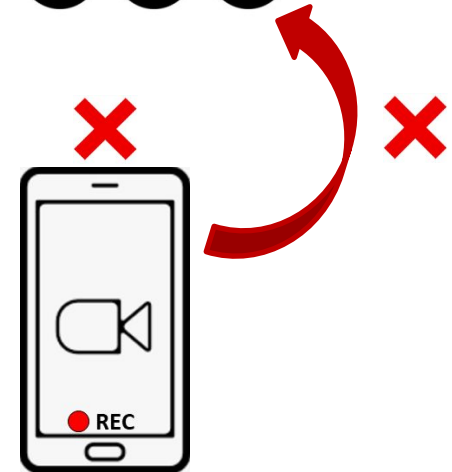
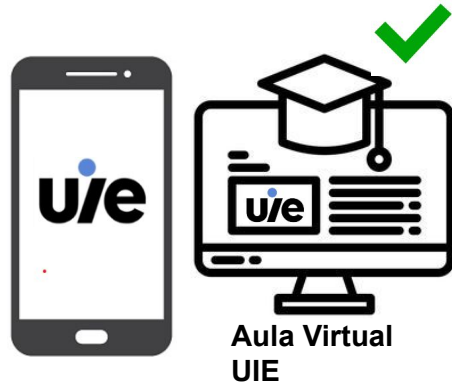
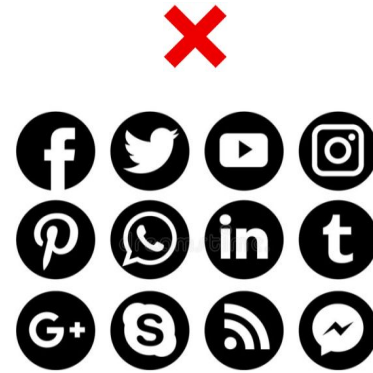
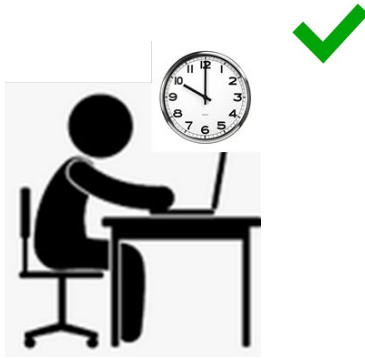


Asignatura

Computación gráfica

Profesor

David Cereijo Graña







Participar

Sesión 09

Unidad III

Gráficos 3D

Tema 3.2 y 3.3

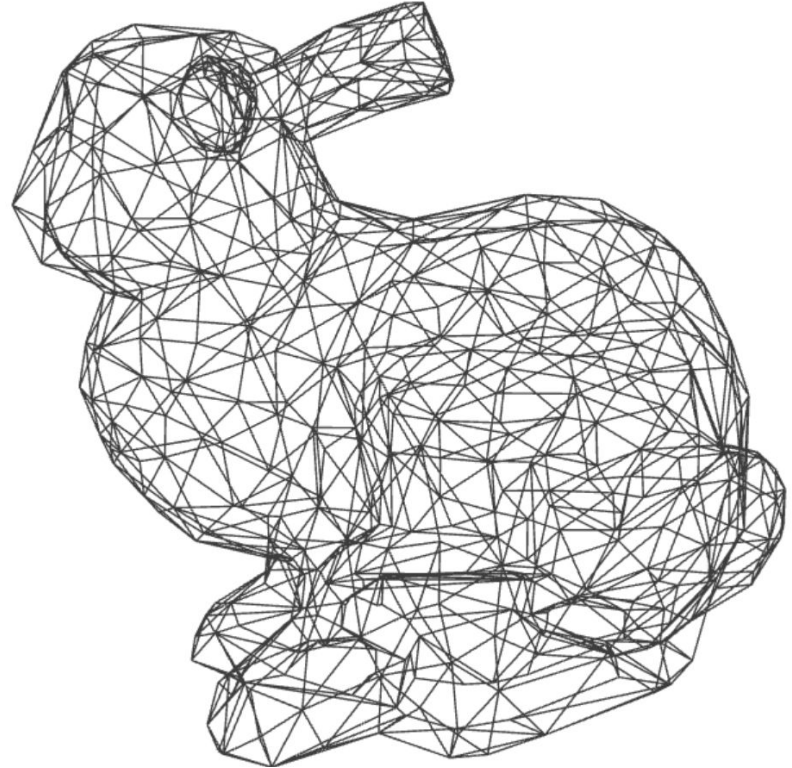
Modelado geométrico 3D y Algoritmos para gráficos 3D

Malla poligonal

Una **malla poligonal** es una red de polígonos conectados entre sí por medio de vértices y aristas compartidas para formar una superficie continua.

Las mallas poligonales surgen de la necesidad de representar objetos tridimensionales complejos, y las más utilizadas en computación gráfica son las **mallas triangulares** (*triangle mesh*).


Las GPU actuales están optimizadas para renderizar de forma eficiente mallas triangulares.



Malla poligonal

La representación de objetos 3D mediante mallas triangulares es relativamente sencilla, pero tiene algunas desventajas:

- **Representación aproximada:** al tratarse de una representación discreta, las superficies curvas se deben representar de forma aproximada, de modo que para conseguir un buen nivel de detalles es necesario utilizar un elevado número de triángulos.
- **Elevada cantidad de datos:** como consecuencia de lo anterior, para representar un modelo de forma detallada, es necesario almacenar y procesar una gran cantidad de datos.

En la siguiente figura podemos ver un mismo objeto  modelado utilizando un número diferente de triángulos:



290 triángulos.



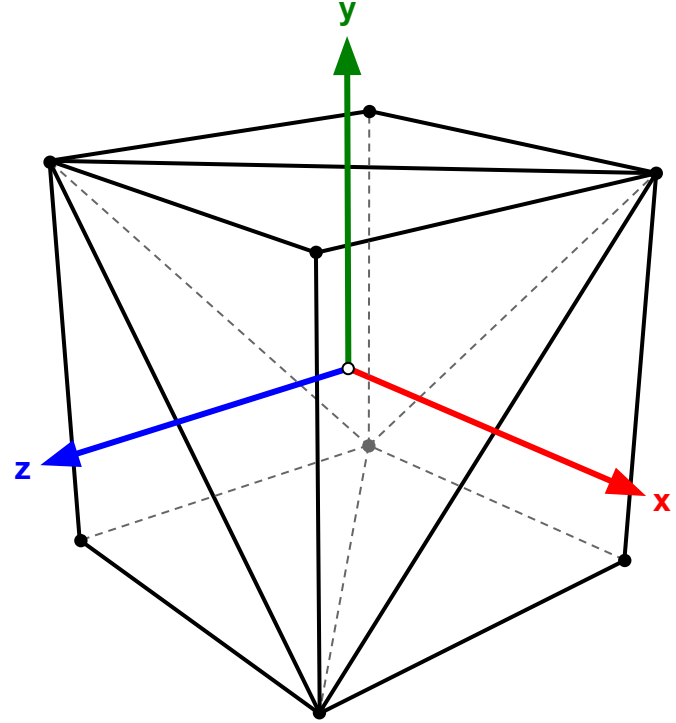
15.744 triángulos.

Descripción de una malla poligonal

La descripción mínima de una malla consiste en definir un conjunto de triángulos y las posiciones de los vértices que los conforman.

No obstante, para poder aplicar color, efectos de iluminación y texturas, es habitual almacenar otros atributos adicionales a la posición en cada vértice, como su color, normal y atributos de textura.

Por tanto, para poder representar una malla necesitamos construir una estructura de datos que permita almacenar esta información de manera eficiente.



Descripción de una malla poligonal

La descripción mínima de una malla consiste en definir un conjunto de triángulos y las posiciones de los vértices que los conforman. No obstante, para poder aplicar color, efectos de iluminación y texturas, es habitual almacenar otros atributos además de la posición en cada vértice, como su color, normal y atributos de textura.

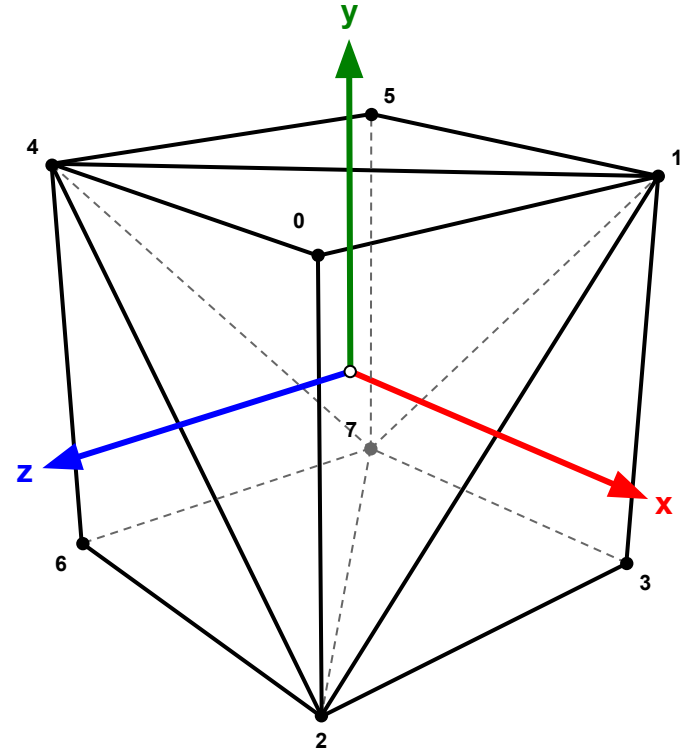
Por tanto, para poder representar una malla necesitamos construir una estructura de datos que permita almacenar esta información de manera eficiente, lo cual se puede hacer de dos grandes formas:

- **Índices compartidos:** cada triángulo se representa mediante tres índices de la lista de vértices, donde cada vértice compartido se representa una sola vez. Esta representación no permite almacenar información
- **Caras independientes:** cada triángulo se representa mediante tres índices una lista de vértices, donde cada vértice se representa una vez para cada triángulo del cual forma parte.

Modelado geométrico de un cubo. Índices compartidos

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	-0,5	-0,5
4	-0,5	+0,5	+0,5
5	-0,5	+0,5	-0,5
6	-0,5	-0,5	+0,5
7	-0,5	-0,5	-0,5

TRIÁNGULOS				
F	T	v_1	v_2	v_3
R	0	0	1	2
	1	1	2	3
F	2	0	2	4
	3	2	4	6
L	4	4	5	7
	5	4	6	7
B	6	1	3	7
	7	1	5	7
U	8	0	1	4
	9	1	4	5
D	10	2	3	7
	11	2	6	7

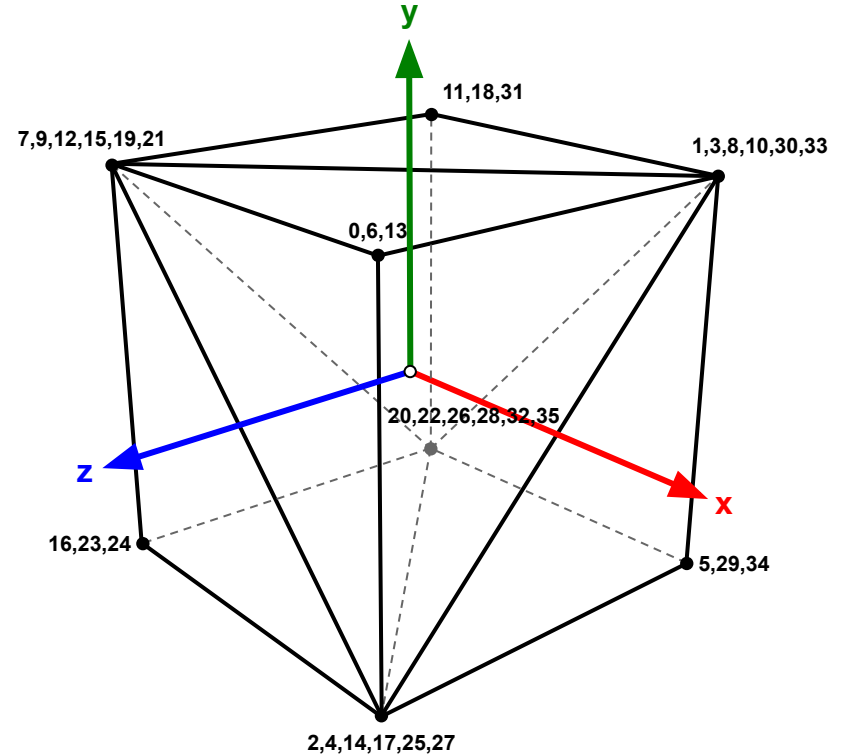


Modelado geométrico de un cubo. Caras independientes

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	+0,5	-0,5	+0,5
17	-0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

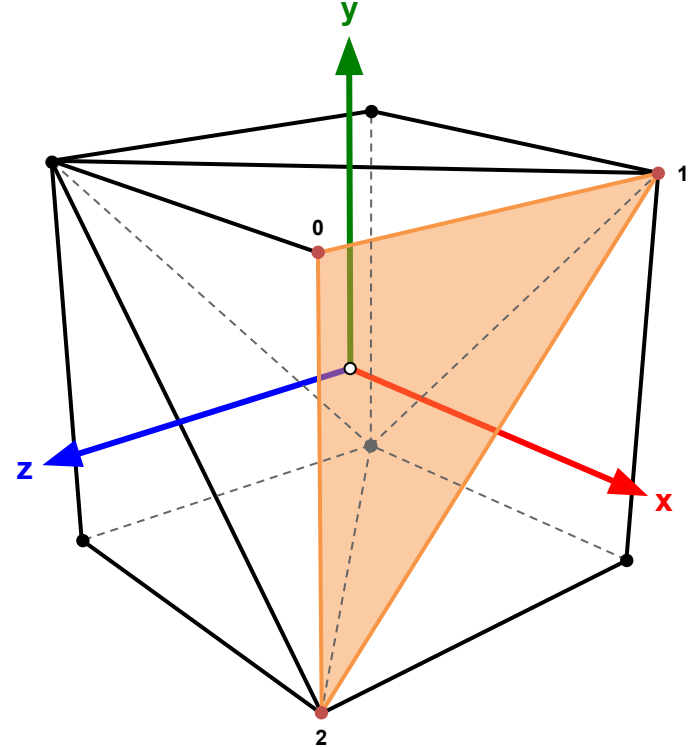


Modelado geométrico de un cubo. Cara R (Right): $x = +0.5$. Triángulo 0

VÉRTICES			
V	x	y	z
0	+0.5	+0.5	+0.5
1	+0.5	+0.5	-0.5
2	+0.5	-0.5	+0.5
3	+0.5	+0.5	-0.5
4	+0.5	-0.5	+0.5
5	+0.5	-0.5	-0.5
6	+0.5	+0.5	+0.5
7	-0.5	+0.5	+0.5
8	+0.5	+0.5	-0.5
9	-0.5	+0.5	+0.5
10	+0.5	+0.5	-0.5
11	-0.5	+0.5	-0.5
12	-0.5	+0.5	+0.5
13	+0.5	+0.5	+0.5
14	+0.5	-0.5	+0.5
15	-0.5	+0.5	+0.5
16	+0.5	-0.5	+0.5
17	-0.5	-0.5	+0.5
18	-0.5	+0.5	-0.5
19	-0.5	+0.5	+0.5
20	-0.5	-0.5	-0.5
21	-0.5	+0.5	+0.5
22	-0.5	-0.5	-0.5
23	-0.5	-0.5	+0.5
24	-0.5	+0.5	+0.5
25	+0.5	-0.5	+0.5
26	-0.5	-0.5	-0.5
27	+0.5	-0.5	+0.5
28	+0.5	-0.5	-0.5
29	-0.5	-0.5	-0.5
30	+0.5	+0.5	-0.5
31	-0.5	+0.5	-0.5
32	-0.5	-0.5	-0.5
33	+0.5	+0.5	-0.5
34	+0.5	-0.5	-0.5
35	-0.5	-0.5	-0.5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

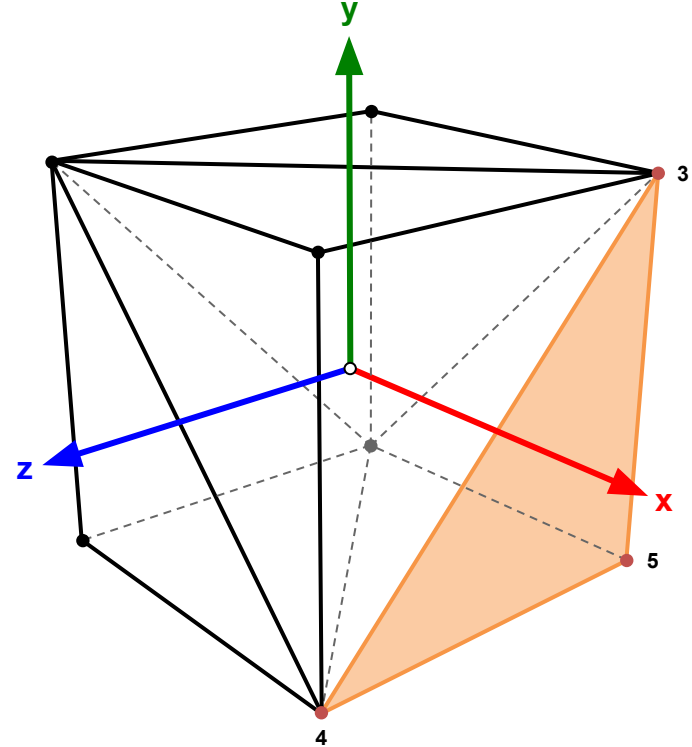


Modelado geométrico de un cubo. Cara R (Right): $x = +0.5$. Triángulo 1

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	+0,5	-0,5	+0,5
17	-0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

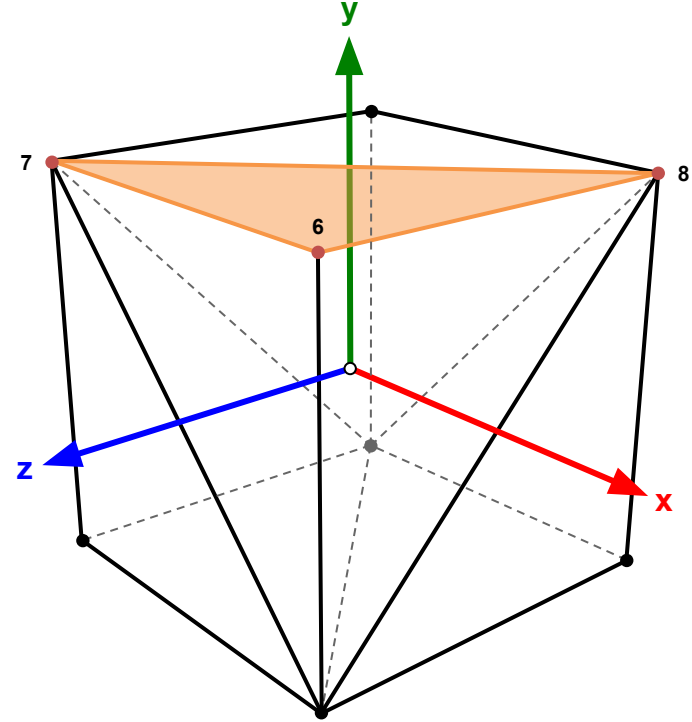


Modelado geométrico de un cubo. Cara U (Up): $y = +0.5$. Triángulo 2

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	+0,5	-0,5	+0,5
17	-0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

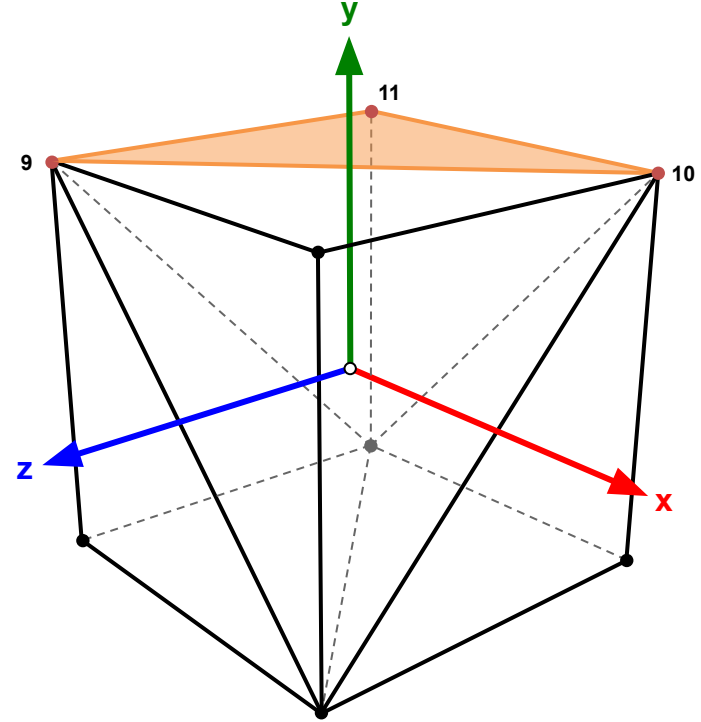


Modelado geométrico de un cubo. Cara U (Up): $y = +0.5$. Triángulo 3

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	-0,5	-0,5	+0,5
17	+0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

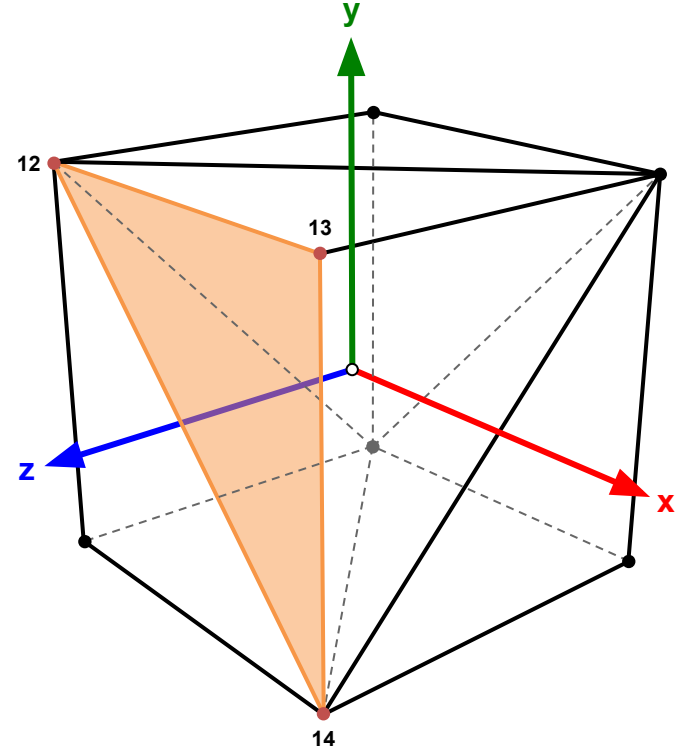


Modelado geométrico de un cubo. Cara F (Front): $z = +0.5$. Triángulo 4

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	-0,5	-0,5	+0,5
17	+0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

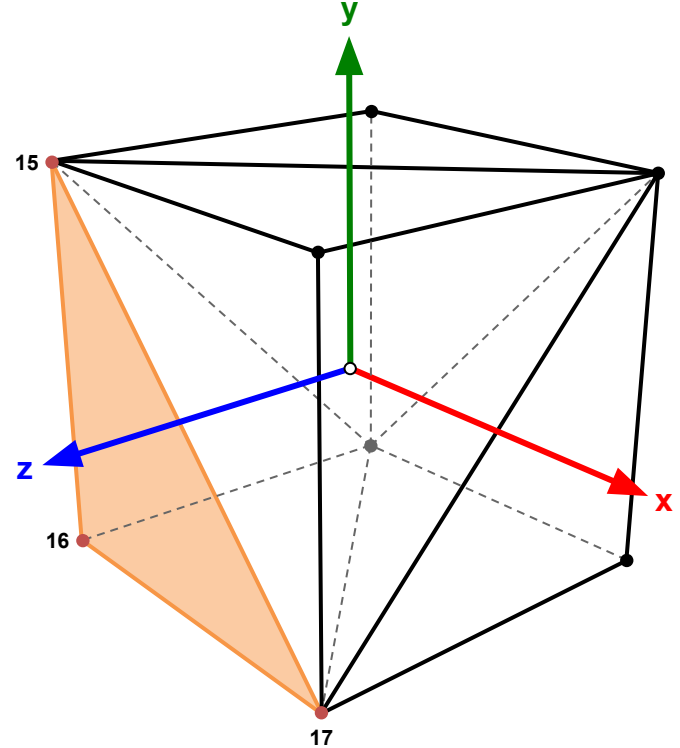


Modelado geométrico de un cubo. Cara F (Front): $z = +0.5$. Triángulo 5

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	-0,5	-0,5	+0,5
17	+0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

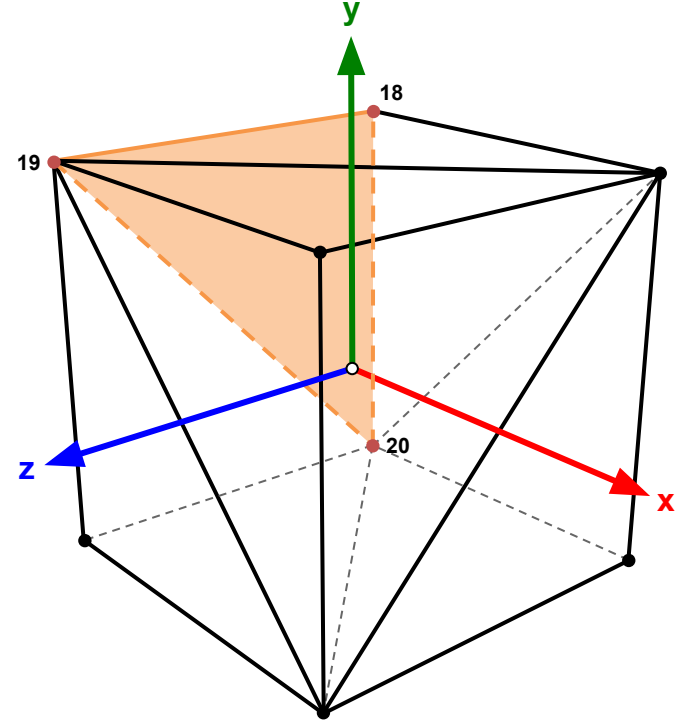


Modelado geométrico de un cubo. Cara L (Left): $x = -0.5$. Triángulo 6

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	-0,5	-0,5	+0,5
17	+0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
U	0	0	1	2
	1	3	4	5
	2	6	7	8
F	3	9	10	11
	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
	10	30	31	32
B	11	33	34	35

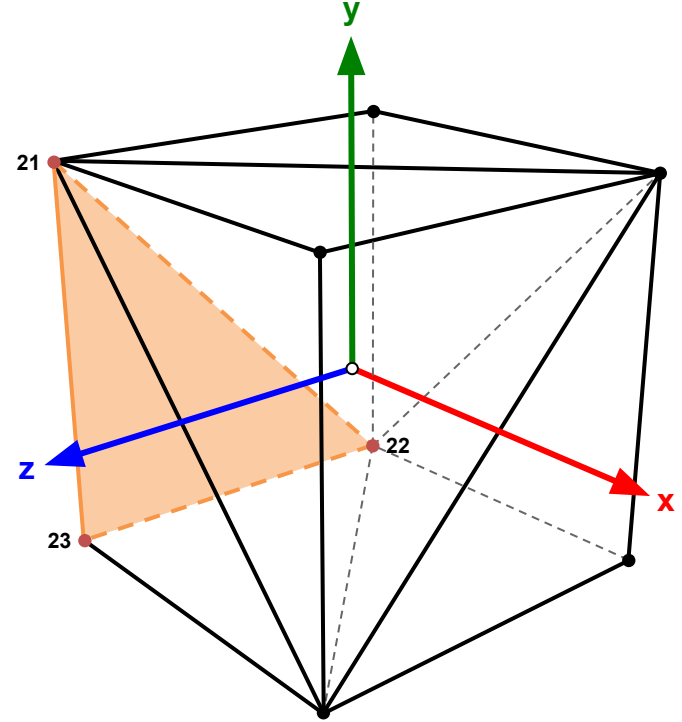


Modelado geométrico de un cubo. Cara L (Left): $x = -0.5$. Triángulo 7

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	-0,5	-0,5	+0,5
17	+0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

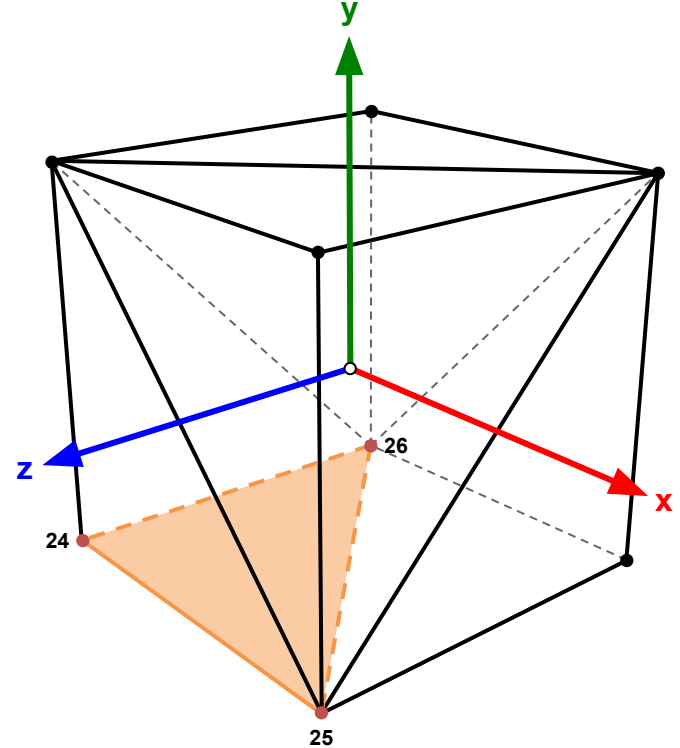


Modelado geométrico de un cubo. Cara D (Down): $y = -0.5$. Triángulo 8

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	-0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	-0,5	-0,5	+0,5
17	+0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	-0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

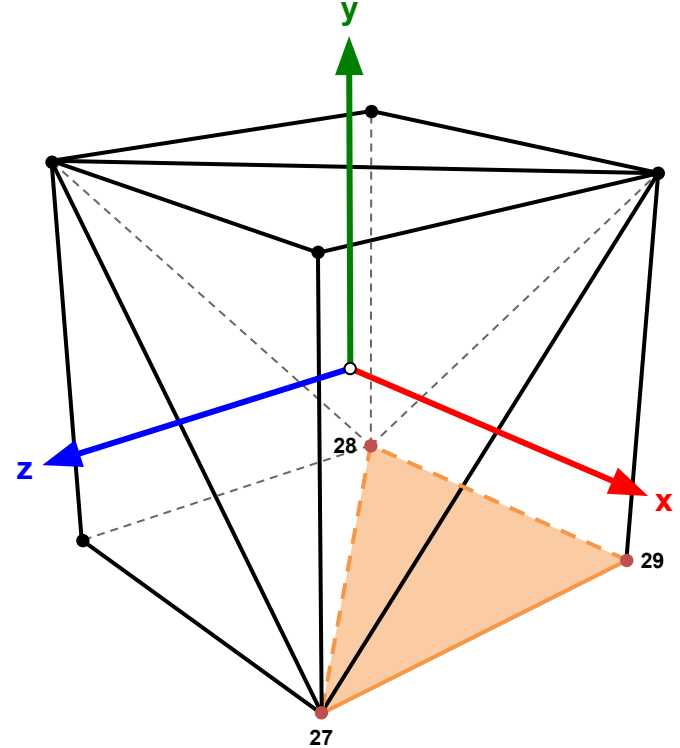


Modelado geométrico de un cubo. Cara D (Down): $y = -0.5$. Triángulo 9

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	-0,5	-0,5	+0,5
17	+0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

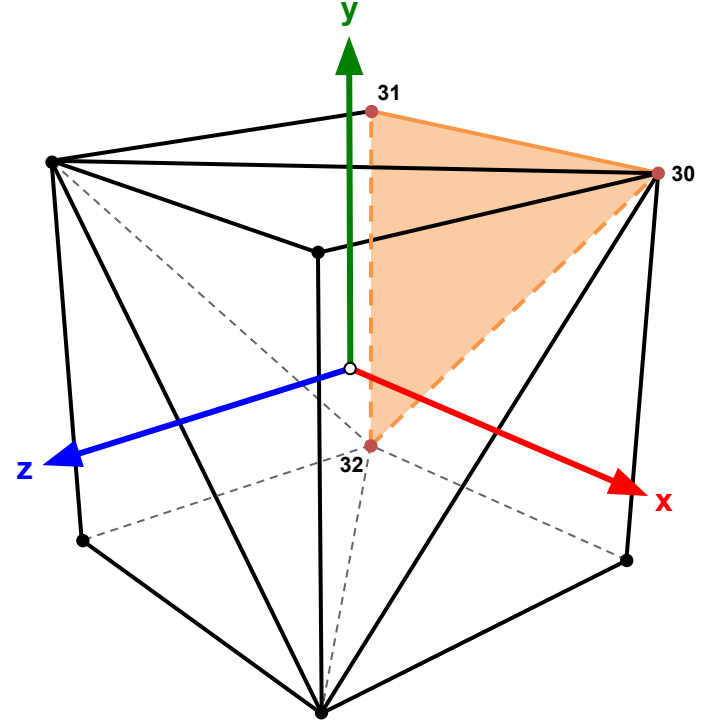


Modelado geométrico de un cubo. Cara B (Back): $z = -0.5$. Triángulo 10

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	+0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	-0,5	-0,5	+0,5
17	+0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35

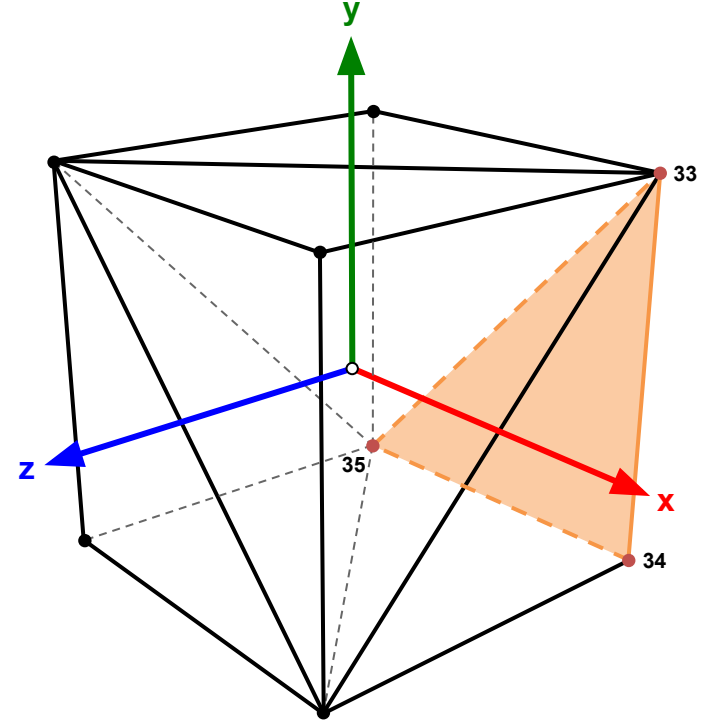


Modelado geométrico de un cubo. Cara B (Back): $z = -0.5$. Triángulo 11

VÉRTICES			
V	x	y	z
0	+0,5	+0,5	+0,5
1	+0,5	+0,5	-0,5
2	+0,5	-0,5	+0,5
3	+0,5	-0,5	-0,5
4	+0,5	-0,5	+0,5
5	+0,5	-0,5	-0,5
6	+0,5	+0,5	+0,5
7	-0,5	+0,5	+0,5
8	+0,5	+0,5	-0,5
9	-0,5	+0,5	+0,5
10	+0,5	+0,5	-0,5
11	-0,5	+0,5	-0,5
12	-0,5	+0,5	+0,5
13	+0,5	+0,5	+0,5
14	+0,5	-0,5	+0,5
15	-0,5	+0,5	+0,5
16	-0,5	-0,5	+0,5
17	+0,5	-0,5	+0,5
18	-0,5	+0,5	-0,5
19	-0,5	+0,5	+0,5
20	-0,5	-0,5	-0,5
21	-0,5	+0,5	+0,5
22	-0,5	-0,5	-0,5
23	-0,5	-0,5	+0,5
24	-0,5	+0,5	+0,5
25	+0,5	-0,5	+0,5
26	-0,5	-0,5	-0,5
27	+0,5	-0,5	+0,5
28	-0,5	-0,5	-0,5
29	+0,5	-0,5	-0,5
30	+0,5	+0,5	-0,5
31	-0,5	+0,5	-0,5
32	-0,5	-0,5	-0,5
33	+0,5	+0,5	-0,5
34	+0,5	-0,5	-0,5
35	-0,5	-0,5	-0,5

NORMALES			
V	x	y	z
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	0	0	1
15	0	0	1
16	0	0	1
17	0	0	1
18	-1	0	0
19	-1	0	0
20	-1	0	0
21	-1	0	0
22	-1	0	0
23	-1	0	0
24	0	-1	0
25	0	-1	0
26	0	-1	0
27	0	-1	0
28	0	-1	0
29	0	-1	0
30	0	0	-1
31	0	0	-1
32	0	0	-1
33	0	0	-1
34	0	0	-1
35	0	0	-1

TRIÁNGULOS				
F	T	v1	v2	v3
R	0	0	1	2
	1	3	4	5
U	2	6	7	8
	3	9	10	11
F	4	12	13	14
	5	15	16	17
L	6	18	19	20
	7	21	22	23
D	8	24	25	26
	9	27	28	29
B	10	30	31	32
	11	33	34	35



Fichero cubo_vertices_compartidos.py

cubo_vertices_compartidos.py

```
7  from OpenGL.GL import * # Importa las funciones de OpenGL necesarias para renderizar
8
9  # Definición de los vértices únicos del cubo, cada uno representado por coordenadas (x, y, z)
10 vertices = [
11     (+0.5, +0.5, +0.5), # Vértice 0
12     (+0.5, +0.5, -0.5), # Vértice 1
13     (+0.5, -0.5, +0.5), # Vértice 2
14     (+0.5, -0.5, -0.5), # Vértice 3
15     (-0.5, +0.5, +0.5), # Vértice 4
16     (-0.5, +0.5, -0.5), # Vértice 5
17     (-0.5, -0.5, +0.5), # Vértice 6
18     (-0.5, -0.5, -0.5), # Vértice 7
19 ]
```

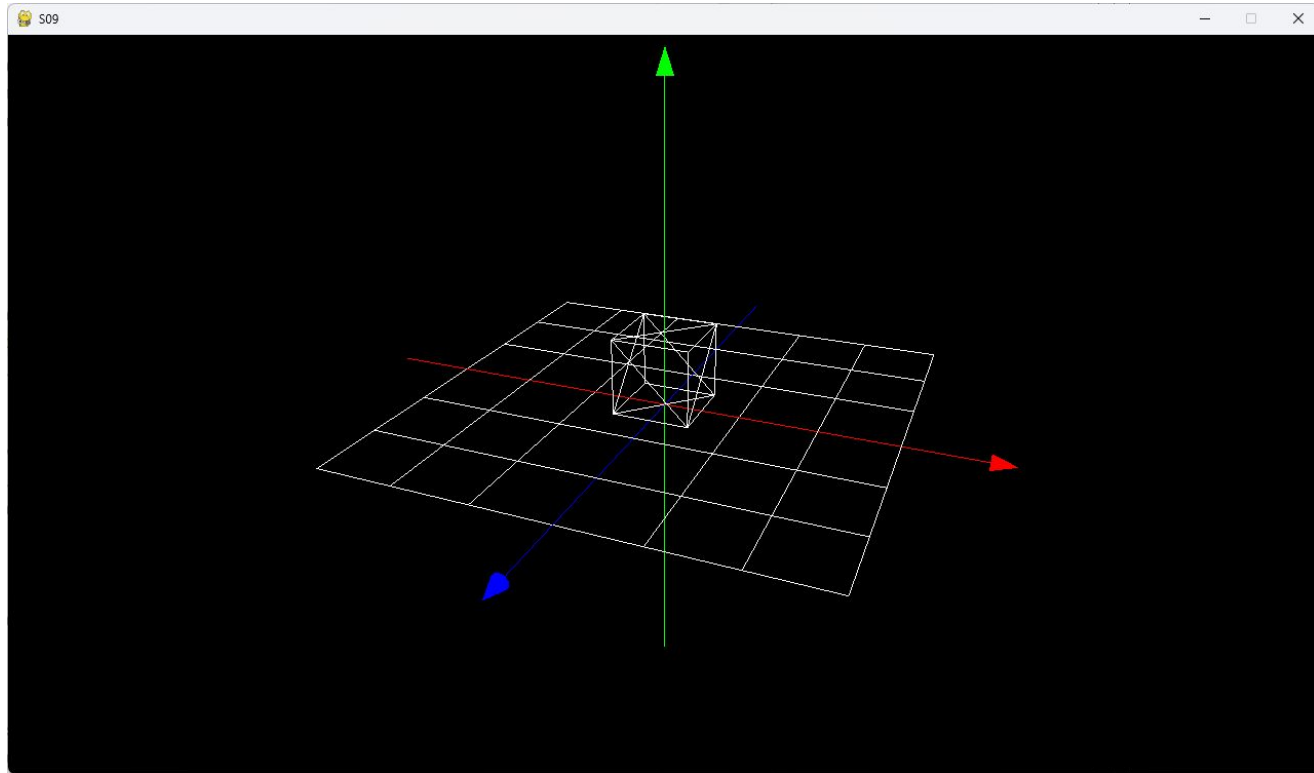
cubo_vertices_compartidos.py

```
21 # Definición de triángulos.
22 triangulos = [
23     # Cara R (Right): x = +0.5
24     [0, 1, 2],
25     [3, 2, 1],
26     # Cara U (Up): y = +0.5
27     [0, 4, 1],
28     [5, 1, 4],
29     # Cara F (Front): z = +0.5
30     [0, 2, 4],
31     [6, 4, 2],
32     # Cara L (Left): x = -0.5
33     [4, 5, 6],
34     [7, 6, 5],
35     # Cara D (Down): y = -0.5
36     [2, 3, 6],
37     [7, 6, 3],
38     # Cara B (Back): z = -0.5
39     [1, 5, 3],
40     [7, 3, 5],
41 ]
```

cubo_vertices_compartidos.py

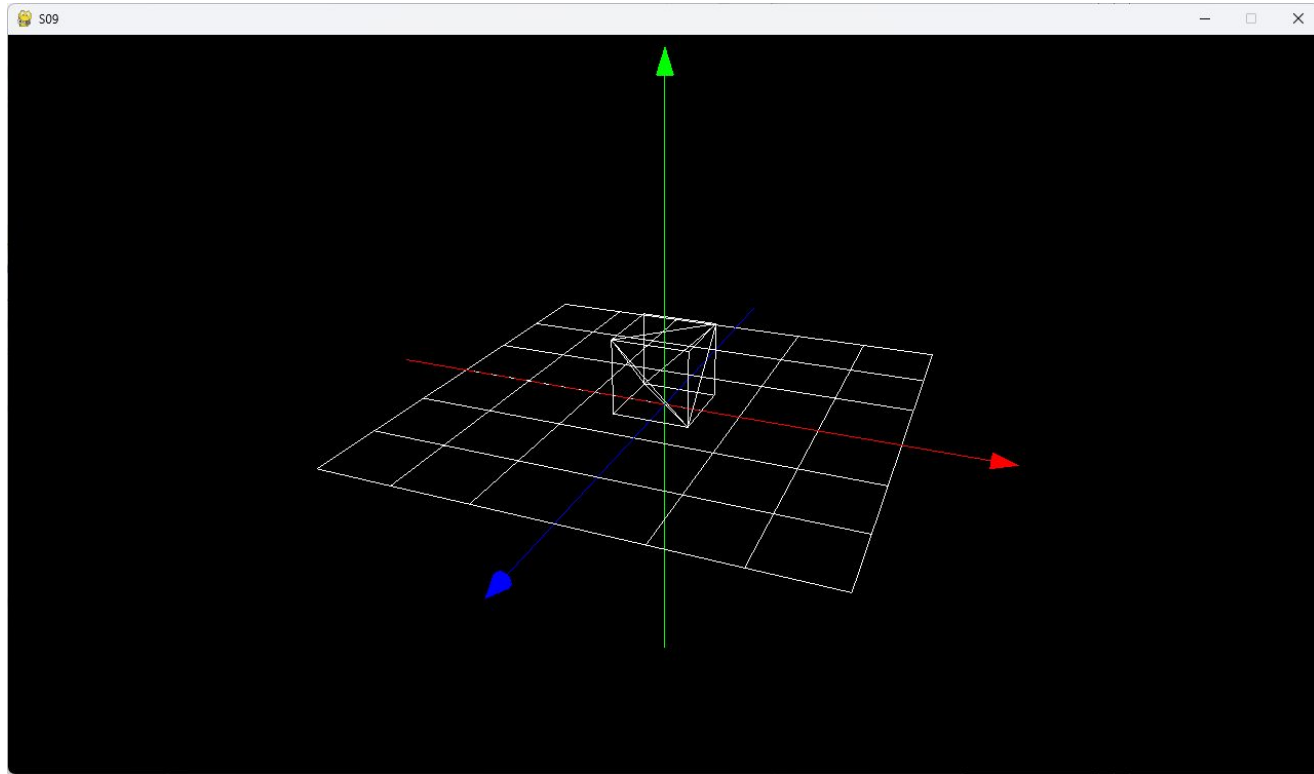
```
43 def cubo():
44     """Dibuja el modelo del cubo utilizando líneas para cada triángulo.
45
46     Recorre la lista de triángulos y dibuja cada uno en el espacio 3D utilizando
47     la función GL_LINE_LOOP de OpenGL, que conecta los tres vértices de cada triángulo
48     con líneas para delinear la estructura del cubo.
49     """
50     for triangulo in triangulos:
51         glBegin(GL_LINE_LOOP) # Inicia el modo de renderizado de líneas en bucle
52         glVertex3fv(vertices[triangulo[0]]) # Primer vértice del triángulo
53         glVertex3fv(vertices[triangulo[1]]) # Segundo vértice del triángulo
54         glVertex3fv(vertices[triangulo[2]]) # Tercer vértice del triángulo
55         glEnd() # Finaliza el dibujo del triángulo actual
```

cubo_vertices_compartidos.py



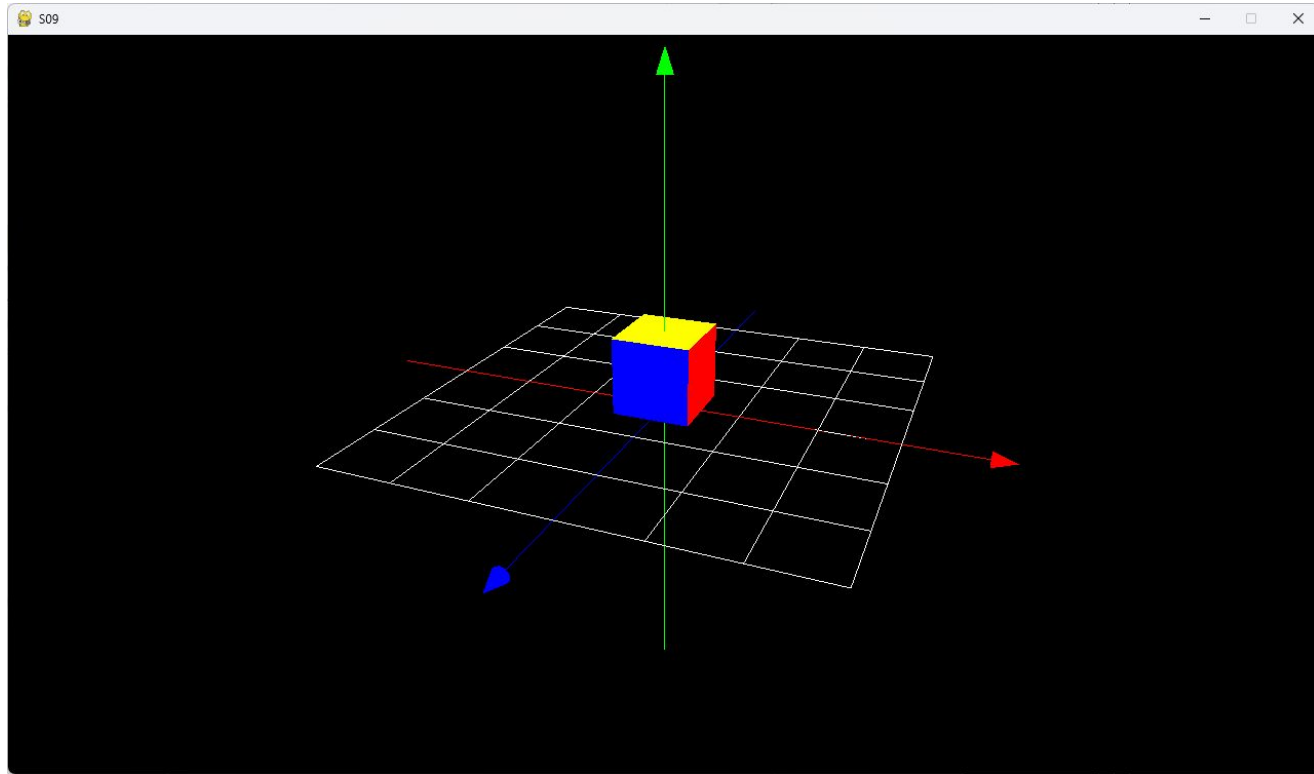
Fichero cubo_caras_independientes.py

cubo_vertices_compartidos.py



Fichero cubo_caras_independientes_color.py

cubo_vertices_compartidos.py



Fichero main.py

main.py

```
19 # Crea una instancia de la cámara con valores iniciales.
20 camara = Camara()
21
22 # Inicialización de la escena.
23 def inicializar_escena(): 1usage
24     """Inicializa la ventana de Pygame y configura los ajustes de OpenGL para la escena 3D.
25
26     Returns:
27         screen: Objeto de la ventana gráfica creada por Pygame.
28     """
29     pygame.init() # Inicializa Pygame
30     screen = pygame.display.set_mode( size: (SCREEN_WIDTH, SCREEN_HEIGHT), DOUBLEBUF | OPENGGL) # Crea la ventana
31     pygame.display.set_caption('S09') # Establece el título de la ventana
32
33     # Configuración de OpenGL
34     glClearColor( red: 0, green: 0, blue: 0, alpha: 1) # Fondo negro
35     glEnable(GL_DEPTH_TEST) # Activa el z-buffer para la profundidad
36     glMatrixMode(GL_PROJECTION) # Selecciona la matriz de proyección
37     gluPerspective(FOV, SCREEN_ASPECT_RATIO, NEAR_PLANE, FAR_PLANE) # Configura la proyección en perspectiva
38     glMatrixMode(GL_MODELVIEW) # Selecciona la matriz de modelo-vista
39     glLoadIdentity() # Restablece la matriz a la identidad
40     return screen # Devuelve la referencia a la ventana creada
```

main.py

```
42 # Dibuja un cubo en una posición específica con transformaciones opcionales
43 def dibujar_cubo(t_x=0.0, t_y=0.0, t_z=0.0, angulo=0.0, eje_x=0, eje_y=0, eje_z=1.0, sx=1.0, sy=1.0, sz=1.0):
44     """Dibuja un cubo aplicando transformaciones de traslación, rotación y escalado.
45
46     Args:
47         t_x, t_y, t_z (float): Posición de traslación en los ejes X, Y, Z.
48         angulo (float): Ángulo de rotación.
49         eje_x, eje_y, eje_z (float): Ejes de rotación.
50         sx, sy, sz (float): Factores de escalado en X, Y, Z.
51     """
52     transformar(t_x, t_y, t_z, angulo, eje_x, eje_y, eje_z, sx, sy, sz, cubo)
53
54 def dibujar_objetos(): 1usage
55     """Dibuja todos los objetos de la escena."""
56     dibujar_cubo(t_y=0.5) # Dibuja un cubo centrado y elevado ligeramente en el eje Y
```

main.py

```
58 # Renderiza la escena
59 def renderizar():
60     """Renderiza los elementos de la escena, incluyendo la cámara, ejes, rejilla y objetos."""
61     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) # Limpia los buffers de color y profundidad
62     glLoadIdentity() # Restablece la matriz a la identidad
63
64     # Configura la posición y orientación de la cámara
65     glRotatef(camara.roll, x: 0, y: 0, z: 1) # Aplica rotación en el eje Z según el roll de la cámara
66     cam_x, cam_y, cam_z = camara.obtener_posicion() # Obtiene la posición de la cámara
67     gluLookAt(cam_x, cam_y, cam_z, 0, 0, 0, 0, 1, 0) # Configura la cámara para observar la escena
68
69     # Dibuja elementos auxiliares y objetos en la escena
70     dibujar_elementos_auxiliares(ejes=True, rejilla=True) # Dibuja ejes y rejilla
71     dibujar_objetos() # Dibuja los objetos
```

main.py

```
73     # Configuración e inicialización del entorno
74     screen = inicializar_escena() # Crea la ventana y configura la escena
75     clock = pygame.time.Clock()  # Inicializa el reloj de Pygame para medir el tiempo entre fotogramas
76     ejecutando = True # Define la bandera que controla el bucle de renderizado
```

main.py

```
78 # Bucle principal de la aplicación
79 while ejecutando:
80     # Calcula el tiempo transcurrido en segundos para suavizar los movimientos
81     delta_time = clock.tick(FPS) / MILLISECONDS_PER_SECOND
82
83     # Procesa los eventos de Pygame, incluyendo el cierre de la ventana y eventos de ratón
84     for evento in pygame.event.get():
85         if evento.type == pygame.QUIT:
86             ejecutando = False # Termina el bucle si se cierra la ventana
87         elif evento.type in (pygame.MOUSEBUTTONDOWN, pygame.MOUSEBUTTONUP, pygame.MOUSEMOTION, pygame.MOUSEWHEEL):
88             procesar_eventos_raton(evento, camara) # Procesa los eventos de ratón para controlar la cámara
89
90     # Consulta el estado del teclado y ajusta la cámara en consecuencia
91     consultar_estado_teclado(camara, delta_time)
92
93     # Actualiza la posición y orientación de la cámara
94     camara.actualizar_camara()
95     renderizar() # Renderiza todos los elementos de la escena en el espacio 3D
96     pygame.display.flip() # Intercambia los buffers para actualizar el fotograma en pantalla
```

Fichero usuario.py

usuario.py

```
10 boton_izquierdo_presionado = False # Bandera que indica si el botón izquierdo está presionado
11 ultimo_x, ultimo_y = 0, 0 # Última posición del ratón en los ejes X e Y.
```


usuario.py

```
13 def procesar_eventos_raton(evento, camara): 8 usages
14     global boton_izquierdo_presionado, ultimo_x, ultimo_y
15
16     # Verifica si se ha presionado el botón izquierdo del ratón
17     if evento.type == pygame.MOUSEBUTTONDOWN and evento.button == BOTON_IZQUIERDO_RATON:
18         boton_izquierdo_presionado = True # Activa el seguimiento del movimiento
19         ultimo_x, ultimo_y = evento.pos # Guarda la posición inicial del clic
20
21     # Verifica si se ha liberado el botón izquierdo del ratón
22     elif evento.type == pygame.MOUSEBUTTONUP and evento.button == BOTON_IZQUIERDO_RATON:
23         boton_izquierdo_presionado = False # Detiene el seguimiento del movimiento
24
25     # Verifica si el ratón se mueve mientras el botón izquierdo está presionado
26     elif evento.type == pygame.MOUSEMOTION and boton_izquierdo_presionado:
27         # Calcula el desplazamiento del ratón en ambos ejes
28         dx, dy = evento.pos[0] - ultimo_x, evento.pos[1] - ultimo_y
29         # Ajusta la rotación horizontal (yaw) en función del desplazamiento en X
30         camara.ajustar_yaw(dx * SENSIBILIDAD_ROTACION * INVERTIR_CONTROLES)
31         # Ajusta la rotación vertical (pitch) en función del desplazamiento en Y
32         camara.ajustar_pitch(-dy * SENSIBILIDAD_ROTACION * INVERTIR_CONTROLES)
33         # Actualiza la última posición del ratón
34         ultimo_x, ultimo_y = evento.pos
35
36     # Verifica si la rueda del ratón se ha movido
37     elif evento.type == pygame.MOUSEWHEEL:
38         # Ajusta el zoom de la cámara en función del movimiento de la rueda
39         camara.ajustar_radio(-evento.y * SENSIBILIDAD_ZOOM * INVERTIR_CONTROLES, RADIO_MIN, RADIO_MAX)
```

usuario.py

```

55 def consultar_estado_teclado(camara, delta_time): #usages
56     keys = pygame.key.get_pressed() # Obtiene el estado actual de todas las teclas
57
58     # Calcula las velocidades de rotación y zoom, sensibles al tiempo transcurrido (delta_time)
59     velocidad_rotacion = VELOCIDAD_ROTACION * delta_time * INVERTIR_CONTROLES
60     velocidad_vertical = VELOCIDAD_ROTACION * delta_time
61     velocidad_zoom = VELOCIDAD_ZOOM * delta_time * INVERTIR_CONTROLES
62
63     # Control de inclinación (pitch) vertical hacia arriba.
64     if keys[K_UP]:
65         camara.ajustar_pitch(-velocidad_vertical)
66
67     # Control de inclinación (pitch) vertical hacia abajo.
68     if keys[K_DOWN]:
69         camara.ajustar_pitch(velocidad_vertical)
70
71     # Control de rotación horizontal (yaw) hacia la izquierda.
72     if keys[K_LEFT]:
73         camara.ajustar_yaw(-velocidad_rotacion)
74
75     # Control de rotación horizontal (yaw) hacia la derecha.
76     if keys[K_RIGHT]:
77         camara.ajustar_yaw(velocidad_rotacion)

```

```

79     # Control de zoom (radio) para alejar la cámara.
80     if keys[K_q]:
81         camara.ajustar_radio(-velocidad_zoom, RADIO_MIN, RADIO_MAX)
82
83     # Control de zoom (radio) para acercar la cámara.
84     if keys[K_a]:
85         camara.ajustar_radio(velocidad_zoom, RADIO_MIN, RADIO_MAX)
86
87     # Control del roll de la cámara en sentido horario.
88     if keys[K_e]:
89         camara.ajustar_roll(velocidad_rotacion)
90
91     # Control del roll de la cámara en sentido antihorario.
92     if keys[K_d]:
93         camara.ajustar_roll(-velocidad_rotacion)
94
95     # Restablece la cámara a su estado inicial.
96     if keys[K_r]:
97         camara.reset()

```

Fichero camara.py

Cámara

La cámara es un objeto más en la escena y, aunque no lo podamos ver, estamos mirando a través de él, y lo podemos trasladar y rotar del mismo modo que cualquier otro objeto. Por tanto, moviendo la cámara y apuntándola en una dirección particular, podemos ver la escena desde cualquier punto.

OpenGL permite hacer esto con el comando **gluLookAt()**.

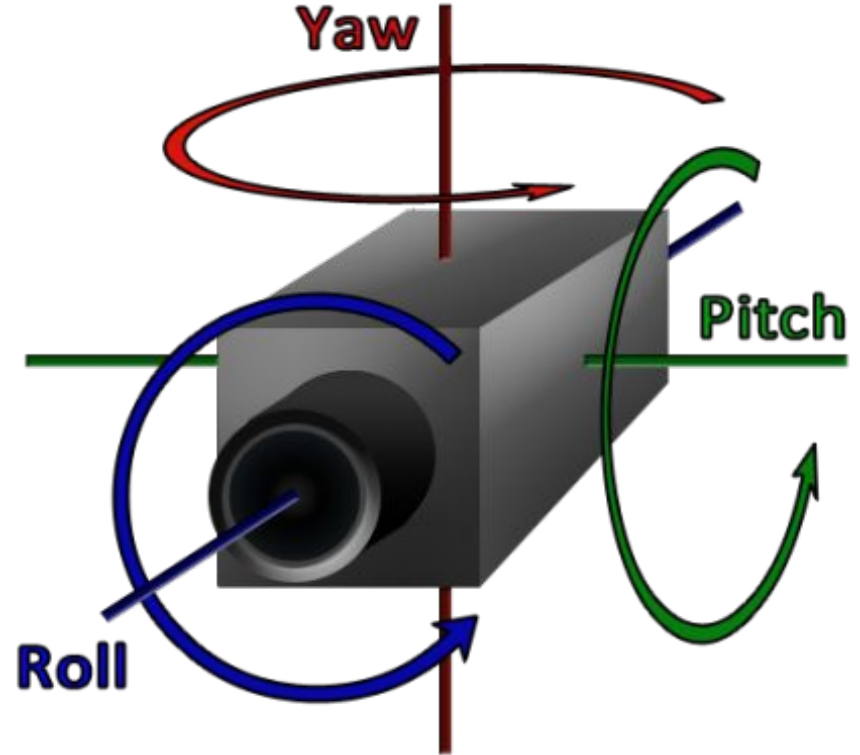
gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ):

- **(eyeX, eyeY, eyeZ):** especifica la posición del ojo u observador.
- **(centerX, centerY, centerZ):** especifica la posición del punto de referencia, hacia el cual apunta la cámara.
- **(upX, upY, upZ):** especifica la dirección del vector “up”, es decir, la dirección vertical de la cámara en el espacio.

Cámara

En el contexto de la computación gráfica **3D**, la orientación de la cámara queda descrita por tres ángulos denominados **yaw**, **pitch** y **roll**:

- **Yaw (guiñada)**: es el ángulo de rotación horizontal, en torno al eje vertical, **Y**, lo que hace que la cámara mire hacia la izquierda o hacia la derecha.
- **Pitch (inclinación)**: es el ángulo de inclinación vertical, en torno al eje **X**, lo que mueve la vista hacia arriba o hacia abajo, cambiando el ángulo de observación.
- **Roll (alabeo)**: es el ángulo de rotación lateral, en torno al eje **Z**, lo que gira la vista como se inclinara hacia un lado.



camara.py

```
5  MIN_PITCH = -90 # Ángulo mínimo permitido para la inclinación (pitch) de la cámara
6  MAX_PITCH = 90  # Ángulo máximo permitido para la inclinación (pitch) de la cámara
7
8  class Camara: 16 usages
9      """Clase que representa una cámara en un entorno 3D.
10
11      La cámara permite controlar su orientación en los 3 ejes (pitch, yaw y roll),
12      así como su distancia (radio) respecto al punto de observación en el espacio.
13      """
```

camara.py

```

15 def __init__(self, pitch=0, yaw=0, roll=0, radio=10.0):
16     """Inicializa una instancia de la clase Camara con valores predeterminados.
17
18     Args:
19         pitch (float): Ángulo inicial de inclinación de la cámara.
20         yaw (float): Ángulo inicial de rotación horizontal de la cámara.
21         roll (float): Ángulo inicial de rotación lateral de la cámara.
22         radio (float): Distancia inicial de la cámara respecto al punto de origen.
23     """
24     # Almacena los valores iniciales para poder reiniciarlos si es necesario
25     self.pitch_inicial = pitch
26     self.yaw_inicial = yaw
27     self.roll_inicial = roll
28     self.radio_inicial = radio
29
30     # Asigna los valores actuales de pitch, yaw, roll y radio
31     self.pitch = pitch
32     self.yaw = yaw
33     self.roll = roll
34     self.radio = radio
35
36     # Establece la posición inicial de la cámara en el espacio 3D
37     self.cam_x, self.cam_y, self.cam_z = 0, 0, radio

```

camara.py

```
39     def actualizar_camara(self): 9 usages
40         """Actualiza la posición de la cámara en el espacio en función de pitch, yaw y radio.
41
42         Calcula las coordenadas de la cámara en el espacio 3D usando trigonometría,
43         basándose en los ángulos actuales y el radio de distancia.
44         """
45         self.cam_x = self.radio * np.sin(np.radians(self.yaw)) * np.cos(np.radians(self.pitch))
46         self.cam_y = self.radio * np.sin(np.radians(self.pitch))
47         self.cam_z = self.radio * np.cos(np.radians(self.yaw)) * np.cos(np.radians(self.pitch))
```


camara.py

```
49     def obtener_posicion(self): 8 usages
50         """Devuelve la posición actual de la cámara.
51
52         Returns:
53             tuple: Coordenadas (cam_x, cam_y, cam_z) que representan la posición en el espacio 3D.
54         """
55         return self.cam_x, self.cam_y, self.cam_z
```

camara.py

```
57     def ajustar_pitch(self, incremento): 3 usages (3 dynamic)
58         """Ajusta el ángulo de inclinación (pitch) dentro de los límites establecidos.
59
60         Args:
61             incremento (float): Valor a sumar al pitch actual, limitado entre MIN_PITCH y MAX_PITCH.
62         """
63         self.pitch = np.clip(self.pitch + incremento, MIN_PITCH, MAX_PITCH)
```

camara.py

```
65     def ajustar_yaw(self, incremento): 3 usages (3 dynamic)
66         """Ajusta el ángulo de rotación horizontal (yaw) sin límites específicos.
67
68         Args:
69             incremento (float): Valor a sumar al yaw actual.
70         """
71         self.yaw += incremento
```

camara.py

```
73     def ajustar_roll(self, incremento): 2 usages (2 dynamic)
74         """Ajusta el ángulo de rotación lateral (roll) de la cámara.
75
76         Args:
77             incremento (float): Valor a sumar al roll actual.
78         """
79         self.roll += incremento
```

camara.py

```
81     def ajustar_radio(self, incremento, min_radio, max_radio): 3 usages (3 dynamic)
82         """Ajusta la distancia (radio) de la cámara dentro de límites permitidos.
83
84         Args:
85             incremento (float): Valor a sumar al radio actual.
86             min_radio (float): Límite mínimo permitido para el radio.
87             max_radio (float): Límite máximo permitido para el radio.
88         """
89         self.radio = np.clip(self.radio + incremento, min_radio, max_radio)
```

camara.py

```
91     def reset(self): 1 usage (1 dynamic)
92         """Restaura la orientación y distancia de la cámara a sus valores iniciales."""
93         self.pitch = self.pitch_inicial
94         self.yaw = self.yaw_inicial
95         self.roll = self.roll_inicial
96         self.radio = self.radio_inicial
97         self.actualizar_camara()
```

Fichero transformaciones.py

transformaciones.py

```
13 def trasladar(x, y, z): 1usage
14     """Aplica una traslación en el espacio 3D.
15
16     Desplaza un objeto en el espacio 3D en función de las coordenadas especificadas.
17
18     Args:
19         x (float): Desplazamiento en el eje X.
20         y (float): Desplazamiento en el eje Y.
21         z (float): Desplazamiento en el eje Z.
22     """
23     glTranslatef(x, y, z) # Aplica la traslación en el espacio 3D
```


transformaciones.py

```
26 def rotar(angulo, x, y, z): 1 usage
27     """
28     Aplica una rotación en el espacio 3D.
29
30     Rota un objeto en torno a un eje especificado (definido por x, y, z) en el espacio 3D.
31
32     Args:
33         angulo (float): Ángulo de rotación en grados.
34         x (float): Componente X del eje de rotación.
35         y (float): Componente Y del eje de rotación.
36         z (float): Componente Z del eje de rotación.
37     """
38     glRotatef(angulo, x, y, z) # Aplica la rotación en el espacio 3D
```

transformaciones.py

```
41 def escalar(sx, sy, sz): 1usage
42     """Aplica un escalado en el espacio 3D.
43
44     Modifica el tamaño de un objeto en el espacio 3D, escalándolo en los ejes X, Y y Z.
45
46     Args:
47         sx (float): Factor de escalado en el eje X.
48         sy (float): Factor de escalado en el eje Y.
49         sz (float): Factor de escalado en el eje Z.
50     """
51     glScalef(sx, sy, sz) # Aplica el escalado en el espacio 3D
```

transformaciones.py

```
54 def transformar(t_x, t_y, t_z, angulo, eje_x, eje_y, eje_z, sx, sy, sz, objeto): 17 usages
55     glPushMatrix() # Guarda la matriz de transformación actual en la pila
56
57     # Aplica las transformaciones en el orden de traslación, rotación y escalado
58     trasladar(t_x, t_y, t_z)
59     rotar(angulo, eje_x, eje_y, eje_z)
60     escalar(sx, sy, sz)
61
62     # Llama a la función que dibuja el objeto con las transformaciones aplicadas
63     objeto()
64
65     glPopMatrix() # Restaura la matriz de transformación previa de la pila
```

Fichero configuracion.py

configuracion.py

```
6      # Definición de colores
7      COLOR_ROJO      = (1.0, 0.0, 0.0)      # Color rojo primario
8      COLOR_VERDE     = (0.0, 1.0, 0.0)      # Color verde primario
9      COLOR_AZUL      = (0.0, 0.0, 1.0)      # Color azul primario
10     COLOR_AMARILLO   = (1.0, 1.0, 0.0)      # Color amarillo
11     COLOR_NARANJA    = (1.0, 0.5, 0.0)      # Color naranja
12     COLOR_BLANCO     = (1.0, 1.0, 1.0)      # Color blanco
13     COLOR_NEGRO      = (0.0, 0.0, 0.0)      # Color negro
```

configuracion.py

```
15  # Configuración de la ventana gráfica
16  SCREEN_WIDTH      = 800                # Anchura de la ventana gráfica
17  SCREEN_HEIGHT     = 600                # Altura de la ventana gráfica
18  SCREEN_ASPECT_RATIO = SCREEN_WIDTH / SCREEN_HEIGHT # Relación de aspecto de la ventana gráfica
```

configuracion.py

```
20 # Configuración de la proyección
21 FOV          = 45    # (Field Of View) Campo o ángulo de vision desde el centro de la cámara.
22 NEAR_PLANE   = 0.1   # Distancia del plano más cercano a la cámara.
23 FAR_PLANE    = 50    # Distancia del plano más lejano a la cámara.
```

configuracion.py

```
25     # Configuración del bucle de renderizado
26     FPS = 60
27     MILLISECONDS_PER_SECOND = 1000.0    # Número de milisegundos en un segundo.
```


configuracion.py

```
29  # Configuración de la interacción del usuario
30  BOTON_IZQUIERDO_RATON    = 1      # ID del botón izquierdo del ratón.
31  SENSIBILIDAD_ROTACION    = 0.2    # Sensibilidad a la rotación cuando se utiliza el ratón.
32  SENSIBILIDAD_ZOOM        = 0.3    # Sensibilidad al zoom cuando se usa el ratón.
33  RADIO_MAX                 = 15.0   # Distancia máxima de la cámara al origen cuando se hace zoom.
34  RADIO_MIN                 = 1.0    # Distancia mínima de la cámara al origen cuando se hace zoom.
35  INVERTIR_CONTROLES        = -1     # Ajustar a 1 o -1 para invertir el sentido de los controles cuando se rota la escena.
36  VELOCIDAD_ROTACION        = 135    # Velocidad de rotación cuando se usa el teclado, en grados por segundo.
37  VELOCIDAD_ZOOM            = 10     # Velocidad de zoom cuando se usa el teclado, en unidades por segundo.
```

configuracion.py

```
39  # Configuración de los ejes
40  LONGITUD_EJE          = 4          # Longitud de los segmentos de recta que representan los ejes
41  EJE_X_MIN             = -LONGITUD_EJE # Coordenada mínima del eje X
42  EJE_X_MAX             = LONGITUD_EJE  # Coordenada máxima del eje X
43  EJE_Y_MIN             = -LONGITUD_EJE # Coordenada mínima del eje Y
44  EJE_Y_MAX             = LONGITUD_EJE  # Coordenada máxima del eje Y
45  EJE_Z_MIN             = -LONGITUD_EJE # Coordenada mínima del eje Z
46  EJE_Z_MAX             = LONGITUD_EJE  # Coordenada máxima del eje Z
47  COLOR_EJE_X           = COLOR_ROJO   # Color del eje X
48  COLOR_EJE_Y           = COLOR_VERDE  # Color del eje Y
49  COLOR_EJE_Z           = COLOR_AZUL   # Color del eje Z
50  EJE_FLECHA_BASE       = 0.1          # Radio de la base de la flecha
51  EJE_FLECHA_PUNTA      = 0.0          # Radio de la punta de la flecha
52  EJE_FLECHA_LONGITUD   = 0.3          # Longitud de la flecha
53  EJE_FLECHA_REBANADAS  = 10           # Número de subdivisiones en el eje Z de la flecha
54  EJE_FLECHA_PILAS      = 10           # Número de subdivisiones en el eje Y de la flecha
```

configuracion.py

```
56  # Configuración de la rejilla
57  REJILLA_COLOR      =  COLOR_BLANCO      # Color de las líneas de la rejilla
58  REJILLA_TAMANO     =  3                 # Tamaño de la rejilla en cada eje (positivo y negativo)
59  REJILLA_PASO       =  1                 # Tamaño de cada celda de la rejilla
```

Fichero utilidades.py

utilidades.py

```
13 def dibujar_elementos_auxiliares(ejes=False, rejilla=False): 8 usages
14     """Dibuja elementos auxiliares en la escena, como ejes y rejilla.
15
16     Args:
17         ejes (bool): Si es True, dibuja los ejes.
18         rejilla (bool): Si es True, dibuja la rejilla.
19     """
20     if ejes:
21         dibujar_ejes()
22     if rejilla:
23         dibujar_rejilla()
```

utilidades.py

```
26 def dibujar_ejes(): 1 usage
27     """Dibuja los ejes del mundo y sus puntas de flecha en cada extremo."""
28     # Dibuja los ejes en X, Y y Z
29     dibujar_eje_con_flecha(EJE_X_MIN, y1: 0, z1: 0, EJE_X_MAX, y2: 0, z2: 0, COLOR_EJE_X, rotacion=(90, 0, 1, 0))
30     dibujar_eje_con_flecha(x1: 0, EJE_Y_MIN, z1: 0, x2: 0, EJE_Y_MAX, z2: 0, COLOR_EJE_Y, rotacion=(-90, 1, 0, 0))
31     dibujar_eje_con_flecha(x1: 0, y1: 0, EJE_Z_MIN, x2: 0, y2: 0, EJE_Z_MAX, COLOR_EJE_Z)
```

utilidades.py

```
34 def dibujar_eje_con_flecha(x1, y1, z1, x2, y2, z2, color, rotacion=None): 3 usages
35     """Dibuja un eje con una flecha en el extremo positivo.
36
37     Args:
38         x1, y1, z1: Coordenadas de inicio del eje.
39         x2, y2, z2: Coordenadas de fin del eje.
40         color: Color del eje y la flecha.
41         rotacion: Tupla con los parámetros de rotación (angulo, x, y, z) para la flecha.
42     """
43     dibujar_segmento(x1, y1, z1, x2, y2, z2, color)
44     glColor3f(*color)
45     glPushMatrix()
46     glTranslatef(x2, y2, z2) # Posiciona en el extremo positivo del eje
47     if rotacion:
48         glRotatef(*rotacion) # Aplica la rotación necesaria para orientar la flecha
49     dibujar_cono()
50     glPopMatrix()
```

utilidades.py

```

53 def dibujar_segmento(x1, y1, z1, x2, y2, z2, color): 1 usage
54     """Dibuja un segmento de línea en 3D.
55
56     Args:
57         x1, y1, z1 (float): Coordenadas de inicio del segmento.
58         x2, y2, z2 (float): Coordenadas de fin del segmento.
59         color (tuple): Color en formato RGB.
60     """
61     glBegin(GL_LINES)
62     glColor3f(*color)
63     glVertex3f(x1, y1, z1)
64     glVertex3f(x2, y2, z2)
65     glEnd()

```


utilidades.py

```
68     def dibujar_cono(): 1 usage
69         """Dibuja un cono que representa la punta de una flecha de un eje."""
70         cone = gluNewQuadric()
71         gluCylinder(cone,
72                     EJE_FLECHA_BASE,
73                     EJE_FLECHA_PUNTA,
74                     EJE_FLECHA_LONGITUD,
75                     EJE_FLECHA_REBANADAS,
76                     EJE_FLECHA_PILAS)
77
```

utilidades.py

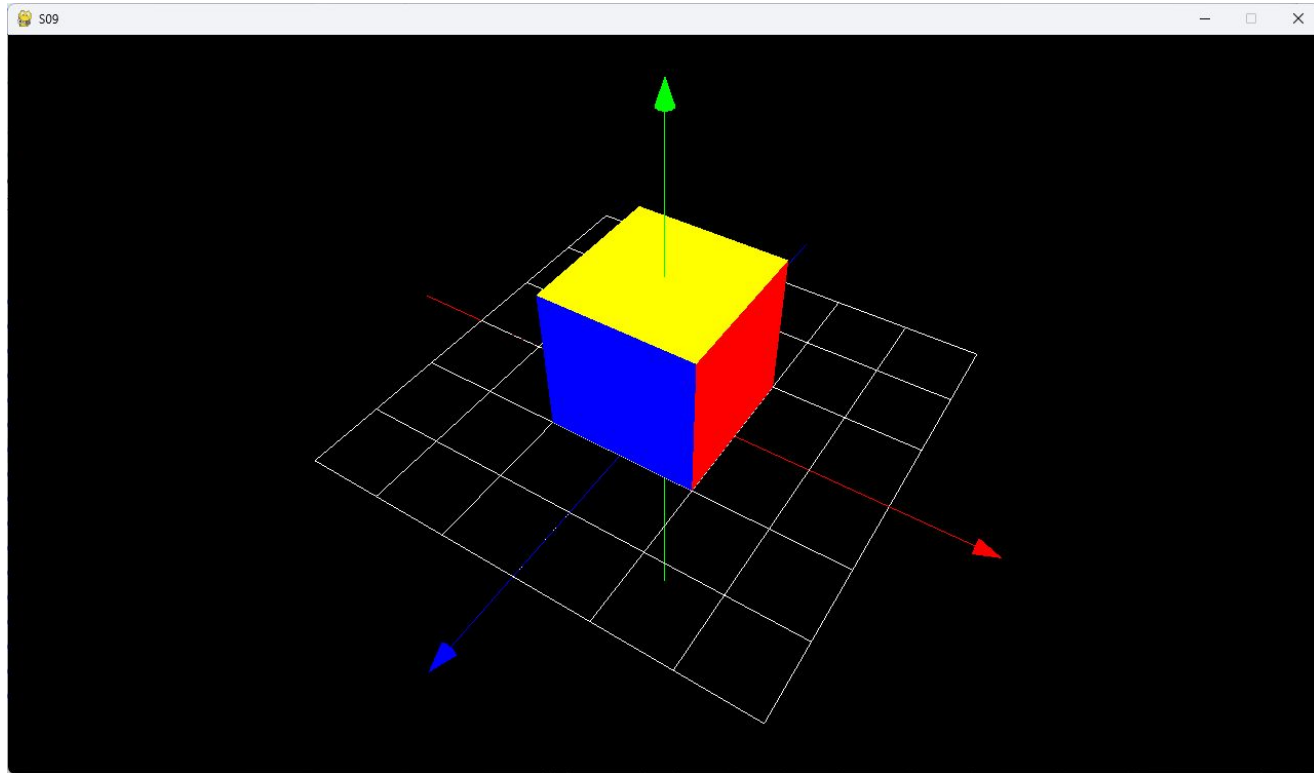
```
79 def dibujar_rejilla(): 1usage
80     """Dibuja una rejilla en el plano XZ utilizando líneas."""
81     glColor3f(*REJILLA_COLOR)
82     glBegin(GL_LINES)
83     for i in range(-REJILLA_TAMANO, REJILLA_TAMANO + 1):
84         # Líneas paralelas al eje X
85         dibujar_linea(-REJILLA_TAMANO, y1: 0, i, REJILLA_TAMANO, y2: 0, i)
86         # Líneas paralelas al eje Z
87         dibujar_linea(i, y1: 0, -REJILLA_TAMANO, i, y2: 0, REJILLA_TAMANO)
88     glEnd()
```

utilidades.py

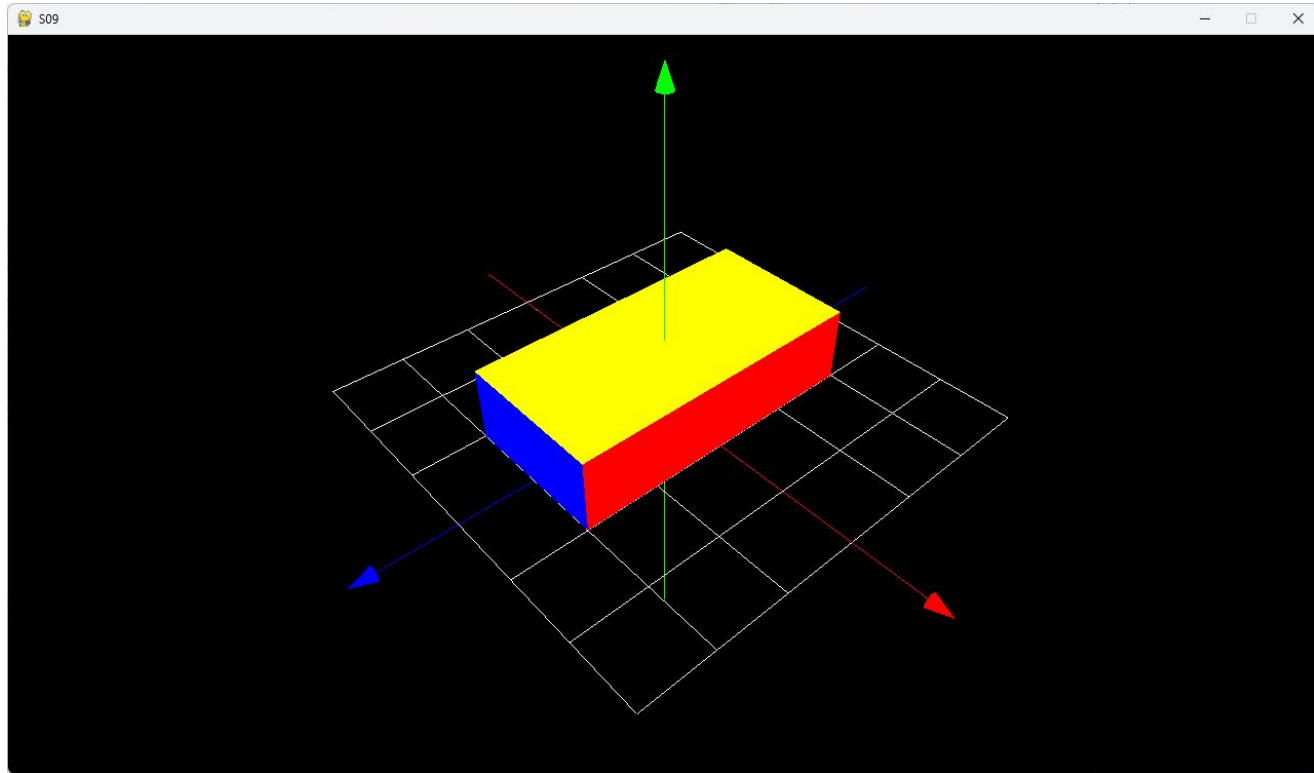
```
91 def dibujar_linea(x1, y1, z1, x2, y2, z2): 2 usages
92     """Dibuja una línea entre dos puntos.
93
94     Args:
95         x1, y1, z1 (float): Coordenadas del primer punto.
96         x2, y2, z2 (float): Coordenadas del segundo punto.
97     """
98     glVertex3f(x1 * REJILLA_PASO, y1, z1 * REJILLA_PASO)
99     glVertex3f(x2 * REJILLA_PASO, y2, z2 * REJILLA_PASO)
```

Ejercicios

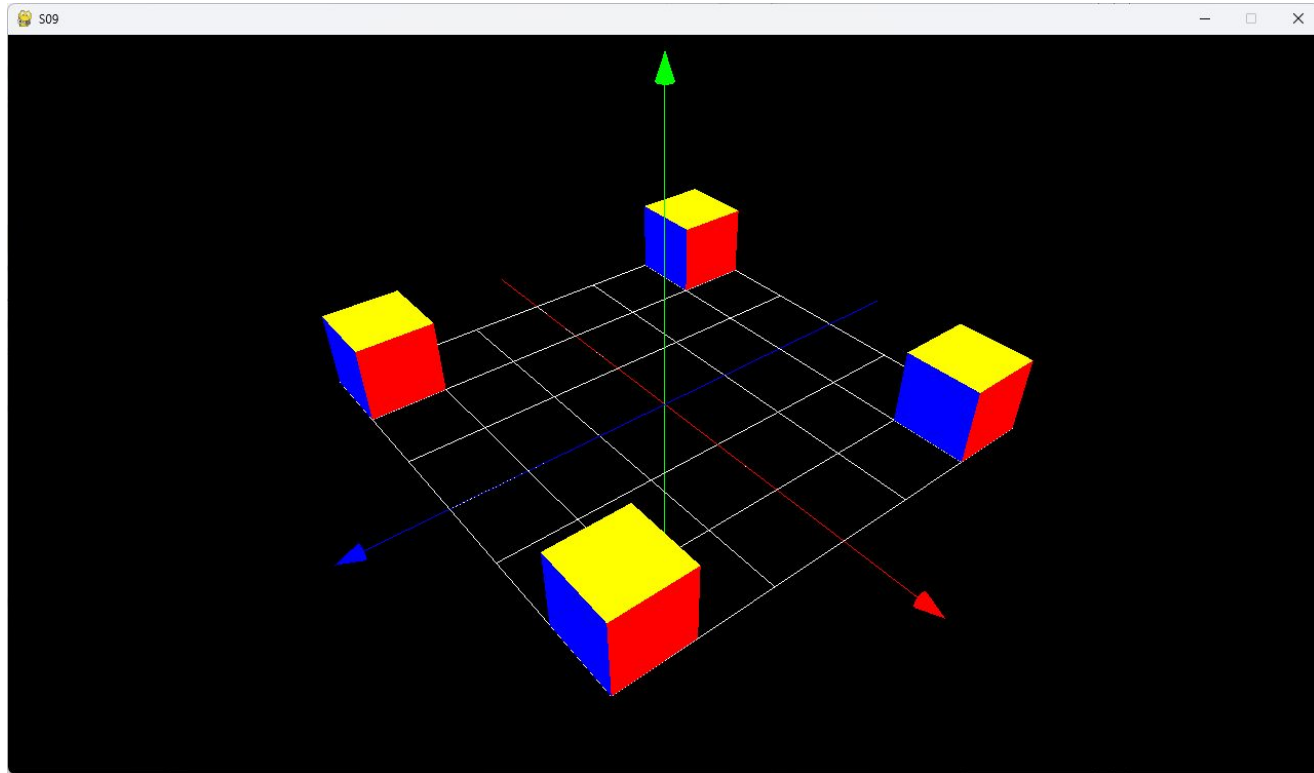
Ejercicio 1



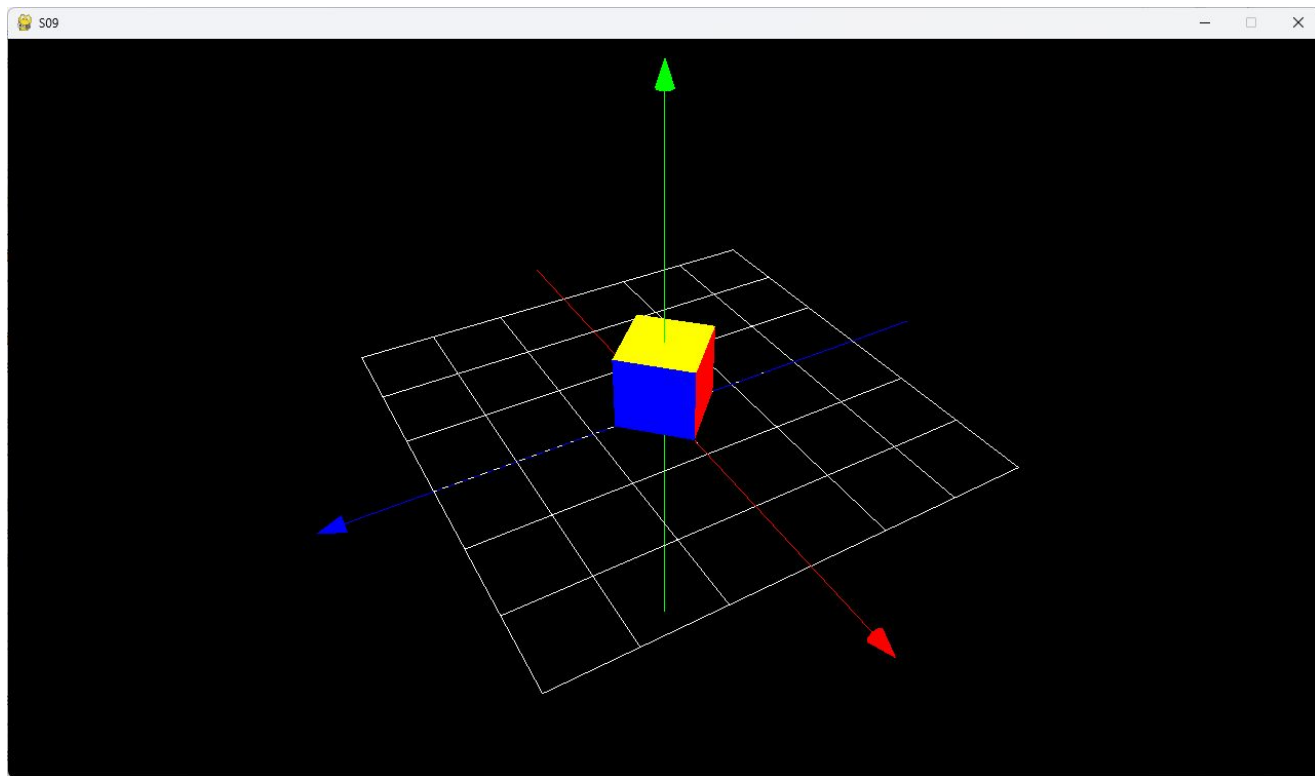
Ejercicio 2



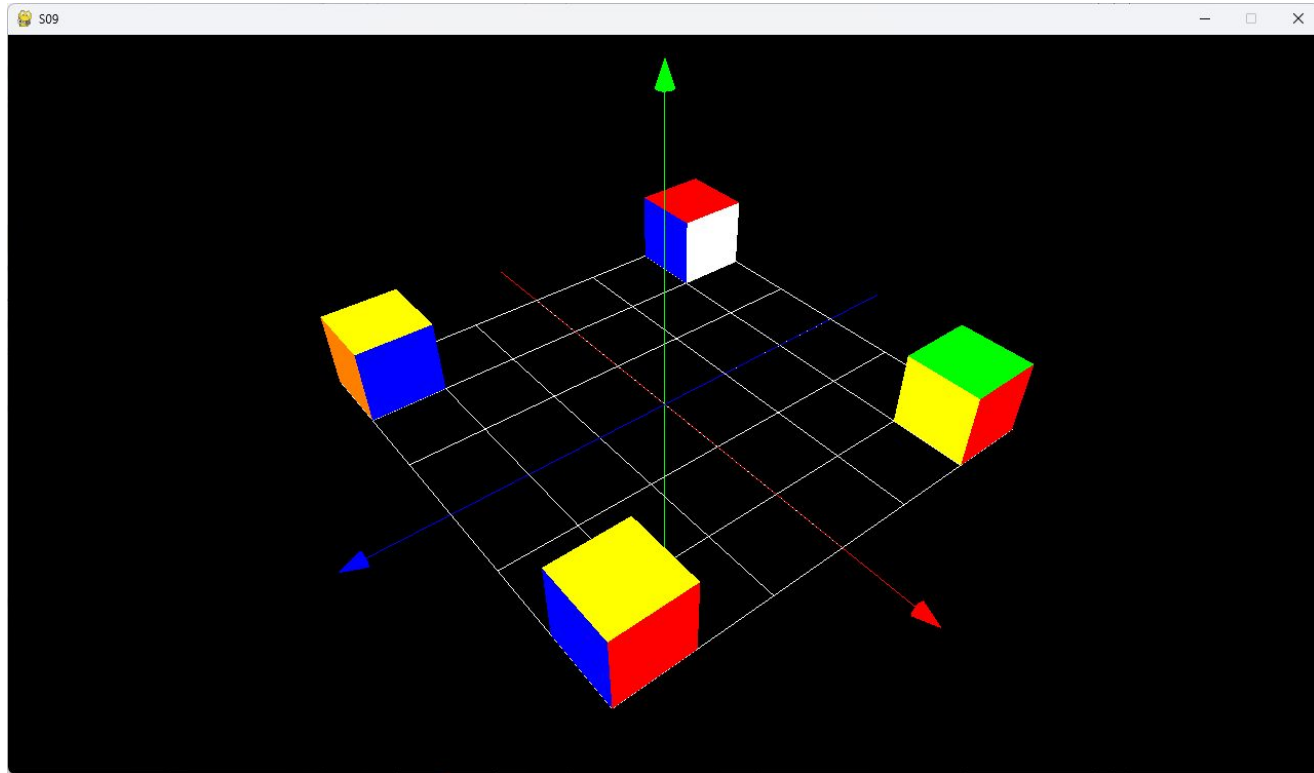
Ejercicio 3



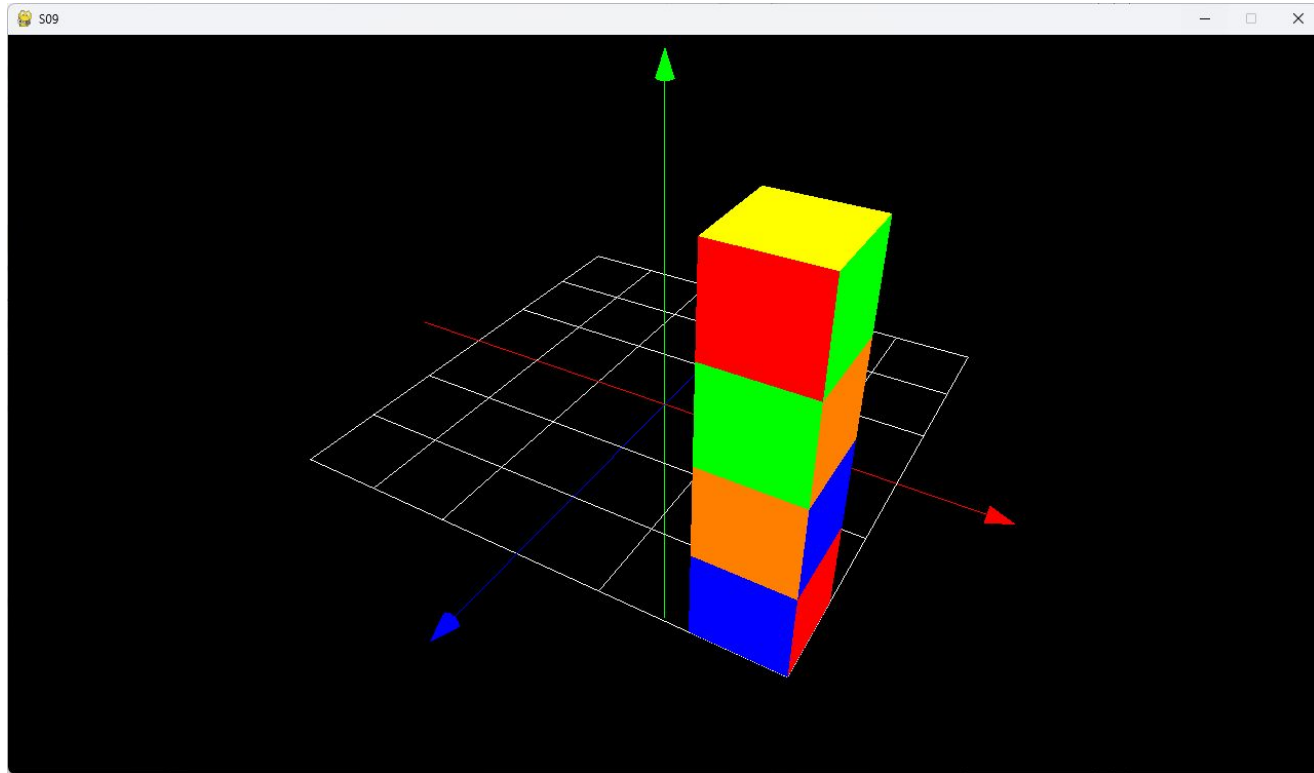
Ejercicio 4



Ejercicio 5



Ejercicio 6



Object files

Object file

Un *object file*, o archivo de objetos, es un archivo que contiene la representación de un modelo **3D**. Los *object files* almacenan información sobre la geometría de un objeto, como vértices, caras, normales y coordenadas de textura.

Existen diversos formatos de archivo para representar objetos, como **.OBJ**, **.FBX**, **.STL** o **.3DS**, siendo el más común el formato **.OBJ** desarrollado por Wavefront Technologies.

Los archivos **.OBJ** permiten especificar las siguientes características :

- **Geometría:** definen los vértices, aristas y caras que forman la estructura del objeto **3D**.
- **Normales:** incluyen información sobre las normales de las superficies. Las normales son vectores perpendiculares a las caras y son esenciales para calcular cómo reacciona la luz sobre la superficie del objeto.
- **Coordenadas de textura:** las coordenadas de textura permiten mapear texturas **2D** en la superficie 3D del objeto, especificando cómo se debe envolver la textura alrededor del modelo para lograr el efecto visual deseado.
- **Materiales:** mediante archivos **.MTL** es posible especificar materiales, y definir las propiedades visuales del objeto, como el color, la reflectividad y la transparencia.

Estructura de un fichero .OBJ

Cada línea de un archivo .OBJ comienza con una letra o palabra clave que indica el tipo de información que contiene esa línea:

- **Vértices (v):** cada vértice se define con una línea que empieza con la letra v, seguida por las coordenadas x, y, y z en el espacio 3D.
`Ej.: v 1.000000 2.000000 3.000000`
- **Normales de los vértices (vn):** cada normal se especifica con vn, seguida de sus componentes x, y y z.
`Ej.: n 0.0000 0.0000 1.0000`
- **Coordenadas de textura (vt):** cada coordenada vt, también llamada UV, se especifica con vt, seguida de sus componentes u, v y, opcionalmente, w.
`Ej.: vt 0.500 1.000`
- **Caras (f):** cada cara se especifica con la letra f, seguida de una lista de índices con hasta tres elementos: el índice de vértice, el índice de textura y el índice de la normal, separados por “/”.
`f 1/1/1 2/2/1 3/3/1`
- **Comentarios (#):** cualquier línea que empiece por # es un comentario.
`# Esto es un comentario`
- **Grupos y materiales:** las palabras reservadas g, usemtl y mtl lib definen grupos de vértices, materiales, y archivos .MTL, respectivamente.
`g Grupo1`
`usemtl MaterialEjemplo`
`mtllib ejemplo.mtl`

Fichero cube.obj

cube.py

```
1  # This file uses centimeters as units for non-parametric coordinates.
2
3  mtllib cube.mtl
4  g default
5  v -0.500000 -0.500000 0.500000
6  v 0.500000 -0.500000 0.500000
7  v -0.500000 0.500000 0.500000
8  v 0.500000 0.500000 0.500000
9  v -0.500000 0.500000 -0.500000
10 v 0.500000 0.500000 -0.500000
11 v -0.500000 -0.500000 -0.500000
12 v 0.500000 -0.500000 -0.500000
```

cube.py

```

27 vn 0.000000 0.000000 1.000000
28 vn 0.000000 0.000000 1.000000
29 vn 0.000000 0.000000 1.000000
30 vn 0.000000 0.000000 1.000000
31 vn 0.000000 1.000000 0.000000
32 vn 0.000000 1.000000 0.000000
33 vn 0.000000 1.000000 0.000000
34 vn 0.000000 1.000000 0.000000
35 vn 0.000000 0.000000 -1.000000
36 vn 0.000000 0.000000 -1.000000
37 vn 0.000000 0.000000 -1.000000
38 vn 0.000000 0.000000 -1.000000
39 vn 0.000000 -1.000000 0.000000
40 vn 0.000000 -1.000000 0.000000
41 vn 0.000000 -1.000000 0.000000
42 vn 0.000000 -1.000000 0.000000
43 vn 1.000000 0.000000 0.000000
44 vn 1.000000 0.000000 0.000000
45 vn 1.000000 0.000000 0.000000
46 vn 1.000000 0.000000 0.000000
47 vn -1.000000 0.000000 0.000000
48 vn -1.000000 0.000000 0.000000
49 vn -1.000000 0.000000 0.000000
50 vn -1.000000 0.000000 0.000000

```


cube.py

```
51     g pCube1  
52     usemtl initialShadingGroup
```

cube.py

```

53      f 1/1/1 2/2/2 3/3/3
54      f 3/3/3 2/2/2 4/4/4
55      f 3/3/5 4/4/6 5/5/7
56      f 5/5/7 4/4/6 6/6/8
57      f 5/5/9 6/6/10 7/7/11
58      f 7/7/11 6/6/10 8/8/12
59      f 7/7/13 8/8/14 1/9/15
60      f 1/9/15 8/8/14 2/10/16
61      f 2/2/17 8/11/18 4/4/19
62      f 4/4/19 8/11/18 6/12/20
63      f 7/13/21 1/1/22 5/14/23
64      f 5/14/23 1/1/22 3/3/24

```

Fichero modelo.py

modelo.py

```
7  class Modelo: 8 usages
8      """Clase para cargar y dibujar un modelo 3D en formato .obj."""
9
10     def __init__(self, filename, draw_type=GL_TRIANGLES):
11         """
12         Inicializa el modelo cargando el archivo .obj.
13
14         Parámetros:
15             filename (str): La ruta al archivo .obj.
16             draw_type (GLenum): Tipo de dibujo de OpenGL (GL_TRIANGLES o GL_LINE_LOOP).
17         """
18         # Inicializa listas para guardar vértices y triángulos
19         self.vertices = []
20         self.triangles = []
21         self.filename = filename # Nombre del archivo a cargar
22         self.draw_type = draw_type # Tipo de dibujo de OpenGL
23
24         # Llama al método que carga el modelo desde el archivo .obj
25         self.cargar_modelo()
```

modelo.py

```

27     def cargar_modelo(self): 1usage
28         """Lee el archivo .obj y extrae los vértices y triángulos."""
29         # Abre el archivo .obj y lo lee línea por línea
30         with open(self.filename) as file:
31             for line in file:
32                 # Si la línea comienza con 'v ', es una línea de vértice
33                 if line.startswith("v "):
34                     # Divide la línea en palabras y convierte las últimas tres a números decimales (float)
35                     partes = line[2:].strip().split()
36                     x = float(partes[0])
37                     y = float(partes[1])
38                     z = float(partes[2])
39                     # Guarda el vértice en la lista de vértices
40                     self.vertices.append((x, y, z))
41
42                 # Si la línea comienza con 'f ', es una línea de triángulo (cara)
43                 elif line.startswith("f "):
44                     # Divide la línea en palabras y procesa cada índice de vértice
45                     partes = line[2:].strip().split()
46                     indices = []
47                     for parte in partes:
48                         # Divide cada parte (por ejemplo, '1/1/1') en los índices correspondientes
49                         indice = int(parte.split('/')[0]) - 1 # Convertimos a entero y ajustamos el índice
50                         indices.append(indice)
51
52                     # Solo guarda el triángulo si tiene tres vértices
53                     if len(indices) == 3:
54                         self.triangles.append(tuple(indices))

```

modelo.py

```

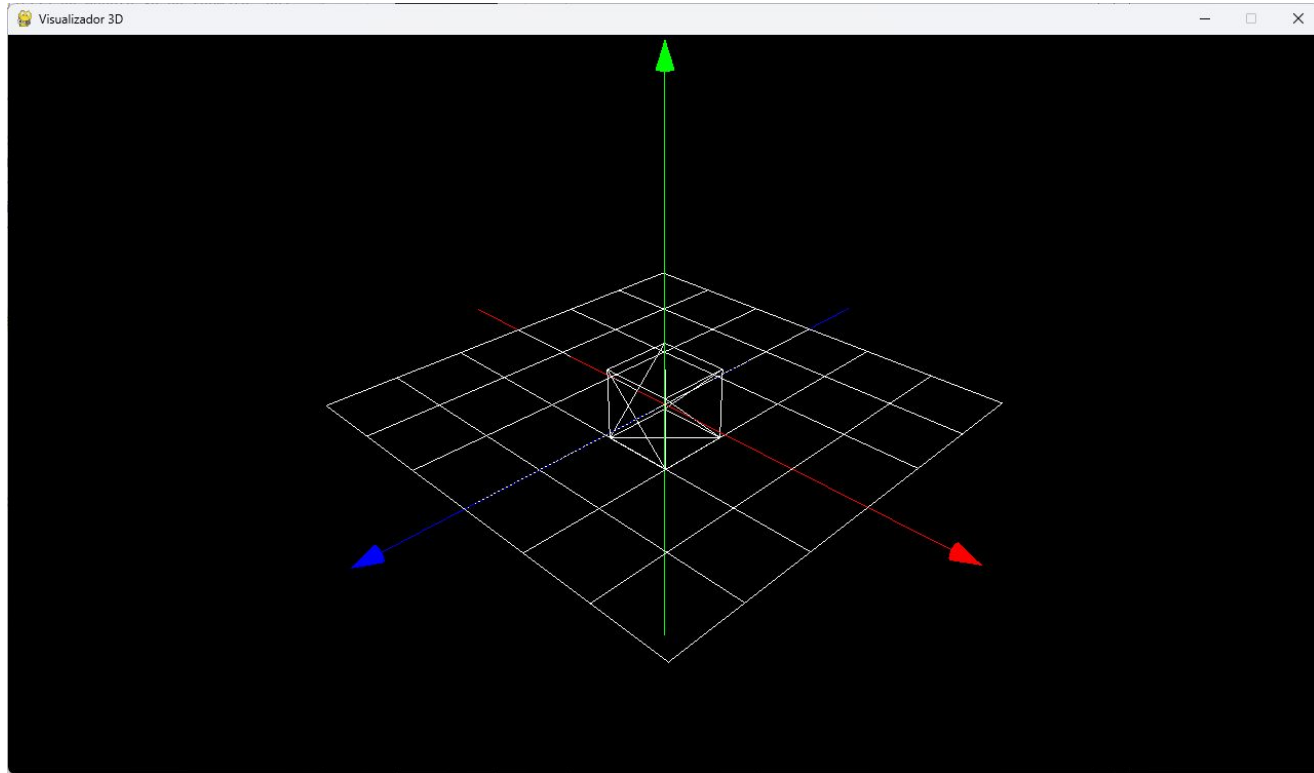
56     def dibujar(self, x=0, y=0, z=0, angulo=0, eje_x=0, eje_y=1, eje_z=0, sx=1, sy=1, sz=1):
57         """
58         Dibuja el modelo en la posición especificada y aplica transformaciones.
59
60         Parámetros:
61             x, y, z (float): Posición del modelo en el espacio.
62             angulo (float): Ángulo de rotación.
63             eje_x, eje_y, eje_z (float): Ejes de rotación.
64             sx, sy, sz (float): Escala del modelo.
65         """
66         # Llama a la función de transformación antes de dibujar el objeto
67         transformar(x, y, z, angulo, eje_x, eje_y, eje_z, sx, sy, sz, self._dibujar_objeto)

```

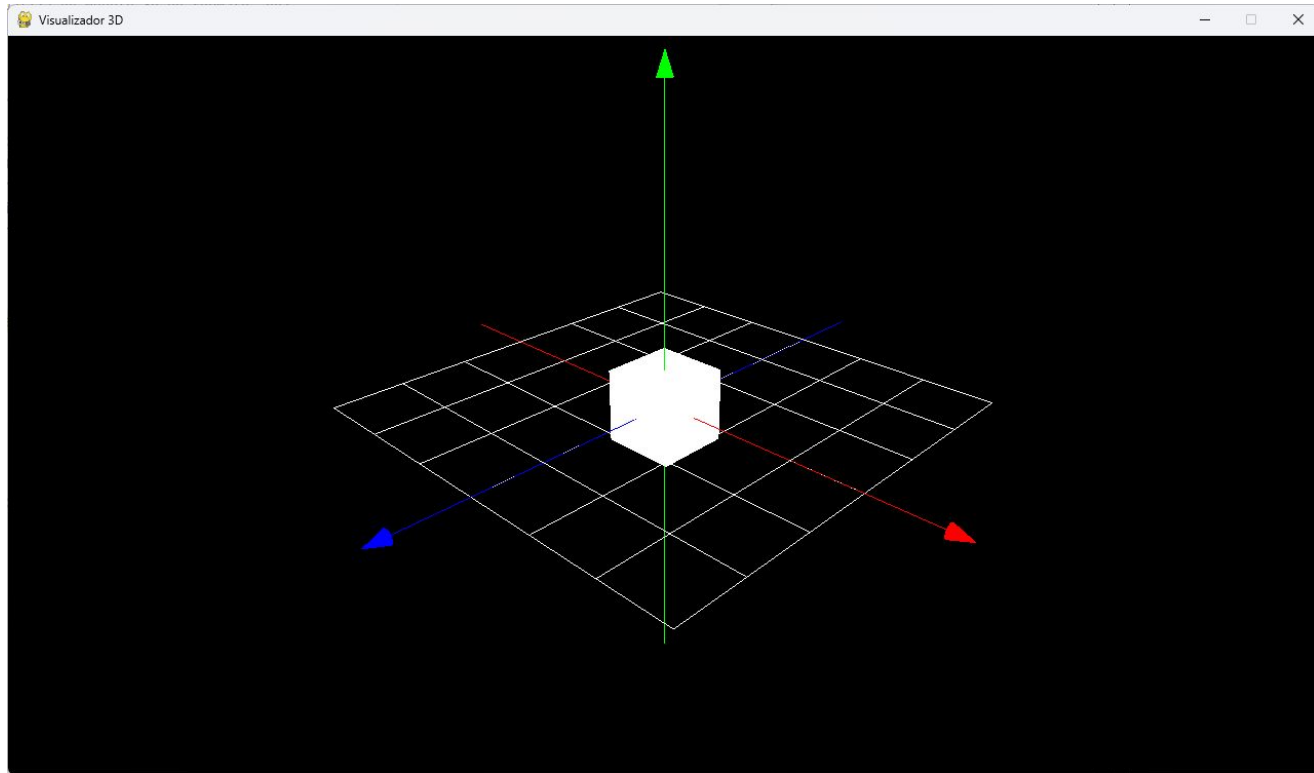
modelo.py

```
69     def _dibujar_objeto(self): 1 usage
70         """Dibuja cada triángulo del modelo usando los vértices."""
71         # Para cada triángulo en la lista, dibuja los tres vértices
72         for v1, v2, v3 in self.triangles:
73             glBegin(self.draw_type) # Inicia el dibujo según el tipo (triángulo o contorno)
74
75             # Dibuja cada vértice del triángulo
76             glVertex3fv(self.vertices[v1]) # Primer vértice
77             glVertex3fv(self.vertices[v2]) # Segundo vértice
78             glVertex3fv(self.vertices[v3]) # Tercer vértice
79
80             glEnd() # Termina el dibujo del triángulo
```

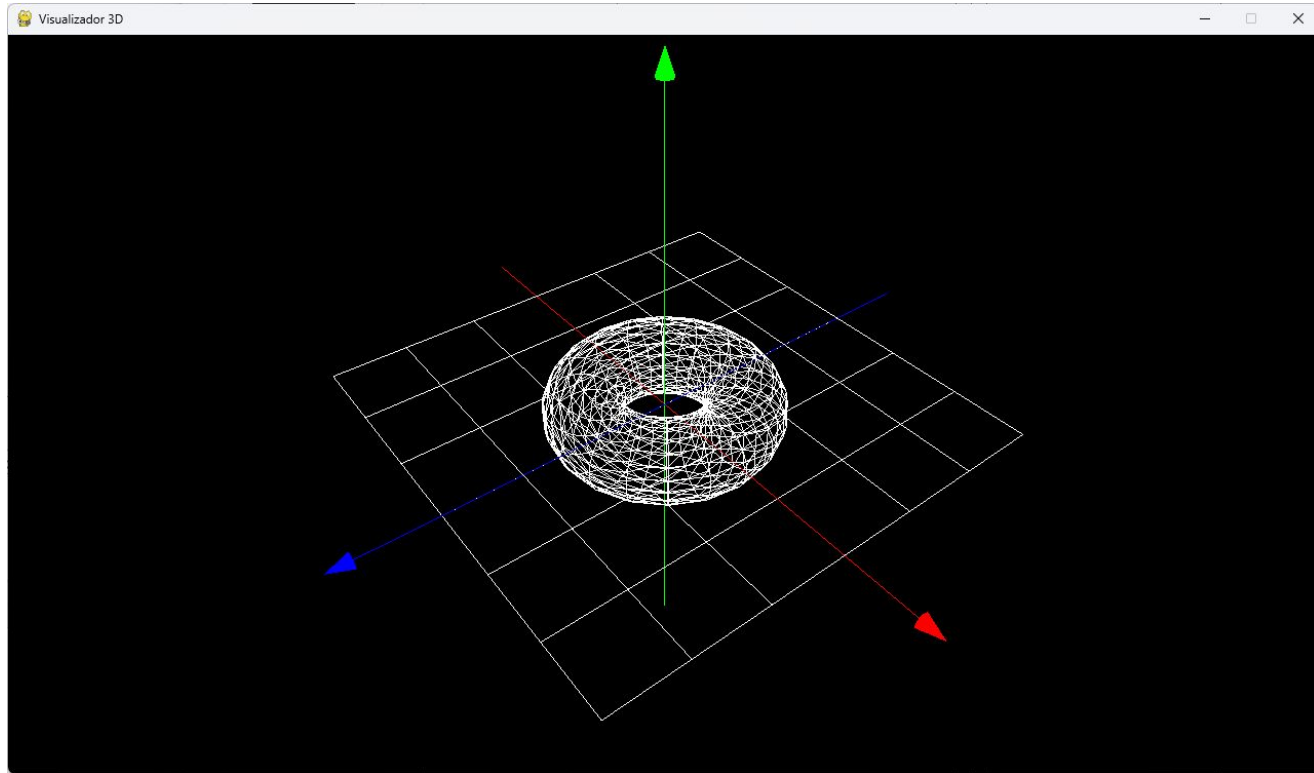
Object file: cube.obj. Draw type = GL_LINE_LOOP



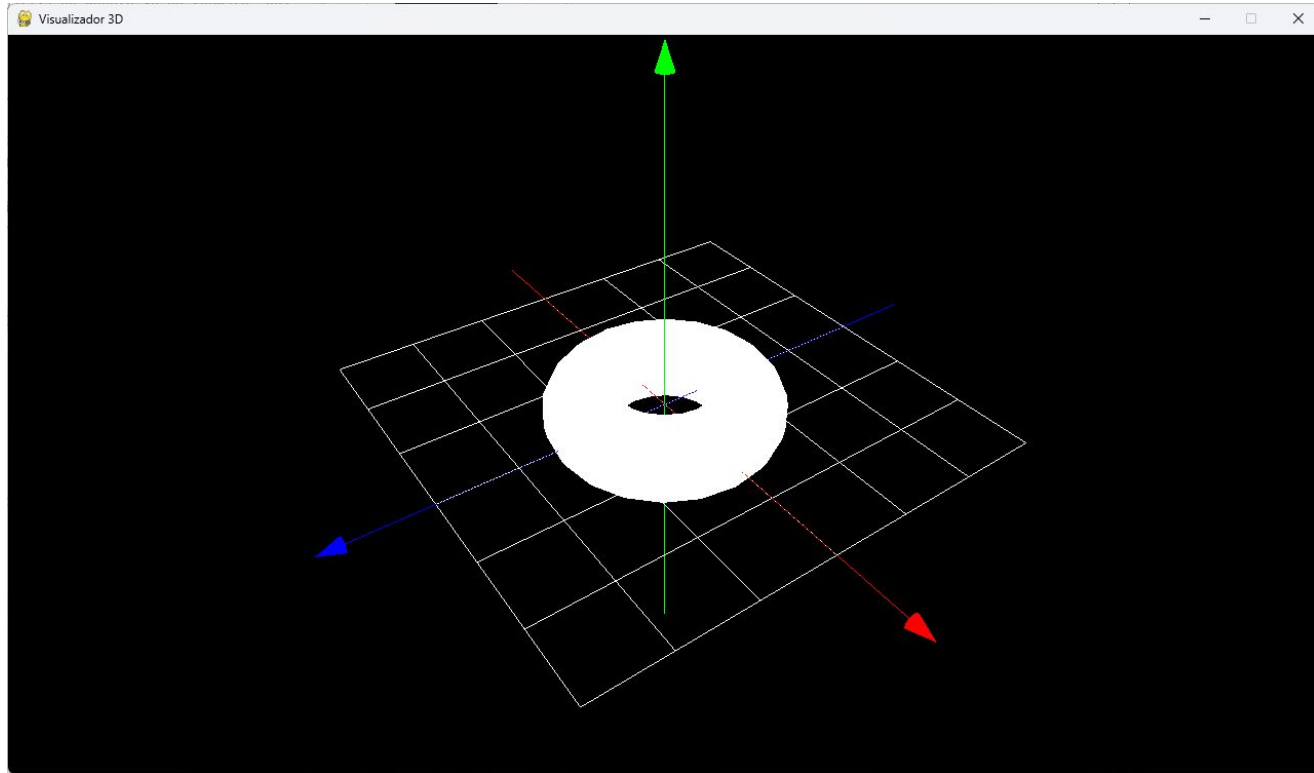
Object file: cube.obj. Draw type = GL_TRIANGLES



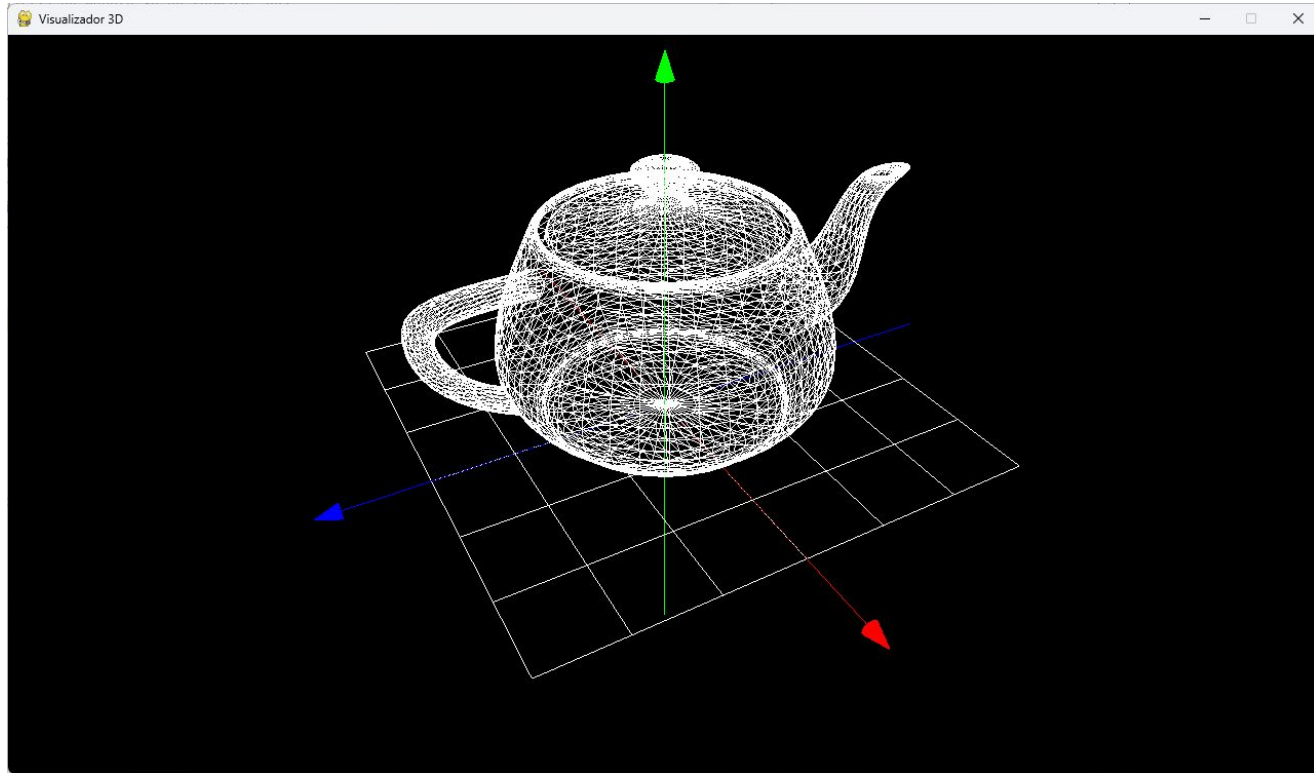
Object file: donut.obj. Draw type = GL_LINE_LOOP



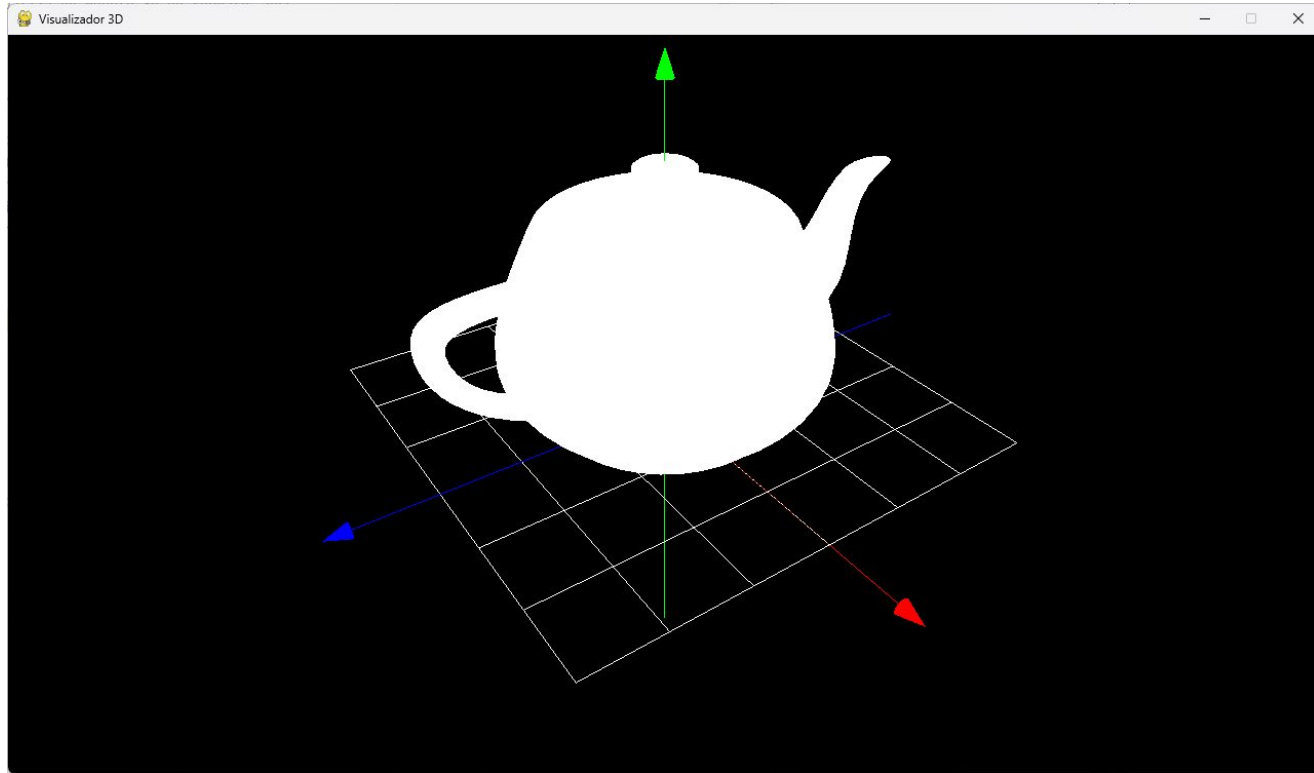
Object file: donut.obj. Draw type = GL_TRIANGLES



Object file: teapot.obj. Draw type = GL_LINE_LOOP

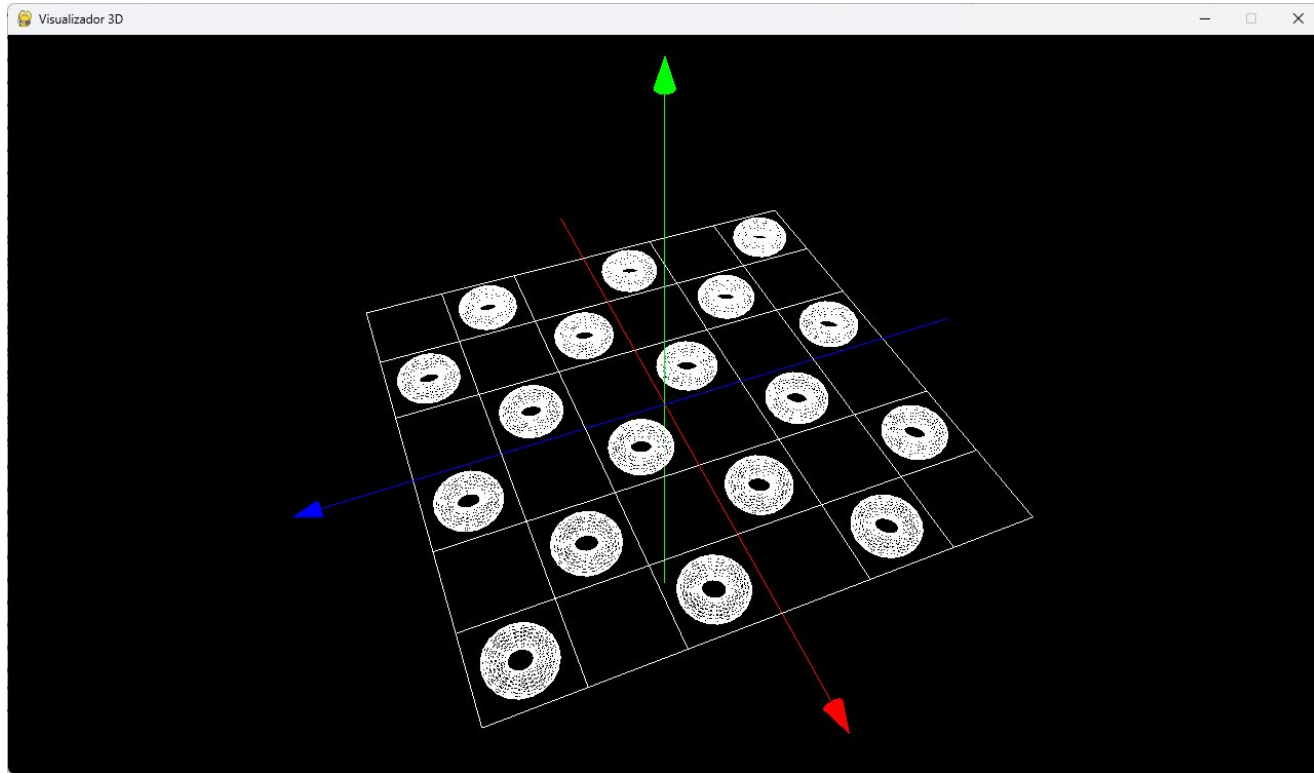


Object file: teapot.obj. Draw type = TRIANGLES

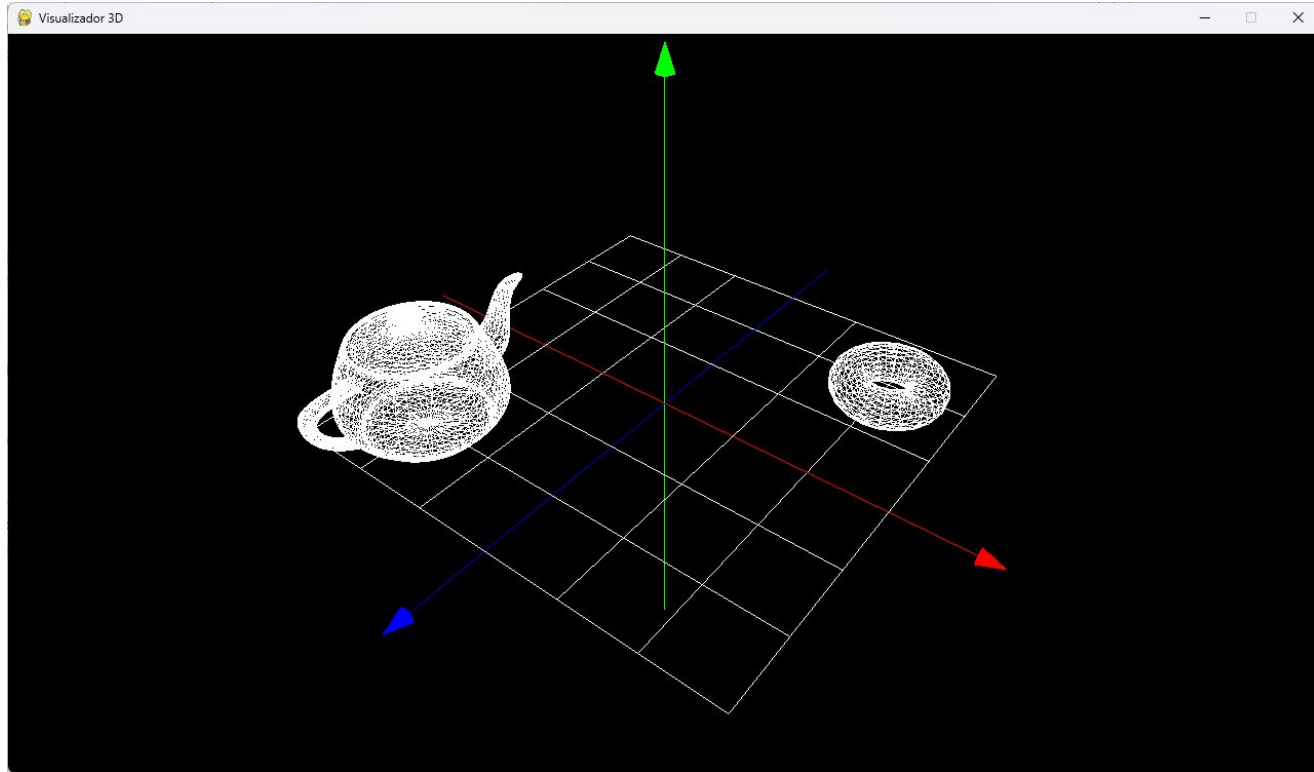


Ejercicios

Ejercicio 7 (Factor de escala donut: 0,25)



Ejercicio 7 (Factor de escala donut y teapot: 0,5)



Referencias bibliográficas

- Foley, J.D., Van Dam, A., Feiner, S.K., Hughes, J.F., Phillips, R.L. (1993). *Introduction to Computer Graphics*. Addison-Wesley Publishing Company.
- Méndez, M. (2022). *Introducción a la graficación por computadora*.
<https://proyectodescartes.org/iCartesiLibri/PDF/GraficacionComputadora.pdf>