Modelado de superficies en 3D

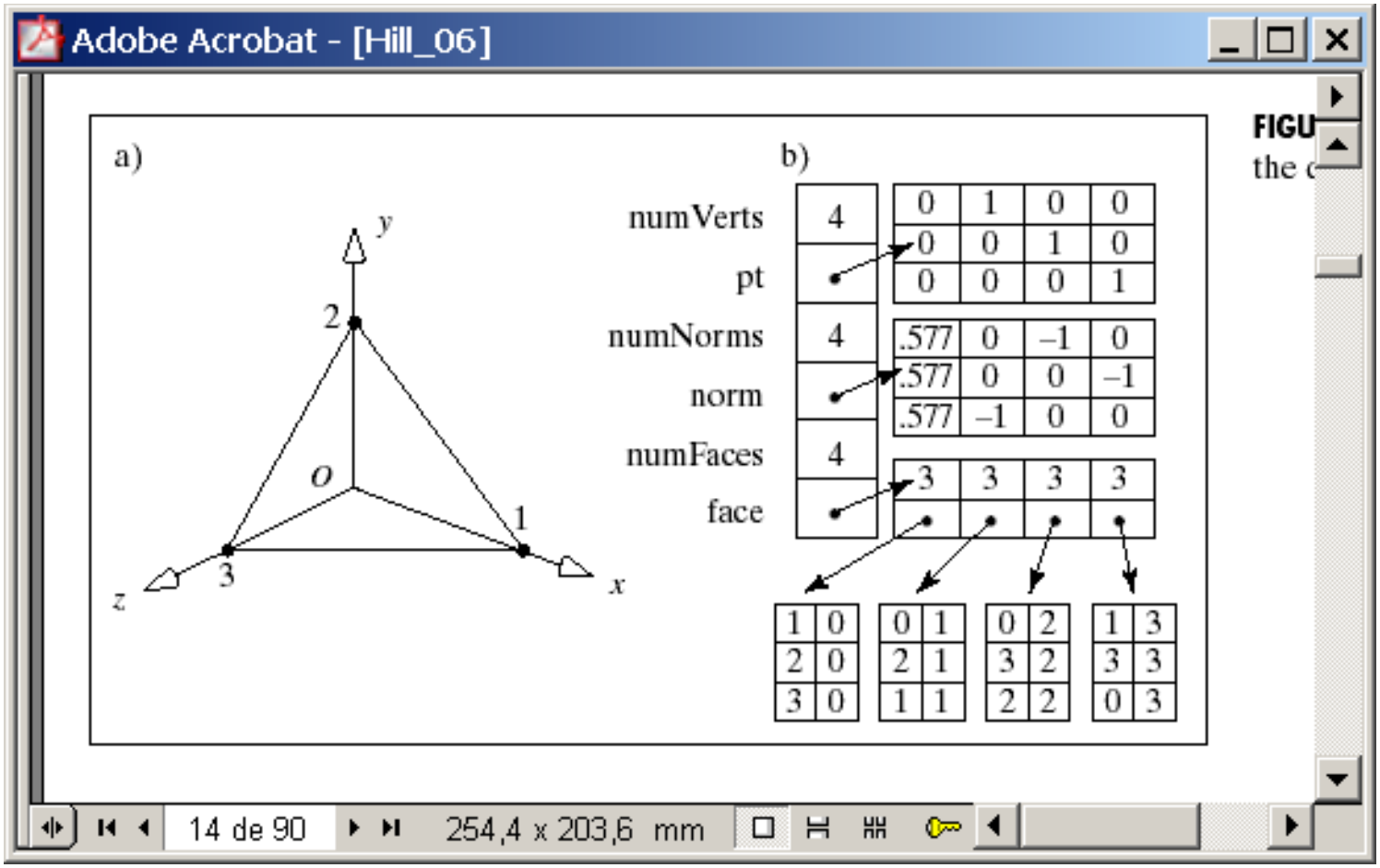
P. J. Martín, A. Gavilanes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Formas de representación de superficies

- 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
- Elemento más en la representación de un objeto (color, material, textura)

- Elementos de una malla
  - **Tabla de vértices,** que proporciona la información posicional o geométrica del objeto mediante las coordenadas de los vértices.
  - **Tabla de normales,** que proporciona la información sobre la orientación de las caras mediante las coordenadas de las normales.
  - **Tabla de caras,** que proporciona la información topológica sobre la conectividad en el objeto mediante la definición de los polígonos.

Mallas de polígonos



- Cada componente del array de vértices es un puntero a un vértice.
- Cada componente del array de normales es un puntero a un vector normal.
  - Vector normal por vértice vs vector normal por cara.
  - Vector normal normalizado.
  - Los vectores normales apuntan hacia el exterior del objeto.
  - Vector normal y sombreado del objeto.
  - El comando **`glEnable(GL_NORMALIZE);`**

- Cada cara es un array de tantos pares de la forma (índice de vértice, índice de normal), como vértices formen la cara.
  - Se sigue el convenio de proporcionar los vértices de una cara en sentido anti-horario (CCW), cuando la cara se mira desde el exterior del objeto.
  - El sentido de los vértices permite a OpenGL identificar las caras frontales (**`GL_FRONT`**) y las traseras (**`GL_BACK`**).
  - El comando

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

- La clase PV3D (de PuntoVector3D):

```
class PV3D {
  private:
        GLfloat x, y, z;
        int pv;
        …
  public:
        void normaliza();
        PV3D* clona();
        Glfloat productoEscalar(PV3D* v) {…}
        PV3D* productoVectorial(PV3D* v) {…}
        …
}
```

- La clase Cara:

```
class Cara {
  private:
        int numVertices;
        VerticeNormal** arrayVN;
  …
}
```

- La clase VerticeNormal:

```
class VerticeNormal {
  private:
        int indiceVertice, indiceNormal;
  …
}
```

La clase Malla:

```
class Malla {
  protected:
     int numVertices;
     PV3D** vertice;
     int numNormales;  //=numCaras, frecuentemente
     PV3D** normal;
     int numCaras;
     Cara** cara;
  public:
     void dibuja();
     …
}
```

El método dibuja():

```
void dibuja() {
    for (int i=0; i<numeroCaras; i++) {
      glLineWidth(1.0);
      glBegin(GL_POLYGON);   //o glBegin(GL_LINE_LOOP);
      for (int j=0; j<cara[i]->getNumeroVertices(); j++) {
          int iN=cara[i]->getIndiceNormal(j);
          int iV=cara[i]->getIndiceVertice(j);
          glNormal3f(normal[iN]->getX(),normal[iN]->getY(),normal[iN]->getZ());
          //Si hubiera coordenadas de textura, aquí se suministrarían
          //las coordenadas de textura del vértice j con glTexCoor2f(…);
          glVertex3f(vertice[iV]->getX(),vertice[iV]->getY(),vertice[iV]->getZ());
      }
      glEnd();
    }
}
```

Cálculo de vectores normales

- Producto vectorial
- Formas de calcular el vector normal a una cara formada por los vértices {v0, v1, …, vn}
  - Usando el producto vectorial:

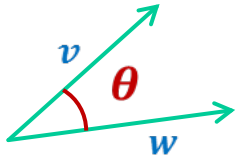
    n=(v2-v1)x(v0-v1)

    Inconvenientes
  - Usando el método de Newell

- v y w son dos vectores en 3D

$$v \times w = \begin{pmatrix} v_y w_z - v_z w_y \\ v_z w_x - v_x w_z \\ v_x w_y - v_y w_x \end{pmatrix}$$

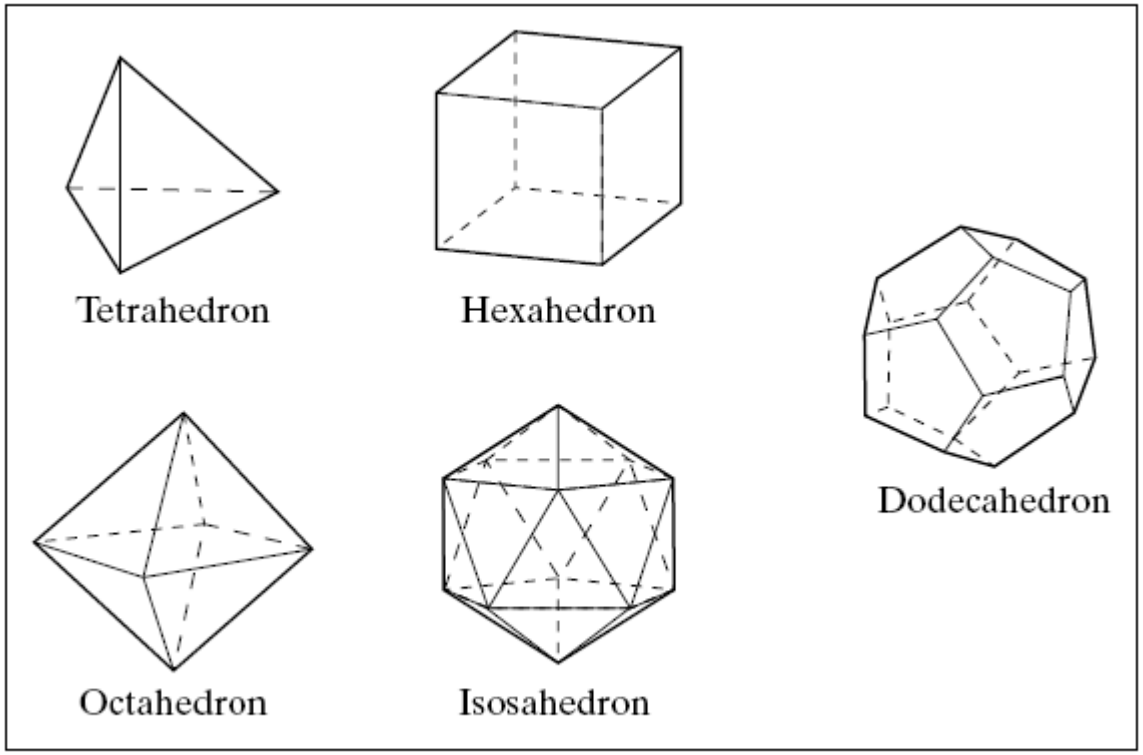
- $v \times w$ es un vector perpendicular a v y w, el sentido lo da la regla de la mano derecha
- Propiedades algebraicas y geométricas
  - $v \times w = -(w \times v)$
  - $v \times (w + u) = v \times w + v \times u$
  - $k(v \times w) = (kv) \times w$
  - $\|v \times w\| = \|v\| \, \|w\| \, |sin\, \theta|$
  - $\|v \times w\| = 0 \iff v$ y w son paralelos



$0 \leq \theta \leq 180°$

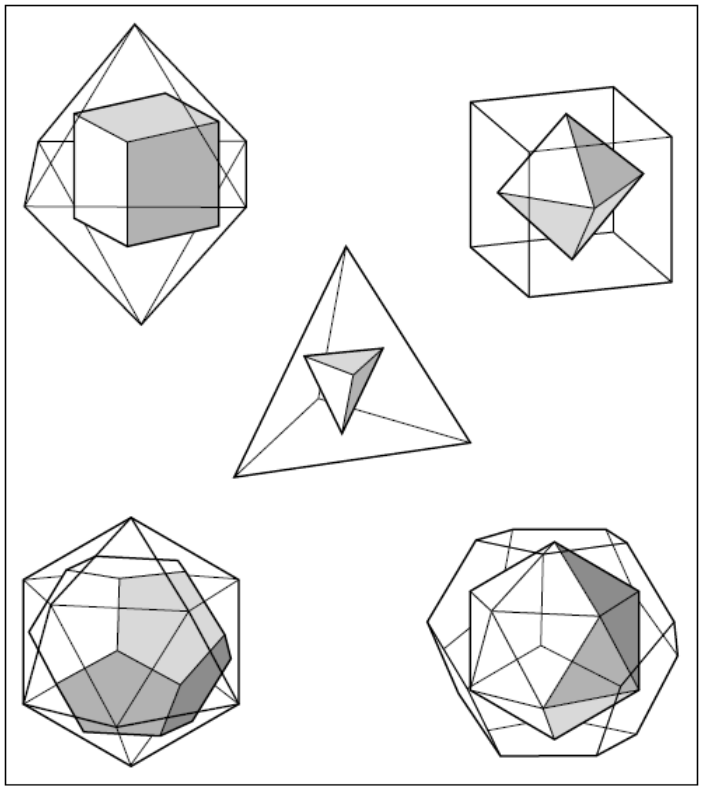
```
CalculoVectorNormalPorNewell(Cara C)
   n = (0, 0, 0);
   for i=0 to C.numeroVertices {
      vertActual=vertice[C->getIndiceVertice(i)];
      vertSiguiente=vertice[C->getIndiceVertice((i+1) % C.numeroVertices)];
      n.x+=(vertActual->getY()-vertSiguiente->getY())*
           (vertActual->getZ()+vertSiguiente->getZ());
      n.y+=(vertActual->getZ()-vertSiguiente->getZ())*
           (vertActual->getX()+vertSiguiente->getX());
      n.z+=(vertActual->getX()-vertSiguiente->getX())*
           (vertActual->getY()+vertSiguiente->getY());
   }
   return normaliza(n.x, n.y, n.z);
```

Sólidos platónicos

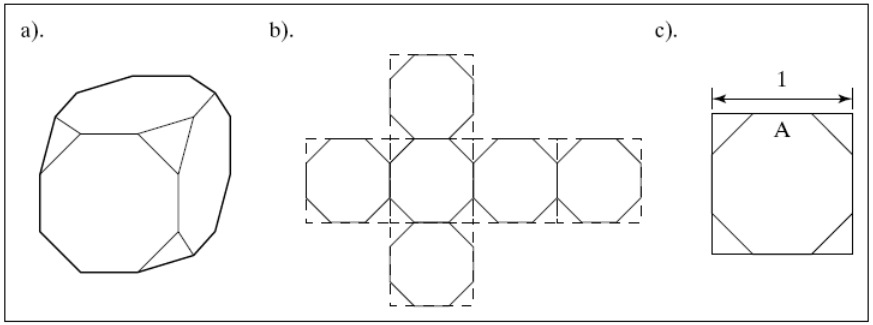


Sólidos platónicos y dualidad

Solid	V	F	E	Schläfli
Tetrahedron	4	4	6	(3, 3)
Hexahedron	8	6	12	(4, 3)
Octahedron	6	8	12	(3, 4)
Icosahedron	12	20	30	(3, 5)
Dodecahedron	20	12	30	(5, 3)



- Sólidos por truncamiento. Cubo truncado

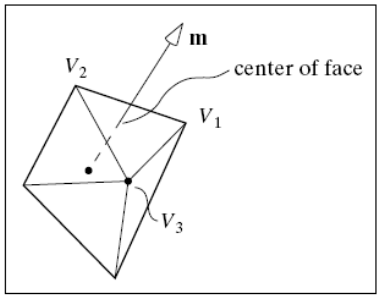


- Vértices obtenidos al truncar la arista CD

$$V=((1+A)/2)\ C + ((1-A)/2)\ D$$

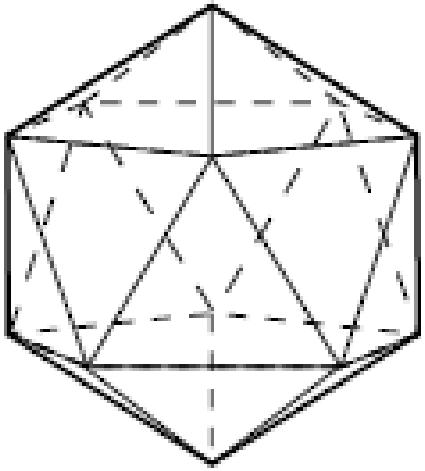
$$W=((1-A)/2)\ C + ((1+A)/2)\ D$$

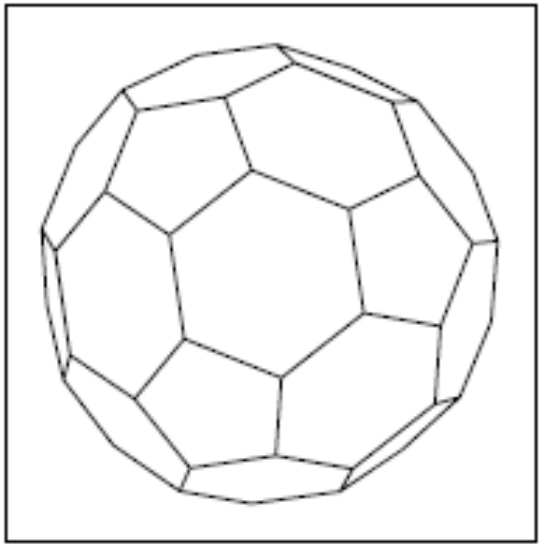
- Normales
  - **m**=(V1+V2+V3)/3



Icosaedro truncado

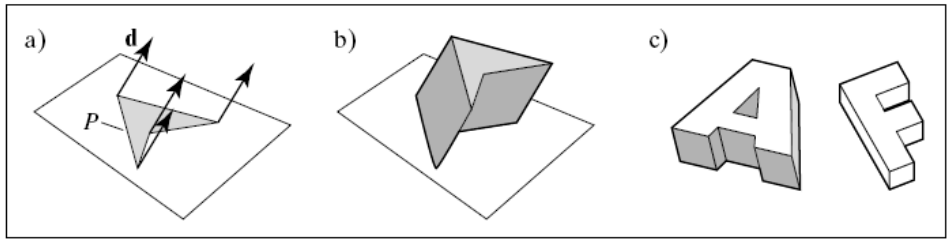
- Buckyball (fulereno), por Buckminster Fuller



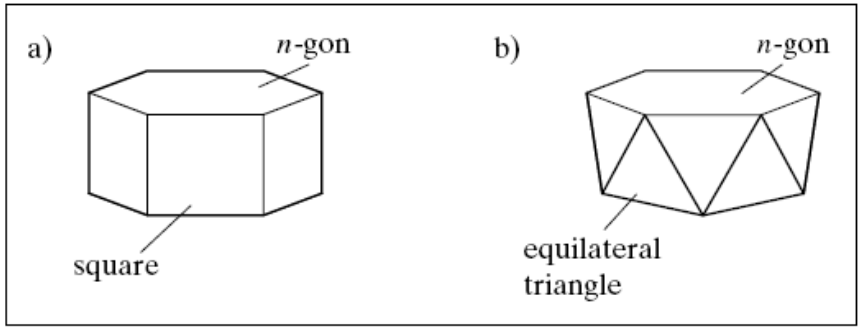


Prismas y antiprismas

- Extrudir (tr): Dar forma a una masa metálica, plástica, etc., haciéndola salir por una abertura especialmente dispuesta.
- Un prisma es un poliedro obtenido mediante extrusión de un polígono a lo largo de una línea recta.
  - Caras cuadrangulares



- Un antiprisma se obtiene a partir de un prisma, rotando la cara n-gonal superior, 180/n grados.
  - Caras triangulares
  - El octaedro es un antiprisma (Why?)



Malla para un prisma recto

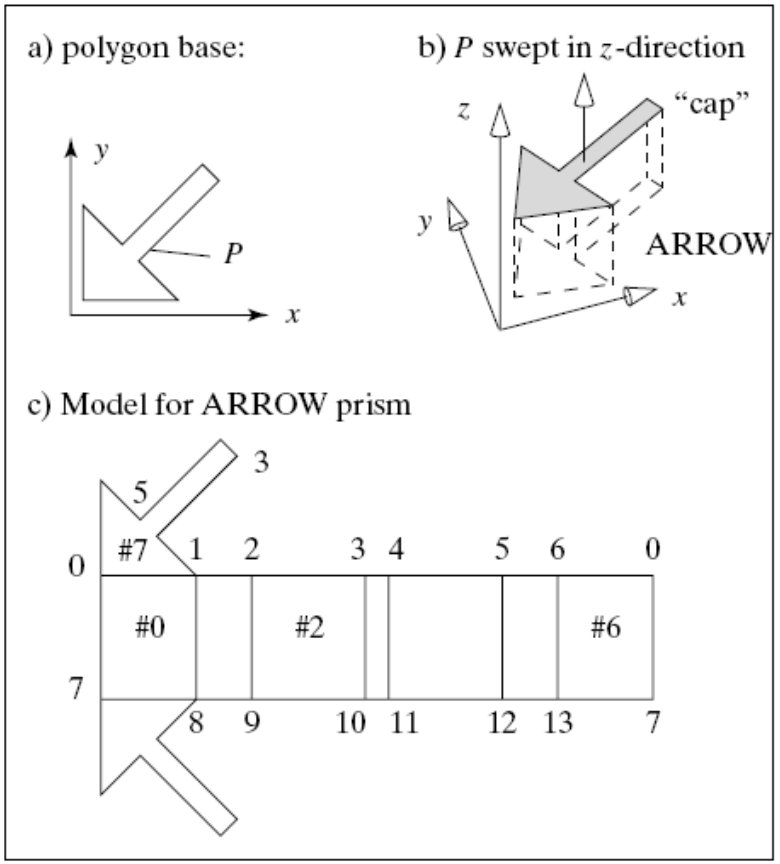
- Vértices:

  (xi, yi, 0), (xi, yi, H), 0≤i≤N-1

  donde N es el número de lados del polígono que origina el prisma y H es su altura

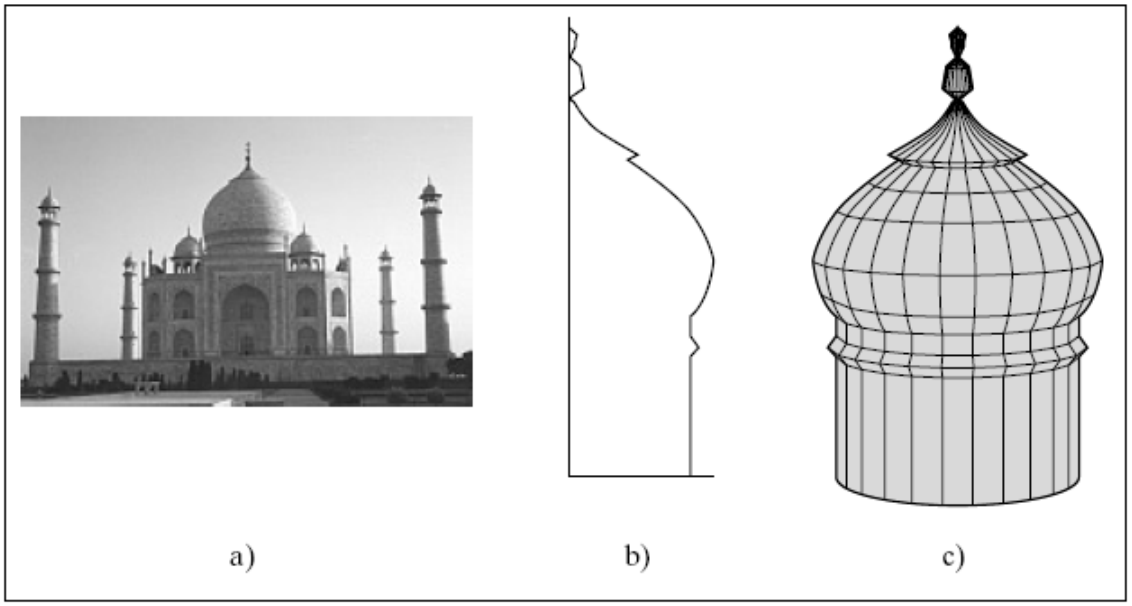
- Caras:
  - Laterales

    (j, suc(j), suc(j)+N, j+N), 0≤j≤N-1
  - Base (0, 6, 5, 4, 3, 2, 1)
  - Tapa (7, 8, 9, 10, 11, 12,13)
- Normales: Por Newell
- Ejemplo: Flecha de base heptagonal (N=7)

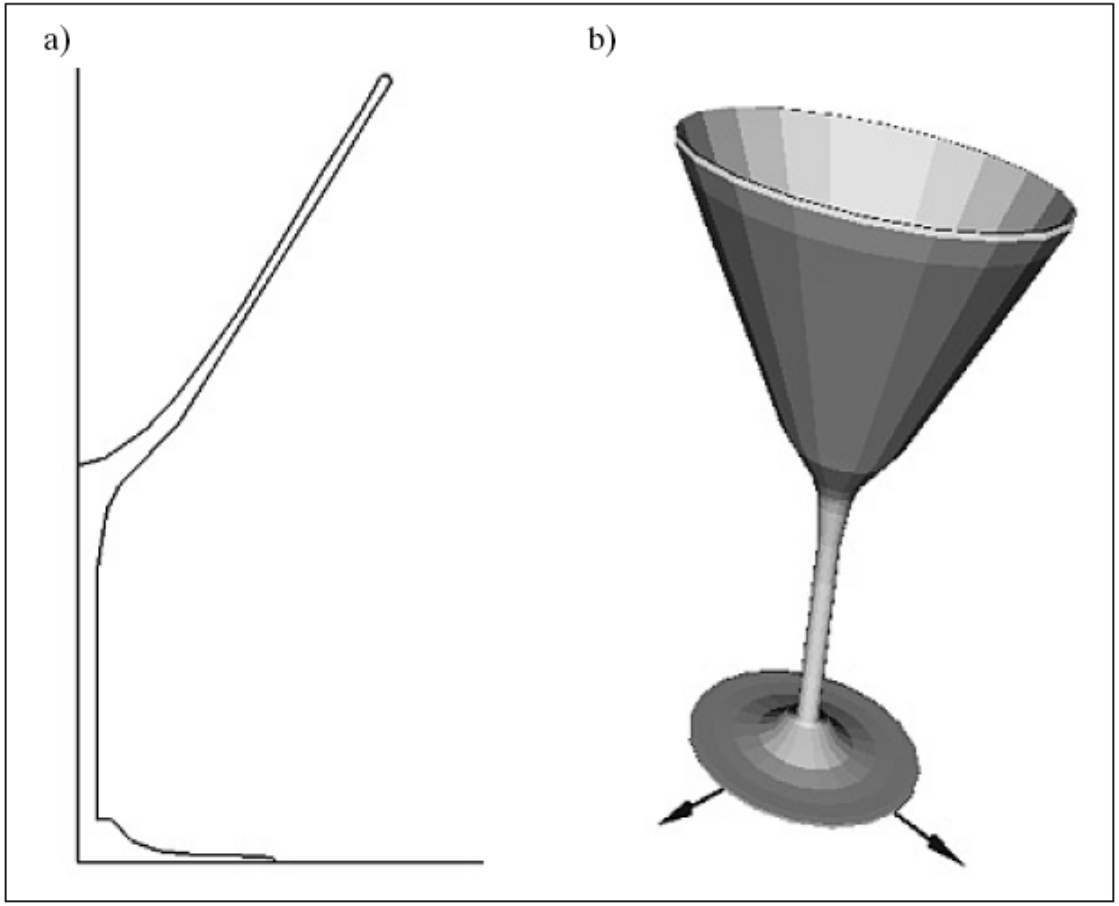


Mallas por revolución

- Ingredientes para definir una malla por revolución
  - Un perfil formado por los puntos {P_0, ..., P_{n-1}} sobre el plano XY
  - Vértices: los obtenidos aplicando las sucesivas rotaciones R(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
  - Caras: las cuadrangulares obtenidas uniendo dos puntos de un perfil con los dos correspondientes del perfil siguiente
  - Normales: Por Newell

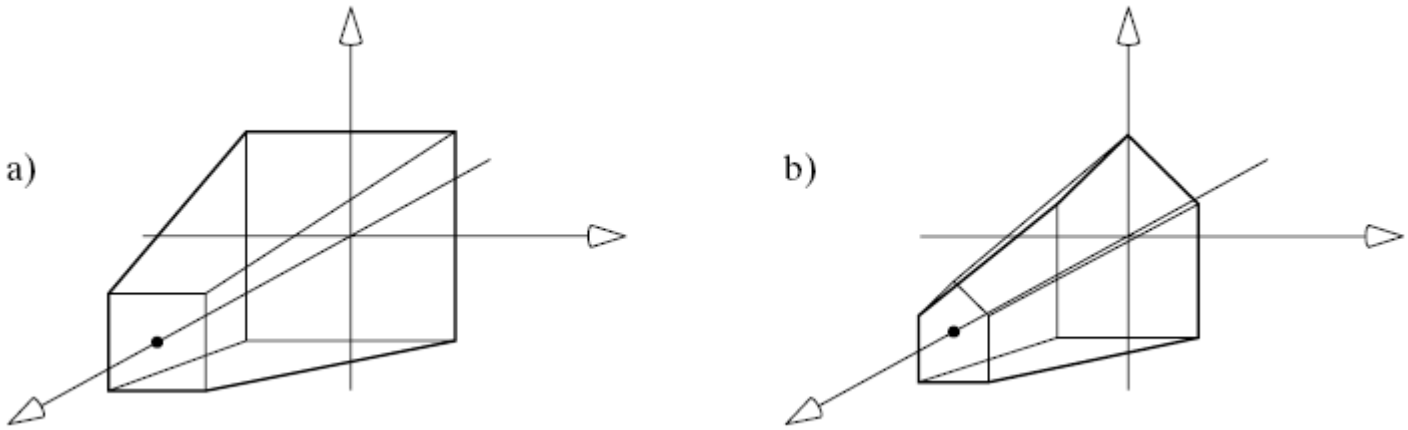


Mallas por revolución

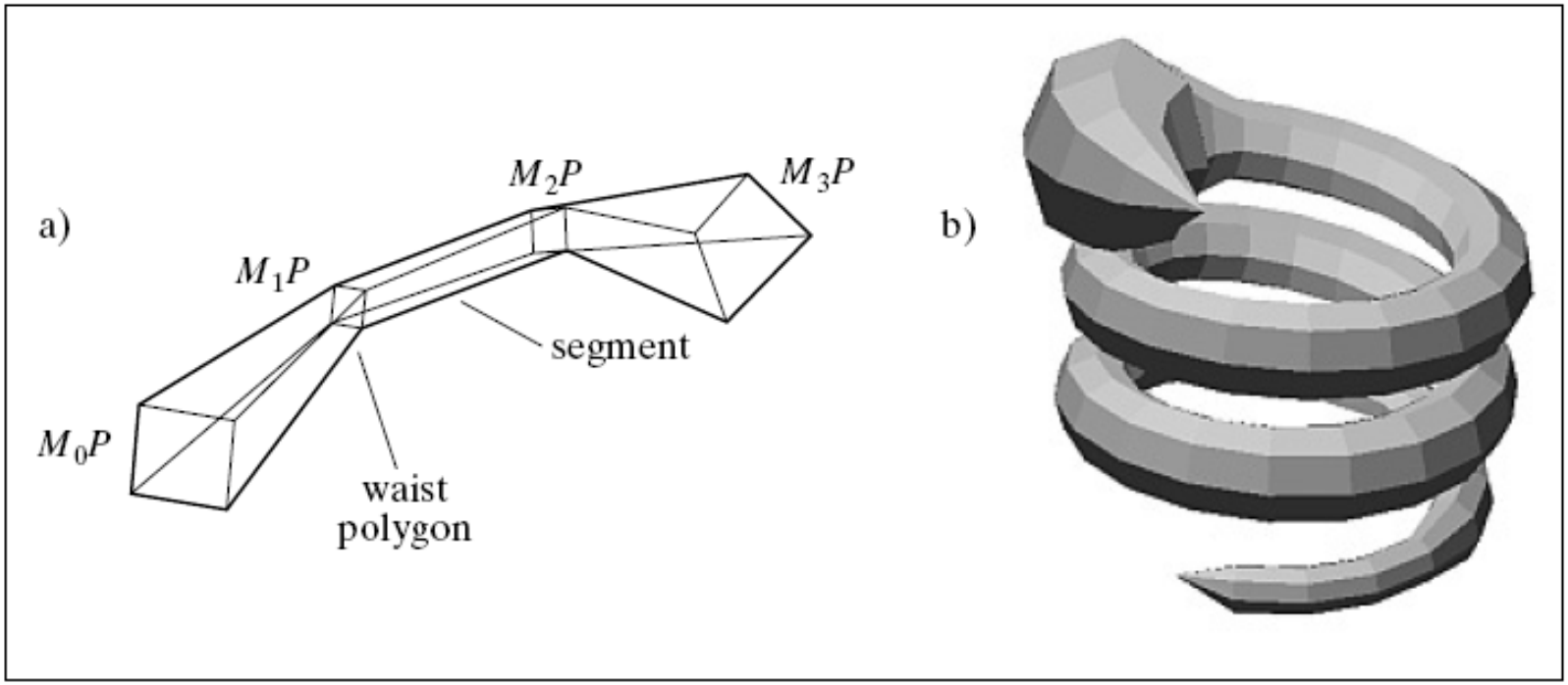


Mallas por extrusión

- Ingredientes para definir una malla por extrusión
  - Un polígono formado por los puntos $\{P_0, ..., P_{n-1}\}$
  - Una serie de transformaciones (posiblemente iguales) $\{M_0, ..., M_k\}$
  - Vértices del perfil j-ésimo: los obtenidos aplicando la transformación M_j a los puntos del polígono inicial $M_j(p_i)$, $0 \leq i \leq n-1$, $0 \leq j \leq k$
  - Caras: las cuadrangulares obtenidas como se vio con las caras laterales en los prismas rectos
  - Normales: Por Newell

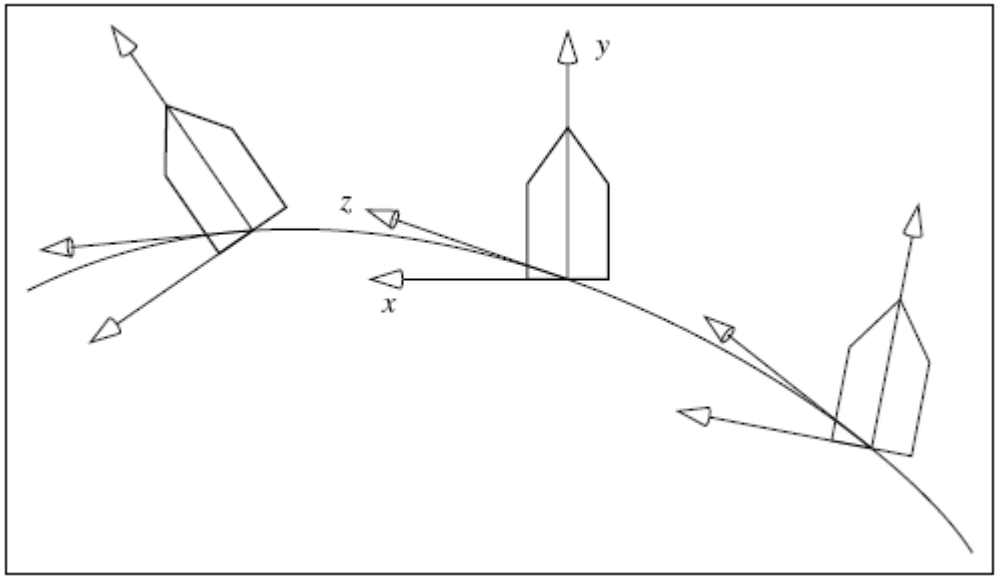


Mallas por extrusión



Marco de Frenet

- Se parte de **un perfil dado** y se producen sucesivos perfiles, por aplicación de una transformación que coloca el sistema de coordenadas local, que hemos usado para modelar el perfil, sobre un punto de la escena con la orientación deseada. Estos puntos recorren **una curva dada**.
- Los puntos de los sucesivos perfiles se unen, como se hace en cualquier malla por extrusión.



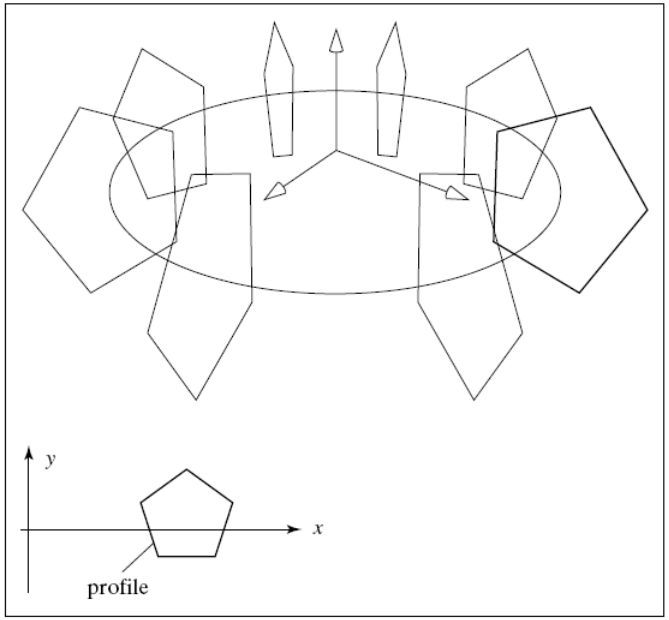
Toro usando el marco de Frenet

- Perfil inicial: polígono regular sobre el plano XY de nP lados, inscrito en una circunferencia de radio r

```
inc=(2*PI/nP);
for (int i=0; i<nP; i++)
   perfil[i]=new PV3D(r*cos(i*inc),
                      r*sin(i*inc),0,1);
```

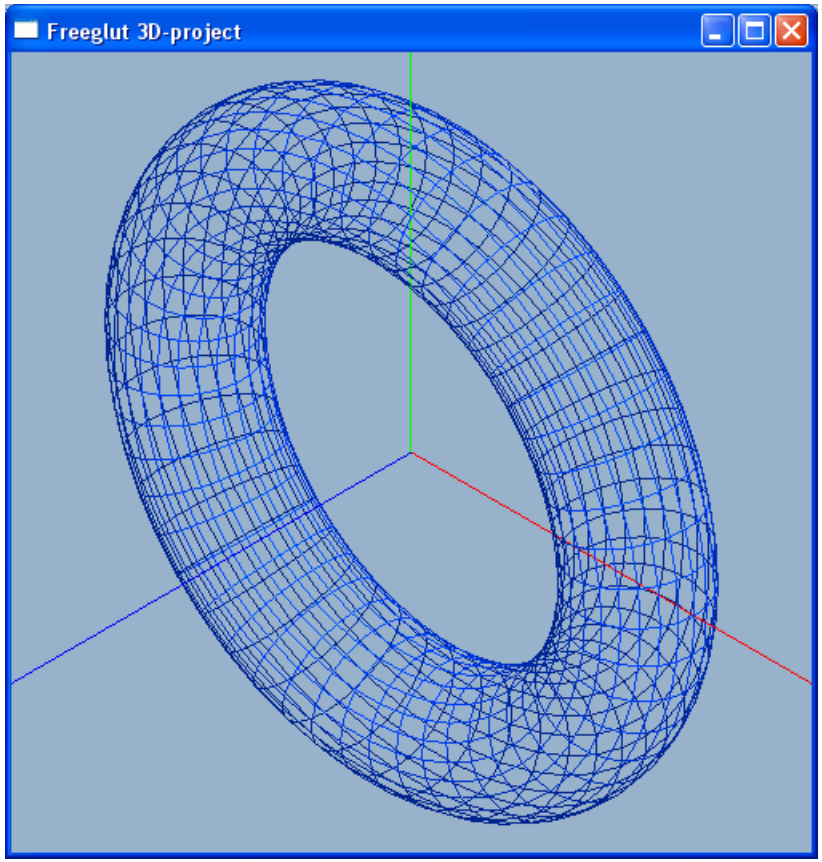
Coordenada homogénea:
1 para puntos, 0 para vectores

- Curva dada: C(t)=(R*cos(t), 0, R*sin(t)) (ecuaciones paramétricas de la circunferencia centrada en el origen, sobre el plano XZ, de radio R).

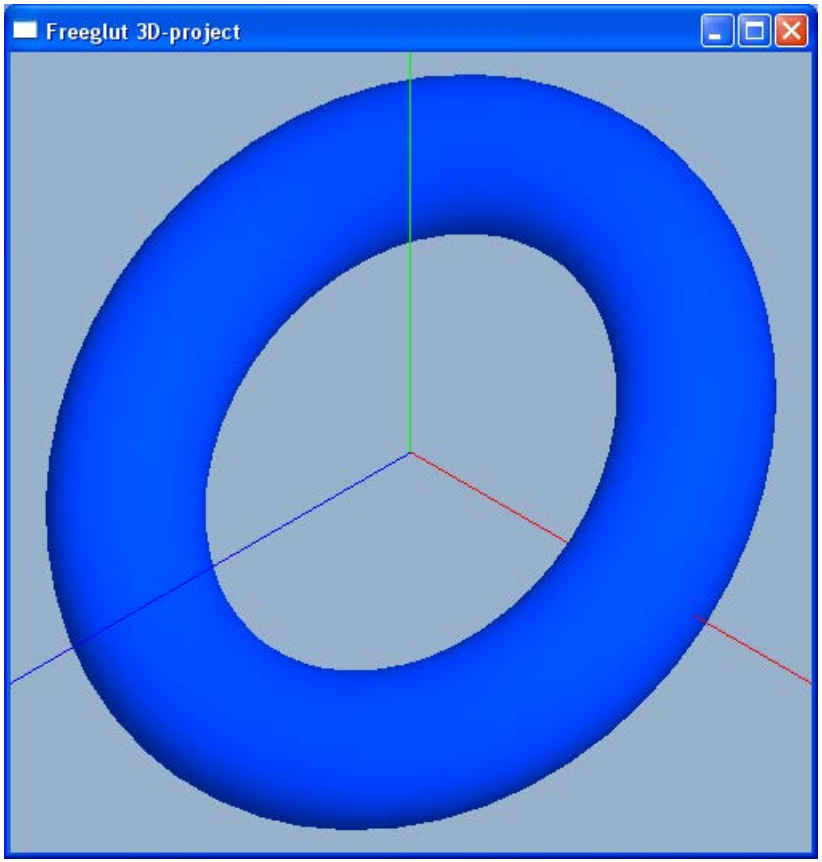


Toro usando las glut

```
glutWireTorus(2, 8, 25, 50);
```



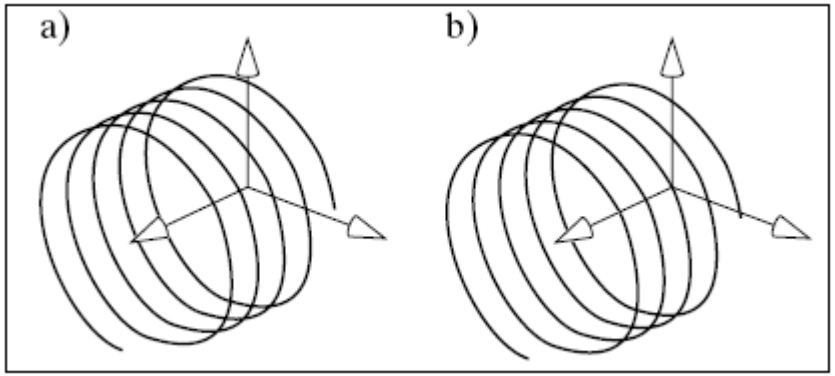

```
glutSolidTorus(2, 8, 25, 50);
```



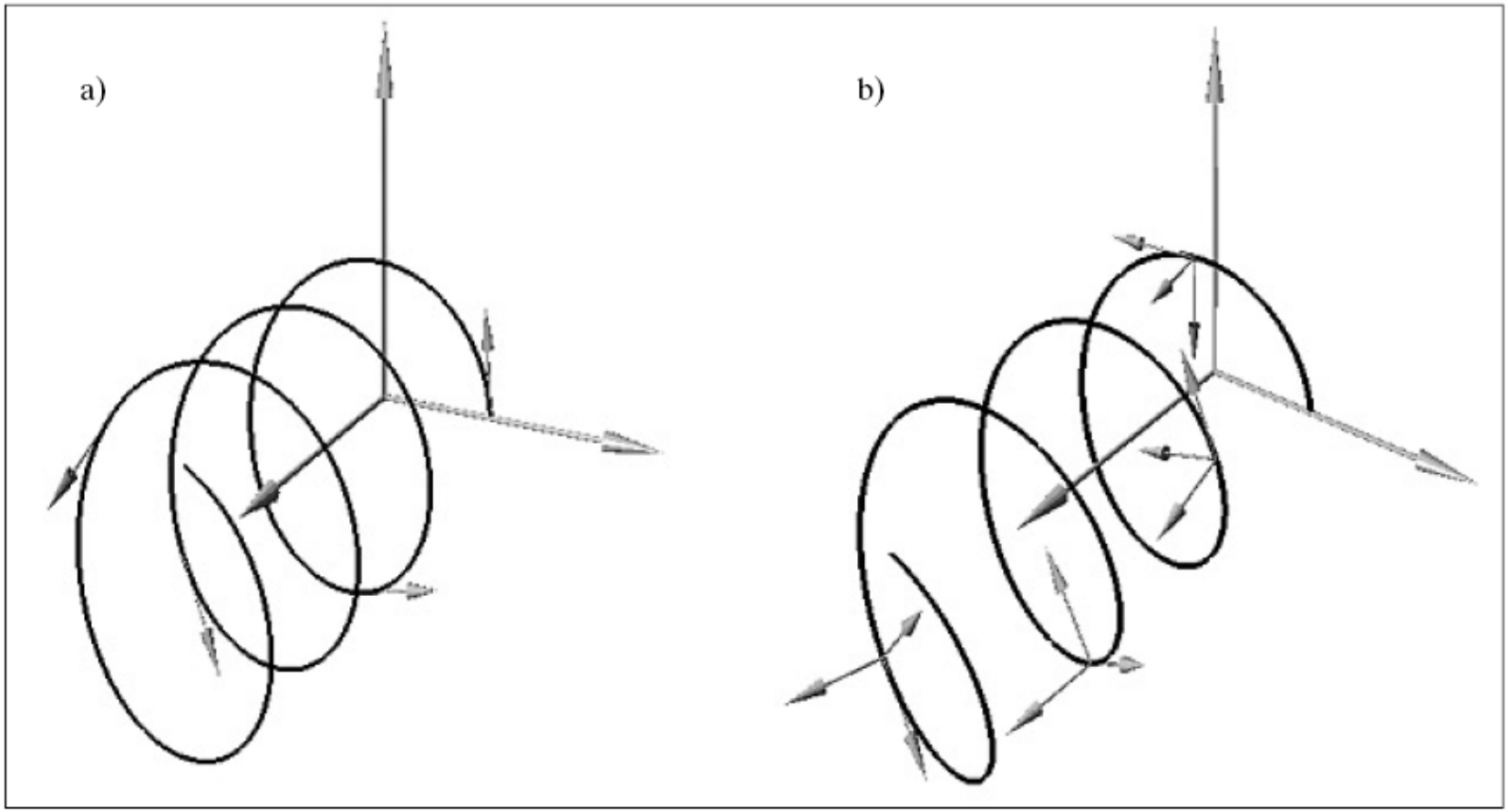
- Se busca obtener la transformación que coloca el sistema de referencia local (del perfil) en un punto de C(t). Además queremos orientar el sistema de referencia que resulta para que su plano XY sea perpendicular a C(t).
- La primera derivada: C'(t) (vector tangente a C(t)).
- La segunda derivada: C''(t) (vector curvatura de C(t)).
- El vector T(t) es la normalización de C'(t).
- El vector binormal B(t) es la normalización de C'(t)×C''(t).
- El vector N(t) es B(t)×T(t). Ya está normalizado.
- La transformación es la matriz 4×4 M(t)=(**N(t)**, **B(t)**, **T(t)**, C(t)), donde las tres primeras columnas son vectores y la última, punto.
- Ejemplo sobre la hélice:

  C(t)=(cos(t), sin(t), b*t)

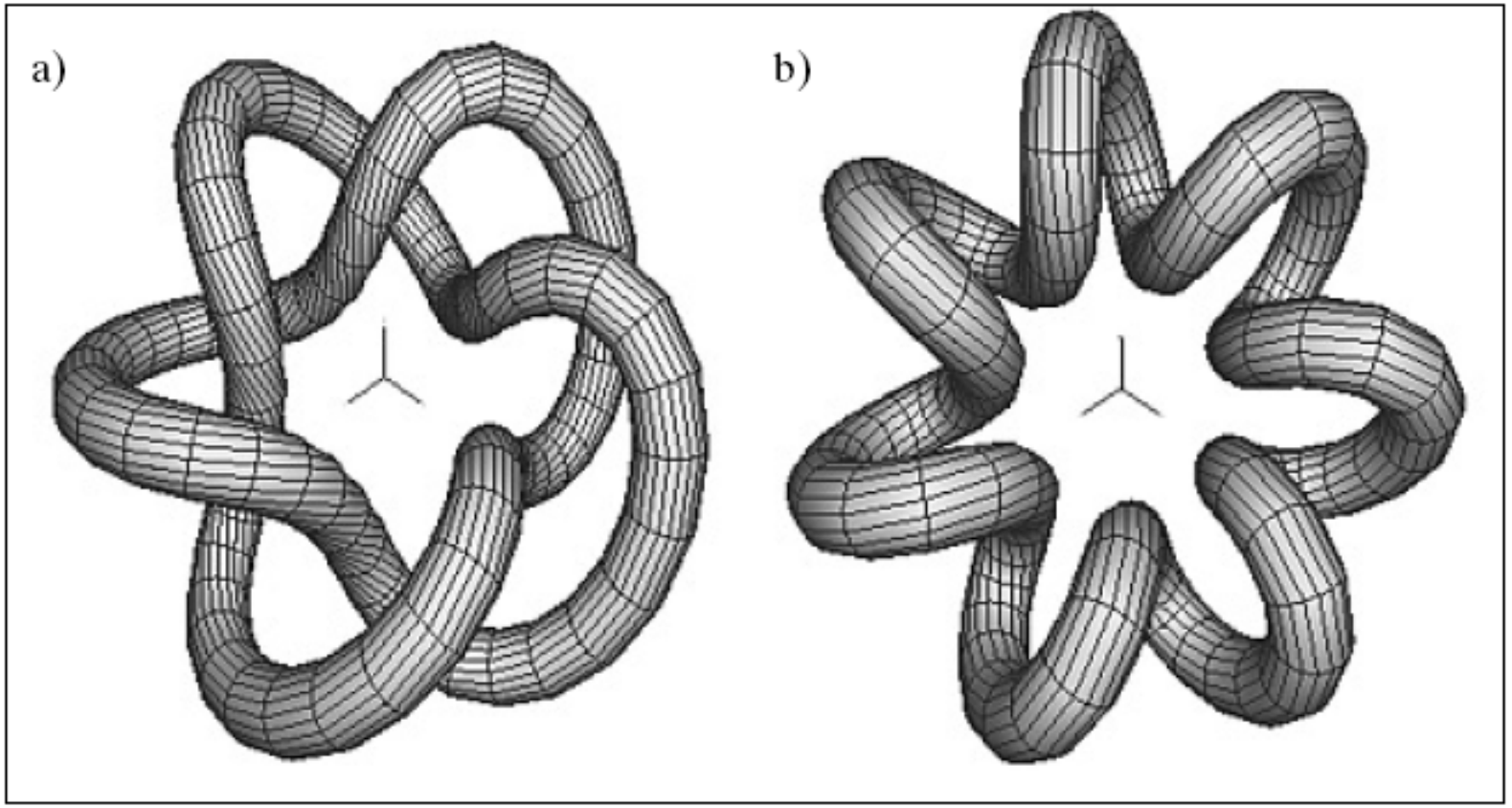
  b constante



Marco de Frenet



Marco de Frenet



- Ecuación de un plano:
  - Forma paramétrica: $P(u, v) = C+u*\mathbf{a}+v*\mathbf{b}$

  donde C es un punto del plano, y **a**, **b** son vectores del plano l.i.
  - Forma implícita: $F(x, y, z): a*x + b*y + c*z + d= 0$

  El vector **n** = (a, b, c) es normal al plano.
- Cuando la superficie está dada en forma paramétrica, el producto vectorial de las derivadas parciales con respecto a los parámetros $(\partial P/\partial u)\times(\partial P/\partial v)$ es un vector normal a la superficie.
- Cuando la superficie está dada en forma implícita, el gradiente de la ecuación $\nabla F=(\partial F/\partial x, \partial F/\partial y, \partial F/\partial z)$ es un vector normal a la superficie.
- Ejemplo: La esfera de radio R, centrada en el origen
  - Forma paramétrica: $P(\phi, \theta) =(R*\cos(\phi)*\cos(\theta), R*\text{sen}(\phi), R*\cos(\phi)*\text{sen}(\theta))$

  donde $-\pi/2\leq\phi<\pi/2$ (latitud), $0\leq\theta<2*\pi$ (longitud)
  - Forma implícita: $x*x + y*y + z*z = R*R$

Construcción de la malla de una superficie

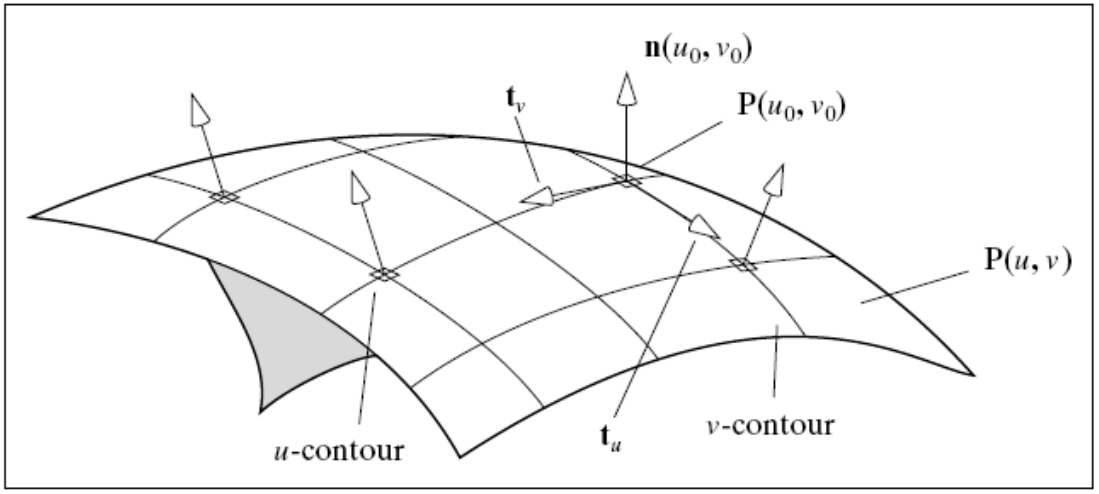
- Las ecuaciones paramétricas de la superficie están dadas por:

$$P(u, v) = (X(u, v), Y(u, v), Z(u, v))$$

- El vector normal a un punto P(u, v) de la superficie está dado por:

$$\mathbf{n}(u, v) = (\mathbf{nX}(u, v), \mathbf{nY}(u, v), \mathbf{nZ}(u, v))$$

- El método construye una malla teniendo en cuenta que:
  - Los rangos de los parámetros son $u \in [uMin, uMax]$, $v \in [vMin, vMax]$
  - La superficie se divide en nU partes, sobre u, y en nV partes, sobre v



Construcción de la malla de una superficie

```
void hazMallaSuperficie() {
   //Dimensiones de la superficie
   GLdouble uMin = …; uMax = …; vMin = …; vMax = …;
   //Número de divisiones
   int nU = …; nV = …;
   //Incrementos
   GLdouble incU = (uMax-uMin)/(nU-1);
   GLdouble incV = (vMax-vMin)/(nV-1);

   //Tamaños de los arrays
   numVertices = nU*nV;
   numNormales = numVertices;
   numCaras = (nU-1)*(nV-1);
   //Creación de los arrays
   vertice = new PV3D*[numVertices];
   normal = new PV3D*[numNormales];
   cara = new Cara*[numCaras];
```

Construcción de la malla de una superficie

```
for (int i=0, u=uMin; i<nU; i++, u+=incU)
   for (int j=0, v=vMin; j<nV; j++, v+=incV) {
        int indice = i*nV+j;
        //Coordenadas del vértice y de la normal, el mismo índice!!!
        vertice[indice] =
                new PV3D(X(u,v), Y(u,v), Z(u,v), 1);
        normal[indice] =
                new PV3D(nX(u,v), nY(u,v), nZ(u,v), 0);
        normal[indice]->normalizar();
        //Construcción de caras cuadrangulares
        if (i>0 && j>0) {
          int indiceCara = (i-1)*(nV-1)+(j-1);
          VerticeNormal** vn = new VerticeNormal*[4];
          vn[0]=new VerticeNormal(indice,indice);
          vn[1]=new VerticeNormal(indice-nV,indice-nV);
          vn[2]=new VerticeNormal(indice-nV-1,indice-nV-1);
          vn[3]=new VerticeNormal(indice-1,indice-1);
          cara[indiceCara] = new Cara(4, vn);
        }//if
    }//for
}//método
```