Costruzione di Interfacce
Lezione 12
Esercitazione C++,

cignoni@iei.pi.cnr.it
http://vcg.iei.pi.cnr.it/~cignoni

Introduzione

- Oggi cercheremo di costruire da zero gli elementi di base di una gui, in particolare uno slider, cercando di capire quali siano le problematiche che ci sono dietro.
- Lo scopo è costruire, rimanendo in glut, un programmino che permetta di modificare interattivamente alcuni parametri di rendering di opengl

Skeleton Glut (tanto per cambiare)

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

void myRedrawFunc()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glutSwapBuffers();
}
void myIdle() { glutPostRedisplay();}
int main(int argc, char *argv[])
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
   glutIdleFunc(myIdle);
   glutCreateWindow("Colors");
   glutDisplayFunc(myRedrawFunc);
   glutMainLoop();     //Passare al SO il controllo.
   return 0;
}
```

Init Opengl Ragionevole

```
void myInitGL()
{
   // Abilita la possibilita' di cambiare il alcune componenti colore del
   // materiale tramite un semplice glcolor, anzichè fare glMaterial(...)
   glColorMaterial(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE);
   glEnable(GL_COLOR_MATERIAL);

   // Abilita il calcolo dell'illuminazione
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);

   // anche le normali attraversano la pipeline, quindi devono essere
   // rinormalizzate, prima del calcolo dell'illuminazione, altrimenti
   // il calcolo dell'illuminazione viene sbagliato.
   glEnable(GL_NORMALIZE);

   // Abilita il test sullo zbuffer e quindi l'eliminazione delle
   // superfici nascoste
   glEnable(GL_DEPTH_TEST);
}
```

Scopo dell'Esercitazione

Provare interattivamente le varie proprietà dei materiali di opengl

Il che implica:

- utilizzare qualche meccanismo di interazione diretto tra l'applicazione e l'utente;
- Usare il mouse
- Implementare uno slider

Glut e mouse

Al solito Callbacks

- glutMouseFunc(button,state,x,y);
  - Chiamata quando cambia lo stato di uno dei bottoni del mouse
- glutMotionFunc(button,state,x,y);
  glutPassiveMotionFunc(button,state,x,y)
  - Chiamate quando il mouse si muove con o senza I bottoni premuti.

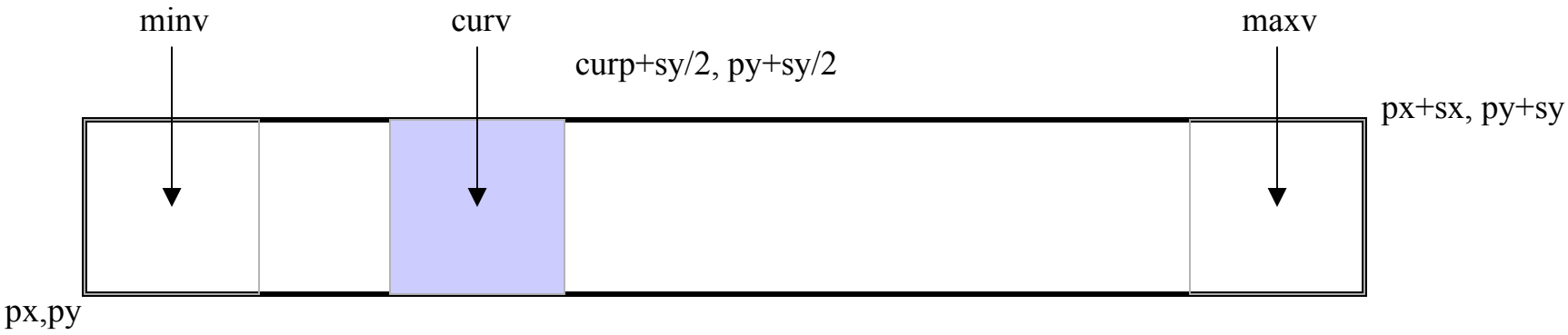
- Non è l'unica interfaccia possibile
- Windows struttura I messaggi in maniera differente da glut

Slider

- Vogliamo mediare in maniera un po'pulita l'input dell'utente, tramite il mouse,
- Costruiremo da zero uno slider che serva a mappare il movimento del mouse in una certa regione della finestra nel cambiamento di qualche proprieta' della nostra applicazione

La geometria di uno slider

- Prima cosa ragioniamo su come è fatto uno slider e cosa serve per memorizzarlo, disegnarlo e capire dove sto cliccando.



Interfaccia di uno slider

- Stato dello slider
  - Dove:
    Posizione e dimensione sulla finestra:
    - In che sistema di riferimento?
    - Scegliamo quello opengl (0,0 in basso a sx)
  - Come:
    range di valori su cui deve funzionare
  - Valore corrente
  - Altro (e.g. se stiamo draggando ecc.)

Interfaccia dello slider

- Funzioni di modifica diretta dello stato dello slider
  - SetPos(); // pos e dim nella finestra
  - SetRange();
  - SetVal();
- Interrogazione
  - GetVal();

Interfaccia Interattiva

- Gestione eventi mouse, primo abbozzo
  - MouseDown();
  - MouseUp();
  - MouseMove();

Prima interfaccia di glSlider

```
class glSlider
{
public:
   glSlider(void);
   ~glSlider(void);
   void SetPos(int posx, int posy, int sizex, int sizey);
   void Draw();

   void SetRange(float minval, float maxval);
   float  GetVal();
   void SetVal(int val);

   void MouseDown(int mx, int my);
   void MouseMove(int mx, int my);
   void MouseUp(int mx, int my);
private:
   // Size and position
   int px,py,sx,sy;

   float minv,maxv; // range
   float curv;      // current position
};
```

Prima implementazione

- Scriviamo subito le funzioni facili:

```
void glSlider::SetRange(float minval, float maxval)
{
   minv=minval;
   maxv=maxval;
}

void glSlider::SetVal(float val)
{
   curv=val;
}

void glSlider::SetPos(int posx, int posy, int sizex, int sizey)
{
   px=posx;
   py=posy;
   sx=sizex;
   sy=sizey;
}
```

La funzione di Draw (dirty!)

```
void glSlider::Draw()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0,300,0,300,-1,1);  // ATTENZIONE HACK
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

  int curp=CurPos();

    glDisable(GL_LIGHTING);
    glBegin(GL_LINE_LOOP);
          glVertex2i(px   , py   );
          glVertex2i(px+sx, py   );
          glVertex2i(px+sx, py+sy);
          glVertex2i(px   , py+sy);
  glEnd();
    glBegin(GL_QUADS);
          glVertex2i(curp-sy/2,py);
          glVertex2i(curp+sy/2,py);
          glVertex2i(curp+sy/2,py+sy);
          glVertex2i(curp-sy/2,py+sy);
  glEnd();
}
```

Caveat

- L’implementazione fatta finora contiene alcune parti palesemente sbagliate/arrangiate non generali (la draw!).
- Pero’ serve per testare il resto.
- L’importante è ricordarselo.

Primo test

- Aggiungiamo una variabile globale di tipo glSlider al nostro prog, un po' di init nel main

```
// Variabili Globali
glSlider tt; // Lo Slider
…
int main(int argc, char *argv[]){
…
    tt.SetPos(5,5,200,50);
    tt.SetRange(0,100);
    tt.SetVal(50);
…
}
```

- E proviamo a disegnarlo

```
void myRedrawFunc()
{
    glClear(GL_COLOR_BUFFER_BIT);
    tt.Draw();
   glutSwapBuffers();
}
```

Prima ripulitura

- La draw non era fatta bene
- Assunzione sulle matrici di proj che troviamo:
  - glOrto con coord == alla dim della finestra in pixel
- Aggiungiamo la gestione della resize, e memorizziamoci le dim della finestra in due var globali

```
// Stato della finestra corrente
int WinW,WinH;

void myReshapeFunc(int w, int h){
    WinW=w;
    WinH=h;
    glViewport(0,0,w,h);
}
int main(int argc, char *argv[]){
…
    glutReshapeFunc(myReshapeFunc);
…
}
```

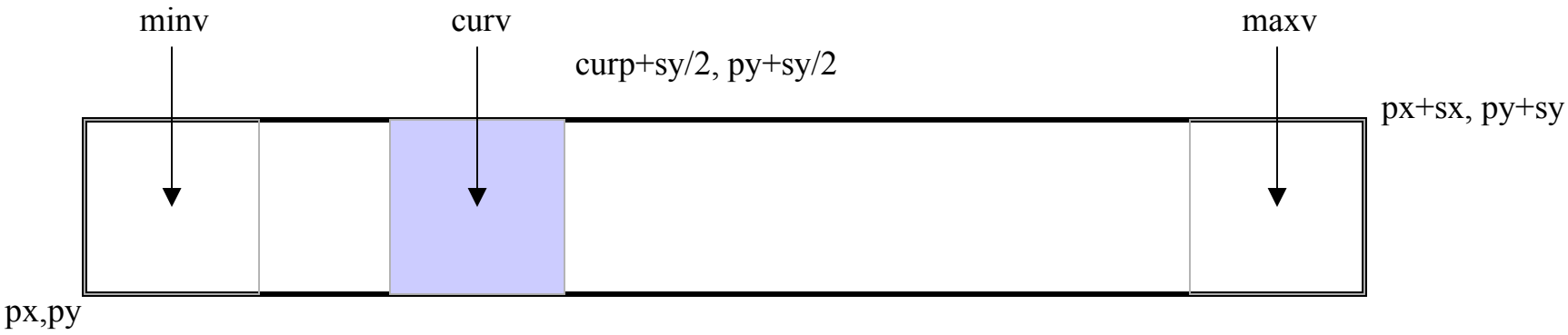
Nuova glSlider::Draw e display

```
void myRedrawFunc()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0,WinW,0,WinH,-1,1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    tt.Draw();
    glutSwapBuffers();
}

void glSlider::Draw()
{
  int curp=CurPos();

  glDisable(GL_LIGHTING);
…
Come prima ma senza l'inizializzazione delle matrici
…
}
```

Hit Test

- Di sicuro ci serve un test per sapere se ho colpito con il mouse il nostro slider

```
bool glSlider::Hit(int x, int y)
{
    if(x>=px && y >=py)
         if(x<px+sx && y<py+sy) return
    true;
    return false;
}
```



Gestione mouse

- Aggiungiamo le callback di glut per il mouse,
- Notare l'inversione della y dovuta ai differenti sistemi di riferimento
  - Mouse ha lo 0 in alto a sx
  - Opengl ha lo 0 in basso a sx

```
main()
…
    glutMouseFunc(myMouseFunc);
    glutMotionFunc(myMotionFunc);
…

void myMouseFunc(int button, int state, int x, int y){
    if(button==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
          if(tt.Hit(x,WinH-y)) tt.MouseDown(x,WinH-y);
    }
}
void myMotionFunc(int x, int y){}
```

Problema

- A questo punto conviene fermarsi e prima di continuare l'implementazione farsi alcune domande:
  - Cosa succede se voglio avere più slider?
  - Chi distribuisce gli eventi tra i vari slider della finestra?
  - Ogni slider ha diritto a ricevere eventi anche quando il mouse non e' sopra di lui?
  - Occorre una classe che distribuisca gli eventi bene…

Classe glDialogManager

- Aggiungiamo a questo scopo un'altra classe conosca tutti i widget nella finestra
  - Ha una lista di puntatori a glSlider
- Distribuisce bene gli eventi del mouse
  - Il nostro programma passa gli eventi del mouse al manager che fa tutto lui
- Si occupa di disegnare tutti i widget

glDialogManager

```
class glDialogManager
{
public:
   glDialogManager(void);
   ~glDialogManager(void);
   void AddSlider(glSlider *sl);
   void Draw();

   void MouseDown(int mx, int my);
   void MouseMove(int mx, int my);
   void MouseUp(int mx, int my);

private:
   list<glSlider *> L;
};
```

Memento sulle stl

- Il dialog manager usa una lista stl per tenere i puntatori a tutti gli slider che deve gestire
- Le liste stl sono contenitori generici templatati su un tipo scelto dall'utente
- push_back per aggiungere un elemento ad un container.
- Iteratori (e.g. astrazione di puntatori) per accedere agli elementi di un contenitore generico.

Iteratori STL

- Per scandire gli elementi di una lista occorre un iteratore a quel tipo di lista e poi basta un semplice for.

```
list<glSlider *>::iterator li;
for(li=L.begin();li!=L.end();++li)
        DoSomethingWith(*li)
```

Draw degli Slider

- Gestita dal DialogManager
- Semplicemente un for:

```
void glDialogManager::Draw()
{
   list<glSlider *>::iterator li;
   for(li=L.begin();li!=L.end();++li)
               (*li)->Draw();
}
```

Gestione degli eventi

- Cosa succede quando si clicca su uno slider
  - Lo slider colpito diventa “attivo”
  - Lo slider attivo (ha il ***focus***) finché il mouse è premuto riceve tutti gli eventi del mouse (anche se il mouse è fuori dall’area dello slider
  - Quando si rilascia il bottone del mouse si smette di aggiornare lo slider

Gestione Focus

- Si aggiunge un campo Focus al DialogManager, che contiene un puntatore allo slider che ha correntemente il focus.
- Il campo Focus viene inizializzato dal costruttore, e modificato dalla MouseDown e dalla MouseUp

```
class glDialogManager
{
…
private:
…
    glSlider *Focus;
};
```

Gestione Focus

```
glDialogManager::glDialogManager(void){ Focus=0; }

void glDialogManager::MouseDown(int mx,int my){
   list<glSlider *>::iterator li;
   bool found=false;
   for(li=L.begin();li!=L.end() && !found;++li)
        {
                if((*li)->Hit(mx,my))
                {
                        Focus=*li;
                        Focus->MouseDown(mx,my);
                }
        }
}

void glDialogManager::MouseMove(int mx,int my){
   if(!Focus) return;
   else Focus->MouseMove(mx,my);
}
void glDialogManager::MouseUp(int mx,int my) { Focus=0; }
```

Gestione Mouse nello Slider

- Cambio di sistema di riferimento tra spazio del mouse nella finestra e valori dello slider
- Conviene aggiungere due funzioni che facciano la conversione di frame

```
class glSlider
{
private:
…
   int Val2Pos(float val);
   float Pos2Val(int pos);
 …
}
```

Conversione Frame

```
int glSlider::Val2Pos(float val)
{
   int minp=px+sy/2;
   int maxp=px+sx-sy/2;
   if(val<minv) return minp;
   if(val>maxv) return maxp;
   return (minp+((val-minv)/(maxv-minv))*float(maxp-
   minp));
}

float glSlider::Pos2Val(int pos)
{
   int minp=px+sy/2;
   int maxp=px+sx-sy/2;
   if(pos<minp) return minv;
   if(pos>maxp) return maxv;
   return minv+(float(pos-minp)/(maxp-minp))* (maxv-
   minv);
}
```

Gestione Mouse

- MouseDown e MouseUp in pratica non fanno nulla (avrebbero un senso se si volesse ad esempio cambiare il modo in cui si disegna lo slider mentre si dragga)
- MouseMove semplicemente converte da spazio di mouse a spazio di valori dello slider

Gestione Mouse

```
void glSlider::MouseDown(int mx,int my)
{
}

void glSlider::MouseMove(int mx,int my)
{
   float newval=Pos2Val(mx);
   SetVal(newval);
}

void glSlider::MouseUp(int mx,int my)
{
}
```

Callback

- Per gestire in maniera consistente gli slider un modo è quello di copiare glut e affidare ad ogni slider una callback da chiamare quando lo slider cambia valore;
- Una callback si memorizza con un puntatore a funzione…

Puntatori a funzione

- Caso semplice

```
void (*funcPtr)();
```

- Puntatore ad una funzione che non ha argomenti e ritorna void
- Trucco: Si leggono da dentro verso fuori, destra-sinistra-destra
  - `funcPtr` funcPtr…
  - `(*funcPtr)` … è un puntatore …
  - `(*funcPtr)();` … a funzione senza argomenti …
  - `void (*funcPtr)();` … che ritorna void

Puntatori a funzione

- `bool (*cbfunc)(float,float);`
- puntatore a funzione che prende in ingresso due float e ritorna un booleano

- `float (*(*fp2)(int,int,float))(int);`
- Puntatore a funzione che prende in ingresso due interi e un float e ritorna un puntatore ad una funzione che prende in ingresso un intero e ritorna un float

Slider Con Callback

- Aggiungiamo quindi alla nostra classe

```
class glSlider
{
public:
…
   void SetCallback(void (*func)(float));
…
private:
   void (*cbfunc)(float);
…
}
```

Slider con CallBack

```
glSlider::glSlider(void)
{
   cbfunc=0;
}
…
void glSlider::SetCallback(void (*func)(float ))
{
   cbfunc=func;
}
…
void glSlider::MouseMove(int mx,int my)
{
   float newval=Pos2Val(mx);
   SetVal(newval);
   if(cbfunc) cbfunc(newval);
}
```

Tutto in pratica!

- Usiamo gli slider per costruire un'app. in cui si possa modificare interattivamente, componente diffusa, speculare e shininess di un oggetto.
- Tre slider globali, ognuno con una sua callback che modifica una variabile globale usata nella funzione di redraw per definire le proprietà del materiale

Global

```
// Gli Slider
glSlider tt1,tt2,tt3;

// il Dialog Manager
glDialogManager DM;

// variabili gestite dagli slider
float Shininess=1;
float GraySpec=.5f;
float GrayDiff=.5f;

void SetShinyCallback(float val) {Shininess=val;}
void SetSpecCallback(float val)  {GraySpec=val; }
void SetDiffCallback(float val)  {GrayDiff=val; }
```

Nel Main

```
tt1.SetPos(10,10,200,30);
tt1.SetRange(0,255);
tt1.SetVal(50);
tt1.SetCallback(SetShinyCallback);

tt2.SetPos(10,50,200,30);
tt2.SetRange(0,1);
tt2.SetVal(.5f);
tt2.SetCallback(SetSpecCallback);

tt3.SetPos(10,90,200,30);
tt3.SetRange(0,1);
tt3.SetVal(.5f);
tt3.SetCallback(SetDiffCallback);

DM.AddSlider(&tt1);
DM.AddSlider(&tt2);
DM.AddSlider(&tt3);
```

Nella Display

```
void myRedrawFunc()
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0,WinW,0,WinH,-1,1);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   DM.Draw();  // Disegna tutti gli Slider
   myInitGL(); // Setta lo stato Opengl
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(45,1,1,10);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt(0,0,5,0,0,0,0,1,0);
   glRotatef(glutGet(GLUT_ELAPSED_TIME)/50,1,1,0);
   // glutSolidTeapot(1);
   glutSolidTorus(.7,1,60,120);
   glutSwapBuffers();
}
```

Settaggio dello stato Opengl

```
void myInitGL()
{
   // Settaggio materiale
   float spec[4]={GraySpec,GraySpec,GraySpec,1};
   glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR , spec);
   float diff[4]={GrayDiff,GrayDiff,GrayDiff,1};
   glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE , diff);
   glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,Shininess);

   // Abilita il calcolo dell'illuminazione
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);

   // anche le normali attraversano la pipeline, quindi devono essere
   // rinormalizzate, prima del calcolo dell'illuminazione, altrimenti
   // il calcolo dell'illuminazione viene sbagliato.
   glEnable(GL_NORMALIZE);

   // Abilita il test sullo zbuffer e quindi l'eliminazione delle
   // superfici nascoste
   glEnable(GL_DEPTH_TEST);
}
```