Lab 2 - Modelli Geometrici 3D

2.1 Caricamento e visualizzazione modelli geometrici

Il primo punto dell'esercitazione richiedeva di implementare le seguenti funzioni:

1. Caricamento e visualizzazione modelli geometrici di tipo mesh in formato .m
2. Visualizzazione superfici quadriche dalla libreria GLU (es. Sfere, cilindri, tori)
3. Verifica della gestione della visualizzazione dei modelli poligonali a mesh tramite display list.
4. Calcolo e memorizzazione delle normali ai vertici per i modelli mesh poligonali. Visualizzazione con normali ai vertici in modalità smooth.

I punti 1 3 e 4 sono stati applicati nella funzione `loadMesh()` che viene richiamata nel main un numero di volte pari a MESH, una costante dell'applicativo fissata a 4 che rappresenta il numero di oggetti da disegnare. Nel main viene quindi generata una display list per ogni oggetto, riempita opportunamente nella funzione `loadMesh()` con le informazioni necessarie per disegnarlo. Gli oggetti sono stati scelti dalla cartella `../data` presente nel template dell'esercitazione e sono *pig.m*, *cactus.m*, *teapot.m*.

La funzione `loadMesh()` è stata creata sulla base della funzione `init()` già presente nel template ed esegue nell'ordine:

- apertura del file *.m
- lettura di tutte le linee e quindi vertici e facce dell'oggetto
- per ogni faccia viene eseguito il calcolo della normale alla faccia
- per ogni vertice eseguito il calcolo della normale al vertice
- chiusura del file
- inizializzazione della display list relativa all'oggetto e riempimento di essa con il disegno di ogni triangolo della mesh.

2.1.1 Normale alla faccia

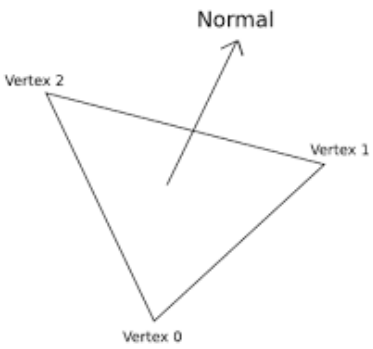


Figura 2.1

Ogni faccia è costituita da 3 vertici poiché ogni faccia è un triangolo. La normale alla faccia è ottenuta attraverso la cross-correlazione di due vettori ottenuti dalle sottrazioni $v_2 - v_0$ e $v_1 - v_0$. Il risultato della cross-correlazione viene poi normalizzato ottenendo la normale alla faccia:

- $v_a = v_2 - v_0$
- $v_b = v_1 - v_0$
- $norm = cross\_product(v_a, v_b)$
- $normalize(norm)$

Questi passaggi sono stati applicati nel codice attraverso le funzioni già presenti nella libreria `v3d.h`

2.1.2 Normale al vertice

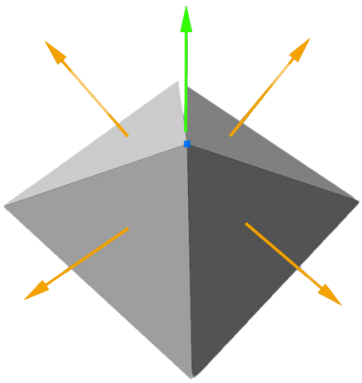


Figura 2.2

La normale al vertice è ottenuta calcolando la media di tutte le normali. I passi implementati nel codice sono i seguenti:

Per ogni vertice i:

- Crea vettore somma S
- Crea contatore k
- Per ogni faccia j:
  - Se la faccia j contiene il vertice i allora somma ad S la normale alla faccia e incrementa il contatore k
- dividi S per k
- normalizza S

2.2 Controllo interattivo della scena

Le funzioni da implementare erano le seguenti:

- **Zoom**
- **Proiezione**
- **Culling**
- **Wireframe**
- **Shading**
- **Sistemare movimento della TrackBall**
- **Esplorazione della scena tramite un'animazione.**

2.2.1 Zoom

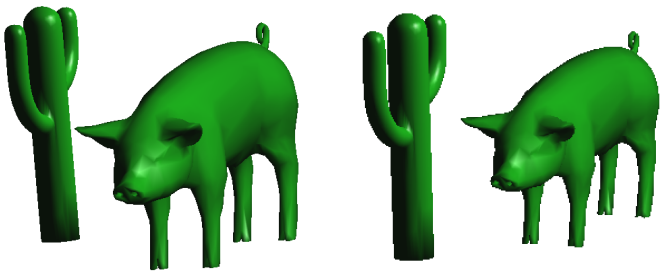
Lo zoom è stato implementato in due modi:

- **tramite l'utilizzo della funzione richiamabile dal menu a tendina (con f,F)**: nella funzione `keyboard()` sono stati aggiunti due *case* per la pressione dei tasti 'f' e 'F'. Il risultato è la variazione del parametro *fovy* ovvero il *field of view* della camera.
- **tramite lo scorrimento della rotellina del mouse (o del trackpad)**: il principio è lo stesso di quello sopra, ma occorre aggiungere al main la primitiva `glutMouseWheelFunc(mouseWheel);` che richiama appunto la *mouseWheel()* ad ogni scorrimento della rotellina del mouse.

2.2.2 Proiezione

Attraverso la variabile globale *orpro* si tiene conto di quale sia la proiezione corrente. Nella `display()` sotto alla `glMatrixMode(GL_PROJECTION)`, si applica la ortografica o la prospettiva a seconda di *orpro*:

- se *vera*: si usa la funzione `glOrtho(-fovr, fovr, -fovr, fovr, -fovr, fovy)` inserendo i valori dei piani di clipping relativi al *field of view* della camera. In questo modo è possibile usare lo zoom anche in ortografica poiché i piani di clipping variano in base al parametro *fovr* che si basa sul valore di *fovy*
- se *falsa*: si usa la funzione `gluPerspective(fovy, aspect, 1, 100);`



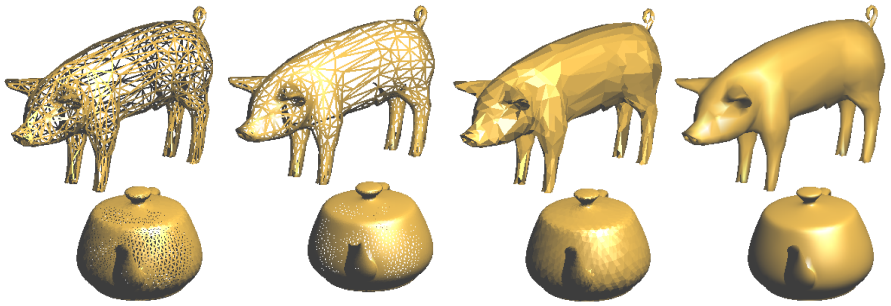
(a) Ortografica (b) Prospettiva

Come si può notare, nella prospettiva il maialino ha una dimensione minore, mentre nell'ortografica si ha una proiezione parallela e quindi le dimensioni rimangono le stesse.

2.2.3 Culling, Wireframe, Shading

Queste tre funzioni sono state abilitate nella funzione `display` a seconda del corrispettivo parametro intero che ne determina l'abilitazione. Le tre sono state abilitate grazie alle primitive:

- **Culling**: abilitata con `glEnable(GL_CULL_FACE)` e disabilitata con `glDisable(GL_CULL_FACE)`
- **Wireframe**: abilitata con `glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)` e disabilitata con `glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)`
- **Shading**: abilitato con `glShadeModel(GL_SMOOTH)` e disabilitato con `glShadeModel-(GL_FLAT)`



(a) Wireframe ON (b) Culling ON (c) Shading Flat (d) Shading Smooth

Nell'immagine Culling ON si può notare che le facce interne non vengono visualizzate, in quanto il culling viene abilitato a default per GL_BACK, ovvero il retro delle facce.

2.2.4 Camera Motion

Per fare in modo che la telecamera seguisse una traiettoria attorno alla scena è necessario creare una curva di Bézier chiusa a coordinata Y costante. Dati alcuni punti di controllo che creano la curva e applicando l'algoritmo di De Casteljau usato nella prima esercitazione, è bastato abilitare la funzione `idle()` con un incremento del parametro t ogni 50 millisecondi e una chiamata alla routine `moveCamera()` che applica l'algoritmo di De Casteljau. Ogni punto trovato con l'algoritmo sarà assegnato al punto C della camera, ovvero l'occhio della camera (eye). La funzione è selezionabile dal menù a tendina e l'effetto finale è un'animazione che sorvola la scena.

2.2.5 Trackball

La trackball è stata implementata in due modi diversi, ma solo il secondo modo ha portato ad una rotazione corretta. Per completezza riporto entrambi i metodi.

- **Primo metodo**: nella funzione `motion()` si mantiene in memoria l'ultima rotazione e l'ultima posizione relativa degli assi. Basta applicare l'operatore $+=$ ad *tbAngle* e *tbAxis* e ad ogni movimento aggiornare l'ultima posizione con `v3dSet(lastPosition, currentPosition);` dove *lastPosition* e *currentPosition* sono rispettivamente tbW e tbV. Questo metodo permette di salvare l'ultima posizione della trackball e continuare la rotazione, ma questa avviene sempre ad un angolo positivo e non permette mai di cambiare direzione di rotazione. Questo metodo è stato mantenuto fino al completamento del terzo punto, il quale ha permesso di comprendere meglio il funzionamento dello stack delle matrici e di applicare quando imparato anche alla rotazione della trackball.

- **Secondo metodo**: si basa sul concetto che ad ogni trascinamento, prima viene calcolato ciò che riguarda la rotazione e poi viene disegnato il risultato. Si sfrutta questo concetto nella `motion()` creando la matrice di rotazione della trackball che verrà poi applicata nella `display()` moltiplicandola alla fine di tutto (quindi in cima ad ogni altra operazione sullo stack delle matrici di trasformazione) con la `glMultMatrixf(WCS[TRACKBALL])`. Il codice è il seguente:

```
glLoadIdentity();
glRotatef(tbAngle, tbAxis[0], tbAxis[1], tbAxis[2]);
glMultMatrixf(WCS[TRACKBALL]);
glGetFloatv(GL_MODELVIEW_MATRIX, WCS[TRACKBALL]);
```

Ulteriori dettagli sulla sequenza di queste operazioni saranno dati nella sezione che segue.

2.3 Manipolazione dello stack delle matrici di trasformazione

Le funzioni richieste sono state aggiunte al menù e alla funzione `keyboard()`. Ciò che verrà trattato in dettaglio riguarda la serie di operazioni eseguite sulle matrici di trasformazione per ottenere traslazioni e rotazioni attorno a OCS e WCS. È necessario distinguere i due casi in cui applicando una trasformazione, questa venga applicata in OCS o in WCS.

Per questo punto si è fatto uso di 3 matrici per mantenere lo stato di ogni oggetto (compresa la trackball):

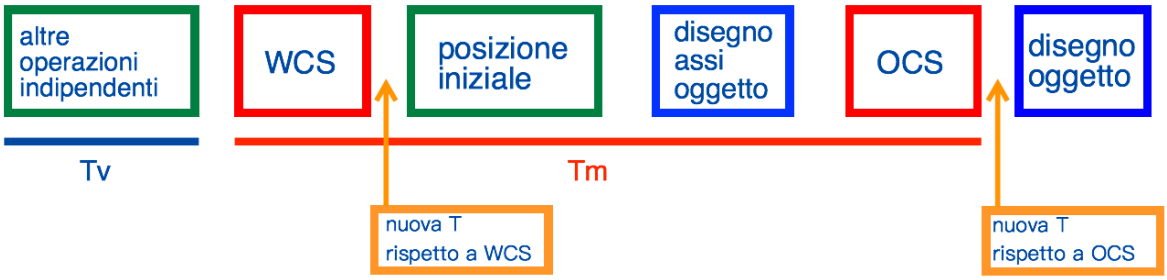
- `WCS[MESH][16]`: matrice che contiene lo stato delle trasformazioni di ogni oggetto rispetto al sistema di riferimento della scena
- `OCS[MESH][16]`: matrice che contiene lo stato delle trasformazioni di ogni oggetto rispetto al proprio sistema di riferimento
- `initialPosition[MESH][16]`: matrice che contiene la posizione iniziale di ogni oggetto.

Le prime due matrici vengono aggiornate ogni volta che l'utente seleziona traslazione o rotazione rispetto a WCS o OCS. Questa modifica viene effettuata nella funzione `keyboard()` ogni volta che l'utente preme x,y,z richiamando la funzione `applyTransform()`. Questa funzione, analogamente a ciò che avviene per la trackball, carica una matrice identità, moltiplica per la matrice precedente (che rappresenta la posizione precedente dell'oggetto) e applica la nuova traslazione o rotazione. Ovviamente la funzione distingue i casi in cui l'utente abbia selezionato OCS o WCS e traslazione e rotazione.

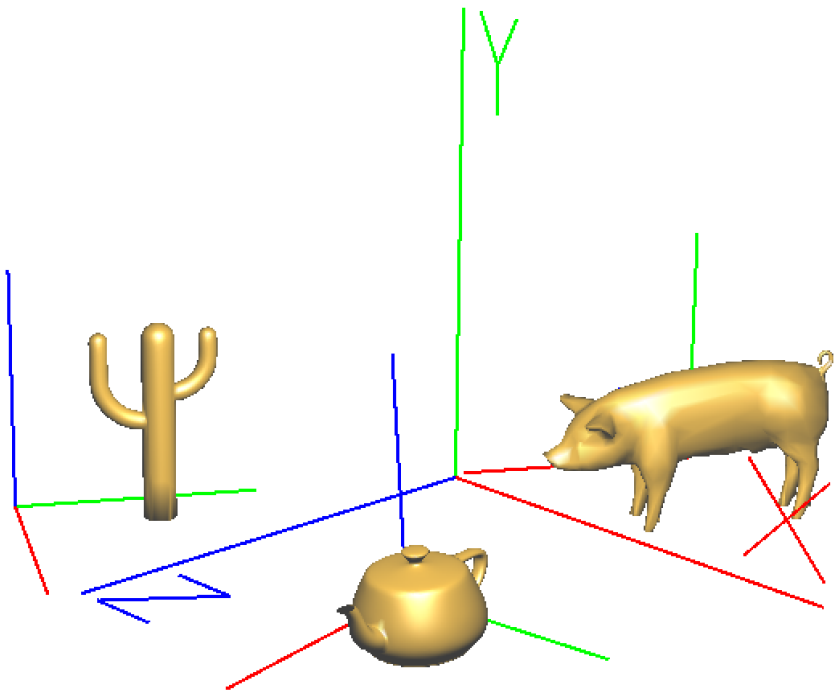
Questa applicazione permette di non *sporcare* la `display()` con rotazioni e traslazioni e permette di applicare sempre la stessa sequenza di operazioni. Per ogni mesh della scena si applicano le seguenti operazioni:

- `glPushMatrix()` : rilevo dallo stack una copia della matrice attuale applicando le seguenti operazioni solo a ciò che viene disegnato all'interno della push e della pop.
- `glMultMatrixf(WCS[mesh])` : applico le trasformazioni rispetto al WCS
- `glMultMatrixf(initialPosition[mesh])` : applico la posizione iniziale rispetto al WCS
- `drawAxis(1, 0)` : disegno gli assi nella posizione iniziale dell'oggetto

- `glMultMatrixf(OCS[mesh])` : applico le trasformazioni rispetto all'OCS
- `glCallList(mesh)` : disegno l'oggetto nella posizione derivante da WCS, posizione iniziale, OCS
- `glPopMatrix()` applico le modifiche allo stack



Un esempio del risultato finale, con gli oggetti nella posizione iniziale. Per visualizzare gli assi, è necessario utilizzare la funzione DEBUG del menù a tendina.



Note:

- è stata creata una funzione che inizializza i colori `initColors()`, in modo da rendere più leggibile il codice nella `display()`
- si è scelto di fissare la luce in modo tale che fosse solidale alla trackball.