Grafica Computazionale

Laboratorio: Le trasformazioni

Fabio Ganovelli
fabio.ganovelli@isti.cnr.it
a.a. 2005-2006

Obiettivi

- (oggi) le trasformazioni della pipeline grafica in softogl
- (domani) clipping di segmenti

Le trasformazioni: i passi

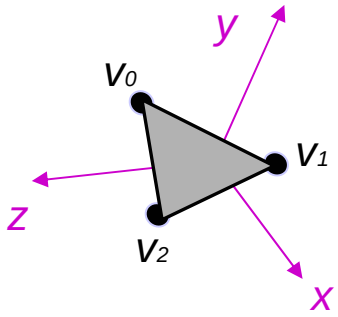
- Nell’attuale versione di softogl trovate i tipi Point3dHC,Vector3dHC e Matrix44 implementati
- 1) implementare le funzioni che costruiscono le matrici di:
  - Traslazione
  - Rotazione intorno agli assi x,y,z
  - Shear
  - Scaling
- 2) implementare la funzione che costruisce la matrice di rotazione attorno a un asse specificato
- 3) implementare la funzione che costruisce la matrice di cambiamento di frame dato un punto di vista, una direzione un vettore “up” (vedi lez. 5)

Le matrici in softogl

- Nella pipeline, e quindi nella classe softogl, teniamo solo tre matrici (per ora)
  - Trasformazione di modello (modelview_matrix): dal frame in cui è definito l'oggetto al frame di vista
  - Trasformazione di proiezione (projection_matrix): dal volume di vista al volume di vista canonico
  - Trasformazione di viewport (viewport_matrix): dalle window coordinates alle viewport coordinates

Dalla lezione 5

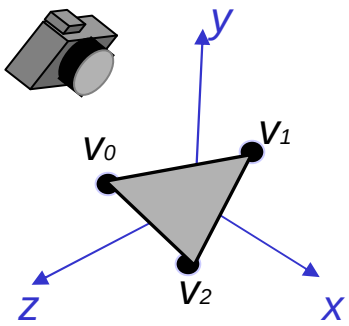
0) transformazione di modellazione
1) transformazione di vista
2) transformazione di proiezione
3) transformazione di viewport



object Coordinates

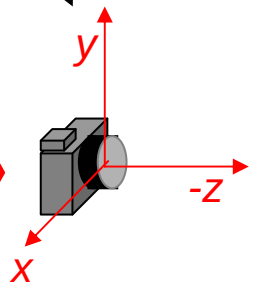
0

modelview_matrix

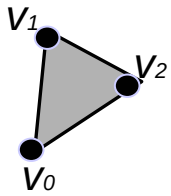


world Coordinates

1

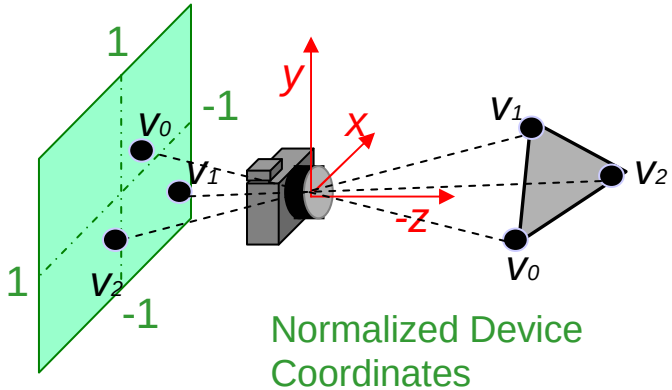


view Coordinates
(a.k.a. eye Coordinates)



2

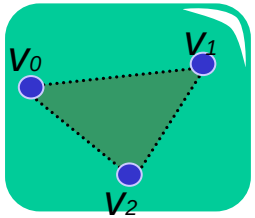
projection_matrix



Normalized Device
Coordinates

viewport_matrix

3



screen Space

4) Modifica delle matrici (lato modelview)

- Manteniamo un puntatore alla matrice corrente (quale tra modelview_matrix e projection_matrix si va modificando): vedi `MatrixMode e current_matrix in` softogl.h
- `Translate(x,y,z)` rimpiazza (`*current_matrix`) con (`*current_matrix`)`*T` dove `T` è la matrice di traslazione che trasla il punto di x,y,z unità
- `Rotate(..),Scale(..), LookAt(..)` operano nello stesso modo

5) Modifica delle matrici (lato projection)

- `Orto(…)` rimpiazza (`*current_matrix)` con (`*current_matrix)*T` dove `T` è la matrice che realizza la trasformazione dal volume di vista della proiezione *ortogonale* al volume di vista canonico
- `Frustum(…)` rimpiazza (`*current_matrix)` con (`*current_matrix)*T` dove `T` è la matrice che realizza la trasformazione dal volume di vista della proiezione *prospettica* al volume di vista canonico

6) Definizione di viewport_matrix

- `Viewport(minx,miny,maxx,maxy)` **definisce** la matrice viewport_matrix che trasforma dalla finestra di vista canonica [-1,1]x[-1,1] alle coordinate del device (pixels) specificate

Uso di templates

- Le classi template sono classi la cui definizione può essere paramerizzata su altre classi o tipi enumerati
- Es:

```
template <class tipo_numero>
class Punto2dim{
tipo_numero x,y:
}

Punto2dim <float> punto_float;
Punto2dim <double> punto_double;
Punto2dim <int> punto_int;
```

Uso di templates

- Le funzioni template sono funzioni la cui definizione può essere paramerizzata su classi o tipi enumerati
- Es:

```
template <class tipo_numero >
tipo_numero somma(tipo_numero a, tipo_numero b){
return a+b:
}

int s=Somma<int>(2,3);
float d=Somma<float>(2.1,3);
```

Standard template library

- Libreria che implementa strutture dati e algoritmi di base: vettori,liste,code,hashmap…inserzione, cancellazione ordinamento
- ES: un vettore di interi

```
…….
std::vector<int> h;
 h.push_back(3);
 h.push_back(2);
```

H[0] vale 3

H[1] vale 2

H[2] darebbe errore, nel vettore si sono solo due elementi

Standard template library

```
std::list<int> h;
 h.push_back(3);
 h.push_back(2);
```

Il primo elemento della lista contiene 3

Il secondo elemento contiene 2

- Le liste non sono contigue in memoria e l'operatore [] non è disponibile
- Come si fa a scorrere una lista? (ma anche un vettore.?)

I container

- Vettori, liste, code,stack etc…sono tutti **container stl**
- Tutti i container stl hanno delle caratteristiche in comune, una di queste è:
- L'iteratore

```
void func(mio_tipo a){….}
void main(){
std::vector<mio_tipo> mio_vettore;
….
std::vector<mio_tipo>::iterator i;
for(i = mio_vettore.begin(); i != mio_vettore.end(); ++i)
func(*i);
}
```

Esempio “utile”

```
struct EdgeTableElement{…..};
Typedef std::list< EdgeTableElement > ScanLine;
Typedef std::vector<ScanLine> EdgeTable;
```