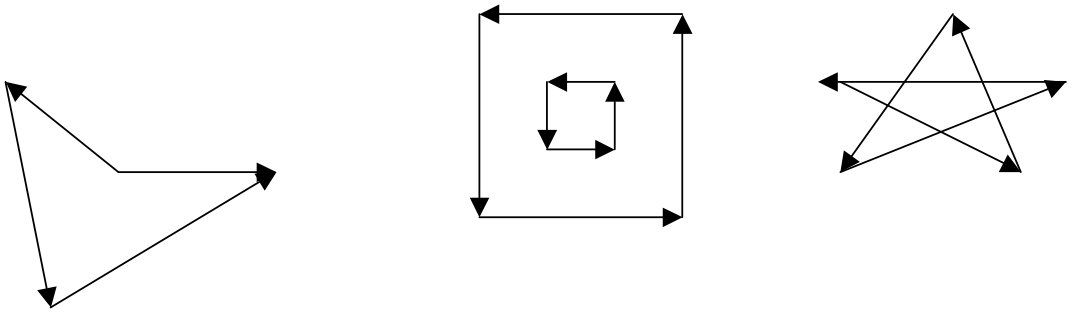
Teselación de polígonos:
OpenGl sólo es capaz de representa polígonos convexos simples:

1. Las aristas sólo se intersectan en los vértices.
2. No hay duplicidad de vértices.
3. En un vértice sólo inciden dos aristas.

En el caso de que tengamos que representar polígonos cóncavos, con aristas que se intersectan o huecos, se deben subdividir primero en polígonos más simples y convexos para que puedan ser representados. Esta subdivisión de polígonos se llama **teselación de polígonos.**



Si un polígono necesita teselación, hay que seguir estos pasos:

1. Crear un objeto teselación nuevo con **gluNewTess().**
2. Usar varias veces **gluTessCallback()** varias veces para registrar las funciones callback para desarrollar operaciones durante la teselación. El caso más complicado para una función callback es cuando el algoritmo de teselación detecta una intersección y debe llamar a la función registrada para GLU_TESS_COMBINE.
3. Espcificar las propiedades de teselación llamando a **gluTessProperty().** Crear y representar los polígonos teselados especificando los contornos de uno o más polígonos cerrados. Si los datos para el objeto son estáticos, encerrarlo en una lista de visualización.
4. Si se necesita teselar algo más, hay que destruir previamente el actual objeto teselación con **gluDeleteTess()**.

Cuádricas en Opengl:
La librería base de Opengl proporciona el soporte modelización y representación de: puntos, líneas, y polígonos convexos rellenos. Sin embargo no dispone de ningún procedimiento que permita representar ningún objeto 3D, ni siquiera círculos 2D. Sin embargo, la libería GLU proporciona rutinas para modelar y representar aproximaciones teseladas de formas 2D y 3D que se calculan con ecuaciones de cuádricas, es decir, una de superficies cuádricas en una variedad de estilos y orientaciones.
Ecuación general de una superficie cuádrica:

$$a_1x^2 + a_2y^2 + a_3z^2 + a_4xy + a_5yz + a_6xz + a_{7x} + a_8y + a_9z + a_{10} = 0$$

Para usar objetos cuádricos hay que seguir los siguientes pasos:

1. Crearlo: **gluNewQuadric()**
2. Especificar las propiedades:

   a. **gluQuadricOrientation()** para controlar la dirección (¿) y diferenciar el interior del exterior.
   b. **gluQuadricStyle()** para elegir entre representar el objeto como puntos, líneas o polígonos rellenos.
   c. **gluQuadricNormals()** para especificar una normal por vértice o una normal por cara. Por defecto no se genera ninguna.
   d. **gluQuadricTexture()** Si se quieren generar coordenadas de textura.
3. Prepararse para poder invocar las rutinas de manipulación de errores.
4. LLamar a la rutina de representación de la cuádrica elegida: gluSphere(), **gluCylinder()**, **gluDisk()**, **gluPartialDisk().** Para mayor eficacia encapsular el objeto en una lista de despliegue si no va a variar.
5. Cuando ya no se va a usar más destruirlo: **gluDeleteQuadric()**

GLU_quadricObj * gluNewquadric(void); crea un nuevo objeto cuádrico y devuelve un puntero a él. Si se produce algún fallo devuelve NULL.

void gluDeleteQuadric(GLUquadricObj *qobj) Destruye el objeto cuádrico qobj y libera cualquier memoria usada por él.

void gluQuadricCallback(GLUquadricObj qobj, GLenum wich, void (*fn)()): Define una función fn a llamarse en circustancias especiales. GLU_ERROR es el único valor legal, por lo que fn se llama si ocurre un error. Si fn es NULL cualquier callback existente se borra. Para GLU_ERROR, fn se llama con un parámetro que es el código de error. gluErrorString se puede usar para convertir el código de error a un string ASCII.

Las siguientes rutinas afectan ala tipo de datos generados por las cuádricas. Se deben usar antes de especifica las primitivas.

void gluQuadricDrawSytle(GLUquadricObj *qobj, GLenum drawStyle) Controla el tipo de representanción del objeto:

drawStyle	Tipo de representación
GLU_POINT	punto en cada vértice
GLU_LINE	Aristas uniendo los vértices
GLU_SILHOUETTE	Líneas, excepto las que separan caras coplanares, se usa con gluDisk, gluPartialDisk
GLU_FILL	Polígonos rellenos, los polígonos se dibujan en el sentido contrario a las agujas del reloj con respecto a las normales. Esto puede ser modificado con gluQuadricOrientation

voidgluQuadricOrientation(GLUquadricObj *qobj, GLenum orientation) Controla la dirección en que apuntan las normales. Orientation vale: GLU_OUTSIDE (por defecto), o GLU_INSIDE. Para glusphere() y gluCylinder(), las definiciones de fuera y dentro son obvias. Para gludisk() y gluPartialDisk(), la cara z positiva se considera fuera.

void gluQuadricNormals(GLUquadricObj *quobj, GLenum normals)

normals	Efecto
GLU_NONE	No genera vectores normales, sin iluminación
GLU_FLAT	Una normal para cada cara, sombreado plano.
GLU_SMOOTH	Una normal por vértice, Gouraud.

void gluQuadricTexture(GLUquadricObj *qobj, GLboolean texture coords): cords puede valer GL_TRUE o GL_FALSE(por defecto) para generar o no coordenadas de textura respectivamente.