Open Inventor
5 - Texture Mapping

Claudio Baldazzi *Maurizio Patrignani*

Università degli Studi di Roma Tre
Dip. di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

30 aprile 1998

1 Introduzione

Questi appunti introducono all'uso del *texture mapping* che permette di aggiungere dettagli realistici agli oggetti della scena. In Inventor si crea un'immagine 2D chiamata *texture map* che si applica sulla supeficie di un oggetto solido 3D. L'immagine 2D rettangolare è modificata per essere applicata sugli oggetti 3D secondo certe regole da impostare nel grafo della scena. Le informazioni, le figure e i listati contenuti in questo documento, sono tratti, salvo diversa indicazione, dal libro "The Inventor Mentor" di Josie Wernecke edito dalla Addison Wesley.

2 Overview e Texture Mapping di default

Texture Map Un'immagine 2D, chiamata *Texture Map*, è un array bidimensionale contenente informazioni sui pixel dell'immagine stessa. Inventor, come OpenGL, usa la lettera s per la coordinata texture orizzontale e la lettera t per quella verticale ed entrambe variano nel range $[0.0, 1.0]$. Una *Texture Map* è quindi un quadrato con lato 1. Le coordinate texture sono assegnate ad ogni vertice di un poligono dell'oggetto, esplicitamente dal programma utente oppure automaticamente da Inventor. Se i pixel nella texture non coincidono esattamente con i pixel del poligono, Inventor attua un processo di filtraggio che deforma la texture map per poter assegnare i pixel della texture map all'oggetto. La texture map è letta da file oppure da memoria.

Inventor Texture Mapping Per effettuare il texture mapping Inventor mette a disposizione i seguenti nodi:

- **SoTexture2** Nel nodo viene specificata la texture map da usare e alcuni parametri sulla modalità con cui viene effettuato il texture mapping. Si tornerà in seguito su questo nodo, dopo aver chiarito i vari modelli di mapping.

- **SoTextureCoordinate2** In questo nodo viene esplicitamente definito l'insieme delle coordinate texture da associare sequenzialmente ai vertici degli oggetti seguenti nel grafo della scena.

- **SoTextureCoordinateBinding** Il nodo specifica come le coordinate texture correnti sono legate ai nodi rappresentanti gli oggetti.

- **SoTextureCoordinatePlane** e **SoTextureCoordinateEnvironment** Sono due nodi che permettono di usare una funzione per il texture mapping.

- **SoTextureCoordinateDefault** Questo nodo elimina ogni precedente texture mapping così che tutti gli oggetti, seguenti nel grafo della scena, usino le loro coordinate texture di default.

- **SoTexture2Transform** Tale nodo permette di definire una trasformazione (ovviamente 2D) per la texture map.

Il nodo **SoComplexity** ha un campo chiamato **textureQuality** che può assumere valori da 0.0 a 1.0. Un valore pari a 0.0 rende il rendering molto veloce, ma il texture ha bassissima definizione. Al contrario un valore pari a 1.0 rende il texture molto realistico ma rallenta il rendering della scena. Il valore di default usato da Inventor è 0.5.

Texture Mapping di default Per effettuare un semplice mapping, basta avere un'immagine 2D su file e usare il nodo **SoTexture2**. L'esempio seguente mostra come impostare il su un cubo usando tutti parametri di default di Inventor ed il file `brick.1.rgb`.

```
/*--------------------------------------------------------------
 *  This is an example from the Inventor Mentor,
 *  chapter 7, example 1.
 *
 *  This example displays a textured cube (default
 *  texture coords).
 *------------------------------------------------------------*/

#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoTexture2.h>

int main( int, char** argv )
{
   Widget myWindow = SoXt::init(argv[0]);
   if(myWindow == 0) return 1;

   SoSeparator* root = new SoSeparator;
   root->ref();

   // Choose a texture
   SoTexture2* rock = new SoTexture2;
   root->addChild(rock);
   rock->filename.setValue("brick.1.rgb");

   // Make a cube
   root->addChild(new SoCube);
```

```
    SoXtExaminerViewer* myViewer =
            new SoXtExaminerViewer(myWindow);
    myViewer->setSceneGraph(root);
    myViewer->setTitle("Default Texture Coords");

    // In Inventor 2.1, if the machine does not have hardware texture
    // mapping, we must override the default drawStyle to display textures.
    myViewer->setDrawStyle(SoXtViewer::STILL, SoXtViewer::VIEW_AS_IS);

    myViewer->show();

    SoXt::show(myWindow);
    SoXt::mainLoop();

    return 0;
}
```

Per un uso semplice del mapping è bastato creare un file 2D che contiene la texture map e farlo leggere dal nodo **SoTexture2**. A tutti gli oggetti che vengono inseriti dopo tale nodo verrà applicato il texture mapping con tutti i parametri di default impostati da Inventor.

3 Texture Mapping

Concetti di base In questa sezione si analizzeranno alcuni modi per cambiare il mapping di una texture map su di un oggetto. Quello che si può cambiare con Inventor è:

- il modo con cui la texture map avvolge l'oggetto;
- il modo con cui la texture map interagisce con il colore dell'oggetto.

Avvolgere una texture intorno all'oggetto Il valore delle coordinate della texture map varia, lungo tutte e due le dimensioni, da 0.0 a 1.0. Se il poligono su cui applicare la texture ha le coordinate in un diverso range in una delle direzioni, si hanno due scelte possibili:

- **REPEAT**: la texture viene ripetuta tante volte quanto è necessario per coprire completamente la faccia.
- **CLAMP**: l'ultima riga di pixel della texture map viene usata per riempire il resto del poligono.

Interazione di una texture con il colore dell'oggetto Inventor usa tre tipi di modelli per l'interazione con il colore dell'oggetto:

- **MODULATE**: moltiplica il colore dell'oggetto per il colore della texture (default). Se la texture ha una componente *alpha*, tale valore modula la trasparenza dell'oggetto.
- **DECAL**: rimpiazza il colore dell'oggetto con quello della texture map. Se la texture ha una componente *alpha*, essa specifica la trasparenza della texture, permettendo al colore dell'oggetto di essere visto in trasparenza.

- **BLEND**: l'intensità della texture viene combinata tra il colore dell'oggetto e uno specifico colore costante chiamato *blend color*.

Il modello **MODULATE** può essere usato con qualsiasi tipo di file texture. Il modello **BLEND** può essere usato con i file ad una o due componenti ed il modello **DECAL** è usato con i file a tre o quattro componenti (si veda in seguito).

4 Memorizzazione di una texture map

Una texture map può essere letta da file o da memoria. Il formato dei file immagine supportati sono molteplici e dipendono dal sistema operativo, ad esempio in Windows vengono supportati i file bmp, jpeg e gif.

Le texture map possono essere memorizzate anche come oggetti di tipo **SoSFImage**. Una texture map di tipo **SoSFImage** viene poi specificarla nel campo **image** del nodo **SoTexture2**. Una texture map di questo tipo può contenere da una a quattro componenti per pixel.

- *Texture map ad una componente* — contiene solo il valore di intensità. Alcune volte questa texture map viene chiamata *intensity map*.
- *Texture map a 2 componenti* — contiene il valore di intensità e il valore *alpha* per la trasparenza.
- *Texture map a 3 componenti* — contiene un'immagine RGB, quindi le tre componenti sono il colore RGB del pixel.
- *Texture map a 4 componenti* — contiene un'immagine RGB con in più il fattore *alpha* per la trasparenza.

L'immagine può risiedere, come si è detto, su file oppure si può usare il metodo setValue() del nodo *SoTexture2* per leggere l'immagine dalla memoria. Il metodo vuole come argomenti la larghezza e l'altezza della texture, il numero di componenti ed un array di **unsigned char** con valori da 0 a 255 che rappresentano le componenti dell'immagine. Ad esempio per una texture ad una componente si può scrivere:

```
SoTexture2* textureNode = new SoTexture2;

unsigned char image[] = {
  255, 0,
  0, 255,
  255, 0
};

textureNode->image.setValue( SbVec2s(3,2), 1, image );
```

5 Nodo SoTexture2

I campi del nodo **SoTexture2** sono:

- **filename** (SoSFName) In questo campo si specifica il nome del file contenente la texture map. I formati del file letti sono molteplici.

- **image** (SoSFImage) Nel campo viene specificato il colore e il numero di pixel di una texture map.
- **wrapS** e **wrapT** (SoSFEnum) In questi campi viene specificato come l'immagine é mappata nelle direzioni s (orizzontale) e t (verticale). I valori possibili sono `REPEAT` e `CLAMP`.
- **model** (SoSFEnum) Il campo specifica il tipo di modello di interazione con il colore dell'oggetto. I valori possibili sono `MODULATE`, `DECAL` e `BLEND`.
- **blendColor** (SoSFColor): specifica il colore di blend nel modello `BLEND`.

Inoltre il nodo **SoComplexity** ha un campo chiamato **textureQuality** che può assumere valori da 0.0 a 1.0. Un valore pari a 0.0 rende il rendering molto veloce, ma il texture ha bassissima definizione. Al contrario un valore pari a 1.0 rende il texture molto realistico ma rallenta il rendering della scena. Il valore di default usato da Inventor è 0.5.

Nodo SoTexture2Transform Si può effettuare una trasformazione sulla texture map inserendo nella scena un nodo di tipo **SoTexture2Transform**. Le trasformazioni memorizzate in questo nodo vengono applicate alle coordinate di texture s e t ed hanno un effetto cumulativo quando si hanno più nodi di questo tipo. I campi del nodo sono simili a quelli di un nodo **SoTransform**, ma ora la dimensione dello spazio è 2 e non 3. Si avranno quindi i campi:

- **translation** in cui si specifica una traslazione delle coordinate texture dell'oggetto.
- **rotation** in cui si inserisce una rotazione, in radianti, delle coordinate texture dell'oggetto.
- **scaleFactor** che serve a scalare la texture map sull'oggetto. Le coordinate s e t vengono moltiplicate per il fattore di scala. In questo modo un fattore di scala di (2.0,2.0) fa in modo che la texture map appaia più piccola sull'oggetto, mentre un fattore di scala di (0.5,0.5) fa apparire la texture map ingrandita.
- **center** in cui si specifica il cantro di scalamenti e rotazioni. Il default é [0.0,0.0], l'angolo in basso a sinistra della texture map.

6 Mapping della texture map sull'oggetto

Esistono in Inventor 3 tecniche per mappare una texture map 2D su di un oggetto 3D:

1. Usare le coordinate texture di default (**SoTextureCoordinateDefault**).
2. Per gli oggetti derivati dal nodo **SoVertexShape** si possono specificare esplicitamente le coordinate di texture attraverso il nodo **SoTextureCoordinate2** in cui vanno specificate le coordinate di texture per ogni vertice dell'oggetto.

3. Usare uno dei nodi funzioni-texture che mappano la texture sull'oggetto attraverso un algoritmo: **SoTextureCoordinatePlane** e **SoTexture-CoordinateEnvironment**.

Le tecniche 1 e 3 sono automaticamente gestite da Inventor e, quindi, facili da usare, la tecnica 2 richiede la generazione esplicita delle coordinate.

Texture mapping di default Inventor usa la stessa tecnica per generare le coordinate di texture di default per ogni oggetto derivato da **SoVertexShape**. Inventor esegue il mapping nel seguente modo:

1. Viene calcolato il box di contenimento dell'oggetto 3D.

2. Il lato più lungo del box di contenimento viene usato come asse orizzontale (s) della texture. Il lato successivo viene invece usato come asse verticale (t) della texture.

3. Il valore di s sarà nel range $[0.0, 1.0]$ da un estremo all'altro del box di contenimento, mentre quello di t andrà da 0.0 a n, con n pari al rapporto tra il secondo lato più lungo e il lato più lungo scelto per le s.

L'effetto è quello che la texture viene applicata al lato più lungo del box di contenimento senza distorsione, mentre sugli altri lati si puó avere distorsione dell'immagine.

Per oggetti non derivati da **SoVertexShape** il mapping viene generato diversamente a seconda dell'oggetto. (Si lascia per esercizio al lettore la generazione del default mapping per gli oggetti **SoSphere**, **SoCube**, **SoCylinder**, **SoCone**, **SoNurbsSurface** e **SoText3**.

Texture Mapping con specificazione delle coordinate esplicita Per specificare esplicitamente le coordinate di texture per ogni vertice di un oggetto, derivato da **SoVertexShape**, si usa il nodo **SoTextureCoordinate2**, in cui si specifica l'insieme delle coordinate di texture da applicare a ogni vertice dell'oggetto. Le coordinate vanno specificate a coppia, (s,t).

```
/*--------------------------------------------------------------
 *  This is an example from the Inventor Mentor
 *  chapter 7, example 2.
 *
 *  This example illustrates using texture coordindates on
 *  a Face Set.
 *------------------------------------------------------------*/

#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoFaceSet.h>
#include <Inventor/nodes/SoNormal.h>
#include <Inventor/nodes/SoNormalBinding.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoTexture2.h>
#include <Inventor/nodes/SoTextureCoordinate2.h>
#include <Inventor/nodes/SoTextureCoordinateBinding.h>

int main( int ,char** argv )
{
```

```
Widget myWindow = SoXt::init(argv[0]);
if(myWindow == 0) return 1;

SoSeparator* root = new SoSeparator;
root->ref();

// Choose a texture
SoTexture2* brick = new SoTexture2;
root->addChild(brick);
brick->filename.setValue("brick.1.rgb");

// Using the new SoVertexProperty node is more efficient
SoVertexProperty* myVertexProperty = new SoVertexProperty;

// Define the square's spatial coordinates
myVertexProperty->vertex.set1Value(0, SbVec3f(-3, -3, 0));
myVertexProperty->vertex.set1Value(1, SbVec3f( 3, -3, 0));
myVertexProperty->vertex.set1Value(2, SbVec3f( 3,  3, 0));
myVertexProperty->vertex.set1Value(3, SbVec3f(-3,  3, 0));

// Define the square's normal
myVertexProperty->normal.set1Value(0, SbVec3f(0, 0, 1));

// Define the square's texture coordinates
myVertexProperty->texCoord.set1Value(0, SbVec2f(0, 0));
myVertexProperty->texCoord.set1Value(1, SbVec2f(1, 0));
myVertexProperty->texCoord.set1Value(2, SbVec2f(1, 1));
myVertexProperty->texCoord.set1Value(3, SbVec2f(0, 1));

// SoTextureCoordinateBinding node is now obsolete--in Inventor 2.1,
// texture coordinates will always be generated if none are
// provided.
/*
  SoTextureCoordinateBinding* tBind =
         new SoTextureCoordinateBinding;
  root->addChild(tBind);
  tBind->value = SoTextureCoordinateBinding::PER_VERTEX;
*/
// Define normal binding
myVertexProperty->normalBinding = SoNormalBinding::OVERALL;

// Define a FaceSet
SoFaceSet* myFaceSet = new SoFaceSet;
root->addChild(myFaceSet);
myFaceSet->numVertices.set1Value(0, 4);

myFaceSet->vertexProperty.setValue(myVertexProperty);

SoXtExaminerViewer* myViewer =
         new SoXtExaminerViewer(myWindow);
myViewer->setSceneGraph(root);
myViewer->setTitle("Texture Coordinates");

// In Inventor 2.1, if the machine does not have hardware texture
// mapping, we must override the default drawStyle to display textures.
myViewer->setDrawStyle(SoXtViewer::STILL, SoXtViewer::VIEW_AS_IS);

myViewer->show();

SoXt::show(myWindow);
SoXt::mainLoop();
```

```
    return 0;
}
```

Nel codice, al posto di usare i nodi **SoCoordinate3** e **SoTextureCoordinate2**, viene usato un unico nodo **SoVertexProperty** i cui campi **vertex** e **texCoord** sostituiscono la definizione dei due nodi sopramenzionati. Dopo aver dichiarato quindi un oggetto di questo tipo, si specifica l'oggetto 3D attraverso il seguente codice:

```
// Define the square's spatial coordinates
myVertexProperty->vertex.set1Value(0, SbVec3f(-3, -3, 0));
myVertexProperty->vertex.set1Value(1, SbVec3f( 3, -3, 0));
myVertexProperty->vertex.set1Value(2, SbVec3f( 3,  3, 0));
myVertexProperty->vertex.set1Value(3, SbVec3f(-3,  3, 0));

// Define the square's normal
myVertexProperty->normal.set1Value(0, SbVec3f(0, 0, 1));
```

e dopo per ogni vertice vengono specificate le coordinate di texture:

```
// Define the square's texture coordinates
myVertexProperty->texCoord.set1Value(0, SbVec2f(0, 0));
myVertexProperty->texCoord.set1Value(1, SbVec2f(1, 0));
myVertexProperty->texCoord.set1Value(2, SbVec2f(1, 1));
myVertexProperty->texCoord.set1Value(3, SbVec2f(0, 1));
```

Texture mapping usando nodi funzioni-texture Un nodo funzione-texture effettua il mapping definendo le coordinate texture per un oggetto basandosi sulla posizione di ogni vertice. Ogni funzione usa un diverso algoritmo per calcolare le coordinate di texture. Queste funzioni permettono di specificare il mapping in maniera generale, senza richiedere la definizione esplicita delle coordinate texture, e ignorano le coordinate texture correnti specificate dai nodi **SoTextureCoordinate2**.

Inventor include 2 funzioni:

- **SoTextureCoordinatePlane** che proietta la texture map sull'oggetto.
- **SoTextureCoordinateEnvironment** la funzione di questo nodo fa in modo che l'oggetto a cui è applicata la texture map appare come se riflettesse l'ambiente che lo circonda. La texture map deve rappresentare l'ambiente circostante.

SoTextureCoordinatePlane Questa è probabilmente la funzione più usata per il mapping, proietta il piano della texture map sull'oggetto. Si possono definire le direzioni di s e t. La coordinata di texture s è poi definita dalla seguente equazione, in cui *coord* è la coordinata dell'oggetto:

$$s = \left(\frac{1}{directionS}\right) * coord.$$

I campi del nodo sono:

- **directionS** — direzione di proiezione della coordinata s (default = [1.0,0.0,0.0]).
- **directionT** — direzione di proiezione della coordinata t (default = [0.0,1.0,0.0]).

Il modulo del vettore direzione è pari all'intervallo di ripetizione della texture.

L'esempio seguente disegna tre sfere con una texture map. La texture map é applicata sulle tre sfere cambiando il modulo del vettore in modo da ripeterla sull'oggetto diverse volte.

```
/*--------------------------------------------------------------
 *  This is an example from the Inventor Mentor,
 *  chapter 7, example 3.
 *
 *  This example illustrates using texture functions to
 *  generate texture coordinates on a sphere.
 *  It draws three texture mapped spheres, each with a
 *  different repeat frequency as defined by the fields of the
 *  SoTextureCoordinatePlane node.
 *------------------------------------------------------------*/

#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/nodes/SoTexture2.h>
#include <Inventor/nodes/SoTexture2Transform.h>
#include <Inventor/nodes/SoTextureCoordinatePlane.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>

int main( int, char** argv )
{
   Widget myWindow = SoXt::init(argv[0]);
   if(myWindow == 0) return 1;

   SoSeparator* root = new SoSeparator;
   root->ref();

   // Choose a texture.
   SoTexture2* faceTexture = new SoTexture2;
   root->addChild(faceTexture);
   faceTexture->filename.setValue("sillyFace.rgb");

   // Make the diffuse color pure white
   SoMaterial* myMaterial = new SoMaterial;
   myMaterial->diffuseColor.setValue(1,1,1);
   root->addChild(myMaterial);

   // This texture2Transform centers the texture about (0,0,0)
   SoTexture2Transform* myTexXf = new SoTexture2Transform;
   myTexXf->translation.setValue(.5,.5);
   root->addChild(myTexXf);

   // Define a texture coordinate plane node.  This one will
   // repeat with a frequency of two times per unit length.
   // Add a sphere for it to affect.
   SoTextureCoordinatePlane* texPlane1 = new SoTextureCoordinatePlane;
   texPlane1->directionS.setValue(SbVec3f(2,0,0));
   texPlane1->directionT.setValue(SbVec3f(0,2,0));
   root->addChild(texPlane1);
   root->addChild(new SoSphere);

   // A translation node for spacing the three spheres.
   SoTranslation* myTranslation = new SoTranslation;
   myTranslation->translation.setValue(2.5,0,0);
```

```
   // Create a second sphere with a repeat frequency of 1.
   SoTextureCoordinatePlane* texPlane2 = new SoTextureCoordinatePlane;
   texPlane2->directionS.setValue(SbVec3f(1,0,0));
   texPlane2->directionT.setValue(SbVec3f(0,1,0));
   root->addChild(myTranslation);
   root->addChild(texPlane2);
   root->addChild(new SoSphere);

   // The third sphere has a repeat frequency of .5
   SoTextureCoordinatePlane* texPlane3 = new SoTextureCoordinatePlane;
   texPlane3->directionS.setValue(SbVec3f(.5,0,0));
   texPlane3->directionT.setValue(SbVec3f(0,.5,0));
   root->addChild(myTranslation);
   root->addChild(texPlane3);
   root->addChild(new SoSphere);

   SoXtExaminerViewer* myViewer =
            new SoXtExaminerViewer(myWindow);
   myViewer->setSceneGraph(root);
   myViewer->setTitle("Texture Coordinate Plane");

   // In Inventor 2.1, if the machine does not have hardware texture
   // mapping, we must override the default drawStyle to display textures.
   myViewer->setDrawStyle(SoXtViewer::STILL, SoXtViewer::VIEW_AS_IS);

   myViewer->show();

   SoXt::show(myWindow);
   SoXt::mainLoop();

   return 0;
}
```

SoTextureCoordinateEnvironment Questo nodo specifica che gli oggetti seguenti devono riflettere il loro ambiente circostante (come gli ornamenti di natale). Per migliori risultati conviene che la texture map sia una mappa sferica di riflessione (vedi OpenGL).

7 FAQ

D.: Ho fatto un programma per disegnare dei poligoni concavi, in particolare il frammento seguente di codice disegna un poligono a forma di “L”, ma il visualizzatore di Inventor fa strane cose e non visualizza la “L” ma un’altra figura.

```
static float vertices[6][3] =
    { {0,2,0},{0,2,1},{0,.5,1},{0,.5,2},{0,0,2},{0,0,0} };

static int numerovertici[1] = {6};

SoSeparator* mkbraccio()
{
    SoSeparator* braccio = new SoSeparator;
    braccio->ref();

    SoCoordinate3* myCoord = new SoCoordinate3;
    myCoord->point.setValues(0, 6, vertices);
    braccio->addChild(myCoord);
```

```
    SoFaceSet* myFace = new SoFaceSet;
    myFace->numVertices.setValues(0, 1, numerovertici);
    braccio->addChild(myFace);

    braccio->unrefNoDelete();
    return braccio;
}

int main( int, char** argv )
{
    // Initialize Inventor and Xt
    Widget myWindow = SoXt::init(argv[0]);
    if(myWindow == 0) return 1;

    SoSeparator* root = new SoSeparator;
    SoMaterial* myMat = new SoMaterial;
    myMat->diffuseColor.setValue(1,0,0);

    root->ref();

    root->addChild(myMat);
    root->addChild(mkbraccio());

    SoXtExaminerViewer *myViewer = new SoXtExaminerViewer(myWindow);
    myViewer->setSceneGraph(root);
    myViewer->setTitle("SoFaceSet Errata");
    myViewer->show();

    SoXt::show(myWindow);
    SoXt::mainLoop();

    return 0;
}
```

R.: Il problema sta nel fatto che Inventor di default "crede" che i poligoni, quindi gli oggetti **SoFaceSet**, siano convessi e non concavi come la "L". Esiste un nodo in Inventor chiamato **SoShapeHints** che indica ad Inventor come disegnare i poligoni. I campi di questo nodo sono:

- **vertexOrdering** che puo' assumere i valori **SoShapeHints::UNKNOWN_ORDERING** (il default di Inventor), **SoShapeHints::CLOCKWISE** e **SoShapeHints::COUNTERCLOCKWISE** a seconda dell'ordine con cui si inseriscono i vertici dei poligoni nel nodo **SoCoordinate3**.
- **shapeType** che puo' assumere i valori **SoShapeHints::SOLID** e **SoShapeHints::UNKNOWN_SHAPE_TYPE** (il default) ed indica che l'oggetto è sicuramente chiuso oppure no.
- **faceType** che puo' assumere i valori **SoShapeHints::UNKNOWN_FACE_TYPE** e **SoShapeHints::CONVEX** (il default di Inventor). Di default quindi Inventor assume che tutte le facce date siano convesse.
- **creaseAngle**, campo interno per generare automaticamente le normali.

Di seguito c'è il codice (solo la funzione main) che disegna correttamente il poligono inserendo il nodo **SoShapeHints** prima del nodo **SoFaceSet**.

```
int main( int, char** argv )
{
```

```
    // Initialize Inventor and Xt
    Widget myWindow = SoXt::init(argv[0]);
    if(myWindow == 0) return 1;

    SoSeparator *root = new SoSeparator;
    SoMaterial* myMat = new SoMaterial;
    myMat->diffuseColor.setValue(1,0,0);

    // SoShapeHints
    // the default of faceType is SoShapeHints::CONVEX
    SoShapeHints* ssh = new SoShapeHints;
    ssh->faceType = SoShapeHints::UNKNOWN_FACE_TYPE;    // no convex polygons
    //

    root->ref();

    root->addChild(ssh);    // add SoShapeHints
    root->addChild(myMat);
    root->addChild(mkbraccio());

    SoXtExaminerViewer *myViewer = new SoXtExaminerViewer(myWindow);
    myViewer->setSceneGraph(root);
    myViewer->setTitle("SoFaceSet OK");
    myViewer->show();
    SoXt::show(myWindow);
    SoXt::mainLoop();

    return 0;
}
```