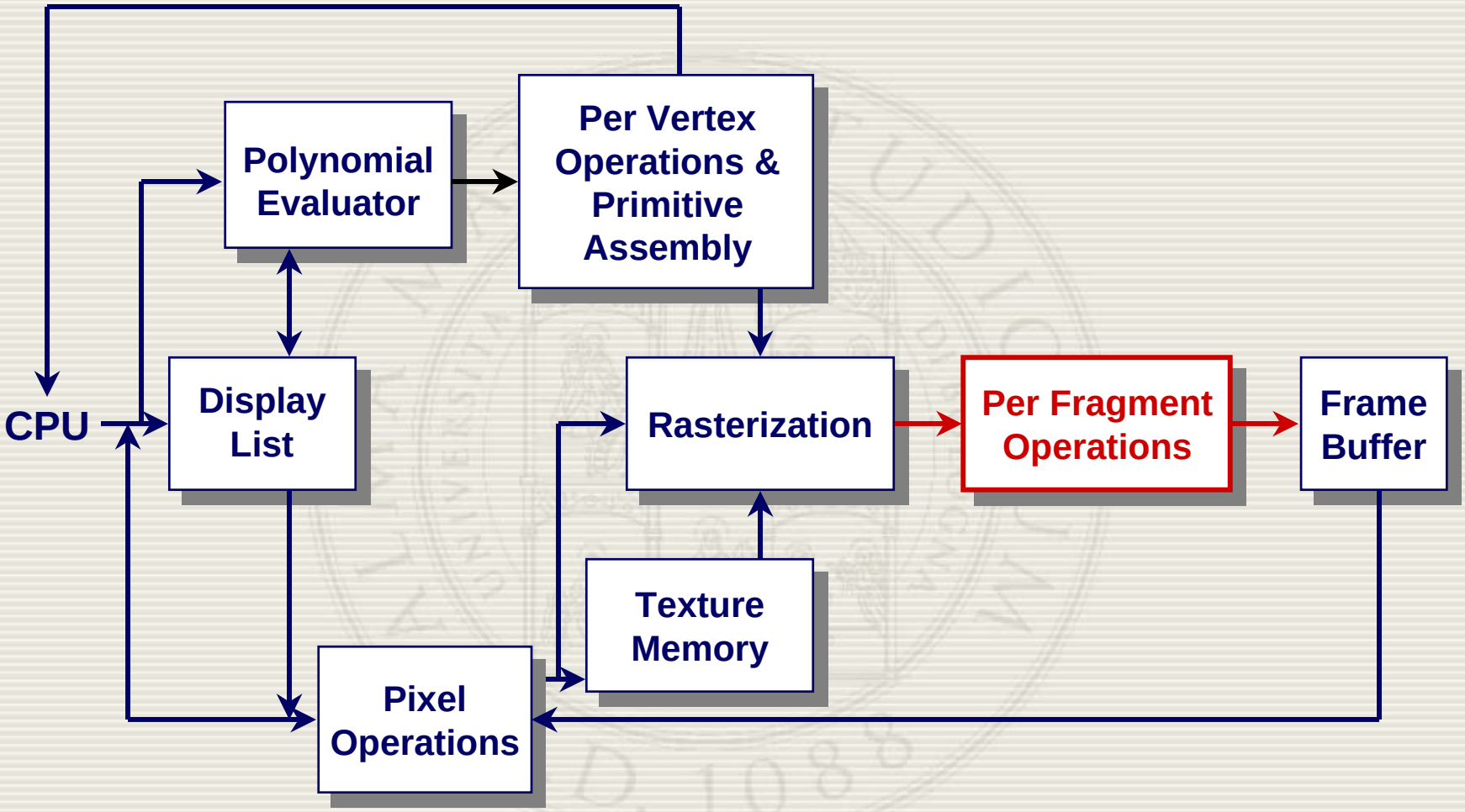
Rendering Avanzato in OpenGL e glut/SDL

OpenGL Architecture

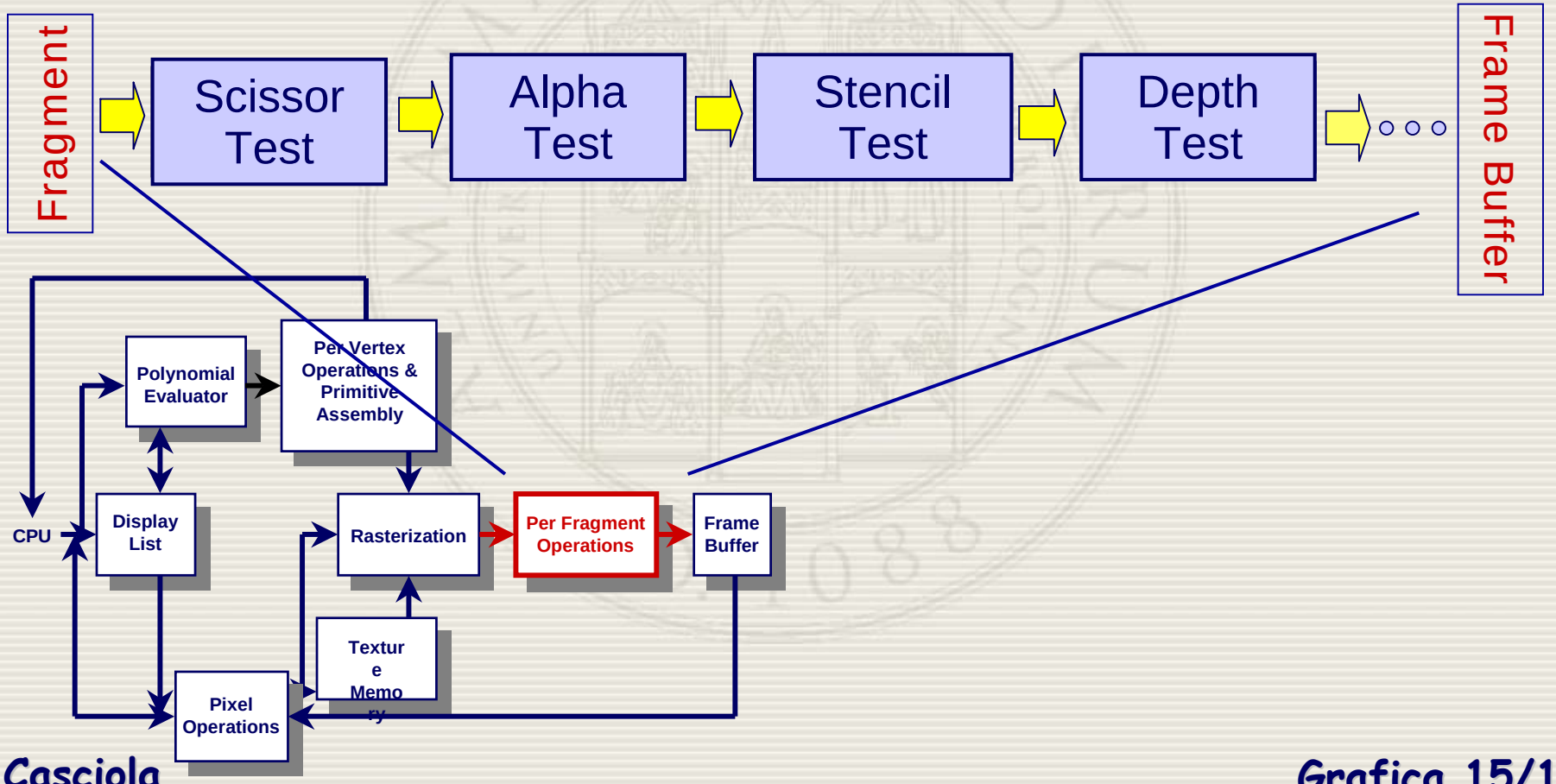


Conoscendo il funzionamento della pipeline grafica è possibile sfruttarne i meccanismi per implementare tecniche di rendering avanzato

Per-Fragment Operation

Fase della pipeline significativa per le tecniche di rendering

- Prevede l'utilizzo di **buffer**
- Prevede una **serie di test**: i “pixel” che sopravvivono a tutti i test vengono scritti nel buffer definitivo (**Color/Frame Buffer**) per essere disegnati



OpenGL Buffer

- I buffer OpenGL memorizzano/conservano un certo numero di informazioni per ogni pixel
- 5 tipi di buffer:
  - Color/Frame buffer
  - Depth buffer
  - Stencil buffer
  - Accumulation buffer
  - Auxiliary buffer
- Ogni buffer ha una ben precisa funzione; comunque si possono usare in un qualunque modo. Alla base delle tecniche di rendering c'è la manipolazione dei buffer
- Per essere usato, un buffer deve essere **allocato**

glutInitDisplayMode()

SDL_GL_SetAttribute()

Frame Buffer

- Il frame buffer o Color Buffer memorizza i pixel che verranno accesi sullo schermo
- Un applicazione può usare 1 o 2 Frame Buffer
  - Quando si usa 1 frame buffer si dice "singlebuffer"
  - Quando si usano 2 frame buffer si dice "doublebuffer" (questo è il modo più comune e di default in SDL 2.0)

```
glutInitDisplayMode(GLUT_DOUBLE)

SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER,1)
```

Esempio: Double Buffering

- Si richiede color buffer e double buffer

  glutInitDisplayMode*(GLUT_RGB|GLUT_DOUBLE);*

  SDL_GL_SetAttribute(SDL_GL_BUFFER_SIZE,24)
  SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER,1)

- Si inzializza (clear) il color buffer

  glClear*(GL_COLOR_BUFFER_BIT);*

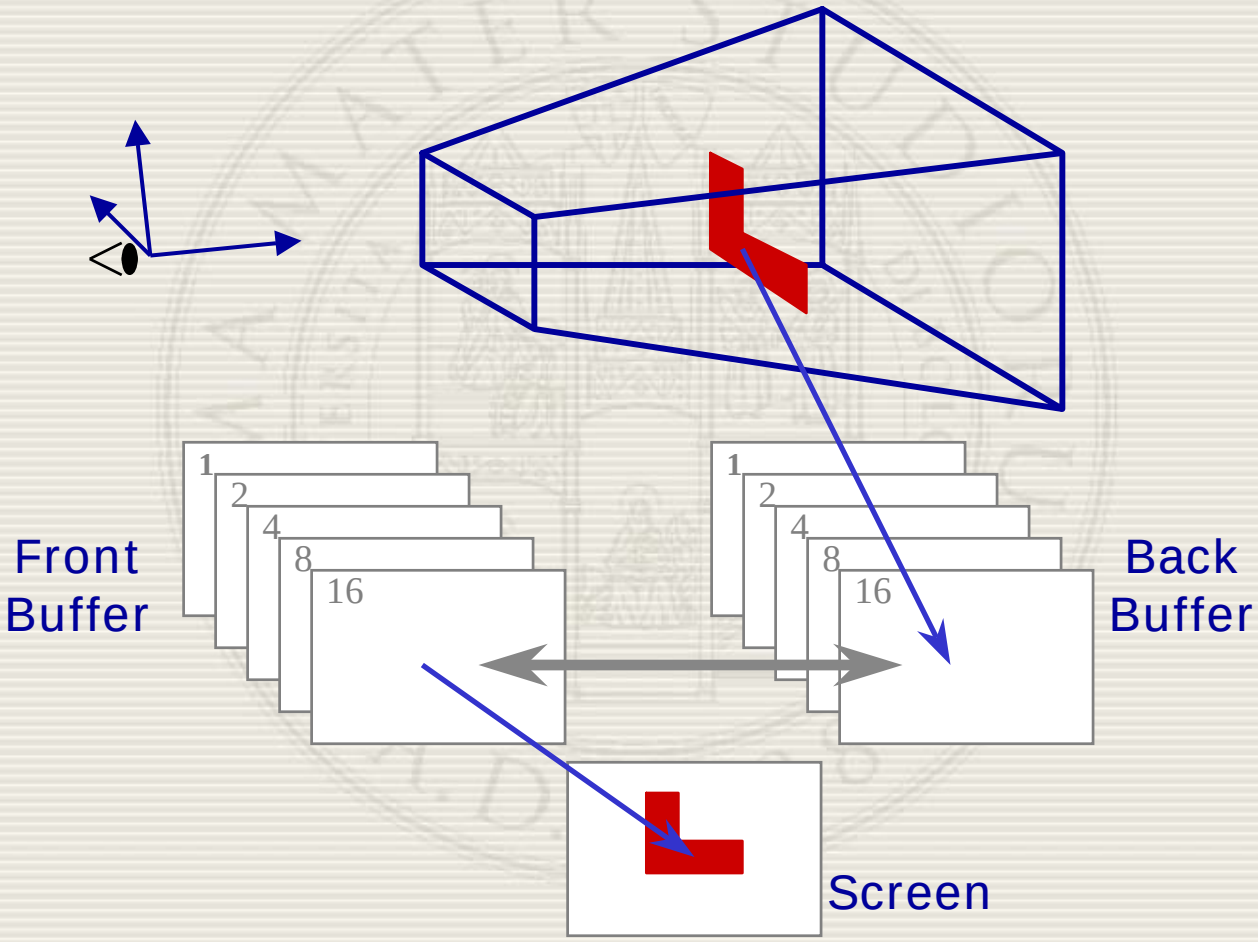
- Si rende la scena ...

- Si scambiano (swap) il front e il back buffer

  glutSwapBuffers()
  SDL_GL_SwapWindow()

- Si ripetono i passi precedenti

Double Buffering



OpenGL Test

- Collegati ai buffer ci sono i test:
  - Scissor test
  - Alpha test
  - Depth test
  - Stencil test
- Si tratta di un'espressione booleana valutata per ogni fragment
  - Se il test risulta True, il test passa e il fragment continua la pipeline;
  - Se il test fallisce, il fragment viene scartato;
  - Tutti i test vengono eseguiti nella fase di "fragment-processing" della pipeline.
- Per avere effetto, ogni test deve essere abilitato:

glEnable()

Buffer & Test

- I buffer e i test sono associati

  Abbiamo già visto un esempio di questo nel depth buffer che implementa l'algoritmo Z-buffer
- Ricorda:
  - **Allocate** buffer
  - **Enable** test

Buffer	Corresponding Test
--	Scissor Test
Color Buffers	Alpha Test
Depth Buffer	Depth Test
Stencil Buffer	Stencil Test
Accumulation Buffer	--

Depth Buffering con glut

- Alloca un depth buffer

```
glutInitDisplayMode( GLUT_RGB | GLUT_DEPTH );

SDL_GL_SetAttribute(SDL_GL_BUFFER_SIZE,24)
SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE,24)
```

- Abilita il depth test

```
glEnable( GL_DEPTH_TEST );
```

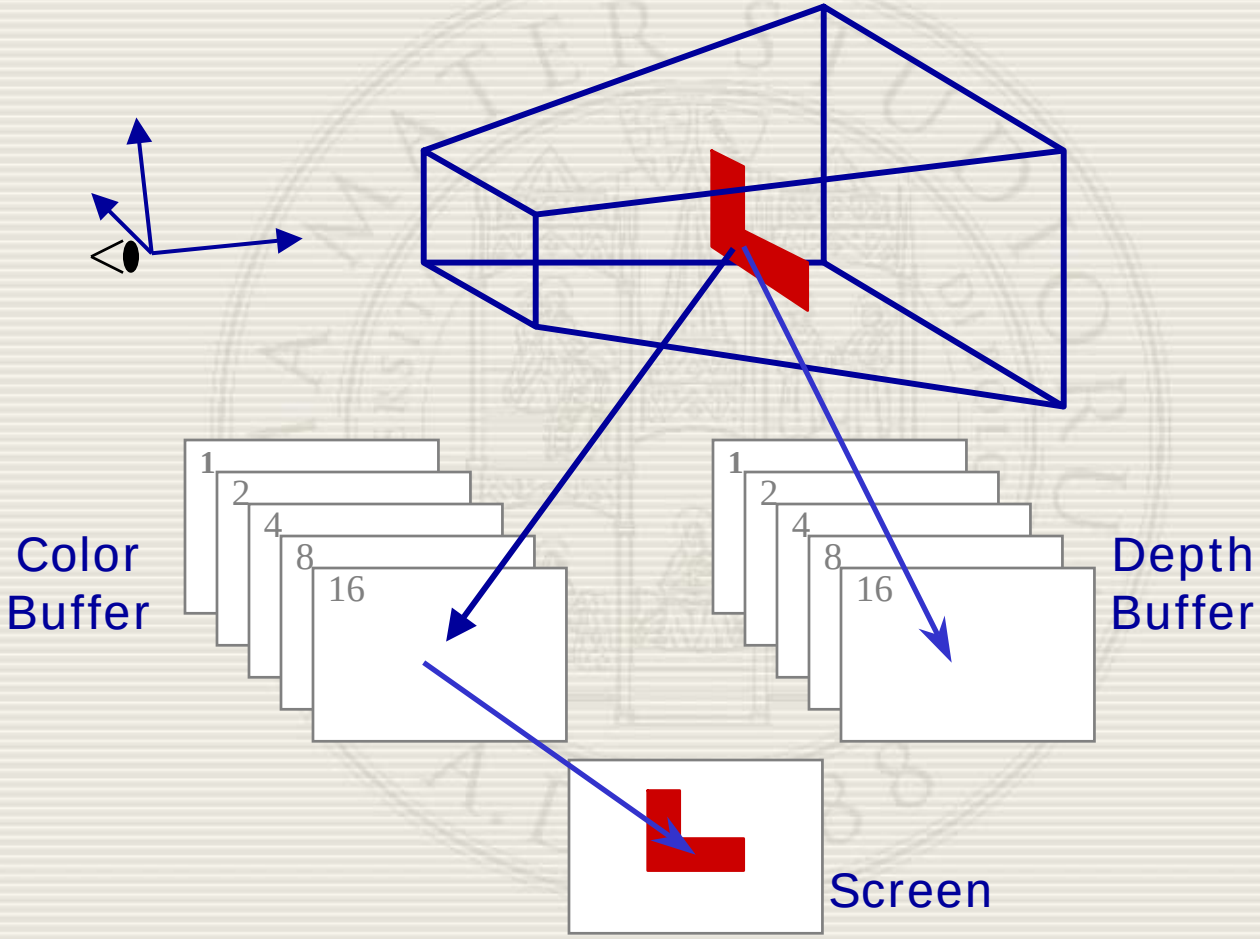
- Inizializza (clear) il color e il depth buffer

```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

- Si rende la scena, quindi

```
glFlush();
```

Depth Buffering per Hidden Surface Removal



Clearing

- I Buffer vengono inizializzati (clear) con glClear

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |
        GL_STENCIL_BUFFER_BIT | GL_ACCUM_BUFFER_BIT);
```

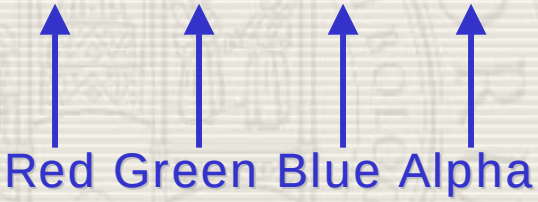
- Il valore da memorizzare nel buffer per ogni pixel viene settato con:

```
glClearColor(GLclampf red, GLclampf green,
             GLclampf blue, GLclampf alpha )
glClearDepth(GLclampd depth )
glClearStencil(GLint s )
glClearAccum(GLfloat red, GLfloat green,
             GLfloat blue, GLfloat alpha )
```

Masking

- Le maschere (Mask) determinano se un buffer (o parte di un buffer) deve essere scritto
- Per esempio,

```
glColorMask(false, true, true, true);
```

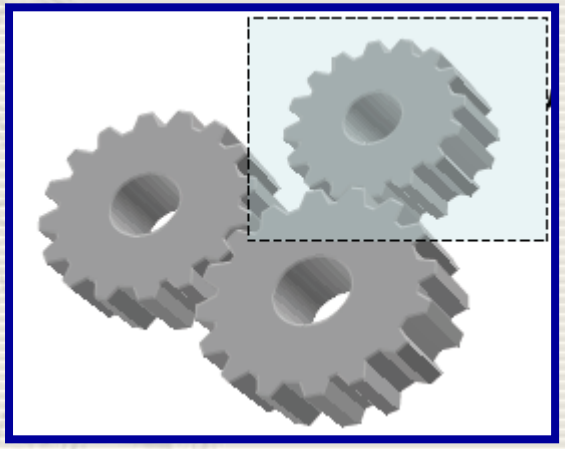


Significa che la componente Red del Color buffer non dovrà essere cambiata, le altre sì

- Un Mask influisce su tutti i comandi che dovrebbero cambiare il buffer, anche **glClear**

Scissor Test

- Il test più semplice è lo "*scissor test*"
  - Permette di restringere il disegno ad una porzione rettangolare della viewport
  - Per abilitarlo:
    glEnable(GL_SCISSOR_TEST);
  - Per definirlo:
    glScissor(*x*, *y*, *width*, *height*);
    i parametri sono come quelli per la **glViewport**.



Alpha Test

- glAlphaFunc(func, value);
  - Scarta pixel sulla base del suo valore alpha;
  - Per abilitarlo: glEnable(GL_ALPHA_TEST);
  - Usa alpha come un mask;
  - func può essere uno dei seguenti:
    - **GL_NEVER,**
    - **GL_LESS,**
    - **GL_EQUAL,**
    - **GL_LEQUAL,**
    - **GL_GREATER,**
    - **GL_NOEQUAL,**
    - **GL_NOTEQUAL,**
    - **GL_ALWAYS**

Accumulation Buffer (AB)

- L'AB permette di miscelare insieme (blend) differenti immagini 2D
- L'AB permette di memorizzare colori RGBA, come il Color buffer
- Comandi speciali permettono di miscelare un color buffer con l'AB (anche più volte) e poi di trasferire il contenuto dell'AB ad un color buffer
- Per allocare l'AB si usa:

```
glutInitDisplayMode(GLUT_ACCUM)

SDL_GL_SetAttribute(SDL_GL_ACCUM_RED_SIZE,8)
SDL_GL_SetAttribute(SDL_GL_ACCUM_GREEN_SIZE,8)
SDL_GL_SetAttribute(SDL_GL_ACCUM_BLUE_SIZE,8)
SDL_GL_SetAttribute(SDL_GL_ACCUM_ALPHA_SIZE,8)
```

- Non c'è nessun Test da abilitare (enable)

Accedere all'AB

- Si possono effettuare cinque operazioni sull'AB. Vengono realizzate sull'intero buffer in una volta:
  - L'AB può essere inizializzato (cleared)
  - Il contenuto di un color buffer può essere moltiplicato per un valore e poi copiato sull' AB.
  - Il contenuto di un color buffer può essere moltiplicato per un valore e poi aggiunto all'AB.
  - Su ogni pixel dell'AB si può fare un'operazione aritmetica (× or +)
  - Il contenuto dell'AB può essere moltiplicato per un valore e copiato in un color buffer
- La prima operazione descritta (clear), viene fatta con **`glClear`**:

```
glClearAccum(R, G, B, A);          // specifica un colore
                                   // (come glClearColor)
                                   // per l'AB
glClear(GL_ACCUM_BUFFER_BIT);      // Clear AB
```

- Le altre quattro operazioni si effettuano con il comando glAccum

Accedere all'AB

- glAccum ha due parametri:
  - **GLenum** dice quale operazione effettuare
  - **GLfloat** passa un valore costante
- Per moltiplicare il contenuto di un color buffer per un valore e copiare il risultato sull'AB:

  glAccum(GL_LOAD, *value*);

  - Usa il color buffer selezionato per leggere
    Si usi **glReadBuffer** per cambiarlo

- Per moltiplicare il contenuto di un color buffer per un valore e aggiungere il risultato all'AB:

  glAccum(GL_ACCUM, *value*);

Accedere all'AB

- Per moltiplicare il contenuto dell'AB per un valore:

  glAccum(GL_MULT, *value*);

  - C'e anche **GL_ADD**, per addizionare anziché moltiplicare

- Per moltiplicare il contenuto dell'AB per un valore e copiare il risultato su un color buffer:

  glAccum(GL_RETURN, *value*);

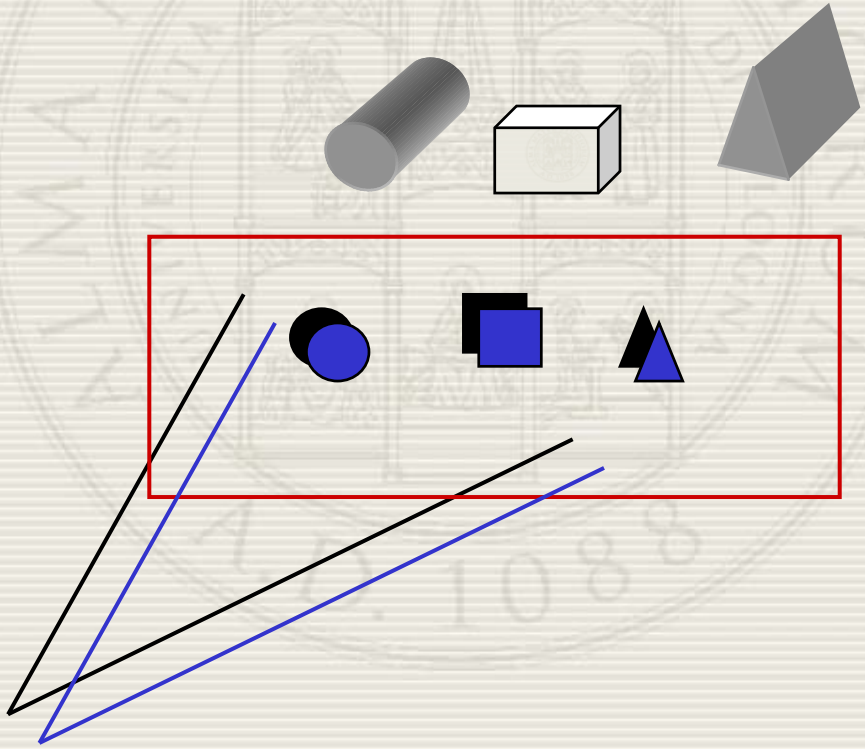
  - Usa il color buffer selezionato per il disegno
    Si usi **glDrawBuffer** per cambiarlo

Accumulation Buffer: Applicazioni

- Compositing
  Combina più immagini in una singola immagine
- Full Scene Antialiasing
  Media (smooth) i lati/bordi a scalini di tutti gli oggetti
- Depth of Field
  Simula la messa a fuoco di una camera su un singolo oggetto
- Filtering
  Rende la stessa immagine con un piccolo offset di pixel
- Motion Blur
  Rende la stessa immagine da differenti posizioni camera

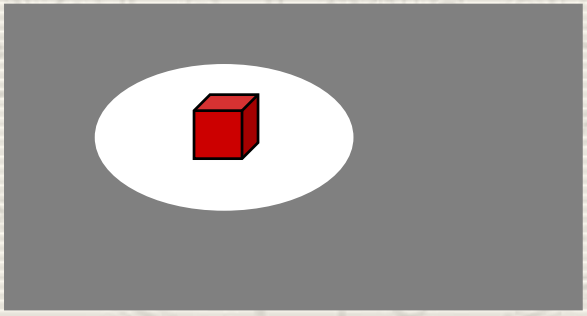
Full Scene Antialiasing

- Ogni volta che l'osservatore si sposta, l'immagine risulta un po' spostata
  - Ci saranno differenti effetti aliasing in ciascuna immagine
  - Mediando le immagini, con l'utilizzo dell'AB, si mediano e quindi rimuovono gli artefatti dovuti all'aliasing.



Stencil Buffer

- Viene usato per controllare il disegno basato su valori
  - I fragment che falliscono lo stencil test non sono disegnati
  - Esempio: si crea un Mask nello stencil buffer e si disegnano solo gli oggetti non nella Mask area



Controllare lo Stencil Buffer

glStencilFunc(*func, ref, mask*);

- Confronta i valori nel buffer con **ref** usando **func**
- Si applica solo per i pixel nel **mask** che sono 1
- **func** è una delle funzioni standard di confronto

glStencilOp(*fail, zfail, zpass*);

- Permette modifiche nello stencil buffer basate sul passaggio o meno dello stencil test e depth test: **GL_KEEP**, **GL_INCR**

Creare un Mask

```
glInitDisplayMode( ...|GLUT_STENCIL|... );
glEnable( GL_STENCIL_TEST );
glClearStencil( 0x0 );

glStencilFunc( GL_ALWAYS, 0x1, 0x1 );
glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );
```

Disegna il mask

Usare lo Stencil Mask

1. Costruisce uno stencil buffer solo in lettura (read-only)

   glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

2. Disegna oggetti dove lo stencil vale 1

   glStencilFunc(*GL_EQUAL, 0x1, 0x1*);

3. Disegna oggetti dove lo stencil ha valori diversi da 1

   glStencilFunc(GL_NOT_EQUAL, 0x1, 0x1);

Riflessioni planari

Il Dinosauro viene riflesso dal pavimento.
Si disegna dino due volte, la seconda volta si usa
glScalef(1,-1,1) per generare il riflesso attraverso il pavimento

opengl_1516/opengl2/reflection/reflectdino

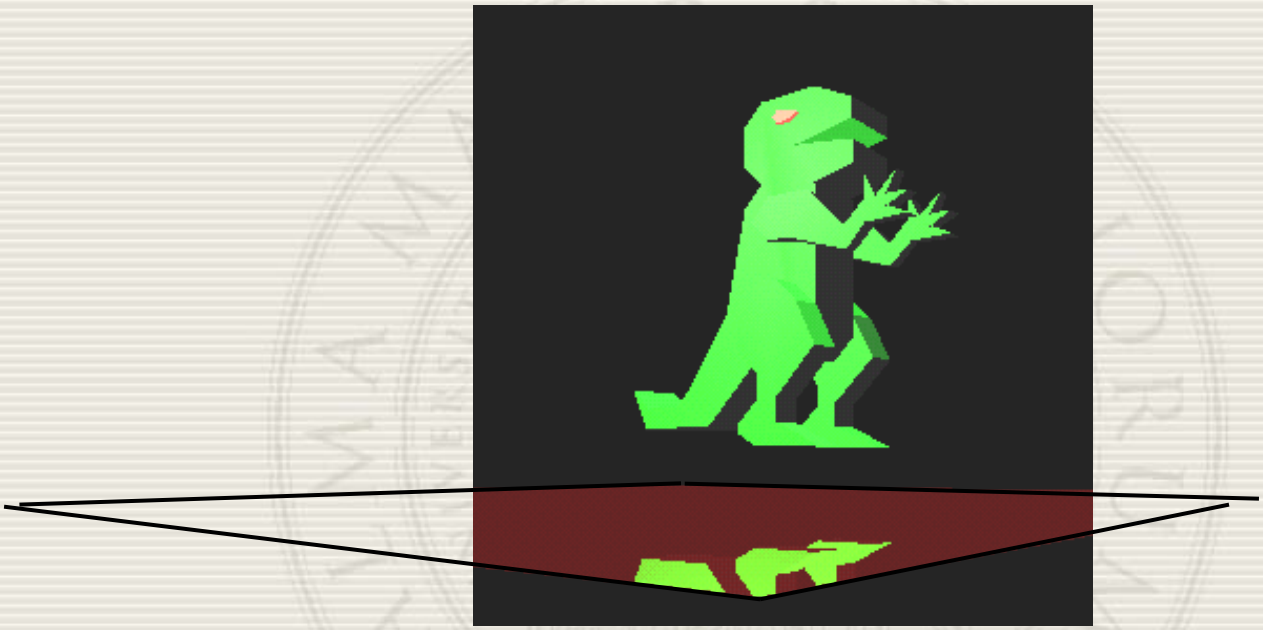
Confronta le due versioni

Bene

Male

Si noti che l'immagine riflessa di destra esce dal pavimento!

Lo Stencil mantiene il pavimento



- Inizializza lo stencil a zero.
- Disegna il poligono del pavimento nello stencil con valore 1
- Disegna il secondo Dino (la riflessione) solo dove lo stencil è 1

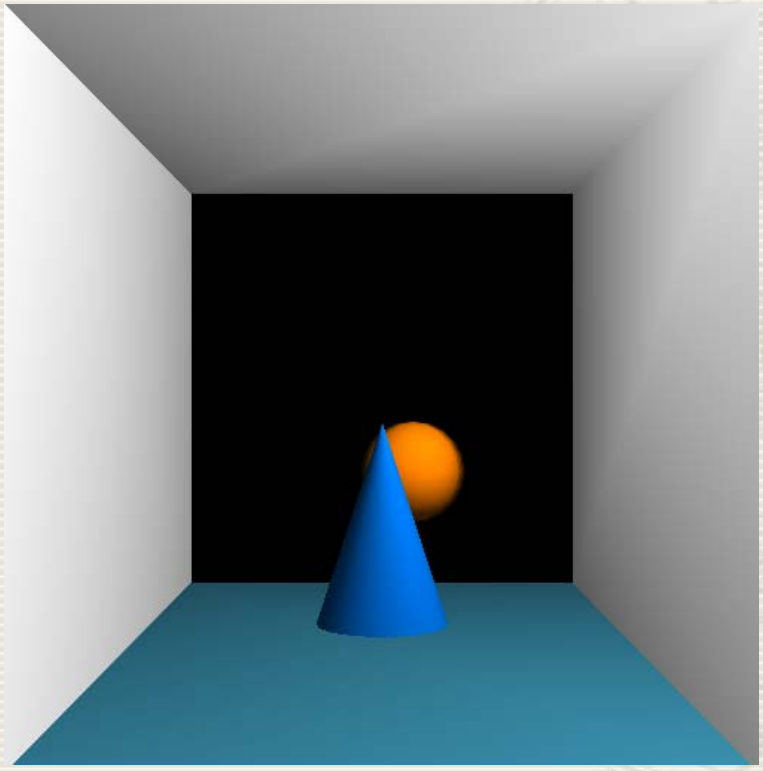
Auxilliary Buffer

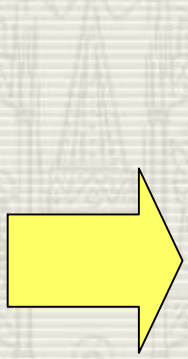
- OpenGL supporta un numero di buffer ausiliari
  - Di solito (mai) vengono supportati via hardware
    - Cioè i dati non vengono mostrati sullo schermo a meno che non vengano trasferiti sul frame buffer
    - Per questo motivo sono raramente usati
  - Sono chiamati **GL_AUXn**

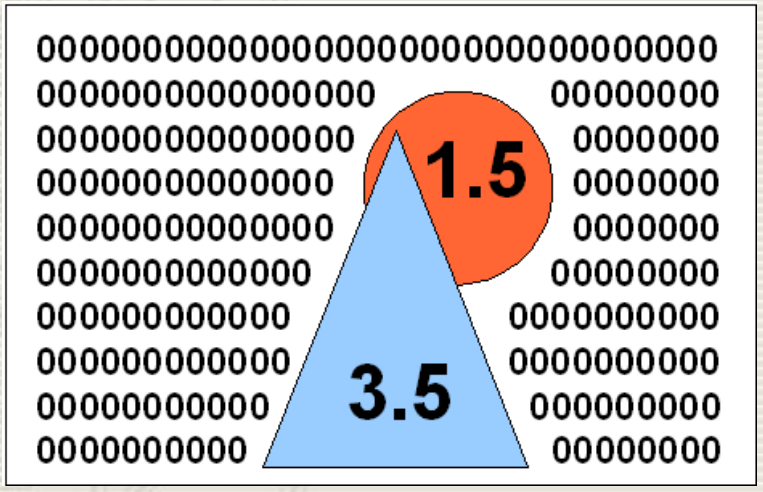
    n è il numero del buffer
- Possono essere usati come buffer di rendering

  glDrawBuffers(GL_AUXn)

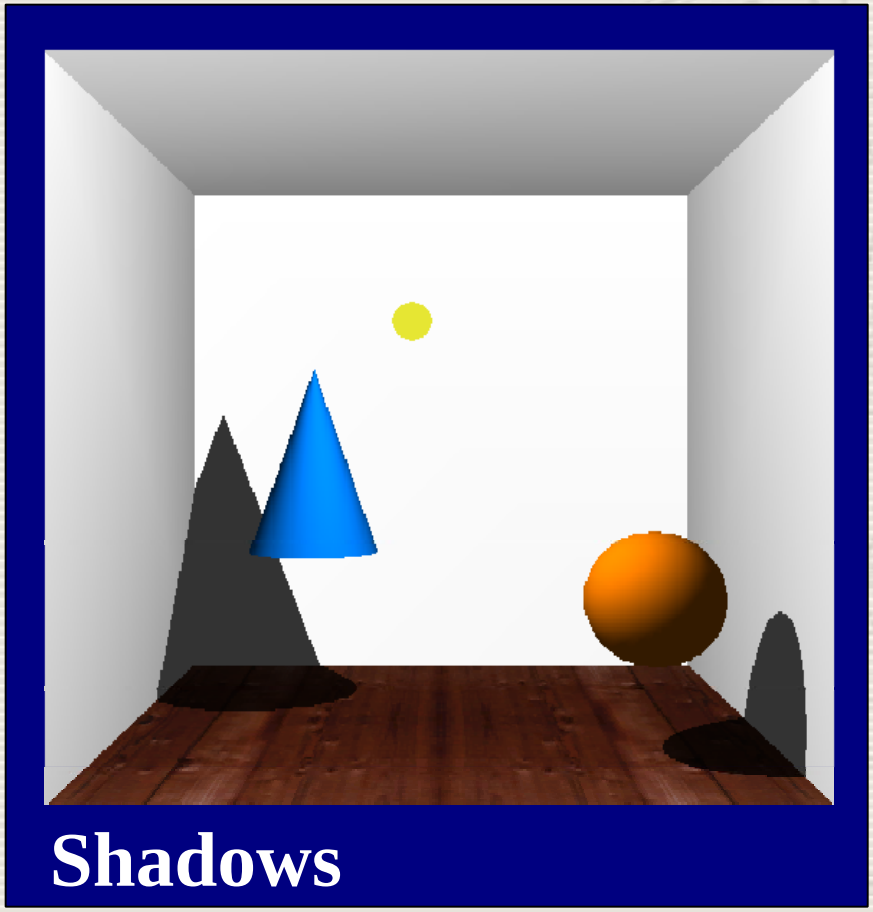
Depth Buffer: esempio



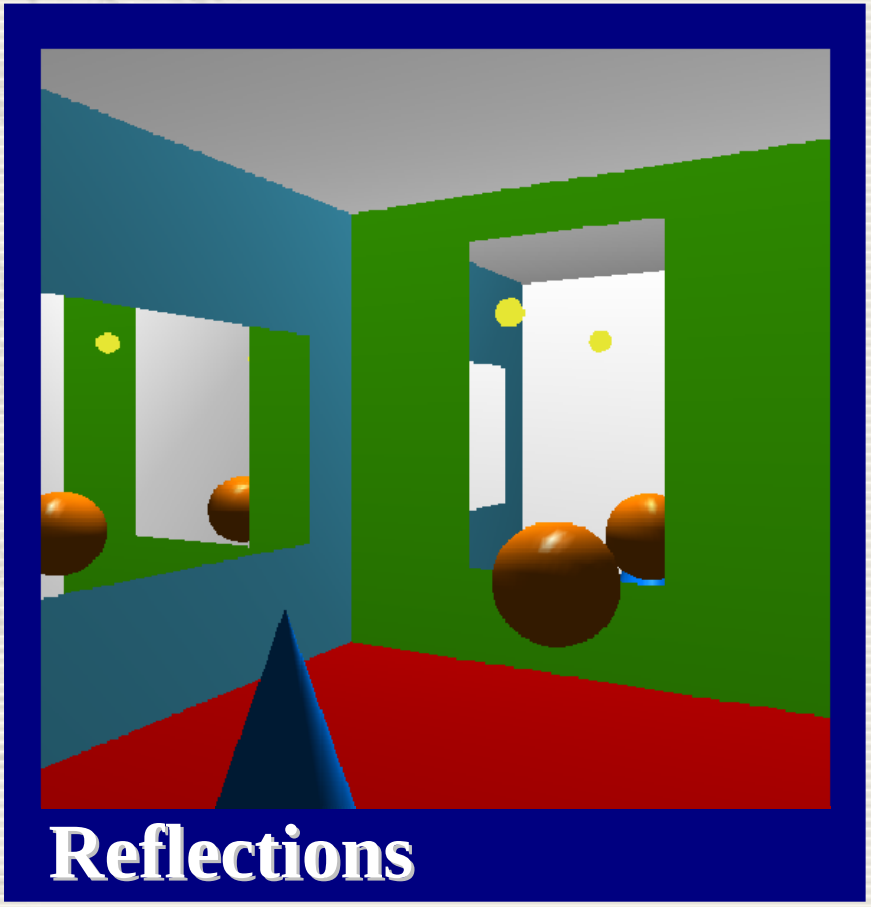




Stencil Buffer: esempi

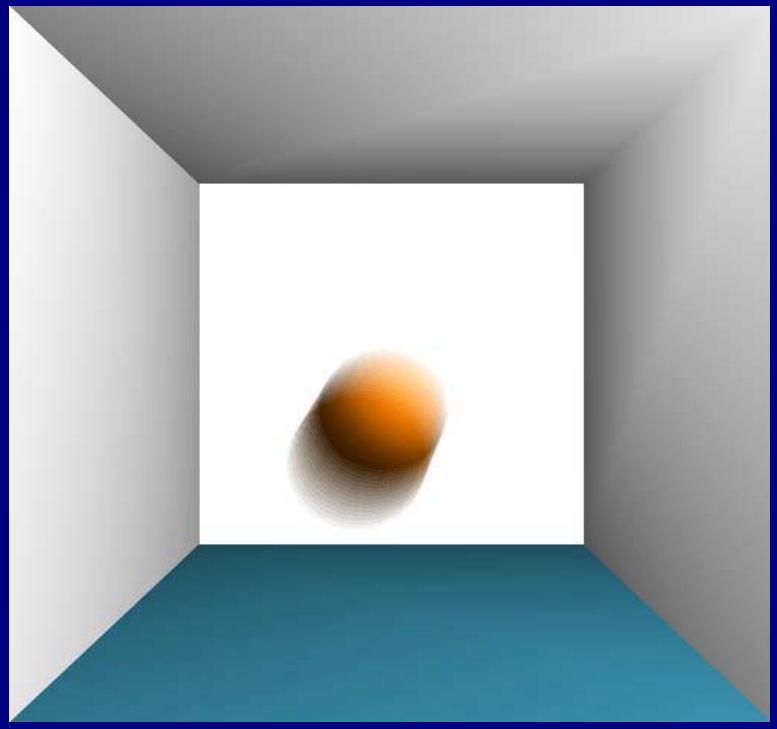


Shadows

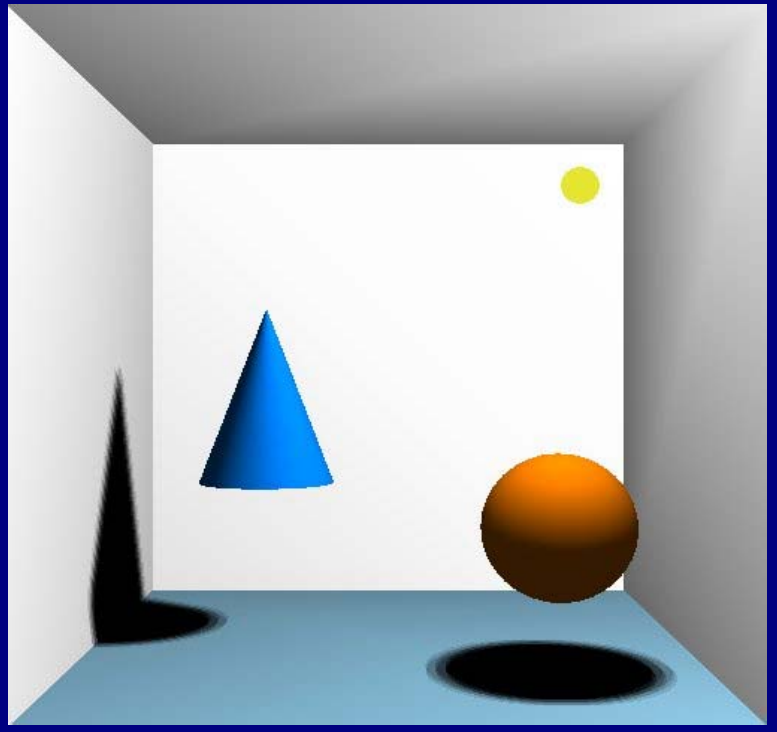


Reflections

Accumulation Buffer: esempi



Motion Blur



Soft Shadow