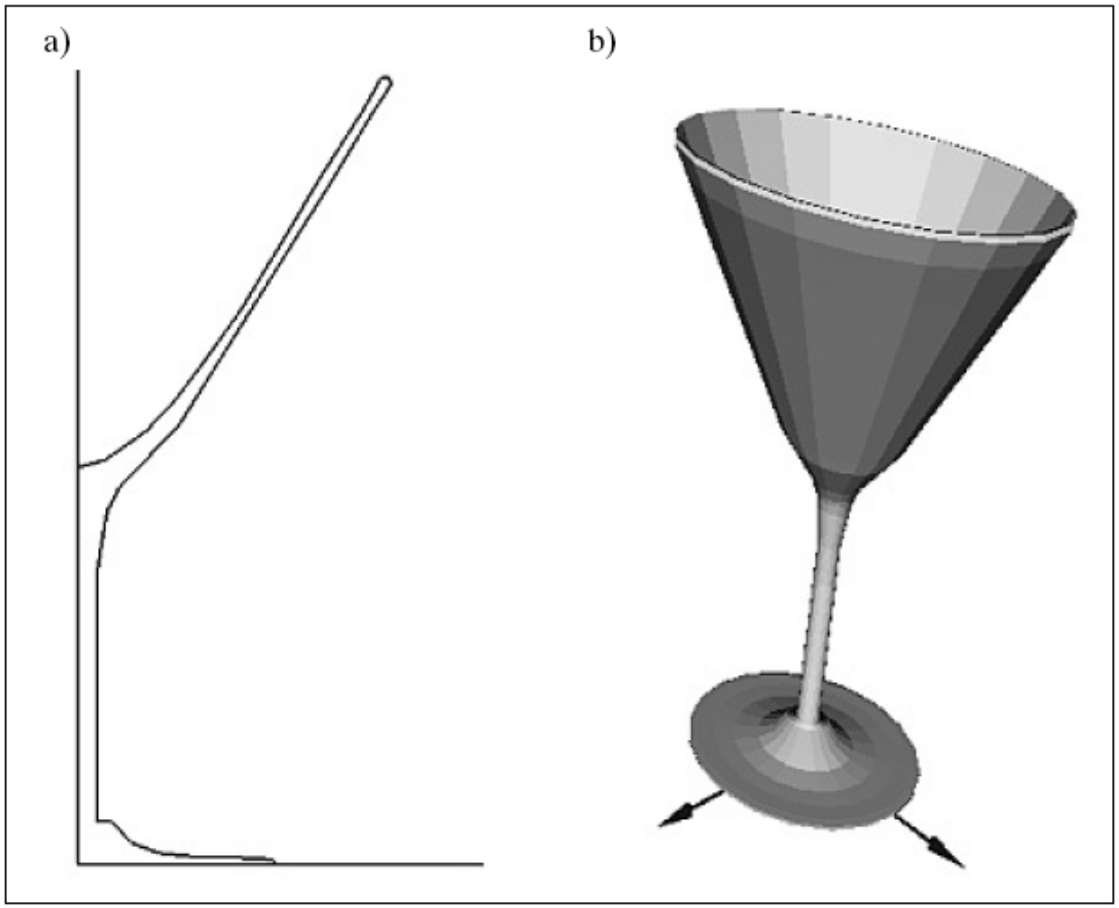
Construcción de una malla por revolución en 3D

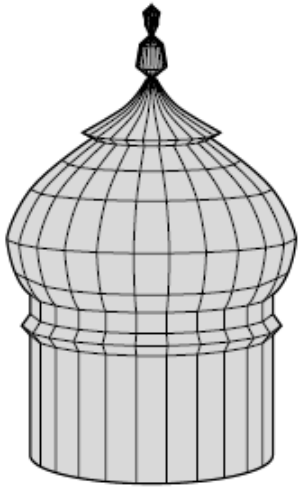
A. Gavilanes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Mallas por revolución



Mallas por revolución

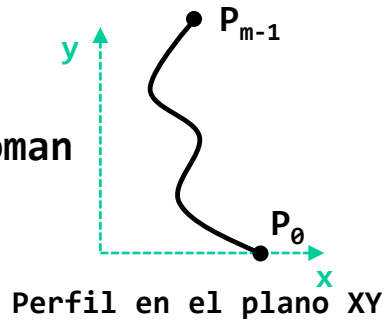
- Ingredientes para definir una malla por revolución
  - Un perfil formado por los puntos **{P0, …, Pm-1}** sobre el plano **XY**
  - Vértices: los obtenidos aplicando las sucesivas rotaciones **Rot(360/n,(0,1,0))** a los puntos del perfil, donde **n** es el número de veces que se van a tomar muestras durante una revolución alrededor del eje **Y**
  - Algunos vértices pueden repetirse. Por ejemplo, el punto más alto de la cúpula del Taj Mahal
  - Caras: las cuadrangulares obtenidas uniendo dos puntos de un perfil con los dos correspondientes del perfil siguiente



Construcción de una malla por revolución

(1) Crear una nueva clase **MBR** (de **Mesh By Revolution**) que hereda de la clase **Mesh** y que tiene los siguientes atributos:

```
(int) m: número de puntos del perfil
(int) n: número de rotaciones (muestras) que se toman
(dvec3*) perfil: perfil original en el plano XY
```



Perfil en el plano XY

En las explicaciones que siguen se supone que los puntos del perfil van de abajo arriba, tal como se muestra en la figura adjunta.

(2) La constructora de la clase **MBR** tiene tres parámetros y da valor a los atributos de la forma obvia.

(3) Definir en la clase **MBR** el método **void vertexBuilding()** que, invocado desde la constructora, obtiene los vértices de la malla.

Construcción de una malla por revolución

```
void vertexBuilding() {
    ... // Definir el array vertices de tamaño numVertices(=n*m)
    for (int i=0; i<n; i++) {
      // Cada vuelta genera la muestra i-ésima de vértices
      double theta = i*2*PI / n;
      double c = cos(theta);
      double s = sin(theta);
      // R_y de más abajo es la matriz de rotación sobre el eje Y
      for (int j=0; j<m; j++) {
          int indice = i*m+j;
          // Aplicar la matriz al punto j-ésimo del perfil
          double x =  c*perfil[j][0] + s*perfil[j][2];
          double z = -s*perfil[j][0] + c*perfil[j][2];
          dvec3 p = dvec3(x, perfil[j][1], z);
          vertices[indice] = p;
      }
}
```

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

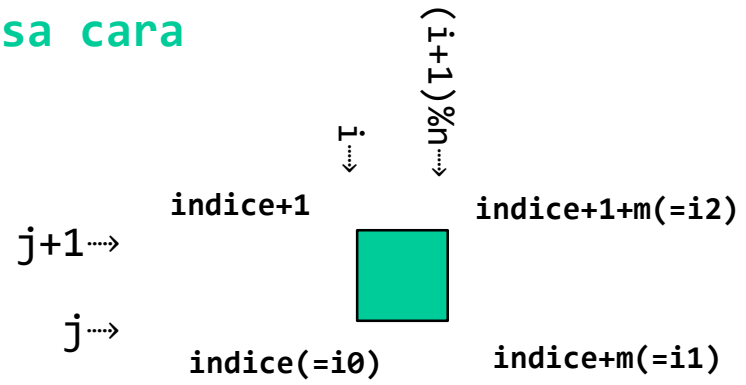
(4) Definir en la clase **MBR** el método **void normalize()** que, invocado desde la constructora, obtiene las normales a los vértices de la malla a base de hacer un recorrido que, por cada uno de ellos, suma al vector normal obtenido hasta ese momento para ese vértice, el de la cara a la que pertenece como esquina inferior izquierda

```
void normalize() {
    ... // Definir el array normals de tamaño numVertices
    // e inicializar sus componentes al vector cero(=dvec3(0,0,0))

    for (int i = 0; i < n; i++)
      for (int j = 0; j < m-1; j++) {
        // Recorrido de todos los vértices
        // Ojo, i<n (obliga a usar %(n*m))
        // y j<m-1 (para excluir los vértices del borde superior)
        int indice = i*m + j;
```

Construcción de una malla por revolución

```
void normalize() {
      // (continuación)
      // Por cada cara en la que el vértice ocupa la esquina
      // inferior izquierda, se determinan 3 índices i0, i1, i2
      // de 3 vértices consecutivos de esa cara
      dvec3 aux0 = vértices[i0];
      dvec3 aux1 = vértices[i1];
      dvec3 aux2 = vértices[i2];
```



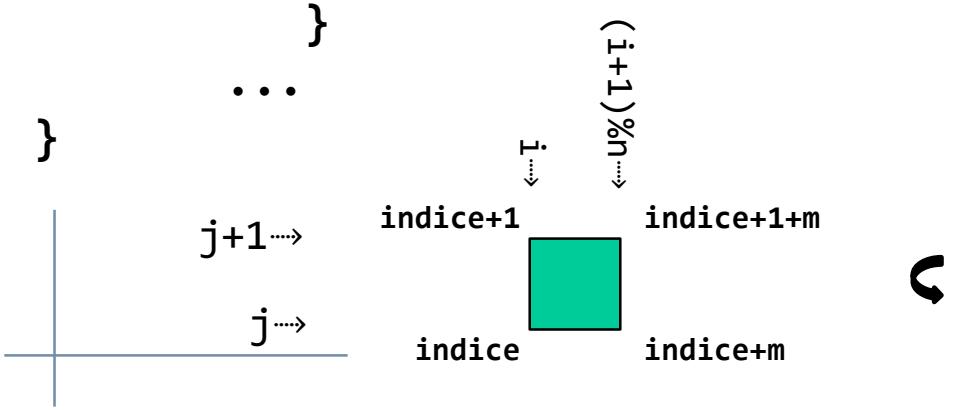

```
      dvec3 norm = glm::cross(aux2 – aux1, aux0 – aux1);
      normals[i0] += norm; normals[i1] += norm;
      normals[i2] += norm; normals[...] += norm;
      }  // Fin del for j
      // Se normalizan todos los vectores normales
      ...
      normals[i] = glm::normalize(normals[i]);
   }
```

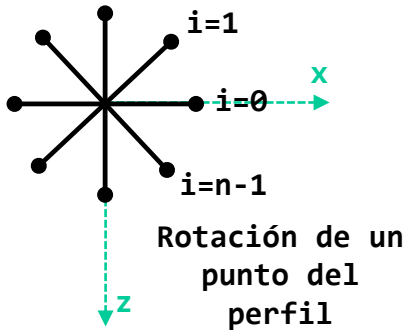
(5) Se reescribe el método **render()** de la clase **MBR**

```
void MPR::render() {
        if (vertices != nullptr) {
                glEnableClientState(GL_VERTEX_ARRAY);
                glVertexPointer(3, GL_DOUBLE, 0, vertices);
                // Activación de vertex arrays de colores y
                // coordenadas de textura, si hace el caso.
                // No olvidar desactivarlos
                if (normals != nullptr) {
                        glEnableClientState(GL_NORMAL_ARRAY);
                        glNormalPointer(GL_DOUBLE, 0, normals);
```

Construcción de una malla por revolución usando índices

```
void render() {
        // (continuación)
        primitive = GL_POLYGON; // o GL_LINE_LOOP
        // Se dan índices de vértices de caras cuadrangulares
        for (int i=0; i<n; i++)
           // Unir muestra i-ésima con (i+1)%n-ésima
           for (int j=0; j<m-1; j++) {
               // Empezar en esquina inferior izquierda de la cara
               int indice = i*m+j;
               unsigned int index[] =
                       {indice, (indice + m)%(n*m),
                       (indice + m + 1)%(n*m), indice + 1};
               glDrawElements(primitive, 4, GL_UNSIGNED_INT, index);
           }
        ...
}
```





Rotación de un punto del perfil

Ejemplo. Construcción de un cono por revolución

- Definir una nueva entidad llamada **Cone**
- Definir la constructora de **Cone**:

```
Cone::Cone(GLdouble h, GLdouble r) {
    // h=altura del cono, r=radio de la base
    int m = 3; // m=número de puntos del perfil
    dvec3* perfil = new dvec3[m];
    perfil[0] = dvec3(0.0, 0.0, 0.0);
    perfil[1] = dvec3(r, 0.0, 0.0);
    perfil[2] = dvec3(0.0, h, 0.0);
    this->mesh = new MBR(m, 50, perfil);
}
```