Grafica Computazionale

Efficienza

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

Efficienza della pipeline

- Il modo di specificare i poligoni da disegnare visto finora è chiamato *immediate mode:*

```
glBegin(GL_TRIANGLES);
glVertex3f(…);
glVertex3f(…);
glVertex3f(…);
…
glEnd();
```

- Ogni istruzione trasferisce dati da ram a memoria video
- Molto spesso l'insieme di primitive geometriche (triangoli) non cambia da frame a frame
- Quanto costa il rendering di un triangolo? Come si potrebbe risparmiare?

Seguiamo il triangolo nella strada verso il monitor

- …e vediamo se qualche passo rimane invariato nell'assunzione che l'insieme dei triangoli non cambi

- Sottosistema geometrico
  - Trasformazioni geometriche
  - Clipping
  - Ligthting
- Sottosistema raster
  - Triangle setup
  - Scan conversion
    - Interpolazione colore
    - Interpolazione z
    - Interpolazione texture
    - Interpolazione !

E questo da dove esce?

Triangle setup

- Calcolo di tutto quello che servirà durante la rasterizzazione
- In generale: gradienti
  - Pendenza degli spigoli
  - Gradiente colore
  - Gradiente coordinate texture

Invarianti…

- Se cambio il punto di vista e luci, devo ricalcolare tutto.
- E se cambio solo le luci?
  - Rimangono valide le edge functions e il gradiente delle coordinate textures
- E se cambio solo la posizione?
  - Rimangono valide le edge function e le componenti view independent del lighting per vertice (tutto tranne la componente speculare)
- Un po’ pochino, e impossibile da mettere a frutto

La vera invariante

- Se l'insieme dei triangoli non cambia, la vera invariante è che l'insieme dei triangoli non cambia!
- Come viene sfruttata?
  - I dati (vertici, normali etc..) permangono in memoria video
- Vantaggi
  - Una volta specificato l'insieme, la CPU deve solo invocare il disegno di quell'insieme (una istruzione) e non rispecificarlo tutto per ogni frame: una sola chiamata alla API grafica
  - Si elimina la latenza dovuta al trasferimento dei dati
- Il nome: i modi di rendering che mantengono una copia dei dati in memoria video sono chiamati *retained modes*

Retained modes: Display lists

Creazione

```
// inizia a definire una lista.
glNewList( id_list, GL_COMPILE );
//disegna
glBegin(GL_*);
glVertex3f(…);
…
glEnd();

glBegin(GL_*);
glVertex3f(…);
…
glEnd();
…
..
glEndList();//memorizza su memoria
      video tutto quello disegnato da
      glNewList in avanti
```

Uso

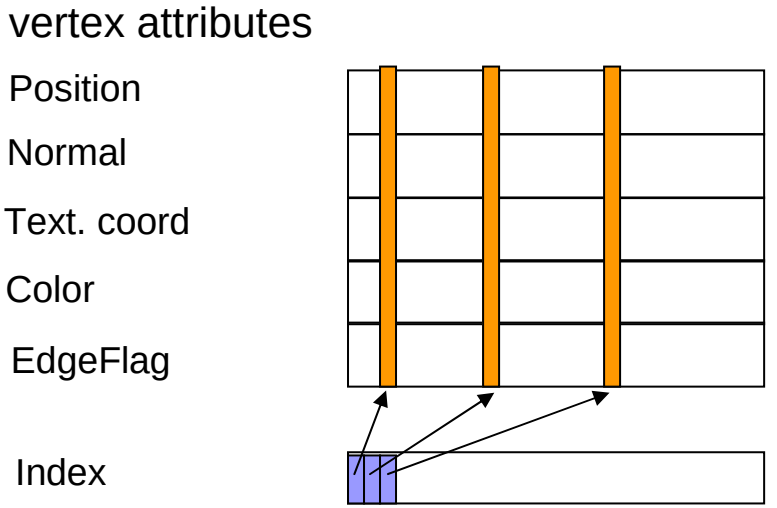
```
// ridisegna tutto quello memorizzato
      nella displaylist id_list
glCallList( id_list);
```

Vantaggi:
- una sola chiamata alla libreria
- Dati memorizzati in memoria video

Limiti
- Non molto flessibile
- Memory intensive

Retained modes: Vertex arrays

- Nella memoria lato client si memorizza l'insieme dei vertici e relativi attibuti
- Con apposite chiamate si comunica al server dove sono i dati memorizzati (puntatori in memoria)




```
float   positions [100*3];
float   normals [100*3];
// riempo i vettori positions e normals
…
// dichiaro dove sono
glVertexPointer(3,FLOAT,0, positions );
glVertexPointer(3,FLOAT,0, normals );
// abilito
EnableClientState (VERTEX_ARRAY) ;
EnableClientState (NORMAL_ARRAY) ;
// disegno
glDrawArrays (GL_VERTEX, 0, 100 ) ;
```

Equivale a:

```
glBegin(GL_VERTEX);
for(int i=0; i <100;++i){
  glNormal(normals[i*3],
           normals[i*3+1],
           normals[i*3+2]);
  glVertex(positions[i*3],
          positions [i*3+1],
positions[i*3+2]);
}
glEnd();
```

Retained modes: Vertex arrays

- Vantaggi
  - Riduce le chiamate alla API
  - Possiamo cambiare dinamicamente gli attributi (posizioni, normali etc..)
- Limiti
  - I dati devono **comunque** essere trasferiti in memoria video ogni volta che viene invocata glDrawArrays o glDrawElements

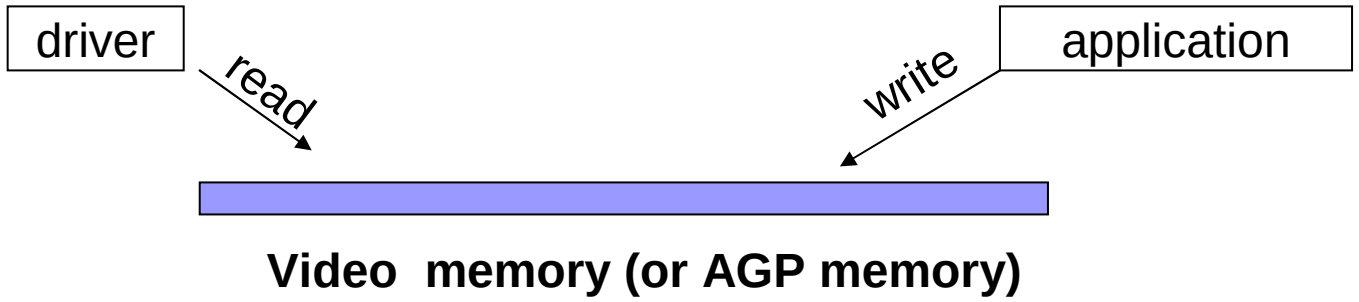
Retained modes: Vertex Array Range

- GL_NV_vertex_array_range
- Il principio: dare accesso diretto alla memoria video e alla memoria AGP
- Così da programma si riempiono direttamente gli array letti dal driver della scheda
- Un problema non da poco: sincronizzazione
  - Il driver legge e dove l'applicazione scrive
  - Cosa succede se rimpiazziamo un array proprio mente il driver lo sta leggendo?

Retained modes: VAR

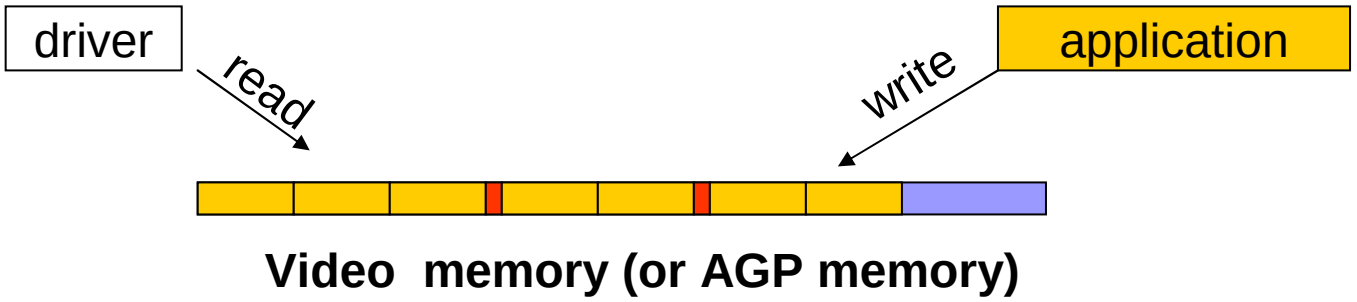
- Soluzione 1: glFinish()
  - Aspettiamo che tutte le chiamate alla API fatte prima di glFinish() siano terminate
  - Non va, se tra glDrawElements e glFinish() ho fatto altre cose devo aspettare che finisca tutto
- Soluzione 2: GL_NV_FENCE
  - È un meccanismo di sincronizzazione a grana più fine. Invece di aspettare la fine possiamo piazzare dei punti di sincronizzazione in punti qualunque della lista di primitive
  - glSetFenceNV(..): pone un token in una lista di comandi
  - glTestFenceNV(..): ritorna true se la posizione in cui è stato messo il token nella lista di comandi è stata raggiunta
  - glFinishFenceNV (..): aspetta che il fence specificato sia stato raggiunto

Retained modes: VAR



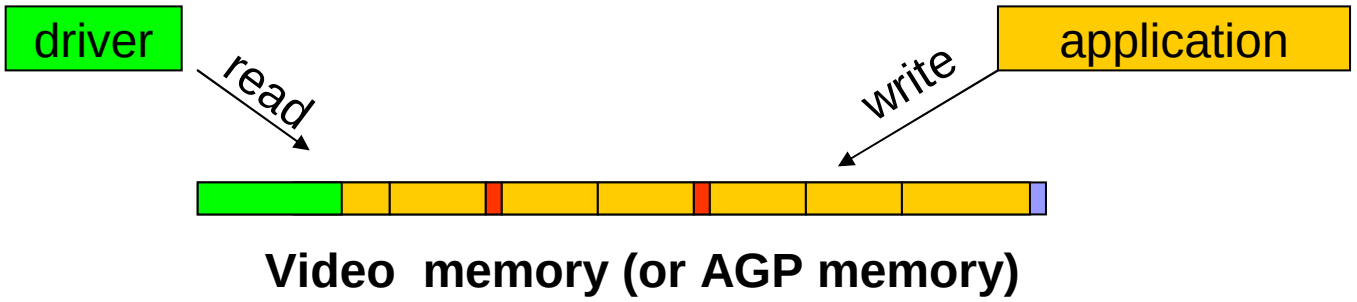
- AGP memory: memento. Porzione della RAM acceduta direttamente dai driver della scheda via la Accellerated Graphics Port (AGP)

Retained modes: VAR



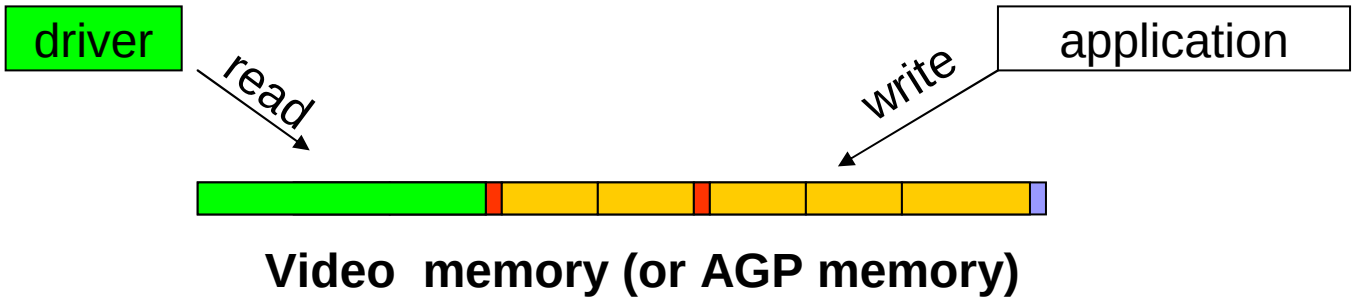
L’applicazione alloca una porzione di memoria video (o di memoria AGP) e ci copia degli array (in senape). Ogni tanto di mette anche un FENCE (in rosso)

Retained modes: VAR



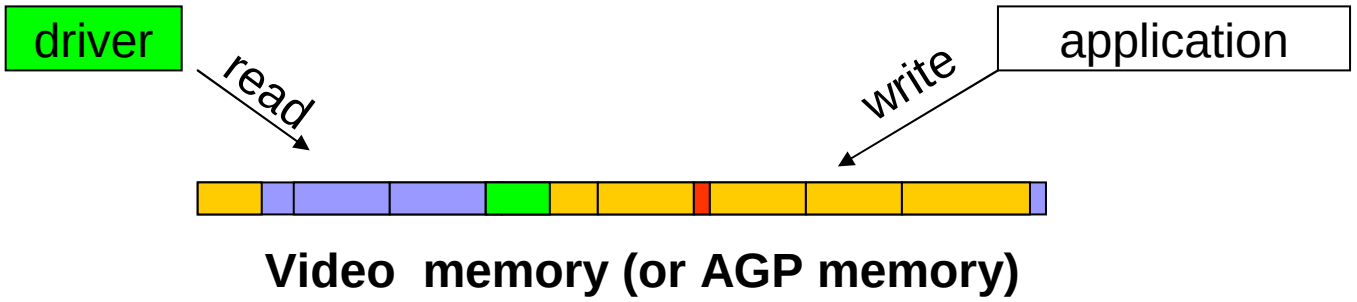
Il driver incomincia a leggere mentre l’applicazione scrive da un’altra parte

Retained modes: VAR



L’applicazione si blocca su un glFinishFenceNV(..)

Retained modes: VAR



L’applicazione può di nuovo scrivere sulla memoria già letta

Retained modes: VAR

- Problemi con i VAR:
  - richiedono all'utente di "gestire" la memoria in modo da avere la GPU o la CPU disoccupate il mino tempo possibile. Meglio mai.
  - Aprono la memoria video all'utente, violando così il paradigma client/server
  - Si tratta di **un'estensione nvidia** (_NV_), **non** presente nelle specifiche OpenGL attuali (2.0)
  - C'è qualcosa di meglio….

Retained modes: Vertex Buffer Objects

Esempi (dalle specifiche di OpenGL 2.0):

DYNAMIC DRAW The data store contents will be respecified **repeatedly** by the application, and used many times as the source for GL drawing commands.
STATIC DRAW The data store contents will be specified **once** by the application, and used many times as the source for GL drawing commands.
…..

```
// Costruisci un buffer object
BufferData(ARRAY_BUFFER, size, const
void *data, enum usage );
```

i dati

Informazione sull'uso del buffer per ottimizzare l'uso della memoria