

Corso di
Informatica Grafica

Introduzione
a
OpenGL

parte 4

Selezionare gli oggetti

Molte applicazioni grafiche richiedono la diretta interazione dell'utente con la scena disegnata (si pensi ad esempio a tutti i software di tipo CAD), solitamente attraverso il mouse.

L'atto di fare click con il mouse su un oggetto disegnato e attivare uno stato particolare dell'applicazione è chiamato “picking” e si basa su un meccanismo presente in OpenGL chiamato selezione, che permette di conoscere quanti e quali oggetti vengono disegnati all'interno di un certo volume di visuale. Normalmente il picking consiste di una selezione dopo aver modificato la matrice di proiezione in modo tale da restringere il viewing volume attorno al punto in cui si è fatto click.

Picking

La selezione è una modalità di rendering, in cui nessun pixel viene copiato all'interno del frame buffer, mentre ogni primitiva che rientra nel viewing volume produce un “hit record” in un buffer di selezione (che è un array di interi).

Prima di entrare in modalità selezione bisogna predisporre tale buffer di selezione e dare un nome ad ogni primitiva (o gruppo di primitive) che si vuole selezionare: il nome per una primitiva consiste in un numero intero che viene aggiunto ad uno stack dei nomi.

- 1) Creare il buffer che conterrà i risultati della selezione
- 2) Salvare la matrice di proiezione corrente
- 3) Entrare in modalità selezione e caricare la matrice identità nella matrice di proiezione
- 4) Ridurre il viewing volume attorno all'area su cui si è fatto click col mouse
- 5) Disegnare la scena (in modalità selezione, la scena non viene visualizzata)
- 4) Ripristinare la matrice di proiezione originale e caricare la matrice di MODELVIEW
- 5) Uscire dalla modalità selezione e **chiamare una funzione che elabora i risultati**

Gestire lo stack dei nomi

Le funzioni che OpenGL mette a disposizione per gestire i nomi da assegnare alle primitive sono :

- void **glInitNames()**

inizializza lo stack alla sua condizione iniziale, senza alcun nome. E' la prima funzione da chiamare quando si utilizza la selezione

- void **glLoadName**(*GLuint name*)

pone il nome specificato nella prima posizione in cima allo stack. Il nome corrente sullo stack è sostituito da *name*

- Void **glPopName()**

rimuove dalla cima dello stack un nome. Togliere un nome da uno stack vuoto genera un errore in OpenGL

Gestire lo stack dei nomi

- void **glPushName**(*GLuint name*)

aggiunge un nome in cima allo stack. Aggiungere nomi oltre la capacità massima dello stack produce un errore di OpenGL. La differenza tra questa e la precedente funzione è che questa è utilizzata nel picking gerarchico per identificare parti di una primitiva che è possibile selezionare. Di solito si chiama “glPushName(0);” prima di manipolare lo stack per essere sicuri che contenga qualcosa e non incorrere in errori.

glLoadName(name) equivale a

glPopName();

glPushName(name);

Per modificare il viewing volume e restringerlo all'area attorno al puntatore del mouse bisogna chiamare: void **gluPickMatrix**(*GLdouble x, GLdouble y, GLdouble width, GLdouble height, GLint viewport[4]*) seguito dalla funzione **gluPerspective** (con gli stessi parametri di proiezione definiti nella callback di reshape), in cui *x* e *y* sono le coordinate del puntatore rispetto al vertice in basso a sinistra della finestra, *width* e *height* le dimensioni del nuovo viewing volume (minori sono, più preciso è il picking, ma anche più difficoltoso per l'utente) e *viewport* un array di interi contenente una descrizione della viewport ottenuta con void **glGetIntegerv**(*GLenum pname, GLint* params*) ,
pname=GL_VIEWPORT .

Specificare il buffer di selezione

Una volta predisposto il buffer di selezione (che è soltanto un array di GLint), bisogna indicare a OpenGL che quello è il buffer in cui memorizzare i risultati, con

```
void glSelectBuffer(GLsizei size, GLuint *buffer)
```

in cui size è la dimensione dell'array e buffer l'array stesso. La selezione produrrà un hit record per ogni primitiva che rientra nel viewing volume specificato con gluPickMatrix. Ogni hit record, che verrà memorizzato in buffer finchè ci sarà spazio, ha la seguente forma:

- Numero di nomi sul name stack quando si è verificato l'hit
- La minima e massima profondità della parte delle primitive grafiche che hanno intersecato il viewing volume (valore unsigned int, tanto più piccolo quanto più la primitiva grafica si avvicina all'osservatore)
- Il contenuto dello stack dei nomi al verificarsi dell'hit, a cominciare dall'elemento **più a monte nella struttura gerarchica**

Entrare in modalità selezione

Una volta specificato il buffer da utilizzare, si entra in modalità selezione chiamando (la prima volta) la funzione

GLint glRenderMode(*GLenum mode*)

in cui mode vale GL_SELECT (altri 2 possibilità: GL_RENDER e GL_FEEDBACK)

Il valore restituito dalla funzione varia a seconda di cosa valeva mode l'ultima volta che è stata chiamata: se mode era valeva GL_RENDER allora il valore ritornato sarà 0, se mode valeva GL_SELECT il valore ritornato è il numero di hit records scritti nel buffer di selezione.

A questo punto si salva la matrice di proiezione, si carica la matrice identità e si riduce il viewing volume con gluPickMatrix (specificando anche una matrice di proiezione), quindi si chiama la funzione di rendering della scena in modalità selezione (che non disegna la scena), operazione in cui verrà caricato/aggiunto un nome sullo stack per ogni primitiva o gruppo di primitive.

Infine si esce dalla selezione (sempre con glRenderMode chiamata la seconda volta, adesso con parametro GL_RENDER) e si processano i risultati

Esempio 1

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

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

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

```
// Define object names
```

```
#define SUN      1
```

```
#define MERCURY 2
```

```
#define VENUS    3
```

```
#define EARTH    4
```

```
#define MARS     5
```

```
Void RenderScene(void) {
```

```
// Clear the window with current clearing color
```

```
glClear(GL_COLOR_BUFFER_BIT |  
        GL_DEPTH_BUFFER_BIT);
```

```
// Save the matrix state and do the rotations
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glPushMatrix();
```

```
// Translate the whole scene out and into view
```

```
glTranslatef(0.0f, 0.0f, -300.0f);
```

```
// Initialize the names stack
```

```
glInitNames();
```

```
glPushName(0);
```

Esempio 1 (2)

// nomina e disegna la Terra

```
glColor3f(1.0f, 1.0f, 0.0f);  
glLoadName(SUN);  
glutSolidSphere(15.0,40,40);  
  
// nomina e disegna Mercurio  
glColor3f(0.5f, 0.0f, 0.0f);  
glPushMatrix();  
glTranslatef(24.0f, 0.0f, 0.0f);  
glLoadName(MERCURY);  
glutSolidSphere(2.0,40,40);  
glPopMatrix();
```

```
glColor3f(0.5f, 0.5f, 1.0f);  
glPushMatrix();  
glTranslatef(60.0f, 0.0f, 0.0f);  
glLoadName(VENUS);  
glutSolidSphere(4.0,40,40);  
glPopMatrix();  
glColor3f(0.0f, 0.0f, 1.0f);  
glPushMatrix();  
glTranslatef(100.0f,0.0f,0.0f);  
glLoadName(EARTH);  
glutSolidSphere(8.0,40,40);  
glPopMatrix();
```

Esempio 1 (3)

```
// Marte

glColor3f(1.0f, 0.0f, 0.0f);

glPushMatrix();

glTranslatef(150.0f, 0.0f, 0.0f);

glLoadName(MARS);

glutSolidSphere(4.0,40,40);

glPopMatrix();

glPopMatrix();// matrice Modelview iniziale

glutSwapBuffers();

}
```

```
Void MouseCallback(int button, int state, int x, int y)

{

if(button == GLUT_LEFT_BUTTON && state ==
    GLUT_DOWN)

ProcessSelection(x, y);

}

#define BUFFER_LENGTH 64

void ProcessSelection(int xPos, int yPos){

GLfloat fAspect;

// buffer di selezione

static GLuint selectBuff[BUFFER_LENGTH];

// contatore degli Hit e vettore per lo stato della
    viewport

GLint hits, viewport[4];
```

Esempio 1 (4)

// seleziona il buffer per la selezione

```
glSelectBuffer(BUFFER_LENGTH, selectBuff);
```

// ritrova le dimensioni della viewport

```
glGetIntegerv(GL_VIEWPORT, viewport);
```

```
glMatrixMode(GL_PROJECTION);
```

```
glPushMatrix();
```

// cambia la modalità di rendering

```
glRenderMode(GL_SELECT);
```

```
glLoadIdentity();
```

```
gluPickMatrix(xPos, viewport[3] - yPos, 2,2,  
             viewport);
```

// Applica una proiezione prospettica

```
fAspect = (float)viewport[2] / (float)viewport[3];
```

```
gluPerspective(45.0f, fAspect, 1.0, 425.0);
```

// disegna la scena

```
RenderScene();
```

// ripristina la matrice di proiezione

```
glMatrixMode(GL_PROJECTION);
```

```
glPopMatrix();
```

```
glMatrixMode(GL_MODELVIEW);
```

Esempio 1 (5)

```
// ritrova il numero di hits
```

```
hits = glRenderMode(GL_RENDER);
```

```
if(hits == 1)
```

```
ProcessPlanet(selectBuff[3]);
```

```
else
```

```
glutSetWindowTitle("Nothing was clicked on!");
```

```
}
```

```
void SetupRC(){
```

```
glEnable(GL_DEPTH_TEST);
```

```
glFrontFace(GL_CCW);
```

```
glEnable(GL_CULL_FACE);
```

```
glClearColor(0.60f, 0.60f, 0.60f, 1.0f );
```

```
}
```

```
void ChangeSize(int w, int h){
```

```
GLfloat fAspect;
```

```
// previene una divisione per zero
```

```
if(h == 0)
```

```
h = 1;
```

```
// setta le dimensioni della viewport
```

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

```
// calcola l'aspect ratio della finestra
```

```
fAspect = (GLfloat)w/(GLfloat)h;
```

```
glMatrixMode(GL_PROJECTION);
```

```
glLoadIdentity();
```

Esempio 1 (6)

```
// Field of view di 45 gradi, piani near and far 1.0  
    and 425
```

```
gluPerspective(45.0f, fAspect, 1.0, 425.0);
```

```
// reset della matrice Modelview
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity();
```

```
}
```

```
int main(int argc, char* argv[]){
```

```
    glutInit(&argc, argv);
```

```
    glutInitDisplayMode(GLUT_DOUBLE |  
        GLUT_RGB | GLUT_DEPTH);
```

```
    glutInitWindowSize(800,600);
```

```
    glutCreateWindow("Pick a Planet");
```

```
    glutReshapeFunc(ChangeSize);
```

```
    glutMouseFunc(MouseCallback);
```

```
    glutDisplayFunc(RenderScene);
```

```
    SetupRC();
```

```
    glutMainLoop();
```

```
    return 0;
```

```
}
```

Picking gerarchico

OpenGL prevede la possibilità di salvare nello stack dei nomi più etichette corrispondenti alla gerarchia degli oggetti della scena. Nell'esempio precedente lo stack dei nomi è stato manipolato esclusivamente con `glLoadName`, che modifica il nome in cima allo stack e rappresenta il nome dell'oggetto nel suo insieme. E' possibile usare **`glPushName`** in modo tale da aggiungere allo stack più nomi (etichette) delle varie parti di un oggetto.

Nel prossimo esempio viene mostrato un possibile utilizzo di tale tecnica: sulla scena sono presenti il pianeta Terra e Marte, ciascuno con le sue lune; facendo click su una luna il titolo della finestra sarà modificato e mostrerà il nome del pianeta a cui la luna appartiene, oltre che al nome della luna

Esempio 2

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

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

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

```
#include<string.h>
```

```
// nomi degli oggetti
```

```
#define EARTH    1
```

```
#define MARS     2
```

```
#define MOON1    3
```

```
#define MOON2    4
```

```
void RenderScene(void)
```

```
{
```

```
    glClear(GL_COLOR_BUFFER_BIT |  
            GL_DEPTH_BUFFER_BIT);
```

```
    glMatrixMode(GL_MODELVIEW);
```

```
    glPushMatrix();
```

```
    // trasformazione di viewing
```

```
    glTranslatef(0.0f, 0.0f, -300.0f);
```

```
    // inizializza lo stack dei nomi
```

```
    glInitNames();
```

```
    glPushName(0);
```

Esempio 2 (2)

// disegna la Terra

```
glPushMatrix();  
  
glColor3f(0.0f, 0.0f, 1.0f);  
  
glTranslatef(-100.0f,0.0f,0.0f);  
  
glLoadName(EARTH);  
  
glutSolidSphere(30.0f,40.0f,40.0f);
```

// disegna la Luna

```
glTranslatef(45.0f, 0.0f, 0.0f);  
  
glColor3f(0.85f, 0.85f, 0.85f);  
  
glPushName(MOON1);  
  
glutSolidSphere(5.0f,40.0f,40.0f);  
  
glPopName(); glPopMatrix();
```

// disegna Marte

```
glColor3f(1.0f, 0.0f, 0.0f);  
  
glPushMatrix();  
  
glTranslatef(100.0f, 0.0f, 0.0f);  
  
glLoadName(MARS);  
  
glutSolidSphere(20.0f,40.0f,40.0f);
```

// disegna la prima luna

```
glTranslatef(-40.0f, 40.0f, 0.0f);  
  
glColor3f(0.85f, 0.85f, 0.85f);  
  
glPushName(MOON1);  
  
glutSolidSphere(5.0f,40.0f,40.0f);  
  
glPopName();
```

Esempio 2 (3)

```
// disegna la seconda luna
glTranslatef(0.0f, -80.0f, 0.0f);
glPushName(MOON2);
glutSolidSphere(5.0f,40.0f,40.0f);
glPopName();
glPopMatrix();

// ripristina la trasformazione iniziale
glPopMatrix();// matrice Modelview
glutSwapBuffers();
}
```

```
#define BUFFER_LENGTH 64

void ProcessSelection(int xPos, int yPos) {
    GLfloat fAspect;    // aspect ratio
    // selection buffer
    GLuint selectBuff[BUFFER_LENGTH];

    // contatore degli Hit e array della viewport
    GLint hits, viewport[4];

    glSelectBuffer(BUFFER_LENGTH, selectBuff);

    // ritrova le dimensioni della viewport
    glGetIntegerv(GL_VIEWPORT, viewport);
```

Esempio 2 (4)

// cambia lo stack corrente e salva la matrice di proiezione

```
glMatrixMode(GL_PROJECTION);
```

```
glPushMatrix();
```

// cambia la modalità di rendering

```
glRenderMode(GL_SELECT);
```

```
glLoadIdentity();
```

```
gluPickMatrix(xPos, viewport[3] - yPos, 2,2,  
             viewport);
```

```
fAspect = (float)viewport[2] / (float)viewport[3];
```

```
gluPerspective(45.0f, fAspect, 1.0, 425.0);
```

// disegna la scena

```
RenderScene();
```

// ripristina la matrice di proiezione

```
glMatrixMode(GL_PROJECTION);
```

```
glPopMatrix();
```

```
glMatrixMode(GL_MODELVIEW);
```

// ritrova il numero di hits

```
hits = glRenderMode(GL_RENDER);
```

Esempio 2 (5)

```
// se ho selezionato qualcosa, controlla cosa ho  
    selezionato
```

```
if(hits == 1)
```

```
    ProcessPlanet(selectBuff);
```

```
else
```

```
    glutSetWindowTitle("You clicked empty  
        space!");
```

```
}
```

```
void MouseCallback(int button, int state, int x, int  
    y)
```

```
{
```

```
    if(button == GLUT_LEFT_BUTTON && state  
        == GLUT_DOWN)
```

```
        ProcessSelection(x, y);}
```

```
void ProcessPlanet(GLuint *pSelectBuff){
```

```
    int id,count;
```

```
    char cMessage[64];
```

```
    strcpy(cMessage,"Error, no selection detected");
```

```
    // quanti nomi sul name stack
```

```
    count = pSelectBuff[0];
```

```
    // valore più in basso sul name stack
```

```
    id = pSelectBuff[3];
```

```
    switch(id){
```

```
        case EARTH:
```

```
            strcpy(cMessage,"You clicked Earth.");
```

Esempio 2 (6)

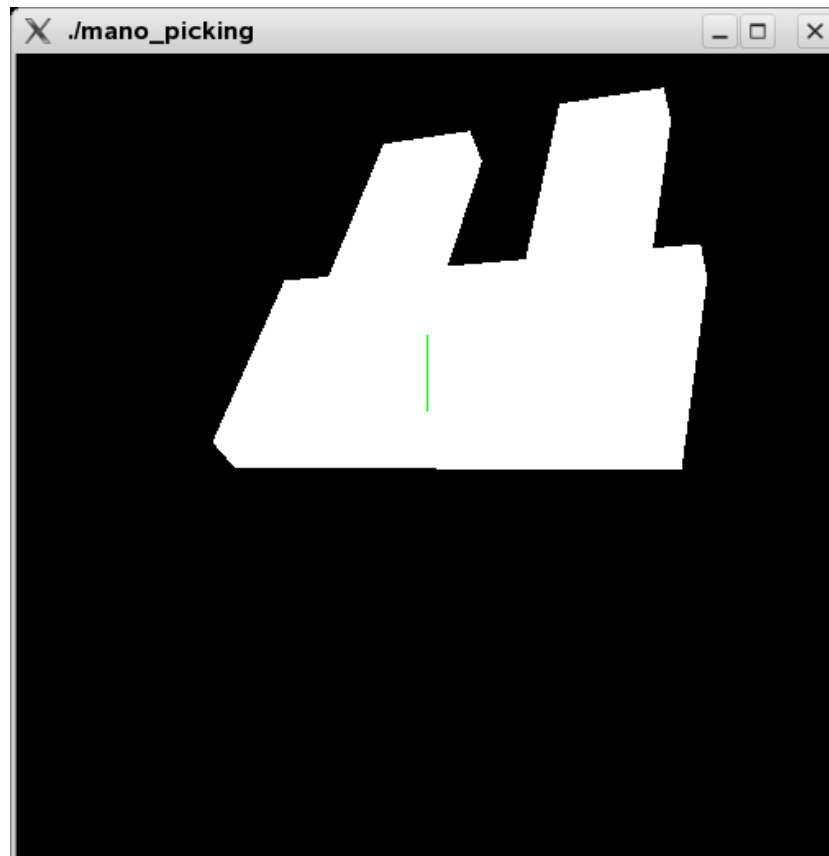
```
if(count == 2)
    strcat(cMessage," - Specifically the moon.");
break;
case MARS:
    strcpy(cMessage,"You clicked Mars.");
// sappiamo che lo stack dei nomi era profondo 2
// la luna scelta sarà in pSelectBuff[4]
if(count == 2){
    if(pSelectBuff[4] == MOON1)
        strcat(cMessage," - Specifically Moon #1.");
```

```
Else
    Strcat(cMessage," - Specifically Moon #2.");
}
break;
}
glutSetWindowTitle(cMessage);
}
void SetupRC(),
void ChangeSize(int w, int h)
int main(int argc, char **argv)
come nell'esempio precedente
```

Esercizio

Modificare l'esercizio della mano introducendo la funzionalità di picking. Quando con il mouse si seleziona un elemento della mano (polso, dito1, dito2) il programma visualizza un messaggio che contiene il nome dell'elemento selezionato e riporta la mano nella sua configurazione iniziale (polso lungo l'asse x).

Osservazione: visualizzare gli elementi della mano come solidi pieni e non in wireframe (perche' altrimenti il picking funziona solo sui bordi)



Migliorare le prestazioni: Display list e Vertex array

In un' implementazione di OpenGL un possibile “collo di bottiglia” a sfavore delle prestazioni è la fase di traduzione dei comandi che disegnano le primitive in istruzioni hardware. Quando si hanno molti oggetti, più o meno complessi, ma con geometria fissa è possibile risparmiare il tempo necessario a compilare i comandi (glVertex ecc...) raggruppandoli in una display list, alla quale è associato un numero intero utilizzato per richiamarla. Una display list è quindi costituita da un gruppo di comandi OpenGL immagazzinati per una successiva esecuzione. Il vantaggio sta nell'avere insieme di comandi già tradotti in istruzioni pronte per essere eseguite dall'hardware.

Funzioni per la manipolazione delle display list:

`void glNewList(GLuint list, GLenum mode)`

crea una nuova lista identificata da list, secondo la modalità mode, che può valere GL_COMPILE (memorizza solo i comandi) o GL_COMPILE_AND_EXECUTE (memorizza ed esegue i comandi). Questa funzione inizia anche la memorizzazione dei comandi, che termina con void **glEndList()**.

Migliorare le prestazioni: Display list e Vertex array (2)

Void **glCallList**(*GLuint list*)

esegue i comandi memorizzati nella display list identificata da *list*

void **glCallLists**(*GLsizei n, GLenum type, const GLvoid *lists*)

chiama le *n* display list memorizzate in *lists* (che contiene valori di tipo *type*, ad es: GL_BYTE, GL_SHORT...)

GLuint **glGenLists**(*GLsizei range*)

genera un numero *range* identificativi contigui per display list e restituisce il primo. Lo scopo è quello di procurarsi *n* display list vuote da utilizzare successivamente e di evitare di assegnare manualmente valori che potrebbero già essere utilizzati in altre occasioni.

Migliorare le prestazioni:

Display list e Vertex array (2)

GLboolean **glIsList**(*GLuint list*)

restituisce true se l'intero passato come argomento è già associato ad una lista, false altrimenti.

void **glDeleteLists**(*GLuint list, GLsizei range*)

cancella un numero range di display list a partire da list e libera i corrispondenti id.

Vertex array

I vertex array permettono di utilizzare un numero di chiamate (glVertex ecc...) decisamente inferiore in confronto a quanto visto fino ad ora. Al contrario delle d. l., i vertex array si adattano anche a casi in cui la geometria rappresentata non è fissa. L'idea alla base dell'utilizzo dei vertex array è quella di memorizzare in uno o più array le informazioni relative a vertici, colori, normali, texture ecc... per tutti i poligoni che formano un oggetto (fissi o modificabili), quindi con poche chiamate utilizzare il contenuto di tutti gli array senza specificare ogni singolo vertice, normale ecc...

Esempio 3

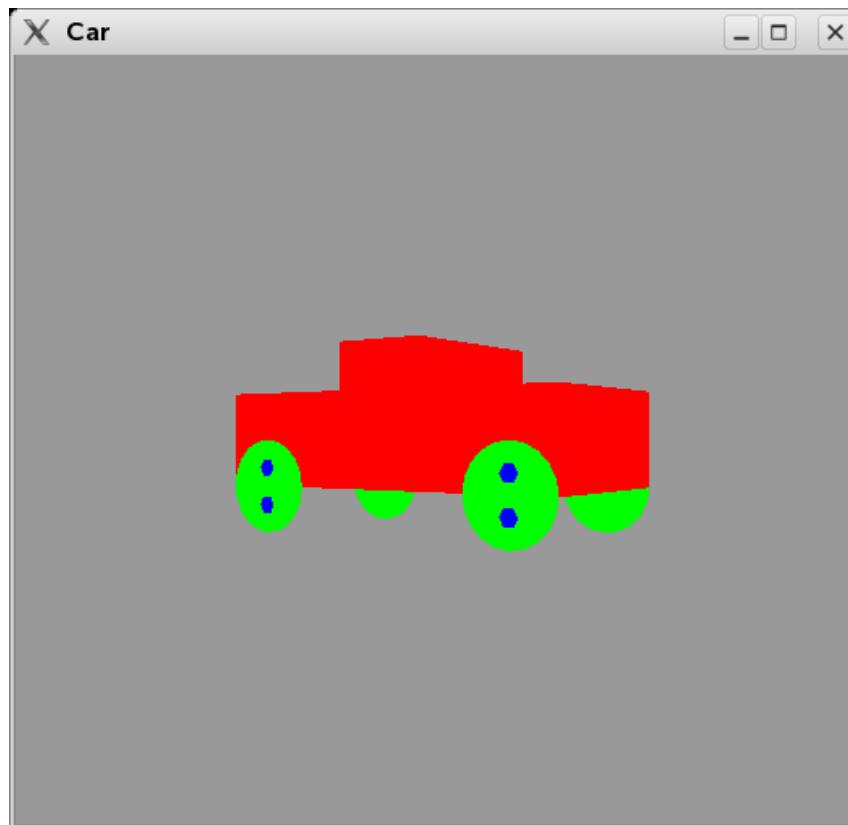
Esempio di Display List. Si definisce una Display List per disegnare un triangolo rosso e successivamente si disegnano 10 triangoli richiamando la display list. La display list contiene una traslazione lungo x per consentire il posizionamento del triangolo successivo rispetto al triangolo corrente. Successivamente si disegna un segmento orizzontale. Il segmento viene disegnato in rosso perche' il rosso e' lo stato del colore corrente che viene impostato all'interno della display list.



Esercizio

Il programma “auto.cpp” disegna il corpo di un'auto e imposta una rotazione rispetto all'asse y controllata da tastiera. Inoltre viene definita una display list per disegnare le viti (cerchi blu) delle ruote. Completare il programma scrivendo

- 1) il codice di una display list per disegnare una ruota. Una ruota e' un cerchio verde sopra il quale vengono applicate due viti come mostrato in figura.
- 2) il codice per disegnare 4 ruote posizionate alle estremita' della macchina.



Funzioni vertex array

void **glEnableClientState**(*GLenum array*) / **glDisableClientState**

abilitano o disabilitano gli array specificati dall'argomento, per specificare un array di vertici si utilizza il parametro `GL_VERTEX_ARRAY`

void **glVertexPointer**(*GLint size, GLenum type, GLsizei stride, const GLvoid *pointer*)

definisce l'array di vertici (pointer), il numero di vertici per coordinata (size), il tipo delle coordinate (type, ad es `GL_FLOAT`), l'offset di byte tra i vertici nell'array; un valore pari a 0 indica che non c'è spazio tra i dati di un vertice e il successivo.

void **glDrawArrays**(*GLenum mode, GLint first, GLsizei count*)

esegue il rendering di primitive di tipo mode (ad es. `GL_POINTS`) partendo dai dati all'indice first nell'array abilitato, fino all'indice count (in questo caso count è il numero di vertici, non di valori nell'array)

Esempio 4

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

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

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

```
#include <math.h>
```

```
typedef GLfloat GLTVector2[2];
```

```
#define SMALL_STARS 150
```

```
GLTVector2 vSmallStars[SMALL_STARS][2];
```

```
#define MEDIUM_STARS 40
```

```
GLfloat vMediumStars[MEDIUM_STARS][2];
```

```
#define LARGE_STARS 15
```

```
GLTVector2 vLargeStars[LARGE_STARS][2];
```

```
#define SCREEN_X 800
```

```
#define SCREEN_Y 600
```

```
void SetupRC(){
```

```
int i;
```

```
for(i = 0; i < SMALL_STARS; i++)
```

```
{
```

```
    vSmallStars[i][0] = (GLfloat)(rand() %  
    SCREEN_X);
```

```
    vSmallStars[i][1] = (GLfloat)(rand() %  
    (SCREEN_Y - 100))+100.0f;
```

```
}
```

Esempio 4 (2)

```
for(i = 0; i < MEDIUM_STARS; i++)  
{  
    vMediumStars[i][0] = (GLfloat)(rand() %  
    SCREEN_X * 10)/10.0f;  
    vMediumStars[i][1] = (GLfloat)(rand() %  
    (SCREEN_Y - 100))+100.0f;  
}  
  
for(i = 0; i < LARGE_STARS; i++)  
{  
    vLargeStars[i][0] = (GLfloat)(rand() %  
    SCREEN_X*10)/10.0f;  
    vLargeStars[i][1] = (GLfloat)(rand() %  
    (SCREEN_Y - 100)*10.0f)/ 10.0f +100.0f;  
}  
  
void RenderScene(void) {  
    GLfloat x = 700.0f; //posizione e raggio della  
    luna  
    GLfloat y = 500.0f;  
    GLfloat r = 50.0f;  
    GLfloat angle = 0.0f;  
    glClear(GL_COLOR_BUFFER_BIT |  
    GL_DEPTH_BUFFER_BIT);  
}
```

Esempio 4 (3)

```
glColor3f(1.0f, 1.0f, 1.0f);  
  
glEnableClientState(GL_VERTEX_ARRAY);  
  
glPointSize(1.0f);  
  
glVertexPointer(2, GL_FLOAT, 0, vSmallStars);  
  
glDrawArrays(GL_POINTS, 0, SMALL_STARS);  
  
glPointSize(3.05f);  
  
glVertexPointer(2, GL_FLOAT, 0, vMediumStars);  
  
glDrawArrays(GL_POINTS, 0, MEDIUM_STARS);  
  
glPointSize(5.5f);  
  
glVertexPointer(2, GL_FLOAT, 0, vLargeStars);  
  
glDrawArrays(GL_POINTS, 0, LARGE_STARS);
```

```
glBegin(GL_TRIANGLE_FAN);  
  
    glVertex2f(x, y);  
  
    for(angle = 0; angle < 2.0f * 3.141592f;  
        angle += 0.1f)  
        glVertex2f(x + (float)cos(angle) * r, y +  
            (float)sin(angle) * r);  
  
    glVertex2f(x + r, y);  
  
glEnd();  
  
glutSwapBuffers();  
  
}
```

aggiungere void ChangeSize(int w, int h) con proiezione ortogonale 2d e int main(int argc, char **argv) come solito

Curve e Superfici NURBS

GLU offre una interfaccia di alto livello per rappresentare curve e superfici NURBS (Non-Uniform Rational B-Spline).

- 1) Si utilizza la funzione **gluNewNurbsRenderer()** per creare un puntatore ad un oggetto di tipo NURBS.
- 2) **gluNurbsProperty()** è utilizzata per definire le proprietà di rendering, come ad esempio il massimo numero di linee da utilizzare nell'approssimazione della curva.
- 3) Per specificare la primitiva si utilizza la funzione **gluNurbsCurve()** (**gluNurbsSurface()**) che accetta come parametri i punti di controllo, i pesi, la sequenza dei nodi e l'ordine delle basis function.
- 3) Le funzioni precedenti devono essere contenute all'interno di due chiamate che inizializzano e terminano la rappresentazione della curva o della superficie: **gluBeginCurve()** o (**gluBeginSurface()**) e **gluEndCurve()** o **gluEndSurface()**

Curve e Superfici NURBS

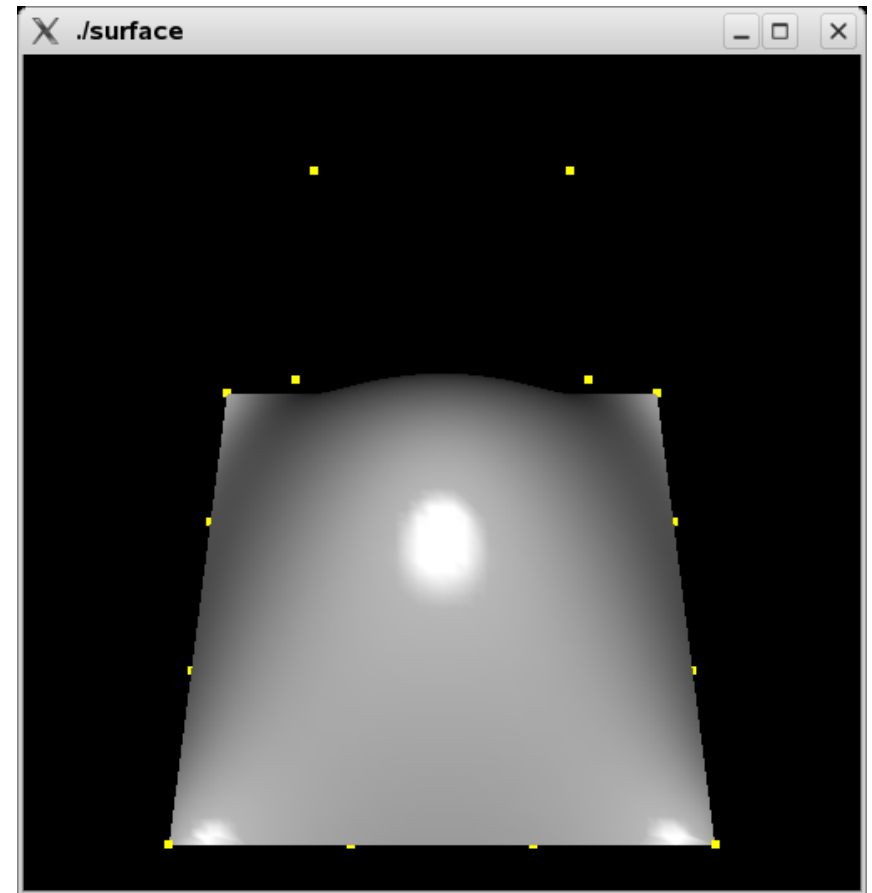
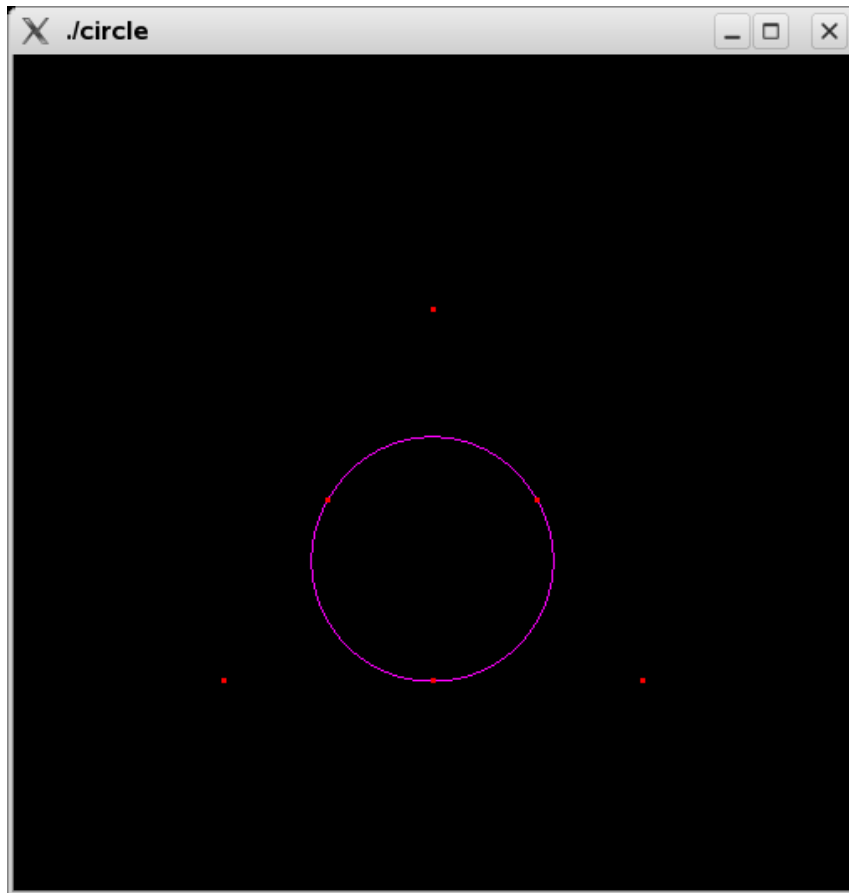
Prototipo della funzione gluNurbsCurve:

```
void gluNurbsCurve(GLUnurbsObj *nobj, // puntatore alla curva
                  GLint nknots,    // numero di nodi, nodi= pti_controllo + ordine
                  GLfloat *knot,   // vettore non decrescente di nodi
                  GLint stride,    // offset tra due successivi punti di controllo
                  GLfloat *ctlarray, // vettore di punti di controllo
                  GLint order, // ordine della curva= grado + 1
                  GLenum type) // tipo di curva
```

Esempio di utilizzo:

```
gluBeginCurve(nobj);
    gluNurbsCurve(nobj, ..., GL_MAP1_VERTEX_4);
gluEndCurve(nobj);
```

Curve e Superfici NURBS



Esercizio

Completare il programma `telecamera_movimento.cpp` nelle funzioni “SpecialKeys” e “Draw” utilizzando la funzione `gluLookAt` in modo che la telecamera possa essere traslata e ruotata nello spazio premendo i tasti cursore.

La telecamera e' solidale ad un osservatore che si muove nella scena.