Costruzione di Interfacce
Lezione 27
Xml for dummies Parte 2

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

Introduzione

- XML read e write
- Scene Graph design
- Adattare lo scene graph

Vcg/Xml

- Due classi:
  - XmlDoc
    - Rappresenta tutto un documento xml dopo che è stato parsato da un file
    - XmlDoc xd; xd.read("myfile.xml");
    - La radice dell'albero xml è in Xd.start;
  - Xml nodo dell'albero;
  - Ogni nodo contiene:
    - string id; // il tag
    - string content;
    - map<string,string> attr;
    - vector<Xml> children;

Memory leaks

- Nota: la classe xml quando parsa un file fa dei memory leaks…
- Colpa del parser che usa una serie di variabili globali (buffers) che non vengono deallocate alla fine.
- Non pericoloso.

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;

Usiamo lo SceneGraph

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

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

Usiamo lo SceneGraph

- Nella OnDraw della classe GLView buttiamo via tutto quello dopo la trasf della trackball e aggiungiamo:

```
pd->Scene.root.glDraw(pd->ElapsedSec);
```

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

Vcg/xml

- Dopo che si è parsato un file xml, occorre fare a mano la creazione delle classi corrispondenti ai vari nodi
- Molto facile se si fa tutto con la stessa interfaccia.
- Per ogni classe che si vuole rendere persistente si aggiunge due funzioni
  - XMLWrite(FILE *fp)
  - XMLRead(Xml& xml);
- E si usa tutto ricorsivamente

Salviamo in XML

- Seguiamo la via del praticone:
- Aggiungiamo al nodo base

```
virtual void XMLWrite(FILE *fp)=0;
```

- Che ci obbliga ad implementare tale metodo in TUTTI gli altri nodi da esso derivati

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


XMLWrite

```
void CSGAnimZPrecession::XMLWrite(FILE *fp)
{
  fprintf(fp,"<CSGAnimZPrecession\n");
  fprintf(fp,"  DeclinationDeg = \"%f\"\n", DeclinationDeg);
  fprintf(fp,"  AngularSpeedDPS = \"%f\"\n", AngularSpeedDPS );
  fprintf(fp,"  StartAngleDeg = \"%f\"\n", StartAngleDeg);
  fprintf(fp,"/>\n");
}
void MoebiusStrip::XMLWrite(FILE *fp)
{
   fprintf(fp,"<MoebiusStrip\n");
   fprintf(fp,"  BlockNum = \"%i\"\n",BlockNum);
   fprintf(fp,"/>\n");
}
```

- Il parsing alla prossima puntata!

XMLRead

- Il meccanismo di base è che un si passa ad un oggetto un elemento xml da cui l'oggetto pesca tutto quello che gli serve (simmetrico della write)
- Difficoltà
  - gestione puntatori
  - funzioni virtuali…
- Qui abbiamo entrambe…
- In più dobbiamo riuscire a mappare tutto quello che avevamo nella nostra app come nodi dello scene graph.

Virtuali

- Nel nodo del gruppo c'è una lista di puntatori a generici
- list<CSGNode *> Sons;
- I meccanismi di inheritance e polimorfismo del c++ ci aiutano a far si che ogni nodo sappia come comportarsi:
  - Come disegnarsi
  - Come salvarsi

Read

- La lettura è più problematica perchè in realtà facciamo due cose:
  - Creazione della struttura dello scene graph
  - Settaggio dei parametri dei vari nodi
- Il meccanismo

  Virtual void myClass::XMLRead(Xmlelem& xml)
- Funziona bene se ho già la struttura dell'albero
  - Ogni nodo non standard sa come leggersi perchè lo scrivo io

Read

- Schema base:
- Lettura fatta tramite funzione virtuale pura
- Ogni classe ha la propria funzione XMLRead
  - Controllo che il tag dell'elemento che mi è stato passato sia compatibile con me.
  - Per ogni membro della classe cerco l'attributo corrispondente e se non c'è metto un valore di default

Creazione della struttura

- Nel progettare lo scene graph voglio che sia estendibile.
  - Deve essere possibile definire nuovi classi di nodi (e.g. md2 models) senza dover cambiare nulla nelle strutture di base
- Nel ricostruire l'albero dello scene graph il problema è che la creazione dei nodi viene fatta dal nodo gruppo.
- Il nodo gruppo non può conoscere tutti i possibili nodi futuri.

Incapsulare

- Una soluzione ragionevole consiste nell'incapsulare la conoscenza di tutti i nodi possibili invece che nel gruppo, nello scene graph stesso
  - Assunzione ragionevole lo scene graph *conosce* tutti I possibili tipi di nodi
  - Chi definisce nuovi tipi di nodi deve subclassare lo scene graph
- Perchè non subclassare il gruppo?
  - Perchè non è una classe finale.
  - Posso immaginare classi derivate da group (e.g. switch).

Creazione di nodi

- Per rendere flessibile la creazione aggiungo allo scene graph una funzione che mi crea dinamicamente un nuovo nodo di un tipo specificato a runtime
- `CSGNode *CSG::Allocate(const string &classname)`
  - Se voglio nuovi nodi basta subclassare CSG e ridefinire Allocate per gestire nuovi nomi di classi
  - Secondo vantaggio: la scene può sapere tutto della vita dei suoi figli

Allocate base

```
CSGNode *CSG::Allocate(const string &classname)
{
  CSGNode *pt=0;
  if(classname=="CSGGroup") pt= new CSGGroup;
  if(classname=="CSGTransformation") pt = new CSGTransformation;
  if(classname=="CSGAnimRotation") pt = new CSGAnimRotation;
  if(classname=="CSGAnimZPrecession") pt = new CSGAnimZPrecession;
  return pt;
}
```

Specializzazione Scene Graph

- A questo punto iniziamo a specializzare il nostro scene graph.
  - Deve gestire nuovi tipi di nodi
  - È vincolato ad avere un solo nodo di tipo Moebius che voglio riferire facilmente.
  - Mi tengo un membro puntatore a Moebius strip

```
class CSGMb : public CSG
{
public:
  CSGMb() {m=0;}
  virtual CSGNode *Allocate(const string &classname);
  MoebiusStrip *m;
};
```


Specializzazione Scene Graph

```
CSGNode *CSGMb::Allocate(const string &classname)
{
    if(classname=="MoebiusStrip") {
          assert(m==0);
          m=new MoebiusStrip;
          return m;
    }
    CSGNode *nn=CSG::Allocate(classname);
  if(nn) return nn;
    return 0;
}
```

- Il membro puntatore a moebius è quello che uso nel doc

Picking

- Cosi'come abbiamo ristrutturato ora non funziona più. Perché?
- Il Picking era membro di moebius e per funzionare assumeva che venisse fatto partire con la matrice di modellazione come durante il rendering
- Adesso abbiamo una popmatrix alla fine che toglie il moto di precessione
- Ogni gruppo è isolato.

Picking

- Il picking è una feature dello scene graph.
- Ogni nodo che disegna qualcosa è responsabile di mettere il suo nome all'inizio
- lasciare sullo stack dei nomi alla fine qualcosa di sensato (-1).
- Problema: possibili collisioni di nomi.
  - Si dovrebbe usare anche lo stack dei nomi
  - Insieme dei nomi locale al gruppo.

Picking

```
int CSG::Pick(int x, int y, const float &DocTime)
```

- Notare che:
  - Ha bisogno del tempo
  - Funziona purché sia fatta partire nello stesso stato di quando si fa partire glDraw della scena intera

Aggiungiamo la pallina

- Facciamo una classe derivata da CIMesh e da CSGNode

```
class CSGSphereMesh : public CSGNode, CIMesh
{
public:
  float Radius;
  int Parallel;
  int Meridian;
  CSGSphereMesh(){};
  virtual   ~CSGSphereMesh(void){};

  virtual void glDraw(const float DocTime);
  virtual void XMLWrite(FILE *fp);
  virtual void XMLRead(Xml &xml, CSG *Base);
};
```

CSGSphereMesh

```
CSGSphereMesh(float r,int p,int m)
{
 Radius=r;
 Parallel=p;
 Meridian=m;
 CIShape::Sphere(*this,r,p,m);
};
```



```
void CSGSphereMesh::glDraw(const float DocTime){
   Draw<true,true>();
}
void CSGSphereMesh:: XMLWrite(FILE *fp)
{
   fprintf(fp,"<CSGSphereMesh\n");
   fprintf(fp,"  Radius = \"%f\" Parallel = \"%i\" Meridian= \"%i\"\n",
       Radius,Parallel,Meridian);
   fprintf(fp,"/>\n");
}
void CSGSphereMesh::XMLRead(Xml &xml, CSG *Base)
{
   assert(xml.id == "CSGSphereMesh");
   Radius   = (float)atof(xml["Radius"].c_str());
   Parallel =  atoi(xml["Parallel"].c_str());
   Meridian = atoi(xml["Meridian"].c_str());
   CIShape::Sphere(*this,Radius,Parallel,Meridian);
}
```


Allocate

- Avendo aggiunto al un nuovo nodo allo scene graph devo ricordarmi di gestirlo nella allocate

```
CSGNode *CSGMb::Allocate(const string &classname)
{
   if(classname=="MoebiusStrip") {
        assert(m==0);
        m=new MoebiusStrip;
        return m;
   }
   if(classname=="CSGSphereMesh") {
        return new CSGSphereMesh;
   }
   CSGNode *nn=CSG::Allocate(classname);
   if(nn) return nn;
   return 0;
}
```


Movimento sul nastro

- Aggiungiamo un nuovo nodo che gestisca il movimento di un oggetto
- Avevamo una classe MoebiusStrip::pos che faceva gran parte del lavoro sporco
- Tipico uso:
  - `MoebiusStrip::Pos pp;`
  - `pp.CurSide=0;`
  - `pp.ThetaDeg=45+CurAngleDeg/2;`
  - `pd->m.Transform(pp);`

MoebiusStrip::Transform

```
void MoebiusStrip::Transform(const MoebiusStrip::Pos &p)
{
   // di quanto si gira per ogni turn (e.g.
   float TurnStepDeg = (360.0f/SideNum);
   // di quanto si avvolge per ogni degree
   float TwistPerDeg = (TurnStepDeg * Turns )/360.0f;
   float HalfFilletAngleDeg=(TurnStepDeg*FilletRatio)/2;
   float BaseAngleDeg = StartTwistDeg-TurnStepDeg/2.0f;

   // Rotazione Intorno all'asse principale
   glRotatef(p.ThetaDeg, 0,0,1);
   glTranslatef(Radius,0,0);

   // Avvolgimento della striscia.
   glRotatef(TurnStepDeg*p.CurSide+p.ThetaDeg*TwistPerDeg, 0,1,0);

   // Spostamento verso la superficie del'nastro
   glTranslatef(Cos(ToRad(BaseAngleDeg+HalfFilletAngleDeg))*InnerRadius,0,0);
}
```


Incapsuliamo

- Al solito

```
class CSGAnimMoebiusTransf  : public CSGNode
{
public:
   CSGMb *b;
   virtual     ~CSGAnimMoebiusTransf(void){};
   float AngularSpeedDPS; //Degree Per Sec;
     float StartAngleDeg;
   int StartSide;

   virtual void glDraw(const float DocTime);
   virtual void XMLWrite(FILE *fp);
   virtual void XMLRead(Xml &xml, CSG *Base);
};
```

- Notate come sia usato il puntatore alla scene dentro a questo nodo per recuperare la moebius strip;
- Non tutti I nodi hanno bisogno di sapere qual'è lo scene graph cui appartengono
  - Un nodo può appartenere a più scene graphs?

```
void CSGAnimMoebiusTransf::glDraw(const float DocTime)
{
   MoebiusStrip::Pos pp;
   float CurAngleDeg = StartAngleDeg + DocTime*AngularSpeedDPS;
   pp.CurSide=StartSide;
   pp.ThetaDeg=CurAngleDeg;
   b->m->Transform(pp);
}
```



```
void CSGAnimMoebiusTransf:: XMLWrite(FILE *fp)
{
   fprintf(fp,"<CSGAnimMoebiusTransf\n");
   fprintf(fp,"  AngularSpeedDPS = \"%f\" "
              "  StartAngleDeg = \"%f \" "
              "  StartSide = \"%f\"\n",
         AngularSpeedDPS, StartAngleDeg, StartSide);
   fprintf(fp,"/>\n");
}

void CSGAnimMoebiusTransf::XMLRead(Xml &xml, CSG *Base)
{
   assert(xml.id == "CSGAnimMoebiusTransf");
   AngularSpeedDPS = (float)atof(xml["AngularSpeedDPS"].c_str());
   StartAngleDeg   = (float)atof(xml["StartAngleDeg"].c_str());
   StartSide       =  atoi(xml["StartSide"].c_str());
   b=(CSGMb *)Base;
}
```


OnNewDocument

```
BOOL CMBDoc::OnNewDocument()
{
…
   Scene.root.Sons.push_back(new CSGAnimZPrecession());
   Scene.m=new MoebiusStrip();
   m=Scene.m;
   m->Generate();
   Scene.root.Sons.push_back(m);
   CSGAnimMoebiusTransf *ma=new CSGAnimMoebiusTransf;
   ma->b=&Scene;
   ma->StartAngleDeg=0;
   ma->AngularSpeedDPS=40;
   Scene.root.Sons.push_back(ma);
   Scene.root.Sons.push_back(new CSGSphereMesh(.5f,20,40));
`
   return TRUE;
}
```


Esempio di xml

```
<CSGGroup >
<CSGAnimZPrecession
  DeclinationDeg = "30.0" AngularSpeedDPS = "10.0" StartAngleDeg= "0.0"
/>
<MoebiusStrip
  SideNum       = "2"  Turns          = "1.0"
  Radius        = "4.0"  BlockNum       = "12"
  InnerRadius   = "0.800000"  FilletRatio   = "0.100000"
  StartTwistDeg = "0.0"  StepNum        = "4"
/>
<CSGGroup >
<CSGAnimMoebiusTransf
  AngularSpeedDPS = "40.0" StartAngleDeg = "0" StartSide="1"/>
<CSGAnimRotation
  AxisX = "0.0" AxisY = "0.0" AxisZ = "1.0" AngularSpeedDPS = "720.0" StartAngleDeg= "20.0"
/>
<CSGSphereMesh
  Radius = "0.500000" Parallel = "20" Meridian= "40"
/>
</CSGGroup>
<CSGGroup >
<CSGAnimMoebiusTransf
  AngularSpeedDPS = "40.0" StartAngleDeg = "80" StartSide="0"
/>
<CSGSphereMesh
  Radius = "0.500000" Parallel = "20" Meridian= "40"
/>
</CSGGroup>
</CSGGroup>
```