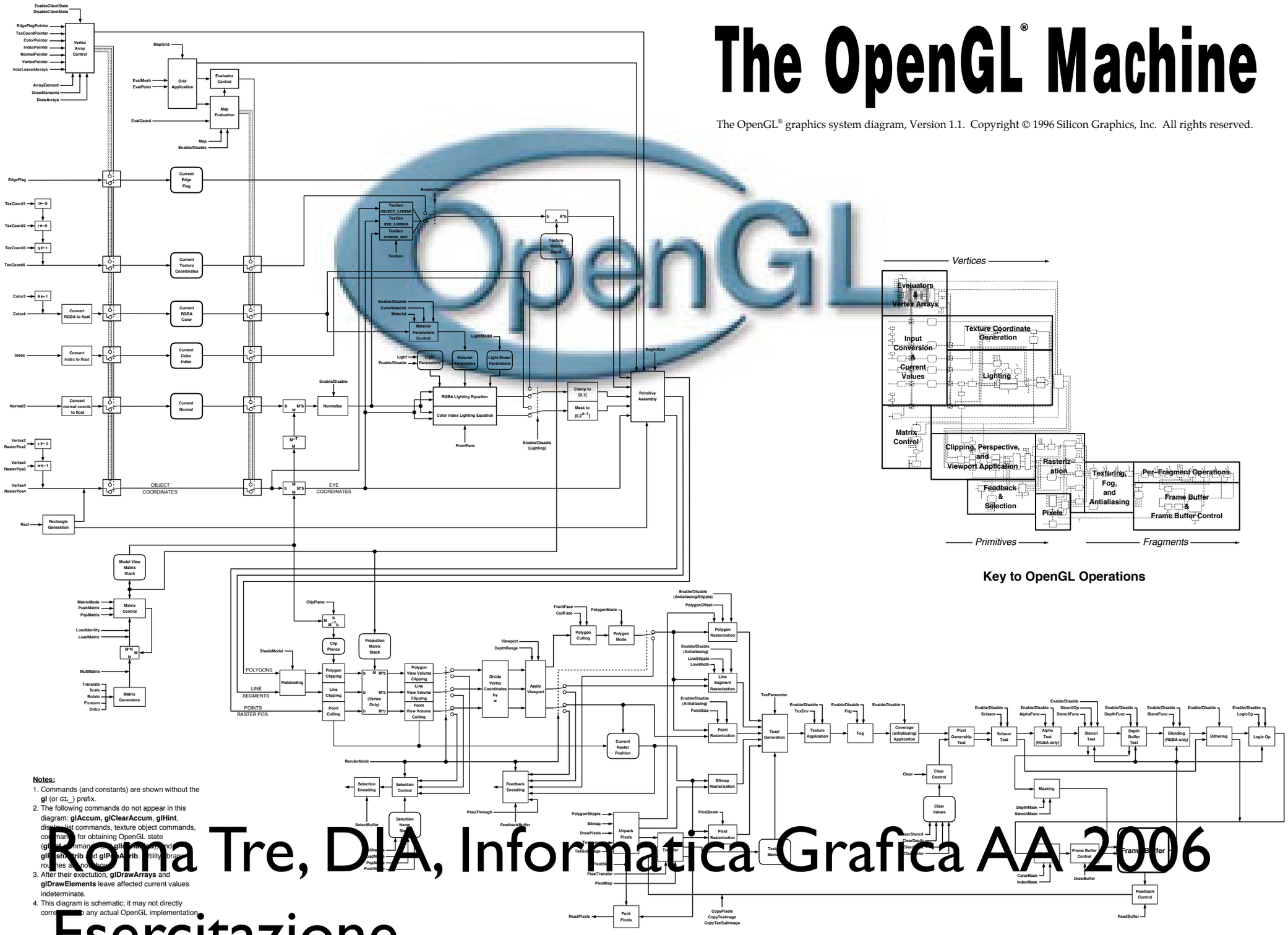
The OpenGL® Machine



Key to OpenGL Operations

Roma Tre, DIA, Informatica Grafica AA 2006

Esercitazione

8/3/2005

Portuesi Simone

Menu del giorno
Primitive Disegno OpenGL

* Generalità e generalità di disegno
* Punti e Primitive
* Attributi Disegno
* Colori e Punti
* Esempi
* Vertex Arrays

rtype **Name**{ϵ**1234**}{ϵ **b s i f d ub us ui**}{ϵ**v**}

(*[args ,] T arg1 , . . . , T argN [, args]*) ;

* Tipo di ritorno
* Nome Funzione
* Numero di argomenti (se variabile)
* Tipo argomenti
* Vettore di argomenti?
  T arg1, ... T arnN -> T* argV
* Altri argomenti

Letter	Corresponding GL Type
b	`byte`
s	`short`
i	`int`
f	`float`
d	`double`
ub	`ubyte`
us	`ushort`
ui	`uint`

Tipo dati OpenGL

Tipo C: GL___

GL Type	Minimum Bit Width	Description
`boolean`	1	Boolean
`byte`	8	signed 2's complement binary integer
`ubyte`	8	unsigned binary integer
`short`	16	signed 2's complement binary integer
`ushort`	16	unsigned binary integer
`int`	32	signed 2's complement binary integer
`uint`	32	unsigned binary integer
`sizei`	32	Non-negative binary integer size
`enum`	32	Enumerated binary integer value
`bitfield`	32	Bit field
`float`	32	Floating-point value
`clampf`	32	Floating-point value clamped to $[0, 1]$
`double`	64	Floating-point value
`clampd`	64	Floating-point value clamped to $[0, 1]$

Pulire Buffer

* Attributi

void **glClearColor**(GLclampf *red*, GLclampf *green*, GLclampf *blue*, GLclampf *alpha*)

void **glClearDepth**(GLclampd *depth*)

* Azione

void **glClear**(GLbitfield *mask*)

Buffer	Name
Color buffer	GL_COLOR_BUFFER_BIT
Depth buffer	GL_DEPTH_BUFFER_BIT
Accumulation buffer	GL_ACCUM_BUFFER_BIT
Stencil buffer	GL_STENCIL_BUFFER_BIT

* Esempio

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClearDepth(0.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Forzare il disegno

Inizia a disegnare

force execution of GL commands in finite time

void **glFlush**(void)

Finisci di disegnare

block until all GL execution is complete

void **glFinish**(void)

Glut double buffer

```
void glutSwapBuffers(void);
```

An implicit `glFlush` is done by `glutSwapBuffers` before it returns. Subsequent OpenGL commands can be issued immediately after calling `glutSwapBuffers`, but are not executed until the

buffer exchange is completed.

Z-Buffer

A depth buffer works by associating a depth, or distance from the viewpoint, with each pixel on the window. Initially, the depth values for all pixels are set to the largest possible distance using the **glClear()** command with GL_DEPTH_BUFFER_BIT, and then the objects in the scene are drawn in any order.

Eg:

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

Colori ed Attributi

With OpenGL, the description of the shape of an object being drawn is independent of the description of its color.

Whenever a particular geometric object is drawn, it's drawn using the currently specified coloring scheme. The coloring scheme might be as simple as "draw everything in fire-engine red," or might be as complicated as "assume the object is made out of blue plastic, that there's a yellow spotlight pointed in such and such a direction, and that there's a general low-level reddish-brown light everywhere else."

In general, an OpenGL programmer first sets the color or coloring scheme, and then draws the objects. Until the color or coloring scheme is changed, all objects are drawn in that color or using that coloring scheme.

In general OpenGL colors results from enabling certain rendering capabilities and setting the corresponding attribute while drawing as needed.

* OpenGL ha una serie di stati,modi e valori che rimangono in essere fino a quando cambiati.

* Molti modi sono legati a delle capacità di rendering da attivare/disattivare:

void **glEnable**(GLenum *cap*)

void **glDisable**(GLenum *cap*)

* Query modalità:

GLboolean **glIsEnabled**(GLenum *cap*)

* Query Variabile di Stato:

void **glGetBooleanv**(GLenum *pname*, GLboolean **params*)
void **glGetDoublev**(GLenum *pname*, GLdouble **params*)
void **glGetFloatv**(GLenum *pname*, GLfloat **params*)
void **glGetIntegerv**(GLenum *pname*, GLint **params*)

pname: Parametro da richiedere.
params: Puntatore ad area di memoria ove mettere uno o più valore/i

Eg: GLfloat[4][4] matrix;
glGetFloatv(**GL_MODELVIEW_MATRIX**, matrix)

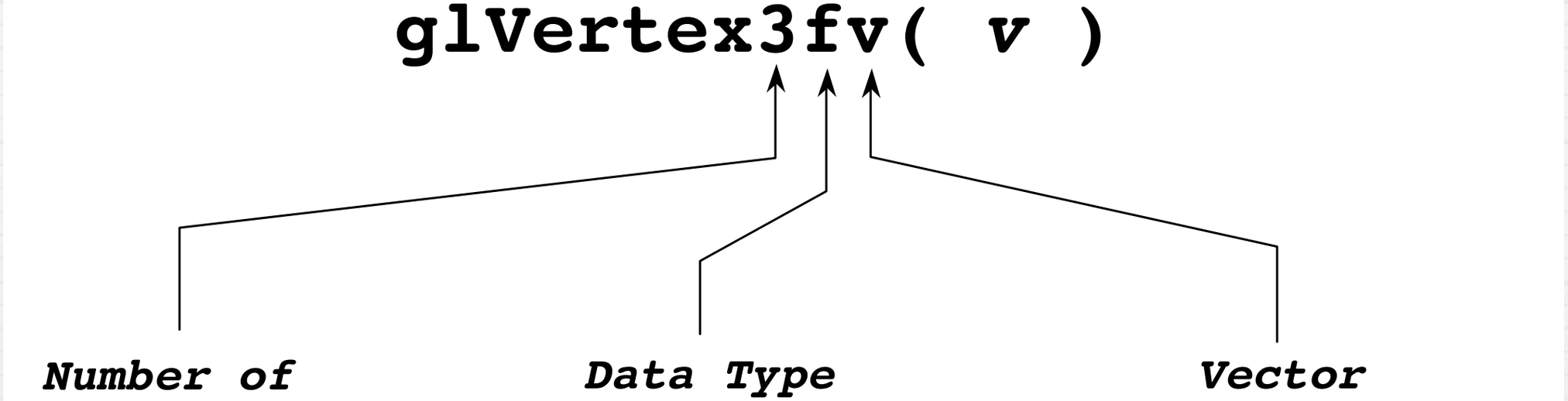
* Stack Variabili di Stato

void **glPushAttrib**(GLbitfield *mask*)
void **glPopAttrib**(void)
mask: Serie di bit indicanti classi di valore da salvare
EG: **GL_TRANSFORM_BIT:** Modalità di trasformazione ed associati
GL_VIEWPORT_BIT: Viewport
GL_LIGHTING_BIT: Materiali e luci

Primitive di Disegno

* Inizia Comando
  `glBegin(command);`
* Elenca Punti Comando
  (ed eventualmente altro ... dopo)
  `glVertex...(....);`
* Fine Comando
  `glEnd();`

Vertici



Number of components

```
2 - (x,y)
3 - (x,y,z)
4 - (x,y,z,w)
```

Data Type

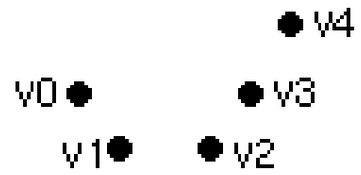
```
b  - byte
ub - unsigned byte
s  - short
us - unsigned short
i  - int
ui - unsigned int
f  - float
d  - double
```

Vector

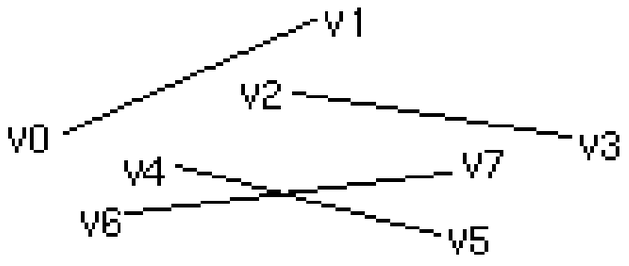
```
omit “v” for
scalar form

glVertex2f( x, y )
```

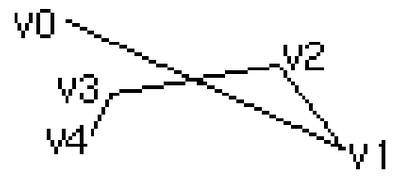
GL_POINTS	individual points
GL_LINES	pairs of vertices interpreted as individual line segments
GL_LINE_STRIP	series of connected line segments
GL_LINE_LOOP	same as above, with a segment added between last and first vertices
GL_TRIANGLES	triples of vertices interpreted as triangles
GL_TRIANGLE_STRIP	linked strip of triangles
GL_TRIANGLE_FAN	linked fan of triangles
GL_QUADS	quadruples of vertices interpreted as four-sided polygons
GL_QUAD_STRIP	linked strip of quadrilaterals



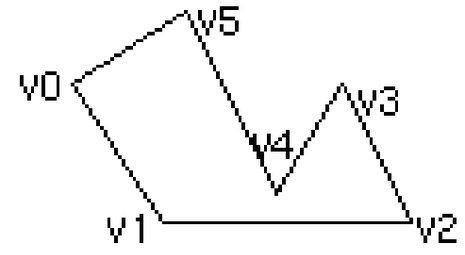
GL_POINTS



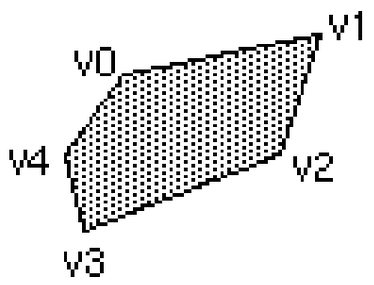
GL_LINES



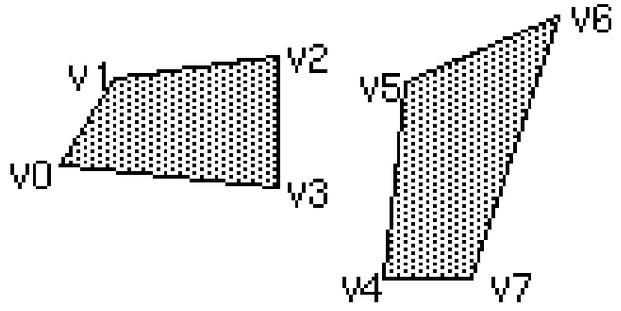
GL_LINE_STRIP



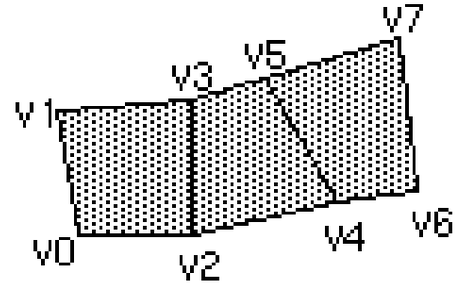
GL_LINE_LOOP



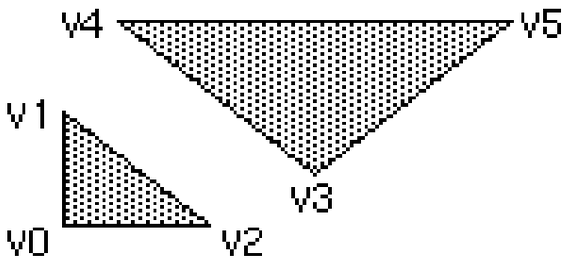
GL_POLYGON



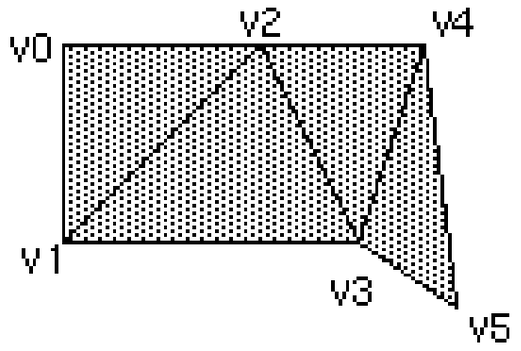
GL_QUADS



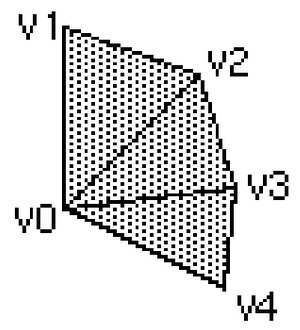
GL_QUAD_STRIP



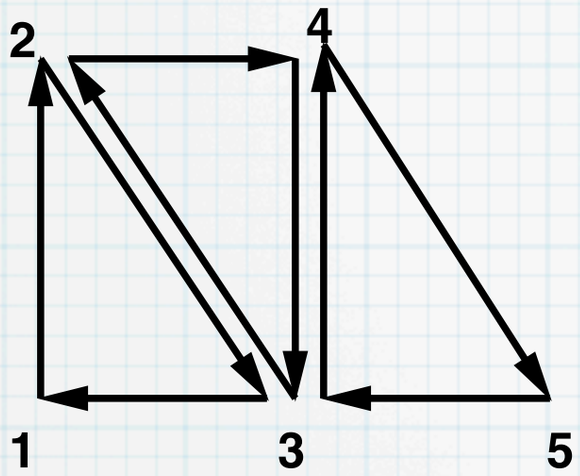
GL_TRIANGLES



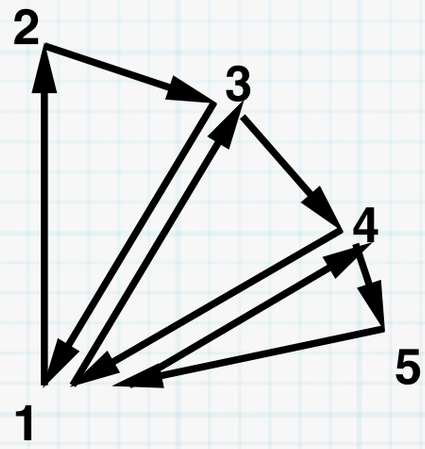
GL_TRIANGLE_STRIP



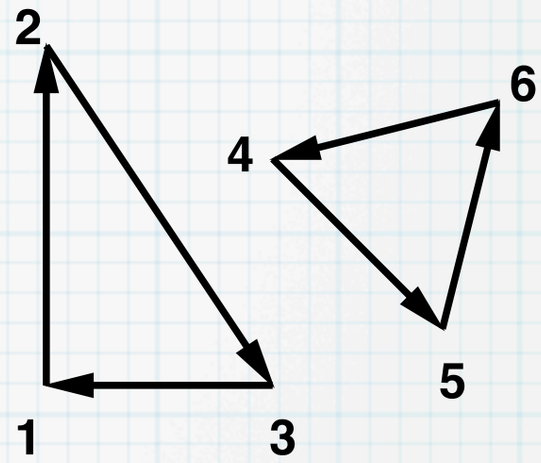
GL_TRIANGLE_FAN



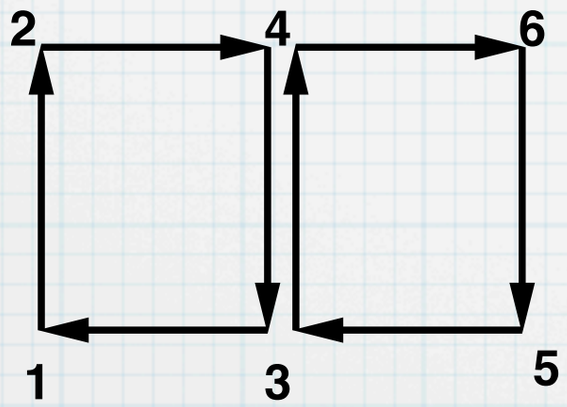
(a)



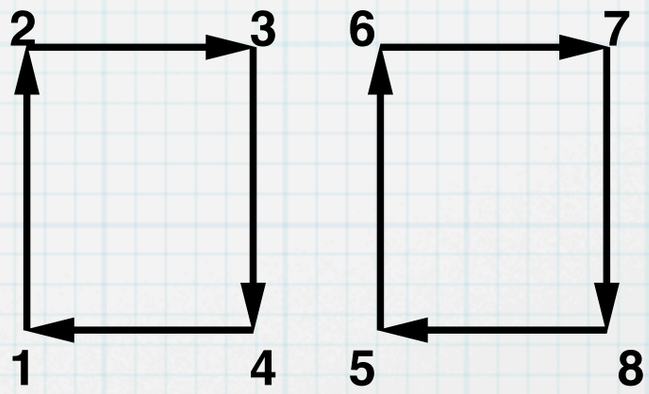
(b)



(c)



(a)



(b)

Poligoni

- Vincoli per i poligoni
  - Convessi
  - I bordi non devono incrociarsi
  - Planari
    - Ordinati
- Triangoli sono sempre validi
- Poligoni hanno due faccie: Front and Back
  - CCW - Punti in senso anti-orario: Front
  - CW - Punti in senso orario: Back

* Modalità disegno poligono

void **glPolygonMode**(GLenum *face*, GLenum *mode*)
face: Faccia da disegnare
GL_FRONT, **GL_BACK**, **GL_FRONT_AND_BACK**
mode: Modalità di disegno
GL_POINT, **GL_LINE**, **GL_FILL**

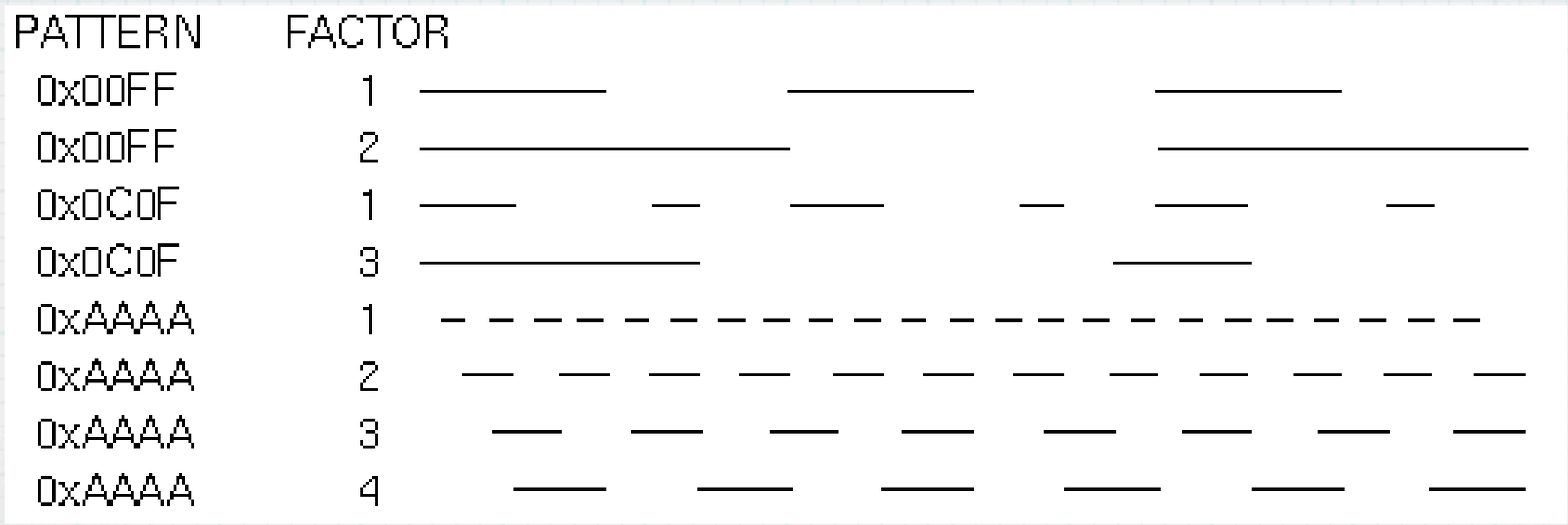
void **glFrontFace**(GLenum *mode*)
mode: Modalità orientamento faccia
GL_CCW Senso anti-orario è Front
GL_CW Senso orario è Front

void **glCullFace**(GLenum *mode*)
mode: Test visibilità faccie viewpoint

GL_FRONT, GL_BACK e **GL_FRONT_AND_BACK**

glEnable(GL_CULL_FACE)

- **glPointSize** - Diametro in punti di un punto al momerto del disegno
  void **glPointSize**(GLfloat *size*)
- **glLineWidth** - Diametro in punti di una linea al momerto del disegno
  void **glLineWidth**(GLfloat *width*)
- **glLineStipple** - Tratteggio linea
  void **glLineStipple**(GLint *factor*, GLushort *pattern*)
  pattern: Un intero a 16bit la cui rappresentazione in binario è il pattern per il tratteggio
  factor: Ogni quanti punti (di disegno) viene usato un nuovo bit di pattern



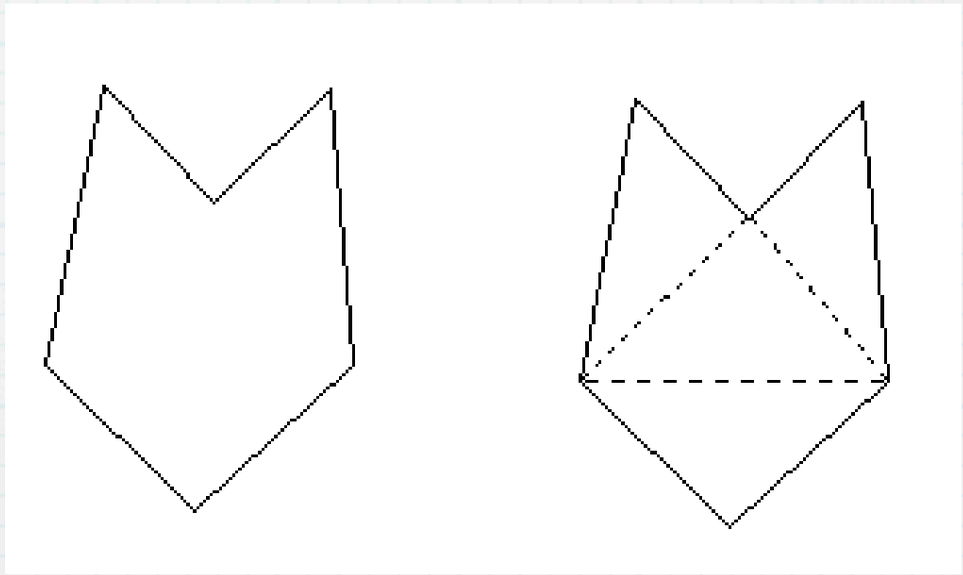
* Tratteggio Poligono

void **glPolygonStipple**(const GLubyte **mask*)
mask: Bitmap 32x32 (array da 8x32 byte oppure 1x32 interi a 32 bit) da utilizzare come tratteggio

```
glEnable(GL_POLYGON_STIPPLE);
```

* Bordi interni

void **glEdgeFlag**(GLboolean *flag*)



```
void model() {
  int i;
  GLubyte polystipple[32][8];
  GLubyte linestipple[8];

  /* Set drawing mode */
  for(i=0;i<32;i++)
    memset(polystipple[i] , (GLubyte) i%2?0xAA:0x55 , 8);
  memset(linestipple, 0xB6, 8);
  glPolygonMode(GL_FRONT,GL_FILL);
  glEnable(GL_POLYGON_STIPPLE);
  glPolygonStipple (polystipple);
  glColor3f(0.8,0.8,0.8);
  glPointSize(3);
  glLineWidth(2.0);

  /* First draw as is */
   glBegin(primtype);
  mkpoints();
  glEnd();
```

```
/* Current model */
void mkpoints() {
 int i;

 for (i=0;i < 12; i ++) {
    glVertex2f(cos(M_PI/6*i),
               sin(M_PI/6*i));
    }
}
```

```
/* Change Drawing mode */
glDisable(GL_POLYGON_STIPPLE);
glEnable(GL_LINE_STIPPLE);
glLineStipple(1,linestipple);
glDisable(GL_DEPTH_TEST);
/* Drawn lines, draw points too */
if(primtype>=GL_LINES && primtype < GL_TRIANGLES) {
  glBegin(GL_POINTS);
  mkpoints();
  glEnd();
}
/* Drawn poligons, draw points & lines */
if(primtype>=GL_TRIANGLES) {
  glPolygonMode(GL_FRONT_AND_BACK,GL_POINT);
  glColor3f(1.0,1.0,1.0);
      glBegin(primtype);
  mkpoints();
  glEnd();
  glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
  glColor3f(1.0,1.0,1.0);
      glBegin(primtype);
  mkpoints();
  glEnd();
 }
 /* Disable stipple */
 glDisable(GL_LINE_STIPPLE);
 glEnable(GL_DEPTH_TEST);
}
```

Tra GLBegin e GLEnd

glVertex*()	set vertex coordinates
glColor*()	set current color
glIndex*()	set current color index
glNormal*()	set normal vector coordinates
glTexCoord*()	set texture coordinates
glEdgeFlag*()	control drawing of edges
glMaterial*()	set material properties
glArrayElement()	extract vertex array data
glEvalCoord*(), glEvalPoint*()	generate coordinates
glCallList(), glCallLists()	execute display list(s)

* glVertex è ancora importante

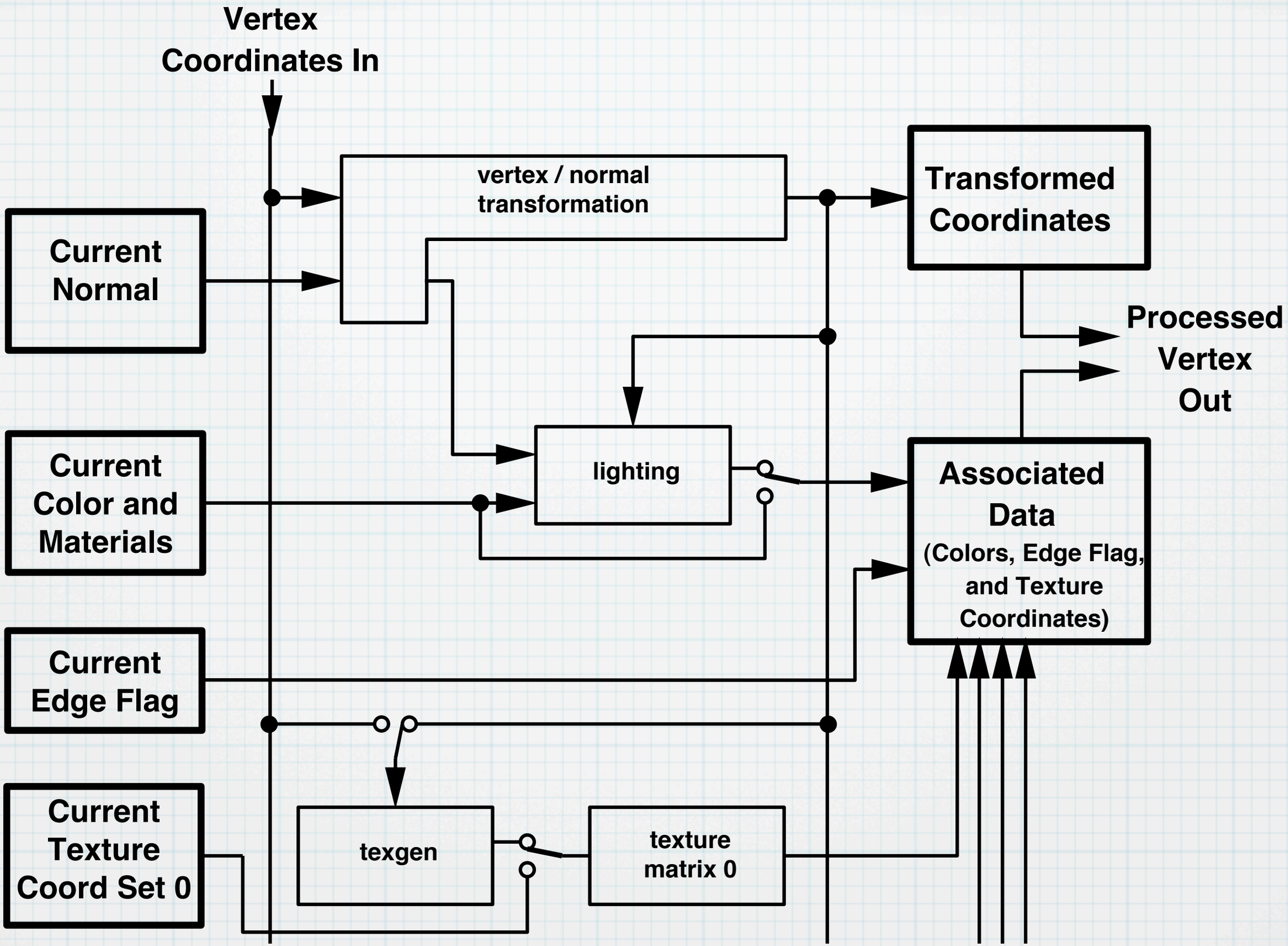
```
glBegin(GL_POINTS);
   glColor3f(0.0, 1.0, 0.0);                    /* green */
   glColor3f(1.0, 0.0, 0.0);                    /* red */
   glVertex(...);
   glColor3f(1.0, 1.0, 0.0);                    /* yellow */
   glColor3f(0.0, 0.0, 1.0);                    /* blue */
   glVertex(...);
   glVertex(...);
glEnd();
```

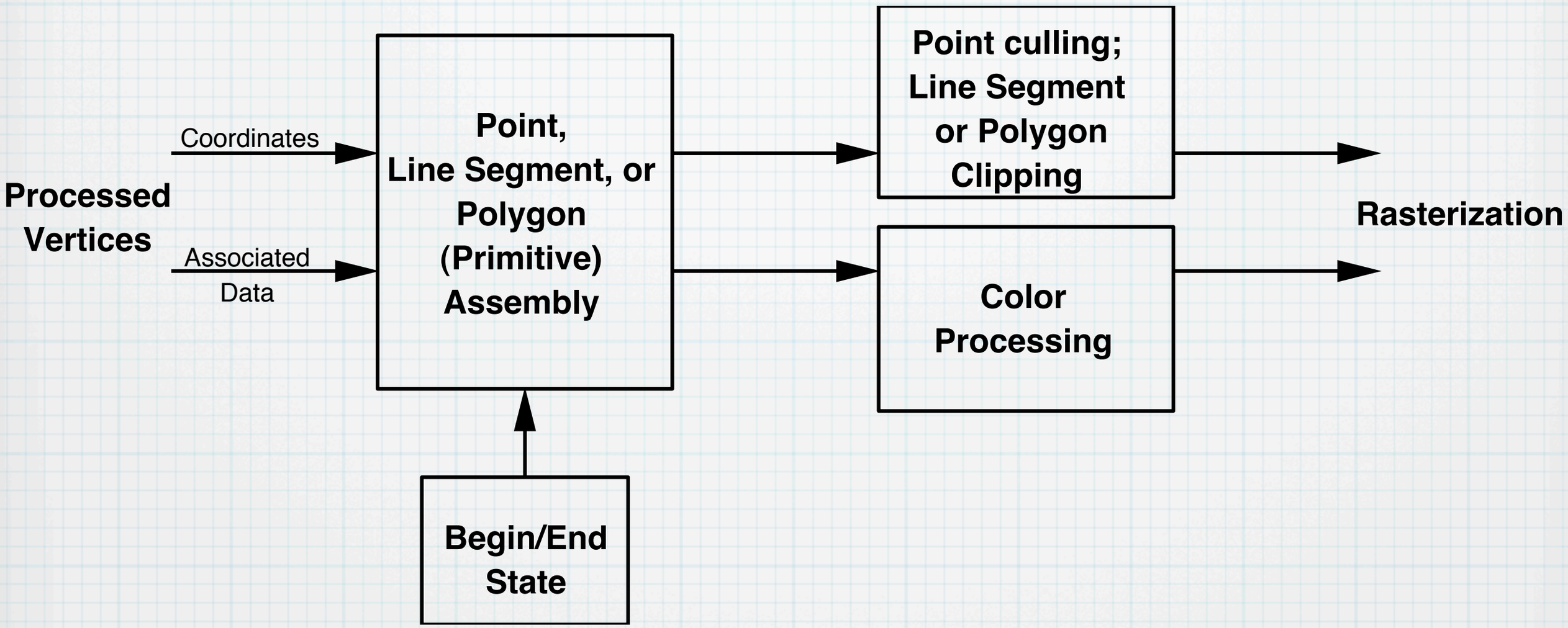
* L'ordine è quello di esecuzione

```
GLint circle_points = 100;
glBegin(GL_LINE_LOOP);
for (i = 0; i < circle_points; i++) {
   angle = 2*PI*i/circle_points;
   glVertex2f(cos(angle), sin(angle));
}
glEnd();
```

Colore per Vertice o per faccia

glShadeModel - select flat or smooth shading
void **glShadeModel**(GLenum *mode*)
*mode*_ Specifies a symbolic value representing a shading technique. Accepted values are **GL_FLAT** and **GL_SMOOTH**. The default is **GL_SMOOTH**.


Vertex
Coordinates In
vertex / normal
transformation
Transformed
Coordinates
Current
Normal
Processed
Vertex
Out
lighting
Current
Color and
Materials
Associated
Data
(Colors, Edge Flag,
and Texture
Coordinates)
Current
Edge Flag
Current
Texture
Coord Set 0
texgen
texture
matrix 0


Processed Vertices
Coordinates
Associated Data
Point, Line Segment, or Polygon (Primitive) Assembly
Begin/End State
Point culling; Line Segment or Polygon Clipping
Color Processing
Rasterization

Vedere esempio Shapes in

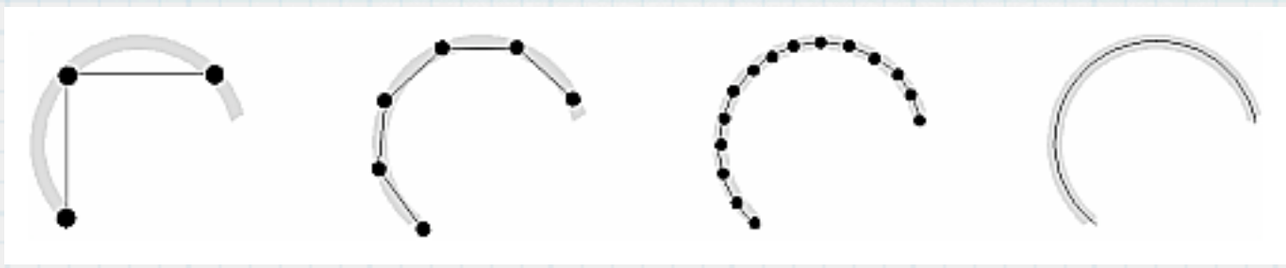
http://www.xmission.com/~nate/tutors.html

Approssimazione Superfici

* Approssimare superficie con poligoni
* Un poligono ha solo una normale
* Normali -> illuminazione

* Normale per ogni vertice

void **glNormal3**{bsidf}(*TYPEnx*, *TYPEny*, *TYPEnz*);
void **glNormal3**{bsidf}**v**(const **TYPE** *v);



```
int shownormals=0;

void rotateprofile(GLfloat profile[][2], int n_profile, int n) {
  int i,j,k;
  GLfloat angle0,angle1;
  GLfloat step=2.0 * M_PI / n;
  GLfloat profileN[2];
  GLfloat d;

  /* For each segment */
  for(i=0;i<n_profile-1;i++) {
    /* For each slice */
    for(j=0;j<n;j++) {
      /* Calculate angles */
      angle0=j * step;
      angle1=((j+1) % n) * step;

      /* Calculate normal of profile */
      profileN[0] = profile[i][1] -  profile[i+1][1];
      profileN[1] = profile[i+1][0] -  profile[i][0];
      d=sqrt(profileN[0] * profileN[0] + profileN[1] * profileN[1]);
      profileN[0] /=d;
      profileN[1] /=d;
```

```
if(shownormals) {
  glDisable(GL_LIGHTING);

  glBegin(GL_LINES);
  glVertex3f(profile[i][0] * cos(angle0),
      profile[i][1] ,
      profile[i][0] * sin(angle0));
  glVertex3f((profile[i][0] + profileN[0])  * cos(angle0),
      profile[i][1] + profileN[1],
      (profile[i][0] + profileN[0])  * sin(angle0) );
  glEnd();

      glBegin(GL_LINES);
      glVertex3f(profile[i+1][0] * cos(angle1),
          profile[i+1][1] ,
          profile[i+1][0] * sin(angle1));
      glVertex3f((profile[i+1][0] + profileN[0])  * cos(angle1),
          profile[i+1][1] + profileN[1],
          (profile[i+1][0] + profileN[0])  * sin(angle1) );
     glEnd();

      glEnable(GL_LIGHTING);
}
```

```
/* Draw face, updating the angle */
glBegin(GL_POLYGON);
glPolygonMode(GL_FRONT,GL_FILL);

/* first point of profile projected on first angle */
glNormal3f(profileN[0] * cos(angle0),
           profileN[1],
           profileN[0] * sin(angle0));
glVertex3f( profile[i][0] * cos(angle0),
            profile[i][1],
            profile[i][0] * sin(angle0) );

/* second point of profile projected on first angle */
glNormal3f(profileN[0] * cos(angle0),
           profileN[1],
           profileN[0] * sin(angle0));
glVertex3f( profile[i+1][0] * cos(angle0),
            profile[i+1][1],
            profile[i+1][0] * sin(angle0) );
```

```
            /* second point of profile projected on second angle */
            glNormal3f(profileN[0] * cos(angle1),
                       profileN[1],
                       profileN[0] * sin(angle1));
            glVertex3f( profile[i+1][0] * cos(angle1),
                        profile[i+1][1],
                        profile[i+1][0] * sin(angle1));

            /* first point of profile projected on second angle */
            glNormal3f(profileN[0] *  cos(angle1),
                       profileN[1],
                                profileN[0] *  sin(angle1));
                    glVertex3f( profile[i][0] * cos(angle1),
                                profile[i][1],
                                profile[i][0] * sin(angle1));

                    glEnd();

                }
            }
        }
```

```
void model3 () {
  GLfloat light_pos[]={ 3.0, 3.0, 3.0 };
  GLfloat profile[][2] = { {0.1, 1    },
                           {0.8, 0.5  },
                           {1  , 0    },
                           {0.8, -0.5 },
                           {1,   -1   },
                           {0.8, -1   },
                           {0.1,  0   } };
  int n_profile=7;

  /* Fiat Lux */
  glEnable(GL_LIGHTING);
  glEnable(GL_NORMALIZE);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_POSITION, light_pos );

  /* Make profile */
  glColor3f(0.8,0.8,0.8);
  rotateprofile(profile,n_profile,8);

  /* Turn off lighting */
  glDisable(GL_LIGHTING);
}
```

Vertex Arrays (>1.1)

* Permettono di utilizzare interi array nella memoria per poterli poi riferire solo per indice
* Attivare/Disattivare

void **glEnableClientState**(GLenum *array*)

void **glDisableClientState**(GLenum *array*)

array: GL_VERTEX_ARRAY,
GL_COLOR_ARRAY,
GL_NORMAL_ARRAY
GL_TEXTURE_COORD_ARRAY
GL_EDGE_FLAG_ARRAY
...

* Specificare Array

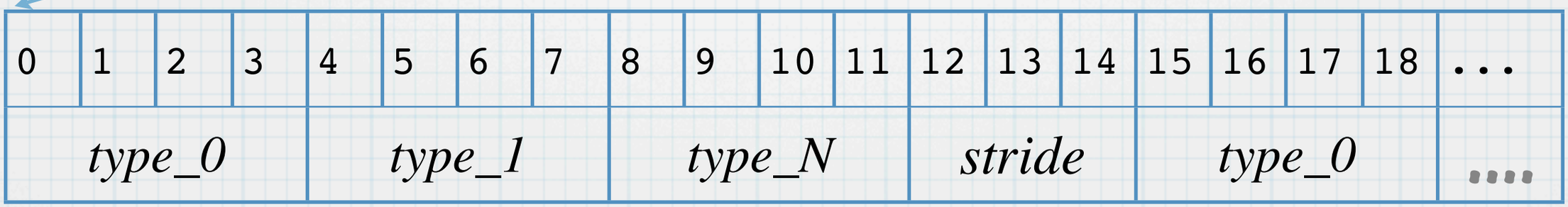
`void` **glVertexPointer**(`int` *N*, `enum` *type*, `sizei` *stride*, `void` **pointer*);

pointer: `Puntatore ad area di memoria contenente i dati`

N: `Numero di coordinate per vertice (2,3,4)`

type: `Tipo elemento base del vertice`
`(GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE ...)`

stride: `Spazio in byte tra un vertice ed un altro`



* Inoltre:

`void` **glEdgeFlagPointer**(`sizei` *stride*, `void` **pointer*);
`void` **glTexCoordPointer**(`int` *size*, `enum` *type*, `sizei` *stride*, `void` **pointer*);
`void` **glColorPointer**(`int` *size*, `enum` *type*, `sizei` *stride*, `void` **pointer*);
`void` **glIndexPointer**(`enum` *type*, `sizei` *stride*, `void` **pointer*);
`void` **glNormalPointer**(`enum` *type*, `sizei` *stride*, `void` **pointer*);
`void` **glVertexPointer**(`int` *size*, `enum` *type*, `sizei` *stride*, `void` **pointer*);

Command	Sizes	Types
VertexPointer	2,3,4	`short`, `int`, `float`, `double`
NormalPointer	3	`byte`, `short`, `int`, `float`, `double`
ColorPointer	3,4	`byte`, `ubyte`, `short`, `ushort`, `int`, `uint`, `float`, `double`
IndexPointer	1	`ubyte`, `short`, `int`, `float`, `double`
TexCoordPointer	1,2,3,4	`short`, `int`, `float`, `double`
EdgeFlagPointer	1	`boolean`

Table 2.4: Vertex array sizes (values per vertex) and data types.

Selezionare elementi in tutti gli array attivi:

`void` **glArrayElement**(`int` *i*);
Seleziona i-esimo elemento da tutti gli array attivi

`void` **glDrawArrays**(`enum` *mode*, `int` *first*, `sizei` *count*);

```
Begin(mode);
for (i=0; i < count ; i++)
    ArrayElement(i);
End();
```

`void` **glDrawElements**(`enum` *mode*, `sizei` *count*, `enum` *type*, `void` **indices*);

```
Begin(mode);
for (i=0; i < count ; i++)
    ArrayElement(indices[i]);
End();
```

`void` **glDrawRangeElements**(`enum` *mode*, `uint` *start*, `uint` *end*, `sizei` *count*, `enum` *type*, `void` **indices*);

`void` **glMultiDrawElements**(`enum` *mode*, `sizei` **count*, `enum` *type*, `void` ***indices*, `sizei` *primcount*);

`void` **glMultiDrawArrays**(`enum` *mode*, `int` **first*, `sizei` **count*, `sizei` *primcount*);

* Array mischiati

`void` **glInterleavedArrays**(`enum` *format*, `sizei` *stride*, `void` **pointer*);

format	e_t	e_c	e_n	s_t	s_c	s_v	t_c
V2F	*False*	*False*	*False*			2	
V3F	*False*	*False*	*False*			3	
C4UB_V2F	*False*	*True*	*False*		4	2	UNSIGNED_BYTE
C4UB_V3F	*False*	*True*	*False*		4	3	UNSIGNED_BYTE
C3F_V3F	*False*	*True*	*False*		3	3	FLOAT
N3F_V3F	*False*	*False*	*True*			3	
C4F_N3F_V3F	*False*	*True*	*True*		4	3	FLOAT
T2F_V3F	*True*	*False*	*False*	2		3	
T4F_V4F	*True*	*False*	*False*	4		4	
T2F_C4UB_V3F	*True*	*True*	*False*	2	4	3	UNSIGNED_BYTE
T2F_C3F_V3F	*True*	*True*	*False*	2	3	3	FLOAT
T2F_N3F_V3F	*True*	*False*	*True*	2		3	
T2F_C4F_N3F_V3F	*True*	*True*	*True*	2	4	3	FLOAT
T4F_C4F_N3F_V4F	*True*	*True*	*True*	4	4	4	FLOAT

format	p_c	p_n	p_v	s
V2F			0	$2f$
V3F			0	$3f$
C4UB_V2F	0		c	$c+2f$
C4UB_V3F	0		c	$c+3f$
C3F_V3F	0		$3f$	$6f$
N3F_V3F		0	$3f$	$6f$
C4F_N3F_V3F	0	$4f$	$7f$	$10f$
T2F_V3F			$2f$	$5f$
T4F_V4F			$4f$	$8f$
T2F_C4UB_V3F	$2f$		$c+2f$	$c+5f$
T2F_C3F_V3F	$2f$		$5f$	$8f$
T2F_N3F_V3F		$2f$	$5f$	$8f$
T2F_C4F_N3F_V3F	$2f$	$6f$	$9f$	$12f$
T4F_C4F_N3F_V4F	$4f$	$8f$	$11f$	$15f$

```
void model2() {
  GLfloat cpoints[8][3] = { { 0.0, 0.0, 0.0 },
                            { 1.0, 0.0, 0.0 },
                            { 0.0, 1.0, 0.0 },
                            { 1.0, 1.0, 0.0 },
                            { 0.0, 0.0, 1.0 },
                            { 1.0, 0.0, 1.0 },
                            { 0.0, 1.0, 1.0 },
                            { 1.0, 1.0, 1.0 }};
  GLubyte indices[6][4] = { { 0, 2, 3, 1 }, //back
                            { 4, 5, 7, 6 }, //front
                            { 0, 4, 6, 2 }, //left
                            { 1, 5, 7, 3 }, //right
                            { 0, 1, 5, 4 }, //rear
                            { 2, 6, 7, 3 } };

  int i,j;
  glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
```

```
/* For each face and point */
  glBegin(GL_QUADS);
  for(i=0;i<6;i++) {
    for(j=0;j<4;j++) {
      glColor3fv(cpoints[indices[i][j]]);
      glVertex3fv(cpoints[indices[i][j]]);
       }
    }
  glEnd();

  /* Oppure */
  glEnableClientState(GL_VERTEX_ARRAY);
  glEnableClientState(GL_COLOR_ARRAY);
  glVertexPointer( 3, GL_FLOAT, 0, cpoints);
  glColorPointer( 3, GL_FLOAT, 0, cpoints);
  glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, indices );
  glDisableClientState( GL_VERTEX_ARRAY );
  glDisableClientState( GL_COLOR_ARRAY );ù
}
```