REPRESENTACION COMPUTACIONAL DE SOLIDOS

Universidad Tecnológica Nacional
Facultad Regional Haedo
Secretaría de Ciencia y Tecnología
Centro de Estudios de Informática

Ing. Juan C. Polidoro
jpolidoro@frh.utn.edu.ar

Representación de sólidos

Que es?

- Es la representación no ambigua de las partes sólidas de un objeto, de manera tal que sea tratable por métodos computacionales.

Objetivo al iniciar el desarrollo

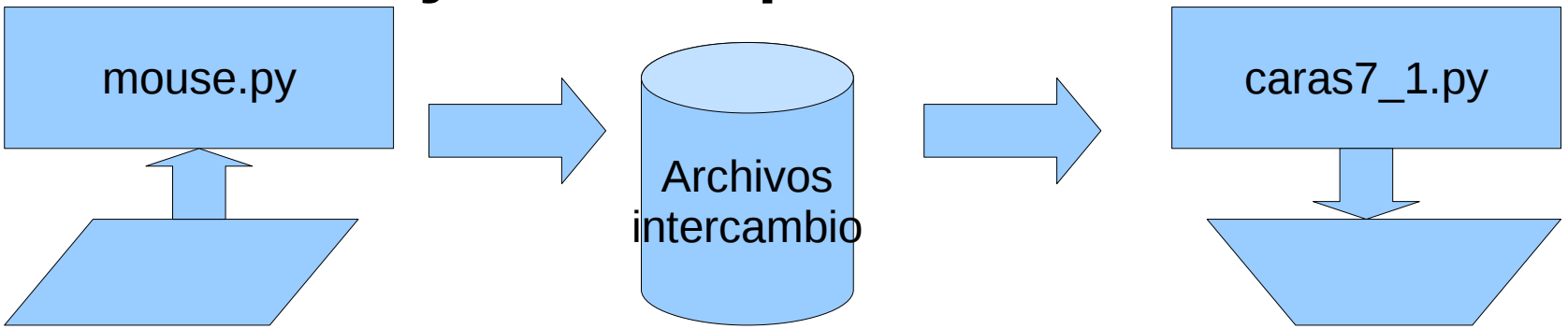
- Obtener los conocimiento básicos para poder generar y manipular un objeto en 3D, desarrollando sofware que adhiere a la política de la Universidad al respecto. (SL, Linux, Python, Tkinter).

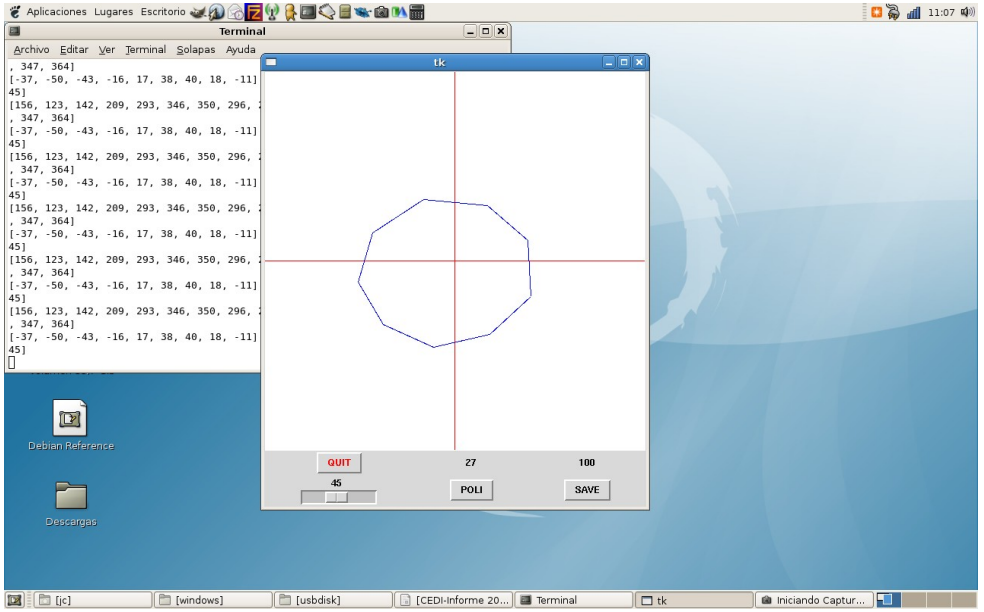
➔ Principales formas de representación

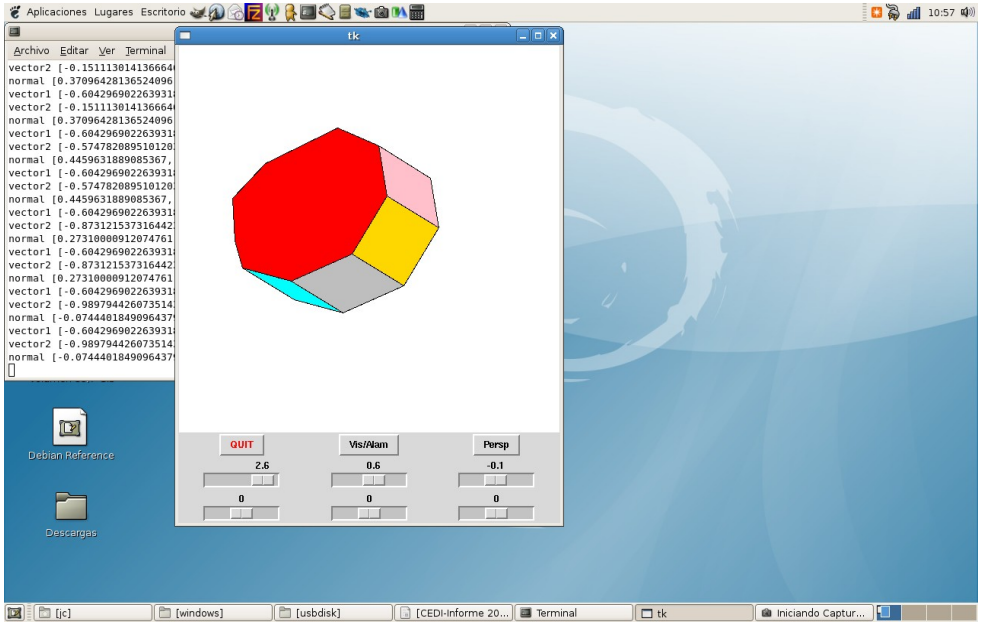
- CGS (Formas primitivas, operaciones booleanas)

- B-Rep (Sólido representado por superficies límites)
  - Internamente se representan por un conjunto de datos:
    - Geométricos (vertices)
    - Topológicos (Forma en que estan relacionados los elementos geométricos, bordes y caras)

Flujo de operaciones



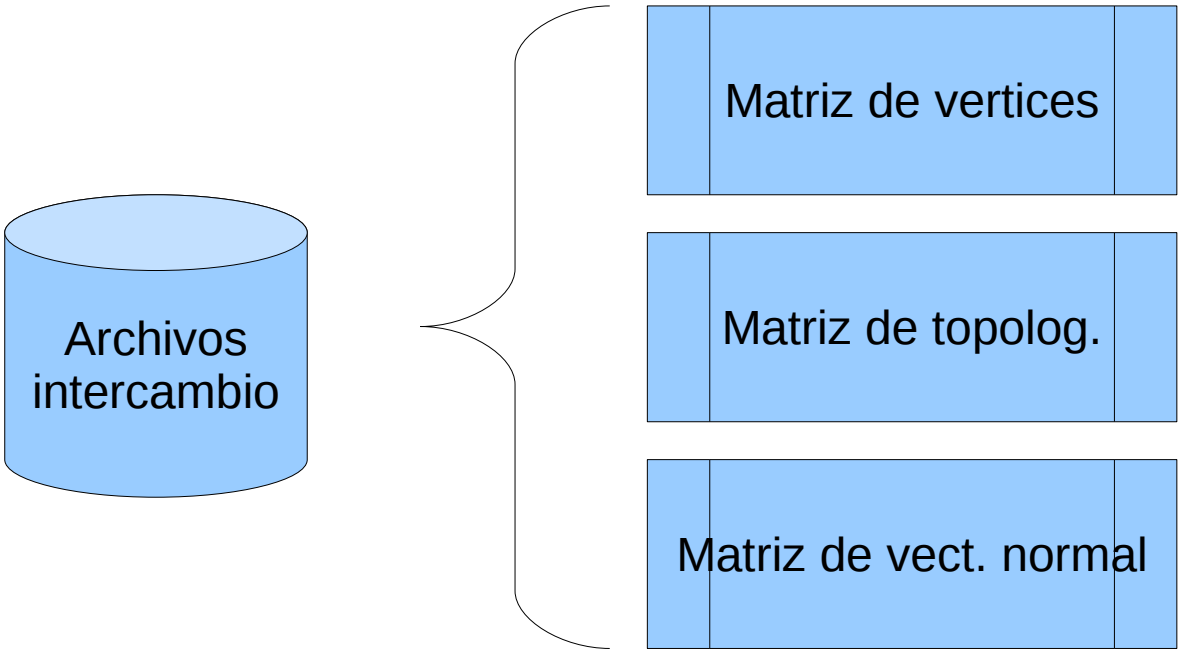




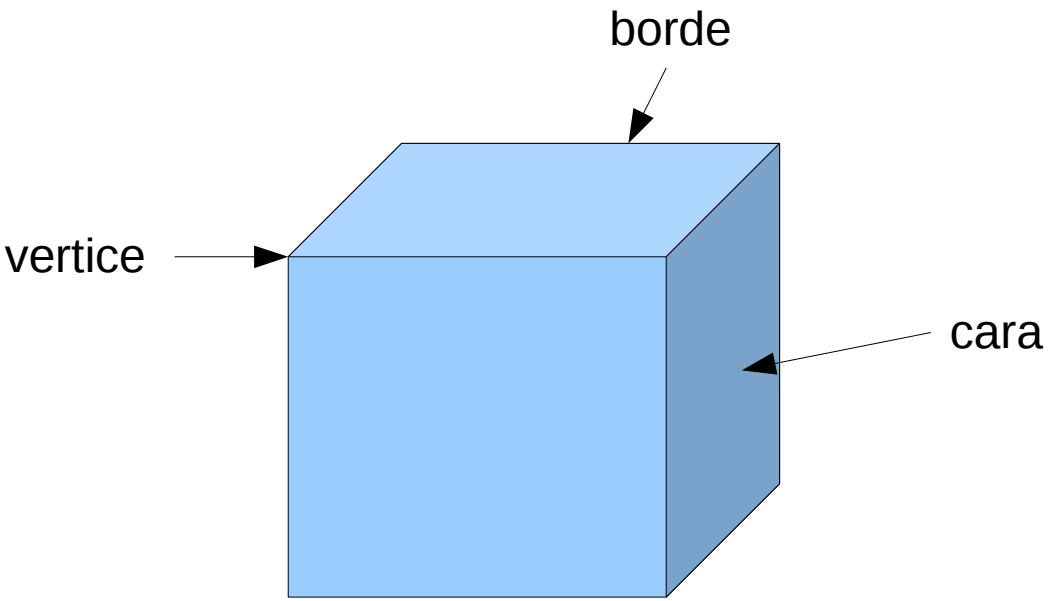
Ingreso de datos geometricos y topolog.

Exhibicion y manejo de datos

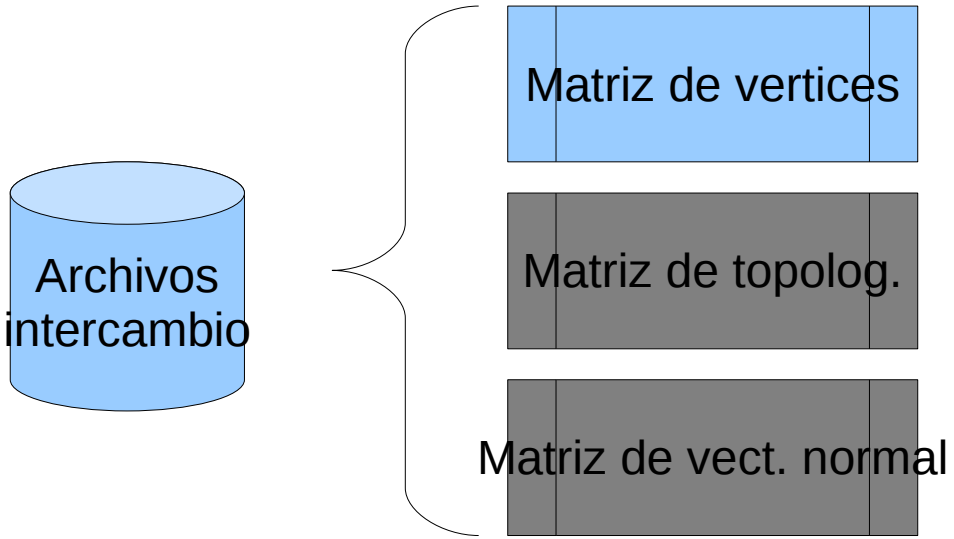
Intercambio de datos



Nomenclatura



Estructura de la matriz de geometría



La geometría de los vértices se guarda en una matriz vert[ ] de dos dimensiones

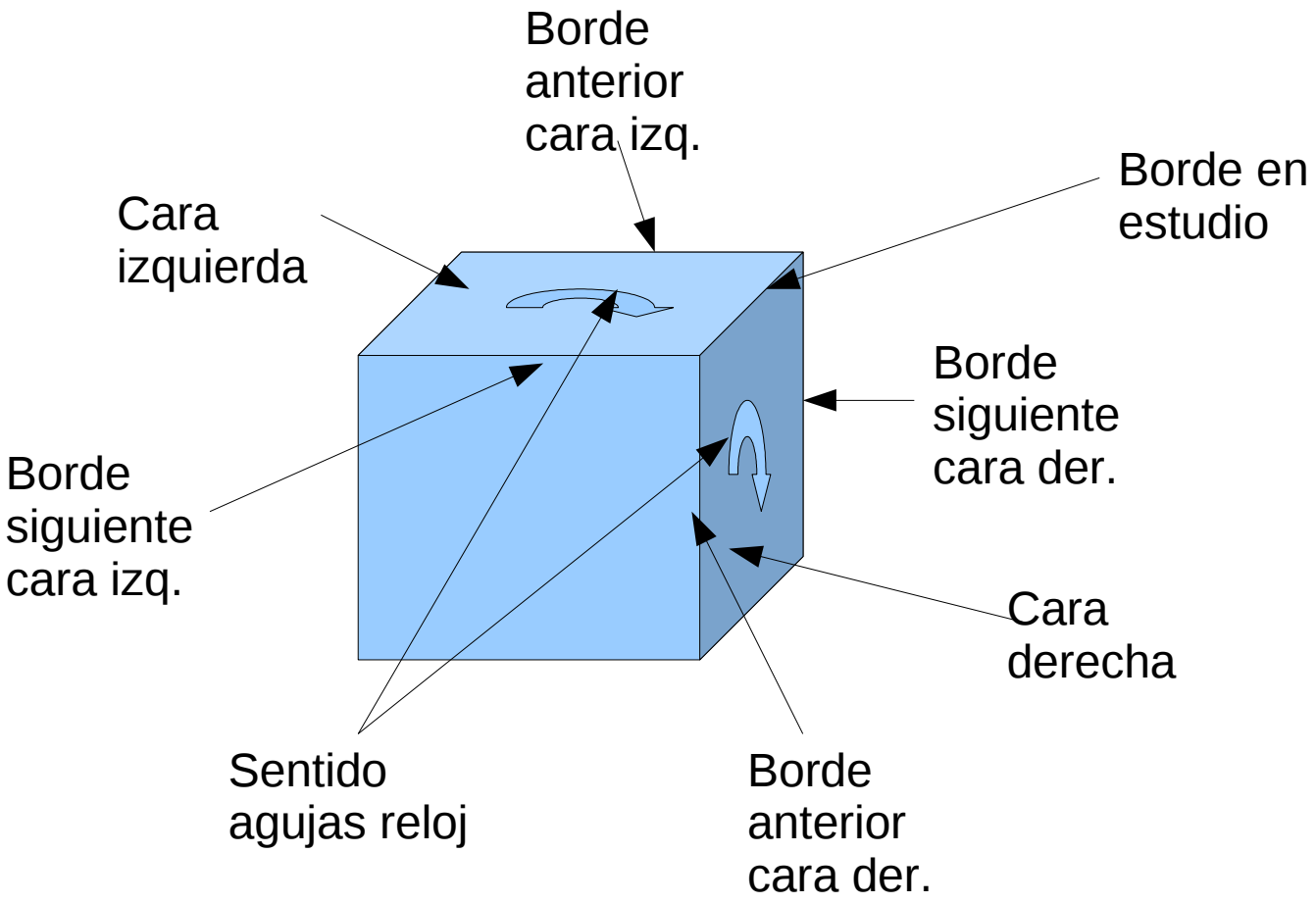
vert [ ]

Nro vertices 1....n

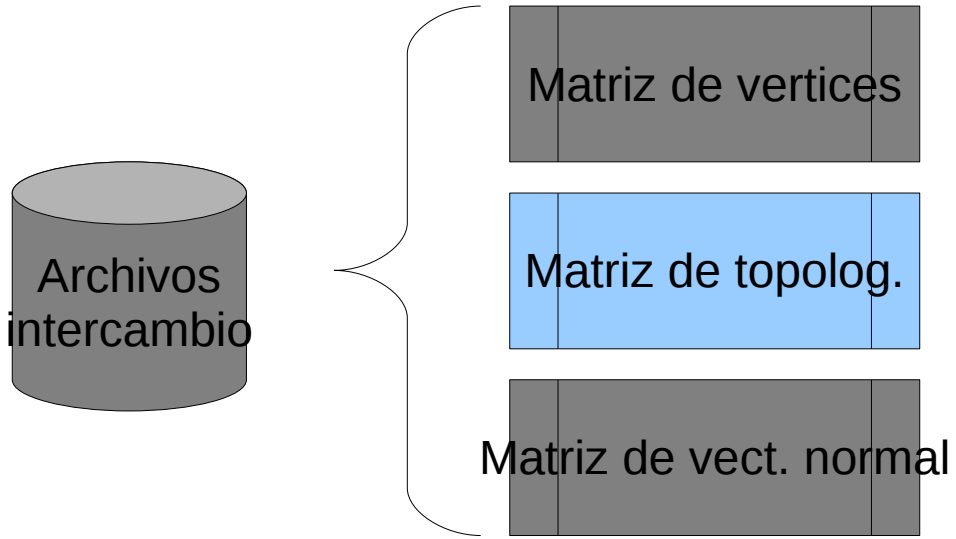
X	Y	Z	1

Nomenclatura utilizada por Baumgart

Bruce Baumgart -Stanford 1975



Estructura matriz de topología



La topología del cuerpo esta guardada en una matriz e[ ] de dos dimensiones con la siguiente estructura para los bordes. (*Estructura de datos de Baumgart).*

e [ ]

Nro borde 1.....n

Vertice inicio	Vertice fin	Cara izquierda	Cara derecha	Borde anterior Cara izq	Borde siguiente Cara izq	Borde anterior Cara der	Borde siguiente Cara der

Ejemplo matriz topología

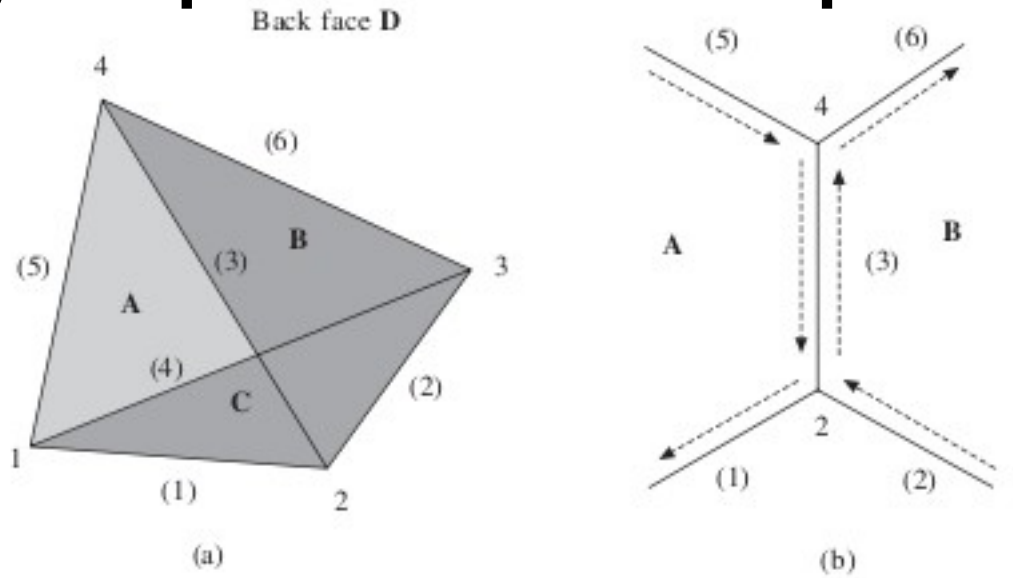
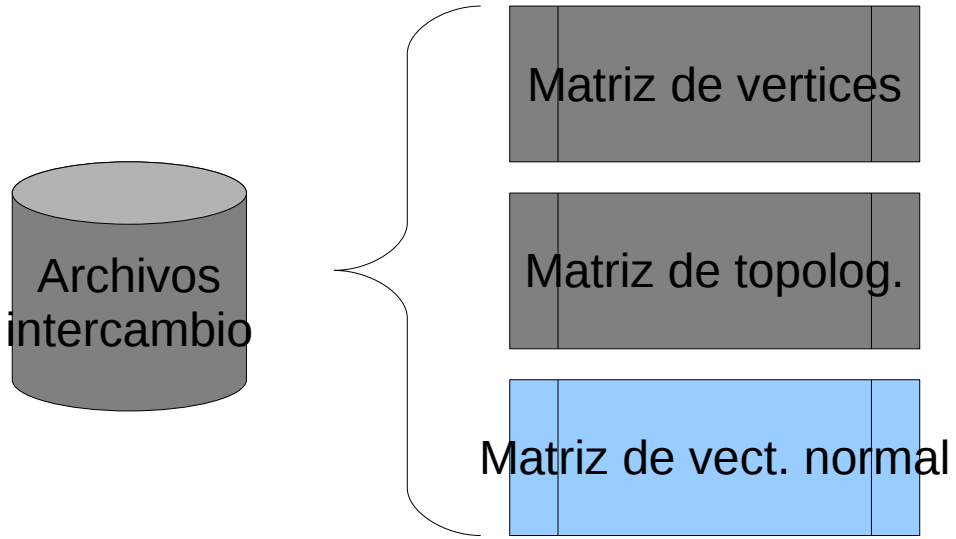


Figure 8.17 Winged-edge data structure for a tetrahedron

Edge table

Edge	Vertices		Faces		Clockwise traverse on left face		Clockwise traverse on right face	
Name	Start	End	Left	Right	Preceding edge	Suceeding edge	Preceding edge	Suceeding edge
(3)	2	4	**A**	**B**	(5)	(1)	(2)	(6)
(1)	1	2	**A**	**C**	(3)	(5)	(4)	(2)
(2)	2	3	**B**	**C**	(6)	(3)	(1)	(4)
(4)	1	3	**D**	**C**	(5)	(6)	(2)	(1)
(5)	1	4	**D**	**A**	(6)	(4)	(1)	(3)
(6)	3	4	**B**	**D**	(3)	(2)	(4)	(5)

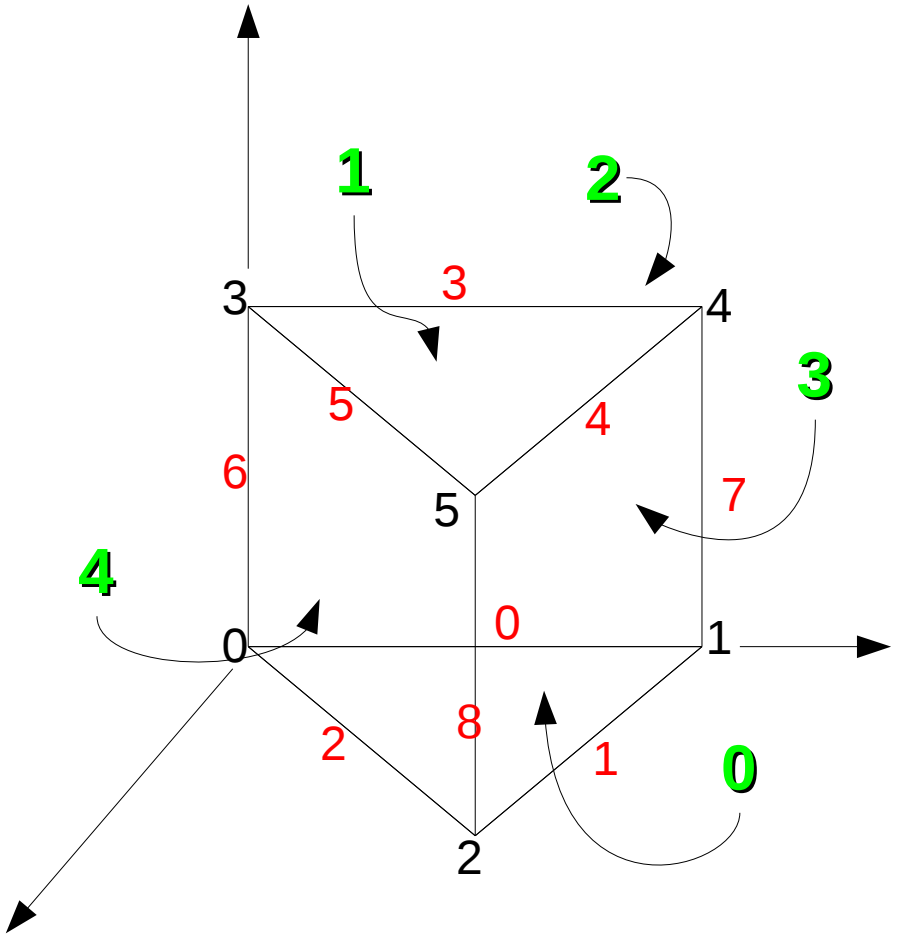
Estructura matriz de vector normal



Y se completa con una matriz f[ ] de dos dimensiones que contiene un borde perteneciente a la cara y las componentes del vector normal a la misma.

Nro cara 1....n

Borde perteneciente a cara	No usado	No usado	No usado	No usado	Componente versor normal	Componente versor normal	Componente versor normal



Vertices

```
vert[]----------------------
0  [[  0.   0.   0.   1.]
1   [  0.  50.   0.   1.]
2   [ 25.  25.   0.   1.]
3   [  0.   0.  40.   1.]
4   [  0.  50.  40.   1.]
5   [ 25.  25.  40.   1.]]
```

Bordes

```
e[]----------------------
0  [[0 1 2 0 7 6 2 1]
1   [1 2 3 0 8 7 0 2]
2   [2 0 4 0 6 8 1 0]
3   [3 4 1 2 4 5 6 7]
4   [4 5 1 3 5 3 7 8]
5   [5 3 1 4 3 4 8 6]
6   [0 3 4 2 5 2 0 3]
7   [1 4 2 3 3 0 1 4]
8   [2 5 3 4 4 1 2 5]]
```

Caras

```
f[]----------------------
0  [[0 0 0 0 0 0 0 0]
1   [3 3 3 3 3 3 3 3]
2   [6 6 6 6 6 6 6 6]
3   [7 7 7 7 7 7 7 7]
4   [8 8 8 8 8 8 8 8]]
```

```
def search_edge(xe,xf):
        # esta es una busqueda recursiva de los vertices que componen un lado
        # xe es el borde
        # xf es la cara
        global e
        global pp       # es la variable de entorno que guarda el borde
                        #con la que estoy llamando inicialmente
        global flag     #bandera para diferenciar la primera vez
                        #que encuentro la cara
        global puntos           # lista de entorno para guardar los puntos de la cara
        print "----"," borde ",xe,"cara ",xf,"flag",flag,"borde llamad",pp
        if (xe==pp and flag >0): # or flag>2:
                return
        else:
                flag=flag+1
                if e[xe][2]==xf:
                        e_next=e[xe][5]
                        f_next=e[xe][2]
                        puntos.append(e[xe][1]) # guardo el punto final del borde
                elif e[xe][3]==xf:
                        e_next=e[xe][7]
                        f_next=e[xe][3]
                        puntos.append(e[xe][0]) # guardo el punto final del borde
                else:
                        e_next=99
                        f_next=98
                print " prox vert",e_next,"prox cara",f_next

                search_edge(e_next,f_next)
```

```
     0  1 2 3 4 5 6 7
e[]----------------------
0  [[0 1 2 0 7 6 2 1]
1   [1 2 3 0 8 7 0 2]
2   [2 0 4 0 6 8 1 0]
3   [3 4 1 2 4 5 6 7]
4   [4 5 1 3 5 3 7 8]
5   [5 3 1 4 3 4 8 6]
6   [0 3 4 2 5 2 0 3]
7   [1 4 2 3 3 0 1 4]
8   [2 5 3 4 4 1 2 5]]
f[]----------------------
0  [[0 0 0 0 0 0 0 0]
1   [3 3 3 3 3 3 3 3]
2   [6 6 6 6 6 6 6 6]
3   [7 7 7 7 7 7 7 7]
4   [8 8 8 8 8 8 8 8]]
```

xe	xf	pp	e_next	f_next	puntos
0	0	0	1	0	0
1	0		2	0	1
2	0		0	0	2
0	0	0			

Interfase usuario mouse.py

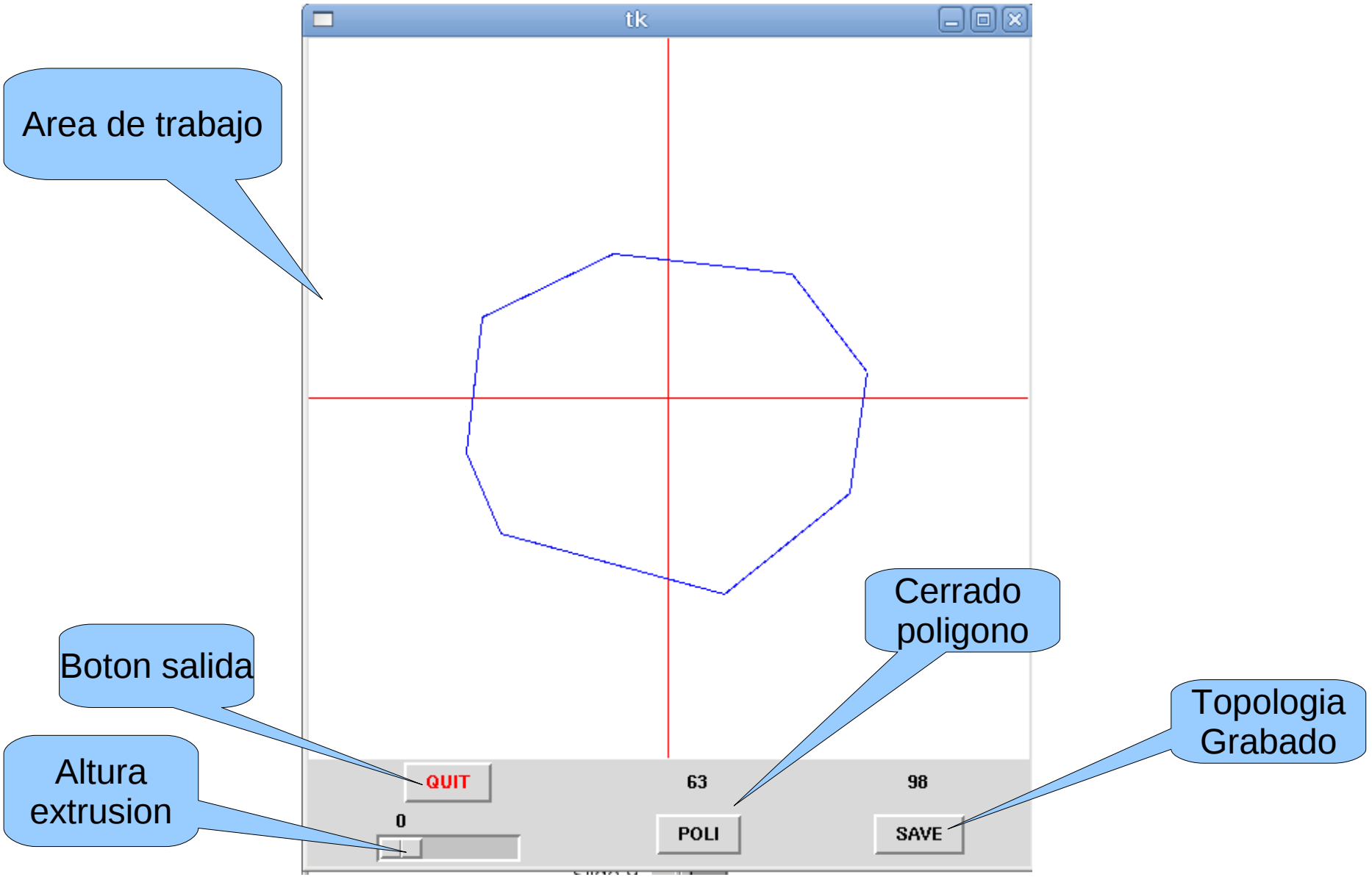
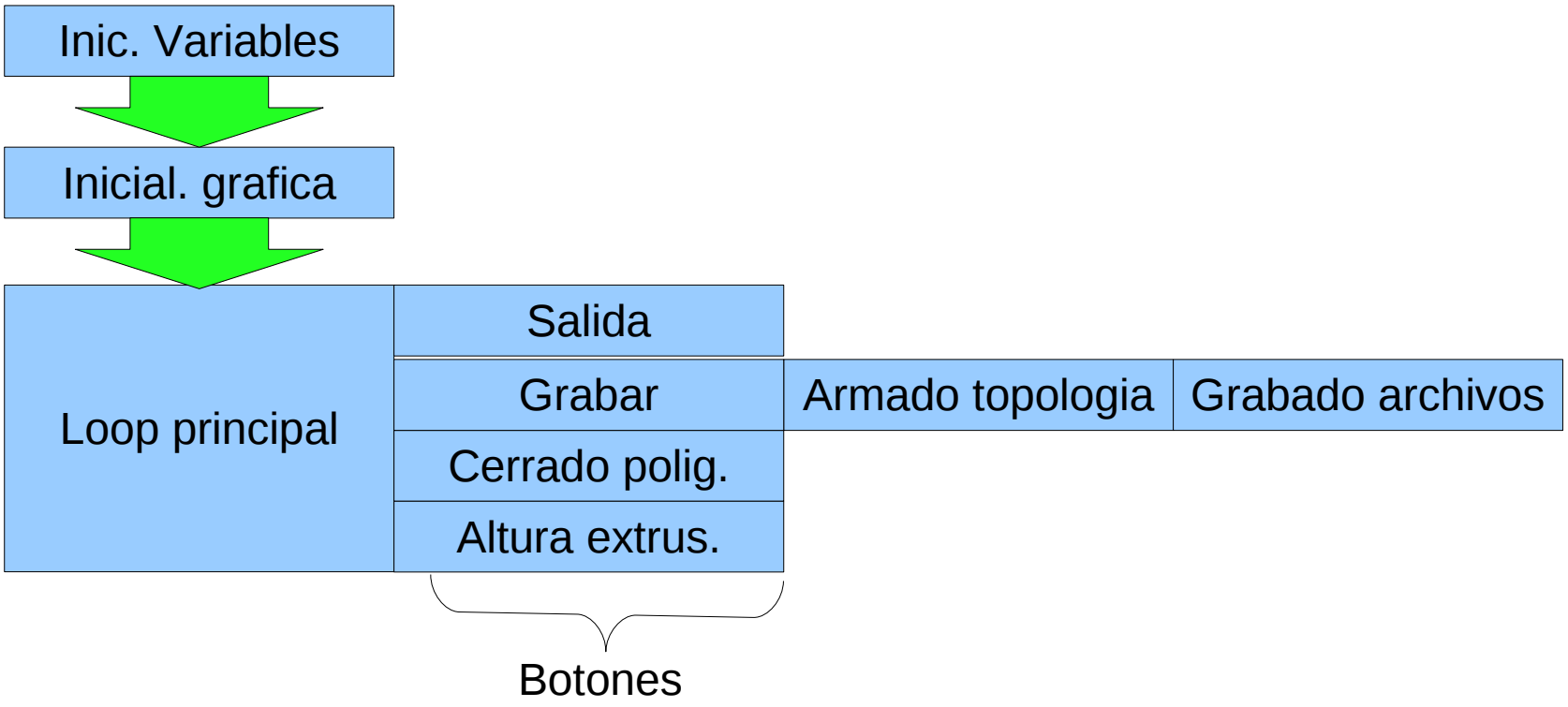


Diagram. bloques de mouse.py



Interfase usuario caras7_1.py

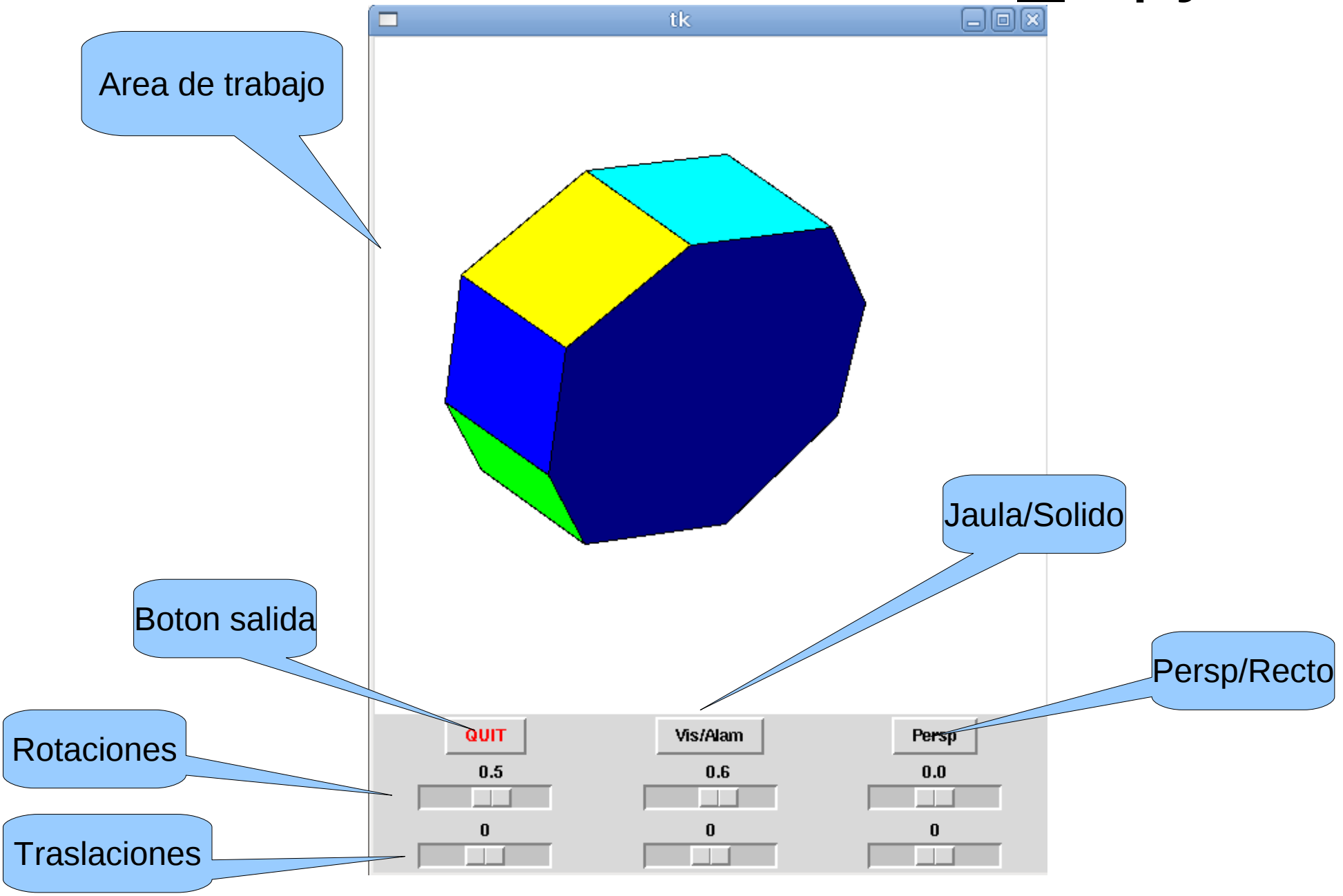
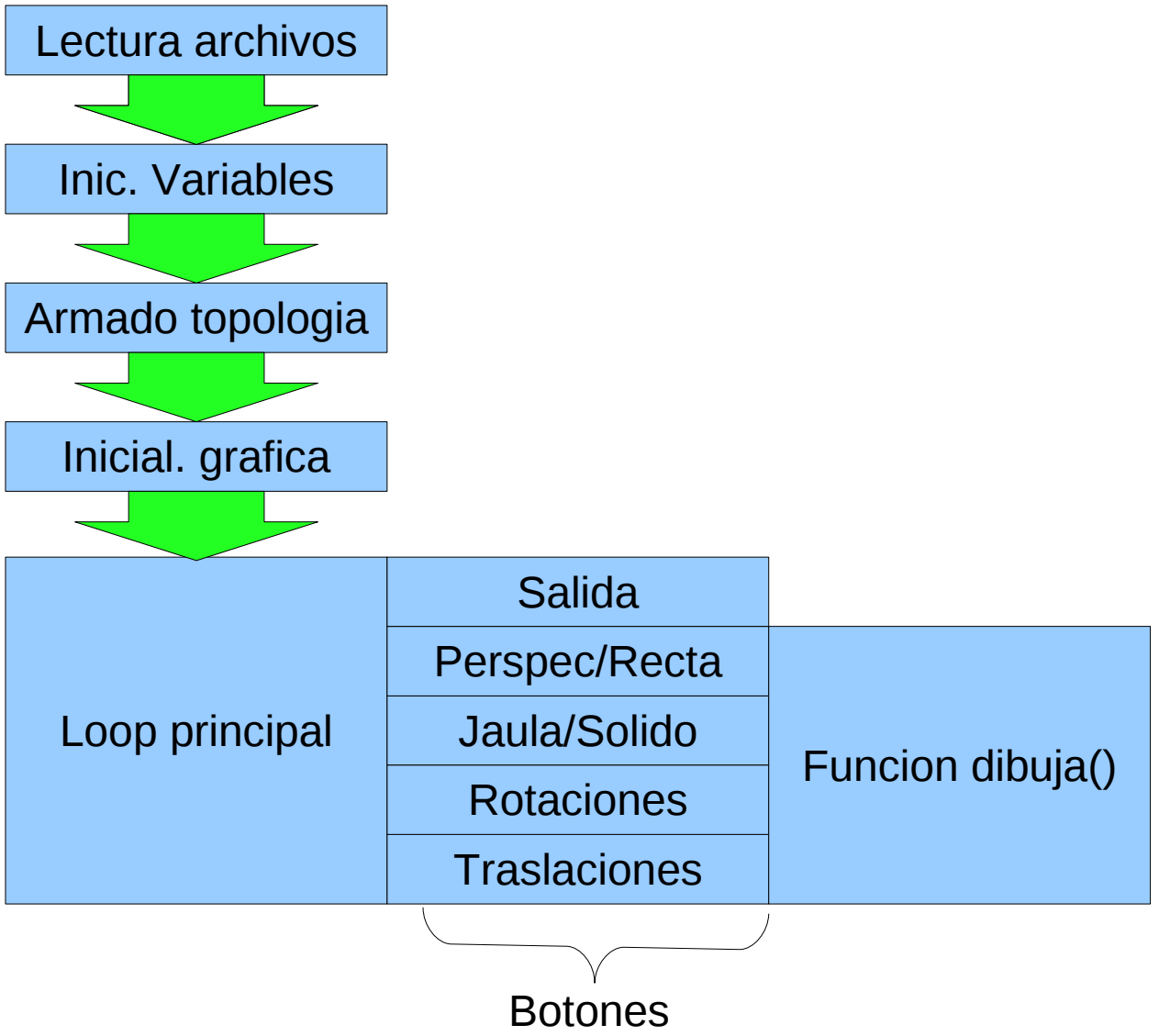


Diagram bloques de caras7_1.py



Próximos objetivos

- Caras_7 rota la figura y traslada sobre un sistema de coordenadas locales, hace falta modificar el sistema para mover al punto de vista.
- No soporta polígonos cóncavos (falta un algoritmo mas elaborado para la visibilidad).
- Problemas con el clipping (no maneja limites fuera de la ventana).
- Mouse.py no soporta extrudados no perpendiculares a la base.
- Y la lista continua....