

Costruzione di Interfacce Lezione 26 XML read e write / GI Selection e picking

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

XML writing

- ❖ Ogni classe
 - ❖ scrive un elemento
 - ❖ e' responsabile del contenuto del nodo xml
 - ❖ ricorsiva
 - ❖ si usa semplicemente gli stream

XML Write

```
void CISG::XMLWrite(QTextStream &xstrm)
{
    xstrm << "<CISG>\n";
    root.XMLWrite(xstrm);
    xstrm << "</CISG>\n";
}

void CISGGroup::XMLWrite(QTextStream &xstrm)
{
    xstrm << "<CISGGroup>\n";
    iterator i;
    for(i=Sons.begin(); i!=Sons.end(); ++i)
        (*i)->XMLWrite(xstrm);
    xstrm << "</CISGGroup>\n";
}
```

```
class CISGRotation :public CISGNode
{public:
    CISGRotation() :Axis(Point3f(0,0,1)),AngleDeg(0){};
    CISGRotation(float ad, Point3f &v): Axis(v),AngleDeg(ad) {};

    Point3f Axis;
    float AngleDeg;
    virtual void glDraw(const float )
        {glRotatef(AngleDeg,Axis[0],Axis[1],Axis[2]);}
    virtual void XMLWrite(QTextStream &xstrm);
    virtual bool XMLRead(QDomElement n, CISG *Base);
};

void CISGRotation::XMLWrite(QTextStream &xstrm)
{
    xstrm << "<CISGRotation>\n";
    xstrm << Axis << " " << AngleDeg ;
    xstrm << "</CISGRotation>\n";
}
```

Overload << e >>

```
QTextStream& operator<< ( QTextStream& os, Point3f& m )
{
    os<< m[0] << " " << m[1] << " " << m[2] << " ";
    return os;
}
QTextIStream& operator>> ( QTextIStream& os, Point3f& m )
{
    os >> m[0] >> m[1] >> m[2];
    return os;
}
QTextStream& operator<< ( QTextStream& os, Matrix44f& m )
{
    for(int i=0; i<4;++i)
        for(int j=0; j<4;++j)
            os<< m[i][j];
    return os;
}
```

XML read

- ❖ l'xml salvato e' formato da un nodo di tipo CISG che contiene come unico sottoalbero un CISGGroup.
- ❖ Problema principale:
 - ❖ creazione dinamica di oggetti in base a stringa
 - ❖ se trovo un nodo xml con tag CISGRotation devo istanziare un oggetto di tipo CISGRotation ecc.
 - ❖ Delegare l'allocazione al SG

Lettura SG

- ❖ Lo scene graph *deve* conoscere tutti i tipi di nodi
 - ❖ cio' non era necessario per la scrittura
- ❖ Allocate(string) alloca in base al tag.
- ❖ Come si fa ad aggiungere nuovi nodi allo sg
 - ❖ e.g. moebius e md2?

lettura Scene Graph

```
bool CISG::XMLRead(QIODevice &io)
{
    QDomDocument doc;
    if ( !doc.setContent( &io ) ) return false;

    QDomElement docElem = doc.documentElement();
    if(docElem.tagName() != "CISG" ) return false;

    QDomNode xroot = docElem.firstChild();
    if(xroot.isNull()) return false;
    QDomElement xre = xroot.toElement(); // try to convert the
    node to an element.
    if( xre.tagName() != "CISGGroup" ) return false;
    return root.XMLRead(xre, this);
}
```

lettura gruppo

```
bool CISGGroup::XMLRead(QDomElement en, CISG *Base)
{
    Sons.clear();
    if(en.tagName() != "CISGGroup") return false;
    QDomNode n=en.firstChild();
    while( !n.isNull() ) {
        QDomElement e = n.toElement(); // try to convert the node to
        an element.
        if( !e.isNull() ) {
            CISGNode *np=Base->Allocate(e.tagName());
            if(!np) return false;
            cout << e.tagName().ascii() << endl; // the
            node really is an element.
            np->XMLRead(e, Base);
            Sons.push_back(np);
        }
        n = n.nextSibling();
    }
    return true;
}
```

lettura nodo generico

```
/****** CISGRotation *****/
void CISGRotation::XMLWrite(QTextStream &xstrm)
{
    xstrm << "<CISGRotation>\n";
    xstrm << Axis << " " << AngleDeg ;
    xstrm << "</CISGRotation>\n";
}

bool CISGRotation::XMLRead(QDomElement n, CISG *)
{
    if(n.tagName() != "CISGRotation" || !n.firstChild().isText()
    )
        return false;
    cout << n.tagName().ascii() << ": " << n.text().ascii() <<
    endl;
    QTextIStream(&n.text()) >> Axis >> AngleDeg;
    return true;
}
```

allocate standard

```
CISGNode *CISG::Allocate(const QString &classname)
{
    CISGNode *pt=0;
    if(classname=="CISGGroup") pt= new CISGGroup;
    if(classname=="CISGGround") pt= new CISGGround;
    if(classname=="CISGRotation") pt= new CISGRotation;
    if(classname=="CISGTranslation") pt= new CISGTranslation;
    if(classname=="CISGScaling") pt= new CISGScaling;
    if(classname=="CISGTransformation") pt= new
    CISGTransformation;
    if(classname=="CISGAnimRotation") pt= new CISGAnimRotation;
    if(classname=="CISGAnimZPrecession") pt= new
    CISGAnimZPrecession;
    return pt;
}
```

allocate derivata

Per poter gestire nodi personalizzati basta subclassare la classe principale dello scenegraph, ridefinire la allocate in modo da coprire le allocazioni di nuovi nodi

```
class CISGM : public CISG
{
public :
    virtual CISGNode *Allocate(const QString &classname)
    {
        CISGNode *ret=CISG::Allocate(classname);
        if(classname=="CIMoebius") ret=new CIMoebius();
        return ret;
    }
};
```

Selezione e picking

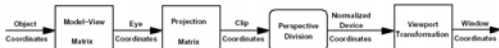
- ❖ Per capire quello che si trova sotto il mouse nel mondo 3d
- ❖ Due strategie
 - ❖ `glselection`, ad alto livello, permette di sapere quale *oggetto* ho cliccato con il mouse
 - ❖ `gluUnproject` dato un punto sullo schermo e la *z* ad esso associata sullo *zbuffer*, permette di trovare le coord 3d in nel sistema di riferimento originale

glSelection

- ❖ Meccanismo fornito da `opengl` per capire cosa fa a finire in una data porzione dello schermo.
- ❖ Quindi per capire, con precisione, cosa si trova sotto il mouse
- ❖ Sfrutta la pipeline di rendering
- ❖ Non fa rasterizzazione

glSelection

- ❖ In pratica si definisce una nuova matrice di proiezione che inquadra solo la regione scelta
- ❖ Si rimanda tutta la geometria della scena, dando ad ogni entità un nome
- ❖ La geometria attraversa solo la prima parte della pipeline (no rasterization)
- ❖ I nomi di quello non viene clippato tutto fuori dal volume di vista vengono salvati in un buffer.



glSelection in pratica

- ❖ Per scoprire cosa passa nel punto *x,y* occorre:
- ❖ Passare alla modalità di rendering selection
 - ❖ `glRenderMode(GL_SELECT);`
 - ❖
- ❖ Inizializzare lo stack dei nomi
 - ❖ `glInitNames();`
 - ❖ `// LoadName() won't work with no names on the stack`
 - ❖ `glLoadName(-1);`
- ❖ Preparare un buffer dove `opengl` metterà i nomi delle entità che cadono nella zona d'interesse
 - ❖ `static unsigned int selectBuf[16384];`
 - ❖ `glSelectBuffer(16384, selectBuf);`

glSelection

- ❖ Modificare la matrice di proiezione in modo che mi clippi quello che mi interessa
- ❖ Occorre PRE-moltiplicarla per una matrice fatta apposta e relativa al punto che vogliamo e al viewport corrente:

```
int viewport[4];
glGetIntegerv(GL_VIEWPORT, viewport);
glMatrixMode(GL_PROJECTION);
double mp[16];
glGetDoublev(GL_PROJECTION_MATRIX, mp);
glPushMatrix();
glLoadIdentity();
gluPickMatrix(x, y, 4, 4, viewport);
glMultMatrixd(mp);
```

glSelection

- ❖ Rendering con i nomi per oggetti che voglio scegliere
 - ❖ `glLoadName(int)`
- ❖ Nota è uno stack
 - ❖ `glPushName()`
 - ❖ `glPopName()`

glSelect

- ❖ Dopo il rendering si rimette tutto a posto

```
glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
int hits = glRenderMode(GL_RENDER);
```

- ❖ E nel buffer che avevamo preparato opengl ha messo una serie di record con la seguente struttura:

- ❖ int: Numero nomi nello stack
- ❖ int,int: Minima e max depth
- ❖ int ... int Lista nomi sullo stack

Parsing del selection buffer

- ❖ Se non abbiamo usato lo stack (se non abbiamo fatto glPushName) i record sono tutti lunghi uguali e parsarli è facile:

```
vector< pair<double,unsigned int> > H;
for(int ii=0;ii<hits;ii++){
    TRACE("%ui %ui %ui\n",
          selectBuf[ii*4],selectBuf[ii*4+1],
          selectBuf[ii*4+2],selectBuf[ii*4+3]);
    H.push_back( make_pair(
                  selectBuf[ii*4+1]/4294967295.0,
                  selectBuf[ii*4+3])
                );
}
sort(H.begin(),H.end());
TRACE("\n Closest is %i\n",H[0].second);
```

In pratica

- ❖ Non fare la pick direttamente nell'handler del mouse
- ❖ Nel 90% dei casi va, ma potrebbe non essere disponibile il contesto opengl
- ❖ Salvarsi posizione del mouse;

tipico code snip

```
long hits;
static unsigned int selectBuf[16384];
glSelectBuffer(16384, sBuf);
glRenderMode(GL_SELECT);
glInitNames(); glPushName(-1);
int viewport[4]; glGetInteger(GL_VIEWPORT, viewport);
double mp[16]; glGetDoublev(GL_PROJECTION_MATRIX, mp);
glMatrixMode(GL_PROJECTION);
glPopMatrix();
glLoadIdentity();
gluProject(x, y, z, viewport[3]-y, 4, 4, viewport);
glMultiTexCoord1s();
glMatrixMode(GL_MODELVIEW);
Draw(); // RENDER all the scene as always loading names before objects
glPopMatrix();
glMatrixMode(GL_PROJECTION);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
hits = glRenderMode(GL_RENDER);
if (hits <= 0) return 0;
vector< pair<double,unsigned int> > H;
for(int ii=0;ii<hits;ii++)
    H.push_back( make_pair(sBuf[ii*4+1]/4294967295.0,sBuf[ii*4+3]));
sort(H.begin(),H.end());
return H[0].second;
```

Alternativa gluUnproject

```
int gluUnProject(
GLdouble winx, GLdouble winy, GLdouble winz,
const GLdouble modelMatrix[16],
const GLdouble projMatrix[16],
const GLint viewport[4],
GLdouble *objx, GLdouble *objy, GLdouble *objz );
```

- ❖ dove

- ❖ winx e winz sono le coord del mouse
- ❖ winz e' la z del pixel presa con
glReadPixel(winx, winy, 1, 1,
GL_DEPTH_COMPONENT, GL_FLOAT, &winz);

- ❖ Restituisce la posizione del punto indicato dal mouse nel sistema di riferimento definito al momento in cui sono state prese le matrici model e proj.

- ❖ **ATTENZIONE A QUANDO PRENDETE LE MATRICI!!!!**