Informática Gráfica 3D

Antonio Gavilanes Franco
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

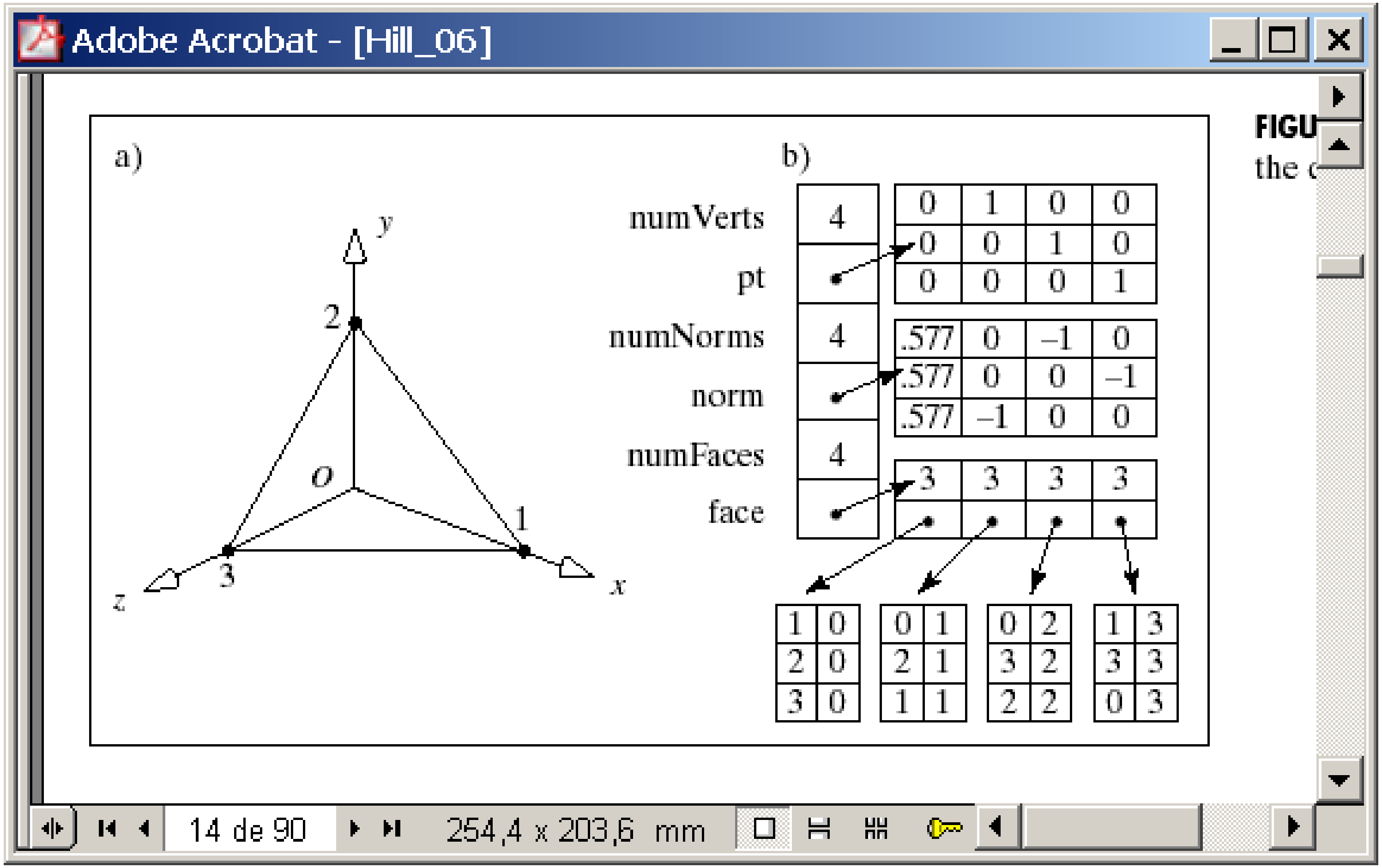
Programa

- Formas de representación de superficies
- Transformaciones afines
- Cámara y proyecciones
- Modelo jerárquico
- Coloreado, iluminación y texturas
- Bibliografía
  - Francis S. Hill Jr., Computer Graphics usin OpenGL
  - Sumanta Guha, Computer Graphics through OpenGL

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


Adobe Acrobat - [Hill_06]
a)
y
2
O
1
3
z
x
b)
numVerts
pt
numNorms
norm
numFaces
face
4
4
4
0 1 0 0
0 0 1 0
0 0 0 1
.577 0 –1 0
.577 0 0 –1
.577 –1 0 0
3 3 3 3
1 0
2 0
3 0
0 1
2 1
1 1
0 2
3 2
2 2
1 3
3 3
0 3
FIGU
the c
14 de 90
254,4 x 203,6 mm

- 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]->getIndiceNormalK(j);
          int iV=cara[i]->getIndiceVerticeK(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();
    }
}
```

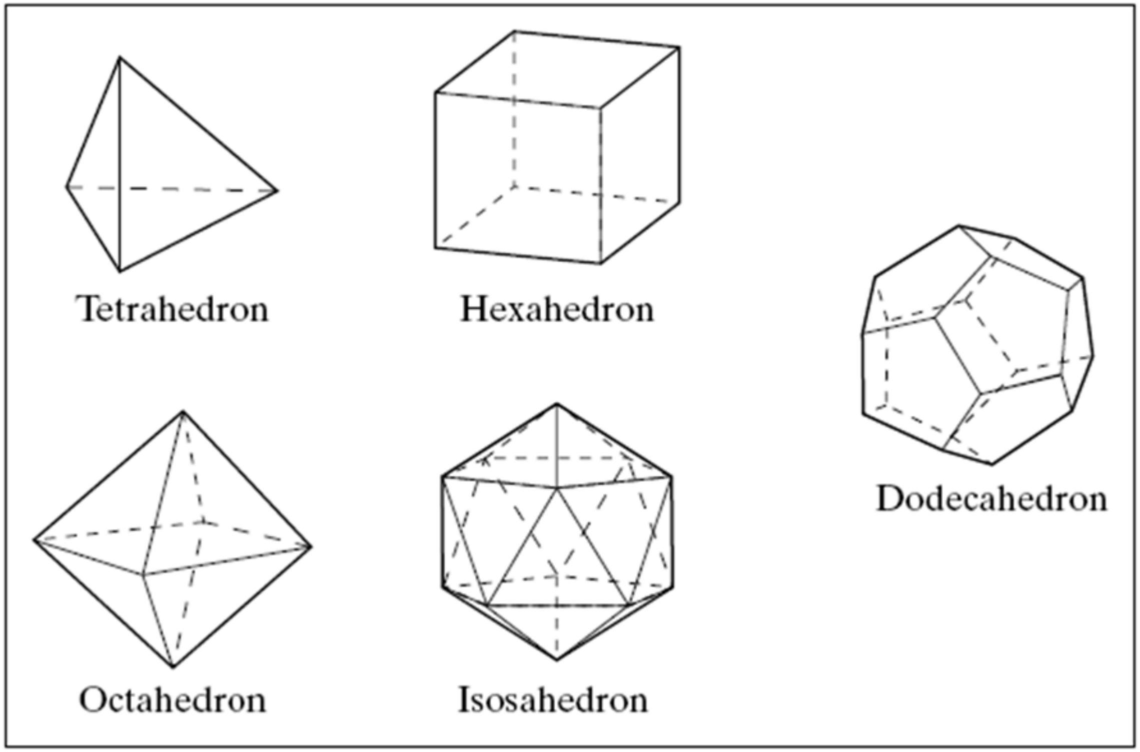
- Producto vectorial
- Formas de calcular el vector normal a una cara formada por los vértices `{v0, v1, …, vn}`
  - Usando el producto vectorial:

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

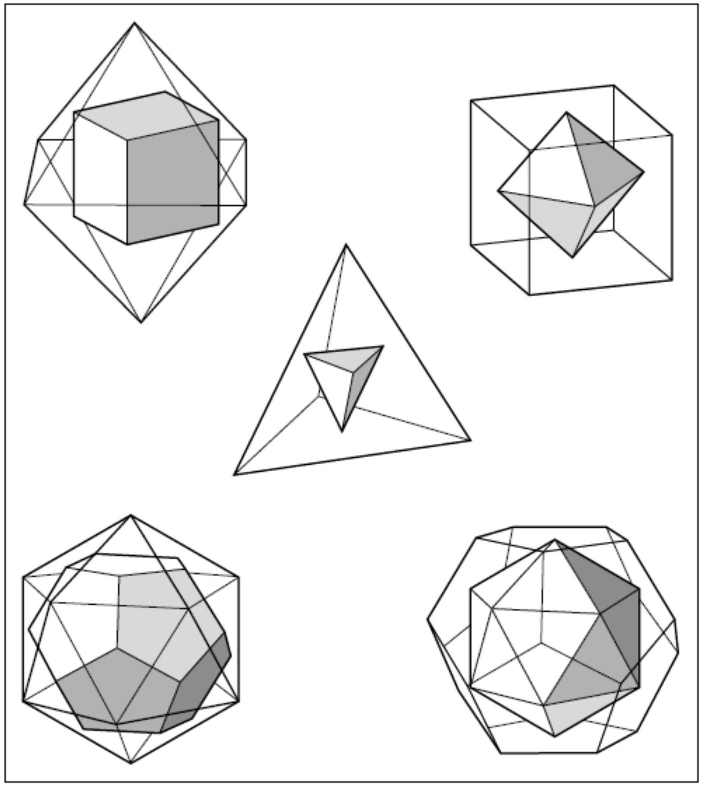
    Inconvenientes
  - Usando el método de Newell

```
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);
```

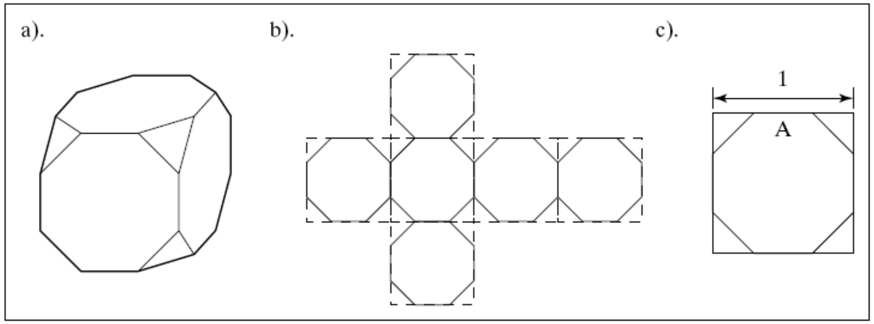
Sólidos platónicos



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

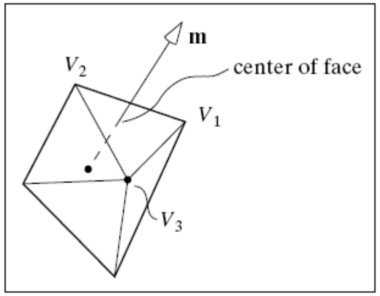


- Vértices obtenidos al truncar la arista CD

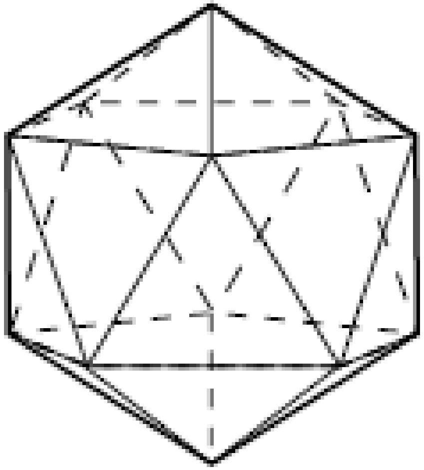
$$V=((1+A)/2)\ C + ((1-A)/2)\ D$$

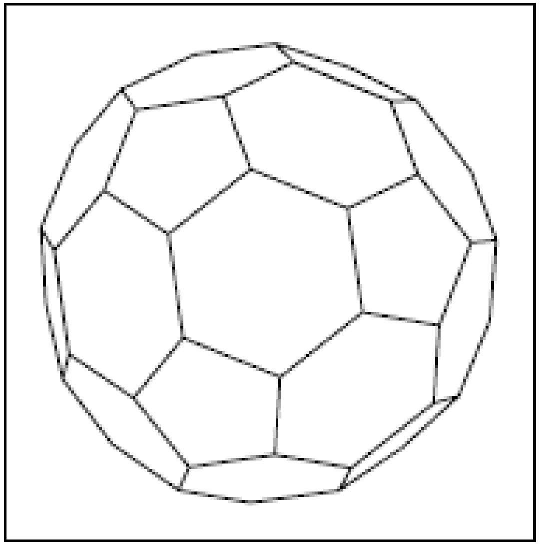
$$W=((1-A)/2)\ C + ((1+A)/2)\ D$$

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

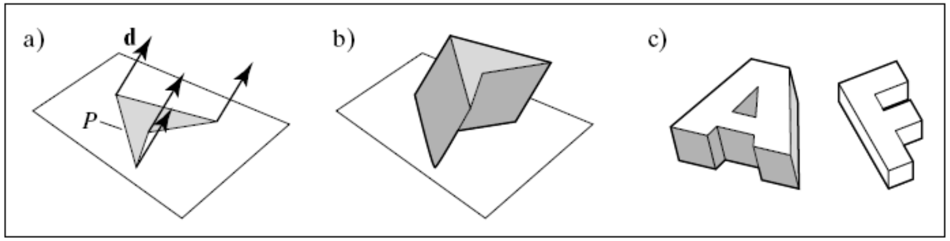


- Buckyball (fulereno), por Buckminster Fuller

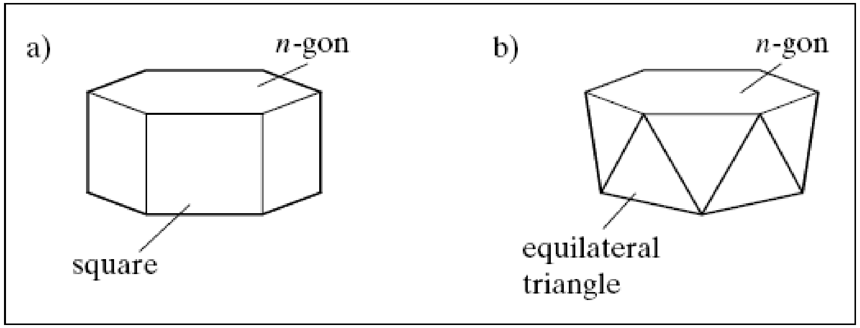




- Extrudir (tr): Dar forma a una masa metálica, plástica, etc., haciéndola salir por una abertura especialmente dispuesta.
- Un prisma es un poliedro obtenido mediante extrusión de un polígono a lo largo de una línea recta.
  - Caras cuadrangulares



- 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?)



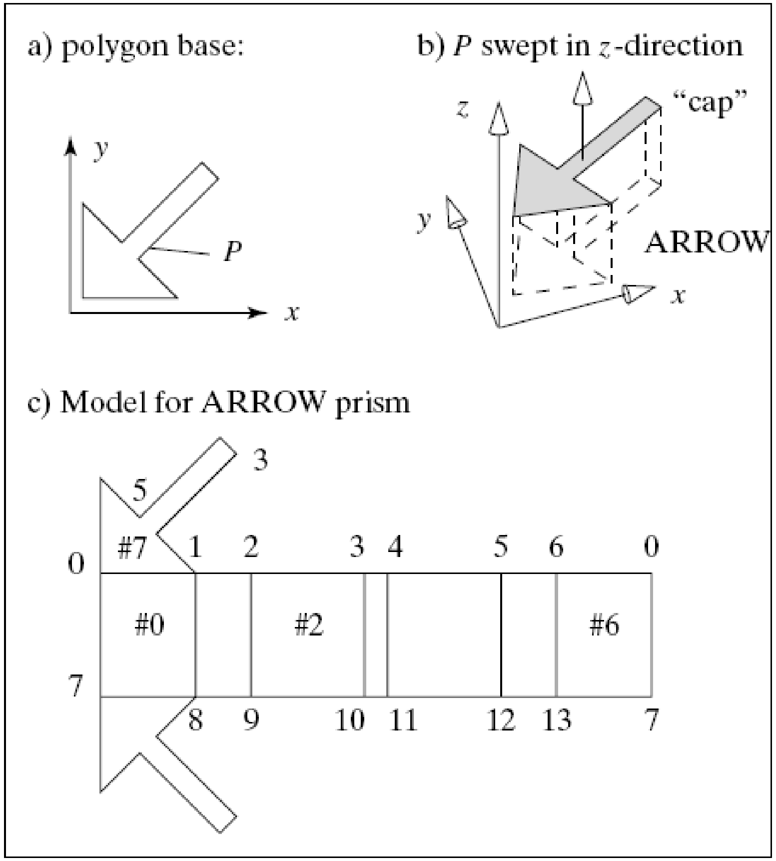
- Vértices:

  (xi, yi, 0), (xi, yi, H), 0≤i≤N-1

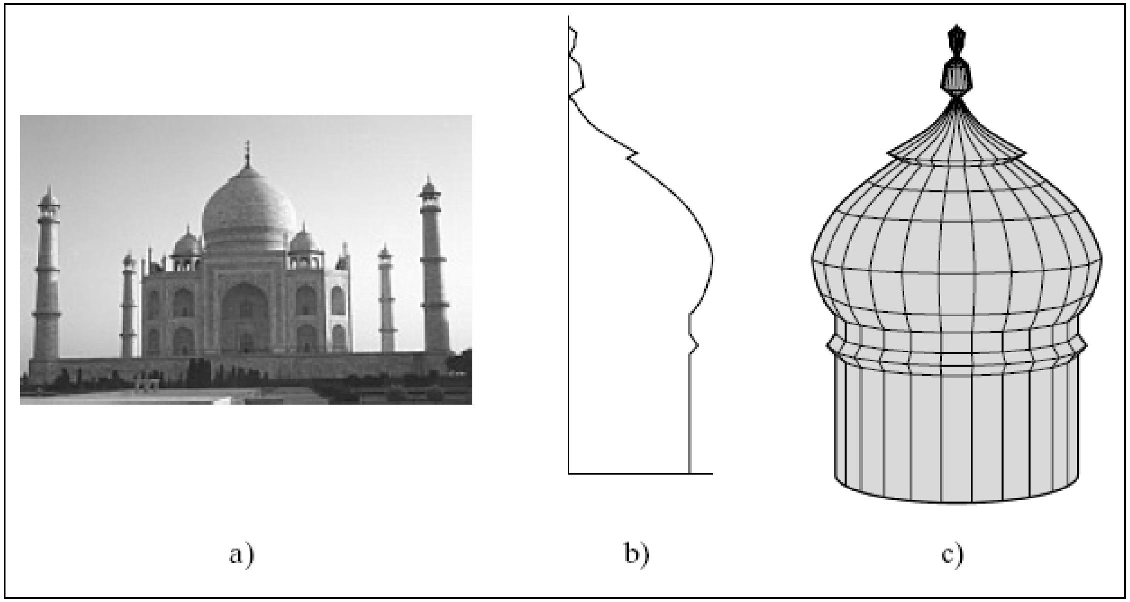
  donde N es el número de lados del polígono que origina el prisma y H es su altura

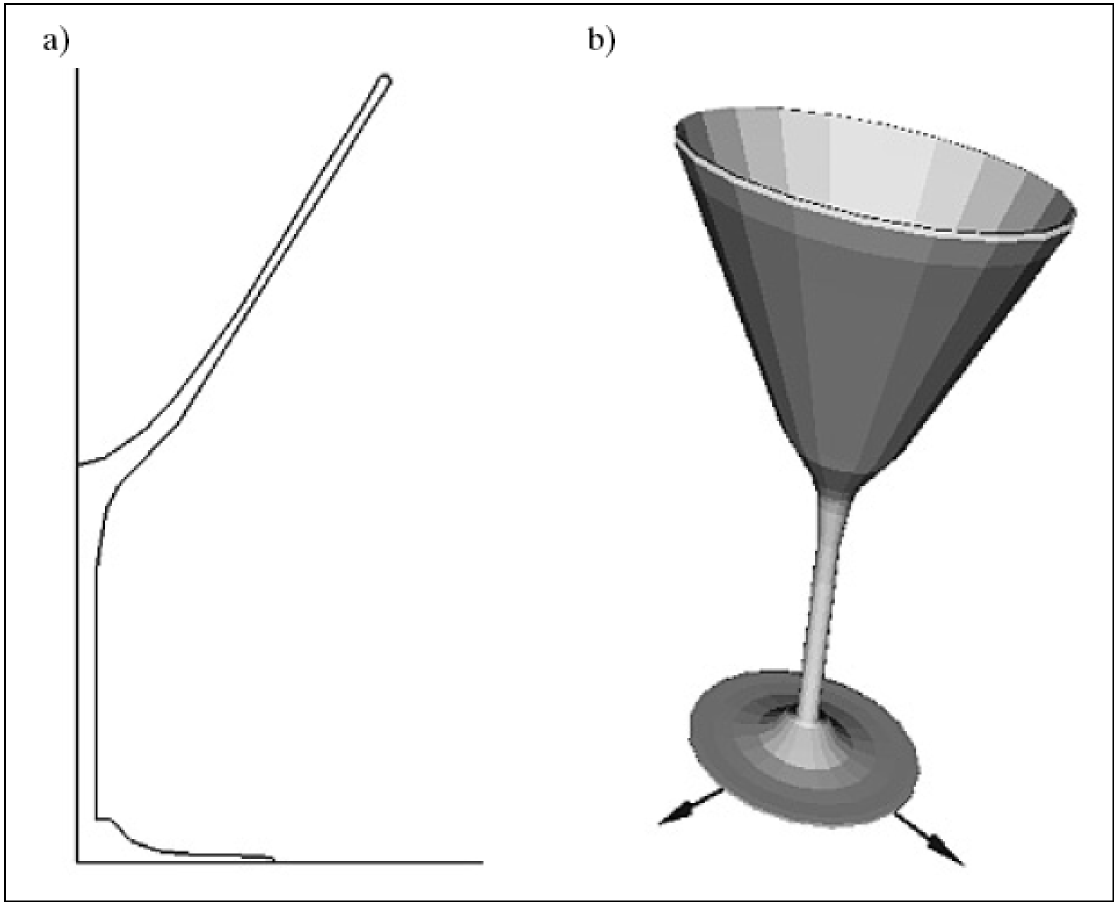
- Caras:
  - Laterales

    (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)

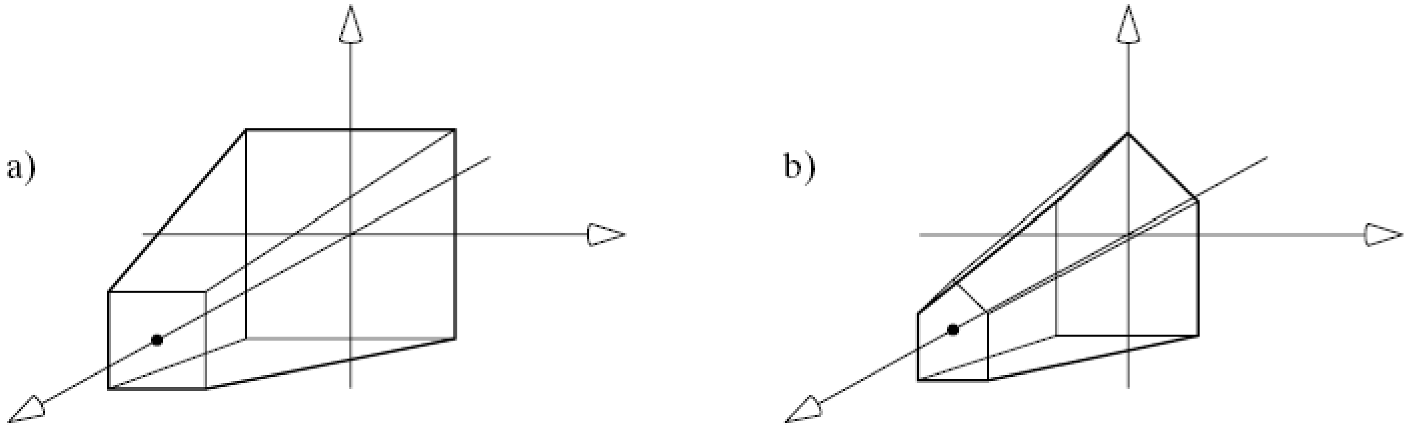


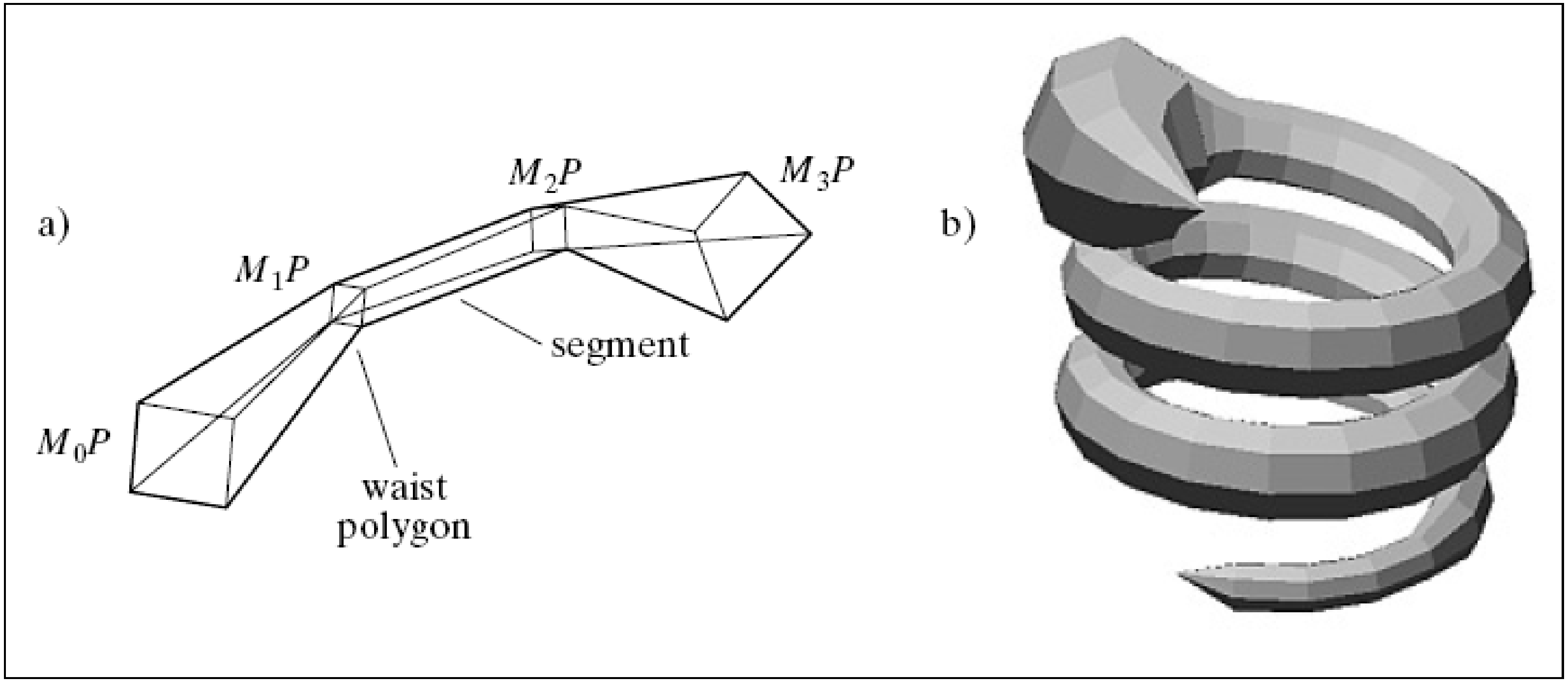
- Ingredientes para definir una malla por revolución
  - Un perfil formado por los puntos $\{P_0, \ldots, P_{n-1}\}$ sobre el plano XY
  - Vértices: los obtenidos aplicando las sucesivas rotaciones R(360/n, (0, 1, 0)) a los puntos del perfil, donde n es el número de veces que se van a tomar muestras durante una revolución
  - Caras: las cuadrangulares obtenidas uniendo dos puntos de un perfil con los dos correspondientes del perfil siguiente
  - Normales: Por Newell




a)
b)

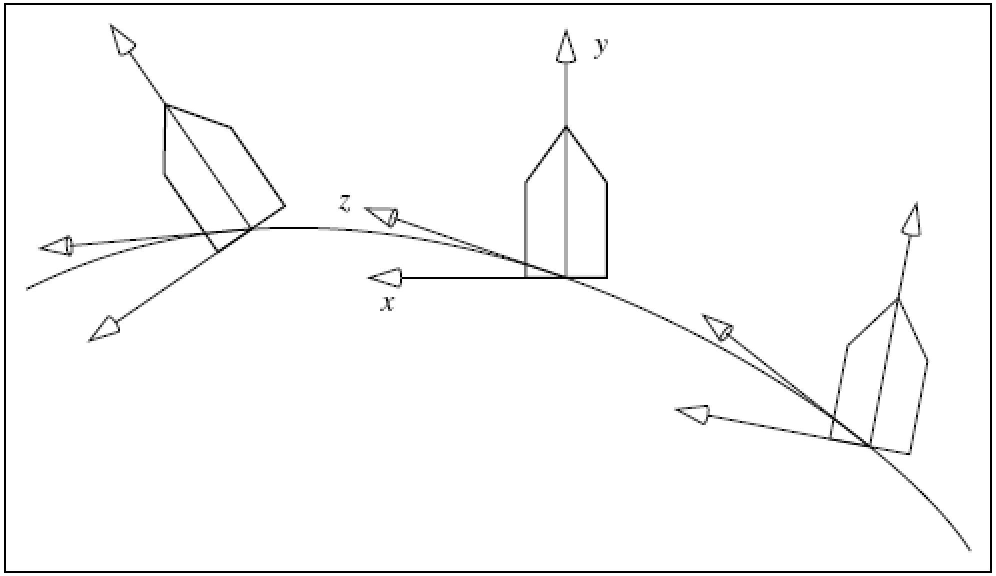
- Ingredientes para definir una malla por extrusión
  - Un polígono formado por los puntos $\{P_0, \ldots, P_{n-1}\}$
  - Una serie de transformaciones (posiblemente iguales) $\{M_1, \ldots, M_k\}$
  - Vértices: los obtenidos aplicando sucesivamente las transformaciones a los puntos del último polígono obtenido $M_j(M_{j-1}(\ldots(M_1(p_i))\ldots))$, $0 \le i \le n-1$, $2 \le j \le k$
  - Caras: las cuadrangulares obtenidas como se vio con las caras laterales en los prismas
  - Normales: Por Newell




a)
M_0P
M_1P
M_2P
M_3P
segment
waist polygon
b)

Marco de Frenet

- Se parte de **un perfil dado** y se producen sucesivos perfiles, por aplicación de una transformación del sistema de coordenadas global en un sistema de coordenadas local cuyo origen se encuentra en los puntos de **una curva dada**.
- Los puntos de los sucesivos perfiles se unen, igual que se hace en una malla por extrusión.

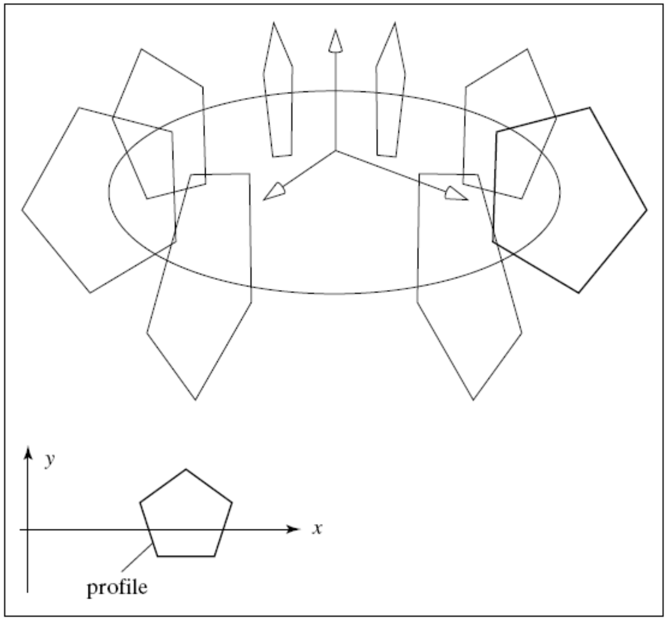


Toro usando el marco de Frenet

- Perfil dado: 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(2*PI-i*inc),
                      r*sin(2*PI-i*inc),0,1);
```

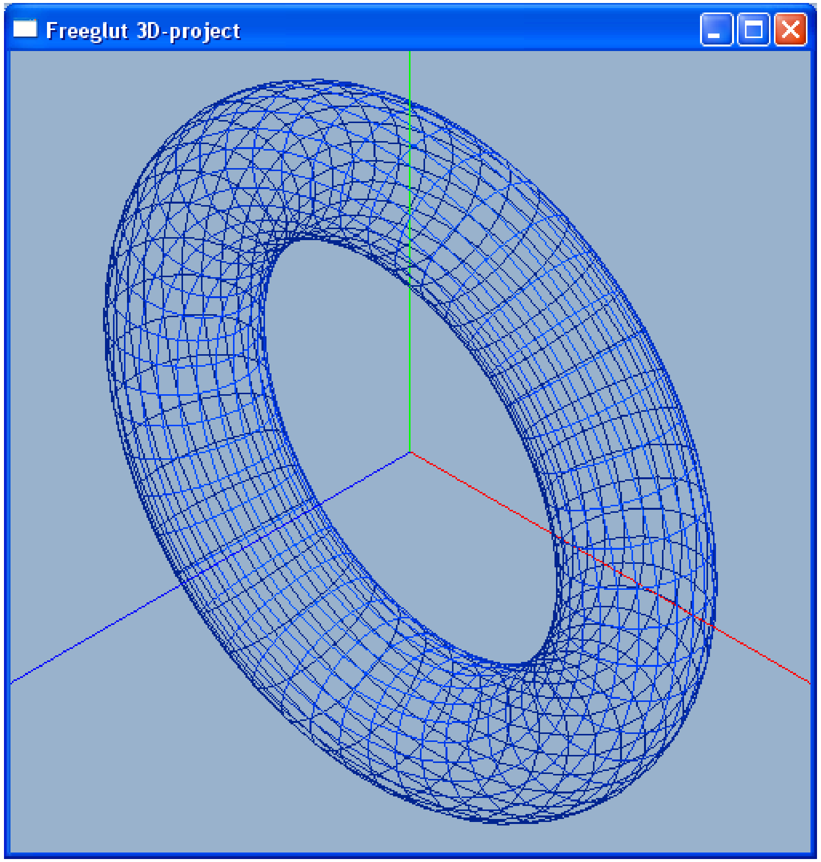
- Curva dada: C(t)=(R*cos(t), 0, R*sin(t)) (ecuaciones paramétricas de la circunferencia centrada en el origen, sobre el plano XZ, de radio R).

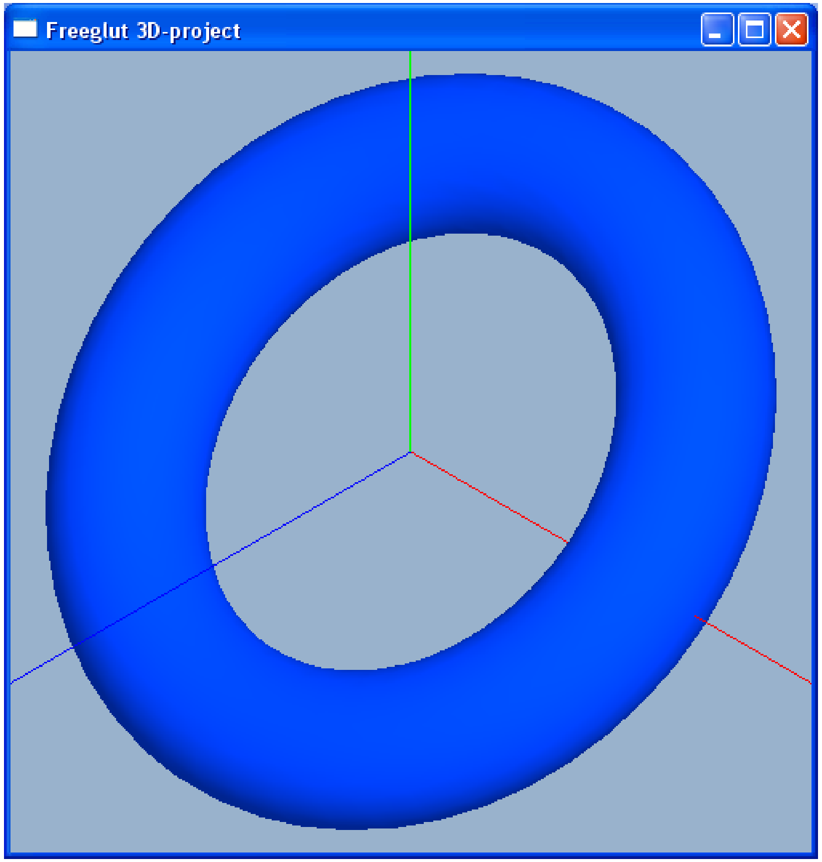


Toro usando las glut

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

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

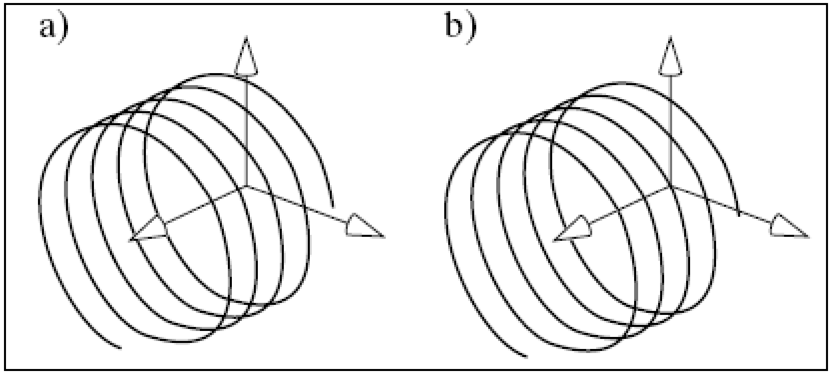


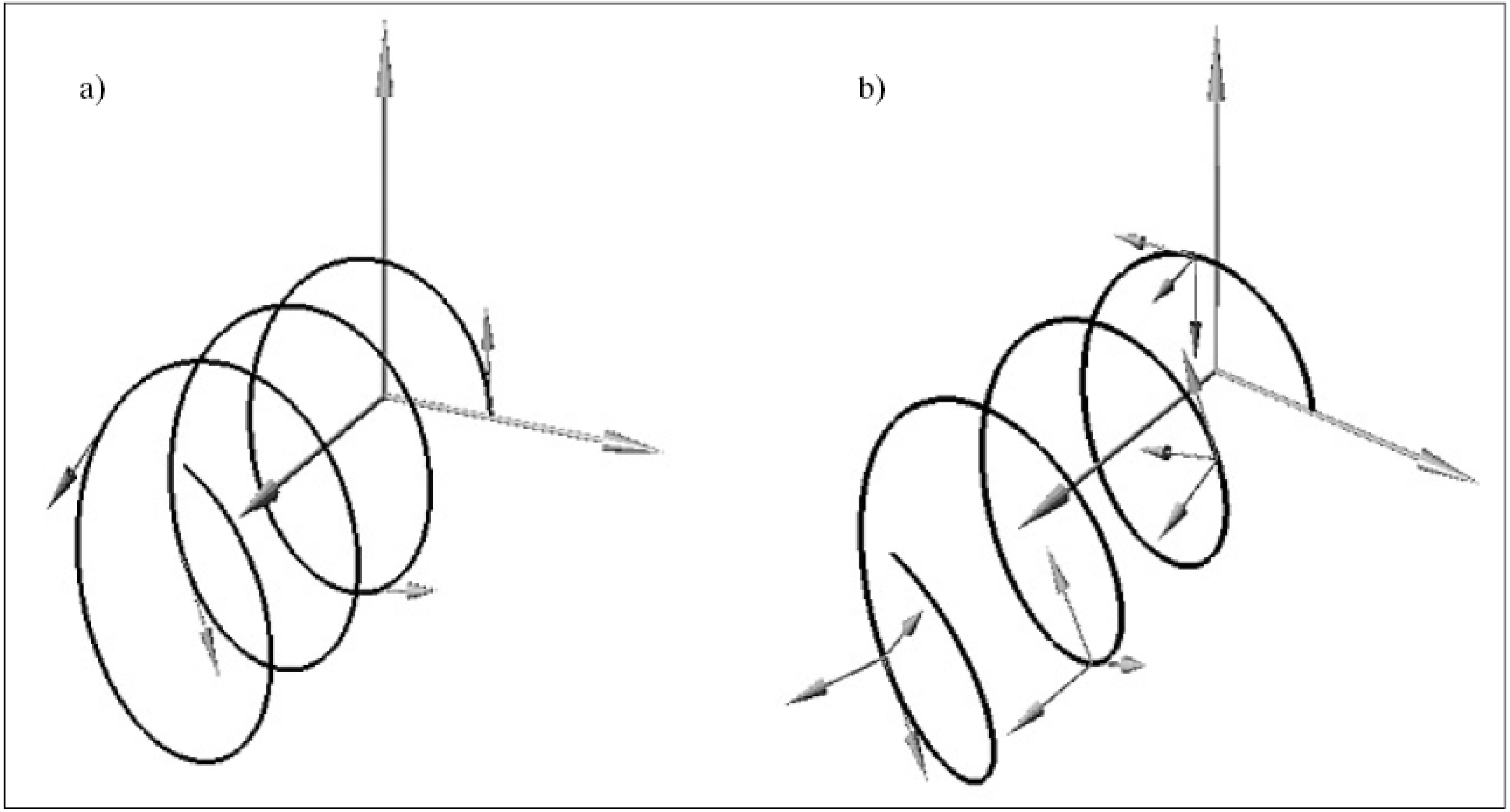


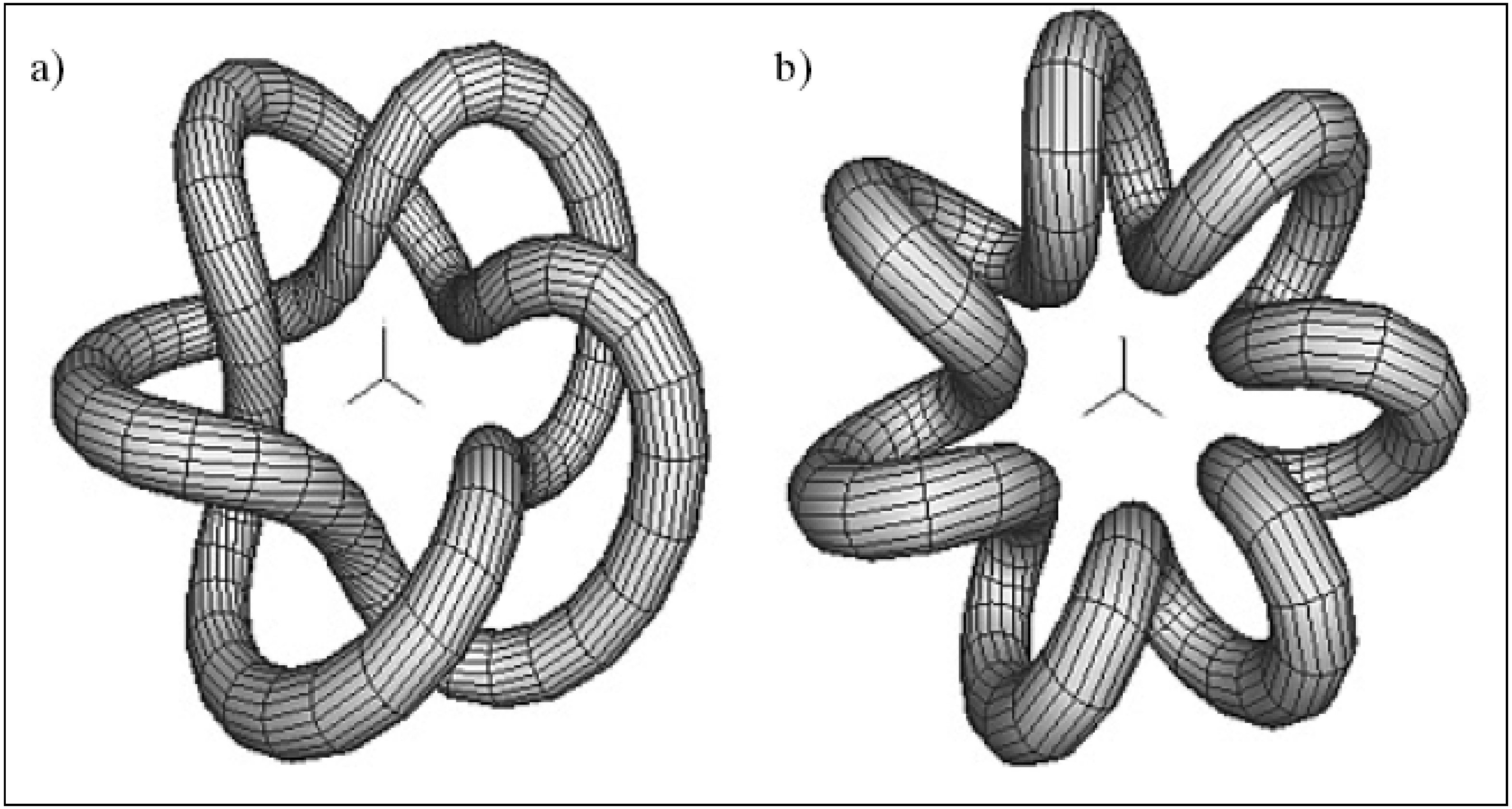
- Se busca obtener la transformación del sistema de coordenadas global en un sistema de coordenadas local cuyo origen se encuentra en los puntos de C(t).
- La primera derivada: C'(t) (vector tangente a C(t)).
- La segunda derivada: C''(t) (vector curvatura de C(t)).
- El vector T(t) es la normalización de C'(t).
- El vector binormal B(t) es la normalización de $C'(t) \times C''(t)$.
- El vector N(t) es $B(t) \times T(t)$. 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




a)
b)


a)
b)

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

  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: $P(\phi, \theta) =(R*\cos(\phi)*\cos(\theta), R*\mathrm{sen}(\phi), R*\cos(\phi)*\mathrm{sen}(\theta))$

  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

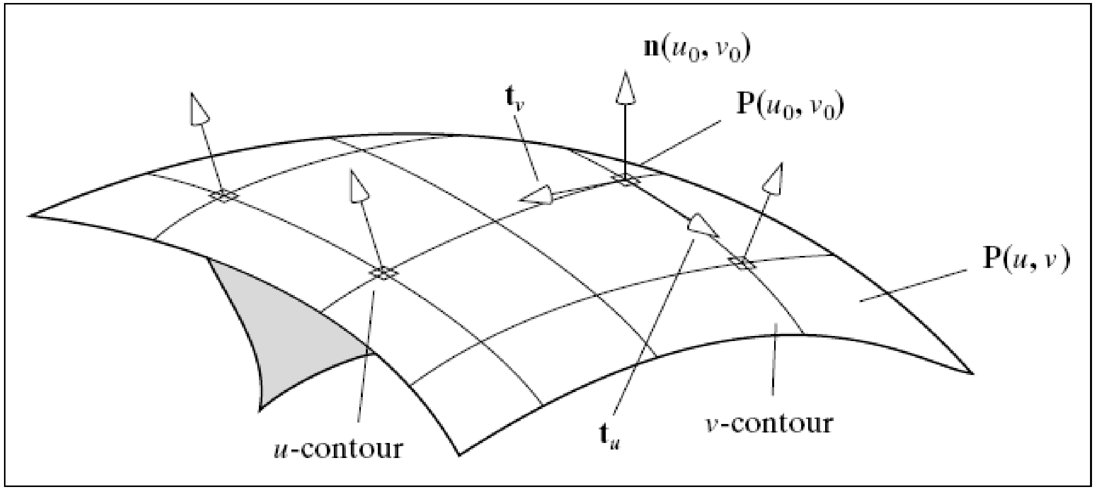
- Las ecuaciones paramétricas de la superficie están dadas por:

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

- 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∈[uMin, uMax], v∈[vMin, vMax]
  - La superficie se divide en nU partes, sobre u, y en nV partes, sobre v



```
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];
```

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