LAB 2 – Navigazione interattiva in scena con modelli geometrici 3D

Fondamenti di Computer Graphics M
Erika Gardini – Ingegneria Informatica A.A. 2016/2017

Per prima cosa si è caricato ed eseguito il programma. A default avviene il caricamento del modello geometrico di tipo mesh in formato .m della figura pig. Cambiando il nome del file .m è possibile caricare nella finestra altre figure con lo stesso formato.
Per la visualizzazione di superfici quadriche, invece, si può utilizzare la libreria GLU che mette a disposizione una serie di comandi. Per esempio i comandi sotto indicati producono rispettivamente i seguenti risultati:

```
gluCylinder(myReusableQuadric, 0.5, 0.2, 0.5, 12, 12);
glutSolidCube(1);
gluSphere(myReusableQuadric, 1.0, 12, 12);
gluDisk(myReusableQuadric, 0.5, 1.0, 10, 10);
gluPartialDisk( myReusableQuadric, 0.5, 1.0, 10, 10, 0.0, 45.0);
```

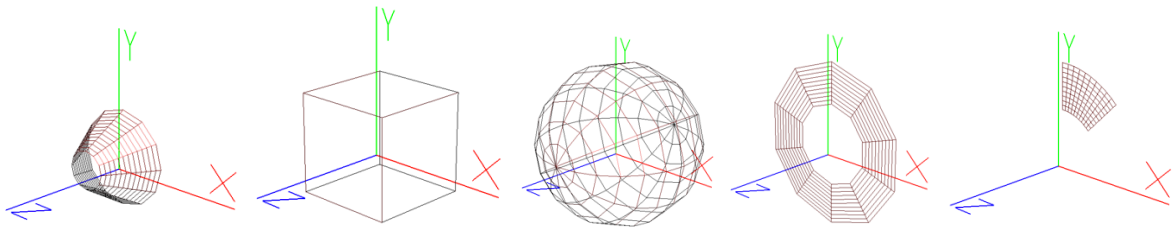


Figura 1: quadriche con libreria GLU

Per la visualizzazione dei modelli poligonali a mesh occorre per prima cosa creare la display list mesh mediante i seguenti comandi:

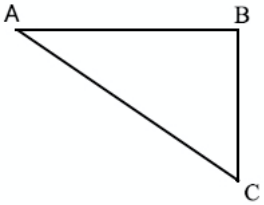
```
listname=glGenLists(1);  //ID della lista e' listname
glNewList(listname,GL_COMPILE);
...
glEndList();
```

la *listname* contiene le informazioni sui vertici della mesh da disegnare (caricati dal file).
Una volta creata la display list mesh, per la visualizzazione del modello poligonale a mesh occorre invocare nel metodo display il seguente comando:

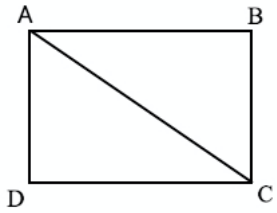
```
glCallList(listname);
```

Una volta analizzate le funzionalità già presenti nell'applicazione fornite, è stato possibile passare alla prima funzionalità da realizzare. In particolare, veniva richiesto di calcolare e memorizzare le normali ai vertici per i modelli mesh poligonali. Queste consentono di visualizzare le zone in ombra e le zone illuminate della figura mostrata in esecuzione.
In particolare, un punto è tanto più illuminato quanto piccolo è l'angolo formato dalla normale a quel punto con la direzione della luce.
La normale al vertice A si ottiene eseguendo i passi sotto indicati:

- ottenere il vettore B-A sottraendo il punto B al punto A;
- ottenere il vettore C-A sottraendo il punto C al punto A;
- calcolare il cross product fra i due vettori trovati, così da calcolare il vettore normale;
- normalizzare, così da ottenere il versore normale alla faccia ABC.



I poligoni mostrati in esecuzione, però sono costituiti da più triangoli fra loro adiacenti. La normale effettiva ad un vertice è quindi data dalla somma di tutti i contributi. Nell'esempio riportato, quindi, la normale al vertice A è calcolata in questo modo:



- ottenere il vettore B-A sottraendo il punto B al punto A;
- ottenere il vettore C-A sottraendo il punto C al punto A;
- calcolare il cross product fra i vettori trovati, così da calcolare il vettore normale (primo contributo);
- ottenere il vettore C-A sottraendo il punto C al punto A;
- ottenere il punto D-A sottraendo il punto D al punto A;
- calcolare il cross product fra i vettori trovati così da calcolare il vettore normale (secondo contributo);
- sommare fra loro i vari contributi e normalizzare.

Applicando questi passaggi a tutti i punti i vertici costituenti la mesh poligonale è possibile ottenere il seguente risultato:

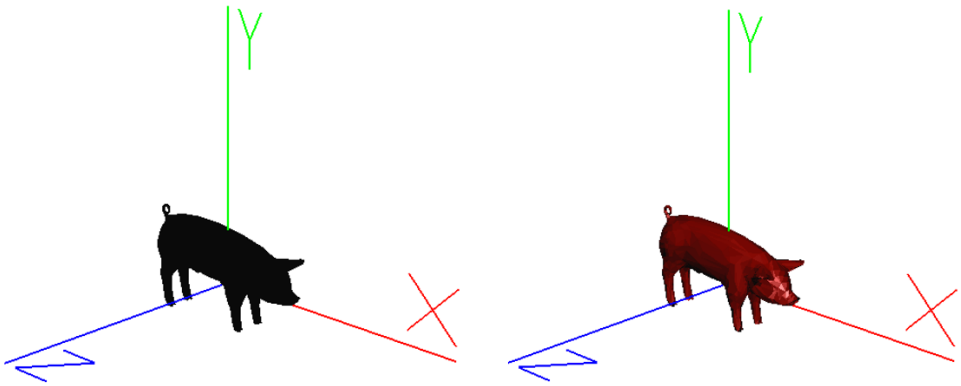


Figura 2: aspetto della mesh poligonale con/senza luce

Affinché il risultato sia visibile è necessario assegnare una posizione alla luce ed abilitarla mediante i seguenti comandi:

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

Completata la prima parte, è possibile procedere con l'inserimento delle prossime funzionalità richieste. In particolare, nella seconda parte dell'esercitazione veniva richiesto di implementare le seguenti funzionalità:

1) ***change eye point – modificare le coordinate del punto di vista alla pressione dei tasti x, X, y, Y, z, Z***
   Per realizzare la seguente funzionalità è stato necessario aggiungere, nel metodo keyboard dell'applicazione, nel caso in cui viene selezionata la modalità change eye pos, la posizione attuale del punto di vista (ossia camE, cioè la posizione della camera) e lo step di cui occorre spostarsi alla pressione dei tasti x, X, y, Y, z, Z.
   Il risultato ottenuto è il seguente:

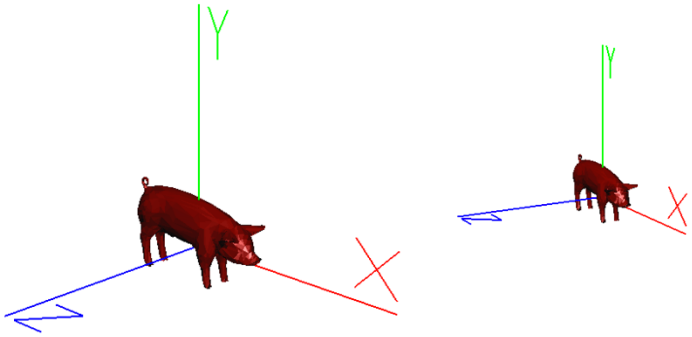


Figura 3: spostamento del punto di vista rispetto x

2) ***rotate model – rotazione del modello alla pressione dei tasti x, X, y, Y, z, Z***

Per realizzare la seguente funzionalità è stato necessario aggiungere, nel metodo keyboard dell'applicazione, nel caso in cui viene selezionata la modalità rotate model, la posizione attuale del modello (ossia l'angolo di rotazione attuale rispetto alla posizione di partenza) e lo step di cui occorre ruotare alla pressione dei tasti x, X, y, Y, z, Z.

Il risultato ottenuto è il seguente:

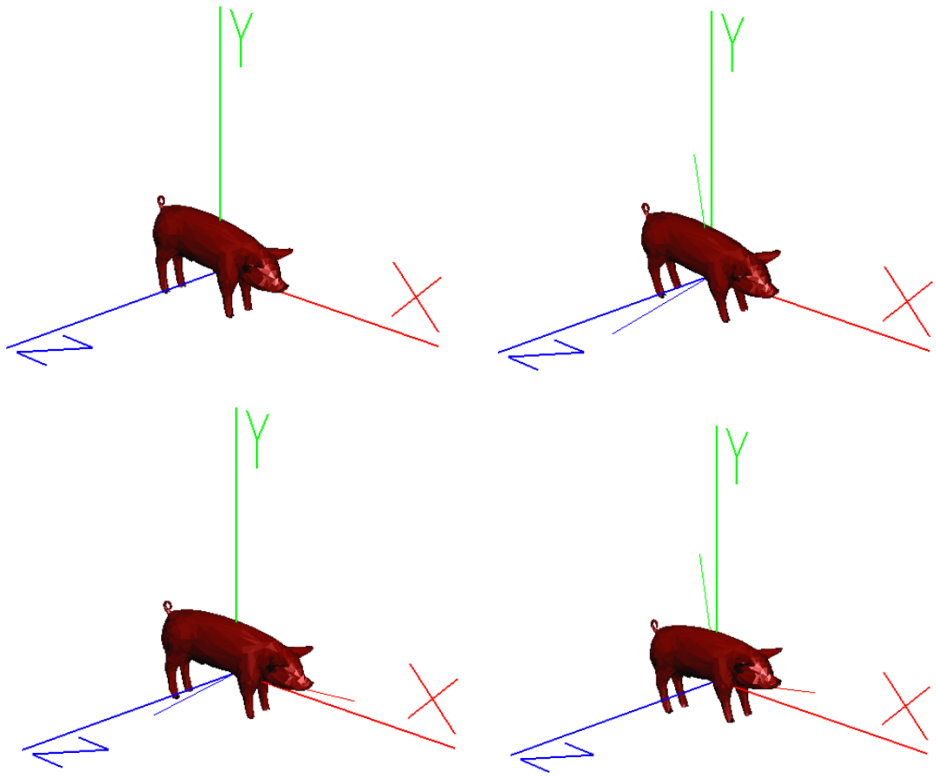


Figura 4: rotazione di 10 step rispetto x, y, z

3) ***zoom in/out – modifica dell'angolo (field of view) al vertice della piramide di vista alla pressione dei tasti f, F***

Per realizzare la seguente funzionalità è stato necessario aggiungere, nel metodo keyboard dell'applicazione, nel caso in cui viene selezionata la modalità change zoom, la posizione attuale del modello (ossia il campo di vista attuale rispetto alla posizione di partenza) e lo step di cui occorre avvicinarsi/allontanarsi alla pressione dei tasti f, F.

Il risultato ottenuto è il seguente:

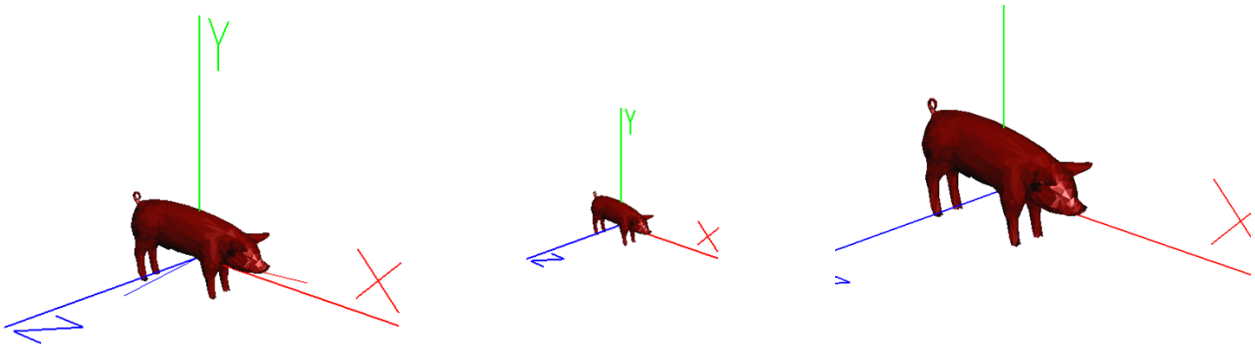
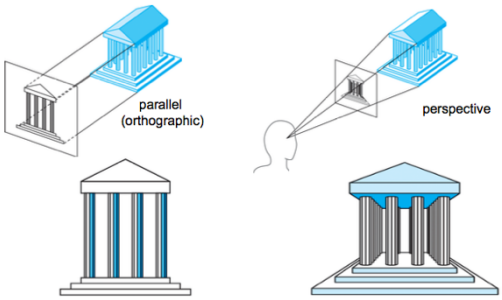


Figura 5: effetto dello zoom in/out

4) ***proiezione – modifica il tipo di proiezione e quindi il volume di vista***

Per realizzare la seguente funzionalità è necessario cambiare modalità di disegno da prospettica ad ortogonale a seconda di ciò che viene richiesto dal menù. I comandi utilizzati sono i seguenti:

```
gluPerspective(fovy, aspect, 1, 100);
glOrtho(-1.0, 1.0, -1.0, 1.0, -10, 100);
```



Il primo viene utilizzato per mostrare il modello in modalità prospettica, il secondo viene utilizzato per mostrare il modello in modalità ortogonale.
Nella modalità ortogonale le proiezioni avvengono in modo ortogonale ad ogni punto della figura (i raggi proiettori sono paralleli), mentre nella modalità prospettica i punti della figura vengono proiettati in un unico punto di vista. La modalità prospettica risulta essere molto più vicina alla realtà rispetto alla modalità ortogonale. Nelle due modalità il volume di vista è differente: nella modalità prospettica è una piramide troncata, mentre nella modalità ortogonale è un parallelepipedo.
Il risultato ottenuto è il seguente:

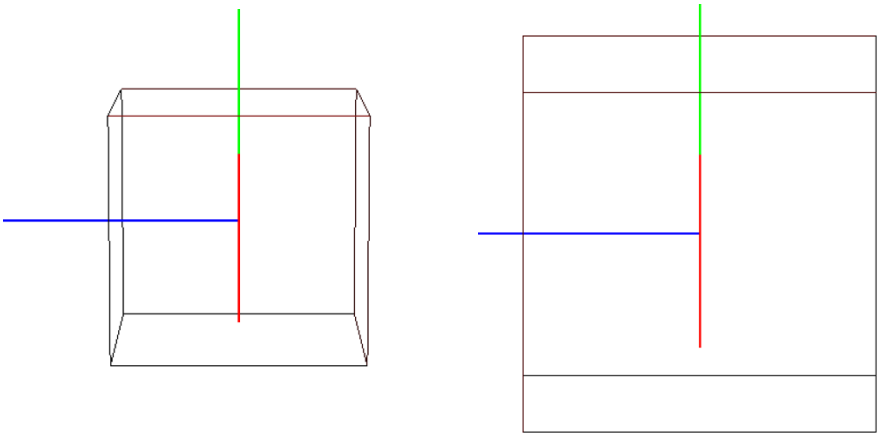


Figura 6: prospettica vs. ortogonale

Per ottenere il risultato mostrato è stato necessario spostare la camera in una posizione in cui il risultato finale fosse significativo. Inoltre, è stato utilizzato un cubo in modo tale che la differenza fra le due modalità fosse messa in evidenza.

5) ***culling – abilitare/disabilitare il back face culling***

Per realizzare la seguente funzionalità è necessario abilitare/disabilitare il culling attraverso i seguenti comandi:

```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);

glDisable(GL_CULL_FACE);
```

In particolare, quando da menù viene attivato il culling, la parte non visibile della mesh poligonale viene tagliata (l'effetto è quello di vedere un numero minore di triangoli).

Il risultato ottenuto è il seguente:

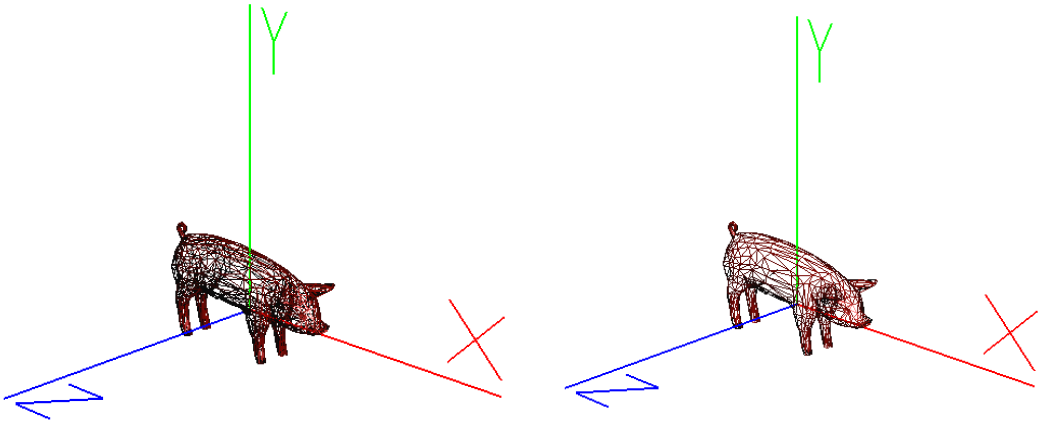


Figura 7: culling

6) ***wireframe – cambiare la modalità di rendering dei poligoni***
Per realizzare la seguente funzionalità è necessario specificare una modalità di disegno a seconda che il wireframe sia attivato o meno. A tal scopo, sono stati utilizzati i seguenti comandi:

```
glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );
glPolygonMode( GL_FRONT_AND_BACK, GL_FILL );
```

Quando da menù è attivata la modalità wireframe, vengono mostrati i triangoli vuoti (primo comando), quando invece la modalità wireframe è disattivata i triangoli vengono riempieti (secondo comando).
Il risultato ottenuto è il seguente:

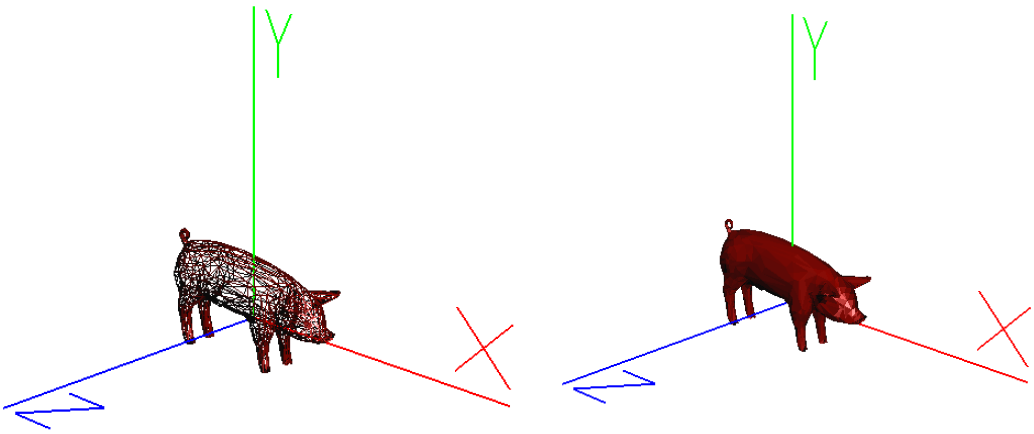


Figura 8: wireframe

7) ***shading – cambiare la modalità di shading***
Per realizzare la seguente funzionalità è necessario specificare una modalità di disegno a seconda che lo smooth sia attivato o meno. A tal scopo, sono stati utilizzati i seguenti comandi:

```
glShadeModel(GL_SMOOTH);
glShadeModel(GL_FLAT);
```

Quando da menù è attivata la modalità smooth (shading sempre attivo), vengono smussati i triangoli che compongono la mesh (primo comando), quando invece la modalità smooth viene disattivata (shading sempre attivo) i triangoli non sono più smussati (secondo comando).

Il risultato ottenuto è il seguente:

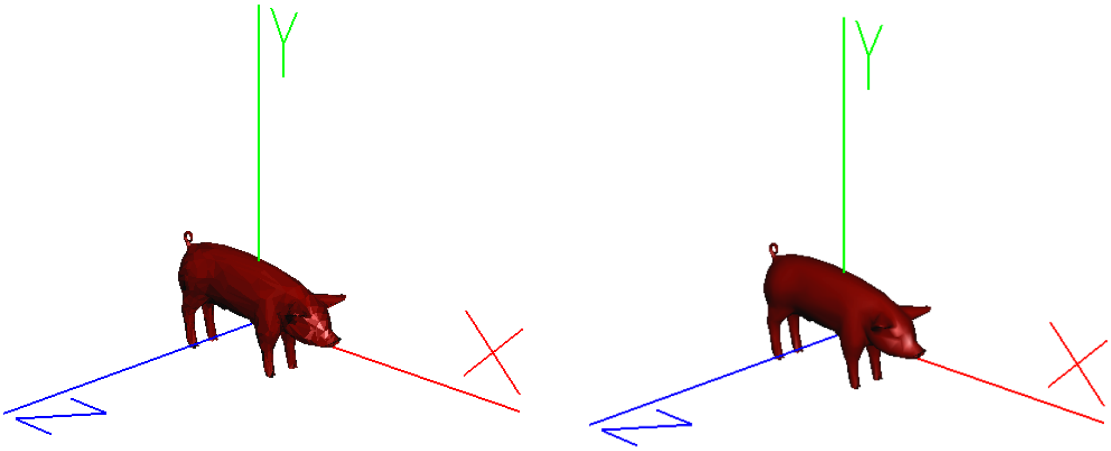


Figura 9: smooth

8) ***trackball – rendere sempre attiva la trackball virtuale per il controllo della camera***
Per realizzare questa funzionalità è stato necessario modificare la funzionalità trackball già fornita dall'applicazione. In particolare, la trackball fornita consente di ruotare le coordinate WCS rispetto gli assi x, y e z alla pressione del tasto sinistro del mouse. Al rilascio del mouse gli assi rimangono nella posizione stabilita, ma il successivo spostamento riparte dalla posizione iniziale e non da quella attuale. Per renderla sempre attiva occorre, quindi, incrementare di volta in volta le posizioni degli assi e l'angolo di rotazione in modo tale che risulti essere la somma dei contributi forniti ad ogni click del mouse.

9) ***camera motion – muovere la camera lungo un percorso***
Per realizzare questa funzionalità si è deciso di far compiere alla camera un percorso lungo una curva di Beziér alla pressione del tasto 's' della tastiera. In particolare, i passi seguiti per la realizzazione della funzionalità sono stati i seguenti:

- inserimento del seguente comando:

  ```
  glutIdleFunc(idle);
  ```

  Inserendo questo comando, la funzione idle passata come argomento viene invocata quando non ci sono input inseriti dall'utente;
- aggiunta, nella funzione keyboard, di una "reazione" alla pressione del tasto 's'. In particolare, quando il tasto indicato viene premuto viene attivata/disattiva la funzione camera motion;
- scrivere la funzione idle: la funzione idle, quando la funzionalità camera motion è attivata, esegue l'algoritmo di De Casteljau su una serie di punti di controllo "fissi" (stabiliti dal programmatore) e sposta la camera nel punto restituito dall'algoritmo (un punto della curva di Bèzier costruita a partire dai punti di controllo). Il procedimento viene ripetuto finché la modalità non viene disattivata.

Terminato l'inserimento delle funzionalità richieste nella seconda parte dell'esercitazione è stato possibile procedere con la terza parte. In particolare, in questa parte dell'esercitazione venivano richieste le seguenti funzionalità:

1) ***posizionare almeno tre diversi modelli geometrici nella scena sfruttando le funzioni openGL: glPushMatrix(), glPopMatrix()***

Per realizzare questa prima richiesta è stato in primo luogo necessario modificare il metodo init dell'applicazione fornita. In questo metodo, infatti, venivano caricati i vertici di una mesh poligonale (pig.m). Per fare in modo che vengano caricate tre o più differenti mesh, è necessario ripetere le operazioni già presenti per tutte le mesh desiderate (a tal fine è stato realizzato un metodo apposito,

loadMesh(…) che dato in ingresso il path del file carica tutti i valori dei vertici, delle facce e delle normali ai vertici ed alle facce). Le mesh poligonali che si è deciso di caricare sono: pig.m, cactus.m e teapot.m.
Svolte queste prime operazioni il risultato ottenuto è il seguente:

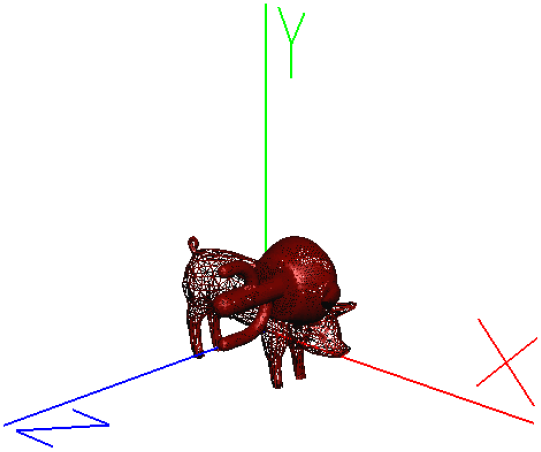


Figura 10: caricamento di tre diverse mesh poligonali

Come si può notare dalla figura i tre oggetti posizionati in scena risultano essere sovrapposti. Per posizionare gli oggetti in diverse zone del modello è necessario moltiplicare la posizione attuale di ciascuno di essi per una matrice 4x4 contenente le informazioni sulla traslazione e rotazione dell'oggetto rispetto al WCS (World Coordinate System).
La matrice 4x4 ha la seguente struttura:

- le prime tre righe e colonne (sottomatrice 3x3) contengono informazioni sulla rotazione;
- le ultime tre righe contengono informazioni sulla traslazione;
- l'ultima riga contiene tre zeri (vettore) ed un uno (punto).

Il metodo nel quale occorre inserire i comandi sulle posizioni delle mesh è il metodo display, che dovrà quindi essere modificato.
La posizione iniziale degli oggetti è data dalle coordinate dei vertici caricati da file.
Gli oggetti risultano essere sovrapposti in quanto nella display è presente il seguente comando, che posizione il modello al centro della scena:

```
gluLookAt(camE[0],camE[1],camE[2],
          camC[0],camC[1],camC[2],
           camU[0],camU[1],camU[2]);
```

Ogni volta che si lavora sulle posizioni delle mesh è necessario inserire i seguenti comandi:

```
glPushMatrix();
glPopMatrix();
```

Il primo comando carica il contenuto dei comandi che seguono nello stack; l'ultimo rimuove le informazioni inserite dallo stack.
Chiariti i passaggi da seguire e create n matrici 4x4 (dove n è il numero di mesh caricate), è necessario inserire i seguenti comandi:

```
glPushMatrix();
glMultMatrixf(matrix[i]);
glCallList(listname[i]);
glPopMatrix();
```

Si va quindi a moltiplicare con la funzione glMultMatrixf(…) il contenuto della matrice rotazione/traslazione alla posizione attuale degli elementi nella mesh e poi si vanno a disegnare le mesh poligonali con il comando glCallList(…) analizzato all'inizio della relazione.
Il risultato ottenuto è il seguente:

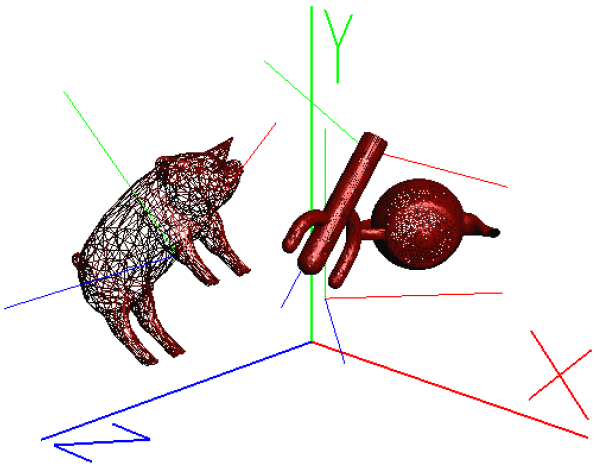


Figura 11: riposizionamento delle mesh poligonali

Per ciascuna mesh è stato disegnato il sistema di coordinate Object (utile in seguito) con il seguente comando:

```
drawAxis( 1.0, 0 );
```

Il metodo drawAxis(…) è fornito dall'applicazione di partenza e consente di disegnare gli assi.

2) ***permettere la traslazione e rotazione dei singoli oggetti rispetto ai sistemi di riferimento WCS ed OCS (World e Object Coordinate Systems)***

Per realizzare questa richiesta il procedimento da seguire è simile a quello al passo precedente. In particolare, sarà necessario gestire le informazioni sulle rotazioni/traslazioni riferite al WCS attraverso un'apposita matrice e lo stesso vale per le informazioni sulle rotazioni/traslazioni riferite all'OCS. In particolare saranno necessarie n matrici relative al WCS ed n matrici relative all'OCS (dove n è il numero di mesh caricate). A questo punto è necessario modificare il metodo keyboard in modo tale che vengano rilevate le pressioni dei tasti richiesti. Ogni volta che viene richiesta una rotazione/traslazione di un oggetto rispetto ad un asse indicato e rispetto al WCS o all'OCS sarà necessario aggiornare la matrice già esistente (contenente le informazioni sulla rotazione/traslazione rispetto al sistema di interesse) aggiungendo lo "step" di rotazione/traslazione da eseguire.
I comandi da eseguire sono quindi i seguenti:

```
glPushMatrix();
glLoadIdentity();
step = 5.0;
if(key == 'x'){
      glRotatef(step, 1, 0, 0);
}else if(key == 'X'){
      glRotatef(-step, -1, 0, 0);
}else if(key == 'y'){
      glRotatef(step, 0, 1, 0);
}else if(key == 'Y'){
      glRotatef(-step, 0, 1, 0);
}else if(key == 'z'){
      glRotatef(step, 0, 0, 1);
}else if(key == 'Z'){
```

```
        glRotatef(-step, 0, 0, 1);
    }
    glMultMatrixf(matrixWCS[indexObjectToMode]);
    glGetFloatv(GL_MODELVIEW, matrixWCS[indexObjectToMode]);
    glPopMatrix();
```

Quindi:

- si attiva il caricamento sullo stack;
- si carica sullo stack la matrice identità;
- si moltiplica la matrice identità per la matrice di rotazione dello step desiderato (glRotatef(…));
- si moltiplica questa matrice alla matrice matrixWCS[…] contenente già tutte le informazioni sulla posizione rispetto al WCS dell'oggetto selezionato;
- si memorizza il risultato nella moltiplicazione nella stessa matrice matrixWCS[…] (glGetFloatv - sovrascrive alla precedente la nuova);
- infine si disattiva il caricamento sullo stack.

La nuova matrice rotazione deve essere inserita nello stack prima di quella attuale (matrixWCS[…]) in quanto vale la seguente regola:

$$\mathbf{v}' = \mathbf{M}_4 \cdot (\mathbf{M}_3 \cdot (\mathbf{M}_2 \cdot (\mathbf{M}_1 \cdot \mathbf{v})))$$

L'ordine di moltiplicazione delle matrici è importante, in quanto non vale la regola di commutatività. L'ultima modifica deve essere sempre inserita per prima.
Il caso mostrato è quello della rotazione rispetto al WCS, ma lo stesso vale per tutte le altre richieste possibili.
Avendo a disposizione tutte le matrici aggiornate, sia rispetto al WCS che rispetto all'OCS, la display dovrà semplicemente essere modificata come segue:

```
glPushMatrix();
glMultMatrixf(matrixWCS[i]);
glMultMatrixf(matrix[i]);
drawAxis( 1.0, 0 );
glMultMatrixf(matrixOCS[i]);
glCallList(listname[i]);
glPopMatrix();
```

Si aggiungono, cioè, le moltiplicazioni delle matrici WCS ed OCS.
Anche in questo caso l'ordine di moltiplicazione delle matrici è rilevante; infatti in questo modo i calcoli applicati sono i seguenti:

- $Pos_{WCS} = T_{WCS} \cdot T$ → si ruota e trasla la mesh in base al contenuto della matrice WCS. In questo modo si ottiene la posizione rispetto al sistema di coordinate WCS;
- $Pos_{OCS} = Pos_{WCS} \cdot T_{OCS}$ → si ruota e trasla la mesh in base al contenuto della matrice OCS. In questo modo si ottiene la posizione rispetto al sistema di coordinate OCS;
- $Pos_{OCS} \cdot vertex$ → si disegnano i vertici in base alle posizioni calcolate nei punti sopra.