Mallas indexadas y vectores normales

Informática Gráfica I

Material de: **Antonio Gavilanes**
Adaptado por: **Elena Gómez** y **Rubén Rubio**
{mariaelena.gomez,rubenrub}@ucm.es

UNIVERSIDAD COMPLUTENSE MADRID

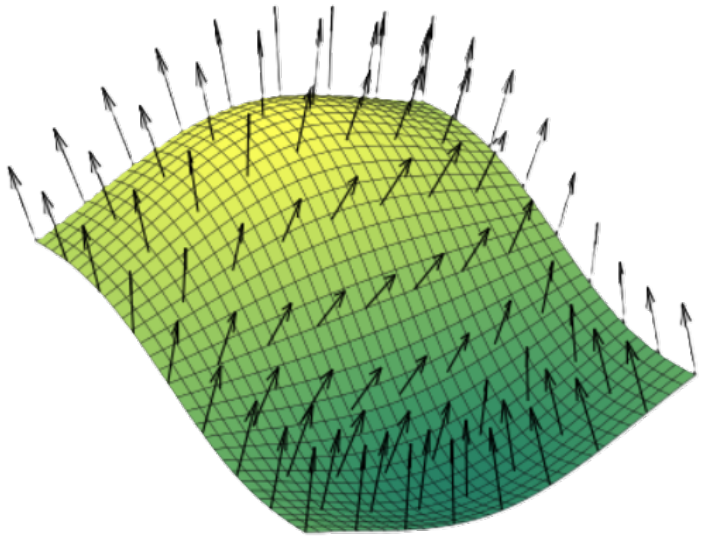
Contenido

Mallas

- Una **malla** (en inglés, *mesh*) es una colección de polígonos que se usa para representar la superficie de un objeto tridimensional.
- Estándar de representación de objetos gráficos:
  - Facilidad de implementación y transformación.
  - Propiedades sencillas.
- Representación exacta o aproximada de un objeto.
- Un elemento más en la representación de un objeto (color, material, textura).

Mallas y vectores normales

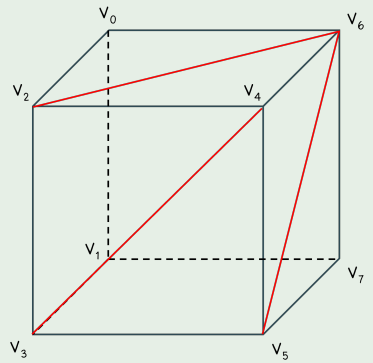
- Muchas caras de las mallas comparten vértices.
- Cada vértice y cada cara tiene un vector **normal**.
- La normal es importante porque:
  - descarta el renderizado de las caras posteriores
  - añade efectos de luz y sombra en función del ángulo por la normal y la dirección del foco



Repetición de información

Cubo de lado 2 y centrado en el origen

- El cubo tiene 8 vértices (numerados del 0 al 7).
- La primitiva que se usa es `GL_TRIANGLES` y, se necesitan 36 vértices (12 triángulos, 2 por cara).
- En el array de vértices, cada uno se repetiría 4 o 5 veces (pues 4 vértices forman parte de 4 triángulos y los otros 4, de 5 triángulos).



Repetición de información

Cubo de lado 2 y centrado en el origen

En lugar de repetir los vértices para formar los triángulos se añade a una tabla de índices.

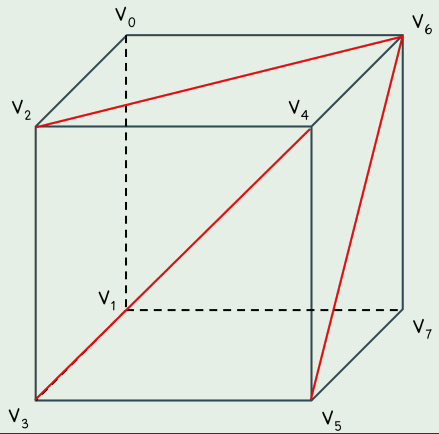
Vértices (8)

0	$(1, 1, -1)$
1	$(1, -1, -1)$
2	$(1, 1, 1)$
3	$(1, -1, 1)$
4	$(-1, 1, 1)$
5	$(-1, -1, 1)$
6	$(-1, 1, -1)$
7	$(-1, -1, -1)$

Vértices (36) = (3 vértices × 12 triángulos)

$0, 1, 2, 2, 1, 3,$
$2, 3, 4, 4, 3, 5,$
$4, 5, 6, 6, 5, 7,$
$6, 7, 0, 0, 7, 1,$
$4, 6, 2, 2, 6, 0,$
$1, 7, 3, 3, 7, 5$

Vértice de la **posición 5** (¡el que no se ve en la figura!) de la tabla de vértices (**se repite 4 veces**)



Repetición de la información

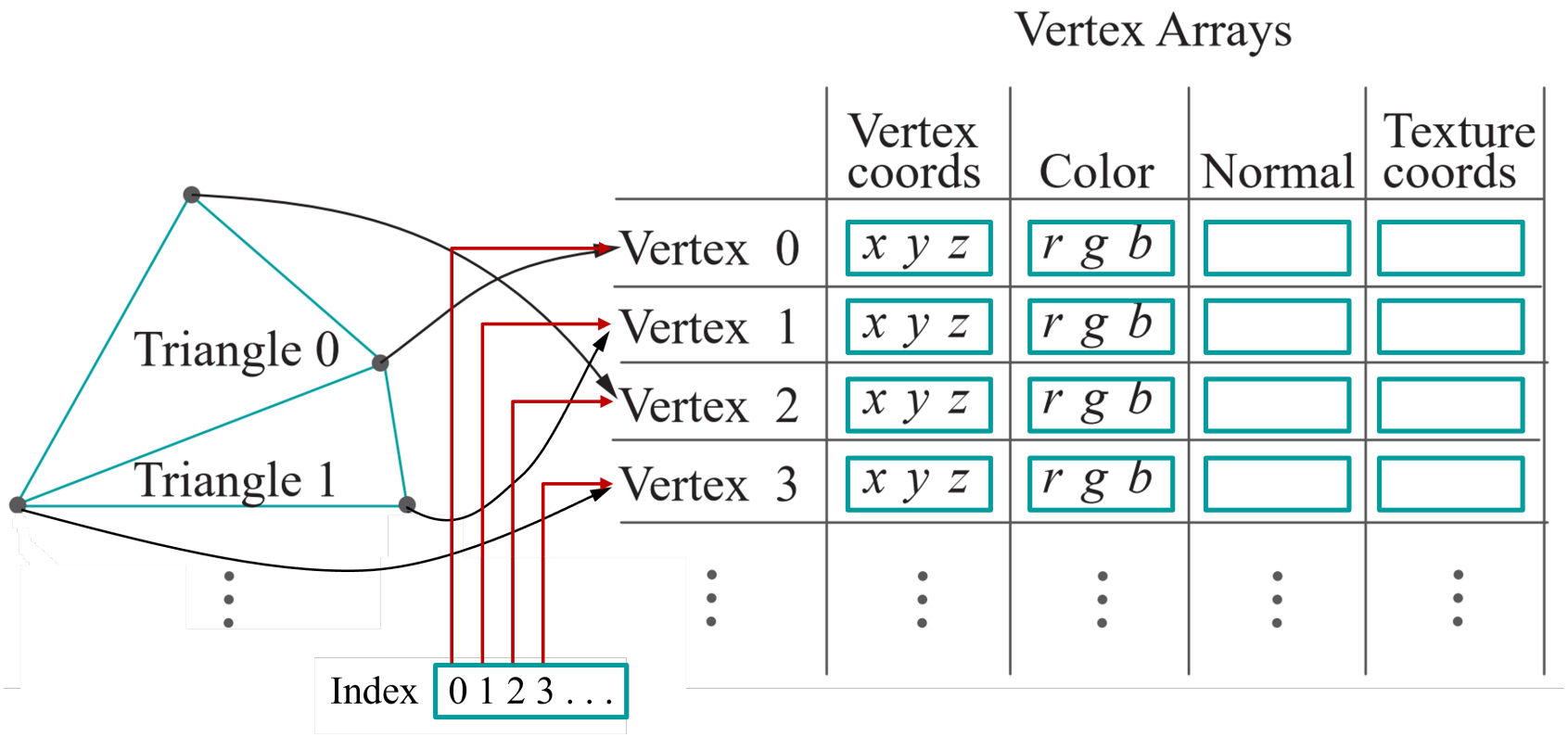
Evitan la repetición de la información mediante el uso de índices. Están formadas por columnas de:

- **Vértices**: contiene las coordenadas de los vértices de la malla (información de posición).
- **Normales**: contiene las coordenadas de los vectores normales a los vértices (información de orientación).
- **Colores**: contiene información de los colores de los vértices
- **Coordenadas de textura**: contiene las coordenadas de textura de cada vértice.

Estas últimas tablas tienen que ser del mismo tamaño que la tabla de vértices (`numVertices`).

Mallas indexadas

Añadimos índices (posiciones en la tabla de vértices) a las mallas para poder repetir vértices sin tener que copiar sus datos.



Implementación

Supongamos una clase `IndexMesh` que extiende `Mesh` con

- un atributo `vector<GLuint> vertices` con los índices.
- un atributo `GLuint mIBO` con la referencia al VBO de los índices.
- sobrescrituras de los métodos `load`, `unload` y `draw`.

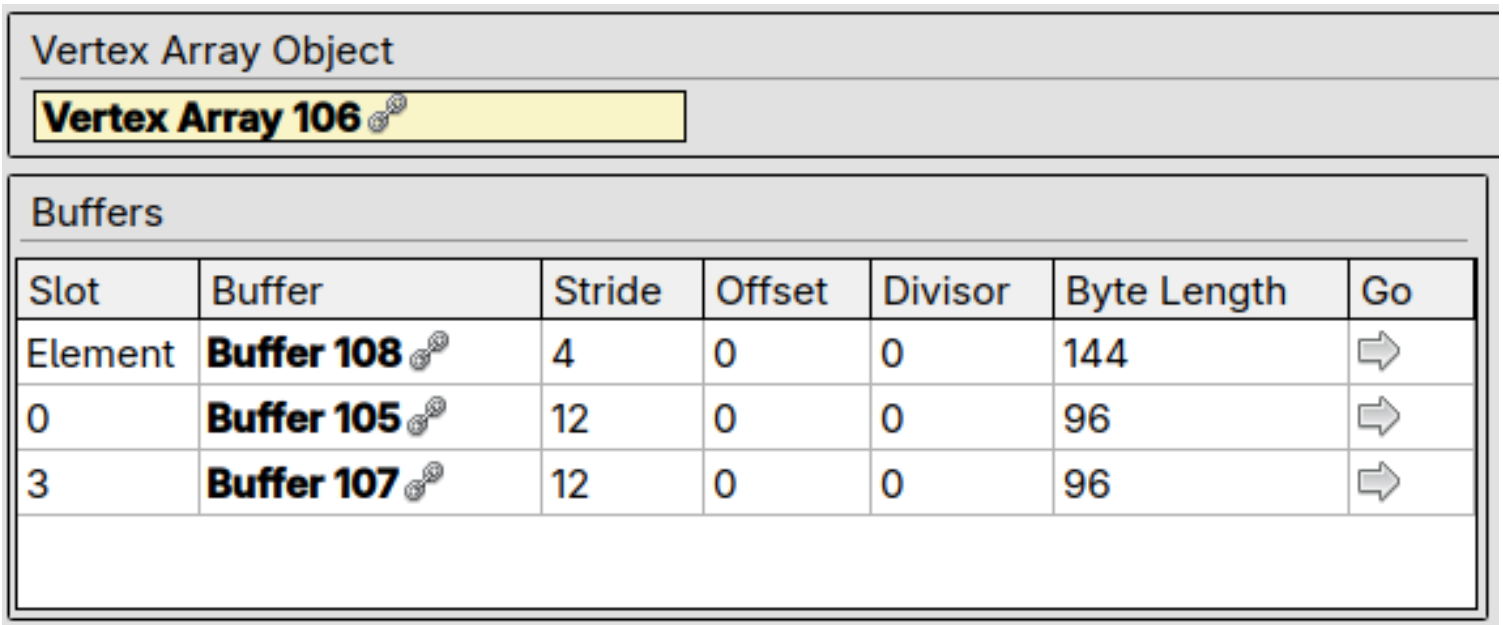
La carga de los índices se hace como se vio el tema anterior:

```
void IndexMesh::load() {
   Mesh::load(); glBindVertexArray(mVAO);
   glGenBuffers(1, &mIBO);
   glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, mIBO);
   glBufferData(GL_ELEMENT_ARRAY_BUFFER,
                vIndices.size() * sizeof(GLuint),
                vIndices.data(), GL_STATIC_DRAW);
   glBindVertexArray(0);
}
```

Implementación

El VBO de índices queda asociado al VAO activo en el momento de crearlo (como en la diapositiva anterior), pero la vinculación se puede hacer alternativamente con:

```
glVertexArrayElementBuffer (mVAO, mIBO);
```



Implementación

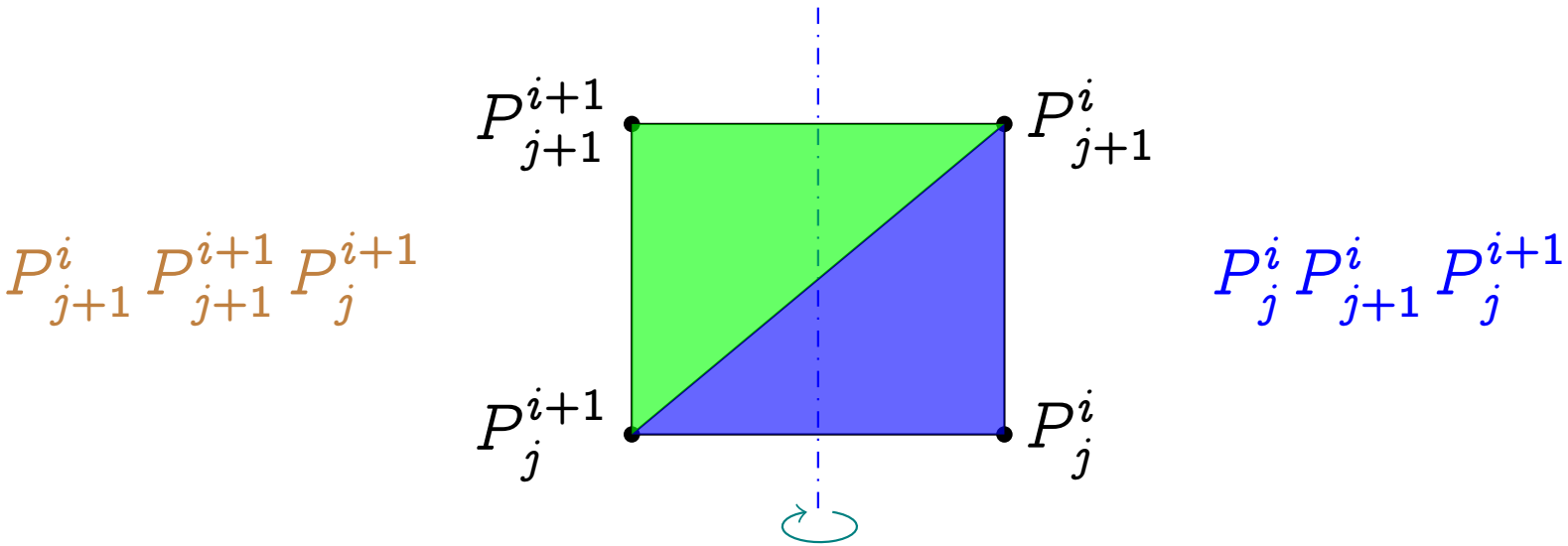
La malla se renderiza con `glDrawElements`:

```
void IndexMesh::draw() const
{
   glDrawElements(
      mPrimitive,       // primitiva (GL_TRIANGLES, etc.)
      vIndices.size(),  // número de índices
      GL_UNSIGNED_INT,  // tipo de los índices
      nullptr           // offset en el VBO de índices
   );
}
```

Si el VAO no contiene un VBO de índices estos se pueden pasar por el cuarto argumento como un puntero al vector en memoria.

Ejemplo: mallas por revolución

- Se generan `nMuestras` · `tamPerfil` vértices
- Se especifican las caras refiriéndose a esos vértices



- El índice de P^i_j es i · `tamPerfil` $+ j$.
- Se especifican las caras refiriéndose a esos vértices

Constructor de la malla indexada (1/2)

```
IndexMesh* IndexMesh::generateByRevolution(
      const vector<vec2>& perfil, int nMuestras) {
   IndexMesh* mesh = new IndexMesh;
   mesh->mPrimitive = GL_TRIANGLES;
   int tamPerfil = perfil.size();
   mesh->vVertices.reserve(nMuestras * tamPerfil);

   // Genera los vértices de las muestras
   GLdouble theta1 = 2 * numbers::pi / nMuestras;

   for (int i = 0; i ≤ nMuestras; ++i) {  // muestra i-ésima
      GLdouble c = cos(i * theta1), s = sin(i * theta1);
      for (auto p : perfil)  // rota el perfil
         mesh->vVertices.emplace_back(p.x * c, p.y, - p.x * s);
   }
```

Constructor de la malla indexada (2/2)

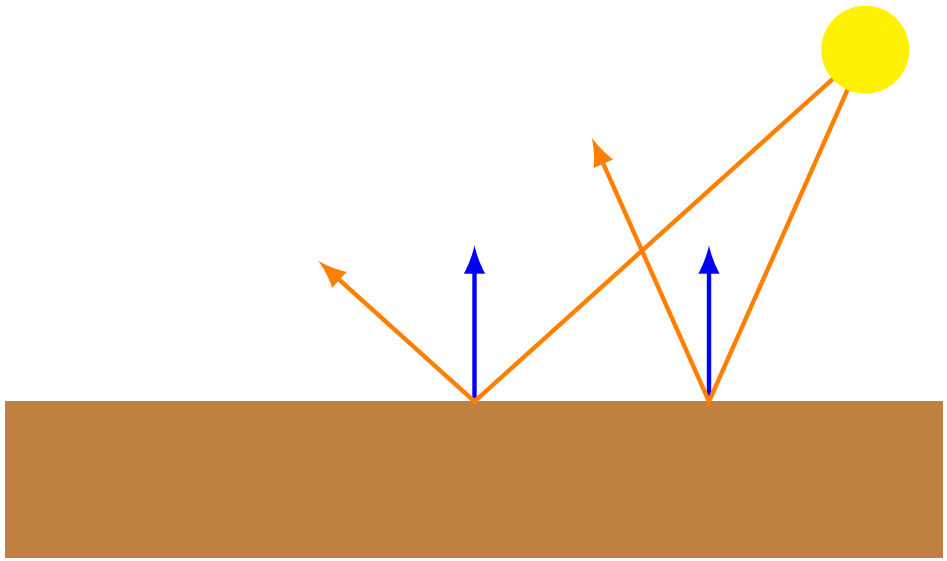
```
for (int i = 0; i < nMuestras; ++i)  // caras i a i + 1
  for (int j = 0; j < tamPerfil - 1; ++j) { // una cara
    if (perfil[j].x ≠ 0.0)  // triángulo inferior
      for (auto [s, t] : {pair{i, j}, {i, j+1}, {i+1, j}})
        mesh->vIndices.push_back(s * tamPerfil + t);

    if (perfil[j + 1].x ≠ 0.0)  // triángulo superior
      for (auto [s, t] : {pair{i, j+1}, {i+1, j+1}, {i+1, j}})
        mesh->vIndices.push_back(s * tamPerfil + t);
  }

  mesh->mNumVertices = mesh->vVertices.size();
  return mesh;
}
```

Vectores normales

En la iluminación de los objetos influyen factores como el color, la distancia a la fuente de luz o la orientación de sus superficies. La orientación queda definida por los vectores normales.



Vectores normales

- Cada componente de la tabla o array de normales es un **vector normal** de un vértice, es decir, es perpendicular a la tangente en ese vértice al objeto *malleado*.
- Hay tantos vectores normales como vértices.

Vectores normales

- Cada vector normal:
  - constituye un atributo más del vértice.
  - es perpendicular a la cara en ese vértice (es decir, el producto escalar con el vector tangente es igual a 0).
  - apunta hacia el exterior del objeto.
  - debe estar normalizado (vector de módulo 1).
- Vector normal y sombreado del objeto (**shading model**).

Vectores normales

- Los vectores normales de una cara se pueden calcular a partir de los vértices que forman la cara.
- Se sigue el convenio de proporcionar los índices de los vértices de cada cara en sentido antihorario (`GL_CCW`) según se mira la cara del objeto desde el exterior del mismo.
- Este orden permite distinguir el interior y el exterior del objeto y a OpenGL le permite diferenciar entre caras frontales (`GL_FRONT`) y caras traseras (`GL_BACK`).
- Los vectores normales se utilizan en el proceso de iluminación para determinar el color de los vértices.

Mallas con vectores normales

- Para tener en cuenta los vectores normales añadimos a la clase `Mesh` un atributo para el array de vectores normales y otro para el identificador de su VBO:

```
class Mesh {
protected:
  std::vector<glm::vec3> vNormals;  // en la CPU
  GLuint mNBO;                      // en la GPU
  ...
public:
  virtual void render();
  ...
};
```

Mallas con vectores normales

- Es preciso también extender `Mesh::load` como de costumbre

```
if (vNormals.size() > 0) {    // upload normals
   glGenBuffers(1, &mNBO);

   glBindBuffer(GL_ARRAY_BUFFER, mNBO);
   glBufferData(GL_ARRAY_BUFFER,
                vNormals.size() * sizeof(vec3),
                vNormals.data(), GL_STATIC_DRAW);
   glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE,
                         sizeof(vec3), nullptr);
   glEnableVertexAttribArray(3);
}
```

Usaremos (arbitrariamente) el atributo 3 del VAO.

Cálculo de los vectores normales

- En el renderizado de objetos malleados, los vectores normales son perpendiculares a las caras, pero se asocian a los vértices.
- Lo habitual es calcular el vector normal a una cara y usarlo como parte del vector normal para los vértices de esa cara.
- Como hay tantos vectores normales como vértices, el vector normal de un vértice se puede calcular como la suma (normalizada) de los vectores normales de las caras en las que participa el vértice.

Cálculo de los vectores normales

- A veces se usan sumas ponderadas por el ángulo o por el área que forman las caras que concurren en un vértice.
- En el renderizado de objetos con superficies curvas de ecuaciones conocidas (esferas, cilindros, etc.): el vector normal de un vértice se calcula a partir de las ecuaciones (paramétricas o implícita) de la superficie.

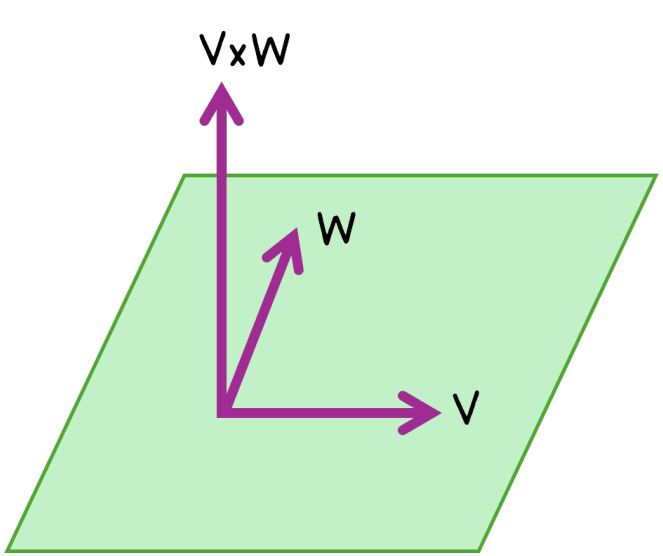
Producto vectorial de dos vectores

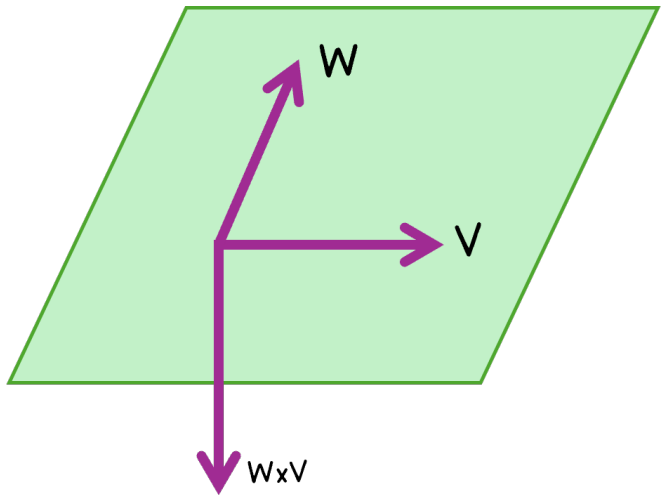
$$\left.\begin{aligned} V &= (v_1, v_2, v_3) \\ W &= (w_1, w_2, w_3) \end{aligned}\right\}$$

$$V \times W = \begin{vmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix}$$

$$|V \times W| = |V| \cdot |W| \cdot \sin(\theta)$$

Vector normal al plano formado por V y W





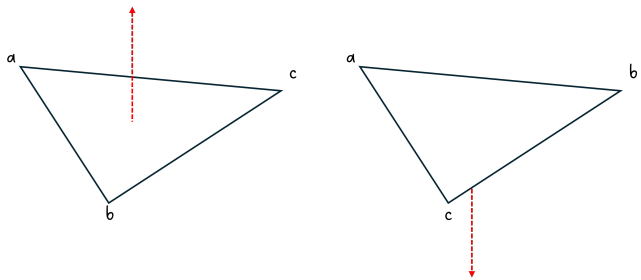
Cálculo del vector normal a una cara

- Vector normal a un triángulo (`a`, `b`, `c`) dado en el orden CCW es un vector unitario (módulo 1) perpendicular al plano determinado por los vértices del triángulo y que apunta hacia fuera.
- Se puede obtener como el producto vectorial de dos vectores en CCW, normalizando: $\overrightarrow{ab} \times \overrightarrow{ac}$, $\overrightarrow{ba} \times \overrightarrow{bc}$, ...

```
normalize(cross(b - a, c - a))
```

- El orden de los vértices es importante:

```
vector_normal(vértices CW) = -vector_normal(vértices CCW)
```



Cálculo del vector normal a una cara

- El vector `n` normal a una cara formada por los vértices de índices `ind0`, `ind1` e `ind2` se puede calcular:
  - Usando el producto vectorial. Sea `vi` el vértice de índice `indi`:

    ```
    n = normalize(cross(v2 - v1, v0 - v1))
    ```

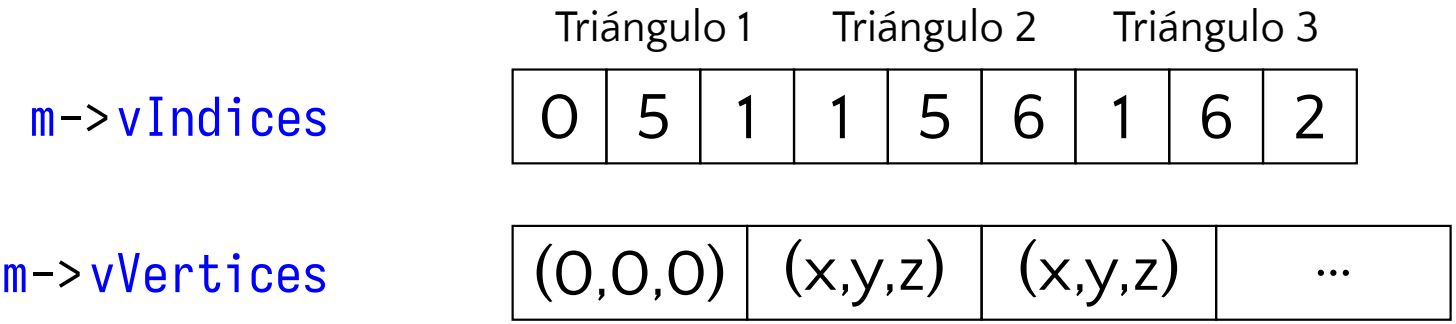
  - O también:

    ```
    n = normalize(cross(v1 - v0, v2 - v0))
    ```

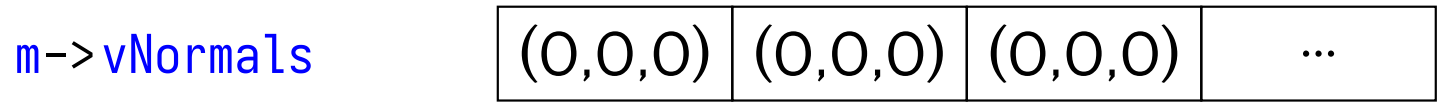
    Inconvenientes de este método
  - Usando el método de Newell.

Método de Newell

- Construir el vector de normales del mismo tamaño que el de vértices



- Inicializar las componentes del vector de normales al vector 0



Método de Newell

- Recorrer los triángulos, es decir, recorrer `m->indices` haciendo:
  - Extraer los índices del triángulo `a`, `b`, `c`.
  - Calcular el vector `n` normal al triángulo tal como se ha explicado.
  - Sumar `n` al vector normal de cada vértice del triángulo.

- Normalizar los vectores de `m->normals`

Método de Newell

Normales n_j para una malla con n vértices v_j y $3m$ índices i_k:

n_j = {0.0, 0.0, 0.0} para $j = 1, \ldots, n$

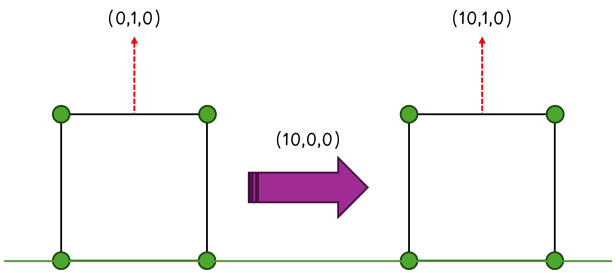
```
for (k = 0; k < 3m; k += 3) {
    vec3 normal = normalize(cross(v_{i_{k+1}} - v_{i_k}, v_{i_{k+2}} - v_{i_k}));

    n_{i_k}   += normal;      // suma la normal del triángulo
    n_{i_{k+1}} += normal;    // a todos sus vértices
    n_{i_{k+2}} += normal;
}
```

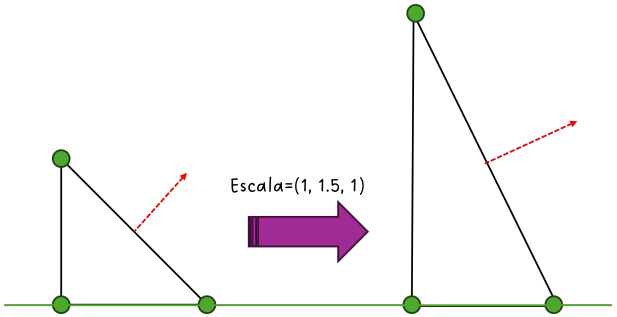
n_j = normalize(n_j) para $j = 1, \ldots, n$

Transformación de vectores normales

- **Vectores normales y matriz de modelado**

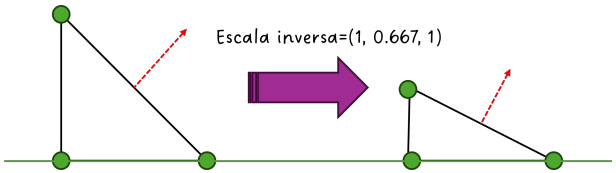


Si la normal se transforma

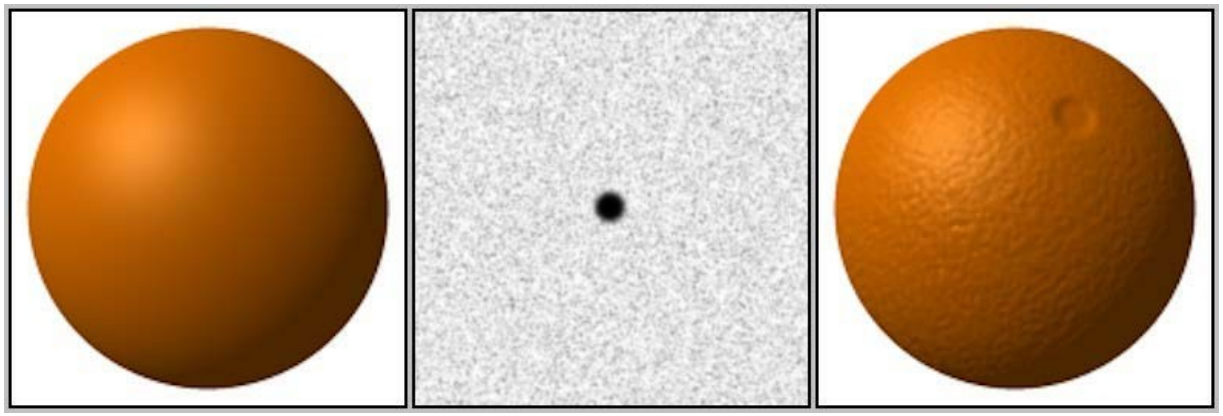


Si la normal se transforma

- **Matriz de transformación de vectores normales**: ignora la translación e invierte la escala



Bump mapping



- Cuando se usan ciertas técnicas (*bump mapping* o aspecto de piel de naranja) se pueden especificar vectores normales por fragmento.
- Permite dar cierto aspecto (por ejemplo, rugosidad) sin cambiar la geometría del objeto.
- Las normales se realinean siguiendo un cierto patrón.
- Método desarrollado por James Blinn.