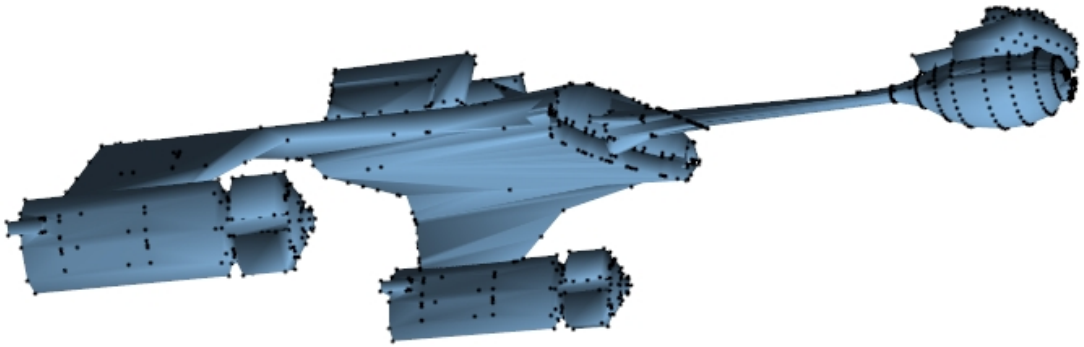


Tema 1: Modelado poligonal

José Ribelles

Departamento de Lenguajes y Sistemas Informáticos, Universitat Jaume I

SIU020 - Síntesis de Imagen y Animación

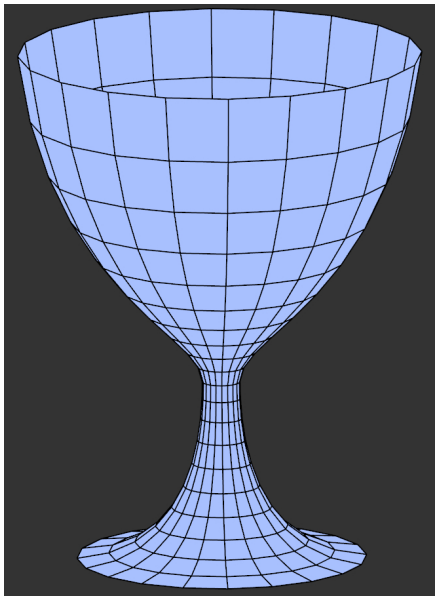
Contenido

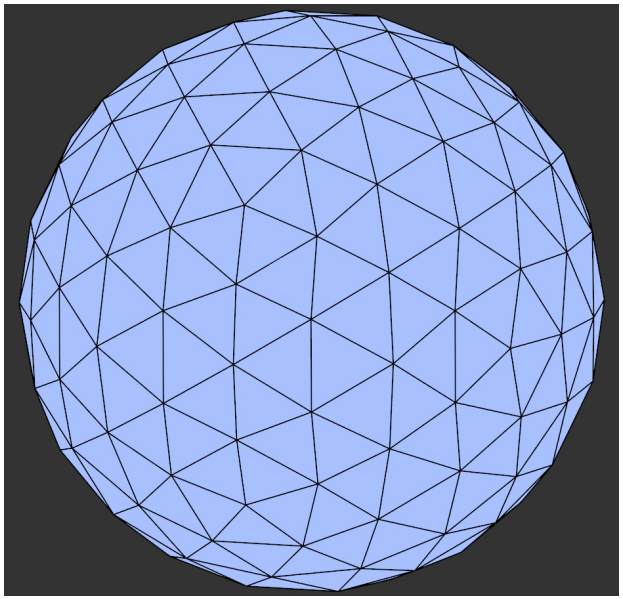
Hoy veremos...

Introducción

Modelado poligonal

- Se denomina modelo al conjunto de datos que describe a un objeto y que puede ser utilizado por un sistema gráfico para ser visualizado.
- Modelado poligonal es cuando se utilizan polígonos para describir la geometría.
- El triángulo es la primitiva más utilizada.

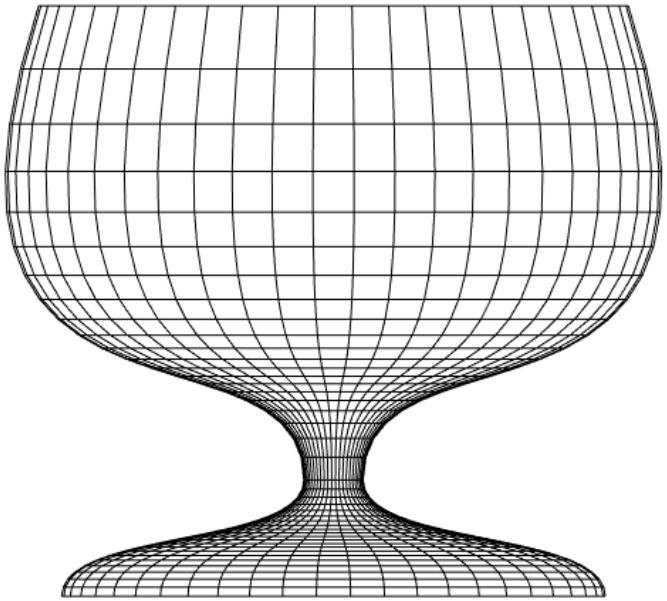


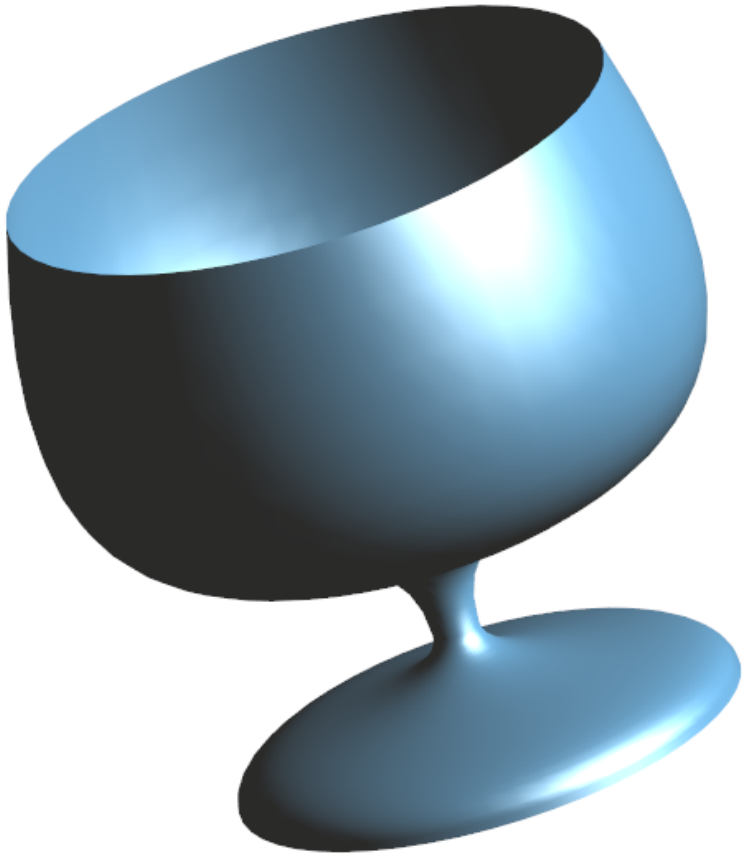


Introducción

Ventajas

- Exactitud **visual**.
- Velocidad.

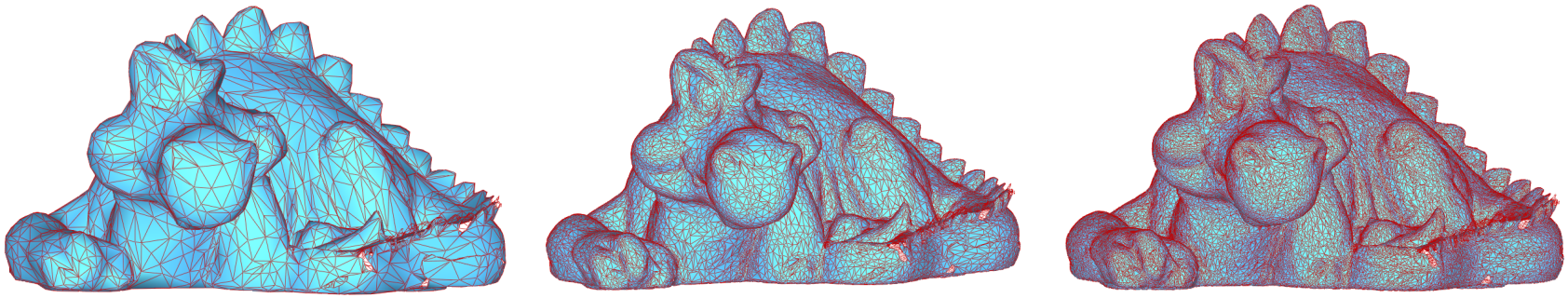




Ejercicio

Busca en la bibliografía información de los siguientes conceptos asociados al modelado poligonal:

- **Teselado**
- **Triangulación**
- **Optimización**
- **Simplificación**



Hoy veremos...

Orígenes de información poligonal

Posibles fuentes

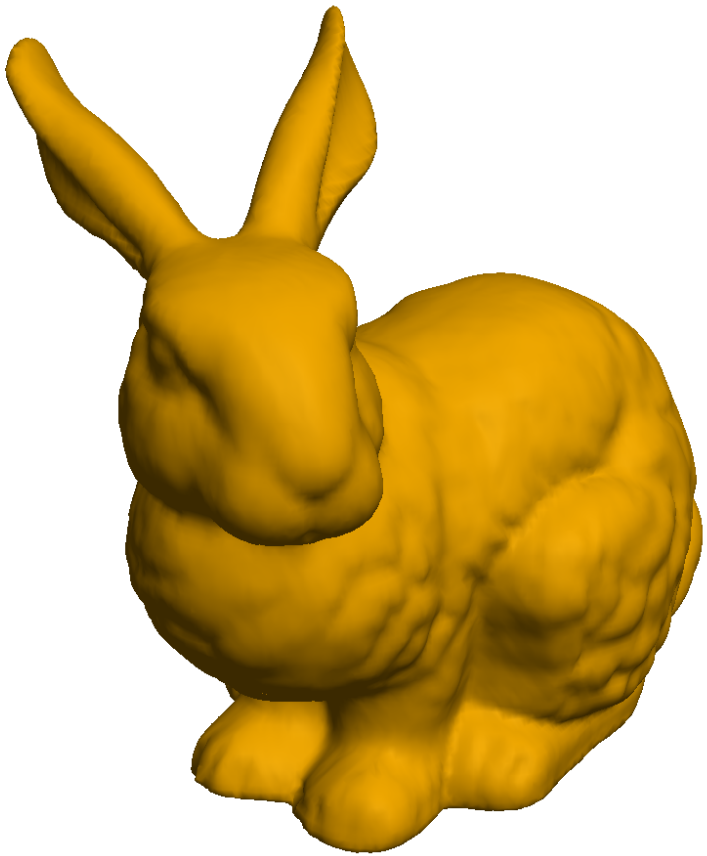
- Describir la geometría a mano.
- Modelado procedural: programas que generan la geometría.
- Transformar datos de modelos de superficies o volumétricos.
- Programas de modelado.
- Muestreo de un modelo real, digitalizador 3D.
- Fotogrametría: reconstrucción 3D a partir de imágenes.
- Escaner 3D.

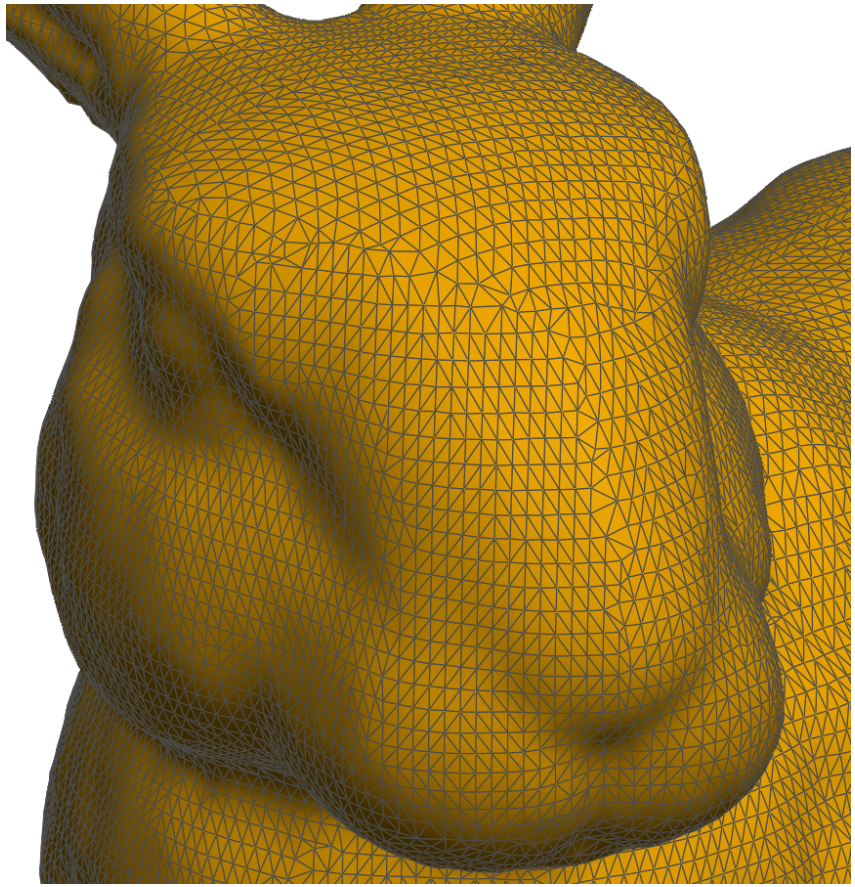
Hoy veremos...

Mallas de triángulos

- La mayor parte de los modelos poligonales lo forman mallas de triángulos.
- Una malla representa una superficie donde vértices y aristas se comparten.
- Es importante representarlos de manera eficiente para:
  - Reducir el espacio de almacenamiento.
  - Reducir el consumo de ancho de banda.
  - Reducir el tiempo de dibujado.

Mallas de triángulos

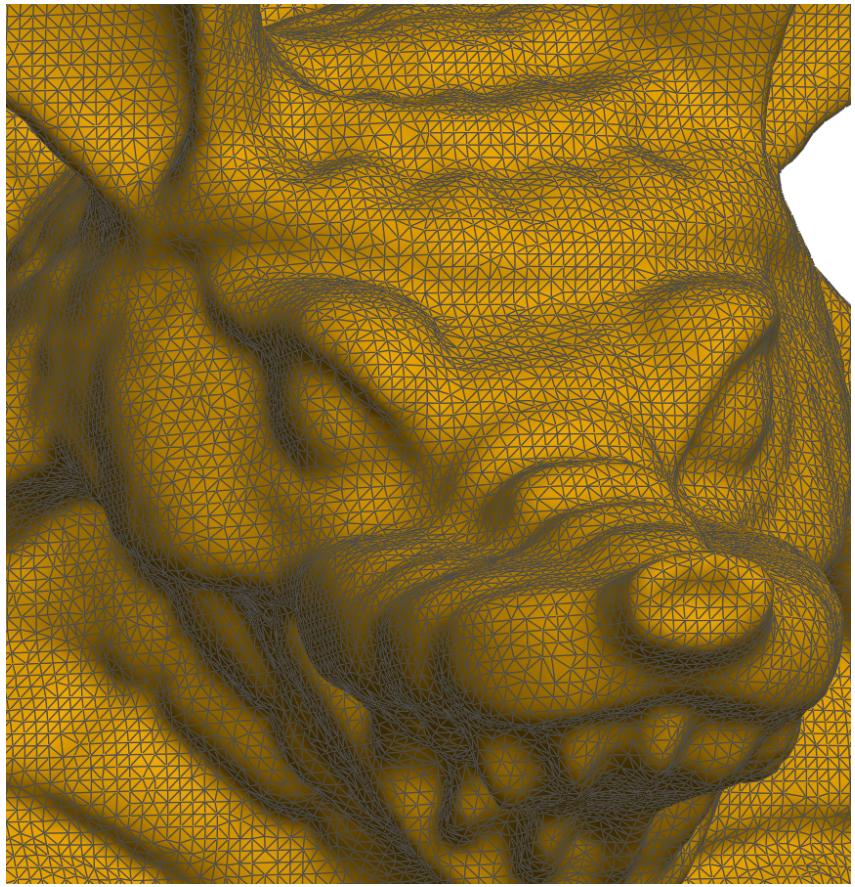




35.947 vértices y 69.451 triángulos

Modelo *Bunny* cortesía del *Stanford Computer Graphics Laboratory*.

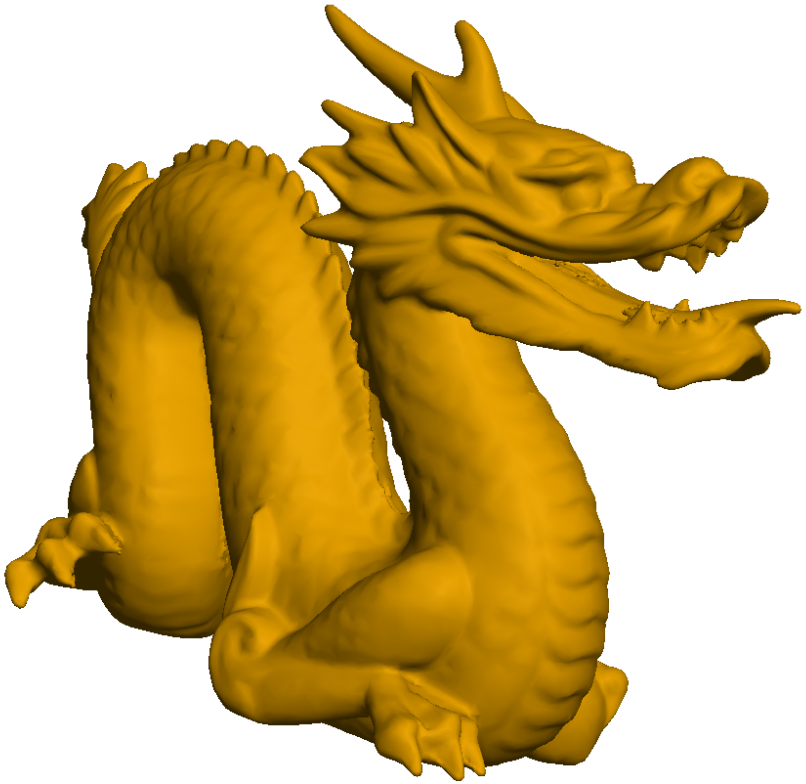
Mallas de triángulos

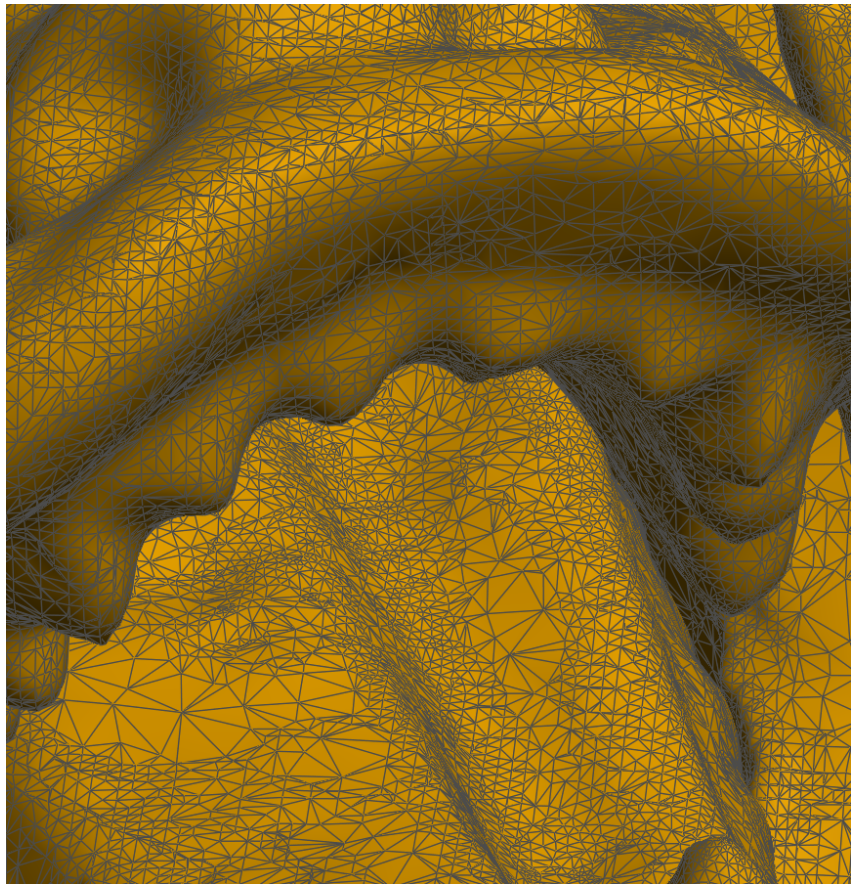


172.974 vértices y 345.944 triángulos

Modelo *Armadillo* cortesía del *Stanford Computer Graphics Laboratory*.

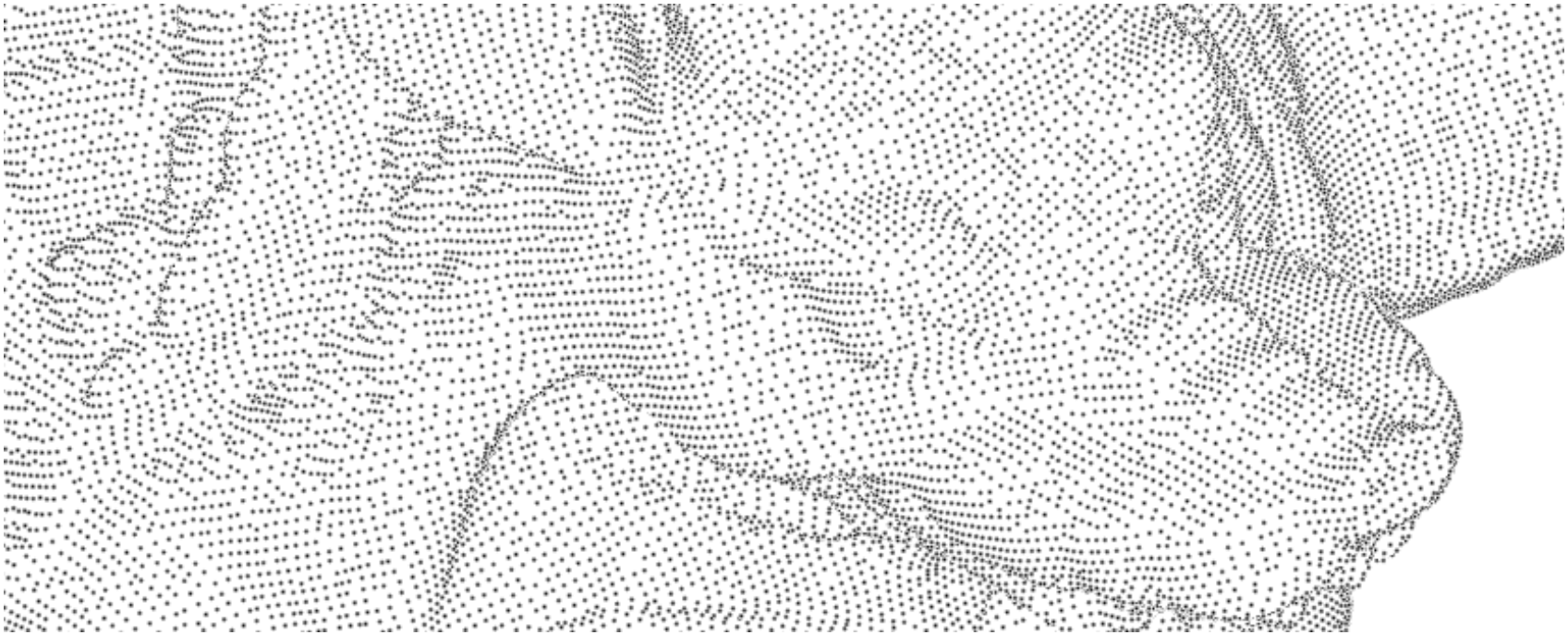
Mallas de triángulos





435.545 vértices y 871.306 triángulos
Modelo *Dragon* cortesía del *Stanford Computer Graphics Laboratory*.

Mallas de triángulos

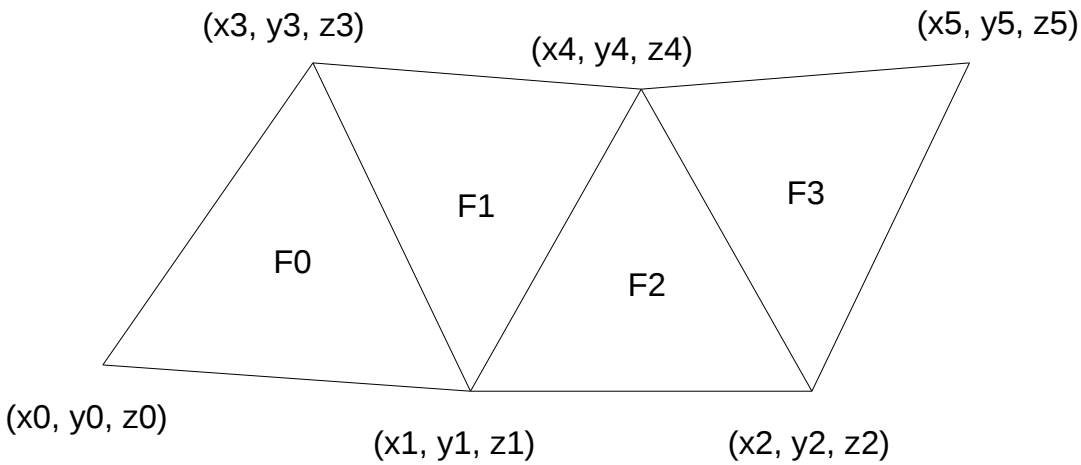


- Tipos de representación:
  - Caras independientes.
  - Vértices compartidos.
  - Tiras y abanicos de triángulos.

Caras independientes

Cada cara almacena sus vértices.

```
struct triangulo {
  vector3 coordenadas[3];
} Triangulos[nTriangulos];
```

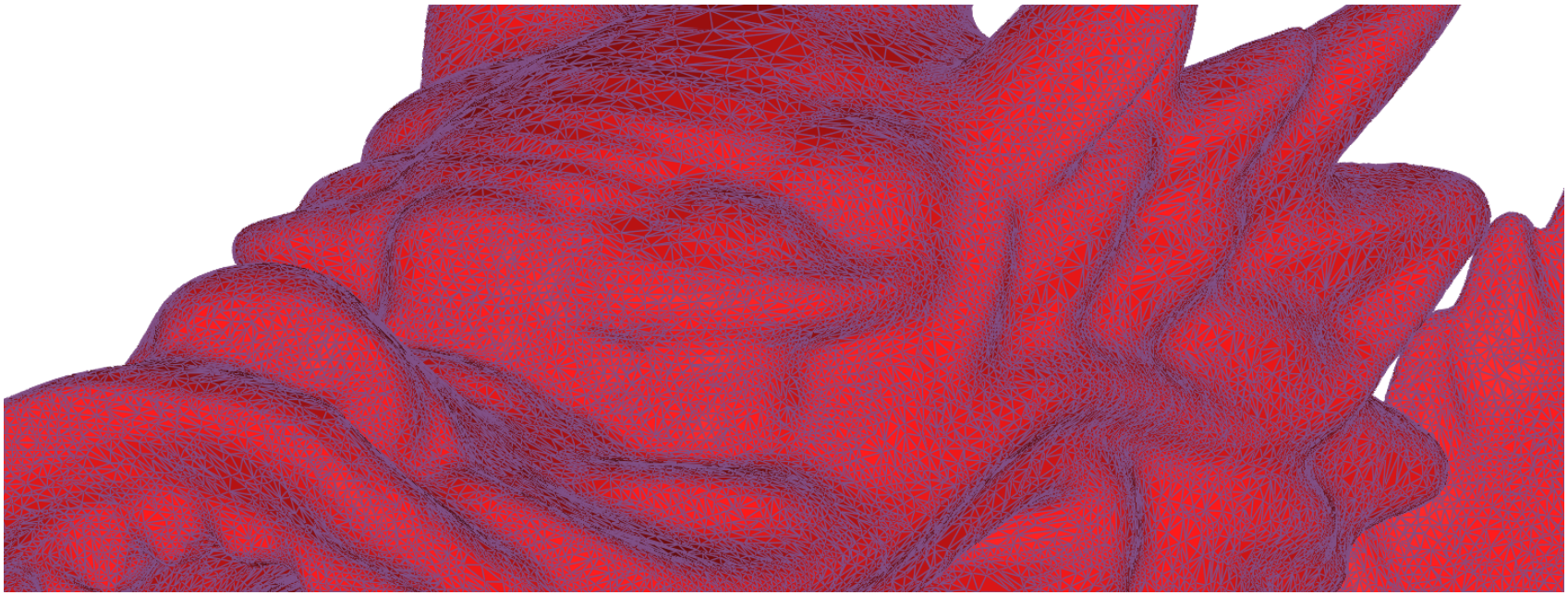


Triangulos[] = { { x0, y0, z0, x1, y1, z1, x3, y3, z3 },
{ x1, y1, z1, x4, y4, z4, x3, y3, z3 },
{ x1, y1, z1, x2, y2, z2, x4, y4, z4 },
{ x2, y2, z2, x5, y5, z5, x4, y4, z4 } }

Caras independientes

Ejercicio

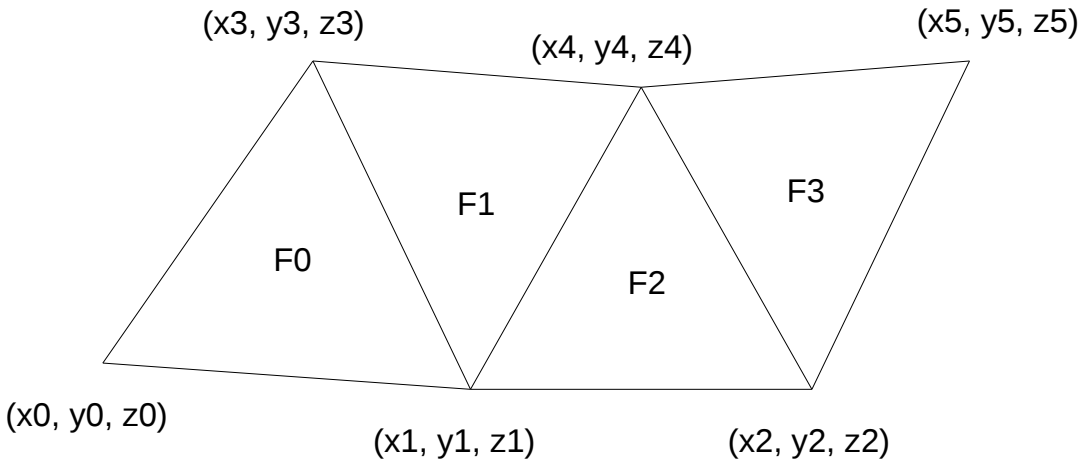
Calcula el coste de almacenamiento en *bytes* del modelo del *Dragon* utilizando la estructura de caras independientes si almacenar cada coordenada cuesta 4 bytes.



Vértices compartidos

Cada cara almacena índices a una lista de vértices.

```
struct vertice {
  float coordenadas[3];
} Vertices[nVertices];
struct triangulo {
  unsigned int indices[3];
} Triangulos[nTriangulos];
```



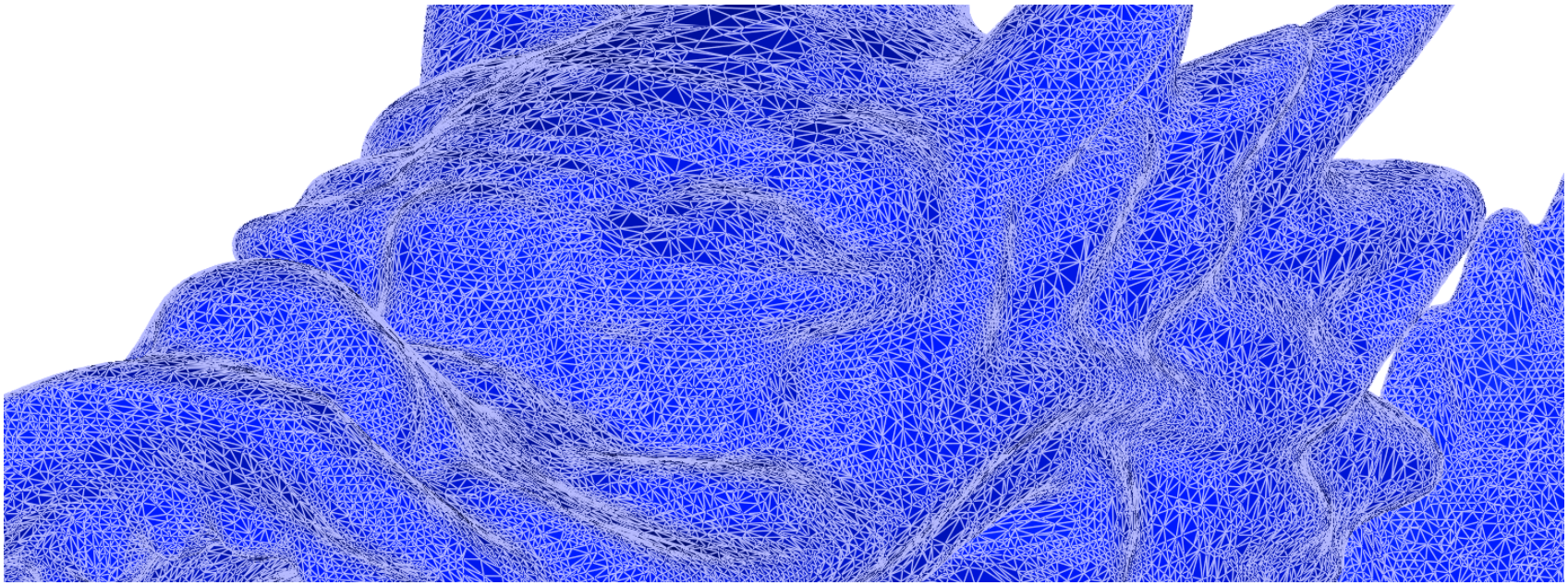
Vertices[] = { {x0, y0, z0},
{x1, y1, z1},
{x2, y2, z2},
{x3, y3, z3},
{x4, y4, z4},
{x5, y5, z5} }

Triangulos[] = { { 0, 1, 3 },
{ 1, 4, 3 },
{ 1, 2, 4 },
{ 2, 5, 4 } }

Caras independientes

Ejercicio

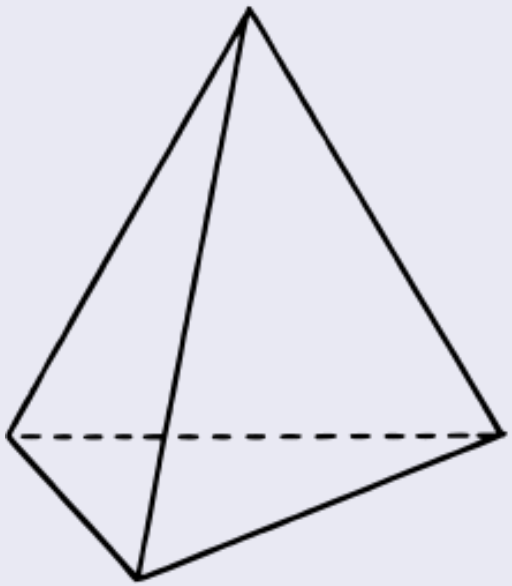
Calcula el coste de almacenamiento en *bytes* del modelo del *Dragon* utilizando la estructura de vértices compartidos si almacenar una coordenada o un índice cuesta 4 bytes cada uno.



Ejercicio

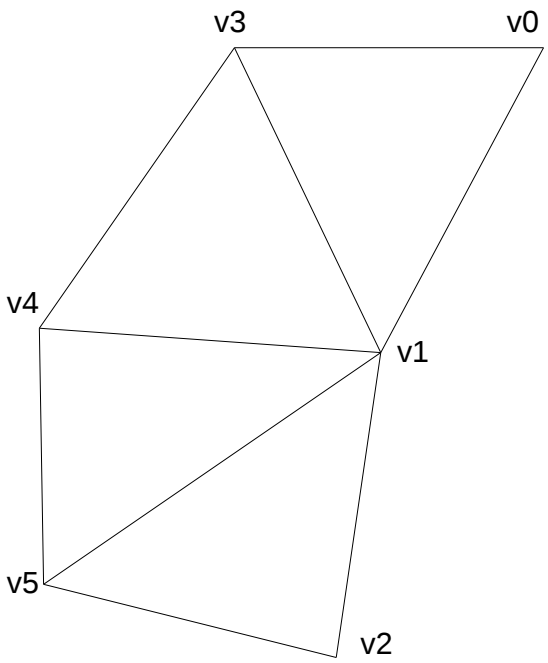
Dado un tetraedro, calcula la diferencia en bytes que existe entre almacenarlo utilizando el método de caras independientes y el de vértices compartidos.
Asume que:

- cada vértice almacena tres coordenadas, y que
- almacenar una coordenada o un índice cuesta los mismo, es decir, 4 bytes.

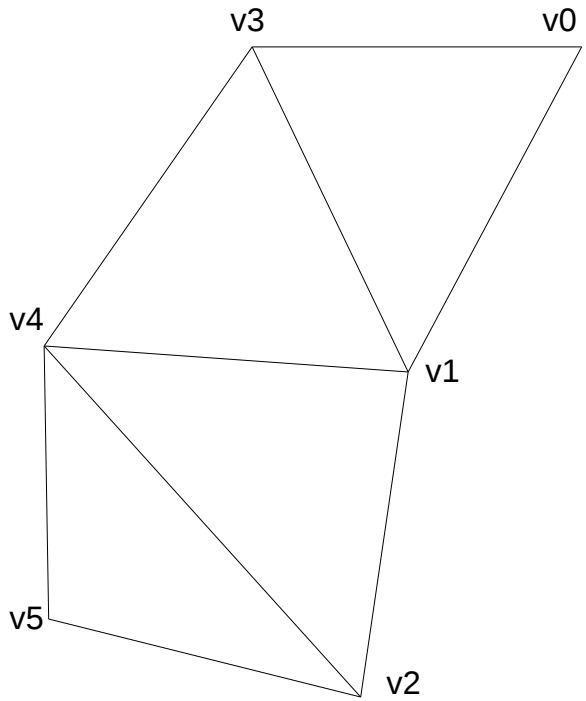


Tiras y abanicos de triángulos

Incremento de prestaciones al enviar grupos de triángulos que comparten vértices, ¿por qué se produce?, ¿se produce siempre?, ¿qué podemos concluir?



AbanicoTriangulos[] = { 1, 0, 3, 4, 5, 2 }

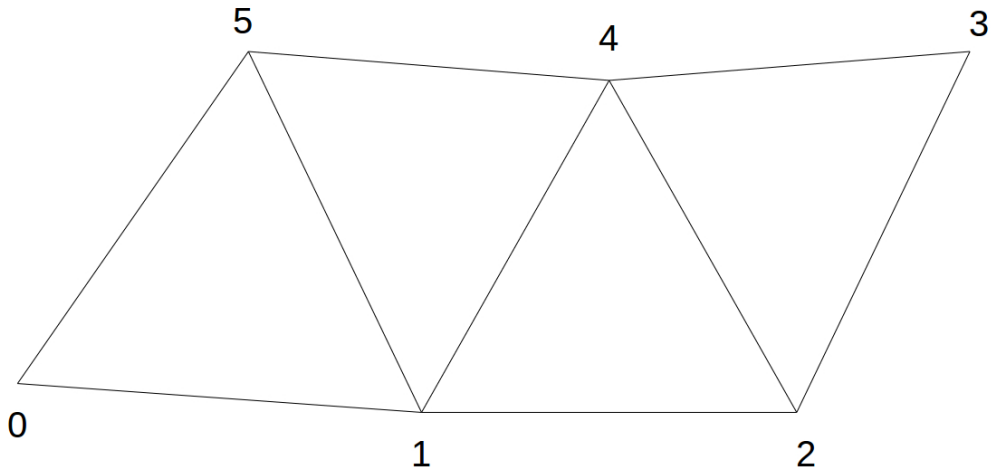


TiraTriangulos[] = { 0, 3, 1, 4, 2, 5 }

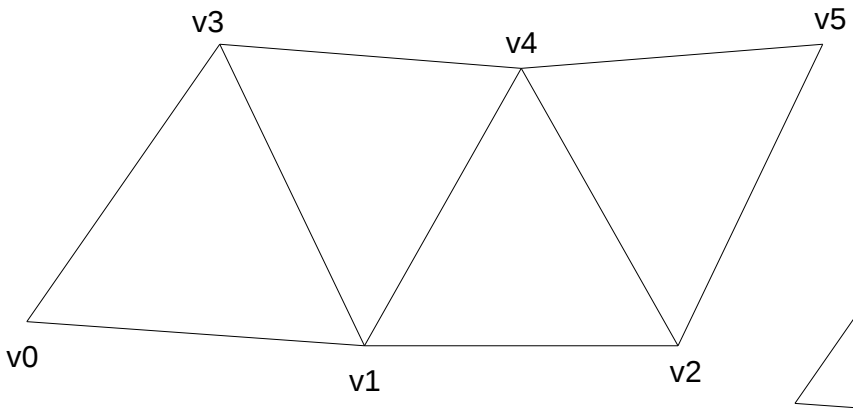
Ejercicio

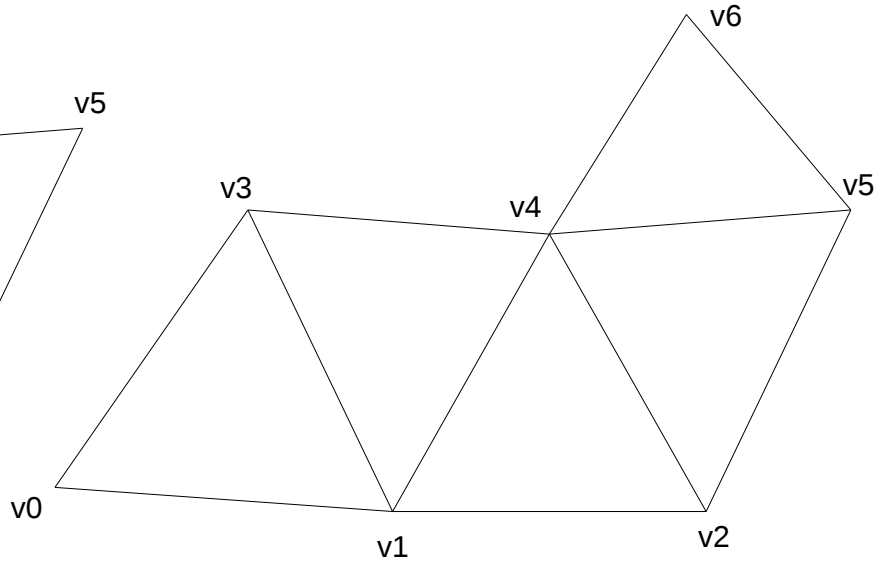
Señala las secuencias de vértices correctas que representan la siguiente malla poligonal como una tira de triángulos (pueden haber varias respuestas correctas):

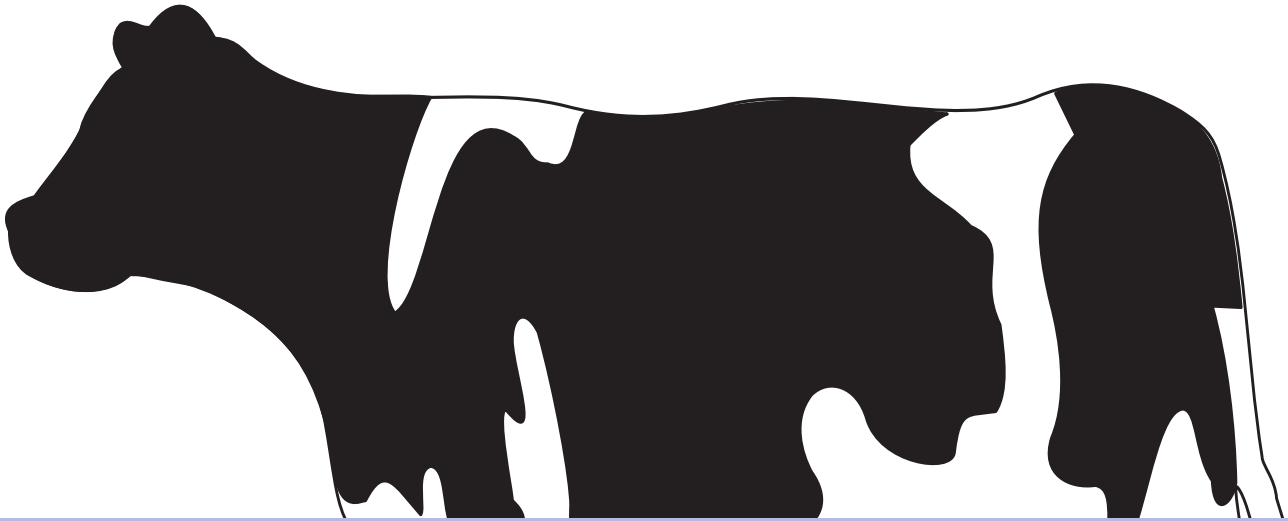
1. 3 2 4 1 0 5
2. 0 1 5 4 2 3
3. 0 5 1 4 2 3
4. 3 2 4 1 5 0



Tiras de triángulos generalizadas

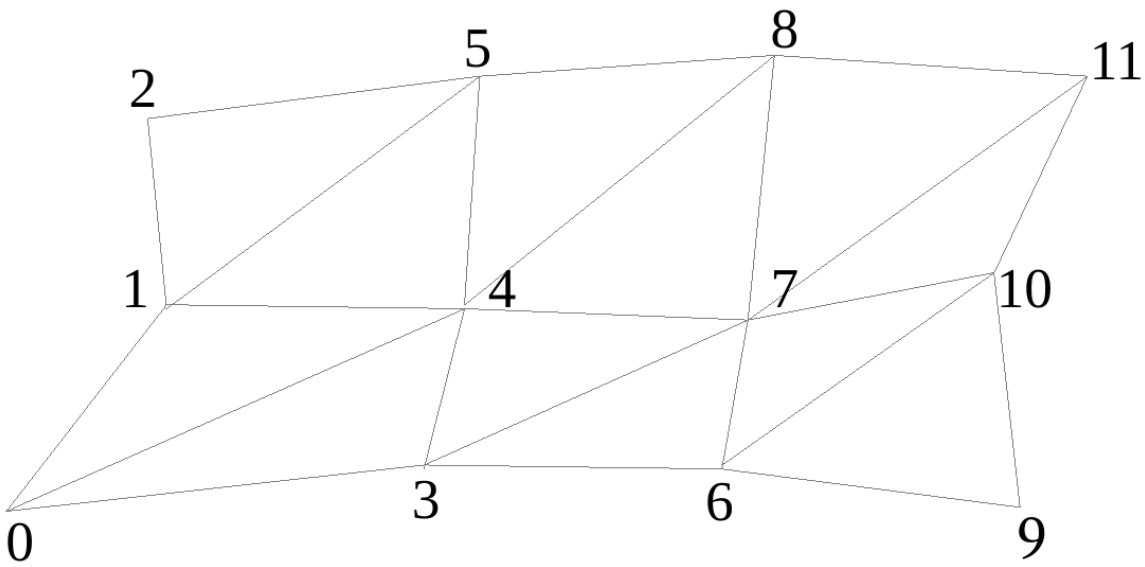






Ejercicio

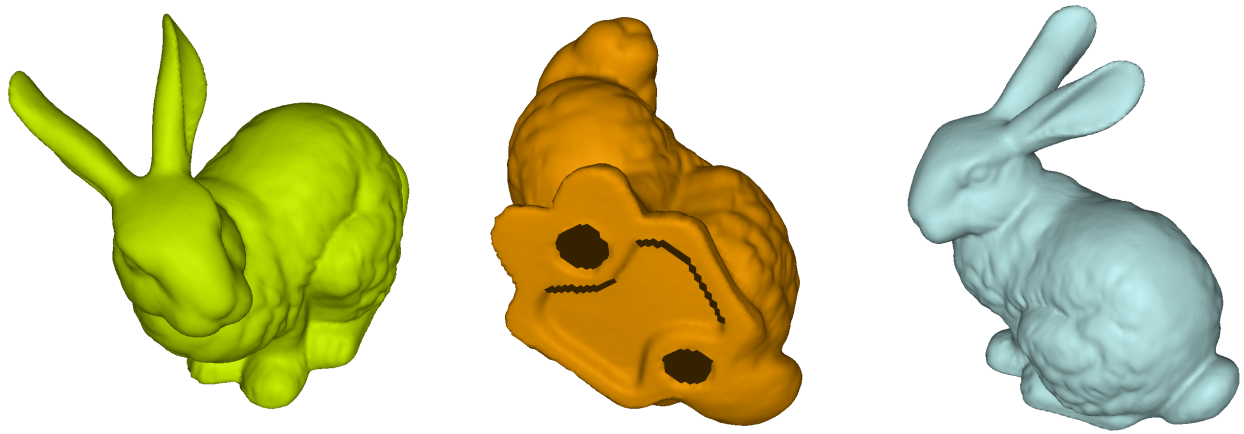
Obtén una secuencia de vértices que represente la siguiente malla poligonal como una única tira de triángulos.



Ejercicio

Completa la tabla y calcula para el modelo *Bunny* los costes de almacenamiento en *bytes* si lo representas mediante una única tira de triángulos de 102.997 índices. Asume un coste de 4 bytes por coordenada o índice.

Primitiva	**Estructura**	**Bytes**	**Fps**
Triángulo	Caras independientes	2.500.236	760
	Vértices compartidos	1.264.776	760
Tira de triángulos	Caras independientes	?	1100
	Vértices compartidos	?	1100



Ejercicio

Observa la siguiente descripción poligonal de un objeto. Las líneas que comienzan por *v* se corresponden con los vértices e indican sus coordenadas. El primero es el número 1 y los demás se enumeran de forma consecutiva. Las líneas que comienzan por *f* se corresponden con las caras e indican qué vertices lo forman.

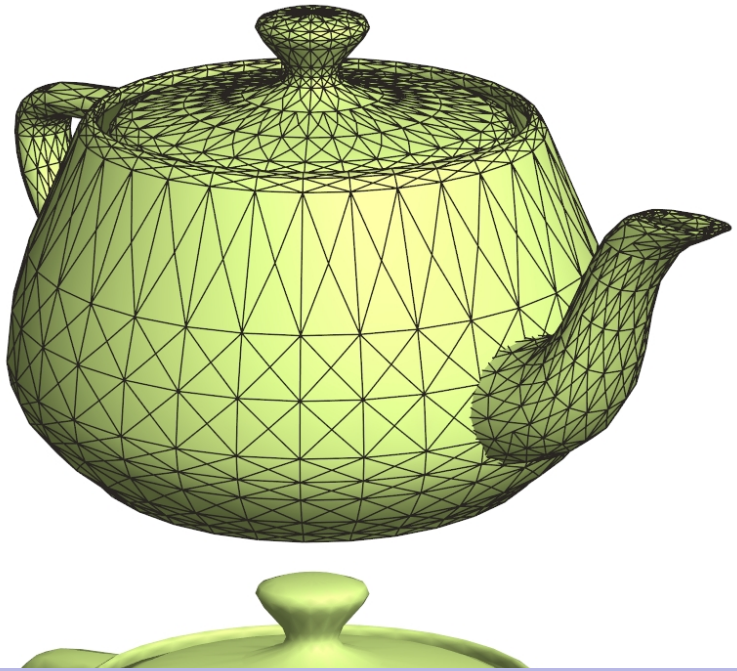
```
v 0 0 0        f 1 3 2
v 0 0 1        f 1 4 3
v 1 0 1        f 1 2 5
v 1 0 0        f 2 6 5
v 0 1 0        f 3 2 6
v 0 1 1        f 3 6 7
v 1 1 1        f 3 4 7
v 1 1 0        f 4 8 7
               f 4 1 8
               f 1 5 8
```

- Dibújalo en papel, ¿qué objeto representa?
- ¿Están todas sus caras definidas en el mismo orden?
- ¿En qué sentido están definidas, horario o antihorario?

Hoy veremos...

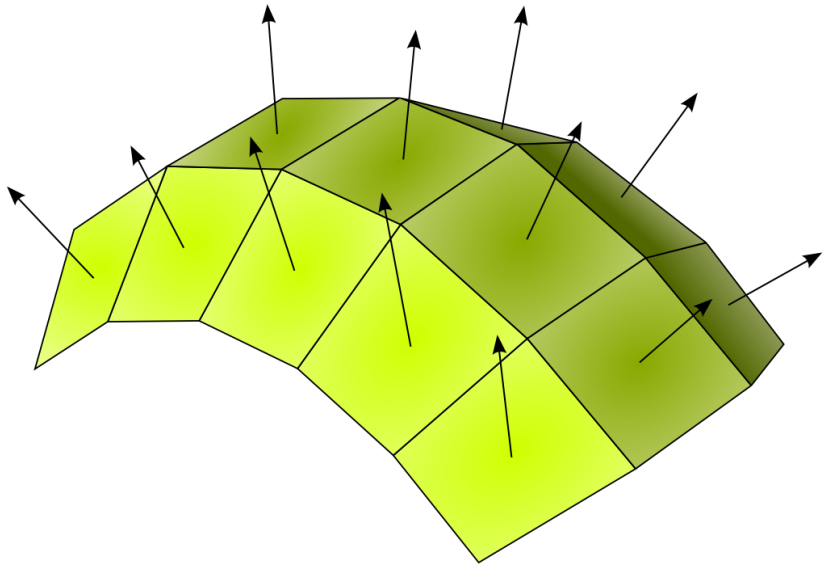
La Normal

- Es habitual que para cada vértice se almacenen atributos como el color, la normal o las coordenadas de textura.
- Estos atributos se interpolan a través de cada triángulo para definir una función continua sobre el total de la superficie de la malla.
- Y son necesarios para mejorar significativamente la calidad visual.



La Normal

- Uno de los atributos más importantes es la Normal.
- La normal es un vector perpendicular a la superficie en un punto, ¿cómo obtenerla?
- Ya que el producto vectorial no es commutativo, hay que establecer un orden y obtener todas las normales de manera consistente.



Ejercicios

- ¿Cómo puedes saber si todas las caras de una malla están definidas en el mismo orden (horario o antihorario) y cómo lo arreglas en el caso de encontrar caras definidas en sentidos distintos?
- ¿Cómo puedes saber si una malla es cerrada o si por contra presenta algún agujero?
- ¿Cómo calcularías la normal en un vértice de una malla? ¿Y si el vértice pertenece a una esquina o a un canto del modelo?
- Describe un modelo poligonal de un Cubo de lado 1 centrado en el origen de coordenadas:
  1. Dibújalo en papel.
  2. Obten la lista de vértices y la de triángulos.
  3. Ten cuidado y define todas sus caras en el mismo orden.
  4. ¿Cuántos vértices te salen?
  5. Añade ahora las normales para cada vértice.
  6. ¿Tienes el mismo número de vértices?

Hoy veremos...

Primitivas geométricas

- OpenGL no proporciona mecanismos para modelar objetos sino para especificar cómo han de ser dibujados.
- El punto, el segmento de línea y el triángulo.
- Cada primitiva se define especificando sus respectivos vértices.
  - Dibujo de puntos:
    - gl.POINTS
  - Dibujo de líneas:
    - Segmentos sueltos: gl.LINES
    - Secuencia o tira de segmentos: gl.LINE_STRIP
    - Secuencia cerrada de segmentos: gl.LINE_LOOP
  - Triángulos
    - Triángulos sueltos: gl.TRIANGLES
    - Tira de triángulos: gl.TRIANGLE_STRIP
    - Abanico de triángulos: gl.TRIANGLE_FAN
- Las primitivas geométricas se forman agrupando vértices, y estas se agrupan a su vez para definir objetos de mayor complejidad.

Modelado

- Solemos asociar el concepto de vértice con las coordenadas.
- El concepto de vértice es más general entendiéndose como una agrupación de datos llamados atributos.
- Los atributos más utilizados son la posición, la normal y el color.
- El programador puede incluir como atributo cualquier información.
- OpenGL requiere que los datos a dibujar se dispongan en vectores.

```
struct vertice
{
  GLfloat coordenadas[3];
  GLfloat color[3];
}
Vertices[nVertices];

struct triangulo
{
  GLuint indices[3];
}
Triangulos[nTriangulos];
```

Visualización

- Tres pasos previos:
  1. Almacenar el modelo poligonal en *buffer objects*.
  2. Obtener los índices de las variables del *Shader* que representan los atributos de los vértices.
  3. Especificar para cada atributo dónde y cómo se encuentran almacenados así como habilitar los vectores correspondientes.
- Dibujar indicando tipo de primitiva y número de elementos.

Listado 1: Paso 1

```
idBufferVertices = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, idBufferVertices);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(Vertices), gl.STATIC_DRAW);

idBufferIndices = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, idBufferIndices);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(Triangulos), gl.STATIC_DRAW);
```

Visualización

Listado 2: Paso 2

```
<script id="vertexShaderSource" type="x-shader/x-vertex">#version 300 es

  in  vec3 VertexPosition;
  in  vec3 VertexColor;

  out vec4 colorOut;

  void main()
  {
     colorOut    = vec4 (VertexColor, 1.0);
     gl_Position = vec4 (VertexPosition, 1.0);
  }

</script>

<script id="fragmentShaderSource" type="x-shader/x-fragment">#version 300 es

  precision mediump float;
  in  vec4 colorOut;
  out vec4 fragmentColor;

  void main(void)
  {
     fragmentColor = colorOut;
  }

</script>

... // compila y enlaza el Shader

vertexPositionAttribute = gl.getAttribLocation(program, "VertexPosition");
vertexColorAttribute    = gl.getAttribLocation(program, "VertexColor");
```

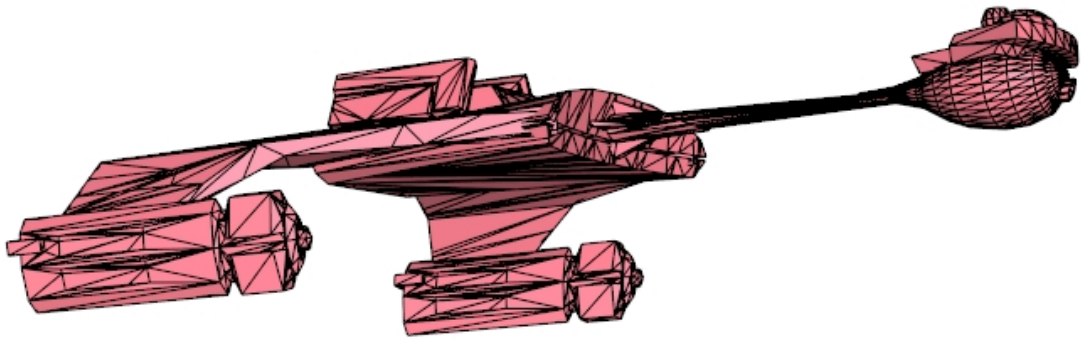
Visualización

Listado 3: Paso 3 y dibujado

```
gl.bindBuffer(gl.ARRAY_BUFFER, idBufferVertices);
gl.vertexAttribPointer (vertexPositionAttribute, 3, gl.FLOAT, false, 2*3*4, 0);
gl.vertexAttribPointer (vertexColorAttribute,    3, gl.FLOAT, false, 2*3*4, 3*4);

gl.enableVertexAttribArray(vertexPositionAttribute);
gl.enableVertexAttribArray(vertexColorAttribute);

gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, idBufferIndices);
gl.drawElements(gl.TRIANGLES, nTriangulos*3, gl.UNSIGNED_SHORT, 0);
```



Ejercicio

Contesta de manera razonada a cada una de las siguientes preguntas:

- ¿Cuál es el número mínimo de buffers que necesitas en WebGL para almacenar un modelo poligonal que utiliza la representación de vértices compartidos si para cada vértice almacenas coordenadas y normal?
- Cuántos triángulos crees que se pintarían con la siguiente orden (ten en cuenta que al tratarse de una tira pueden haber triángulos degenerados):
  - gl.drawElements(gl.TRIANGLE_STRIP, 30, gl.UNSIGNED_SHORT, 12)
- Discute la veracidad de la siguiente afirmación:
  - es interesante hacer que todas las caras de una malla poligonal estén orientadas de la misma forma ya que así una malla poligonal representada por vértices compartidos ocuparía menos espacio.