Informática Gráfica. Grupo C Curso 2015-16. Teoría. (25/01/16)

Apellidos: _____Nombre: _____

1. **(2.5pt)** Escriba el código necesario para disponer de la funcionalidad "Zoom +" y "Zoom -", tanto en una cámara con proyección perspectiva como en una cámara con proyección ortogonal.

Suponiendo que para proyección perspectiva se utiliza una llamada con los siguientes parámetros:

```
glFrustum(left, right, bottom, top, near, far);
```

y para la proyección ortogonal se utiliza una llamada con los siguientes parámetros

```
glOrtho(left, right, bottom, top, near, far);
```

siendo `left,right, bottom,top,near` y `far` atributos de una clase `Camara`, tendríamos los siguientes métodos:

```
void Camara::zoomMas( ) {
    left/=1.1;
    right/=1.1;
    bottom/=1.1;
    top/=1.1;
}

void Camara::zoomMenos( ) {
    left*=1.1;
    right*=1.1;
    bottom*=1.1;
    top*=1.1;
}
```

2. (2.5pt) Supongamos que 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 T
  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).

Cada arista aparece dos veces (en dos triángulos). Como siempre están en sentido antihorario, en uno aparecerá el segmento en el orden i-*j* y en el otro aparecerá como j-*i,* es suficiente con establecer un criterio que sólo pinte una de las dos apariciones. P.ej., *"pintar la arista sólo si j>i"*

b) Escribe el método en pseudo-código.

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

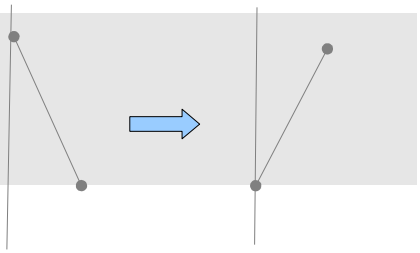
c) Escribe una implementación en OpenGL de ese método, usando glBegin/glEnd o glDrawElements (incluir esto último depende de si han visto en clase glDrawElements)

```
// SOLUCION 1:  Con glBegin/ glEnd
for (t=0; t<triangles.size();t++) {
      v1=vertices[t.i]; v2=vertices[t.j];v3=vertices[t.k];
      glBegin(GL_LINES);
      if (t.j>t.i) {
            glVertex3f(v1.x,v1.y,v1.z);
            glVertex3f(v2.x,v2.y,v2.z);
      }
      if (t.k>t.j) {
            glVertex3f(v2.x,v2.y,v2.z);
            glVertex3f(v3.x,v3.y,v3.z);
      }
      if (t.i>t.k){
            glVertex3f(v3.x,v3.y,v3.z);
            glVertex3f(v1.x,v1.y,v1.z);
      }
      glEnd();
}

// SOLUCION 2: Con glDrawArrays
// *************** En el constructor ***********
std::vector<float> aristas;
for (t=0; t<triangles.size();t++) {
   v1=vertices[t.i]; v2=vertices[t.j];v3=vertices[t.k];
   if (t.j>t.i) {
      aristas.pushBack(v1.x); aristas.pushBack(v1.y);aristas.pushBack(v1.z);
      aristas.pushBack(v2.x); aristas.pushBack(v2.y);aristas.pushBack(v2.z);
   }
   if (t.k>t.j) {
      aristas.pushBack(v2.x); aristas.pushBack(v2.y);aristas.pushBack(v2.z);
      aristas.pushBack(v3.x); aristas.pushBack(v3.y);aristas.pushBack(v3.z);
      }
   if (t.i>t.k){
      aristas.pushBack(v3.x); aristas.pushBack(v3.y);aristas.pushBack(v3.z);
      aristas.pushBack(v1.x); aristas.pushBack(v1.y);aristas.pushBack(v1.z);
      }
}

// **********en el método de dibujar ***********
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, aristas);
glDrawArrays(GL_LINES, 0, 3);
```

3. **(2.5pt)** Tenemos un cono de altura d, con el ápice en (0,d,0) y la base en el plano y=0. Realizar en pseudocódigo el programa para una animación que transforme el cono anterior, en un cono con ápice en (0,0,0) y base en el plano y=d.



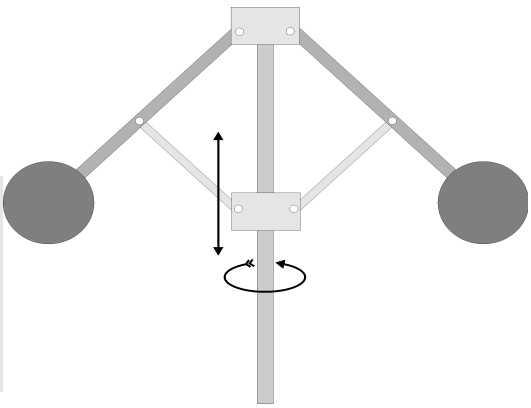
```
// Solución 1: tenemos el cono como una malla triangular
// y suponiendo que el ápice es el vértice de índice [0]
void Objeto3D::Animacion(int pasoActual, int totalPasos ) {
      float fraccion=(float)pasoActual/(float)totalPasos;
      bool invertida=false;
      if (v[0].y > 0) {
            v[0].y-=d * (1-fraccion);
            for (i=1;i<v.size();i++)
                  v[i].y=d*fraccion;

            if ((v[0].y< v[1].y) && !invertida) {
                  invertirCaras();
                  invertida=true;
            }
            calcularNormales( );
      }

      redibujar( );
}
```

```
// Solución 2: Recalcular la revolución
void Objeto3D::Animacion(int pasoActual, int totalPasos ) {
      float fraccion=(float)pasoActual/(float)totalPasos;
      if (v[0].y > 0) {
            v[0].y-=d * (1-fraccion);
            v[1].y=d*fraccion;
            this->generarRevolucion(v,20);
      }
      redibujar();
}
```

4. **(2.5pt)** Generar el grafo de escena incluyendo las transformaciones, tal que partiendo de una esfera unidad y un cilindro unidad centrados en el origen permite obtener un modelo de un regulador de Watt. Hacer un dibujo del posicionamiento y dimensiones de las piezas.



Hay varias aproximaciones, similares, que podrían ser correctas. Por lo que no se incluye ninguna como solución "absoluta" al problema.