Costruzione di Interfacce
Lezione 21
Trackball, Generazione TexCoords

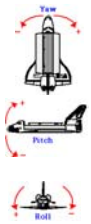
cignoni@iei.pi.cnr.it
http://vcg.iei.pi.cnr.it/~cignoni

Interfacce di rotazione

- Come può un utente specificare una rotazione tramite un interfaccia?
- Due modalità:
  - Diretta: specifica valori numerici esatti
  - Interattiva: tramite movimenti del mouse
- Come rappresento una rotazione?
  - Euler Angle
  - Axis/angle
  - Quaternions

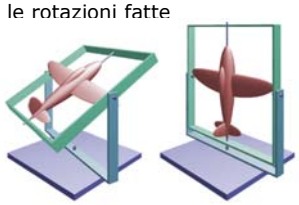
Euler Angle

- Una rotazione viene espressa come una serie di tre rotazioni sui tre assi.
- Deriva dal modo con cui si descrive l'orientamento di un aereo
  - Yaw
  - Pitch
  - Roll
- Intuitivo per piccoli valori di pitch e roll



Euler Angle

- Problema ordine rotazione
  - Il risultato dipende dall'ordine in cui faccio le tre rotazioni
- Problema Gimbal Lock
  - In alcune situazioni le rotazioni fatte su un asse possono su un altro asse
  - Se il pitch è a 90 gradi yaw e roll si possono annullare a vicenda.

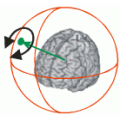


Gimbal lock nelle interfacce

- Capita ad esempio quando cerco di far specificare gli euler angle interattivamente all'utente:
  - Up/down: rot asse x
  - Left/right: rot asse y
  - Pgup/pgdn: rot asse z
- Si incarta.

Axis/angle

- Approccio OpenGL
  - Si specifica un'asse di rotazione e un angolo di rotazione
  - Molto generico
  - Poco intuitivo
    - Qual'e l'asse di rotazione per girare la testa in modo da guardare in basso a destra?



Quaternioni

- Cos'è un quaternione?
- Un estensione dei numeri complessi,

$$q = w + xi + yj + zk \quad dove \quad i \cdot i = j \cdot j = k \cdot k = -1$$

- Spesso rappresentato come una coppia scalare-vettore:

$$q = [w, \mathbf{v}] \quad dove \quad \mathbf{v} = (x, y, z)$$

Quaternioni

- Magnitudine

$$\|q\| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

- Normalizzazione a quaternione unitario

$$q = q / \|q\|$$

Somma e prodotto

- Dati due quaternioni

$$q_1 = w_1 + x_1 i + y_1 j + z_1 k \quad e \quad q_2 = w_2 + x_2 i + y_2 j + z_2 k$$

$$q_1 = [w_1, \mathbf{v}_1] \; e \; q_2 = [w_2, \mathbf{v}_2] \quad dove \quad \mathbf{v}_1 = (x_1, y_1, z_1) \; e \; \mathbf{v}_2 = (x_2, y_2, z_2)$$

- Si definisce

$$q_1 + q_2 = [w_1 + w_2, \quad \mathbf{v}_1 + \mathbf{v}_2]$$

$$q_1 * q_2 = [w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, \quad \mathbf{v}_1 w_2 + w_1 \mathbf{v}_2 + \mathbf{v}_1 \times \mathbf{v}_2]$$

- Identità
  - somma $q_{1+} = [0, \quad (0,0,0)]$
  - prodotto $q_{1*} = [1, \quad (0,0,0)]$

Quaternioni e rotazioni

- Ogni quaternione unitario corrisponde ad una rotazione in 3d
- La moltiplicazione tra due quaternioni corrisponde alla composizione delle due rotazioni

Conversioni

- Da quaternione a matrice

$$\begin{bmatrix} 1-2x^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2xy-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}$$

- Da quaternione ad axis/angle

$$q = [w, \mathbf{v}] \quad axis = \mathbf{v} / |\mathbf{v}| \; e \; angle = 2 \arccos(w)$$

Conversioni

- Da axis angle a quaternioni

$$axis = (a_x, a_y, a_z) \; e \; angle = \theta$$

$$q = [\mathbf{v}, w]$$

$$\mathbf{v} = (a_x \cdot \sin(\tfrac{\theta}{2}), a_y \cdot \sin(\tfrac{\theta}{2}), a_z \cdot \sin(\tfrac{\theta}{2}))$$

$$w = \cos(\tfrac{\theta}{2})$$

- Da euler angle a quaternion

$$q_x = [\cos(\tfrac{a}{2}), \quad (\sin(\tfrac{a}{2}),0,0)]$$

$$q_y = [\cos(\tfrac{b}{2}), \quad (0,\sin(\tfrac{b}{2}),0)]$$

$$q_z = [\cos(\tfrac{c}{2}), \quad (0,0,\sin(\tfrac{c}{2}))]$$

$$q = q_x * q_y * q_z$$

Evitare il gimbal lock

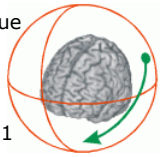
- Euler angle è molto intuitivo per piccole rotazioni: se ruoto di angoli piccoli quello che ottengo è esattamente quello che mi aspetto
- Soluzione:
  - Tenere la rotazione come un quaternione
  - Ad ogni pressione di tasto generare un quaternione corrispondente al piccolo euler angle
  - Ad es. se premo *left* genero un quaternione

$$q_y = [\cos(\tfrac{\delta}{2}),\quad (0,\sin(\tfrac{\delta}{2}),0)]$$

  - Comporre il risultato con moltiplicazione tra quaternioni e tenere il risultato come base;

Trackball

- Come si mappa il movimento del mouse in una rotazione?
  - Si immagina una sfera circoscritta all'oggetto con cui si vuole interagire
  - Ogni drag del mouse definisce due punti p1 e p2(inizio e fine del drag) sulla sfera
  - Si considera la rotazione che descrive l'arco di cerchio sulla superficie sferica delimitato da p1 e p2



Trackball

- La rotazione così calcolata viene trasformata in un quaternione e composta con la trasf corrente
- Se una volta rilasciato il mouse, si continua comporre con l'ultimo quaternione calcolato, si ottiene l'effetto di spinning.

CITrackball

- Classe che implementa una trackball
- Di solito è un oggetto della glview
- Interfaccia fondamentale nella glview
  - Si deve gestire Mousedown/mouseup/Mousemove
  - Avere un membro dove tenere la mat di rotazione corrente
  - Comunicare Resize alla trackball

CITrackball

```
class CMBGLView : public CView
{
…
   // Gestione trackball
  CITrackball m_tb;
  Matrix44f m_matRot;
  bool m_bCaptured;
…
};

void CMBGLView::OnInitialUpdate()
{
…
   CRect rc;
   GetClientRect(&rc);
   m_tb.Init(rc.right,rc.bottom);
   m_matRot.SetIdentity();
}
void CMBGLView::OnSize(UINT nType, int cx, int cy)
{
…
   m_tb.Resize( cx,cy );
…
}
```


CITrackball

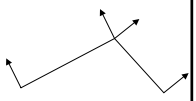
```
void CMBGLView::OnLButtonDown(UINT nFlags, CPoint point)
{
   if(!m_bCaptured) {
        m_tb.MouseDown( point.x, point.y, 0 );
        SetCapture();
        m_bCaptured = TRUE;
   }
   CView::OnLButtonDown(nFlags, point);
}
void CMBGLView::OnLButtonUp(UINT nFlags, CPoint point)
{
   if(m_bCaptured) {
        m_tb.MouseUp(point.x,point.y);
        ReleaseCapture();
        m_bCaptured = FALSE;
   }
   CView::OnLButtonUp(nFlags, point);
}
void CMBGLView::OnMouseMove(UINT nFlags, CPoint point)
{
   if(m_bCaptured) {
        m_tb.CalcRotMatrix( m_matRot, point.x, point.y );
        Invalidate();
   }
   CView::OnMouseMove(nFlags, point);
}
```

CITrackBall

```
void CMBGLView::OnDraw(CDC* pDC)
{
…
   // World To Camera Transformation
   gluLookAt(0,0,10,0,0,0,0,1,0);

   // Apply the trackball
   glMultMatrix(m_matRot);

   glRotatef( -90,0,1,0);
   glRotatef( 90,1,0,0);

   // moto di precessione: una rotazione il cui asse ruota
   intorno all'asse z
   glRotated( 10,cos(ToRad(-45+CurAngleDeg*.5)),sin(ToRad(-
   45+CurAngleDeg*.5)),0);
…
   if(pd->m.Empty()) pd->m.Generate();
   pd->m.Ring.Draw<true,true>();
   SwapBuffers(m_pDC->GetSafeHdc());
}
```


Modifiche a MoebiusStrip

- Resa piu' generale
- Non piu width/ height ma innerradius e filletratio
- La Generate è ora divisa in due parti:
  - Prima si genera una sezione
  - Poi si estrude
- Gestione CreaseAngle tramite duplicazione vertici
  - Introduzione Matrix44
  - Gestione normali corretta
  - Generazione coordinate texture
  - Draw templated



Metaprogramming

- "the art of programming programs that read, transform or write other programs"
- Risultati
  - Compile-time computations
  - Compile-time control structure
  - Performance
- Metaprogramming è una gran disciplina, questo è sono un minimo assaggio.

Passaggio parametri

- Normalmente il passaggio di parametri è fatto attraverso variabili/oggetti
- `double square(x) {return x*x;}`
- Valori specifici vengono passati a run time quando la funzione viene invocata
- `…square(-3)…`
- In c++ si puo' fornire placeholder per risolvere tutto a runtime
- `template<type t> inline square(T x) { return x*x;}`

Template valori

- Ma si può templatare anche in base a valori

```
template <unsigned n>
inline double pow(double x) {
double ans=1.0;
for(unsigned k=0;k<n;++k)
ans*=k;
return ans;
}
```

- Oppure ancora meglio

```
template <unsigned n>
inline double pow(double x) {
return pow<n%2>(x) * pow<n/2> (x*x);
}
template<> pow<1u>(double x) {return x;}
```


Template

- In questo modo si puo far fare al compilatore parte del processing

```
template <bool NormFlag, bool TexFlag> Draw()
{
 glBegin(GL_TRIANGLES);
 vector<CIFace>::const_iterator fi;
 for(fi=face.begin();fi!=face.end();++fi)
      {
      if(NormFlag) glNormal((*fi).v[0]->n);
      if(TexFlag) glTexCoord2f((*fi).v[0]->s,(*fi).v[0]->t);
      glVertex((*fi).v[0]->p);
…
      if(NormFlag) glNormal((*fi).v[2]->n);
      if(TexFlag)  glTexCoord2f((*fi).v[2]->s,(*fi).v[2]->t);
      glVertex((*fi).v[2]->p);
      }
 glEnd();
}
```

Conclusioni

- Stiamo ancora sperimentando
- Mancano gli oggetti che si muovono sull'anello
- Dobbiamo decidere meglio le specifiche dell'app.
- L'animazione ora è gestita interamente dentro la view. Non portabile…
- Butteremo via ancora varie volta quasi tutto…