Preguntas

A) Exponer las 4 características de la proyección de perspectiva.

B) Dadas las siguientes configuraciones, dibujar como se vería un cubo unitario (vértices en (0,0,0) y en (1,1,1)). Se supone que hay un plano de recorte delantero (PLANO FRONTAL), que coincide con el plano de proyección. El PRP está en el eje z a una distancia d. Mostrar también los ejes. El sistema es coordenado derecho.

VRP	VPN	VUP
(0,0,2)	(1,0,1)	(0,1,0)
(0,10,0)	(0,2,0)	(0,0,-1)
(-1,-1,-1)	(-1,-1,-1)	(0,1,0)
(-1,0,-1)	(-1,0,-1)	(1,1,0)
(2,0,0)	(-1,0,-1)	(0,-1,0)

C) Implementar en seudocódigo la rotación con respecto a un eje arbitrario, definido por dos puntos.

D) ¿Por qué es conveniente definir el PRP, PD, PT y ventana en coordenadas de vista y no en coordenadas de mundo? ¿Cómo afecta, para obtener la transformación de vista en OpenGL, la definición del PRP?

E) Qué son las coordenadas homogéneas y por qué son necesarias. Como se aplican.

F) Marcar las 4 características de la proyección de perspectiva (las incorrectas cuentan negativamente)

Distorsión topológica ☐
Acortamiento perspectivo ☐
Inversión de vista ☐
Puntos de fuga ☐
Líneas oblicuas al horizonte ☐
Acercamiento del infinito ☐
Perseverancia cíclica ☐
Ajuste de vista ☐

G) Dado el siguiente VPN(25,13,20) , realizar los cálculos para obtener los dos ángulos para alinearlo con el eje Z.

H) Diferencia entre geometría y topología. Modos de almacenamiento de ambas.

I) Explicar por qué ocurre lo siguiente: al poner la ventana de la aplicación a tamaño completo, generalmente de forma rectangular, la imagen de la

escena se mantiene cuadrada, desaprovechando el espacio dejado a izquierda y derecha. ¿Cómo se arreglaría?

J) Demostrar que una reflexión respecto al eje X seguida de una reflexión con respecto al eje Y es equivalente a una rotación de 180º.

K) Al cambiar la ventana de una aplicación su tamaño pasa a ser AnchoxAlto. Dada las distancia d1 al plano delantero y d2 al plano trasero, indicar como quedaría la llamada a glFrustum ()

L) ¿Qué diferencia hay entre modo inmediato y modo retenido en OpenGL.

M) Se desea obtener una relación 1:1 en un espacio 2D sin deformación en OpenGL. ¿Cuales deberían ser los valores de glOrtho?

N) Implementar en seudocódigo el cálculo de las normales de los puntos, dado que poseemos un vector con las normales de las caras.

O) Hemos importado un modelo PLY y queremos comprobar que es cerrado. Escribir el seudocódigo (indicar las estructuras de datos que se usan)

P) Dado la clase cilindro, que crea un cilindro unitario centrado en el origen y con las tapas en el eje Y, crear la clase rueda.

Q) La luna está a 384.400km de la tierra y tiene un diámetro de 3.476km. El sol tiene un diámetro de 1.392.000km. Sabiendo que en un eclipse total de sol, la luna cubre totalmente al sol, ¿cómo podríamos calcular la distancia a la que está? ¿Cual es la distancia?

R) Calcular la matriz de transformación resultante de aplicar las siguientes instrucciones OpenGL:

```
glLoadIdentity();
glRotatef(30,0,1,0);
glScale3f(1.0,0.5,2.0);
glTranslate3i(30,10,20);
```

S) Indicar que hace cada una de estas primitivas de OpenGL. Poner un ejemplo dibujado, numerando los vértices. (GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_POLYGON, GL_QUADS, GL_QUAD_STRIP, GL_TRIANGLES, GL_TRIANGLE, GL_TRIANGLE_FAN)

T) Indicar que hacen estas funciones:

glutDisplayFunc(funcion);
glutReshapeFunc(funcion);
glutKeyboardFunc(funcion);
glutSpecialFunc(funcion);

U) Implementar el siguiente procedimiento que dibuja un cilindro
void dibujar_cilindro(float Radio, float Altura, int Num_divisiones)

V) Suponiendo que existen las primitivas cubo y cilindro, de tamaño unitario y centrados en el origen, crear un procedimiento **void dibujar_coche()** que dibuje un coche (lo importante son las transformaciones)

W) Se han generado los puntos que representan a una esfera (m puntos para la curva generatriz, n divisiones). Se han guardado en un vector, por meridianos. Implementar un procedimiento que almacene las caras.

X) Calcular los valores de los parámetros de la transformación de vista, dados el VRP, VPN y VUP. Mostrar el código OpenGL que permitiría aplicar la Transformación de vista.

Y) Implementar el procedimiento que usando una función senoidal simula un movimiento de aceleración y desaceleración.

INFORMÁTICA GRÁFICA

Relación de ejercicios.

1. Dado un cubo unidad situado con un vértice en el origen del sistema de coordenadas, ver figura 1. Indicar los parámetros a usar para visualizarlo desde las siguientes posiciones:
   a) Desde el eje Y, a una distancia de 5 unidades mirando hacia el origen.
   b) Desde la bisectriz del tercer cuadrante mirando hacia el extremo inferior del cubo situado sobre el eje X, con Y y Z a cero.

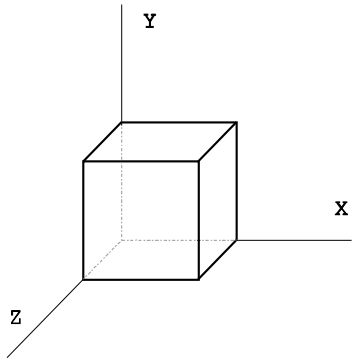


Figura 2.1

2. Describir la imagen que se generaría en cada uno de los casos anteriores.

3. Redactar procedimientos que efectúen los siguientes recorridos de cámara:
   a) Recorrido circular de radio 5 unidades mirando hacia el origen sobre el eje y sobre el plano y=0.
   b) Recorrido en forma de escalera de caracol, de radio 5 unidades, partiendo de una altura de y=5 hasta y=-5, mirando hacia el origen.
   c) Alejamiento de la cámara, mirando desde la bisectriz del primer cuadrante al origen de coordenadas (comenzando en una distancia de 5 y finalizando a una distancia de 20).

4. Implementar un algoritmo para realizar un efecto zoom en OpenGL

5. Se pretende simular una caída en barrena vertical. Describir un algoritmo que realice este efecto en OpenGL . El observador siempre mira al origen de coordenadas.

6. Para definir una vista se puede utilizar el siguiente conjunto de parámetros:
   PCamara: Posición del observador.
   PAtencion: Posición a la que mira.
   Apertura: Angulo de apertura del cono de visión.
   Spin: Angulo de giro de cámara respecto a la vertical.
   Indicar como se fijarían los parámetros de vista de OpenGL a partir de estos.

7. Construir la función *Transform*(Sx, Sy, a, tx,ty) para transformar un símbolo 2D. Se quiere que cuando el símbolo tenga tamaño 1x1 se instancie teniendo tamaño final (Sx,Sy), y aparezca rotado un ángulo a, con el punto que inicialmente está en el origen en la posición (tx,ty).

8. ¿Cómo se puede calcular automáticamente la caja englobante o marco de un modelo (mínimo paralelepípedo alineado con los ejes que lo encierra)?

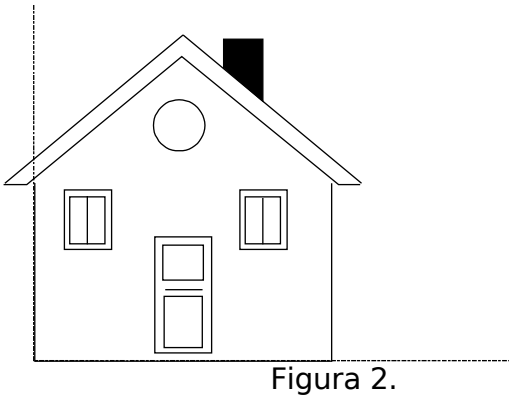


Figura 2.

9. Dado el modelo 2D de la figura 2
   - Indicar cómo se podría construir con OpenGL .

10. Encontrar una estructura para el modelo de una aplicación de diseño del amueblamiento de una planta de un edificio. ¿Tendría alguna ventaja utilizar un modelo jerárquico?

11. Se dispone de dos procedimientos que visualizan respectivamente un cubo de tamaño unidad y una pirámide de base cuadrada y lado uno. Redactar un procedimiento que genere una figura formada por un cubo central de tamaño 10 y seis pirámides cuadradas de lado 5 adosadas a sus caras.

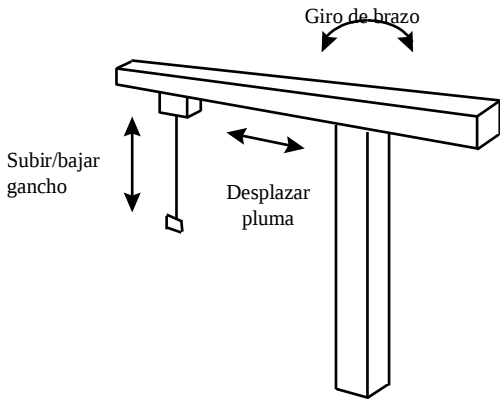


Figura 3

12. Se desea usar la librería OpenGL para diseñar la grua de la figura 3. El modelo se debe diseñar de forma que sea posible, y fácil, girar el brazo, desplazar la pluma, y subir y bajar el gancho. Debe tenerse en cuenta las restricciones de movimiento. Realizar el grafo del modelo, indicando de forma clara el contenido de cada nodo (primitivas y transformaciones). Tener en cuenta que la cuerda debe cambiar de tamaño al moverse el gancho. Especificar las transformaciones que sería necesario introducir para realizar los movimientos anteriores.

13. Describir la estructura que tendría en OpenGL el modelo del sistema de la figura 4. El sistema está formado por una secuencia de tres segmentos conectados mediante articulaciones, el primero de los cuales se une a un cuarto segmento fijo mediante otra articulación. Los segmentos poseen una guía central que fuerza a estos a mantener una altura constante, haciendo que de hecho el sistema posea un solo grado de libertad. Dibujar el grafo de estructuras, indicando las transformaciones geométricas, y el proceso que se debe seguir para modificar el ángulo.

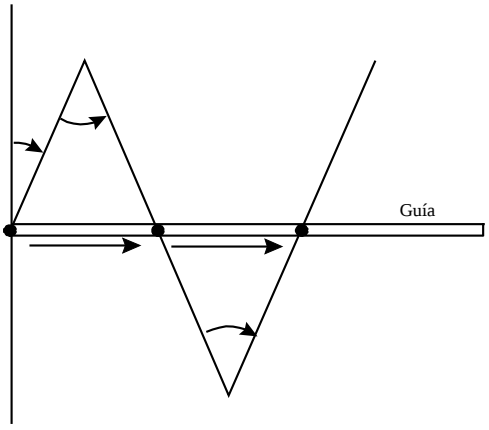


Figura 4

14. Realizar en OpenGL el modelo mostrado en la figura 5. Los cubos poseen dimensión unidad, y se dispone de un procedimiento que crea un cubo definido en el origen.

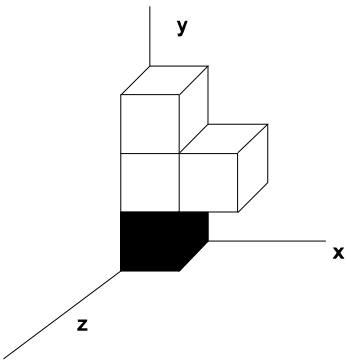


Figura 5

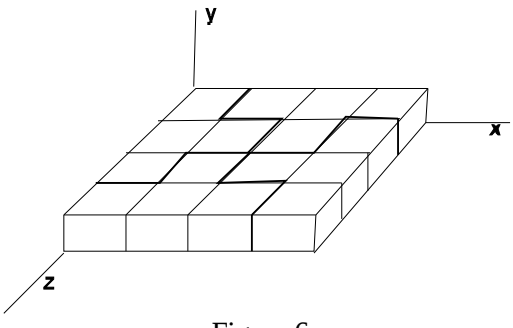


Figura 6

15. Generar el modelo necesario para obtener un mosaico de la figura 6, partiendo de la figura generada en el ejercicio anterior.

16. Indicar la forma en que se podría construir en OpenGL el modelo de la figura 7, de tal modo que simplemente añadiendo una transformación geométrica se puedan separar los dos cubos de forma simétrica del centro (suponer definido el cubo).

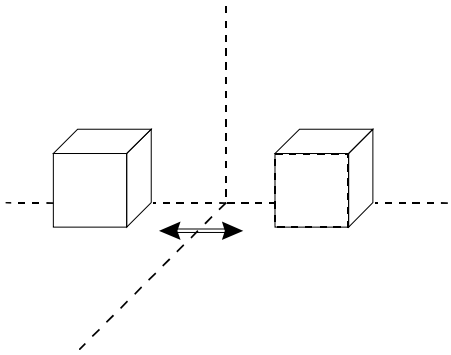


Figura 7

17. Se desea usar la librería OpenGL para diseñar el robot de la figura 8. Para ello, se dispone de un conjunto básico de elementos que se muestra en la figura 9. El robot se compone de cabeza, tronco, base, brazos y pinza, estando situado sobre el origen de coordenadas, mirando a lo largo del eje z. El extremo inferior izquierda de todas las piezas está situado sobre el origen de coordenadas. El tamaño de cada una de las piezas es el siguiente:

Cabeza: Xmáx = 30, Ymáx = 20, Zmáx = 20
Tronco: Xmáx = 50, Ymax = 80, Zmáx = 20
Brazo: Xmáx = 15, Ymáx = 40, Zmáx = 20
Pinza: Xmáx = 10, Ymáx = 5, Zmáx = 20
Base: Xmáx = 40, Ymáx= 40, Zmáx = 20 (La parte superior de la base tiene un ancho de 20)
Crear el grafo del modelo y diseñar el robot usando OpenGL, partiendo del conjunto de piezas ya creadas.

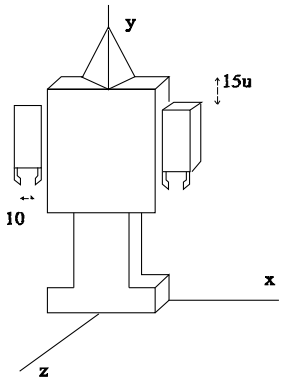


Figura 8

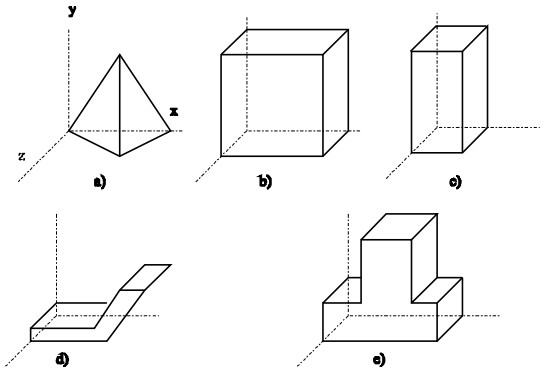


Figura 9

18. Introducir en el diseño anterior los siguientes movimientos:
a) giro del tronco 90º en sentido antihorario
b) elevación del brazo izquierdo 90º

19. Considerar y explicar razonadamente cómo quedaría el grafo (DAG) si el movimiento de elevación de los dos brazos fuese solidario, conservándose el movimiento independiente de cada una de las muñecas y pinzas.

20. Queremos obtener una modelo de una persiana de varillas, en el que podamos subir y bajar la persiana y rotar las varillas solidariamente (figura 10). Diseñar el grafo del modelo y codificar el modelo usando OpenGL.

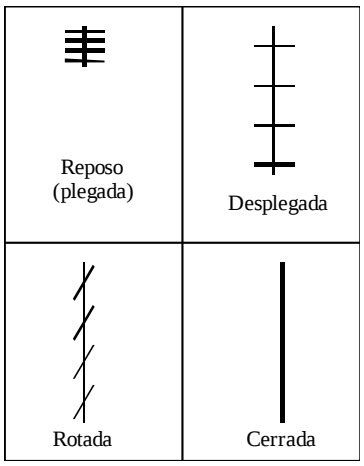


Figura 10

21. Modelar una balanza como la mostrada en la figura 11. Redactar un procedimiento de edición para girar la balanza un determinado ángulo (siendo la posición del dibujo de reposo).

22. Se desea modelar el mecanismo de una biela (figura 12). Dado el giro de la pieza b, se debe modificar la posición de la biela (pieza a) de tal modo que su longitud permanezca constante. EL pistón (c) se moverá verticalmente (subiendo y bajando) dependiendo del ángulo de giro.

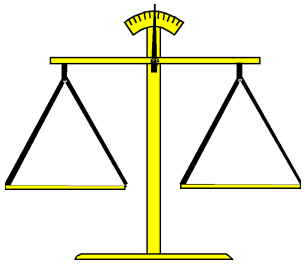


Figura 11

23. Diseñar un modelo jerárquico para el objeto de la figura 13, a partir de un cilindro de altura uno, y radio de base uno. Detallar todas las transformaciones geométricas utilizadas. Los tamaños de las piezas se indican en la tabla adjunta.

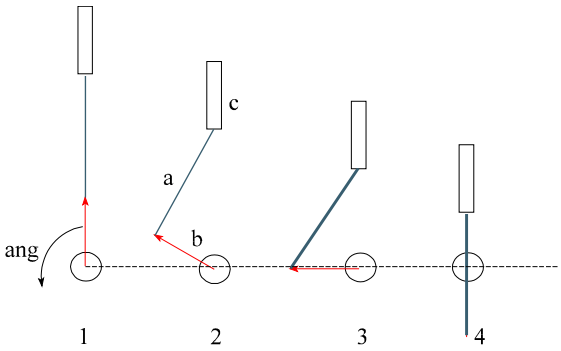


Figura12

24. Redactar un procedimiento en C que cree un modelo jerárquico para el objeto de la figura 14, usando openGL. Detallar todas las transformaciones geométricas utilizadas. Suponer que las piezas están previamente definidas, decidir e indicar los tamaños para cada pieza base, así como las dimensiones usadas para el modelo. Las piezas A y C son fijas. La pieza B está unida a A mediante una bisagra vertical. D se une a B con una articulación, y el pasador E se desliza a través de D.

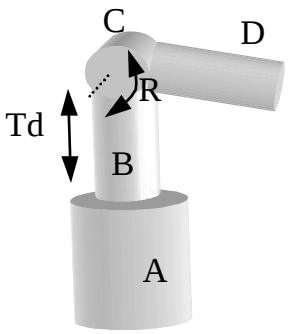


Figura 13

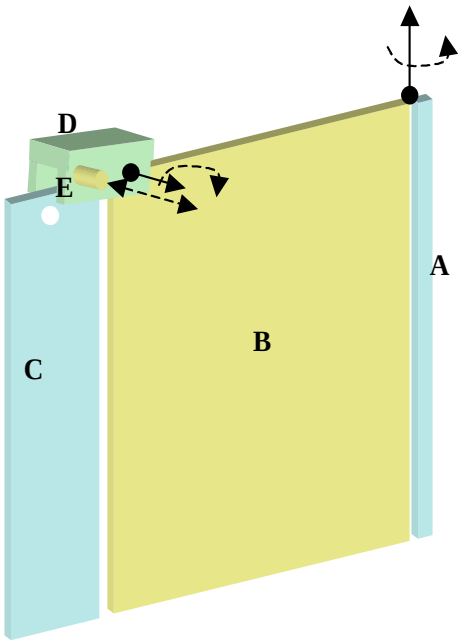


Figura 14

Informática Gráfica.
Relación de Enunciados de Problemas.
Curso 2012-13.

Carlos Ureña
Dpt. Lenguajes y Sistemas Informáticos
ETSI Informática y de Telecomunicación
Universidad de Granada

Documento generado: January 2, 2013

Contents

Tema 1

Problemas del tema 1

1

El producto escalar de dos vectores es una operación binaria (simbolizada por $\cdot$) que produce un valor real y que puede definirse por sus propiedades, en concreto, las siguientes (para cualquier trio de vectores $\mathbf{a}$, $\mathbf{b}$ y $\mathbf{c}$, y para cualquier valor real t)

(1) Conmutativa: $\mathbf{a} \cdot \mathbf{b} \equiv \mathbf{b} \cdot \mathbf{a}$

(2) Distributiva respecto de la suma: $\mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) \equiv \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c}$

(3) Conmutatividad con producto externo: $(t\mathbf{a}) \cdot \mathbf{b} \equiv t(\mathbf{a} \cdot \mathbf{b})$

adicionalmente, asumimos por definición que los tres ejes o versores del sistema de coordenadas del mundo ($\mathbf{x}$,$\mathbf{y}$ y $\mathbf{z}$) son de longitud unidad y perpendiculares entre sí. Esto se expresa en terminos de propiedades del producto escalar de estos vectores, en concreto, usamos estos dos axiomas adicionales:

(4) Longitud unidad: $\mathbf{x} \cdot \mathbf{x} \equiv \mathbf{y} \cdot \mathbf{y} \equiv \mathbf{z} \cdot \mathbf{z} \equiv 1$

(5) Perpendicularidad: $\mathbf{x} \cdot \mathbf{y} \equiv \mathbf{y} \cdot \mathbf{z} \equiv \mathbf{z} \cdot \mathbf{x} \equiv 0$

todos estos axiomas también se cumplen en 2D, usando lógicamente dos ejes en lugar de tres.

A partir de estos cinco axiomas, **demuestra** que si $\mathbf{a} = (x_0, y_0, z_0)$ y $\mathbf{b} = (x_1, y_1, z_1)$ entonces se cumple que:

$$\mathbf{a} \cdot \mathbf{b} = x_0 x_1 + y_0 y_1 + z_0 z_1$$

para hacer esta demostración, se debe tener en cuenta que las coordenadas cartesianas de un vector son tres valores que indican como obtener dicho vector como suma ponderada de $\mathbf{x}$,$\mathbf{y}$ y $\mathbf{z}$, es decir, decir que $\mathbf{v} = (a, b, c)$ es lo mismo que decir que $\mathbf{v} = a\mathbf{x} + b\mathbf{y} + c\mathbf{z}$

Responde a esta pregunta: ¿ sigue siendo esto válido para vectores en 2D ? (se cumplen los mismos

axiomas, incluyendo los dos versores **x** e **y** en lugar de tres)

2

En 2D, la rotación entorno al origen (un ángulo ϕ) de un vector (x,y) se expresa como la multiplicación por una matriz 2x2 (que llamamos R_ϕ):

$$\begin{pmatrix} x' \\ y' \end{pmatrix} \;=\; R_\phi \begin{pmatrix} x \\ y \end{pmatrix} \;=\; \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \;=\; \begin{pmatrix} cx - sy \\ sx + cy \end{pmatrix}$$

donde $c \equiv \cos(\phi)$ y $s \equiv \sin(\phi)$.

Demuestra que el producto escalar es invariante por rotación, es decir, que para cualquier ángulo ϕ y vectores $\mathbf{a} = (x_0, y_0)$ y $\mathbf{b} = (x_1, y_1)$ se cumple:

$$(R_\phi \mathbf{a}) \cdot (R_\phi \mathbf{b}) \;=\; \mathbf{a} \cdot \mathbf{b}$$

para ello se deben usar las propiedades de los senos y cosenos, en particular la igualdad $c^2 + s^2 = 1$.

3

Demuestra que, en 2D, la matriz de rotación por un ángulo ϕ (que llamamos R_ϕ) tiene como inversa la matriz de rotación por un ángulo $-\phi$. Es decir:

$$\left(R_\phi\right)^{-1} \;=\; R_{-\phi}$$

Para esto, usa las propiedades de simetría y anti-simetría de la funciones seno y coseno, es decir, las igualdades $\cos(-\phi) = \cos(\phi)$ y $\sin(-\phi) = -\sin(\phi)$, que son ciertas para cualquier ϕ, así como los datos del enunciado del problema 2.

Indica si el resultado también es cierto en 3D para rotaciones cuyo eje son los ejes de coordenadas, y para rotaciones arbitrarias.

4

Demuestra que en 2D las rotaciones no modifican la longitud de un vector, es decir, que para cualquier ángulo ϕ y vector $\mathbf{v} = (x,y)$ se cumple $\|R_\phi \mathbf{v}\| = \|\mathbf{v}\|$. Describe razonadamente si este resultado puede generalizarse a 3D.

5

Sea ϕ el ángulo entre un vector no nulo $\mathbf{a} = (a_x, a_y, a_z)$ y otro vector no nulo $\mathbf{b} = (b_x, 0, 0)$ (es

paralelo al eje X). **Demuestra** entonces que:

$$\mathbf{a} \cdot \mathbf{b} \quad = \quad \|\mathbf{a}\| \, \|\mathbf{b}\| \cos(\phi)$$

usa el hecho de que, en un triángulo rectángulo, el coseno de uno de sus ángulos es igual a la longitud del cateto contiguo dividido por la longitud de la hipotenusa.

6

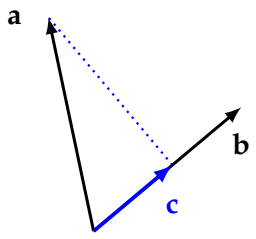
Generaliza el resultado del problema anterior para cualquier par de vectores **a** y **b**, usando la invariancia por rotación demostrada en un problema anterior.

7

Sean **a** y **b** dos vectores no nulos, y sea **c** la componente de **a** paralela a **b** (el vector **c** se suele escribir como $\mathbf{a}_{\|}\mathbf{b}$). **Demuestra** que:

$$\mathbf{c} \quad \equiv \quad \mathbf{a}_{\|}\mathbf{b} \quad = \quad \left(\frac{\mathbf{a} \cdot \mathbf{b}}{\mathbf{b} \cdot \mathbf{b}} \right) \mathbf{b}$$

la relación de **c** con **a** y **b** se aprecia en esta figura, en la que se ve que **c** se obtiene proyectando **a** sobre **b** en dirección perpendicular a **b**



(la figura es en 2D, en 3D es similar si se observa perpendicularmente al plano formado por **a** y **b**)

8

El producto vectorial de dos vectores es una operación binaria (simbolizada por $\times$) que produce un tercer vector perpendicular a los dos primeos. Al igual que el producto esalar, el producto vectorial puede definirse por sus propiedades, en concreto, las siguientes (para cualquier trio de vectores **a**, **b** y **c**, y para cualquier valor real t)

(1) Anti-conmutativa: $\mathbf{a} \times \mathbf{b} \equiv -\mathbf{b} \times \mathbf{a}$

(2) Distributiva respecto de la suma: $\mathbf{a} \times (\mathbf{b} + \mathbf{c}) \equiv \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$

(3) Conmutatividad con producto externo: $(t\mathbf{a}) \times \mathbf{b} \equiv t(\mathbf{a} \times \mathbf{b})$

adicionalmente, puesto que asumimos por definición que los tres ejes o versores del sistema de coordenadas del mundo (**x**,**y** y **z**) son perpendiculares entre sí, entonces como consecuencia debemos asumir también que se cumplen estas igualdades:

(4) $\mathbf{x} \times \mathbf{y} \equiv \mathbf{z} \quad \mathbf{y} \times \mathbf{z} \equiv \mathbf{x} \quad \mathbf{z} \times \mathbf{x} \equiv \mathbf{y}$

(5) $\mathbf{x} \times \mathbf{x} \equiv \mathbf{y} \times \mathbf{y} \equiv \mathbf{z} \times \mathbf{z} \equiv (0,0,0)$

Demuestra, usando estos 5 axiomas, que si $\mathbf{a} = (x_0, y_0, z_0)$ y $\mathbf{b} = (x_1, y_1, z_1)$ entonces:

$$\mathbf{a} \times \mathbf{b} \quad \equiv \quad (y_0 z_1 - z_0 x_1 \, , \, z_0 x_1 - x_0 z_1 \, , \, x_0 y_1 - y_0 x_1 \,)$$

9

Sea R_ϕ una rotación entorno al eje Z (un ángulo ϕ) en 3D. Demuestra que el producto escalar es invariante por ese tipo rotaciones, es decir:

$$(R_\phi \mathbf{a}) \times (R_\phi \mathbf{b}) \quad = \quad R_\phi(\mathbf{a} \times \mathbf{b})$$

¿ se puede generalizar el resultado a cualquier rotación en 3D ?

10

Supongamos que $\mathbf{a} = (a_x, a_y, a_z)$ y $\mathbf{b} = (b_x, b_y, b_z)$ son dos vectores cualesquiera no nulos, siendo ϕ el ángulo entre ellos. **Demuestra** que:

$$\|\mathbf{a} \times \mathbf{b}\| \quad = \quad \|\mathbf{a}\| \|\mathbf{b}\| \, \sin(\phi)$$

realiza la demostración para el caso particular en el cual $\mathbf{a} = \mathbf{x}$ y $\mathbf{b}$ está en el plano XY (es decir $b_z = 0$), y después generalizalo a cualquier par **a** y **b** usando las propiedades que hemos visto del producto vectorial.

11

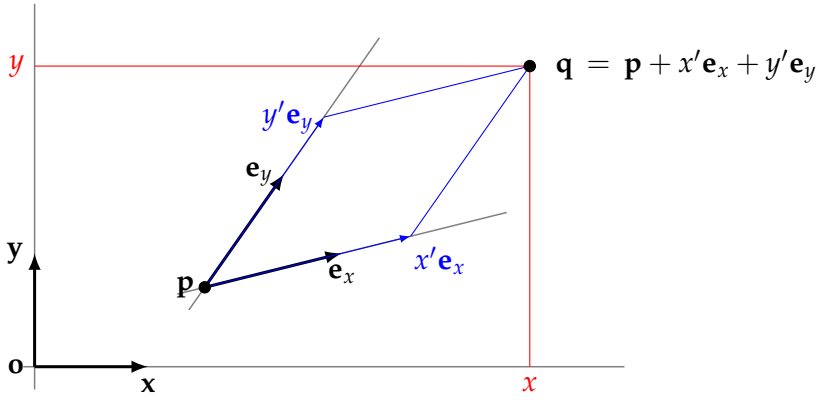
Consideremos en 3D tres vectores ($\mathbf{e}_x$,$\mathbf{e}_y$ y $\mathbf{e}_z$) linealmente independientes, es decir ninguno de ellos puede expresarse como combinación lineal de los otros dos, y consideremos un punto **p** cualquiera. Las coordenadas de los vectores y el punto son las que se indican a continuación:

$$\mathbf{e}_x \equiv (e_{xx}, e_{xy}, e_{xz}) \quad \mathbf{e}_y \equiv (e_{yx}, e_{yy}, e_{yz}) \quad \mathbf{e}_z \equiv (e_{zx}, e_{zy}, e_{zz}) \quad \mathbf{p} \equiv (p_x, p_y, p_z)$$

Estas coordenadas son relativas al origen $\mathbf{o} = (0,0,0)$ y los ejes **x**,**y** y **z**. Los vectores $\mathbf{e}_x$,$\mathbf{e}_y$ y $\mathbf{e}_z$, junto con $\mathbf{p}$ (como origen), forman un **sistema (o marco) de referencia local**, o también un *sistema de coordenadas local* (*reference frame* o *local frame*), que llamaremos $R \equiv (\mathbf{p}, \mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$. Cualquier punto $\mathbf{q}$ puede expresarse usando tres coordenadas $(x', y', z')_R$ relativas al sistema dicho referencia local R, esto es lo mismo que decir:

$$\mathbf{q} \quad = \quad \mathbf{p} + x'\mathbf{e}_x + y'\mathbf{e}_y + z'\mathbf{e}_z$$

en esta figura se observa la relación de las coordenadas locales (x', y', z') con los vectores descritos (en 2D, ignorando la Z)



Describe como calcular las coordenadas (x, y, z) de $\mathbf{q}$ respecto del sistema de coordenadas global $(\mathbf{o}, \mathbf{x}, \mathbf{y}, \mathbf{z})$, a partir de las coordenadas (x', y', z') relativas al sistema de coordenadas local $R = (\mathbf{p}, \mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$

12

Considera la transformación T que se ha visto en el problema anterior. Escribe razonadamente una expresión matricial para la transformación inversa T^{-1} (similar a la obtenida para T), transformación que convierte las coordenadas (x, y, z) de $\mathbf{q}$ (relativas al sistema de referencia global) en las coordenadas (x', y', z'), relativas al sistema de referencia local R. Suponer ahora que los vectores $\mathbf{e}_x$,$\mathbf{e}_y$ y $\mathbf{e}_z$ son de longitud unidad y además perpendiculares entre si.

Para este problema, ten en cuenta que el vector $\mathbf{q} - \mathbf{p}$ puede escribirse como suma de tres de vectores, cada uno de ellos paralelo a $\mathbf{e}_x$, $\mathbf{e}_y$ y $\mathbf{e}_z$, respectivamente. Para ello se puede usar el operador $\|$ que hemos visto antes (que usa proyección perpendicular a un vector), lo cual nos da fácilmente las coordenadas que se piden.

13

Escribe las transformaciones T y T^{-1} como transformaciones lineales en coordenadas homogéneas

usando matrices 4x4.

14

En el problema anterior (problema 13), la transformación T^{-1} (que transforma coordenadas globales en locales) solo es válida si los ejes locales ($\mathbf{e}_x$,$\mathbf{e}_y$ y $\mathbf{e}_z$) son perpendiculares entre si y de longitud unidad. **Escribe** la matriz correspondiente a esa transformación en cada uno de estos dos casos:

(a) Los ejes del sistema de coordenadas local son perpendiculares pero no necesariamente de longitud unidad

(b) Los ejes del sistema de coordenadas local son tres vectores cualesquiera, no necesariamente perpendiculares ni de longitud unidad (simplemente se les exige que sean linealmente independientes).

15

Supongamos que tenemos definida una clase **SistRefLocal** para almacenar los cuatro vectores que definen un sistema de referencia local. Cada instancia de **SistRefLocal** tiene cuatro miembros publicos de nombre `orig` (origen), `ejex`, `ejey` y `ejez` (ejes locales X,Y y Z), todos de tipo **_vertex3f**

Con esta definición de clase, **escribe el código** de un método público de **SistRefLocal** con esta cabecera

```
void SistRefLocal::apilar_matriz_locales_a_mundo()  { /* ... */ }
```

este método compone la matriz de transformación actual de OpenGL con la matriz de transformación de coordenadas locales a coordenadas de mundo, de forma que a partir de su llamada los vértices enviados con **glVertex** verán sus coordenadas interpretadas como coordenadas locales del sistema de referencia (siempre que el método se llame en modo *modelview*).

Para este método es útil usar la función de OpenGL **glMultMatrixf**. Es necesario tener en cuenta que **glMultMatrixf** recibe como parámetro un puntero a un vector de 16 valores tipo **GLfloat** (llamémosles $m_0, m_1, \ldots, m_{15}$) que codifican una matriz 4×4 puesta por columnas, es decir, la siguiente matriz M:

$$M = \begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

16

Sea $\mathbf{p} = (x,y,z)$ un punto cualquiera, con $z < 0$. Si proyectamos ese punto en el plano $z = 0$ (usando proyección perspectiva con foco en el punto $(0,0,1)$) ¿ cuales son las coordenadas del punto proyectado $\mathbf{p}' = (x',y',0)$?

Eescribe la solución al problema anterior interpretando la proyección como una transformación lineal en coordenadas homogéneas (escribe la matriz 4x4 asociada a la proyección).

17

En el problema anterior, supongamos que no hacemos $z' = 0$ sino que hacemos $z' = g(z) = z/(1-z)$.

- Escribe la correspondiente matriz de transformación en coordenadas homogéneas.
- Escribe el rango de valores que se obtiene para z', teniendo en cuenta que $z < 0$, es decir $z \in (-\infty,0)$.
- Además, demuestra que si $z_0 < z_1 < 0$ entonces $z'_0 < z'_1 < 0$ (donde $z'_0 = g(z_0)$ y $z'_1 = g(z_1)$). Esto equivale a verificar que la proyección conserva el orden en Z.

18

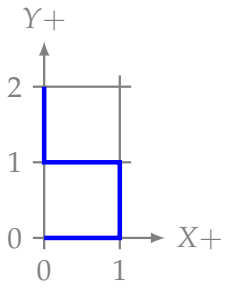
En el problema anterior, supongamos que existen dos valores positivos n y f, con $0 < n < f$ de forma que sabemos que los puntos a proyectar están a una distancia del observador (el foco de la proyección) entre n y f (ambos incluidos). Esto es lo mismo que decir que los valores de z cumplen $1-f \leq z \leq 1-n$ (ya que el observador está a una unidad del origen en Z+). En estas condiciones, responde a estas dos cuestiones:

- ¿ cual será el rango de valores de $g(z)$?
- ¿ como podríamos hacer para que, después de la proyección, los valores de Z estuviesen siempre en el rango $(0,1)$, siendo $g(-n+1) = 0$ y $g(-f+1) = 1$?

19

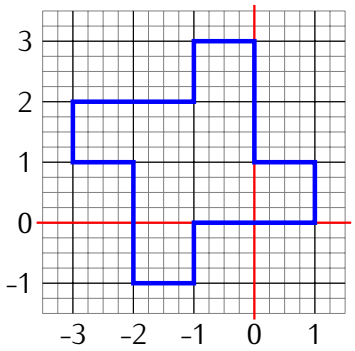
Escribe una función en OpenGL, llamada **gancho** para dibujar con OpenGL la figura azul de la izquierda (en la cual cada segmento recto tiene longitud unidad, y el extremo inferior está en el

origen), usando para ello **glBegin**, **glVertex2f** y **glEnd** (la función debe ser neutra respecto de la matriz modelview, el color o el grosor de la línea, es decir, usará la modelview, el color y grosor del estado de OpenGL en el momento de la llamada).



20

Usando (exclusivamente) la función **gancho** del problema anterior, construye otra función (**gancho_x4**) para dibujar con OpenGL el polígono que aparece en la figura:



En esta función no se podrá usar **glPushMatrix** ni **glPopMatrix**, se usará exclusivamente **glTranslatef** y **glRotate** (la longitud de cada segmento seguirá siendo la unidad). Hay que tener en cuenta que la figura se puede obtiener exclusivamente usando rotaciones de la original, pero en esas rotaciones el centro no es el origen.

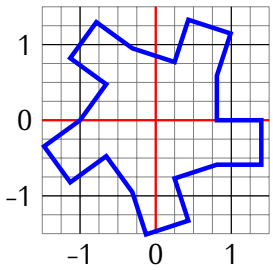
21

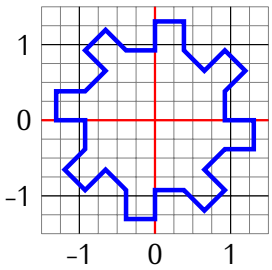
Escribe el pseudocódigo OpenGL otra función (**gancho_2p**) para dibujar esa misma figura, pero escalada y rotada de forma que sus extremos coincidan con dos puntos arbtrarios $\mathbf{p}_0 = (x_0, y_0)$ y

$\mathbf{p}_1 = (x_1, y_1)$, que se pasan como parámetro a dicha función.

22

Usa la función del problema anterior para construir estas dos nuevas figuras, en las cuales hay un número variable de instancias de la figura original, dispuestas en circulo (vemos los ejemplos para 5 y 8 instancias, respectivamente).





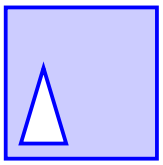
Tema 2

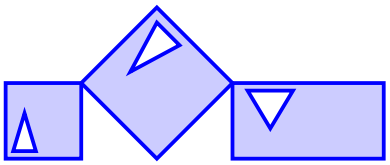
Problemas del tema 2

23

Supón que dispones de una función llamada `figura_simple` que dibuja con OpenGL la figura que aparece a la izquierda (un cuadrado de lado unidad con la esquina inferior izquierda en el origen, con un triángulo inscrito). Esta función llama a **`glVertex`** para especificar las posiciones de los vértices de los polígonos de dicha figura.

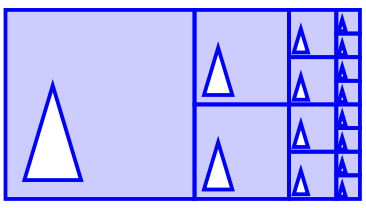
Usando exclusivamente llamadas a dicha función, construye otra función llamada **`figura_compleja`** que dibuja la figura de la derecha. Para lograrlo puedes usar manipulación de la pila de la matriz modelview (**`glPushMatrix`** y **`glPopMatrix`**), junto con **`glTranslatef`** y **`glScalef`**.





24

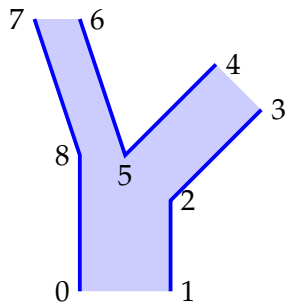
Usando de nuevo la función `figura_simple`, escribe otra función llamada **`figura_compleja_rec`**. Esta nueva función dibuja la figura que aparece aquí.



El código incluirá una única llamada a `figura_simple`, que se ejecutará 15 veces. Resuelve el problema con una función recursiva.

25

Escribe el código OpenGL de una función (llamada **`tronco`**) que dibuje la figura que aparece a aquí. El código dibujará el polígono relleno de color azul claro, y las aristas que aparecen de color azul oscuro (ten en cuenta que no todas las aristas del polígono relleno aparecen).

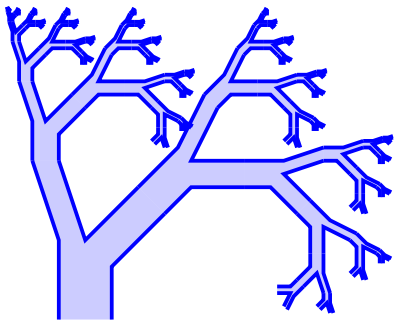


Índice de vértice	Coordenadas de vértice
0	$(+0.0, +0.0)$
1	$(+1.0, +0.0)$
2	$(+1.0, +1.0)$
3	$(+2.0, +2.0)$
4	$(+1.5, +2.5)$
5	$(+0.5, +1.5)$
6	$(+0.0, +3.0)$
7	$(-0.5, +3.0)$
8	$(+0.0, +1.5)$

A la derecha aparecen las coordenadas de los 9 vértices, empezando en el vértice en el extremo inferior izquierdo, y continuando en el sentido contrario a las agujas del reloj.

26

Usando el código del problema anterior, escribe una función **`arbol`** que, llamando a **`tronco`** (y manipulando la pila de matrices modelview), dibuje el árbol que aparece en la figura de la derecha.
En este árbol cada tronco conecta con dos copias del mismo más pequeñas situadas en sus dos ramas. El tronco raíz es semejante al del problema anterior. En total hay 63 troncos de 6 tamaños o niveles distintos. Diseña el código usando recursividad, de forma que el número de niveles sea un parámetro modificable en dicho código.



27

Supongamos que tenemos una malla de triangulos representada en memoria como un vector de vértices (**`ver`**) y un vector de índices de vértices de cada triángulo (**`ivt`**), de forma que:

- hay un total de n vértices, y m triángulos.

- la expresión **ver**$[i]$ es una tupla con la coordenadas cartesianas del punto donde está el i-ésimo vértice, con i entre 0 y $n-1$.
- la expresión **ivt**$[j]$**._**k designa el índice del k-ésimo vértice del j-ésimo triángulo, con k igual a 0,1 o 2, y j entre 0 y $m-1$ (ambos incluidos).

Escribe el pseudocódigo de la función para calcular la tabla de normales de triángulos, que será un vector **nort** con m tuplas, cada una de ellas con las coordenadas de la normal de un triángulo (asumir que se diponen de los operadores usuales de tuplas o vectores, es decir $+, -, \cdot, \times, \| \ \|, \ldots$). Estas normales deben quedar normalizadas.

Asumir que los vectores que se usan son todos ellos vectores dinámicos, que se pueden redimensionar o a los que podemos añadirles entradas al final.

28

Extiende el pseudo-código de la función anterior para calcular también la tabla de normales de vértices, **norv**, con n entradas (los vértices quedan normalizados). Intenta encontrar una solución simple que no requiera tablas auxiliares.

29

Algunos autores recomiendan que, para el cálculo de la normales de un vértice, en lugar de promediar usando el mismo peso para todas las normales de los triángulos adyacentes, lo que se haga sea usar pesos distintos para cada normal de triángulo, en concreto pesos proporcionales al área de cada triángulo.

Escribe una versión del pseudo-código del problema anterior que implemente esta opción.

30

Escribe el pseudo-código de una función para calcular la tabla de aristas de la malla del problema anterior. Esta tabla será un vector **ari**, con tantas entradas como aristas. La expresión **ari**$[i]$**._**j es un entero con el índice del j-ésimo vértice de la i-ésima arista, con j siendo 0 o 1, e i entre 0 y $a-1$, donde a es el número de aristas.

Escribe una solución que no use tablas auxiliares, aunque la complejidad en tiempo sea alta.

31

Examina y describe razonadamente el orden de complejidad en tiempo de tu solución al problema anterior. Si dicha complejidad es superior a $O(a)$ (donde a es el número de aristas), escribe una nueva versión del algoritmo con tiempo en $O(a)$ (tiempo proporcional al número de aristas), aunque se emplee (durante la ejecuión del algoritmo) más memoria que en la primera versión (usa tablas auxiliares temporales).