

# IGejerciciosresueltos.pdf



patriciacorhid



Informática Gráfica



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación  
Universidad de Granada



**Que no te escriban poemas de amor  
cuando terminen la carrera** ►►►►►►►►

☺  
(a nosotros por  
suerte nos pasa)

**WUOLAH**



// Registrar tabla:

```
glBindBuffer(GL_ARRAY_BUFFER, 0);  
glVertexAttribPointer(3, GL_FLOAT, 0, vertices.data());  
glEnableClientState(GL_VERTEX_ARRAY);
```

} No hace falta volver a registrar la tabla

// Dibujar:

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);  
glDrawArrays(GL_POLYGON, 0, vertices.size());
```

// Restaura variables:

```
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);  
glLineWidth(1);  
glEnable(GL_DEPTH_TEST);  
glBindVertexArray(0);
```

Con Begin - End:

```
void dibujarPoligono(int n) {
```

// Crear vértices + desactivar profundidad igual que en la otra función

// Relleno

```
glColor3f(0, 1, 1);  
glBegin(GL_POLYGON);  
for (int i = 0; i < n; i++)  
    glVertex3fv(vertices[i]);  
glEnd();
```

// Contorno

```
glLineWidth(2);  
glColor3f(0, 0.2, 0.8);  
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);  
glBegin(GL_POLYGON);  
for (int i = 0; i < n; i++)  
    glVertex3fv(vertices[i]);  
glEnd();
```

glBindVertexArray(0);  
// Restaura variables + glBindVertexArray(0);

}

# Que no te escriban poemas de amor cuando terminen la carrera

(a nosotros por suerte nos pasa)



Ayer a las 20:20

Oh Wuolah wuolitah  
Tu que eres tan bonita

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

No si antes decirte  
Lo mucho que te voy a recordar



Envía un mensaje...



**WUOLAH**



③ Con VAO:

```
void dibujarPoligono (int n) {
    // Crear vértices + desactivar profundidad igual que en la otra función
    glColor3f(0, 1, 1);
    // VAO
    GLuint id_vao;
    glGenVertexArrays(1, &id_vao); // Crear 1 VAO
    glBindVertexArray(id_vao); // Activar VAO
    // Crear VBO unsigned long tam = sizeof(Tupla3f)*(unsigned long) vertices.size();
    GLuint id_vbo;
    glGenBuffers(1, &id_vbo); // Crear 1 VBO
    glBindBuffer(GL_ARRAY_BUFFER, id_vbo); // Activar VBO
    glBufferData(GL_ARRAY_BUFFER, tam, vertices.data(), GL_STATIC_DRAW);
    // RDM → GPU
    // Bind Buffer (GL_ARRAY_BUFFER, 0), // No dejar activado el VBO
    // Dibujar relleno:
    glBindBuffer(GL_ARRAY_BUFFER, id_vbo);
    glVertexAttribPointer(0, GL_FLOAT, 0, 0);
    glEnableClientState(GL_VERTEX_ARRAY);
    // Dibujar contorno:
    glLineWidth(2);
    glColor3f(0, 0.2, 0.8);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glDrawArrays(GL_POLYGON, 0, vertices.size());
    glDrawArrays(GL_ARRAY_BUFFER, 0); // No dejar activado el VBO
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    // Restaurar variables + glBindVertexArray(0);
}
```

④ llenar limpia TODA la ventana. (Pag 107)

Con glClearColor (r, g, b, opacidad) cambiamos el color con el que limpiamos (el que se queda tras borrar).

Opacidad  $\in [0, 1]$

⑤ glRect: dibuja un rectángulo.

void glRectf(GLfloat x<sub>1</sub>, GLfloat y<sub>1</sub>, GLfloat x<sub>2</sub>, GLfloat y<sub>2</sub>);

un vértice

el vértice opuesto

GL-COLOR-BUFFER-BIT | GL-DEPTH-BUFFER-BIT;

glClearColor (0.4, 0.4, 0.4); glClear (GL-COLOR-BUFFER-BIT | GL-DEPTH-BUFFER-BIT);  
int m = min { ancho, alto }; glColor3f (1, 1, 1);  
glRectf ( ancho - m, alto - m, ancho + m, alto + m );





## (a nosotros por suerte nos pasa)

No si antes decíte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

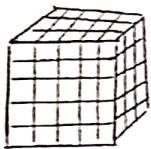
Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

### Tema 2

#### ⑩. Enumeración espacial:



Hay  $K$  cuadraditos en cada fila. Luego hay en total de  $K^3$  celdas. Como cada una ocupa un bit, se requieren  $\frac{K^3}{8}$  bytes.

#### • Modelo de fronteras:



Hay  $K$  meridianos y  $K$  paralelos, como en cada meridiano hay  $K$  vértices y hay  $K \Rightarrow$  hay  $K^2$  vértices. A su vez, cada vértice son 3 componentes que ocupan 4 bytes cada una. Luego son  $3 \cdot 4 \cdot K^2$  bytes para los vértices.

Por otro lado, puedo identificar cada cuadradito por uno de sus vértices, luego hay  $K^2$  cuadraditos ( $2K^2$  triángulos). Para cada triángulo, guardo 3 enteros de 4 bytes cada uno, que son 3  $\cdot$  4  $\cdot$   $2K^2$  bytes para triángulos.

En total se almacenan  $12K^2 + 24K^2 = 36K^2$  bytes.

#### ⑭ Caso 1:

Por cada entrada de la tabla triángulos hay 3 posibles aristas. Para comprobar si hemos añadido ya la arista, usamos una matriz triangular (pares), donde cada fila corresponde a un vértice y contiene los vértices con mayor índice que son adyacentes a este y para los que ya hemos almacenado la arista. Esto tiene complejidad  $O(n)$ .

```
void generarAri() {
    vector<vector<int>> pares(n - 1);
```

```
    int mayor, menor;
```

```
    for (int i = 0; i < tri.size(); i++) {
```

```
        for (int j = 0; j < 3; j++) {
```

```
            menor = min(tri[i][j], tri[i][((j + 1) % 3)]);
```

```
            mayor = max(tri[i][j], tri[i][((j + 1) % 3)]);
```

```
i & (find(pares[menor].begin(), pares[menor].end(), mayor) == pares[menor].end());
```

```
            if (find(pares[menor].begin(), pares[menor].end(), mayor) == pares[menor].end()) {
```

```
                ari.push_back({menor, mayor}); // añadimos la arista
```

```
            }
```

```
            pares[menor].push_back(mayor);
```

s: La arista no  
estaba añadida

ari.push\_back({menor, mayor}); // añadimos la arista

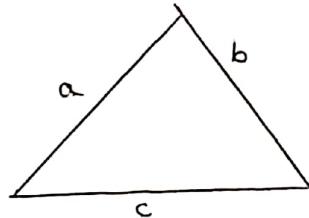


## Caso 2:

Cada arista aparece 2 veces, en una nos da un como  $(a, b)$  y en otra como  $(b, a)$ . Solo metemos una vez, cuando la 1<sup>a</sup> componente es menor que la 2<sup>a</sup>.

void generarArC() {

```
int a, b;
for (int i=0; i<tri.size(); i++) {
    for (int j=0; j<3; j++) {
        a = tri[i][j];
        b = tri[i][((j+1)%3)];
        if (a < b)
            ar.push_back({a, b});
    }
}
```



## (15) Fórmula de Herón:

Área =  $\sqrt{(s-a)(s-b)(s-c)s}$ , donde  $s = \frac{a+b+c}{2}$

• Propiedad del producto vectorial:

Área triángulo =  $\frac{\|\vec{a} \times \vec{b}\|}{2}$  (Pag 162 del Tema 1)

double Área (const Malla& malla)

double a = 0; //área

Tupla3F p, q, r, u, v;

for (int i=0; i<malla.triangulos.size(); i++) {

//Vértices del triángulo

p = malla.vertices[malla.triangulos[i][0]];

q = malla.vertices[malla.triangulos[i][1]];

r = malla.vertices[malla.triangulos[i][2]];

//Dos aristas del triángulo

u = q - p;

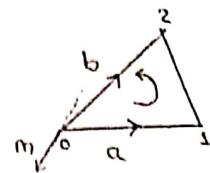
v = r - p;

a += sqrt(u.cross(v).lengthSq());

a /= 2.0;

return a;

## (17) uoc ecalcular Normales Triangulos (12)



Type of a,b,m

```
for (int i=0; i<triangles.size(); ++i){
```

$$\alpha = \text{vertices}[\text{triangles}[i][j]] - \text{vertices}[\text{triangles}[i][k]]$$

$a = \text{vertices}[\text{triangulos}[i][j]]$  - vertices [triangulos[i][j]]  
 $b = \text{vertices}[\text{triangulos}[i][k]]$  - vertices [triangulos[i][k]]

$m = \alpha \cdot \text{cross}(b)$

$m = a \cdot \text{gross}(b)$ ,  
 if ( $m[X] \neq 0$  or  $m[Y] \neq 0$  or  $m[Z] \neq 0$ )

```
m = m.normalized()
```

nor\_in.push\_back(m);

۱

```
void calcularNormales();
```

calculer Normales Triangulaires();

```
for (int i=0; i<vertices.size(); i++)
```

nor-ver. push-back (<0.0,0.0,0.0>);  
e.g.  $\text{S} \rightarrow \text{S} \cup \{\text{f}\}$ : (++) // Par

```
for( int i=0; i < triangulos.size(); i++ ) { // recorre los vértices
    for( int j=0; j < 3; j++ ) { // recorre los vértices [i][j], [i][j+1] y [i][j+2]
        if( triangulos[i][j].size() == 3 ) { // si el triángulo es simple
            triangulos[i][j].normal = normala( vertices[triangulos[i][j]], vertices[triangulos[i][j+1]], vertices[triangulos[i][j+2]] );
        }
    }
}
```

```
for( int j=0; j<3; j++ ) {
    nor-ver[triangles[i][j]] = nor-ver[triangles[i][j]] + size(j); j++)
}
```

for (int i = 0; i < nor\_ver.size(); i++)  
 for (int j = 0; j < nor\_ver[i].size(); j++)  
 if (nor\_ver[i][j] == 0) nor\_ver[i][j] = 1;

```
for (int i = 0; i < nor_ver.size(); i++) {  
    if (nor_ver[i].j & j == 0) {  
        nor_ver[i].j = nor_ver[i].normalized();  
    }  
}
```

nor-ve [i] = nor-ve ve ve

{ en 1972), el nuevo vértice será:

18) Dado un vértice  $(x_1, y_1, z)$ , el nuevo vértice  $(x_1 + N_x, y_1 + N_y, z + N_z)$  es su normal.

19) Con el test de profundidad desactivado, si pintemos las aristas  $(x, y, z) + N(x, y, t)$ , —

Con el test de profundidad desactivado, si pintemos las aristas de lante del relleno, después se nos quedan todas las aristas de lante del relleno, como las aristas no sirve. Con el test de profundidad, como las aristas tienen la misma  $z$  que los triángulos, por pequeños errores de cálculo algunas aristas pueden quedar ocultas por los triángulos. Para solucionarlo, desplazaremos los vértices de la malla en dirección de su normal, para que quede por encima del relleno. Se escribe lo siguiente encima del código:

glEnable(GL\_POLYGON\_OFFSET\_FILL); *Fill para que afecte al relleno, Line para las líneas.*  
 glPolygonOffset(1.0, 1.0);

Los valores positivos del primer parámetro meten el dibujo hacia adentro, y los negativos lo sacan para afuera.

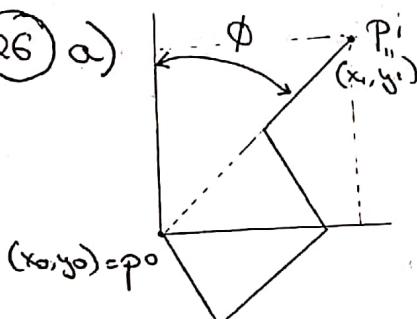
(Y ya podemos añadir aquí código como en el ejercicio ③ para dibujar la figura).

## 24 void gancho()

```
glBegin(GL_UNESTRIPE);
    glVertex2f(0, 0);
    glVertex2f(1, 0);
    glVertex2f(1, 1);
    glVertex2f(0, 1);
    glVertex2f(0, 2);
    glEnd();
```

{

## 26 a)



Componemos la matriz Modelview para trasladar el gancho y que aparezca como en la figura.  
 Como las rotaciones se hacen en sentido antihorario, rotamos con ángulo -phi.

```
void gancho_2p_a(float x0, float y0, float x1, float y1)
{
    glMatrixMode(GL_MODELVIEW); glLoadIdentity();
    float phi = atan2(x1 - x0, y1 - y0); // ángulo phi
    float mod = sqrt((x0 - x1) * 2 + (y0 - y1) * 2); // ||p1 - p0||

    float esc = mod / 2.0;
    glTranslatef(x0, y0, 0);
    glRotatef(-phi * 180 / M_PI, 0, 0, 1);
    glScalef(esc, esc, esc);
    gancho();
}
```

{



(a nosotros por suerte nos pasa)

No si antes decirte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

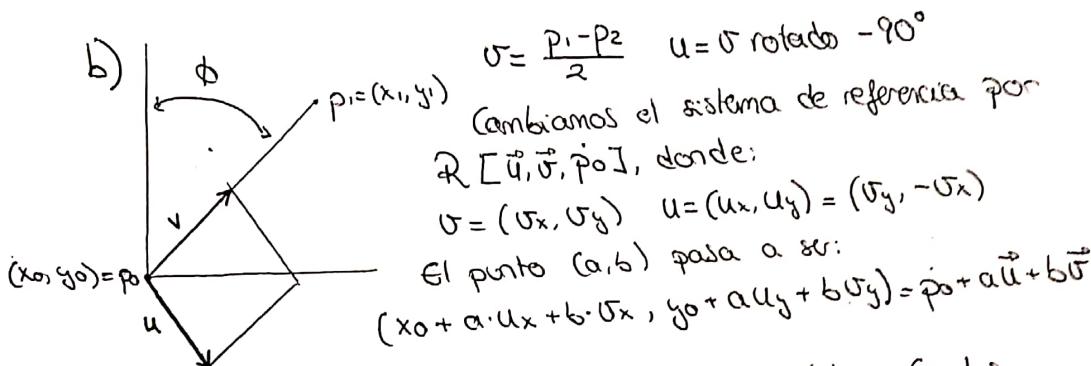
Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

②6) a) NOTA 1: atan2( $\Delta x, \Delta y$ ) hace la arctangente bien, ya que tiene en cuenta los signos de  $\Delta x$  y  $\Delta y$  para devolver el ángulo (en radianes) en el cuadrante correcto.

NOTA 2: El escalado y la rotación pueden intercambiarse por ser el escalado uniforme. Si no, el escalado iría primero.

NOTA 3: El factor de esc =  $\frac{\|p_1 - p_0\|}{2}$  porque la longitud del gancho (2 unidades) la lleva a  $\frac{\|p_1 - p_0\|}{2}$ , luego cada unidad de escala  $\frac{\|p_1 - p_0\|}{2}$



$$v = \frac{p_1 - p_0}{2} \quad u = v \text{ rotado } -90^\circ$$

Cambiemos el sistema de referencia por  $R [u, v, p_0]$ , donde:

$$v = (v_x, v_y) \quad u = (u_x, u_y) = (v_y, -v_x)$$

El punto  $(a, b)$  pasa a ser:

$$(x_0 + a \cdot u_x + b \cdot v_x, y_0 + a \cdot u_y + b \cdot v_y) = p_0 + a \bar{u} + b \bar{v}$$

②7) Para pintar una figura con n ganchos. Los vértices (puntos iniciales y finales de éstos) serán las raíces n-ésimas de la unidad.

Como el primer gancho de la figura no empieza en el  $(1, 0, 0)$ , sino que empieza a mitad, metemos un desfase al ángulo de la mitad del ángulo que usaremos:

void ensamblaje (int n) {

    float alpha, beta;

    for (int i=0; i<n; i++) {

        alpha = (float)(i-0.5)\*2.0\*M\_PI/n;

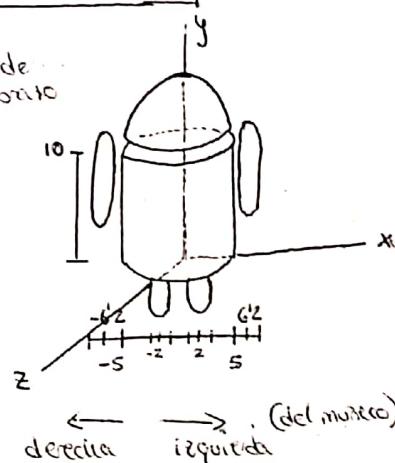
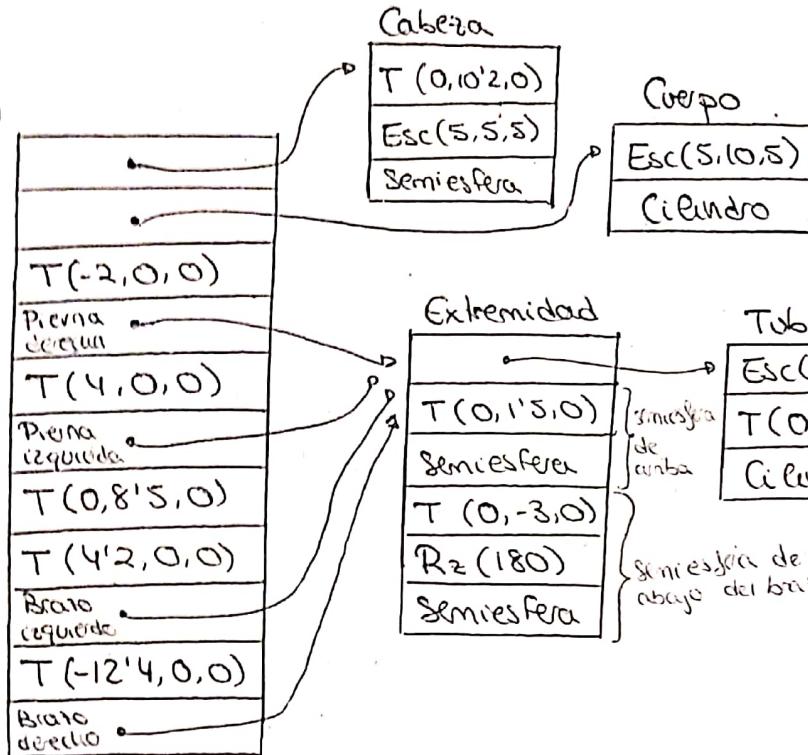
        beta = (float)(i+0.5)\*2.0\*M\_PI/n;

        ganchos - 2p - b(cos(alpha), sin(alpha), cos(beta), sin(beta));

    }

en vez de ser  $i, (i+1)$ , con el desfase queda  $(i-0.5), (i+0.5)$ .

(29) a)



b) void Tubo()

```
glPushMatrix();
    glScalef(1, 3, 1);
    glTranslatef(0, -0'5, 0);
    Cilindro();
    glPopMatrix();
```

```
} void Extremidad() {
```

```
    glPushMatrix();
        Tubo();
        glTranslatef(0, 1'5, 0);
        Semiesfera();
        glTranslatef(0, -3, 0);
        glRotatef(180, 0, 0, 1);
        Semiesfera();
        glPopMatrix();
```

```
} void Cabeza() {
```

```
    glPushMatrix();
        glTranslatef(0, 10'2, 0);
        glScalef(5, 5, 5);
        Semiesfera();
        glPopMatrix();
```

```
}
```

(ejer 29)

```
void Extremidad(float alpha) {
    glPushMatrix();
    glTranslatef(0, 1'5, 0);
    glRotatef(alpha, 0, 0, 1);
    glTranslatef(0, -1'5, 0);
    :
```

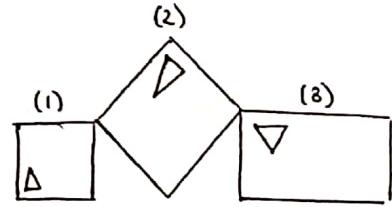
(ejer 30)

```
void Cabeza(float plui) {
    glPushMatrix();
    glRotatef(plui, 0, 1, 0);
    :
```

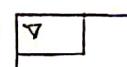
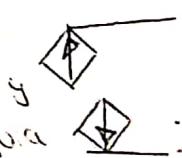
(29) b) void Cuerpo() {  
 glPushMatrix();  
 glScalef(5, 10, 5);  
 Cilindro();  
 glPopMatrix();  
}

(ejer 30)

void Android() { //void Android effekt alpha, kai beta, fka. fka. fka.  
 Cabeza(); //Cabeza (phi)  
 Cuerpo();  
 glTranslatef(-2, 0, 0);  
 Extremidad(); //Extremidad (0)  
 glTranslatef(4, 0, 0);  
 Extremidad(); //Extremidad (0)  
 glTranslatef(0, 8.5, 0);  
 glTranslatef(4.2, 0, 0);  
 Extremidad(); //Extremidad (alpha)  
 glTranslatef(-12.4, 0, 0);  
 Extremidad(); //Extremidad (beta)  
}



(31) void figura\_completa() {  
 glMatrixMode(GL\_MODELVIEW);  
 glLoadIdentity();  
 glPushMatrix();  
 glScalef(0.5, 0.5, 0.5); //Rescalado todo la figura  
 figura\_simple(); (1)  
 glPushMatrix();  
 glTranslatef(2, 2, 0); //Reflejo respecto eje y  
 glScalef(1, -1, 1); //Reflejo respecto eje y  
 glRotatef(45, 0, 0, 1); //Roto la figura  
 glScalef(sqrt(2), sqrt(2), sqrt(2));  
 figura\_simple(); (2)  
 figura\_simple(); (3)  
 glPopMatrix();  
 glPopMatrix();  
 glTranslatef(3, 1, 0);  
 glScalef(2, -1, 1); //Escala y reflejo a la vert ~  
 figura\_simple(); (3)  
 glPopMatrix();  
}



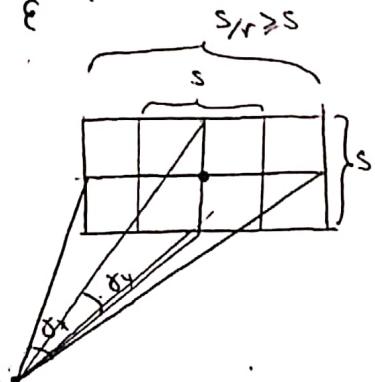
(32) void figura-compleja-rec (int altura){  
 glMatrixMode(GL\_MODELVIEW);  
 glLoadIdentity();  
 rec(altura);  
 }

void rec (int altura){  
 if (altura == 0) // Si soy de los chiquitos, me dibujo  
 figura-simple();

else{  
 figura-simple();  
 glTranslatef(1, 0, 0);  
 glScalef(0.5, 0.5, 0.5);  
 rec(altura - 1);  
 glTranslatef(0, 1, 0);  
 rec(altura - 1);  
 }

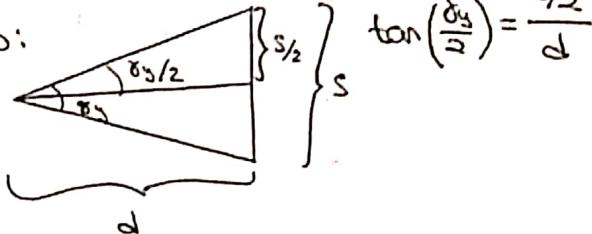
// Si no:  
 Dibujo el fractal de altura - 1, lo  
 desplazo para arriba y dibujo  
 debajo otro igual.  
 deabajo izq, lo traslado y "me  
 escalo" (red), lo traslado y "me  
 dibujo" (dibujo al grande).

(41)



$$\text{si ancho} > \text{alto} \Rightarrow r = \frac{\text{alto}}{\text{ancho}} \leq 1 \quad y \quad \delta_m = \delta_y$$

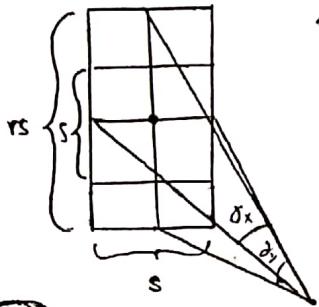
por tanto:



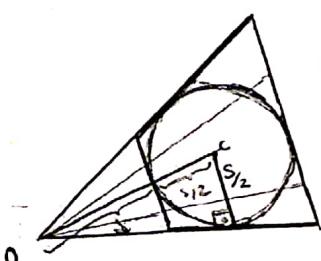
$$\text{si alto} < \text{ancho} \Rightarrow r > 1: \quad y \quad \delta_m = \delta_x$$

La manera de sacar d es análoga

$$\tan\left(\frac{\delta_m}{2}\right) = \frac{s/2}{d} \Rightarrow d = \frac{s}{2 \tan\left(\frac{\delta_m}{2}\right)}$$



(42)



Como en el (41), lo único que necesitas  
 para aplicar eso es calcular la d de  
 forma diferente.

$$\sin\left(\frac{\delta_m}{2}\right) = \frac{s/2}{d+s/2} \Rightarrow d = \frac{s/2}{\sin\left(\frac{\delta_m}{2}\right)} - \frac{s}{2}$$

Mismo código que en el (41) y solo cambio el d.

Que no te escriban poemas de amor  
cuando terminen la carrera ➤➤➤➤➤



WUOLAH

(a nosotros por suerte nos pasa)

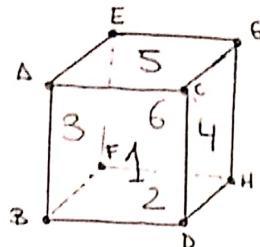
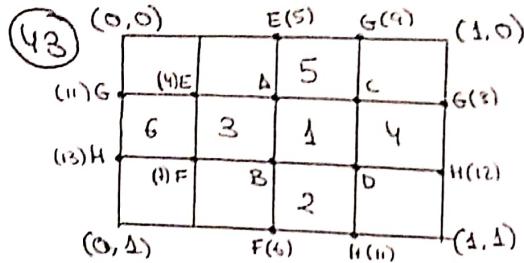
No si antes decirte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita



- Recortar los triángulos en sentido antihorario
- Recortar los triángulos en sentido horario

a) El modelo tendrá como mínimo 14 vértices.  
(El punto A tiene una única consideración de textura,  
pero el F tiene 2, por eso hay que duplicarlo).

b) Dado:: Dado () {

vertices = {  
 <0,1,1> //A 0  
 <0,0,1> //B 1  
 <1,1,1> //C 2  
 <1,0,1> //D 3  
 <0,1,0> //E 4  
 <0,1,0> //E 5  
 <0,0,0> //F 6  
 <0,0,0> //F 7  
 <1,1,0> //G 8  
 <1,1,0> //G 9  
 <1,1,0> //G 10  
 <1,0,0> //H 11  
 <1,0,0> //H 12  
 <1,0,0> //H 13
 }

triángulos = {  
 <0,1,2> //Cara 1  
 <1,3,2>  
 <3,1,6> //Cara 2  
 <6,11,3>  
 <0,4,7> //Cara 3  
 <0,7,1>  
 <2,3,12> //Cara 4  
 <2,12,8>  
 <0,2,5> //Cara 5  
 <5,2,9>  
 <4,10,7> //Cara 6  
 <10,13,7>
 }

cc\_tt\_ver = {  
 <0,5,1/8>, //A  
 <0,5,2/8>, //B  
 <0,75,1/3>, //C  
 <0,75,2/3>, //D  
 <0,25,1/3>, //E  
 <0,5,0>, //E  
 <0,5,2>, //F  
 <0,25,2/3>, //F  
 <1,1/8>, //G  
 <0,75,0>, //G  
 <0,1/8>, //G  
 <0,75,2>, //H  
 <1,2/8>, //H  
 <0,2/3>, //H
 }

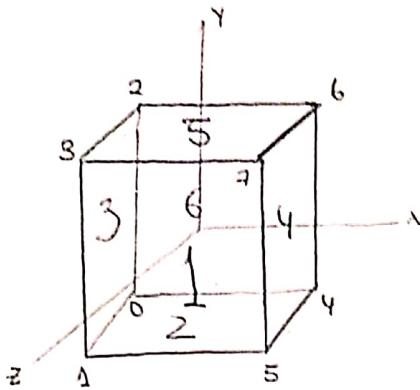
NodoDado:: NodoDado () {

```
  Textura * text = new Textura("dice.jpg");
  Material * mat = new Material({1, 1, 1, 1});
  agregar(mat);
  agregar(new Dado);
}
```

(44)

			10	14		
		2	5	15		
22	18	2	3	19	23	7
6	3	1	17	21	5	4
20	16	0				4
			9	13		
			2	12		
			8			

Con el cubo de  
centro  $(0,0,0)$  y  
lado 2



Para que funcione necesito triplicar todos los vértices, teniendo 24 en total (porque necesito que las normales estén bien calculadas).

Dado 24:: Dado 24(): mallaInd ("Dado24")

vertices = {  
 $\langle -1, -1, -1 \rangle // 0, 8, 16$   
 $\langle -1, -1, +1 \rangle // 1, 9, 17$   
 $\langle -1, +1, -1 \rangle // 2, 10, 18$   
 $\langle -1, +1, +1 \rangle // 3, 11, 19$   
 $\langle 1, -1, -1 \rangle // 4, 12, 20$   
 $\langle 1, -1, +1 \rangle // 5, 13, 21$   
 $\langle 1, +1, -1 \rangle // 6, 14, 22$   
 $\langle 1, +1, +1 \rangle // 7, 15, 23$   
 $\vdots$ 
}

f;

cc-tt-ver = {  
 $\langle 1/4, 2/3 \rangle // 0$   
 $\langle 1/2, 2/3 \rangle // 1$   
 $\langle 1/4, 1/3 \rangle // 2$   
 $\langle 1/2, 1/3 \rangle // 3$   
 $\langle 1, 2/3 \rangle // 4$   
 $\langle 3/4, 2/3 \rangle // 5$   
 $\langle 1, 1/3 \rangle // 6$   
 $\langle 3/4, 1/3 \rangle // 7$   
 $\langle 1/2, 1/4 \rangle // 8$   
 $\langle 1/2, 2/3 \rangle // 9$   
 $\langle 1/2, 0 \rangle // 10$   
 $\langle 1/2, 1/3 \rangle // 11$   
 $\langle 3/4, 1/4 \rangle // 12$   
 $\langle 3/4, 2/3 \rangle // 13$   
 $\langle 3/4, 0 \rangle // 14$   
 $\langle 3/4, 1/3 \rangle // 15$ 
}

$\langle 1/4, 2/3 \rangle // 16$   
 $\langle 1/2, 2/3 \rangle // 17$   
 $\langle 1/4, 1/3 \rangle // 18$   
 $\langle 1/2, 1/3 \rangle // 19$   
 $\langle 0, 2/3 \rangle // 20$   
 $\langle 3/4, 2/3 \rangle // 21$   
 $\langle 0, 1/3 \rangle // 22$   
 $\langle 3/4, 1/3 \rangle // 23$

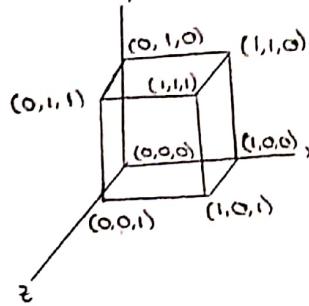
f;

NORMALES DE LOS VÉRTICES:

$0, 1, 2, 3: \langle -1, 0, 0 \rangle$   
 $4, 5, 6, 7: \langle 1, 0, 0 \rangle$   
 $8, 9, 12, 13: \langle 0, -1, 0 \rangle$   
 $10, 11, 14, 15: \langle 0, 1, 0 \rangle$   
 $16, 18, 20, 22: \langle 0, 0, -1 \rangle$   
 $17, 19, 21, 23: \langle 0, 0, 1 \rangle$

f

(45) (Cubo 24)

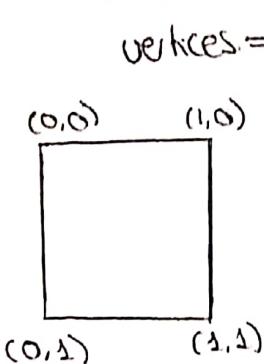


cc-tt-ver2

$\{0,1\}$  //0  
 $\{1,1\}$  //1  
 $\{0,0\}$  //2  
 $\{1,0\}$  //3  
 $\{0,1\}$  //4  
 $\{1,1\}$  //5  
 $\{0,0\}$  //6  
 $\{1,0\}$  //7  
 $\{0,1\}$  //8  
 $\{1,0\}$  //9  
 $\{0,0\}$  //10  
 $\{1,1\}$  //11  
 $\{0,1\}$  //12  
 $\{1,1\}$  //13  
 $\{0,0\}$  //14  
 $\{1,1\}$  //15  
 $\{0,0\}$  //16  
 $\{1,0\}$  //17  
 $\{0,1\}$  //18  
 $\{1,1\}$  //19  
 $\{0,0\}$  //20  
 $\{1,1\}$  //21  
 $\{0,0\}$  //22  
 $\{1,0\}$  //23

8;

(Cubo 24): MallaInd ("Cubo 24")



vertices = {

{0,0}	{1,0}
{0,1}	{1,1}

}

$\{0,0,0\}$  //0  
 $\{0,0,1\}$  //1  
 $\{0,1,0\}$  //2  
 $\{0,1,1\}$  //3  
 $\{1,0,0\}$  //4  
 $\{1,0,1\}$  //5  
 $\{1,1,0\}$  //6  
 $\{1,1,1\}$  //7  
  
 $\{0,0,0\}$  //8  
 $\{0,0,1\}$  //9  
 $\{0,1,0\}$  //10  
 $\{0,1,1\}$  //11  
 $\{1,0,0\}$  //12  
 $\{1,0,1\}$  //13  
 $\{1,1,0\}$  //14  
 $\{1,1,1\}$  //15  
  
 $\{0,0,0\}$  //16  
 $\{0,0,1\}$  //17  
 $\{0,1,0\}$  //18  
 $\{0,1,1\}$  //19  
 $\{1,0,0\}$  //20  
 $\{1,0,1\}$  //21  
 $\{1,1,0\}$  //22  
 $\{1,1,1\}$  //23

(46) a) La reflectividad del material en  $P$  (pag 96) da la iluminación en cada punto viene dada por:

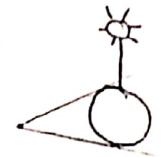
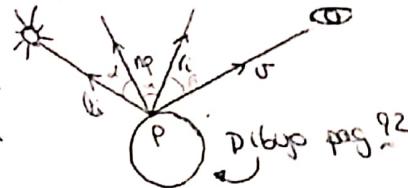
$$C_i = M_s(p) + M_d(p) \max(0, n \cdot l_i) + M_g(p) \cdot d_i [\max(0, n \cdot v)]^e$$

$$\text{En este caso, } C_i = \max(0, n \cdot l_i) \times (1, 1, 1)$$

Como  $n \cdot l_i = \|n\| \|l_i\| \cos(\alpha)$  donde  $\alpha$  es el ángulo que forman, el punto con máxima iluminación es aquel tal que  $d_i = 0$ . ( $n = l_i$ )

Por tanto, el punto con máxima iluminación es el  $(0, 1, 0)$ , el polo norte de la esfera.

Ese punto no es visible por el observador, que está en el punto  $(2, 0, 0)$ , solo puede ver hasta el cono tangente a la esfera.



b) Modelo de Blinn-Phong (pag 93) da reflectividad pseudo-especular del material se obtiene:

$$f_{rs}(p, v, l_i) = M_s(p) d_i [n_p \cdot u_i]^e$$

El punto de máxima iluminación  $d_i$  es 1 si  $n_p \cdot l_i > 0 \Rightarrow$  el punto de máxima iluminación está en el ~~hemisferio~~ hemisferio norte.

Este es el caso.  $M_s(p)$  es  $(1, 1, 1)$  en nuestro caso. Para maximizar  $f_{rs}(p, v, l_i)$  hay que maximizar  $n_p \cdot u_i$ , y eso es máximo cuando el ángulo que forman es 0, es decir, cuando  $u_i = n_p$ .

$$u_i = \frac{l_i + v}{\|l_i + v\|} \text{ bisectriz de } l_i \text{ y } v.$$

$$l_i = \overrightarrow{p \cdot q_i} = \frac{(-p_x, 2-p_y, -p_z)}{\|(-p_x, 2-p_y, -p_z)\|}$$

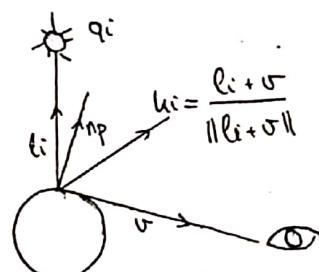
$$v_i = \overrightarrow{p \cdot o} = \frac{(2-p_x, -p_y, -p_z)}{\|(2-p_x, -p_y, -p_z)\|}$$

$$\text{Tomo } p = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0\right), n_p = \overrightarrow{0p} = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0\right)$$

$$l_i = \left(-\frac{\sqrt{2}}{2}, 2 - \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right) / 1.6345 \quad \Rightarrow \quad u_i = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0\right) \Rightarrow u_i = n_p$$

$$v_i = \left(2 - \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right) / 1.6345$$

Por tanto  $p$  es un punto de máxima iluminación.





## (a nosotros por suerte nos pasa)

No si antes decirte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

### (46) b) PARA SABER SI UN PUNTO ES VISIBLE:

Trazamos la recta que pasa por el punto y por el observador, y la intersecamos con la esfera.

De las soluciones del sistema, solo es observable la solución más cercana al observador.

La recta que une  $p = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$  con  $(2,0)$  es:

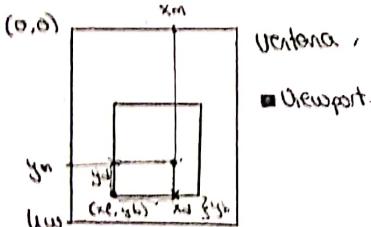
$$y = \frac{-\sqrt{2}}{4-\sqrt{2}}(x-2), \text{ luego:}$$

$$\begin{cases} x^2 + y^2 = 1 \\ y = \frac{-\sqrt{2}}{4-\sqrt{2}}(x-2) \end{cases} \dots$$

### TGMA 4

### (47) • Coordenadas de dispositivo:

Son aquellas relativas al viewport.



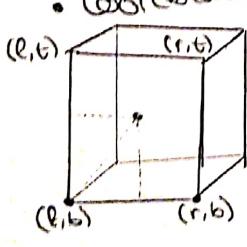
Posemos las coordenadas en función de la ventana (la ventana superior  $\Rightarrow$  es el  $(0,0)$  y la  $y$  crece para abajo) a coordenadas relativas al viewport ( $(x_l, y_b)$  es el  $(0,0)$  y la  $y$  crece para arriba) relativos a la esquina inferior izq.

$$x_d = x_m - x_l$$

$$y_d = l_w - y_m - y_b \quad (\text{en pixeles})$$

### • Coordenadas normalizadas de dispositivo:

Normalizamos las coordenadas de dispositivo



• Coordenadas de mundo:

El viewport es la cara delantera del view-frustum, que está en el mundo. Como las coordenadas normalizadas de dispositivo son la proporción donde está el punto si el ancho y alto fuese 1, tenemos que:

$$x_w = l + \frac{x_{ndc}}{\text{ancho total}}(r-l)$$

$$y_w = b + \frac{y_{ndc}}{\text{proporción alto total}}(t-b)$$

Sumamos  $(l, b)$  porque no mediamos respecto al  $(0, 0)$ .

## TEMAS 5

48 (Pág 62)

El rayo parte de  $O$  y tiene vector director  $d$ , luego la ec. de la semirecta que lo describe es:  $p(t) = O + t d$  con  $t > 0$ .

Si el rayo es paralelo al plano que contiene al triángulo  $\Rightarrow$  no hay intersección.

$\Rightarrow$  no hay intersección.

Si no es paralelo  $\Rightarrow$  interseca con el plano, hay que ver:

Si la intersección cae dentro o fuera del triángulo.

Todo punto del triángulo tiene las coordenadas barycentricas:

$$q = a(\vec{v}_1 - \vec{v}_0) + b(\vec{v}_2 - \vec{v}_0) + \vec{v}_0 \quad a \geq 0, b \geq 0, a+b \leq 1$$

La intersección con el plano se da si  $\exists t > 0$  tal que:

$$(*) O + t d = \vec{v}_0 + a(\vec{v}_1 - \vec{v}_0) + b(\vec{v}_2 - \vec{v}_0)$$

Tomamos el sistema de referencia  $R[\vec{v}_1 - \vec{v}_0, \vec{v}_2 - \vec{v}_0, \vec{n}, \vec{v}_0]$ , donde  $\vec{n}$  es la normal del triángulo. En este sistema de referencia, el triángulo está en el plano  $z=0$ .

A partir de ahora trabajamos con coordenadas respecto a dicho sistema de referencia.

$O R = (0_x, 0_y, 0_z) \quad d_R = (d_x, d_y, d_z)$ . Luego, (\*) quedaría:

$$O R + t d_R = (a, b, 0). \quad \text{El sistema a resolver es:}$$

$$\begin{cases} 0x + t d_x = a \\ 0y + t d_y = b \\ 0z + t d_z = 0 \end{cases} \Rightarrow$$

1) Si  $d_z = 0 \Rightarrow$  el rayo es paralelo al plano (o está contenido), luego no hay intersección.

2) Si  $d_z \neq 0$ :

$$t = \frac{-0z}{dz}, \quad a = 0x + \frac{-0z}{dz} d_x, \quad b = 0y - \frac{0z}{dz} d_y.$$

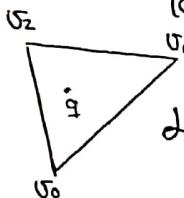
El rayo interseca con el triángulo si:

$$-a + b \leq 1$$

$$-a \geq 0, b \geq 0$$

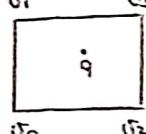
$$-t > 0$$

Pasamos  $(a, b, 0)$  al sistema de coordenadas del mundo y calculamos su distancia con  $O$ . (coordenadas de  $q$  y  $t$  pedidos)



(48) AMPLIACIÓN:

Si en vez de un triángulo tengo un paralelogramo,  
los puntos del cuadrado son:



$$q = a(v_1 - v_0) + b(v_2 - v_0) + v_0 \text{ con } a \geq 0, b \geq 0, \max\{a, b\} \leq 1$$

(porque en mi sistema de referencia  $\|v_2 - v_0\| = \|v_1 - v_0\| = 1$ )

(49) Sea  $q = a(v_1 - v_0) + b(v_2 - v_0) + v_0$  el punto de intersección dentro del triángulo, entonces:

$$q = (1-a-b)v_0 + av_1 + bv_2$$

Luego su normal va a ser:

$$n_q = \frac{(1-a-b)n_0 + an_1 + bn_2}{\|(1-a-b)n_0 + an_1 + bn_2\|}$$

Del mismo modo, sus coordenadas de texture serán:

$$cc\_tt_q = (1-a-b)cc\_tt_0 + acc\_tt_1 + bcc\_tt_2$$

O los del color:

$$c_q = (1-a-b)c_0 + ac_1 + bc_2$$

(50) Superficie de la esfera:  $\lambda p |F(p)| = 0$  con

$$F(p) = \|p - c\|^2 - r^2, \text{ con } p = o + td, t > 0.$$

$F(p) = \|o + td - c\|^2 - r^2$ , si  $p$  en el interior de la esfera

$$F(p) = \begin{cases} < 0 & \text{si } p \text{ en el interior de la esfera} \\ = 0 & \text{" " en la esfera} \\ > 0 & \text{" " el exterior de la esfera} \end{cases}$$

La intersección del rayo con la esfera es la solución

$$\text{de } \|o + td - c\|^2 - r^2 = 0 \Rightarrow$$

$$\text{de } \|o + td - c\|^2 - r^2 = 0 \Rightarrow (ox + tdx - cx)^2 + (oy + tdy - cy)^2 + (oz + tdz - cz)^2 - r^2 = 0$$

$$\Rightarrow (ox + tdx - cx)^2 + (oy + tdy - cy)^2 + (oz + tdz - cz)^2 = r^2$$

Vemos si  $\exists t > 0$  que sea posible

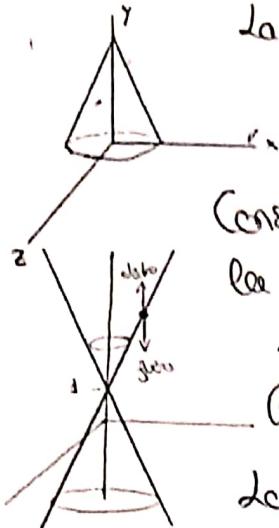
Vemos si hay 2 soluciones: tomamos la más pequeña

- Si hay 2 soluciones: toma la más pequeña

- Si hay 1 solución: esa es la intersección

- Si no hay solución: no hay intersección

(51) • Cono: Habrá como máximo 2 intersecciones.



La ecuación del cono es:

$$\left\{ \begin{array}{l} x^2 + z^2 = (y-1)^2 \\ \text{cono} \end{array} \right. \quad 0 \leq y \leq 1 \quad \cup \quad \left\{ \begin{array}{l} x^2 + z^2 \leq 1, y=0 \\ \text{base} \end{array} \right. \quad y=0$$

Consideramos el cono infinito  $x^2 + z^2 = (y-1)^2$ , y

la función  $F(x, y, z) = x^2 + z^2 - (y-1)^2$

$$F(x, y, z) = \begin{cases} < 0 & \text{si } (x, y, z) \text{ fuera del cono infinito} \\ = 0 & " " \quad \text{en el cono infinito} \\ > 0 & " " \quad \text{dentro del cono infinito} \end{cases}$$

(signo circular)

La intersección del rayo  $0+td$  con el cono infinito es la solución de la ecuación:

$$(0x + tdy)^2 + (0z + tdz)^2 - (0y + tdy - 1)^2 = 0$$

tales que  $t > 0$ .

De esas soluciones, nos quedamos con aquellas tales que  $0y + tdy \in [0, 1]$ , que son las que intersectan con nuestro cono original.

\* Si salen 2 soluciones distintas de este tipo, estas son nuestras intersecciones.

\* En otro caso, calculemos los puntos de corte con la tapa.

- Si  $dy = 0$ , nunca corta la tapa (paralelo a ésta).

- Si  $dy \neq 0$ : Tomo  $t_0 = \frac{-0y}{dy}$  (despejo de  $0y + tdy = 0$ )

Si  $t_0 < 0$ , no corta la tapa

Si otro caso, habrá intersección con la tapa

$$\text{Si } (0x + t_0 dy)^2 + (0z + t_0 dz)^2 \leq 1. \quad (x^2 + z^2 \leq 1)$$

Si  $(0x + t_0 dy)^2 + (0z + t_0 dz)^2 > 1$

De las soluciones obtenidas, la intersección del rayo con el objeto será la más cercana al observador.  
(la que tiene el  $t$  menor)

Que no te escriban poemas de amor  
cuando terminen la carrera ➡➡➡➡➡



WUOLAH

(a nosotros por suerte nos pasa)

No si antes decirte  
Lo mucho que te voy a recordar

Pero me voy a graduar.  
Mañana mi diploma y título he de  
pagar

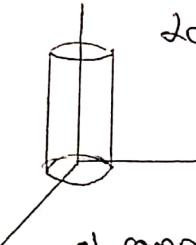
Llegó mi momento de despedirte  
Tras años en los que has estado mi  
lado.

Siempre me has ayudado  
Cuando por exámenes me he  
agobiado

Oh Wuolah wuolah  
Tu que eres tan bonita

51

• Cilindro: Habrá como máximo 2 intersecciones



La ecuación de nuestro cilindro es:

$$\begin{cases} x^2 + z^2 = 1, 0 \leq y \leq 1 \\ x^2 + z^2 \leq 1, y = 0 \end{cases}$$
$$x^2 + z^2 \leq 1, y = 1$$

Consideramos el cilindro infinito  $x^2 + z^2 = 1$  y  
el campo escalar  $F(p) = \begin{cases} < 0 & \text{si } p \text{ está dentro del cilindro} \\ = 0 & \text{si } p \text{ está en el cilindro} \\ > 0 & \text{si } p \text{ " fuera del cilindro} \end{cases}$

con  $F(x, y, z) = x^2 + z^2 - 1$ .  
La intersección del rayo  $O + t\vec{d}$  con el cono infinito es la  
solución de la ecuación:

$$(Ox + t\vec{d}x)^2 + (Oz + t\vec{d}z)^2 - 1 = 0 \quad (\text{tales que } t > 0)$$

De estas soluciones nos quedamos con las que  
son las que intersectan con  
 $Oy + t\vec{d}y \in [0, 1]$ , que son las que intersectan con  
nuestro cilindro original.

- \* Si salen dos soluciones distintas de este tipo, estas  
son nuestras intersecciones.
- \* En otro caso, calculamos los puntos de corte con  
las tapas.

- Si  $\vec{d}y = 0$ : nunca corta las tapas (paralelo a estas)

- Si  $\vec{d}y \neq 0$ :

Tapa inferior: Está en el plano  $y=0$ . Despejo de  $Oy + t\vec{d}y = 0$ .

Tomo  $t_0 = \frac{-Oy}{\vec{d}y}$ . Si  $t_0 < 0$ , no corta la tapa.

Tapa superior: Habrá intersección con la tapa si:

$$(Ox + t\vec{d}x)^2 + (Oz + t\vec{d}z)^2 \leq 1$$

$$(Ox + t\vec{d}x)^2 + (Oz + t\vec{d}z)^2 \leq 1 \quad (\text{despejo de } Oy + t\vec{d}y = 1)$$

Tapa superior: Está en el plano  $y=1$ . Despejo de  $Oy + t\vec{d}y = 1$ .

Tomo  $t_0 = \frac{1-Oy}{\vec{d}y}$ . Si  $t_0 < 0$ , no corta la tapa.

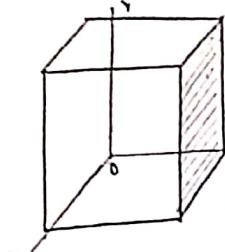
En otro caso, habrá intersección con la tapa si:

$$(Ox + t\vec{d}x)^2 + (Oz + t\vec{d}z)^2 \leq 1$$

De las soluciones obtenidas, la intersección del rayo  
con el objeto será la que tenga la  $t$  menor.

• **Cubo:** Consideramos el cubo de lado dos y centro

$(0,0,0)$ . Habrá como máximo dos intersecciones con el cubo, mientras que no las encontramos, seguimos buscando.



Para cada cara del cubo, hay que ver si el rayo

interseca con esta.

Tomemos por ejemplo la cara  $\pi$ , situada en el plano  $x=1$ .

Buscamos las soluciones de  $0x + tdx = 1$ , que será la intersección del rayo con el plano. Si  $dx=0$ , paralelo,

no cortará ni el plano  $x=1$  ni el  $x=-1$ .

Si  $dx \neq 0$ , tomo  $t_0 = \frac{1-0x}{dx}$ . Si  $t_0 < 0$ , no hay

intersección con el plano.

Si  $t_0 > 0$ , comprobar si cae dentro de la cara del

cubo, es decir que:

$$-1 < 0y + t_0 dy < 1 \quad y \quad -1 < 0z + t_0 dz < 1.$$

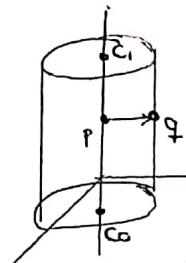
(Este es igual para todas las caras, repetir hasta encontrar 2 intersecciones, que es el máximo que puede haber, aunque puede haber menos).

De las soluciones encontradas, nos quedamos con aquella con  $t$  menor.

⑤ La idea es normalizar el objeto y aplicarle esas mismas transformaciones al rayo. Usamos los ejercicios anteriores para calcular la intersección y a esa solución le aplicamos las transformaciones inversas.

(Lo hacemos con un cambio de sistema de referencia)

### 53) • Cilindro:



sea  $q$  el punto donde quiero calcular la normal.

- Si  $q$  no está en las tapas:

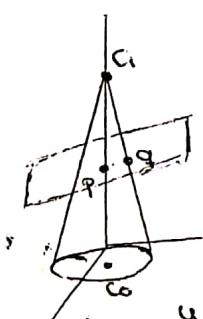
Tomo el plano paralelo a las tapas que pasa por  $q$ . Tomo el punto  $p$ : corte de ese plano con el eje del cilindro. La normal es el vector  $\vec{pq}$  normalizado.

NOTA: El eje del cilindro es la recta que pasa por uno de sus centros,  $(x_0, y_0, z_0)$  y tiene de vector director la resta de ambos centros,  $\sigma = (x_0, y_0, z_0) - (x_1, y_1, z_1)$ .

NOTA: Sean  $v_0, v_1, v_2$  puntos de un plano,  $n = (v_1 - v_0) \times (v_2 - v_0)$  vector normal al plano de coordenadas  $(n_1, n_2, n_3)$ . La ecuación del plano es:  $n_1 x + n_2 y + n_3 z + D = 0$ , sustituyendo en un punto del plano, obtengo el  $D$ .

- Si  $q$  está en la tapa:
  - si está en la tapa de arriba, la normal es  $c_1 - c_0 (\vec{c_0 c_1})$  normalizado.
  - si está en la tapa de abajo, la normal es  $c_0 - c_1 (\vec{c_1 c_0})$  normalizado.

### • Cono:



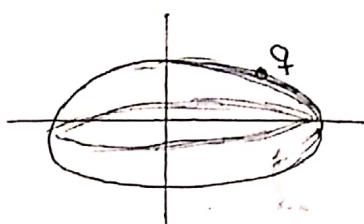
- Si  $q$  está en la tapa:
  - su normal es  $c_0 - c_1 (\vec{c_1 c_0})$  normalizado.

- Si  $q$  no está en la tapa:

Calculo el plano perpendicular a  $\vec{q c_1}$  que pasa por  $q$  (el que tiene a  $\vec{q c_1}$  como normal), y su intersección con el eje del cono,  $p$ .

La normal será el vector  $\vec{pq}$  normalizado.

## • Elipsoide:



## \* Esfera:

La normal es el vector que une el centro con el punto  $q$  normalizado.

## \* Elipsoide: Tomamos $q$ punto en la elipsoide.

Sea  $A$  la matriz que transforma la esfera en el elipsoide.

Como las traslaciones no afectan a la normal

podemos suponer que  $A$  es de  $3 \times 3$ .

Sea  $n$  la normal en la esfera de  $A^{-1}q$ , entonces

la normal de  $q$  es  $n_q = (A^T)^{-1} \cdot n$  (Tema 2, pag 78)

normalizada.

54) (Tema 3, pag 22 y 23)

## Rasterización:

para cada primitiva  $P \neq \emptyset$   
encontrar el conjunto  $S$  de píxeles  $\sim P$

El algoritmo tiene complejidad  $O(n \cdot p)$ .

## Raytracing:

para cada pixel  $q \sim P$   
para cada primitiva  $\sim P \} \max(n, n) = n$

calcular  $T$ , conjunto de primitivas  $\sim P$ .

para cada primitiva  $\sim P$ .

Complejidad  $O(n \cdot p)$ , pero con indexación especial,  
~~supuestamente~~ para cálculo de  $T$  en cada pixel,  
~~supuestamente~~ conseguimos  $O(p \log(n))$ .