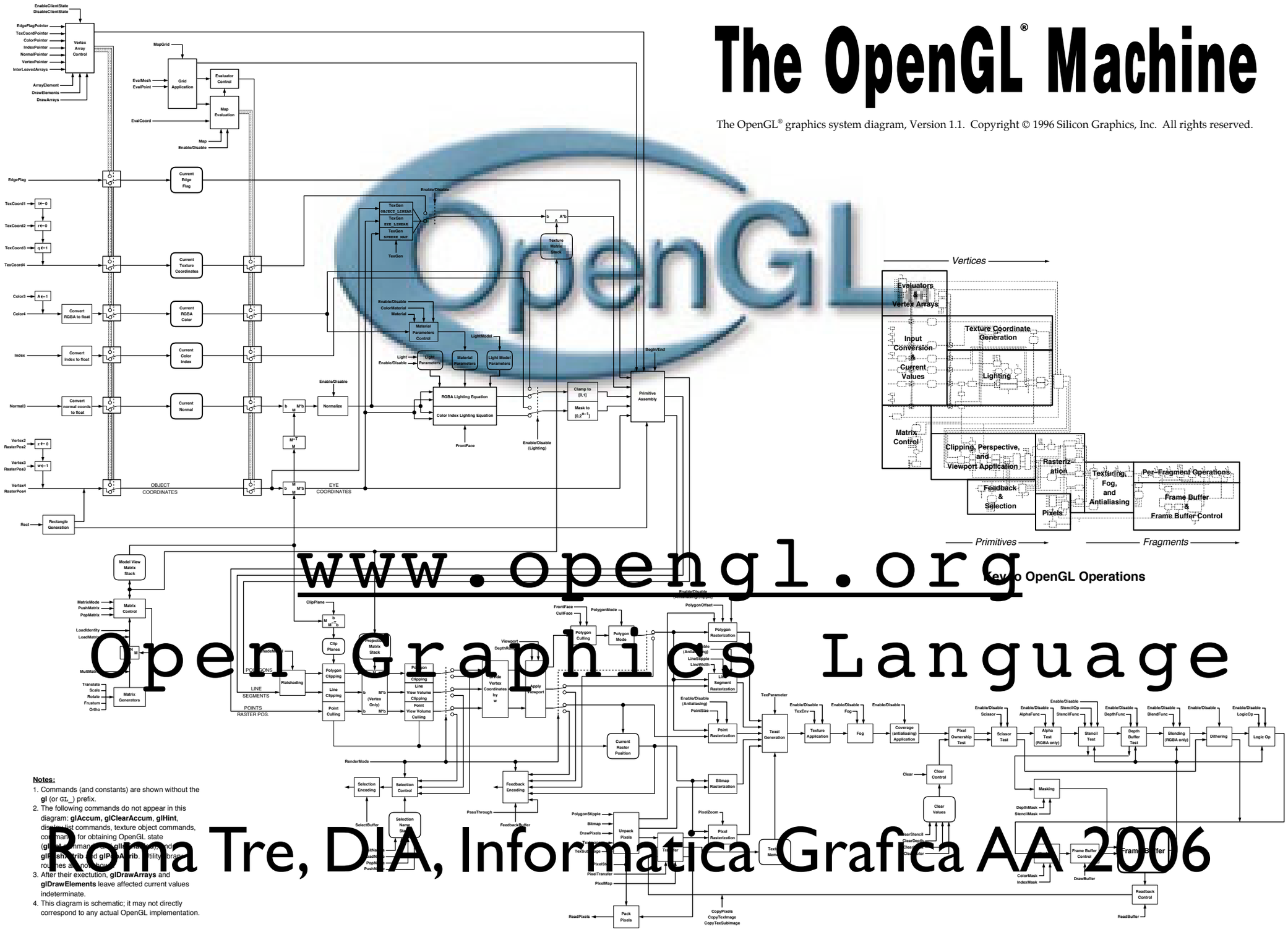
The OpenGL® Machine



Notes:

1. Commands (and constants) are shown without the **gl** (or GL_) prefix.
2. The following commands do not appear in this diagram: **glAccum, glClearAccum**, **glHint**, display list commands, texture object commands, commands for obtaining OpenGL state (**glGet** commands, **glIsEnabled**, and **glPushAttrib** and **glPopAttrib**), utility library routines and toolkit routines.
3. After their exectution, **glDrawArrays** and **glDrawElements** leave affected current values indeterminate.
4. This diagram is schematic; it may not directly correspond to any actual OpenGL implementation.

Oggi

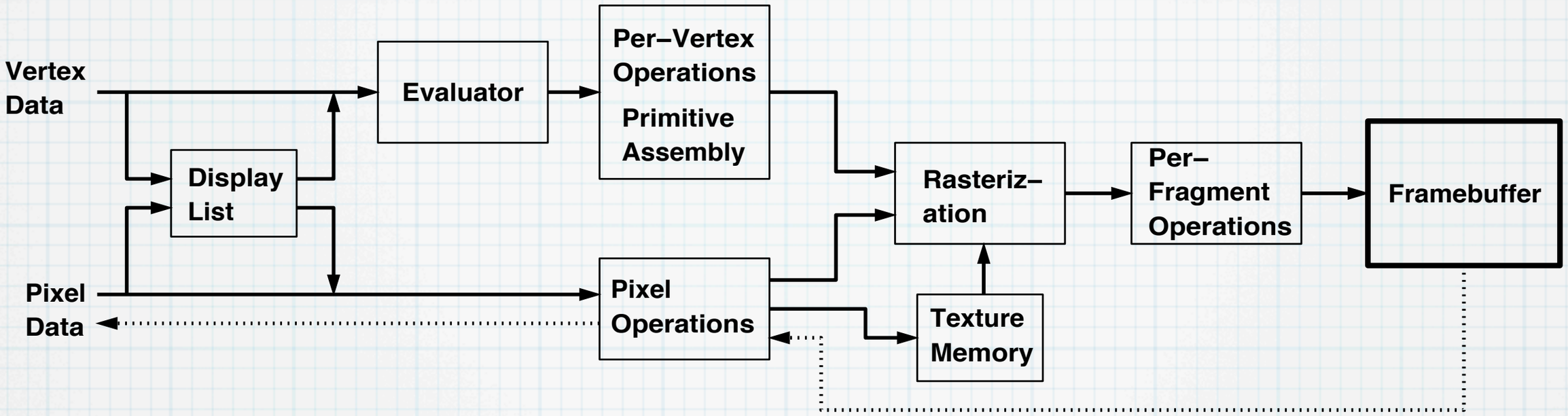
* Cos'è OpenGL
  * Modello
  * Librerie Associate
* Primo Programma
* Trasformazioni
* Secondo Programma ...

Prossime Lezioni

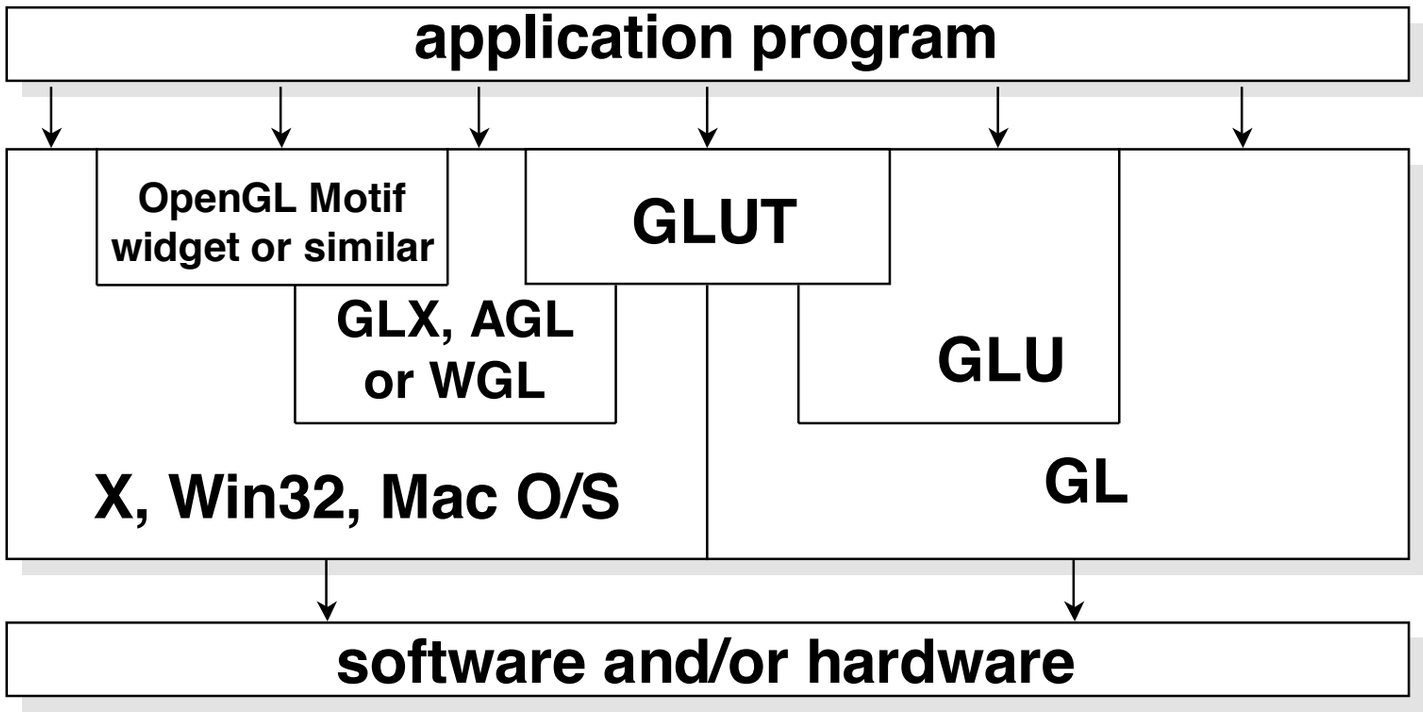
* 08/03 : Primitive OpenGL
* 15/03 : Modelli di vista
* 22/03 : Texture mapping
* 29/03 : Luci e materiali
* 05/04 : Curve e Superfici

Cos'è OpenGL?

* API per il rendering grafico
  (API: Application Programming Interface)
* Immagini di alta qualità costruite da primitive d'immagine e geometriche
* Indipendente da Hardware, Sistema Operativo, e Sistema Grafico
* What You See is What You Wanted
  Quello che ottieni è quello che hai detto

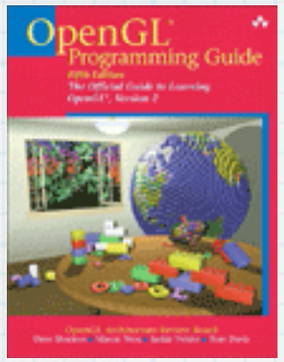


* Modello ad automa (hardware astratto)
(può essere implementato in hardware vero)
  * Comandi
  (Disegna Qualcosa)
  Primitive Geometriche o d'immagine
  Texturing
  * Stato
  (Come OpenGL disegna)
  Colore, Materiale, Luci, etc

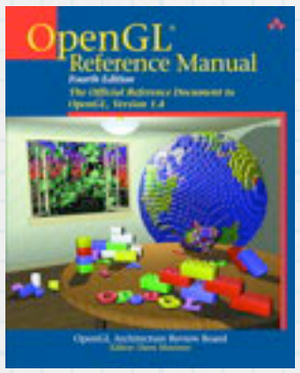


* GLU (Open GL Utility Library)
  Funzioni composte e d'utilità
  NURBS, tesselators, Viewpoint ...
* AGL, GLX, WGL
  "Ponte con sistema operativo e grafico"
* GLUT (Open GL Utility Toolkit)
  (AGL, GLX, WGL indipendente da S.O)

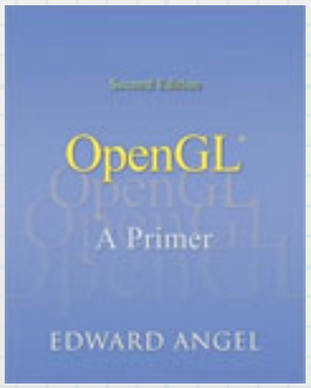
Libri consigliabili



* **The OpenGL Programming Guide - The Redbook**
  The OpenGL Programming Guide 4th Edition The Official Guide to Learning OpenGL Version 1.4



* **The OpenGL Reference Manual - The Bluebook**
  The OpenGL Reference Manual 4th Edition The Official Reference Document to OpenGL, Version 1.4



* **OpenGL : A Primer (2nd Edition)**

Link utili

* www.opengl.org
  Trovate tutto lì
* Tutorial GLUT:
  http://www.lighthouse3d.com/opengl/glut/index.php?2
* Programmi Tutorial Sofisticati
  http://www.xmission.com/~nate/tutors.html
* WGL: http://nehe.gamedev.net/

Compilazione (GCC) singolo file

```
gcc <IncludeDir> <LibDir> <Lib> <Opt>\
    file.c -o file

<IncludeDir> : Non default include directory
               -I <dir1> -I <dir2> ...
               Eg. -I /usr/GLUT/include
                   -I ../include
<LibDir> : Non default libraries directory
           -L <dir1> -L <dir2> ...
           Eg. -L /usr/GLUT/lib
<Lib> : Non default libraries
        -lfile1 -lfile2 ...
        Eg: -lm -lXext -lX11
            -lGL -lGLU -lglut
    (Apple) -framework OpenGL -framework GLUT
<Opt> : Opzioni varie ed eventuali
        Eg: -g -O2 -DPROVA -Wall
```

Compilazione (GCC) singolo file

```
gcc <IncludeDir> <LibDir> <Lib> <Opt>\
    file.c -o file

<IncludeDir> : Non default include directory
               -I <dir1> -I <dir2> ...
               Eg. -I /usr/GLUT/include
```

```
Ubuntu Linux:
     gcc -g -lGL -lGLU -lglut main.c -o main

MacOSX:
     gcc -g -framework OpenGL -framework GLUT \
        main.c -o main
```

```
               -lGL -lGLU -lglut
          (Apple) -framework OpenGL -framework GLUT
       <Opt> : Opzioni varie ed eventuali
               Eg: -g -O2 -DPROVA -Wall
```

Compilazione (GCC) file multipli

```
Per ogni file sorgente:
gcc -c <IncludeDir> <CompOpt> file1.c
gcc -c <IncludeDir> <CompOpt> file2.c
...
-> file1.o file2.o ...
```

```
Per l'eseguibile:
gcc <LibDir> <Lib> <LinkOpt>\
    file1.o file2.o ... -o exefile
```

```
Conviene fare Makefile:
Vedere Esempi e Tutors
```

Comp

Per c
gcc -
gcc -
...
-> f

Per
gcc

Convi
Vedei

```
HEADERS  = glm.h materials.h
SOURCES  = fog.c glm.c lightmaterial.c lightposition.c projection.c shapes.c \
           texture.c transformation.c
DEPENDS  = $(SOURCES:.c=.d)
OBJECTS  = $(SOURCES:.c=.o)
PROGRAMS = fog lightmaterial lightposition projection shapes texture \
           transformation

ifdef DEBUG
OPTFLAGS = -g
else
OPTFLAGS = -O3 -s
endif

CC      = gcc
CFLAGS  = -Wall -Wno-format $(OPTFLAGS)
LDFLAGS = -lgl -lglut

all: $(PROGRAMS)

$(PROGRAMS):
      $(CC) $(CFLAGS) $^ $(LDFLAGS) -o $@

define PROGRAM_template
$(1): $(addsuffix .o,$(1)) glm.o
endef
$(foreach t,$(PROGRAMS),$(eval $(call PROGRAM_template,$(t))))

clean:
      $(RM) $(OBJECTS) $(DEPENDS)
      $(RM) $(PROGRAMS)

.PHONY: all clean

%.o: %.c
      $(CC) -c $(CFLAGS) $< -o $@

%.d: %.c
      $(CC) -MM $(CFLAGS) $< > $@


ifneq ($(MAKECMDGOALS),clean)
-include $(DEPENDS)
endif
```

Progetto Visual Studio

http://csf1 1.acs.uwosh.edu/cs371/visualstudio/
http://www.css.tayloru.edu/~btoll/resources/graphics/opengl/xp/visualc.html

0. Install GLUT

1. Create an *empty* C++ Win32 Console Project.

2. Add at least one source file to your project.

3A. Open the project's `Property Pages` dialog and select `All Configurations` from the `Configuration:` list box.

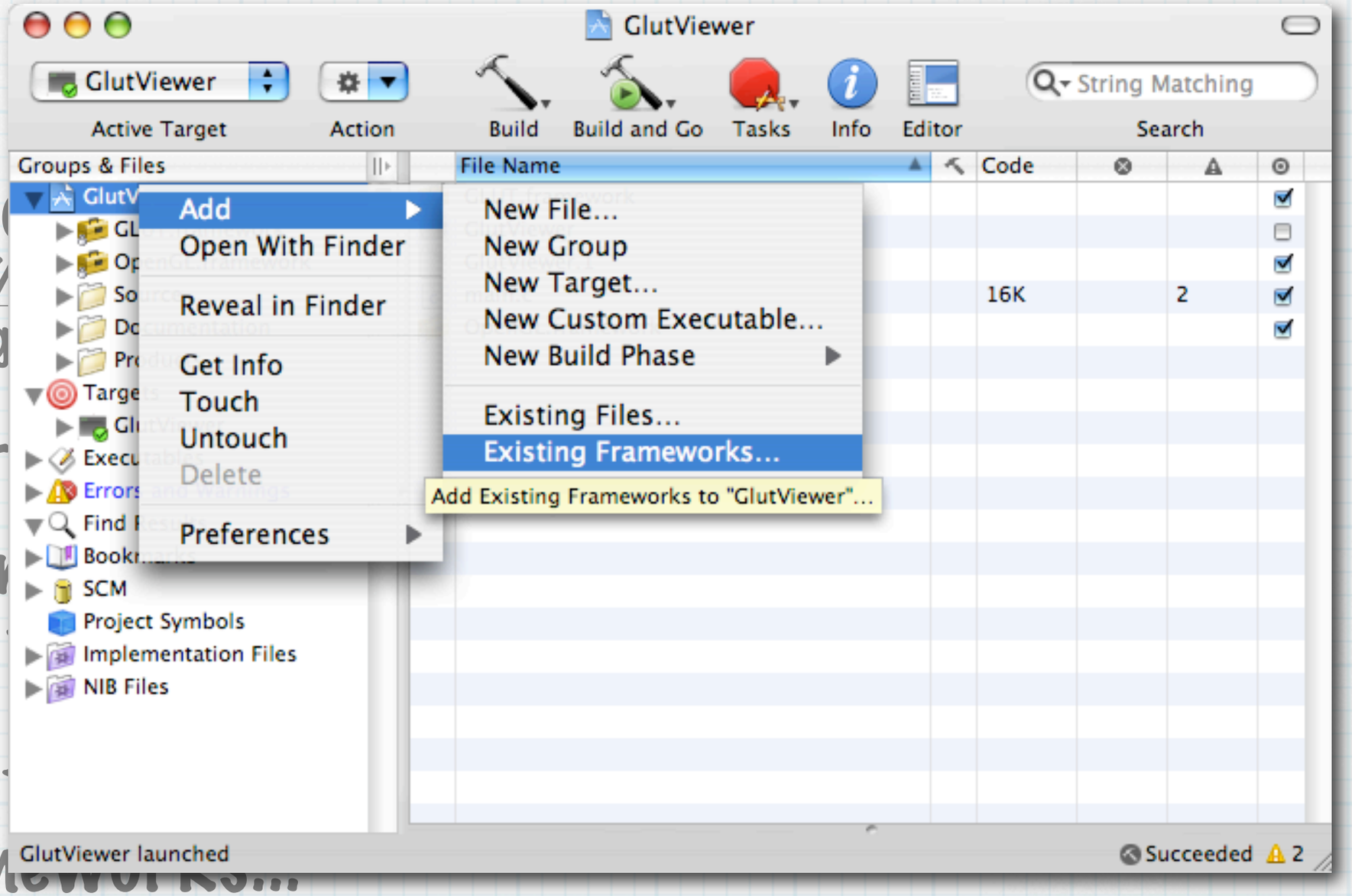
4A..NET: In the `Linker → Input` subtree, add the following to the `Additional Dependencies` text box:
`opengl32.lib glu32.lib glut32.lib`

4B.VS7: Select "Settings..." from the "Project" menu
Select the "Link" tab
Scroll to the end of the "Object/library models:" text box and add the following additional libraries: glut32.lib glut32.lib opengl32.lib

5OPT. To disable the console window, go to the `Linker → Command Line` subtree and add the following code to the `Additional Options:` text box:
`/SUBSYSTEM:WINDOWS /ENTRY:mainCRTStartup`

* Progetto XCODE
  http://razorblade.or.id/2005/10/using-xcode-project-to-do-manage-your-opengl-applications/
* File -> New Project....
* Command Line Utility -> Standard Tool (C++ Tool)
* Right click on <project> -> Add -> Existing Frameworks...
  /System/Library/Frameworks
  OpenGL.framework
  GLUT.framework.

* Prog
http://
manag
* File -
* Comm
(C++
* Righ
Fram
/System/Library/Frameworks
OpenGL.framework
GLUT.framework.



Primo Programma

```
/* Include GL, GLU & GLUT */
#ifdef WIN32
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#endif

#ifdef __APPLE__
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#include <GLUT/glut.h>
#endif

#ifdef linux
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#endif

/* Include some standard stuff */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```c
int main (int argc, const char** argv) {

    /* Init Glut */
    glutInit(&argc, argv);

    /* Initial parameters */
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_SINGLE);
    glutInitWindowSize(800, 600);
    glutInitWindowPosition(0, 0);

    /* Create main Window */
    glutCreateWindow("Axes");

    /* Set Callbacks */
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);

    /* Glut Main Loop */
    glutMainLoop();

    /* Never reached, keep compiler happy */
    return 0;
}
```

```c
int main (int argc, const char** argv) {

   /* Init Glut */
   glutInit(&argc, argv);

   /* Initial parameters */
   glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_SINGLE);
   glutInitWindowSize(800, 
   glutInitWindowPosition(0

   /* Create main Window */
   glutCreateWindow("Axes");
```

```c
void glutInit(int *argcp, char **argv);
```

Inizializza la libreria ed la sua interazione con il sistema grafico. Se fallisce tutto il programma verrà terminato.

```c
   /* Set Callbacks */
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);

   /* Glut Main Loop */
   glutMainLoop();

   /* Never reached, keep compiler happy */
   return 0;
}
```

```c
int main (int argc, const char** argv) {

   /* Init Glut */
   glutInit(&argc, argv);

   /* Initial parameters */
   glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_SINGLE);
   glutInitWindowSize(800, 600);
   glutInitWindowPosition(0, 0);

   /* Create main Window */
```

```
void glutInitDisplayMode(unsigned int mode);
```

`mode:` OR di FLAG. Comuni:

`GLUT_RGBA`	Modalità colore pieno (in genere 32 bit (8: Red 8:Green 8:Blue 8:Alfa)
`GLUT_INDEX`	Modalità colore indicizzato (o con palette). Ormai poco comune
`GLUT_SINGLE`	Singolo buffer immagine per finestra (Statico)
`GLUT_DOUBLE`	Doppio buffer (Dinamico)
`GLUT_DEPTH`	Coordinata Z nel buffer (3D + HFR)
....	(Vedi Manuale)

```c
   /* Never reached, keep compiler happy */
   return 0;
}
```

```c
int main (int argc, const char** argv) {

   /* Init Glut */
   glutInit(&argc, argv);

   /* Initial parameters */
   glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_SINGLE);
   glutInitWindowSize(800, 600);
   glutInitWindowPosition(0, 0);

   /* Create main Window */
   glutCreateWindow("Axes");
```

```c
void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y);
```

```c
   /* Glut Main Loop */
   glutMainLoop();

   /* Never reached, keep compiler happy */
   return 0;
}
```

Vedi anche:

```
int glutCreateSubWindow(int win,
                        int x, int y, int width, int height);
void glutDestroyWindow(int win);
void glutPositionWindow(int x, int y);
void glutReshapeWindow(int width, int height);
void glutShowWindow(void);
void glutHideWindow(void);
void glutIconifyWindow(void);
void glutSetWindowTitle(char *name);
void glutSetIconTitle(char *name);
```

```
/* Create main window */
glutCreateWindow("Axes");
```

```
int glutCreateWindow(char *name);
```

`glutCreateWindow` crea una nuova finestra top level.

Ritorna un identificatore di finestra per identificarlo in caso si hanno più finestre.

Esiste il concetto di finestra corrente. Tutte le funzioni GLUT operano su questa.

Per cambiare finestra corrente:

```
void glutSetWindow(int win);
int glutGetWindow(void);
```

Callbacks:
GLUT è basato su un modello ad eventi.
Esiste una coda di eventi.
Ogni evento è gestito da una funzione a seconda del tipo. Ciclo gestione eventi.
Un programma può registrare le proprie funzioni per una gestione specifica.

Il ciclo gestione eventi è attivato da:

```
void glutMainLoop(void);
```

Tale funzione deve essere chiamata solo una volta nel programma GLUT.
Una volta chiamato non ritornerà mai. Tuttavia chiamerà le funzioni di callback registrate.

```
    /* Create main Window */
    glutCreateWindow("Axes");

    /* Set Callbacks */
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);

    /* Glut Main Loop */
    glutMainLoop();

    /* Never reached, keep compiler happy */
    return 0;
}
```

```
void glutDisplayFunc(void (*func)(void));
```

Installa `void func()` come callback di visualizzazione per la finestra corrnete.

Quando GLUT determina che la finestra deve essere (ri)-disegnate `func` è chiamata.

Si può forzare GLUT a ridisegnare una finestra da un'altro callback con `glutPostRedisplay.`

Questo callback deve essere fornito a GLUT.

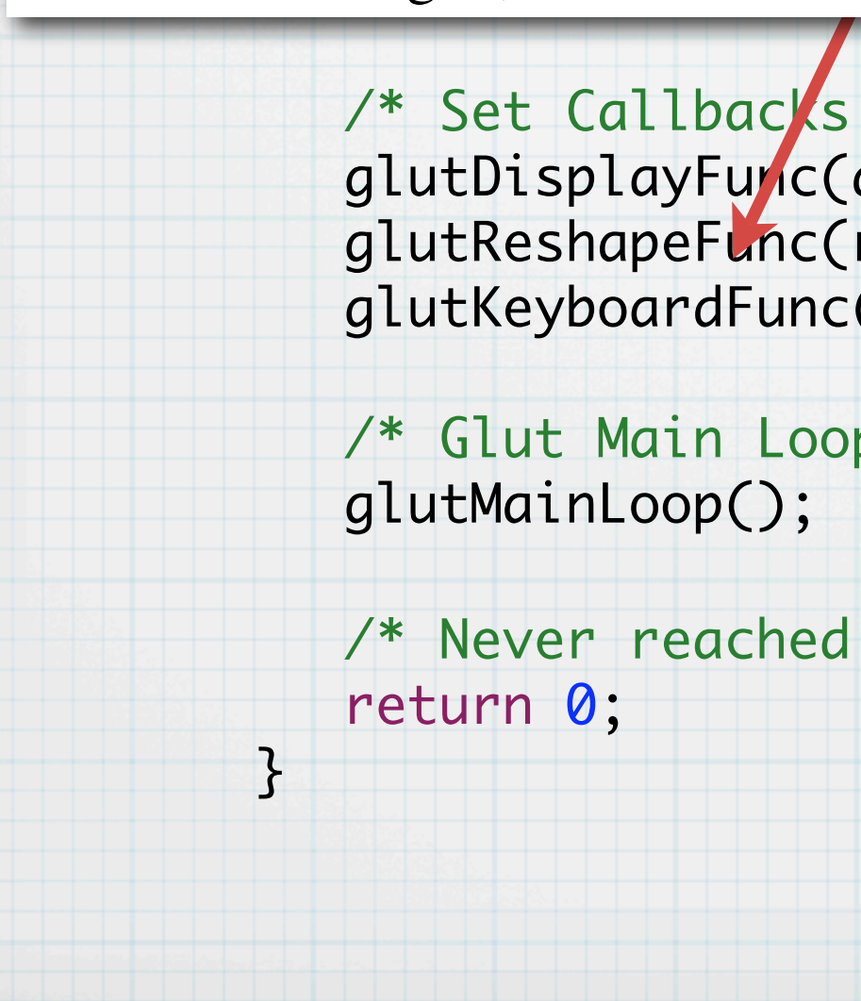
```
    /* Create main Window */
    glutCreateWindow("Axes");

    /* Set Callbacks */
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);

    /* Glut Main Loop */
    glutMainLoop();

    /* Never reached, keep compiler happy */
    return 0;
}
```

```
void glutReshapeFunc(void (*func)(int width, int height));
```

Installa `void func(int width, int height)` come callback di reshape (ridimensionamento). Tale callback è chiamato ogni qual volta è ridimensionata. Viene chiamata anche quando la finestra è creata.

Non è necessario registrarla. Esiste una di default. Tuttavia per programmi openGL non banali occorre registrarne una nuova.

Se una finestra contiene sotto-finestre, queste non sono ridimensionate automaticamente. Tale compito è responsabilità del callback della finestra padre. Se la finestra padre modifica le dimensioni di una finestra figlia, l'evento di ridimensionamento è messo in coda per la finestra figlia.



```
    /* Set Callbacks */
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);

    /* Glut Main Loop */
    glutMainLoop();

    /* Never reached, keep compiler happy */
    return 0;
}
```

```
int main (int argc, const char** argv) {
```

```
void glutKeyboardFunc(void (*func)(unsigned char key,
                                   int x, int y));
```

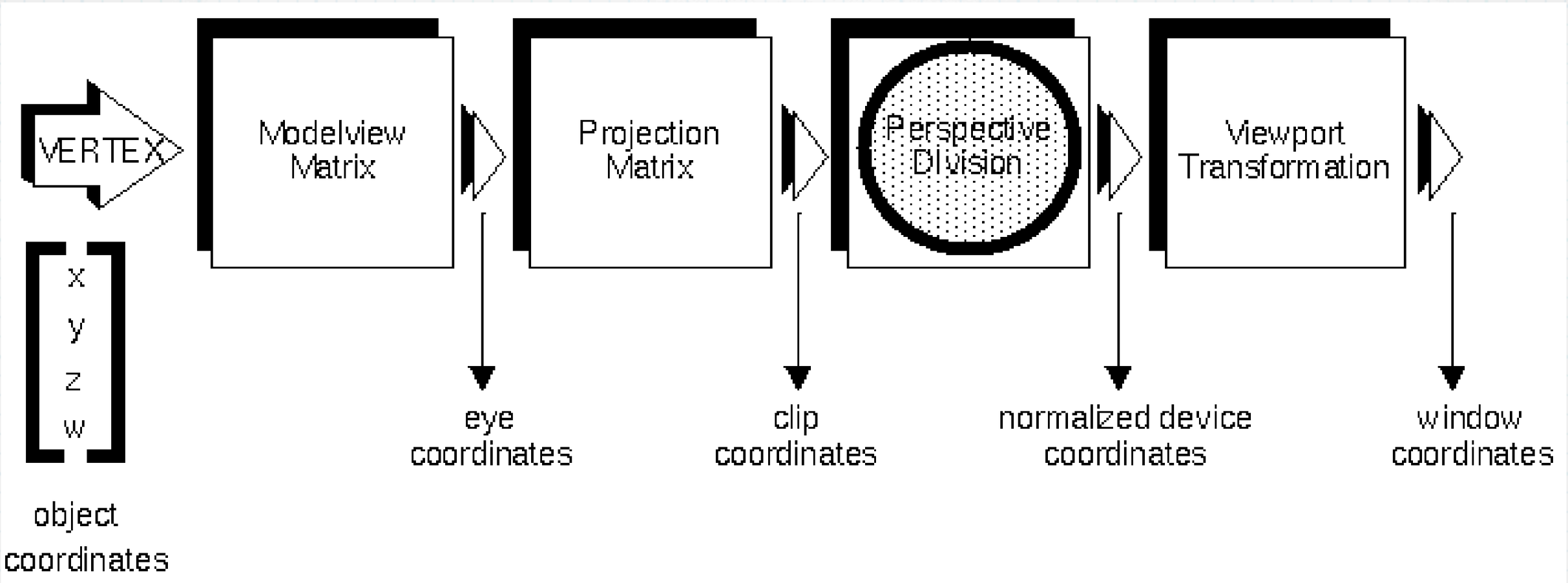
Installa `void (*func)(unsigned char key, int x, int y)` comme callback per la tastiera. Se l'utente digita sulla finestra questa funzione è chiamata.

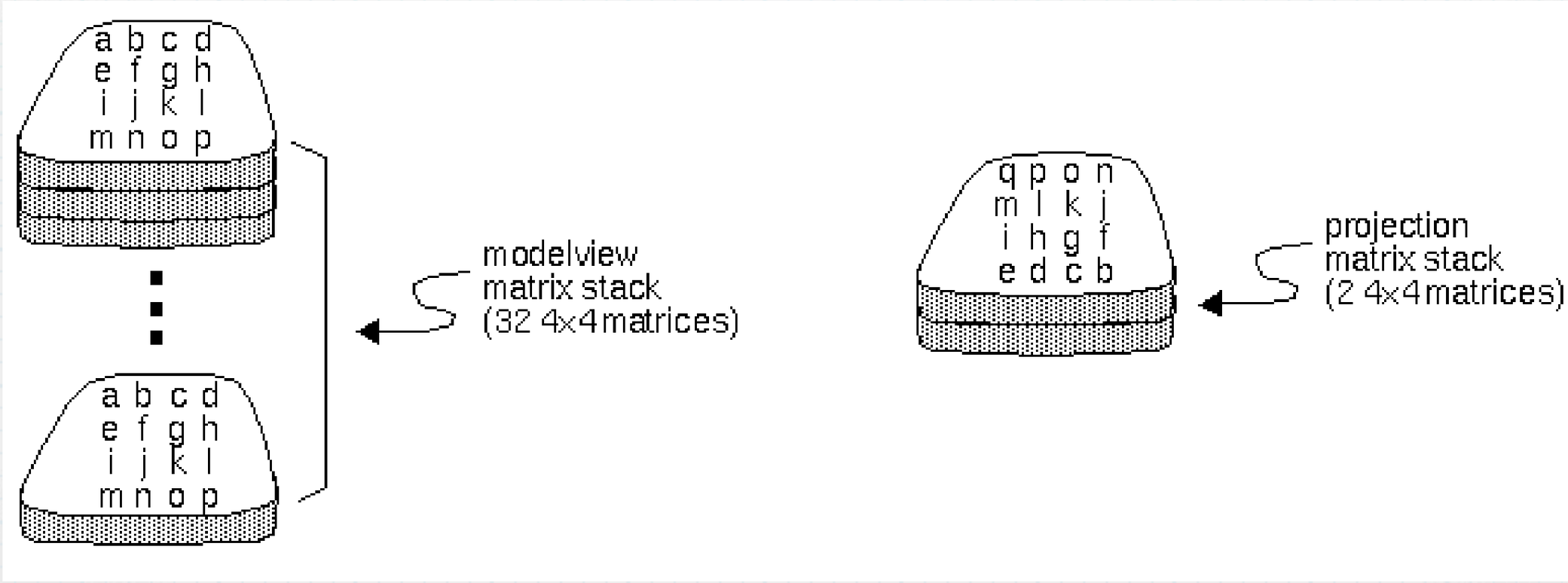
`key` è il codice ascii del tasto.
`x y` sono le coordinate correnti del mouse.

`int glutGetModifiers(void)` può essere chiamata per determinare eventuali tasti modificatori (control, alt, ...).

Per caratteri non ascii esiste la `void glutSpecialFunc(void (*func)(int key, int x, int y));`

```
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);

    /* Glut Main Loop */
    glutMainLoop();

    /* Never reached, keep compiler happy */
    return 0;
}
```


VERTEX
x
y
z
w
object
coordinates
Modelview
Matrix
eye
coordinates
Projection
Matrix
clip
coordinates
Perspective
Division
normalized device
coordinates
Viewport
Transformation
window
coordinates


a b c d
e f g h
i j k l
m n o p
a b c d
e f g h
i j k l
m n o p
modelview
matrix stack
(32 4x4 matrices)
q p o n
m l k j
i h g f
e d c b
projection
matrix stack
(2 4x4 matrices)

```c
int perspective=0;
void reshape(int w, int h) {
    /* Current aspect ratio */
    float ratio=((float) w) / ((float) h);

    /* Set viewport */
    glViewport(0, 0, w, h);

    /* Projection Matrix */
    glMatrixMode(GL_PROJECTION);

    /* Load identity */
    glLoadIdentity();

    /* The projection */
    if (!perspective) {
        /* Orthografic */
        /* Keep 1:1 ratio, shortest side must be at least [-1,1] */
        if (ratio>1) glOrtho( -ratio, ratio, -1, 1, 1, 50);
        else glOrtho( -1, 1, -1/ratio, 1/ratio, 1, 50);
    } else {
        /* OR Perspective */
        gluPerspective(35, ratio, 1, 50 );
    }

    /* Redraw this window */
    glutPostRedisplay();
}
```

```c
int perspective=0;
void reshape(int w, int h) {
    /* Current aspect ratio */
    float ratio=((float) w) / ((float) h);

    /* Set viewport */
    glViewport(0, 0, w, h);

    /* Projection Matrix */
    glMatrixMode(GL_PROJECTION);

    /* Load identity */
    g

    /
    i

    }

    }
    /* Redraw this window */
    glutPostRedisplay();
}
```

void **glViewport**(GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*)

Imposta l'area della finestra da disegnare con OpenGL.

Implicitamente imposta anche la matrice di trasformazione di dispositivo:

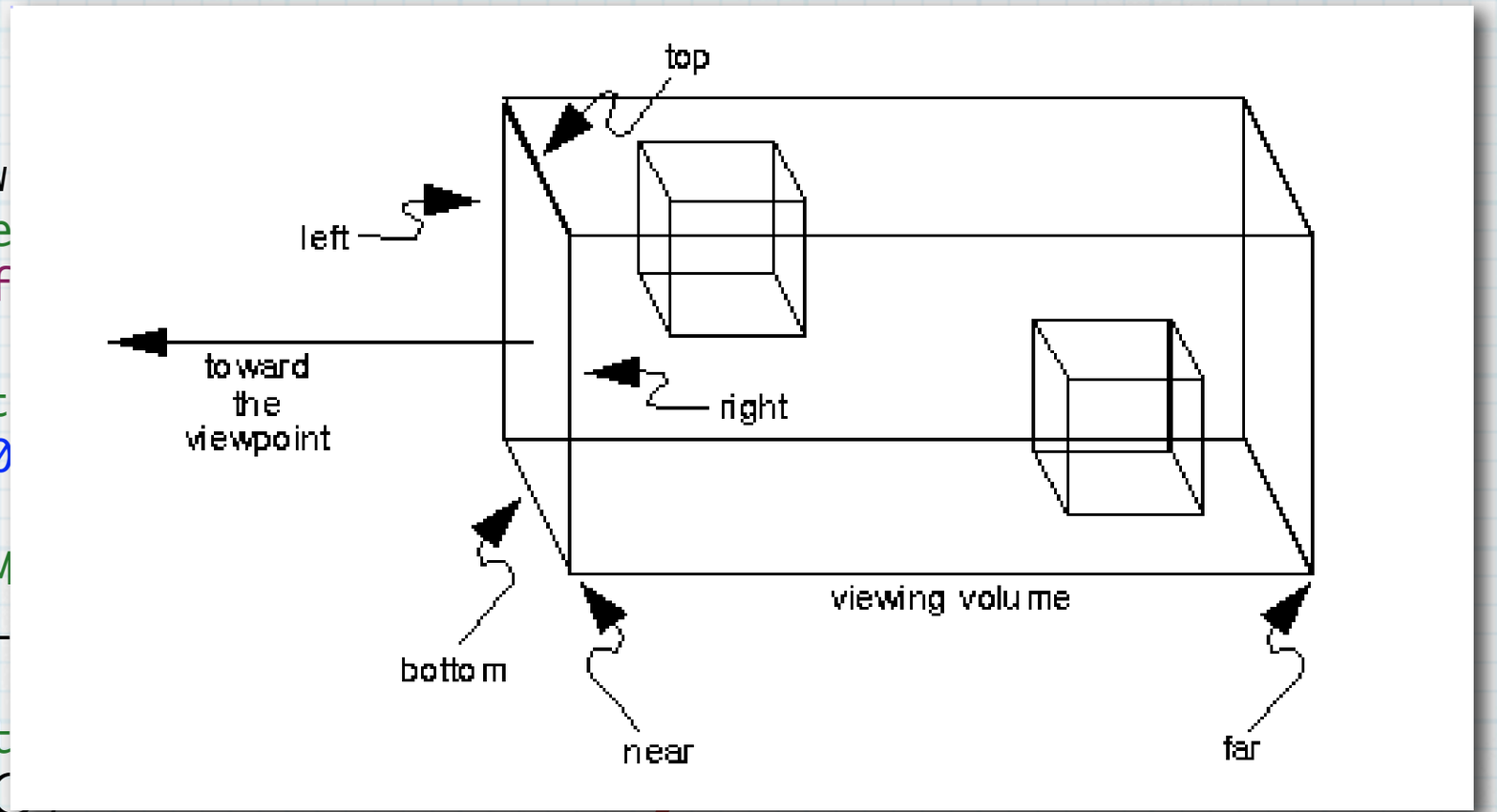
[-1,1]x[-1,1] -> [0, h] x [w, 0]

void **glMatrixMode**(GLenum *mode*)

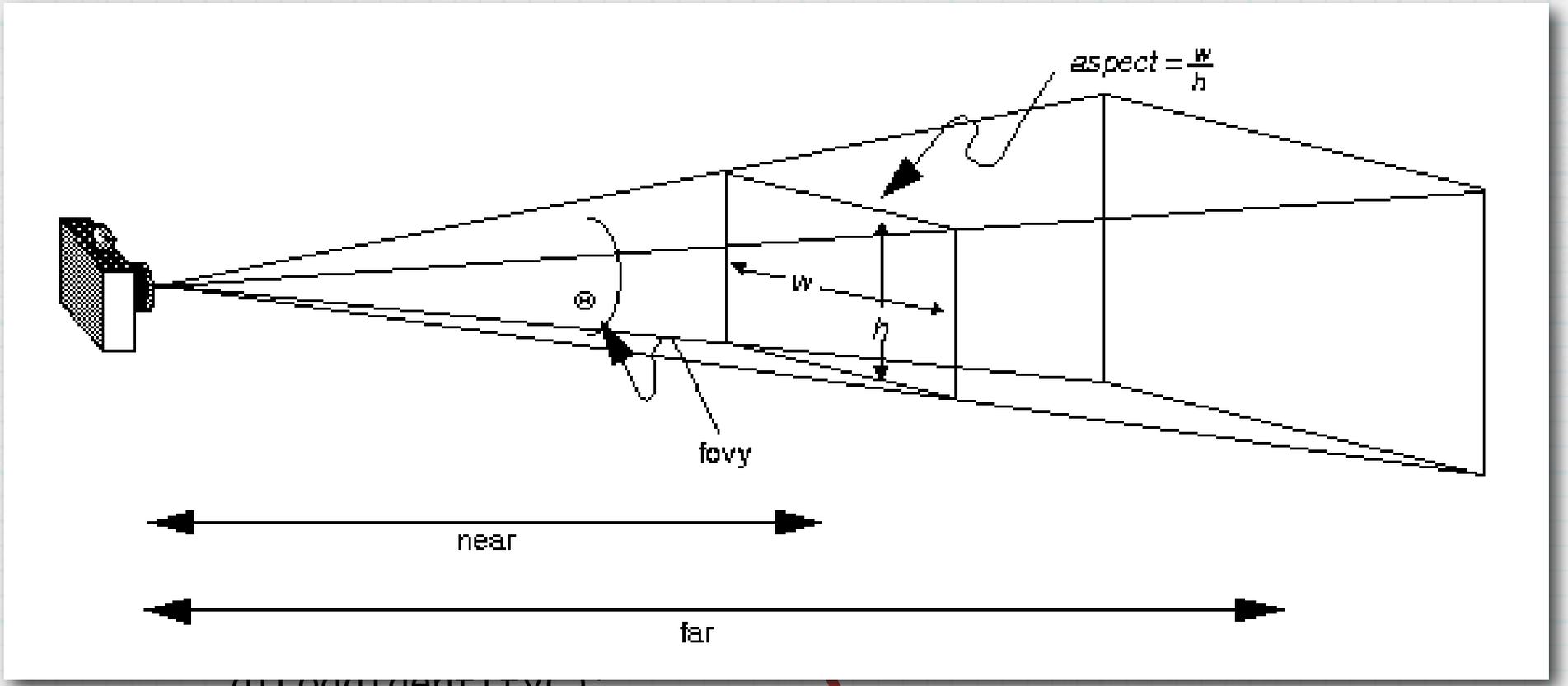
Specifica lo stack di matrici su cui si applicano le prossime operazioni:

GL_MODELVIEW, **GL_PROJECTION**, and **GL_TEXTURE**.

```
int perspective=0;
void reshape(int w
    /* Current aspe
    float ratio=((f

    /* Set viewport
    glViewport(0, 0

    /* Projection M
    glMatrixMode(GL

    /* Load identit
    glLoadIdentity(
    if (!perspective) {
        /* Orthografic */
        /* Keep 1:1 ratio, shortest side must be at least [-1,1] */
        if (ratio>1) glOrtho( -ratio, ratio, -1, 1, 1, 50);
        else glOrtho( -1, 1, -1/ratio, 1/ratio, 1, 3);
    } else {
        /* OR Perspective */
        gluPerspective(35, ratio, 1, 50 );
    }

    /* Redraw this window */
    glutPostRedisplay();
}
```



void **glOrtho**(GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*, GLdouble *near*, GLdouble *far*)




```
glLoadIdentity();

/* The projection */
if (!perspective) {
   /* Orthografic */
   /* Keep 1:1 ratio, shortest side must be at least [-1,1] */
   if (ratio>1) glOrtho( -ratio, ratio, -1, 1, 1, 50);
   else glOrtho( -1, 1, -1/ratio, 1/ratio, 1, 3);
} else {
   /* OR Perspective */
   gluPerspective(35, ratio, 1, 50 );
}

/* Redraw this window */
```

void **gluPerspective**(GLdouble *fovy*, GLdouble *aspect*, GLdouble *zNear*, GLdouble *zFar*)

```
void display(void) {
   /* Clear Buffer */
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   /* Position Model */
   position();

   /* Draw model */
   axes();

   /* Start drawing if you haven't */
   glFlush();
}
```

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

GL_COLOR_BUFFER_BIT

GL_DEPTH_BUFFER_BIT

GL_ACCUM_BUFFER_BIT

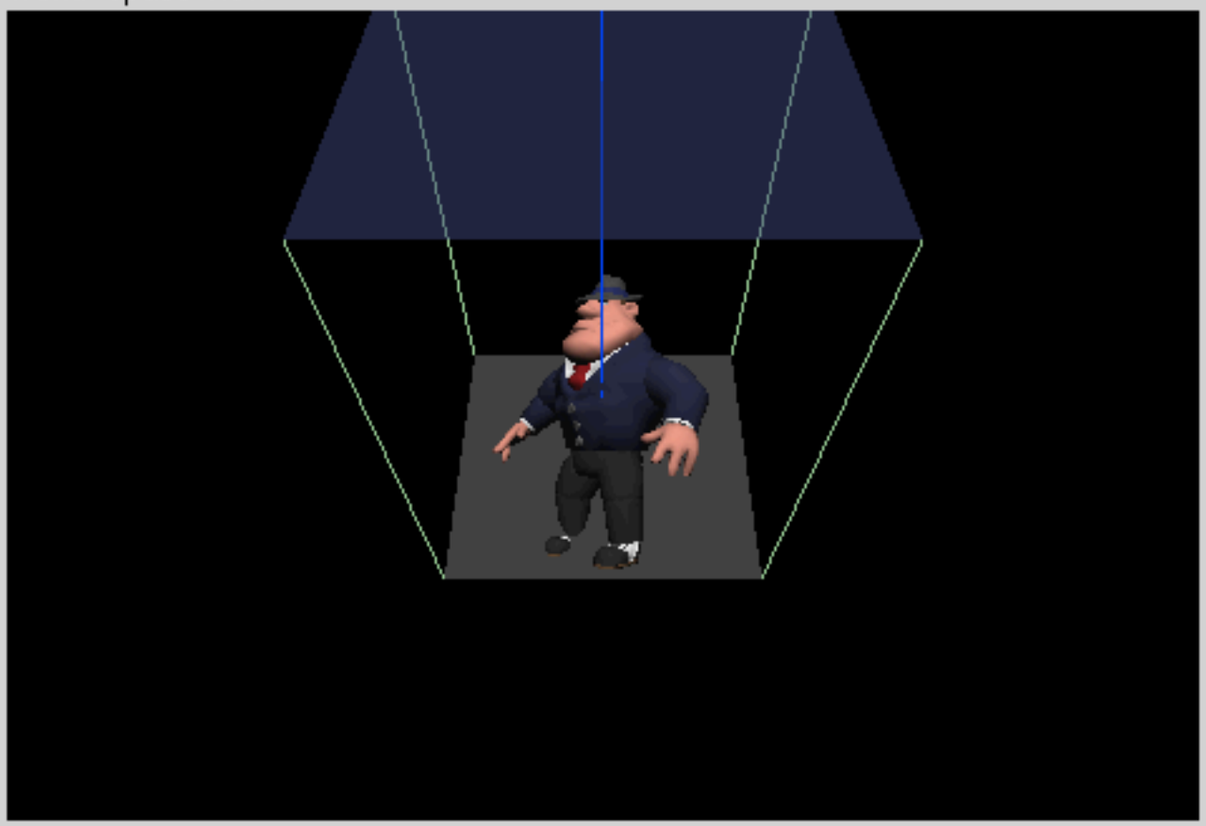
GL_STENCIL_BUFFER_BIT

```
void display(void) {
    /* Clear Buffer */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* Position Model */
    po

    /* Draw model */
    axes();

    /* Start drawing if you haven't */
    glFlush();
}
```

```
void position() {
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt( 2, 2, 2, 0, 0, 0, 1, 0, 0 );
}
```

Projection

World-space view



Screen-space view

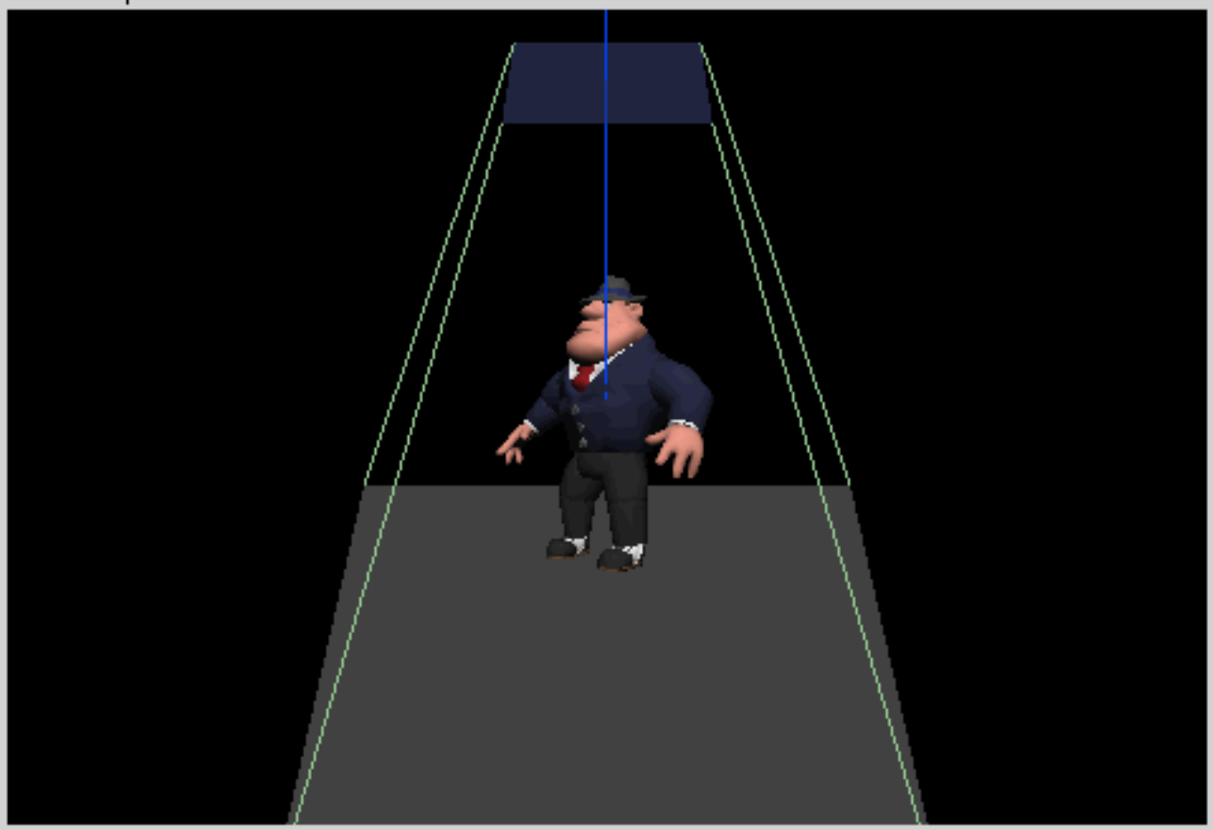
Command manipulation window

```
              left     right  bottom    top    near    far
   glOrtho(  -1.10 , 1.10   , -1.00 , 1.00   , 1.00   , 4.00  );
         gluLookAt( 2.00   , 2.00   , 2.00   ,     <- eye
                    0.00   , 0.00   , 0.00   ,     <- center
                    0.00   , 1.00   , 0.00   );    <- up
```

Click on the arguments and move the mouse to modify values.

Projection

World-space view



Screen-space view

Command manipulation window

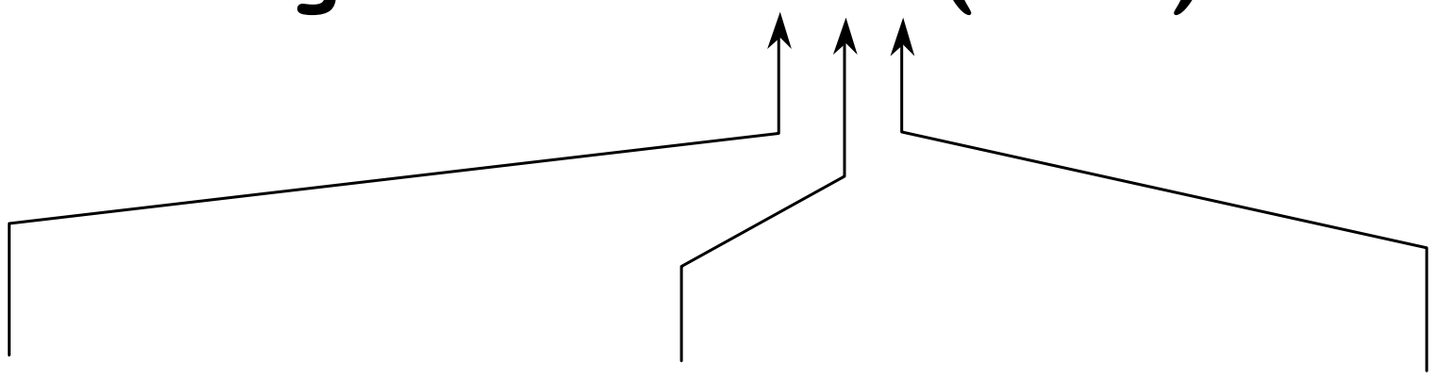
```
                  fovy  aspect  zNear  zFar
gluPerspective( 35.0  , 1.33  , 1.0   , 10.0  );
    gluLookAt( 2.00   , 2.00  , 2.00  ,    <- eye
               0.00   , 0.00  , 0.00  ,    <- center
               0.00   , 1.00  , 0.00  );   <- up
```

Click on the arguments and move the mouse to modify values.

```
void keyboard(unsigned char key, int x, int y) {
   /* Things to remember between calls */
   static int fullscreen=0;
   static int old_w,old_h;
   static int old_x,old_y;
   switch(key) {
      /* Q or esc: Quit */
      case 27:
      case 'q':
      case 'Q':
         exit(0);
         break;
/* F: Fullscreen toogle */
case 'f':
case 'F':
   /* Not in full screen */
   if (!fullscreen) {
      /* Save window size and position */
      old_w=glutGet(GLUT_WINDOW_WIDTH);
      old_h=glutGet(GLUT_WINDOW_HEIGHT);
      old_x=glutGet(GLUT_WINDOW_X);
      old_y=glutGet(GLUT_WINDOW_Y);
      /* Go fullscreen */
      glutFullScreen();
   } else {
```

```
            /* Set window size and shape to original values */
            glutReshapeWindow(old_w, old_h);
            glutPositionWindow(old_x, old_y);
            fullscreen=0;
        }
        glutPostRedisplay();
        break;
    /* p: Perspective */
    case 'p':
    case 'P':
        perspective=1;
        /* Call reshape */
            reshape(glutGet(GLUT_WINDOW_WIDTH),  glutGet
            (GLUT_WINDOW_HEIGHT));
            glutPostRedisplay();
            break;
        case 'o':
        case 'O':
            perspective=0;
            reshape(glutGet(GLUT_WINDOW_WIDTH),  glutGet
            (GLUT_WINDOW_HEIGHT));
            glutPostRedisplay();
            break;
    }
}
```

OpenGL Command Formats



Number of components

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

Data Type

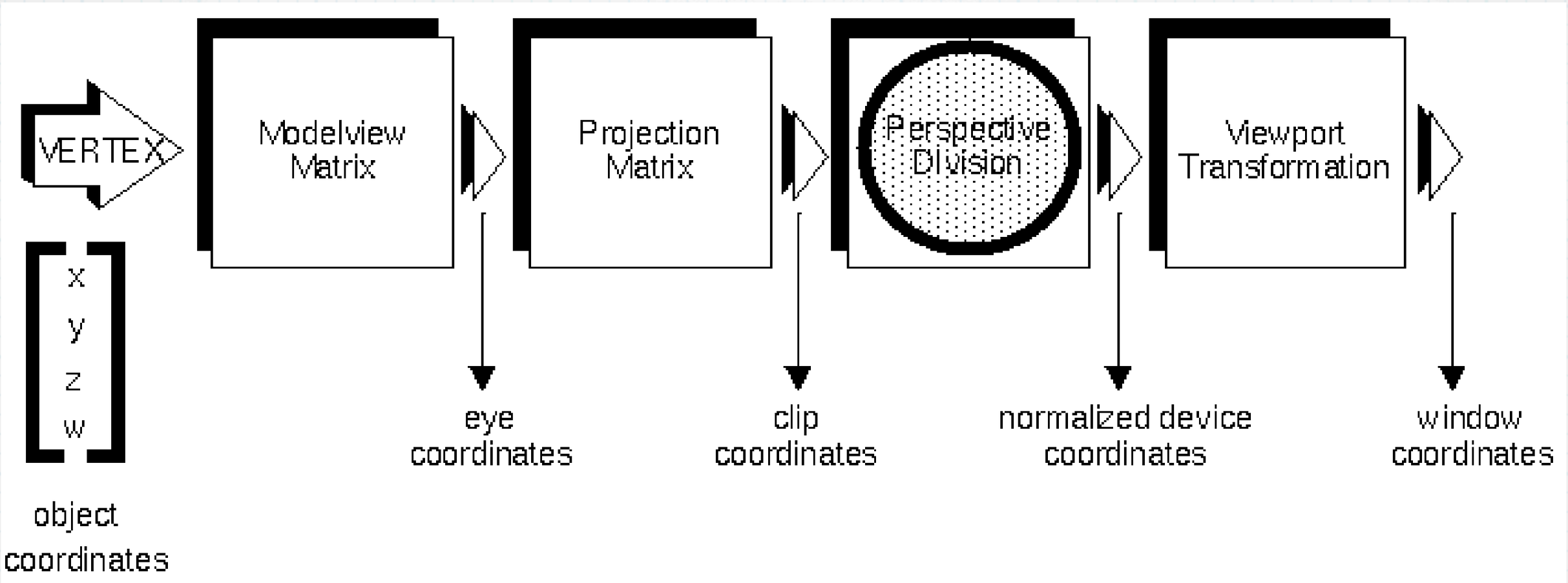
```
b  - byte
ub - unsigned byte
s  - short
us - unsigned short
i  - int
ui - unsigned int
f  - float
d  - double
```

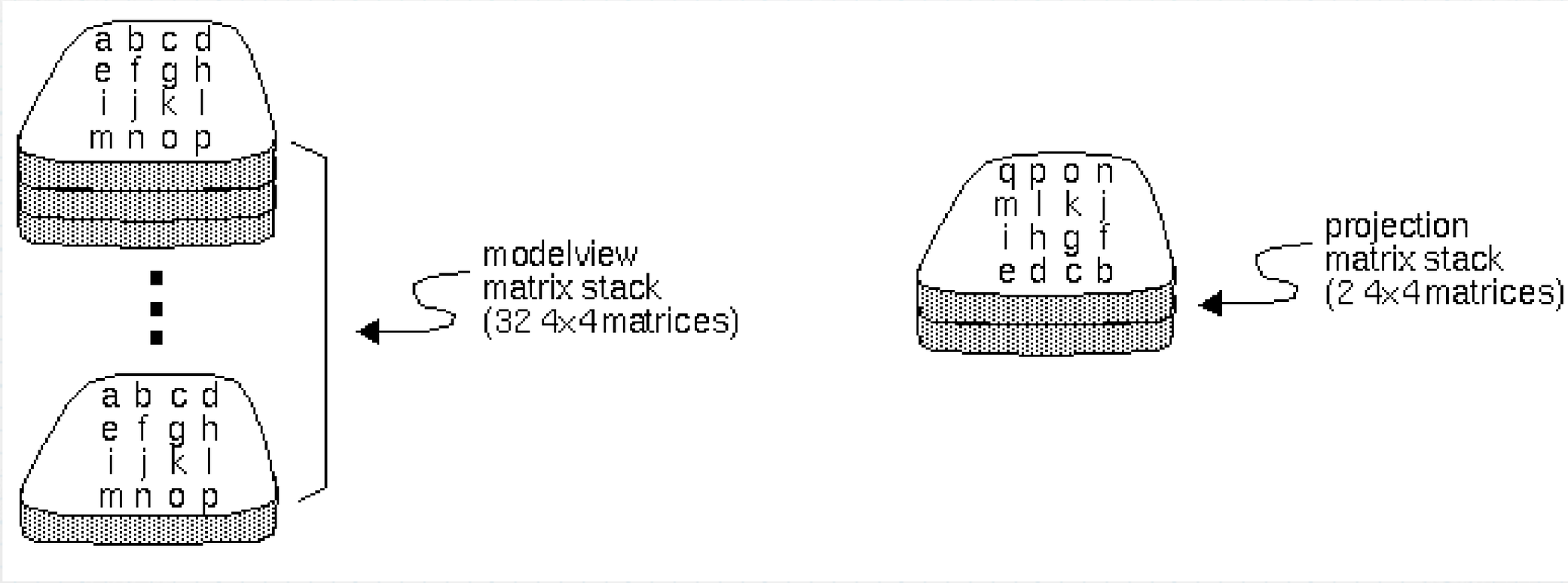
Vector

```
omit “v” for
scalar form

glVertex2f( x, y )
```

```
/* Draw standard Axis */
void axes() {
    int i;

    /* Data for just the standandard X axis */
    /* Others will be permutation on coordinates */
    GLfloat axis[][3] =   {  { -1  , 0    , 0    },
                             { 1   , 0    , 0    },
                             { 0.9 , 0.1  , 0    },
                             { 0.9 , -0.1 , 0    },
                             { 1   , 0    , 0    },
                            { 0.9 , 0    , 0.1  },
                            { 0.9 , 0    ,  -0.1 },
                            { 1   , 0    ,   0  } };

    /* X axis */
    glColor3ub(255, 0, 0);
    glBegin(GL_LINE_STRIP);
    //for(i=0;i<8;i++) glVertex3f(axis[i][0],axis[i][1],axis[i][2]);
    for(i=0;i<8;i++) glVertex3fv(axis[i]);
    glEnd();

    /* Y axis */
    glColor3ub(0, 255, 0);
    glBegin(GL_LINE_STRIP);
    for(i=0;i<8;i++) glVertex3f(axis[i][2],axis[i][0],axis[i][1]);
    glEnd();

    /* Z axis */
    glColor3ub(0, 0, 255);
    glBegin(GL_LINE_STRIP);
    for(i=0;i<8;i++) glVertex3f(axis[i][1],axis[i][2],axis[i][0]);
    glEnd();

}
```

Trasformazioni


VERTEX
x
y
z
w
object
coordinates
Modelview
Matrix
eye
coordinates
Projection
Matrix
clip
coordinates
Perspective
Division
normalized device
coordinates
Viewport
Transformation
window
coordinates


a b c d
e f g h
i j k l
m n o p
a b c d
e f g h
i j k l
m n o p
modelview
matrix stack
(32 4x4 matrices)
q p o n
m l k j
i h g f
e d c b
projection
matrix stack
(2 4x4 matrices)

Stack Matrici su cui operare

void **glMatrixMode**(GLenum *mode*)
GL_MODELVIEW, **GL_PROJECTION**, and **GL_TEXTURE**.

Carica Matrice identità

void **glLoadIdentity**(void)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Carica matrice

void **glLoadMatrixd**(const GLdouble *m*)
void **glLoadMatrixf**(const GLfloat *m*)

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

Multiplica per matrice

void **glMultMatrixd**(const GLdouble *m*)
void **glMultMatrixf**(const GLfloat *m*)

glMultMatrix multiplies the current matrix with the one specified in *m*. That is, if M is the current matrix and T is the matrix passed to **glMultMatrix**, then M is replaced with MT.

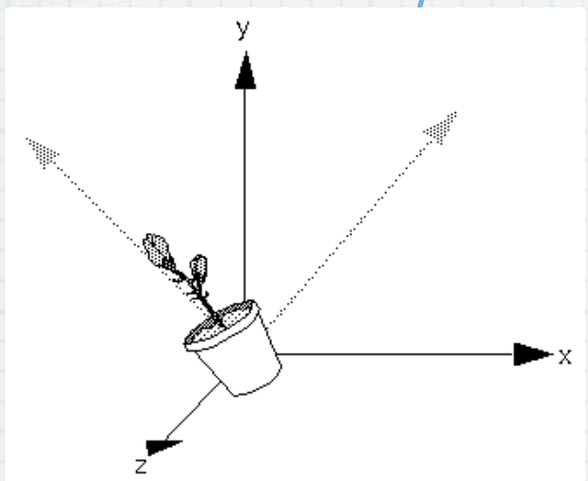
void **glRotated**(GLdouble *angle*, GLdouble *x*, GLdouble *y*, GLdouble *z*)
void **glRotatef**(GLfloat *angle*, GLfloat *x*, GLfloat *y*, GLfloat *z*)

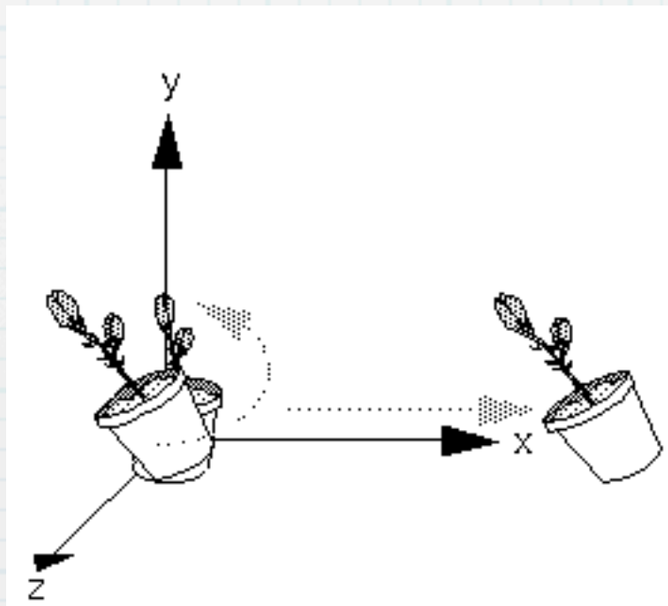
void **glScaled**(GLdouble *x*, GLdouble *y*, GLdouble *z*)
void **glScalef**(GLfloat *x*, GLfloat *y*, GLfloat *z*)

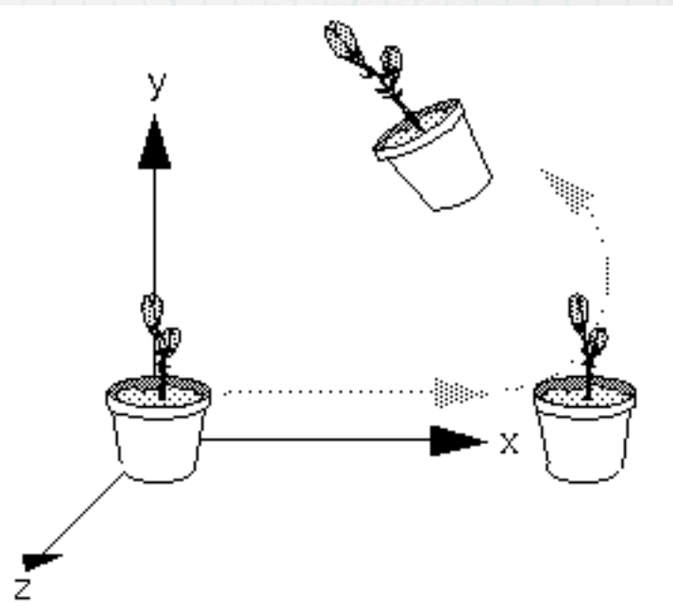
void **glTranslated**(GLdouble *x*, GLdouble *y*, GLdouble *z*)
void **glTranslatef**(GLfloat *x*, GLfloat *y*, GLfloat *z*)

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



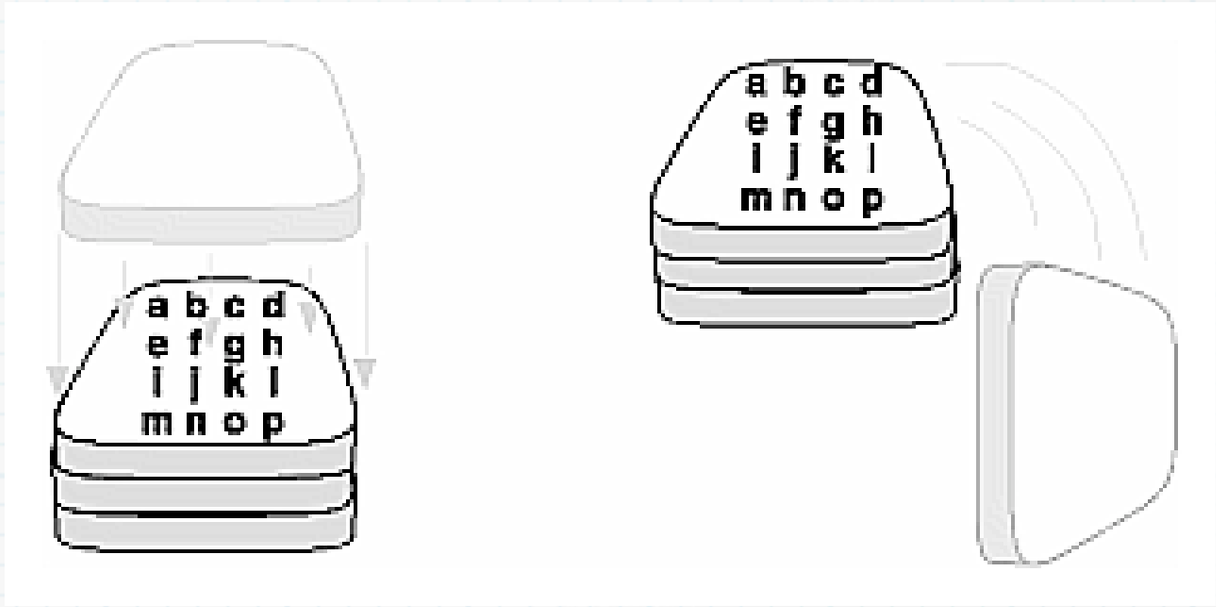




C SPECIFICATION

void **glPushMatrix**(void)

void **glPopMatrix**(void)



DESCRIPTION

There is a stack of matrices for each of the matrix modes. In **GL_MODELVIEW** mode, the stack depth is at least 32. In the other two modes, **GL_PROJECTION** and **GL_TEXTURE**, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode.

glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a **glPushMatrix** call, the matrix on the top of the stack is identical to the one below it.

glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack.

Secondo Programma

```c
int main (int argc, const char** argv) {

    /* Init Glut */
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);

    /* Main Window */
    glutInitWindowSize(800, 600);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Axes");

    /* Set Callbacks */
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(mousemotion);

    /* Glut Main Loop */
    glutMainLoop();

    /* Never reached, keep compiler happy */
    return 0;
}
```

```
void display(void) {

  /* Clear Buffer */
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glEnable(GL_DEPTH_TEST);

 position();

  axes();

  /* Draw model */
    model();

  /* SwapBuffer */
  glutSwapBuffers();
}
```

```
void model() {
  GLfloat light_pos[]={ 2.0, 2.0, 3.0 };

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

  /* Animated move */
   moveposition();

  /* Model */
  glColor3ub(100, 100, 100);
  glutSolidTeapot(1);

  /* Goto darkness */
  glDisable(GL_LIGHTING);
}
```

```
struct Viewpoint {
  GLfloat center[3];
   GLfloat alpha;
  GLfloat beta;
  GLfloat gamma;
  GLfloat zoom;
};
struct Viewpoint init_viewpoint = { { 0, 0, 0}, 0, 0, 0, 1};
struct Viewpoint viewpoint = { { 0, 0, 0}, 0, 0, 0, 1};
```

```
enum Motion_mode { NONE=0, ROTATE, DIST,  PAN };

struct Motion {
  int mode;
  int startx;
  int starty;
};

struct Motion motion = { NONE, 0, 0 };
```

```
void mouse(int button, int state, int x, int y) {
  if(state==GLUT_DOWN) {
    motion.startx=x;
    motion.starty=y;

    switch (button) {
      case GLUT_LEFT_BUTTON:
        motion.mode=ROTATE;
        break;
      case GLUT_MIDDLE_BUTTON:
        motion.mode=DIST;
        break;
      case GLUT_RIGHT_BUTTON:
        motion.mode=PAN;
        break;
      default:
        motion.mode=NONE;
        }
  } else {
    motion.mode=NONE;
    }

}
```

```
void mousemotion(int x, int y) {

   switch (motion.mode) {

      case ROTATE:
         viewpoint.alpha+=(GLfloat) ((x-motion.startx) % 360);
         viewpoint.beta+=(GLfloat) ((y-motion.starty) % 360);
         break;
      case DIST:
         viewpoint.zoom+=4.0 * ((GLfloat) (y-motion.starty))/((GLfloat) glutGet(GLUT_WINDOW_HEIGHT));
         break;
      case PAN:
         viewpoint.center[0]+=4.0 * ((GLfloat) (x-motion.startx)) / ((GLfloat) glutGet(GLUT_WINDOW_WIDTH));
         if(glutGetModifiers()==GLUT_ACTIVE_SHIFT) { // Non funge!!!
            viewpoint.center[2]-=4.0 * ((GLfloat) (y-motion.starty)) /
                                    ((GLfloat) glutGet(GLUT_WINDOW_HEIGHT));
         } else {
            viewpoint.center[1]-=4.0 * ((GLfloat) (y-motion.starty)) /
                                    ((GLfloat) glutGet(GLUT_WINDOW_HEIGHT));
                  }
         break;
       default:
         break;
       }

   motion.startx=x;
   motion.starty=y;

   glutPostRedisplay();
}
```

```
void moveposition() {
  glMatrixMode(GL_MODELVIEW);

  glTranslatef(viewpoint.center[0]-viewpoint.center[2],
               viewpoint.center[1],
               viewpoint.center[2]-viewpoint.center[0]
               );
  glRotatef(viewpoint.beta, cos(M_PI_4), 0, -sin(M_PI_4) );
  glRotatef(viewpoint.alpha, 0, 1, 0 );
  glScalef( viewpoint.zoom, viewpoint.zoom, viewpoint.zoom);
}
```