntamento, come ad es. il click del mouse.
- Il `TouchSensor` individua il tocco dell'utente e produce eventi che descrivono quando e dove l'utente ha toccato la forma sensibile.
- Anche i nodi `CylinderSensor`, `PlaneSensor` e `SphereSensor` evidenziano quando un utente tocca la forma sensibile e fornisce un output concepito per agevolare cambi di posizione e di orientamento della forma.

Strumenti di puntamento

- Quando l'utente sposta un dispositivo 2-D o 3-D sullo schermo si osserva il movimento del cursore. Al muoversi del cursore viene generato un raggio immaginario dal cursore al mondo virtuale. Se il raggio incontra una forma, si dice che il raggio **è sopra** la forma. Il punto in cui il raggio incontra la forma è detto hit point. Al muoversi del cursore, l'hit point cambia.
- Molti computer dispongono di puntatori 2-D, come mouse, trackball, trackpad e joystick. I browser VRML li accettano in genere come strumenti di puntamento. Se il dispositivo ha più tasti in genere è il tasto di sinistra ad essere assunto come strumento di puntamento.
- Strumenti di puntamento 3-D sono i guanti digitali o sistemi tipo CAVE. In questo caso il sensore della forma è attivato quando si tocca la forma con il puntatore 3-D

Sensori multipli (i)

- Quando più sensori sono presenti in gruppi diversi, nidificati tra loro, il sensore del nodo interno predomina su quello esterno.
- Per esempio se stiamo rappresentando una lampada, potremmo prevedere un nodo **`PlaneSensor`** associato alla forma della intera lampada, per poter spostare la lampada sul tavolo ed un nodo **`TouchSensor`**, definito in un gruppo più interno con solo la forma dell'interruttore, per rappresentare l'interruttore con il quale accendere e spegnere la lampada.
- Si potrebbe ipotizzare l'instaurarsi di un conflitto visto che sull'interruttore agiscono entrambi i sensori. Solo il nodo più interno intercetterà lo spostamento e l'azione dell'utente, in quanto ha assegnata una maggiore priorità.

Intercettazione delle azioni (i)

- Un nodo **TouchSensor** può essere aggiunto a qualsiasi gruppo, come quelli creati dai nodi **Group** e **Transform**.
- Quando è presente in tali gruppi, il nodo **TouchSensor** intercetta quando un utente tocca o trascina una forma del gruppo.
- Il nodo **TouchSensor** può intercettare un movimento dell'utente. Quando il cursore passa sopra a delle forme sensibili il nodo produce il valore **TRUE** mediante l'**eventOut isOver**. Quando il cursore esce dalle forme sensibili, il nodo produce per lo stesso **eventOut** il valore **FALSE**.
- Si può istradare l'output di **isOver** nell' **eventIn set_enabled** del nodo **TimeSensor** ed attivare e disattivare un'animazione in concomitanza dei movimenti del

Intercettazione delle azioni (ii)

- Il nodo **TouchSensor** può intercettare movimenti dell'utente di click e di trascinamento. Quando il cursore passa sopra a delle forme sensibili e l'utente preme un tasto del mouse, il nodo produce il valore **TRUE** mediante l'**eventOut isActive**. Quando l'utente rilascia il tasto, il nodo produce il valore **FALSE** nello stesso **eventOut** ed il valore del tempo assoluto nell'**eventOut touchTime**.
- Si può istradare l'output di **touchTime** nell' **eventIn set_startTime** del nodo **TimeSensor** per attivare un'animazione quando viene premuto un bottone su una forma sensibile.
- Durante gli spostamenti, i click e i trascinamenti fatti con il mouse, il nodo produce informazioni sulla posizione del mouse che possono essere usate per gestire animazioni

Spostamenti di forme

- In un programma di grafica di tipo visuale, quando l'utente trascina una forma col mouse, questa si sposta.
- Si può emulare questo comportamento in VRML utilizzando i nodi speciali **`CylinderSensor`**, **`PlaneSensor`** e **`SphereSensor`**. Questi nodi si comportano in modo analogo a **`TouchSensor`**, però offrono in più informazioni per muovere e orientare le forme.
- Questi nodi possono essere inseriti in gruppi di forme e rendere sensibili tutte le forme del gruppo. Quando il cursore passa sopra a delle forme sensibili e l'utente preme un tasto del mouse, il nodo produce il valore **`TRUE`** mediante l' **`eventOut`** **`isActive`**. Quando l'utente rilascia il tasto, il nodo produce per lo stesso **`eventOut`** il valore **`FALSE`**.

Spostamenti di forme (i)

- Il nodo **PlaneSensor** intercetta movimenti di trascinamento con il mouse da parte dell'utente, calcola lo spostamento e produce il risultato nell'**eventOut translation_changed**. Istradando questo campo (mediante VRML wiring) sull' **eventIn set_translation** del nodo **Transform**, l'utente può spostare la forma nel mondo virtuale. Il movimento avviene sul piano associato al sensore in corrispondenza dei movimenti del mouse. Per es. può essere usato per spostare una sedia nella stanza o per spostare un quadro sul muro.
- Il nodo **SphereSensor** intercetta movimenti di trascinamento con il mouse da parte dell'utente, calcola asse e angolo di rotazione e produce il risultato nell' **eventOut rotation_changed**. Istradando questo campo sull' **eventIn set_rotation** del nodo **Transform**, l'utente può ruotare la forma nel mondo virtuale muovendo il mouse. Per l'utente la forma è come una sfera che si può ruotare in ogni direzione.

Spostamenti di forme (ii)

- Anche il nodo **CylinderSensor** intercetta movimenti di trascinamento con il mouse da parte dell'utente, calcola asse e angolo di rotazione e produce il risultato nell'**eventOut rotation_changed**. Istradando questo campo sull' **eventIn set_rotation** del nodo **Transform**, l'utente può ruotare la forma nel mondo virtuale. Per l'utente la forma è come un disco o un cilindro che può essere ruotato rispetto ad un asse. Può essere usato per aprire la portiera di una macchina, una porta girevole, per girare una sedia su se stessa.
- I nodi **PlaneSensor** e **CylinderSensor** hanno dei campi che possono limitare l'intervallo del movimento consentito. Nel caso di un nodo **CylinderSensor** si può limitare l'angolo di rotazione della forma tra un minimo ed un massimo. Analogamente per un nodo **PlaneSensor**, si può limitare lo spostamento entro una regione rattangolare definita da un massimo e da un minimo.

Sensori multipli

- E' possibile utilizzare sensori multipli, ciascuno in grado di intercettare azioni dell'utente relativamente ad una data forma. In questo caso i sensori possono sostituirsi uno all'altro o lavorare sinergicamente, a seconda del modo con cui sono stati definiti nel gruppo.
- Quando più sensori sono fratelli nello stesso gruppo, essi intercettano simultaneamente le azioni nello stesso gruppo di forme.
- Per esempio si può usare il nodo **`TouchSensor`** per individuare quando viene spostato il mouse sulla forma ed il nodo **`PlaneSensor`** per consentire all'utente di spostare la forma. Entrambi i sensori possono essere inclusi nello stesso gruppo per intercettare i movimenti dell'utente sulle forme del gruppo.

Sensor offsets

- Quando un utente sposta una forma a cui è associato un nodo **`CylinderSensor`**, **`PlaneSensor`** o **`SphereSensor`** il sensore produce una serie di valori che sono istradati ad un nodo **`Transform`** in modo da spostare la forma. Quando l'utente rilascia il mouse il movimento si ferma. Se l'utente trascina di nuovo la forma, la forma si può spostare da dove era stata lasciata o dalla posizione iniziale.
- La scelta tra i due comportamenti è regolata dal campo **`autoOffset`** del nodo sensore.
- Se **`autoOffset`** è posto **`TRUE`** il nodo sensore ricorda l'ultima posizione della forma e lo spostamento ricomincia da quel punto.
- Sfruttando questo parametro si possono generare comportamenti diversi nelle animazioni.

Il nodo TouchSensor

- Il nodo **TouchSensor** intercetta le azioni dell'utente e produce in uscita informazioni atte a modificare il mondo virtuale a seguito di tali azioni mediante animazioni.

```
TouchSensor {
        enabled             TRUE            # exposedField SFBool
        isActive                            # eventOut     SFBool
        isOver                              # eventOut     SFBool
        touchTime                           # eventOut     SFTime
        hitPoint_changed                    # eventOut     SFVec3f
        hitNormal_changed                   # eventOut     SFVec3f
        hitTexCoord_changed                 # eventOut     SFVec2f
        }
```

Il nodo TouchSensor

- Il valore dell'exposed-field **`enabled`** specifica se il nodo è attivo (**`TRUE`**) e produce output, oppure no (**`FALSE`**). Il default è **`TRUE`**.
- Quando l'utente muove il cursore sopra una forma resa sensibile dal nodo **`TouchSensor`**, il sensore produce in output il valore **`TRUE`** usando l'`eventOut` **`isOver`**. Quando il cursore viene spostato fuori dalla forma, viene prodotto il valore **`FALSE`**.
- Quando il cursore si muove sulla forma, viene prodotto in output il valore del punto di contatto in forma di coordinata 3-D rispetto al sistema di coordinate della forma attraverso l'`eventOut` **`hitPoint_changed`**.

Il nodo TouchSensor (i)

- A seguito della modifica dell'hit point, viene prodotto in output anche il valore del vettore normale e della coordinata di texture, rispettivamente negli `eventOut` **`hitTexCoord_changed`** e **`hitNormal_changed`**.
- Quando l'utente preme il bottone del mouse mentre il cursore è sulla forma sensibile, il nodo sensore invia il valore **`TRUE`** mediante l'`eventOut` **`isActive`**. Quando il bottone è rilasciato viene inviato il valore **`FALSE`** e il tempo assoluto attuale viene trasmesso attraverso l' `eventOut` **`touchTime`**
- Mentre il bottone è premuto il sensore prende assoluto controllo del dispositivo di puntamento e nel mondo non può esserne usato un altro.
- Se il sensore è definito con **`DEF`** e **`USE`**, l'azione del sensore si propaga su ogni forma che lo istanzia

Il nodo PlaneSensor

- Il nodo **PlaneSensor** intercetta le azioni dell'utente e produce in uscita informazioni atte a muovere forme in un piano 2-D.

```
PlaneSensor {
        enabled             TRUE              # exposedField SFBool
        autoOffset          TRUE              # exposedField SFBool
        offset              0.0 0.0 0.0       # exposedField SFVec3f
        maxPosition         -1.0 -1.0         # exposedField SFVec2f
        minPosition         0.0 0.0           # exposedField SFVec2f
        isActive                              # eventOut     SFBool
        translation_changed                   # eventOut     SFVec3f
        trackPoint_changed                    # eventOut     SFVec3f
        }
```

Il nodo PlaneSensor

- Quando il cursore si muove sopra la forma, il punto di hit point identifica un track plane che passa per l'hit point ed è parallelo all'asse XY. L'hit point funge da origine del track plane
- Quando l'utente si muove sulla superficie, la posizione relativa è tracciata dal track point, il quale si sposta nel track plane. Movimenti orizzontali del mouse spostano orizzontalmente il track point, mentre quelli verticali lo spostano verticalmente. Mano mano che l'utente si muove nel piano la coordinata 3-D del track point viene trasmessa attraverso l' `eventOut` **`trackPoint_changed`**
- I campi **`autoOffset`**, **`offset`**, **`maxPosition`** e **`minPosition`** agiscono per calcolare il valore dell' `eventOut` **`translation_changed`** in particolare questi sarà dato dalla somma delle coordinate del dipositivo di puntamento e del campo **`offset`**

Il nodo SphereSensor

- Il nodo **SphereSensor** intercetta le azioni dell'utente e produce in uscita informazioni atte a ruotare forme sferiche.

```
SphereSensor {
        enabled             TRUE                # exposedField SFBool
        autoOffset          TRUE                # exposedField SFBool
        offset              0.0 1.0 0.0 0.0     # exposedField SFRotation
        isActive                                # eventOut     SFBool
        rotation_changed                        # eventOut     SFRotation
        trackPoint_changed                      # eventOut     SFVec3f
        }
```

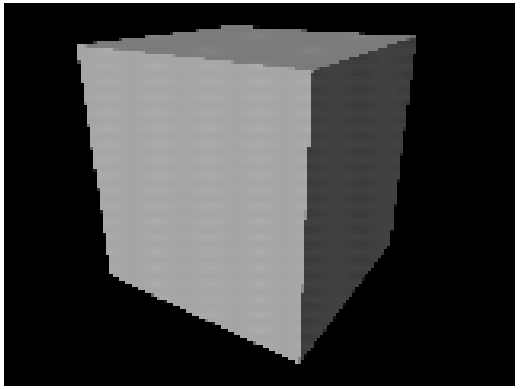
Il nodo CylinderSensor

- Il nodo **`CylinderSensor`** intercetta le azioni dell'utente e produce in uscita informazioni atte a ruotare forme cilindriche rispetto ad un asse.

```
CylinderSensor {
        enabled             TRUE            # exposedField SFBool
        autoOffset          TRUE            # exposedField SFBool
        offset              0.0             # exposedField SFFloat
        diskAngle           0.262           # exposedField SFFloat
        maxAngle            -1.0            # exposedField SFFloat
        minAngle            0.0             # exposedField SFFloat
        isActive                            # eventOut     SFBool
        rotation_changed                    # eventOut     SFRotation
        trackPoint_changed                  # eventOut     SFVec3f
        }
```

Sensori

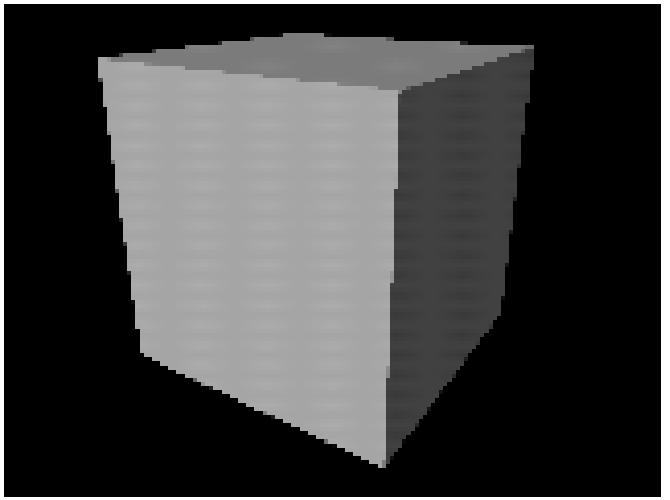
```
#VRML V2.0 utf8
Group {
   children [
# Rotating Cube
        DEF Cube Transform {
            children Shape {
                appearance Appearance {
                    material Material { } }
                geometry Box { } } },
# Sensor
        DEF Touch TouchSensor { },
# Animation clock
        DEF Clock TimeSensor {
                enabled FALSE
                cycleInterval 4.0
                loop TRUE },
# Animation path
        DEF CubePath OrientationInterpolator {
            key [ 0.0, 0.50, 1.0 ]
            keyValue [ 0.0 1.0 0.0 0.0,
                       0.0 1.0 0.0 3.14,
                       0.0 1.0 0.0 6.28 ] }
   ]
}
ROUTE Touch.isOver TO Clock.set_enabled
ROUTE Clock.fraction_changed TO CubePath.set_fraction
ROUTE CubePath.value_changed TO Cube.set_rotation
```



"see the VRML scene

Animazioni con un tocco

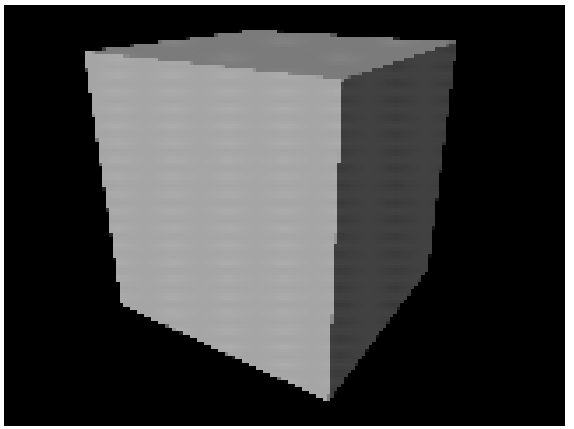
```
#VRML V2.0 utf8
Group {
   children [
# Rotating Cube
        DEF Cube Transform {
            children Shape {
                appearance Appearance {
                    material Material { } }
                geometry Box { } } },
# Sensor
        DEF Touch TouchSensor { },
# Animation clock
        DEF Clock TimeSensor {
                cycleInterval 4.0 },
# Animation path
        DEF CubePath OrientationInterpolator {
            key [ 0.0, 0.50, 1.0 ]
            keyValue [ 0.0 1.0 0.0 0.0,
                       0.0 1.0 0.0 3.14,
                       0.0 1.0 0.0 6.28 ] }
   ]
}
ROUTE Touch.touchTime TO Clock.set_startTime
ROUTE Clock.fraction_changed TO CubePath.set_fraction
ROUTE CubePath.value_changed TO Cube.set_rotation
```



"see the VRML scene

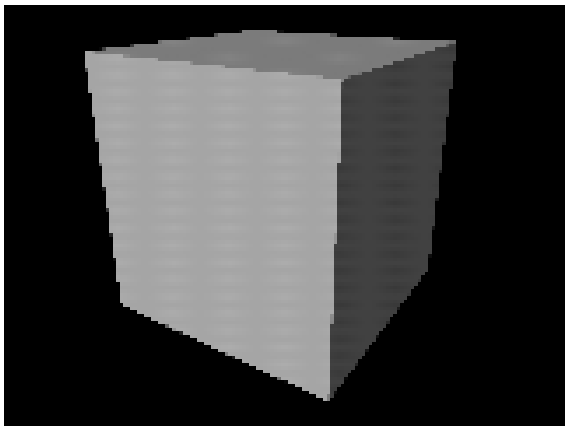
Spostamento di forme nel piano

```
#VRML V2.0 utf8
Group {
   children [
       DEF Cube Transform {
           children Shape {
               appearance Appearance {
                   material Material { } }
               geometry Box { } } },
# Sensor
       DEF Sensor PlaneSensor { },
   ]
}
ROUTE Sensor.translation_changed TO Cube.set_translation
```



"see the VRML scene

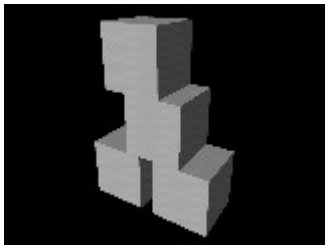
Spostamento di forme nel piano



```
#VRML V2.0 utf8
Group {
   children [
       DEF Cube Transform {
           rotation 1.0 0.0 0.0 -1.57
           children DEF Cube Transform {
               children Shape {
                   appearance Appearance {
                       material Material { } }
                   geometry Box { } } } },
# Sensor
       DEF Sensor PlaneSensor {
               minPosition -2.0 -2.0
               maxPosition  2.0  2.0},
   ]
}
ROUTE Sensor.translation_changed TO Cube.set_translation
```

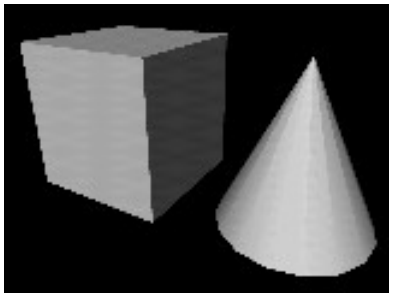
"see the VRML scene

Spostamento di forme nel piano



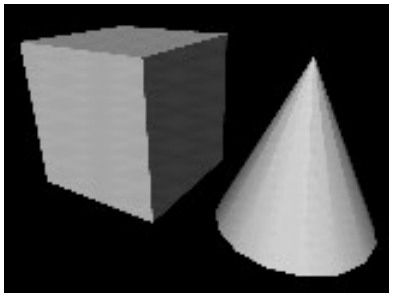
```
#VRML V2.0 utf8
Group {
   children [
     Group {
       children [
          DEF Block1 Transform {
             children DEF BlockShape Shape {
               appearance Appearance { material Material { } }
               geometry Box { } } },
          DEF Block1Sensor PlaneSensor { offset 0.0 0.0 0.0 } ] },
     Group {
        children [
           DEF Block2 Transform { translation 2.5 0.0 0.0
                                  children USE BlockShape },
          DEF Block2Sensor PlaneSensor { offset 2.5 0.0 0.0 } ] },
     Group {
        children [
           DEF Block3 Transform { translation 1.5 2.0 0.0
                                  children USE BlockShape },
          DEF Block3Sensor PlaneSensor { offset 1.5 2.0 0.0 } ] },
     Group {
        children [
           DEF Block4 Transform { translation 0.75 4.0 0.0
                                  children USE BlockShape },
           DEF Block4Sensor PlaneSensor { offset 0.75 4.0 0.0 } ] }
   ]
}
ROUTE Block1Sensor.translation_changed TO Block1.set_translation
ROUTE Block2Sensor.translation_changed TO Block2.set_translation
ROUTE Block3Sensor.translation_changed TO Block3.set_translation
ROUTE Block4Sensor.translation_changed TO Block4.set_translation
```

Rotazione sferica di forme



```
#VRML V2.0 utf8
Group {
   children [
       Group {
           children [
               DEF Shape1 Transform {
                   children Shape {
                       appearance DEF White Appearance {
                               material Material { } }
                        geometry Box { } } },
               DEF Shape1Sensor SphereSensor { }
               ] },
       Group {
           children [
               DEF Shape2 Transform {
                   translation 2.5 0.0 0.0
                   children Shape {
                       appearance USE White
                       geometry Cone { } } },
               DEF Shape2Sensor SphereSensor { }
               ] }
        ]
 }
ROUTE Shape1Sensor.rotation_changed TO Shape1.set_rotation
ROUTE Shape2Sensor.rotation_changed TO Shape2.set_rotation
```

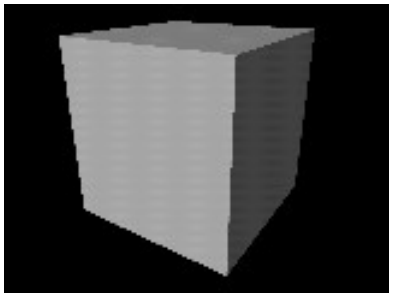
Rotazione cilindrica di forme



```
#VRML V2.0 utf8
Group {
   children [
       Group {
           children [
               DEF Shape1 Transform {
                   children Shape {
                       appearance DEF White Appearance {
                               material Material { } }
                        geometry Box { } } },
               DEF Shape1Sensor CylinderSensor { }
               ] },
       Group {
           children [
               DEF Shape2 Transform {
                   translation 2.5 0.0 0.0
                   children Shape {
                       appearance USE White
                       geometry Cone { } } },
               DEF Shape2Sensor CylinderSensor { }
               ] }
        ]
 }
ROUTE Shape1Sensor.rotation_changed TO Shape1.set_rotation
ROUTE Shape2Sensor.rotation_changed TO Shape2.set_rotation
```

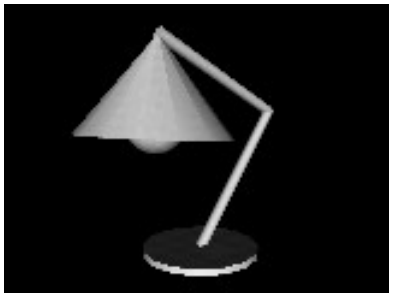
Uso di sensori multipli nello stesso gruppo

```
#VRML V2.0 utf8
Group {
   children [
    # Rotating Cube
    DEF Cube Transform { children Shape {
       appearance Appearance { material Material { } }
       geometry Box { } } },
   # Sensors
   DEF Drag PlaneSensor { },
   DEF Touch TouchSensor { },
   # Animation clock
   DEF Clock TimeSensor {
       enabled FALSE
       cycleInterval 4.0
       loop TRUE },
   # Animation path
   DEF CubePath OrientationInterpolator {
       key [ 0.0, 0.50, 1.0 ]
       keyValue [ 0.0 1.0 0.0 0.0, 0.0 1.0 0.0 3.14, 0.0 1.0 0.0
   6.28 ] }
   ]
 }
ROUTE Touch.isOver TO Clock.set_enabled
ROUTE Clock.fraction_changed TO CubePath.set_fraction
ROUTE CubePath.value_changed TO Cube.set_rotation
ROUTE Drag.translation_changed TO Cube.set_translation
```



"see the VRML scene

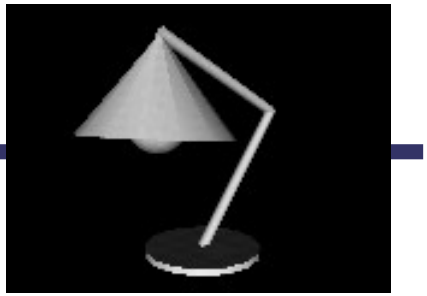
Uso di sensori multipli in gruppi annidati



```
#VRML V2.0 utf8
Group {
   children [
   # Lamp
    DEF MoveLamp PlaneSensor { },
    DEF Lamp Transform {
      children [
         # Lamp base
         Shape {
            appearance DEF White Appearance { material Material { } }
            geometry Cylinder { radius 0.1 height 0.01 } },
         # Base - First arm joint
         Group {
            children [
                 DEF MoveFirstArm SphereSensor { offset 1.0 0.0 0.0 -0.7 },
                 DEF FirstArm Transform { translation 0.0 0.15 0.0
                     rotation 1.0 0.0 0.0 -0.7
                     center 0.0 -0.15 0.0
                     children [
                       # Lower arm
                       DEF LampArm Shape {
                          appearance USE White
                          geometry Cylinder { radius 0.01 height 0.3 } },
                          # First arm - second arm joint
                              Group {
                                children [
                                  DEF MoveSecondArm SphereSensor {
                                        offset 1.0 0.0 0.0 1.9 },
                                  DEF SecondArm Transform {
                                   translation 0.0 0.3 0.0 rotation 1.0 0.0 0.0 1.9
                                   center 0.0 -0.15 0.0
                                  children [
                                     # Second arm
                                     USE LampArm,
```


continua

```
                # Second arm - shade joint
                Group {
                     children [
                          DEF MoveLampShade SphereSensor {
                               offset 1.0 0.0 0.0 -1.25 },
                          DEF LampShade Transform {
                               translation 0.0 0.075 0.0
                               rotation 1.0 0.0 0.0 -1.25
                               center 0.0 0.075 0.0
                               children [
                                    # Shade
                                    Shape { appearance USE White
                                      geometry Cone { height 0.15 bottomRadius 0.12 bottom FALSE} },
                                    # Light bulb Transform {
                                      translation 0.0 -0.05 0.0
                                      children Shape { appearance
                                      USE White geometry Sphere { radius 0.05 } } }
                                              ]
                                            }
                                          ]
                                        }
                                      ]
                                    }
                                  ]
                                }
                              ]
                            }
                          ]
                     }
                ]
           }
      ]
}
ROUTE MoveLamp.translation_changed TO Lamp.set_translation
ROUTE MoveFirstArm.rotation_changed TO FirstArm.set_rotation
ROUTE MoveSecondArm.rotation_changed TO SecondArm.set_rotation
ROUTE MoveLampShade.rotation_changed TO LampShade.set_rotation
```



"see the VRML scene

Virtual Reality Modeling Language

VRML

Controllo dell'Apparenza delle forme

Apparenza delle forme

- Si può controllare l'apparenza di qualsiasi forma specificando gli attributi del materiale con il quale è fatta. Gli attributi comprendono il colore, se la forma riflette e che colore riflette, se è semitrasparente e quanto lo è.
- L'apparenza delle forme viene definita attraverso i nodi `Appearance` e `Material`.
- L'apparenza delle forme è definita da due tipi di proprietà:
  - Quelle che indicano *il colore* del materiale con cui è fatto
  - Quelle che indicano la *variazione di colore* della superficie, o *texture*
- La separazione dell'aspetto dalla geometria della forma consente di creare degli standard, ripetibili nel mondo.

Apparenza dei materiali

- Analogamente anche le proprietà del materiale e quelle di texture sono definiti dai nodi che definiscono i campi `material` e `texture` del nodo `Appearance`.
- Il VRML definisce i colori utilizzando la notazione RGB, cioè indicando con una terna di numeri reali compresi tra 0.0 e 1.0, rispettivamente la quantità di rosso (R), verde (G) e blu (B) che vengono mescolati per dare il colore risultante.

Rosso	Verde	Blu	descrizione
1.0	0.0	0.0	Rosso puro
0.0	1.0	0.0	Verde puro
0.0	0.0	1.0	Blu puro
1.0	1.0	1.0	Bianco
0.0	0.0	0.0	Nero
1.0	1.0	0.0	Giallo
0.0	1.0	1.0	Cyan
1.0	0.0	1.0	Magenta
0.75	0.75	0.75	Grigio chiaro
0.25	0.25	0.25	Grigio scuro

Ombreggiatura

- Le forme vengono rappresentate dal browser VRML mediante la tecnica dell'ombreggiatura, che conferisce loro un aspetto 3-D. Le ombre sono calcolate dal browser considerando una sorgente di luce di default in ogni mondo, denominata *headlight* e posta idealmente sulla testa dell'avatar, la quale assomiglia alla luce che hanno in testa i minatori.
- Al mondo possono essere aggiunte luci addizionali attraverso i nodi `PointLight`, `DirectionalLight` e `SpotLight`.
- Sia che si usino sorgenti di luce addizionali o meno, l'eventuale capacità di trasparenza e l'ombreggiatura delle forme viene controllata mediante il nodo `Material`.
- Però va sottolineato che le forme VRML **non generano ombre** nel mondo virtuale!!!

Effetto risplendente di alcune forme

- Alcune forme (neon, lampadine, schermi di computer) emettono luce propria (glowing effect). In questo caso tale effetto copre l'effetto di ombreggiatura e la forma appare come una superficie illuminata in modo uniforme.
- Attraverso la specifica dell'emissive color nel nodo `Material` si può specificare il colore con cui la superficie si illumina.
- Quando ad esempio si rappresenta una lampadina di forma sferica, quando è spenta viene illuminata dal browser ed appare ombreggiata come una sfera solida.
- Quando la lampada è illuminata emette così tanta luce da apparire una massa circolare di luce.

Animazione delle proprietà dei materiali

- Analogamente a quanto visto per la traslazione e la rotazione, VRML fornisce due nodi di interpolazione per animare colore e trasparenza delle forme: **`ColorInterpolator`** e **`ScalarInterpolator`** .
- **`ColorInterpolator`** utilizza una lista dei tempi frazionari e dei corrispondenti colori nei campi **`key`** e **`keyvalue`**. Quando un nodo **`TimeSensor`** invia un tempo frazionario, il nodo calcola il colore attuale mediante interpolazione lineare dei dati forniti. Il risultato è trasferito mediante l' **`eventOut`** **`value_changed`**.
- Si può collegare questo valore con i nodi che definiscono le proprietà di colore di una forma per animarle, mediante gli opportuni campi del nodo **`Material`**.
- Analogamente, il nodo **`ScalarInterpolator`** consente di animare le proprietà di trasparenza di una forma istradando l'output della interpolazione al campo **`transparency`**

Colori HSV

- Il nodo **`ColorInterpolator`** interpola tra due colori per calcolare un colore intermedio che fornisce in output usando l'**`eventOut` `value_changed`**.
- Il nodo dovrebbe essere disegnato per interpolare un colore tra i valori di rosso verde e blu, in modo analogo a quanto avviene nel caso delle coordinate X, Y e Z nel caso del nodo **`PositionInterpolator`**, ma non è così: in questo modo infatti vengono prodotti colori diversi da quelli desiderati.
- Il nodo **`ColorInterpolator`** trasforma il colore RGB in una nuova forma di specifica, denominata colore **HSV** (*hue*, *saturation*, *value*). L'interpolazione del colore HSV produce un risultato molto più vicino a quello desiderato. Una volta calcolato, il colore HSV viene riconvertito in colore RGB.
- La conversione automatica indica che l'utente non deve lavorare direttamente con i colori HSV.

Colori HSV (i)

- Hue è un numero tra `0.0` e `1.0` che identifica un colore nell'asse rosso-verde-blu-rosso. E' la proprietà che noi identifichiamo con il colore: una mela rossa ha un hue rosso.
- Saturation è un numero tra `0.0` e `1.0` che identifica quanto bianco deve essere mescolato insieme al colore. Il valore `0.0` indica che deve essere mescolata una grande quantità di bianco. Incrementando il valore di saturation del colore, questi assume una colorazione di tipo pastello: un colore rosso con un valore di saturation di `0.75` viene trasformato in rosa.
- Value è un numero tra `0.0` e `1.0` che indica il grado di brillantezza del colore: il valore `0.0` indica che il colore è assente e produce nero, il valore `1.0` indica la massima brillantezza del colore.

Esempio di interpolazione RGB

Tempo fraz.	Chiave RGB	RGB calc.	RGB risult.	Descrizione
0.00	1.00 0.00 0.00	1.00 0.00 0.00	1.00 0.00 0.00	Rosso brill.
0.25		0.75 0.00 0.25	0.75 0.00 0.25	Rosso-blu scuro
0.50		0.50 0.00 0.50	0.50 0.00 0.50	Grigio medio
0.75		0.25 0.00 0.75	0.25 0.00 0.75	Blu-rosso scuro
1.00	0.00 0.00 1.00	0.00 0.00 1.00	0.00 0.00 1.00	Blu brill.

Esempio di interpolazione HSV

Tempo fraz.	Chiave RGB	HSV calc.	RGB risult.	Descrizione
0.00	1.00 0.00 0.00	1.00 0.00 0.00	1.00 0.00 0.00	Rosso brill.
0.25		0.92 0.00 0.00	1.00 0.00 0.50	Rosso-blu brill.
0.50		0.83 0.00 0.00	1.00 0.00 1.00	porpora brill.
0.75		0.75 0.00 0.00	0.50 0.00 1.00	Blu-rosso brill.
1.00	0.00 0.00 1.00	0.67 0.00 0.00	0.00 0.00 1.00	Blu brill.

Nodo Shape

- Come visto, tutte le forme in VRML vengono rappresentate attraverso il nodo **Shape**

```
Shape {
        appearance   NULL      # exposedField      SFNode
        geometry     NULL      # exposedField      SFNode
      }
```

- Abbiamo già discusso il campo **geometry** e presentato le diverse forme predefinite
- Il valore del campo **appearance** specifica un nodo che definisce l'apparenza della forma, compreso il colore e la texture.

Nodo Shape (i)

- Un tipico valore del campo **appearance** comprende il nodo **Appearance**
- L'apparenza e la geometria di una forma possono essere modificate inviando valori agli **eventIn** impliciti **set_appearance** e **set_geometry** degli exposed-field **appearance** e **geometry**.
- Una volta ricevuti i valori, i valori dei campi vengono modificatie ed i nuovi valori vengono propagati attraverso gli **eventOut** impliciti **appearance_changed** e **geometry_changed** degli exposed-field sopracitati.

Il nodo Appearance

- Il nodo **Appearance** specifica gli attributi di apparenza di una forma e può essere specificato come valore del campo **appearance** del nodo **Shape**.

```
Appearance {
        material            NULL            # exposedField SFNode
        texture             NULL            # exposedField SFNode
        textureTransform    NULL            # exposedField SFNode

                    }
```

- Vedremo in seguito i campi **texture** e **textureTransform**

Il nodo Appearance

- Il valore del campo **material** specifica un nodo (tipicamente **Material**) che definisce gli attributi materiali dell'apparenza della forma.
- L'apparenza della forma può essere cambiata inviando un valore al campo **eventIn** implicito **set_material** dell'exposed-field **material**.
- Ricevuto il valore in input, viene cambiato il corrispondente valore del campo e il nuovo valore propagato attraverso l' **eventOut** implicito **material_changed** dell'exposed-field **material**.

Il nodo Material

- Il nodo **Material** specifica gli attributi di apparenza di una forma e può essere specificato come valore del campo **material** del nodo **Appearance**.

```
Material {
        diffusecolor        0.8 0.8 0.8   # exposedField SFColor
        emissivecolor       0.0 0.0 0.0   # exposedField SFColor
        transparency        0.0           # exposedField SFFloat
        ambientIntensity    0.2           # exposedField SFFloat
        specularColor       0.0 0.0 0.0   # exposedField SFColor
        shininess           0.2           # exposedField SFFloat
                    }
```

Il nodo Material (i)

- Il valore dell'exposed-field `diffuseColor` indica il colore RGB della forma.
- Il valore dell'exposed-field `emissiveColor` indica il colore RGB emesso dalla forma. Il default è il colore nero, che indica materiali che non emettono luce propria.
- Forme che hanno codificato `emissiveColor` non illuminano le altre forme della scena. Se si disattivasse l'headlight di default del browser VRML, queste superfici sarebbero ancora visibili, mentre le altre sarebbero nere e invisibili

Il nodo Material (ii)

- Il valore dell'exposed-field **`transparency`** indica il fattore di trasparenza (`0.0` indica una superficie opaca, `1.0` una superficie completamente trasparente). Il valore di default è `0.0`.
- La trasparenza viene resa dai browser VRML con tecniche diverse.
  - Alcuni browser utilizzano una tecnica denominata screen door effect che pone davanti alle superfici trasparenti una specie di rete, denominata dither pattern, più o meno fitta a seconda del valore del campo **`transparency`**. Per un alto valore di trasparenza viene realizzata una maglia attraverso la quale si vede molto.
  - Altri browser usano la tecnica denominata color-blending effect, che calcolano il modo con cui il colore si trasforma all'aumentare della trasparenza. Questo effetto è più realistico del precedente.

Il nodo Material (iii)

- Il colore e la trasparenza della forma possono essere cambiati inviando valori agli **`eventIn`** impliciti **`set_diffuseColor set_emissiveColor set_transparency`** degli exposed-field **`diffuseColor emissiveColor`** e **`transparency`**.
- Ricevuto il valore in input, viene cambiato il corrispondente valore del campo e il nuovo valore propagato attraverso gli **`eventOut`** impliciti **`diffuseColor_changed emissiveColor_changed transparency_changed`**.
- I campi **`ambientIntensity specularColor`** e **`shininess`** verranno discussi più avanti.

Il nodo ColorInterpolator

- Il nodo **`ColorInterpolator`** descrive una serie di colori chiave che possono essere utilizzati per l'animazione del colore di una forma.

```
ColorInterpolator {
        key             [ ]         # exposedField MFFloat
        keyValue        [ ]         # exposedField MFColor
        set_fraction                # eventIn      SFFloat
        value_changed               # eventOut     SFFloat
                                }
```

Il nodo ColorInterpolator

- Il valore dell'exposed-field `key` specifica una lista di frazioni di tempo. In genere questi tempi sono compresi tra `0.0` e `1.0`, come quelli emessi dall' `eventOut fraction_changed` di un nodo `TimeSensor`. Le frazioni di tempo possono essere comunque valori positivi o negativi di ogni grandezza. I tempi devono essere elencati in ordine non decrescente. Il valore di default è una lista vuota.
- Il valore dell'exposed-field `keyValue` specifica una lista di colori chiave RGB. Ogni colore chiave RGB è una terna di valori compresi tra `0.0` e `1.0`. Il valore di default è una lista vuota.

Il nodo ColorInterpolator (i)

- I tempi e i colori sono utilizzati congiuntamente, in modo che il primo tempo indichi il primo colore chiave, il secondo il successivo colore, e così via. Le liste possono indicare qualsiasi numero di colori, ma entrambe devono avere lo stesso numero di valori.
- Quando un nodo **`ColorInterpolator`** riceve un valore di frazione di tempo tramite l' **`eventIn set_fraction`**, esso calcola il nuovo colore sulla base dei valori presenti nella lista dei colori chiave e delle relative frazioni di tempo. Il nuovo colore calcolato viene propagato attraverso l'**`eventOut value_changed`**. Il nuovo colore viene calcolato mediante interpolazione lineare dei colori HSV corrispondenti ai due colori RGB le cui frazioni di tempo associate contengono la frazione di tempo ricevuta.

Il nodo ColorInterpolator (ii)

- In genere le frazioni di tempo sono inviate attraverso l'**eventIn** **set_fraction** creando una route con l'**eventOut** **fraction_changed** di un nodo **TimeSensor**. Frazioni di tempo possono pervenire anche da nodi che generino valori reali generici, come il nodo **ScalarInterpolator**
- La lista delle posizioni e dei tempi possono essere cambiate inviando dei valori agli **eventIn** impliciti **set_key** e **set_keyValue**, ed i nuovi valori propagati attraverso gli **eventOut** **key_changed** e **keyValue_changed**.
- Il nodo **ColorInterpolator** non produce forme e non ha effetti visibili nel mondo. Un nodo **ColorInterpolator** può essere inserito in ogni nodo che raggruppa oggetti, ma è indipendente dal sistema di coordinate. In genere un nodo **ColorInterpolator** viene posto alla fine del gruppo più esterno di un file VRML.

Il nodo ScalarInterpolator

- Il nodo **ScalarInterpolator** descrive una serie di valori chiave che possono essere utilizzati per l'animazione della trasparenza di una forma.

```
ScalarInterpolator {
        key                 [ ]             # exposedField MFFloat
        keyValue            [ ]             # exposedField MFFloat
        set_fraction                        # eventIn      SFFloat
        value_changed                       # eventOut     SFFloat
                                }
```

Il nodo ScalarInterpolator (i)

- Il nodo è costituito da una lista di valori reali contenuti nel campo **key** e da una lista di valori reali chiave corrispondenti contenuti in **keyValue**; entrambe le liste devono avere lo stesso numero di elementi.
- Quando un nodo **ScalarInterpolator** riceve un valore di frazione di tempo tramite l' **eventIn set_fraction**, esso calcola il nuovo colore sulla base dei valori presenti nella lista dei valori reali chiave e delle relative frazioni di tempo. Il nuovo valore reale calcolato viene propagato attraverso l'**eventOut value_changed**. Il nuovo valore reale viene calcolato mediante interpolazione lineare dei valori reali le cui frazioni di numeri reali associate contengono la frazione di numero reale ricevuto.

Il nodo ScalarInterpolator (ii)

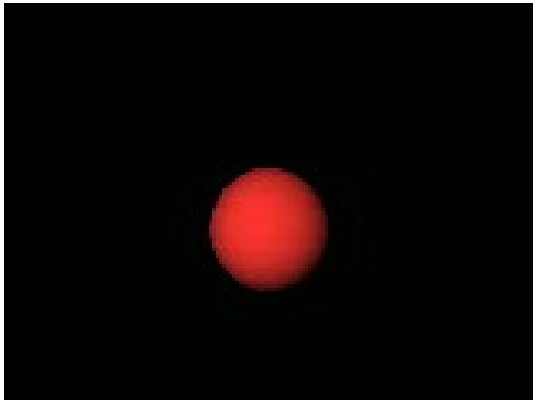
- In genere le frazioni di numero reale sono inviate attraverso l'`eventOut` **`set_fraction`**.
- La lista dei numeri frazione e dei numeri reali chiave possono essere cambiate inviando dei valori agli `eventIn` impliciti **`set_key`** e **`set_keyValue`**, ed i nuovi valori propagati attraverso gli `eventOut` **`key_changed`** e **`keyValue_changed`**.
- Il nodo **`ScalarInterpolator`** non produce forme e non ha effetti visibili nel mondo. Un nodo **`ScalarInterpolator`** può essere inserito in ogni nodo che raggruppa oggetti, ma è indipendente dal sistema di coordinate. In genere un nodo **`ScalarInterpolator`** viene posto alla fine del gruppo più esterno di un file VRML.

Il nodo ScalarInterpolator (ii)

- Questo nodo, differisce dai nodi di interpolazione visti finora, in quanto è un nodo che genera numeri reali di qualsiasi tipo e in qualsiasi ordine, per cui agisce sulle animazioni in modo del tutto differente.
- Anche questo nodo usa una interpolazione lineare tra coppie di valori che comprendono il tempo frazionario fornito in input per calcolare la grandezza da interpolare.
- Un uso interessante di questo nodo di interpolazione è quello di affiancarlo ad un nodo `TimeSensor` per alterarne il funzionamento (ad es: il nodo `TimeSensor` genera numeri casuali nell'intervallo `0.0-1.0`: affiancando il nodo `ScalarInterpolator` si può ottenere una variazione tra `1.0` e `0.0` (basta usare `key = [0 1]` e `keyValue = [1 0]`).

Uso dei nodi Appearance e Material

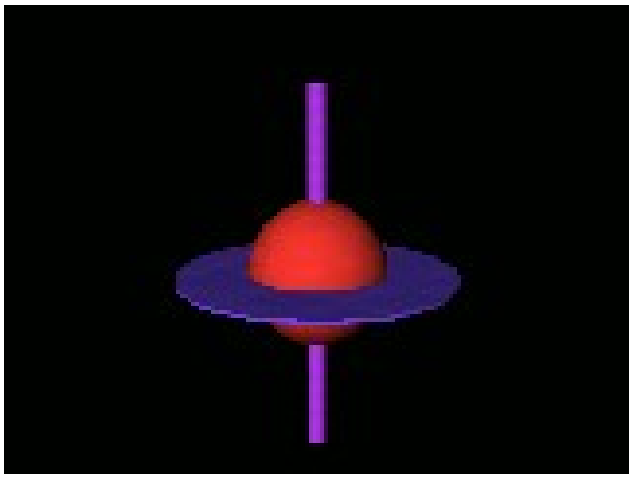
```
#VRML V2.0 utf8
     Shape {
             appearance Appearance {
                             material Material
                                     { diffuseColor
   1.0 0.0 0.0 }
     }
             geometry Sphere { }
 }
```



"see the VRML scene

Uso dei nodi Appearance e Material (i)

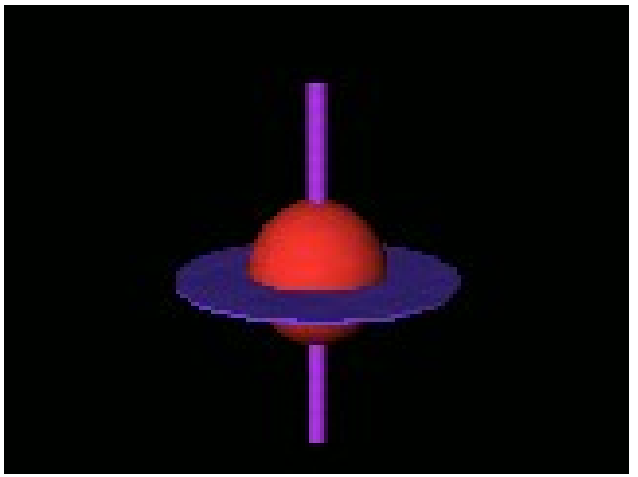
```
#VRML V2.0 utf8
Group {
   children [
     # Station Shapes
      Shape { appearance Appearance {
                    material Material
   { diffuseColor 1.0 0.0 0.0 } }
                  geometry Sphere { } },
      Shape { appearance Appearance {
                    material Material
   { diffuseColor 0.5 0.25 1.0 } }
   height 0.05 } }, geometry Cylinder { radius 2.0
      Shape { appearance Appearance {
                    material Material
   { diffuseColor 0.75 0.0 1.0 } }
   height 5.0 } }     geometry Cylinder { radius 0.15
                  ]
            }
```



"see the VRML scene

Uso dei nodi Appearance e Material (ii)

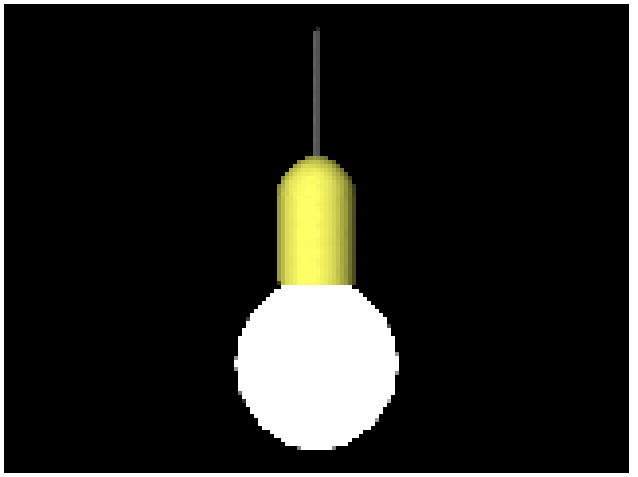
```
#VRML V2.0 utf8
Group {
   children [
     # Station Shapes
      Shape {
           appearance Appearance {
                  material DEF BallColor  Material {
                             diffuseColor 1.0 0.0 0.0
                              } }
                    geometry Sphere { } },
      Shape {
           appearance Appearance {
                  material Material {
                             diffuseColor 0.5 0.25 1.0
                             } }
                    geometry Cylinder { radius 2.0 height 0.05
   } },
      Shape {
            appearance Appearance {
                   material Material {
                               diffuseColor 0.75 0.0 1.0
                              } }
                     geometry Cylinder { radius 0.15 height
   5.0 }},
       # Animation clock
       DEF Clock TimeSensor {
                  cycleInterval 4.0 loop TRUE },
        # Animation path
        DEF ColorPath ColorInterpolator {
                  key [ 0.0, 0.33, 0.67, 1.0 ]
                  keyValue [ 1.0 0.0 0.0, 0.0 1.0 0.0, 0.0
0.0 1.0, 1.0 0.0 0.0, ]
       ]
   }
```



"see the VRML scene

Forme che emettono luce

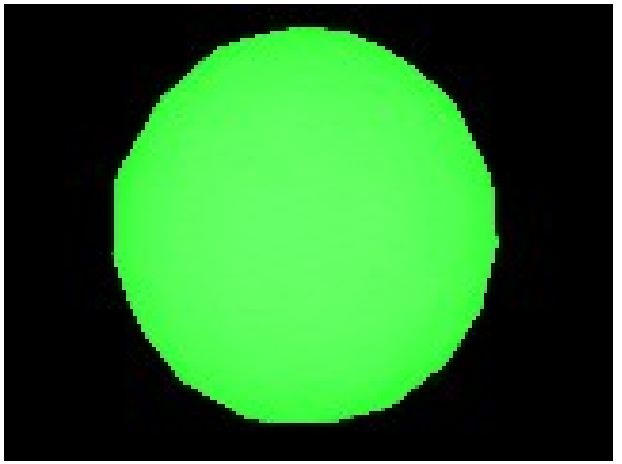
```
#VRML V2.0 utf8
Group {
    children [
    # Dark gray light bulb hanging wire
        Shape {
            appearance Appearance {
   0.4 } }          material Material { diffuseColor 0.4 0.4
            geometry Cylinder { radius 0.05 height 2.0 } },
    # Yellowish light bulb socket
        Transform {
             translation 0.0 -1.0 0.0
                 children Shape {
                          appearance Appearance {
   { diffuseColor 1.0 1.0 0.4 } }    material Material
                          geometry Sphere { radius 0.5 }
         } },
        Transform {
               translation 0.0 -1.5 0.0
                   children Shape {
                           appearance Appearance {
   { diffuseColor 1.0 1.0 0.4 } } material Material
   height 1.0 } } },       geometry Cylinder { radius 0.5
         # White light bulb
         Transform {
                translation 0.0 -2.95 0.0
                     children Shape {
                              appearance Appearance {
   { diffuseColor 1.0 1.0 1.0      material Material
          emissiveColor 1.0 1.0 1.0 } }
                              geometry Sphere { }} }
           ]
}
```



"see the VRML scene

Forme lampeggianti

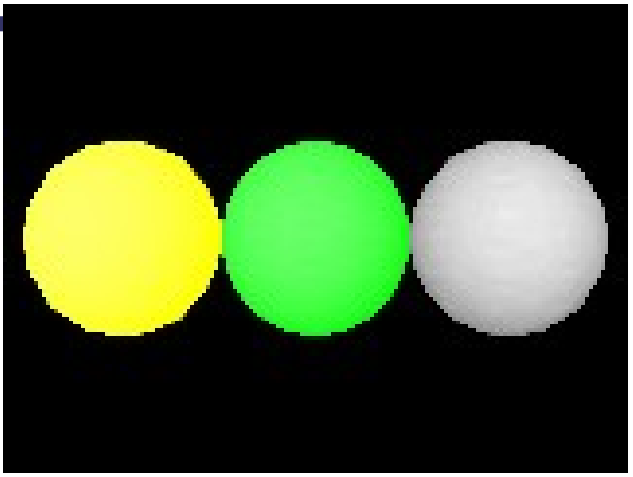
```
#VRML V2.0 utf8
Group {
     children [
     # Blinking ball
         Shape {
              appearance Appearance {
                   material DEF BallColor Material {
diffuseColor 0.4 0.4 0.4 } }
              geometry Sphere { } },
     # Animation clock DEF Clock TimeSensor {
                  cycleInterval 1.0 loop TRUE },
     # Animation path DEF ColorPath ColorInterpolator {
                  key [ 0.0, 0.5, 0.5, 1.0 ]
                  keyValue [ 0.0 1.0 0.0, 0.0 1.0 0.0, 0.0
0.0 1.0, 0.0 0.0 1.0, ] }
     ]
}
ROUTE Clock.fraction_changed TO ColorPath.set_fraction
ROUTE ColorPath.value_changed TO BallColor.set_emissiveColor
```



"see the VRML scene

Forme lampeggianti (i)

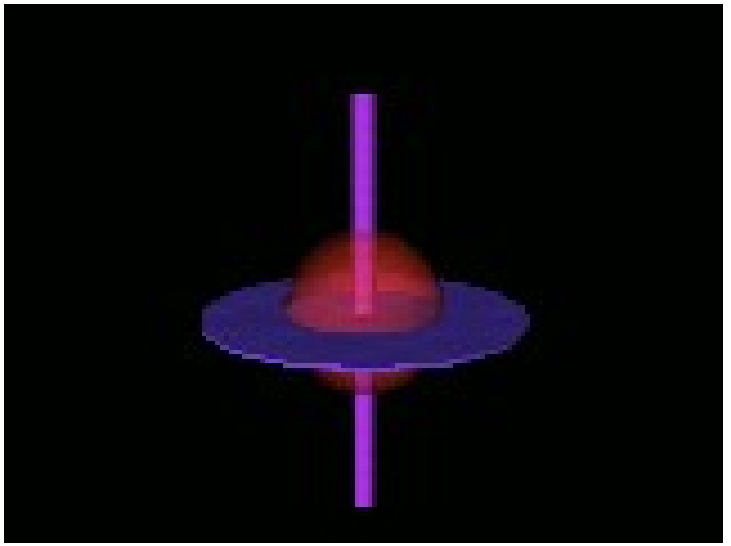
```
#VRML V2.0 utf8
Group {
    children [
    # Blinking balls
        Transform {
            translation -2.0 0.0 0.0
            children Shape {
                appearance Appearance {
                    material DEF Ball1Color Material
                        {diffuseColor 0.4 0.4 0.4 } }
                geometry DEF ABall Sphere { } } },
        Shape {
            appearance Appearance {
                material DEF Ball2Color Material
                    { diffuseColor 0.4 0.4 0.4 } }
            geometry USE ABall },
        Transform {
            translation 2.0 0.0 0.0
            children Shape {
                appearance Appearance {
                    material DEF Ball3Color Material
                    {diffuseColor 0.4 0.4 0.4 } }
                geometry USE ABall } },
        # Animation clocks
        DEF Clock1 TimeSensor { cycleInterval 1.0 loop TRUE },
        DEF Clock2 TimeSensor { cycleInterval 1.3 loop TRUE },
        DEF Clock3 TimeSensor { cycleInterval 0.7 loop TRUE },
        # Animation paths
        DEF Color1Path ColorInterpolator {
            key [ 0.0, 0.5, 0.5, 1.0 ]
            keyValue [ 1.0 1.0 0.0, 1.0 1.0 0.0, 0.0 1.0 0.0, 0.0 1.0 0.0, ] },
        DEF Color2Path ColorInterpolator {
            key [ 0.0, 0.5, 0.5, 1.0 ]
            keyValue [ 0.0 1.0 0.0, 0.0 1.0 0.0, 0.0 0.0 1.0, 0.0 0.0 1.0, ] },
        DEF Color3Path ColorInterpolator {
            key [ 0.0, 0.5, 0.5, 1.0 ]
            keyValue [ 0.5 0.5 0.5, 0.5 0.5 0.5, 1.0 1.0 1.0, 1.0 1.0 1.0, ] }
    ]
}
ROUTE Clock1.fraction_changed TO Color1Path.set_fraction
ROUTE Clock2.fraction_changed TO Color2Path.set_fraction
ROUTE Clock3.fraction_changed TO Color3Path.set_fraction
ROUTE Color1Path.value_changed TO Ball1Color.set_emissiveColor
ROUTE Color2Path.value_changed TO Ball2Color.set_emissiveColor
ROUTE Color3Path.value_changed TO Ball3Color.set_emissiveColor
```



"see the VRML scene

Transparenza

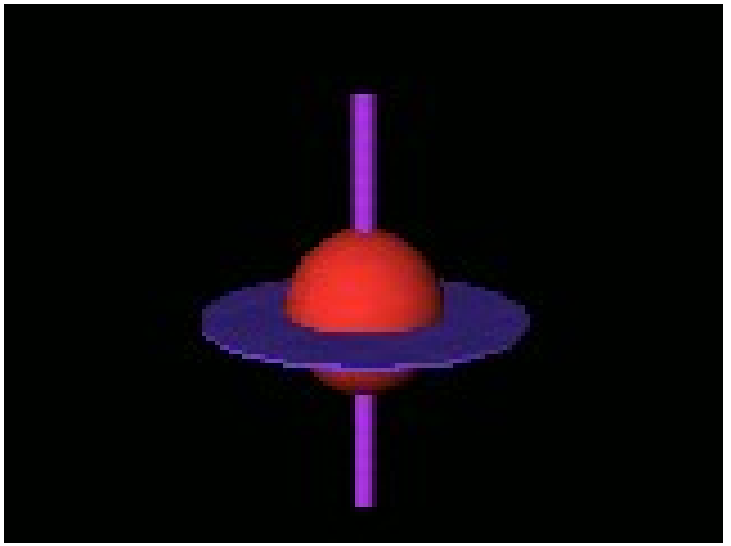
```
#VRML V2.0 utf8
Group {
   children [
     # Station Shapes
      Shape {
           appearance Appearance {
                  material Material {
                              diffuseColor 1.0 0.0 0.0
               transparency 0.5
                              } }
                    geometry Sphere { } },
      Shape {
           appearance Appearance {
                  material Material {
                              diffuseColor 0.5 0.25 1.0
                              } }
                    geometry Cylinder { radius 2.0 height 0.05 } },
      Shape {
            appearance Appearance {
                    material Material {
                                diffuseColor 0.75 0.0 1.0
                               } }
                     geometry Cylinder { radius 0.15 height 5.0 }}
        ]
   }
```



"see the VRML scene

Transparenza (i)

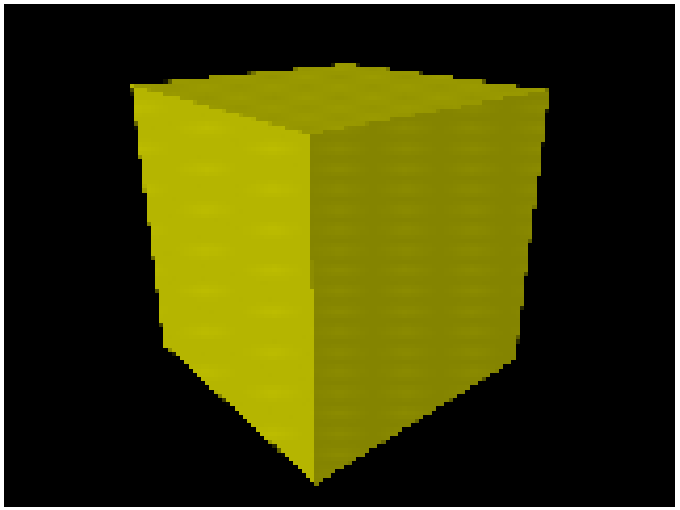
```
#VRML V2.0 utf8
Group {
   children [
     # Station Shapes
      Shape {
           appearance Appearance {
                  material DEF BallColor Material {
                             diffuseColor 1.0 0.0 0.0
                   } }
                    geometry Sphere { } },
      Shape {
           appearance Appearance {
                  material Material {
                              diffuseColor 0.5 0.25 1.0
                              } }
                    geometry Cylinder { radius 2.0 height 0.05 } },
      Shape {
            appearance Appearance {
                    material Material {
                                diffuseColor 0.75 0.0 1.0
                               } }
                     geometry Cylinder { radius 0.15 height 5.0 }},
       # Animation clock
       DEF Clock TimeSensor { cycleInterval 4.0 loop TRUE },
       # Animation path
       DEF TransparencyPath ScalarInterpolator {
              key [ 0.0, 0.5, 1.0 ]
              keyValue [ 0.0, 1.0, 0.0 ] }
        ]
   }
ROUTE Clock.fraction_changed TO TransparencyPath.set_fraction
ROUTE TransparencyPath.value_changed TO BallColor.set_transparency
```



"see the VRML scene

Animazione con ScalarInterpolator

```
#VRML V2.0 utf8
Group {
    children [
        # Moving box
        DEF Cube Transform {
               children Shape {
                     appearance Appearance {
                            material Material
                             { diffuseColor 1.0 1.0 0.0 } }
                     geometry Box { size 1.0 1.0 1.0 } } },
        # Animation clock
        DEF Clock TimeSensor {
                cycleInterval 4.0 loop TRUE },
        # Animation controller
        DEF CubeController ScalarInterpolator { key [ 0.0, 0.5, 1.0 ]
                        keyValue [ 0.0, 1.0, 0.0 ] },
        # Animation path
        DEF CubePath PositionInterpolator {
               key [ 0.00, 0.11, 0.17,
                        0.22, 0.33, 0.44,
                        0.50, 0.55, 0.66,
                        0.77, 0.83, 0.88,
                        0.99 ]
                keyValue [ 0.0 0.0 0.0, 1.0 1.96 1.0,
                                1.5 2.21 1.5, 2.0 1.96 2.0,
                                3.0 0.0 3.0, 2.0 1.96 3.0,
                                1.5 2.21 3.0, 1.0 1.96 3.0,
                                0.0 0.0 3.0, 0.0 1.96 2.0,
                                0.0 2.21 1.5, 0.0 1.96 1.0,
                                0.0 0.0 0.0 ] }
       ]
}
ROUTE Clock.fraction_changed TO CubeController.set_fraction
ROUTE CubeController.value_changed TO CubePath.set_fraction
ROUTE CubePath.value_changed TO Cube.set_translation
```



"see the VRML scene

Virtual Reality Modeling Language

VRML

Come raggruppare nodi

Raggruppare nodi

- Si può raggruppare un numero qualsiasi di nodi e poi manipolare il gruppo come un insieme. Per esempio, si possono raggruppare forme per rappresentare un edificio mediante un nodo `Transform` che poi può essere traslato, ruotato, scalato per inserirlo come unica entità in un blocco di costruzioni.
- Il raggruppamento più semplice si ottiene con il nodo `Group`.
- Il nodo `Switch` estende le capacità di base del nodo `Group` considerando il gruppo di `children` come una lista di possibili scelte tra cui selezionare la forma da rappresentare.
- Utilizzando il campo `wichChoice` del nodo si può selezionare quale forma rappresentare.

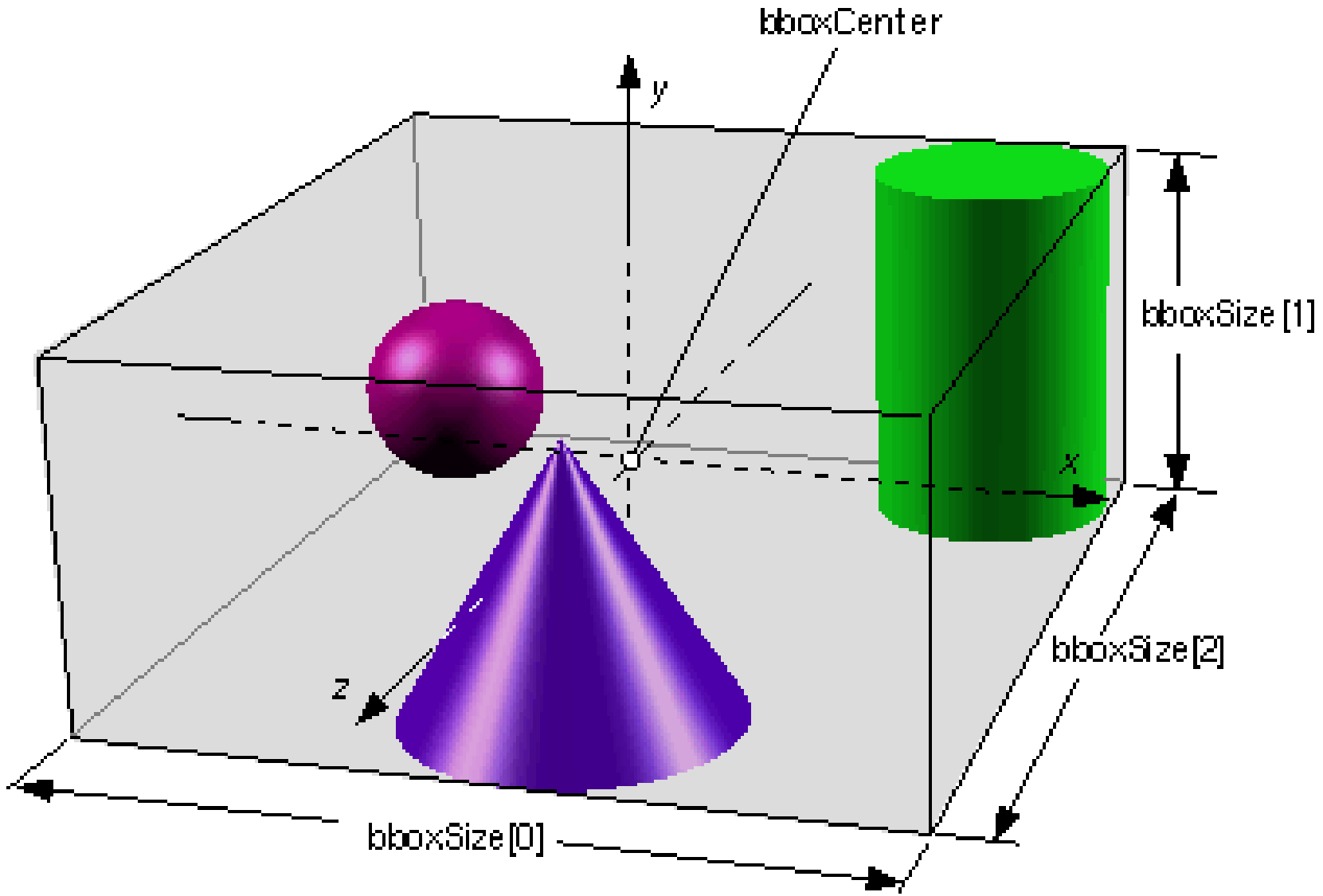
Raggruppare nodi (i)

- Il nodo `Transform` estende anch'esso le capacità di base del nodo `Group` , creando un nuovo sistema di coordinate per il gruppo. Il nuovo sistema di coordinate può essere traslato, ruotato e scalato, in modo che il gruppo di oggetti può essere posto in qualsiasi posizione del mondo virtuale, orientato e ridimensionato a seconda dell'uso.
- Il nodo `Billboard` invece crea uno speciale sistema di coordinate (billboard coordinate) che ruota sempre verso l'osservatore, anche se l'osservatore si muove intorno al gruppo. Un nodo `Billboard` può essere usato per una serie di applicazioni: per metter avvisi, informazioni sullo stato del sistema, pannelli di controllo, etc.
- Si può specificare l'asse di rotazione del gruppo `Billboard`, gli estremi di rotazione, etc. Si può impostare il gruppo anche in modo che segua sempre e comunque il movimento dell'osservatore.

Scelta del campo visivo

- Per velocizzare la rappresentazione del mondo, VRML analizza la scena e calcola quali forme sono visibili rispetto all'osservatore e quali sono nascoste (e pertanto da non rappresentare). La suddivisione tra forme visibili e forme non visibili è detta visibility culling: tanto più rapida è tale operazione, tanto più veloce è il browser VRML.
- Per velocizzare il processo di visibility culling il browser costruisce una serie di scatole invisibili denominate bounding boxes, intorno a ogni gruppo di forme raggruppate dai nodi `Group`, `Billboard` e `Transform`.
- La definizione del bounding box serve ad accelerare certe operazioni del browser, come quella di decidere se il gruppo è rappresentabile oppure no.

Bounding Boxes dei nodi di raggruppamento



Bounding box

- Queste scatole devono essere grandi abbastanza da contenere le forme contenute nel gruppo. Invece di verificare la visibilità delle singole forme, il browser verifica prima quella del bounding box.
- Se il bounding boxes non è visibile, non lo sono neanche le forme in esso contenute, pertanto non si procede al test delle singole forme.
- Se il bounding box è visibile, allora il browser verifica se sono visibili le singole forme.
- Il valore di default (`-1 -1 -1`) indica che non viene definito un bounding box, pertanto sarà calcolato dal browser.
- Il volore (`0 0 0`) indica un punto dello spazio ed è un valore ammesso.
- Se il valore specificato non è (`>0 >0 >0`) o uno dei due valori mostrati sopra, il comportamento è indefinito. Lo stesso accade se il bounding box specificato è più piccolo del dovuto.
- Il bounding box può variare dinamicamente se vengono usati gli eventIn **`addChildren`** e **`removeChildren`**

Il nodo Group

- Il nodo **Group** raggruppa nodi VRML.

```
Group {
        children          [ ]             # exposedField MFNode
        bboxSize          -1.0 -1.0 -1.0  # field        SFVec3f
        bboxCenter        0.0 0.0 0.0     # field        SFVec3f
        addChildren                       # eventIn      MFNode
        removeChildren                    # eventIn      MFNode
      }
```

Il nodo Group

- Il valore dell'exposed-field `children` specifica una lista di nodi da includere nel gruppo. Un valore tipico di `children` è rappresentato dal nodo `Shape` o da un altro nodo di raggruppamento.
- Il valore di `bboxSize` specifica le dimensioni di un bounding box sufficientemente grande da contenere le forme del gruppo. I valori indicano le dimensioni nelle direzioni X, Y e Z rispettivamente.
- Il valore di `bboxCenter` indica il centro del bounding box. Il valore del campo è una coordinata 3-D nel sistema di coordinate del gruppo che consente di spostare il centro del bounding box rispetto all'origine del sistema di coordinate, che è rappresentato dal valore di default. Quando il browser calcola le dimensioni della scatola, calcola anche le coordinate del centro.
- Il bounding box deve essere l'unione dei bounding box relativi alle forme presenti in `children`. Non comprende trasformazioni operate dal gruppo in quanto definito nel sistema di coordinate locale

Il nodo Group (i)

- La lista di `children` nel gruppo può essere modificata utilizzando gli **`eventIn`** `addChildren` e `removeChildren`, rimpiazzata dalla nuova lista.
- Quando una lista di nodi viene inviata all'**`eventIn`** `addChildren`, la lista dei nodi nel campo `children` viene aggiornata di conseguenza ed i nodi della lista vengono aggiunti a quelli contenuti nel campo `children`. Se un nodo è già presente non viene fatto l'aggiornamento.
- Quando una lista di nodi viene inviata all'**`eventIn`** `removeChildren`, i nodi della lista vengono confrontati con quelli presenti nel campo `children`, e se vengono trovati vengono cancellati dal campo `children`.
- In tutti i casi i nuovi valori vengono propagati attraverso l'**`eventOut`** `children_changed` dell'exposed-field `children`.

Il nodo Switch

- Il nodo **Switch** raggruppa nodi VRML.

```
Switch {
        choice              [ ]             # exposedField MFNode
        whichChoice         -1              # exposedField SFInt32
                    }
```

Il nodo Switch

- Il valore dell'exposed-field `choice` specifica una lista di nodi da includere nel gruppo. Un valore tipico di `choice` è rappresentato dal nodo `Shape` o da un altro nodo di raggruppamento. Il browser VRML costruisce una sola delle forme o dei gruppi di forme elencati. Il default è una lista vuota.
- Il valore dell'exposed-field `whichChoice` specifica quale è la forma o il nodo figlio selezionato per la rappresentazione nel mondo virtuale. I nodi figli sono numerati partendo da `0` per il primo elemento della lista di `choice`.
- Se il valore di `whichChoice` è inferiore a `0` o superiore al numero di elementi di `choice`, non viene rappresentata nessuna forma. Il valore di default è `-1`, che significa che nessuna delle forme viene rappresentata.

Il nodo Switch (i)

- Il valore dell'exposed-field **whichChoice** può essere modificato inviando un valore al suo **eventIn** implicito **set_whichChoice**.
- Quando il valore è ricevuto in input, il corrispondente valore viene modificato e propagato attraverso l'**eventOut** implicito **whichChoice_changed** dell'exposed-field **whichChoice**.
- La lista delle forme nel gruppo può essere modificata inviando un valore all'**eventIn** implicito **set_choice** dell'exposed-field **choice**.
- Quando il valore è ricevuto in input, il corrispondente valore viene modificato e propagato attraverso l'**eventOut** implicito **choice_changed** dell'exposed-field **choice**.
- Tutti i nodi del nodo **Switch** continuano a ricevere ed inviare eventi, a prescindere dal valore attuale del campo **whichChoice**

Il nodo Transform

- Il nodo **Transform** crea un nuovo sistema di coordinate relativo al suo sistema di coordinate padre. Le forme create come **children** del nodo **Transform** , sono create centrate nell'origine del nuovo sistema di coordinate. I vari campi sono stati già presentati.

```
Transform {
        children    [ ]                          # exposedField MFNode
        translation 0.0 0.0 0.0                  # exposedField SFVec3f
        rotation  0.0 0.0 1.0 0.0                # exposedField SFRotation
        scale   1.0  1.0 1.0                     # exposedField SFVec3f
        scaleOrientation 0.0 0.0 1.0 0.0         # exposedField SFRotation
        bboxCenter 0.0 0.0 0.0                   # Field SFVec3F
        bboxSize -1.0 -1.0 -1.0                  # Field SFVec3F
        center 0.00 0.0 0.0                      # exposedField SFVec3f
        addChildren                              # eventIn MFNode
        removeChildren                           # eventIn MFNode
}
```

Il nodo Billboard

- Il nodo **Billboard** crea un nuovo sistema di coordinate relativo al suo sistema di coordinate padre. Le forme create come **children** del nodo **Billboard** , sono create centrate nell'origine del nuovo sistema di coordinate.

```
Billboard {
        children    [ ]                         # exposedField MFNode
        axisOfRotation   0.0 1.0 0.0        # exposedField SFVec3f
        bboxCenter 0.0 0.0 0.0              # Field SFVec3F
        bboxSize -1.0 -1.0 -1.0             # Field SFVec3F
        center 0.00 0.0 0.0                 # exposedField SFVec3f
        addChildren                         # eventIn MFNode
        removeChildren                      # eventIn MFNode
}
```

Il nodo Billboard

- Il valore dell'exposed-field `axisOfRotation` specifica un asse arbitrario rispetto al quale ruotare automaticamente il gruppo. Al muoversi del visitatore all'interno del mondo, viene automaticamente aggiornato l'angolo di rotazione in modo tale che il gruppo punti sempre verso l'osservatore.
- L'asse di rotazione può essere un qualsiasi asse. Per default l'asse è orientato lungo l'asse Y. Un asse di rotazione di `0.0 0.0 0.0` indica che il gruppo può essere ruotato arbitrariamente, senza considerare uno specifico asse di rotazione.
- Il valore dell'exposed-field `axisOfRotation` può essere cambiato mediante istradamento di un evento all' eventIn implicito `set_axisOfRotation`.
- Quando l'evento è ricevuto, viene definito il campo `axisOfRotation` e il nuovo valore viene propagato utilizzando l'eventOut implicito `axisOfRotation_changed`.

Il nodo Billboard

- Il resto dei campi sono stati già presentati nella descrizione degli altri nodi.
- Ad un nodo **`Billboard`** può essere assegnato un nome mediante l'uso di **`DEF`**.
- Si possono invocare diverse istanze del nodo mediante l'uso di **`USE`**.
- Ciascuna istanza si muove in modo indipendente, in base alla posizione nel mondo virtuale, in modo da assecondare sempre i movimenti del visitatore.

Scambio di forme (a)

```
#VRML V2.0 utf8
  Switch {
      whichChoice 0
      choice [
      # Version 0: simple round background
       Group {
              children [
                   DEF Cafe Shape {
                        appearance DEF GlowWhite
Appearance {
                                material Material
{ emissiveColor 1.0 1.0 1.0

     diffuseColor 0.0 0.0 0.0 } }
                                geometry Text { string
"Cafe"
                                        fontStyle FontStyle
{ justify "MIDDLE" } } },
                   DEF BlueDisk Transform {
                                translation 0.0 0.3 -0.10
                                rotation 1.0 0.0 0.0 -1.57
                                children Shape {
                                      appearance Appearance
{
                                           material
Material { diffuseColor 0.0 0.3 0.8 } }
                                      geometry Cylinder
{ radius 1.3 height 0.1 } } }
                  ] },
       # Version 1: round background with white edge
       Group {
                 children [
                       USE Cafe,
                       USE BlueDisk,
                       DEF WhiteDisk Transform {
                              translation 0.0 0.3 -0.10
                              rotation 1.0 0.0 0.0 -1.57
                              children Shape {
                                   appearance USE GlowWhite
                                        geometry
Cylinder { radius 1.4 height 0.08 } } }
        ] },
```


```
# Version 2: round background, white edge,
    red box
 Group {
      children [
              USE Cafe,
              USE BlueDisk,
              USE WhiteDisk,
              DEF RedAndWhiteBoxes Transform
   {
                   translation 0.0 0.3 -0.10
                   children [
                        Shape { appearance
   Appearance {
                             material Material
   {

   diffuseColor 0.8 0.0 0.0 } }
                             geometry Box
   { size 4.0 1.2 0.06 } },
                        Shape { appearance
   USE GlowWhite
                              geometry Box
   { size 4.2 1.4 0.04 } }
                         ]
                    }
              ]
         }
    ]
}
```

"scena VRML

Scambio di forme (b)

```
#VRML V2.0 utf8
      Switch {
           whichChoice 1
           choice [
           # Version 0: simple round background
            Group {
                    children [
                         DEF Cafe Shape {
                              appearance DEF GlowWhite
    Appearance {
                                   material Material
    { emissiveColor 1.0 1.0 1.0

         diffuseColor 0.0 0.0 0.0 } }
                                   geometry Text { string
    "Cafe"
                                        fontStyle FontStyle
    { justify "MIDDLE" } } },
                         DEF BlueDisk Transform {
                                   translation 0.0 0.3 -0.10
                                   rotation 1.0 0.0 0.0 -1.57
                                   children Shape {
                                        appearance Appearance
    {
                                             material
    Material { diffuseColor 0.0 0.3 0.8 } }
                                        geometry Cylinder
    { radius 1.3 height 0.1 } } }
                        ] },
           # Version 1: round background with white edge
           Group {
                    children [
                         USE Cafe,
                         USE BlueDisk,
                         DEF WhiteDisk Transform {
                                 translation 0.0 0.3 -0.10
                                 rotation 1.0 0.0 0.0 -1.57
                                 children Shape {
                                      appearance USE GlowWhite
                                           geometry
    Cylinder { radius 1.4 height 0.08 } } }
           ] },
```


```
# Version 2: round background, white edge, red box
 Group {
      children [
              USE Cafe,
              USE BlueDisk,
              USE WhiteDisk,
              DEF RedAndWhiteBoxes Transform {
                   translation 0.0 0.3 -0.10
                   children [
                        Shape { appearance Appearance
    {
                             material Material {
                                  diffuseColor 0.8
    0.0 0.0 } }
                             geometry Box { size 4.0
    1.2 0.06 } },
                        Shape { appearance USE
    GlowWhite
                             geometry Box { size 4.2
    1.4 0.04 } }
                        ]
                   }
              ]
         }
    ]
}
```

"Scena VRML

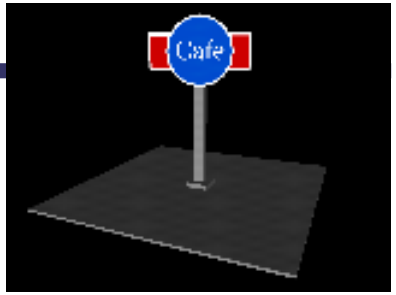
Scambio di forme (c)

```
#VRML V2.0 utf8
Switch {
    whichChoice 2
    choice [
    # Version 0: simple round background
     Group {
            children [
                 DEF Cafe Shape {
                      appearance DEF GlowWhite
Appearance {
                           material Material
{ emissiveColor 1.0 1.0 1.0

     diffuseColor 0.0 0.0 0.0 } }
                           geometry Text { string
"Cafe"
                                fontStyle FontStyle
{ justify "MIDDLE" } } },
                 DEF BlueDisk Transform {
                           translation 0.0 0.3 -0.10
                           rotation 1.0 0.0 0.0 -1.57
                           children Shape {
                                appearance Appearance
{
                                     material
Material { diffuseColor 0.0 0.3 0.8 } }
                                geometry Cylinder
{ radius 1.3 height 0.1 } } }
                ] },
     # Version 1: round background with white edge
     Group {
            children [
                 USE Cafe,
                 USE BlueDisk,
                 DEF WhiteDisk Transform {
                        translation 0.0 0.3 -0.10
                        rotation 1.0 0.0 0.0 -1.57
                        children Shape {
                             appearance USE GlowWhite
                                  geometry
Cylinder { radius 1.4 height 0.08 } } }
       ] },
```


```
# Version 2: round background, white edge,
   red box
 Group {
      children [
            USE Cafe,
            USE BlueDisk,
            USE WhiteDisk,
            DEF RedAndWhiteBoxes Transform
   {
                 translation 0.0 0.3 -0.10
                 children [
                      Shape { appearance
   Appearance {
                           material Material
   {

   diffuseColor 0.8 0.0 0.0 } }
                           geometry Box
   { size 4.0 1.2 0.06 } },
                      Shape { appearance
   USE GlowWhite
                            geometry Box
   { size 4.2 1.4 0.04 } }
                          ]
                     }
            ]
       }
    ]
}
```

"scena VRML

Nodo Billboard

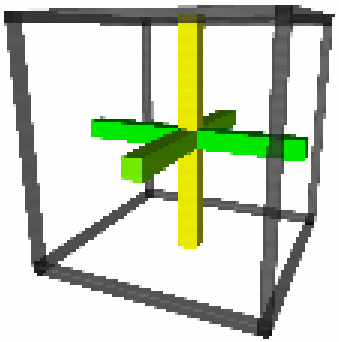



```
#VRML V2.0 utf8
Group {
    children [
    # Cafe Sign in a Billboard group
    Billboard {
          axisOfRotation 0.0 1.0 0.0
          children [
               DEF Cafe Shape {
                     appearance DEF GlowWhite Appearance {
    1.0                    material Material { emissiveColor 1.0 1.0
    diffuseColor 0.0 0.0 0.0 } }
                      geometry Text { string "Cafe" fontStyle FontStyle {
                                      justify "MIDDLE" } } },
               DEF BlueDisk Transform {
                        translation 0.0 0.3 -0.10
                        rotation 1.0 0.0 0.0 -1.57
                        children Shape {
                                appearance Appearance {
                                        material Material {diffuseColor
    0.0 0.3 0.8 } }
    } } },                      geometry Cylinder { radius 1.3 height 0.1
              DEF WhiteDisk Transform {
                        translation 0.0 0.3 -0.10
                        rotation 1.0 0.0 0.0 -1.57
                        children [
                              Shape { appearance USE GlowWhite
    height 0.08 } } ] },                      geometry Cylinder { radius 1.4
               DEF RedAndWhiteBoxes Transform {
                        translation 0.0 0.3 -0.10
                        children [
                                Shape { appearance Appearance { material
    Material {                                          diffuseColor 0.8 0.0
    0.0 } }                                   geometry Box { size 4.0
    1.2 0.06 } },
                                 Shape { appearance USE GlowWhite
                                               geometry Box { size 4.2
    1.4 0.04 } } ]
                           } ] },
```

```
 # Non-billboard sign pole and ground
  DEF Pole Transform {
            translation 0.0 -3.1 -0.10
Appearance {children Shape { appearance DEF Gray
material Material {
diffuseColor 0.6 0.6 0.6 }}
geometry Box { size 0.4 4.0 0.4 } } },
   DEF PoleBase Transform {
             translation 0.0 -5.2 -0.10
             children Shape { appearance USE Gray
1.0 } } },          geometryBox { size 1.0 0.2
    DEF Ground Transform {
             translation 0.0 -5.35 -0.10
             children Shape { appearance USE Gray
10.0 } }            geometry Box { size 10.0 0.1
             } ]
}
```

"scena VRML

Bounding boxes

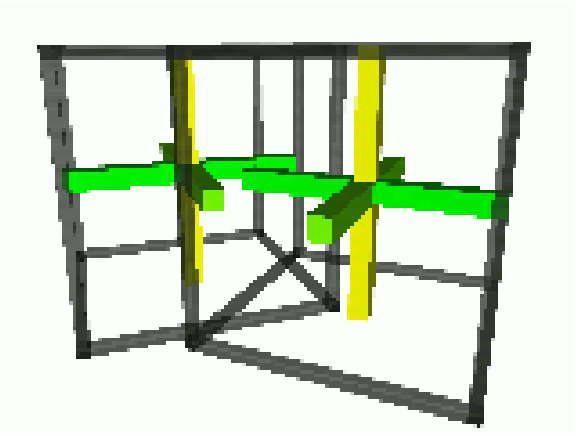


```
#VRML V2.0 utf8
Group {
      bboxCenter 0.0 0.0 0.0
      bboxSize 25.0 25.0 25.0
      children [
            Shape { appearance Appearance {
                        material Material { diffuseColor 0.0 1.0 0.0 } }
                        geometry Box { size 25.0 2.0 2.0 } },
            Shape { appearance Appearance {
                        material Material { diffuseColor 1.0 1.0 0.0 } }
                        geometry Box { size 2.0 25.0 2.0 } },
            Shape { appearance Appearance {
                         material Material { diffuseColor 0.5 1.0
   0.0 } }
                         geometry Box { size 2.0 2.0 25.0 } }
              ]
 }
```

"scena VRML

Bounding boxes

```
#VRML V2.0 utf8
Group {
     children [
      # Translated Shape
         Transform {
                   translation 20.0 0.0 0.0
                   bboxCenter 0.0 0.0 0.0
                   bboxSize 25.0 25.0 25.0
                  children [ DEF Widest Shape {
                         appearance Appearance {
                               material Material {diffuseColor
   0.0 1.0 0.0 } }
                               geometry Box { size 25.0 2.0 2.0
   } },
                        DEF Tallest Shape { appearance
   Appearance {
                               material Material { diffuseColor
   1.0 1.0 0.0 } }
                               geometry Box { size 2.0 25.0 2.0
   } },
                        DEF Deepest Shape { appearance
   Appearance {
                                material Material
   { diffuseColor 0.5 1.0 0.0 } }
                                geometry Box { size 2.0 2.0
   25.0 } } ] },
                        # Rotated Shape Transform {
                                       rotation 0.0 1.0 0.0
   0.785
                                       bboxCenter 0.0 0.0 0.0
                                        bboxSize 25.0 25.0
   25.0
                               children [
                                      USE Widest,
                                      USE Tallest,
                                      USE Deepest ]
                                  }
                ]
 }
```



"scena VRML

Virtual Reality Modeling Language

VRML

Inlining files

Inclusione di files

- La costruzione di mondi virtuali complessi in un unico file può risultare scomoda, anche perché il programmatore può organizzare gli elementi del mondo virtuale in singoli piccoli file, da includere di volta in volta nei vari mondi virtuali.
- La costruzione del mondo VRML può avvenire utilizzando nel file dei nodi `Inline`, ciascuno associato al nome del file da includere nella definizione del mondo virtuale.
- Il browser VRML legge i file specificati nei nodi `Inline` e costruisce il mondo virtuale risultante.
- In genere l'argomento del nodo `Inline` è il nome di un file presente localmente sul computer, ma può essere anche specificata una URL completa.

Inclusione di files (i)

- L'utilità di questo modo di organizzare la produzione di mondi può essere schematizzata nel modo seguente:
  - Le singole componenti possono essere testate e standardizzate una volta per tutte
  - E' semplice costruirsi una libreria di forme standard
  - L'uso in molteplici contesti dell'insieme delle componenti è immediato
  - Se si riscontra un errore, correggendo il file originario, la modifica viene automaticamente propagata in tutti i mondi che lo utilizzano
  - La possibilità di utilizzare una URL, consente di usare anche componenti non propri, ma resi disponibili in rete. Si possono specificare più URL per lo stesso file, in modo di essere sicuri che venga trovato.
  - Consente di pubblicare in rete le proprie componenti per consentirne l'uso pubblico

Inclusione di files (ii)

- Anche il nodo `Inline` è un nodo che raggruppa le forme, come i nodi `Group`, `Billboard` e `Transform`: solo che in questo caso il file non contiene la specifica delle forme, ma la/e URL relativa/e ai file che contengono dette definizioni.
- Il nodo `Inline` contempla la definizione di un bounding box. Se il bounding box è fuori dall'area visibile, il file non viene neanche letto: lo sarà al momento in cui diventa visibile.
- Quando si usa il nodo `Inline` occorre tenere in considerazione che il campo di esistenza dei nomi dei nodi è limitato al singolo file: eventuali nomi assegnati mediante `DEF` possono essere istanziati solo all'interno del file contenete la `DEF` e nomi definiti all'interno del file contenente la specifica del nodo `Inline` non possono essere utilizzati all'interno dei file da includere.

Il nodo Inline

- Il nodo **Inline** consente l'inclusione di file per la definizione delle forme VRML. Il nodo **Inline** può essere usato come componente di un qualsiasi nodo di raggruppamento.

```
Inline {
        url             [ ]                 # exposedField MFString
        bboxCenter 0.0 0.0 0.0              # Field SFVec3F
        bboxSize -1.0 -1.0 -1.0             # Field SFVec3F
        }
```

Il nodo Inline

- Il valore dell'exposed-field `url` specifica una lista con priorità di URL, ordinata dalla priorità maggiore alla minore.
- Il browser VRML cerca di aprire il file definito dalla prima URL, se il file non può essere trovato, prova con la seconda URL, e così via.
- Quando il file viene trovato, viene letto e le definizioni in esso contenute vengono trattate come se fossero `children` di un nodo `Group`.
- Se nessun file può essere aperto, il nodo `Inline` viene trattato come se fosse un nodo `Group` privo di `children`.
- Il valore di default del campo `url` è una lista vuota di URL. La lista può essere modificata inviando un evento all'eventIn implicito `set_url` dell'exposed-field `url`.Il nuovo valore viene propagato mediante l'eventOut implicito `url_changed`.

Il nodo Inline (i)

- La ridefinizione del campo `url` mediante l'eventIn `set_url` comporta la cancellazione delle definizioni del file incluso precedentemente e l'attivazione delle definizioni lette nel nuovo file.
- Se il vecchio file conteneva dei nodi `Script`, che vedremo più avanti, eventuali programmi in esecuzione ed associati al nodo `Script` vengono cancellati prima di caricare le definizioni dei nuovi nodi.
- Il file incluso mediante un nodo `Inline` deve essere un file VRML.

Nodo Inline: arch.wrl

```
#VRML V2.0 utf8
Group {
    children [
    # First archway
    # Left Column
    DEF LeftColumn Transform {
          translation -2.0 3.0 0.0
          children DEF Column Shape {
                appearance DEF White Appearance {
                material Material { } }
                geometry Cylinder { radius 0.3 height 6.0 } } },
     # Right Column
     DEF RightColumn Transform {
                translation 2.0 3.0 0.0
                children USE Column },
      # Archway span
      DEF ArchwaySpan Transform {
                translation 0.0 6.05 0.0
                children Shape {
                     appearance USE White
                     geometry Box { size 4.6 0.4 0.6 } } },
       # Left Roof
       DEF LeftRoof Transform {
                translation -1.15 7.12 0.0
           rotation 0.0 0.0 1.0 0.524
          children DEF Roof Shape {
          appearance USE White
          geometry Box { size 2.86 0.4 0.6 } } },
    # Right Roof
    DEF LeftRoof Transform {
            translation 1.15 7.12 0.0
            rotation 0.0 0.0 1.0 -0.524
            children USE Roof }
 ] }
```

" scena VRML

Nodo Inline

```
#VRML V2.0 utf8
Group {
    children [
        DEF Arch Inline {
            bboxSize 5.0 8.0 2.0
            bboxCenter 1.0 4.0 0.0
            url "arch.wrl" },
        Transform {
            translation 0.0 0.0 -2.0
            children USE Arch },
        Transform {
            translation 0.0 0.0 -4.0
            children USE Arch },
        Transform {
            translation 0.0 0.0 -6.0
            children USE Arch },
        Transform {
            translation 0.0 0.0 -8.0
            children USE Arch }
    ]
}
```

"scena VRML

Nodo Inline (i)

```
#VRML V2.0 utf8
Group {
      children [
      # Ground
           Shape {
               appearance Appearance {
                      material Material { diffuseColor 0.0 1.0
   0.0 } }
               geometry Box { size 50.0 0.1 50.0 } },
      # Back archway row
          Transform {
                translation 0.0 0.0 -4.0
                children DEF ArchRow Inline {
                          bboxSize 5.0 8.0 10.0
                          bboxCenter 1.0 4.0 -5.0
                          url "archrow.wrl" } },
       # Front archway row
           Transform {
                  translation 0.0 0.0 4.0
                  rotation 0.0 1.0 0.0 3.14
                  children USE ArchRow },
        # Left archway row
            Transform {
                   translation -4.0 0.0 0.0
                   rotation 0.0 1.0 0.0 1.57
                   children USE ArchRow },
        # Right archway row
            Transform {
                   translation 4.0 0.0 0.0
                   rotation 0.0 1.0 0.0 -1.57
                   children USE ArchRow }
    ]
}
```

"scena VRML

Virtual Reality Modeling Language

VRML

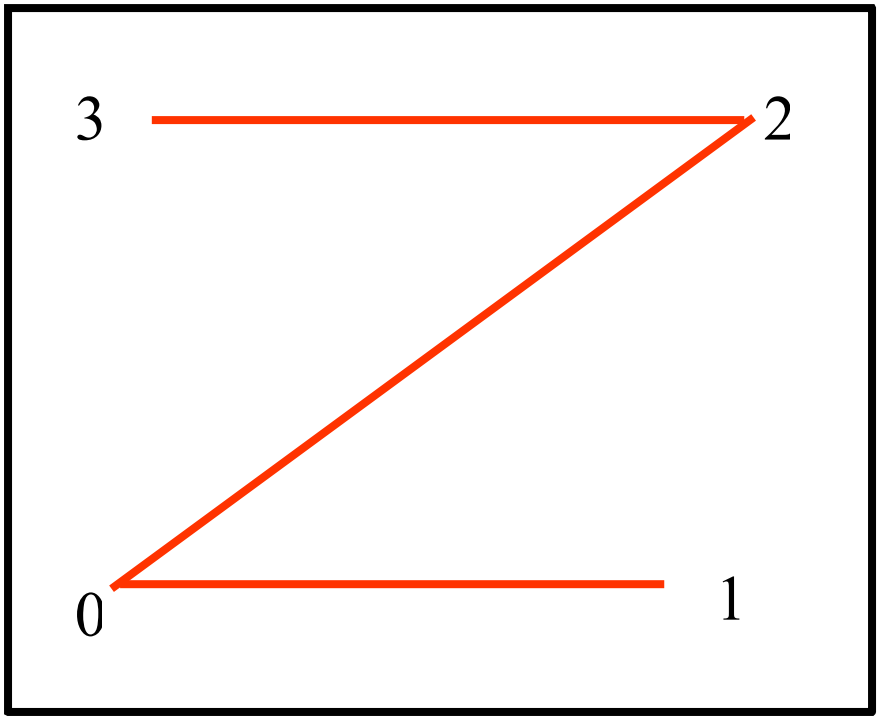
Costruzione di Forme con Punti, Linee e Facce

Costruire Forme

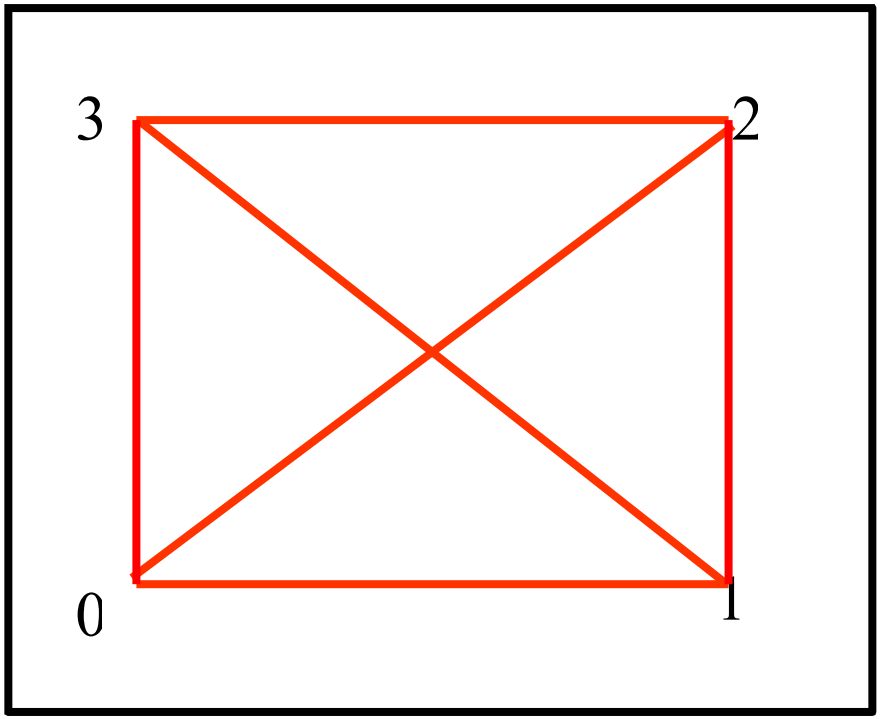
- Pur fornendo un ampio insieme di primitive per la costruzione di forme predefinite, per comprendere appieno le potenzialità di VRML è importante vedere come poter realizzare forme di qualsiasi tipo.
- VRML fornisce i nodi per poter costruire superfici che seguano un qualunque andamento, utilizzando punti, linee e facce.
- Un **punto** VRML è un punto situato nel mondo 3-D, un insieme di punti è chiamato *point set*: utilizzando il nodo `PointSet` si possono realizzare scatter plot.
- Una **linea** VRML è una linea retta che connette 2 punti del mondo 3-D; un insieme di linee è chiamato *line set*. Utilizzando il nodo `IndexedLineSet` si possono realizzare line plot, curve e griglie.

Polyline

A



B



In A la lista delle coordinate è: `3, 2, 0, 1`

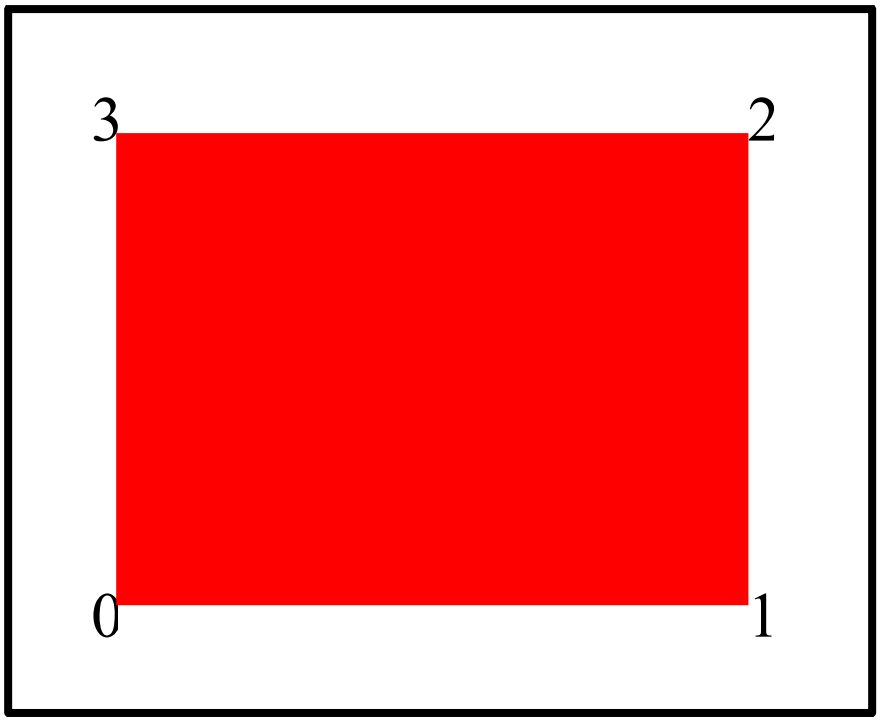
In B è: `0, 1, 2, 3, 0, -1, 0, 2, -1, 1, 3`

Dove la coordinata speciale `-1` indica la fine della polyline

Costruire Forme (i)

- Una **faccia** VRML è una forma piatta, come un triangolo, un quadrato o un ottagono. Il perimetro della faccia si ottiene connettendo gli angoli della stessa. Il browser VRML riempie e ombreggia l'interno della faccia.
- Accostando diversi insiemi di facce in un nodo `IndexedFaceSet` si possono realizzare forme complesse e sfaccettate. Maggiore è il numero delle facce e minori sono le loro dimensioni, maggiore è la risoluzione delle forme realizzate.
- I nodi `PointSet`, `IndexedLineSet` e `IndexedFaceSet` usano il nodo `Coordinate` per specificare le posizioni 3-D in un insieme di punti, gli estremi delle linee in un insieme di linee e gli angoli in un insieme di facce.

Facce



Questa volta la lista delle coordinate è: `0, 1, 2, 3`.

In questo caso non serve specificare che la curva si chiude (bastano 4 coordinate)…

CoordinateInterpolator

- Utilizzando un nodo **`CoordinateInterpolator`** si possono animare le posizioni 3-D di un nodo **`Coordinate`**, pertanto si possono realizzare delle forme che cambiano durante un'animazione.
- Il nodo **`CoordinateInterpolator`** specifica una lista di tempi parziali e di coordinate chiave nei campi **`key`** e **`keyValue`**.
- Sotto l'effetto di una serie di tempi parziali generati da un nodo **`TimeSensor`**, il nodo **`CoordinateInterpolator`** calcola le posizioni intermedie alle posizioni chiave mediante interpolazione lineare. Ciascuna posizione intermedia calcolata viene propagata mediante l'eventOut **`value_changed`**. Si può creare un route tra questo valore ed un nodo **`Coordinate`** per animare le coordinate di una forma.

Morphing geometrico

- Se al tempo parziale `0.0` corrispondono le coordinate di una sfera e al tempo parziale `1.0` corrispondono le forme di una pillola piatta, l'animazione trasforma la sfera in una pillola. Questo è noto con il nome di morphing geometrico.
- I nodi `PointSet`, `IndexedLineSet` e `IndexedFaceSet` posssono essere usati come valori del campo `geometry` di un nodo `Shape` e le caratteristiche della forma possono essere definite mediante il nodo `Appearance`.
- In VRML si possono connettere diversi punti mediante una *polyline*. Una polyline ha una coordinata di partenza, diverse coordinate intermedie ed una coordinata finale. Il browser connette i punti intermedi mediante dei segmenti.
- I punti di una polyline sono specificati mediante il nodo `Coordinate`

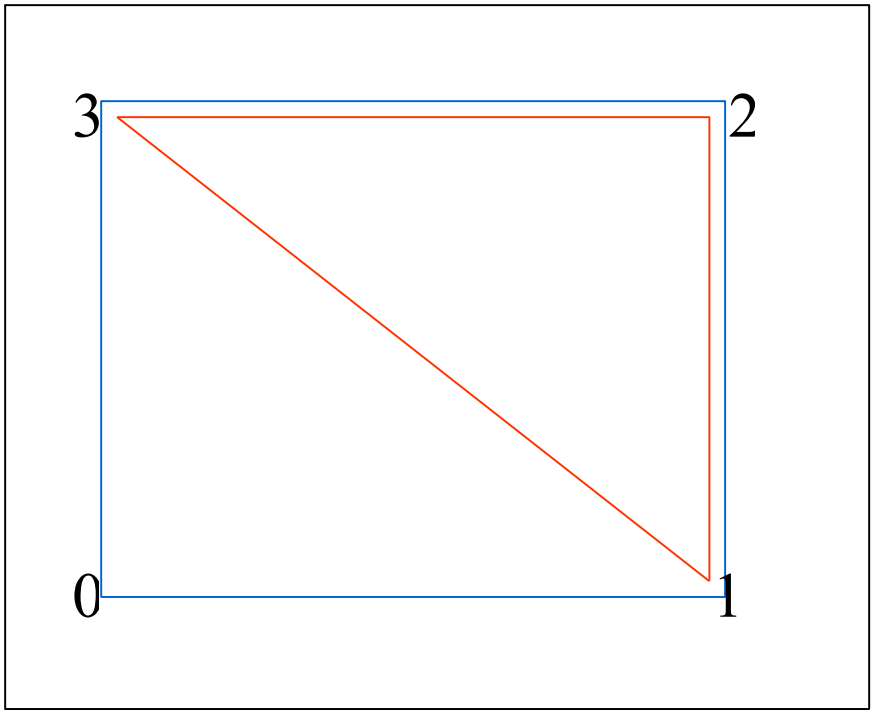
Costruire Forme (iv)

Coordinate di una scatola 3-D

Indice della coordinata	Coordinata (X,Y,Z)
0	-1.0 1.0 1.0
1	1.0 1.0 1.0
2	1.0 1.0 -1.0
3	-1.0 1.0 -1.0
4	-1.0 -1.0 1.0
5	1.0 -1.0 1.0
6	1.0 -1.0 -1.0
7	-1.0 -1.0 -1.0

Costruire Forme (v)

- Usando lo stesso insieme di coordinate si possono creare diverse forme semplicemente connettendo le coordinate in modo differente:



Il nodo Coordinate

- Il nodo **Coordinate** crea una lista di coordinate e può essere usato come valore per il campo di nodi geometrici basati su coordinate, come **PointSet**, **IndexedLineSet** e **IndexedFaceSet**.

```
Coordinate {
        point   [ ]                     # exposedField MFVec3f
        }
```

- Il valore dell'exposed-field **point** contiene una lista di coordinate 3D usate come coordinate di una forma. Ciascuna coordinata è espressa da una terna di valori reali che indicano le componenti X Y e Z della distanza dall'origine della coordinata. Il default è una lista vuota.
- Le coordinate possono essere cambiate inviando un valore all'eventIn implicito **set_point** dell'exposed field **point** (per es. da un nodo **CoordinateInterpolator**). Ricevuto un valore, la lista delle coordinate viene modificata e propagata attraverso l'eventOut **point_changed** .

Il nodo CoordinateInterpolator

- Il nodo **CoordinateInterpolator** descrive una lista di coordinate che possono essere usate in un'animazione.

```
CoordinateInterpolator {
        key         [ ]                 # exposedField MFFloat
        keyValue    [ ]                 # exposedField MFVec3d
        set_fraction                    # eventIn      SFFloat
        value_changed                   # eventOut     MFVec3f
        }
```

- Il valore dell'exposed-field **key** contiene una lista di tempi parziali. Valori tipici sono compresi tra **0.0** e **1.0** come quelli generati dall'eventOut **fraction_changed** di un nodo **TimeSensor**.
- I tempi chiave devono essere elencati in ordine non decrescente.
- Il valore di default di **key** è una lista vuota.

Il nodo CoordinateInterpolator

- Il valore dell'exposed-field **`keyValue`** è una lista di coordinate 3-D. Ciascuna coordinata è espressa da una terna di valori reali che indicano le componenti X Y e Z della distanza dall'origine della coordinata. Il default è una lista vuota.
- Quando il nodo riceve un tempo parziale attraverso l'eventIn **`set_fraction`** il nodo calcola una lista di coordinate sulla base delle coordinate chiave specificate in **`keyValue`**. La coordinata calcolata vien propagata attraverso l'eventOut **`fraction_changed`**.
- La lista dei tempi e delle coordinate chiave può essere cambiata inviando valori negli eventIn impliciti **`set_key`** e **`set_keyValue`** degli exposed-field **`key`** e **`keyValue`**.
- Il nodo **`CoordinateInterpolator`** non crea forme e non ha effetti visibili nel mondo virtuale. Un nodo **`CoordinateInterpolator`** può essere figlio di qualsiasi nodo di raggruppamento ed è indipendente dal sistema di coordinate del sistema.

Il nodo PointSet

- Il nodo **PointSet** crea geometrie puntuali e può essere usato come valore del campo **geometry** in un nodo **Shape**.

```
PointSet {
        coord       NULL                    # exposedField SFNode
        color       NULL                    # exposedField SFNode
         }
```

- Il valore dell'exposed-field **coord** specifica un nodo che contenga la lista delle coordinate usate per identificare ciascun punto dell'insieme di punti. Un valore tipico del campo **coord** è un nodo **Coordinate**.
- Il valore di default **NULL** indica una lista vuota e che nessun punto costituisce l'insieme.
- I punti sono rappresentati uno alla volta, in ordine, a partire dalla prima coordinata della lista.

Il nodo PointSet

- I valori delle coordinate del nodo possono essere modificati inviando un valore all'eventIn implicito **`set_coord`** dell'exposed-field **`coord`**. Il valore viene cambiato e propagato attraverso l'eventOut implicito **`coord_changed`**.
- Il nodo **`Appearance`** del nodo **`Shape`** specifica il colore globale dell'insieme di punti. Le forme del nodo sono rappresentate usando il campo **`emissiveColor`** del nodo **`Material`** del nodo **`Appearance`**.
- Occorre fare attenzione al fatto che il colore di default di **`emissiveColor`** è nero, così come è nero il background di default.
- Si possono colorare i singoli punti utilizzando il nodo **`Color`** come valore del campo **`color`** del nodo **`PointSet`**.
- Alle forme associate al nodo **`PointSet`** non possono essere applicate texture.

Il nodo IndexedLineSet

- Il nodo **IndexedLineSet** crea geometrie di tipo polyline e può essere usato come valore del campo **geometry** in un nodo **Shape**.

```
IndexedLineSet {
        coord           NULL            # exposedField SFNode
        coordIndex      [ ]             # field        MFInt32
        color           NULL            # exposedField SFNode
        colorIndex      [ ]             # field        MFInt32
        colorPerVertex  TRUE            # field        SFBool
        set_coordIndex                  # eventIn      MFInt32
        set_colorIndex                  # eventIn      MFInt32

         }
```

Il nodo IndexedLineSet

- Il valore dell'exposed-field **coord** specifica un nodo che contenga la lista delle coordinate usate per identificare ciascun punto dell'insieme di punti. Un valore tipico del campo **coord** è un nodo **Coordinate**.
- Il valore di default **NULL** indica una lista vuota e che nessun punto costituisce l'insieme.
- Il valore del campo **coordIndex** specifica una lista di indici di coordinate che descrivono il percorso di una o più spezzate. Ciascun valore è un indice intero che specifica una coordinata nella lista del campo **coord**. Il valore di default è una lista vuota, indicante che nessuna linea deve essere rappresentata.
- La lista di coordinate può specificare una o più spezzate. Ciascuna spezzata connette insieme una sequenza di indici di coordinate fino alla fine della spezzata o fino a che non si incontra l'indice **-1**. L'indice successivo al valore **-1** inizia una nuova spezzata che viene rappresentata fino al prossimo valore **-1** o fino alla fine della lista.
- I valori delle coordinate del nodo possono essere modificati inviando un valore all'eventIn implicito **set_coord** dell'exposed-field **coord**. Il valore viene cambiato e propagato attraverso l'eventOut implicito **coord_changed**.

Il nodo IndexedLineSet

- L'indice delle coordinate può essere modificato inviando un valore all'eventIn **`set_coordIndex`**.
- Il nodo **`Appearance`** del nodo **`Shape`** specifica il colore globale dell'insieme di punti. Le forme del nodo sono rappresentate usando il campo **`emissiveColor`** del nodo **`Material`** del nodo **`Appearance`**.
- Occorre fare attenzione al fatto che il colore di default di **`emissiveColor`** è nero, così come è nero il background di default. Si ottengono per default delle linee nere su sfondo nero, che pertanto non vengono visualizzate.
- Si possono colorare le singole linee utilizzando i campi **`color`**, **`colorIndex`**, **`set_colorIndex`** e **`colorPerVertex`**.
- Alle forme associate al nodo **`IndexedLineSet`** non possono essere applicate texture.

Il nodo IndexedFaceSet

- Il nodo **IndexedFaceSet** crea geometrie a facce e può essere usato come valore del campo **geometry** in un nodo **Shape**.

```
IndexedFaceSet {
        coord           NULL            # exposedField SFNode
        coordIndex      [ ]             # field       MFInt32
        texCoord        NULL            # exposedField SFNode
        texCoordIndex   [ ]             # field       MFInt32
        color           NULL            # exposedField SFNode
        colorIndex      [ ]               # field          MFInt32
        colorPerVertex  TRUE            # field          SFBool
        normal          NULL              # exposedField SFNode
        normalIndex     [ ]             # field          MFInt32
        normalPerVertex TRUE            # field          SFBool
        ccw             TRUE            # field          SFBool
        convex          TRUE            # field          SFBool
        solid           TRUE            # field          SFBool
        creaseAngle     0.0             # field          SFFloat
        set_coordIndex                  # eventIn     MFInt32
        set_texCoordIndex               # eventIn     MFInt32
        set_colorIndex                  # eventIn     MFInt32
        set_normalIndex                 # eventIn     MFInt32
        }
```

Il nodo IndexedFaceSet

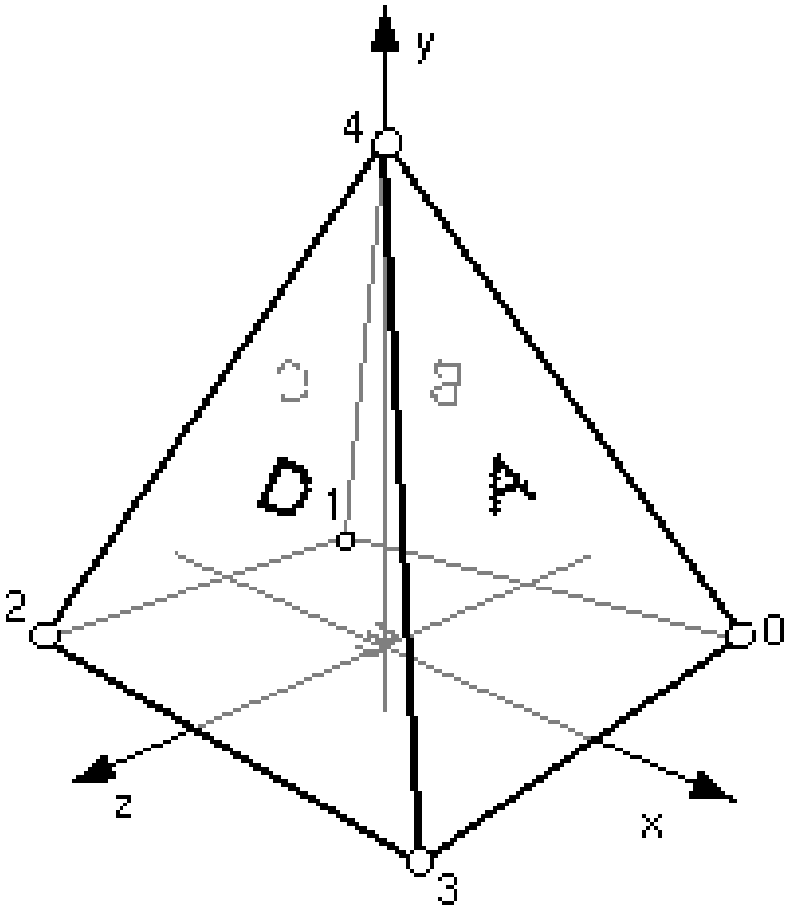
- Il valore dell'exposed-field **coord** specifica un nodo che contenga la lista delle coordinate usate per identificare ciascun punto dell'insieme di punti. Un valore tipico del campo **coord** è un nodo **Coordinate**.
- Il valore di default **NULL** indica una lista vuota e che nessun punto costituisce l'insieme.
- Il valore del campo **coordIndex** specifica una lista di indici di coordinate che descrivono i vertici di una o più facce. Ciascun valore è un indice intero che specifica una coordinata nella lista del campo **coord**. Il valore di default è una lista vuota, il che implica che non deve essere rappresentata nessuna linea.
- La lista di coordinate può specificare una o più facce. Ciascuna faccia connette insieme una serie di coordinate specificata dalla sequenza del vettore indice (sequenza che termina con l'indice **-1** o con l'ultimo elemento del vettore). La faccia viene automaticamente chiusa connettendo il primo vertice con l'ultimo. L'indice successivo al valore **-1** indica la coordinata dalla quale ha inizio una nuova faccia.
- Il campo **ccw** specifica i valori **TRUE** o **FALSE**. Il valore **TRUE** indica che i vertici sono collegati in senso orario per realizzare la faccia.
- I lati posteriori delle facce sono realizzati in base al valore del campo **solid**, il quale indica se la forma è o no un solido pieno.

Il nodo IndexedFaceSet

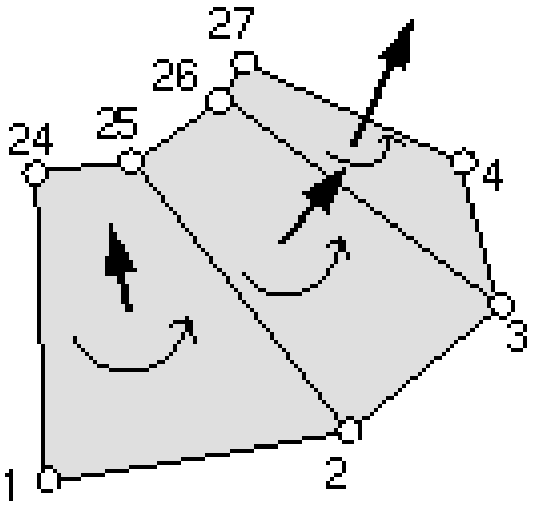
- Il campo **convex** specifica se le facce sono convesse (**TRUE**). Quando il valore è **FALSE** tutte le facce concave sono automaticamente suddivise in tante facce convesse.
- Il campo **creaseAngle** indica un valore limite: facce vicine con un angolo inferiore di **creaseAngle** vengono realizzate con spigoli arrotondati, mediante interpolazione. Le facce con angolo superiore vengono realizzate formando un angolo vivo, spigoloso.
- I valori delle coordinate del nodo possono essere modificati inviando un valore all'*eventIn* implicito **set_coord** dell'exposed-field **coord**. Il valore viene cambiato e propagato attraverso l'*eventOut* implicito **coord_changed**.
- La lista degli indici delle coordinate può essere cambiata inviando la nuova lista all'*eventIn* **set_coordIndex** .
- Il nodo **Appearance** del nodo **Shape** specifica il colore globale dell'insieme di punti. Le forme del nodo sono rappresentate usando il campo **emissiveColor** del nodo **Material** del nodo **Appearance**.

Il nodo IndexedFaceSet

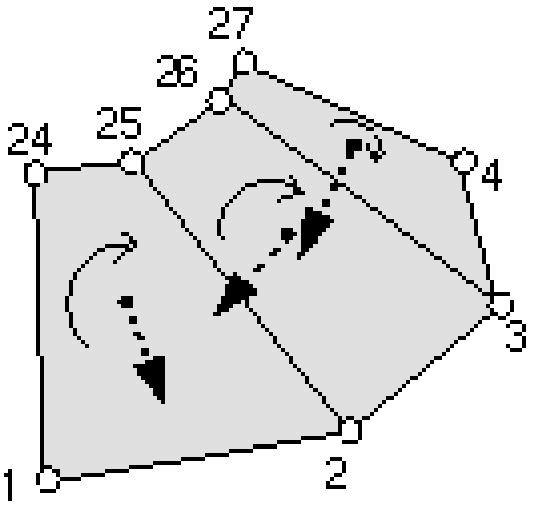
```
IndexedFaceSet {
   coord Coordinate {
       point [ 1 0 -1,
               -1 0 -1,
               -1 0 1,
               1 0 1,
               0 2 0 ] }
   coordIndex [ 0 4 3 -1 # face A, right
                  1 4 0 -1 # face B, back
                  2 4 1 -1 # face C, left
                  3 4 2 -1 # face D, front
                  0 3 2 1 ] # face E, bottom
       }
```



Il campo `ccw`

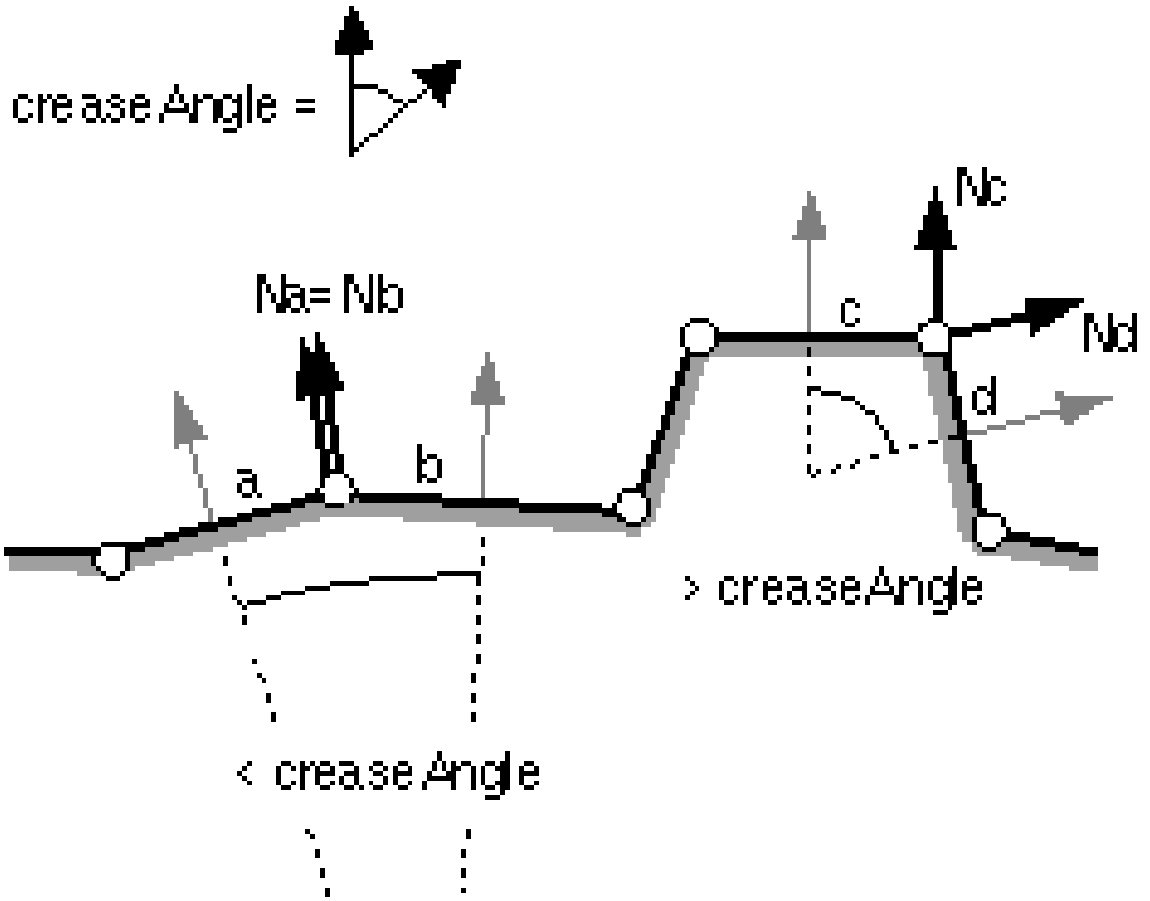



```
IndexedFaceSet {
  coordIndex [1 2 25 24 -1
              2 3 26 25 -1
              3 4 27 26]
  ccw TRUE
  ....
}
```




```
IndexedFaceSet {
  coordIndex [1 2 25 24 -1
              2 3 26 25 -1
              3 4 27 26]
  ccw FALSE
  ....
}
```

Il campo `creaseangle`

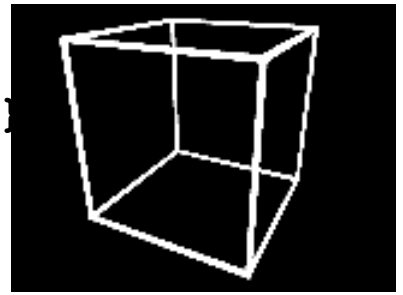


Esempi: PointSet

```
#VRML V2.0 utf8
Shape {
   appearance Appearance {
          material Material { emissiveColor 1.0 1.0 1.0 } }
   geometry PointSet {
          coord Coordinate {
               point [
              # Coordinates around the top of the cube
                        -1.0 1.0 1.0, 1.0 1.0 1.0, 1.0 1.0 -1.0,
                        -1.0 1.0 -1.0,
             # Coordinates around the bottom of the cube
                        -1.0 -1.0 1.0, 1.0 -1.0 1.0, 1.0 -1.0 -1.0,
                        -1.0 -1.0 -1.0 ]
      }
   }
 }
```

- scena VRML

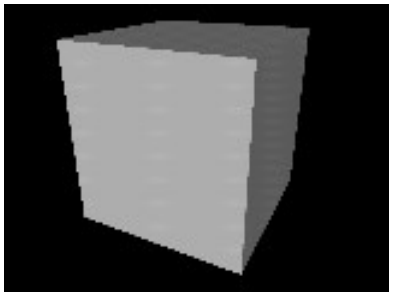
Esempi: IndexedLineSet



```
#VRML V2.0 utf8
Shape {
   appearance Appearance {
          material Material { emissiveColor 1.0 1.0 1.0 } }
   geometry IndexedLineSet {
          coord Coordinate {
               point [
              # Coordinates around the top of the cube
                        -1.0 1.0 1.0, 1.0 1.0 1.0, 1.0 1.0 -1.0,
                        -1.0 1.0 -1.0,
             # Coordinates around the bottom of the cube
                        -1.0 -1.0 1.0, 1.0 -1.0 1.0, 1.0 -1.0 -1.0,
                        -1.0 -1.0 -1.0 ]
          }
          coordIndex [
              # top
              0, 1, 2, 3, 0, -1,
              # bottom
              4, 5, 6, 7, 4, -1,
              # vertical edges
              0, 4, -1, 1, 5, -1, 2, 6, -1, 3, 7
          ] }
  }
 }
```

scena VRML

Esempi: IndexedFaceSet



```
#VRML V2.0 utf8
Shape {
   appearance Appearance {
          material Material {   } }
   geometry IndexedFaceSet {
          coord Coordinate {
               point [
              # Coordinates around the top of the cube
                        -1.0 1.0 1.0, 1.0 1.0 1.0, 1.0 1.0 -1.0,
                        -1.0 1.0 -1.0,
             # Coordinates around the bottom of the cube
                        -1.0 -1.0 1.0, 1.0 -1.0 1.0, 1.0 -1.0 -1.0,
                        -1.0 -1.0 -1.0 ]
         }
         coordIndex [
                  0, 1, 2, 3, -1, # top
                7, 6, 5, 4, -1, # bottom
                0, 4, 5, 1, -1, # front
                1, 5, 6, 2, -1, # right
                2, 6, 7, 3, -1, # back
                3, 7, 4, 0      # left ] }
 }
```

scena VRML

Facce concave

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
Shape {
    appearance Appearance { material Material { diffuseColor 1.0 1.0 0.0 } }
    geometry IndexedFaceSet {
       coord Coordinate { point [
           # Lighting bolt tip
           0.0 0.0 0.0,
           # Front perimeter
           5.5 5.0 0.88, 4.0 5.5 0.968, 7.0 8.0 1.408, 4.0 9.0 1.584,
                         1.0 5.0 0.88, 2.5 4.5 0.792,
           # Back perimeter
           5.5 5.0 -0.88, 4.0 5.5 -0.968, 7.0 8.0 -1.408, 4.0 9.0 -1.584,
                         1.0 5.0 -0.88, 2.5 4.5 -0.792,
       ] }
      coordIndex [
           # Front
           0, 1, 2, 3, 4, 5, 6, -1,
           # Back
           0, 12, 11, 10, 9, 8, 7, -1,
           # Sides
           0, 7, 1, -1, 1, 7, 8, 2, -1, 2, 8, 9, 3, -1, 3, 9, 10, 4, -1, 4,
                         10, 11, 5, -1, 5, 11, 12, 6, -1, 6, 12, 0, -1,
       ]
    convex FALSE
    }
}
```

scena VRML

Forme non solide

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
Shape {
    appearance Appearance { material Material { } }
    geometry IndexedFaceSet {
       coord Coordinate { point [
          # Circular arc
           -1.0 0.0 1.0, -1.0 0.26 0.97, -1.0 0.5 0.87, -1.0 0.71 0.71,
           -1.0 0.87 0.5, -1.0 0.97 0.26,-1.0 1.0 0.0, -1.0 0.97 -0.26,
           -1.0 0.87 -0.5, -1.0 0.71 -0.71, -1.0 0.5 -0.87, -1.0 0.26 -0.97,
           -1.0 0.0 -1.0,
           # Angled circular arc
           -1.0 0.0 1.0, -0.97 0.26 0.97, -0.87 0.5 0.87, -0.71 0.71 0.71, -0.5 0.87 0.5,
           -0.26 0.96 0.26, 0.0 1.0 0.0,  -0.26 0.96 -0.26, -0.5 0.87 -0.5,
           -0.71 0.71 -0.71, -0.87 0.5 -0.87, -0.97 0.26 -0.97, -1.0 0.0 -1.0,
        ] }
      coordIndex [ 0, 13, 14, 1, -1, 1, 14, 15, 2, -1, 2, 15, 16, 3, -1, 3, 16, 17, 4, -1,
                   4, 17, 18, 5, -1, 5, 18, 19, 6, -1, 6, 19, 20, 7, -1, 7, 20, 21, 8, -1,
                   8, 21, 22, 9, -1, 9, 22, 23, 10,-1, 10, 23, 24, 11, -1, 11, 24, 25, 12,
                   -1,
        ]
      solid FALSE
   }
}
```

scena VRML

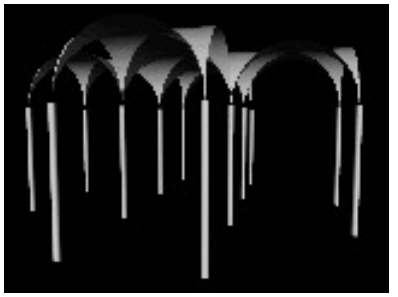
Forme non solide smooth (`creaseAngle`)

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
Shape {
    appearance Appearance { material Material { } }
    geometry IndexedFaceSet {
       coord Coordinate { point [
          # Circular arc
           -1.0 0.0 1.0, -1.0 0.26 0.97, -1.0 0.5 0.87, -1.0 0.71 0.71,
           -1.0 0.87 0.5, -1.0 0.97 0.26,-1.0 1.0 0.0, -1.0 0.97 -0.26,
           -1.0 0.87 -0.5, -1.0 0.71 -0.71, -1.0 0.5 -0.87, -1.0 0.26 -0.97,
           -1.0 0.0 -1.0,
           # Angled circular arc
           -1.0 0.0 1.0, -0.97 0.26 0.97, -0.87 0.5 0.87, -0.71 0.71 0.71, -0.5 0.87 0.5,
           -0.26 0.96 0.26, 0.0 1.0 0.0,  -0.26 0.96 -0.26, -0.5 0.87 -0.5,
           -0.71 0.71 -0.71, -0.87 0.5 -0.87, -0.97 0.26 -0.97, -1.0 0.0 -1.0,
        ] }
      coordIndex [ 0, 13, 14, 1, -1, 1, 14, 15, 2, -1, 2, 15, 16, 3, -1, 3, 16, 17, 4, -1,
                   4, 17, 18, 5, -1, 5, 18, 19, 6, -1, 6, 19, 20, 7, -1, 7, 20, 21, 8, -1,
                   8, 21, 22, 9, -1, 9, 22, 23, 10,-1, 10, 23, 24, 11, -1, 11, 24, 25, 12,
                   -1,
        ]
      solid FALSE
      creaseAngle 0.785
   }
}
```

scena VRML

Combinazione di Facce

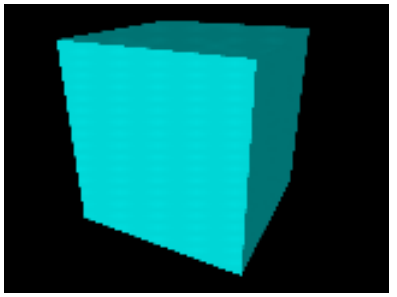
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
Group { children [
    # Center vaulted ceiling
      DEF Ceiling Transform {
              translation 0.0 2.0 0.0
              children [  DEF Vault Inline { url "vault.wrl" },
                   Transform { rotation 0.0 1.0 0.0 1.57  children USE Vault },
                   Transform { rotation 0.0 1.0 0.0 3.14 children USE Vault },
                   Transform { rotation 0.0 1.0 0.0 -1.57 children USE Vault }
               ]
       },
    # Left, right, front, and back vaulted ceilings
      Transform { translation -2.0 0.0 0.0 children USE Ceiling },
      Transform { translation 2.0 0.0 0.0 children USE Ceiling },
      Transform { translation 0.0 0.0 -2.0 children USE Ceiling },
      Transform { translation 0.0 0.0 2.0 children USE Ceiling },
    # Columns supporting the vaulted ceilings
      Transform { translation -3.0 1.0 -1.0 children
             DEF Column Shape { appearance Appearance { material Material { } }
                                                  geometry Cylinder { height 2.0 radius
     0.05 } } },
       Transform { translation -1.0 1.0 -1.0 children USE Column },
       Transform { translation 1.0 1.0 -1.0 children USE Column },
       Transform { translation 3.0 1.0 -1.0 children USE Column },
       Transform { translation -3.0 1.0 1.0 children USE Column },
       Transform { translation -1.0 1.0 1.0 children USE Column },
       Transform { translation 1.0 1.0 1.0 children USE Column },
       Transform { translation 3.0 1.0 1.0 children USE Column },
       Transform { translation -1.0 1.0 -3.0 children USE Column },
       Transform { translation -1.0 1.0 3.0 children USE Column },
       Transform { translation 1.0 1.0 -3.0 children USE Column },
       Transform { translation 1.0 1.0 3.0 children USE Column }
    ]
}
```



scena VRML

Animazione delle coordinate

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
Group { children [
   # Morphing shape
   Shape { appearance Appearance {
          material Material { diffuseColor 0.0 1.0 1.0 } }
          geometry IndexedFaceSet {
               coord DEF CubeCoordinates Coordinate { point [
                  # Coordinates around top of cube
                     -1.0 1.0 1.0, 1.0 1.0 1.0, 1.0 1.0 -1.0, -1.0 1.0 -1.0,
                  # Coordinates around bottom of cube
                     -1.0 -1.0 1.0, 1.0 -1.0 1.0, 1.0 -1.0 -1.0, -1.0 -1.0 -1.0
                  ] }
               coordIndex [
                  # top
                     0, 1, 2, 3, -1,
                  # bottom
                     7, 6, 5, 4, -1,
                  # front
                       0, 4, 5, 1, -1,
                  # right
                       1, 5, 6, 2, -1,
                  # back
                       2, 6, 7, 3, -1,
                  # left
                       3, 7, 4, 0      ]
               }
        },
     # Animation clock
     DEF Clock TimeSensor { cycleInterval 4.0 loop TRUE },
     # Animation morph
     DEF CubeMorph CoordinateInterpolator { key [ 0.0, 0.5, 1.0 ] keyValue [
               # time 0.0 coordinates (cube)
                  -1.0 1.0 1.0, 1.0 1.0 1.0, 1.0 1.0 -1.0, -1.0 1.0 -1.0, -1.0 -1.0 1.0,
                   1.0 -1.0 1.0, 1.0 -1.0 -1.0, -1.0 -1.0 -1.0,
               # time 0.5 coordinates (warped cube)
                  -1.5 1.0 1.5, 1.5 1.0 1.5, 1.5 1.0 -1.5, -1.5 1.0 -1.5, -0.5 -1.0 0.5,
                   0.5 -1.0 0.5, 0.5 -1.0 -0.5, -0.5 -1.0 -0.5,
               # time 1.0 coordinates (cube)
                  -1.0 1.0 1.0, 1.0 1.0 1.0, 1.0 1.0 -1.0, -1.0 1.0 -1.0, -1.0 -1.0 1.0,
                   1.0 -1.0 1.0, 1.0 -1.0 -1.0, -1.0 -1.0 -1.0
                ] }
   ]
}
ROUTE Clock.fraction_changed TO CubeMorph.set_fraction
ROUTE CubeMorph.value_changed TO CubeCoordinates.set_point
```



scena VRML

Virtual Reality Modeling Language

VRML

Costruzione di superfici 3D – Elevation Grid

Costruire Elevation Grid

- Uno dei problemi più frequenti che si riscontrano nel disegno di mondi virtuali è quello di costruire forme che rappresentino il paesaggio in termini di terreni e montagne.
- Utilizzando il nodo VRML **`ElevationGrid`** è possibile costruire forme complesse definendo una griglia di facce quadrate adiacenti. Per ciascuna faccia i valori delle coordinate X e Z indicano le coordinate degli angoli della faccia su tutta la griglia (grid) e il valore di Y, indica l'altezza (elevation) del punto
- Questo tipo di griglia è talmente utilizzata in VRML che gli autori hanno implementato un nodo apposito, **`ElevationGrid`**, che ne semplifica la rappresentazione.

Costruire Elevation Grid (i)

- Usando il nodo **`ElevationGrid`** si specificano le dimensioni le dimensioni della griglia del terreno e l'altezza di ciascun punto della griglia
- Le facce vengono costruite in modo automatico per rappresentare la forma del terreno.
- Nello specificare le dimensioni della griglia l'utente seleziona il numero di righe e di colonne, la spaziatura tra le righe e tra le colonne della stessa.
- Per ciascun punto della griglia vengono fornite le quote lungo l'asse Y
- In una griglia con 5 punti in X (**`xDimension=5`**) e 4 punti in Z **`zDimension=4`** l'altezza del punto P(4,2), che è il quinto punto lungo l'asse x (i=4) ed il terzo lungo Z (j=2), è associata all'elemento **`i+j*xDimension`** della griglia dei valori delle quote (4 + 2* 5= 14)

Il nodo ElevationGrid

- Il nodo **ElevationGrid** crea delle facce e può essere usato come valore del campo **geometry** del nodo **Shape**:

```
ElevationGrid {
        xDimension      0               # field         SFInt32
        xSpacing        0.0             # field         SFFloat
        zDimension      0               # field         SFInt32
        zSpacing        0.0             # field         SFFloat

        height          [ ]             # field         SFFloat
        color           NULL            # exposedField  SFNode
        colorPerVertex  TRUE            # field         SFBool
        normal          NULL            # exposedField  SFNode
        normalPerVertex  TRUE           # field         SFBool
        texCoord        NULL            # exposedField  SFNode
        ccw             TRUE            # field         SFBool
        solid           TRUE            # field         SFBool

        creaseAngle     0.0             # field         SFFloat
        set_height                      # eventIn       MFFloat
  } _
```

Il nodo ElevationGrid

- I valori dei campi **xSpacing** e **zSpacing** specificano la distanza tra le colonne (direzione X) e le righe (direzione Z) della griglia. Il volore di dafault è **0.0** in entrambe le direzioni.
- Il valore del campo **height** specifica la lista delle quote lungo l'asse Y dei punti della griglia. I valori sono elencati per righe di dimensione **zDimension**, ciascuna con **xDimension** valori. I valori delle quote possono essere positivi o negativi. Il valore di default è una lista vuota.
- Quando la forma viene costruita, il primo punto della griglia viene posto nell'origine del sistema di coordinate.
- Il valore del campo **ccw** specifica se le facce devono essere costruite in senso orario (**TRUE**) o antiorario (**FALSE**). Se **ccw** è **TRUE** le facce di fronte all'osservatore guardano in su lungo l'asse Y. Il valore di default è **TRUE** .

Il nodo ElevationGrid (i)

- A seconda del valore del campo `solid` si ha che la rappresentazione del retro delle facce può essere omessa (`TRUE`) o meno per aumentare le prestazioni del browser. Il fronte delle facce viene sempre rappresentato. Il valore di default di `solid` è `TRUE`, che indica la natura solida delle facce, nel qual caso il lato posteriore è nascosto da quello anteriore.
- Il valore del campo `creaseAngle` specifica un angolo di soglia in radianti, come già visto. Il valore deve essere maggiore o uguale a `0.0` . Il valore di default è `0.0`.
- La lista delle quote può essere modificata dinamicamente inviando una lista di quote all'*eventIn* `set_height`.

Il nodo ElevationGrid (ii)

- Il nodo **Appearance** del nodo **Shape** relativo alla forma definita dal nodo **ElevationGrid** che definisce il campo **geometry** del nodo **Shape** stesso, specifica il colore globale con cui viene rappresentata la forma.
- Questo colore può essere modificato per ogni faccia della griglia usando i campi **color** e **colorPerVertex**.
- Il valore dell'exposed-field **color** specifica un nodo che contenga la lista dei colori usati per colorare le linee dell'insieme. Il valore tipico del campo **color** è il nodo **Color**.
- Il valore del campo **colorIndex** specifica una lista di indici di colori da usare nella colorazione delle linee dell'insieme. Ciascun valore è un indice intero che specifica un elemento nella lista del campo **color**. Il valore di default è una lista vuota.

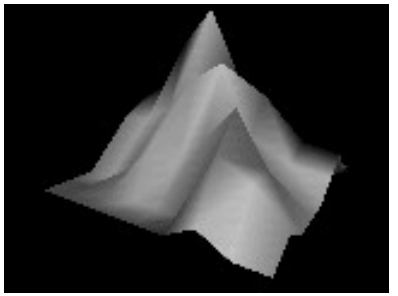
Il nodo ElevationGrid (iii)

- Il campo **colorPerVertex** specifica un valore **TRUE** o **FALSE**, che indica quando i colori della lista del nodo sono utilizzati per ogni spezzata(**FALSE**) o per ciascuna coordinata di ogni spezzata (**TRUE**). Il valore di defalt è **TRUE**.
- Quando il campo **colorIndex** non è vuoto:
  - Se **colorPerVertex** è **FALSE** : indica che ciascun colore della lista viene usato per colorare una spezzata dell'insieme. Devono essere presenti almeno tanti colori, quante sono le spezzate dell'insieme.
  - Se **colorPerVertex** è **TRUE**: indica che ciascun colore della lista viene usato per colorare ciascuna coordinata di ciascuna spezzata dell'insieme. Devono essere presenti almeno tanti colori, quante sono le coordinate, più i valori **-1** per separare le spezzate.

Il nodo ElevationGrid (iv)

- Se il campo **colorIndex** è vuoto, i colori vengono assegnati usando i valori presenti in **coordinateIndex**:
  - Se **colorPerVertex** è **FALSE** : ciascun colore della lista viene usato, nell'ordine per colorare una spezzata dell'insieme. Devono essere presenti almeno tanti colori, quante sono le spezzate dell'insieme.
  - Se **colorPerVertex** è **TRUE**: per assegnare i colori vengono usati i valori degli indici delle coordinate: il primo indice stabilisce la coordinata ed il colore della prima coordinata della prima spezzata dell'insieme, il secondo i valori della seconda coordinata, etc. Devono essere presenti almeno tanti colori, quante sono le coordinate, più i valori **-1** per separare le spezzate.
- I colori delle linee possono essere stabiliti mediante il nodo **Color**, mediante il nodo **Material** o mediante la specifica di entrambi:
  - Se nessuno dei due è specificato, le linee sono bianche
  - Se è specificato solo il nodo **Material**, allora le linee sono rappresentate secondo quanto contenuto nel nodo **Material**.
  - Se è specificato solo il nodo **Color**, allora le linee sono rappresentate secondo quanto contenuto nel nodo **Color**.
  - Se entrambi sono specificati, le caratteristiche vengono definite dal nodo

Esempi: costruzione di una montagna



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (c) 1997
# Andrea L. Ames, David R. Nadeau, and John L.
   Moreland
Shape { appearance Appearance { material Material {
   } }
   geometry ElevationGrid {
       xDimension 9 zDimension 9 xSpacing 1.0
   zSpacing 1.0
       solid FALSE creaseAngle 0.785
       height [ 0.0, 0.0, 0.5, 1.0, 0.5, 0.0, 0.0,
   0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.5, 0.5, 0.0,
   0.0, 0.0, 0.0, 0.0, 0.5, 0.5, 3.0, 1.0, 0.5,
   0.0, 1.0, 0.0, 0.0, 0.5, 2.0, 4.5, 2.5, 1.0,
   1.5, 0.5, 1.0, 2.5, 3.0, 4.5, 5.5, 3.5, 3.0,
   1.0, 0.0, 0.5, 2.0, 2.0, 2.5, 3.5, 4.0, 2.0,
   0.5, 0.0, 0.0, 0.0, 0.5, 1.5, 1.0, 2.0, 3.0,
   1.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0,
   1.5, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5,
   0.0, 0.0, ]
   }
}
```

Scena VRML

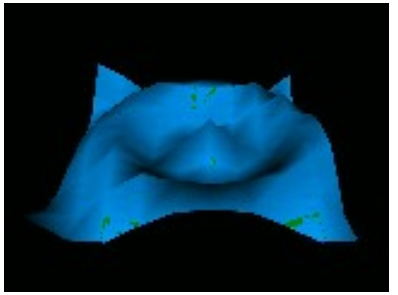
costruzione di superfici

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L.
   Moreland
Shape {
   appearance Appearance {
   material Material {
       diffuseColor 1.0 1.0 0.0
   }
   }
   geometry ElevationGrid {
   xDimension  2
   zDimension  20
   xSpacing    1.0
   zSpacing    4.0
   solid       FALSE
   creaseAngle 0.785
   height [
       0.00,  0.59,  0.95,  0.95,  0.59,
       0.00, -0.59, -0.95, -0.95, -0.59,
       0.00,  0.59,  0.95,  0.95,  0.59,
       0.00, -0.59, -0.95, -0.95, -0.59,
       0.00,  0.59,  0.95,  0.95,  0.59,
       0.00, -0.59, -0.95, -0.95, -0.59,
       0.00,  0.59,  0.95,  0.95,  0.59,
       0.00, -0.59, -0.95, -0.95, -0.59,
   ]
   }
}
```

Scena VRML

costruzione di superfici (i)

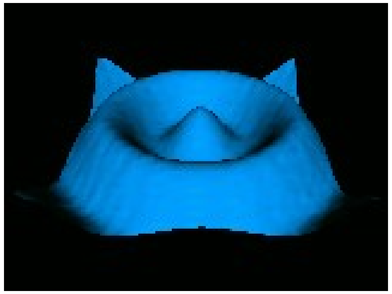
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
         diffuseColor 0.0 0.6 1.0
    }
    }
    geometry ElevationGrid {
    xDimension  10
    zDimension  10
    xSpacing    0.100000
    zSpacing    0.100000
    solid       FALSE
    creaseAngle 3.14
    height [
          0.08,  0.03, -0.05, -0.08, -0.07,
         -0.07, -0.07, -0.08, -0.05,  0.03,
          0.03, -0.06, -0.07, -0.01,  0.04,
          0.05,  0.04, -0.01, -0.07, -0.06,
         -0.05, -0.07,  0.02,  0.08,  0.06,
          0.05,  0.06,  0.08,  0.02, -0.07,
         -0.08, -0.01,  0.08,  0.02, -0.05,
         -0.07, -0.05,  0.02,  0.08, -0.01,
         -0.07,  0.04,  0.06, -0.05, -0.06,
         -0.02, -0.06, -0.05,  0.06,  0.04,
         -0.07,  0.05,  0.05, -0.07, -0.02,
          0.08, -0.02, -0.07,  0.05,  0.05,
         -0.07,  0.04,  0.06, -0.05, -0.06,
         -0.02, -0.06, -0.05,  0.06,  0.04,
         -0.08, -0.01,  0.08,  0.02, -0.05,
         -0.07, -0.05,  0.02,  0.08, -0.01,
         -0.05, -0.07,  0.02,  0.08,  0.06,
          0.05,  0.06,  0.08,  0.02, -0.07,
          0.03, -0.06, -0.07, -0.01,  0.04,
          0.05,  0.04, -0.01, -0.07, -0.06,
    ]
    }
}
```



Scena VRML

costruzione di superfici (ii)

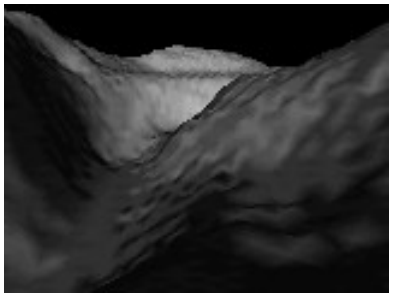
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
        appearance Appearance {
         material Material {
                  diffuseColor 0.0 0.6 1.0
         }
        }
        geometry ElevationGrid {
         xDimension  40
         zDimension  40
         xSpacing    0.025000
         zSpacing    0.025000
         solid       FALSE
         creaseAngle 3.14
         height [
                  0.08, 0.08, 0.07, 0.05, 0.03,
                  0.01, -0.01, -0.03, -0.05, -0.06,
                  [illegible]
                  -0.07, -0.05, -0.02, 0.01, 0.05,
                  [illegible]
                  0.06, 0.07, 0.07, 0.07, 0.07,
                  [illegible]
                  -0.02, 0.00, 0.03, 0.05, 0.06,
         ]
```



Scena VRML

costruzione di superfici (iii)

```
#VRML V2.0 utf8
#  The VRML 2.0 Sourcebook
#  Copyright [1997] By
#  Andrea L. Ames, David R. Nadeau, and John L. Moreland
Transform {
    translation -0.9 0.0 -0.9
    scale 10.0 10.0 10.0
    rotation 0.0 1.0 0.0 1.57
    center 0.9 0.0 0.9
    children Shape {
    appearance Appearance {
         material Material { }
    }
    geometry ElevationGrid {
          xDimension   90
          zDimension   90
          xSpacing     0.02
          zSpacing     0.02
          solid        FALSE
          creaseAngle 0.785
          height [
                0.39,
                0.40,
                0.40,
                0.40,
                0.40,
                0.40,
                0.40,
                0.40,
                0.40,
                0.40,
                0.39,
                0.39,
                0.39,
                0.39,
                0.38,
                0.38,
                0.14,
                0.12,
                0.10,
                0.08,
                0.06,
                0.04,
                0.02,
                0.01,
                0.01,
                0.01,

            ..................................

                0.09,
                0.06,
                0.05,
                0.04,
                0.04,
                0.05,
          ]
    }
    }
}
```



Scena VRML

costruzione di superfici (iv)

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    DEF ThreeRibbon Group {
         children [
              DEF OneRibbon Transform {
                   translation 0.0 0.0 -2.0
                   children Inline {
                        url "ribbon.wrl"
                   }
              },
              Transform {
                   rotation 0.0 1.0 0.0  0.52
                   children USE OneRibbon
              },
              Transform {
                   rotation 0.0 1.0 0.0  1.05
                   children USE OneRibbon
              }
         ]
    },
    Transform {
         rotation 0.0 1.0 0.0  1.57
         children USE ThreeRibbon
    },
    Transform {
         rotation 0.0 1.0 0.0  3.14
         children USE ThreeRibbon
    },
    Transform {
         rotation 0.0 1.0 0.0  4.71
         children USE ThreeRibbon
    }
    ]
}
```

Scena VRML

Virtual Reality Modeling Language

VRML

Associazione di Colori a Punti, Linee, Facce e Coordinate

Il nodo Color

- Usando i nodi `Appearance` e `Material`, si assegna un colore unico ad un'intera forma. Usando il nodo `Color` si possono associare colori diversi ai singoli punti, alle linee, alle facce e alle coordinate di una forma.
- Si possono assegnare pertanto diversi colori ai punti, alle spezzate ed alle facce di nodi `PointSet`, `IndexedLineSet` e `IndexedFaceSet`.
- Quando il browser VRML rappresenta una forma con coordinate colorate, nel passare da una coordinata all'altra cambia il colore della superficie in modo dolce .
- Nel nodo `Color`, vengono elencati i colori da associare alle coordinate di un nodo `Coordinate`.

Il nodo Color (i)

- I colori sono elencati in un nodo mediante terne di valori RGB associate ai singoli colori, eventualmente separati da virgola.
- Come nel nodo `Coordinate`, i colori elencati in un nodo `Color`, vengono implicitamente indicizzati, assegnando l'indice `0` al primo colore, `1` al secondo, etc.
- La colorazione di punti, spezzate e facce avviene assegnando i colori elencati nel nodo `Color`, alle coordinate elencate nel nodo `Coordinate` del campo `coord`. Si possono colorare punti, spezzate e facce diverse in modo diverso.
- Si possono colorare in modo diverso anche i singoli punti di una spezzata o di una faccia, in funzione della specifica che viene fatta nel nodo `Color` del campo `ColorPerVertex`.

Il nodo Color

- Il nodo **Color** crea una lista di colori e può essere usato come valore del campo **color** nei nodi geometrici basati su coordinate, come **PointSet**, **IndexedLineSet**, **IndexedFaceSet** e **ElevationGrid**.

```
Color {
        color       NULL                    # exposedField MFColor
         }
```

- Il valore dell'exposed-field **color** specifica una lista di colori usati per colorare una forma in funzione dei colori assunti dalle varie coordinate.
- Il valore di default è una lista vuota.
- La lista di colori può essere modificata inviando dei valori all'eventIn implicito **set_color** dell'exposed-field **color**. Quando il nuovo valore viene ricevuto, viene aggiornato il valore del campo **color**, il quale viene propagato attraverso l'eventOut **color_changed**.

Color nel nodo PointSet

- Vediamo il ruolo del nodo **Color** quando è usato come valore ddel campo **color** del nodo **PointSet**:

```
PointSet {
        coord       NULL                    # exposedField SFNode
        color       NULL                    # exposedField SFNode
         }
```

- Il valore dell'exposed-field **color** specifica la lista dei colori necessari a colorare i vari punti dell'insieme. Il primo punto del nodo **Coordinate** viene colorato con il primo colore specificato nel nodo **color**, e così via.
- Nel nodo **Color** devono essere presenti almeno tanti colori quante sono le coordinate specificate nel nodo **Coordinate**.
- Il valore di default **NULL** indica una lista vuota.

Color nel nodo IndexedLineSet

- Vediamo il ruolo del nodo **`Color`** quando è usato come valore del campo **`color`** del nodo **`IndexedLineSet`**:

```
IndexedLineSet {
        coord           NULL        # exposedField SFNode
        coordIndex      [ ]         # field        MFInt32
        color           NULL        # exposedField SFNode
        colorIndex      [ ]         # field        MFInt32
        colorPerVertex  TRUE        # field        SFBool
        set_coordIndex              # eventIn      MFInt32
        set_colorIndex              # eventIn      MFInt32

         }
```

Color nel nodo IndexedLineSet

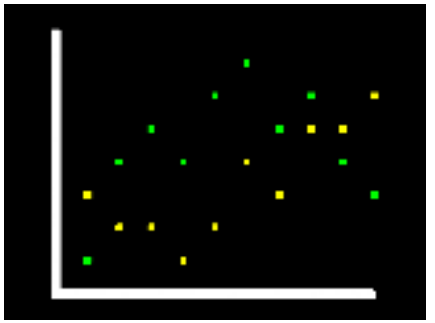
- Il valore dell'exposed-field **color** specifica un nodo che contenga la lista dei colori usati per colorare le linee dell'insieme. Il valore tipico del campo **color** è il nodo **Color**.
- Il valore del campo **colorIndex** specifica una lista di indici di colori da usare nella colorazione delle linee dell'insieme. Ciascun valore è un indice intero che specifica un colore nella lista del campo **color**. Il valore di default è una lista vuota.
- Il campo **colorPerVertex** specifica un valore **TRUE** o **FALSE**, che indica quando i colori della lista del nodo sono utilizzati per ogni spezzata(**FALSE**) o per ciascuna coordinata di ogni spezzata (**TRUE**). Il valore di defalt è **TRUE**.
- Quando il campo **colorIndex** non è vuoto:
  - Se **colorPerVertex** è **FALSE** : indica che ciascun colore della lista viene usato per colorare una spezzata dell'insieme. Devono essere presenti almeno tanti colori, quante sono le spezzate dell'insieme.
  - Se **colorPerVertex** è **TRUE**: indica che ciascun colore della lista viene usato per colorare ciascuna coordinata di ciascuna spezzata dell'insieme. Devono essere presenti almeno tanti colori, quante sono le coordinate, compresi i valori **-1** per separare le spezzate.

Color nel nodo IndexedLineSet

- Se il campo **colorIndex** è vuoto, sono usati i valori presenti in **coordinateIndex** per assegnare i colori:
  - Se **colorPerVertex** è **FALSE** : ciascun colore della lista viene usato per colorare una spezzata dell'insieme, nell'ordine. Devono essere presenti almeno tanti colori, quante sono le spezzate dell'insieme.
  - Se **colorPerVertex** è **TRUE**: per assegnare i colori vengono usati i valori degli indici delle coordinate: il primo indice stabilisce la coordinata ed il colore della prima coordinata della prima spezzata dell'insieme, il secondo i valori della seconda coordinata, etc. Devono essere presenti almeno tanti colori, quante sono le coordinate, compresi i valori **-1** per separare le spezzate.
- I colori delle linee possono essere stabiliti mediante il nodo **Color**, mediante il nodo **Material** o mediante la specifica di entrambi:
  - Se nessuno dei due è specificato, le linee sono bianche
  - Se è specificato solo il nodo **Material**, allora le linee sono rappresentate secondo quanto contenuto nel nodo **Material**.
  - Se è specificato solo il nodo **Color**, allora le linee sono rappresentate secondo quanto contenuto nel nodo **Color**.
  - Se entrambi sono specificati, le caratteristiche vengono definite dal nodo **Material** ad eccezione del colore che è definito dal nodo **Color**

Esempi: coloring PointSet

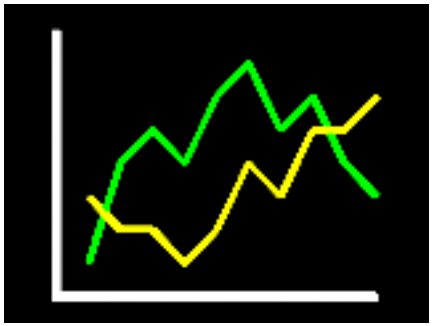
```
#VRML V2.0 utf8
Group {
    children [
    # Axes
        Shape {
            appearance Appearance {
                material Material {
                    emissiveColor 1.0 1.0 1.0
                    diffuseColor  1.0 1.0 1.0
                }
            }
            geometry IndexedLineSet {
                coord Coordinate {
                    point [
                        0.0 0.0 0.0, 10.0 0.0 0.0, 0.0 8.0 0.0
                    ]
                }
                coordIndex [ 0, 1, -1,  0, 2, -1 ]
            }
        },
    # Scatter plot with different color points
        Shape {
            # no appearance, use emissive coloring
            geometry PointSet {
                coord Coordinate {
                    point [
                    # Green points
                        1.0 1.0 0.0,   2.0 4.0 0.0,
                        3.0 5.0 0.0,   4.0 4.0 0.0,
                        5.0 6.0 0.0,   6.0 7.0 0.0,
                        7.0 5.0 0.0,   8.0 6.0 0.0,
                        9.0 4.0 0.0,  10.0 3.0 0.0,
                    # Yellow points
                        1.0 3.0 0.0,   2.0 2.0 0.0,
                        3.0 2.0 0.0,   4.0 1.0 0.0,
                        5.0 2.0 0.0,   6.0 4.0 0.0,
                        7.0 3.0 0.0,   8.0 5.0 0.0,
                        9.0 5.0 0.0,  10.0 6.0 0.0,
                    ]
                }
                color Color {
                    color [
                    # Green points
                        0.0 1.0 0.0,   0.0 1.0 0.0,
                        0.0 1.0 0.0,   0.0 1.0 0.0,
                        0.0 1.0 0.0,   0.0 1.0 0.0,
                        0.0 1.0 0.0,   0.0 1.0 0.0,
                        0.0 1.0 0.0,   0.0 1.0 0.0,
                    # Yellow points
                        1.0 1.0 0.0,   1.0 1.0 0.0,
                        1.0 1.0 0.0,   1.0 1.0 0.0,
                        1.0 1.0 0.0,   1.0 1.0 0.0,
                        1.0 1.0 0.0,   1.0 1.0 0.0,
                        1.0 1.0 0.0,   1.0 1.0 0.0,
                    ]
                }
            }
        }
    ]
}
```



scena VRML

Esempi: coloring lines

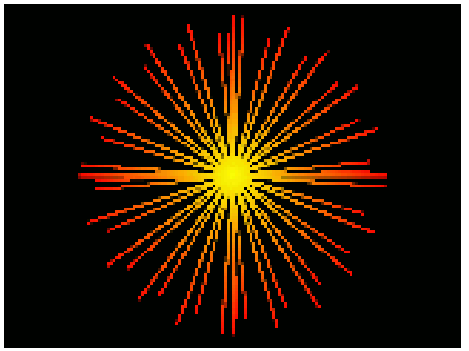
```
#VRML V2.0 utf8
Group {
    children [
    # Axes
        Shape {
            appearance Appearance {
                material Material {
                    emissiveColor 1.0 1.0 1.0
                    diffuseColor  1.0 1.0 1.0
                }
            }
            geometry IndexedLineSet {
                coord Coordinate {
                    point [
                        0.0 0.0 0.0, 10.0 0.0 0.0, 0.0 8.0 0.0
                    ]
                }
                coordIndex [ 0, 1, -1,  0, 2, -1 ]
            }
        },
    # Line plot with different color lines
        Shape {
            # no appearance, use emissive coloring
            geometry IndexedLineSet {
                coord Coordinate {
                    point [
                    # Green line
                        1.0 1.0 0.0,   2.0 4.0 0.0,
                        3.0 5.0 0.0,   4.0 4.0 0.0,
                        5.0 6.0 0.0,   6.0 7.0 0.0,
                        7.0 5.0 0.0,   8.0 6.0 0.0,
                        9.0 4.0 0.0,  10.0 3.0 0.0,
                    # Yellow line
                        1.0 3.0 0.0,   2.0 2.0 0.0,
                        3.0 2.0 0.0,   4.0 1.0 0.0,
                        5.0 2.0 0.0,   6.0 4.0 0.0,
                        7.0 3.0 0.0,   8.0 5.0 0.0,
                        9.0 5.0 0.0,  10.0 6.0 0.0,
                    ]
                }
                coordIndex [
                # Green line
                    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -1,
                # Yellow line
                    10, 11, 12, 13, 14, 15, 16, 17, 18, 19, -1
                ]
                colorPerVertex FALSE
                color Color {
                    color [ 0.0 1.0 0.0,   1.0 1.0 0.0 ]
                }
                colorIndex [ 0, 1 ]
            }
        }
    ]
}
```



scena VRML

Esempi: coloring coordinates

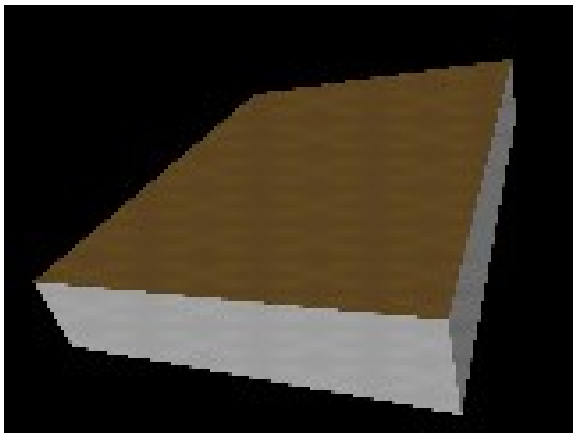
```
#VRML V2.0 utf8
Group {
    children [
        DEF Burst Shape {
            # no appearance, use emissive coloring
            geometry IndexedLineSet {
                coord Coordinate {
                    point [
                         0.00  0.00  0.00,   1.00  0.00  0.00,
                         0.92  0.38  0.00,   0.71  0.71  0.00,
                         0.38  0.92  0.00,   0.00  1.00  0.00,
                        -0.38  0.92  0.00,  -0.71  0.71  0.00,
                        -0.92  0.38  0.00,  -1.00  0.00  0.00,
                        -0.92 -0.38  0.00,  -0.71 -0.71  0.00,
                        -0.38 -0.92  0.00,   0.00 -1.00  0.00,
                         0.38 -0.92  0.00,   0.71 -0.71  0.00,
                         0.92 -0.38  0.00,
                    ]
                }
                coordIndex [
                    0,  1, -1,   0,  2, -1,
                    0,  3, -1,   0,  4, -1,
                    0,  5, -1,   0,  6, -1,
                    0,  7, -1,   0,  8, -1,
                    0,  9, -1,   0, 10, -1,
                    0, 11, -1,   0, 12, -1,
                    0, 13, -1,   0, 14, -1,
                    0, 15, -1,   0, 16, -1
                ]
                colorPerVertex TRUE
                color Color {
                    color [
                        1.0 1.0 0.0,  # burst center color
                        1.0 0.3 0.3   # burst ends color
                    ]
                }
                colorIndex [
                    0,  1,  0,   0,  1,  0,
                    0,  1,  0,   0,  1,  0,
                    0,  1,  0,   0,  1,  0,
                    0,  1,  0,   0,  1,  0,
                    0,  1,  0,   0,  1,  0,
                    0,  1,  0,   0,  1,  0,
                    0,  1,  0,   0,  1,  0,
                    0,  1,  0,   0,  1,  0,
                ]
            }
        },
        Transform { rotation 0.0 1.0 0.0  0.785 children USE Burst },
        Transform { rotation 0.0 1.0 0.0  1.57  children USE Burst },
        Transform { rotation 0.0 1.0 0.0  2.355 children USE Burst }
    ]
}
```



Scena VRML

Esempi: coloring faces

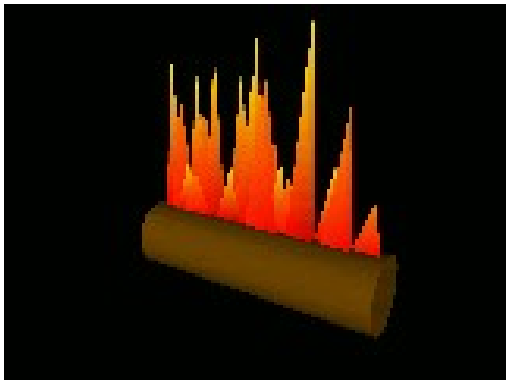
```
#VRML V2.0 utf8
Shape {
    appearance Appearance {
        material Material { }
    }
    geometry IndexedFaceSet {
        coord Coordinate {
            point [
            # Around the front of the book
                -0.095 -0.115 0.04,   0.095 -0.115 0.04,
                 0.095  0.115 0.04,  -0.095  0.115 0.04,
            # Around the back of the book
                -0.095 -0.115 0.00,   0.095 -0.115 0.00,
                 0.095  0.115 0.00,  -0.095  0.115 0.00
            ]
        }
        coordIndex [
        # Cover front, back, and spine cover (brown)
            0, 1, 2, 3, -1,  7, 6, 5, 4, -1,  0, 3, 7, 4, -1,
        # Paper bottom, right, and top edges (white)
            0, 4, 5, 1, -1,  1, 5, 6, 2, -1,  2, 6, 7, 3
        ]
        colorPerVertex FALSE
        color Color {
            color [
                0.7 0.5 0.2, # cover
                0.8 0.8 0.8  # paper edges
            ]
        }
        colorIndex [
            0, 0, 0, # cover
            1, 1, 1  # paper edges
        ]
    }
}
```



Scena VRML

Esempi: coloring coordinates in face sets

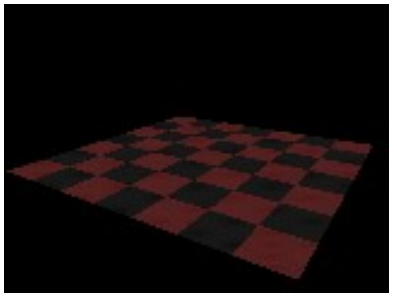
```
#VRML V2.0 utf8
Group {
    children [
    # A log
        Transform {
            translation 0.0 -0.4 0.0
            rotation    0.0 0.0 1.0 -1.57
            children Shape {
                appearance Appearance {
                    material Material {
                        diffuseColor 0.5 0.3 0.0
                    }
                }
                geometry Cylinder {
                    height 2.9
                    radius 0.4
                }
            }
        },
    # A set of flames
        DEF Flames Shape {
            # No appearance, use emissive coloring
            geometry IndexedFaceSet {
                coord Coordinate {
                    point [
                        -0.7 0.0 0.0,  -0.8 1.5 0.0,  -1.0 0.0 0.0,
                        -0.5 0.0 0.01, -0.7 1.2 0.01, -0.9 0.0 0.01,
                        -0.1 0.0 0.0,  -0.2 1.6 0.0,  -0.4 0.0 0.0,
                         0.3 0.0 0.01,  0.2 1.0 0.01,  0.0 0.0 0.0,
                    ]
                }
                coordIndex [
                    0, 1, 2, -1,  3,  4,  5, -1,
                    6, 7, 8, -1,  9, 10, 11, -1
                ]
                solid FALSE
                colorPerVertex TRUE
                color Color {
                    color [
                        1.0 0.0 0.0,  1.0 0.5 0.0,  1.0 0.1 0.0,
                        0.8 0.0 0.0,  1.0 0.9 0.0,  1.0 0.0 0.0,
                    ]
                }
                colorIndex [
                    3, 4, 5, 0,   0, 1, 2, 0,
                    3, 4, 5, 0,   0, 1, 2, 0
                ]
            }
        },
```



```
# Repeat the flames to make a roaring fire
        Transform {
            translation 0.8 0.0 0.02
            scale 1.0 1.3 1.0
            children USE Flames
        },
        Transform {
            translation 1.1 0.0 0.04
            scale 1.0 0.5 1.0
            children USE Flames
        },
        Transform {
            translation -0.3 0.0 0.06
            scale 1.0 1.1 1.0
            children USE Flames
        },
        Transform {
            translation -0.1 0.0 0.08
            scale 1.0 0.4 1.0
            children USE Flames
        },
        Transform {
            translation 0.8 0.0 0.10
            scale 1.0 1.1 1.0
            children USE Flames
        }
    ]
}
```

Scena VRML

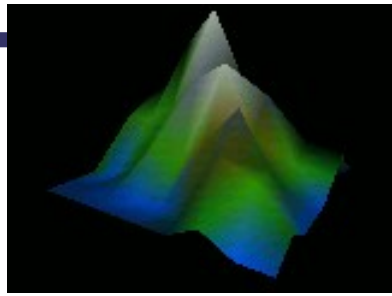
Scacchiera



```
#VRML V2.0 utf8

# The VRML 2.0 Sourcebook

# Copyright (c) 1997

# Andrea L. Ames, David R. Nadeau, and John L. Moreland

Shape { appearance Appearance { material Material { } }

        geometry ElevationGrid { xDimension 8 zDimension 8 xSpacing 1.0 zSpacing 1.0 solid FALSE height
   [ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
   0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
   0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
   0.0, 0.0, 0.0, 0.0, 0.0, ]

   colorPerVertex FALSE

   color DEF check Color {

   color [ 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3,
   0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3
   0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0
   0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2,
   1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2
   0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2
   0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, 1.0 0.3 0.3, 0.2 0.2 0.2, ]

    }

  }

}
```

Scena VRML

Montagna

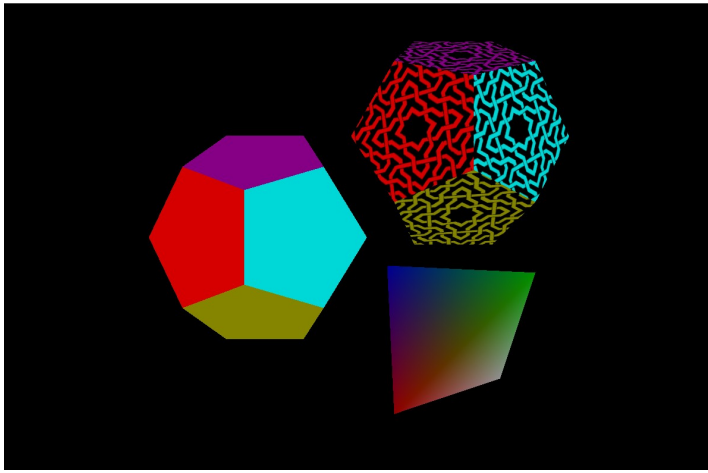


```
#VRML V2.0 utf8

# The VRML 2.0 Sourcebook

# Copyright (c) 1997 # Andrea L. Ames, David R. Nadeau, and John L. Moreland

Shape { appearance Appearance { material Material { } } geometry ElevationGrid {

   xDimension 9 zDimension 9 xSpacing 1.0 zSpacing 1.0 solid FALSE creaseAngle 0.785

   height [ 0.0, 0.0, 0.5, 1.0, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.5, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.5,
    3.0, 1.0, 0.5, 0.0, 1.0, 0.0, 0.0, 0.5, 2.0, 4.5, 2.5, 1.0, 1.5, 0.5, 1.0, 2.5, 3.0, 4.5, 5.5, 3.5, 3.0, 1.0, 0.0, 0.5,
    2.0, 2.0, 2.5, 3.5, 4.0, 2.0, 0.5, 0.0, 0.0, 0.0, 0.5, 1.5, 1.0, 2.0, 3.0, 1.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0,
    1.5, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0, ]

   colorPerVertex TRUE color Color {

   color [ 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.5 0.1, 0.2 0.6 0.0, 0.0 0.5 0.1, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3
    1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.5 0.4 0.0, 0.0 0.5 0.1, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3
    1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.5 0.1, 0.0 0.5 0.1, 0.5 0.4 0.0, 0.2 0.6 0.0, 0.0 0.5 0.1, 0.0 0.3 1.0, 0.2 0.6
    0.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.5 0.1, 0.4 0.3 0.1, 0.7 0.7 0.7, 0.5 0.4 0.0, 0.2 0.6 0.1, 0.3 0.6 0.6, 0.0 0.5
    0.1, 0.2 0.6 0.0, 0.5 0.4 0.0, 0.5 0.4 0.0, 0.7 0.7 0.7, 0.8 0.8 0.8, 0.5 0.5 0.7, 0.5 0.5 0.7, 0.2 0.6 0.0, 0.0 0.3
    1.0, 0.0 0.5 0.1, 0.2 0.6 0.1, 0.2 0.6 0.1, 0.2 0.6 0.1, 0.5 0.5 0.7, 0.7 0.7 0.7, 0.5 0.4 0.0, 0.0 0.5 0.1, 0.0 0.3
    1.0, 0.0 0.5 0.1, 0.0 0.3 1.0, 0.0 0.5 0.1, 0.2 0.6 0.1, 0.2 0.6 0.0, 0.5 0.4 0.0, 0.5 0.5 0.7, 0.2 0.6 0.0, 0.0 0.3
    1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.5 0.1, 0.5 0.4 0.0, 0.2 0.6 0.0, 0.0 0.5
    0.1, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.3 1.0, 0.0 0.5 0.1, 0.0 0.3 1.0, 0.0 0.3
    1.0, ]

    }

  }

}
```

Scena VRML

Facce colorate

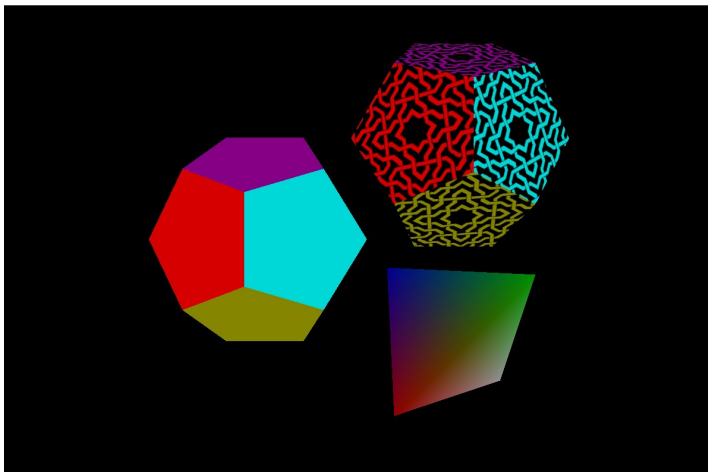
```
#VRML V2.0 utf8
# Three IndexedFaceSets, showing:
#   - Color applied per-face, indexed
#   - Color applied per-vertex
#   - Texture coordinates applied per-vertex
# A dodecahedron: 20 vertices, 12 faces.
# Six colors (primaries:RGB and complements:CMY) mapped to the faces.
Transform {
  translation -1.5 0 0
  children Shape {
    appearance DEF A Appearance { material Material { } }
    geometry DEF IFS IndexedFaceSet {
      coord Coordinate {
        point [  # Coordinates and indices derived from book "Jim Blinn's Corner"
          1 1 1, 1 1 -1, 1 -1 1, 1 -1 -1,
          -1 1 1, -1 1 -1, -1 -1 1, -1 -1 -1,
          .618 1.618 0, -.618 1.618 0, .618 -1.618 0, -.618 -1.618 0,
          1.618 0 .618, 1.618 0 -.618, -1.618 0 .618, -1.618 0 -.618,
          0 .618 1.618, 0 -.618 1.618, 0 .618 -1.618, 0 -.618 -1.618
        ]
      }
      coordIndex [
        1 8 0 12 13 -1,  4 9 5 15 14 -1,  2 10 3 13 12 -1,  7 11 6 14 15 -1,
        2 12 0 16 17 -1,  1 13 3 19 18 -1,  4 14 6 17 16 -1,  7 15 5 18 19 -1,
        4 16 0 8 9 -1,  2 17 6 11 10 -1,  1 18 5 9 8 -1,  7 19 3 10 11 -1,
      ]

      color Color {  # Six colors:
        color [ 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0 ]
      }
      colorPerVertex FALSE  # Applied to faces, not vertices
      # This indexing gives a nice symmetric appearance:
      colorIndex [ 0, 1, 1, 0, 2, 3, 3, 2, 4, 5, 5, 4 ]

      # Five texture coordinates, for the five vertices on each face.
      # These will be re-used by indexing into them appropriately.
      texCoord TextureCoordinate {
        point [  # These are the coordinates of a regular pentagon:
          0.654508 0.0244717,  0.0954915 0.206107
          0.0954915 0.793893,  0.654508 0.975528, 1 0.5,
        ]
      }
      # And this particular indexing makes a nice image:
      texCoordIndex [
        0 1 2 3 4 -1,  2 3 4 0 1 -1,  4 0 1 2 3 -1,  1 2 3 4 0 -1,
        2 3 4 0 1 -1,  0 1 2 3 4 -1,  1 2 3 4 0 -1,  4 0 1 2 3 -1,
        4 0 1 2 3 -1,  1 2 3 4 0 -1,  0 1 2 3 4 -1,  2 3 4 0 1 -1,
       ]

    }
  }
}
```



scena VRML

Facce colorate

```
# A tetrahedron, with a color at each vertex:
Transform {
  translation 1.5 -1.5 0
  children Shape {
    appearance USE A  # Use same default material as dodecahedron
    geometry IndexedFaceSet {
      coord Coordinate {
        point [  # Coordinates and indices derived from book "Jim Blinn's Corner"
          1 1 1, 1 -1 -1, -1 1 -1, -1 -1 1,
        ]
      }
      coordIndex [
        3 2 1 -1,  2 3 0 -1,  1 0 3 -1,  0 1 2 -1,
      ]
      color Color {  # Four colors:
        color [ 0 1 0, 1 1 1, 0 0 1, 1 0 0 ]
      }
     # Leave colorPerVertex field set to TRUE.
      # And no indices are needed, either-- each coordinate point
      # is assigned a color (or, to think of it another way, the same
      # indices are used for both coordinates and colors).
    }
  }
}

# The same dodecahedron, this time with a texture applied.
# The texture overrides the face colors given.
Transform {
  translation 1.5 1.5 0
  children Shape {
    appearance Appearance {
      texture ImageTexture { url "CELTIC.GIF" }
      material Material { }
    }
    geometry USE IFS
  }
}
```



scena VRML

Virtual Reality Modeling Language

VRML

Costruzione di Forme Estruse

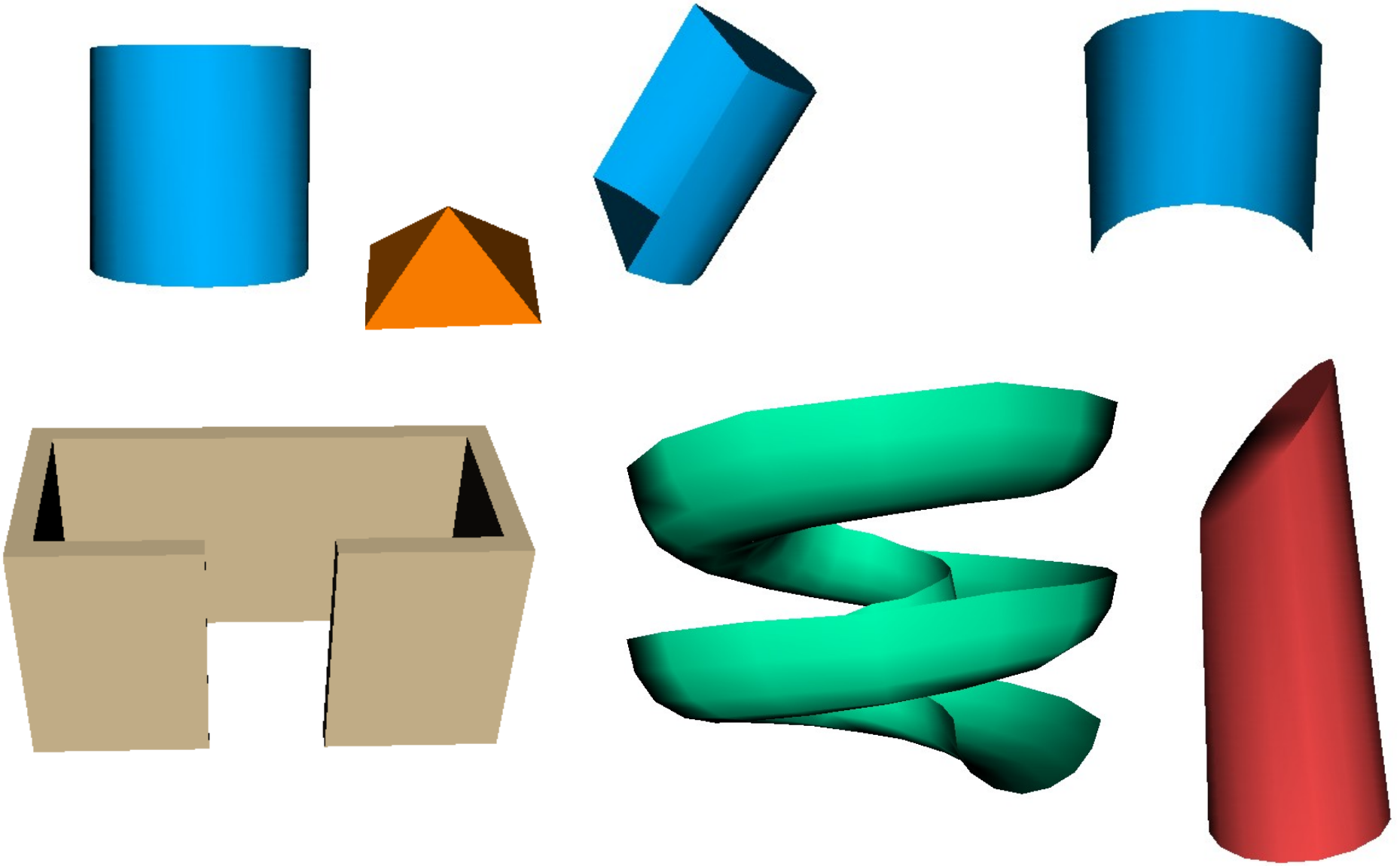
Forme estruse

- L'estrusione è un processo del mondo reale che forza il passaggio di materiale grezzo entro un piatto con un foro: ne risulta una forma allungata avente la sezione coincidente con la forma del foro.
- Queste forme vengono prodotte in modo analogo a come vengono lavorati gli insaccati di maiale (salsiccie, salami), forzando la carne macinata a passare entro un canale nel quale viene avvolto il budello contenitore.
- Il nodo **`Extrusion`** consente di riprodurre questo processo del mondo reale per produrre forme di vario tipo. Viene specificata una forma 2D (cross-section) che funge da guida per il processo di estrusione. La guida viene poi propagata lunga una curva 3D (spine) in modo da creare la geometria estrusa per il nodo **`Shape`**.

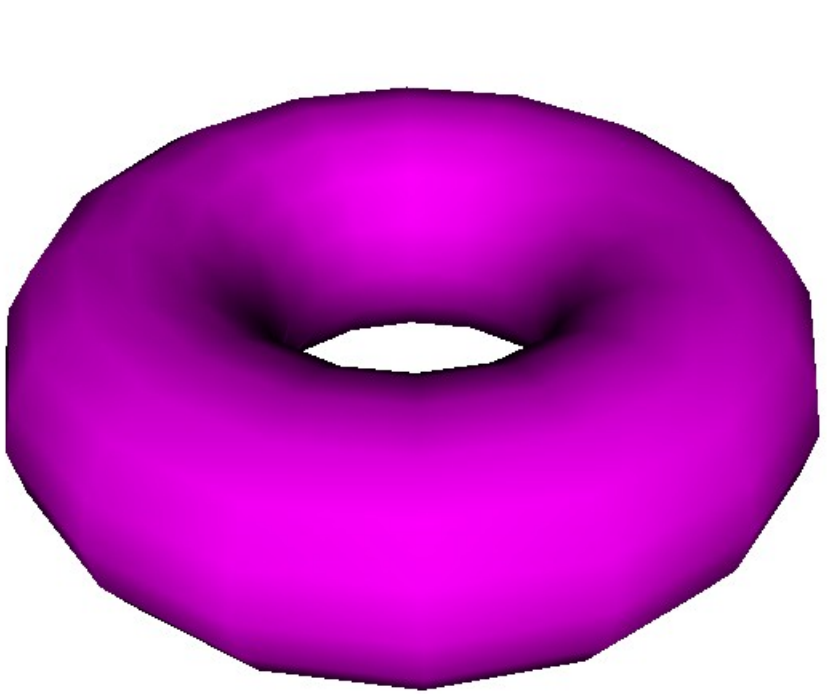
Nodo Extrusion

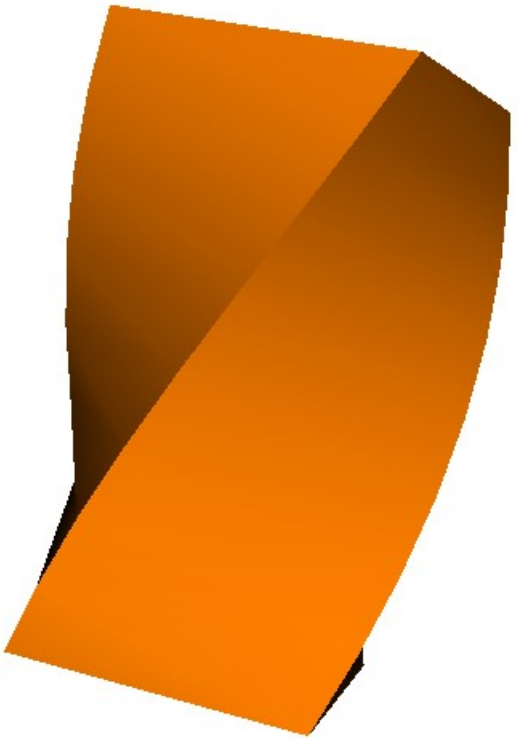
- I nodi **Extrusion** ed **ElevationGrid** sono gli unici nodi aggiunti nella specifica VRML 2.0 rispetto a quella del VRML 1.0.
- **Extrusion** è un nodo molto utilizzato perché consente di creare molti tipi di forme ed è estremamente più leggero del nodo **IndexedFaceSet**.
- I nodi **Extrusion** sono molto più convenienti da utilizzare rispetto ad **IndexedFaceSet** perché essendo nota la topologia della forma, è più semplice la generazione delle normali alle superfici.
- Anche le coordinate di texture sono più semplici da calcolare e non possono essere specificate dall'utente.
- Per il completo controllo del texture mapping occorre utilizzare il nodo **IndexedFaceSet**.

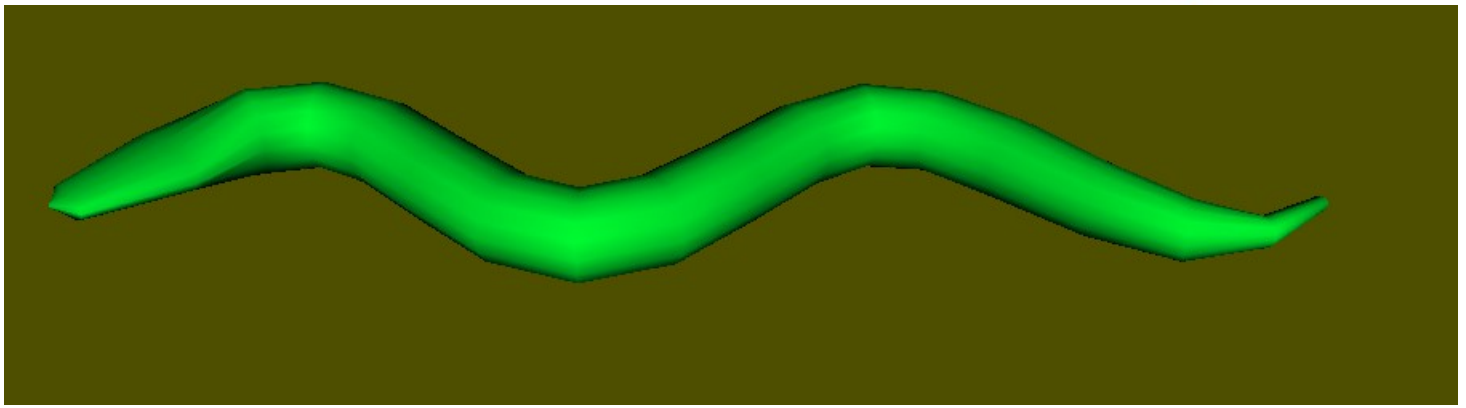
Esempi di forme estruse



Esempi di forme estruse







Cross-section

- La cross-section è una forma 2D, ad es. un cerchio, un semicerchio, un arco, un quadrato.
- Si specifica il perimetro della forma piana, mediante un insieme di coordinate 2D, come si fa per specificare le coordinate di una faccia nel nodo **`IndexFaceSet`**.
- Queste coordinate sono specificate nel campo **`crossSection`** del nodo **`Extrusion`**.
- Una cross-section può essere una forma aperta o chiusa. Una forma chiusa inizia e termina con la stessa coordinata 2D, come nel caso del cerchio. Una forma aperta inizia e finisce in coordinate differenti.

Spine

- Una spine è un percorso 3D che guida il processo di estrusione, come ad es. una linea retta o una curva.
- Analogamente ad una polyline del nodo **`IndexedFaceSet`** una spine viene definita da un insieme di coordinate 3D.
- Le coordinate 3D vengono specificate nel campo **`spine`** del nodo **`Extrusion`**.
- Una spine può essere chiusa (inizia e finisce con la stessa coordinata 3D) o aperta (inizia e finisce in coordinate 3D differenti), come nel caso di una spirale.

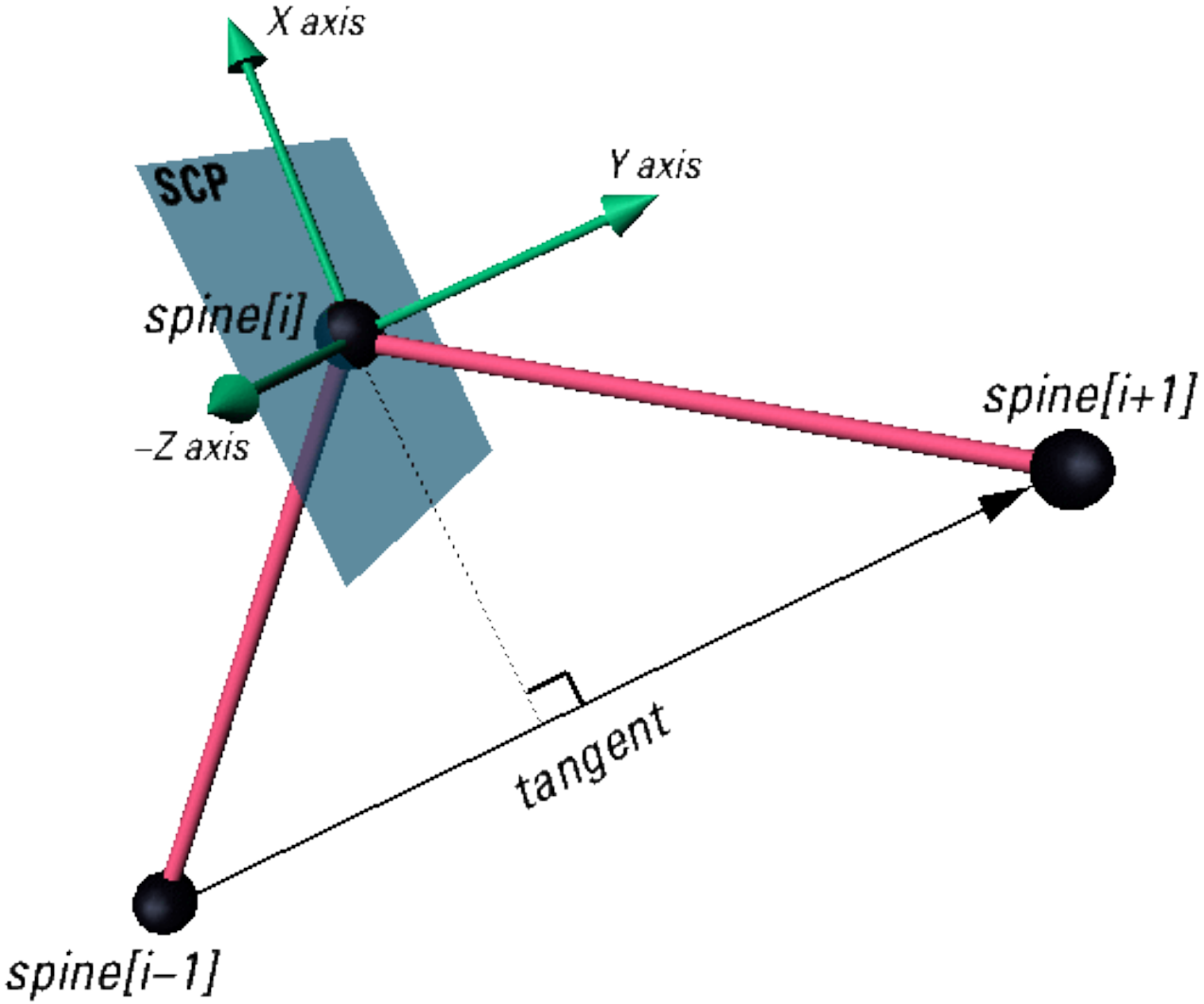
Estrusione

- Per default VRML costruisce le facce all'inizio ed alla fine della forma estrusa (caps).
- Il processo di estrusione procede per tappe:
  1. La forma della cross-section viene posta nella prima coordinata della spine
  2. Una copia della cross-section viene posta nella coordinata successiva della spine
  3. Vengono disegnate le facce che uniscono la prima cross-section alla seconda, creando la porzione relativa di forma estrusa.
  4. I passi 2 e 3 vengono ripetuti fino alla fine della forma spine.
- Ogni volta che la cross-section viene posta nella successiva coordinata della spine, si può usare il campo `scale` per alterare la forma estrusa. Si può scalare verso l'alto o verso il basso.

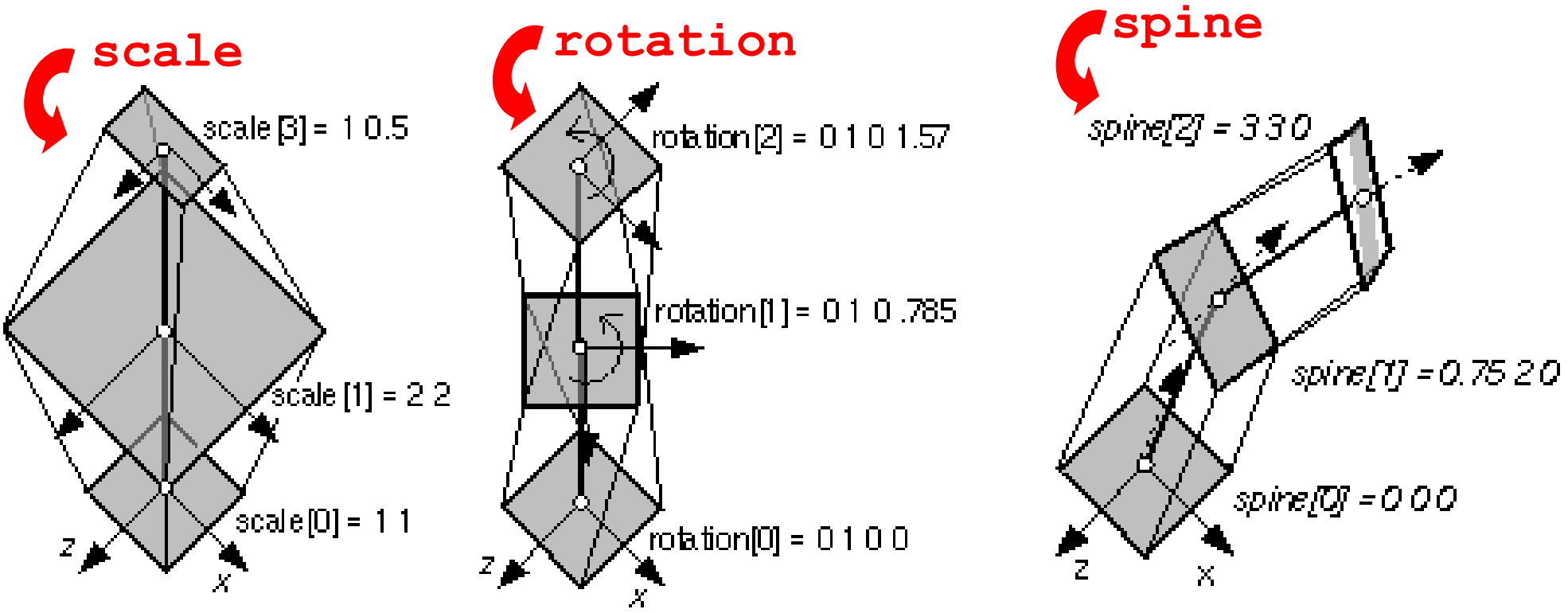
Orientamento della cross-section

- Durante il processo di estrusione la cross-section viene orientata automaticamente in modo da assecondare il percorso della spine
- Questa orientazione può essere arbitrariamente modificata per alterare il processo di estrusione.
- La rotazione della cross-section viene indicata mediante un asse di rotazione.
- La rotazione della cross-section può avvenire su qualsiasi asse e per qualsiasi angolo.

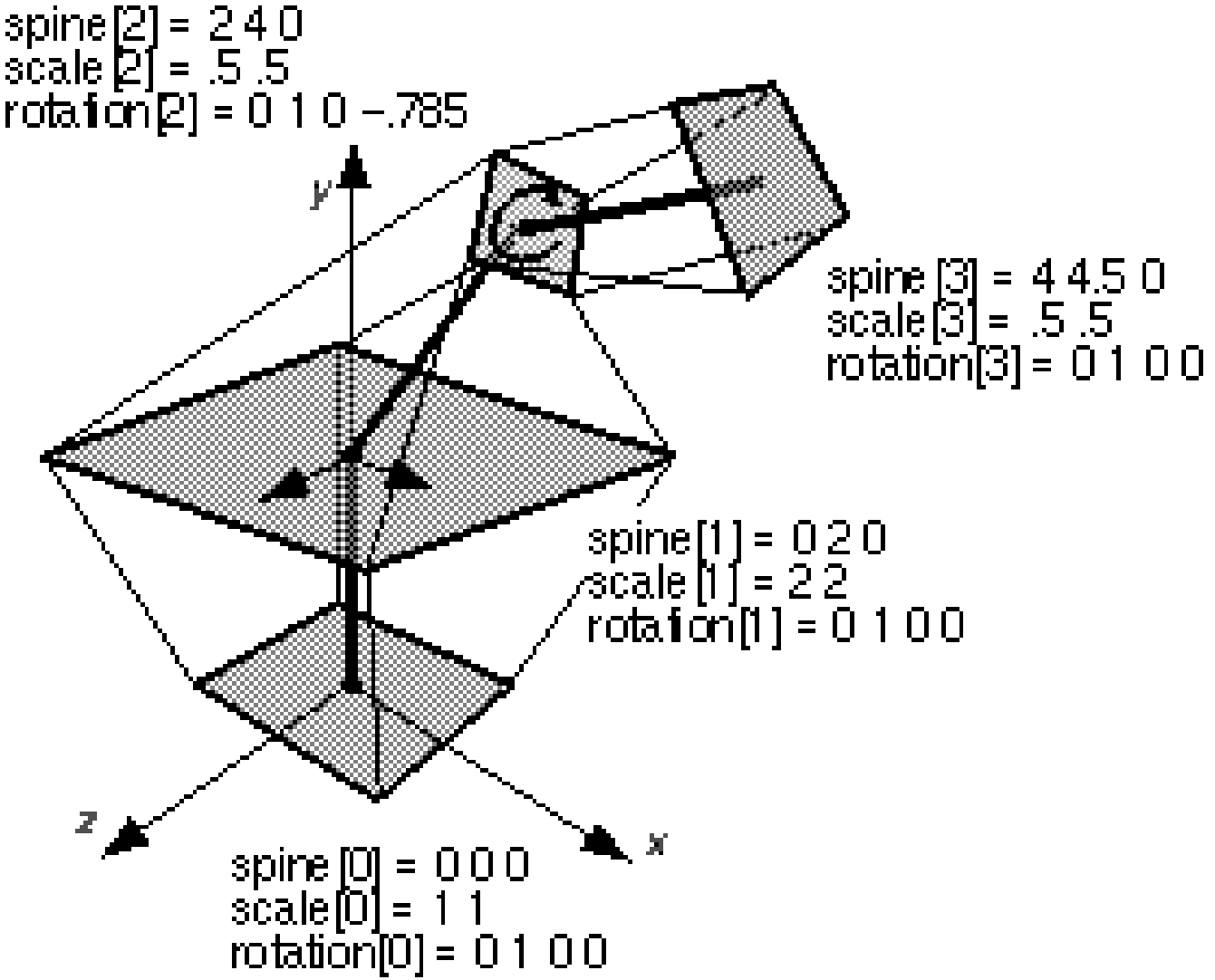
Spine-aligned cross-section plane (SCP)



Esempio di uso di `scale rotation spine`



Esempio di uso di `scale rotation spine`



Il nodo Extrusion

- Il nodo **Extrusion** crea delle facce e può essere usato come valore del campo **geometry** del nodo **Shape**:

```
Extrusion {
        crossSection [ 1.0 1.0,
                       1.0, -1.0,
                      -1.0, -1.0,
                      -1.0, 1.0,
                       1.0, 1.0]          # field          MFVec2f
        spine         [ 0.0, 0.0, 0.0,
                        0.0, 1.0, 0.0]    # field          MFVec3f
        scale           1.0 1.0           # field          MFVec2f
        orientation     0.0 0.0 1.0 0.0    # field          MFRotation
        beginCap        TRUE              # field          SFBool
        endCap          TRUE              # field          SFBool
        ccw             TRUE              # field          SFBool
        solid           TRUE              # field          SFBool
        convex          TRUE              # field          SFBool
        creaseAngle     0.0               # field          SFFloat
        set_spine                         # eventIn        MFVec3f
        set_crossSection                  # eventIn        MFVec2f
        set_scale                         # eventIn        MFVec2f
        set_orientation                    # eventIn        MFRotation
         }
```

Il nodo Extrusion

- Il valore del campo `crossSection` specifica una lista di coordinate 2D che definisce una forma aperta o chiusa attraverso cui avviene il processo di estrusione.
- Le coordinate 2D sono relative al piano XZ, in modo che l'estrusione proceda lungo l'asse Y.
- Il valore di default per il campo `crossSection` è un quadrato.
- Il valore del campo `spine` specifica una lista di coordinate 3D ce definiscono un percorso aperto o chiuso attraverso cui evolve il processo di estrusione.
- Il valore di default è un percorso rettilineo lungo la direzione positiva dell'asse Y

Il nodo Extrusion (i)

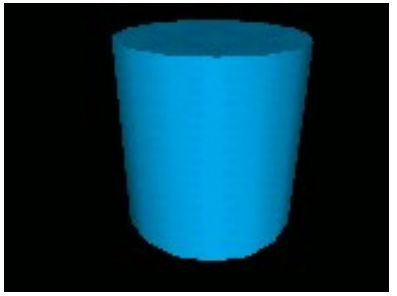
- Il valore del campo `Scale` specifica una lista di coppie di fattori di scala della cross-section. Il primo valore specifica il fattore di scala in X, mentre il secondo il fattore di scala in Z.
- Valori tra `0.0` e `1.0` producono una riduzione della cross-section, mentre valori superiori a `1.0` producono un ingrandimento della cross-section.
- Un fattore di scala pari a `1.0` lascia inalterata la cross-section.
- Il campo `scale` deve contenere una coppia di valori di scala o tante coppie quante sono le coordinate specificate per la spine.

Il nodo Extrusion (ii)

- I campi `beginCap` e `endCap` specificano un valore `TRUE` o `FALSE` e stanno ad indicare quando devono essere disegnate le facce all'inizio e alla fine della forma estrusa
- Quando il valore è `TRUE`, viene disegnata una forma piatta coincidente con le coordinate 2D del campo `crossSection`.
- Quando la cross-section specifica una forma aperta, la faccia viene realizzata connettendo l'ultima coordinata alla prima.
- Quando il valore è `FALSE`, la corrispondente faccia non viene disegnata. Il default è `TRUE`.

Forme estruse: cilindro

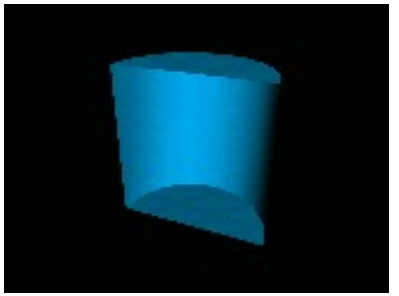
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
         diffuseColor 0.0 0.7 1.0
    }
    }
    geometry Extrusion {
    creaseAngle 0.785
    crossSection [
          1.00  0.00,    0.92 -0.38,
          0.71 -0.71,    0.38 -0.92,
          0.00 -1.00,   -0.38 -0.92,
         -0.71 -0.71,   -0.92 -0.38,
         -1.00 -0.00,   -0.92  0.38,
         -0.71  0.71,   -0.38  0.92,
          0.00  1.00,    0.38  0.92,
          0.71  0.71,    0.92  0.38,
          1.00  0.00
    ]
    spine [ 0.0 -1.0 0.0,  0.0 1.0 0.0 ]
    }
}
```



Scena VRML

Forme estruse: tronco di cilindro

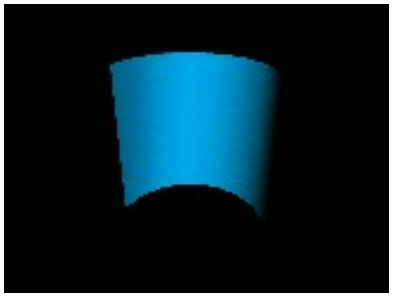
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
         diffuseColor 0.0 0.7 1.0
    }
    }
    geometry Extrusion {
    solid FALSE
    creaseAngle 0.785
    crossSection [
          1.00  0.00,    0.92 -0.38,
          0.71 -0.71,    0.38 -0.92,
          0.00 -1.00,   -0.38 -0.92,
         -0.71 -0.71,   -0.92 -0.38,
         -1.00 -0.00,
    ]
    spine [ 0.0 -1.0 0.0,  0.0 1.0 0.0 ]
    }
}
```



Scena VRML

beginCap, endCap

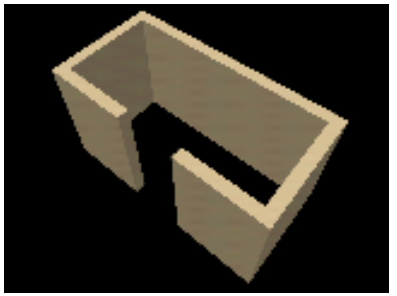
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
         diffuseColor 0.0 0.7 1.0
    }
    }
    geometry Extrusion {
    solid FALSE
    beginCap FALSE
    endCap  FALSE
    creaseAngle 0.785
    crossSection [
          1.00  0.00,    0.92 -0.38,
          0.71 -0.71,    0.38 -0.92,
          0.00 -1.00,   -0.38 -0.92,
         -0.71 -0.71,   -0.92 -0.38,
         -1.00 -0.00,
    ]
    spine [ 0.0 -1.0 0.0,  0.0 1.0 0.0 ]
    }
}
```



Scena VRML

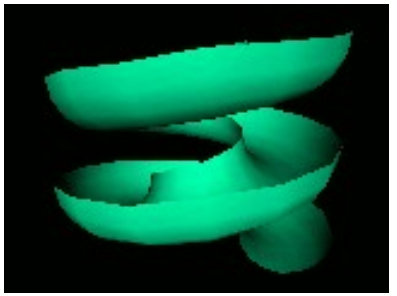
Forme estruse: pareti

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor 1.0 0.9 0.7
            }

        }
        geometry Extrusion {
            convex FALSE
            crossSection [
            # Room outline
                -0.5  1.0,  -0.5  0.8,
                -1.8  0.8,  -1.8 -0.8,
                 1.8 -0.8,   1.8  0.8,
                 0.5  0.8,   0.5  1.0,
                 2.0  1.0,   2.0 -1.0,
                -2.0 -1.0,  -2.0  1.0,
                -0.5  1.0
            ]
            spine [
            # Straight-line
                0.0 0.0 0.0,
                0.0 2.0 0.0
            ]
        }

    }
    ]
}
```



Scena VRML

spine curva

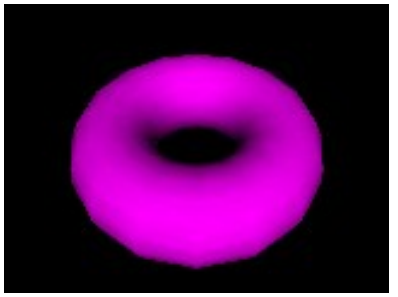


```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
        diffuseColor 0.0 1.0 0.7
    }
    }
    geometry Extrusion {
    creaseAngle 1.57
    endCap   FALSE
    beginCap FALSE
    solid    FALSE
    crossSection [
    # Half-circle
        -1.00  0.00,  -0.92 -0.38,
        -0.71 -0.71,  -0.38 -0.92,
         0.00 -1.00,   0.38 -0.92,
         0.71 -0.71,   0.92 -0.38,
         1.00  0.00,
    ]
    spine [
    # Helix
          2.00 0.00 -0.00,   1.85 0.12 -0.77,
          1.41 0.24 -1.41,   0.77 0.36 -1.85,
          0.00 0.48 -2.00,  -0.77 0.61 -1.85,
        -1.41 0.73 -1.41,  -1.85 0.85 -0.77,
        -2.00 0.97  0.00,  -1.85 1.09  0.77,
        -1.41 1.21  1.41,  -0.77 1.33  1.85,
         0.00 1.45  2.00,   0.77 1.58  1.85,
         1.41 1.70  1.41,   1.85 1.82  0.77,
         2.00 1.94  0.00,   1.85 2.06 -0.77,
         1.41 2.18 -1.41,   0.77 2.30 -1.85,
         0.00 2.42 -2.00,  -0.77 2.55 -1.85,
        -1.41 2.67 -1.41,  -1.85 2.79 -0.77,
        -2.00 2.91  0.00,  -1.85 3.03  0.77,
        -1.41 3.15  1.41,  -0.77 3.27  1.85,
         0.00 3.39  2.00,   0.77 3.52  1.85,
         1.41 3.64  1.41,   1.85 3.76  0.77,
         2.00 3.88  0.00,
    ]
    }
}
```

Scena VRML

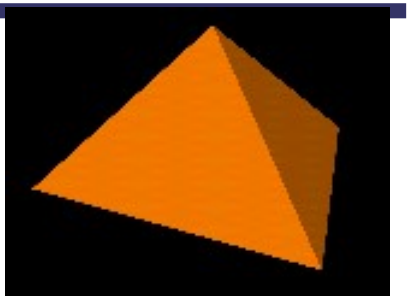
toro

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
        diffuseColor 1.0 0.0 1.0
    }
    }
    geometry Extrusion {
    creaseAngle 1.57
    beginCap FALSE
    endCap   FALSE
    crossSection [
    # Circle
          1.00  0.00,    0.92 -0.38,
          0.71 -0.71,    0.38 -0.92,
          0.00 -1.00,   -0.38 -0.92,
         -0.71 -0.71,   -0.92 -0.38,
         -1.00 -0.00,   -0.92  0.38,
         -0.71  0.71,   -0.38  0.92,
          0.00  1.00,    0.38  0.92,
          0.71  0.71,    0.92  0.38,
          1.00  0.00
    ]
    spine [
    # Circle
          2.00 0.0  0.00,    1.85 0.0 0.77,
          1.41 0.0  1.41,    0.77 0.0 1.85,
          0.00 0.0  2.00,   -0.77 0.0 1.85,
         -1.41 0.0  1.41,   -1.85 0.0 0.77,
         -2.00 0.0  0.00,   -1.85 0.0 -0.77,
         -1.41 0.0 -1.41,   -0.77 0.0 -1.85,
          0.00 0.0 -2.00,    0.77 0.0 -1.85,
          1.41 0.0 -1.41,    1.85 0.0 -0.77,
          2.00 0.0  0.00,
    ]
    }
}
```



Scena VRML

piramide



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
         diffuseColor 1.0 0.5 0.0
    }
    }
    geometry Extrusion {
    crossSection [
    # Square
         -1.0  1.0,    1.0  1.0,
          1.0 -1.0,   -1.0 -1.0,
         -1.0  1.0
    ]
    spine [ 0.0 0.0 0.0,   0.0 1.0 0.0 ]
    scale [ 1.0 1.0,       0.01 0.01 ]
    }
}
```

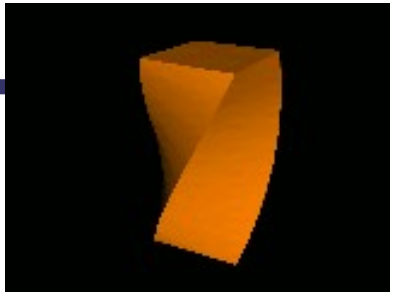
Scena VRML

vaso

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
        diffuseColor 1.0 0.8 0.0
    }
    }
    geometry Extrusion {
    creaseAngle 1.57
    endCap FALSE
    solid  FALSE
    crossSection [
    # Circle
          1.00  0.00,    0.92 -0.38,
          0.71 -0.71,    0.38 -0.92,
          0.00 -1.00,   -0.38 -0.92,
         -0.71 -0.71,   -0.92 -0.38,
         -1.00 -0.00,   -0.92  0.38,
         -0.71  0.71,   -0.38  0.92,
          0.00  1.00,    0.38  0.92,
          0.71  0.71,    0.92  0.38,
          1.00  0.00
    ]
    spine [
    # Straight-line
        0.0 0.0 0.0,   0.0 0.4 0.0,
        0.0 0.8 0.0,   0.0 1.2 0.0,
        0.0 1.6 0.0,   0.0 2.0 0.0,
        0.0 2.4 0.0,   0.0 2.8 0.0,
        0.0 3.2 0.0,   0.0 3.6 0.0,
        0.0 4.0 0.0
    ]
    scale [
        1.8  1.8,   1.95 1.95,
        2.0  2.0,   1.95 1.95
        1.8  1.8,   1.5  1.5
        1.2  1.2,   1.05 1.05,
        1.0  1.0,   1.05  1.05,
        1.15 1.15,
    ]
    }
}
```

Scena VRML

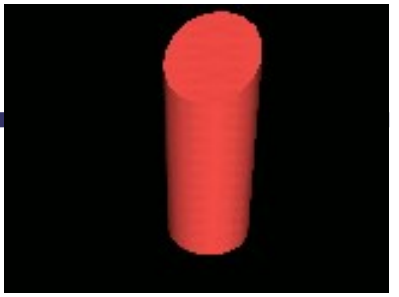
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
         diffuseColor 1.0 0.5 0.0
    }
    }
    geometry Extrusion {
    creaseAngle 0.785
    crossSection [
    # Square
         -1.0  1.0,   1.0  1.0,
          1.0 -1.0,  -1.0 -1.0,
         -1.0  1.0
    ]
    spine [
    # Straight-line
         0.0 0.0 0.0,
         0.0 0.5 0.0,
         0.0 1.0 0.0,
         0.0 1.5 0.0,
         0.0 2.0 0.0,
         0.0 2.5 0.0,
         0.0 3.0 0.0,
         0.0 3.5 0.0,
         0.0 4.0 0.0
    ]
    orientation [
         0.0 1.0 0.0 0.0,
         0.0 1.0 0.0 0.175,
         0.0 1.0 0.0 0.349,
         0.0 1.0 0.0 0.524,
         0.0 1.0 0.0 0.698,
         0.0 1.0 0.0 0.873,
         0.0 1.0 0.0 1.047,
         0.0 1.0 0.0 1.222,
         0.0 1.0 0.0 1.396,
    ]
    }
}
```



Cross-section ruotata

Scena VRML

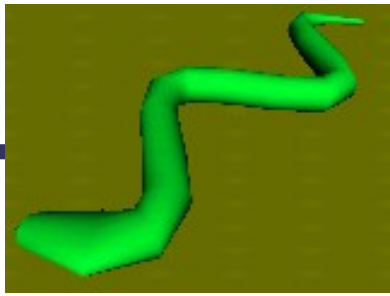
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material {
         diffuseColor 1.0 0.3 0.3
    }
    }
    geometry Extrusion {
    creaseAngle 1.57
    crossSection [
    # Circle
          1.00  0.00,    0.92 -0.38,
          0.71 -0.71,    0.38 -0.92,
          0.00 -1.00,   -0.38 -0.92,
         -0.71 -0.71,   -0.92 -0.38,
         -1.00 -0.0,    -0.92  0.38,
         -0.71  0.71,   -0.38  0.92,
          0.00  1.00,    0.38  0.92,
          0.71  0.71,    0.92  0.38,
          1.00  0.00
    ]
    spine [
    # Straight-line
         0.0 0.0 0.0,  0.0 4.0 0.0,
    ]
    orientation [
         0.0 1.0 0.0 0.0,  1.0 0.0 0.0 0.785
    ]
    }
}
```



Cross-section ruotata

Scena VRML

Spine animata



Scena VRML

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Ground
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor 0.6 0.6 0.0
            }
        }
        geometry Box { size 20.0 0.01 20.0 }
    },
    # Snake shape
    Transform {
        translation 0.0 0.3 0.0
        children Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 0.0 1.0 0.2
                }
            }
            geometry DEF Snake Extrusion {
                creaseAngle 1.57
                crossSection [
                # Circle
                     1.00  0.00,    0.92 -0.38,
                     0.71 -0.71,    0.38 -0.92,
                     0.00 -1.00,   -0.38 -0.92,
                    -0.71 -0.71,   -0.92 -0.38,
                    -1.00 -0.00,   -0.92  0.38,
                    -0.71  0.71,   -0.38  0.92,
                     0.00  1.00,    0.38  0.92,
                     0.71  0.71,    0.92  0.38,
                     1.00  0.00
                ]
                spine [
                # Sine wave
                    -4.100 0.0  0.000,   -4.000 0.0  0.000,
                    -3.529 0.0  0.674,   -3.059 0.0  0.996,
                    -2.588 0.0  0.798,   -2.118 0.0  0.184,
                    -1.647 0.0 -0.526,   -1.176 0.0 -0.962,
                    -0.706 0.0 -0.895,   -0.235 0.0 -0.361,
                     0.235 0.0  0.361,    0.706 0.0  0.895,
                     1.176 0.0  0.962,    1.647 0.0  0.526,
                     2.118 0.0 -0.184,    2.588 0.0 -0.798,
                     3.059 0.0 -0.996,    3.529 0.0 -0.674,
                     4.000 0.0  0.000,
                ]
                scale [
                    0.050 0.020,   0.200 0.100,
                    0.400 0.150,   0.300 0.300,
                    0.300 0.300,   0.300 0.300,
                    0.300 0.300,   0.300 0.300,
                    0.300 0.300,   0.300 0.300,
                    0.290 0.290,   0.290 0.290,
                    0.290 0.290,   0.280 0.280,
                    0.280 0.280,   0.250 0.250,
                    0.200 0.200,   0.100 0.100,
                    0.050 0.050,
                ]
            }
        }
    },
    # Animation clock
    DEF Clock TimeSensor {
        cycleInterval 4.0
        loop TRUE
    },
    # Animation morph
    DEF SnakeWiggle CoordinateInterpolator {
        key [ 0.0, 0.25, 0.50, 0.75, 1.0 ]
        keyValue [
        # time 0.0 position
            -4.100 0.0  0.000,   -4.000 0.0  0.000,
            -3.529 0.0  0.674,   -3.059 0.0  0.996,
            -2.588 0.0  0.798,   -2.118 0.0  0.184,
            -1.647 0.0 -0.526,   -1.176 0.0 -0.962,
            -0.706 0.0 -0.895,   -0.235 0.0 -0.361,
             0.235 0.0  0.361,    0.706 0.0  0.895,
             1.176 0.0  0.962,    1.647 0.0  0.526,
             2.118 0.0 -0.184,    2.588 0.0 -0.798,
             3.059 0.0 -0.996,    3.529 0.0 -0.674,
             4.000 0.0  0.000,
        # time 0.25 position
            -4.100 0.0 -1.000,   -4.000 0.0 -1.000,
            -3.529 0.0 -0.739,   -3.059 0.0 -0.992,
            -2.588 0.0  0.603,   -2.118 0.0  0.983,
            -1.647 0.0  0.850,   -1.176 0.0  0.274,
            -0.706 0.0 -0.446,   -0.235 0.0 -0.932,
             0.235 0.0 -0.932,    0.706 0.0 -0.446,
             1.176 0.0  0.274,    1.647 0.0  0.850,
             2.118 0.0  0.983,    2.588 0.0  0.603,
             3.059 0.0 -0.092,    3.529 0.0 -0.739,
             4.000 0.0 -1.000,
        # time 0.50 position
            -4.100 0.0  0.000,   -4.000 0.0  0.000,
            -3.529 0.0 -0.674,   -3.059 0.0 -0.996,
            -2.588 0.0 -0.798,   -2.118 0.0 -0.184,
            -1.647 0.0  0.526,   -1.176 0.0  0.962,
            -0.706 0.0  0.895,   -0.235 0.0  0.361,
             0.235 0.0 -0.361,    0.706 0.0 -0.895,
             1.176 0.0 -0.962,    1.647 0.0 -0.526,
             2.118 0.0 -0.184,    2.588 0.0  0.798,
             3.059 0.0  0.996,    3.529 0.0  0.674,
             4.000 0.0  0.000,
        # time 0.75 position
            -4.100 0.0  1.000,   -4.000 0.0  1.000,
            -3.529 0.0  0.739,   -3.059 0.0  0.992,
            -2.588 0.0 -0.603,   -2.118 0.0 -0.983,
            -1.647 0.0 -0.850,   -1.176 0.0 -0.274,
            -0.706 0.0  0.446,   -0.235 0.0  0.932,
             0.235 0.0  0.932,    0.706 0.0  0.446,
             1.176 0.0 -0.274,    1.647 0.0 -0.850,
             2.118 0.0 -0.983,    2.588 0.0 -0.603,
             3.059 0.0  0.092,    3.529 0.0  0.739,
             4.000 0.0  1.000,
        # time 1.0 position
            -4.100 0.0  0.000,   -4.000 0.0  0.000,
            -3.529 0.0  0.674,   -3.059 0.0  0.996,
            -2.588 0.0  0.798,   -2.118 0.0  0.184,
            -1.647 0.0 -0.526,   -1.176 0.0 -0.962,
            -0.706 0.0 -0.895,   -0.235 0.0 -0.361,
             0.235 0.0  0.361,    0.706 0.0  0.895,
             1.176 0.0  0.962,    1.647 0.0  0.526,
             2.118 0.0 -0.184,    2.588 0.0 -0.798,
             3.059 0.0 -0.996,    3.529 0.0 -0.674,
             4.000 0.0  0.000,
            ]
        }
    ]
}
ROUTE Clock.fraction_changed    TO SnakeWiggle.set_fraction
ROUTE SnakeWiggle.value_changed TO Snake.set_spine
```

Virtual Reality Modeling Language

VRML

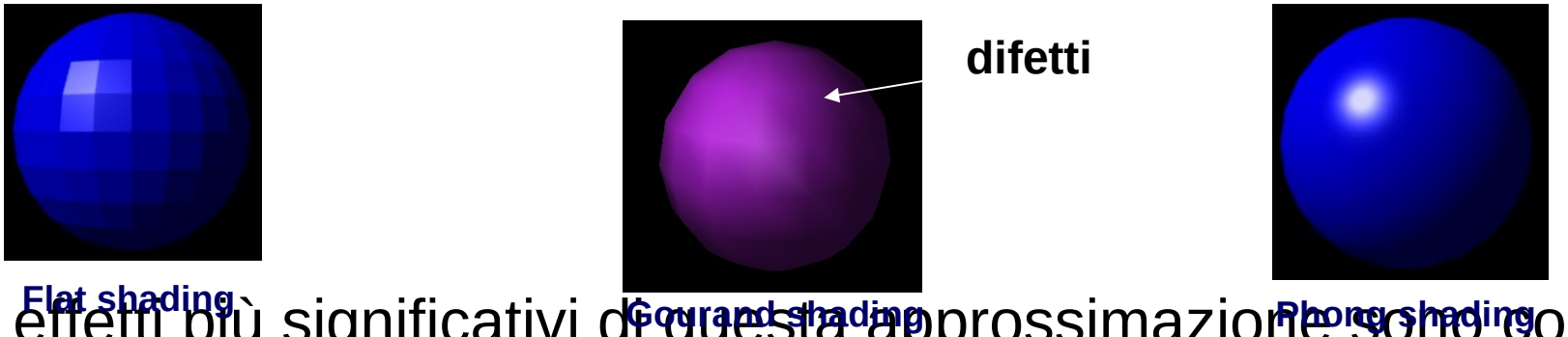
Illuminare i mondi VRML

Il modello di illuminazione VRML

- Il modello di illuminazione VRML definisce le equazioni matematiche che consentono di calcolare il colore da applicare agli oggetti geometrici.
- Per ogni oggetto la luce che colpisce un oggetto viene combinata con le caratteristiche dei nodi **`Material`**, **`Color`**, **`Texture`** e con le proprietà dell'eventuale nodo **`Fog`** attivo.
- Queste equazioni cercano di riprodurre le proprietà fisiche della luce che colpisce una superficie.
- Una forma non viene illuminata se si verifica una delle due condizioni:
  - Il nodo **`Material`** è **`NULL`** (default)
  - Il nodo **`Appearance`** è **`NULL`** (default)

Il modello di illuminazione VRML

- I nodi **`PointSet`**, **`IndexedFaceSet`** e **`IndexedLineSet`** hanno proprietà particolari legate al fatto che il campo **`colorPerVertex`** sia **`TRUE`** o **`FALSE`**.
- Il modello effettua i calcoli per l illuminazione secondo l algoritmo Gouraud Shading interpolando linearmente il colore tra i vertici di ciascun poligono (non per ogni pixel della figura, come avviene per l algoritmo denominatoPhong Shading).



- Gli effetti più significativi di questa approssimazione sono costituiti da margini confusi o non accurati per superfici con brillantezza a specchio e luci spot, dato che lo spezzettamento delle facce incide sui calcoli che vengono effettuati. L approssimazione può essere migliorata aumentando il numero di vertici e facce.

Illuminare un mondo VRML

- Le luci nel mondo VRML assolvono esattamente alle stesse funzioni del mondo reale: illuminano oggetti, mettono in risalto alcune caratteristiche.
- VRML supporta tre tipi di nodi per controllare l'illuminazione: `PointLight`, `DirectionalLight` e `SpotLight`.
- L'illuminazione fornisce un contributo incredibile al realismo del mondo virtuale.
- Le luci in VRML non hanno forma: esse descrivono come illuminare il mondo, ma non danno indicazioni sulla forma che la luce ha. Si vedono solo gli effetti.
- Ai tre nodi VRML corrispondono tre modi diversi di illuminazione della scena: luci puntuali, luci direzionali, luci spot

Luci omnidirezionali: `PointLight`

- Una luce puntiforme è una luce posta nel mondo virtuale che emana luce in tutte le direzioni, come se i raggi luminosi provengano tutti da una stessa sorgente.
- Nel mondo reale una tale luce è rappresentata da una lampadina accesa e senza paralume.
- In VRML una point light viene creata specificando la sua coordinata 3D, la sua intensità e il suo colore.
- Il colore della luce è al solito espresso in notazione RGB. Una point light colorata conferisce colore al mondo circostante

Luci direzionali: `DirectionalLight`

- Una luce direzionale è una luce posta nel mondo virtuale ad una grande distanza dal mondo stesso, di modo che i raggi di luce sono paralleli tra loro e puntano tutti nella stessa direzione.
- Nel mondo reale una luce di questo tipo è rappresentata dal sole.
- In VRML una luce direzionale viene creata specificando la sua posizione come coordinata 3D e la sua direzione di puntamento.
- La direzione di puntamento si esprime mediante la coordinata 3D, essendo l'asse identificato dalla coordinata e dall'origine del mondo VRML.

Luci dirette: `SpotLight`

- Una SpotLight è una luce posta nel mondo virtuale orientata in una direzione specifica, i cui raggi si propagano a forma di cono. Le forme che cadono dentro alla superficie del cono vengono illuminate, le altre no.
- Nel mondo reale questo tipo di luce è rappresentata da un faretto
- La luce viene creata specificando la sua posizione come coordinata 3D, la sua direzione di puntamento e due parametri che consentono di stabilire le caratteristiche del cono luminoso: l'angolo complessivo che il cono forma (angolo tra l'asse del cono e il raggio più esterno) e le dimensioni del cono più piccolo all'interno del quale i raggi luminosi hanno la stessa intensità. I raggi tra il cono piccolo e quello grande diminuiscono progressivamente di intensità dall'interno all'esterno.

Luci colorate

- I principi di Fisica ci insegnano che la luce bianca è una luce che presenta tutte le lunghezze d'onda.
- Quando una luce bianca illumina una superficie colorata, alcuni raggi sono assorbiti, mentre altri sono riflessi e possono colpire il nostro occhio.
- La discriminazione tra raggi assorbiti e riflessi dipende dal colore della superficie. Una superficie blu riflette le lunghezze d'onda del blu ed assorbe le altre.
- Il colore che vediamo nelle cose è determinato dai raggi di luce riflessa. Una superficie bianca riflette tutti i raggi, una nera li assorbe tutti.

Luci colorate (i)

- In VRML i principi ed i comportamenti sono analoghi al mondo reale.
- La Headlight è bianca e non può essere colorata, mentre le altre luci possono essere colorate.
- Quando una luce colorata illumina una superficie, il colore della superficie cambia in quanto non è più irradiata da tutte le lunghezze d'onda (luce bianca) e pertanto la luce riflessa (e quindi il colore) dalla superficie cambia.
- Quando una luce colorata illumina una superficie colorata, si ottengono effetti particolari: una superficie blu riflette solo i raggi blu. Se viene illuminata da una luce rossa, la superficie apparirà nera, perché la luce non contiene il colore blu, pertanto non ci sono raggi che possono essere riflessi.

Luce ambiente

- Nel mondo reale le cose vengono illuminate da sorgenti luminose dirette e dalla luce riflessa da altre superfici
- La luce riflessa ha un tono più diffuso e caldo (es: un salotto illuminato attraverso lampade dirette verso il soffitto, che generano una luce soffusa e calda)
- La luce ambiente origina da raggi luminosi che vengono riflessi da varie superfici.
- Ogni tipo di luce in VRML ha un parametro che ne regola la quantità di luce ambiente generata, attraverso il campo `ambientIntensity`.

Luce ambiente (i)

- Un basso valore del parametro `ambientIntensity` significa che la luce in oggetto non contribuisce molto alla luce ambiente, un alto valore implica un contributo significativo alla luce ambiente.
- Quando il browser VRML disegna le forme del mondo ai fini dell'illuminazione delle forme, considera la somma dei contributi sia di sorgenti di luce diretta che quello della luce ambiente.
- La luce ambiente fornisce alla scena una luce più soffusa e meno contrastata, come se esplorassimo il mondo virtuale al lume di candela. Un alto valore di luce ambiente "lava via" la scena, annullando l'effetto ombreggiatura

Attenuazione della luce

- L'attenuazione della luce descrive il modo con cui l'illuminazione della luce gradualmente diminuisce all'aumentare della distanza dalla sorgente luminosa.
- Utilizzando l'attenuazione della luce si può fare in modo che alcune luci illuminino una larga porzione della scena, mentre delle altre illuminino solo i loro immediati dintorni.
- I nodi `PointLight` e `SpotLight` forniscono due controlli per l'attenuazione: un insieme di tre parametri di attenuazione e il raggio della luce. Il raggio della luce controlla l'area massima di illuminazione della luce. Con un raggio piccolo viene illuminato solo un piccolo dettaglio della scena.
- I parametri di attenuazione controllano come l'illuminazione descresce allontanandosi lungo il raggio, dall'asse del cono di luce.

Luci multiple

- Nel mondo VRML si possono usare diverse luci e di diverso colore.
- L'effetto delle singole luci può essere regolato mediante il controllo dell'attenuazione e del raggio della luce nel caso dei nodi `PointLight` e `SpotLight`. Nel caso del nodo `DirectionalLight` il controllo si ha attraverso le forme che vengono poste nello stesso gruppo del nodo della luce, dato che l'effetto di questo tipo di luce è limitato soltanto ad esse (e alle forme di un nodo di raggruppamento più esterno se questo è un nodo di tipo `Transform`).

Headlight

- L'Headlight è una luce bianca di tipo `DirectionalLight` che viene aggiunta automaticamente dal browser al nostro mondo virtuale. Quando è attiva, l'Headlight illumina il mondo di fronte all'osservatore (è posta immaginariamente sulla fronte dell'osservatore stesso).
- L'Headlight è attivabile/disattivabile sia dal menu del browser VRML che attraverso il nodo `NavigationInfo`.
- Quando si definiscono luci nel mondo può tornare comodo disattivare Headlight per avere il pieno controllo dell'illuminazione della scena.

Ombre

- Nel mondo reale le forme generano ombre quando intercettano i raggi di luce.
- In VRML le forme non generano ombre
- Il calcolo delle ombre è troppo oneroso e complesso per essere effettuato interattivamente.
- Poiché le forme VRML non generano ombre, una forma può essere illuminata anche se è posta dietro ad altre forme.
- L'assenza di ombre, rende le forme VRML delle entità molto più difficili da comprendere.
- Se si vuole dare realismo al proprio mondo occorre disegnare la forma dell'ombra mediante ad es: un nodo `IndexedFaceSet`, colorandola di grigio e assegnandole proprietà di trasparenza, magari animandone la forma…

Illuminazione di forme piatte

- Quando una forma piatta come la faccia di un cubo viene illuminata nel mondo reale da una luce tipo faretto si crea un gradiente di illuminazione procedendo dal centro della faccia verso l'esterno.
- Il comportamento in VRML dipende dal browser, ma in generale tale effetto si perde, perché il calcolo della luminosità viene condotto solo sulle coordinate della faccia, che sono equidistanti dalla luce.
- Una soluzione per dare più realismo al mondo è quella di usare meshed shapes (usando ad esempio i nodi `IndexedfaceSet` o `ElevationGrid`), in modo che lo stesso calcolo sulla griglia risultante produce delle facce diversamente illuminate in base alla loro posizione rispetto a quella della luce.

Il nodo `PointLight`

- Il nodo `PointLight` crea una luce direzionale I cui raggi si diffondono a raggiera in ogni direzione

```
PointLight {
        on                TRUE              # exposedField SFBool
        location          0.0 0.0 0.0       # exposedField SFVec3f
        radius            100.0             # exposedField SFFloat
        intensity         1.0               # exposedField SFFloat

        ambientIntensity 0.0                # exposedField SFFloat
        color             1.0 1.0 1.0       # exposedField SFColor
        attenuation        1.0 0.0 0.0      # exposedField SFVec3f
          }
```

Il nodo `PointLight` (i)

- Il valore dell'exposed-field `location` specifica la coordinata che indica la posizione della luce. Il valore di default è nell'origine del sistema di coordinate
- Il valore dell'exposed-field `radius` specifica il raggio di una sfera di illuminazione centrata nella posizione della lampada.
- Il valore dell'exposed-field `intensity` specifica l'intensità e la brillantezza della sorgente luminosa. Un valore `0.0` rende la lampada scura, `1.0` al massimo della brillantezza.
- Il valore dell'exposed-field `ambientIntensity` specifica l'effetto che la luce ha sulla luce d'ambiente. Un valore `0.0` vuol dire che la luce non influenza la luce ambientale, `1.0` che ha un fortissimo effetto.

Il nodo `PointLight` (ii)

- Il valore dell'exposed-field `color` specifica il colore RGB della luce. Il valore di default è il colore bianco.
- I campi `intensity` e `color` si combinano tra loro per determinare il livello di luce colorata della sorgente luminosa, secondo la relazione:

  `lightColor = color x intensity`

- Il campo `intensity` si comporta come una manopola che regola la brillantezza del colore selezionato nel campo `color`.
- I campi `intensity`, `ambientIntensity` e `color` si combinano tra loro per determinare il contributo della luce colorata alla luce ambientale, secondo l'espressione:

  `lightAmbientColor = color x intensity ambientIntensity`

- I campi `intensity` e `ambientIntensity` si comportano come manopole che controllano il colore della luce ambientale selezionata dal campo `color`.

Il nodo `PointLight` (iii)

- Il valore dell'exposed-field `attenuation` specifica come l'intensità della luce varia allontanandosi dal centro della sfera di illuminazione, lungo il raggio. Per una coordinata dentro alla sfera di illuminazione, ad una distanza d dalla posizione della luce, la brillantezza del colore della luce è determinata da:

$$\texttt{attenuatedColor} = \texttt{lightColor} / (\texttt{attenuation[0]} + \texttt{attenuation[1]} \times d + \texttt{attenuation[2]} \times d^2)$$

- I tre parametri di `attenuation` possono essere usati insieme per generare una vasta gamma di effetti. Alcuni browser VRML non implementano a pieno le funzionalità espresse da `attenuation`. Il valore di default crea una luce con illuminazione costante all'interno della sfera.

Il nodo `PointLight` (iv)

- Il comportamento e gli attributi del nodo `PointLight` possono essere modificati dinamicamente inviando dei valori attraversi gli eventIn impliciti dei vari exposed-field: `set_on`, `set_location`, `set_radius`, `set_intensity`, `set_ambientIntensity`, `set_color` e `set_attenuation`.
- I nuovi valori potranno essere propagati attraverso gli eventOut impliciti dei vari exposed-field: `on_changed`, `location_changed`, `radius_changed`, `intensity_changed`, `ambientIntensity_changed`, `color_changed` e `attenuation_changed`.
- Le luci influenzano le proprietà dei materiali.

Il nodo `DirectionalLight`

- Il nodo `DirectionalLight` crea una luce diretta i cui raggi si diffondono in parallelo in una specifica direzione

```
DirectionalLight {
        on                TRUE              # exposedField SFBool
        intensity         1.0               # exposedField SFFloat

        ambientIntensity 0.0                # exposedField SFFloat
        color             1.0 1.0 1.0       # exposedField SFColor
        direction         0.0 0.0 -1.0      # exposedField SFVec3f
        }
```

Il nodo `DirectionalLight` (i)

- Il significato degli exposed-field `on`, `intensity`, `ambientIntensity` e `color` è lo stesso visto per il nodo `PointLight`.
- Il valore dell'exposed-field `direction` specifica un coordinata 3D. I raggi luminosi correranno tutti paralleli alla linea immaginaria che collega la coordinata 3D all'origine del mondo VRML. Il valore di default indica una direzione di propagazione dei raggi luminosi nella direzione negativa dell'asse Z.
- La luce proveniente da un nodo `DirectionalLight` illumina tutte le forme costruite nel gruppo contenete il nodo e tutte le forme contenute nei sottogruppi dello stesso gruppo contenente il nodo `DirectionalLight`. Non essendo posizionata in un punto dello spazio preciso, non sono possibili effetti di attenuazione.

Il nodo `DirectionalLight` (ii)

- Il comportamento e gli attributi del nodo `DirectionalLight` possono essere modificati dinamicamente inviando dei valori attraversi gli eventIn impliciti dei vari exposed-field: `set_on`, `set_direction`, `set_intensity`, `set_ambientIntensity` e `set_color`.
- I nuovi valori potranno essere propagati attraverso gli eventOut impliciti dei vari exposed-field: `on_changed`, `direction_changed`, `intensity_changed`, `ambientIntensity_changed` e `color_changed`.
- Un nodo `DirectionalLight` influenza le proprietà dei materiali.

Il nodo SpotLight

- Il nodo **SpotLight** crea una luce i cui raggi si propagano in modo radiale secondo la forma di una superficie conica. Le forme che cadono nel cono sono illuminate, le altre no.

```
SpotLight {
        on                TRUE            # exposedField SFBool
        location          0.0 0.0 0.0     # exposedField SFVec3f
        direction         0.0 0.0 -1.0    # exposedField SFVec3f
        radius            100.0           # exposedField SFFloat
        intensity         1.0             # exposedField SFFloat

        ambientIntensity 0.0              # exposedField SFFloat
        color             1.0 1.0 1.0     # exposedField SFColor
        attenuation       1.0 0.0 0.0     # exposedField SFVec3f
        beamWidth         1.570796        # exposedField SFFloat
        cutOffAngle       0.785398        # exposedField SFFloat
            }
```

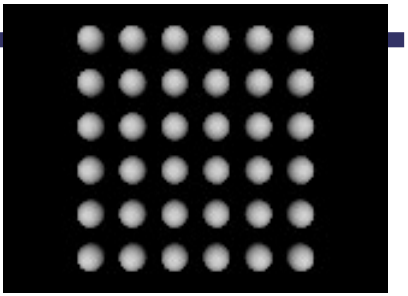
Il nodo `SpotLight` (i)

- Il significato degli exposed-field `on`, `location`, `radius`, `intensity`, `ambientIntensity` e `color` è lo stesso visto per il nodo `PointLight`.
- Il valore dell'exposed-field `cutOffAngle` specifica l'ampiezza dell'angolo del cono di illuminazione, il cui vertice coincida con la posizione della lampada e il cui asse coincida con la direzione dell'asse di propagazione della luce. L'angolo esprime l'ampiezza dell'angolo in radianti formato tra l'asse e le pareti del cono. Angoli maggiori aumentano il cono di luce emanato. Il valore di default è 45.0 gradi (`0.78398` radianti).
- Il valore dell'exposed-field `beamWidth` indica l'angolo di apertura del cono, compreso nel cono più ampio specificato da `cutOffAngle`, in cui l'illuminazione è costante. Anche l'asse ed il verice di questo cono coincidono con quelli del cono più grande

Il nodo `SpotLight` (ii)

- Il comportamento e gli attributi del nodo `SpotLight` possono essere modificati dinamicamente inviando dei valori attraversi gli eventIn impliciti dei vari exposed-field: `set_on`, `set_location`, `set_direction`, `set_radius`, `set_intensity`, `set_ambientIntensity`, `set_color`, `set_attenuation`, `set_beamWidth` e `set_cutOffAngle`.
- I nuovi valori potranno essere propagati attraverso gli eventOut impliciti dei vari exposed-field: `on_changed`, `location_changed`, `direction_changed`, `radius_changed`, `intensity_changed`, `ambientIntensity_changed`, `color_changed`, `attenuation_changed`, `beamWidth_changed` e `cutOffAngle_changed`.
- Un nodo `SpotLight` influenza le proprietà dei materiali.

Headlight



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Transform {
   translation -7.5 -7.5 0.0
   children [
   DEF BallRow Group {
        children [
            DEF Ball Shape {
                appearance Appearance {
                    material Material { }
                }
                geometry Sphere { }
            },
            Transform { translation  3.0 0.0 0.0 children USE Ball },
            Transform { translation  6.0 0.0 0.0 children USE Ball },
            Transform { translation  9.0 0.0 0.0 children USE Ball },
            Transform { translation 12.0 0.0 0.0 children USE Ball },
            Transform { translation 15.0 0.0 0.0 children USE Ball }
        ]
   },
   Transform { translation 0.0  3.0 0.0 children USE BallRow },
   Transform { translation 0.0  6.0 0.0 children USE BallRow },
   Transform { translation 0.0  9.0 0.0 children USE BallRow },
   Transform { translation 0.0 12.0 0.0 children USE BallRow },
   Transform { translation 0.0 15.0 0.0 children USE BallRow }
   ]
}
```

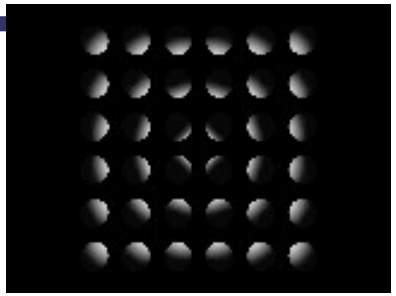
Scena VRML

Test mesh

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Transform {
   translation -8.25 0.0 -8.25
   children Shape {
   appearance Appearance {
        material Material { }
   }
   geometry ElevationGrid {
        xDimension 12
        zDimension 12
        xSpacing   1.5
        zSpacing   1.5
        solid      FALSE
        height [
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        ]
   }
   }
}
```

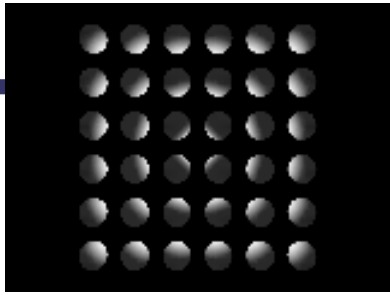
PointLight

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright 1997 By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    PointLight {
        location 0.0 0.0 0.0
        radius   12.0
    },
    Inline {
        url "spheres.wrl"
        bboxCenter 0.0 0.0 0.0
        bboxSize   16.0 16.0 1.0
    }
    ]
}
```



Scena VRML

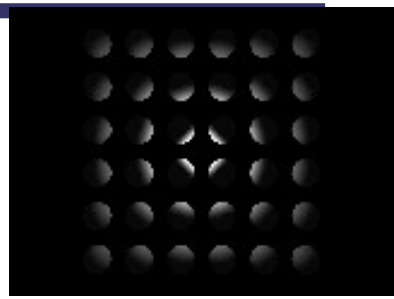
Pointlight + ambientIntensity



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright 1997 By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    PointLight {
        location 0.0 0.0 0.0
        radius   12.0
        ambientIntensity 0.8
    },
    Inline {
        url "spheres.wrl"
        bboxCenter 0.0 0.0 0.0
        bboxSize   16.0 16.0 1.0
    }
    ]
}
```

Scena VRML

PointLight + attenuation control

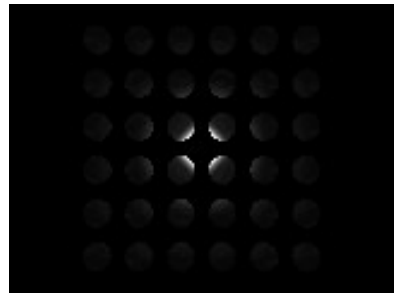


```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    PointLight {
        location 0.0 0.0 0.0
        radius   12.0
        attenuation 0.0 0.3 0.0
    },
    Inline {
        url "spheres.wrl"
        bboxCenter 0.0 0.0 0.0
        bboxSize   16.0 16.0 1.0
    }
    ]
}
```

Scena VRML

PointLight + small radius

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    PointLight {
        location 0.0 0.0 0.0
        radius   5.0
        ambientIntensity 0.8
    },
    Inline {
        url "spheres.wrl"
        bboxCenter 0.0 0.0 0.0
        bboxSize   16.0 16.0 1.0
    }
    ]
}
```



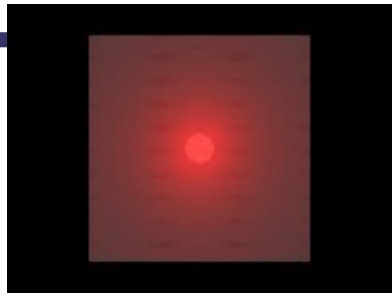
Scena VRML

Superfici illuminate con PointLight

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
      children [
      # Vaulted ceiling and columns
      Inline {
              url "vaulted.wrl"
              bboxCenter 0.0 1.0 0.0
              bboxSize   6.0 2.0 6.0
      },
      # Floor
      Inline {
              url "mesh.wrl"
              bboxCenter 0.0 0.0 0.0
              bboxSize   15.0 0.0 15.0
      },
      # Glowing sphere
      Transform {
              translation 0.0 1.0 0.0
              children [
                      PointLight {
                              location  0.0 0.0 0.0
                              radius    10.0
                              intensity 1.0
                              ambientIntensity 0.2
                              color 0.7 0.5 0.0
                      },
                      Shape {
                              appearance Appearance {
                                      # No material, use emissive texturing
                                      texture ImageTexture {
                                              url "fire.jpg"
                                      }
                              }
                              geometry Sphere { radius 0.2 }
                      }
              ]
      },
      # Pedestal pyramid
      Shape {
              appearance DEF White Appearance {
                      material Material { }
              }
              geometry IndexedFaceSet {
                      coord Coordinate {
                              point [
                              # Around the base
                                      -0.12 0.03  0.12,   0.12 0.03  0.12,
                                       0.12 0.03 -0.12,  -0.12 0.03 -0.12,
                              # Tip
                                       0.0  0.63  0.0,
                              ]
                      }
                      coordIndex [
                              0, 1, 4, -1,  1, 2, 4, -1,
                              2, 3, 4, -1,  3, 0, 4, -1,
                      ]
                      solid TRUE
              }
      },
      # Pedestal base
      Transform {
              translation 0.0 0.015 0.0
              children Shape {
                      appearance USE White
                      geometry Box { size 0.4 0.03 0.4 }
              }
      }
      ]
}
```

Scena VRML

Luci animate

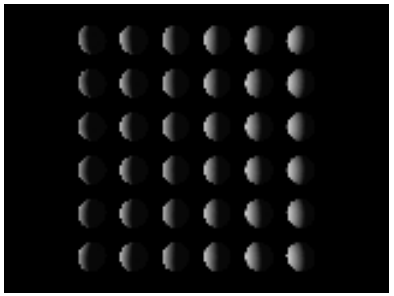


```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Generic lighting for ambience
    DirectionalLight {
          intensity 0.2
          ambientIntensity 1.0
    },
    # Light bulb
    DEF BulbLight PointLight {
          radius 16.0
          color 1.0 0.0 0.0
    },
    # Light bulb shape
    Shape {
          appearance Appearance {
                material DEF BulbColor Material {
                      emissiveColor 1.0 0.3 0.3
                      diffuseColor  0.0 0.0 0.0
                }
          }
          geometry DEF Bulb Sphere { }
    },
    # Wall
    Transform {
          translation 0.0 0.0 -1.1
          rotation 1.0 0.0 0.0 1.57
          children Inline { url "mesh.wrl" }
    },
    # Animation clock
    DEF Clock TimeSensor {
          cycleInterval 4.0
          loop TRUE
    },
    # Animation brightness and colors
    DEF BulbIntensity ScalarInterpolator {
          key [ 0.0, 0.5, 0.5, 1.0 ]
          keyValue [ 1.0, 1.0, 0.0, 0.0 ]
    },
    DEF BulbDiffuse ColorInterpolator {
          key [ 0.0, 0.5, 0.5, 1.0 ]
          keyValue [
                0.0 0.0 0.0,  0.0 0.0 0.0,
                1.0 0.3 0.3,  1.0 0.3 0.3
          ]
    },
    DEF BulbEmissive ColorInterpolator {
          key [ 0.0, 0.5, 0.5, 1.0 ]
          keyValue [
                1.0 0.3 0.3,  1.0 0.3 0.3,
                0.0 0.0 0.0,  0.0 0.0 0.0,
          ]
    }
    ]
}
ROUTE Clock.fraction_changed      TO BulbIntensity.set_fraction
ROUTE Clock.fraction_changed      TO BulbDiffuse.set_fraction
ROUTE Clock.fraction_changed      TO BulbEmissive.set_fraction
ROUTE BulbIntensity.value_changed TO BulbLight.set_intensity
ROUTE BulbDiffuse.value_changed   TO BulbColor.set_diffuseColor
ROUTE BulbEmissive.value_changed  TO BulbColor.set_emissiveColor
```

Scena VRML

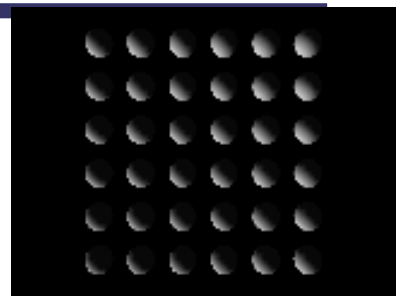
DirectionalLight da dx

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright 1997 By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    DirectionalLight {
        direction 1.0 0.0 0.0
    },
    Inline {
        url "spheres.wrl"
        bboxCenter 0.0 0.0 0.0
        bboxSize   16.0 16.0 1.0
    }
    ]
}
```



Scena VRML

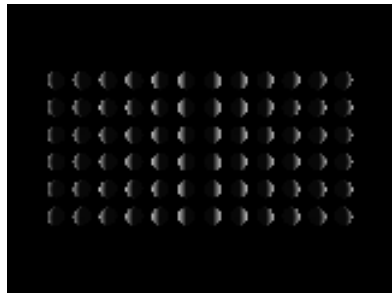
DirectionalLight dal basso a sx



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright 1997 By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    DirectionalLight {
        direction 0.5 0.5 0.0
    },
    Inline {
        url "spheres.wrl"
        bboxCenter 0.0 0.0 0.0
        bboxSize   16.0 16.0 1.0
    }
    ]
}
```

Scena VRML

2 DirectionalLight da opposte direzioni

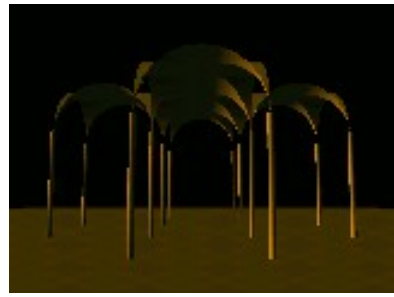


```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    Transform {
        translation -9.0 0.0 0.0
        children [
            DirectionalLight {
                direction 1.0 0.0 0.0
            },
            DEF Spheres Inline {
                url "spheres.wrl"
                bboxCenter 0.0 0.0 0.0
                bboxSize   16.0 16.0 1.0
            },
        ]
    },
    Transform {
        translation 9.0 0.0 0.0
        children [
            DirectionalLight {
                direction -1.0 0.0 0.0
            },
            USE Spheres
        ]
    }
    ]
}
```

Scena VRML

DirectionalLight che emula il sole

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    DirectionalLight {
        direction  0.8 -0.2 -0.2
        intensity 1.0
        ambientIntensity 0.3
        color 1.0 0.6 0.0
    },
    # Vaulted ceiling and columns
    Inline {
        url "vaulted.wrl"
        bboxCenter 0.0 1.0 0.0
        bboxSize   6.0 2.0 6.0
    },
    # Floor
    Shape {
        appearance Appearance {
            material Material { }
        }
        geometry Box { size 16.0 0.01 16.0 }
    }
    ]
}
```



Scena VRML

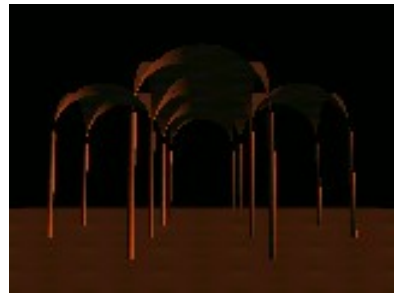
DirectionalLight che emula il pomeriggio

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    DirectionalLight {
         direction  0.0 -1.0 -0.2
         intensity 1.0
         ambientIntensity 0.7
         color 1.0 0.9 0.7
    },
    # Vaulted ceiling and columns
    Inline {
         url "vaulted.wrl"
         bboxCenter 0.0 1.0 0.0
         bboxSize   6.0 2.0 6.0
    },
    # Floor
    Shape {
         appearance Appearance {
             material Material { }
         }
         geometry Box { size 16.0 0.01 16.0 }
    }
    ]
}
```

Scena VRML

DirectionalLight che emula il tramonto

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    DirectionalLight {
        direction -0.8 -0.2 -0.2
        intensity 1.0
        ambientIntensity 0.2
        color 1.0 0.3 0.1
    },
    # Vaulted ceiling and columns
    Inline {
        url "vaulted.wrl"
        bboxCenter 0.0 1.0 0.0
        bboxSize   6.0 2.0 6.0
    },
    # Floor
    Shape {
        appearance Appearance {
            material Material { }
        }
        geometry Box { size 16.0 0.01 16.0 }
    }
    ]
}
```



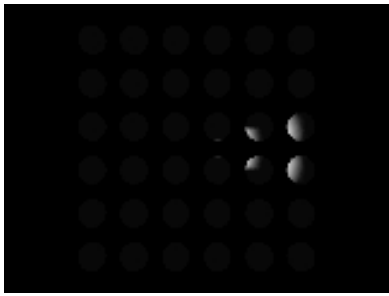
Scena VRML

Luce con sole in movimento

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
     children [
     # Animated sunlight
     DEF SunLight DirectionalLight {
           direction  0.8 -0.2 -0.2
           intensity 1.0
           ambientIntensity 0.5
           color 1.0 0.6 0.0
     },
     # Animation clock
     DEF Clock TimeSensor {
           cycleInterval 10.0
           loop TRUE
     },
     # Animation directions
     DEF LightDirection PositionInterpolator {
           key [ 0.0, 0.5, 1.0 ]
           keyValue [ 0.8 -0.2 -0.2, 0.0 -1.0 -0.2, -0.8 -0.2, -0.2 ]
     },
     # Animation colors and ambient intensity
     DEF LightColor ColorInterpolator {
           key [ 0.0, 0.5, 1.0 ]
           keyValue [ 1.0 0.6 0.0, 1.0 0.9 0.7, 1.0 0.3 0.1 ]
     },
     DEF LightAmbient ScalarInterpolator {
           key [ 0.0, 0.5, 1.0 ]
           keyValue [ 0.3, 0.7, 0.2 ]
     },
     # Vaulted ceiling and columns
     Inline {
           url "vaulted.wrl"
           bboxCenter 0.0 1.0 0.0
           bboxSize   6.0 2.0 6.0
     },
     # Floor
     Shape {
           appearance Appearance {
                 material Material { }
           }
           geometry Box { size 16.0 0.01 16.0 }
     }
     ]
}
ROUTE Clock.fraction_changed       TO LightDirection.set_fraction
ROUTE Clock.fraction_changed       TO LightColor.set_fraction
ROUTE Clock.fraction_changed       TO LightAmbient.set_fraction
ROUTE LightDirection.value_changed TO SunLight.set_direction
ROUTE LightColor.value_changed     TO SunLight.set_color
ROUTE LightAmbient.value_changed   TO SunLight.set_ambientIntensity
```

Scena VRML

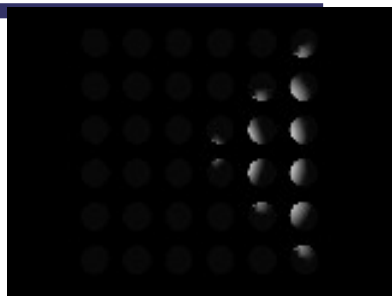
SpotLight dal centro a sx



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   SpotLight {
       location  0.0 0.0 0.0
       direction 1.0 0.0 0.0
       radius   12.0
       cutOffAngle 0.393
   },
   Inline {
       url "spheres.wrl"
       bboxCenter 0.0 0.0 0.0
       bboxSize   16.0 16.0 1.0
   }
   ]
}
```

Scena VRML

SpotLight con un grande angolo

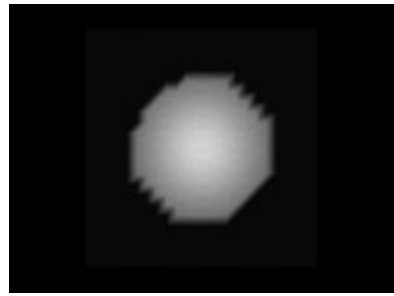


```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    SpotLight {
        location  0.0 0.0 0.0
        direction 1.0 0.0 0.0
        radius    12.0
        cutOffAngle 0.785
    },
    Inline {
        url "spheres.wrl"
        bboxCenter 0.0 0.0 0.0
        bboxSize   16.0 16.0 1.0
    }
    ]
}
```

Scena VRML

Mesh + SpotLight

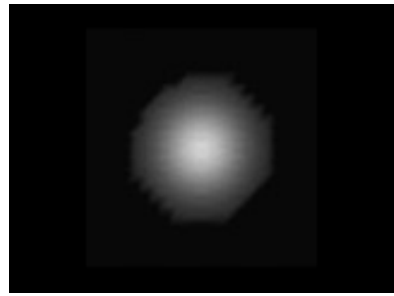
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    SpotLight {
         location  0.0 5.0 0.0
         direction 0.0 -1.0 0.0
         radius    12.0
         intensity 1.0
         cutOffAngle 0.785
         beamWidth   0.785
    },
    Inline {
         url "mesh.wrl"
         bboxCenter 0.0 0.0 0.0
         bboxSize   16.0 1.0 16.0
    }
    ]
}
```



Scena VRML

Mesh + SpotLight

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    SpotLight {
         location  0.0 5.0 0.0
         direction 0.0 -1.0 0.0
         radius    12.0
         intensity 1.0
         cutOffAngle 0.785
         beamWidth   0.392
    },
    Inline {
         url "mesh.wrl"
         bboxCenter 0.0 0.0 0.0
         bboxSize   16.0 1.0 16.0
    }
    ]
}
```



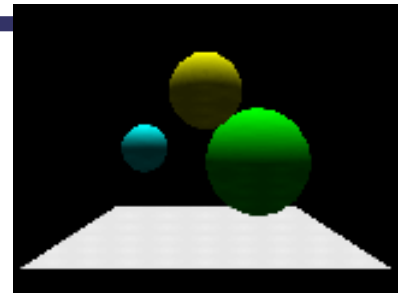
Scena VRML

Mesh + SpotLight

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    SpotLight {
        location  0.0 5.0 0.0
        direction 0.0 -1.0 0.0
        radius    12.0
        intensity 1.0
        cutOffAngle 0.785
        beamWidth   0.09
    },
    Inline {
        url "mesh.wrl"
        bboxCenter 0.0 0.0 0.0
        bboxSize   16.0 1.0 16.0
    }
    ]
}
```

Scena VRML

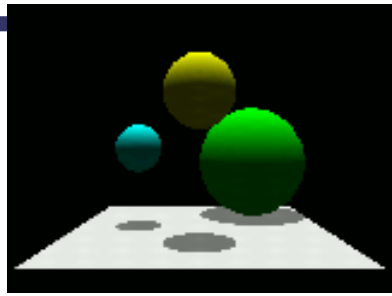
Mondo senza ombre



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
     children [
     # Lighting
     DirectionalLight {
           direction 0.0 -1.0 0.0
           ambientIntensity 0.5
     },
     # Floor
     Shape {
           appearance Appearance {
                 material Material { }
           }
           geometry Box { size 8.0 0.01 8.0 }
     },
     # Spheres
     Transform {
           translation 0.0 4.0 2.0
           children Shape {
                 appearance Appearance {
                       material Material { diffuseColor 1.0 1.0 0.0 }
                 }
                 geometry Sphere { }
           }
     },
     Transform {
           translation 2.0 2.0 -2.0
           children Shape {
                 appearance Appearance {
                       material Material { diffuseColor 0.0 1.0 0.0 }
                 }
                 geometry Sphere { radius 2.0 }
           }
     },
     Transform {
           translation -2.0 2.5 0.0
           children Shape {
                 appearance Appearance {
                       material Material { diffuseColor 0.0 1.0 1.0 }
                 }
                 geometry Sphere { radius 0.75 }
           }
     }
     ]
}
```

Scena VRML

Mondo con ombre



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
	children [
	# Lighting
	DirectionalLight {
		direction 0.0 -1.0 0.0
		ambientIntensity 0.5
	},
	# Floor
	Shape {
		appearance Appearance {
			material Material { }
		}
		geometry Box { size 8.0 0.01 8.0 }
	},
	# Spheres and fake shadows
	Transform {
		translation 0.0 4.0 2.0
		children [
			Shape {
				appearance Appearance {
					material Material { diffuseColor 1.0 1.0 0.0 }
				}
				geometry Sphere { }
			},
		# Shadow
			Transform {
				translation 0.0 -3.95 0.0
				children Shape {
					appearance DEF ShadowAppearance Appearance {
						material Material {
							diffuseColor 0.0 0.0 0.0
							transparency 0.5
						}
					}
					geometry Cylinder {
						height 0.0
						side   FALSE
						bottom FALSE
					}
				}
			}
		]
	},
	Transform {
		translation 2.0 2.0 -2.0
		children [
			Shape {
				appearance Appearance {
					material Material { diffuseColor 0.0 1.0 0.0 }
				}
				geometry Sphere { radius 2.0 }
			},
		# Shadow
			Transform {
				translation 0.0 -1.95 0.0
				children Shape {
					appearance USE ShadowAppearance
					geometry Cylinder {
						radius 2.0
						height 0.0
						side   FALSE
						bottom FALSE
					}
				}
			}
		]
	},
	Transform {
		translation -2.0 2.5 0.0
		children [
			Shape {
				appearance Appearance {
					material Material { diffuseColor 0.0 1.0 1.0 }
				}
				geometry Sphere { radius 0.75 }
			},
		# Shadow
			Transform {
				translation 0.0 -2.45 0.0
				children Shape {
					appearance USE ShadowAppearance
					geometry Cylinder {
						radius 0.75
						height 0.0
						side   FALSE
						bottom FALSE
					}
				}
			}
		]
	}
	]
}
```

Scena VRML