

# I Gejerciciosresueltos.pdf



patriciacorhid



Informática Gráfica



4º Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas



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



**Descarga la APP de Wuolah.**  
Ya disponible para el móvil y la tablet.



**WUOLAH**



**El más PRO del lugar puedes ser Tú.**



**¿Quieres eliminar toda la publi de tus apuntes?**



**¡Fuera Publi!**  
Concéntrate al máximo



**Apuntes a full.**  
Sin publi y sin gastar coins

Para los amantes de la inmediatez, para los que no desperdician ni un solo segundo de su tiempo o para los que dejan todo para el último día.

**Quiero ser PRO**

4,95 / mes

## IG: Soluciones Relación General

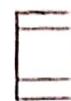
### Tema 1

① `glViewport(x, y, ancho, alto);` (Pag 111)

coordenadas de la esquina inferior izquierda (en pixels)

- Si  $\text{ancho} \leq \text{alto}$ :

`glViewport(0,  $\frac{\text{alto}-\text{ancho}}{2}$ , ancho, ancho);`



- Si  $\text{alto} > \text{ancho}$ :

`glViewport( $\frac{\text{ancho}-\text{alto}}{2}$ , 0, alto, alto);`



- Caso general:

$$m = \min\{\text{ancho}, \text{alto}\};$$

`glViewport( $\frac{\text{ancho}-m}{2}$ ,  $\frac{\text{alto}-m}{2}$ , m, m);`

③ `void dibujarPoligono(int n) {`

`std::vector<Tupla3f> vertices;`

`for (int i=0; i<n; i++) {`

$$\text{float alpha} = i * 2.0 * M\_PI / n;$$

`vertices.push_back( $0.75 * \cos(\alpha)$ ,  $0.75 * \sin(\alpha)$ , 0);`

`}`

`glDisable(GL_DEPTH_TEST); // Desactiva el test de profundidad`

`glColor3f(0, 1, 1);`

`// Registrar tabla:`

`glBindBuffer(GL_ARRAY_BUFFER, 0);`

`glBufferData(GL_ARRAY_BUFFER, 0, vertices.data(), GL_DYNAMIC_DRAW); // Define array de vértices`

`glVertexAttribPointer(3, GL_FLOAT, 0, vertices.data(), GL_VERTEX_ARRAY); // Se hace con tablas de tipo GL_ARRAY_BUFFER`

`glEnableClientState(GL_VERTEX_ARRAY); // Dibuja lo último registrado`

`// Relleno:`

`glDrawArrays(GL_POLYGON, 0, vertices.size());`

`~ indice donde empieza a pintar`

`// Contorno:`

`glLineWidth(2);`

`glColor3f(0, 0.2, 0.8);`

`glDrawArrays(GL_LINES, 0, 2);`

```

// 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());
}

// Restaurar 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);

} // Restaurar variables + glBindVertexArray(0);

③ Con VAO:

```
void dibujarPolígono (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
    glBindBuffer(GL_ARRAY_BUFFER, id_vao); // id_vao
    glBufferData(GL_ARRAY_BUFFER, tam, vertices.size()); // Crear VBO unsigned long tam = sizeof(Tupla3f)*(unsigned long) vertices.size();
    // 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);
    // RAM → GPU
    // No dejar activado el VBO
    glBindBuffer(GL_ARRAY_BUFFER, 0); // No dejar activado el VBO
    // Registrar tabla:
    // glBindBuffer(GL_ARRAY_BUFFER, id_vbo);
    glEnableVertexAttribArray(0, GL_FLOAT, 0, 0);
    glEnableClientState(GL_VERTEX_ARRAY);
    // Dibujar relleno:
    glDrawArrays(GL_POLYGON, 0, vertices.size());
    // Dibujar contorno:
    glLineWidth(2);
    glColor3f(0, 0, 0);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glDrawArrays(GL_POLYGON, 0, vertices.size());
    glBindBuffer(GL_ARRAY_BUFFER, 0); // No dejar activado el VBO
    glBindBuffer(GL_ARRAY_BUFFER, 0); // No dejar activado el VBO
    // Restaurar variables + glBindVertexArray(0);
```

{

- ④ glClear limpia TODA la ventana. (Pag 107)  
 Con glClearColor (r, g, b, opacidad) cambiamos el color con el que se limpian los vértices (el que se queda tras borrar).  
 Opacidad  $\in [0, 1]$
- ⑤ glRect: dibuja un rectángulo.  
void glRectf(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2);  
un vértice el vértice opuesto  
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 ( $\frac{\text{ancho}-m}{2}$ ,  $\frac{\text{alto}-m}{2}$ ,  $\frac{\text{ancho}+m}{2}$ ,  $\frac{\text{alto}+m}{2}$ );

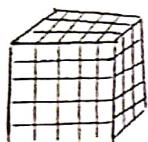
⑥



**INFORMATE EN GRANADA**  
958 222 914 | esgerencia.com

Tema 2

## (10) Enumeración espacial.



Hay  $k$  cuadraditos en cada fila. Luego hay un 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.  
Δ 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 tenemos 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 tenemos almacenada la arista. Esto tiene complejidad  $O(n)$ .

void generarArC() {

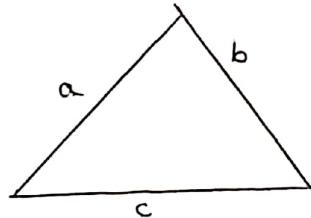
```
vector<vector<int>> pares(n-1);
vector<int> menor;
int mayor, menor;
for (int i=0; i<pri.size(); i++) {
    for (int j=0; j<3; j++) {
        menor = min(pri[i][j], tri[i][((i+1)%3)]);
        mayor = max(pri[i][j], tri[i][(j+1)%3]);
        if (find(pares[menor].begin(), pares[menor].end(), mayor) == pares[menor].end())
            if (pri[i][j] == menor)
                menor.push_back(menor, mayor); // añadimos la arista
            else
                mayor.push_back(menor, mayor);
        pares[menor].push_back(mayor);
    }
}
}
```

## Caso 2:

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

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

• Propiedad del producto vectorial:

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

double Area (const Malla& malla) {

double a = 0; //área

Tuple3F p, q, r, u, v;

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]];

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

//Dos aristas del triángulo

u = q - p;

v = r - p;

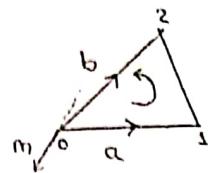
a += sqrt(u.cross(v).lengthSq());

}

a /= 2.0;

return a;

(17) void calcularNormalesTriangulos()



Tipo 3f a, b, m;

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

a = vertices[triangulos[i][1]] - vertices[triangulos[i][0]]; // vértices de los triángulos

b = vertices[triangulos[i][2]] - vertices[triangulos[i][0]]; // vértices de los triángulos

m = a.cross(b);

if (m[X] != 0 || m[Y] != 0 || m[Z] != 0)

m = m.normalized();

nor\_ver.push\_back(m);

}

void calcularNormales()

calcularNormalesTriangulos();

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

nor\_ver.push\_back(<0.0, 0.0, 0.0>);

// Para cada triángulo, suma su normal a los vértices que lo componen.

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

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

nor\_ver[triangulos[i][j]] = nor\_ver[triangulos[i][j]] + nor\_ver[i];

for (int i=0; i<nor\_ver.size(); i++)

if (nor\_ver[i][Z] != 0 || nor\_ver[i][Y] != 0 || nor\_ver[i][Z] != 0)

nor\_ver[i] = nor\_ver[i].normalized();

{

Dado un vértice  $(x, y, z)$ , el nuevo vértice sería:

$(x, y, z) + N(x, y, z) \cdot E$ , donde  $N(x, y, z)$  es su normal.

(18) Con el test de profundidad desactivado, si pintamos las aristas después se nos quedan todas las aristas deante 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 meter el dibujo hacia adentro, y los negativos sacan para afuera.

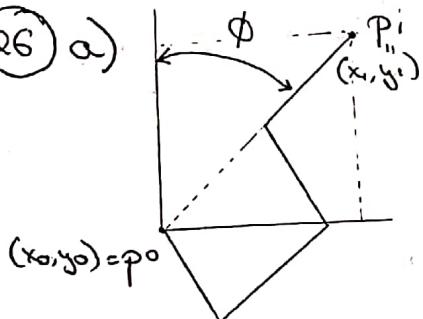
(Y ya podemos añadir aquí código como en el ejercicio ③ para dibujar la figura).

## 24) void gancho()

```
glBegin(GL_UNE_STRIP);
    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_sp_a(float x0, float y0, float x1, float y1)
{
    glMatrixMode(GL_MODELVIEW); glLoadIdentity();
    glTranslatef(x0, y0, 0);
    glRotatef(-phi * 180/M_PI, 0, 0, 1);
    glScalef	esc, esc, esc);
    gancho();
}
```

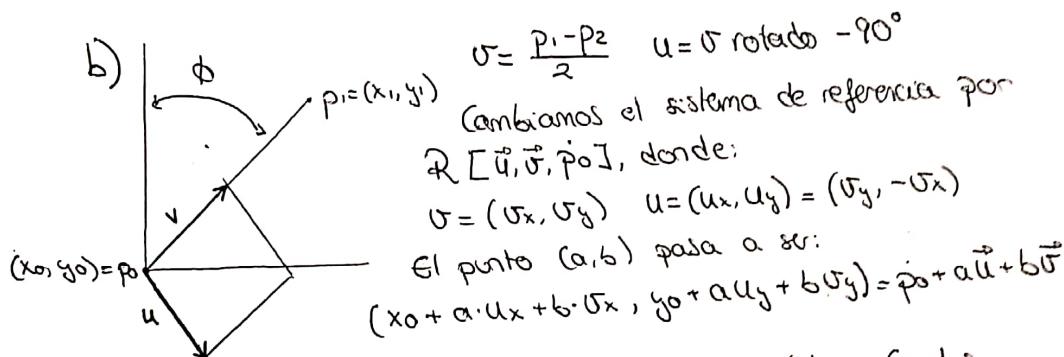
{

INFORMATE EN GRANADA  
958 222 914 | esgerencia.com

26) a) NOTA 1:  $\text{atan2}(\Delta x, \Delta y)$  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 escalamiento y la rotación pueden intercambiarse por ser el escalamiento uniforme. Si no, el escalamiento iría primero.

NOTA 3: El factor de esc =  $\frac{\|\vec{p}_1 - \vec{p}_0\|}{2}$  porque la longitud del gancho (2 unidades) la lleva a  $\|\vec{p}_1 - \vec{p}_0\|$ , luego cada unidad se escala  $\frac{\|\vec{p}_1 - \vec{p}_0\|}{2}$



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 ensenare (int n){

    float alpha, beta;

    for (int i=0; i<n; i++) {

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

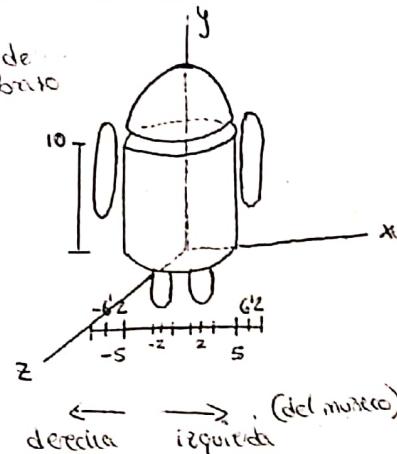
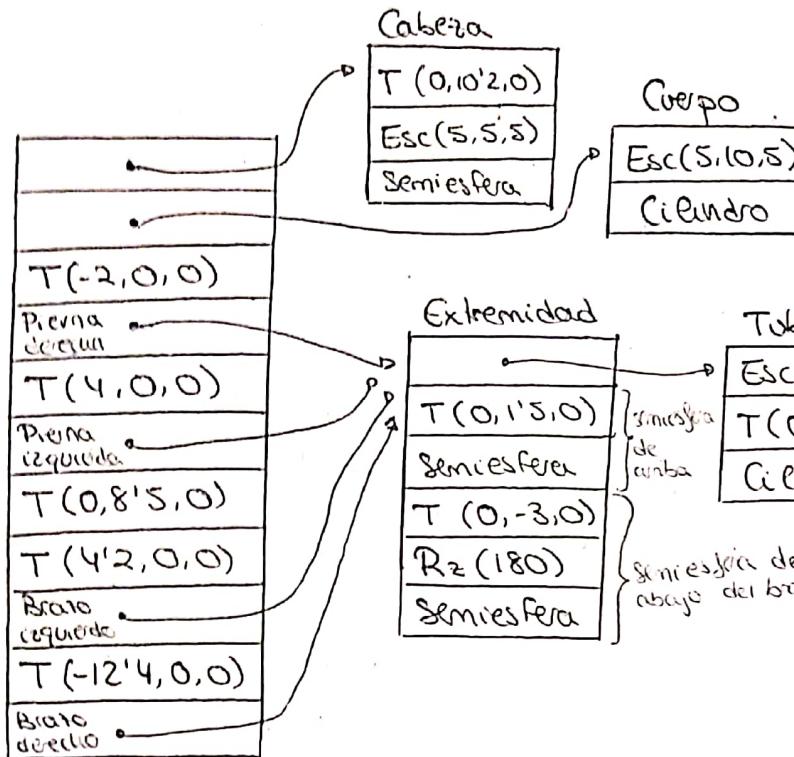
        beta = (float)(i+0.5)\*2.0\*M\_PI/n;

        ganco - 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);
    glTranslatef(0, -3, 0);
    glRotatef(180, 0, 0, 1);
    Semiesfera();
    Semiesfera();
    glPopMatrix();
}

void Cabeza() {
    glPushMatrix();
    glTranslatef(0, 10^2, 0);
    glScalef(5, 5, 5);
    Semiesfera();
    glPopMatrix();
}
```

(ejer 20)

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

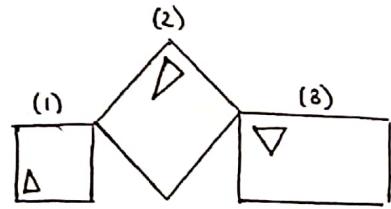
(ejer 20)

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

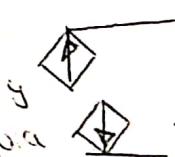
(29) b) void Cuerpo() {  
 glPushMatrix();  
 glScalef(5, 10, 5);  
 Cilindro();  
 glPopMatrix();  
}

(ejer 30)

void Android() { //void Android ffect alpha, ffect beta, ffect phi  
 Cabera(); //Cabera (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)  
}



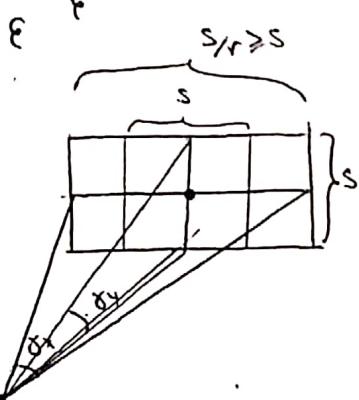
(31) void figura\_completa() {  
 glMatrixMode(GL\_MODELVIEW);  
 glLoadIdentity();  
 glPushMatrix();  
 glScalef(0.5, 0.5, 0.5); //Reescala todo la figura  
 figura\_simple(); (1)  
 glPushMatrix();  
 glTranslatef(2, 2, 0); //Reflejo respecto eje Y  
 glScalef(1, -1, 1); //Rojo la figura  
 glRotatef(45,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



32 void figuraComplejaRec (int altura) {  
 glMatrixMode(GL\_MODELVIEW);  
 glLoadIdentity();  
 rec (altura);

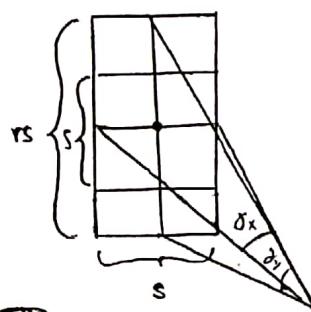
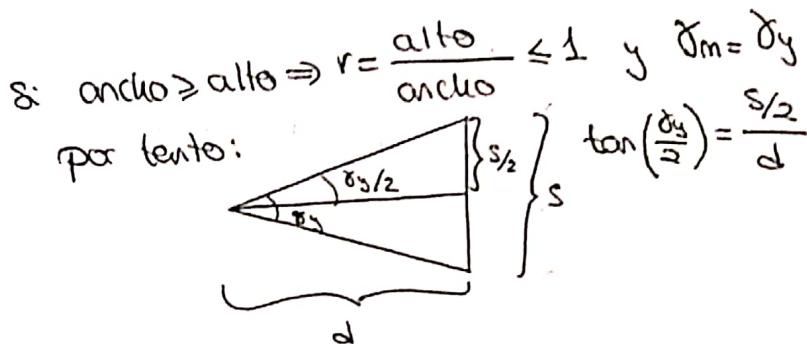
{ void rec (int altura) {  
 if (altura == 0) // Si soy de los chiquiticos, me dibujo  
 figuraSimple();

else { figuraSimple();  
 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 otra vez igual.  
 Escalo todo, lo traslado y "me  
 dibujo" (dibujo al grande).

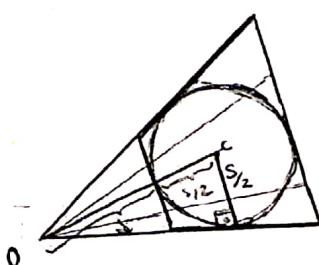
41



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

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

42



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

# AHORA

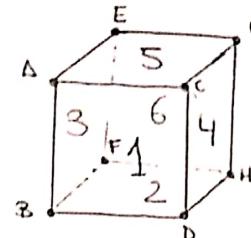
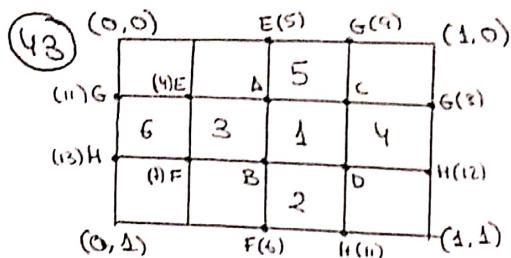
Tu Campus de ESIC en Granada.

MBAs, Marketing, Finanzas, Comercio Internacional,  
Recursos Humanos, Ventas o Economía Digital.



**INFORMATE EN GRANADA**  
958 222 914 | esgerencia.com

## 4<sup>a</sup> Escuela de Negocios de mayor prestigio de España.



- Recorrer los triángulos en sentido antihorario
- Recorrer los triángulos en sentido horario

a) El modelo tendrá como mínimo 14 vértices.

(El punto A tiene tres vértices considerados de textura, pero el F tiene 2, por eso hay que duplicarlo).

b) Dado:: Dado () {

vertices = {  
 2, 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 } // F 5  
 { 0, 0, 0 } // G 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
 }

cc\_tt\_ver = {  
 { 2, 0, 5, 1, 2 }, // A  
 { 0, 5, 2, 3, 4 }, // B  
 { 0, 7, 5, 1, 3 }, // C  
 { 0, 7, 5, 2, 3 }, // D  
 { 0, 2, 5, 1, 3 }, // E  
 { 0, 5, 0, 4, 1 }, // F  
 { 0, 5, 2, 4, 1 }, // G  
 { 0, 2, 5, 2, 3 }, // H  
 { 1, 1, 3, 4 }, // A  
 { 0, 7, 5, 0, 4 }, // B  
 { 2, 0, 1, 2 }, // C  
 { 0, 1, 7, 5, 2 }, // D  
 { 0, 5, 2, 4, 1 }, // E  
 { 2, 0, 2, 3 }, // F  
 { 0, 2, 0, 2, 3 }, // G  
 { 0, 0, 2, 1, 3 }, // H
 }

triangulos = {  
 { 0, 1, 2 } // Cara 1  
 { 1, 3, 2 } // Cara 2  
 { 3, 1, 6 } // Cara 2  
 { 6, 1, 3 } // Cara 3  
 { 0, 4, 7 } // Cara 3  
 { 0, 7, 1 } // Cara 4  
 { 2, 3, 12 } // Cara 4  
 { 2, 12, 8 } // Cara 5  
 { 0, 2, 5 } // Cara 5  
 { 5, 2, 9 } // Cara 6  
 { 4, 10, 7 } // Cara 6  
 { 10, 13, 7 } // Cara 7
 }

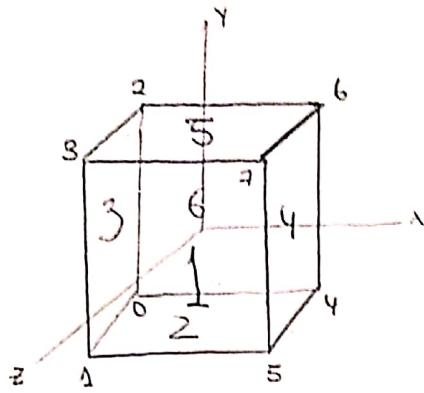
};

NodoDado:: NodoDado () {  
 Textura \* text = new Textura ("drc.jpg");  
 Material \* mat = new Material (.2, .2, .2, .4);  
 agregar (mat);  
 agregar (new Dado);  
} E

(44)

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

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(): mallahd ("Dado 24")

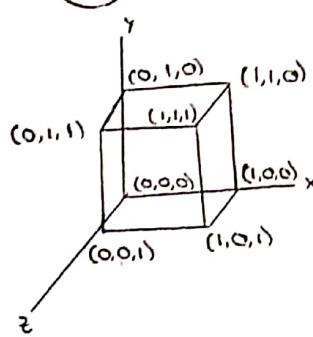
vertices = {  
 {2,-1,-1,-1} //0, 8, 16  
 {-1,-1,+1} //1, 9, 17  
 {-1,+1,-1} //2, 10, 18  
 {-1, 1, 1} //3, 11, 19  
 {1,-1,-1} //4, 12, 20  
 {1,-1, 1} //5, 13, 21  
 {1, 1,-1} //6, 14, 22  
 {1, 1, 1} //7, 15, 23  
 ...  
 };

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

NORMALES DE LOS VERTICES:

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

(45) (Cubo 24)



cc-tt-ver2

220,24 //0

221,24 //1

20,04 //2

21,04 //3

22,24 //4

20,24 //5

22,04 //6

20,04 //7

20,04 //8

22,04 //9

22,04 //10

20,04 //11

20,24 //12

22,24 //13

22,24 //14

20,24 //15

22,24 //16

20,24 //17

22,04 //18

20,04 //19

20,24 //20

22,24 //21

20,04 //22

22,04 //23

8;

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

vertices = { 20,0,04 //0

20,0,14 //1

20,1,04 //2

20,1,14 //3

21,0,04 //4

21,0,14 //5

21,1,04 //6

21,1,14 //7

20,0,04 //8

20,0,14 //9

20,1,04 //10

20,1,14 //11

21,0,04 //12

21,0,14 //13

21,1,04 //14

21,1,14 //15

20,0,04 //16

20,0,14 //17

20,1,04 //18

20,1,14 //19

21,0,04 //20

21,0,14 //21

21,1,04 //22

21,1,14 //23

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

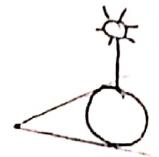
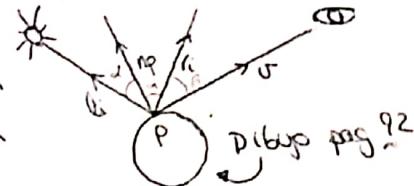
$$C_i = M_s(p) + M_d(p) \max(0, n \cdot l_i) + M_s(p) \cdot d_i [\max(0, n \cdot v)]^e$$

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

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

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)

La reflectividad pseudo-especular del material se obtiene:

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

$d_i$  es 1 si  $n_p \cdot l_i > 0 \Rightarrow$  el punto de máxima iluminación

está en el ~~hemisferio~~ hemisferio norte.

$M_s(p)$  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,

y eso es decir, cuando  $u_i = n_p$ .

es decir, cuando  $u_i = n_p$

$$u_i = \frac{\vec{l}_i + \vec{v}}{\|\vec{l}_i + \vec{v}\|} \text{ bisectriz de } \vec{l}_i \text{ y } \vec{v}$$

$$\vec{l}_i = \frac{\vec{p} - \vec{q}_i}{\|\vec{p} - \vec{q}_i\|} = \frac{(-\vec{p}_x, 2 - \vec{p}_y, -\vec{p}_z)}{\|(-\vec{p}_x, 2 - \vec{p}_y, -\vec{p}_z)\|}$$

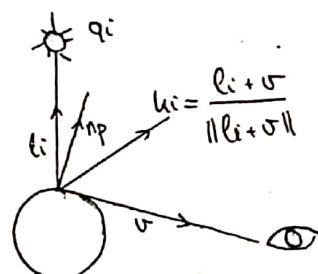
$$\vec{v}_i = \vec{p} - \vec{q}_i = \frac{(2 - \vec{p}_x, -\vec{p}_y, -\vec{p}_z)}{\|(2 - \vec{p}_x, -\vec{p}_y, -\vec{p}_z)\|}$$

$$\text{Tomo } \vec{q} = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0\right), n_p = \vec{n}_p = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0\right)$$

$$\vec{l}_i = \left(-\frac{\sqrt{2}}{2}, 2 - \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right) / \sqrt{63} \quad \Rightarrow \quad u_i = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0\right) \Rightarrow u_i = n_p$$

$$\vec{v}_i = \left(2 - \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}\right) / \sqrt{63}$$

Por tanto  $\vec{p}$  es un punto de máxima iluminación.





(46) b) PARA SABER SI UN PUNTO ES VIABLE:

Tendremos la recta que pasa por el punto y por el observador, y la intersección con la esfera.

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

do recto que une  $P = \left(\frac{12}{2}, \frac{12}{2}\right)$  con  $(2,0)$  es:

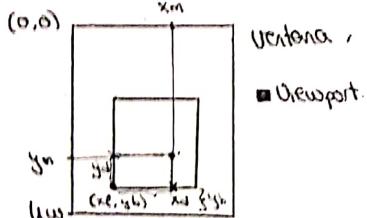
$$y = \frac{-\sqrt{2}}{4-\sqrt{2}}(x-2), \text{ luego:}$$

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

TGMD 4

(47) Coordenadas de dispositivos:

Son aquellas relativas al viewport. Son coordenadas en función de los pixeles.



al viewport.  
Pasamos las coordenadas en función de la ventana (la ventana superior 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.

$$x_d = x_m - x_l$$

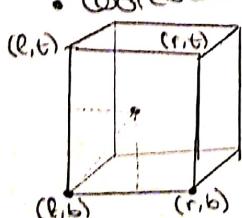
$$\begin{aligned} x_d &= x_m - x_l \\ y_d &= y_w - y_m - y_b \quad (\text{en pixeles}) \end{aligned} \quad \begin{array}{l} (\text{entre } 0 \text{ y } 1, \text{ sin}) \\ \text{ancho de dispositivo:} \end{array}$$

- Coordenadas normalizadas de dispositivo

• Normalizamos las coordenadas de los vértices de cada triángulo

$$x_{ndc} = x_d / w$$

Conferencias de mundo:



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

$$x_w = \ell + \underbrace{x_{ndc}}_{\text{proximal archio}} (r - \ell) \underbrace{\text{arcio}}_{\text{total}}$$

$$y_w = b + \underbrace{y_{dec}}_{\text{proporción}} \underbrace{(t-b)}_{\text{alto total}}$$

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

## TEMAS 5

48) (Pag 62)

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

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

$\Rightarrow$  no hay intersección.

s: no es paralelo  $\Rightarrow$  interseca con el plano, hay que ver si: la intersección cae dentro o fuera del triángulo.

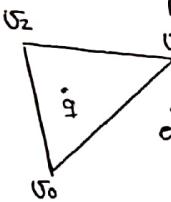
s: la intersección cae dentro o fuera del triángulo.

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  $R[\vec{v}_1 - \vec{v}_0, \vec{v}_2 - \vec{v}_0, \vec{n}, \vec{v}_0]$ , en este sistema 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.

$O_R = (O_x, O_y, O_z)$   $d_R = (d_x, d_y, d_z)$ . Luego, (\*) quedará:

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

$$\begin{cases} O_x + t d_x = a \\ O_y + t d_y = b \\ O_z + t d_z = 0 \end{cases} \Rightarrow$$

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

2) Si  $d_z \neq 0$ :

$$t = \frac{-O_z}{d_z}, \quad a = O_x + \frac{-O_z}{d_z} d_x, \quad b = O_y + \frac{-O_z}{d_z} d_y.$$

El rayo interseca con el triángulo si:

$$-a + b \leq 1$$

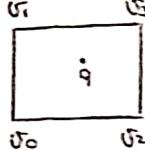
$$-a \geq 0, b \geq 0$$

$$-t > 0$$

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

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

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

La intersección del rayo con la esfera es la solución

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

$$\Rightarrow (ox + tdx - cx)^2 + (oy + tdy - cy)^2 + (oz + tdz - cz)^2 - r^2 = 0$$

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

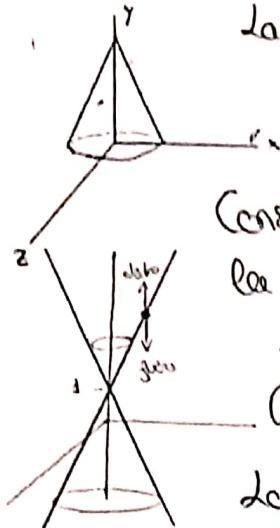
Vemos si  $\exists t > 0$  que sea completa

• Si hay 2 soluciones: tomamos 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:

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

$0 \leq y \leq 1 \wedge \sqrt{x^2 + z^2} \leq 1, y = 0$

báse

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

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

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

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

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

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, calculamos 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  $t_0 > 0$ , intersección con el plano  $y=0$

En otro caso, habrá intersección con la tapa

$$\text{Si } (0x+tdx)^2 + (0z+tdz)^2 \leq 1. (x^2+z^2 \leq 1)$$

Si  $(0x+tdx)^2 + (0z+tdz)^2 > 1$

De las soluciones obtenidas, la intersección del rayo

con el objeto será la más cercana al observador.

(la que tiene el  $t$  menor)

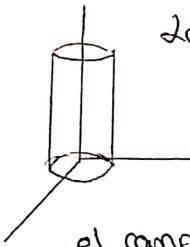
## AHORA

Tu Campus de ESIC en Granada.

MBAs, Marketing, Finanzas, Comercio Internacional,  
Recursos Humanos, Ventas o Economía Digital.INFORMATE EN GRANADA  
958 222 914 | esgerencia.com

(51)

- Cilindro: habrá como máximo 2 intersecciones



La ecuación de nuestro cilindro es:

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

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

Consideramos el cilindro infinito  $x^2 + z^2 = 1$  yel 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  $O+td$  con el cono infinito es la solución de la ecuación:

$$(Ox + tdx)^2 + (Oz + tdz)^2 - 1 = 0 \text{ tales que } t > 0.$$

De estas soluciones nos quedamos con las que  $Oy + tdy \in [0, 1]$ , que son las que intersecan 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  $dy = 0$ : nunca corta las tapas (paralelo a estas)- Si  $dy \neq 0$ :Tapa inferior: Está en el plano  $y=0$ . Despejo de  $Oy + tdy = 0$ .Tomo  $to = \frac{-Oy}{dy}$ . Si  $to < 0$ , no corta la tapa.

En otro caso, habrá intersección con la tapa si:

$$(Ox + tdx)^2 + (Oz + tdz)^2 \leq 1$$

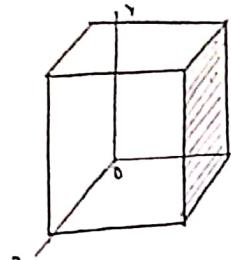
Tapa superior: Está en el plano  $y=1$ . Despejo de  $Oy + tdy = 1$ :Tomo  $to = \frac{1-Oy}{dy}$ . Si  $to < 0$ , no corta la tapa.

En otro caso, habrá intersección con la tapa si:

$$(Ox + tdx)^2 + (Oz + tdz)^2 \leq 1$$

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

- **Cubo:** Consideramos el cubo de lado  $a$  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.

Tomamos por ejemplo la cara  $\square$ , situada en el plano  $x=1$ . Buscamos las soluciones de  $Ox + tdx = 1$ , 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  $t_0 = \frac{1-Ox}{dx}$ . 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:

$$-1 < Oy + t_0 dy < 1 \quad -1 < Oz + t_0 dz < 1$$

$-1 < Oy + t_0 dy < 1$  y  $-1 < Oz + t_0 dz < 1$ .

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

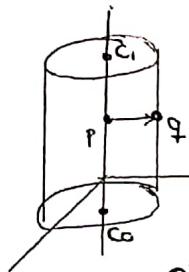
De las soluciones encontradas, nos quedamos con aquella con  $t$  menor.

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:

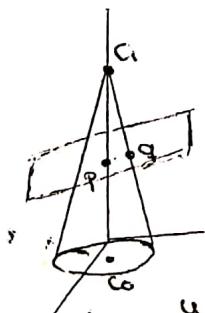
Tomo el plano paralelo a las tapas que pasa por  $q$ . Tomo el punto  $p$ : corte de ese plano con el eje del cilindro. La normal es el vector  $\vec{pq}$  normalizado.

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  $C_1 - C_0 (\vec{C_0C_1})$  normalizado.
  - Si está en la tapa de abajo, la normal es  $C_0 - C_1 (\vec{C_1C_0})$  normalizado.

### • Cono:



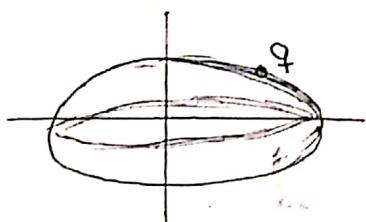
- Si  $q$  está en la tapa:
  - su normal es  $C_0 - C_1 (\vec{C_1C_0})$  normalizado.

- Si  $q$  no está en la tapa:

Calculo el plano perpendicular a  $\vec{qC_1}$  que pasa por  $q$  (el que tiene a  $\vec{qC_1}$  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 trasformaciones 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.

⑤ 4) (Tema 3, pag 22 y 23)

### Rasterización:

para cada primitiva  $P \neq n$

encontrar el conjunto  $S$  de pixeles  $\sim P$

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

### Raytracing:

para cada pixel  $q \neq n$

calcular  $T$ , conjunto de primitivas  $\sim n \quad \} \max(n, n) = n$

para cada primitiva  $\sim n$ .

Complejidad  $O(n \cdot p)$ , pero con indexación especial,

supuestamente para cálculo de  $T$  en cada pixel,

supuestamente para cálculo de  $T$  en cada pixel,

supuestamente conseguimos  $O(p \log(n))$ .

supuestamente conseguimos  $O(p \log(n))$ .