- INTRODUZIONE
- DRAWING
- EVENT MANAGEMENT
- VIEWING
- DOUBLE BUFFERING
- Z-BUFFERING
- LIGHTING

RIEPILOGO

DRAWING

- creare una finestra
- inizializzare i buffer
- gestire il colore
  - in modalita` indicizzata
  - in modalita` rgb
- disegnare vertici e primitive

EVENTI

- OpenGL non possiede gestione degli eventi
- alternative possibili:
  - utilizzare direttamente la gestione eventi fornita dal window system
  - utilizzare una libreria che fornisca un livello di astrazione (es. glut)

EVENTI CON GLUT

- eventi gestiti da glut:
  - eventi generati da windows system
  - eventi generati da keyboard
  - eventi generati da mouse
  - eventi generati da altri dispositivi
- gestione tramite callback:
  - si definisce una funzione in grado di gestire un evento
  - si associa la funzione all'evento tramite una chiamata a glut
  - ogni volta che si verifica l’evento la funzione viene eseguita

RESHAPE

- gestione evento di “reshape”

```
void glutReshapeFunc(void(*f)(int,int))
```

  - permette di associare la funzione **f** all’evento di “finestra ridimensionata” (reshape)
  - l’evento reshape viene generato anche quando la finestra viene creata
  - tipicamente la funzione **f** ha il compito di settare i parametri di viewing

EXPOSE

- gestione evento di “expose”

```
void glutDisplayFunc(void(*f)(void))
```

  - permette di associare la funzione **f** all’evento di “finestra scoperta” (expose)
  - l’evento expose viene generato anche quando la finestra viene creata e dopo ogni evento di reshape
  - tipicamente la funzione **f** ha il compito di (ri)disegnare la scena

GLUT MAIN LOOP

- per attivare la gestione degli eventi

```
void glutMainLoop( void )
```

  - attiva il loop degli eventi
  - deve essere chiamata per ultima (non restituisce il controllo)

ESERC.: EVENTS

```
#include <stdio.h>
#include <GL/glut.h>

void reshape( int w, int h )
{
printf( "event: reshaped to %d x %d\n", w, h );
}

void expose( void )
{
printf( "event: exposed\n" );
}

void main( int argc, char** argv )
{
  glutInitDisplayMode   ( GLUT_RGB );
  glutInitWindowPosition( 300, 100 );
  glutInitWindowSize    ( 150, 150 );
  glutCreateWindow      ( argv[0]  );

  glutReshapeFunc ( reshape );
  glutDisplayFunc ( expose  );
  glutMainLoop    (         );
}
```

KEYBOARD

- getione caratteri ASCII

```
void glutKeyboardFunc( void(*f)
                       (char k, int x, int y) )
```

  - permette di associare la funzione **f** all'evento "pressione di un tasto corrispondente ad un carattere ASCII"
  - **k** e` il carattere
  - **x, y** sono le coordinate del mouse al momento del verificarsi dell'evento

- N.B.: **<DEL>**, **<BS>**, **<ESC>** sono caratteri ASCII

KEYBOARD (cont.)

- getione caratteri estesi

```
void glutSpecialFunc( void(*f)
                      (int k, int x, int y) )
```

  - permette di associare la funzione **f** all'evento "pressione di un tasto corrispondente ad un carattere speciale"
  - **x**, **y** sono le coordinate del mouse
  - **k** puo` valere:

```
GLUT_KEY_F1, ..., GLUT_KEY_F12,
GLUT_KEY_LEFT, GLUT_KEY_UP,
GLUT_KEY_RIGHT, GLUT_KEY_DOWN,
GLUT_KEY_HOME, GLUT_KEY_PAGE_UP,
GLUT_KEY_END, GLUT_KEY_PAGE_DOWN,
GLUT_KEY_INSERT
```

ESERC.: KEYBOARD

```
#include <stdio.h>
#include <GL/glut.h>

#define BS    8
#define ESC  27
#define DEL 127

void key( unsigned char k, int x, int y )
{
   switch( k )
   {
   case  'a': printf("\"a\" pressed\n");break;
   case  'b': printf("\"b\" pressed\n");break;

   case  ESC: exit(0);
   }
}
...
void main( int argc, char** argv )
{
   ...
   glutKeyboardFunc( key     );
   glutReshapeFunc ( reshape );
   glutDisplayFunc ( expose  );
   glutMainLoop    (         );
}
```


MOUSE

- getione eventi generati dal mouse

```
void glutMouseFunc( void(*f)(int b, int s,
                             int x, int y))
```

  - permette di associare la funzione **f** all'evento "click di un bottone del mouse"
  - valori possibili per **b** :
    `GLUT_LEFT_BUTTON`
    `GLUT_MIDDLE_BUTTON`
    `GLUT_RIGHT_BUTTON`
  - valori possibili per **s** :
    `GLUT_DOWN GLUT_UP`
  - **x**, **y** sono le coordinate del mouse al momento del verificarsi dell'evento

ESERC.: MOUSE

```
...

void mouse( int but, int state, int x, int y )
{
char *b, *left="left", *middle="middle",
         *right="right";
char *s, *press="pressed", *rel="released";

   switch( but ) {
   case GLUT_LEFT_BUTTON  : b=left;   break;
   case GLUT_MIDDLE_BUTTON: b=middle; break;
   case GLUT_RIGHT_BUTTON : b=right;  break; }

   switch( state ) {
   case GLUT_DOWN : s=press; break;
   case GLUT_UP   : s=rel;   break; }

   printf( "%s mouse button %s in %d, %d\n",
           b, s, x, y );
}

void main( int argc, char** argv )
{
   ...
   glutMouseFunc( mouse );
   ...
}
```


ALTRE CALLBACK

- “idle”

```
void glutIdleFunc( void(*f)(void) )
```

  - permette di specificare una funzione **f** da eseguire quando il sistema e` idle
  - tipicamente questa caratteristica viene utilizzata per fare del “background” processing

- timer

```
void glutTimerFunc(int ms, void(*f)(int), int)
```

  - permette di specificare una funzione **f** da eseguire al termine di un intervallo di tempo fissato

ESERC.: IDLE

```
#include <stdio.h>
#include <GL/glut.h>

void idle( void )
{
        printf("background processing\n");
        sleep(1);
}

...

void main( int argc, char** argv )
{
   ...

   glutReshapeFunc ( reshape );
   glutDisplayFunc ( expose  );
   glutIdleFunc    ( idle    );
   glutMainLoop    (         );
}
```

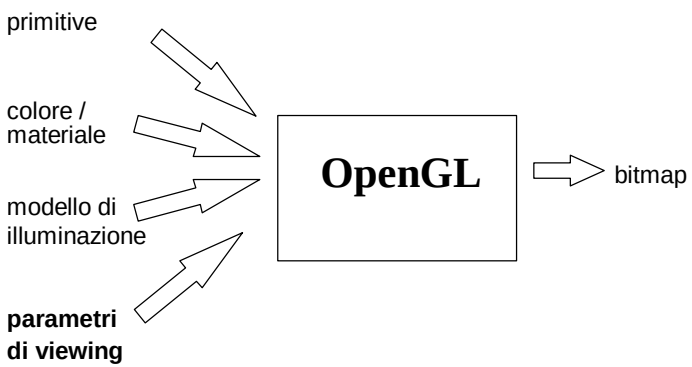

ESERC.: DRAWING

```
#include <GL/glut.h>

void resize( int w, int h )
{
   printf("resized to %d x %d\n", w, h );
}

void redraw( void )
{
   /* drawing code here */
}

void main( int argc, char** argv )
{
   glutInitDisplayMode   ( GLUT_RGB );
   glutInitWindowPosition( 300, 100 );
   glutInitWindowSize    ( 200, 200 );
   glutCreateWindow      ( argv[0]  );

   glutReshapeFunc ( resize  );
   glutDisplayFunc ( redraw  );
   glutMainLoop    (         );
}
```

- INTRODUZIONE
- DRAWING
- EVENT MANAGEMENT
- VIEWING
- DOUBLE BUFFERING
- Z-BUFFERING
- LIGHTING

RENDERING

descrizione della scena ⟶ **RENDERING** ⟶ **image**

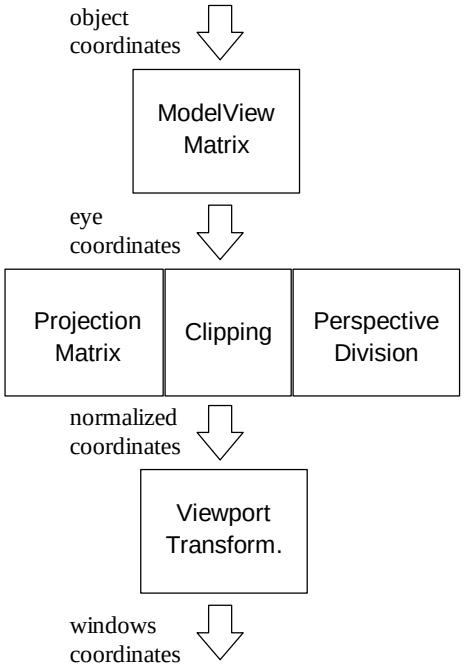


COORDINATE SYSTEMS

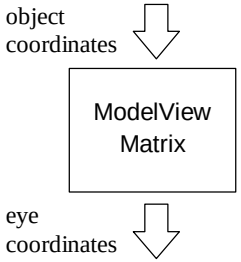
- object coordinates
  - coordinate relative ad un sistema di riferimento solidale con l'oggetto da visualizzare
- eye coordinates
  - coordinate relative ad un sistema di riferimento solidale con l'osservatore
- normalized device coordinates
  - coordinate normalizzate rispetto alle dimensioni della finestra
- window coordinates
  - coordinate (in pixel) relative alla finestra utilizzata
- device coordinates
  - coordinate (in pixel) relative al display

VIEWING

- le coordinate dei vertici delle primitive subiscono una sequenza di trasformazioni



MODELVIEW MATRIX



- definisce la relazione tra object coordinates ed eye coordinates
- usata per definire il punto di vista (viewing)
- usata per definire la scena (modeling)
- puo` essere manipolata direttamente

OPERAZIONI SULLE MATRICI

- OpenGL possiede tre stack di matrici:
  - modelview
  - projection
  - texture
- le operazioni vengono effettuate sulla matrice corrente (la matrice che si trova sulla cima dello stack corrente)
- le matrici sono organizzate per colonne (mentre il C memorizza le matrici per righe)

$$\begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

MATRICI (cont.)

- selezionare lo stack di matrici

```
void glMatrixMode(GLenum s)
```

  - valori possibili per **s**
    GL_MODELVIEW
    GL_PROJECTION
    GL_TEXTURE
  - lo stack **s** diviene lo stack corrente
  - la matrice sulla cima dello stack **s** diviene la matrice corrente

MATRICI (cont.)

- operazione di push

```
void glPushMatrix(void)
```

  - una copia della matrice corrente viene posta sulla cima dello stack
  - la nuova matrice diviene la matrice corrente

- operazione di pop

```
void glPopMatrix(void)
```

  - la matrice corrente viene eliminata dalla cima dello stack
  - la matrice sottostante diviene la nuova matrice corrente

MATRICI (cont.)

- inizializzare la matrice corrente

```
void glLoadIdentity(void)
```

  – la matrice corrente viene inizializzata alla matrice identita`

```
void glLoadMatrixf( const GLfloat *m )
```

  – la matrice corrente viene inizializzata con la matrice **m**
  – **m** e` un vettore di 16 float organizzati per colonne

MATRICI (cont.)

- manipolare la matrice corrente

```
void glMultMatrixf( const GLfloat *m )
```

  – la matrice corrente viene moltiplicata per la la matrice **m**
  – **m** e` un vettore di 16 float organizzati per colonne
  – il risultato viene copiato nella matrice corrente

```
void glTranslatef( x, y, z )
```

  – la matrice corrente viene moltiplicata per una matrice che rappresenta una traslazione delle quantita` (**x**,**y**,**z**) lungo gli assi corrispondenti
  – il risultato viene copiato nella matrice corrente

MATRICI (cont.)

```
void glRotatef( a, x, y, z )
```

- la matrice corrente viene moltiplicata per una matrice che rappresenta una rotazione attorno al vettore (**x**,**y**,**z**), di **a** gradi in senso antiorario
- il risultato viene copiato nella matrice corrente

```
void glScalef( x, y, z )
```

- la matrice corrente viene moltiplicata per una matrice che rappresenta una scalatura dei fattori (**x**,**y**,**z**) lungo gli assi corrispondenti
- il risultato viene copiato nella matrice corrente

MODELVIEW (cont.)

- relazione tra object coordinates ed eye coordinates

```
void gluLookAt( eyeX, eyeY, eyeZ, targX,
                targY, targZ, upX, upY, upZ )
```

- i parametri sono espressi in object coordinates
- la matrice corrente viene moltiplicata per una matrice che rappresenta il posizionamento dell'osservatore in (**eyeX**, **eyeY**, **eyeZ**), rivolto in direzione (**targX**, **targY**, **targZ**) e con il "naso" orientato secondo il vettore (**upX**, **upY**, **upZ**)
- il risultato viene copiato nella matrice corrente

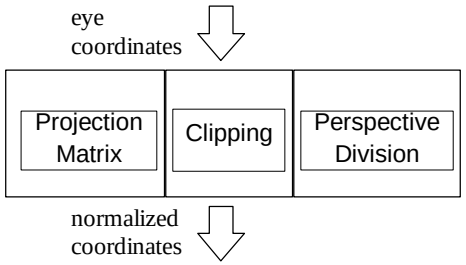
COMPOSIZIONE DI TRASF.

- il modeling della scena puo` rendere necessario comporre trasformazioni
- occorre preparare la matrice modelview prima di effettuare il drawing
- osservando che:

```
A,B        matrici
v          vettore
Av = v'
Bv'= v"
v" = Bv' = BAv = (BA)v
```

le matrici vanno composte in ordine inverso a quello desiderato delle trasformazioni che rappresentano

PROJECTION MATRIX

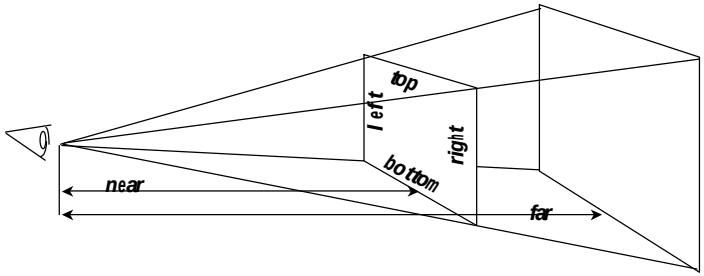


- definisce il viewing volume
  - la forma del viewing volume determina il tipo di proiezione
  - le dimensioni del viewing volume determinano il clipping
- e` possibile manipolarla direttamente
- e` conveniente utilizzare le routines predefinite
  - proiezione prospettica
  - proiezione ortogonale parallela

PROJECTION (cont.)

- proiezione prospettica

`void` **`glFrustum`**`(`**`left`**`,`**`right`**`,`**`bottom`**`,`**`top`**`,`**`near`**`,`**`far`**`)`

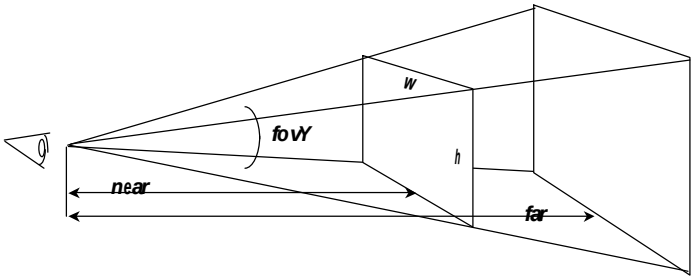


– il viewing volume e` un frustum
– **`left`**, **`right`**, **`bottom`**, **`top`** sono espressi in eye coordinates
– **`near`**, **`far`** rappresentano la distanza dall'osservatore
– e` possibile creare frustum asimmetrici
– modifica la matrice corrente

PROJECTION (cont.)

- proiezione prospettica (cont.)

`void` **`gluPerspective`**`(`**`fovY`**`,`**`aspect`**`,`**`near`**`,`**`far`**`)`

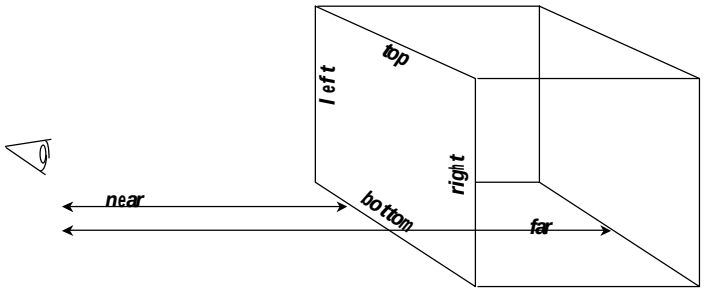


– **`fovY`** e` il "field of view"
– **`aspect`** esprime il rapporto **`w`**/**`h`**
– **`near`**, **`far`** rappresentano la distanza dall'osservatore
– solo frustum simmetrici
– modifica la matrice corrente

PROJECTION (cont.)

- proiezione ortogonale parallela

`void` **`glOrtho`**`(`**`left`**`,`**`right`**`,`**`bottom`**`,`**`top`**`,`**`near`**`,`**`far`**`)`



– il viewing volume e` un parallelepipedo
– **`left`**, **`right`**, **`bottom`**, **`top`** sono espressi in eye coordinates
– **`near`**, **`far`** rappresentano la distanza dall'osservatore
– modifica la matrice corrente

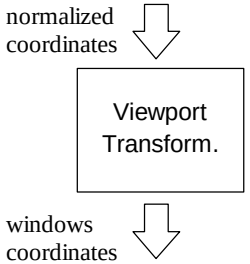
PROJECTION (cont.)

- proiezione ortogonale parallela "2D"

`void` **`gluOrtho2D`**`(`**`left`**`,`**`right`**`,`**`bottom`**`,`**`top`**`)`

– **`left`**, **`right`**, **`bottom`**, **`top`** sono espressi in eye coordinates
– **`(near`**, **`far)`** assumono il valore `(-1.0,1.0)` (si assume che in 2D z valga sempre 0.0)
– modifica la matrice corrente

VIEWPORT TRANSFORM.



- mappa le coordinate dell'immagine sulla finestra

```
void glViewport(x,y,width,height)
```

  - scala l'immagine su una viewport che ha origine in **`(x,y)`** e ha dimensioni **`(width,height)`**
  - i parametri sono in window coordinates

INDICE ESERCITAZIONI

- trasformazione di viewport
- proiezione ortogonale "2D"
- proiezione ortogonale 3D
- proiezione prospettica
- modeling

TIPS

- selezionare la matrice corrente prima di specificare trasformazioni
- specificare la trasformazione di viewing prima di effettuare il modeling
- specificare la projection matrix e la trasformazione di viewport in qualsiasi ordine, ma prima del drawing

ESERC.: VIEWPORT

```
#include <GL/glut.h>

void resize( int w, int h )
{
   glViewport( 0, 0, w, h );
}

void redraw( void )
{
   /* drawing code here */
}

void main( int argc, char** argv )
{
   glutInitDisplayMode   ( GLUT_RGB );
   glutInitWindowPosition( 300, 100 );
   glutInitWindowSize    ( 200, 200 );
   glutCreateWindow      ( argv[0]  );

   glutReshapeFunc( resize );
   glutDisplayFunc( redraw );
   glutMainLoop   (        );
}
```

ESERC.: VIEWPORT (2)

La viewport puo` avere dimensioni inferiori a quelle della finestra:

- definire una viewport corrispondente ad un quadrante a scelta della finestra

ESERC.: ORTHO2D

```
#include <GL/glut.h>

void resize( int w, int h )
{
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();

   gluOrtho2D( -3.0, 3.0, -3.0, 3.0 );

   glViewport( 0, 0, w, h );
}

void redraw( void )
{
   /* drawing code here */
}

void main( int argc, char** argv )
{
   ...
   glutReshapeFunc ( resize );
   glutDisplayFunc ( redraw );
   glutMainLoop    (        );
}
```

ESERC.: ORTHO2D (2)

- definire il viewing volume in modo che l'aspect ratio della immagine sia indipendente dall'aspect ratio della finestra
- suggerimento: considerare i due casi **w**>=**h** e **w**<**h**