POLIEDROS REGULARES UNA VISION POR COMPUTADOR

Diego Luis Hoyos
Profesor Departamento de Matematicas
Universidad del Valle

RESUMEN

En éste artículo se desarrollan los métodos para lograr el despliegue gráfico de los cinco poliedros regulares, utilizando programación en Turbo Pascal. Inicialmente se da un resumen de las rutinas gráficas utilizadas, así como las matemáticas de los diferentes métodos de proyección. Finalmente, se desarrollan las rutinas para evitar las lineas ocultas.

INTRODUCCION

Un poliedro en tres dimensiones es un cuerpo sólido limitado por porciones de planos llamados caras; es "regular", según Euclides, si todas sus caras son polígonos congruentes con lados y ángulos iguales. Platón ya conocía la existencia de únicamente cinco poliedros regulares. Sorprendido por éste hecho, les atribuyó propiedades mágicas, construyendo un sistema planetario basado en ellos, hecho que permaneció como verdad hasta los trabajos de Kepler.

Euclides muestra, en el libro XIII de sus "Elementos", la forma de inscribir éstos poliedros en una esfera encontrando relaciones entre el díametro de la esfera y sus aristas, así como las relaciones entre las longitudes de las aristas de los diferentes poliedros.

Utilizando el hecho de que cada cara tiene el mismo número de lados, que en cada vértice concurren el mismo número de caras y que satisfacen la relación de Euler vertices + caras - aristas = 2, se puede dar una demostración topológica de la existencia de únicamente cinco poliedros regulares en dimensión tres (6).

En el espacio cartesiano, las coordenadas de los vértices se pueden hallar utilizando las relaciones dadas por Euclides, o utilizando la propiedad de dualidad. Dos poliedros se dicen duales si los vértices de uno son los centros de las caras del otro. Así son duales el cubo y el octaedro, el dodecaedro y el icosaedro; el tetraedro es autodual. Por ejemplo, para un cubo con centro en el origen, de aristas paralela a los ejes coordenados y de longitud dos, sus 8 vértices tienen coordenadas $(\pm 1, \pm 1, \pm 1)$; el octaedro dual tiene como vértices las permutaciones de $(\pm 1, 0, 0)$. Los vértices del tetraedro se obtienen seleccionando vértices alternos del cubo. Los vértices del icosaedro son los doce vértices de tres rectángulos aureos que están en planos perpendiculares entre sí (1).

Las características de los cinco poliedros regulares se resumen en la siguiente tabla, donde r es la proporción áurea $(1 + \sqrt{5})/2$.

Nombre	V	A	C	Coordenas de los vértices
Tetraedro	4	6	4	$(1,1,1),(-1,-1,1),(-1,1,-1),(1,-1,-1)$
Cubo	8	12	6	$(\pm 1, \pm 1, \pm 1)$
Octaedro	6	12	8	$(\pm 1, 0, 0)$ permutados
Dodecaedro	20	30	12	Permutaciones pares de $(\pm 1/r, \pm r, 0)$
Icosaedro	12	30	20	$(0, \pm r, \pm 1),(\pm 1, 0, \pm r),(\pm r, \pm 1, 0)$

Tabla 1:

En éste artículo no probaremos los resultados citados, o resumidos en la tabla anterior, sino que los usaremos para desarrollar sencillos programas en Turbo Pascal que permitan el despliegue gráfico de dichos cuerpos geométricos.

Inicialmente veamos un breve resumen de las rutinas gráficas del Turbo Pascal que utilizaremos.

GRAFICAS EN TURBO PASCAL

La unidad GRAPH.TPU contiene las rutinas para el despliegue gráfico, dibujo de lineas y sus atributos, manejo de ventanas y de color, llenado de polígonos, texto gráfico etc.

El sistema de coordenadas coloca el origen en la esquina superior izquierda de la pantalla, y en la esquina inferior derecha, la media resolución de 640×200 de la targeta EGA en un AT compatible con IBM.

El procedimiento moveto (x, y : integer), mueve la posición corriente al punto de coordenadas x,y; de tal forma que si queremos colocar el origen de coordenadas en el centro de la pantalla, podemos declarar como constantes $x = 320, y = 100$.

El procedimiento lineto (x, y : integer), dibuja una linea recta desde la posición corriente al punto x, y. Por ejemplo, éstas dos rutinas dibujan una linea recta del punto (100, 50), al punto (300, 200):

moveto (100, 50);
lineto (300, 200);

Podemos, multiplicando por un factor (escala), convertir las coordenadas de pixel, a valores mas apropiados; por ejemplo, definiendo $ox := 320, oy := 100, t1 := 40, t2 := 20, x1 := 1, y1 := 2, x2 := 4, y2 := 3$, las 2 rutinas

moveto ($ox + t1 * x2, oy - t2 * y1$);
lineto ($ox + t1 * x2, oy - t2 * y2$);

dibujan una recta del punto (1,2), al punto (4,3). Con estos valores de $t1$ y $t2$, la esquina superior izquierda tiene coordenas (-8,5) y la esquina inferior derecha (8,-5). Para futuras referencias, se da un procedimiento que dibuja una linea del punto ($x1, y1$), al punto ($x2, y2$):

```
PROCEDURE union (x1,y1,x2,y2: real);
var
  xasp,yasp:word;
  t1,t2,a:real;
begin
  getaspectratio(xasp,yasp);
  t1:=a;
  t2:=round(xasp/yasp)*t1);
  moveto(round(ox + t1 * x1), round(oy - t2 * y1);
  lineto(round(ox + t1 * x2), round(oy - t2 * y2)
end;
```

(Como es conocido, el punto y coma al final del end es requerido por el compilador).

Getaspectratio es un procedimiento de Turbo Pascal que devuelve los valores apropiados para calcular la razón de aspecto del modo gráfico que se esté utilizando. Esto se usa para evitar distorsiones en el despliegue gráfico.

Para cambiar el estilo de linea, se tiene el procedure setlinestyle (1,p,w), y para los diferentes colores, el procedure setcolor (color).

Remito al lector al "manual de referencia" de Turbo Pascal versión 4 o 5 para un estudio completo de todas las rutinas gráficas.

GRAFICACION TRIDIMENSIONAL

Puesto que Turbo Pascal versión 5 no tiene rutinas tridimensionales, para representar un objeto tridimensional en el plano es necesario realizar una proyección. Hay dos métodos básicos de proyección: proyección paralela, en la cual todos los puntos del objeto se proyectan a lo largo de lineas paralela, y proyección perspectiva, en la cual los puntos se proyectan a lo largo de lineas que convergen hacia un punto denominado centro de proyección. La proyección paralela puede ser ortogonal, si los rayos proyectores son perpendiculares al plano de proyección, y se obtiene colocando una de las coordenadas igual a cero; y oblicua si los rayos forman con el plano, un ángulo distinto de cero. En nuestro caso, la proyección ortogonal no daría una visión realista del objeto. Por lo tanto solamente estudiaremos las proyecciones oblicua y en perspectiva. En la pantalla del computador con el origen en el centro, el eje y positivo es el eje horizontal hacia la derecha y el eje z positivo es el eje vertical hacia arriba; el eje x positivo es perpendicular al ojo del, usuario hacia afuera de la pantalla.

Proyeccion Oblicua

Sea (x, y, z) un punto en el espacio coordenado, y sea $(p1, p2)$ su proyección oblicua en el plano yz.

Como se observa en la Figura 1, a es el ángulo entre la linea de proyección y la linea que une $(p1, p2)$ con el punto (y, z) (coordenadas de la proyección ortogonal). Si l es la longitud de ésta linea, y b es el ángulo que forma con el eje y, entonces se tienen las relaciones:

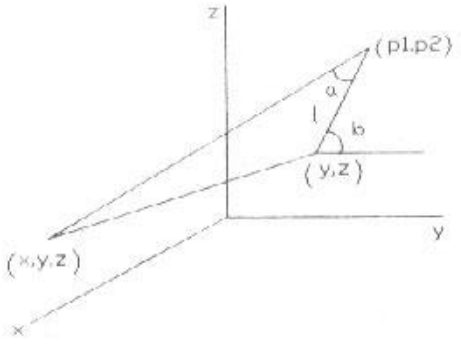

Figura 1

$$\begin{aligned} p1 &= y + l * \cos(b) \\ p2 &= z + l * \sin(b) \end{aligned}$$

Por otro lado,

$$1 = x/\tan(a);$$

de aquí que

$$\begin{aligned} p1 &= y + m1 * x \\ p2 &= z + m2 * x \end{aligned}$$

donde

$$\begin{aligned} m1 &= \cos(b)/\tan(a) \\ m2 &= \sin(b)/\tan(a) \end{aligned}$$

Generalmente se toman a y b de tal manera que $m1 = m2 = -0.5$ (b igual a 45 grados y a aproximadamente -55 grados), pero se pueden ensayar otros valores.

Proyección Perspectiva

Tomando el centro de proyección en el eje x positivo a una distancia d del origen , la ecuación vectorial del rayo proyector es

$$(x', y', z') = (x, y, z) + t(x - d, y, z) \text{ para } 0 \leq t \leq 1 \ .$$

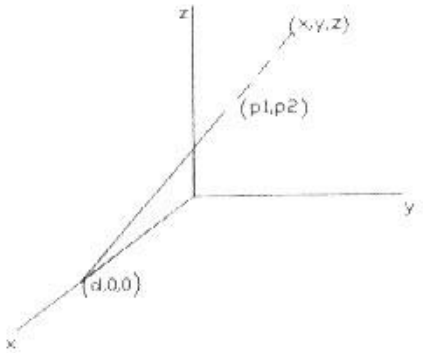

Figura 2

y sus ecuaciones paramétricas

$$\begin{aligned} x' &= x + t(x-d) \\ y' &= y + ty \\ z' &= z + tz \end{aligned}$$

Puesto que proyectamos al plano yz, para hallar $p1$ y $p2$ hacemos $x = 0$ y despejamos t.

$$t = x/(d-x)\ ;$$

al reemplazar éste valor de t en y' y z', obtenemos

$$\begin{aligned} p1 &= yd/(d-x) \\ p2 &= zd/(d-x)\ . \end{aligned}$$

PROGRAMACION EN PASCAL

El procedimiento que realiza la proyección es el mismo para todos los poliedros; solamente deben cambiarse las coordenadas de los vértices para cada poliedro en particular. Estas coordenadas se cargan en una matriz (array) bidimensional. Por ejemplo, para el cubo podemos escribir

```
type
  cubo = array[1..8,1..3] of real;
const
  c:cubo=(((1),(1),(1)),((-1),(1),(1)),((-1),(-1),(1)),
          ((1),(-1),(1)),((1),(1),(-1)),((-1),(1),(-1)),
          ((-1),(-1),(-1)),((1),(-1),(-1)));
```

Para saber cuales vértices son los que se unen, es conveniente numerarlos, por ejemplo, como se muestra en la figura 3.

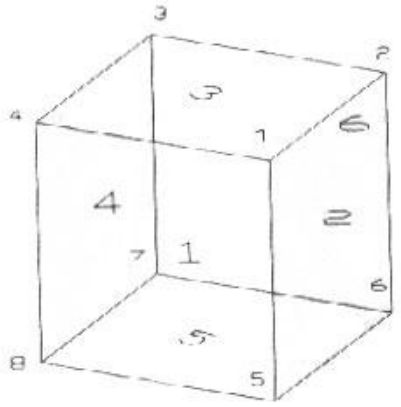

Figura 3

Estas marcas se cargan en arreglos unidimensionales de la siguiente forma:

```
type
  comienzo=array[1..12] of integer;
  final=array[1..12] 0f integer;
const
  com:comienzo=(1,1,1,2,2,3,3,4,5,5,6,7);
  fin:final=(5,4,2,6,3,7,4,8,8,6,7,8);
```

Para una proyección oblicua, el procedimiento proyección es entonces como sigue:

```
PROCEDURE proyeccion;
var
  arista:integer;
```

```
  s, f:integer;
  x1,y1,x2,y2:real;
begin
  for arista:=1 to 12 do
  begin
    s := com[arista];
    f := fin[arista];
    x1 := c[s,2] - 0.5 * c[s,1];
    y1 := c[s,3] - 0.5 * c[s,1];
    x2 := c[f,2] - 0.5 * c[f,1];
    y2 := c[f,3] - 0.5 * c[f,1];
    union(x1,y1,x2,y2)
  end
end;
```

Aquí esta el programa principal completo:

```
PROGRAM cubo3d;
uses
  graph;
type
  cubo= array[1..8,1..3] of real;
  comienzo=array[1..12] of integer;
  final= array[1..12] of integer;
const
  ox =320;
  oy=100;
  m1=-0.5;
  m2=-0.5;
  c:cubo=(((1),(1),(1)),((-1),(1),(1)),((-1),(-1),(1)),
          ((1),(-1),(1)),((1),(1)(-1)),((-1),(1),(-1)),
          ((-1),(-1),( 1)),((1),(-1),(-1)));
  com:comienzo=(1,1,1,2,2,3,3,4,5,5,6,7);
  fin:final =(5,4,2,6,3,7,4,8,8,6,7,8);
var
  gd,gm:integer;

PROCEDURE union(x1,y1,x2,y2:real);
var
  xasp,yasp:word;
  t1,t2:real;
begin
  getaspectratio(xasp,yasp);
  t1:=80;
  t2:=(xasp/yasp)*t1;
```

```
  moveto(round(ox+t1*x1),round(oy-t2*y1));
  lineto(round(ox+t1*x2),round(oy-t2*y2));
end;

PROCEDURE proyeccion;
var
  arista:integer;
  s:integer;
  f:integer;
  x1,y1,x2,y2:real;
begin
  for arista:=1 to 12 do
    begin
    s:=com[arista];
    f:=fin[arista];
    x1:=c[s,2]+m1*c[s,1];
    y1:=c[s,3]+m2*c[s,1];
    x2:=c[f,2]+m1*c[f,1];
    y2:=c[f,3]+m2*c[f,1];
    union(x1,y1,x2,y2)
  end
end;
begin
  gd:=detect;
  initgraph(gd,gm,'bgi');
  proyeccion;
  readln
end.
```

Produce por salida la gráfica de un cubo (ver fig 4).

Podemos rotar el objeto alrededor de cualquiera de los 3 ejes coordenados y con cualquier ángulo añadiendo al programa principal la siguiente rutina:

```
PROCEDURE rotacion(a1,a2:integer; theta:real):
const
  pi=3.14159;
  rad=pi/180;
var
  x,y:real;
  i:integer;
begin
  for i:=1 to 8 do
  begin
```

```
      x:=c[i,a1];
      y:=c[i,a2];
      c[i,a1]:=x*cos(theta*rad) - y*sin(theta*rad);
      c[i,a2]:=x*sin(theta*rad) + y*cos(theta*rad)
    end
end;
```

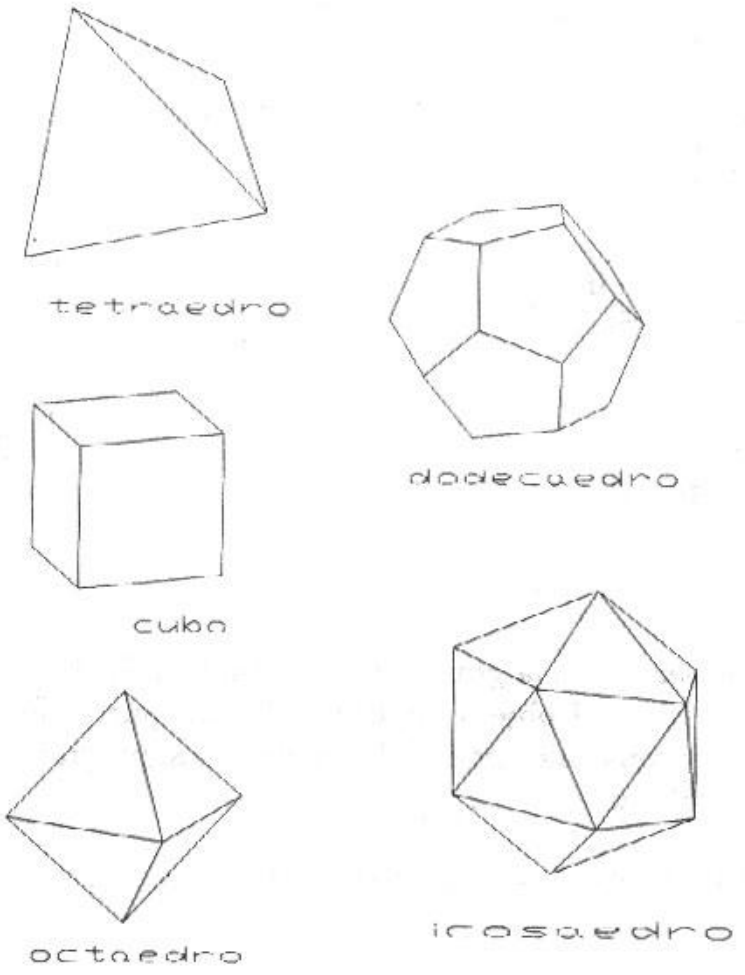

Figura 4

En la figura 4 se muestra la proyección oblicua de los cinco poliedros regulares con distintos ángulos de rotación. Para la proyección en perspectiva, solamente es necesario cambiar el procedimiento proyección por el siguiente:

```
PROCEDURE proyeccion (d:real);
var
  arista:integer;
  s:integer;
  f:integer;
  x1,y1,x2,y2,m:real;
begin
  for arista:=1 to 12 do
    begin
      s:=com[arista];
      f:=fin[arista];
      m:=d/(d-c[s,1]);
      x1:=c[s,2]*m;
      y1:=c[s,3]*m;
      m:=d/(d-c[f,1]);
      x2:=c[f,2]*m;
      y2:=c[f,3]*m;
      union(x1,y1,x2,y2)
    end
end;
```

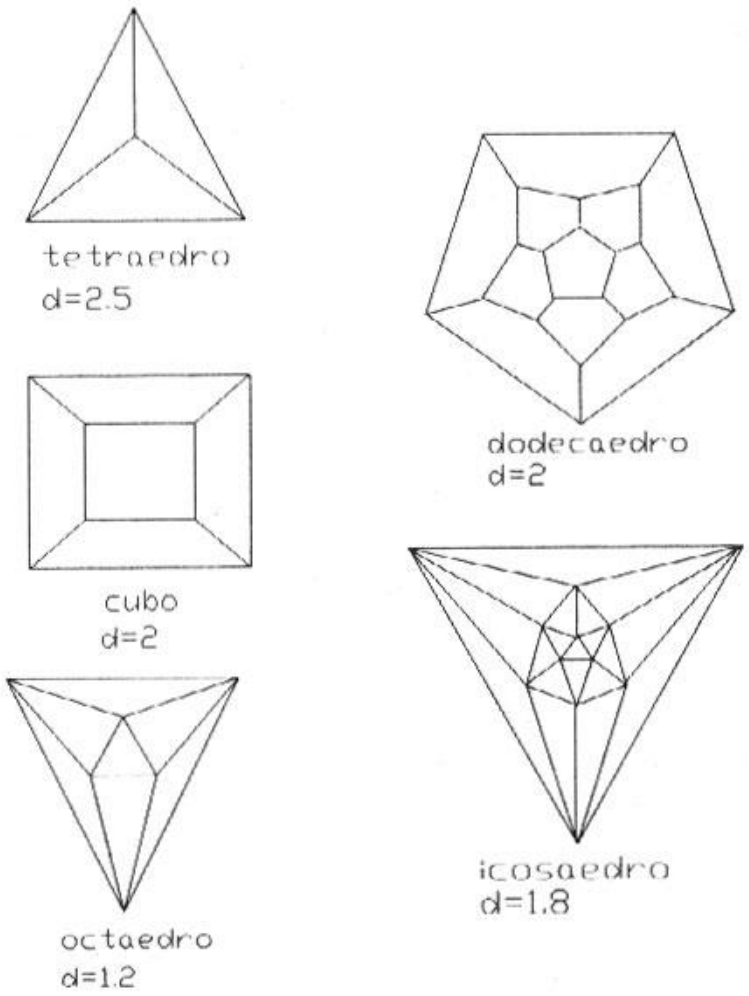

Figura 5

La figura 5 muestra la proyección perspectiva con el foco a poca distancia del centro de una de las caras de cada poliedro, como aseguraban correctamente D.Hilbert y S.Cohn-Vossen en [4], y la figura 6, con $d = 4$.

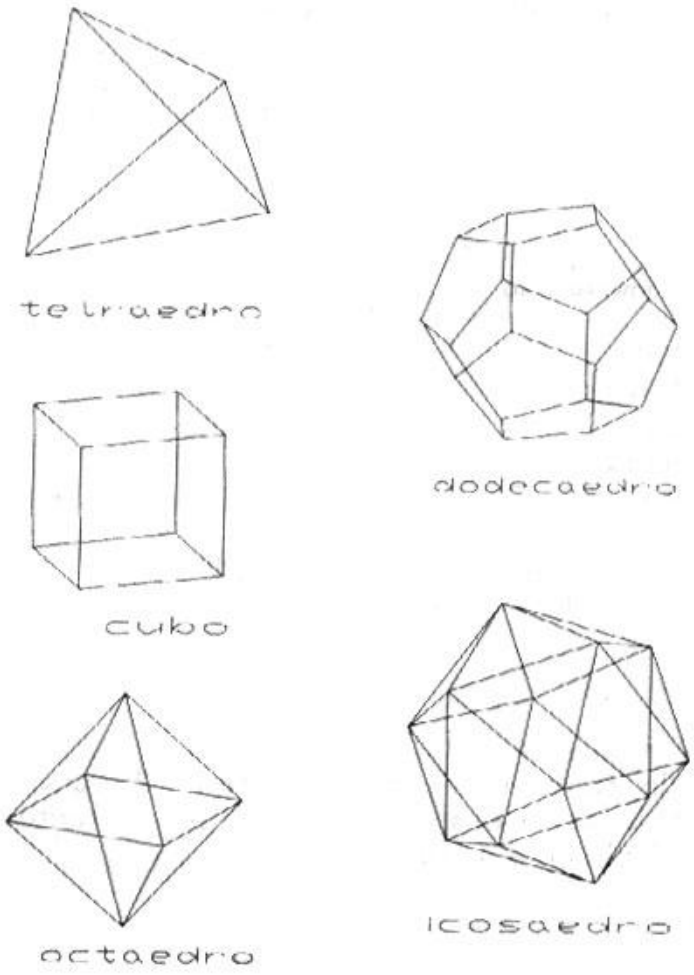

Figura 6

Hay dos cosas que se deben aclarar. Primero, ¿Cómo se encuentran los arreglos comienzo y final? Como el poliedro es convexo, dos vértices se unen si la distanciaentre ellos es la mínima. El programa que se muestra a continuación, calcula la distancia entre el vértice K y los restantes $n - k$:

```
PROGRAM distancia;
type
  poliedro=array[1..a,1..3] of real; {a es el numero de vertices}
const
  c:poliedro= { aqui se escriben las coordenadas de los vertices}
var
  i,k:integer;
  d:real;
  a1,a2,a3,b1,b2,b3:real;
  begin
    for i:=1 to a do
    begin
      for k:=a-i downto 1 do
      begin
        a1:=C[i,1];
        b1:=C[i+k,1];
        a2:=C[i,2];
        b2:=C[i+k,2];
        a3:=C[i,3];
        b3:=C[i+k,3];
        d:=sqrt(sqr(a1-b1)+sqr(a2-b2)+sqr(a3-b3));
        write(' la distancia de ',i , ' a ', i+k , ' es ', d);
      end
  end
end.
```

Por ejemplo, en el caso del icosaedro ($a = 12$), la salida muestra que la menor distancia es menor que 2.1; por lo tanto, agregando despues del cálculo de d, la instrucción

```
if d < 2.1 then
writeln(' El vertice ',i, ' se une con el vertice ', i+k);
```

da por salida los arrays buscados.

Segundo, el lector habrá notado que en la figura 6 las lineas ocultas no están dibujadas. ¿Cómo se logra ésto?

ARISTAS OCULTAS

Los algoritmos para no dibujar lineas ocultas son conocidos y en general de dificil aplicación [2] y [3]; pero para los poliedros convexos en perspectiva son relativamente fáciles (en [5] podrá encontrarse una versión en BASIC), porque como los ángulos interiores son menores

de 180 grados, desde cualquier punto de vista cada cara es completamente visible o está completamente oculta.

Puesto que una arista pertenece a dos caras, dicha arista está oculta, si y sólo si, las dos caras están ocultas. Para determinar si una cara está oculta, observamos el ángulo que forma el vector normal exterior de la cara, con el vector que va desde el pié de la normal al ojo del observador (centro de proyección). Puesto que en nuestro sistema de coordenadas, el ojo se encuentra en el eje x positivo, una cara está oculta si dicho ángulo es mayor de 90 grados.

Para hallar el vector normal de cada cara, escogemos 3 de sus vértices A,B,C, en el sentido de las manecillas del reloj, y efectuamos el producto vectorial entre los vectores BA y BC:

$$N = (B - A) \times (B - C)$$

Si 0 es el centro de proyección, entonces el signo del producto triple

$$sp = N.(B \quad 0)$$

da la solución : si $sp < 0$ entonces la cara está oculta.

Para el programa en Pascal, utilizamos un array bidimensional de 3 columnas y c filas (el número de caras); cada fila contiene los tres vértices elegidos de cada cara. Así en el caso del cubo tenemos: (ver figura 1).

```
type
vertices=array[1..6,1..3] of integer;
const
vert:vertices=(((1),(5),(8)),((1),(2),(6)),((1),(4),(3)),
               ((3),(4),(8)),((8),(5),(6)),((6),(2),(3)));
```

Debemos también numerar las caras del poliedro y declaramos un array de 2 columnas y a filas (el número de aristas): Cada columna contiene las dos caras de cada arista y deben estar en el mismo orden en que se escogió unir sus vértices; por ejemplo, en el cubo, el primer elemento del array que contiene las caras debe tener por primer elemento el par 1, 2 de acuerdo a como hemos enumerado las caras del cubo (ver figura 1); así tenemos la declaración

```
type
  caras = array[1..12,1..2] of integer;
const
```

```
cr:caras=(((1),(2)),((1),(3)),((2),(3)),((2),(6)),((3),(6)),
          ((6),(4)),((3),(4)),((4),(1)),((5),(1)),((5),(2)),
          ((5),(6)),((4),(5)));
```

Los tres procedimientos siguientes realizan el trabajo; el primero determina cuales caras son visibles y cuales ocultas; el segundo reemplaza el procedimiento proyección del programa principal y el tercero hace el gráfico.

```
PROCEDURE test(d:real);
var
  i,j:integer;
  s1,s2,s3:integer;
  sp:real;
  p,q,n,e:array[1..3] of real;
begin
  for i:=1 to 6 do
    begin
      flag[i]:=1;
      s1:=vert[i,1];
      s2:=vert[i,2];
      s3:=vert[i,3];
      for j:=1 to 3 do
        begin
          p[j]:=c[s1,j]-c[s2,j];
          q[j]:=c[s3,j]-c[s2,j];
        end;
        n[1]:=p[2]*q[3]-p[3]*q[2];
        n[2]:=p[3]*q[1]-p[1]*q[3];
        n[3]:=p[1]*q[2]-p[2]*q[1];
        e[1]:=d-c[s2,1];
        e[2]:=-c[s2,2];
        e[3]:=-c[s2,3];
        sp:=e[1]*n[1]+e[2]*n[2]+e[3]*n[3];
        if sp < 0 then flag[i]:=0;
    end;
end;
```

En éste primer procedimiento, flag es una variable global de tipo array de una fila y seis columnas, que se usa para marcar si la cara es visible o no; si flag[i] es uno, la cara es visible y si flag[i] es cero, la cara está oculta.

```
PROCEDURE proyeccion(d:real);
```

```
var
  x1,y1,x2,y2,m:real;
  s,f:integer;
begin
  s:=com[arista]; arista debe ser variable global de tipo integer
  f:=fin[arista];
  m:=d/(d-c[s,1]);
  x1:=c[s,2]*m;
  y1:=c[s,3]*m;
  m:=d/(d-c[f,1]);
  x2:=c[f,2]*m;
  y2:=c[f,3]*m;
  union(x1,y1,x2,y2);
end;
```

El anterior procedimiento es esencialmente al mismo procedimiento proyeccion (d:real), en la vista perspectiva dado anteriormente, pero sin el bucle.

```
PROCEDURE grafica(dreal);
var
  v1,v2:integer;
begin
  test(d);
  arista:=1;
  repeat
    v1:=flag[cr[arista,1]];
    v2:=flag[cr[arista,2]];
    if (v1<>0) or (v2<>0) then
       proyeccion(d);
    arista:=arista+1
    until arista > 12;
end;
```

Estos mismos métodos de proyección pueden utilizarse no solamente para graficar poliedros en tres dimensiones, sino también para cualquier dimensión $n > 3$ así como para cualquier tipo de curvas y superficies en el espacio.

Bibliografía

[1] Coxeter, H.S.M., Fundamentos de Geometría , editorial Limusa, México, 1971.

[2] Foley, J.; Van Dam, A., Fundamentos of Interactive Computer Graphics; Addison-Wesley; Massachussetts, 1984.

[3] Hearn, D; Baker, M., Gráficas por Computador, Prentice Hall, éxico, 1988.

[4] Hilbert, D; Cohn-Vossen, S., Geometry and the Imagination, Chelsea Publishing Company, 1952.

[5] Oldknow, A., Microcomputers in Geometry; John Willey and Sons, n. y, 1987.

[6] Rademacher, H; Toeplitz, O., Números y Figuras; Alianza editorial, S. A.; Madrid, 1970.