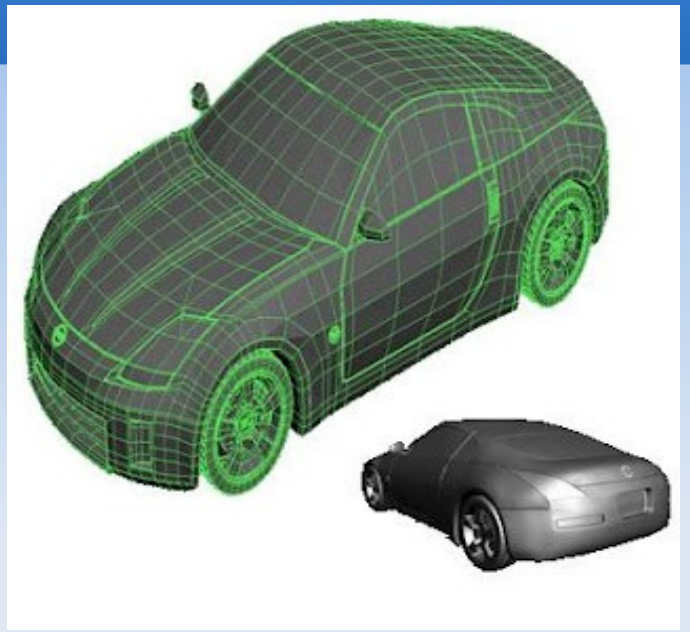
Modelación de objetos 3D
Parte I:
Mallas de polígonos y triangulaciones

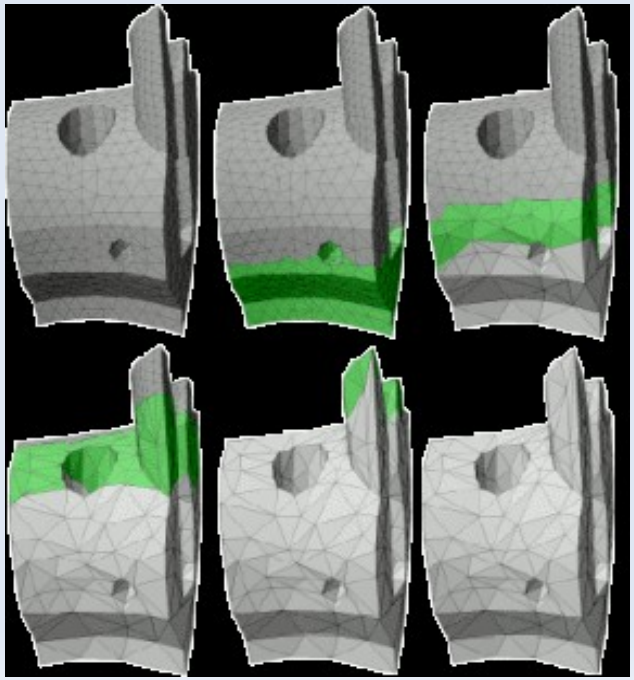
Contenido

- Motivación
- Polígonos simples 3D
- Almacenamiento de mallas de polígonos
- OpenGl y mallas de polígonos
- Triangulaciones
- Triangulaciones y curvas de nivel

Motivación

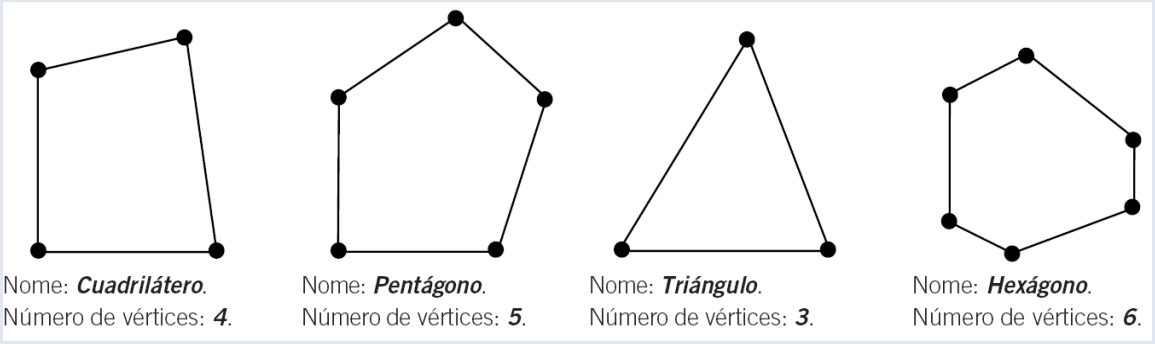
- Qué conceptos geométricos necesitamos?





Polígonos simples 3D

- Figura plana definida por tres o más vértices delimitada por un borde
  - Arcos adyacentes se intersectan solo en uno de sus vértices extremos
  - No hay arcos que se intersecten (excepto caso anterior)
  - Ejemplo: triángulos, cuadriláteros, etc

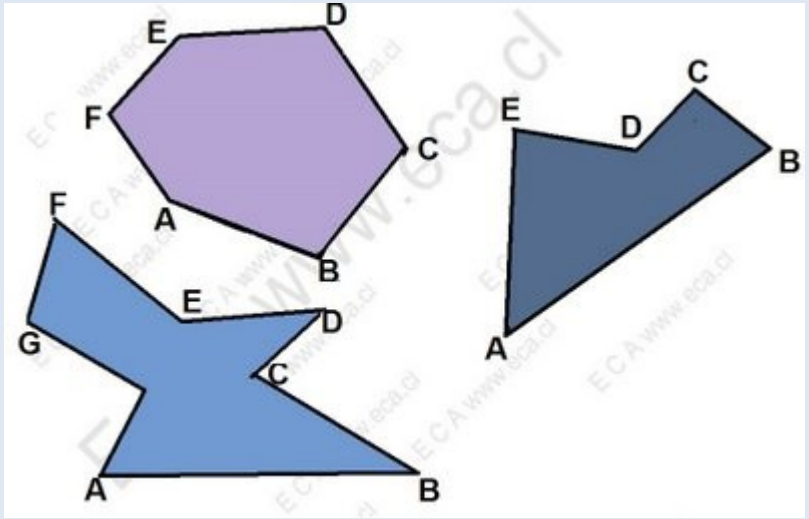


Nome: ***Cuadrilátero***.
Número de vértices: ***4***.

Nome: ***Pentágono***.
Número de vértices: ***5***.

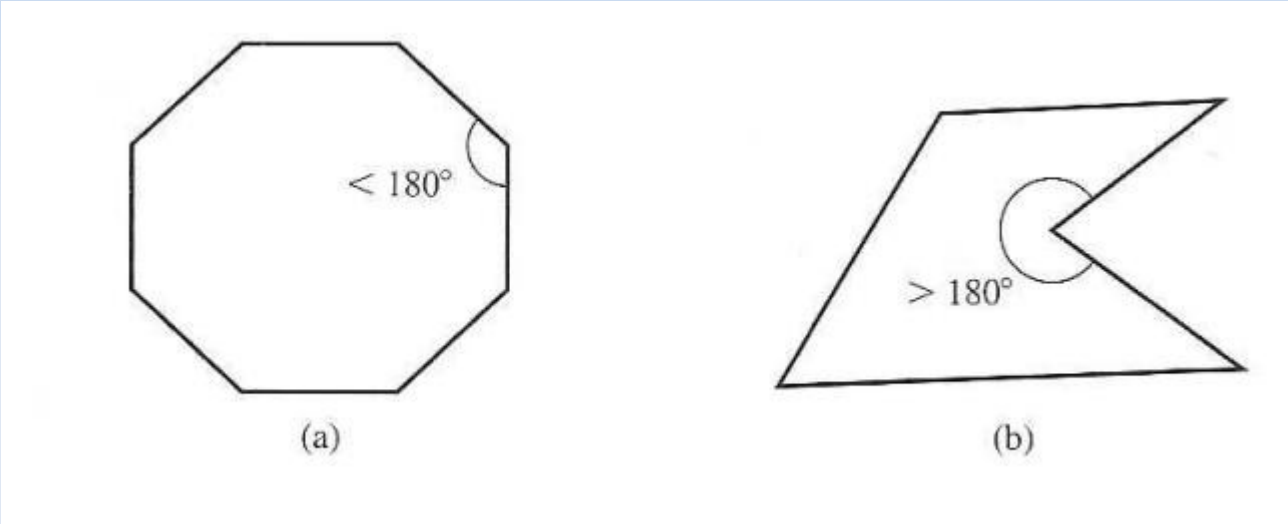
Nome: ***Triángulo***.
Número de vértices: ***3***.

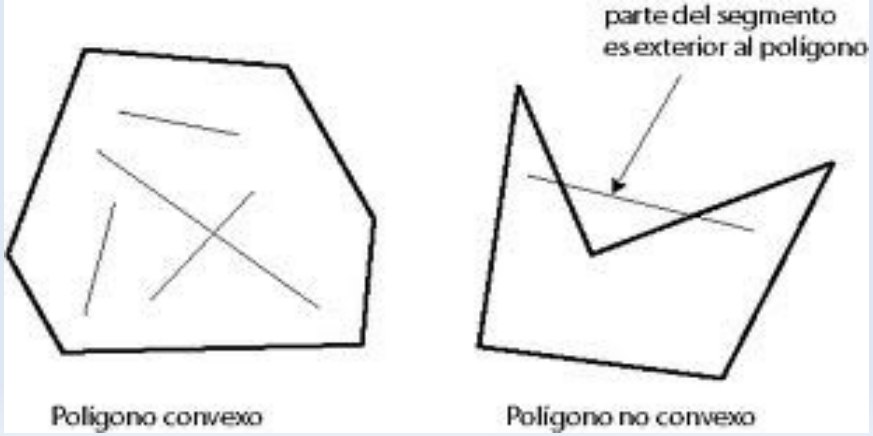
Nome: ***Hexágono***.
Número de vértices: ***6***.



Polígonos simples

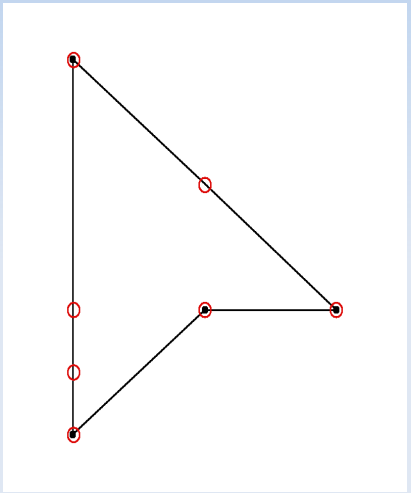
- Clasificación:
  - Convexos: ángulos interiores menor a 180 (a)
  - Cóncavos: (b)

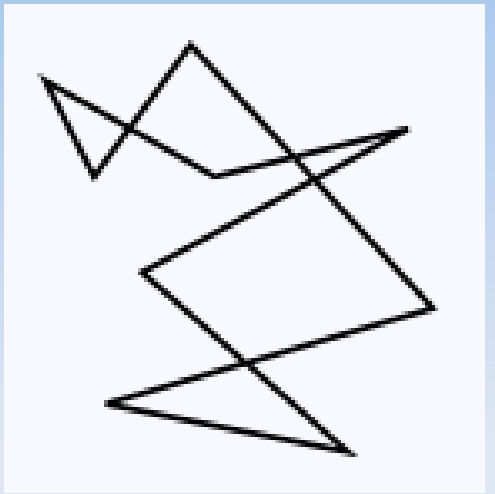




Polígonos simples

- Polígonos degenerados:
  - Puntos colineales
  - Arcos que se cruzan
  - Area igual a 0

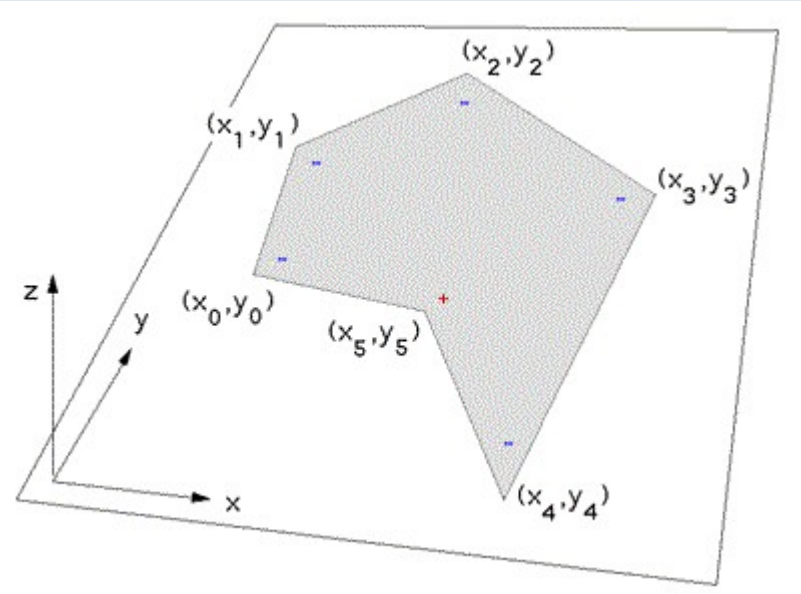




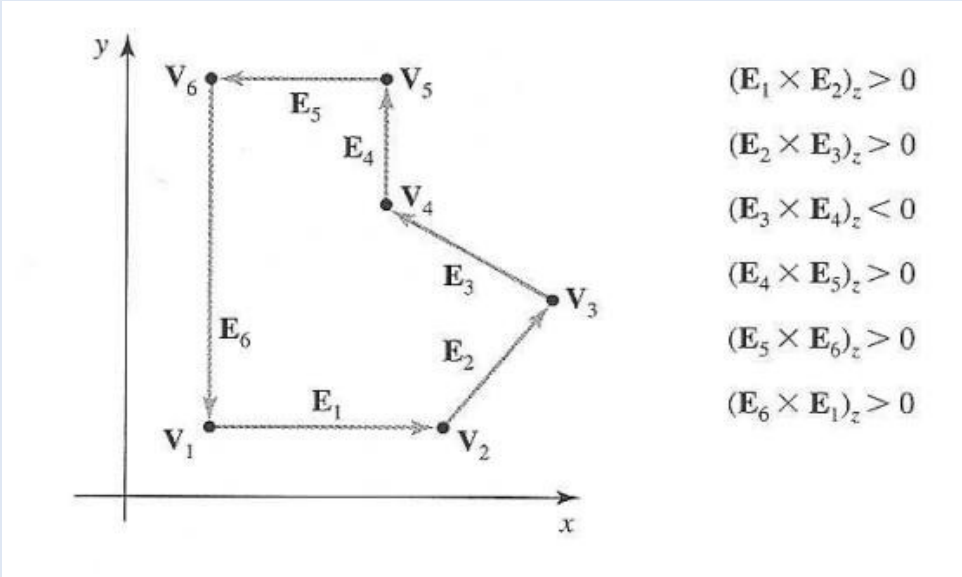
- Importante:
  - Librerías gráficas requieren polígonos simples
  - OpenGl (primitivas de relleno) pueden presentar problemas con polígonos simples concavos
  - Qué pasa si puntos no son coplanares?

Polígonos simples

- Cómo transformar un polígono cóncavo en convexo?
  - Reconocer un polígono concavo? Usar producto cruz entre arcos adyacentes
    - Todos positivos o negativos => convexo



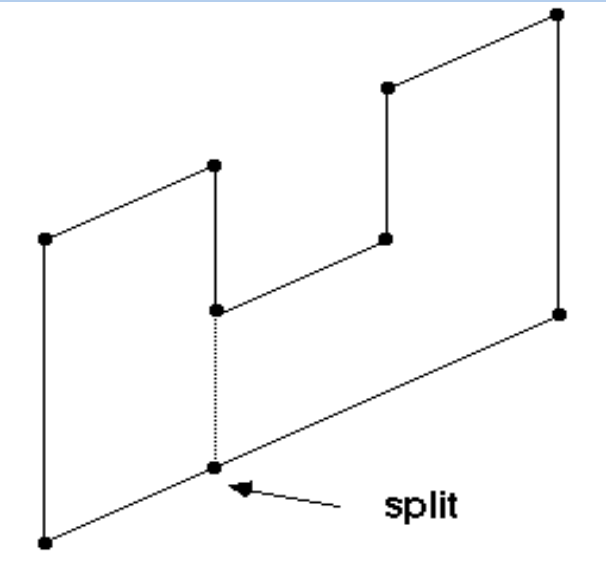
Vértices en orden CW (clock-wise)

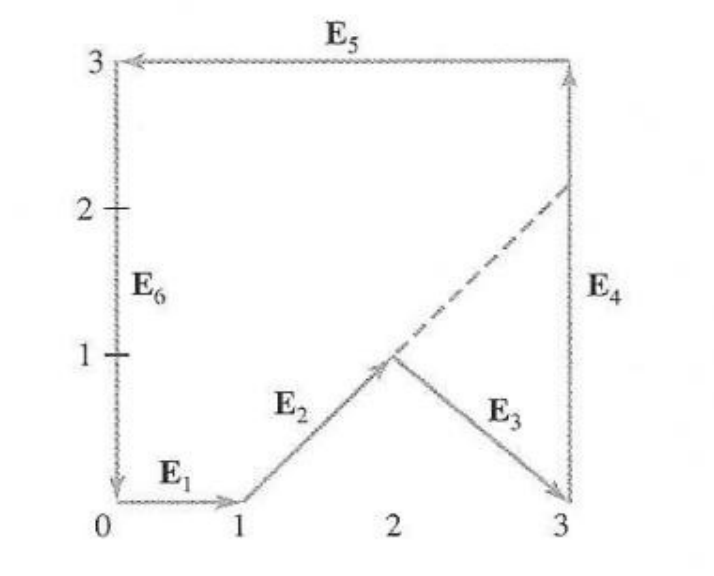


Vértices ne orden CCW (counter clock wise)

Polígonos simples

- Algoritmo:
  - Encontrar vértice reflejo (angulo interior > 180)
  - Dividir el polígono en dos o más polígonos usando la línea que pasa por los arcos que contienen este vértice
  - (Repetir proceso en cada polígono generado hasta que todos los poligonos sean convexos)

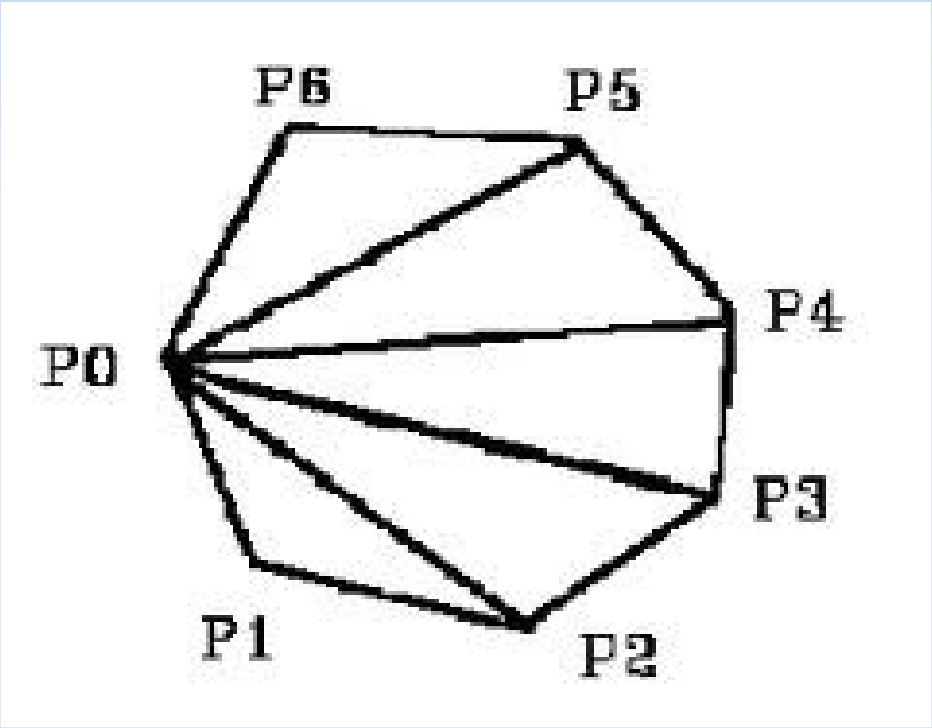




Polígonos simples

- Polígono convexo en triángulos:
  - Escoger P0
  - Formar triángulos P0,Pi,Pi+1, i=1..n-1
  - (n: número de vértices)

- Nota: El uso de triángulos evita tratar polígonos con puntos no coplanares



Almacenamiento de mallas polígonos simples

- Cómo almacenar la información?
  Ideal no repetir la información
- Usar tablas:
  - Tabla de vértices contiene las coordenadas de los puntos
  - Tabla de arcos contiene índices (punteros) a sus puntos extremos en la tabla de vértices
  - Tabla de póligonos contiene índices (punteros) a los arcos en la tabla de arcos

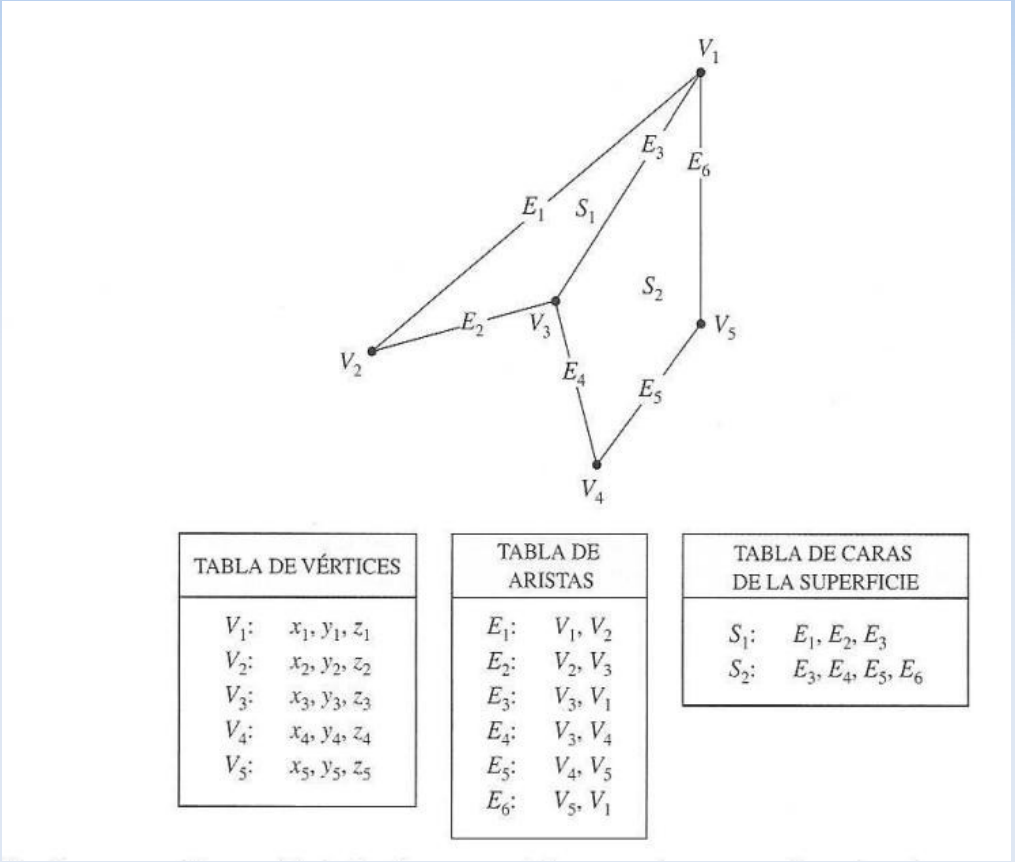


TABLA DE VÉRTICES	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

TABLA DE ARISTAS	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

TABLA DE CARAS DE LA SUPERFICIE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

Almacenamiento de mallas polígonos simples

- Qué otra información puede ser interesante?
  - Vecindad entre triángulos: Arco E_3 es compartido por ambos los polígonos S_1 y S_2

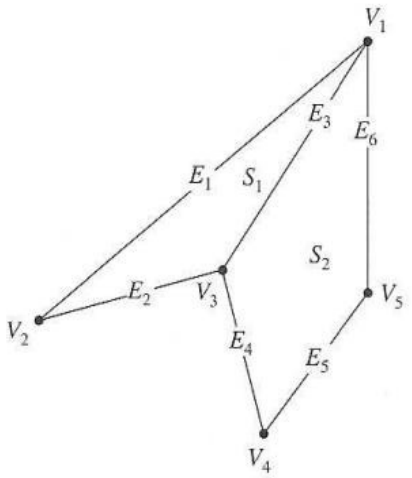


TABLA DE VÉRTICES	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

TABLA DE ARISTAS	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

TABLA DE CARAS DE LA SUPERFICIE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

E_1:	V_1, V_2, S_1
E_2:	V_2, V_3, S_1
E_3:	V_3, V_1, S_1, S_2
E_4:	V_3, V_4, S_2
E_5:	V_4, V_5, S_2
E_6:	V_5, V_1, S_2

Almacenamiento de mallas polígonos simples

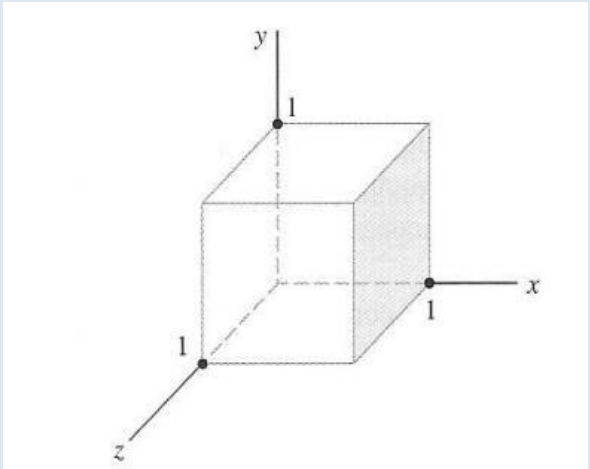
- Para qué es útil?
  - Validar los polígonos (las librerías gráficas suponen que son correctos) y que la malla sea conexa (no hay vértices, arcos ni caras aisladas)
    - Pertenece cada vertice extremo a al menos dos arcos?
    - Pertenece cada arco a uno o dos polígonos?
    - Es cada polígono cerrado?
    - Tiene cada polígono al menos un arco compartido?
    - Cada arco debe referenciar por lo menos un polígono
    - ...
  - Obtener elementos vecinos de manera eficiente (no recorriendo todos los vertices, caras o arcos)

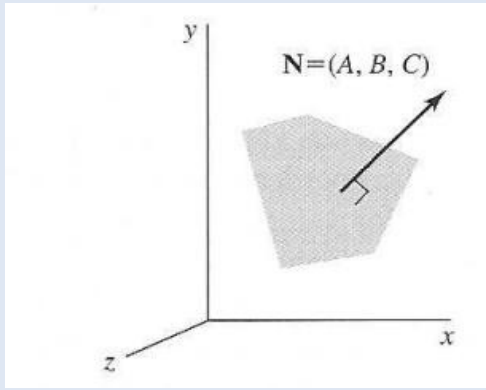
Almacenamiento de mallas polígonos simples

- Conceptos inportantes:
  - Una malla de polígonos tiene una geometría y topología asociada
    - Elementos topológicos: vértices, arcos y caras
    - Elementos geométricos: puntos, segmentos y polígonos
    - La topología asociada la definen las relaciones de vecindad
  - Validación? Debe ser geométrica y topológica

Polígonos: vector normal y cara frontal y posterior

- Cómo calcular el vector normal aproximado de un polígono?
  - Ecuación del plano: $Ax + By + Cz + D = 0$ o $(A,B,C)(x,y,z) = -D$
  - Cada tres vértices calcular una normal
  - Usar como normal del polígono el promedio de las normales
- Cara frontal y posterior del polígono
  - $Ax + By + Cz + D < 0$ => (x,y,z) detrás del plano (polígono)
  - $Ax + By + Cz + D > 0$ => (x,y,z) al frente del plano (polígono)



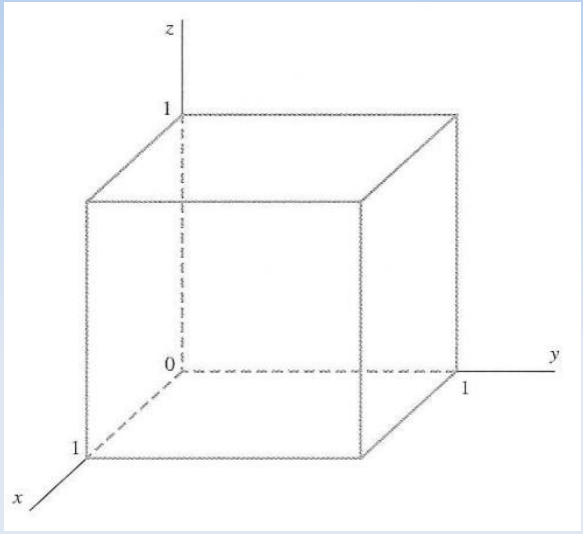


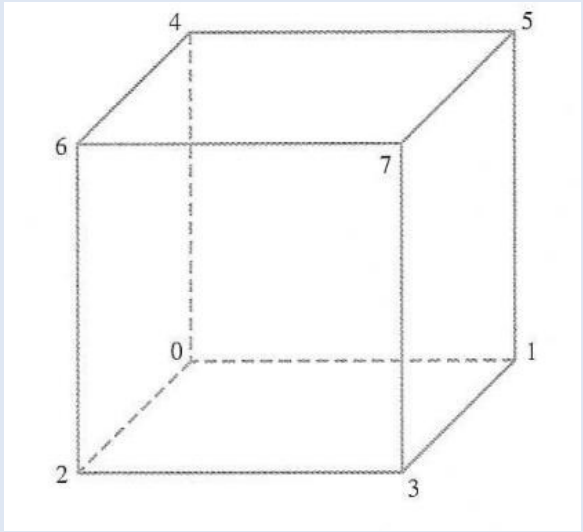
Mallas de polígonos y OpenGl

- Cuando una escena tiene más de cien o mil polígonos, no podemos especificarlos de a uno. En general estos objetos se modelan y crean usando modeladores de superficies y sólidos tales como solid Edge, Blender, SolidWorks, etc o a través de algoritmos de procesamiento de imágenes tales como marching-cubes
- OpenGl provee una manera de dibujar esta escena pasando todos los datos geométricos a través de la llamada a una sola función
- Veamos como dibujar un Cubo con vertex arrays

Mallas de polígonos y OpenGl

- Usando Opengl de la manera vista hasta ahora

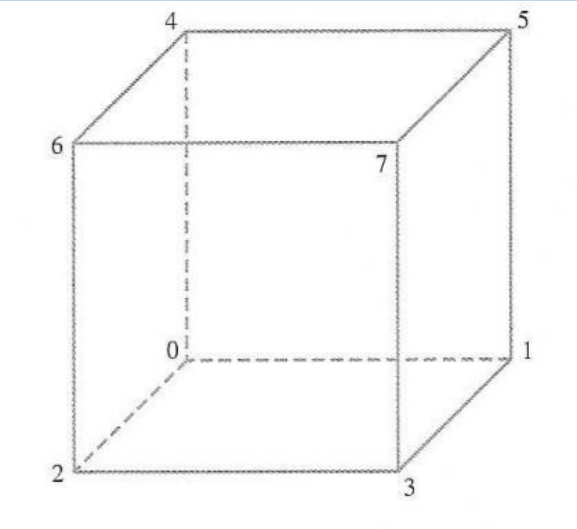





```
typedef GLint vertex3 [3];

vertex3 pt [8] = { {0, 0, 0}, {0, 1, 0}, {1, 0, 0}, {1, 1, 0},
                   {0, 0, 1}, {0, 1, 1}, {1, 0, 1}, {1, 1, 1} };

void quad (GLint n1, GLint n2, GLint n3, GLint n4)
{
    glBegin (GL_QUADS);
        glVertex3iv (pt [n1]);
        glVertex3iv (pt [n2]);
        glVertex3iv (pt [n3]);
        glVertex3iv (pt [n4]);
        glEnd ( );
}

void cube ( )
{
    quad (6, 2, 3, 7);
    quad (5, 1, 0, 4);
    quad (7, 3, 1, 5);
    quad (4, 0, 2, 6);
    quad (2, 0, 1, 3);
    quad (7, 5, 4, 6);
}
```

Mallas de polígonos y OpenGl

- Usando Opengl con vertex array




```
typedef GLint vertex3 [3];

vertex3 pt [8] = { {0, 0, 0}, {0, 1, 0}, {1, 0, 0}, {1, 1, 0},
                   {0, 0, 1}, {0, 1, 1}, {1, 0, 1}, {1, 1, 1} };
```

```
glEnableClientState (GL_VERTEX_ARRAY);
glVertexPointer (3, GL_INT, 0, pt);

GLubyte vertIndex [ ] = (6, 2, 3, 7, 5, 1, 0, 4, 7, 3, 1, 5,
     4, 0, 2, 6, 2, 0, 1, 3, 7, 5, 4, 6);

glDrawElements (GL_QUADS, 24, GL_UNSIGNED_BYTE, vertIndex);
```

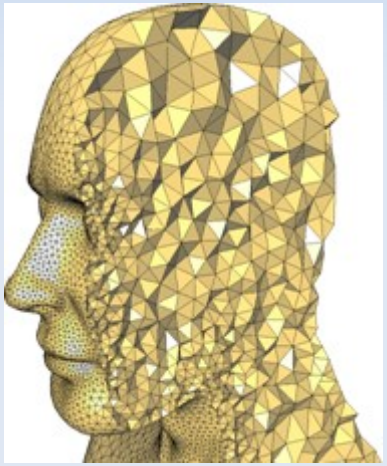
```
glDisableClientState (GL_VERTEX_ARRAY);
```

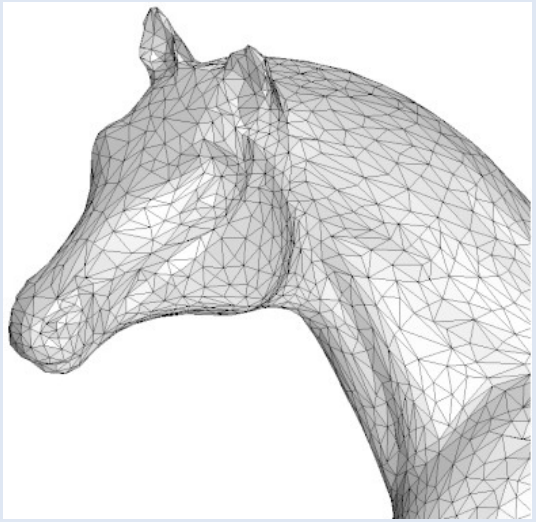
Mallas de polígonos y OpenGl

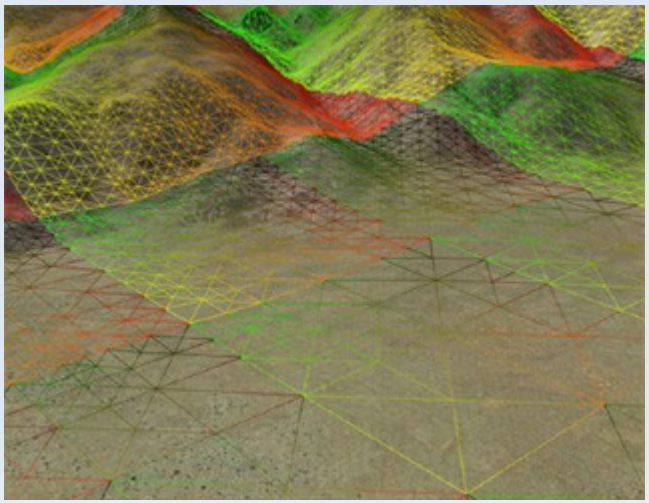
- Recordar que los pasos involucrados son:
  - Invocar a glEnableClientState(GL_VERTEX_ARRAY) para activar este modo
  - Usar glVertexPointer para especificar las coordenadas de los puntos
  - Dibujar la escena usando glDrawElements
  - Invocar a glEnableClientState(GL_VERTEX_ARRAY) para desactivar este modo
- glVertexPointer(size,type,stride,array):
  - size: numero de coordenadas usada para cada vertice
  - type: GL_INT, GL_FLOAT, etc
  - stride: desplazamiento en número de bytes comienzo siguiente coordenada
- glDrawElements(prim,num,type,array):
  - prim: GL_QUADS, GL_TRIANGLES, etc
  - num: número de elementos en array
  - Type: GL_UNSIGNED_BYTE (para índice pequeños), GL_INT, etc

Triangulaciones

- Cada polígono de la malla es un triángulo
- Aplicaciones: representar la superficie de objetos para:
  - Simulación numérica en aplicaciones científicas y de ingeniería
  - Generación de una malla de volumen (tetraedros, hexaedros o mixta)
  - Visualización en juegos y entretenimiento
  - Modelación de terrenos

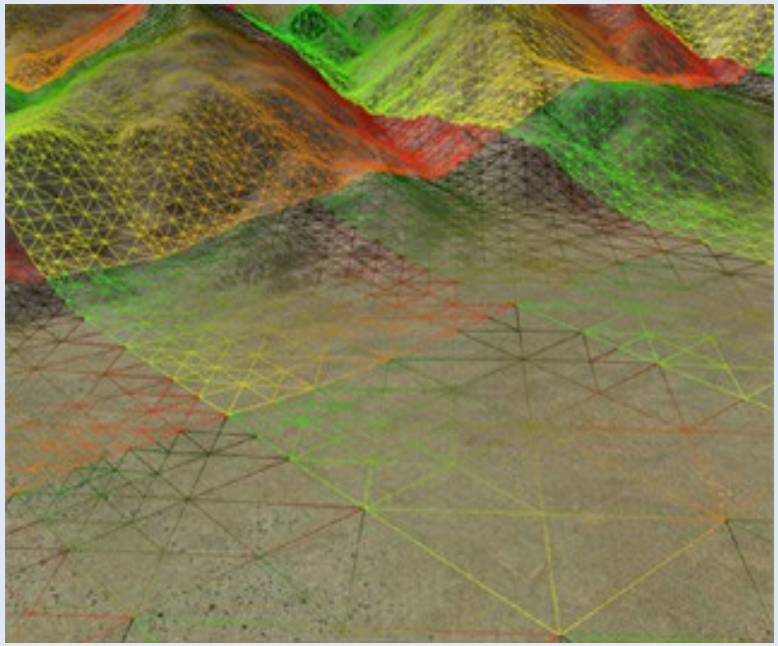






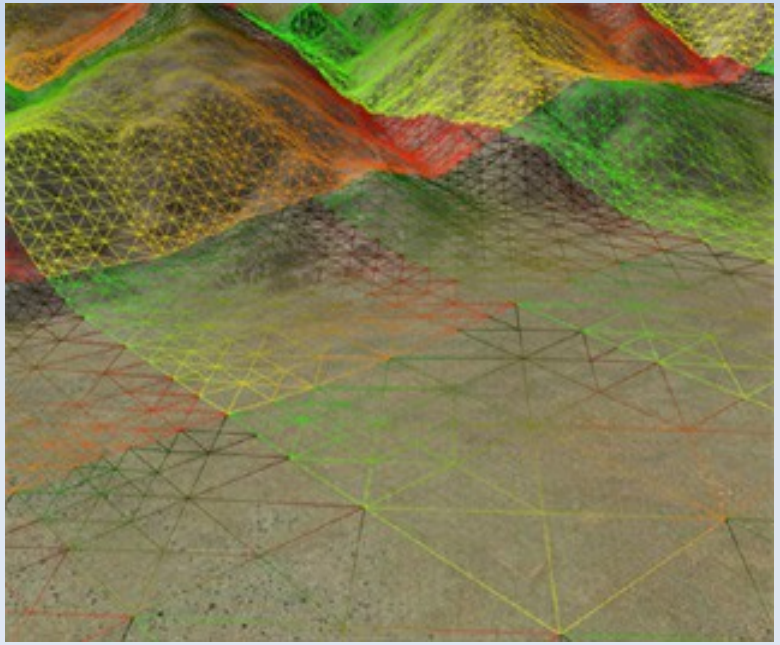
Triangulaciones para modelación de terrenos

- Mallas de triángulos en 3d (2 1/2 dimensiones)
- Datos de entrada:
  - Terreno función h(x,y) en el espacio 3D
  - Función h definida dentro de un cuadrado/rectángulo
- Salida:
  - Malla uniforme(triángulos iguales) o malla no uniforme(triángulos de distinto tamaño)
- Nota: un terreno tambien se puede modelar con mallas de cuadriláteros



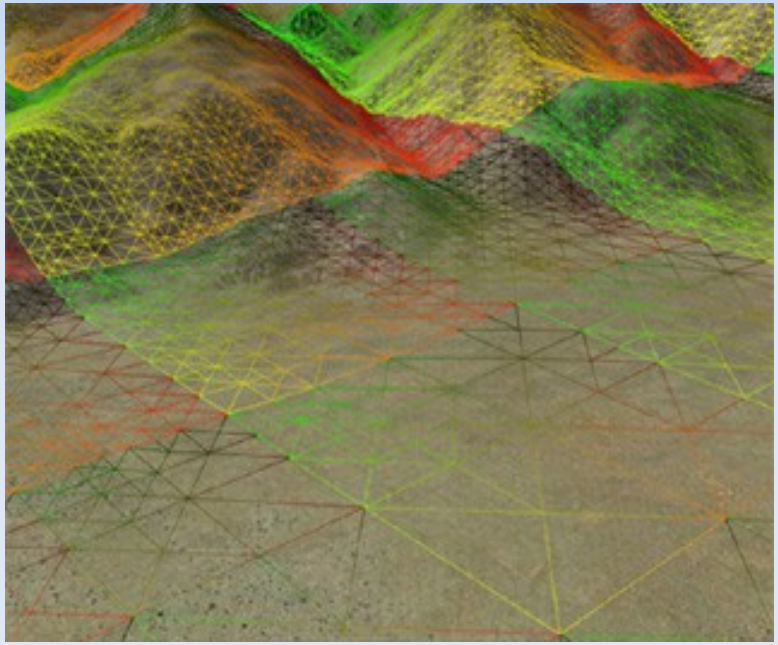
Triangulaciones para modelación de terrenos

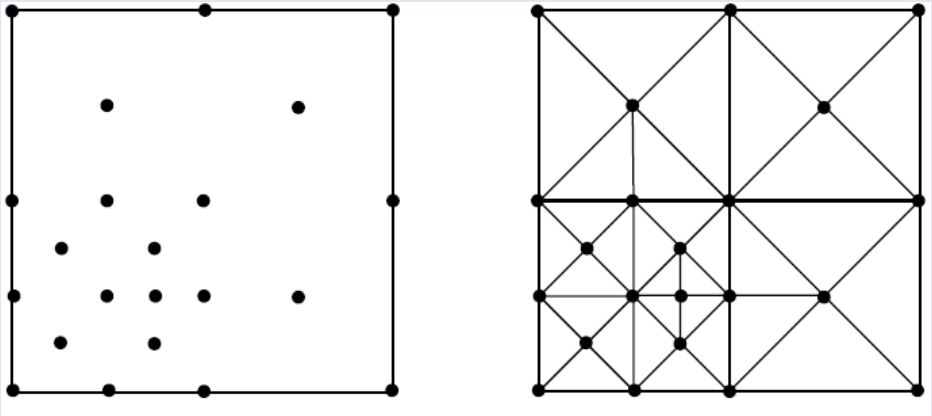
- Mallas de triángulos en 3d (2 1/2 dimensiones)
- Algoritmo 1
  - Entrada: datos equi-espaciados en x e y
  - Generar grilla rectangular
  - Dividir en triángulos
  - Altura de los puntos dada por h(x,y)
  - Eliminar triángulos en zonas de altura parecida



Triangulaciones para modelación de terrenos

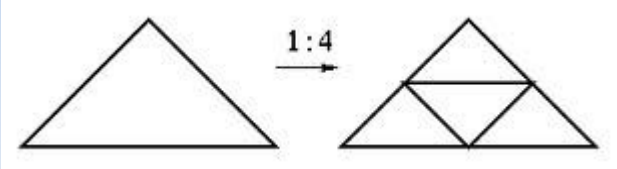
- Mallas de triángulos en 3d (2 1/2 dimensiones)
- Algoritmo 2
  - Entrada: datos equi-espaciados en x e y
  - Generar el rectángulo más grande
  - Dividir en dos triángulos
  - Dividir triángulos más pequeños si el terreno no está a la misma altura





Triangulaciones para modelación de terrenos

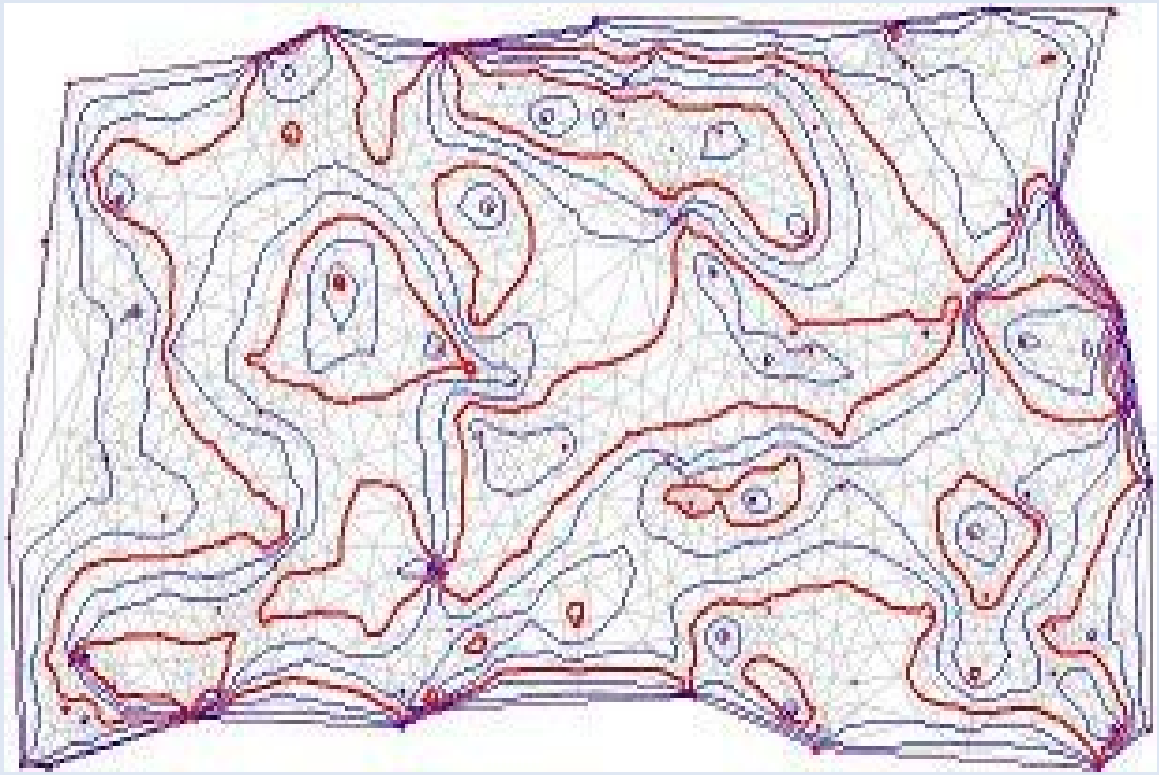
- Usando fractales: (wikipedia fractals animación y lectura)
  - Cada triángulo se divide en 4 triángulos iguales



  - Valores asociados a h(x,y) (elevación) a partir de puntos medios de los arcos y se suma (o resta) un valor aleatorio (no se usan datos reales)

Triangulaciones: curvas de nivel

- Qué datos podriamos querer visualizar de un terreno?
  - Zonas (líneas) a la misma altura
  - Usa, por ejemplo, curvas de nivel

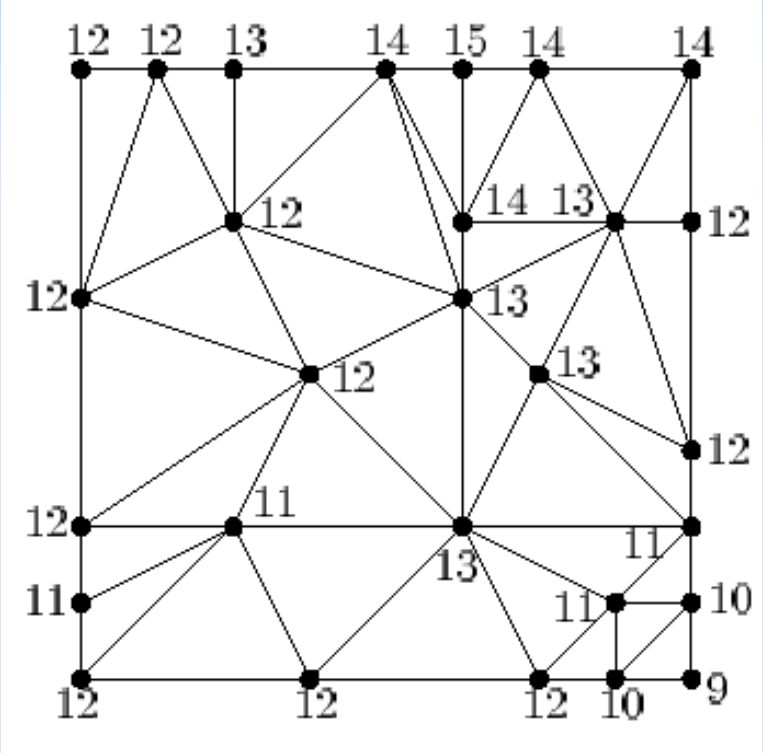


Triangulaciones: curvas de nivel

- Cómo calcular curvas de nivel?
  - Entrada: triangulación y valor (o valores de la curvas de nivel) de alturas que se quieren visualizar
  - Salida: malla, curvas de nivel aproximadas por segmentos de líneas
  - Algoritmo:
    - Encontrar un triángulo t que corte la “curva” definida a altura H: vértices del triángulo tiene h(x,y) mayor y menor que que el valor H
    - Unir los puntos a igual altura H con un segmento
    - while exista triángulo vecino cortado por la curva de altura H distinto de t
      - Encontrar arcos que que intersectan la curva
      - Unir puntos de intersección con un segmento

Triangulaciones: curvas de nivel

- Buscar curvas de nivel H = 12.5 y H = 11.5



¿Y si buscamos para H = 12?