Virtual Reality Modeling Language

VRML

Textures

Texture mapping

- Il mondo reale è straordinariamente pieno di dettagli visivi.
- Se consideriamo un albero, da lontano appare come una massa verde, poi piano piano che ci si avvicina di intravedono i rami, le foglie …
- Avvicinandoci ancora di più vediamo le fumature di colore delle foglie, le nervature, etc
- In Computer Graphics il dettaglio visivo è noto con il termine di **texture**.
- Immaginare di riprodurre in dettaglio l'albero con VRML è impensabile.
- Quello che è possibile è *mappare*, *incollare*, una foto ad una qualsiasi forma VRML: questo procedimento è noto con il termine di **texture mapping**

Texture mapping (i)

- Pur con le limitazioni del caso, descrivere una scena ricca di dettaglio diventa semplice: basta scattare una foto e incollarla ad una semplice forma, quale un quadrato o un rettangolo.
- L'effetto genera l'illusione di una scena reale fino a che non ci si avvicina abbastanza da mettere in evidenza una forma con attaccata un'immagine…
- In VRML si usa il campo **`texture`** del nodo **`Appearance`**, insieme con i nodi **`ImageTexture`**, **`PixelTexture`**, e **`MovieTexture`** per specificare la texture da incollare alla forma VRML.

Texture mapping (ii)

- Una immagine Texture può essere immaginata come una griglia 2-D, con ogni quadratino colorato in modo diverso. Ogni quadratino è chiamato **pixel**, che è l'abbreviazione di *picture element*.
- La dimensione dell'immagine in altezza e larghezza è misurata in numero di pixel.
- Pur potendo essere di ogni dimensione, in VRML le immagini sono molto piccole (128x128 pixel, 64x64 pixel, etc).
- I valori dei pixel sono salvati in un file. Si seleziona il file contenente un'immagine mediante una URL.
- Il file può contenere un'immagine o la sequenza di un filmato, attraverso la quale si possono inserire scene in movimento.

Texture mapping (ii)

- I formati delle immagini che VRML gestisce sono PNG (estensione `.png`), JPG (`.jpg`), GIF (`.gif`); mentre per le animazione è gestito il formato MPEG (`.mov`).
- Alcuni browser supportano anche altri standard, ma occorre considerare che usando formati non generali, si rischia che la scena sia rappresentata solo parzialmente.
- Esistono comunque diversi prodotti freeware o shareware che sono in grado di convertire un'immagine da un formato ad un altro.

formato JPEG

- JPEG è sinonimo di Joint Photographic Experts Group, il nome del Comitato che ha inventato questo formato.
- Il formato grafico è concepito per descrivere immagini di alta qualità utilizzando un algoritmo molto intelligente di memorizzazione delle informazioni
- Questo è il formato meglio supportato da VRML per memorizzare texture non in movimento.

formato GIF

- GIF è sinonimo di Graphical Interchange Format.
- Il formato grafico è stato sviluppato come standard da CompuServe per distribuire le immagini sulla rete e sul Web.
- Il formato GIF incorpora la tecnologia di compressione Lempel-Zev-Welch (LZW), coperta da patent da parte di Unisys dal 1994.
- E' supportato da tutti i browser Web.
- Purtroppo la limitazione derivante dal patent Unisys tende a scoraggiare l'uso del formato GIF, anche se è diventato popolarissimo tra gli utenti.

Formato PNG

- PNG è acronimo di Portable Network Graphics.
- Il formato PNG è stato introdotto per rimpiazzare il formato GIF a causa delle pesanti limitazioni dovute al patent di CompuServe su quest'ultimo.
- Anche se il formato PNG ha una gamma interessantissima di funzionalità, tra cui la possibilità di creare immagini trasparenti.
- E' molto importante adottare questo formato come formato standard nelle proprie applicazioni, perché ricco di funzionalità e completamente libero da vincoli.
- L'algoritmo di compressione utilizzto è molto eff ciente e limita alcuni errori dei formati GIF e JPEG.

Formato MPEG

- MPEG è acronimo di Moving Picture Experts Group, il nome degli inventori di questo formato.
- A differenza degli altri formati, MPEG è concepito per archiviare immagini di alta qualità e in movimento, usando diversi eff cienti schemi di compressione.
- Insieme alle immagini codif cate in un opportuno formato video è in grado di memorizzare anche i suoni che accompagnano le immagini.
- MPEG-1 è il solo formato supportato da VRML per memorizzare texture in movimento

Formato MPEG (i)

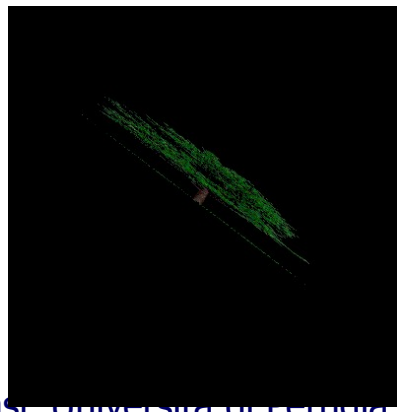
- Il formato MPEG è codif cato da un gruppo internazionale di standardizzazione ed è in continua evoluzione per la memorizzazione e la trasmissione di video e audio.
- MPEG-1, la prima versione è disegnata per eseguire eff cacemente audio e video da Hard disk o CD-ROM.
- La seconda versione, MPEG-2, estende le potenzialità del formato per essere riprodotto attraverso una rete lenta, quale la connessione via modem.
- MPEG-3 è disegnato per gestire la codif ca del segnale televisivo ad alta def nizione (HDTV), è stata rinominata e inclusa in MPEG-2
- MPEG-4 è in corso di def nizione ed è indirizzato a dispositivi video molto lenti, quali i videotelefoni.
- MPEG-1 e MPEG-2 esistono nelle varianti Systems, Audio, Video. MPEG-1 Systems codif ca video e audio sincronizzato, MPEG-1 Audio codif ca solo audio, MPEG-1 Video codif ca video soltanto.
- MPEG-2 e MPEG-1 Audio non sono supportati da VRML.

Tipi di immagine

- Le immagini texture possono essere salvate sia a colori che come scala di grigi. Si possono usare entrambe come texture di forme VRML.
- Quando per una forma si usa una texture colorata, questa risulta essere come dipinta. Eventuali specif che del campo `diffuseColor` del campo `Material` vengono ignorate. Gli altri campi del nodo `Material` però ancora vengono applicati, permettendo di realizzare forme semitrasparenti o a specchio.
- I formati JPG, GIF e PNG supportano immagini a colori e in bianco e nero, mentre il formato MPG supporta solo immagini a colori.

Immagini con trasparenze

- Oltre al colore di ogni pixel, alcuni formati graf ci sono in grado di specif carne il grado di trasparenza.
- Il grado di trasparenza del pixel sta ad indicare se il pixel deve essere opaco, trasparente o avere un comportamento intermedio.
- Usando la trasparenza dei pixel si possono creare delle immagini texture con dei buchi all'interno.
- Quando c'e' un buco trasparente nell'immagine texture, questo genera un buco trasparente anche nella forma.

Immagini con trasparenze

- Quando si usa una immagine con pixel trasparenti, la proprietà di trasparenza dell'immagine prevale sulle caratteristiche specif cate nel campo `transparency` del nodo `Material` .
- I restanti campi del nodo `Material` rimangono attivi e validi.
- Il formato PNG supporta valori di trasparenza dei pixel. I formati JPG e MPEG non possono memorizzare questa caratteristica.
- I formati GIF in genere non prevedono di poter gestire questo aspetto per default, ma alcune versioni possono farlo.

Nodi Texture

- VRML fornisce 3 nodi per specificare una texture da associare ad una forma: **`Imagetexture`**, **`Pixeltexture`** e **`Movietexture`**.
- Ciascuno di questi nodi può essere usato come valore di un campo **`texture`** del nodo **`Appearance`**.
- Il nodo **`Imagetexture`** è il più comune per il texture mapping. Utilizzando questo nodo, si fornisce la URL del file contenente l'immagine in uno dei formati PNG, JPG e GIF. Il browser VRML carica l'immagine dal file e la applica alla forma.
- Il nodo **`Pixeltexture`** fornisce una via alternativa per specificare un'immagine texture. Invece di fornire la URL del file si dichiarano esplicitamente i valori dei pixel dell'immagine, uno ad uno, da sx a dx, dal basso in alto dell'immagine.
- Il nodo **`Movietexture`** viene utilizzato per specificare una texture contenente un video in formato MPEG. Il nodo consente il controllo sul quando il video viene riprodotto e a che velocità.

Applicazione delle texture

- Il modo con cui viene applicata un'immagine texture ad una forma dipende dalla sua geometria:
  - Nodo **`Box`**: il nodo costruisce una scatola con 4 facce laterali, 1 sopra e una sotto. Il texture mapping viene fatto così:
    - L'immagine appare allineata a destra verso l'alto, nelle facce di fronte, dietro a destra e a sinistra se la faccia superiore punta nella direzione positiva dell'asse Y
    - Nella faccia superiore l'immagine appare allineata a destra, verso l'alto se la faccia viene ruotata in faccia all'osservatore, mantenendo la faccia di sx a sx e quella di dx a dx.
    - Nella faccia inferiore l'immagine appare allineata a destra, verso l'alto se la faccia viene ruotata in faccia all'osservatore, mantenendo la faccia di sx a sx e quella di dx a dx.

Applicazione delle texture (i)

- Nodo **`Cone`**: il nodo costruisce un cono con superfi cie laterale e base. Il texture mapping viene fatto così:
  - Sulla superfi cie laterale del cono la texture viene incollata intorno alla stessa in senso antiorario cominciando da dietro (con l'asse del cono lungo la direzione positiva dell'asse Y). L'immagine viene incollata e tagliata in cima in modo da aderire alla forma conica.
  - Nella base l'immagine appare allineata a destra, verso l'alto se la base viene ruotata in faccia all'osservatore, senza ruotare il cono rispetto al proprio asse.

Applicazione delle texture (ii)

- Nodo `Cylinder`: il nodo costruisce un cilindro con superfcie laterale, faccia superiore e faccia inferiore. Il texture mapping viene fatto così:
  - Sulla superfcie laterale del cono la texture viene incollata intorno alla superfcie in senso antiorario cominciando da dietro (con l'asse del cilindro lungo la direzione positiva dell'asse Y). E' come se i bordi sx e dx dell'immagine si ricongiungessero verticalmente nel retro del cilindro.
  - Nella faccia superiore l'immagine appare allineata a destra, verso l'alto se la faccia viene ruotata in faccia all'osservatore, senza ruotare il cilindro rispetto al proprio asse.
  - Nella base l'immagine appare allineata a destra, verso l'alto se la base viene ruotata in faccia all'osservatore, senza ruotare il cilindro rispetto al proprio asse.

Applicazione delle texture (iii)

- Nodo `Sphere`: il nodo costruisce una sfera. Il texture mapping viene fatto così:
  - La texture viene incollata sulla superfcie della sfera tutta intorno, in senso antiorario cominciando da dietro. E' come se i bordi sx e dx dell'immagine si ricongiungano verticalmente nel retro della sfera ed ai poli l'immagine venga raggruppata per aderire alla forma .
- Nodo `Text`: il nodo costruisce testi. Il texture mapping viene fatto così:
  - La texture viene incollata sulla forma dei caratteri del testo, e stirata o compressa sia orizzontalmente che verticalmente, in modo da combaciare con le dimensioni della font utilizzata. Ogni carattere agisce poi come uno stampino che ritaglia l'immagine di modo che i caratteri vengano completamente rivestiti dell'immagine. Eventuali altre copie dell'immagine vengono affancate tra loro per costruire una serie infnita di ffgure in grado di ricoprire il testo.

Applicazione delle texture (iv)

- Nodi **`PointSet`** e **`IndexedLineSet`**: essendo punti e linee entità geometriche prive di dimensione, i due nodi non possono essere ricoperti da texture.
- Nodo **`IndexedFaceSet`**: il nodo costruisce delle forme costituite da facce. Il texture mapping viene fatto sull'intera superf cie così:
  - Il browser VRML calcola un bounding box che contiene la forma. L'immagine texture viene proiettata sulla forma alla stessa maniera di quando si usa un proiettore.
  - Le immagini texture vengono raramente incollate alle facce con questo metodo automatico. Si usa piuttosto una modalità esplicita, utilizzando le proprietà delle coordinate di texture che vedremo più avanti.

Applicazione delle texture (v)

- Nodo **`ElevationGrid`**: il nodo costruisce delle forme partendo dalle altezze di una griglia di punti. Il texture mapping viene fatto sull'intera griglia così:
  - L'immagine viene incollata alla forma in modo da rispettare le dimensioni della griglia.
  - Le immagini texture vengono raramente incollate alle facce con questo metodo automatico. Si usa piuttosto una modalità esplicita, utilizzando le proprietà delle coordinate di texture che vedremo più avanti.

Applicazione delle texture (vi)

- Nodo **`Extrusion`**: il nodo costruisce delle forme propagando una forma 2-D lungo una direttrice di propagazione. Il texture mapping viene fatto sull'intera superficie così:
  - L'immagine viene incollata intorno alla forma estrusa, fino al dietro.
  - L'estremo sinistro della texture viene allineato con la prima coordinata della cross-section, mentre quello destro viene allineato con l'ultima coordinata.

Il nodo Appearance

Il nodo **Appearance** specif ca gli attributi di apparenza delle forme e può essere usato come valore del campo **appearance** del nodo **Shape**.

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

Il nodo Appearance (i)

- Il valore dell'exposed-field **texture** può essere un nodo che consente la specificazione di una texture da incollare ad una forma, come **ImageTexture**, **PixelTexture** e **MovieTexture**.
- Il valore **NULL** di default sta ad indicare che per senza specifiche esplicite non vengono applicate texture.
- La texture può essere modificata inviando un valore all'eventIn implicito **set_texture**, il nuovo valore verrà propagato tra i nodi attraverso l'eventOut implicito **texture_changed** dell'exposed field.
- L'altro campo **textureTrasform**, verrà discusso più avanti.

Il nodo ImageTexture

Il nodo **ImageTexture** specifica gli attributi cin cui viene fatto texture-mapping e può essere usato come valore del campo **texture** del nodo **Appearance**.

```
ImageTexture {
        url             []          # exposedField MFString
        repeatS         TRUE        # field SFBool
        repeatT         TRUE        # field SFBool
         }
```

Il nodo `ImageTexture` (i)

- Il valore dell'exposed-field **`url`** indica una lista di URL con priorità ordinate dalla più alta alla più bassa. Il browser VRML cercherà di aprire il primo file della lista, accedendo al primo URL, se questo non verrà risolto, tenterà di accedere il secondo della lista, etc.
- Quando il file verrà trovato, sarà utilizzato per costruire la texture della forma.
- Per default il campo **`url`** ha una lista vuota.
- La lista può essere modificata inviando volori all'eventIn implicito **`set_url`**; il nuovo valore sarà propagato attraverso l'eventOut implicito **`url_changed`**.
- Le immagini texture devono essere file grafici memorizzati nei formati PNG, JPG o GIF e devono specificare pixel per pixel il colore o la scala dei grigi e la trasparenza dei pixel

Il nodo `ImageTexture` (ii)

- I colori della texture o la scala dei grigi controllano il colore della forma, sovrapponendo o combinandosi con il colore specificato nei nodi **`Material`** e **`Color`**, nella forma seguente:
  - Se la texture è un'immagine a colori, il colore della texture prevale su qualsiasi colore specificato nel nodo **`Color`** o nel campo **`diffuseColor`** del nodo **`Material`**.
  - Se la texture è un'immagine in bianco e nero, i valori della scala di grigio della texture vengono moltiplicati con i valori del nodo **`Color`** o con quelli specificati nel campo **`diffuseColor`** del nodo **`Material`**.
- In entrambi i casi se la forma non ha specificato un colore, I colori della texture vengono utilizzati come emissive color e rendono la forma illuminata con I clori della texture.

Il nodo `ImageTexture` (iii)

- Se invece la forma ha codificato il nodo **`Appearance`** ed ha un nodo **`Material`** con un valore non nullo di **`diffuseColor`**, la forma viene ombreggiata con i colori della texture.
- I valori di trasparenza specificati eventualmente nella texture determinano il comportamento della forma, prevalendo su qualsiasi valore specificato nel nodo **`Material`**.
- I rimanenti campi **`repeatS`** e **`repeatT`** verranno discussi più avanti.

Il nodo `PixelTexture`

Il nodo **`PixelTexture`** specifica gli attributi cin cui viene fatto texture-mapping e può essere usato come valore del campo **`texture`** del nodo **`Appearance`**.

```
PixelTexture {
        image           0 0 0       # exposedField SFImage
        repeatS         TRUE        # field SFBool
        repeatT         TRUE        # field SFBool
         }
```

Il nodo `PixelTexture` (i)

- I valori del campo **`image`** specif cano le dimensioni ed i valori dei pixel dell'immagine usata per colorare la forma. Il signif cato dei primi tre valori del campo **`image`** sono:
  1. La larghezza dell'immagine in pixel
  2. L'altezza dell'immagine in pixel
  3. Il numero di bytes per ogni pixel. I valori validi per questo numero sono i seguenti, essendo alfa il grado di trasparenza del pixel:

Numero di bytes	Significato
0	Disabilita texture della forma
1	Scala di grigi
2	Scala di grigi con alfa
3	RGB
4	RGB con alfa

Il nodo `PixelTexture` (ii)

- I numeri che seguono i primi tre rappresentano i valori interi di ogni pixel dell'immagine (se il numero di bytes per pixel è maggiore di 0).
- Il valore del primo pixel fornisce il colore per il pixel più a sx e più in basso dell'immagine.
- I seguenti valori procedono linea per linea da sx a dx, dal basso verso l'alto.
- I valori interi dei pixel in genere sono spef f cati con notazione esadecimale.

Il nodo MovieTexture

Il nodo **MovieTexture** specifica gli attributi con cui viene fatto texture-mapping e può essere usato come valore del campo **texture** del nodo **Appearance**.

```
MovieTexture {
        url             []          # exposedField MFString
        loop            FALSE       # exposedField SFBool
        speed           1.0         # exposedField SFFloat
        startTime       0.0         # exposedField SFFloat
        stopTime        0.0         # exposedField SFFloat
        repeatS         TRUE        # field SFBool
        repeatT         TRUE        # field SFBool
        isActive                    # eventOut SFBool
        duration_changed            # eventOut SFFloat
         }
```

Il nodo `MovieTexture` (i)

- Quando si usa un file video contenente audio, il nodo **`MovieTexture`** specifica il suono che può essere riprodotto quando il nodo è usato come valore del campo **`source`** del nodo **`Sound`**.
- I file video devono essere in formato MPEG-1. Sono supportate le varienti MPEG-1 Systems (audio e video) e MPEG-1 Video (solo video).
- Il numero di frames presenti nel file video ne determina la durata. Appena caricato in memoria, il browser determina la durata del video e propaga l'informazione attraverso l'eventOut **`duration_changed`**.
- Tale valore è indipendente dal valore del campo **`speed`**.
- Se per qualche motivo il browser non riesce a determinare tale valore, ritorna il valore **`-1`**.

Il nodo `MovieTexture` (ii)

- Il valore dell'exposed-field `startTime` specifica il tempo al quale, deve iniziare la riproduzione del video. 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 valore dell'exposed-field `stopTime` specifica il tempo al quale cessa la riproduzione del video. Il suo valore è un tempo assoluto ed il default è `0.0` sec.
- Il valore dell'exposed-field `loop` specifica se il video viene riprodotto all'infinito o no. Il default è `FALSE` , se il valore è `TRUE` il video viene riprodotto una volta e poi si ferma.

Il nodo `MovieTexture` (iii)

- Fino a quando non viene attivata la riproduzione il video rimane fermo nella prima immagine (o nell'ultima, se duration ha un valore negativo) e questa immagine è usata come texture della forma.
- Quando inizia la riproduzione del video l'eventOut **`isActive`** diventa **`TRUE`** e la forma viene colorata con le frame del video, secondo la velocità indicata nel parametro **`speed`**.
- Il valore dell'exposed-field **`speed`** può essere modificato attraverso l'eventIn implicito **`set_speed`**; se il nuovo valore è ricevuto mentre la riproduzione è in corso, esso viene ignorato.
- Il valore del campo **`loop`** può essere modificato attraverso l'eventIn implicito **`set_loop`**; se il valore **`FALSE`** viene ricevuto mentre è in corso la riproduzione, questa continua fino all'ultimo frame e poi si ferma. Se il valore **`TRUE`** viene ricevuto durante la riproduzione, questa continuerà ciclicamente.

Texture-mapping del cubo

```
#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 { }
    texture ImageTexture {
        url "testimg.jpg"
    }
    }
    geometry Box { }
}
```

testimg.jpg

Texture-mapping del cono

```
#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 { }
    texture ImageTexture {
        url "testimg.jpg"
    }
    }
    geometry Cone { }
}
```

testimg.jpg

Scena VRML

Texture-mapping del 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 { }
    texture ImageTexture {
        url "testimg.jpg"
    }
    }
    geometry Cylinder { }
}
```

testimg.jpg

Scena VRML

Texture-mapping della sfera

```
#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 { }
    texture ImageTexture {
        url "testimg.jpg"
    }
    }
    geometry Sphere { }
}
```

testimg.jpg

Scena VRML

Texture-mapping su facce separate

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Can top
    Shape {
        appearance Appearance {
            material Material { }
            texture ImageTexture {
                url " cantop.jpg "
            }
        }
        geometry Cylinder {
            bottom FALSE
            side FALSE
            height 2.7
        }
    }
    # Can bottom
    Shape {
        appearance Appearance {
            material Material { }
            texture ImageTexture {
                url "canbot.jpg"
            }
        }
        geometry Cylinder {
            top FALSE
            side FALSE
            height 2.7
        }
    }
    # Can side
    Shape {
        appearance Appearance {
            material Material { }
            texture ImageTexture {
                url "canlabel.jpg"
            }
        }
        geometry Cylinder {
            top FALSE
            bottom FALSE
            height 2.7
        }
    }
    ]
}
```

canlabel.jpg

canbot.jpg

cantop.jpg

Scena VRML

Texture-mapping su forme di testo

```
#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 { }
    texture ImageTexture {
         url "brick.jpg"
    }
    }
    geometry Text {
    fontStyle FontStyle {
         style "BOLD"
    }
    string [ "Qwerty", "0123" ]
    }
}
```

brick.jpg

Scena VRML

Texture-mapping sul nodo IndexFaceSet

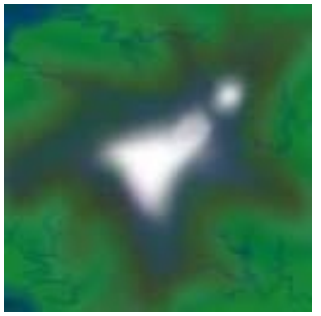
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L.
    Moreland
Shape {
    appearance Appearance {
    material Material { }
    texture ImageTexture {
         url "bolt2.jpg"
    }
    }
    geometry IndexedFaceSet {
    coord Coordinate {
         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
    }
}
```

bolt2.jpg

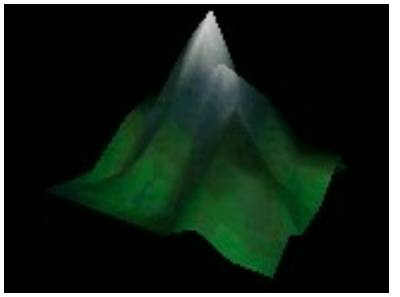
Scena VRML

Texture-mapping sul nodo ElevationGrid

```
#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 { }
    texture ImageTexture {
         url "mount.jpg"
    }
    }
    geometry ElevationGrid {
    xDimension 9
    zDimension 9
    xSpacing   1.0
    zSpacing   1.0
    solid      FALSE
    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,
    ]
    }
}
```



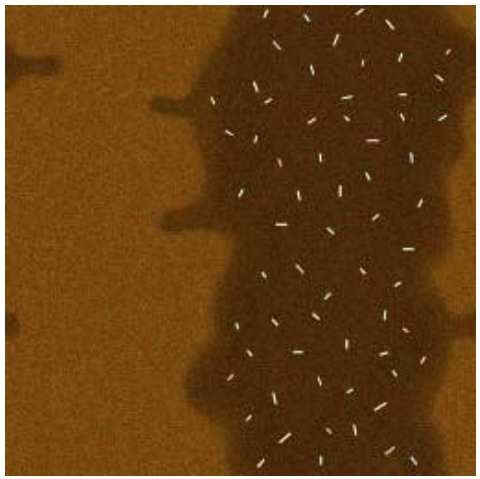
mount.jpg



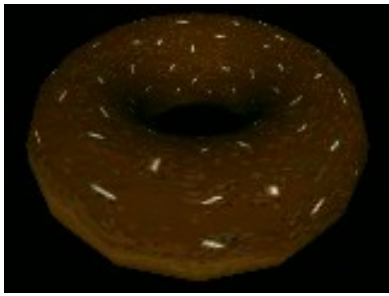
Scena VRML

Texture-mapping su forme estruse

```
#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 { }
    texture ImageTexture {
         url "icing.jpg"
    }
    }
    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,
    ]
    }
}
```



icing.jpg



Scena VRML

Creazione di buchi usando la trasparenza dei pixel

```
#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 { }
         }
         geometry IndexedFaceSet {
              coord Coordinate {
                   point [
                        -5.0 0.0  5.0,  5.0 0.0  5.0,
                         5.0 0.0 -5.0, -5.0 0.0 -5.0,
                   ]
              }
              coordIndex [ 0, 1, 2, 3 ]
              solid FALSE
         }
    },
    # Tree face
    Shape {
         appearance Appearance {
              # No material, use emissive texturing
              texture ImageTexture {
                   url "tree1.png"
              }
         }
         geometry IndexedFaceSet {
              coord Coordinate {
                   point [
                        -1.51 0.0 0.0,   1.51 0.0 0.0,
                         1.51 3.0 0.0,  -1.51 3.0 0.0,
                   ]
              }
              coordIndex [ 0, 1, 2, 3 ]
              solid FALSE
         }
    }
    ]
}
```

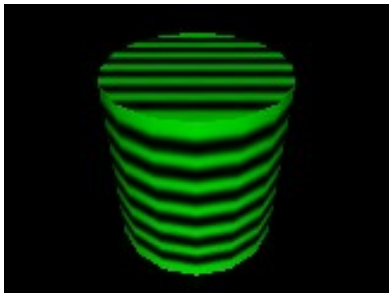
tree1.png

Scena VRML

Texture in scala di grigi

```
#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 0.0
    }
    texture ImageTexture {
         url "lines_g.jpg"
    }
    }
    geometry Cylinder {
    height 2.0
    radius 1.0
    }
}
```

lines_g.jpg



Scena VRML

Colorazione di Texture in scala di grigi

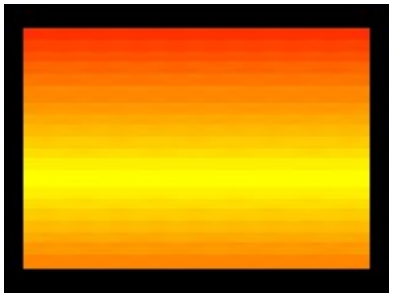
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
      children [
      # Top left
      Shape {
              appearance Appearance {
                      material Material { diffuseColor 1.0 0.35 0.23 }
                      texture DEF wood ImageTexture { url "wood_g.jpg" }
              }
              geometry DEF square IndexedFaceSet {
                      coord Coordinate {
                              point [
                                      0.0 1.0 0.0,  0.0 0.0 0.0,
                                      1.1 0.0 0.0,  1.1 1.0 0.0
                              ]
                      }
                      coordIndex [ 0, 1, 2, 3 ]
              }
      },
      # Top center
      Transform {
              translation 1.25 0.0 0.0
              children Shape {
                      appearance Appearance {
                              material Material { diffuseColor 1.0 0.45 0.23 }
                              texture USE wood
                      }
                      geometry USE square
              }
      },
      # Top right
      Transform {
              translation 2.50 0.0 0.0
              children Shape {
                      appearance Appearance {
                              material Material { diffuseColor 1.0 0.55 0.23 }
                              texture USE wood
                      }
                      geometry USE square
              }
      },
      # Bottom left
      Transform {
              translation 0.0 -1.25 0.0
              children Shape {
                      appearance Appearance {
                              material Material { diffuseColor 1.0 0.65 0.53 }
                              texture USE wood
                      }
                      geometry USE square
              }
      },
      # Bottom center
      Transform {
              translation 1.25 -1.25 0.0
              children Shape {
                      appearance Appearance {
                              material Material { diffuseColor 1.0 0.55 0.43 }
                              texture USE wood
                      }
                      geometry USE square
              }
      },
      # Bottom right
      Transform {
              translation 2.50 -1.25 0.0
              children Shape {
                      appearance Appearance {
                              material Material { diffuseColor 1.0 0.55 0.53 }
                              texture USE wood
                      }
                      geometry USE square
              }
      }
      ]
}
```

wood_g.jpg

Scena VRML

Nodo `PixelTexture`

```
#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 { }
    texture PixelTexture {
         image 1 2 3       # width, height, 3-byte RGB image
              0xFFFF00  # yellow at the bottom
              0xFF0000  # red at the top
    }
    }
    geometry IndexedFaceSet {
    coord Coordinate {
         point [
              -1.5 -1.0 0.0,   1.5 -1.0 0.0,
               1.5  1.0 0.0,  -1.5  1.0 0.0,
         ]
    }
    coordIndex [ 0, 1, 2, 3 ]
    solid FALSE
    }
}
```



Scena VRML

Nodo MovieTexture

```
#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 0.0 0.7 0.0
                  }
            }
            geometry Box { size 10.0 0.01 10.0 }
      },
      # Well wall
      Shape {
            appearance Appearance {
                  material Material { }
                  texture ImageTexture {
                        url "wellwall.jpg"
                  }
            }
            geometry Extrusion {
                  creaseAngle 1.57
                  beginCap FALSE
                  endCap   FALSE
                  crossSection [
                  # upside-down U-shape
                           0.4  0.0,
                           0.4 -0.7,
                          -0.4 -0.7,
                          -0.4  0.0,
                  ]
                  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,
                  ]
            }
      },
      # Well water
      Shape {
            appearance Appearance {
                  # No material, use emissive texturing
                  texture MovieTexture {
                        url "wrlpool.mpg"
                        loop TRUE
                  }
            }
            geometry IndexedFaceSet {
                  solid FALSE
                  coord Coordinate {
                        point [
                        # Circle
                                 2.00 0.6  0.00,   1.85 0.6 0.67,
                                 1.41 0.6  1.41,   0.67 0.6 1.85,
                                 0.00 0.6  2.00,  -0.67 0.6 1.85,
                                -1.41 0.6  1.41,  -1.85 0.6 0.67,
                                -2.00 0.6  0.00,  -1.85 0.6 -0.67,
                                -1.41 0.6 -1.41,  -0.67 0.6 -1.85,
                                 0.00 0.6 -2.00,   0.67 0.6 -1.85,
                                 1.41 0.6 -1.41,   1.85 0.6 -0.67,
                                 2.00 0.6  0.00,
                        ]
                  }
                  coordIndex [
                        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                        11, 12, 13, 14, 15, 16
                  ]
            }
      }
      ]
}
```

Wrlpool.mpg

Scena VRML

Virtual Reality Modeling Language

VRML

Controllo del Texture-mapping

Controllo del Texture mapping

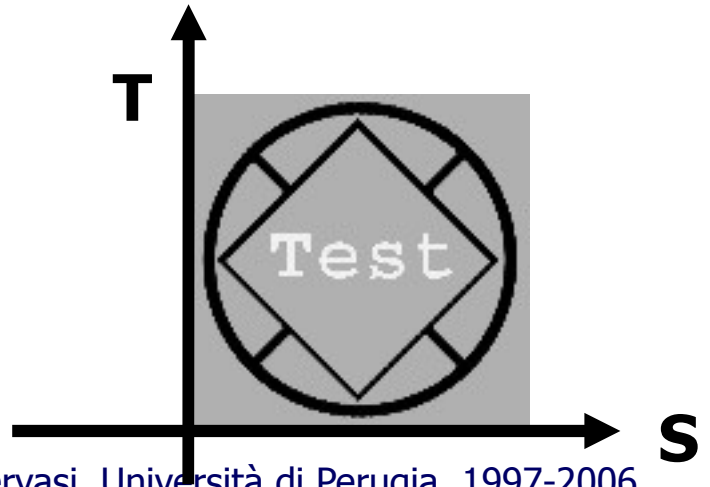
- Mediante il controllo del texture-mapping si può controllare la posizione di una immagine in una forma, allargarla o restringerla, ruotarla o ripeterla varie volte sulla superf cie di una forma.
- Si può prendere una piccola foto contenente una porzione di mattoni e poi ripetere l'immagine in modo da ricoprire le pareti di una casa…
- Il controllo del texture-mapping consente anche di estrarre da una foto una porzione di immagine e di usare questa porzione per ricoprire una forma, invece dell'intera immagine.
- Utilizzando il nodo **`TextureCoordinate`** si può utilizzare una porzione di immagine, mentre usando **`TextureTransform`** si possono traslare, ruotare e scalare coordinate di texture.

Controllo del Texture mapping (i)

- Il processo di texture-mapping va visto per gradi: prima si usano i nodi **`ImageTexture`**, **`PixelTexture`** e **`MovieTexture`** per selezionare l'immagine (texture), che per default verrebbe incollata su tutta la superf cie nel modo visto.
- Successivamente si può modif care tale comportamento standard e operare come con un coltello, sezionando porzioni di immagine, ripetendo la porzione, ruotandola, scalandola, etc…
- Utilizzando le coordinate di texture 2-D si specif ca la porzione di immagine che interessa.
- La porzione di immagine viene incollata alla forma facendo coincidere gli angoli della faccia (che devono essere di uguale numero).

Coordinate di Texture

- La specifica delle coordinate di texture avviene elencando i vertici 2-D che uniti tra loro definiscono la porzione di immagine da utilizzare.
- Ogni coordinata 2-D specifica una posizione in un sistema di coordinate di Texture, dove le distanze sono misurate nelle direzioni S e T.
- L'origine del sistema di coordinate coincide sempre con l'angolo dell'immagine in basso a sx.



Coordinate Texture (i)

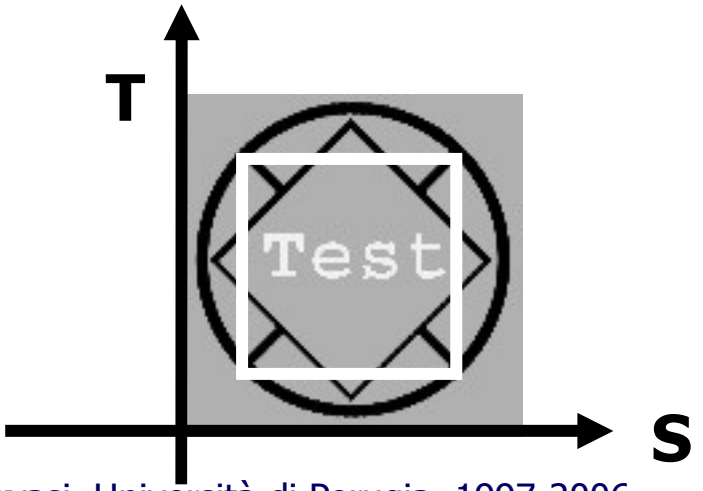
- La coordinata di texture S varia dal valore `0.0` (al bordo sx) a `1.0` (al bordo dx).
- Analogamente la coordinata T varia dal valore `0.0` (al bordo inferiore) al valore `1.0` (al bordo superiore).
- Questo intervallo di valori di S e T è indipendente dalle dimensioni attuali dell'immagine.
- La coordinata di texture `(1.0,1.0)` è sempre l'angolo superiore dx, mentre la coordinata `(0.5,0.5)` è sempre il centro dell'immagine, a prescindere dalle dimensioni attuali dell'immagine.

Coordinate di Texture (ii)

- Per ritagliare una porzione di immagine occorre fornire una lista di coordinate 2-D di texture, del tipo:

Indice della coordinata di texture	Coordinata
0	0.2 0.2
1	0.8 0.2
2	0.8 0.8
3	0.2 0.8

che corrisponde alla porzione di immagine seguente:



Applicare una texture

- Usando il metodo visto è possibile ritagliare una porzione di immagine.
- Per applicare il pezzo di immagine ad una faccia di una forma, ogni angolo dell'immagine è collegato ad uno spigolo della faccia. Se le dimensioni tra immagine e faccia differiscono, l'immagine viene stirata o compressa in modo da coincidere con la forma.
- Figura e faccia devono avere lo stesso numero di coordinate.
- Si può ovviamente appiccicare una immagine quadrata ad una forma rettangolare, ma non si può appiccicare un'immagine triangolare ad una forma quadrata!

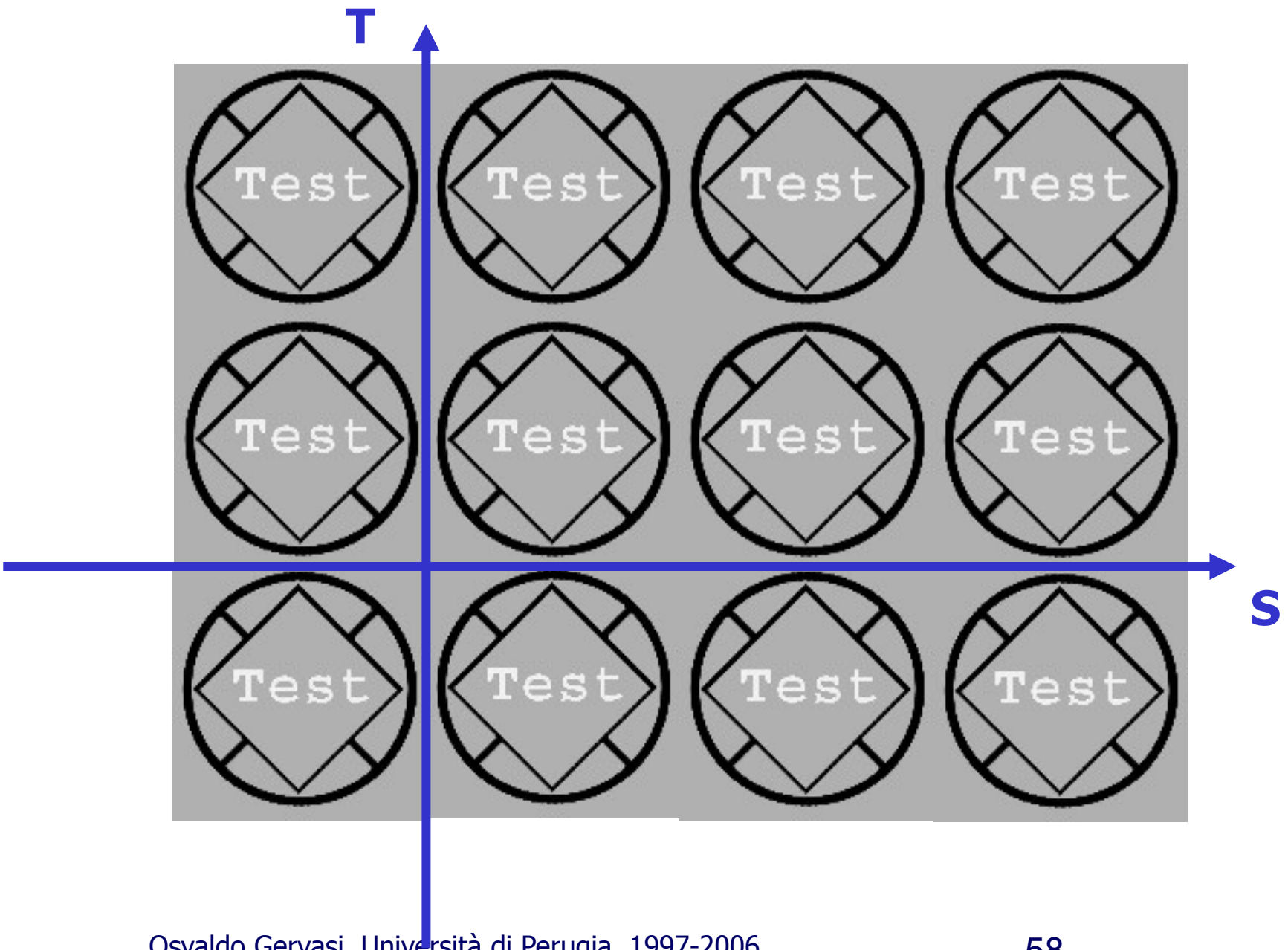
Applicare una texture (i)

- Quando le coordinate di texture sono create automaticamente da VRML, queste vengono allineate alle facce della forma in modo automatico. Ciò consente di applicare immagini a forme quali box, text, elevation grid, etc, senza specif care le coordinate di texture.
- Per forme create con il nodo **`IndexedFaceSet`** è possibile specif care una lista di coordinate di texture indice, una per indice di faccia, nel campo **`texCoordIndex`** del nodo **`IndexedFaceSet`**.
- La prima coordinata indice indica la coordinata di texture associata alla prima coordinata della prima faccia, la seconda coordinata di texture corrisponde alla seconda coordinata della faccia, etc
- Questo processo di assegnazione delle coordinate di texture alle coordinate della faccia è simile all'assegnazione di colori alle coordinate di ciascuna faccia

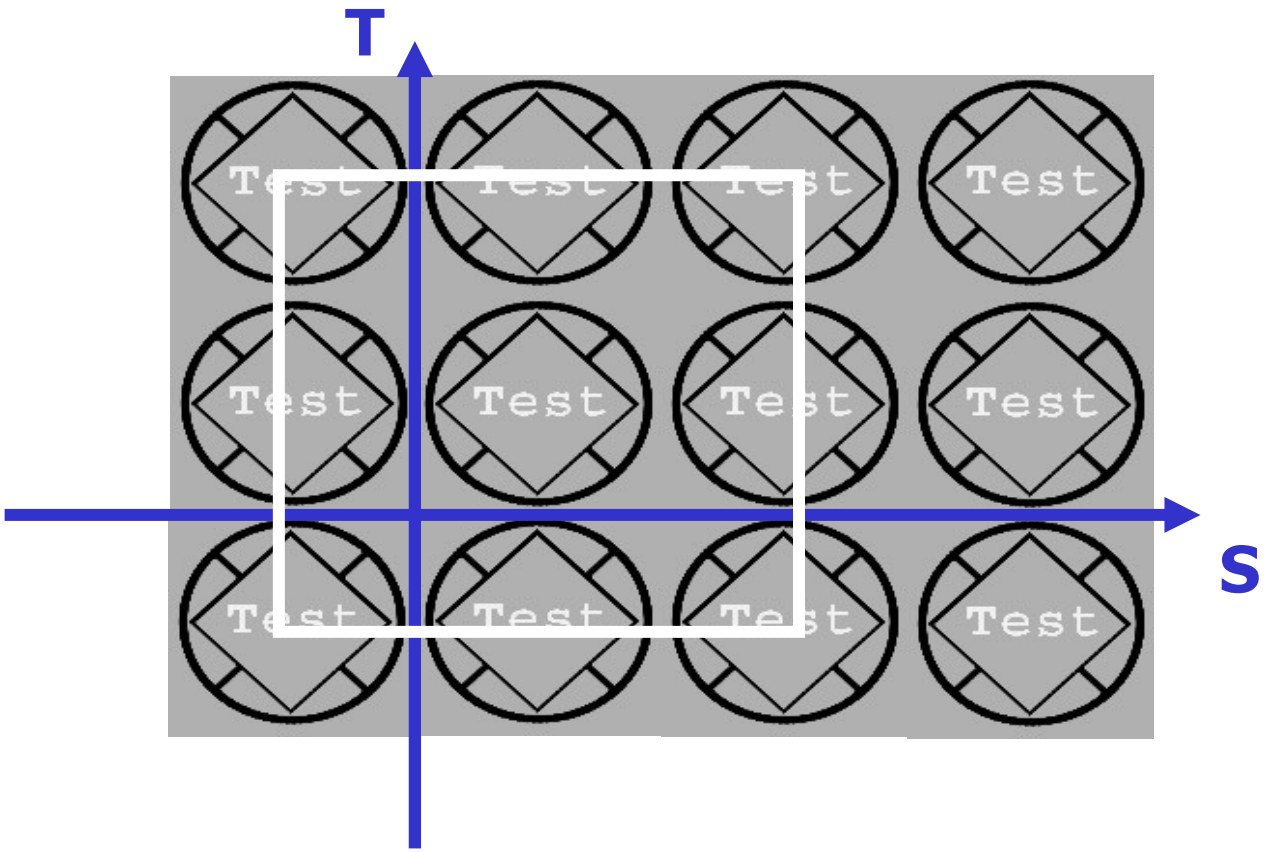
Spostamento della coordinata di texture

- Le coordinate S e T variano da `0.0` a `1.0` all'interno dell'immagine.
- E' possibile selezionare coordinate di texture all'esterno dell'immagine, spostando la coordinata in un'area dove dobbiamo immaginare esista una replica dell'immagine stessa.
- La coordinata S si sposta pertanto dal bordo dx al sx, mentre la coordinata T dal bordo in alto a quello in basso.
- Si deve pensare all'immagine originale come a una sequenza infinita di immagini da sx a dx e dal basso in alto

Spostamento della coordinata di texture (i)



Spostamento della coordinata di texture (ii)



Indice della coordinata di texture	Coordinata
0	-0.5 -0.5
1	1.5 -0.5
2	1.5 1.5
3	-0.5 1.5

Spostamento della coordinata di texture (iii)

- Si può usare lo spostamento delle coordinate di texture per creare strutture ripetitive su una forma.
- Per default lo spostamento delle coordinate di texture è attivo.
- Si può disabilitare lo spostamento della coordinata S o della T o di entrambe, ponendo a **`FALSE`** il campo **`repeatS`** e/o **`repeatT`** dei nodi **`ImageTexture`**, **`PixelTexture`** e **`MovieTexture`**.
- Ovviamente alcune immagini si prestano ad essere ripetute, altre no. Probabilmente un'immagine acquisita con scanner necessita di un piccolo intervento di fotoritocco affinché sia adatta ad essere ripetuta (aggiustamento di colori e forme ai bordi).

Blocco della coordinata di texture

- Se si desidera evitare la deformazione dell'immagine quando la texture viene applicata ad una forma diversa da quella dell'immagine e se si vuole evitare il comportamento dello spostamento della coordinata di texture visto, si possono adottare delle coordinate di texture minori di `0.0` e maggiori di `1.0`, disabilitando lo spostamento delle coordinate (blocco delle coordinate).
- Per migliorare l'effetto di blocco delle coordinate si può aggiungere ai bordi dell'immagine un bordo solido
- Sarà il bordo introdotto che sarà stirato in modo da ricoprire tutta la forma.

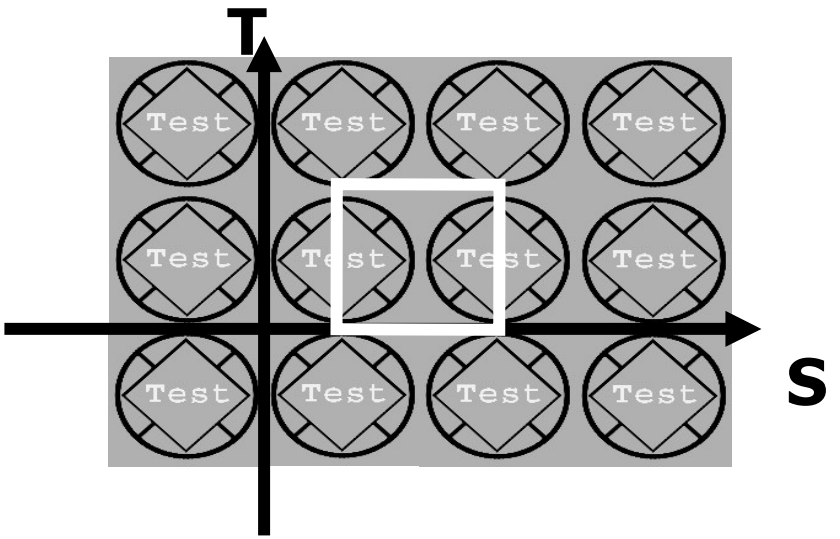
Trasformazione di coordinate di texture

- Utilizzando il nodo **TextureTransform** si può creare un nuovo sistema di coordinate di texture, che può essere usato per ritagliare un'immagine.
- Variando i valori dei campi **translation**, **rotation** e **scale** del nodo **TextureTransform** si possono posizionare, orientare e ridimensionare le forme con cui ritagliare le immagini texture, prima di applicarle alle forme.
- Il nodo **TextureTransform** assegna un valore al campo **textureTransform** del nodo **Appearance**.
- La trasformazione delle coordinate di texture influenza le coordinate se vengono fornite attraverso un nodo **TextureCoordinate** per un nodo **IndexedFaceSet** e se sono automaticamente create per le forme geometriche predefinite. Questo consente di influenzare il texture mapping anche nei casi in cui VRML produce le proprie coordinate di texture come nei nodi **Box**, **Cylinder**, **Cone**, **Sphere**, **Text**, **ElevationGrid** e **Extrusion**.

Traslazione di coordinate di texture

- Le traslazioni delle coordinate di texture sono espresse in forma di distanze dalle direzioni S e T.
- La traslazione sposta la finestra logica con cui viene ritagliata la figura di texture prime di essere attaccata alla forma

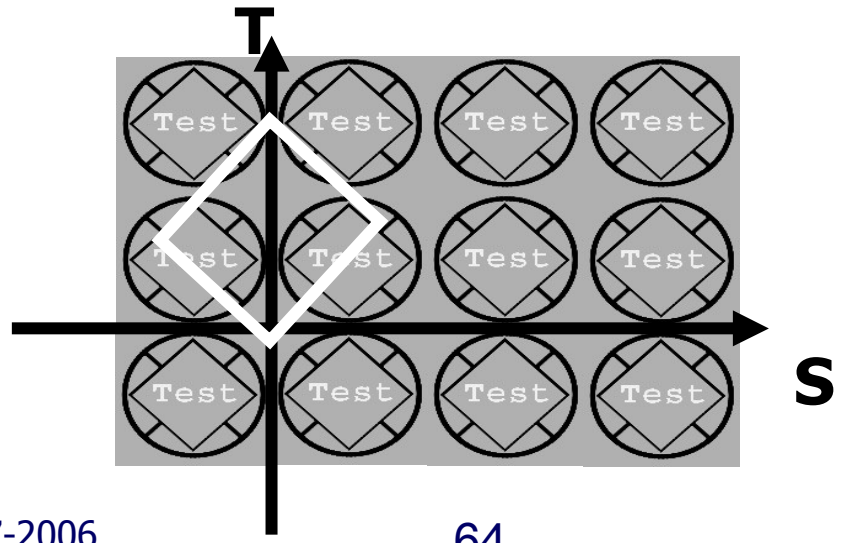
Le coordinate di texture, traslate nel punto (0.5, 0.0)



Rotazione di coordinate di texture

- Le rotazioni delle coordinate di texture sono espresse attraverso un angolo espresso in radianti. Essendo la texture un'immagine 2-D non occorre esprimere un asse di rotazione. La rotazione avviene sempre rispetto ad un immaginario asse ortogonale al piano dell'immagine. La rotazione avviene rispetto all'angolo in basso a sx dell'immagine (origine del sistema di coordinate di texture) ed in senso antiorario).

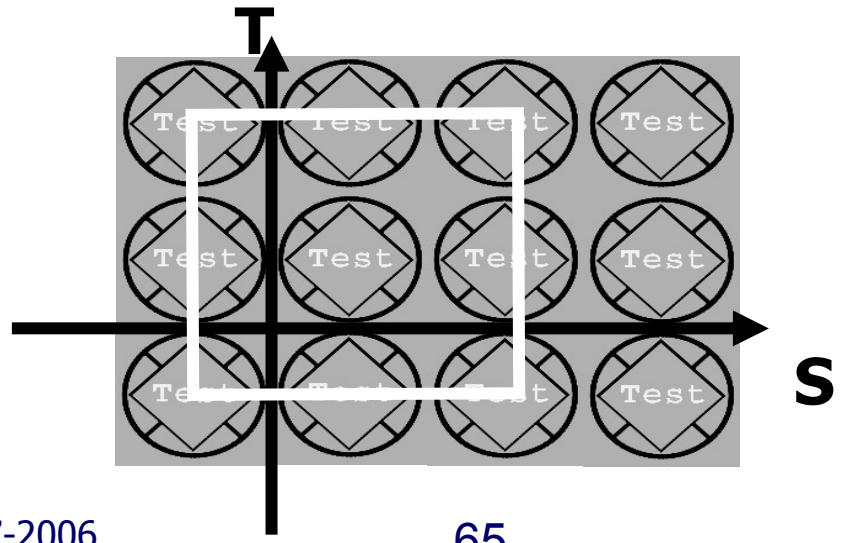
Le coordinate di texture, ruotate di 45° in senso antiorario



Scalatura di coordinate di texture

- La scalatura delle coordinate di texture serve per ingrandire o rimpiccolire la forma da applicare alla immagine texture prima di tagliarla.
- Si può scegliere un punto diverso dall'origine delle coordinate di texture per effettuare la scalatura.

Le coordinate di texture, scalate di un fattore 2.0 e con centro in (0.5, 0.5)



Il nodo Appearance

- Il nodo **Appearance** specifica gli attributi di apparenza delle forme e può essere usato come valore del campo **appearance** del nodo **Shape**.

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

- Il valore dell'exposed-field **textureTransform** specifica un nodo che può essere applicato alle cordinate 2-D di texture quando si incolla la texture alla forma.
- Un valore tipico è il nodo **TextureTransform**.
- Il valore di default **NULL** indica che non viene effettuata nessuna trasformazione

Il nodo ImageTexture

- Il nodo **ImageTexture** specifica gli attributi con cui viene fatto texture-mapping e può essere usato come valore del campo **texture** del nodo **Appearance**.

```
ImageTexture {
        url             []          # exposedField MFString
        repeatS         TRUE        # field SFBool
        repeatT         TRUE        # field SFBool
         }
```

- I campi **repeatS** e **repeatT** specificano quando le coordinate devono essere spostate o no. Quando il valore è **TRUE** la texture può ripetersi lungo le coordinate di texture.
- I campi **repeatS** e **repeatT** sono indipendenti tra loro, consentendo lo spostamento in una, l'altra o entrambe le coordinate di texture.

Il nodo PixelTexture

- Il nodo **PixelTexture** specifica gli attributi cin cui viene fatto texture-mapping e può essere usato come valore del campo **texture** del nodo **Appearance**.

```
PixelTexture {
        image           0 0 0       # exposedField SFImage
        repeatS         TRUE        # field SFBool
        repeatT         TRUE        # field SFBool
         }
```

- I campi **repeatS** e **repeatT** specificano quando le coordinate devono essere spostate o no. Quando il valore è **TRUE** la texture può ripetersi lungo le coordinate di texture.

Il nodo MovieTexture

- Il nodo **MovieTexture** specifica gli attributi con cui viene fatto texture-mapping e può essere usato come valore del campo **texture** del nodo **Appearance**.

```
MovieTexture {
        url             []          # exposedField MFString
        loop            FALSE       # exposedField SFBool
        speed           1.0         # exposedField SFFloat
        startTime       0.0         # exposedField SFFloat
        stopTime        0.0         # exposedField SFFloat
        repeatS         TRUE        # field SFBool
        repeatT         TRUE        # field SFBool
        isActive                    # eventOut SFBool
        duration_changed            # eventOut SFFloat
         }
```

Il nodo TextureCoordinate

- Il nodo **TextureCoordinate** specifica una lista di coordinate di texture e può essere usato come valore del campo **texCoord** dei nodi **IndexedFaceSet** ed **ElevationGrid**.

```
TextureCoordinate {
        point           []          # exposedField MFVec2f
         }
```

- Il valore del campo **point** è una lista di coordinate 2-D di texture che indica le posizioni della texture
- Ogni coordinata è costituita da 2 valori, uno per la coordinata S e uno per la coordinata T. Il default è una lista vuota.
- La lista di coordinate di texture può essere modificata inviando valori all'eventIn implicito **set_point**. I valori vengono propagati attraverso l'eventOut implicito **point_changed**.

Il nodo TextureTransform

- Il nodo **TextureTransform** specifica un nuovo sistema di coordinate di texture rispetto all'originale. Il nodo **TextureTransform** può essere usato come valore del campo **textureTransform** del nodo **Appearance**.

```
TextureTransform {
        translation          0.0 0.0        # exposedField MFVec2f
        rotation             0.0            # exposedField MFFloat
        scale                1.0  1.0       # exposedField MFVec2f
        center               0.0 0.0        # exposedField MFVec2f

         }
```

Il nodo `TextureTransform` (i)

- Il valore dell'exposed-field **`translation`** specifica le distanze lungo S e T, dall'origine del sistema di coordinate di texture, della nuova origine del sistema di coordinate di texture. I due valori possono essere positivi o negativi.
- Il valore dell'exposed-field **`rotation`** specifica di quanto deve essere ruotato il nuovo sistema di coordinate di texture.
- Il valore dell'exposed-field **`scale`** specifica se e di quanto deve essere scalato il nuovo sistema di coordinate di texture. Valori superiori a `1.0` ingrandiscono il nuovo sistema di coordinate, valori inferiori a `1.0` lo rimpiccoliscono.
- Il volore dell'exposed-field **`center`** specifica una coordinata 2-D di texture nel nuovo sistema di coordinate traslato, rispetto al quale deve avvenire la rotazione o la scalatura. Il valore di default è l'origine.
- I valori possono essere modificati inviando valori agli eventIn impliciti **`set_translation`**, **`set_rotation`**, **`set_scale`** e **`set_center`**. I nuovi valori verranno propagati attraverso gli eventOut impliciti **`translation_changed`**, **`rotation_changed`**, **`scale_changed`** e **`center_changed`**.

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 (i)

- Il valore dell'exposed-field **texCoord** fornisce un nodo che specifica le coordinate di texture disponibili per colorare le facce della forma realizzata con il nodo **IndexedFaceSet**. Un valore tipico del campo **texCoord** è il nodo **TextureCoordinate**.
- Il valore dell'exposed-field **texCoordIndex** fornisce una lista di indici di coordinate di texture che descrivono uno o più mascherine per ritagliare l'immagine di texture, una per ogni faccia della forma realizzata con il nodo **IndexedFaceSet**. Ogni valore della lista è un indice intero che corrisponde ad una coordinata di texture della lista specificata nel campo **texCoord**. Il valore di default del campo **texCoordIndex** è una lista vuota.
- Se il campo **texCoordIndex** è vuoto e vengono specificate delle coordinate di texture nel campo **texCoord**, vengono utilizzati gli indici indicati nel campo **coordIndex**.
- I valori del campo **texCoord** possono essere modificati inviando valori all'eventIn implicito **set_texCoord**. I nuovi valori verranno propagati attraverso l'eventOut implicito **texCoord_changed**.
- I valori degli indici delle coordinate di texture possono essere modificati inviando dei dati all'eventIn **set_texCoordIndex**.

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)

- Il valore dell'exposed-field **`texCoord`** fornisce un nodo che specifica le coordinate di texture disponibili per colorare la forma 3-D realizzata dal nodo **`ElevationGrid`**. Un valore tipico del campo **`texCoord`** è il nodo **`TextureCoordinate`**.
- Quando viene specificato il campo Il valore dell'exposed-field **`texCoord`** viene usata una coordinata per ogni punto della griglia. La prima coordinata viene usata per il primo punto a sx della prima riga della griglia. Ci deve essere una coordinata per ogni punto della griglia (dato dal risultato dell'operazione **`xDimension x zDimension`**).
- I valori del campo **`texCoord`** possono essere modificati inviando valori all'eventIn implicito **`set_texCoord`**. I nuovi valori verranno propagati attraverso l'eventOut implicito **`texCoord_changed`**.

Suggerimenti

- E' raccomandato di istanziare i nodi **`ImageTexture`** e **`MovieTexture`** attraverso l'uso di **`DEF`** / **`USE`** in modo da ridurre il tempo di scaricamento dei sorgenti e velocizzare il processo di rappresentazione della scena.
- E' bene disattivare l'illuminazione della scena, omettendo la specif ca del nodo **`Material`**.
- Ridurre al massimo le dimensioni del processo di texture mapping mediante:
  - Scelta della risoluzione minima possibile
  - Uso di immagini monocromatiche
  - Per immagini ripetibili scegliere l'elemento più piccolo possibile e sfruttare al massimo la proprietà di riprodurre lo stesso elemento
  - Le immagini trasparenti agiscono come un coltello che taglia la forma dove l'immagine è trasparente.

Coordinate di texture

```
#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 { }
    texture ImageTexture {
        url "testimg.jpg"
    }
    }
    geometry IndexedFaceSet {
    coord Coordinate {
        point [
            -1.0 -1.0 0.0,
             1.0 -1.0 0.0,
             1.0  1.0 0.0,
            -1.0  1.0 0.0,
        ]
    }
    coordIndex [ 0, 1, 2, 3, ]
    texCoord TextureCoordinate {
        point [
            0.2 0.2,
            0.8 0.2,
            0.8 0.8,
            0.2 0.8,
        ]
    }
    texCoordIndex [ 0, 1, 2, 3, ]
    solid FALSE
    }
}
```

Testimg.jpg

Scena VRML

Coordinate di texture (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 { }
    texture ImageTexture {
         url "pizza.jpg"
    }
    }
    geometry IndexedFaceSet {
    coord Coordinate {
         point [
         # Slice, pulled out of pizza
              0.50 0.0 0.50,   0.88 0.0 1.42,
              1.06 0.0 1.33,   1.21 0.0 1.21,
              1.33 0.0 1.06,   1.42 0.0 0.88
         ]
    }
    coordIndex [ 0, 1, 2, 3, 4, 5 ]
    texCoord TextureCoordinate {
         point [
         # Center point of pizza image
              0.50 0.50,
         # Slice perimeter
              0.68 0.07,  0.76 0.11,
              0.83 0.17,  0.89 0.24,
              0.93 0.32,
         ]
    }
    texCoordIndex [ 0, 1, 2, 3, 4, 5, ]
    solid FALSE
    }
}
```

pizza.jpg

Scena VRML

Coordinate di texture (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 { }
      texture ImageTexture {
              url "pizza.jpg"
      }
      }
      geometry IndexedFaceSet {
      coord Coordinate {
              point [
              # Slice, pulled out of pizza
                       0.50 0.0  0.50,    0.88 0.0  1.42,
                       1.06 0.0  1.33,    1.21 0.0  1.21,
                       1.33 0.0  1.06,    1.42 0.0  0.88,
              # Rest of pizza
                       0.00 0.0  0.00,    0.92 0.0  0.38,
                       0.98 0.0  0.20,    1.00 0.0  0.00,
                       0.98 0.0 -0.20,    0.92 0.0 -0.38,
                       0.83 0.0 -0.56,    0.71 0.0 -0.71,
                       0.56 0.0 -0.83,    0.38 0.0 -0.92,
                       0.20 0.0 -0.98,    0.00 0.0 -1.00,
                      -0.20 0.0 -0.98,   -0.38 0.0 -0.92,
                      -0.56 0.0 -0.83,   -0.71 0.0 -0.71,
                      -0.83 0.0 -0.56,   -0.92 0.0 -0.38,
                      -0.98 0.0 -0.20,   -1.00 0.0  0.00,
                      -0.98 0.0  0.20,   -0.92 0.0  0.38,
                      -0.83 0.0  0.56,   -0.71 0.0  0.71,
                      -0.56 0.0  0.83,   -0.38 0.0  0.92,
                      -0.20 0.0  0.98,    0.00 0.0  1.00,
                       0.20 0.0  0.98,    0.38 0.0  0.92
              ]
      }
      coordIndex [
      # Slice
                0,  1,  2,  3,  4,  5, -1,
      # Rest of pizza
              6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
              17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
              28, 29, 30, 31, 32, 33, 34, 35
      ]
      texCoord TextureCoordinate {
              point [
              # Center point of pizza image
                       0.50 0.50,
              # Pizza perimeter
                       0.96 0.41,  0.97 0.50,
                       0.96 0.59,  0.93 0.68,
                       0.89 0.76,  0.83 0.83,
                       0.76 0.89,  0.68 0.93,
                       0.59 0.96,  0.50 0.97,
                       0.41 0.96,  0.32 0.93,
                       0.24 0.89,  0.17 0.83,
                       0.11 0.76,  0.07 0.68,
                       0.04 0.59,  0.03 0.50,
                       0.04 0.41,  0.07 0.32,
                       0.11 0.24,  0.17 0.17,
                       0.24 0.11,  0.32 0.07,
                       0.41 0.04,  0.50 0.03,
                       0.59 0.04,
              # Slice perimeter
                       0.68 0.07,  0.76 0.11,
                       0.83 0.17,  0.89 0.24,
                       0.93 0.32
              ]
      }
      texCoordIndex [
      # Slice
              0, 28, 29, 30, 31, 32, -1,
      # Rest of pizza
              0,  32,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
              11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
              23, 24, 25, 26, 27, 28
      ]
      solid FALSE
      }
}
```

pizza.jpg

Scena VRML

Coordinate di texture (iii)

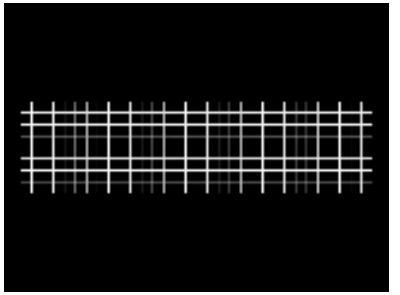
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
      children [
      # Lower-left video screen
      Shape {
              appearance Appearance {
                      # no material, use emissive texturing
                      texture DEF Video ImageTexture {
                              url "grand.jpg"
                      }
              }
              geometry DEF Screen IndexedFaceSet {
                      solid FALSE
                      coord Coordinate {
                              point [
                                      0.0 0.0 0.0,  1.0 0.0 0.0,
                                      1.0 1.0 0.0,  0.0 1.0 0.0,
                              ]
                      }
                      coordIndex [ 0, 1, 2, 3 ]
                      texCoord TextureCoordinate {
                              point [
                                      0.0 0.0,  0.5 0.0,
                                      0.5 0.5,  0.0 0.5,
                              ]
                      }
                      texCoordIndex [ 0, 1, 2, 3 ]
              }
      },
      # Lower-right video screen
      Transform {
              translation 1.1 0.0 0.0
              children Shape {
                      appearance Appearance {
                      # no material, use emissive texturing
                              texture USE Video
                              textureTransform TextureTransform {
                              # Slide to lower-right quadrant
                                      translation 0.5 0.0
                              }
                      }
                      geometry USE Screen
              }
      },
      # Upper-left video screen
      Transform {
              translation 0.0 1.1 0.0
              children Shape {
                      appearance Appearance {
                      # no material, use emissive texturing
                              texture USE Video
                              textureTransform TextureTransform {
                              # Slide to upper-left quadrant
                                      translation 0.0 0.5
                              }
                      }
                      geometry USE Screen
              }
      },
      # Upper-right video screen
      Transform {
              translation 1.1 1.1 0.0
              children Shape {
                      appearance Appearance {
                      # no material, use emissive texturing
                              texture USE Video
                              textureTransform TextureTransform {
                              # Slide to upper-right quadrant
                                      translation 0.5 0.5
                              }
                      }
                      geometry USE Screen
              }
      }
      ]
}
```

grand.jpg

Scena VRML

Scalatura di coordinate di texture

```
#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 { }
    texture ImageTexture {
         url "grill.png"
    }
    textureTransform TextureTransform {
         scale 32.0 8.0
         center 0.5 0.5
    }
    }
    geometry IndexedFaceSet {
    solid FALSE
    coord Coordinate {
         point [
              -4.0 -1.0 0.1,   4.0 -1.0 0.1,
               4.0  1.0 0.1,  -4.0  1.0 0.1,
         ]
    }
    coordIndex [ 0, 1, 2, 3 ]
    texCoord TextureCoordinate {
         point [
              0.0 0.0,  1.0 0.0,
              1.0 1.0,  0.0 1.0,
         ]
    }
    texCoordIndex [ 0, 1, 2, 3 ]
    }
}
```

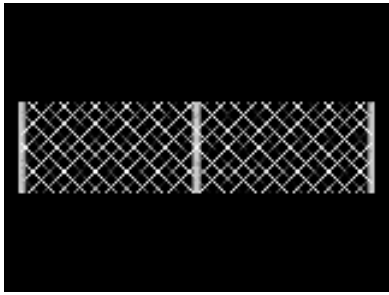


grill.png

Scena VRML

Rotazione di coordinate di texture

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Chain-link fence
    Shape {
         appearance Appearance {
              material Material { }
              texture ImageTexture {
                   url "grill.png"
              }
              textureTransform TextureTransform {
                   rotation 0.785
                   scale 32.0 8.0
                   center 0.5 0.5
              }
         }
         geometry IndexedFaceSet {
              solid FALSE
              coord Coordinate {
                   point [
                        -4.0 -1.0 0.1,   4.0 -1.0 0.1,
                         4.0  1.0 0.1,  -4.0  1.0 0.1,
                   ]
              }
              coordIndex [ 0, 1, 2, 3 ]
              texCoord TextureCoordinate {
                   point [
                        0.0 0.0,  1.0 0.0,
                        1.0 1.0,  0.0 1.0,
                   ]
              }
              texCoordIndex [ 0, 1, 2, 3 ]
         }
    },
    # Fence posts
    DEF Post Shape {
         appearance Appearance {
              material Material { }
         }
         geometry Cylinder {
              height 2.0
              radius 0.1
         }
    },
    Transform { translation -4.0 0.0 0.0  children USE Post },
    Transform { translation  4.0 0.0 0.0  children USE Post }
    ]
}
```

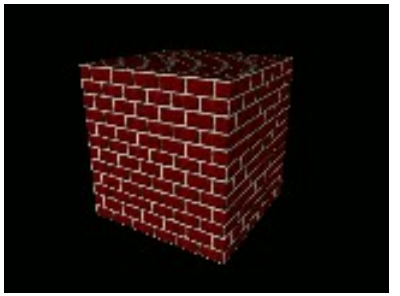


grill.png

Scena VRML

Trasformazione di texture su forme predefinite

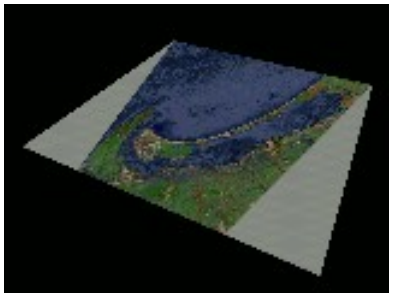
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
    appearance Appearance {
    material Material { }
    texture ImageTexture {
         url "brick.jpg"
    }
    textureTransform TextureTransform {
         scale 3.0 3.0
    }
    }
    geometry Box { }
}
```

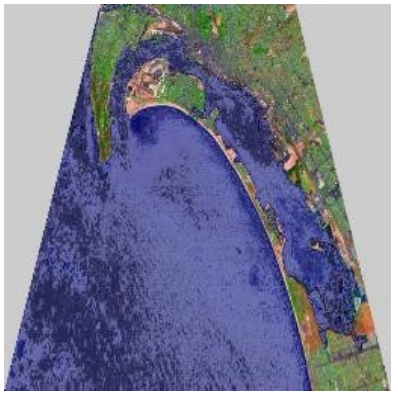


brick.jpg

Controllo di texture nel nodo **ElevationGrid**

```
#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 { }
    texture ImageTexture { url "sdbaywlr.jpg" }
    }
    geometry ElevationGrid {
    xDimension 5
    zDimension 5
    xSpacing   0.2
    zSpacing   0.2
    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
    ]
    }
}
```

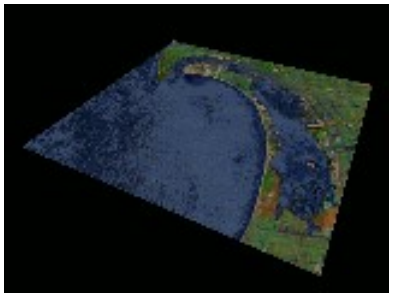


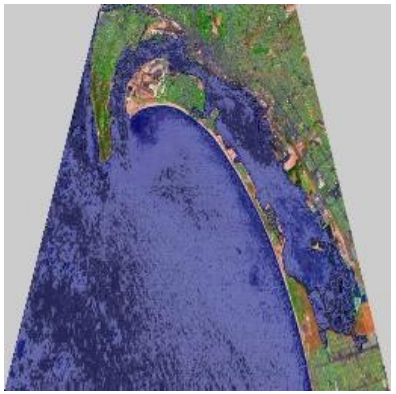


sdbaywlr.jpg

Controllo di texture nel nodo **ElevationGrid**

```
#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 { }
    texture ImageTexture { url "sdbaywlr.jpg" }
    }
    geometry ElevationGrid {
    xDimension 5
    zDimension 5
    xSpacing   0.2
    zSpacing   0.2
    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
    ]
    texCoord TextureCoordinate {
         point [
              0.250 1.000, 0.375 1.000, 0.500 1.000, 0.625 1.000, 0.750 1.000,
              0.188 0.750, 0.344 0.750, 0.500 0.750, 0.656 0.750, 0.812 0.750,
              0.125 0.500, 0.312 0.500, 0.500 0.500, 0.688 0.500, 0.875 0.500,
              0.062 0.250, 0.281 0.250, 0.500 0.250, 0.719 0.250, 0.938 0.250,
              0.000 0.000, 0.250 0.000, 0.500 0.000, 0.750 0.000, 1.000 0.000
         ]
    }
    }
}
```

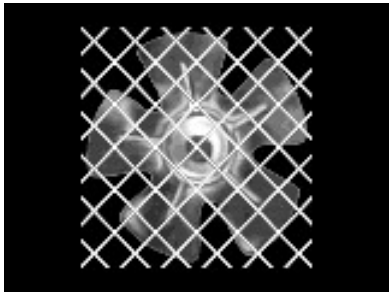




sdbaywlr.jpg

Animazione della trasformazione di texture

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
     children [
     # Rotating fan
     Shape {
            appearance Appearance {
                   material Material { }
                   texture ImageTexture {
                          url "fan.png"
                          repeatS FALSE
                          repeatT FALSE
                   }
                   textureTransform DEF FanRotation TextureTransform {
                          center 0.5 0.5
                   }
            }
            geometry DEF Square IndexedFaceSet {
                   coord Coordinate {
                          point [
                                 -1.0 -1.0 -0.1,   1.0 -1.0 -0.1,
                                  1.0  1.0 -0.1,  -1.0  1.0 -0.1,
                          ]
                   }
                   coordIndex [ 0, 1, 2, 3, ]
                   texCoord TextureCoordinate {
                          point [
                                 0.0 0.0,  1.0 0.0,
                                 1.0 1.0,  0.0 1.0,
                          ]
                   }
                   texCoordIndex [ 0, 1, 2, 3, ]
                   solid FALSE
            }
     },
     # Grill in front
     Transform {
            translation 0.0 0.0 0.1
            children Shape {
                   appearance Appearance {
                          material Material { }
                          texture ImageTexture {
                                 url "grill.png"
                          }
                          textureTransform TextureTransform {
                                 rotation 0.785
                                 scale 8.0 8.0
                                 center 0.5 0.5
                          }
                   }
                   geometry USE Square
            }
     },
     # Animation clock
     DEF Clock TimeSensor {
            cycleInterval 10.0
            loop TRUE
     },
     # Animation path
     DEF FanPath ScalarInterpolator {
            key      [ 0.0, 0.5,  1.0  ]
            keyValue [ 0.0, 3.14, 6.28 ]
     },
     ]
}
ROUTE Clock.fraction_changed TO FanPath.set_fraction
ROUTE FanPath.value_changed  TO FanRotation.set_rotation
```



fan.jpg

grill.png

Virtual Reality Modeling Language

VRML

Controllo dell'ombreggiatura

Ombreggiatura

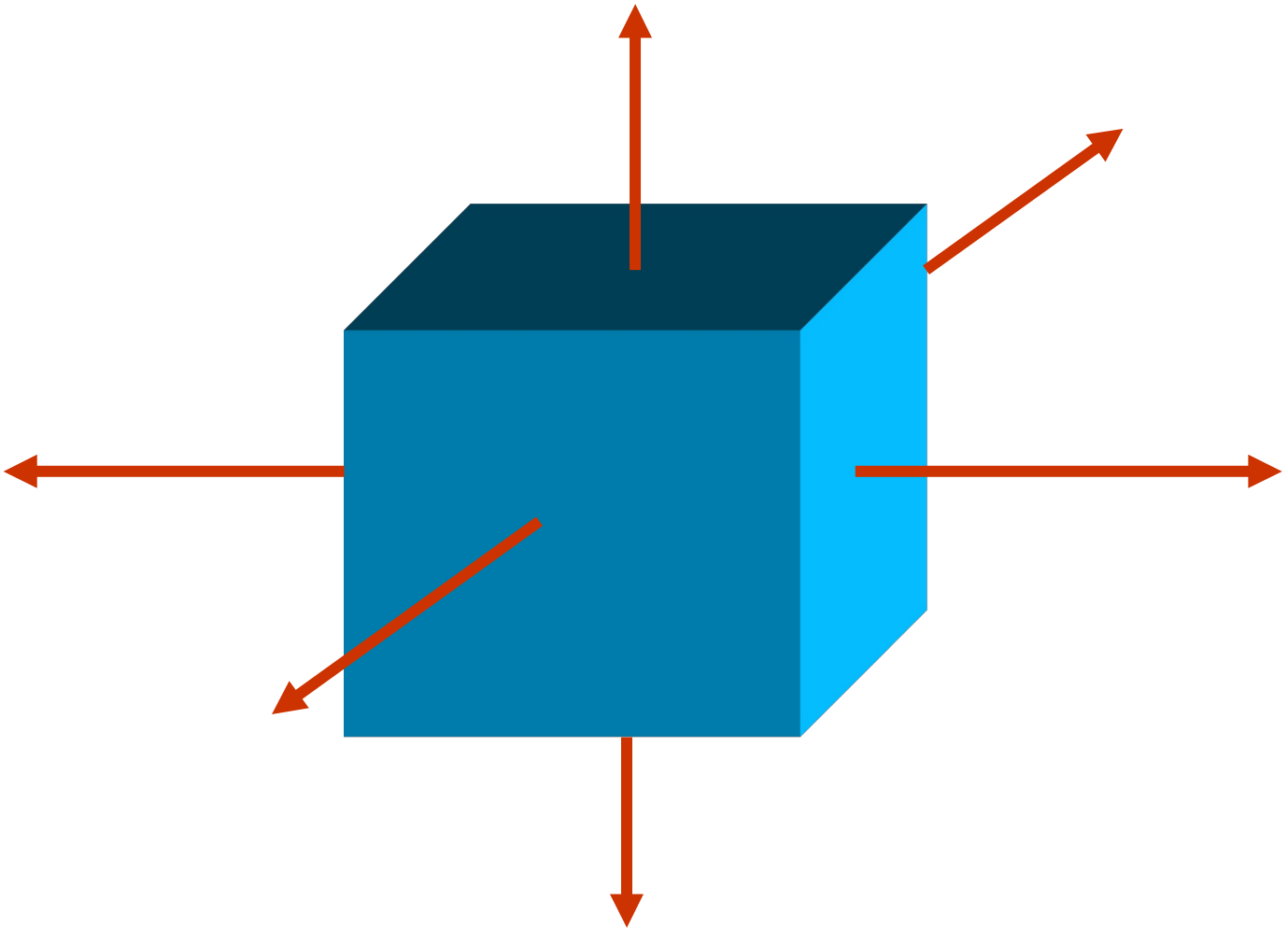
- L'ombreggiatura di una faccia di una forma VRML dipende da quanto questa è illuminata dalla luce del mondo virtuale. Più la faccia è orientata verso una luce, più viene illuminata ed è resa brillante dal browser. Una faccia lontana dalla luce viene resa scura dal browser.
- Per decidere se una faccia è orientata o no verso una luce, il browser calcola automaticamente la normale alla faccia. La normale è un vettore che punta diritto fuori dalla faccia, perpendicolare ad essa.
- Se la faccia è orientata a dx, allora la normale è orientata a dx. se la faccia punta verso l'alto, la normale è orientata verso l'alto.
- Se la normale punta verso la luce, allora la faccia è resa brillante, se punta lontano dalla luce, viene resa scura.

Normale

- Il calcolo delle normali delle facce è compito del browser.
- L'utente può specif care autonomamente le normali e modif care il comportamento di default del browser.
- Questo consente all'utente di controllare esattamente il modo con cui le facce vengono ombreggiate dal browser.
- Questa tecnica va usata in casi ristretti, quando il comportamento del browser non è corretto.
- E' possibile specif care i valori delle normali per le facce relative ai nodi **`IndexedFaceSet`** e **`ElevationGrid`**. In entrambi i casi si specif cano i valori delle normali mediante il nodo **`Normal`** come valore dell'exposed-f eld **`normal`**. Si possono persino animare le normali mediante il nodo **`NormalInterpolator`**.

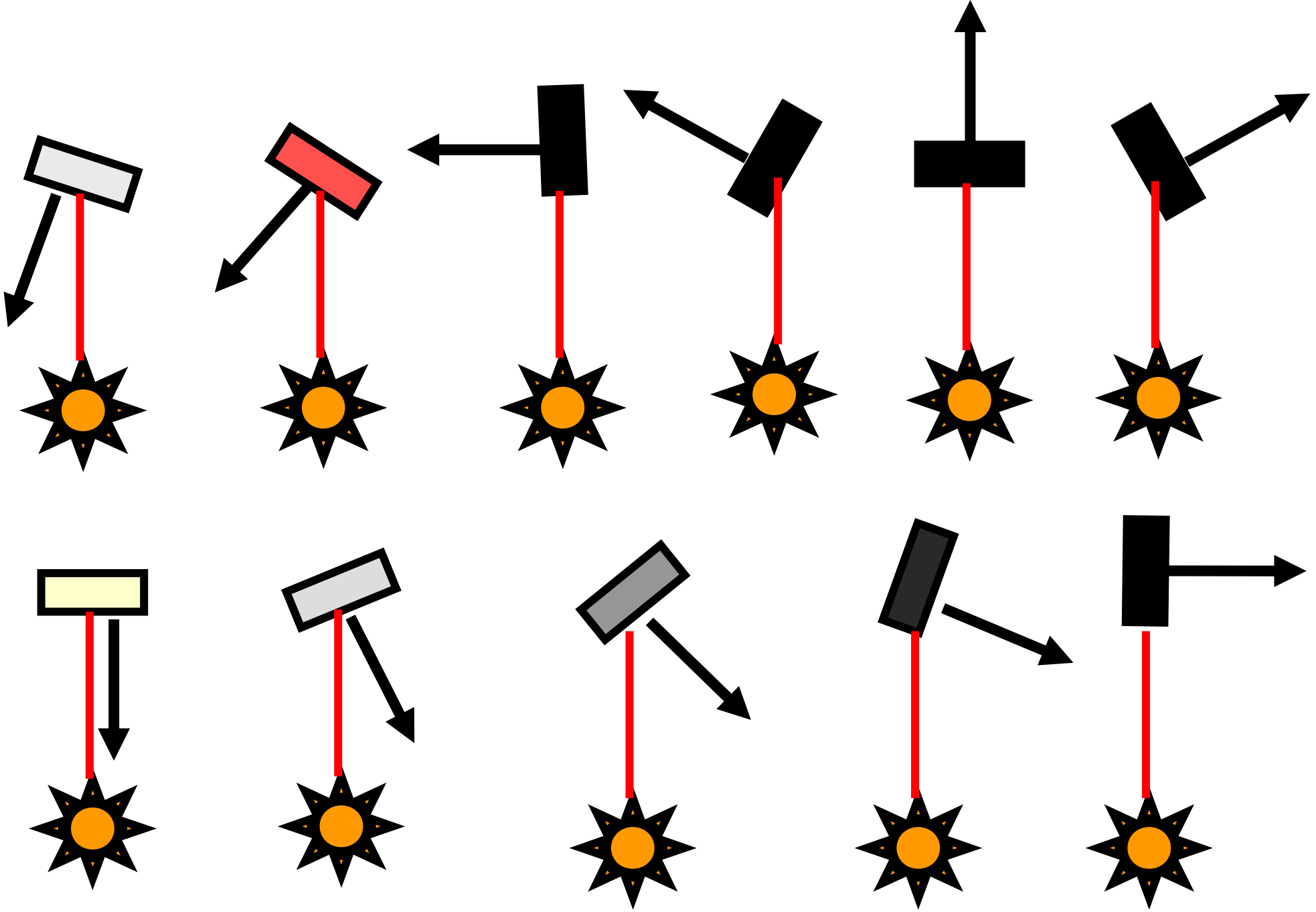
Normali (i)

- La normale è un modo per indicare l'orientazione di una faccia:

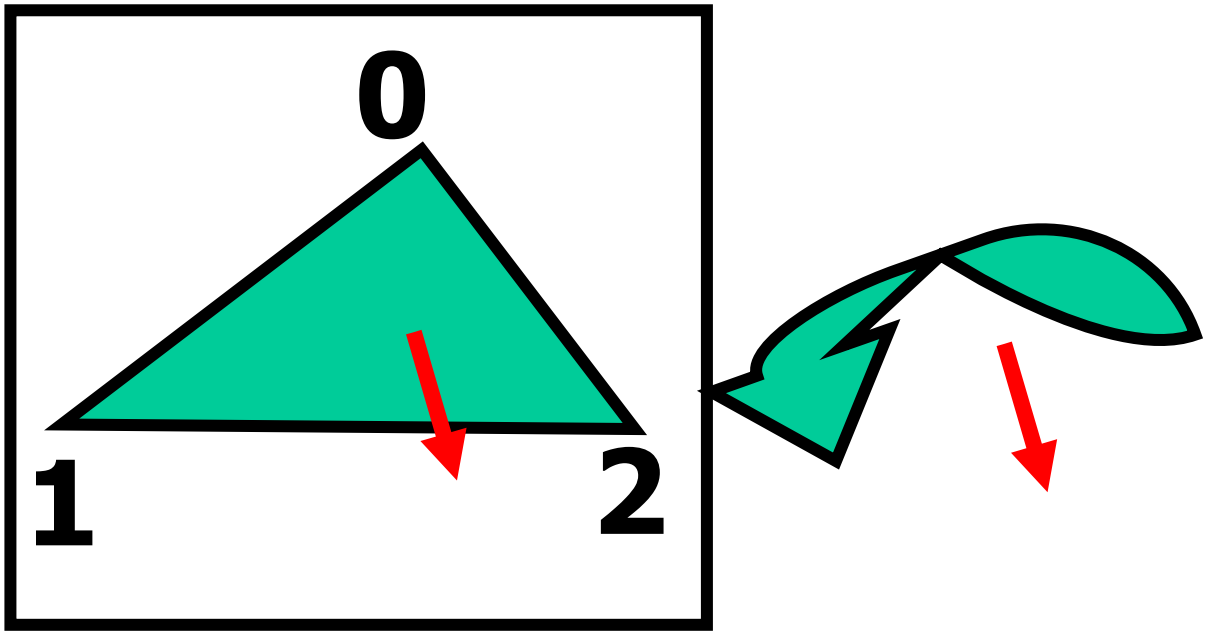


Ombreggiatura in base alla direzione della normale

- Il browser VRML crea sempre una headlight, che analogamente alla lampada sul casco dei minatori, illumina per noi il mondo virtuale.
- Il browser calcola l'angolo che la varie facce formano con l'headlight, che coincide con la testa dell'avatar.
- Quando la faccia è orientata verso il visitatore, la normale forma un angolo piccolo e la faccia è molto illuminata.
- Quando una faccia è orientata lontano dall'osservatore, l'angolo formato aumenta sempre più e la faccia è resa sempre più scura.



Determinazione della direzione della normale

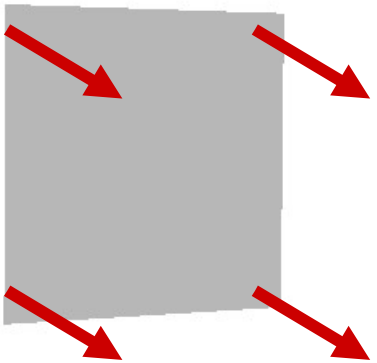


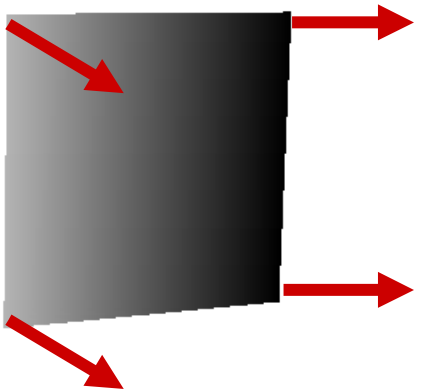
- La normale (freccia rossa) esce ortogonalmente dal foglio, verso l'osservatore, secondo la regola della mano dx orientata con il palmo secondo la freccia grande verde, che segue la numerazione delle facce.

Smooth Shading

- Il modo con cui il browser ombreggia le forme può essere alterato mediante una tecnica denominata Smooth Shading, che consiste nello specificare manualmente le normali delle varie facce, facendole considerare al browser in maniera analoga a forme curve.
- Il calcolo delle normali è comunque abbastanza complesso.
- Il controllo delle normali avviene specificando una lista di normali per i nodi **`IndexedFaceSet`** ed **`ElevationGrid`** in un nodo **`Normal`** che assegna un valore al campo **`normal`** del nodo **`IndexedfaceSet`** o **`ElevationGrid`**.
- Gli indici delle normali vanno specificati nel nodo **`normalIndex`**.

Smooth Shading

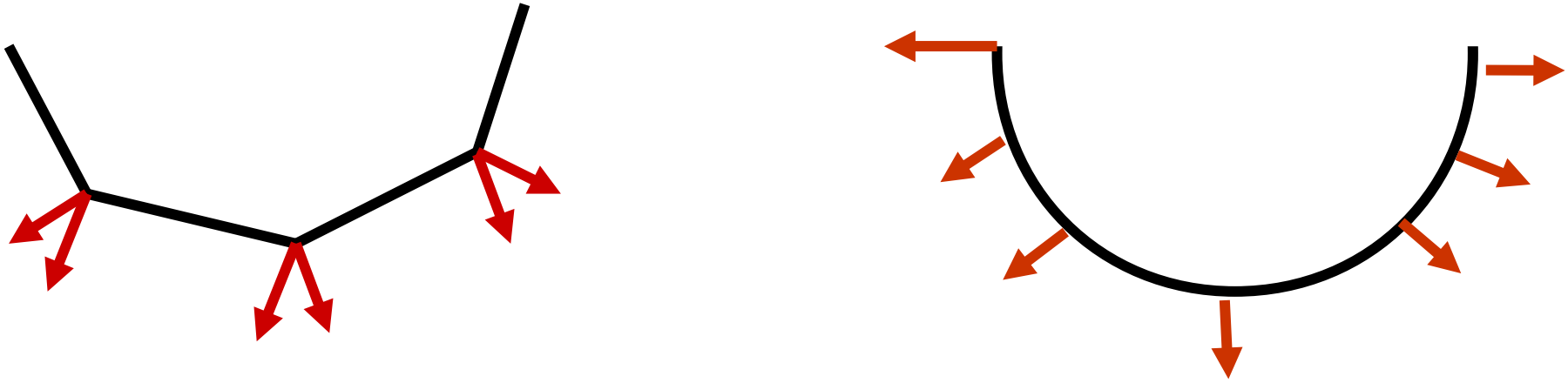




```
normal Normal {
        vector [
            0.0  0.0  1.0,   0.0  0.0  1.0,
            0.0  0.0  1.0,   0.0  0.0  1.0,
        ]
    }
    normalIndex [ 0, 1, 2, 3 ]
```

```
normal Normal {
        vector [
            0.0  0.0  1.0,   1.0  0.0  0.0,
            1.0  0.0  0.0,   0.0  0.0  1.0,
        ]
    }
    normalIndex [ 0, 1, 2, 3 ]
```

Smooth Shading: approx di curve



Talvolta l'uso del campo **`creaseAngle`** genera un effetto di ombreggiatura errato, in conseguenza dell'andamento dei vettori delle normali illustrato in a). L'effetto può essere corretto a mano, esplicitando i vettori delle normali in maniera corretta, come mostrato in b)

Animazioni di normali

- Il nodo **`NormalInterpolator`** è in grado di implementare l'animazione delle coordinate.
- Il nodo **`NormalInterpolator`** usa una lista di frazioni temporali nel campo **`key`** ed una serie di vettori normali nel campo **`keyValue`**.
- Se si alimenta il nodo con un nodo **`TimeSensor`** che produce una serie di tempi variabili, il nodo **`NormalInterpolator`** produrrà una serie di vettori normali per interpolazione lineare dei valori chiave di normale presenti nella definizione del nodo.

Il nodo Normal

Il nodo **Normal** crea una lista di vettori di normali e può essere usato come valore del campo **normal** in un nodo geometrico basato su coordinate, come il nodo **IndexedFaceSet** o il nodo **ElevationGrid**.

```
Normal{
        vector          [ ]                     # exposedField MFVec3f
      }
```

- Il valore dell'exposed-field **vector** specifica una lista di vettori di normali che possono essere utilizzati come vettori normali delle facce della forma. Ogni normale è specificato da tre valori indicanti la componente X, quella Y e quella Z del vettore. Il default è una lista vuota. Il contenuto può essere modificato utilizzando l'eventIn implicito **set_vector**. L'informazione modificata può essere propagata attraverso l'eventOut implicito **vector_changed**.

Il nodo NormalInterpolator

Il nodo **NormalInterpolator** descrive una serie di vettori di normale da poter utilizzare nella animazione di normali.

```
NormalInterpolator{
        key             [ ]             # exposedField MFFloat
        keyValue        [ ]             # exposedField MFVec3f
        set_fraction                    # eventIn      MFFloat
        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 NormalInterpolator

- 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 del vettore normale alla faccia.
- Quando il nodo riceve un tempo parziale attraverso l'eventIn **set_fraction** il nodo calcola un nuovo vettore normale sulla base dei valori di riferimento specificati in **keyValue**. Il valore viene interpolato da punto a punto come se giacesse su una superficie di una sfera di raggio 1.0.Il vettore normale calcolato viene propagato attraverso l'eventOut **fraction_changed**.
- La lista dei tempi e delle normali di riferimento può essere cambiata inviando valori negli eventIn impliciti **set_key** e **set_keyValue** degli exposed-field **key** e **keyValue**.
- Il nodo **NormalInterpolator** non crea forme e non ha effetti visibili nel mondo virtuale. Un nodo **NormalInterpolator** può essere figlio di qualsiasi nodo di raggruppamento ed è indipendente dal sistema di coordinate del sistema.

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 **normal** specifica un nodo che contenga la lista dei vettori normali da associare alle facce della forma. Un valore tipico del campo **normal** è un nodo **Normal**.
- Il valore di default **NULL** indica una lista vuota e le normali alle varie facce vengono calcolate automaticamente.
- Il valore del campo **normalIndex** specifica una lista di indici di vettori normali e ne consentono l'assegnazione alle varie facce della forma. Ciascun valore è un indice intero che specifica una normale nella lista del campo **normal**. Il valore di default è una lista vuota, indicante che la lista delle normali è vuota.
- Il campo **normalPerVertex** specifica se le normali del nodo **Normal** sono usate per ogni faccia (**FALSE**) o per ogni coordinata indice associata ad ogni faccia (**TRUE**). Il valore di default del campo **NormalPerVertex** è **TRUE**.

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

- Il valore dell'exposed-field **normal** specifica un nodo che contenga la lista dei vettori normali da associare alle facce dell'Elevation Grid. Un valore tipico del campo **normal** è un nodo **Normal**.
- Il valore di default **NULL** indica una lista vuota e le normali alle varie facce vengono calcolate automaticamente.
- Il campo **normalPerVertex** specifica se le normali del nodo **Normal** sono usate per ogni quadrato della griglia (**FALSE**) o per ogni punto della griglia (**TRUE**). Il valore di default del campo **NormalPerVertex** è **TRUE**.

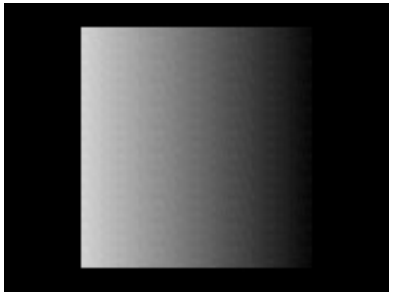
Ex: Una faccia e 4 normali orientate in +Z

```
#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 { }
   }
   geometry IndexedFaceSet {
   coord Coordinate {
       point [
           -1.0 -1.0  0.0,   1.0 -1.0  0.0,
            1.0  1.0  0.0,  -1.0  1.0  0.0,
       ]
   }
   coordIndex [ 0, 1, 2, 3 ]
   normalPerVertex TRUE
   normal Normal {
       vector [
           0.0  0.0  1.0,   0.0  0.0  1.0,
           0.0  0.0  1.0,   0.0  0.0  1.0,
       ]
   }
   normalIndex [ 0, 1, 2, 3 ]
   }
}
```

scena VRML

Ex: Una faccia e 4 normali, 2 orientate in +Z e 2 orientate in +X

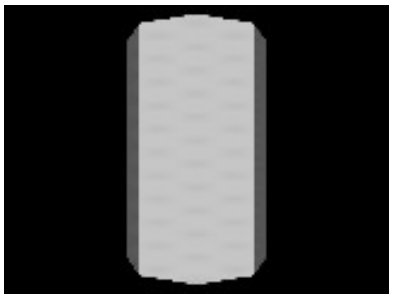
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Shape {
   appearance Appearance {
   material Material { }
   }
   geometry IndexedFaceSet {
   coord Coordinate {
       point [
           -1.0 -1.0  0.0,   1.0 -1.0  0.0,
            1.0  1.0  0.0,  -1.0  1.0  0.0,
       ]
   }
   coordIndex [ 0, 1, 2, 3 ]
   normalPerVertex TRUE
   normal Normal {
       vector [
           0.0  0.0  1.0,   1.0  0.0  0.0,
           1.0  0.0  0.0,   0.0  0.0  1.0,
       ]
   }
   normalIndex [ 0, 1, 2, 3 ]
   }
}
```



scena VRML

Ex: Una semicolonna sfaccettata con uso di `creaseAngle`

```
#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 { }
   }
   geometry IndexedFaceSet {
   creaseAngle 1.57
   coord Coordinate {
        point [
             -2.00  3.00 0.00,
             -2.00 -3.00 0.00,
             -1.41  3.00 1.41,
             -1.41 -3.00 1.41,
              0.00  3.00 2.00,
              0.00 -3.00 2.00,
              1.41  3.00 1.41,
              1.41 -3.00 1.41,
              2.00  3.00 0.00,
              2.00 -3.00 0.00,
        ]
   }
   coordIndex [
        0, 1, 3, 2,   -1,
        2, 3, 5, 4,   -1,
        4, 5, 7, 6,   -1,
        6, 7, 9, 8,   -1,
   ]
   }
}
```

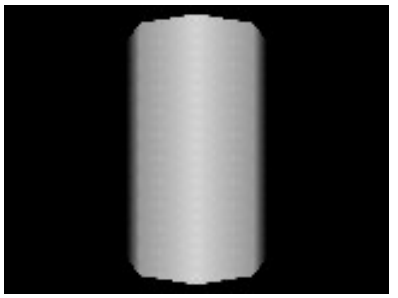


colca1.wrl

scena VRML

Uso di normali definite a mano

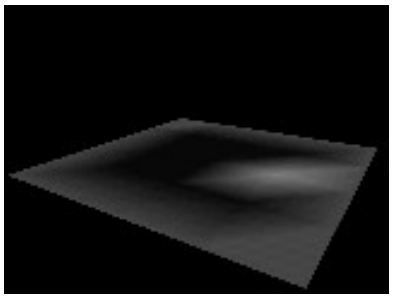
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    Shape {
         appearance Appearance {
              material Material { }
         }
         geometry IndexedFaceSet {
              coord Coordinate {
                   point [
                        -2.00  3.00 0.00,
                        -2.00 -3.00 0.00,
                        -1.41  3.00 1.41,
                        -1.41 -3.00 1.41,
                         0.00  3.00 2.00,
                         0.00 -3.00 2.00,
                         1.41  3.00 1.41,
                         1.41 -3.00 1.41,
                         2.00  3.00 0.00,
                         2.00 -3.00 0.00,
                   ]
              }
              coordIndex [
                   0, 1, 3, 2,   -1,
                   2, 3, 5, 4,   -1,
                   4, 5, 7, 6,   -1,
                   6, 7, 9, 8,   -1,
              ]
              normalPerVertex TRUE
              normal Normal {
                   vector [
                        -1.00 0.00 0.00,
                        -0.71 0.00 0.71,
                         0.00 0.00 1.00,
                         0.71 0.00 0.71,
                         1.00 0.00 0.00,
                   ]
              }
              normalIndex [
                   0, 0, 1, 1,   -1,
                   1, 1, 2, 2,   -1,
                   2, 2, 3, 3,   -1,
                   3, 3, 4, 4,   -1,
              ]

         }
    }
    ]
}
```



scena VRML

Normali e ElevationGrid

```
#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 { }
    }
    geometry ElevationGrid {
    xDimension 5
    zDimension 5
    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,
    ]
    normalPerVertex TRUE
    normal Normal {
         vector [
         # First row
               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,
         # Second row
               0.0 1.0  0.0,  -0.3 0.3 -0.3,
               0.0 0.5 -0.5,   0.3 0.3 -0.3,
               0.0 1.0  0.0,
         # Third row
               0.0 1.0  0.0,  -0.5 0.5  0.0,
              -0.5 0.5  0.0,   0.5 0.5  0.0,
               0.0 1.0  0.0,
         # Fourth row
               0.0 1.0  0.0,  -0.3 0.3 -0.3,
               0.0 0.5 -0.5,   0.3 0.3 -0.3,
               0.0 1.0  0.0,
         # Fifth row
               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,
         ]
    }
    }
}
```



scena VRML

Animazione delle normali

```
#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 shape
    Shape {
         appearance Appearance {
              material Material { }
         }
         geometry IndexedFaceSet {
              coord Coordinate {
                   point [
                        -1.0 -1.0  0.0,   1.0 -1.0  0.0,
                         1.0  1.0  0.0,  -1.0  1.0  0.0,
                   ]
              }
              normal DEF AnimNorm Normal {
                   vector [
                        0.0  0.0  1.0,   0.0  0.0  1.0,
                        0.0  0.0  1.0,   0.0  0.0  1.0,
                   ]
              }
              coordIndex  [ 0, 1, 2, 3 ]
              normalIndex [ 0, 1, 2, 3 ]
              normalPerVertex TRUE
         }
    },
    # Animation clock
    DEF Clock TimeSensor {
         cycleInterval 4.0
         loop TRUE
    },
    # Animation normals
    DEF NormPath NormalInterpolator {
         key [ 0.0, 0.5, 1.0 ]
         keyValue [
         # time 0.0 normals
              0.0  0.0  1.0,   0.0  0.0  1.0,
              0.0  0.0  1.0,   0.0  0.0  1.0,
         # time 0.5 normals
              0.0  0.0  1.0,   1.0  0.0  0.0,
              1.0  0.0  0.0,   0.0  0.0  1.0,
         # time 1.0 normals
              0.0  0.0  1.0,   0.0  0.0  1.0,
              0.0  0.0  1.0,   0.0  0.0  1.0,
         ]
    }
    ]
}
ROUTE Clock.fraction_changed TO NormPath.set_fraction
ROUTE NormPath.value_changed TO AnimNorm.set_vector
```

scena VRML

Virtual Reality Modeling Language

VRML

Aggiunta di sfondi

Aggiunta di sfondi

- Nella costruzione di mondi virtuali può essere utile disporre di funzioni che consentono di aggiungere degli sfondi f ssi. Il visitatore avrà una migliore percezione del mondo virtuale anche se non potrà navigare in quella parte.
- E' inoltre indispensabile poter personalizzare l'aspetto del cielo e del suolo.
- Il nodo **`Background`** consente proprio di controllare i colori del cielo e del suolo, di def nire una immagine di sfondo o un panorama, in modo da creare l'immagine distante di paesaggi montani o cittadini.
- Ciò consente di arricchire il mondo virtuale di ulteriori effetti scenici senza dover def nire queste scene lontane con VRML.

I Colori del cielo VRML

- Il cielo VRML è da immaginare come una sfera inf nitamente larga che circondail mondo virtuale.
- Per default il cielo VRML è nero.
- Utilizzando il nodo **`Background`** è possibile cambiare questo colore, aggiungere gradienti tra colori.
- I gradienti di colore vengono def niti mediante il campo **`skyColor`** del nodo **`Background`**, partendo dal primo colore associato al polo superiore e procedendo in basso f no all'ultimo colore associato la polo inferiore.
- Per ciascun colore, eccetto il primo, occorre anche indicare uno **`skyAngle`** che indica l'angolo a partire dal polo superiore al quale deve essere applicato il colore stesso.

I colori del cielo VRML (i)

- I colori tra due **`skyAngle`** vengono fatti variare in maniera morbida, in modo da creare il gradiente di colore.
- Utilizzando **`skyAngle`** si può definire molto precisamente come devono cambiare i colori della sfera del cielo.
- Ad esempio si può definire **`skyAngle`** come: (`0.785`, `1.571`) e **`skyColor`** come: (`0.0 0.2 0.7`, `0.0 0.5 1.0`, `1.0 1.0 1.0`). In questo nodo si definisce un cielo blu al polo superiore, si crea un gradiente con un colore più scuro fino ad un angolo di 45° (`0.785`). Da qui viene realizzato un altro gradiente fino al colore bianco, ad un angolo di 90° (`1.571`).
- Si possono definire angoli fino a 180° (polo inferiore). Se si definisce con un angolo inferiore a 180°, l'ultimo colore è applicato sfumato fino al polo inferiore.

I colori del suolo VRML

- Analogamente al cielo, il suolo VRML è da immaginare come una sfera infinitamente larga che circonda il mondo virtuale.
- Con tecniche analoghe a quelle viste per il cielo, si possono specificare gradienti di colore anche per il suolo.
- I gradienti di colore del suolo vengono definiti mediante i campi **`groundColor`** e **`groundAngle`** del nodo **`Background`**.
- I colori tra due **`groundAngle`** vengono fatti variare in maniera morbida, in modo da creare il gradiente di colore.

Immagini panoramiche

- Le sfere associate al cielo ed al suolo VRML forniscono un modo per colorare in modo elementare la scena.
- Su questo sfondo colorato possono essere inserite immagini di paesaggi montani e cittadini in modo da rendere ancora più interessante il monod virtuale.
- Ciò viene realizzato concettualmente inserendo una scatola grande all'inf nito dentro le sfere del cielo e del suolo del mondo virtuale.
- Possono essere inserite immagini panoramiche in ciascun lato di questa scatola usando i campi **`frontUrl`**, **`backUrl`**, **`leftUrl`**, **`rightUrl`**, **`topUrl`**, **`bottomUrl`**.

Background binding

- Nel mondo virtuale ci può essere soltanto un background attivo.
- Nel caso in cui serva modificare la scena (un paesaggio che per esempio passi da un paesaggio all'alba al sole splendente) si devono creare diversi nodi `Background` e poi usare in concetto di **background binding** per attivare quello di interesse.
- Il browser VRML mantiene una pila (stack) dei nodi background disponibili. La pila ha una cima ed un fondo. Si può mettere un nodo in cima alla pila o rimuoverlo dal fondo della pila.

Background binding (i)

- Per **Background binding** si intende il processo di inserimento di un nodo **`Background`** in cima alla pila.
- Quando un nodo **`Background`** è bound, viene posto in cima alla pila.
- Quando un nodo **`Background`** è unbound, viene rimosso dalla pila.
- Si può attivare un nodo **`Background`** inviando il valore **`TRUE`** all'eventIn **`set_bind`**, mentre si disattiva inviando il valore **`FALSE`** allo stesso.

Background binding (ii)

- Quando un nodo **`Background`** viene attivato, esso genera il valore **`TRUE`** attraverso l'eventOut **`isBound`**.
- Mediante il binding si possono mettere diversi nodi **`Background`** nella pila. Se avviene l'unbound di un nodo, viene attivato il successivo nodo nella pila che controllerà la colorazione del mondo virtuale.
- Se vengono unbound tutti i nodi della pila, diventa attivo il nodo di default (nero)
- Quando il browser legge il file **`.wrl`**, viene posto in cima alla pila ed attivato il primo nodo **`Background`** che viene incontrato.

Il nodo **Background**

- Il nodo **Background** crea uno sfondo per il mondo virtuale, mediante due sfere inf nite per il cielo e per il suolo ed una scatola inf nita per i panorami di sfondo. Il visitatore non può mai avvicinarsi alle sfere o alla scatola.

```
Background {
        skyColor        [0.0 0.0 0.0]   # exposedField MFColor
        skyAngle        [ ]             # exposedField MFFloat
        groundColor     [ ]               # exposedField MFColor
        groundAngle     [ ]             # exposedField MFFloat
        backUrl         [ ]             # exposedField MFString
        bottomUrl           [ ]           # exposedField MFString
        frontUrl        [ ]             # exposedField MFString
        leftUrl         [ ]             # exposedField FString
        rightUrl        [ ]             # exposedField MFString
        topUrl          [ ]             # exposedField MFString
        set_bind                          # eventIn       SFBool
        isBound                         # eventOut      SFBool
         }
```

Il nodo Background

- Il valore dell'exposed-field **skyColor** indica il colore RGB da usare per colorare la sfera del cielo VRML.
- Il valore dell'exposed-field **skyAngle** indica l'angolo al quale viene usato **skyColor** per colorare la sfera del cielo VRML. Gli angoli devono essere elencat in ordine crescente. Se ci sono *n* **skyColor** devono esserci *n-1* **skyAngle**.
- Il valore dell'exposed-field **groundColor** indica il colore RGB da usare per colorare il suolo VRML.
- Il valore dell'exposed-field **groundAngle** indica l'angolo al quale viene usato **groundColor** per colorare la sfera del suolo VRML. Gli angoli devono essere elencati in ordine crescente e devono essere in numero *n-1* se *n* sono i **groundColor**.
- I valori di **frontUrl**, **backUrl**, **LeftUrl**, **rightUrl**, **topUrl** e **bottomUrl** indicano come accedere ai file associati alle immagini.

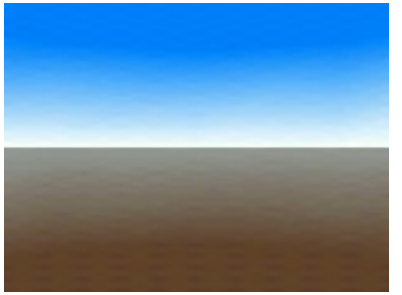
Esempi: colorazione del cielo

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Background {
   skyColor [
   0.0 0.2 0.7,
   0.0 0.5 1.0,
   1.0 1.0 1.0
   ]
   skyAngle [ 1.309, 1.571 ]
}
```

scena VRML

Esempi: colorazione del suolo

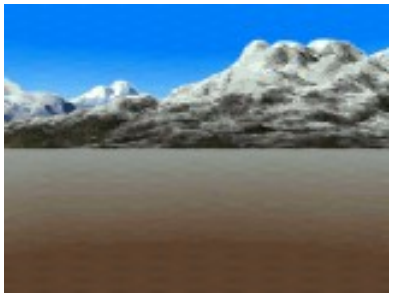
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Background {
   skyColor [
   0.0 0.2 0.7,
   0.0 0.5 1.0,
   1.0 1.0 1.0
   ]
   skyAngle [ 1.309, 1.571 ]
   groundColor [
   0.1 0.10 0.0,
   0.4 0.25 0.2,
   0.6 0.60 0.6,
   ]
   groundAngle [ 1.309, 1.571 ]
}
```



scena VRML

Esempi: aggiunta di un panorama

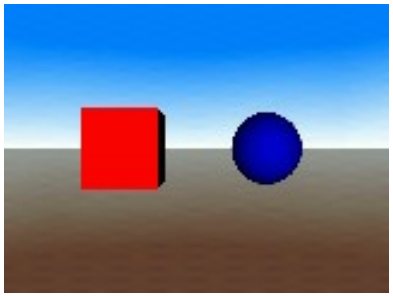
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Background {
   skyColor [
   0.0 0.2 0.7,
   0.0 0.5 1.0,
   1.0 1.0 1.0
   ]
   skyAngle [ 1.309, 1.571 ]
   groundColor [
   0.1 0.10 0.0,
   0.4 0.25 0.2,
   0.6 0.60 0.6,
   ]
   groundAngle [ 1.309, 1.571 ]
   frontUrl "mountns.png"
   backUrl  "mountns.png"
   leftUrl  "mountns.png"
   rightUrl "mountns.png"
}
```



mountns.png

scena VRML

Esempi: background binding



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Initial background
    Background {
          skyColor [ 0.0 0.2 0.7,  0.0 0.5 1.0,  1.0 1.0 1.0 ]
          skyAngle [ 1.309, 1.571 ]
          groundColor [ 0.1 0.1 0.0,  0.4 0.25 0.2,  0.6 0.6 0.6 ]
          groundAngle [ 1.309, 1.571 ]
    },
    # Alternate backgrounds
    DEF AltBack1 Background {
          skyColor [ 1.0 0.0 0.0,  1.0 0.4 0.0,  1.0 1.0 0.0 ]
          skyAngle [ 1.309, 1.571 ]
          groundColor [ 0.1 0.1 0.0,  0.5 0.25 0.2,  0.6 0.6 0.2 ]
          groundAngle [ 1.309, 1.571 ]
    },
    DEF AltBack2 Background {
          skyColor [ 1.0 0.0 0.8,  0.5 0.0 0.8,  0.0 0.0 0.8 ]
          skyAngle [ 1.309, 1.571 ]
          groundColor [ 0.0 0.0 0.1,  0.0 0.1 0.3,  0.3 0.3 0.6 ]
          groundAngle [ 1.309, 1.571 ]
    },
    # Shapes to act as buttons
    Transform {
          translation -2.0 0.0 0.0
          children [
                Shape {
                      appearance Appearance {
                            material Material {
                                  diffuseColor 1.0 0.0 0.0
                            }
                      }
                geometry Box { }
                ,
                DEF TouchBox TouchSensor { }
                   ]
                 },
                Transform {
                            translation 2.0 0.0 0.0
                            children [
                             Shape {
                              appearance Appearance {
                               material Material {
                               diffuseColor 0.0 0.0 0.8
                              }
                             }

                geometry Sphere { }
                            },
                            DEF TouchSphere TouchSensor { }
                              ]
                        }
 ]
}
ROUTE TouchBox.isActive    TO AltBack1.set_bind
ROUTE TouchSphere.isActive TO AltBack2.set_bind
```

scena VRML

Virtual Reality Modeling Language

VRML

Aggiunta di nebbia

Nebbia

- Dopo aver costruito le forme del mondo virtuale, si possono aggiungere gli effetti di un'atmosfera di nebbia virtuale usando il nodo **`Fog`**.
- Il nodo **`Fog`** consente di controllare i colori dell'atmosfera. Come per la nebbia nel mondo reale la nebbia virtuale possiede due attributi: il colore e la densità.
- Per default la nebbia è di colore bianco, ma può essere di qualsiasi colore.
- Lo spessore della nebbia è controllato indicando l'intervallo di visibilità, il quale specifica la distanza tra l'osservatore ed il punto nel quale la nebbia oscura completamente le forme del mondo virtuale. Le forme prossime all'osservatore sono chiaramente visibili, mentre quelle prossime all'intervallo di visibilità sono quasi oscurate.

Nebbia (i)

- Se si usa un intervallo di visibilità molto alto, si finisce per creare un alone che pervade il mondo virtuale senza pregiudicare la visibilità delle forme.
- Se si indica un valore basso, la nebbia risulta molto fitta e pregiudica la visibilità delle forme.
- Si può controllare la profondità della nebbia specificandone il tipo.
- Una nebbia **lineare** ha la proprietà di variare di intensità linearmente.
- Una nebbia di tipo **esponenziale** varia di intensità secondo una legge esponenziale e fornisce una sensazione di nebbia molto più realistica.

Nebbia (ii)

- Nel mondo reale la nebbia origina da una miriade di piccole gocce d'acqua sospese nell'atmosfera.
- Una simulazione rigorosa il computer dovrebbe calcolare gli effetti di ognuna delle particelle di acqua sospese. Un simile sforzo computazionale non è pensabile con le risorse che abbiamo.
- VRML simula la nebbia con un approccio più semplice, riducendo l'effetto nebbia ad un problema di colore della forma: quanto più l'osservatore è vicino alla forma, tanto più la forma apparirà del suo colore; quanto più l'osservatore è lontano dalla forma, tanto più la forma assume il colore della nebbia, fino a sparire alla distanza dell'intervallo di visibilità.

Nebbia (iii)

- La nebbia definita mediante il nodo **`Fog`** non agisce sullo sfondo della scena definito attraverso il nodo **`Background`**.
- Se si intende usare uno sfondo per la scena insieme alla nebbia occorre analizzare i due effetti insieme ed intervenire sull'immagine dello sfondo per applicare l'effetto nebbia desiderato direttamente sull'immagine.
- Nei casi in cui la nebbia sia molto densa, conviene non mettere immagini panoramiche di sfondo, ma colorare il cielo VRML dello stesso colore della nebbia.
- Coordinando le proprietà dello sfondo e della nebbia si possono creare ambienti molto realistici.

Fog binding

- Come nel caso del background, si può specificare il nodo **`Fog`** una sola volta nel mondo virtuale.
- Nel caso si desideri modificare il tipo di nebbia nel mondo virtuale, si possono definire più nodi **`Fog`** ed utilizzare il fog binding per attivare selettivamente i vari nodi modificare l'aspetto della scena.
- Analogamente al nodo **`Background`**, il browser VRML mantiene una pila (stack) di nodi. I nodi possono essere messi in cima alla pila o essere rimossi dalla pila.
- Il nodo in cima alla pila è quello attivo ed è quello usato dal browser per stabilire l'apparenza attuale del mondo virtuale.

Fog binding (i)

- Per **Fog binding** si intende il processo di inserimento di un nodo **`Fog`** in cima alla pila.
- Quando un nodo **`Fog`** è **bound**, viene posto in cima alla pila.
- Quando un nodo **`Fog`** è ***unbound***, viene rimosso dalla pila.
- Si può attivare un nodo **`Fog`** inviando il valore **`TRUE`** all'eventIn **`set_bind`** mentre si disattiva inviando il valore **`FALSE`** allo stesso.
- Quando un nodo **`Fog`** viene attivato, esso genera il valore **`TRUE`** attraverso l'eventOut **`isBound`**.

Fog binding (ii)

- Mediante il binding si possono mettere diversi nodi **`Fog`** nella pila. Se si fa l'**unbound** di un nodo, viene attivato il successivo nodo nella pila che controllerà la nebbia del mondo virtuale.
- Se un nodo viene attivato al posto di un altro, il nodo che viene disattivato invia il valore **`FALSE`** attraverso l'eventOut **`isBound`**. Usando questo output si può tenere traccia di quale nodo è in cima alla pila dei nodi.
- Quando il browser legge il file VRML, il primo nodo **`Fog`** che viene incontrato, viene posto in cima alla pila ed attivato.

Il nodo Fog

- Il nodo **Fog** crea un'atmosfera nebbiosa nel mondo virtuale.

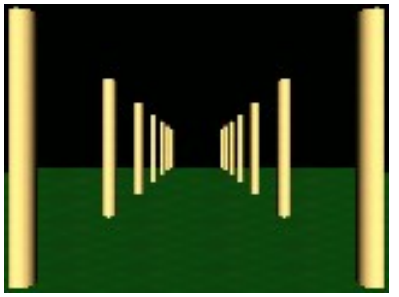
```
Fog {
      color              1.0 1.0 1.0       # exposedField SFcolor
      visibilityRange 0.0                  # exposedField SFFloat
      fogType      "LINEAR"                # exposedField MFcolor
      set_bind                             # eventIn      SFBool
      isBound                              # eventOut     SFBool
}
```

Il nodo Fog

- Il valore dell'exposed-field **color** indica il colore RGB della nebbia in VRML.
- Il valore dell'exposed-field **visibilityRange** indica la distanza dell'osservatore, misurata nel sistema di coordinate attuali, nella quale le forme del mondo virtuale vengono completamente oscurate dalla nebbia. Un valore minore o uguale a **0.0** disabilita il nodo. Il default è **0.0**.
- Il valore dell'exposed-field **fogType** indica il tipo di nebbia. I possibii valori sono **"LINEAR"** per un gradiente lineare nell'intensità della nebbia e **"EXPONENTIAL"** per un gradiente esponenziale. Il valore di default è **"LINEAR"**.
- Il tipo di nebbia del mondo virtuale può essere dinamicamentemodificato inviando degli eventi agli eventIn impliciti **set_color**, **set_visibilityRange** e **set_fogType**. Una volta cambiati i valori, i nuovi vengono propagati usando gli eventOut impliciti **color_changed**, **visibilityRange_changed**, **fogType_changed**.

Struttura base

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Ambient lighting
    DirectionalLight {
         direction 0.0 -1.0 -1.0
         intensity 0.2
         ambientIntensity 1.0
    },
    # Floor
    Shape {
         appearance Appearance {
              material Material {
                   ambientIntensity 0.5
                   diffuseColor 0.2 0.8 0.2
              }
         }
         geometry Box { size 50.0 0.01 50.0 }
    },
    # Pair of columns
    DEF ColumnPair Group {
         children [
              Transform {
                   translation -4.0 3.0 0.0
                   children DEF Column Shape {
                        appearance Appearance {
                             material Material {
                                  diffuseColor 1.0 0.8 0.5
                             }
                        }
                        geometry Cylinder {
                             radius 0.3
                             height 6.0
                        }
                   }
              },
              Transform {
                   translation 4.0 3.0 0.0
                   children USE Column
              }
         ]
    },
    # Several more pairs of columns
    Transform { translation 0.0 0.0  -8.0  children USE ColumnPair },
    Transform { translation 0.0 0.0   8.0  children USE ColumnPair },
    Transform { translation 0.0 0.0 -16.0  children USE ColumnPair },
    Transform { translation 0.0 0.0  16.0  children USE ColumnPair },
    Transform { translation 0.0 0.0 -24.0  children USE ColumnPair },
    Transform { translation 0.0 0.0  24.0  children USE ColumnPair },
    ]
}
```

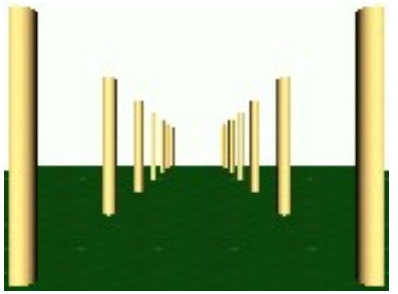


scena VRML

see the VRML scene

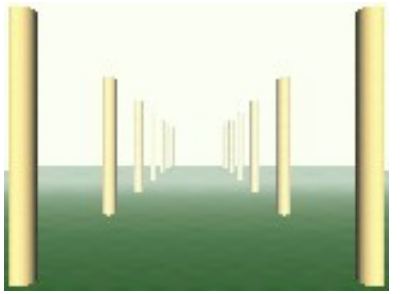
Nebbia Fogtype="LINEAR" visibilityRange=100

```
##VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   Fog {
       color 1.0 1.0 1.0
       fogType "LINEAR"
       visibilityRange 100.0
   },
   Background { skyColor 1.0 1.0 1.0 },
   Inline { url "fogworld.wrl" }
   ]
}
```



scena VRML

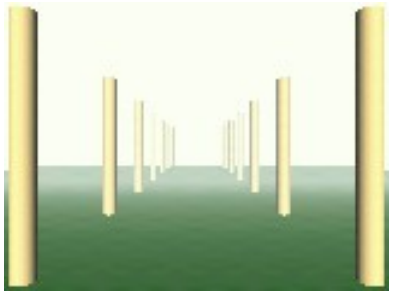
Nebbia Fogtype="LINEAR" visibilityRange=40



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   Fog {
       color 1.0 1.0 1.0
       fogType "LINEAR"
       visibilityRange 40.0
   },
   Background { skyColor 1.0 1.0 1.0 },
   Inline { url "fogworld.wrl" }
   ]
}
```

scena VRML

Nebbia Fogtype="LINEAR" visibilityRange=30



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   Fog {
       color 1.0 1.0 1.0
       fogType "LINEAR"
       visibilityRange 30.0
   },
   Background { skyColor 1.0 1.0 1.0 },
   Inline { url "fogworld.wrl" }
   ]
}
```

scena VRML

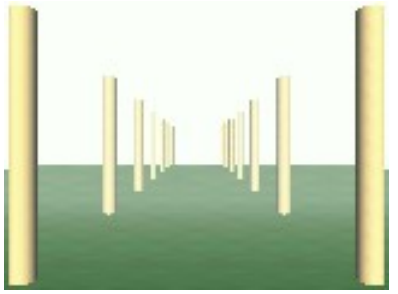
Nebbia Fogtype="LINEAR" visibilityRange=20

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   Fog {
       color 1.0 1.0 1.0
       fogType "LINEAR"
       visibilityRange 20.0
   },
   Background { skyColor 1.0 1.0 1.0 },
   Inline { url "fogworld.wrl" }
   ]
}
```

scena VRML

Nebbia Fogtype="EXPONENTIAL" visibilityRange=100

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   Fog {
       color 1.0 1.0 1.0
       fogType "EXPONENTIAL"
       visibilityRange 100.0
   },
   Background { skyColor 1.0 1.0 1.0 },
   Inline { url "fogworld.wrl" }
   ]
}
```



scena VRML

Nebbia Fogtype="EXPONENTIAL" visibilityRange=40

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   Fog {
       color 1.0 1.0 1.0
       fogType "EXPONENTIAL"
       visibilityRange 40.0
   },
   Background { skyColor 1.0 1.0 1.0 },
   Inline { url "fogworld.wrl" }
   ]
}
```

scena VRML

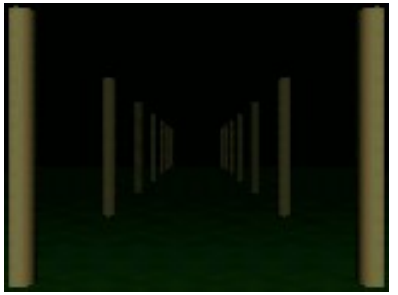
Nebbia Fogtype="EXPONENTIAL" visibilityRange=20

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   Fog {
       color 1.0 1.0 1.0
       fogType "EXPONENTIAL"
       visibilityRange 20.0
   },
   Background { skyColor 1.0 1.0 1.0 },
   Inline { url "fogworld.wrl" }
   ]
}
```


scena VRML

Nebbia colorata

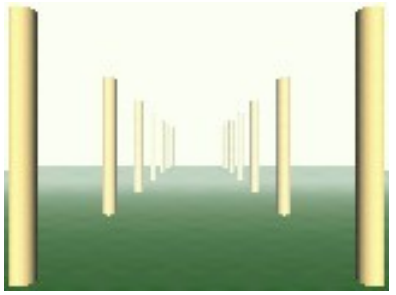
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   Fog {
       color 0.0 0.0 0.0
       fogType "EXPONENTIAL"
       visibilityRange 40.0
   },
   Background { skyColor 1.0 1.0 1.0 },
   Inline { url "fogworld.wrl" }
   ]
}
```



scena VRML

Fog binding

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   # Initial fog and background
   Fog {
       color 1.0 1.0 1.0
       fogType "LINEAR"
       visibilityRange 40.0
   },
   Background { skyColor 1.0 1.0 1.0 },
   # Alternate fog background
   DEF AltFog Fog {
       color 1.0 0.0 0.0
       fogType "LINEAR"
       visibilityRange 30.0
   },
   DEF AltBack Background { skyColor 1.0 0.0 0.0 },
   # Test world
   Transform {
       children [
           Inline { url "fogworld.wrl" },
           DEF TouchWorld TouchSensor { }
       ]
   }
   ]
}
ROUTE TouchWorld.isActive TO AltFog.set_bind
ROUTE TouchWorld.isActive TO AltBack.set_bind
```



scena VRML

Virtual Reality Modeling Language

VRML

Aggiunta di suoni

Aggiunta di suoni

- Si può aumentare il realismo del mondo virtuale aggiungendo suoni e filmati attraverso i nodi **`AudioClip`**, **`MovieTexture`** e **`Sound`**.
- Si possono creare suoni di ambiente, come ad esempio una musica di sottofondo in un negozio.
- Si possono creare suoni che siano animati da circuiti, come ad esempio lo sbattere di una porta che si attiva quando la porta si chiude.
- Si può determinare da quale parte far sentire un suono, se ascoltato mediante una cuffia o delle casse acustiche stereofoniche.

Aggiunta di suoni (i)

- L'aggiunta di suoni al mondo virtuale coinvolge due componenti: la sorgente del suono e il trasmettitore.
- La sorgente audio fornisce un segnale audio, come un lettore CD o un registratore video
- Il trasmettitore audio trasforma il segnale sorgente in un suono udibile attraverso delle casse stereofoniche o una cuffia.
- In VRML la sorgente audio viene specificata attraverso il nodo **`AudioClip`**. Il nodo **`AudioClip`** specifica un campo **`url`** il cui valore indica lo Unified Resource Locator (URL) associato al file contenente il suono da riprodurre.

Aggiunta di suoni (ii)

- Sono supportati diversi tipi di file suono (anche in funzione del software utilizzato come browser VRML):
  - Formato WAV (Waveform), uno dei formati di file audio digitali più popolari
  - Formato General MIDI (Musical Instrument Digital Interface) type 1, che indica al sintetizzatore presente nella scheda audio del computer come riprodurre un pezzo di musica.
  - Formato MPEG (generalmente utilizzato per riprodurre filmato e suono).
  - Formato MP3 (MPEG Layer 3 Audio Codec), il formato più popolare per la riproduzione e la diffusione in rete Internet di brani musicali, definito nello standard ISO 11172-3
- Una sorgente audio può essere specificata anche attraverso il nodo **`MovieTexture`**, il quale consente di specificare sia un film che un suono da riprodurre. Si possono creare presentazioni multimediali sincronizzate usando lo stesso nodo sia per una texture video che per un suono.

Aggiunta di suoni (iii)

- Per esempio si può creare una televisione che riproduce un filmato video in formato MPEG facendo ascoltare dagli altoparlanti virtuali della televisione un file audio.
- Sono disponibili in rete diversi strumenti per la produzione o la manipolazioni di questi tipi di file.
- Per una registrazione fedele digitale di un brano della durata di un minuto possono servire anche 10MB di spazio disco.
- Poiché i file MIDI contengono le istruzioni per la riproduzione, piuttosto della rappresentazione digitale del suono, le loro dimensioni sono molto contenute e sono molto adatti per la riproduzione di musica di sottofondo.

Controllo della sorgente del suono

- I due nodi **`AudioClip`** e **`MovieTexture`** forniscono entrambi gli strumenti per controllare come e quando riprodurre un suono.
- Per entrambi i nodi i campi **`startTime`** e **`stopTime`** definiscono i tempi assoluti di inizio e fine riproduzione sonora. Il campo **`loop`** indica se la riproduzione va effettuata all'infinito.
- La velocità di riproduzione (espressa come fattore relativo) può essere controllata attraverso i campi **`pitch`** per il nodo **`AudioClip`** e **`speed`** per il nodo **`MovieTexture`**. Un valore di **`0.5`** riduce della metà la velocità di riproduzione, mentre **`2.0`** la raddoppia. Il valore di default è **`1.0`**.

Controllo della riproduzione del suono

- Il nodo **`Sound`** descrive un emettitore di suono nel nostro mondo virtuale.
- Il valore del campo **`source`** specifica un nodo **`AudioClip`** o **`MovieTexture`** che specifica la sorgente del suono.
- I campi **`location`** e **`direction`** consentono di posizionare l'emittente del suono ed orientarla in una determinata direzione.
- Il campo **`intensity`** fornisce un controllo di volume principale del suono

Controllo della riproduzione del suono (i)

- Come nel mondo reale, si può fare in modo che l'intensità del suono dipenda dalla distanza dell'osservatore rispetto alla fonte di emissione del suono.
- Si può allo scopo def nire una regione 3-D che circonda il punto di emissione del suono mediante def nizione di un elissoide di intervallo minimo ed un elissoide di intervallo massimo del campo di azione del suono stesso:
  - Finché l'osservatore si trova nell'elissoide di intervallo minimo, sente il suono al massimo volume
  - Tra l'elissoide di intervallo minimo e quello massimo, il suono diminuisce mano mano che ci si allontana
  - Se è fuori dall'elissoide di intervallo massimo, non sente alcun suono perché troppo distante

Controllo della riproduzione del suono (ii)

- L' elissoide di intervallo minimo viene definito attraverso i campi **`minFront`** e **`minBack`** del nodo **`Sound`**.
- L' elissoide di intervallo massimo viene definito attraverso i campi **`maxFront`** e **`maxBack`** del nodo **`Sound`**.
- In entrambi i casi la distanza è misurata lungo una linea immaginaria che parte dall'emittente del suono e va all'osservatore.
- Definendo opportunamente questi due elissoidi si può controllare in dettaglio l'area nella quale il suono è udibile.
- Si può usare un nodo per far udire una musica di sottofondo in una stanza, mentre in un'altra si può far udire del rumore e controllare i campi di azione di entrambe le emittenti di suoni.

Controllo della riproduzione del suono (iii)

- Sound spatialization è una tecnica di elaborazione dei segnali digitali, che fa si che un suono digitale appaia essere emesso in un punto preciso dello spazio 3-D.
- Usando sound spatialization si può fare in modo che il suono si muova in alto e in basso, a sinistra e a destra, avanti e indietro ed addirittura roteare intorno alla testa ...
- Si può attivare sound spatialization mediante il campo booleano `spatialize` del nodo `Sound`.

Il nodo Sound

- Il nodo **Sound** crea un emettitore di suoni, la cui azione è percepibile entro una regione elissoidale,

```
Sound {
        source          NULL            #exposedField MFString
        intensity       1.0             #exposedField SFFloat
        location        0.0 0.0 0.0     #exposedField SFVec3f
        direction       0.0 0.0 1.0     #exposedField SFVec3f
        minFront        1.0             #exposedField SFFloat
        minBack         1.0             #exposedField SFFloat
        maxFront        10.0            #exposedField SFFloat
        maxBack         10.0            #exposedField SFFloat
        priority        0.0             #exposedField SFFloat
        spatialize      TRUE            #field SFBool
}
```

Il nodo AudioClip

- Il nodo **AudioClip** descrive una sorgente di suono e può essere utilizzato come valore del campo **source** del nodo **Sound**.

```
AudioClip {
                url             [ ]     #exposedField MFString
                duration        “”      #exposedField SFString
                startTime   0.0         #exposedField SFTime
                stopTime        0.0     #exposedField SFTime
                pitch       1.0         #exposedField SFFloat
                loop        FALSE       #exposedField SFBool
                isActive                #eventOut SFBool
                duration_changed        #eventOut SFFLoat
}
```

Il nodo MovieTexture

- Il nodo **MovieTexture** descrive un file video contenente suono (MPEG-1) e può essere utilizzato come valore del campo **source** del nodo **Sound**.

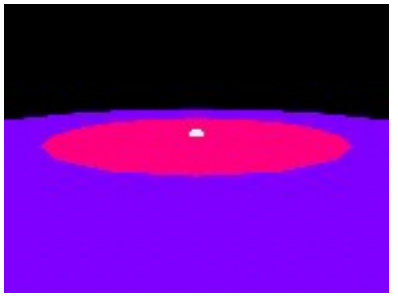
```
MovieTexture {
        url             [ ]         #exposedField MFString
        startTime       0.0         #exposedField SFTime
        stopTime        0.0         #exposedField SFTime
        speed           1.0         #exposedField SFFloat
        loop            FALSE       #exposedField SFBool
        repeatS         TRUE        #field SFBool
        repeatT         TRUE        #field SFBool
        isActive                    #eventOut SFBool
        duration_changed            #eventOut SFFLoat
}
```

Nodo Sound

- Il campo **`intensity`** aggiusta la riproduzione del suono (in decibels) tra un minimo di 0db (non viene udito nessun suono) ed un massimo di 20db. Valori intermedi di **`intensity`** vengono interpolati linearmente nell'intervallo 0-20db.
- Il campo **`location`** rappresenta la posizione nello spazio della sorgente sonora, rispetto al sistema di coordinate corrente.
- Il campo **`priority`** istruisce il browser su quale nodo ha priorità maggiore di riproduzione, qualora siano attivi più nodi in una stessa porzione dello spazio.

Confini dell'emittente

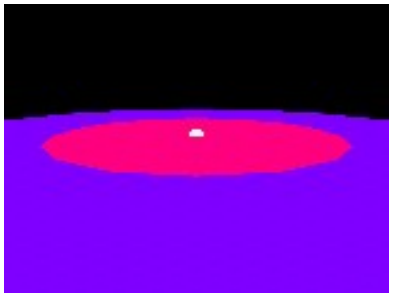
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Emitter marker
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor  0.0 0.0 0.0
                emissiveColor 1.0 1.0 1.0
            }
        }
        geometry Sphere { radius 0.25 }
    },
    # Minimum range ellipsoid (circle) marker
    DEF MinMarker Shape {
        appearance Appearance {
            material Material {
                diffuseColor  0.0 0.0 0.0
                emissiveColor 1.0 0.0 0.5
            }
        }
        geometry Cylinder {
            radius 5.0
            height 0.01
            side   FALSE
            bottom FALSE
        }
    },
    # Maximum range ellipsoid (circle) marker
    DEF MaxMarker Shape {
        appearance Appearance {
            material Material {
                diffuseColor  0.0 0.0 0.0
                emissiveColor 0.5 0.0 1.0
            }
        }
        geometry Cylinder {
            radius 10.0
            height 0.001
            side   FALSE
            bottom FALSE
        }
    },
    ]
}
```



sndmark.wrl

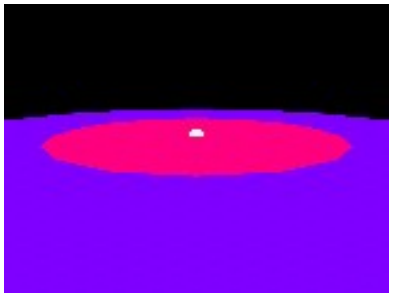
Prova dell'emittente

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   # Sound emitter
   Sound {
       source AudioClip {
           url "willow1.wav"
           loop TRUE
       }
       minFront 5.0
       minBack  5.0
       maxFront 10.0
       maxBack  10.0
   },
   # Sound emitter markers
   Inline { url "sndmark.wrl" }
   ]
}
```

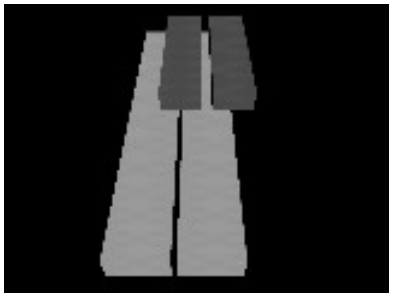


Controllo del suono

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   # Sound emitter
   Sound {
       source DEF Source AudioClip {
           url "tone1.wav"
           loop FALSE
       }
       minFront 5.0
       minBack  5.0
       maxFront 10.0
       maxBack  10.0
   },
   # Sound emitter markers
   Inline { url "sndmark.wrl" },
   # Sensor
   DEF Touch TouchSensor { }
   ]
}
ROUTE Touch.touchTime TO Source.set_startTime
```

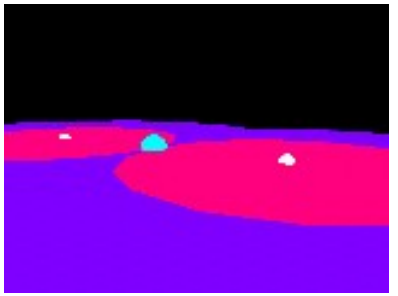


Velocità di riproduzione `pitch`



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Middle C (C4)
    Transform {
        children [
            DEF WhiteKey Shape {
                appearance Appearance {
                    material Material { }
                }
                geometry Box { size 0.23 0.1 1.5 }
            }
            DEF C4 TouchSensor { },
            Sound {
                source DEF PitchC4 AudioClip {
                    url "tone1.wav"
                    pitch 1.0
                }
            }
        ]
    }
    # C# above middle C (Cs4)
    Transform { translation 0.125 0.1 -0.375
        children [
            DEF BlackKey Shape {
                appearance Appearance {
                    material Material {
                        diffuseColor 0.4 0.4 0.4
                    }
                }
                geometry Box { size 0.2 0.1 0.75 }
            }
            DEF Cs4 TouchSensor { }
            Sound {
                source DEF PitchCs4 AudioClip {
                    url "tone1.wav"
                    pitch 1.059
                }
            }
        ]
    }
    # D above middle C (D4)
    Transform { translation 0.25 0.0 0.0
        children [
            USE WhiteKey,
            DEF D4 TouchSensor { }
            Sound {
                source DEF PitchD4 AudioClip {
                    url "tone1.wav"
                    pitch 1.122
                }
            }
        ]
    }
    # D# above middle C (Ds4)
    Transform { translation 0.375 0.1 -0.375
        children [
            USE BlackKey,
            DEF Ds4 TouchSensor { }
            Sound {
                source DEF PitchDs4 AudioClip {
                    url "tone1.wav"
                    pitch 1.189
                }
            }
        ]
    }
    ]
}
ROUTE C4.touchTime  TO PitchC4.set_startTime
ROUTE Cs4.touchTime TO PitchCs4.set_startTime
ROUTE D4.touchTime  TO PitchD4.set_startTime
ROUTE Ds4.touchTime TO PitchDs4.set_startTime
```

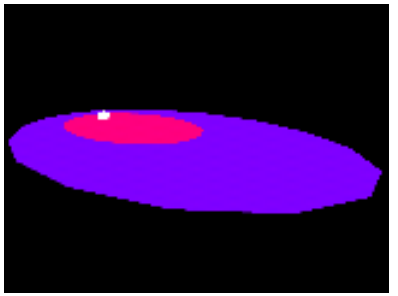
Animazione dei confini dell'emittente



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Origin marker
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor 0.0 0.0 0.0
                emissiveColor 0.0 1.0 1.0
            }
        }
        geometry Sphere { radius 0.5 }
    },
    # Sound emitter 1
    DEF Emitter1 Transform {
        translation 5.0 0.0 0.0
        center -5.0 0.0 0.0
        children [
            Sound {
                source AudioClip {
                    url "willow1.wav"
                    loop TRUE
                }
                intensity 0.5
                minFront 5.0
                minBack  5.0
                maxFront 10.0
                maxBack  10.0
            },
            DEF SoundMarker Inline { url "sndmark.wrl" }
        ]
    },
    DEF Emitter1Clock TimeSensor {
        cycleInterval 15.0
        loop TRUE
    },
    DEF Emitter1Path OrientationInterpolator {
        key [ 0.0, 0.5, 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 ]
    },
    # Sound emitter 2
    DEF Emitter2 Transform {
        translation -5.0 0.0 0.0
        center 5.0 0.0 0.0
        children [
            Sound {
                source AudioClip {
                    url "drone1.wav"
                    loop TRUE
                }
                intensity 0.5
                minFront 5.0
                minBack  5.0
                maxFront 10.0
                maxBack  10.0
            },
            USE SoundMarker
        ]
    },
    DEF Emitter2Clock TimeSensor {
        cycleInterval 7.0
        loop TRUE
    },
    DEF Emitter2Path OrientationInterpolator {
        key [ 0.0, 0.5, 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 Emitter1Clock.fraction_changed TO Emitter1Path.set_fraction
ROUTE Emitter2Clock.fraction_changed TO Emitter2Path.set_fraction
ROUTE Emitter1Path.value_changed     TO Emitter1.set_rotation
ROUTE Emitter2Path.value_changed     TO Emitter2.set_rotation
```

Controllo della direzione del suono

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Directed sound emitter
    Sound {
        source AudioClip {
            url "willow1.wav"
            loop TRUE
        }
        direction 1.0 0.0 0.0
        minFront 5.0
        minBack  1.0
        maxFront 10.0
        maxBack  2.0
    }
    # Emitter marker
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor  0.0 0.0 0.0
                emissiveColor 1.0 1.0 1.0
            }
        }
        geometry Sphere { radius 0.25 }
    }
    # Minimum range ellipsoid marker
    Transform {
        translation 2.0 0.0 0.0
        scale 3.0 2.0 2.0
        children DEF MinMarker Shape {
            appearance Appearance {
                material Material {
                    diffuseColor  0.0 0.0 0.0
                    emissiveColor 1.0 0.0 0.5
                }
            }
            geometry Cylinder {
                radius 1.0
                height 0.01
                side   FALSE
                bottom FALSE
            }
        }
    }
    # Maximum range ellipsoid marker
    Transform {
        translation 4.0 0.0 0.0
        scale 6.0 4.0 4.0
        children DEF MaxMarker Shape {
            appearance Appearance {
                material Material {
                    diffuseColor  0.0 0.0 0.0
                    emissiveColor 0.5 0.0 1.0
                }
            }
            geometry Cylinder {
                radius 1.0
                height 0.001
                side   FALSE
                bottom FALSE
            }
        }
    }
    ]
}
```



Uso di MPEG

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Chassis
    Shape {
        appearance Appearance {
            material Material { diffuseColor 0.3 0.3 0.3 }
        }
        geometry Box { size 5.0 3.5 2.0 }
    }
    # Controls
    Shape {
        appearance Appearance {
            material Material { }
            texture ImageTexture {
                url "tvcntrl.jpg"
                repeatS FALSE
                repeatT FALSE
            }
        }
        geometry IndexedFaceSet {
            coord Coordinate {
                point [
                    1.75 -1.5 1.01,  2.40 -1.5 1.01,
                    2.40  1.5 1.01,  1.75  1.5 1.01,
                ]
            }
            coordIndex [ 0, 1, 2, 3 ]
            texCoord TextureCoordinate {
                point [
                    0.0 0.0,  1.0 0.0,
                    1.0 1.0,  0.0 1.0
                ]
            }
        }
    }
    # Screen
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor 0.0 0.0 0.0
                emissiveColor 1.0 1.0 1.0
            }
            texture DEF TV MovieTexture {
                url "tv.mpg"
                loop FALSE
                repeatS FALSE
                repeatT FALSE
            }
        }
        geometry IndexedFaceSet {
            coord Coordinate {
                point [
                    -2.35 -1.5 1.01,
                     1.65 -1.5 1.01,
                     1.65  1.5 1.01,
                    -2.35  1.5 1.01
                ]
            }
            coordIndex [ 0, 1, 2, 3 ]
        }
    }
    # Sound
    Sound {
        # Use MovieTexture as sound source
        source USE TV
        minFront 30.0
        minBack  30.0
        maxFront 100.0
        maxBack  100.0
    }
    # Trigger on touch
    DEF Touch TouchSensor { }
    ]
}
ROUTE Touch.touchTime TO TV.set_startTime
```

Virtual Reality Modeling Language

VRML

Forme splendenti

Forme splendenti

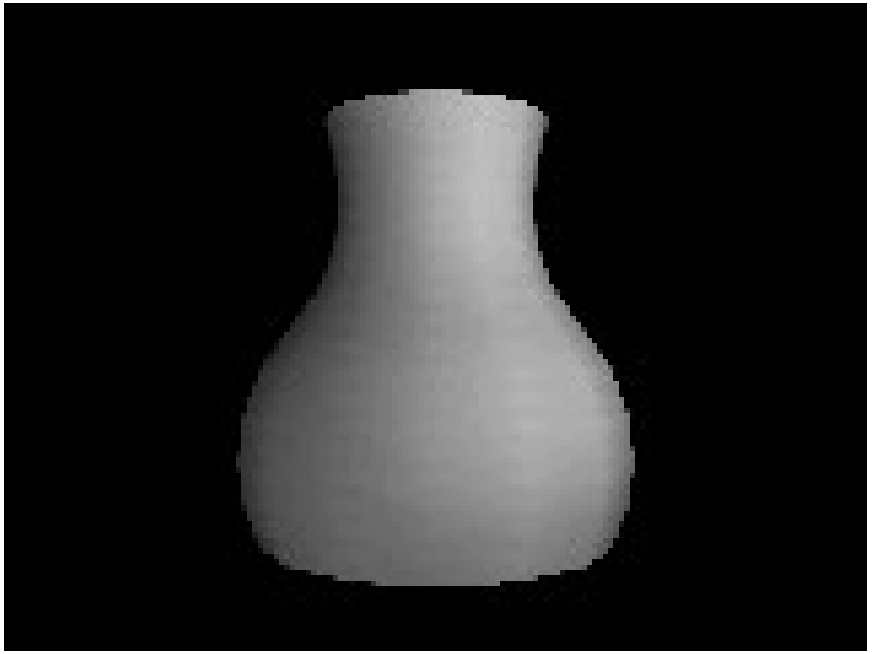
- Utilizzando il nodo `Material` ed il campo `diffuseColor`, si controlla il colore delle forme. Quando la forma viene illuminata dall'headlight o da luci addizionali, la forma appare costituita da materiale opaco.
- Utilizzando altri campi del nodo `Material` è possibile istruire il browser VRML sulla natura splendente della forma e simulare così forme metalliche, plastiche, etc.
- La colorazione e la resa della forma è controllata dai seguenti campi del nodo `Material`: `shininess`, `specularColor`, `ambientIntensity`, che contribuiscono insieme a determinare l'aspetto della forma.

Riflessione della luce

- Nel mondo reale quando la luce colpisce un materiale, alcuni raggi vengono assorbiti, altri vengono rif essi, alcuni sono trasmessi dalla superf cie.
- Una f nestra trasmette gran parte della luce, una parte viene assorbita e rif essa.
- Un tavolo invece, non trasmette luce; rif ette o assorbe la componente di luce del quale appare colorato al nostro occhio.
- La luce eventualmente trasmessa, attraversa il mondo virtuale ed illumina le varie superf ci che i raggi luminosi incontrano nella loro traiettoria.
- Per semplicità VRML usa un metodo approssimato per illuminare le forme in base alla rif essione della luce, che considera la **riflessione diffusa** e la **riflessione speculare** della luce.

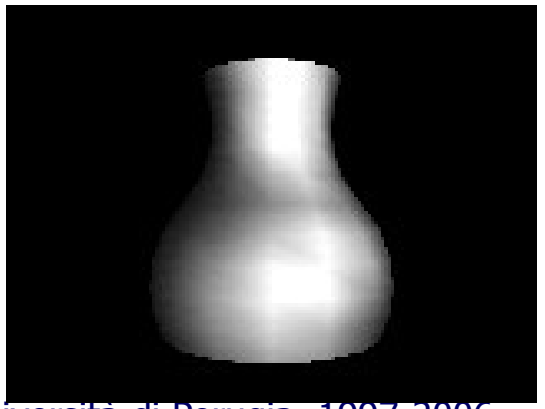
Riflessione diffusa

- La **riflessione diffusa** della luce rif ette i raggi in modo casuale, inviandoli nelle varie direzioni dello spazio.
- L'effetto è una superf cie omogeneamente illuminata, con le ombreggiature tenui.



Riflessione speculare

- La **riflessione speculare** della luce rif ette i raggi in modo matematico e prevedibile, consentendo la rif essione del mondo circostante sulla superf cie stessa .
- La rif essione del mondo circostante è molto complessa da calcolare, pertanto VRML semplif ca questo calcolo consentendo la sola rif essione di luci e non quella delle forme!

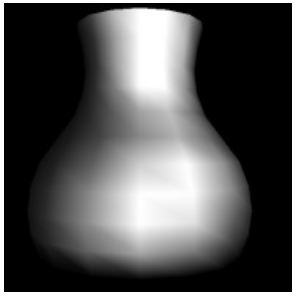
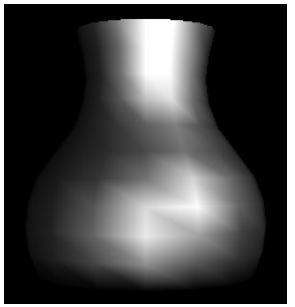
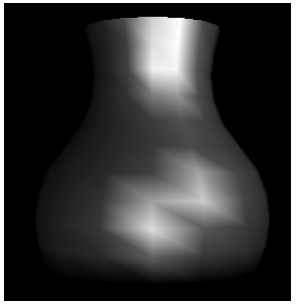


Riflessione speculare e diffusa

- La **riflessione speculare** e la **riflessione diffusa** vengono discriminate dal campo `shininess` del nodo `Material` che determina la componente relativa di rif essione della superf cie. Es:

```
material Material {
          ambientIntensity 0.4
          diffuseColor 0.15 0.15 0.15
          specularColor 0.70 0.70 0.70
          shininess
```

0.00 0.05 0.10 0.2

Specular color

- Il colore della **riflessione speculare** dipende da diversi fattori come le proprietà del materiale di cui è costituita la forma ed il colore della luce rif essa.
- Il calcolo della luce rif essa nel mondo reale che è in continua trasformazione è proibitivo in applicazioni graf che interattive.
- Essendo molto complessa la valutazione del colore della luce rif essa, VRML consente attraverso il campo `specularColor` del nodo `Material` di determinare il colore della luce rif essa dalla superf cie.

Ambient light

- Come già detto, la luce ambiente è data dalla sommatoria delle varie componenti di luce che sono presenti nel nostro mondo virtuale.
- Il campo `ambientIntensity` dei nodi `PointLight`, `SpotLight` e `DirectionalLight` controlla la quantità di luce ambientale nel mondo virtuale.
- E' possibile controllare come una forma reagisce ad una data quantità di luce ambientale attraverso il campo `ambientIntensity` del nodo `Material`.
- Un basso valore di `ambientIntensity` rende la forma meno soggetta ad essere inf uenzata dalla luce ambiente.

Forme plastiche e metalliche splendenti

- Nel mondo reale i metalli rif ettono i raggi luminosi in modo differente, a seconda della loro lunghezza d'onda. Ne consegue che la luce rif essa dal metallo cambia a seconda del tipo di metallo.
- La plastica emette sempre lo stesso colore di luce della luce che illumina la forma

Effetto	ambientIntensity	diffuseColor	specularColor	shininess
Alluminio	0.30	0.3 0.3 0.5	0.7 0.7 0.8	0.10
Rame	0.26	0.3 0.11 0.0	0.75 0.33 0.0	0.08
Oro	0.40	0.22 0.15 0.0	0.71 0.7 0.56	0.16
Porpora metallico	0.17	0.1 0.03 0.22	0.64 0.0 0.98	0.20
Rosso metallico	0.15	0.27 0.0 0.0	0.61 0.13 0.18	0.20
Plastica blu	0.10	0.2 0.2 0.71	0.83 0.83 0.83	0.12

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
                    }
```

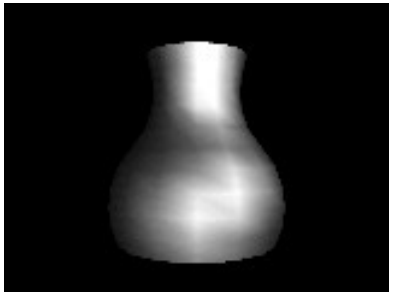
Vaso opaco

```
#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 3.0 3.0 3.0
        ambientIntensity 0.2
    },
    Shape {
        appearance Appearance {
            material Material {
                diffuseColor 0.40 0.40 0.40
            }
        }
        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,
            ]
        }
    }
    ]
}
```

Vaso brillante

```
#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 3.0 3.0 3.0
        ambientIntensity 0.2
    },
    Shape {
        appearance Appearance {
            material Material {
                ambientIntensity 0.4
                diffuseColor 0.15 0.15 0.15
                specularColor 0.70 0.70 0.70
                shininess 0.05
            }
        }
        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,
            ]
        }
    }
    ]
}
```

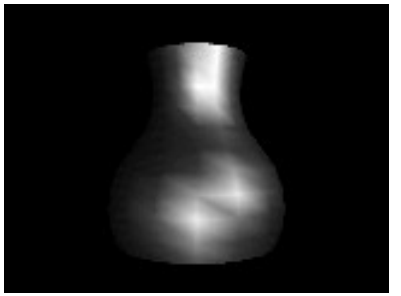
Vaso brillante (i)



```
#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 3.0 3.0 3.0
        ambientIntensity 0.2
    },
    Shape {
        appearance Appearance {
            material Material {
                ambientIntensity 0.4
                diffuseColor 0.15 0.15 0.15
                specularColor 0.70 0.70 0.70
                shininess 0.08
            }
        }
        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,
            ]
        }
    }
    ]
}
```

Vaso brillante (ii)

```
#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 3.0 3.0 3.0
        ambientIntensity 0.2
    },
    Shape {
        appearance Appearance {
            material Material {
                ambientIntensity 0.2
                diffuseColor 0.10 0.10 0.10
                specularColor 0.80 0.80 0.80
                shininess 0.16
            }
        }
        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,
            ]
        }
    }
    ]
}
```



Virtual Reality Modeling Language

VRML

Controllo del dettaglio

Controllo del dettaglio

- Se si desidera rappresentare le forme del mondo virtuale con molto dettaglio, il browser risulterà appesantito da questa funzione e apparirà probabilmente meno interattivo.
- Si è continuamente in presenza di due opposte esigenze: molto dettaglio per ottenere il massimo del realismo e rappresentazione veloce per accelerare l'interattività.
- Si può controllare il livello di dettaglio con cui vengono rappresentate le forme, mediante il nodo **`LOD`**.
- La tecnica è maturata nell'ambito dello sviluppo dei simulatori di volo utilizzati per istruire piloti e basato sulla simulazione di un aereo che vola sopra un paesaggio.
- Il livello di dettaglio della rappresentazione del terreno cambia, passando dalle zone vicine a quelle lontane.

Controllo del dettaglio (i)

- La rappresentazione di un albero, ad esempio può essere fatta con una forma molto grossolana, quando l'albero è molto distante.
- Il dettaglio aumenterà mano mano che l'albero si avvicina alla vista dell'aereo.
- Si parla di livello di dettaglio alto, medio e basso.
- Si può usare la stessa tecnica per la rappresentazione del mondo virtuale, inserendo per le forme complesse diverse forme a livelli di dettaglio diversi.
- Dopo aver creato le diverse forme corrispondenti a diversi livelli di dettaglio, si raggruppano insieme in un nodo **`LOD`** indicando le varie versioni nel campo **`level`**. Il browser viene istruito sul come gestire le varie forme in base alla distanza delle forme dall'osservatore.

Il gruppo LOD

- La distanza tra l'osservatore e la forme del gruppo **LOD** è chiamata *range*. Se si hanno tre rappresentazioni delle forme a diversi livelli di dettaglio, occorre specif care due valori di *range*, nel campo **range** del nodo **LOD**.
- Il primo valore di **range** indica la distanza alla quale effettuare il cambio tra la prima e la seconda rappresentazione della forma, il secondo valore quello al quale effettuare il cambio tra la seconda e la terza.
- Se il valore di **range** viene omesso, non si attiva il controllo manuale ed il browser sceglie autonomamente il criterio con il quale selezionare le forme all'interno del nodo **LOD**.
- La distanza è misurata tra l'osservatore ed il centro della scena della forma specif cata nel nodo **LOD**.
- Per default il centro coincide con il centro della scena. Se la forma è stata traslata, il nuovo centro della scena deve essere specif cato nel campo **center**.

Il nodo LOD

- Il nodo **LOD** raggruppa i vari nodi che descrivono una forma a diversi livelli di dettaglio (level of detail).

```
LOD {
        center      0.0 0.0 0.0  #field          SFVec3f
        level       [ ]          #exposedField   MFNode
        range       [ ]          #field          MFFloat
        }
```

Il nodo LOD

- I valori dell'exposed-field **level** specificano una lista di nodi da includere nel gruppo. Nel suo insieme il nodo è concepito per descrivere una singola forma a diversi livelli di dettaglio.
- La prima forma descrive la rappresentazione associata alla descrizione più dettagliata, continuando con le successive forme a scendere nel livello di dettaglio.
- Il valore di **center** indica la terna di valori associati alla coordinata 3-D del centro della forma descritto dal nodo **LOD**.

Il nodo LOD (i)

- I valori del campo **range** specif cano una lista di distanze osservatore-forma che vengono utilizzate per selezionare la rappresentazione della forma adatta allo specif co livello di dettaglio desiderato.
- Se le rappresentazioni della forma sono pari a *n*, occorrono *n-1* valori di **range** . Se i valori di **range** sono minori di n-1 (ad es. *m*), le ultime (*n-m-1*) rappresentazioni della forma vengono ignorate.
- I valori di **range** devono essere positivi ed elencati in ordine crescente.
- I valori devono essere scelti in modo tale che il cambio della forma associato al successivo livello di dettaglio non sia percepibile dall'osservatore.
- Siccome il browser sceglie da solo i valori di range in assenza di esplicite dichiarazioni, in genere è meglio lasciare questo campo vuoto.

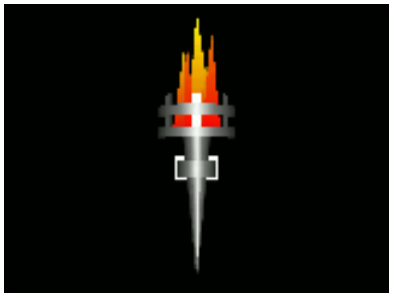
Il nodo LOD (ii)

- Le forme del gruppo **LOD** possono essere modif cati utilizzando l'*eventIn* implicito **set_level** e le nuove informazioni possono essere propagate attraverso l'*eventOut* **level_changed**.
- Le forme vengono rappresentate nel sistema di coordinate attuale. Eventuali trasformazioni effettuare con il nodo **Trasform** inf uenzano il nodo **LOD**.
- I nodi elencati nel nodo **LOD** continuano a ricevere *eventIn* e propagare *eventOut*, pur se non vengono selezionati per essere rappresentati.

Torcia ad alto dettaglio

168 coordinate

torch1.wrl



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Transform {
    translation 0.0 0.0 0.2
    scale 0.5 0.5 0.5
    children [
    # Torch handle
    Transform {
          translation 0.0 -0.75 0.0
          rotation 1.0 0.0 0.0 3.14
          children Shape {
                appearance DEF Gray Appearance {
                      material Material {
                            diffuseColor  0.4 0.4 0.4
                            specularColor 0.7 0.7 0.7
                      }
                }
                geometry Cone {
                      height 1.5
                      bottomRadius 0.15
                }
          }
    },
    # Fire pot
    DEF Ring Shape {
          appearance USE Gray
          geometry Cylinder {
                height 0.1
                radius 0.4
                top    FALSE
                bottom FALSE
          }
    },
    Transform { translation 0.0 0.2 0.0 children USE Ring },
    # Fire pot detail
    DEF Bar Shape {
          appearance USE Gray
          geometry IndexedFaceSet {
                coord Coordinate {
                      point [
                            0.04 0.00 0.38,  0.04 0.35 0.38,
                           -0.04 0.35 0.38, -0.04 0.00 0.38,
                      ]
                }
                coordIndex [ 0, 1, 2, 3 ]
          }
    },
    Transform { rotation 0.0 1.0 0.0 -1.571 children USE Bar },
    Transform { rotation 0.0 1.0 0.0 -0.785 children USE Bar },
    Transform { rotation 0.0 1.0 0.0  0.785 children USE Bar },
    Transform { rotation 0.0 1.0 0.0  1.571 children USE Bar },
    # Mounting bracket
    Transform {
          translation 0.0 -0.35 0.0
          children [
                Shape {
                      appearance USE Gray
                      geometry Cylinder {
                            height 0.15
                            radius 0.20
                      }
                },
                Transform {
                      translation 0.0 0.0 -0.2
                      children Shape {
                            appearance USE Gray
                            geometry Box { size 0.45 0.25 0.39 }
                      }
                }
          ]
    },
```

```
    # Flames
          DEF Flames Shape {
        # No appearance, use emissive shading
              geometry IndexedFaceSet {
                    coord Coordinate {
                          point [
                                0.25 0.0 0.00,  0.15 1.0 0.10,
                                0.05 0.0 0.15,  0.18 0.0 0.05,
                                0.00 1.2 0.05, -0.10 0.0 0.05,
                               -0.00 0.0 0.15, -0.13 0.8 0.10,
                                     -0.25 0.0 0.00,
                                ]
                          }
                    color Color {
                          color [
                                1.0 0.0 0.0,  0.9 0.5 0.0,
                                1.0 0.0 0.0,  0.9 0.3 0.0,
                                1.0 1.0 0.0,  0.9 0.3 0.0,
                                0.7 0.1 0.2,  0.9 0.8 0.0,
                                      1.0 0.0 0.0,
                                ]
                          }
                    coordIndex [
                0, 1, 2, -1,  3, 4, 5, -1,  6, 7, 8, -1,
                          ]
                    }
              },
    # Additional Flames
              Transform {
                rotation 0.0 1.0 0.0 1.57
                scale    0.9 0.9 1.0
                children USE Flames
              },
              Transform {
                rotation 0.0 1.0 0.0 -1.57
                scale    0.9 0.9 1.0
                children USE Flames
              }
          ]
    }
```

Torcia a dettaglio medio

78 coordinate (-46%)

torch2.wrl

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Transform {
    translation 0.0 0.0 0.2
    scale 0.5 0.5 0.5
    children [
    # Torch handle
    Transform {
          translation 0.0 -0.75 0.0
          rotation 1.0 0.0 0.0 3.14
          children Shape {
                appearance DEF Gray Appearance {
                      material Material {
                            diffuseColor  0.4 0.4 0.4
                            specularColor 0.7 0.7 0.7
                      }
                }
                geometry Cone {
                      height 1.5
                      bottomRadius 0.15
                }
          }
    },
    # Fire pot
    DEF Ring Shape {
          appearance USE Gray
          geometry Cylinder {
                height 0.1
                radius 0.4
                top    FALSE
                bottom FALSE
          }
    },
    Transform { translation 0.0 0.2 0.0 children USE Ring },
    # Fire pot detail (eliminated)
    # Mounting bracket (eliminated)
    # Flames
    DEF Flames Shape {
          # No appearance, use emissive shading
          geometry IndexedFaceSet {
                coord Coordinate {
                      point [
                             0.25 0.0 0.00,  0.15 1.0 0.10,
                             0.05 0.0 0.15,  0.18 0.0 0.05,
                             0.00 1.2 0.05, -0.10 0.0 0.05,
                            -0.00 0.0 0.15, -0.13 0.8 0.10,
                            -0.25 0.0 0.00,
                      ]
                }
                color Color {
                      color [
                            1.0 0.0 0.0,  0.9 0.5 0.0,
                            1.0 0.0 0.0,  0.9 0.3 0.0,
                            1.0 1.0 0.0,  0.9 0.3 0.0,
                            0.7 0.1 0.2,  0.9 0.8 0.0,
                            1.0 0.0 0.0,
                      ]
                }
                coordIndex [
                      0, 1, 2, -1,  3, 4, 5, -1,  6, 7, 8, -1,
                ]
          }
    }
    # Additional Flames (eliminated)
    ]
}
```

Torcia a dettaglio basso

14 coordinate (-92%)

torch3.wrl

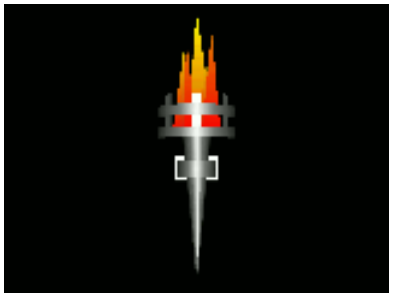
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Transform {
    translation 0.0 0.0 0.2
    scale 0.5 0.5 0.5
    children [
    # Torch handle (simplified)
    Shape {
        appearance DEF Gray Appearance {
            material Material { diffuseColor 0.4 0.4
        }
        geometry IndexedFaceSet {
            coord Coordinate {
                point [
                    -0.15 0.0 0.0,  0.0 -1.5 0.0,
                     0.15 0.0 0.0,
                ]
            }
            coordIndex [ 0, 1, 2 ]
        }
    },
    # Fire pot (simplified)
    DEF Ring Shape {
        appearance USE Gray
        geometry IndexedFaceSet {
            coord Coordinate {
                point [
                # First ring
                    -0.40 -0.05 0.1,  0.40 -0.05 0.1,
                     0.40  0.05 0.1, -0.40  0.05 0.1,
                # Second ring
                    -0.40  0.15 0.1,  0.40  0.15 0.1,
                     0.40  0.25 0.1, -0.40  0.25 0.1,
                ]
            }
            coordIndex [ 0, 1, 2, 3, -1,  4, 5, 6, 7, -1 ]
        }
    },
    # Fire pot detail (eliminated)
    # Mounting bracket (eliminated)
    # Flames (simplified)
    DEF Flames Shape {
        # No appearance, use emissive shading
        geometry IndexedFaceSet {
            coord Coordinate {
                point [
                    0.18 0.0 0.05,  0.00 1.2 0.05,
                    -0.18 0.0 0.05,
                ]
            }
            color Color {
                color [
                    1.0 0.0 0.0,  0.9 0.5 0.0,
                    1.0 0.0 0.0,
                ]
            }
            coordIndex [ 0, 1, 2 ]
        }
    }
    # Additional Flames (eliminated)
    ]
}
```

Cambio di forme

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L.
    Moreland
Group {
    children [
    # High-detail
    Transform {
         translation -0.5 0.0 0.0
         children Inline { url "torch1.wrl" }
    },
    # Medium-detail
    Inline { url "torch2.wrl" },
    # Low-detail
    Transform {
         translation 0.5 0.0 0.0
         children Inline { url "torch3.wrl" }
    }
    ]
}
```

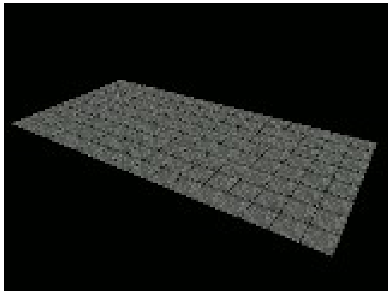
Cambio di forme (LOD)

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L.
    Moreland
LOD {
    center 0.0 0.0 0.0
    range [ 7.5, 12.0 ]
    level [
    # High-detail
     Inline { url "torch1.wrl" },
    # Medium-detail
     Inline { url "torch2.wrl" },
    # Low-detail
     Inline { url "torch3.wrl" }
    ]
}
```



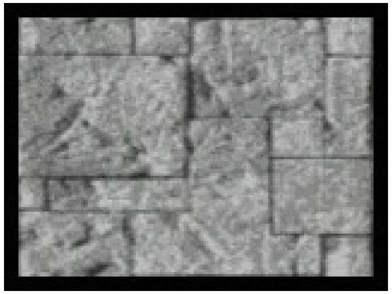
Costruzione di ambienti: pavimento

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L.
    Moreland
Transform {
    translation -5.0 0.0 -2.5
    children Shape {
     appearance Appearance {
          material Material { diffuseColor 1.0 1.0 1.0
    }
          texture ImageTexture { url "stone2.jpg" }
          textureTransform TextureTransform {
               scale 16.0 8.0
          }
     }
     geometry ElevationGrid {
          xDimension 8
          zDimension 8
          xSpacing   1.4285714
          zSpacing   0.7142857
          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,
          ]
     }
    }
}
```



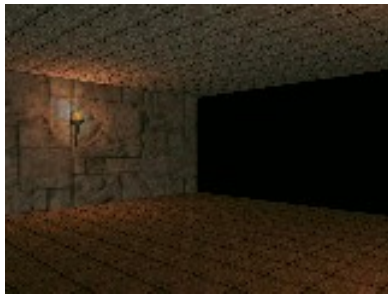
Costruzione di ambienti : muro

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L.
    Moreland
Transform {
    translation 0.0 3.5 -2.5
    rotation 0.0 0.0 1.0 -1.57
    children Shape {
     appearance Appearance {
          material Material { diffuseColor 1.0 1.0 1.0
     }
          texture ImageTexture { url "stonewal.jpg" }
     }
     geometry ElevationGrid {
          xDimension 8
          zDimension 8
          xSpacing   0.5
          zSpacing   0.7142857
          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,
          ]
     }
    }
}
```



Costruzione di ambienti : stanza

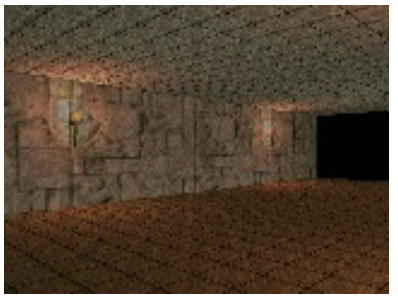
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Floor (two strips)
    Transform {
          translation 0.0 0.0 2.5
          children DEF Floor Inline { url "dfloor.wrl" }
    },
    Transform { translation 0.0 0.0 -2.5 children USE Floor },
    # Ceiling (reuse the floor)
    Transform { translation 0.0 3.5  2.5 children USE Floor },
    Transform { translation 0.0 3.5 -2.5 children USE Floor },
    # Left wall with torch
    Transform {
          translation -5.0 0.0 0.0
          children [
                Transform {
                      translation 0.0 0.0 2.5
                      children DEF Wall Inline { url
    "dwall.wrl" }
                },
                Transform { translation 0.0 0.0 -2.5 children USE
    Wall },
                Transform {
                      translation 0.0 2.25 0.0
                      rotation 0.0 1.0 0.0 1.57
                      children [
                            PointLight {
                                  location 0.0 0.25 0.2
                                  color 1.0 0.4 0.2
                                  intensity 0.8
                                  attenuation 0.0 0.6 0.0
                                  radius 10.0
                            },
                            DEF Torch Inline { url
    "torches.wrl" }
                      ]
                }
          ]
    },
    # Right wall with torch
    Transform {
          translation 5.0 0.0 0.0
          children [
                Transform { translation 0.0 0.0  2.5 children USE
    Wall },
                Transform { translation 0.0 0.0 -2.5 children USE
    Wall },
                Transform {
                      translation 0.0 2.25 0.0
                      rotation 0.0 1.0 0.0 -1.57
                      children [
                            PointLight {
                                  location 0.0 0.25 0.2
                                  color 1.0 0.4 0.2
                                  intensity 0.8
                                  attenuation 0.0 0.6 0.0
                                  radius 10.0
                            },
                            USE Torch
                      ]
                }
          ]
    }
    ]
}
```



droom.wrl

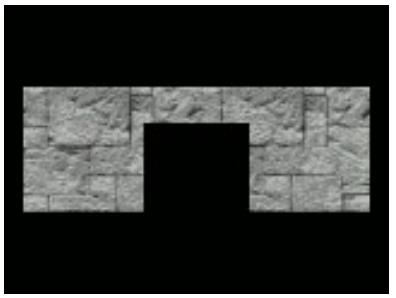
Costruzione di ambienti : 2 stanze controllate da LOD

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # First room
    LOD {
         range [ 20.0 ]
         level [
              Inline { url "droom.wrl" },
              Group { }
         ]
    },
    # Second room
    Transform {
         translation 0.0 0.0 -10.0
         children LOD {
              range [ 20.0 ]
              level [
                   Inline { url "droom.wrl" },
                   Group { }
              ]
         }
    }
    ]
}
```



Costruzione di ambienti : parete con porta

```
#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 1.0 }
    texture ImageTexture { url "stonewal.jpg" }
    }
    geometry IndexedFaceSet {
    coord Coordinate {
        point [
            -5.0 0.0 0.0, -1.5 0.0 0.0,
            -1.5 2.5 0.0,  1.5 2.5 0.0,
             1.5 0.0 0.0,  5.0 0.0 0.0,
             5.0 3.5 0.0, -5.0 3.5 0.0,
        ]
    }
    texCoord TextureCoordinate {
        point [
            0.0 0.0,  0.7 0.0,  0.7 0.7,  1.3 0.7,
            1.3 0.0,  2.0 0.0,  2.0 1.0,  0.0 1.0
        ]
    }
    coordIndex [ 0, 1, 2, 3, 4, 5, 6, 7 ]
    convex FALSE
    solid FALSE
    }
}
```



Costruzione di ambienti : porta

```
#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 1.0 }
    texture ImageTexture { url "panel.jpg" }
    }
    geometry IndexedFaceSet {
    coord Coordinate {
         point [
              -0.75 0.0 0.0,  0.75 0.0 0.0,
               0.75 2.5 0.0, -0.75 2.5 0.0,
         ]
    }
    texCoord TextureCoordinate {
         point [
              0.0 0.0,  1.0 0.0, 1.0 2.0,  0.0 2.0,
         ]
    }
    coordIndex [ 0, 1, 2, 3 ]
    solid FALSE
    }
}
```

Costruzione di ambienti : prigione sotterranea

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # First room
    LOD {
         range [ 20.0 ]
         level [
              Inline { url "droom.wrl" },
              Group { }
         ]
    },
    # Second room
    Transform {
         translation 0.0 0.0 -10.0
         children LOD {
              range [ 20.0 ]
              level [
                   Inline { url "droom.wrl" },
                   Group { }
              ]
         }
    },
    # Wall between first and second rooms
    Transform {
         translation 0.0 0.0 -5.0
         children Inline { url "dwall2.wrl" }
    },
    # Left and right door panels
    Transform {
         translation 0.0 0.0 -4.95
         children [
              DEF LeftDoor Transform {
                   children Transform {
                        translation -0.75 0.0 0.0
                        children DEF Door Inline { url "ddoor.wrl" }
                   }
              },
              DEF RightDoor Transform {
                   children Transform {
                        translation 0.75 0.0 0.0
                        children USE Door
                   }
              },
              DEF TouchDoor TouchSensor { }
         ]
    },
    # Animation clock
    DEF Clock TimeSensor {
         cycleInterval 5.0
    },
```

```
# Animation paths for the left and right doors
                    DEF LeftOpen PositionInterpolator {
                              key [ 0.0, 0.1, 0.9, 1.0 ]
                              keyValue [
                                        0.0 0.0 0.0, -1.3 0.0 0.0,
                                       -1.3 0.0 0.0,  0.0 0.0 0.0
                              ]
                    },
                    DEF RightOpen PositionInterpolator {
                              key [ 0.0, 0.1, 0.9, 1.0 ]
                              keyValue [
                                        0.0 0.0 0.0,  1.3 0.0 0.0,
                                        1.3 0.0 0.0,  0.0 0.0 0.0
                              ]
                    }
          ]
}
ROUTE TouchDoor.touchTime      TO Clock.set_startTime
ROUTE Clock.fraction_changed   TO LeftOpen.set_fraction
ROUTE Clock.fraction_changed   TO RightOpen.set_fraction
ROUTE LeftOpen.value_changed   TO LeftDoor.set_translation
ROUTE RightOpen.value_changed TO RightDoor.set_translation
```

Virtual Reality Modeling Language

VRML

Controllo del punto di vista

Controllo del punto di vista

- Mentre l'osservatore si muove nel mondo virtuale, il browser VRML ricostruisce la scena e la visualizza sullo schermo.
- Ogni volta che ci si sposta, cambia il punto di vista ed il browser usa una macchina fotografica virtuale che fotografa la scena e la riproduce a video.
- I nostri movimenti nel mondo virtuale continuamente orientano e posizionano questa macchina fotografica virtuale.
- Per agevolare la navigazione, è importante identificare dei punti di vista predefiniti, che consentano al navigatore di muoversi più efficacemente e scoprire al meglio le proprietà della scena.
- VRML fornisce due nodi per questo: **`Viewpoint`** e **`NavigationInfo`**.

Viewpoint

- Un punto di vista (viewpoint) è una posizione predef nita sia come posizione che come orientamento, analogamente ad un punto sosta belvedere dal quale il turista può effettuare belle foto.
- Quando un nodo **`Viewpoint`** è def nito in un nodo **`Trasform`**, il punto di vista appartiene al nuovo sistema di coordinate. Se si muove il sistema di coordinate, si muove anche il punto di vista.
- Usando le proprietà del browser VRML si può consentire allo stesso di saltare ai punti di vista selezionati nel nodo **`Viewpoint`**.

Informazioni sulla navigazione

- In realtà virtuale un **avatar** è una rappresentazione simbolica nel mondo virtuale di una persona nel mondo reale.
- Utilizzando un **avatar** una persona del mondo reale può interagire con il mondo virtuale, vedendo ciò che l'**avatar** vede ed interagendo con esso mediante una serie di istruzioni e movimenti che la persona compie attraverso l'**avatar**, istruendolo sul come comportarsi.
- In un ambiente di realtà virtuale multiutente, ognuno sceglie una forma 3-D come rappresentante in questo mondo. In funzione delle caratteristiche dell'ambiente, si può consentire al proprio **avatar** di camminare insieme ad altri **avatar** e di parlare con essi.

L'avatar

- Le informazioni relative all'avatar si dividono in due classi:
  - Quelle che descrivono l'aspetto che l'avatar deve avere
  - Quelle che descrivono come l'avatar si può muovere
- Molti browser hanno dei menu che consentono di modificare a runtime l'aspetto dell'avatar e le sue caratteristiche. Il posto fondamentale dove specificare le proprietà dell'avatar è comunque il nodo **`NavigationInfo`**.
- Il nodo **`NavigationInfo`** agisce di concerto con il nodo **`Viewpoint`**: il nodo **`Viewpoint`** definisce come vedere il mondo virtuale, mentre il nodo **`NavigationInfo`** descrive come muoversi rispetto ad esso.

Tipi di avatar

- La navigazione è il modo con cui ci si muove all'interno del mondo virtuale.
- In analogia con il mondo reale ci si può muovere nel mondo virtuale camminando o volando. Nel primo caso si segue la natura del terreno, nel secondo ci si muove liberamente e indipendentemente dalla natura del terreno stesso.
- Se il mondo è costituito da una prigione sotterranea, il tipo di navigazione più appropriata sarà **walk**, mentre nel caso si proceda a simulazioni di volo, il tipo di navigazione più appropriata sarà **fly**.
- Il tipo di navigazione viene selezionata attraverso il campo `type` del nodo `NavigationInfo`.

Tipi di avatar (i)

Valore di `type`	Descrizione
"WALK"	Abilita l'avatar ad esplorare il mondo virtuale come se stesse **camminando** in esso. Il visitatore ha la continua sensazione del sù e giù, allineati all'asse Y del sistema di coordinate correntemente definito. L'avatar viene spinto lungo l'asse Y per simulare la forza di gravità.
"FLY"	Abilita l'avatar a **volare** nel mondo virtuale, Fornisce al visitatore il senso del su e giù, allineati all'asse Y del sistema di coordinate correntemente definito. L'avatar non è costretto a seguire l'andamento del terreno e non c'è forza di gravità.
"EXAMINE"	Abilita il navigatore a vedere il mondo come una cosa esterna e lontana di fronte a se stesso. Il visitatore può girare il mondo ed effettuare lo zoom per analizzarlo più in dettaglio. L'avatar non è costretto a seguire l'andamento del terreno e non c'è effetto della forza di gravità
"NONE"	La navigazione è **disabilitata** e non c'è modo di navigare nel mondo. Il visitatore può solo sfruttare gli eventuali meccanismi di anmazione attivati dall'autore del mondo virtuale.

Velocità dell'avatar

- In base alle caratteristiche del mondo virtuale creato è bene stabilire anche la velocità più adatta per muoversi in esso.
- La velocità viene stabilita attraverso il campo **`speed`** del nodo **`NavigationInfo`**. Il valore specifica il numero di unità al secondo al quale l'avatar si deve muovere.
- Il valore definisce il valore iniziale con il quale l'avatar si muove nel mondo, anche se questo potrà poi essere modificato dal navigatore durante la visita mediante il ricorso al menu di configurazione del browser VRML.

Dimensioni dell'avatar

- Le dimensioni vengono definite nel campo **`avatarSize`** del nodo **`NavigationInfo`**.

Valore di `avatarSize`	Descrizione
`width`	Specifica fino a quale distanza l'avatar si può avvicinare ad una forma prima di collidere con essa. E' espresso come raggio. **Default**: `0.25`
`height`	Specifica a che altezza dal terreno deve essere tenuto il punto di vista, quando si segue l'andamento del terreno in modalità **`WALK`**. Coincide con l'altezza degli occhi dell'avatar dal terreno. **Default**: `1.60` :
`Step height`	Specifica l'altezza che l'avatar può saltare quando incontra un ostacolo o un'asperità del terreno, usando il tipo di navigazione **`WALK`**. Ostacoli più alti di questa misura potranno essere soltanto aggirati e non superati. Approssimativamente coincide con l'altezza massima a cui l'avatar può portare le gambe. **Default**: `0.75`

Caratteristiche dell'avatar

- Il browser VRML abilita per default una **headlight** che è posta sulla fronte dell'avatar.
- Pur essendo personalizzabile in molti browser da menu, è possibile da programma attivare o disattivare la **headlight** attraverso il campo `headlight` del nodo `NavigationInfo`.
- E' possibile determinare la profondità di veduta che l'avatar può raggiungere, mediante il campo `visibilityLimit` del nodo `NavigationInfo`.
- Si può usare il campo `visibilityLimit` per ridurre il numero di oggetti che il browser deve rappresentare nel mondo virtuale.
- Un valore di `visibilityLimit` uguale a `0.0` elimina questa funzionalità e consente la rappresentazione di tutte le forme presenti nel mondo a prescindere dalla loro distanza dall'osservatore.

Navigation binding

- Analogamente a quanto visto per gli sfondi, la nebbia ed i punti di vista, ci può essere un solo nodo `NavigationInfo` attivo in un certo istante nel mondo virtuale.
- Ci sono situazioni in chi si ha bisogno di diversi nodi `NavigationInfo`, per gestire situazioni differenti: si possono allora def nire diversi nodi e ricorrere al **navigation binding** per l'attivazione selettiva dei diversi nodi.
- Il browser, analogamente ai nodi `Background`, `Fog` e `Viewpoint` mantiene un **navigation stack**: il processo di **navigation binding** consiste nel porre il nodo `NavigationInfo` di interesse in cima al **navigation stack** per renderlo attivo.

Il nodo `Viewpoint`

- Il nodo **`Viewpoint`** specifica un punto di vista all'interno del sistema di coordinate attivo nel mondo virtuale

```
Viewpoint {
      position        0.0 0.0 0.0          #exposedField  SFVec3f
      orientation     0.0 0.0 1.0  0.0     #exposedField  SFRotation
      fieldOfView     0.785398             #exposedField  SFFloat
      description     " "                  #field         SFString
      Jump            TRUE                 #exposedField  SFBool
      set_bind                             #eventIn       SFBool
      isBound                              #eventOut      SFBool
      bindTime                             #eventOut      SFTime
         }
```

Il nodo `Viewpoint`

- L'exposed-field **`position`** specifica la coordinata 3D dove deve essere localizzato il punto di vista.
- L'exposed-field **`orientation`** specifica asse ed angolo di rotazione del punto di vista. Per default non avviene nessuna rotazione.
- L'exposed-field **`fieldOfView`** specifica l'ampiezza del campo visivo. Il valore di default è di 45 gradi.
- Il valore di **`fieldOfView`** viene ignorato se si montano dispositivi per la realtà virtuale immersiva, come l'**head mounted display** (HMD)
- Il campo **`description`** serve a specificare l'etichetta assegnata al punto di vista, per farne meglio comprender valore e significato al navigatore.

Il nodo `Viewpoint` (i)

- Il meccanismo del **viewpoint binding** viene attivato ponendo in cima al viewpoint stack il determinato nodo **`Viewpoint`** inviando all'eventIn **`set_bind`** il valore **`TRUE`**. In tal caso si ha:
  - Posizione e orientazione del visitatore vengono salvate nel nodo **`Viewpoint`** al momento in cima allo stack;
  - Il nodo **`Viewpoint`** attualmente in cima allo stack invia il valore **`FALSE`** utilizzando l'eventOut **`isBound`**.
  - Il nuovo nodo **`Viewpoint`** passa in cima allo stack e diventa il nodo attivo
  - Il nodo attivo invia il valore **`TRUE`** utilizzando l'eventOut **`is_bound`**, insieme al tempo assoluto nell'eventOut **`bindTime`**.
  - Se il valore del campo **`jump`** è **`TRUE`** allora posizione ed orientazione dell'osservatore saltano nei nuovi valori indicati nel nuovo nodo **`Viewpoint`**, altrimenti rimangono invariati, pur cambiando punto di vista.

Il nodo NavigationInfo

- Il nodo **NavigationInfo** specifica le informazioni per la navigazione nel mondo virtuale

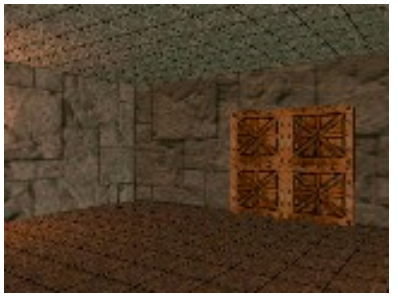
```
NavigationInfo {
      type              "WALK"              #exposedField   MFString
      speed              1.0                #exposedField   SFFloat
      avatarSize        [0.25, 1.6, 0.75]   #exposedField   MFFloat
      headlight         TRUE                #exposedField   SFBool
      visibilityLimit   0.0                 #eventIn        SFBool
      set_bind                              #eventIn        SFBool
      isBound                               #eventOut       SFBool
          }
```

Viewpoint

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    Viewpoint {
         description "Forward view"
         position 0.0 1.6 5.0
    },
    NavigationInfo {
         type "WALK"
         speed 1.0
         headlight FALSE
         avatarSize [ 0.5, 1.6, 0.5 ]
    },
    Inline { url "dungeon.wrl" }
    ]
}
```

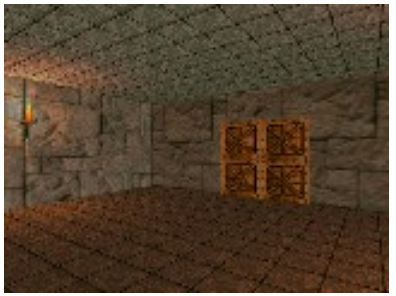
Orientazione punto di vista

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    Viewpoint {
         description "Corner view"
         position 3.0 1.6 3.0
         orientation 0.0 1.0 0.0 0.611
    },
    NavigationInfo {
         type "WALK"
         speed 1.0
         headlight FALSE
         avatarSize [ 0.5, 1.6, 0.5 ]
    },
    Inline { url "dungeon.wrl" }
    ]
}
```



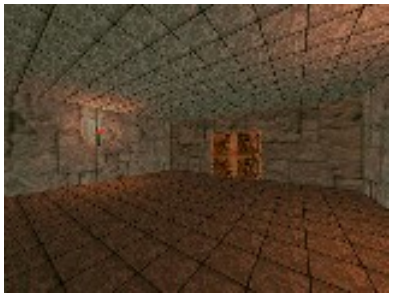
Controllo dell'apertura del campo visivo

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    Viewpoint {
         description "Corner view"
         position 3.0 1.6 3.0
         orientation 0.0 1.0 0.0 0.611
    },
    NavigationInfo {
         type "WALK"
         speed 1.0
         headlight FALSE
         avatarSize [ 0.5, 1.6, 0.5 ]
    },
    Inline { url "dungeon.wrl" }
    ]
}
```



Controllo dell'apertura del campo visivo (i)

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    Viewpoint {
         description "90.0 FOV degree corner view"
         position 3.0 1.6 3.0
         orientation 0.0 1.0 0.0 0.611
         fieldOfView 1.57
    },
    NavigationInfo {
         type "WALK"
         speed 1.0
         headlight FALSE
         avatarSize [ 0.5, 1.6, 0.5 ]
    },
    Inline { url "dungeon.wrl" }
    ]
}
```



Uso di più punti di vista

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Viewpoints
    Viewpoint {
         description "Forward view"
         position 0.0 1.6 5.0
    },
    Viewpoint {
         description "Corner view"
         position 3.0 1.6 3.0
         orientation 0.0 1.0 0.0 0.611
    },
    Viewpoint {
         description "60.0 FOV degree corner view"
         position 3.0 1.6 3.0
         orientation 0.0 1.0 0.0 0.611
         fieldOfView 1.047
    },
    Viewpoint {
         description "90.0 FOV degree corner view"
         position 3.0 1.6 3.0
         orientation 0.0 1.0 0.0 0.611
         fieldOfView 1.57
    },
    # Navigation
    NavigationInfo {
         type "WALK"
         speed 1.0
         headlight FALSE
         avatarSize [ 0.5, 1.6, 0.5 ]
    },
    # World
    Inline { url "dungeon.wrl" }
    ]
}
```

Virtual Reality Modeling Language

VRML

Intercettare l'avvicinamento dell'osservatore

Avvicinamento dell'osservatore

- Per una serie di potenzialità che si possono attivare nel nostro mondo virtuale risulta strategico poter intercettare l'avvicinamento dell'osservatore in certe parti del nostro mondo
- VRML, oltre ai già visti sensori, fornisce tre nodi sensori che sono fondamentali allo scopo: **`VisibilitySensor`**,che indica quando l'osservatore entra nella vista di una parte del mondo virtuale, **`ProximitySensor`** che indica quando l'osservatore entra in una regione specif ca del mondo virtuale e **`Collision`** che indica quando l'osservatore tocca una forma del mondo virtuale

Sensori Visibilità

- Questi sensori si attivano quando una certa scatola risulta visibile in base alla posizione e all'orientamento dell'osservatore.
- Si possono usare questi sensori per attivare o fermare animazioni o per controllare altre azioni nel momento in cui l'osservatore diventa visibile.
- Si indica centro e dimensioni della scatola bersaglio.
- Ogni volta che il bersaglio diventa visibile il nodo produce il tempo assoluto nell'eventOut **`enterTime`**.
- Analogamente ogni volta che il bersaglio esce di scena il nodo produce il tempo assoluto nell'eventOut **`exitTime`**.

Sensori di prossimità

- Questi sensori si attivano quando l'osservatore entra in una regione del mondo virtuale (una scatola definita da un centro e da una dimensione), analogamente ai sensori di visibilità.
- Si possono usare per emulare il comportamento di cellule fotoelettriche, in modo che quando l'osservatore si avvicina ad una porta scorrevole, questa si apra e si richiuda dopo che l'osservatore è uscito dalla scatola.
- Analogamente al sensore di visibilità produce i tempi assoluti di entrata e uscita dalla scatola negli eventOut **`enterTime`** e **`exitTime`**.
- Gli eventOut **`position_changed`** e **`orientation_changed`** invece riportano posizione e orientazione dell'osservatore all'uscita della scatola.

Sensori di collisione

- Questo sensore intercetta quando l'osservatore va a collidere con un gruppo di forme.
- E' un nodo di raggruppamento come il nodo **`Group`**, e come questo prevede un campo **`children`** con un bounding box center e un bounding box size.
- Quando il nodo intercetta una collisione produce il tempo assoluto nell'eventOut **`collideTime`**.
- A questo punto si può eseguire un suono, oppure attivare un video o attivare delle animazioni.
- Si può utilizzare il campo **`collide`** per disattivare il sensore su tutte le forme del gruppo.
- Questo sensore è estremamente oneroso per il browser. Può essere vantaggioso definire una forma in prossimità del sensore da specificare nel campo **`proxy`** del sensore **`Collision`**.

Il nodo `VisibilitySensor`

- Il nodo **`VisibilitySensor`** specifica un sensore che evidenzia quando l'osservatore diventa visibile in un'area del mondo virtuale.

```
VisibilitySensor {
      enabled              TRUE          #exposedField   SFBool
      center         0.0 0.0 0.0         #exposedField   SFVec3f
      size           0.0 0.0 0.0         #exposedField   SFVec3f
      isActive                           #eventOut       SFBool
      enterTime                          #eventOut       SFTime
      exitTime                           #eventOut       SFTime
         }
```

Il nodo `ProximitySensor`

- Il nodo **`ProximitySensor`** specifica un sensore che evidenzia quando l'osservatore si avvicina ad un'area del mondo virtuale.

```
ProximitySensor {
      enabled              TRUE          #exposedField  SFBool
      center         0.0 0.0 0.0         #exposedField  SFVec3f
      size           0.0 0.0 0.0         #exposedField  SFVec3f
      isActive                           #eventOut      SFBool
      enterTime                          #eventOut      SFTime
      exitTime                           #eventOut      SFTime
      position_changed                   #eventOut      SFVec3f
      orientation_changed                #eventOut      SFRotation
         }
```

Il nodo `Collision`

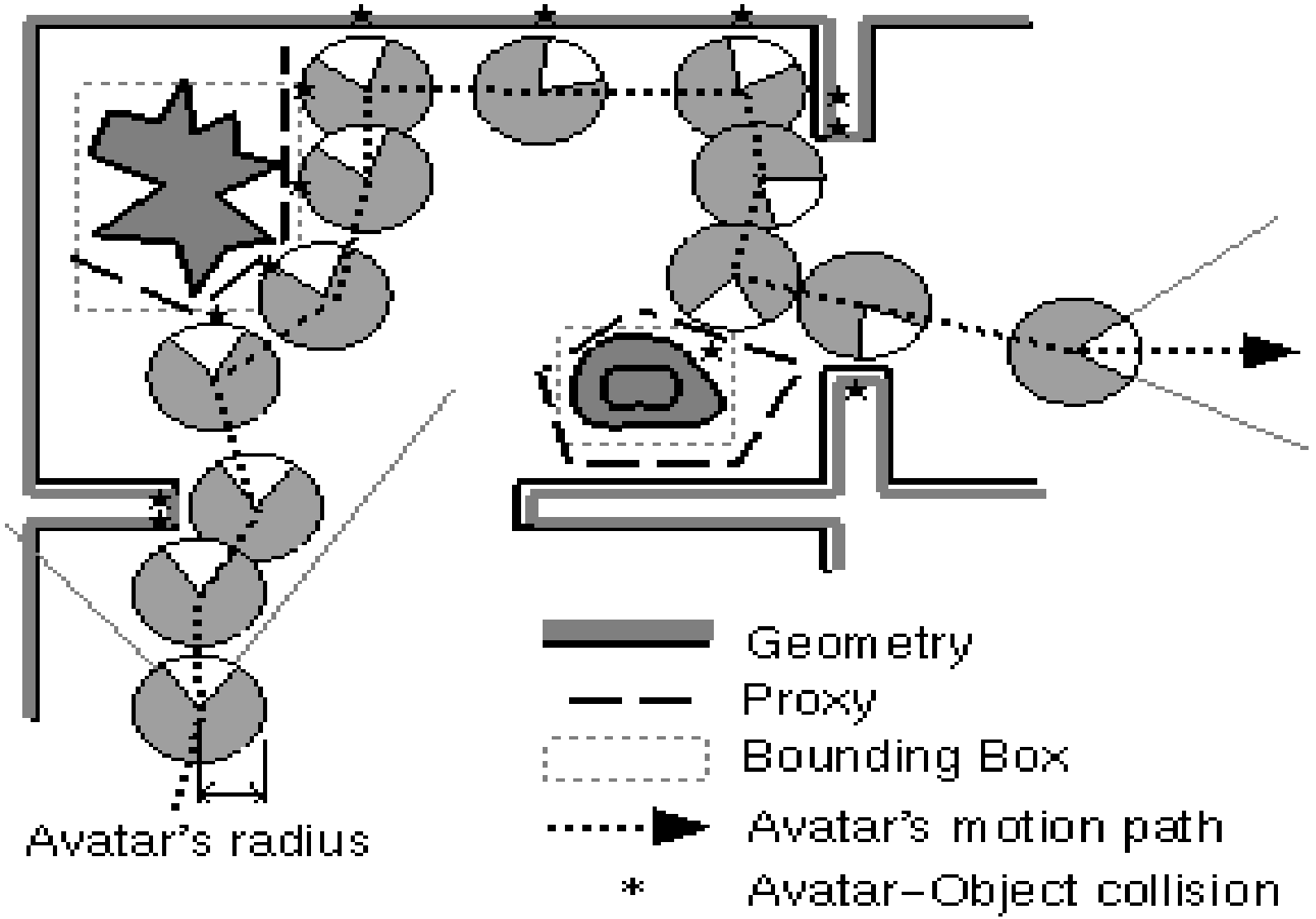
- Il nodo **`Collision`** specifica un sensore che evidenzia quando l'osservatore si avvicina ad un gruppo di forme.

```
Collision {
      children        [ ]                  #exposedField  MFNode
      bboxCenter      0.0 0.0 0.0          #field         SFVec3f
      bboxSize        -1.0 -1.0 -1.0       #field         SFVec3f
      collide         TRUE                 #exposedField  SFBool
      proxy           NULL                 #field         SFNode
      collideTime                          #eventOut      SFTime
      addChildren                          #eventIn       MFNode
      removeChildren                       #eventIn       MFNode
         }
```

Nodo Collision

- Il browser nel corso della navigazione intercetta le collisioni tra l'avatar e le forme geometriche della scena al fine di prevenire che l'avatar penetri nelle forme.
- I nodi `IndexedLineSet`, `PointSet` e `Text` non generano collisioni
- Il nodo *Collision* è un nodo di raggruppamento che può disabilitare attraverso il campo `collide` le collisioni per le forme che raggruppa.
- La definizione di un campo `proxy` può facilitare ed ottimizzare il processo di *collision detection*, usualmente molto costoso per i browser.

Nodo Collision



VisibilitySensor

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    NavigationInfo { headlight FALSE },
    # Room
    Inline { url "droom.wrl" },
    # Wall
    Transform {
        translation 0.0 0.0 -5.0
        children Inline { url "dwall2.wrl" }
    },
    # Left and right door panels
    Transform {
        translation 0.0 0.0 -4.95
        children [
            DEF LeftDoor Transform {
                children Transform {
                    translation -0.75 0.0 0.0
                    children DEF Door Inline { url "ddoor.wrl" }
                }
            },
            DEF RightDoor Transform {
                children Transform {
                    translation 0.75 0.0 0.0
                    children USE Door
                }
            },
        # Visibility Sensor
            DEF DoorSense VisibilitySensor {
                center 0.0 1.75 0.0
                size 3.0 2.5 1.0
            }
        ]
    },
    # Sounds
    Sound {
        source DEF OpenSound AudioClip { url "clunk1.wav" }
        minFront 20.0 minBack 20.0
        maxFront 60.0 maxBack 60.0
    },
    Sound {
        source DEF CloseSound AudioClip { url "clunk1.wav" }
        minFront 20.0 minBack 20.0
        maxFront 60.0 maxBack 60.0
    },
    # Animation clocks
    DEF OpenClock TimeSensor {
        cycleInterval 0.5
    },
    DEF CloseClock TimeSensor {
        cycleInterval 0.5
    },
    # Animation paths for the left and right doors
    DEF LeftOpen PositionInterpolator {
        key [ 0.0, 1.0 ]
        keyValue [ 0.0 0.0 0.0, -1.3 0.0 0.0 ]
    },
    DEF LeftClose PositionInterpolator {
        key [ 0.0, 1.0 ]
        keyValue [ -1.3 0.0 0.0, 0.0 0.0 0.0 ]
    },
    DEF RightOpen PositionInterpolator {
        key [ 0.0, 1.0 ]
        keyValue [ 0.0 0.0 0.0, 1.3 0.0 0.0 ]
    },
    DEF RightClose PositionInterpolator {
        key [ 0.0, 1.0 ]
        keyValue [ 1.3 0.0 0.0, 0.0 0.0 0.0 ]
    }
    ]
}
```

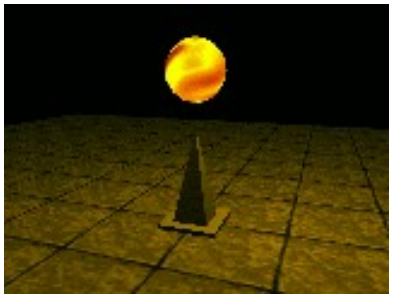
```
ROUTE DoorSense.enterTime          TO OpenSound.set_startTime
ROUTE DoorSense.exitTime           TO OpenSound.set_stopTime
ROUTE DoorSense.enterTime          TO OpenClock.set_startTime
ROUTE DoorSense.exitTime           TO OpenClock.set_stopTime

ROUTE DoorSense.exitTime           TO CloseSound.set_startTime
ROUTE DoorSense.enterTime          TO CloseSound.set_stopTime
ROUTE DoorSense.exitTime           TO CloseClock.set_startTime
ROUTE DoorSense.enterTime          TO CloseClock.set_stopTime

ROUTE OpenClock.fraction_changed   TO LeftOpen.set_fraction
ROUTE OpenClock.fraction_changed   TO RightOpen.set_fraction
ROUTE CloseClock.fraction_changed  TO LeftClose.set_fraction
ROUTE CloseClock.fraction_changed  TO RightClose.set_fraction

ROUTE LeftOpen.value_changed       TO LeftDoor.set_translation
ROUTE LeftClose.value_changed      TO LeftDoor.set_translation
ROUTE RightOpen.value_changed      TO RightDoor.set_translation
ROUTE RightClose.value_changed     TO RightDoor.set_translation
```

ProximitySensor

```
#VRML V2.0 utf8
#  The VRML 2.0 Sourcebook
#  Copyright [1997] By
#  Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    NavigationInfo {
          headlight FALSE
          avatarSize [ 1.0, 1.6, 0.75 ]
    }
    # Room
    Inline { url "droom.wrl" },
    # Wall
    Transform {
          translation 0.0 0.0 -5.0
          children Inline { url "dwall2.wrl" }
    }
    # Left and right door panels in a collision group
    DEF DoorCollide Collision {
          children Transform {
                translation 0.0 0.0 -4.95
                children [
                      DEF LeftDoor Transform {
                            children Transform {
                                  translation -0.75 0.0 0.0
                                  children DEF Door Inline { url "ddoor.wrl" }
                            }
                      },
                      DEF RightDoor Transform {
                            children Transform {
                                  translation 0.75 0.0 0.0
                                  children USE Door
                            }
                      },
                      DEF DoorSense ProximitySensor {
                            center 0.0 1.75 0.0
                            size 6.0 3.5 8.0
                      }
                ]
          }
    }
    # Sounds
    Sound {
          source DEF OpenSound AudioClip { url "clunk1.wav" }
          minFront 20.0 minBack 20.0
          maxFront 60.0 maxBack 60.0
    },
    Sound {
          source DEF CloseSound AudioClip { url "clunk1.wav" }
          minFront 20.0 minBack 20.0
          maxFront 60.0 maxBack 60.0
    }
    # Animation clocks
    DEF OpenClock TimeSensor {
          cycleInterval 0.5
    },
    DEF CloseClock TimeSensor {
          cycleInterval 0.5
    }
    # Animation paths for the left and right doors
    DEF LeftOpen PositionInterpolator {
          key [ 0.0, 1.0 ]
          keyValue [ 0.0 0.0 0.0, -1.3 0.0 0.0 ]
    },
    DEF LeftClose PositionInterpolator {
          key [ 0.0, 1.0 ]
          keyValue [ -1.3 0.0 0.0, 0.0 0.0 0.0 ]
    },
    DEF RightOpen PositionInterpolator {
          key [ 0.0, 1.0 ]
          keyValue [ 0.0 0.0 0.0, 1.3 0.0 0.0 ]
    },
    DEF RightClose PositionInterpolator {
          key [ 0.0, 1.0 ]
          keyValue [ 1.3 0.0 0.0, 0.0 0.0 0.0 ]
    }
    ]
}
```

```
ROUTE DoorCollide.collideTime      TO OpenSound.set_startTime
ROUTE DoorSense.exitTime           TO OpenSound.set_stopTime
ROUTE DoorCollide.collideTime      TO OpenClock.set_startTime
ROUTE DoorSense.exitTime           TO OpenClock.set_stopTime

ROUTE DoorSense.exitTime           TO CloseSound.set_startTime
ROUTE DoorCollide.collideTime      TO CloseSound.set_stopTime
ROUTE DoorSense.exitTime           TO CloseClock.set_startTime
ROUTE DoorCollide.collideTime      TO CloseClock.set_stopTime

ROUTE OpenClock.fraction_changed   TO LeftOpen.set_fraction
ROUTE OpenClock.fraction_changed   TO RightOpen.set_fraction
ROUTE CloseClock.fraction_changed  TO LeftClose.set_fraction
ROUTE CloseClock.fraction_changed  TO RightClose.set_fraction

ROUTE LeftOpen.value_changed       TO LeftDoor.set_translation
ROUTE LeftClose.value_changed      TO LeftDoor.set_translation
ROUTE RightOpen.value_changed      TO RightDoor.set_translation
ROUTE RightClose.value_changed     TO RightDoor.set_translation
```

Collision (proxy)



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    NavigationInfo {
        headlight FALSE
        avatarSize [ 1.0, 1.0, 0.75 ]
    },
    # Floor (two strips)
    Transform {
        translation 0.0 0.0 2.5
        children DEF Floor Inline { url "dfloor.wrl" }
    },
    Transform { translation 0.0 0.0 -2.5 children USE Floor },
    # Collision group
    DEF OrbCollide Collision {
        proxy Transform {
            translation 0.0 0.6 0.0
            children Shape {
                geometry Box { size 0.4 1.2 0.4 }
            }
        }
        children [
        # Glowing orb with sound effects
            DEF OrbSpin Transform {
                translation 0.0 1.0 0.0
                children [
                # Orb light
                    PointLight {
                        location 0.3 0.0 0.0
                        radius 10.0
                        ambientIntensity 0.2
                        color 0.7 0.5 0.0
                    },
                # Orb itself
                    Shape {
                        appearance Appearance {
                            # No material, use emissive texturing
                           texture ImageTexture { url "fire.jpg" }
                        }
                        geometry Sphere { radius 0.2 }
                    },
                # Orb sounds
                    Sound {
                        source AudioClip {
                            url "drone1.wav"
                            loop TRUE
                        }
                        intensity 0.5
                    },
                    DEF WispyAmp Sound {
                        source DEF Wispy AudioClip {
                            url "willow1.wav"
                            stopTime 1.0
                        }
                        intensity 0.0
                    }
                ]
            },
        # Pedestal pyramid
            Shape {
                appearance DEF PedestalColor 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 PedestalColor
                    geometry Box { size 0.4 0.03 0.4 }
                }
            },
        ]
    }
```

```
    # Animation clock
                    DEF Clock TimeSensor {
                        cycleInterval 28.0
                    },
    # Orb animation and volume control
                    DEF OrbSpinner OrientationInterpolator {
                        key [ 0.0, 0.5, 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
                        ]
                    },
                    DEF WispyVolume ScalarInterpolator {
                        key      [ 0.0, 0.1, 0.9, 1.0 ]
                        keyValue [ 0.0, 0.6, 0.6, 0.0 ]
                    }
    ]
}
ROUTE OrbCollide.collideTime    TO Clock.set_startTime
ROUTE OrbCollide.collideTime    TO Wispy.set_startTime
ROUTE Clock.fraction_changed    TO OrbSpinner.set_fraction
ROUTE Clock.fraction_changed    TO WispyVolume.set_fraction
ROUTE OrbSpinner.value_changed  TO OrbSpin.set_rotation
ROUTE WispyVolume.value_changed TO WispyAmp.set_intensity
```

Virtual Reality Modeling Language

VRML

Aggiunta di *Anchor*

Anchors

- Essendo una tecnologia Web, VRML consente di effettuare link tra mondi virtuali.
- Usando un link si può collegare ad es. una porta ad un mondo di destinazione, memorizzato in Internet.
- In questo modo si possono creare dei mondi molto complessi in modo modulare.
- I link a mondi VRML sono collegato o ancorati a delle forme specif che nel nostro mondo virtuale.
- Questa tecnica usa il nodo **`Anchor`** ed una URL che indica l'indirizzo Internet di un mondo virtuale.
- Il f le specif cato nella URL contiene il mondo nel quale si viene a trovare l'utente che clicca sulla forma associata al nodo **`Anchor`**.

Il nodo `Anchor`

- Il nodo **`Anchor`** raggruppa un insieme di forme, quando l'utente clicca su queste forme, viene letto un altro mondo:

```
Anchor {
      children          [ ]                  #exposedField    MFNode
      bboxCenter        0.0 0.0 0.0          #field           SFVec3f
      bboxSize          -1.0 -1.0 -1.0       #field           SFVec3f
      url               [ ]                  #exposedField    MFBool
      parameter         [ ]                  #exposedField    MFNode
      description       ""                   #exposedField    SFString
      addChildren                            #eventIn         MFNode
      removeChildren                         #eventIn         MFNode
          }
```

Il nodo Anchor

- Il valore dell'exposed-field **description** è un breve testo che descrive le caratteristiche del mondo virtuale di destinazione. In genere viene mostrato quando l'utente passa sopra alla forma senza cliccare.
- L'exposed-field **parameter** contiene eventuali parametri addizionali usati dal browser per accedere al meglio il mondo di destinazione.
- La lista è composta da una serie di parole chiave alle quali viene associato un valore attraverso il carattere **=**.
- Ogni URL nel campo può contenere la specifica di un nodo opzionale, aggiungendo alla URL il segno # ed il nome di un nodo **Viewpoint**. Ad es:

  www.da.qualche.parte.org#CoolView
- Dove **CoolView** è il nome assegnato ad un nodo **Viewpoint** nel mondo di destinazione mediante **DEF**.

Il nodo `Anchor`

- Una URL nel campo **`url`** può contenere un punto di vista e un indirizzo web vuoto:

```
#CoolView
```

- Un indirizzo web vuoto indica il file stesso: quando il visitatore clicca sulla forma viene spostata la posizione e l'orientazione dell'osservatore nel nodo **`Viewpoint`** collegato.
- In questo modo si possono creare dei menu che guidano la visitazione del mondo virtuale.
- Un altro uso è quello che codifica il campo mentre il campo viene lasciato vuoto: in questo modo quando il visitatore si avvicina alla forma appare un pannello con la descrizione: utile per *annotare* il nostro mondo virtuale.

Mondo di partenza

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
   children [
   # Viewpoints
   DEF Forward Viewpoint {
        description "Forward view"
        position 0.0 1.6 5.0
   },
   DEF Corner Viewpoint {
        description "Corner view"
        position 3.0 1.6 3.0
        orientation 0.0 1.0 0.0 0.611
   },
   # Navigation
   NavigationInfo { headlight FALSE },
   # World
   Inline { url "dungeon.wrl" }
   ]
}
```

Anchor (I)

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    NavigationInfo { headlight FALSE },
    # Room
    Inline { url "droom.wrl" },
    # Dungeon wall
    Transform {
         translation 0.0 0.0 -5.0
         children DEF Wall Inline { url "dwall2.wrl" },
    },
    # Anchor doors
    Anchor {
         url "dngnwrld.wrl"
         description "The Dungeon"
         children [
         # Left door panel
              Transform {
                   translation -0.75 0.0 -4.95
                   children DEF LeftDoor Transform {
                        children DEF Door Inline { url "ddoor.wrl" }
                   }
              },
         # Right door panel
              Transform {
                   translation 0.75 0.0 -4.95
                   children DEF RightDoor Transform {
                        children USE Door
                   }
              }
         ]
    }
    ]
}
```

Virtual Reality Modeling Language

VRML

linformazioni relative al mondo

Informazioni sul mondo virtuale

- Una volta realizzato il mondo virtuale in linguaggio VRML, ad esso viene assegnato un titolo e va firmato:
  - Inserendo nei sorgenti VRML una intestazione con i dati principali del progetto
  - codificando il nodo **`WorldInfo`**
- Al contrario dei commenti, le informazioni del nodo **`WorldInfo`** possono essere estratte dal browser e visualizzate al visitatore del mondo virtuale
- Ciò consente all'autore di firmare il proprio progetto ed al visitatore di conoscere il tema e l'autore del mondo che sta visitando

Informazioni sul mondo virtuale

- Il nodo **`WorldInfo`** consente di dichiarare il titolo del mondo virtuale ed ogni addizionale informazione che viene ritenuta necessaria:
  - Nome e cognome dell'autore
  - Notizie di Copyright
  - Data in cui il lavoro è stato terminato
  - Storia delle revisioni
  - Alcune parole che descrivano il mondo e quali scene/oggetti sono particolarmente interessanti e significativi in esso.
- Il nodo **`WorldInfo`** non crea forme nel mondo virtuale: le informazioni vengono visualizzate dal browser in una apposita voce di menu. Quando l'utente selezione questa voce, una finestra speciale illustra il contenuto del nodo. Alcuni browser inseriscono in titolo del progetto nel titolo della pagina web.
- Si possono inserire quanti nodi **`WorldInfo`** si vuole nel mondo, ma verrà elaborato soltanto il primo incontrato.

Il nodo `WorldInfo`

- Il nodo **`WorldInfo`** fornisce informazioni sul mondo virtuale:

```
WorldInfo {
      title          " "          #field          SFString
      info           [ ]          #field          MFString
          }
```

- Il valore del campo **`title`** è una stringa di testo che descrive il titolo del mondo virtuale. E' bene che sia suff cientemente corto da entrare nella *title bar* del browser. Il valore di default è una stringa vuota.
- Il valore del campo **`info`** è una serie di stringhe che contengono i commenti al mondo virtale. Il valore di default è una stringa vuota.

WorldInfo

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    WorldInfo {
         title "The Dungeon"
         info  [
              "(c) Copyright 1996, Ames, Nadeau, and
    Moreland",
              "Created for The VRML 2.0 Sourcebook",
              "Published by John Wiley & Sons, 1996."
         ]
    },
    NavigationInfo { headlight FALSE },
    Inline { url "dungeon.wrl" }
    ]
}
```

Virtual Reality Modeling Language

VRML

Definizione di nodi utente

Creazione di nuovi nodi

- VRML fornisce oltre 50 tipi di nodi diversi per creare forme, selezionare colori, def nire animazioni, etc.
- A questo insieme di nodi si possono aggiungere nuovi tipi di nodo def niti dall'utente.
- Ciascun nuovo tipo di nodo creato ha:
  - un nome
  - una lista di campi, `eventIn` ed `eventOut` (**interfaccia del nodo**)
  - un corpo
- Il nome del tipo di nodo può essere un nome qualsiasi, esclusi i nomi riservati. Se il nodo descrive una montagna frattale, si potrà chiamare ad es.: **`FractalMountain`**.

Creazione di nuovi tipi di nodo (i)

- L'**interfaccia del nodo**, costituita dai campi, dagli *exposed-field*, dagli *eventIn* e dagli *eventOut*, descrive le grandezze proprie del nuovo tipo di nodo e quelle che lo interfacciano al mondo VRML restante. Il tipo di nodo **`FractalMountain`** potrà avere dei campi che descrivono l'altezza della montagna, la profondità e la lunghezza, il colore, i dettagli da inserire nella forma della montagna, etc.
- Il **corpo del nodo** specifica che cosa fa e come lo fa.
- Questo viene definito utilizzando qualsiasi combinazione di tipi di nodo predefiniti in VRML. Nel nostro esempio si potrà utilizzare un nodo **`ElevationGrid`** per descrivere la forma della montagna, insieme ad un nodo **`Script`** che calcoli le altezze relative del nodo **`ElevationGrid`**.

La definizione PROTO

- I nuovi tipi di nodo vengono definiti utilizzando la definizione **PROTO**.
- Le proprietà della definizione **PROTO** consentono di definire il nome del nuovo tipo di nodo, i campi, gli *exposed-field*, gli *eventIn* e *eventOut*, ed il corpo del nuovo tipo di nodo.
- Si possono inserire le definizioni **PROTO** in un file separato, creando una sorta di libreria di definizioni contenenti i propri nuovi tipi di nodo VRML.
- Per utilizzare uno dei nuovi tipi di nodo definiti nella libreria si usa la dichiarazione **EXTERNPROTO**.

Struttura del nodo PROTO

proto ::=

PROTO *nodeTypeId* **[** *interfaceDeclarations* **]** **{** *protoBody* **}** ;

protoBody ::=

protoStatements node statements ;

interfaceDeclarations ::=

interfaceDeclaration |

interfaceDeclaration interfaceDeclaration … |

empty ;

restrictedInterfaceDeclaration ::=

eventIn *fieldType eventInId* |

eventOut *fieldType eventOutId* |

field *fieldType fieldId fieldValue* ;

interfaceDeclaration ::=

restrictedInterfaceDeclaration |

exposedField *fieldType fieldId fieldValue* ;

Struttura del nodo EXTERNPROTO

EXTERNPROTO *nodeTypeId* **[** *externInterfaceDeclarations* **]** *URLList* ;

externInterfaceDeclarations ::=

externInterfaceDeclaration |

externInterfaceDeclaration externInterfaceDeclarations |

empty ;

externInterfaceDeclaration ::=

eventIn *fieldType eventInId* |

eventOut *fieldType eventOutId* |

field *fieldType fieldId* |

exposedField *fieldType fieldId* ;

Creazione di prototipi

- Una definizione **PROTO** descrive un nuovo tipo di nodo:

  PROTO *NodeTypeName* [*nodeInterface*] { *nodeBody* }

- La definizione inizia con la parola chiave **PROTO**, seguita dal nome del tipo di nodo.
- Dopo il nome viene specificata tra parentesi quadre la lista dei campi, degli exposed-field, degli eventIn e eventOut che costituiscono l'interfaccia del nuovo tipo di nodo.
- Successivamente viene specificato il corpo del nodo, tra parentesi graffe.

Definizione del nome di prototipi

- Il nome del tipo di nodo segue le regole viste per i nomi da usare in **`DEF`**:
  - I nomi dei nuovi tipi di nodi devono essere differenti da qualsiasi nome di tipo di nodo esistente
  - Devono iniziare con una lettera e possono contenere numeri e il carattere _
  - Non si possono usare i nomi predefiniti in VRML
  - Ogni parola nel nome del tipo di nodo, compresa la prima, deve iniziare con la lettera maiuscola

Definizione dell'interfaccia di prototipi

- Def nizione di un campo del nodo:

```
PROTO MyNode [
        …
    field     fieldType    fieldName    defaultValue
        …
    ]
    {
        …
    }
```

- Il nome del tipo del campo (***fieldType***)indica il tipo di informazione associata al campo e deve essere uno dei tipi previsti in VRML.
- Il nome del campo (***fieldName***) può essere un nome qualsiasi compatibile con la sintassi VRML, con la convenzione che i nomi di campi iniziano con la lettera minuscola e le parole nel nome successive alla prima iniziano con la lettera maiuscola.

Definizione dell'interfaccia di prototipi (i)

- Definizione di un exposed-field del nodo:

```
PROTO MyNode [
    …
  exposedField     fieldType   fieldName   defaultValue
    …
  ]
  {
    …
  }
```

- Il nome del tipo del campo (***fieldType***) indica il tipo di informazione associata al campo e deve essere uno dei tipi previsti in VRML.
- Il nome del campo (***fieldName***) può essere un nome qualsiasi compatibile con la sintassi VRML, con la convenzione che i nomi di campi iniziano con la lettera minuscola e le parole nel nome successive alla prima iniziano con la lettera maiuscola.

Definizione dell'interfaccia di prototipi (ii)

- Definizione di un eventIn del nodo:

```
PROTO MyNode [
    …
  eventIn      eventInType eventInName
    …
  ]
  {
    …
  }
```

- Il nome del tipo dell'eventIn (***eventInType***) indica il tipo di informazione associata all'eventIn e deve essere uno dei tipi previsti in VRML.
- Il nome dell'eventIn (***eventInName***) può essere un nome qualsiasi compatibile con la sintassi VRML, con la convenzione che i nomi degli eventIn iniziano con il prefisso **set_**, a meno che non abbiano a che fare con l'aggiunta o rimozione di children da un gruppo.

Definizione dell'interfaccia di prototipi (iii)

- Definizione di un eventOut del nodo:

```
PROTO MyNode [
    …
  eventOut    eventOutType   eventOutName
    …
]
{
    …
}
```

- Il nome del tipo dell'eventOut (***eventOutType***) indica il tipo di informazione associata all'eventOut e deve essere uno dei tipi previsti in VRML.
- Il nome dell'eventOut (***eventOutName***) può essere un nome qualsiasi compatibile con la sintassi VRML, con la convenzione che i nomi degli eventOut finiscono con il suffisso **_changed**, a meno che non indichino un valore booleano (nel qual caso inizierà con **is**) o un valore di tempo (nel qual caso terminerà in **Time**).

Definizione del corpo di prototipi

- Una volta definita l'interfaccia del nodo, si può definirne il comportamento attraverso il corpo del nodo, che specifica che cosa il nuovo tipo di nodo fa e come lo fa.
- Il corpo del nodo è una lista di nodi VRML, inclusi quelli di gruppo, sensori, interpolatori o nodi **`Script`**. Si può definire un numero qualsiasi di nodi e di qualsiasi tipo. Il primo nodo definito nel prototipo definisce le regole di utilizzabilità del nuovo nodo. Le definizioni **`DEF`** definite fuori dal nodo non sono utilizzabili dentro al nodo, quelle definite dentro al nodo non sono utilizzabili fuori dal nodo stesso.
- La possibilità di includere in un prototipo qualsiasi tipo di nodo predefinito e qualsiasi **`ROUTE`**, è uno dei maggiori punti di forza di VRML.
- La tecnica vista consente di definire dei prototipi per le strutture più ricorrenti, aggiornandone le proprietà attraverso l'interfaccia ogni volta che viene usato il prototipo.

Accesso all'interfaccia dal corpo del prototipo

- Il corpo del nodo può realizzare una connessione tra un elemento dell'interfaccia e un elemento appropriato del corpo del nodo.
- La connessione è specif cata utilizzando la sintassi **IS**:

```
   AnyNode {
      anyField IS anyInterfaceItem
            }
PROTO Cube [
      field SFVec3f cubeSize 0.0 0.0 0.0
         ] {
            Shape {
               appearance Appearance {
                  material Material { }
               }
               geometry Box {
                  size IS cubeSize
                  }
            }
      }
```

Accesso all'interfaccia dal corpo del prototipo (i)

- Quando si usa la connesione `IS`, il tipo di dato dell'elemento dell'interfaccia deve essere lo stesso di quello dell'elemento del corpo del nodo.
- Ogni tipo di dato deve essere associato con lo stesso tipo di dato (campo-campo, eventIn-eventIn, eventOut-eventOut) ad eccezione dell' exposed-field, che può essere collegato a qualsiasi tipo di dato dell'interfaccia (sia campo, che exposed-field, che eventIn o eventOut)

Prototipi esterni

- Una definizione **PROTO** definisce un nuovo tipo di nodo che può essere usato in qualsiasi punto dello stesso file.
- In certi casi può risultare vantaggioso raggruppare i prototipi definiti in un file esterno in modo da usarlo come libreria di nodi.
- In questo caso la definizione **PROTO** può essere acceduta da qualsiasi posizione del file contenente un dato mondo virtuale, usando la dichiarazione **EXTERNPROTO**.

Prototipi esterni (i)

- La definizione **EXTERNPROTO** è una versione ridotta della definizione **PROTO**: specifica il nome del nuovo tipo di nodo, la dichiarazione dell'interfaccia senza specificare i valori di default e invece del corpo del nodo specifica una URL relativa al sorgente del file contenente la definizione **PROTO**.
- Le dichiarazioni di interfaccia specificate nella dichiarazione **PROTO** devono corrispondere ad un sottoinsieme di quelle definite nella dichiarazione **EXTERNPROTO**.

```
EXTERNPROTO NodeTypeName [ NodeInterface ]  [ urls ]
```

La definizione PROTO

- La definizione **PROTO** consente di definire dei nuovi tipi di nodo, secondo le esigenze dell'utente:

```
PROTO NodeTypeName [

# qualsiasi numero di:

   field           fieldType      fieldName      defaultValue

   exposedField    exposedFieldType  exposedFieldName
   defaultValue

   eventIn         eventInType    eventInName

   eventOut          eventOutType      eventOutName

   ] {

         nodeBody

         }
```

La dichiarazione EXTERNPROTO

- La dichiarazione **EXTERNPROTO** crea un nuovo tipo di nodo che specifica una definizione **PROTO** contenuta in un altro file:

```
EXTERNPROTO NodeTypeName [

# qualsiasi numero di:

   field           fieldType            fieldName
   defaultValue

   exposedFfield exposedFieldType     exposedfieldName    defaultValue

   eventIn         eventInType        eventInName

   eventOut        eventOutType       eventOutName

       ]

       url oppure [ urlList ]
```

La connessione IS

- La dichiarazione **IS** crea una connessione tra un campo, un exposed-field, un eventIn o un eventOut dell'interfaccia di un prototipo e un campo, un exposed-field, un eventIn o un eventOut del corpo dello stesso prototipo:

```
fieldName   IS    interfaceItem
```

Prototipi: WallColor

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
PROTO WallColor [ ] {
   Material {
    diffuseColor 0.0 0.6 1.0
    }
}

Shape {
   appearance Appearance {
    material WallColor { }
    }
   geometry Box { size 10.0 2.0 0.1 }
}
```

Prototipi: matlib.wrl

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
PROTO WallColor [ ] {
   Material {
    diffuseColor 0.0 0.6 1.0
    }
}
PROTO Gold [ ] {
   Material {
    ambientIntensity 0.4
    diffuseColor 0.22 0.15 0.0
    specularColor 0.71 0.70 0.56
    shininess 0.16
    }
}
PROTO Aluminum [ ] {
   Material {
    ambientIntensity 0.3
    diffuseColor 0.30 0.30 0.50
    specularColor 0.70 0.70 0.80
    shininess 0.10
    }
}
PROTO Copper [ ] {
   Material {
    ambientIntensity 0.26
    diffuseColor 0.30 0.11 0.00
    specularColor 0.75 0.33 0.00
    shininess 0.08
    }
}
```

Prototipi: uso di matlib.wrl

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland


EXTERNPROTO WallColor [ ] "matlib.wrl#WallColor"

Shape {
   appearance Appearance {
   material WallColor { }
   }
   geometry Box { size 10.0 2.0 0.1 }
}
```

Prototipi: applib.wrl

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
PROTO DarkOak [
   exposedField SFNode textureTransform NULL
] {
   Appearance {
   material Material { diffuseColor 1.0 0.45 0.23 }
   texture ImageTexture { url "wood_g.jpg" }
   textureTransform IS textureTransform
   }
}
PROTO LightOak [
   exposedField SFNode textureTransform NULL
] {
   Appearance {
   material Material { diffuseColor 1.0 0.65 0.53 }
   texture ImageTexture { url "wood_g.jpg" }
   textureTransform IS textureTransform
   }
}
```

Prototipi: uso di applib.wrl

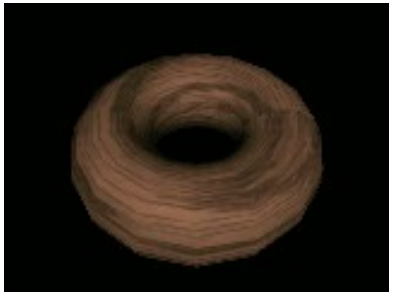
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
EXTERNPROTO DarkOak [
   exposedField SFNode textureTransform
] "applib.wrl#DarkOak"

EXTERNPROTO LightOak [
   exposedField SFNode textureTransform
] "applib.wrl#LightOak"

Group {
   children [
   Transform { translation -1.5 0.0 0.0
        children Shape {
             appearance DarkOak { }
             geometry Box { }
        }
   },
   Transform { translation 1.5 0.0 0.0
        children Shape {
             appearance LightOak {
                  textureTransform TextureTransform {
                       translation 0.5 0.0
                       scale 2.0 1.0
                  }
             }
             geometry Box { }
        }
   }
   ]
}
```

Creazione di prototipi di forme geometriche: donutpro.wrl

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
PROTO Donut [
   field    SFFloat crossSectionRadius     1.0
   field    SFFloat spineRadius            2.0
   field    SFInt32 crossSectionResolution 16
   field    SFInt32 spineResolution        16
   eventIn  SFFloat set_crossSectionRadius
   eventIn  SFFloat set_spineRadius
   eventOut MFVec2f crossSection_changed
   eventOut MFVec3f spine_changed
] {
   DEF Ext Extrusion {
    spine         [ ]
    crossSection  [ ]
    creaseAngle   1.57
    beginCap      FALSE
    endCap        FALSE
    }
   DEF DonutMaker Script {
    url "donutmkr.js"
    field    SFFloat crossSectionRadius     IS crossSectionRadius
    field    SFFloat spineRadius            IS spineRadius
    field    SFInt32 crossSectionResolution IS crossSectionResolution
    field    SFInt32 spineResolution        IS spineResolution
    eventIn  SFFloat set_crossSectionRadius IS set_crossSectionRadius
    eventIn  SFFloat set_spineRadius        IS set_spineRadius
    eventOut MFVec2f crossSection_changed   IS crossSection_changed
    eventOut MFVec3f spine_changed          IS spine_changed
    }
   ROUTE DonutMaker.crossSection_changed TO Ext.set_crossSection
   ROUTE DonutMaker.spine_changed        TO Ext.set_spine
}
```

Uso di prototipi di forme geometriche

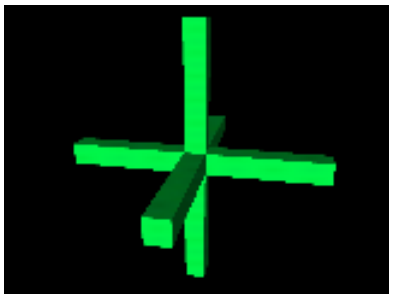
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
EXTERNPROTO Donut [
   field    SFFloat crossSectionRadius
   field    SFFloat spineRadius
   field    SFInt32 crossSectionResolution
   field    SFInt32 spineResolution
   eventIn  SFFloat set_crossSectionRadius
   eventIn  SFFloat set_spineRadius
   eventOut MFVec2f crossSection_changed
   eventOut MFVec3f spine_changed
] "donutpro.wrl#Donut"

EXTERNPROTO LightOak [
   exposedField SFNode textureTransform
] "applib.wrl#LightOak"

Shape {
   appearance LightOak {
    textureTransform TextureTransform {
        translation 0.5 0.0
    }
    }
   geometry Donut {
    crossSectionRadius 1.0
    spineRadius        2.0
    }
}
```



Costruzione di una script per un f ltro booleano:BooleanFilter.wrl

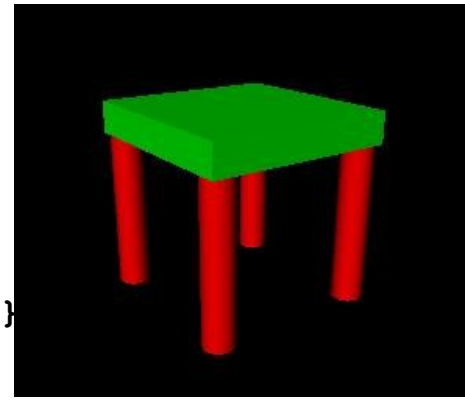
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
PROTO BooleanFilter [
   eventIn  SFBool set_boolean
   eventOut SFBool true_changed
   eventOut SFBool false_changed
] {
   Script {
    eventIn  SFBool set_boolean   IS set_boolean
    eventOut SFBool true_changed  IS true_changed
    eventOut SFBool false_changed IS false_changed
    url "javascript:
        function set_boolean( bool, eventTime ) {
            if ( bool == true ) { true_changed  = true; }
            else                { false_changed = true; }
        }"
   }
}
```

Costruzione di un nodo di Rotazione:SpinGroup.wrl



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
PROTO SpinGroup [
    exposedField MFNode  children      [ ]
    exposedField SFTime  cycleInterval 1.0
    exposedField SFBool  loop          FALSE
    exposedField SFTime  startTime     0.0
    exposedField SFTime  stopTime      0.0
] {
    DEF SpinMe Transform {
    children      IS children
    },
    DEF Clock TimeSensor {
    cycleInterval IS cycleInterval
    loop          IS loop
    startTime     IS startTime
    stopTime      IS stopTime
    },
    DEF Spinner OrientationInterpolator {
    key [ 0.0, 0.5, 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 Spinner.set_fraction
    ROUTE Spinner.value_changed  TO SpinMe.set_rotation
}

SpinGroup {
    cycleInterval 4.0
    loop TRUE
    children [
    Shape {
          appearance DEF Green Appearance {
               material Material { diffuseColor 0.0 1.0 0.3 }
          }
          geometry Box { size 25.0 2.0 2.0 }
    },
    Shape {
          appearance USE Green
          geometry Box { size 2.0 25.0 2.0 }
    },
    Shape {
          appearance USE Green
          geometry Box { size 2.0 2.0 25.0 }
    }
    ]
}
```

Costruzione del prototipo di un tavolo



```
#VRML V2.0 utf8
   PROTO TwoColorTable [
        field SFColor legColor .8 .4 .7
         field SFColor topColor .6 .6 .1 ]
         { Transform
            { children [
                Transform { # table top
                    translation 0 0.6 0
                        children Shape {
                           appearance Appearance {
                            material Material {diffuseColor IS topColor}
                              }
                           geometry Box { size 1.2 0.2 1.2 }
                         }
                 }
                Transform { # first table leg
                    translation -.5 0 -.5
                        children DEF Leg Shape {
                          appearance Appearance { material Material { diffuseColor
   IS legColor } }
                           geometry Cylinder { height 1 radius .1 }
                         }
                  }
                Transform { # another table leg
                    translation .5 0 -.5 children USE Leg }
                    Transform { # another table leg
                    translation -.5 0 .5 children USE Leg }
                    Transform { # another table leg
                        translation .5 0 .5 children USE Leg }
             ] # End of root
           Transform's children } # End of root
       Transform
     } # End of prototype
    # The prototype is now defined. Although it contains a
    # number of nodes, only the legColor and topColor fields
    # are public. Instead of using the default legColor and
    # topColor, this instance of the table has red legs and
    # a green top:
    TwoColorTable { legColor 1 0 0 topColor 0 1 0 }
    NavigationInfo { type "EXAMINE" } # Use the Examine viewer
```

Prototipo per nodo MultiColor

```
#VRML 2.0 UTF-8
PROTO MultiColorInterpolator [
  ##########################################################################
  # Node defined by Ignazio Locatelli (VRML DREAMERS)                      #
  # Use this node as you want in your non-commercial worlds:               #
  # I'm just asking you some little credits ;)                             #
  # Ignazio Locatelli - email: ignazio@logicom.it                          #
  # VRML DREAMERS: first Italian Vrml Group - http://www.lucia.it/vrml #
  ##########################################################################

  eventIn SFFloat set_fraction
  exposedField MFFloat key []
  exposedField MFVec3f keyValue []
  eventOut MFColor value_changed
]

{

  DEF InterpColor CoordinateInterpolator {
    set_fraction IS set_fraction
    key IS key
    keyValue IS keyValue
  }

  DEF S1 Script {
    field SFNode Interp USE InterpColor
    eventIn MFVec3f set_values
    eventOut MFColor newColors IS value_changed

    url "javascript:

      function set_values(value) {
        for (i = 0; i < Interp.keyValue.length; i++) {
          newColors[i][0] = value[i][0];
          newColors[i][1] = value[i][1];
          newColors[i][2] = value[i][2];
        }
      }
    "
  }

  ROUTE InterpColor.value_changed TO S1.set_values
}
# end of proto
```

Uso del prototipo MultiColor

```
#VRML V2.0 utf8

PROTO MultiColorInterpolator [

      ############################################################
      ######################
  # Node defined by Ignazio Locatelli (VRML DREAMERS)              #
  # Use this node as you want in your non-commercial worlds:       #
  # I'm just asking you some little credits ;)                     #
  # Ignazio Locatelli - email: ignazio@logicom.it                  #
  # VRML DREAMERS: first Italian Vrml Group - http://www.lucia.it/vrml #

      ############################################################
      ######################

  eventIn SFFloat set_fraction
  exposedField MFFloat key []
  exposedField MFVec3f keyValue []
  eventOut MFColor value_changed
]

{

  DEF InterpColor CoordinateInterpolator {
    set_fraction IS set_fraction
    key IS key
    keyValue IS keyValue
  }

  DEF S1 Script {
    field SFNode Interp USE InterpColor
    eventIn MFVec3f set_values
    eventOut MFColor newColors IS value_changed

    url "javascript:

      function set_values(value) {
        for (i = 0; i < Interp.keyValue.length; i++) {
          newColors[i][0] = value[i][0];
          newColors[i][1] = value[i][1];
          newColors[i][2] = value[i][2];
        }
      }
    "
  }

   ROUTE InterpColor.value_changed TO S1.set_values
}
# end of proto
Transform {
 translation 0 0 -10
 rotation 0 1 0 1
#
 children [
  DEF ColorCube Shape {
   appearance Appearance { material Material { diffuseColor 0 0 1 } }
   geometry IndexedFaceSet {
    coord Coordinate {
     point [ -2 -2 2, 2 -2 2, 2 -2 -2, -2 -2 -2,
             -2 2 2, 2 2 2, 2 2 -2, -2 2 -2 ]
    }
    coordIndex [
       3,2,1,0,-1,
       0,1,5,4,-1,
       1,2,6,5,-1,
       4,5,6,7,-1,
       0,4,7,3,-1,
       3,7,6,2,-1
    ]
    color DEF C1 Color {
     color [ 1 0 0, 1 1 0, 0 1 0, 0 0 1, 0 1 1, 1 0 1, 1 1 1, 0 0 0 ]
    }
    colorIndex [
       3,2,1,0,-1,
       0,1,5,4,-1,
       1,2,6,5,-1,
       4,5,6,7,-1,
       0,4,7,3,-1,
       3,7,6,2,-1 ]
   }
  }
 ]
}

DEF T1 TimeSensor {
  enabled TRUE
  startTime 1
  loop TRUE
  cycleInterval 5
}

DEF M1 MultiColorInterpolator {
  key [0 .5 1]
  keyValue [
    1 0 0, 1 1 0, 0 1 0, 0 0 1, 0 1 1, 1 0 1, 1 1 1, 0 0 0,
    0 1 0, 1 0 0, 1 1 0, 0 1 0, .5 1 .5, 0 1 0, .2 .7 .2, 0 1 0,
    1 0 0, 1 1 0, 0 1 0, 0 0 1, 0 1 1, 1 0 1, 1 1 1, 0 0 0
  ]
}
ROUTE T1.fraction_changed TO M1.set_fraction
ROUTE M1.value_changed TO C1.set_color
```

Virtual Reality Modeling Language

VRML

Programmi Script

Programmi in VRML

- Abbiamo spesso fatto riferimento alla programmazione in VRML mediante il nodo **`Script`**, in particolare tutte le volte che abbiamo evidenziato la necessità di un supporto di programmazione più evoluto rispetto alle funzionalità di base di VRML
- In particolare la realizzazione di particolari animazioni o interpolazioni il cui comportamento esuli da quelli previsti in VRML ci porta alla necessità di scrivere programmi **`Script`**.
- Il nodo **`Script`** è un contenitore: ha campi, `eventIn` ed `eventOut`, ma non ha azioni proprie.
- L'utente specifica le proprie azioni attraverso un **programma script**, che è una applicazione, spesso di modeste dimensioni, scritta nei linguaggi di programmazione Java, ECMAScript o Javascript.
- Usando un **programma script** ed un nodo **`Script`** si possono realizzare nodi che effettuano azioni molto complesse, dei sensori ed interpolatori personalizzati, estendendo moltissimo le funzioni di base di VRML.

Il nodo Script

- Il nodo necessita di una serie di campi, eventIn ed eventOut e delle descrizione delle azioni che devono essere effettuate su questi campi.
- Quando si definisce un nodo Script, occorre definire in modo rigoroso i vari campi, gli eventIn e gli eventOut che costituiscono l'interfaccia del nodo verso il programma VRML.
- Il nodo fornisce una sintassi specifica per la definizione di campi, eventIn e eventOut
- Il nome assegnato a queste grandezze deve soddisfare le regole generali già viste per l'istruzione DEF, in particolare per convenzione VRML i nomi di nodi iniziano con la maiuscola; i nomi dei campi devono iniziare con la lettera minuscola; ogni parola successiva deve iniziare con la lettera maiuscola; si possono usare numeri ed il carattere _

Definizione di campi nel nodo Script

```
Script {
        …
         field filedType fieldName initialValue
        …
}
```

Es:

```
field  SFBool     enabled              TRUE
field  SFColor        diffuseColor     0.8 0.8 0.8
field  MFColor        spacecraftColor  [1.0 0.0 0.0, 0.0
                                        0.0 1.0]
field  SFFloat        intensity        1.0
field  SFImage        textureImage     0 0 0
field  SFInt32        phoneNumber          0755855048
field  MFInt32        indicatorChoice  [42, 531, 5]
field  SFNode         texture          NULL
field  SFString       robotName        "Marvin"
```

Definizione di eventIn nel nodo Script

```
Script {
        …
         eventIn eventInType eventInName
        …
}
```

Es:

```
eventIn   SFBool       set_enabled
eventIn   MFColor      set_skyColor
eventIn   SFFloat      set_height
eventIn   SFImage      set_texture
eventIn   SFInt32      set_population
eventIn   SFRotation   set_orbitAngle
eventIn   MFNode       addChildren
eventIn   SFString     set_candyFlavor
```

Definizione di eventOut nel nodo Script

```
Script {
      …
       eventOut eventOutType eventOutName
      …
}
```

Es:

```
eventOut    SFBool        isActive
eventOut    SFColor       eyeColor_changed
eventOut    MFFloat       stockIndexes_changed
eventOut    SFImage       panorama_changed
eventOut    SFInt32       age_changed
eventOut    SFRotation    orientation_changed
eventOut    SFNode        shape_changed
eventOut    SFString      fontFamily_changed
```

Uso di campi, eventIn e eventOut

Utilizzando la sintassi vista è possibile definire i campi, gli eventIn e gli eventOut di un nodo **Script**. Ad es:

```
Script {
field       SFFloat     gravity     -9.8
field       SFVec3f     velocity    1.0 0.0 0.0
field       SFVec3f     position    0.0 0.0 0.0
eventIn     SFVec3f     set_position
eventOut    SFVec3f     position_changed
eventIn     SFFloat     set_fraction
}
```

Non si possono creare exposed-field: l'analogo si ottiene definendo un campo (es: **position**), un eventIn (es: **set_position**) ed un eventOut (es: **position_changed**) seguendo la convenzione standard di VRML.

Definizione di un programma script

Una volta definiti campi, eventIn e eventOut del nodo **Script**, occorre definire il **programma script** che compia le azioni volute e ritorni negli eventOut le grandezze richieste. Un programma script viene indicato nel nodo specificando la URL del relativo file:

```
Script {
url             "myscript.js"
field      ….
eventIn         …
eventOut        …
}
```

Quando si usa javascript come linguaggio di programmazione, si può includere il sorgente nel nodo, specificandole nel campo **url** al posto della URL e precedendo le istruzioni con la chiave **javascript:**.

Definizione di un programma script

```
Script {
 url "javascript:
       function set_position (pos, time){
             position = pos;
             }
     "
 field        ….
 eventIn      …
 eventOut     …
 }
```

Il vantaggio dell'inclusione del sorgente del programma direttamente nel nodo risiede nel fatto che in questo modo si evita la frammentazione dei sorgenti in tanti piccoli f le.

Definizione di un programma script

```
Script {
  url ["javascript: ....",     #Custom protocol Javascript
       "http:bar.com/foo.js",       #Std protocol Javascript
       "http://bar.com/foo.class"   #Std protocol Java byte

  }
```

The file extension for Java bytecodes

Controllo del comportamento di un programma script

- Un nodo **`Script`** fornisce due campi addizionali che controllano il comportamento del programma script: **`mustEvaluate`** e **`directOutput`**.
- Per non appesantire troppo il browser con i calcoli inerenti i vari nodi **`Script`** definiti, il calcolo di questi programmi di norma viene posticipato in momenti in cui il browser può sopportare il relativo carico computazionale.
- Se l'utente ha invece bisogno di un comportamento diverso, per cui il programma venga eseguito ogni volta che arrivano dati dall'`EventIn`, allora occorre definire il valore **`mustEvaluate`** come **`TRUE`**.

Controllo del comportamento di un programma script (i)

- E' bene lasciare il valore di default (**FALSE**) nei casi in cui non sia indispensabile un calcolo immediato.
- Ci sono casi in cui un programma script deve avere un controllo superiore alle condizioni standard (ricevere dei dati in input e produrre dei risultati in output). Ad esempio quando deve cambiare dei valori di campi di altri nodi.
- Si può ad esempio scrivere un **programma script** che automaticamente crei i circuiti relativi a delle **ROUTE** tra i principali nodi del nostro mondo.
- La costruzione di **ROUTE** è una azione globale che riguarda tutto il mondo virtuale e richiede in generale dei permessi che normalmente i **programmi script** non hanno.

Controllo del comportamento di un programma script (ii)

- Def nendo `TRUE` il campo `directOutput` si istruisce VRML che il **programma script** deve avere il potere di manipolare direttamente il mondo VRML. Il valore tipico è `FALSE` ed il **programma script** non ha il potere di modif care direttamente il mondo VRML.

Programmi script in Java e Javascript

- Le funzionalità che un browser VRML può fornire ad un programma script di tipo Java o Javascript è classificato in alcune Application Programming Interface (API).
- Le API dei programmi script comprendono una serie di funzionalità afferenti alle seguenti categorie:
  - Accesso ai `campi` di interfaccia e agli `eventOut` del programma del nodo `Script`.
  - Conversione tra i tipi di dati VRML e quelli dei due linguaggi
  - Inizializzazione e shutdown di un programma e capacità di rispondere agli eventi in arrivo.
  - Accesso al browser per modificare il contenuto del mondo o per caricarne uno nuovo.

Accesso ai campi di interfaccia e agli eventOut

- In Java il metodo `getField`, per esempio, fornisce un handle per un campo interfaccia. Usando il valore ritornato dal metodo `getField`, il programma Java può leggere e scrivere campi interfaccia.
- Analogamente il metodo `getEventOut` fornisce un handle per un eventOut. Usando il valore ritornato dal metodo `getEventOut`, il programma Java può inviare eventi usando degli eventOut.
- In Javascript l'API crea automaticamente una variabile di interfaccia per ogni campo interfaccia o eventOut. Usando una variabile interfaccia il programma Javascript può leggere o scrivere campi interfaccia ed inviare venti attraverso eventOut.

Accesso ai campi di interfaccia e agli eventOut (i)

- Ogni evento scritto su un eventOut e ricevuto da un eventIn è composto da due componenti: un event value e un event timestamp.
  - Un event value è un valore, come una posizione 3-D, generata da un nodo e inviata usando un eventOut.
  - Un event timestamp è un valore che contiene il tempo assoluto in cui il valore è stato creato e inviato attraverso una route.
- Gli eventi che arrivano ad un eventIn di un nodo vengono ordinati in modo tale che gli eventi inviati prima arrivino per primi.

Generazione di eventi

- Un programma script può rispondere a tre tipi di attività:
  - Inizializzazione: avviene all'inizio dell'esecuzione e prima di gestire eventi. Sia in Java che Javascript si può scrivere una funzione `Initialize()` che viene chiamata all'inizializzazione.
  - Shutdown: viene eseguita al termina del programma. Sia in Java che Javascript si può scrivere una funzione `Shutdown()` che viene chiamata al termine del programma.
  - Ricevimento di eventi: si verifica quando un nuovo dato viene ricevuto in input attraverso un `eventIn`. In questo caso il programma legge il valore ed esegue le istruzione per produrre un evento da inviare attraverso un `eventOut`.

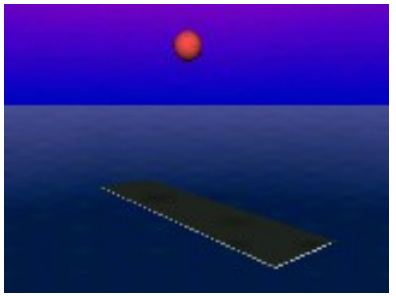
Il nodo `Script`

- Il nodo **`Script`** definisce i campi, gli EventIn e gli EventOut di un nodo contenente un programma script.

```
Script {
      url              [ ]                  #exposedField  MFString
      mustEvaluate     FALSE                #field         SFBool
      directOutput     FALSE                #field         SFBool
# qualsiasi numero di:
      field            fieldType    fieldName initialValue
      eventIn          eventInType  eventInName
      eventOut         eventOutType eventOutName
         }
```

Creazione di un interpolatore

```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    Background {
        skyColor     [ 1.0 0.0 0.8,  0.5 0.0 0.8,  0.0 0.0 0.8 ]
        skyAngle     [ 1.309, 1.571 ]
        groundColor  [ 0.0 0.0 0.1,  0.0 0.1 0.3,  0.3 0.3 0.6 ]
        groundAngle  [ 1.309, 1.571 ]
    },
    # Floor
    Shape {
        appearance Appearance {
            material Material { }
        }
        geometry Box { size 2.0 0.01 0.5 }
    },
    # Animating red ball
    Transform {
        translation 0.0 1.1 0.0
        children DEF BallTransform Transform {
            children Shape {
                appearance Appearance {
                    material Material {
                        diffuseColor 1.0 0.3 0.3
                    }
                }
                geometry Sphere { radius 0.1 }
            }
        }
    },
    # Animation clock
    DEF Clock TimeSensor {
        cycleInterval 4.0
        loop TRUE
    },
    # Script
    DEF Mover Script {
        url "move1.js"
        eventIn  SFFloat set_fraction
        eventOut SFVec3f value_changed
    }
    ]
}
ROUTE Clock.fraction_changed TO Mover.set_fraction
ROUTE Mover.value_changed    TO BallTransform.set_translation
```

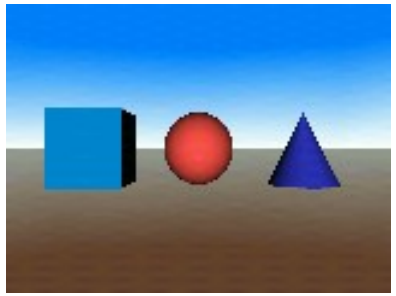


move1.js:

```
// Move a shape in a straight path
function set_fraction( fraction, eventTime ) {
        value_changed[0] = fraction;    // X component
        value_changed[1] = 0.0;         // Y component
        value_changed[2] = 0.0;         // Z component
}
```

Creazione di un filtro di eventi

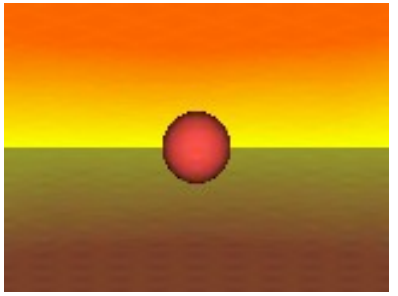
```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Bindable Backgrounds (cyan, red, blue)
    DEF Back1 Background {
            skyColor     [ 0.0 0.2 0.7,  0.0 0.5 1.0,  1.0 1.0 1.0 ]
            skyAngle     [ 1.309, 1.571 ]
            groundColor  [ 0.1 0.1 0.0,  0.4 0.25 0.2,  0.6 0.6 0.6 ]
            groundAngle  [ 1.309, 1.571 ]
    },
    DEF Back2 Background {
            skyColor     [ 1.0 0.0 0.0,  1.0 0.4 0.0,  1.0 1.0 0.0 ]
            skyAngle     [ 1.309, 1.571 ]
            groundColor  [ 0.1 0.1 0.0,  0.5 0.25 0.2,  0.6 0.6 0.2 ]
            groundAngle  [ 1.309, 1.571 ]
    },
    DEF Back3 Background {
            skyColor     [ 1.0 0.0 0.8,  0.5 0.0 0.8,  0.0 0.0 0.8 ]
            skyAngle     [ 1.309, 1.571 ]
            groundColor  [ 0.0 0.0 0.0,  0.0 0.1 0.3,  0.3 0.3 0.6 ]
            groundAngle  [ 1.309, 1.571 ]
    }
    # Shapes to act as buttons (cyan, red, blue)
    Transform { translation -3.0 0.0 0.0
            children [
                    Shape {
                            appearance Appearance {
                                    material Material {
                                            diffuseColor 0.0 0.5 0.8
                                    }
                            }
                            geometry Box { }
                    },
                    DEF BackButton1 TouchSensor { }
            ]
    },
    Group {
            children [
                    Shape {
                            appearance Appearance {
                                    material Material {
                                            diffuseColor 1.0 0.3 0.3
                                    }
                            }
                            geometry Sphere { }
                    },
                    DEF BackButton2 TouchSensor { }
            ]
    },
    Transform { translation 3.0 0.0 0.0
            children [
                    Shape {
                            appearance Appearance {
                                    material Material {
                                            diffuseColor 0.2 0.2 0.8
                                    }
                            }
                            geometry Cone { }
                    },
                    DEF BackButton3 TouchSensor { }
            ]
    }
    # Scripts
    DEF Filter1 Script {
            url "javascript:
                    function set_boolean( bool, eventTime ) {
                            if ( bool == true ) { true_changed  = true; }
                            else                { false_changed = true; }
                    }"
            eventIn  SFBool set_boolean
            eventOut SFBool true_changed
            eventOut SFBool false_changed
    }
    DEF Filter2 Script {
            url "javascript:
                    function set_boolean( bool, eventTime ) {
                            if ( bool == true ) { true_changed  = true; }
                            else                { false_changed = true; }
                    }"
            eventIn  SFBool set_boolean
            eventOut SFBool true_changed
            eventOut SFBool false_changed
    }
    DEF Filter3 Script {
            url "javascript:
                    function set_boolean( bool, eventTime ) {
                            if ( bool == true ) { true_changed  = true; }
                            else                { false_changed = true; }
                    }"
            eventIn  SFBool set_boolean
            eventOut SFBool true_changed
            eventOut SFBool false_changed
    },
    ]
}
```



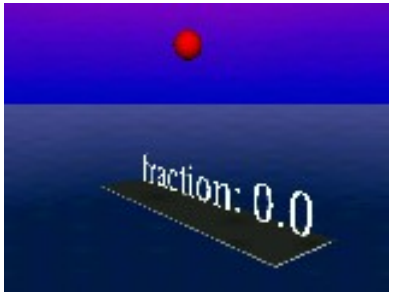
```
ROUTE BackButton1.isActive TO Filter1.set_boolean
ROUTE BackButton2.isActive TO Filter2.set_boolean
ROUTE BackButton3.isActive TO Filter3.set_boolean
ROUTE Filter1.true_changed TO Back1.set_bind
ROUTE Filter2.true_changed TO Back2.set_bind
ROUTE Filter3.true_changed TO Back3.set_bind
```

Creazione di un filtro di tempi

```
#VRML V2.0 utf8
#  The VRML 2.0 Sourcebook
#  Copyright [1997] By
#  Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Background
    Background {
         skyColor     [ 1.0 0.0 0.0,  1.0 0.4 0.0,  1.0 1.0 0.0 ]
         skyAngle     [ 1.309, 1.571 ]
         groundColor  [ 0.1 0.1 0.0,  0.5 0.25 0.2,  0.6 0.6 0.2 ]
         groundAngle  [ 1.309, 1.571 ]
    },
    # On-off switch
    Shape {
         appearance Appearance {
              material Material {
                   diffuseColor 1.0 0.3 0.3
              }
         }
         geometry Sphere { }
    },
    DEF Touch TouchSensor { },
    # Sound
    Sound {
         source DEF Audio AudioClip {
              url "willow1.wav"
              loop TRUE
              stopTime 1.0
         }
    },
    # Filters
    DEF Filter Script {
         url "javascript:
              function set_boolean( bool, eventTime ) {
                   if ( bool == true ) { true_changed  = true; }
                   else                { false_changed = true; }
              }"
         eventIn  SFBool set_boolean
         eventOut SFBool true_changed
         eventOut SFBool false_changed
    },
    DEF TimeFilter Script {
         url "javascript:
              function set_boolean( bool, timeStamp ) {
                   eventTime = timeStamp;
              }"
         eventIn  SFBool set_boolean
         eventOut SFTime eventTime
    }
    ]
}
ROUTE Touch.isActive        TO Filter.set_boolean
ROUTE Filter.true_changed   TO TimeFilter.set_boolean
ROUTE TimeFilter.eventTime  TO Audio.set_startTime
ROUTE Touch.touchTime       TO Audio.set_stopTime
```

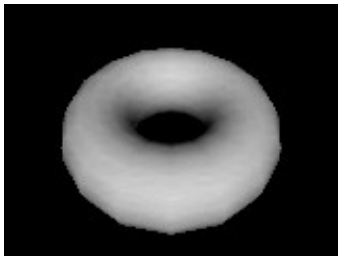


Creazione di un route debugger




```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright (1997) By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    Background {
        skyColor     [ 1.0 0.0 0.0 0.8,  0.5 0.0 0.8,  0.0 0.0 0.8 ]
        skyAngle     [ 1.309, 1.571 ]
        groundColor [ 0.0 0.0 0.0 0.1,  0.0 0.1 0.3,  0.3 0.3 0.6 ]
        groundAngle [ 1.309, 1.571 ]
    }
    # Floor
    Shape {
        appearance Appearance {
            material Material { }
        }
        geometry Box { size 2.0 0.01 0.5 }
    }
    # Animating red ball
    Transform {
        translation 0.0 1.1 0.0
        children DEF BallTransform Transform {
            children Shape {
                appearance Appearance {
                    material Material {
                        diffuseColor 1.0 0.0 0.0
                    }
                }
                geometry Sphere { radius 0.1 }
            }
        }
    }
    # Animation clock
    DEF Clock TimeSensor {
        cycleInterval 4.0
        loop TRUE
    }
    # Script
    DEF Mover Script {
        url "javascript:
            function set_fraction( fraction, eventTime ) {
                value_changed[0] = fraction;
                value_changed[1] = radius * Math.sin( turns * fraction * 6.28 );
                value_changed[2] = radius * Math.cos( turns * fraction * 6.28 );
            }"
        field    SFFloat radius 1.0
        field    SFFloat turns  1.0
        eventIn  SFFloat set_fraction
        eventOut SFVec3f value_changed
    }
    # Debugger
    DEF Debug Script {
        url "javascript:
            function initalize( ) {
                string_changed[0] = label + ':';
            }
            function set_float( f, ts ) {
                string_changed[0] = label + ': ' + f;
            }"
        field    SFString label "fraction"
        eventIn  SFFloat set_float
        eventOut MFString string_changed
    }
    Transform {
        translation 0.0 0.01 -0.15
        children Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 0.0 0.0 0.0
                    emissiveColor 1.0 1.0 1.0
                }
            }
            geometry DEF Message Text {
                fontStyle FontStyle {
                    size 0.35
                    justify "MIDDLE"
                }
            }
        }
    }
    ]
}
ROUTE Clock.fraction_changed  TO Mover.set_fraction
ROUTE Mover.value_changed     TO BallTransform.set_translation
ROUTE Clock.fraction_changed  TO Debug.set_float
ROUTE Debug.string_changed    TO Message.set_string
```

Controllo di una forma con un programma script



```
#VRML V2.0 utf8
# The VRML 2.0 Sourcebook
# Copyright [1997] By
# Andrea L. Ames, David R. Nadeau, and John L. Moreland
Group {
    children [
    # Donut, initially empty
    Shape {
         appearance Appearance {
              material Material { }
         }
         geometry DEF Donut Extrusion {
              crossSection [ ]
              spine [ ]
              creaseAngle 1.57
              beginCap FALSE
              endCap   FALSE
         }
    },
    # Donut maker
    DEF DonutMaker Script {
         url "donutmkr.js"
         field    SFFloat crossSectionRadius     1.0
         field    SFFloat spineRadius            2.0
         field    SFInt32 crossSectionResolution 16
         field    SFInt32 spineResolution        16
         eventIn  SFFloat set_crossSectionRadius
         eventIn  SFFloat set_spineRadius
         eventOut MFVec2f crossSection_changed
         eventOut MFVec3f spine_changed
    }
    ]
}
ROUTE DonutMaker.crossSection_changed TO Donut.set_crossSection
ROUTE DonutMaker.spine_changed        TO Donut.set_spine
```

DonutMaker.js

```
function initialize( ) {
    generateCrossSection( );
    generateSpine( );
}
function set_crossSectionRadius( csr, ts ) {
    crossSectionRadius = csr;
    generateCrossSection( );
}
function set_spineRadius( sr, ts ) {
    spineRadius = sr;
    generateSpine( );
}

function generateCrossSection( ) {
    angle = 0.0;
    delta = 6.28 / crossSectionResolution;
    for ( i = 0; i <= crossSectionResolution; i++ )
{
        crossSection_changed[i][0] =
crossSectionRadius * Math.cos( angle );
        crossSection_changed[i][1] =
-crossSectionRadius * Math.sin( angle );
        angle += delta;
    }
}

function generateSpine( ) {
    angle = 0.0;
    delta = 6.28 / spineResolution;
    for ( i = 0; i <= spineResolution; i++ ) {
        spine_changed[i][0] = spineRadius *
Math.cos( angle );
        spine_changed[i][1] = 0.0;
        spine_changed[i][2] = -spineRadius *
Math.sin( angle );
        angle += delta;
    }
}
```

Virtual Reality Modeling Language

VRML

ECMAScript /Javascript API

Funzioni Javascript

```
void    eventsProcessed( )
void    initialize( )
void    shutdown( )
numeric parseInt( String string, [radix] )
numeric parseFloat( String string )
```

Metodi Javascript

Metodo eseguito prima che il mondo VRML venga presentato all'utente e prima che qualsiasi evento sia ricevuto da qualsiasi nodo:

```
void    initialize( )
```

Metodo che può essere eseguito dopo che un determinato numero di eventi sono stati ricevuti dalla script (non ha parametri):

```
void    eventsProcessed( )
```

Metodi eseguiti per accedere ai campi:

```
numeric parseInt( String string, [radix] )
numeric parseFloat( String string )
```

Metodo eseguito prima che la script venga cancellata dalla memoria o prima che il mondo VRML contenente la script venga scaricato dalla memoria:

```
void    shutdown( )
```

Funzioni matematiche

Uso: `a=Math.abs(x)`

Funzione	uso
`Math.abs(x)`	Returns the absolute value of x.
`Math.acos(x)`	Returns the arc cosine (in radians) of x.
`Math.asin(x)`	Returns the arc sine (in radians) of x.
`Math.atan(x)`	Returns the arc tangent (in radians) of x.
`Math.ceil(x)`	Returns the least integer greater than or equal to x.
`Math.cos(x)`	Returns the cosine of the variable x.
`Math.exp(x)`	Returns e, to the power of x (i.e. e^x).
`Math.floor(x)`	Returns the greatest integer less than or equal to x.
`Math.log(x)`	Returns the natural log of the variable x.
`Math.max(x,z)`	Returns the larger of the two variables x and z.
`Math.min(x,z)`	Returns the smaller of the two variables x and z.
`Math.pow(x,z)`	Returns the value of the variable x to the z^{th} power.
`Math.random(x)`	Returns a random number between 0 and x. If not used x is assumed to equal 1
`Math.round(x)`	Returns the variable x rounded to the nearest integer.
`Math.sin(x)`	Returns the sine of x, where x is expressed in radians.
`Math.sqrt(x)`	Returns the square root of x.
`Math.tan(x)`	Returns the tangent of x, where x is expressed in radians .

Funzioni per vettori

Uso:

```
DEF SCR-VEC3F Script {. . .
eventOut SFVec3f new_translation . . .}
  . . .
tVec = new_translation.divide(2);
new_translation = new_translation.subtract(tVec);
. . .
```

Funzione	uso
.multiply(x)	Multiply x by the f eld values.
.divide(x)	Divide the f eld values by x.
.add(x)	Add x to the f eld values
.subtract(x)	Subtract x by the f eld values.
.dot(x)	Get the dot product of the f eld values and x.
.cross(x)	Get the cross product of the f eld value and x
.normalize()	Returns the normalized value of the f eld.
.negate()	Returns the negative value of the f eld.
.length()	Returns the length of a vector.

Funzioni per i metodi Browser

Browser Object:

```
mymfnode = Browser.createVrmlFromString('Sphere {}');
```

Return value	Method
String	**getName**( )
String	**getVersion**( )
Numeric	**getCurrentSpeed**( )
Numeric	**getCurrentFrameRate**( )
String	**getWorldURL**( )
Void	**replaceWorld**(MFNode nodes)
MFNode	**createVrmlFromString**(String vrmlSyntax)
Void	**createVrmlFromURL**(MFString url, Node node, String event)
Void	**addRoute**(SFNode fromNode, String fromEventOut, SFNode toNode, String toEventIn)
Void	**deleteRoute**(SFNode fromNode, String fromEventOut, SFNode toNode, String toEventIn)
Void	**loadURL**(MFString url, MFString parameter)
Void	**setDescription**(String description)

Funzioni per i metodi SFColor

SFColor Object (corrisponde al campo VRML SFColor):

```
sfColorObjectName = new SFColor(float r, float g, float b)
```

Metodo	Descrizione
void setHSV(float *h*, float *s*, float *v*)	Stabilisce il valore del colore assegnandone i valori *hue*, *saturation*, e *value*.
numeric[3] getHSV( )	Fornisce il valore del colore in un vettore numerico di tre elementi, con il valore di *hue* all'indice 0, quello di *saturation* all'indice 1 e quello di *value* all'indice 0
String toString( )	Fornisce una Stringa contenente il valore della codifica di *r*, *g* e *b*.

Funzioni per i metodi SFNode

SFNode Object (corrisponde al campo VRML SFNode):

***sfNodeObjectName* = new SFNode(String *vrmlstring*)**

Metodo	Descrizione
String toString( )	Fornisce una stringa che se interpretata come valore del campo SFNode, produrrà questo nodo.

Funzioni per i metodi SFRotation

SFRotation Object (corrisponde al campo VRML SFRotation):

```
sfRotationObjectName = new SFRotation(numeric x, numeric y, numeric z, numeric angle)
sfRotationObjectName = new SFRotation(SFVec3f axis, numeric angle)
```

Metodo	Descrizione
SFVec3f getAxis( )	Returns the axis of rotation.
SFRotation inverse( )	Returns the inverse of this object's rotation.
SFRotation multiply(SFRotation *rot*)	Returns the object multiplied by the passed value.
SFVec3f multVec(SFVec3f *vec*)	Returns the value of *vec* multiplied by the matrix corresponding to this object's rotation.
void setAxis(SFVec3f *vec*)	Sets the axis of rotation to the value passed in *vec*.
SFRotation slerp(SFRotation *dest*, numeric *t*)	Returns the value of the spherical linear interpolation between this object's rotation and *dest* at value 0 <= *t* <= 1. For *t* = 0, the value is this object's rotation. For *t* = 1, the value is *dest*.
String toString( )	Returns a String containing the VRML 97 utf8 encoded value of x, y, z, and angle.

Example_1.js

```
DEF Example_1 Script {
        field    SFColor currentColor 0 0 0
    eventIn  SFColor colorIn
    eventOut SFBool  isRed

    url "javascript:
        function colorIn(newColor, ts) {
            // This method is called when a colorIn event is
  received
            currentColor = newColor;
        }

        function eventsProcessed( ) {
            if (currentColor[0] >= 0.5)
                // if red is at or above 50%
                isRed = true;
        }"
}
```

Baa.js

```
DEF Sensor TouchSensor {}

DEF Baa Script {
    field   SFNode myself USE Baa
    field   SFNode fromNode USE Sensor
    eventIn SFBool clicked
    eventIn SFBool trigger_event

    url "javascript:
        function trigger_event(eventIn_value){
            // do something and then add routing
            Browser.addRoute(fromNode, 'isActive', myself,
  'clicked');
        }

        function clicked(value){
            // do something
        }"
}
```

Assigning with reference and assigning by value

```
Script{
    eventIn  SFBool  eI
    eventOut SFVec3f eO
    field    MFVec3f f []

    url "javascript:
        function eI( ) {
            eO = new SFVec3f(0,1,2);  // 'eO' contains the value
                                      // (0,1,2) which will be sent
                                      // out when the function
                                      // is complete.
            a = eO;                   // 'a' contains a SFVec3f
                                      // object with the value (0,1,2)
            b = a;                    // 'b' references the same
                                      // object as 'a'.
            a.x = 3;                  // 'a' and 'b' both contain
                                      // (3,1,2). 'eO' is unchanged.
            f[1] = a;                 // 'f[1]' contains the value
                                      // (3,1,2).
            c = f[1];                 // `c' contains a SFVec3f
                                      // object with the value (3,1,2)
            f[1].y = 4;               // 'f[1]' contains the value
                                      // (3,4,2). 'c' is unchanged.
        }"
}
```

Uso di campi e metodi

```
DEF SCR-VEC3F Script {
    eventIn SFTime touched1
    eventIn SFTime touched2
    eventIn SFTime touched3
    eventIn SFTime touched4
    eventOut SFVec3f new_translation
    field SFInt32 count 1
    field MFVec3f verts  []

    url "javascript:
        function initialize( ) {
            verts[0] = new SFVec3f(0, 0, 0);
            verts[1] = new SFVec3f(1, 1.732, 0);
            verts[2] = new SFVec3f(2, 0, 0);
            verts[3] = new SFVec3f(1, 0.577, 1.732);
        }

        function touched1 (value) {
            new_translation = verts[count]; // move sphere around tetra
            count++;
            if (count >= verts.length) count = 1;
        }

        function touched2 (value) {
            var tVec;
            tVec = new_translation.divide(2);
            new_translation = new_translation.subtract(tVec);
        }

        function touched4 (value) {
            new_translation = new_translation.negate( );
        }

        function touched3 (value) {
            var a;
            a = verts[1].length( );
            a = verts[3].dot(verts[2].cross(verts[1]));
            a = verts[1].x;
            new_translation = verts[2].normalize( );
            new_translation = new_translation.add(new_translation);
        }"
}
```

Example_2.js

```
DEF Example_2 Script {
    field   SFNode myself USE Example_2
    field   SFNode root USE ROOT_TRANSFORM
    field   MFString url "foo.wrl"
    eventIn MFNode   nodesLoaded
    eventIn SFBool   trigger_event

    url "javascript:
        function trigger_event(value, ts){
            // do something and then fetch values
            Browser.createVRMLFromURL(url, myself, 'nodesLoaded');
        }

        function nodesLoaded(value, timestamp){
            if (value.length > 5) {
                 // do something more than 5 nodes in this MFNode...
            }
            root.addChildren = value;
        }"
}
```

ECMAScript (Javascript) resources

http://www.ecma-international.org

http://www.ecma-international.org/publications/standards/Ecma-262.htm

http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

http://developer.netscape.com/docs/manuals/communicator/jsref/

http://www.webreference.com/js/

http://javascript.internet.com/

http://javascript-reference.info/

Virtual Reality Modeling Language

VRML

Java API

Java

- Il campo url del nodo Script può contenere un puntatore al bytecode java:

```
Script { url "http://foo.co.jp/Example.class"
            eventIn SFBool start
        }
```

- L'estensione Java dei bytecode è **.class**
- Il tipo MIME è application/x-java
- Le classi java usate dal nodo **`Script`** devono estendere la classe java **`vrml.node.Script`** in modo da interfacciarsi opportunamente al browser VRML.

Gestione degli EventIn

- Gli eventi del nodo `Script` vengono passati al corrispondente metodo della piattaforma Java nel programma dello script: `processEvents()` o `processEvent()`.
- Per un file bytecode Java specificato nel campo url, si devono verificare le seguenti tre condizioni:
- deve contenere la definizione di classe il cui nome sia esattamente quello del nome del file
- deve essere una sottoclasse della classe Script
- deve essere dichiarata come classe pubblica

Gestione degli eventIn (i)

- Il seguente nodo `Script` ha un `eventIn` il cui nome è `start`.

```
Script {
    url "http://foo.co.jp/Example1.class"
    eventIn SFBool start
      }
```

- Il nodo punta al file script Example1.class, il cui sorgente, Example1.java, è:

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;
public class Example1 extends Script {
... // This method is called when any event is received
public void processEvent(Event e){
// ... perform some operation ...
     }
}
```

Quando l'`eventIn` *start* viene inviato, il metodo `ProcessEvent()` riceve l'`eventIn` e viene eseguito

Passaggio di parametri con oggetti event

- L'oggetto Java riceve l' **`eventIn`** come oggetto event.
- L'oggetto event ha tre campi associati: nome, valore e timestamp, i cui valori sono passati dall'**`eventIn`**. Questi valori possono essere letti utilizzando il corrispondente metodo dell'oggetto event:

```
public class Event implements Cloneable {
  public String getName();
  public ConstField getValue();
  public double getTimeStamp();
  // other methods ...
}
```

Metodo `processEvents()`

- L'autore può definire un metodo **`processEvents()`** in una classe che è chiamata quando lo script riceve degli eventi.
- Il prototipo del metodo **`processEvents()`** è:

```
public void processEvents(int count, Event events[]);
```

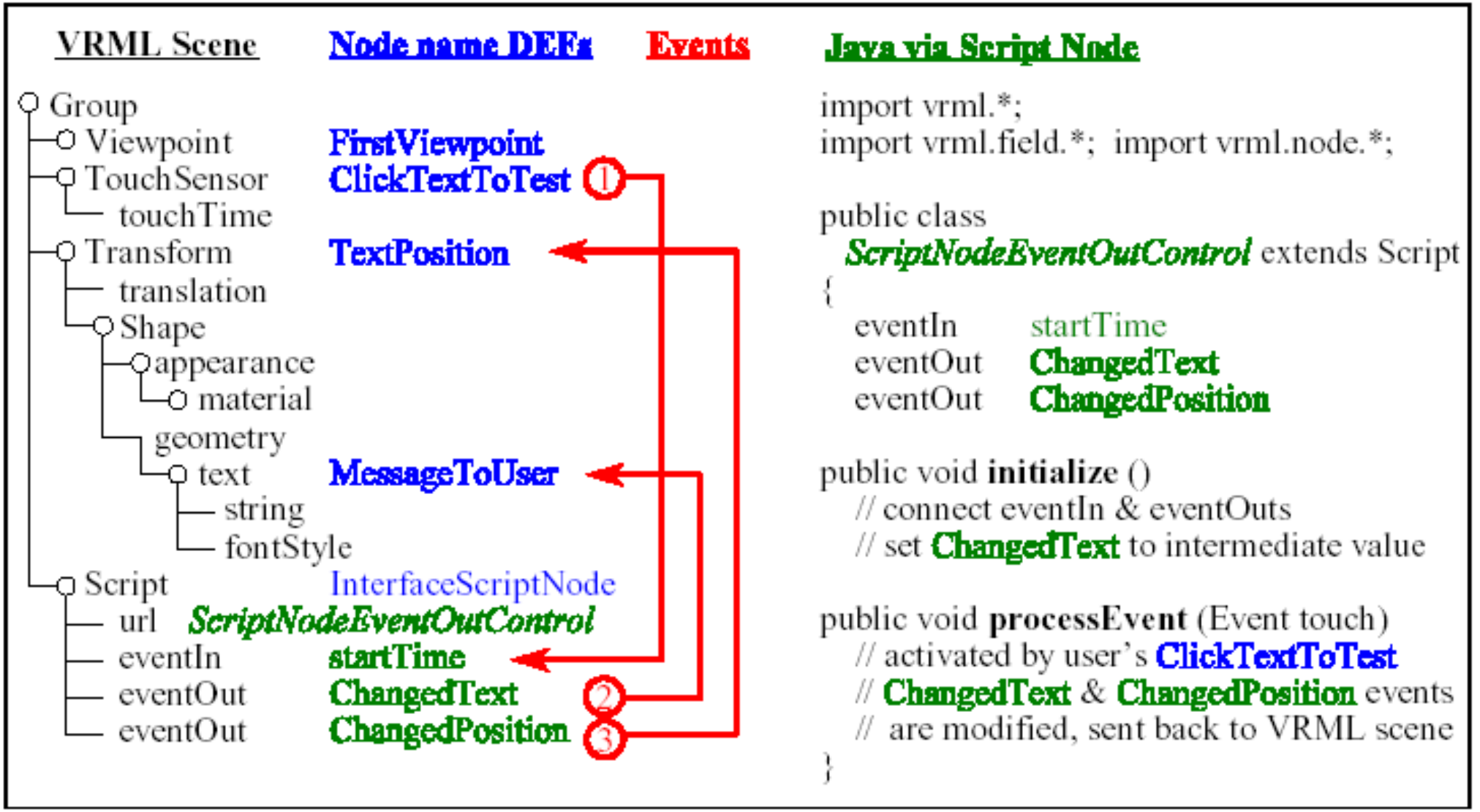
`count` indica il numero di eventi prodotti e **`events`** è il vettore di eventi generati. Il suo comportamento di default è di effettuare un'iterazione sul numero di eventi, invocando ogni volta **`processEvent()`**:

```
public void processEvents(int count, Event events[])
{
   for (int i = 0; i < count; i++)
   { processEvent(events[i]);
   }
}
```

esempio

```
Transform {
   children [
      DEF TS TouchSensor {}
      Shape { geometry Cone {} }
         ]
      }
      DEF SC Script {
         url "Example.class"
         eventIn SFBool isActive
         eventIn SFTime touchTime
         }
ROUTE TS.isActive TO SC.isActive
ROUTE TS.touchTime TO SC.touchTime
```

Esempio (i)



`ScriptNodeEventOutControl.wrl` ScriptNodeEventOutControl.java

Metodo `initialize()`

- Gli autori possono definire un metodo **`initialize()`** in una classe **`Script`** che viene chiamata prima che il browser presenti il mondo all'utente e prima che qualsiasi evento sia gestito da qualsiasi nodo dello stesso file VRML che contiene il programma.
- Il metodo **`initialize()`** è chiamato una volta soltanto nel corso della vita dell'oggetto **`Script`**.
- Il prototipo del metodo **`initialize()`** è il seguente:

```
public void  initialize();
```

- Il suo comportamento di default è quello di non effettuare operazioni.

Metodo `processEvent()`

- Gli autori possono def nire in una classe un metodo **`processEvent()`.**
- Il prototipo del metodo **`processEvent()`** è il seguente:

```
public void processEvent(Event event);
```

- Il suo comportamento di default è quello di non effettuare operazioni.

Metodo `eventsProcessed()`

- Gli autori possono definire un metodo **`eventsProcessed()`** in una classe che viene chiamata dopo che è stato ricevuto un insieme di eventi.
- Ciò consente ai nodi Script che non hanno bisogno di conoscere l'ordine degli eventi ricevuti di generare meno eventi rispetto ad un nodo che produce tanti eventi quanti ne riceve.
- Il prototipo del metodo **`eventsProcessed()`** è il seguente:

```
public void  eventsProcessed();
```

- Il suo comportamento di default è quello di non effettuare operazioni.

Metodo `shutdown()`

- Gli autori possono def nire un metodo **`shutdown()`** in una classe che viene chiamata quando il corrispondente nodo **`Script`** è cancellato o il mondo contenente il nodo **`Script`** è sostituito con un altro mondo.
- Il prototipo del metodo **`shutdown()`** è il seguente:

```
public void  shutdown();
```

- Il suo comportamento di default è quello di non effettuare operazioni.

Accesso a campi ed eventi

- I **campi**, gli *eventIn* e *eventOut* di un nodo `Script` sono accessibili dalla corrispondente classe java .
- Ciascun campo definito nel nodo `Script` è disponibile dalla classe java specificandone il nome.
- Si può leggere il valore del campo e/o modificarlo
- Gli *eventOut* definiti nel nodo `Script` possono essere letti, mentri gli *eventIn* possono essere scritti.

Accesso a campi ed eventi (i)

- L'accesso ai campi di un nodo **`Script`** può essere effettuato mediante uno dei tre tipi di metodi di classe **`Script`**
  - **`Field getField(String fieldName)`**
    - metodo che ottiene il riferimento al `campo` del nodo Script il cui nome è *fieldName*. Il valore ottenuto può essere convertito in un opportuno Java "Class Field"
  - **`Field getEventOut(String eventName)`**
    - metodo che ottiene il riferimento all' `eventOut` del nodo Script il cui nome è *eventName*. Il valore ottenuto può essere convertito in un opportuno Java "Class Field"
  - **`Field getEventIn(String eventName)`**
    - metodo che ottiene il riferimento all' `eventIn` del nodo Script il cui nome è *eventName*. Il valore ottenuto può essere convertito in un opportuno Java "Class Field"

Accesso a campi ed eventi (ii)

- Quando vengono invocati i metodi **`setValue()`**, **`setlValue()`**, **`addValue()`**, **`insertValue()`**, **`delete()`** o **`clear()`** in un oggetto **`Field`** ottenuto con il metodo **`getField()`**, si ha che il valore ottenuto viene memorizzato nel corrispondente campo del nodo VRML.
- Quando vengono invocati i metodi **`setValue()`**, **`setlValue()`**, **`addValue()`**, **`insertValue()`** in un oggetto **`Field`** ottenuto con il metodo **`getEventOut()`**, si ha che il valore specif cato come argomento genera un evento nella scena VRML. L'effetto dipende dall'istruzione ROUTE associata.

Accesso a campi ed eventi (ii)

- Quando vengono invocati i metodi **`setValue()`**, **`setlValue()`**, **`addValue()`**, **`insertValue()`** in un oggetto **`Field`** ottenuto con il metodo **`getEventIn()`**, si ha che il valore specif cato come argomento genera un evento nella scena VRML. L'effetto dipende dall'istruzione ROUTE associata.
- Nel caso dei metodi **`setlValue()`**, **`addValue()`**, **`insertValue()`**, **`delete()`** vengono acceduti tutti gli elementi del nodo VRML dopodiché il valore specif cato in argomento viene sottoposto alla specif ca operazione (aggiunta, cancellazione, etc) e salvato come elemento nel corrispondente campo del nodo VRML.
- Nel caso del metodo **`clear()`** vengono cancellati tutti i campi del nodo VRML.

Esempio

```
Script {
        url         "Example.class"
        eventIn     SFBool start
        eventOut    SFBool on
        field       SFBool state TRUE
    }
```

Example.java:

```
    import vrml.*;
    import vrml.field.*;
    import vrml.node.*;

    class Example extends Script {
       private SFBool state;
       private SFBool on;

       public void initialize(){
            state = (SFBool) getField("state");
            on = (SFBool) getEventOut("on");
        }

        public void processEvent(Event e) {
            if(state.getValue()==true){
                on.setValue(true); // set true to eventOut 'on'
                state.setValue(false);
            }
            else {
                on.setValue(false); // set false to eventOut 'on'
                state.setValue(true);
            }
        }
    }
```

Accesso di fields, eventIn, eventOut di altri nodi

- Se un programma **Script** ha accesso ad un nodo, ogni enventIn, eventOut o exposedField di quel nodo è accessibile usando i metodi getEventIn(), getEventOut() o getExposedField() def niti nella classe del nodo.
- Il modo tipico per un nodo **Script** di avere accesso ad un altro nodo VRML è quello di avere un campo SFNode che fornisce il riferimento ad un altro nodo.
- L'invio di e**ventOut** da script avviene def nendo il valore nell'eventOut dello script usando i metodi **setValue(), setlValue(), addValue()** o **insertValue()**
- L'invio di e**ventIn** da script avviene def nendo il valore nell'eventIn dello script usando i metodi **setValue(), setlValue(), addValue()** o **insertValue().**

Esempio

```
DEF SomeNode Transform {}
Script {
   field SFNode node USE SomeNode # SomeNode is a Transform node
   eventIn SFVec3f pos # new value to be inserted in
                       # SomeNode's translation field
   url "Example2.class"
  }
```

Example2.java:

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

public class Example2 extends Script {
   private SFNode node; // field
   private SFVec3f trans; // translation field captured from remote Transform node
   public void initialize(){
        node = (SFNode) getField("node");
   }

   public void processEvent(Event e){
        // gets the reference to the 'translation' field of the Transform node

        trans = (SFVec3f)((node.getValue()).getExposedField("translation");
        // reset translation to value given in Event e, which is eventIn pos
        // in the VRML Script node.
        trans.setValue((ConstSFVec3f)e.getValue());
    }
}
```

Classi e Metodi Exposed per Nodi e Campi

- Le classi Java per VRML sono definiti nei package **`vrml`**, **`vrml.node`** e **`vrml.field`**
- La classe `Field` estende per default la classe `Object` di Java, per cui `Field` ha la piena funzionalità della classe `Object`, compreso il metodo `getClass()`. Il resto del package definisce una classe `Const` read-only per ogni tipo di campo VRML, con un metodo `getValue()` per ogni classe; ed un'altra classe read/write, per ogni tipo di campo, con entrambi i metodi `getValue()` e `setValue()` per ogni classe. Il metodo getValue() converte il valore di tipo VRML nel valore di tipo Java. Il metodo setValue() converte il valore di tipo Java nel tipo VRML a lo assegna al corrispondente campo VRML. Gli errori sono gestiti da una serie di "throws exception"

Field Class e ConstField Class

- Ogni tipo di dato VRML ha il corrispondente dato Java

```
Class Field {
                }
```

- La classe Field è la radice di ogni tipo di campo. Questa classe ha due sottoclassi, una read-only ed una read-write

Field Class e ConstField Class

- Read-only class
  - Supporta il metodo *getValue()*. Alcune classi supportano metodi per ottenere valori da oggetti in modo conveniente:

    ConstSFBool, ConstSFColor, ConstMFColor, ConstSFFloat, ConstMFFloat, ConstSFImage, ConstSFInt32, ConstMFInt32, ConstSFNode, ConstMFNode, ConstSFRotation, ConstMFRotation, ConstSFString, ConstMFString, ConstSFVec2f, ConstMFVec2f, ConstSFVec3f, ConstMFVec3f, ConstSFTime, ConstMFTime
- Writable class
  - Supporta i metodi *getValue()* e *setValue()*. Se la classe inizia per MF, sono supportati i metodi *setIValue()*, *addValue()* *insertValue()*. Alcune classi supportano metodi per ottenere valori da oggetti in modo conveniente:
  - SFBool, SFColor, MFColor, SFFloat, MFFloat, SFImage, SFInt32, MFInt32, SFNode, MFNode, SFRotation, MFRotation, SFString, MFString, SFVec2f, MFVec2f, SFVec3f, MFVec3f, SFTime, MFTime

Metodi per ottenere e impostare valori

- **getSize**()
  E' il metodo per ottenere il numero di elementi di ogni field class a valore multiplo (MF class).
- **getValue**()
  E' il metodo per convertire un valore di tipo VRML in un valore di tipo Java ed ottenerlo.
- **get1Value**(int *index*)
  E' il metodo per convertire un valore di tipo VRML (*index*-th element) in un valore di tipo Java ed ottenerlo. L'indice del primo elemento è 0.
- **setValue**(*value*)
  E' il metodo per convertire un valore di tipo Java in un valore di tipo VRML e lo assegna al campo VRML.

Metodi per ottenere e impostare valori (i)

- **set1Value**(int *index*, *value*)
  E' il metodo per convertire un valore di tipo Java in uno di tipo VRML e lo assegna all'elemento *index-esimo*.
- **addValue**(*value*)
  E' il metodo per convertire un valore di tipo Java in uno di tipo VRML e lo assegna all'ultimo elemento.
- **insertValue**(int *index*, *value*)
  E' il metodo per convertire un valore di tipo Java in uno di tipo VRML e lo inserisce nella posizione *index*-esima.

Browser class

- Le interfacce pubbliche Java per la classe `Browser` sono le seguenti:

Return value	Method name
String	***getName****()*
String	***getVersion****()*
float	***getCurrentSpeed****()*
float	***getCurrentFrameRate****()*
String	***getWorldURL****()*
void	***replaceWorld****(Node[] nodes)*
Node[]	***createVrmlFromString****(String vrmlSyntax)*
void	***createVrmlFromURL****(String[] url, Node node, String event)*
void	***addRoute****(Node fromNode, String fromEventOut, NodetoNode, String toEventIn)*
void	***deleteRoute****(Node fromNode, String fromEventOut, Node toNode, String toEventIn)*
void	***loadURL****(String[] url, String[] parameter)*
void	***setDescription****(String description)*

Conversione VRML-Java

VRML type	Java type
SFString	*String*
SFFloat	*float*
MFString	*String[]*
MFNode	*Node[]*

Esempio

```
Script {
    field SFColor currentColor 0 0 0
    eventIn SFColor colorIn
    eventOut SFBool isRed
    url "Example3.class"
    }
```

Example3.java:

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

public class Example3 extends Script {
    // Declare field(s)
    private SFColor currentColor;

    // Declare eventOut
    private SFBool isRed;

    // buffer for SFColor.getValue().
    private float colorBuff[] = new float[3];
    public void initialize(){
          currentColor = (SFColor) getField("currentColor");
          isRed = (SFBool) getEventOut("isRed");
        }
    public void processEvent(Event e){
          // This method is called when a colorIn event is received
          currentColor.setValue((ConstSFColor)e.getValue());
       }
    public void eventsProcessed(){
          currentColor.getValue(colorBuff);
          if (colorBuff[0] >= 0.5) // if red is at or above 50%
               isRed.setValue(true);
    }
}
```

External Authoring Interface (EAI)

- Invece di fornire la connettività Java-VRML da dentro la scena VRML mediante il nodo Script, EAI definisce una interfaccia Java o Javascript per un applet esterno che comunica con un browser web esterno.
- applet EAI possono passare messaggi da e per scene VRML che siano *embedded* in pagine HTML.
- Gran parte dell'interfaccia browser è invariata, ma talvolta occorre una sintassi specifica per il passaggio degli eventi ed il controllo di flusso.
- Il maggior vantaggio è rappresentato dalla possibilità di scambio di messaggi tra ambiente Web e mondo VRML (per cui anche altri linguaggi, quali PHP, possono interagire con il mondo VRML)

```
package vrml;

public abstract class Field implements Cloneable
{
   public Object clone();
}

public abstract class ConstField extends Field
{
}

public class ConstMField extends ConstField
{
   public int getSize();
}

public class MField extends Field
{
   public int getSize();
   public void clear();
   public void delete(int index);
}

public class Event implements Cloneable {
  public String getName();
  public double getTimeStamp();
  public ConstField getValue();
  public Object clone();
}

public class Browser {
  // Browser interface
  public String getName();
  public String getVersion();

  public float getCurrentSpeed();

  public float getCurrentFrameRate();
 public String getWorldURL();
  public void replaceWorld(Node[] nodes);

  public Node[] createVrmlFromString(String
   vrmlSyntax)
    throws InvalidVRMLSyntaxException;

  public void createVrmlFromURL(String[]
   url, Node node, String event)
    throws InvalidVRMLSyntaxException;

  public void addRoute(Node fromNode, String
   fromEventOut,
    Node toNode, String toEventIn);
  public void deleteRoute(Node fromNode,
   String fromEventOut,
    Node toNode, String toEventIn);

  public void loadURL(String[] url, String[]
   parameter)
    throws InvalidVRMLSyntaxException;
  public void setDescription(String
   description);
}

//
// This is the general BaseNode class
//
public abstract class BaseNode
{
  // Returns the type of the node.  If the
   node is a prototype
  //   it returns the name of the prototype.
  public String getType();

  // Get the Browser that this node is
   contained in.
  public Browser getBrowser();
}
```

```
package vrml.node;
//
// This is the general Node class
//
public abstract class Node extends BaseNode
{
   // Get an EventIn by name. Return value is write-only.
   // Throws an InvalidEventInException if eventInName isn't a valid
   // eventIn name for a node of this type.
   public final Field getEventIn(String eventInName);

   // Get an EventOut by name. Return value is read-only.
   // Throws an InvalidEventOutException if eventOutName isn't a valid
   // eventOut name for a node of this type.
   public final ConstField getEventOut(String eventOutName);// Get an exposed field by name.

   // Throws an InvalidExposedFieldException if exposedFieldName isn't a valid
   // exposedField name for a node of this type.
   public final Field getExposedField(String exposedFieldName);

   public String toString(); // This overrides a method in Object
}
// This is the general Script class, to be subclassed by all scripts.
// Note that the provided methods allow the script author to explicitly
// throw tailored exceptions in case something goes wrong in the
// script.
//
public abstract class Script extends BaseNode
{
   // This method is called before any event is generated public void initialize();
   // Get a Field by name.
   // Throws an InvalidFieldException if fieldName isn't a valid
   // field name for a node of this type.
   protected final Field getField(String fieldName);

   // Get an EventOut by name.
   // Throws an InvalidEventOutException if eventOutName isn't a valid
   // eventOut name for a node of this type.
   protected final Field getEventOut(String eventOutName);
```

```
// Get an EventIn by name.
   // Throws an InvalidEventInException if eventInName isn't a valid
   // eventIn name for a node of this type.
   protected final Field getEventIn(String eventInName);

   // processEvents() is called automatically when the script receives
   // some set of events. It shall not be called directly except by its subclass.
   // count indicates the number of events delivered.
   public void processEvents(int count, Event events[]);

   // processEvent() is called automatically when the script receives
   // an event.
   public void processEvent(Event event);

   // eventsProcessed() is called after every invocation of processEvents().
   public void eventsProcessed()

   // shutdown() is called when this Script node is deleted.
   public void shutdown();

   public String toString(); // This overrides a method in Object
```

Java Resources

"The Java Virtual Machine Specif cation" by Tim Lindhold and Frank Yellin, Addison Wesley, Reading Massachusetts, 1996, ISBN 0-201-63452-X. **http://java.sun.com/docs/books/vmspec/index.html**

http://www.graphcomp.com/info/specs/sgi/vrml/spec/part1/java.html

http://eureka.lucia.it/vrml/tutorial