Model Viewer

Descrizione

L’applicazione model viewer carica e visualizza sia quadriche predefinite in OpenGL GLU library, sia oggetti mesh poligonali memorizzati in file con estensione `.m`

Obiettivi

1. Culling
2. Wireframe
3. Shading
4. Modifica del punto di vista
5. Zoom in/out
6. Proiezione prospettica e ortogonale
7. Correzione Trackball
8. Camera motion
9. Traslazione e rotazione rispetto a WCS e OCS

Risultati

1. Culling

L'abilitazione del culling viene fatta usando `glEnable/glDisable(GL_CULL_FACE)`. Abilitato il culling si nota come le facce in senso opposto al punto di vista non vengano renderizzate.

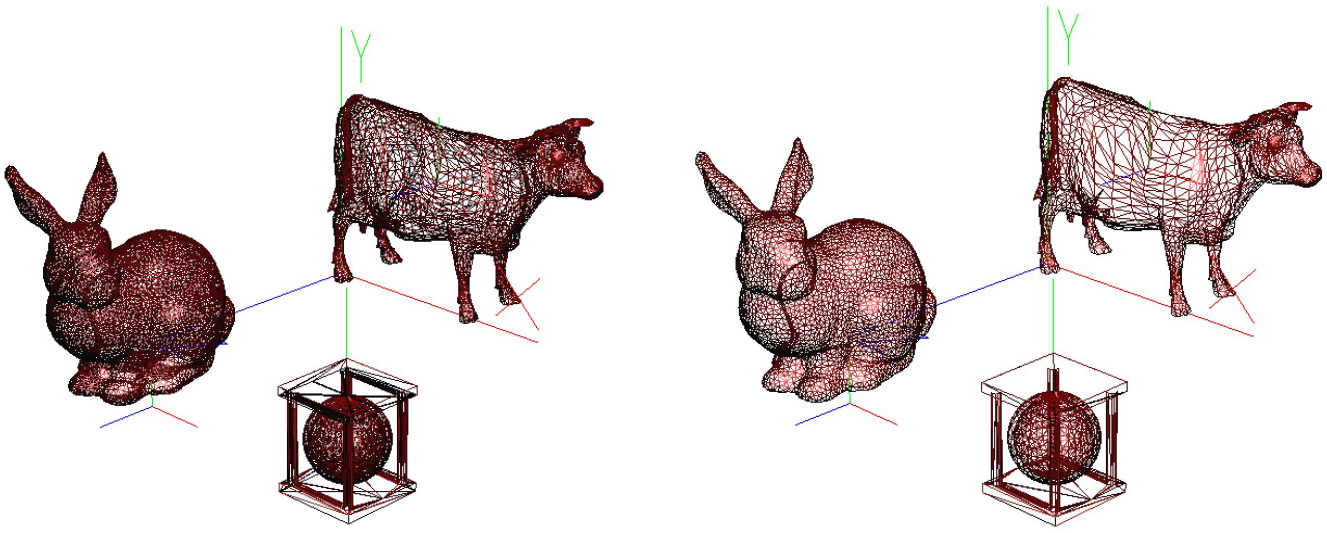

Fig 1. Culling disabilitato (a) culling abilitato (b)

2. Wireframe

`glPolygonMode(GL_FRONT_AND_BACK, _ )` permette di gestire la visualizzazione dei poligoni per fronte e retro di ogni faccia scegliendo come secondo parametro fra `GL_LINE` (solo wireframe) o `GL_FILL` (facce riempite)

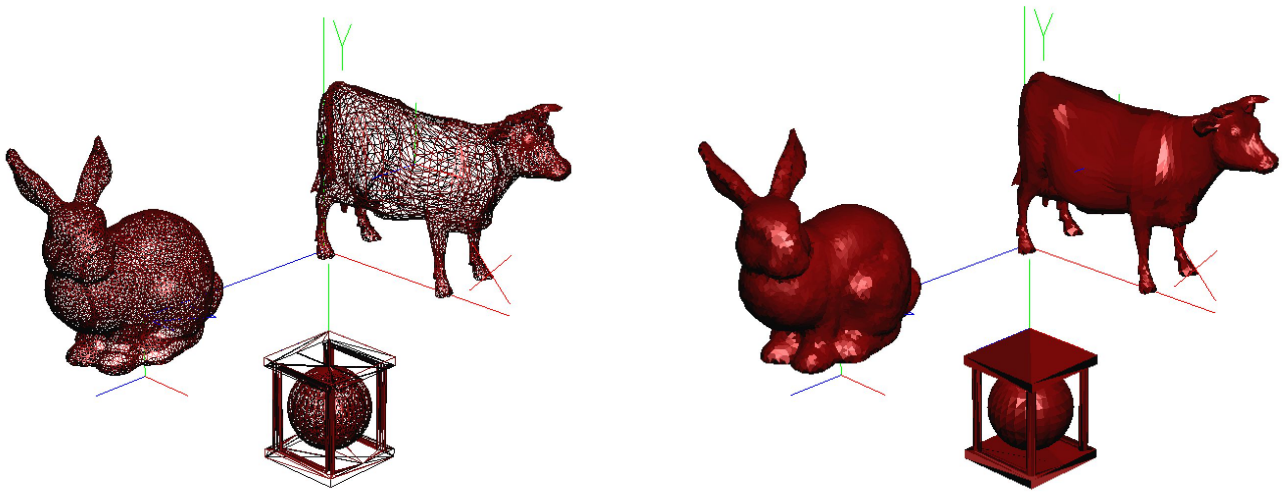

Fig 2. Visualizzazione wireframe (a) o con facce riempite (b)

3. Shading

`glShadeModel( _ )` permette di gestire la modalità di shading scegliendo fra `GL_FLAT` (un colore per faccia) o `GL_SMOOTH` (interpolazione del colore per ogni pixel).

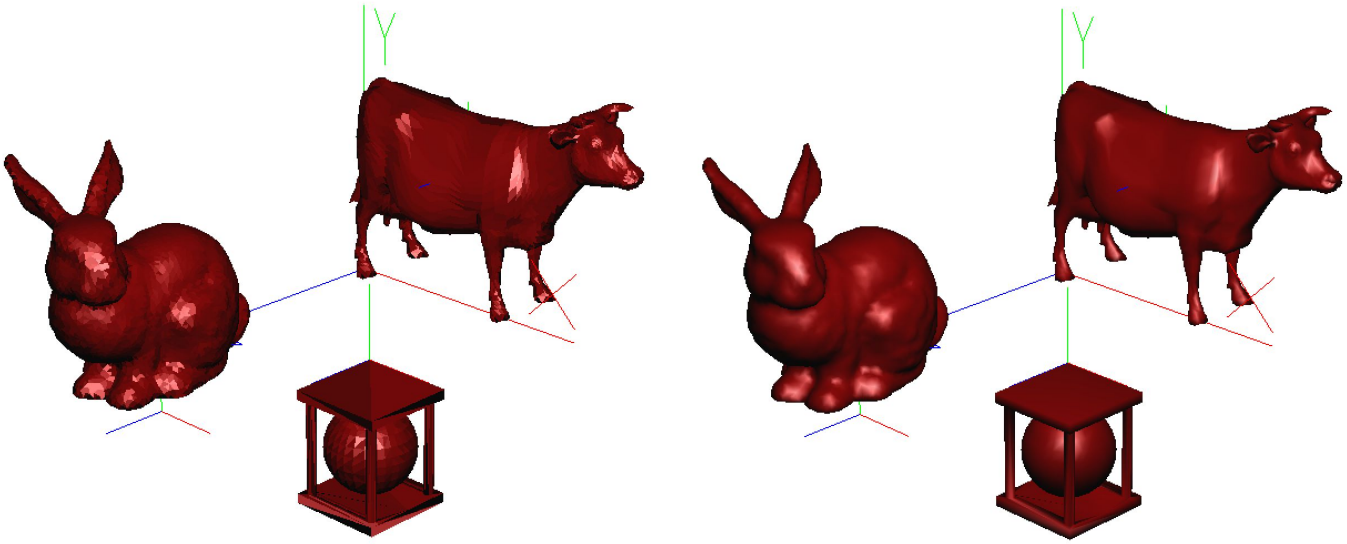

Fig 3. Visualizzazione shading flat (a) o con shading smooth (b)

4. Modifica del punto di vista

Selezionando dal menu la modalità `MODE_CHANGE_EYE_POS` è possibile modificare la posizione del punto di vista con i tasti x, y, z (minuscole corrispondono a diminuizione, maiuscole ad incremento). Lo spostamento è implementato aumentando o diminuendo di `0.5` le coordinate di `camE` (camera eye).

5. Zoom in/out

Lo zoom è stato implementato usando la formula della retta parametrica passante per un punto:

```
camE = camE + ( origin - camE ) * ( 0.1f * direction )
```

dove:

- `camE`, posizione della camera
- `origin`, posizione dell'origine degli assi cartesiani `(0,0,0)`
- `direction`, direzione dello zoom modificabile premendo `v` che può essere `-1` (zoom in) o `1` (zoom out)

6. Proiezione prospettica ed ortogonale

La proiezione prospettica viene effettuata utilizzando la funzione `gluPerspective`, mentre la proiezione ortogonale viene effettuata utilizzando la funzione `glOrtho`.

7. Correzione Trackball

La trackball è stata corretta salvando su una matrice (`tbRotationMatrix`) la rotazione della scena alla fine di ogni azione di drag. Alla fine di ogni drag viene chiamata `glutPostRedisplay()`, il metodo `display()` sovrascrive la matrice di rotazione con il seguente codice:

```
glPushMatrix();
glLoadMatrixf(tbRotationMatrix);
glRotatef(tbAngle, tbAxis[0], tbAxis[1], tbAxis[2]);
glGetFloatv(GL_MODELVIEW_MATRIX, tbRotationMatrix);
glPopMatrix();
```

8. Camera motion

Il movimento della camera è stato implementando spostando la posizione del punto di vista lungo una curva di bézier circolare definita da 5 control points, valutata ad intervalli di `0.001f` fra `0` e `1`.

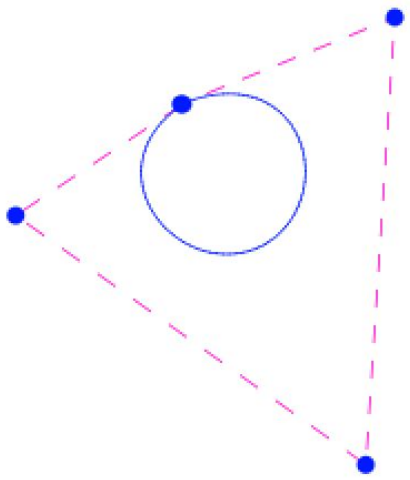

Fig 4. Percorso della camera, il primo control point e l'ultimo coincidono

9. Traslazione e rotazione rispetto a WCS e OCS

Le trasformazioni rispetto ai sistemi di coordinate WCS e OCS sono state implementate salvando le trasformazioni nelle apposite variabili:

- wcsTranslations / wcsRotations
- ocsTranslations / ocsRotations

Per applicare le trasformazioni è stato aggiunto del codice nel metodo `display()` nel ciclo che disegna i modelli in scena, il concetto del funzionamento è spiegato nel seguente pseudocodice

```
per ogni poligono
      applica trasformazioni WCS
      disegna gli assi
      applica trasformazioni OCS
      disegna l'oggetto
```