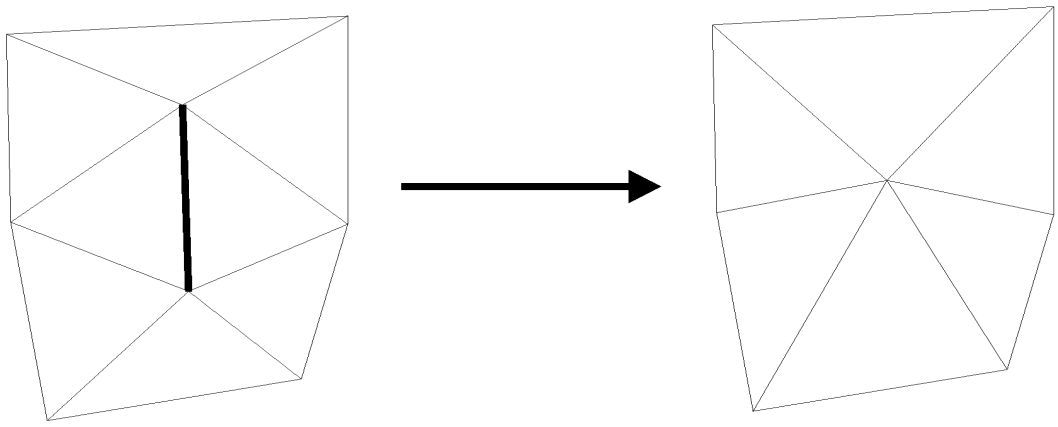
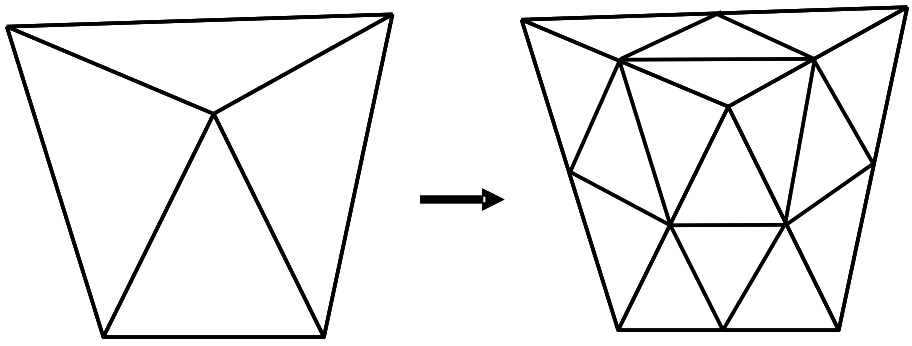
(1) Dado un objeto descrito con vértices y caras (vector<_vertex3f> Vertices; vector<_vertex3i> Triangulos), escribir en pseudocódigo el programa que comprueba si el objeto es abierto o cerrado (abierto=tiene aristas con una sola cara)

(2) Una técnica para reducir el número de caras de un modelo es el colapso de aristas. Dado un objeto descrito con vértices y caras (vector<_vertex3f> Vertices; vector<_vertex3i> Triangulos), escribir en pseudocódigo el programa que lo realizaría



(3) Para un objeto descrito con vértices y caras (vector<_vertex3f> Vertices; vector<_vertex3i> Triangulos), escribir en pseudocódigo un algoritmo que permita añadir nuevos triángulos, cuatro por cada triángulo inicial, partiendo de los puntos medios de cada arista (ver imagen de ejemplo). Indicar como han de ser las nuevas listas de vértices y caras (hay que tener en cuanta que no se han de repetir vértices).



(4) Supongamos un objeto descrito con listas de vértices y caras (vector<_vertex3f> Vertices; vector<_vertex3i> Triangulos) y un punto donde se sitúa el observador. Implementar en pseudocódigo el algoritmo de eliminación de las caras traseras (back face culling) que permite seleccionar las caras visibles a un observador. ¿Qué listas ha de devolver el algoritmo?

(5) Dado un objeto descrito con vértices y caras (vector<_vertex3f> Vertices; vector<_vertex3i> Triangulos), escribir en código en C o C++ el programa que centre el objecto con respecto a su caja frontera.

(6) Queremos dibujar en modo alambre un malla indexada de triángulos, que corresponde a una superficie cerrada (es decir, una en la que cada arista es compartida por exactamente dos caras). El método "estándar" consiste en recorrer todas las caras, y para cada cara dibujar sus aristas. El pseudo-código puede ser este:

Para cada triángulo T de la malla:
Recuperar los tres índices (i,j,k) de los vértices de
Recuperar las coordenadas del mundo (a,b,c) de los vértices con índices (i,j,k)
Dibujar segmento desde 'a' hasta 'b',
Dibujar segmento desde 'b' hasta 'c'.
Dibujar segmento desde 'c' hasta 'a'.

Por tanto, el método descrito dibuja dos veces cada arista, y tarda casi el doble del tiempo necesario. En este contexto, responde a estas cuestiones:

a) Describe razonadamente un método alternativo de dibujo que solamente visualice cada arista una única vez para este tipo de mallas (lo más sencillo posible).
b) Escribe el método en pseudo-código.
c) Escribe una implementación en OpenGL de ese método, usando glBegin/glEnd o glDrawElements

(7) Considera una malla de n triángulos almacenada en memoria con un vector caras (con n entradas), de forma que caras[i][j] es un entero, en concreto el índice del vértice número j de la cara número i (con $0 \leq i < n$ y $0 \leq j < 3$).

- Con esta definición, escribe el código de una función con esta declaración:

  bool comparten_vertice(int c1, int c2) ;

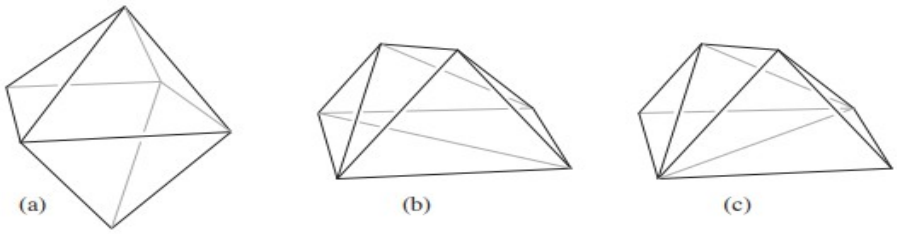
  que devuelve true cuando las caras número c1 y c2 comparten un vértice (devuelve false si esto no es así).

- Escribe el código de otra función

  bool comparten_arista(int c1 , int c2) ;

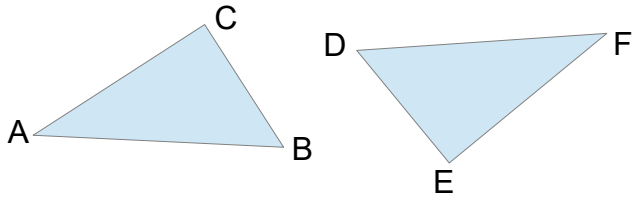
  que devuelve true cuando las caras número c 1 y c 2 comparten una arista (devuelve false si esto no es así).

(8) Mira estas tres mallas:



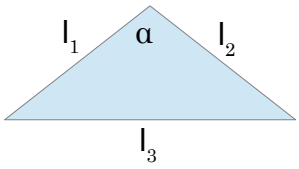
a) ¿Cuales comparten topología? ¿Cuales geometría?
b) Define la tabla de caras cada uno de ellos. No es necesario dar coordenadas de vértices, tan solo marcarlos en el dibujo qué indice tiene cada uno.

(9) Escribir en código C o C++ una función que determine si dos triángulos, ΔABC y ΔDEF, definidos por puntos en 3D, son congruentes o no. Dos triángulos son congruentes si tienen la misma forma y tamaño aunque distinta orientación, es decir, sus correspondientes lados y ángulos son iguales.



Nota: para calcular el ángulo entre dos lados de un triángulo se usa el teorema del coseno, sean l_1,l_2,l_3 los lados de un triángulo y α el ángulo entre los lados l_1 y l_2, tenemos:

$$\alpha = \arccos\left(\frac{(l_1)^2+(l_2)^2-(l_3)^2}{2\,l_1\,l_2}\right)$$



(10) Se dispone de una malla de triángulos descrita con vértices y caras (vector<_vertex3f> Vertices; vector<_vertex3i> Triangulos). Diseña un método o función limpiar() que permita detectar y corregir vértices duplicados (si dos vértices tienen la misma posición se deben sustituir por uno único).

(11) Diseña un algoritmo que permita detectar si todas las caras de un malla tienen orientación consistente (todas con sentido horario o todas con sentido antihorario). ¿Cómo se puede corregir la malla para homogeneizarla en caso de no ser consistente?