Costruzione di Interfacce
Lezione 24
SDL&QT, Scene Graphs e Xml for dummies

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

SDL e QT

- vogliamo integrare alcune feature di SDL con QT
  - SDL gestisce molto bene:
    - fullscreen e resize modalita video
  - QT gestisce molto bene
    - loading immagini
    - parsing xml ecc.
  - facciamo un mix dei due

SDLMoebius.pro

```
TEMPLATE = app
CONFIG += console debug release opengl
INCLUDEPATH += .;c:\code\SDL-1.2.6\include
LIBPATH += c:\code\SDL-1.2.6\lib
unix:LIBS += -lmath -L/usr/local/lib
win32:LIBS +=  sdlmain.lib sdl.lib

# Input
HEADERS += vcg\Matrix44.h vcg\Point3.h vcg\Point4.h
   vcg\Utility.h
SOURCES += main.cpp CIMoebius.cpp
```

main.cpp

```
int main(int argc, char **argv)
{
   QApplication QApp(argc, argv );
   SDL_Init(SDL_INIT_VIDEO) < 0 ); SDL_GL_SetAttribute
   (SDL_GL_DEPTH_SIZE, 24);
   SDL_SetVideoMode(640, 480, 0, SDL_OPENGL |SDL_FULLSCREEN);
   CIMoebius::myShape s;
   s.BuildNGon(3,1);
   ring.GenerateMoebiusRing(s,36,3,120);
   int done = 0, ret;
   myGLReshapeFunc(640,480);
   ...
```

main.cpp

```
while ( ! done ) {
    SDL_Event event;
     ret=SDL_PollEvent(&event);
          if(ret) switch(event.type){
          case SDL_QUIT    : done = 1;            break ;
          case SDL_KEYDOWN :
           if ( event.key.keysym.sym == SDLK_ESCAPE )
                                      done = 1;
            if ( event.key.keysym.sym == SDLK_SPACE )
                       QMessageBox::information(0,"asdfads","Asdfadsf");
                     break;
         case SDL_VIDEORESIZE :
          SDL_SetVideoMode(event.resize.w,event.resize.h,0,SDL_OPENGL |SDL_FULLSCREEN);
          myGLReshapeFunc(event.resize.w,event.resize.h);
          break;
    }
    else DrawGLScene();
  }
  SDL_Quit();
  return 1;
}
```

main.cpp, InitGL

```
QImage tt;    tt.load("Texture.png");
QImage tx = QGLWidget::convertToGLFormat ( tt);
printf("QImage loaded %i %i \n",tt.size().width(),tt.size().height());

GLuint texName;
glGenTextures(1, &texName);
glBindTexture(GL_TEXTURE_2D, texName);
glTexImage2D( GL_TEXTURE_2D, 0, 3, tx.width(), tx.height(), 0, GL_RGBA,
GL_UNSIGNED_BYTE, tx.bits() );
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, tx.width(), tx.height(), GL_RGBA,
GL_UNSIGNED_BYTE, tx.bits() );
glEnable(GL_TEXTURE_2D);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );
glTexParameteri
(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
```

XML

- Cos'è XML?
- acronimo di eXtensible Markup Language
- è un linguaggio estensibile realizzato per poter definire ed utilizzare in modo semplice documenti strutturati
- Nato e studiato per il Web,
- Molto utile al difuori del contesto web per definire generici formati di interscambio di dati.
- XML è molto di più di quello che vedrete qui…

Esempio xml

```
<?xml version="1.0" encoding="utf-8"?>
<business-card>
  <persona>
    <titolo> Sig. </titolo>
    <nome> Mario </nome>
    <cognome> Rossi </cognome>
  </persona>
  <indirizzo>
    <strada> Via Verdi </strada>
    <numero-civico> 12 </numero-civico>
    <cap> 56100 </cap>
    <citta> Pisa </citta>
  </indirizzo>
</business-card>
```

Xml

- Basato sull'uso di markup per delimitare porzioni del documento
- Un mark-up è quello compreso tra <>
- Detto anche tag, o etichetta
- I tag devono essere annidati
- XML non ha tag predefiniti
  - è estensibile
  - consente di definire nuovi linguaggi
  - è un metalinguaggio
- Uno degli obiettivi di progettazione di XML:
  - deve essere in un formato leggibile dall'uomo
  - è ascii.

Xml

I documenti xml sono strutturati in tre parti, (le prime due sono opzionali)

2. *XML processing instruction:* identifica la versione di XML usata e come è codificata e se riferisce altri file:

```
<?xml version="1.0" encoding="UCS2" standalone="yes“?>
```

3. *document type declaration (DTD):* che o contiene al suo interno una dichiarazione formale di quali tag (e come) il doc usa (*internal subset*), o riferisce un file che contiene tali dichiarazioni: (*external subset*),

```
<!DOCTYPE memo SYSTEM "http://www.myco.com//memo.dtd">
```

4. Il doc xml vero e proprio il cui elemento radice corrisponde con il tipo specificato nel dtd. Tutti gli altri tag sono nestati in questo.

Xml dettagli

Sintassi

- È case sensitive

Elemento: ciò che è racchiuso da una coppia <tag> … </tag>

- Un tag può avere *attributi*
  <tag att0="str1" att0="str1" > </tag>
- Valida anche la sintassi
- <tag attr="asdfasd" />

Commenti

- <!-- questo e' un commento. //-->

XML

- Perchè usiamo xml?
- Perchè aiuta molto nella definizione del formato di file della nostra applicazione
  - No problemi di parsing.
  - Estendibilità innata (è facile fare backward compatibility)
    - Facile gestire set non ordinati
    - Facile gestire assenza di elementi
  - Fa sempre comodo conoscere una tecnologia in più

XML e linguaggi oo

- Ad ogni classe corrisponde un elemento
- I membri di una classe sono naturalmente annidati.
- Corrispondenza naturale tra strutture dati e formato xml

Esempio

```
<business-card>
  <persona>
    <titolo> Sig. </titolo>
    <nome> Mario </nome>
    <cognome> Rossi
    </cognome>
  </persona>
  <indirizzo>
    <strada> Via Verdi
    </strada>
    <numero-civico> 12
    </numero-civico>
    <cap> 56100 </cap>
    <citta> Pisa </citta>
  </indirizzo>
</business-card>
```

```
class BusinnesCard
{ public:
  Persona m_pers;
  Indirizzo m_ind;
};
class Persona
{public:
  string m_titolo;
  string m_nome;
  string m_cognome;
};
class Indirizzo
{public:
  string m_strada;
  string m_numero
  int m_cap;
string m_citta;
};
```

Xml in pratica

- Generare xml è assai facile
- La via del praticone:
  - printf("<tag> %i </tag>",val);
  - Facile ma possibilità di fare errori di sintassi xml…
- Appoggiandosi a QT

Modello DOM

- I parser xml appartengono a due grosse categorie
- DOM: Document Object Model
  - Leggono tutto un file xml e restituiscono un oggetto c++ (o in qualunque altro linguaggio) che memorizza tutto l'albero con tutti gli elementi e gli attributi.
  - L'applicazione poi naviga l'albero con calma.
- SAX: Simple Api for Xml
  - Generano eventi durante la lettura che devono essere gestiti dall'applicazione

Scene Graph

- Il problema principale è che il mio documento/scena non è affatto strutturato.
- Scriviamoci il nostro semplicissimo scene graph ad hoc
- Cosa ci deve stare dentro?
  - Mesh
  - trasformazioni
  - Animazioni

CSG

- La scena la assumo strutturata come un albero
- Assumo che ogni nodo sia riferito solo una volta.
- Due classi principali
  - CSG: contiene l'intero scene graph (la radice)
  - CSGNode: generico nodo dello scene graph, lo faccio come classe astratta;

CSGNode

```
class CSG
{public:
   CSG(void);
   ~CSG(void);
   CSGGroup root;
};
class CSGNode
{public:
   virtual ~CSGNode(void){};
   virtual void glDraw(const float DocTime)=0;
};
```

- Notare che
  - La draw prende in ingresso un tempo che verrà usato o meno
  - La draw è una funzione pure virtual, quindi la classe CSGNode non è istanziabile
  - Tutto quello che mi serve lo derivo da CSGNode, specializzando distruttore e glDraw.

CSGGroup

- Nodo principale per rappresentare un gruppo di nodi
- È quello che definisce lo scope delle trasformazioni di modellazione
- Si assume che i figli possano cambiare la modelview modificando lo stato per i successivi fratelli
- L'ordine dei figli è significativo
- È responsabile della disallocazione dei nodi.

CSGGroup

```
class CSGGroup :public CSGNode
{  public:
  virtual ~CSGGroup();
  typedef list<CSGNode *>::iterator iterator;
  list<CSGNode *> Sons;
  virtual void glDraw(const float DocTime);
};

CSGGroup::~CSGGroup()  //distruttore: disalloca tutti i figli
{
  for(iterator i=Sons.begin();i!=Sons.end();++i)
        delete (*i);
}

void CSGGroup::glDraw(const float DocTime)
{
  glPushMatrix();
  for(iterator i=Sons.begin();i!=Sons.end();++i)
    (*i)->glDraw(DocTime);
  glPopMatrix();
}
```

CISGTransformation

Classe che incapsula una generica, statica trasformazione (o composizione di varie trasformazioni)

```
class CSGTransformation :public CSGNode
{  public:
  Matrix44f m;
  virtual void glDraw(const float DocTime);
};

void CSGTransformation::glDraw(const float DocTime)
{ glMultMatrix(m); };
```

CISGAnimRotation

- Classe che incapsula una rotazione animata, notare che il tempo da visualizzare arriva da fuori.

```
class CSGAnimRotation :public CSGNode
{ public:
  Point3f axis;
  float AngularSpeedDPS; //Degree Per Sec;
  float StartAngleDeg;
  virtual void glDraw(const float DocTime);
};

void CSGAnimRotation ::glDraw(const float DocTime)
{
   float CurAngleDeg = StartAngleDeg + DocTime*AngularSpeedDPS ;
   glRotatef(CurAngleDeg , axis[0],axis[1],axis[2]);
};
```

CISGAnimZPrecession

- Moto di precessione animato sull'asse Z

```
class CSGAnimZPrecession :public CSGNode
{public:
  CSGAnimZPrecession();
  virtual ~CSGAnimZPrecession(){};
  float DeclinationDeg;
  float AngularSpeedDPS; //Degree Per Sec;
  float StartAngleDeg;
  virtual void glDraw(const float DocTime);
};
CSGAnimZPrecession::CSGAnimZPrecession()
{  StartAngleDeg=0;AngularSpeedDPS=10;DeclinationDeg=30;}
void CSGAnimZPrecession::glDraw(const float DocTime)
{ // precessione: una rotazione il cui asse ruota intorno all'asse z
   float CurAngleRad=ToRad(StartAngleDeg + DocTime*AngularSpeedDPS);
   glRotated( DeclinationDeg,cos(CurAngleRad),sin(CurAngleRad),0);
};
```

Modifichiamo MoebiusStrip

- Deriviamo CIMoebius da CISGNode
  - In questo modo diventa un oggetto che può far parte dello scene graph.
- Adattiamo la funzione draw
- Aggiungiamo il distruttore virtuale (che non fa nulla)

Usiamo lo SceneGraph

- Nel main.cpp aggiungiamo un membro:
- **`CISG Scene;`**
- OnNewDocument: aggiungiamo la costruzione dello scene graph
- **`Scene.root.Sons.push_back(new CSGAnimZPrecession());`**
- **`CIMoebius *ms=new CIMoebius ();`**
- **`ms->Generate();`**
- **`Scene.root.Sons.push_back(ms);\`**

- Notare come si crei apposta un oggetto CIMoebius con la new anziché passargli l'indirizzo di un oggetto esistente. Questo per evitare che poi il distruttore del gruppo faccia pasticci cercando di disallocare qualcosa non allocato con la new

Usiamo lo SceneGraph

- Nella funzione di disegno della classe aggiungiamo:

- **`Scene.root.glDraw(clock()/1000.0f);`**

- Si dovrebbe fare la stessa cosa con tutto il resto (omini, pallina ecc)

Leggere e scrivere file in QT

- un file e' considerato come un particolare device I/O
- classe QIODevice (*astratta*)
  - un mezzo da cui o su cui si puo leggere/scrivere byte
  - La azioni importanti sono
    - open(int mode)/close()/flush()
    - at(Offset pos)
    - readBlock(char * data, Q_ULONG maxlen)/ writeBlock()
    - readLine/getch/putch

QIODevice

- bool QIODevice::open (int mode)
- mode e' una combinazione di
  - IO_Raw *unbuffered file access*
  - IO_ReadOnly
  - IO_WriteOnly
  - IO_ReadWrite
  - IO_Append
  - IO_Truncate
  - IO_Translate *enables carriage returns and linefeed translation*.

Specializzazioni QIODevice

- Qbuffer
  - buffer in memoria di char
- QSocket e QSocketDevice
  - per l'accesso in rete.
- QFile: quello che ci interessa

- Come ci si scrive piu' ad alto livello?

QFile

- Cose utili:
  - QDir per gestire dir,
    - sempre con lo '/'
    - utile per manipolare pathnames
  - QDir::setCurrent(QString) settare la dir corrente cui fanno riferimento i path specificati successivamente (se non assoluti)

QTextStream

- classe per leggere/scrivere testo su un QIODevice
  - QTextStream::QTextStream (QIODevice * iod)
  - QTextStream::QTextStream (QString * str, int filemode)
- solito stile c++
  - << butta nello stream
  - >> legge dallo stream
  - flags e variabili per dire come si trasf i numeri
    - flags(int) con valori tipo *bin, oct, dec, fixedpos ecc.*
    - *precision(int)*

Esempio pratico

```
// il costruttore di QFile non apre il file!
QFile xfile("pippo.txt");

// cosi' come quello di QDir non cambia dir.
QDir xdir("tmptxt");

// fa riferimento al path corrente dell'app...
xdir.mkdir("");

// ...che adesso viene cambiato
QDir::setCurrent(xdir.path());

// il file viene creato solo ora.
xfile.open(IO_WriteOnly);
QTextStream xstrm(&xfile);
xstrm << "prova";
xfile.close();
```

Dom model per XML

- Interfaccia dom per accedere e modificare file xml
- si costruisce una rappresentazione gerarchica (un albero!) del documento xml in memoria
- si lavora sulla rappresentazione in memoria
  - ***leggere*** ~ attraversare e interpretare l'albero in mem con una qualche visita
  - ***scrivere*** ~ creare l'albero xml in memoria visitando nel giusto ordine le nostre strutture dati e poi invocare un metodo save

QDomDocument

- la classe che rappresenta l'intero documento
- e' la radice dell'albero xml permette l'accesso a tutti gli elementi dell'albero
- e' una specializzazione di QDomNode

QDomNode

- Classe base
  - QDomDocument
  - QDomElement
  - QDomAttr
- metodi
  - accesso,inserzione/rimozione figli
    - che son sempre QDomNode
- si puo’ sapere di che tipo e’
  - isDocument/ isElement/isAttribute

QDomAttr

- Attributo di un elemento
  - name()
  - value()

```
<link href="http://www.trolltech.com" color="red" />

QDomElement e = //...
QDomAttr a = e.attributeNode( "href" );
// stampa "http://www.trolltech.com"
cout << a.value() << endl;
// change the node's attribute
a.setValue( "http://doc.trolltech.com" );
QDomAttr a2 = e.attributeNode( "href" );
// stampa "http://doc.trolltech.com"
cout << a2.value() << endl;
```

QDomElement

- rappresenta un elemento
  - tagName() che puo essere cambiato
  - zero o piu’ attributes
    - QString attribute(name, defval)
    - QDomAttr attributeNode(name)

QDomDocument

- gli elementi non possono esistere scollegati da un QDomDocument
- QDomDocument contiene i metodi per ***fabbricare*** gli elementi
  - I nodi hanno un *owner* che il doc che li ha creati
  - tutte le classi QDom sono solo references
    - saranno cancellati quando sono cancellati tutti gli oggetti che le riferiscono e il document che li possiede
    - si possono importare nodi da altri documenti
      - importNode(..)

QDomDocument

- per settare il contenuto di un doc
  - setContent
    - si puo usare in input
      - Qstring
      - QBytearray
      - QIODevice
- Per salvare il contenuto di un doc
  - save(QTextStream) in uno stream
    - vale anche per singoli nodi
  - toString()

QDomDocument

```
QDomDocument doc( "mydocument" );
   QFile file( "mydocument.xml" );
   if ( !file.open( IO_ReadOnly ) )        return;
   if ( !doc.setContent( &file ) ) {
       file.close();
       return;
   }
   file.close();
   // print out the element names of all elements that are direct children
   // of the outermost element.
   QDomElement docElem = doc.documentElement(); // la radice

   QDomNode n = docElem.firstChild();
   while( !n.isNull() ) {
       QDomElement e = n.toElement(); // try to convert the node to an element.
       if( !e.isNull() ) {
           cout << e.tagName() << endl; // the node really is an element.
       }
       n = n.nextSibling();
   }
```

```
QDomDocument doc( "MyML" );
QDomElement root = doc.createElement( "MyML" );
doc.appendChild( root );

QDomElement tag = doc.createElement( "Greeting" );
root.appendChild( tag );

QDomText t = doc.createTextNode( "Hello World" );
tag.appendChild( t );

QString xml = doc.toString();
```

In pratica

- Per salvare uno scene graph in xml?
  - aggiungere un metodo XMLWrite ad ogni nodo

```
void CISGGroup::XMLWrite(FILE *fp)
{
  fprintf(fp,"<CSGGroup>\n");
  iterator i;
  for(i=Sons.begin();i!=Sons.end();++i)
   (*i)->XMLWrite(fp);
  fprintf(fp,"</CSGGroup>\n");
}
```