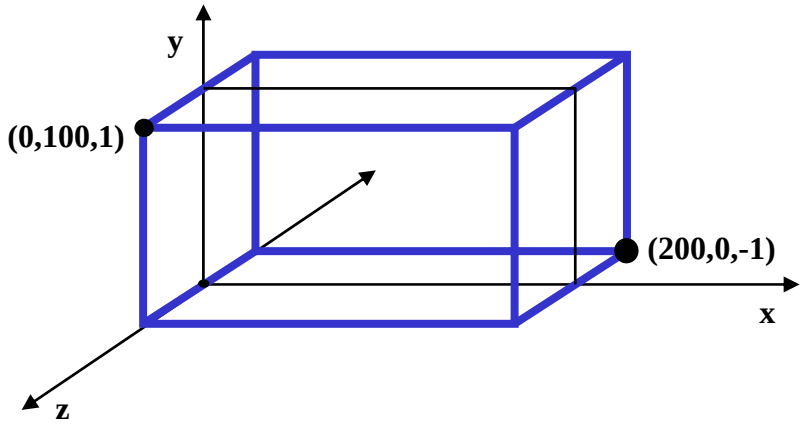
# Proyecciones

# Proyecciones

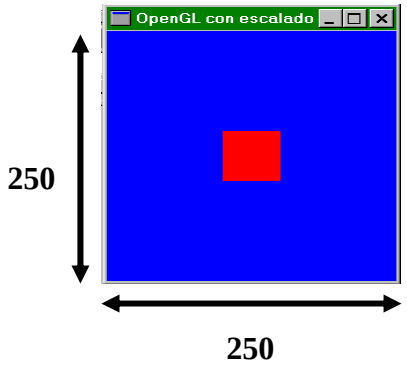
- La matriz de proyección especifica el tamaño y forma del volumen de visualización

```
glMatrixMode(GL_PROJECTION);
// reseteamos la matriz
glLoadIdentity();
// actualizamos la matriz
glOrtho(0,200,0,100,1,-1);
```




- La matriz de transformación a ventana mapea la foto final sobre el área de pantalla especificada

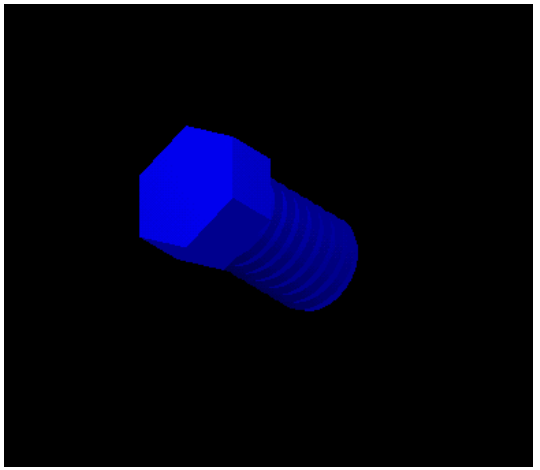
```
glViewport(0, 0, 250, 250);
```



# Orden de las matrices

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \rightarrow \begin{pmatrix} M_m \\ \text{Matriz de modelado} \end{pmatrix} \rightarrow \begin{pmatrix} x_m \\ y_m \\ z_m \\ w_m \end{pmatrix} \rightarrow \begin{pmatrix} M_p \\ \text{Matriz de proyección} \end{pmatrix} \rightarrow \begin{pmatrix} x_p \\ y_p \\ z_p \\ w_p \end{pmatrix} \rightarrow$$

$$\rightarrow \begin{pmatrix} M_v \\ \text{Matriz de mapeo} \end{pmatrix} \rightarrow \begin{pmatrix} x_v \\ y_v \\ z_v \\ w_v \end{pmatrix} \rightarrow$$



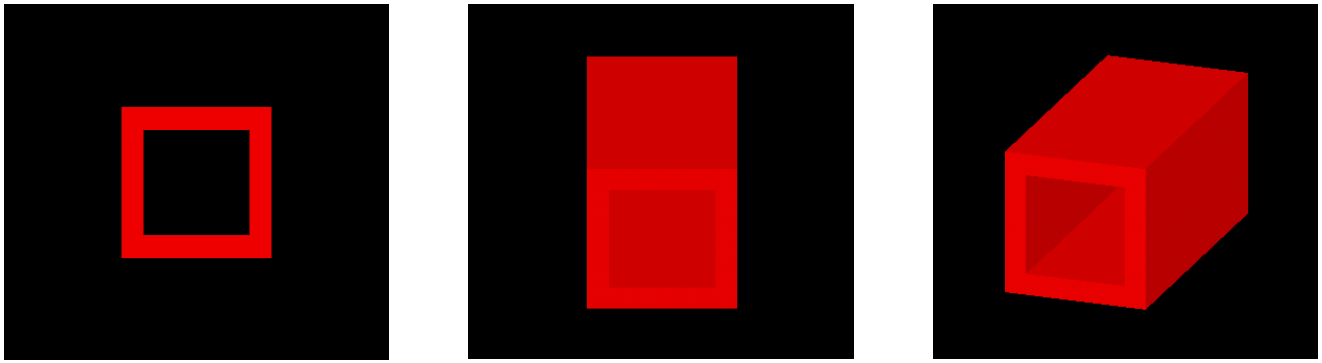
$$P' = M_v \cdot M_p \cdot M_m \cdot P$$

```
void CALLBACK EscalaVentana(GLsizei w, GLsizei h)
{
   // Matriz de mapeo de ventana
   glViewport(0, 0, w, h);

   // Matriz de proyección
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0,250,0,250,1,-1);
}
```

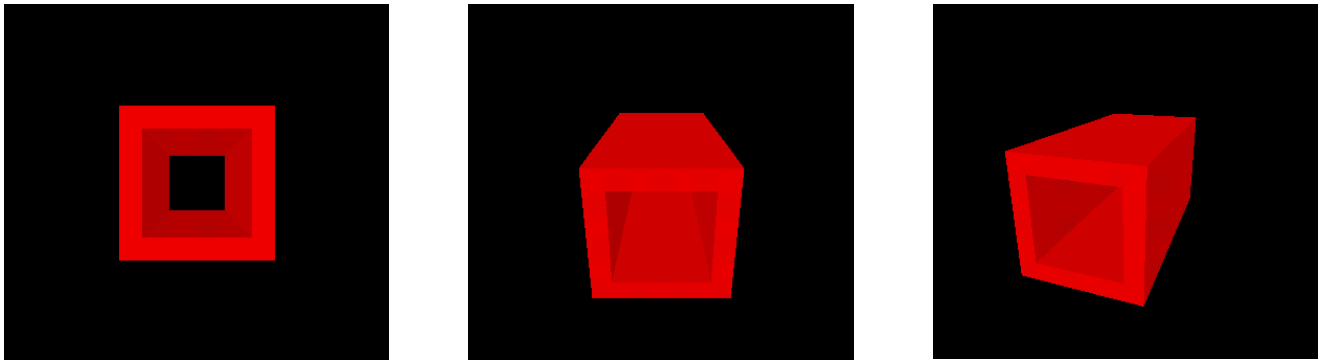

```
void CALLBACK DibujaEscena()
{
   // Matriz de modelado
   glClear (GL_COLOR_BUFFER_BIT);
   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity();
   glRotatef(angulo,0,1,0);
   glTranslatef(-70,0,0);
   ...
}
```

# Proyección paralela / perspectiva

- Proyección paralela



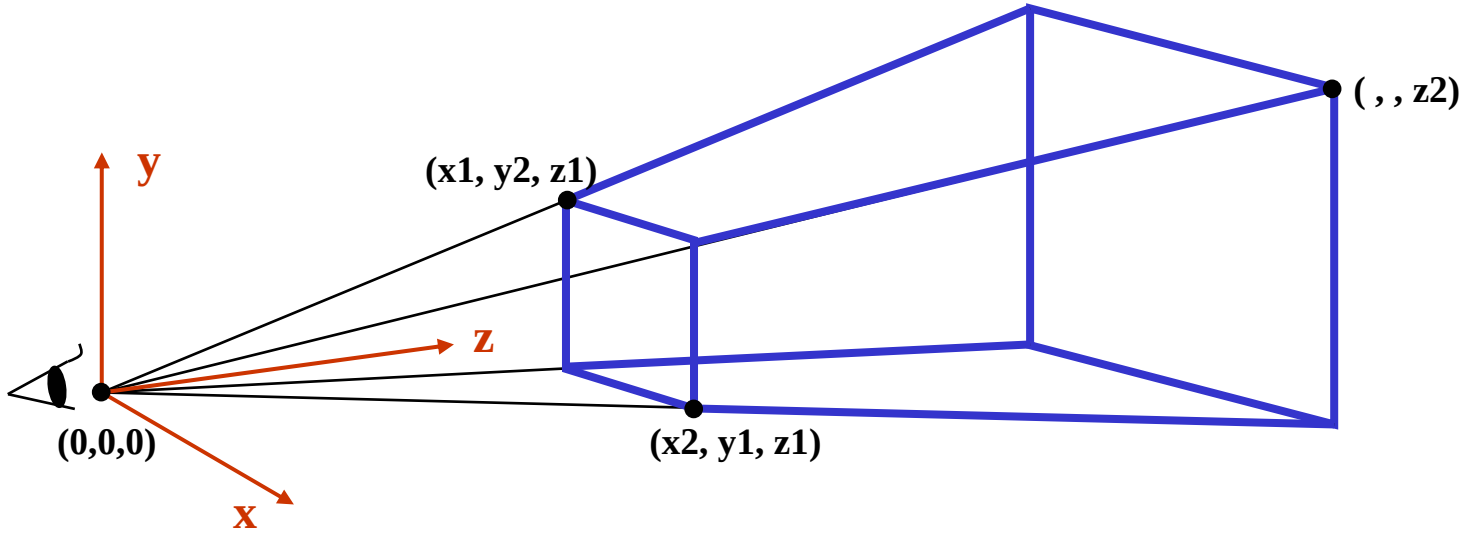
- Proyección perspectiva



# Proyección perspectiva en OpenGL

- El ojo siempre está en el origen
- La dirección de vista es la que une el ojo con el centro de la ventana

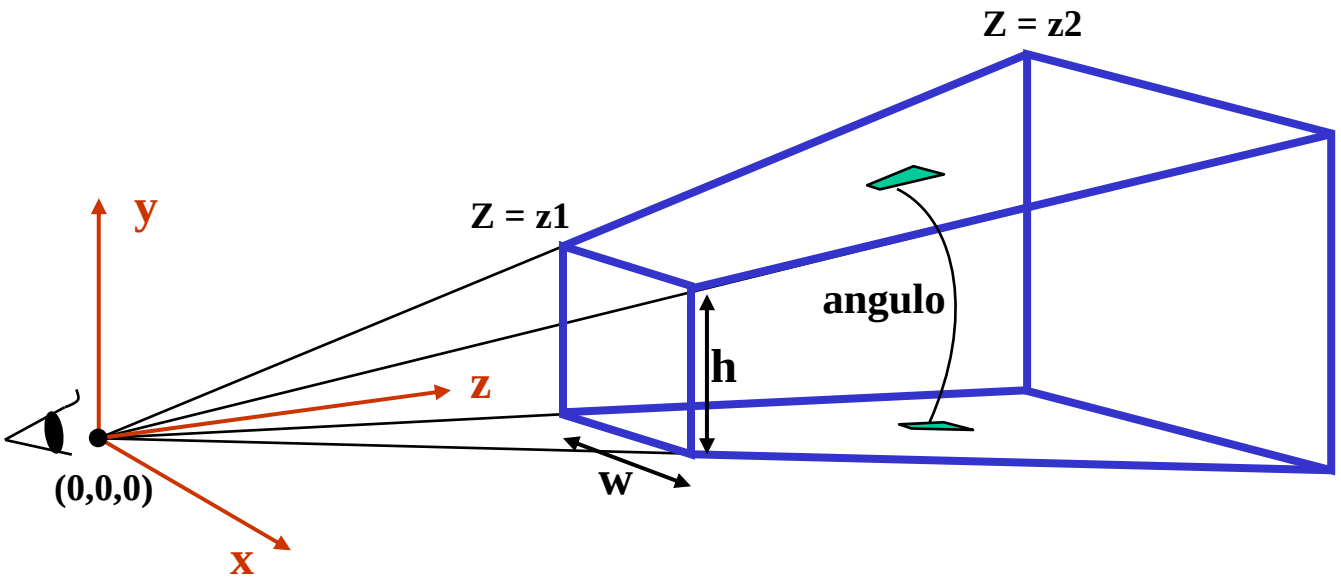
```
void glFrustum(GLdouble x1, GLdouble x2, GLdouble y1,
               GLdouble y2, GLdouble z1, GLdouble z2);
```



# Proyección perspectiva usando la GLU

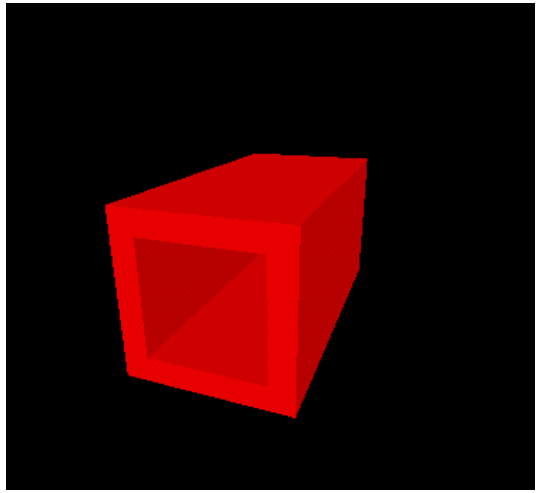
- Es otra forma alternativa de especificar el volumen de visualización
- El ojo sigue estando en el origen
- La dirección de vista coincide con el eje z

```
void gluPerspective(GLdouble angulo, GLdouble aspecto,
                    GLdouble z1, GLdouble z2);
```



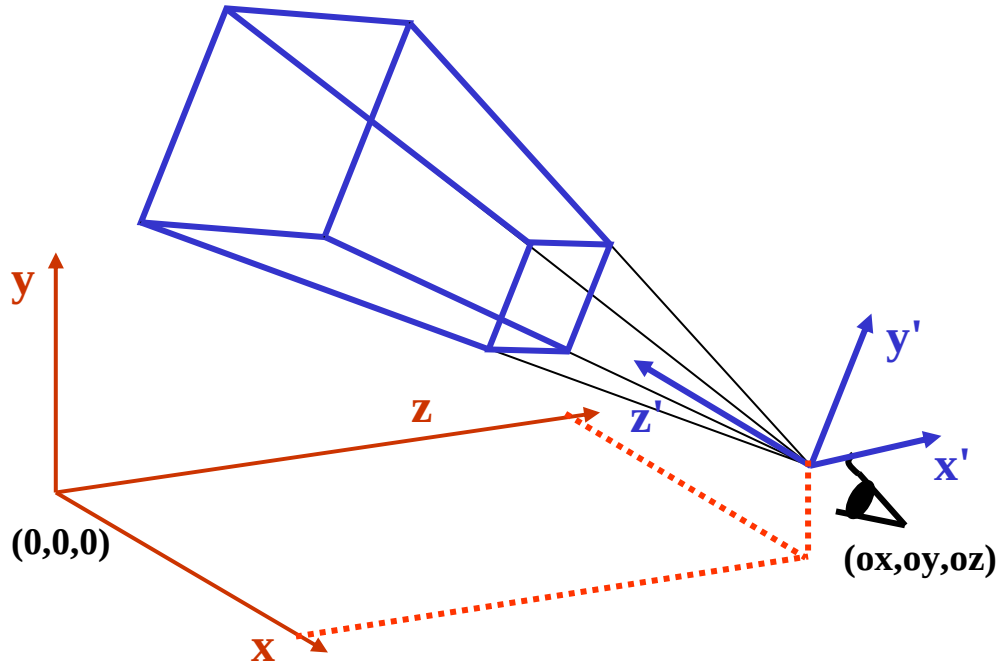

**aspecto = w / h**

```
void CALLBACK EscalaVentana(GLsizei w, GLsizei h)
{
   // Evita una división por cero
   if (h == 0) h = 1;

   // Ajusta la vista a las dimensiones de la ventana
   glViewport(0, 0, w, h);

   // Reinicia la matriz de proyección
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();

   // Establece el volumen de trabajo
   aspecto = (GLfloat) w / h;
   gluPerspective (60,aspecto,1,400);
}
```



# Transformación de vista

- Una vez definido el volumen de visualización, el ojo se encuentra en el origen, mirando en la dirección negativa del eje z

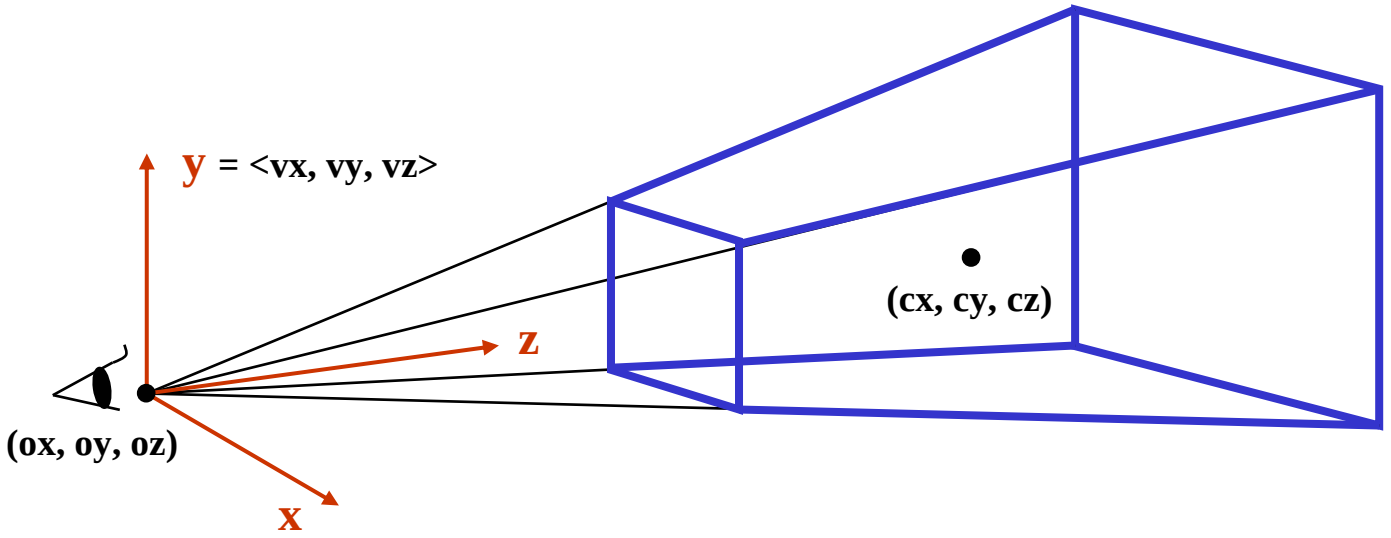



- Para obtener la proyección desde otra posición y desde otra dirección de vista, hay que especificar una transformación de vista

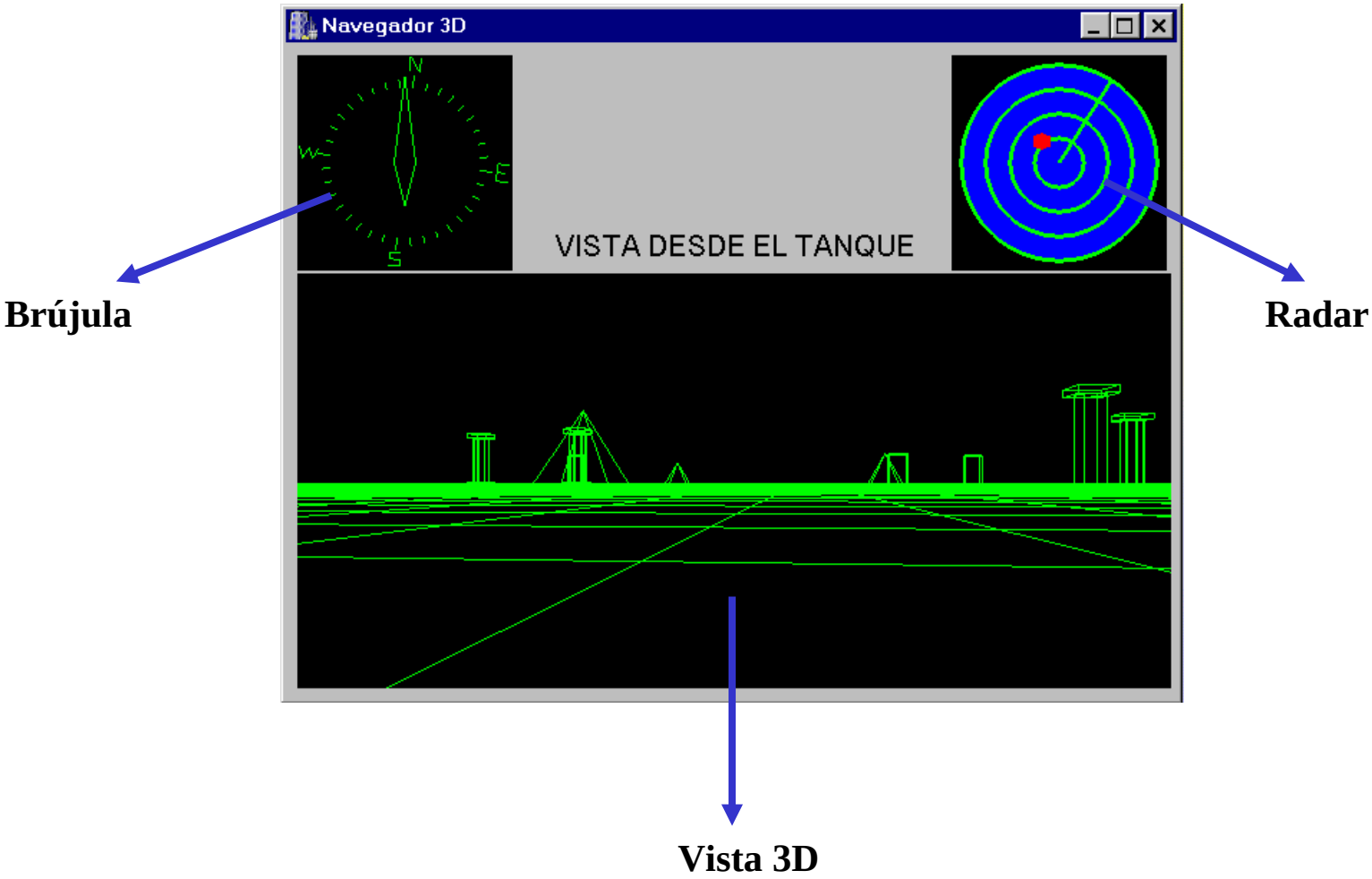
# Transformación de vista en OpenGL

- Para definir la transformación de vista se usa:

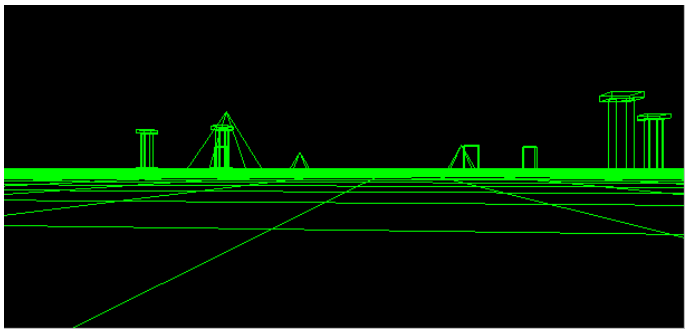
```
void gluLookAt(GLdouble ox, GLdouble oy, GLdouble oz,
               GLdouble cx, GLdouble cy, GLdouble cz,
               GLdouble vx, GLdouble vy, GLdouble vz);
```



# Ejemplo: Implementación de un navegador 3D

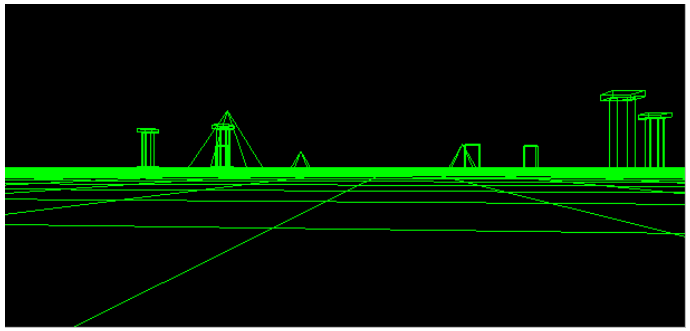


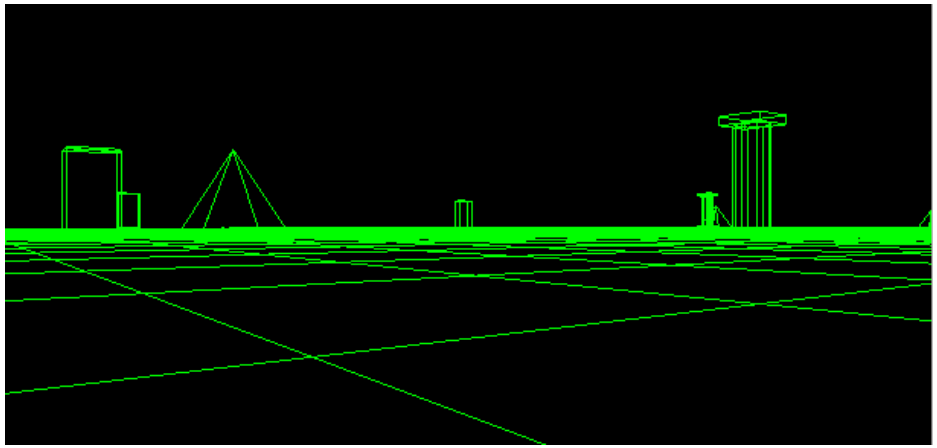
# Panel de vista



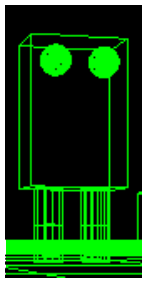
```
void PanelVista_Resize(TObject *Sender)
{
   GLsizei w = PanelVista->Width;
   GLsizei h = PanelVista->Height;
   GLdouble aspecto;

   // Mapeo de la ventana
   glViewport(0, 0, w, h);

   // Inicialización de la matriz de proyección
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   aspecto = (GLdouble) w/h;
   gluPerspective(35.0, aspecto, 1, 2000);

   // Inicialización de la matriz de modelado
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

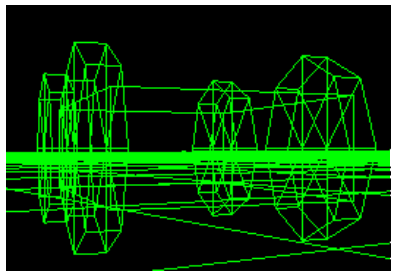
```
// estructura de datos del tanque y del robot
struct OBJETO {
   GLfloat x,z;            // posiciones x y z del objeto
   Gldouble ori;           // ángulo de orientación del objeto
 } tanque, robot, viewer;


void PanelVista_Paint(TObject *Sender)
{
   glPushMatrix();
   glClear(GL_COLOR_BUFFER_BIT);

   // Transformación de vista
   gluLookAt (viewer->x, 5.0f, viewer->z,
         viewer->x+cos(viewer->ori), 5, viewer->z-sin(viewer->ori),
         0.0f, 1.0f, 0.0f);

   // dibujamos todos los objetos
   RenderWorld();
   glPopMatrix();
}
```





```
// Dibuja la escena completa
void RenderWorld()
   {
   // Dibuja todo en verde
   glColor3f(0,255,0);

   // Dibuja el suelo
   DrawGround();           // usando GL_LINES

   // Coloca los objetos en posiciones aleatorias ya precalculadas
   PlacePyramids();        // usando GL_TRIANGLES
   PlaceSlabs();           // usando GL_QUADS
   PlacePillers();         // usando GL_QUAD_STRIP

   // Dibuja el objeto contrario al que seamos
   if (viewer == tank)
       DrawRobot();        // usando auxSphere, auxBox y auxCilynder
       else DrawTank();    // usando GL_TRIANGLE_FAN y GL_QUADS
   }
```

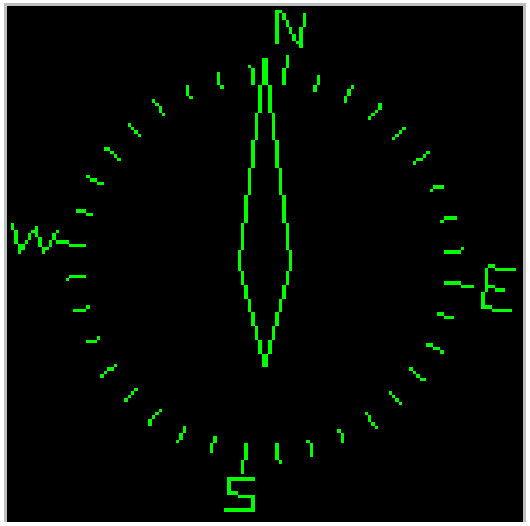




```
void Form1_KeyDown(TObject *Sender, WORD &Key, TShiftState Shift)
{
   switch (Key) {
       case VK_UP:         MoveViewer(10.0); break;
       case VK_DOWN:       MoveViewer(-10.0); break;
       case VK_LEFT:       viewer->ori += PI/30;
                           if(viewer->ori>2*PI) viewer->ori = 0.0;
                           break;
       case VK_RIGHT:      viewer->ori -= PI/30;
                           if(viewer->ori<0.0) viewer->ori = 2*PI;
                           break;

       case T:             viewer = tanque;
                           Label1->Caption = "VISTA DESDE EL TANQUE";
                           break;
       case R:             viewer = robot;
                           Label1->Caption = "VISTA DESDE EL ROBOT ";
                           break;
       default: return;
     }

   // repintamos la vista y la brújula
   PanelVista->Paint();
   PanelBrujula->Paint();
}
```

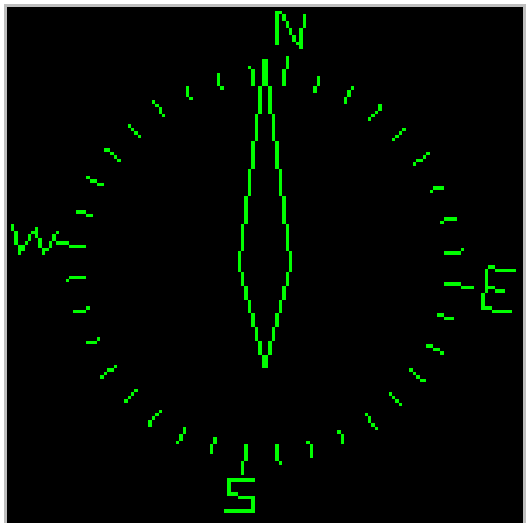
```
// Mueve el tanque (o el robot) hacia delante o hacia atrás
void MoveViewer(GLdouble paso)
   {
   GLdouble xDelta, zDelta;

   /* calcula cuánto hay que avanzar en x y en z, según la
            orientación del objeto */
   xDelta = paso * cos(viewer->ori);
   zDelta = -paso * sin(viewer->ori);

   // actualiza las coordenadas del objeto
   viewer->x += (GLfloat)xDelta;
   viewer->z += (GLfloat)zDelta;

   // comprueba que no nos salgamos de los límites del mundo
   if(viewer->x > 1000.0f)  viewer->x = 1000.0f;
   if(viewer->x < -1000.0f) viewer->x = -1000.0f;
   if(viewer->z > 1000.0f)  viewer->z = 1000.0f;
   if(viewer->z < -1000.0f) viewer->z = -1000.0f;
}
```

# Panel de la brújula



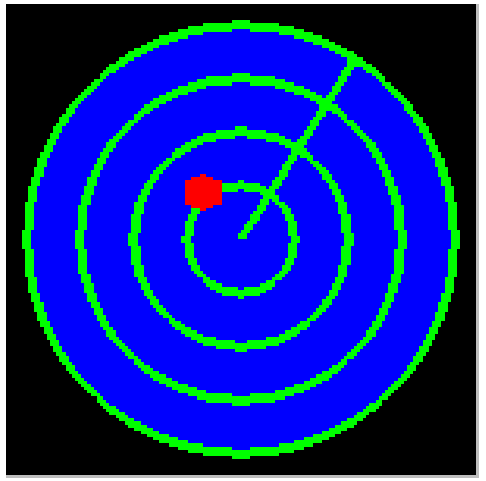

```
void PanelBrujula_Resize(TObject *Sender)
{
   GLsizei w = PanelBrujula->Width;
   GLsizei h = PanelBrujula->Height;

   // Mapeo de la ventana
   glViewport(0, 0, w, h);

   // Inicialización de la matriz de proyección
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-1.0, 1.0, -1.0, 1.0);

   // Inicialización de la matriz de modelado
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

```
void PanelBrujula_Paint(TObject *Sender)
{
   GLfloat grados;

   glClear(GL_COLOR_BUFFER_BIT);
   glPushMatrix();

   // pintamos la aguja, siempre fija en vertical
   DrawAguja();    // usando GL_QUADS

   // convertimos el ángulo de orientación del objeto a grados
   grados = (GLfloat)(viewer->ori) * 180/PI;

   // rotamos la brújula
   grados = 90.0 - grados;
   glRotatef (grados, 0, 0, 1);

   // pintamos el interior de la brújula
   DrawLetras();   // usando GL_LINE_STRIP
   DrawTicks();    // usando GL_LINES

   glPopMatrix();
   glFlush();
}
```

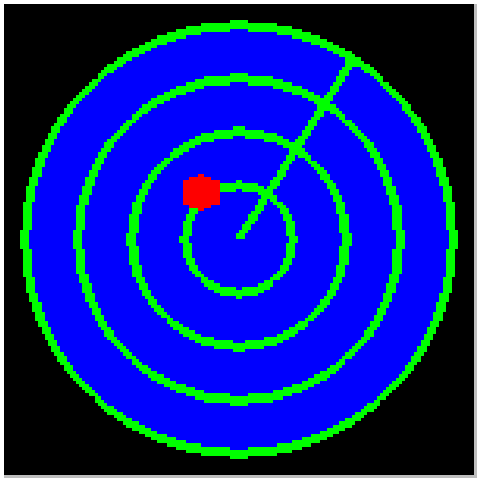


# Panel del radar

```
void PanelRadar_Init(TObject *Sender)
{
   // líneas gruesas
   glLineWidth(3);
}


void PanelRadar_Resize(TObject *Sender)
{
   GLsizei w = PanelRadar->Width;
   GLsizei h = PanelRadar->Height;

   // Mapeo de la ventana
   glViewport(0, 0, w, h);

   // Inicialización de la matriz
   glLoadIdentity();
   gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
}
```



```
GLfloat palito=2*PI;     // indica el ángulo del palito del radar

void Timer_Timer(TObject *Sender)
{
   palito-=PI/50;
   if (palito<0) palito=2*PI;
   PanelBrujula->Paint();
}

void PanelRadar_Paint(TObject *Sender)
{
   glClear(GL_COLOR_BUFFER_BIT);

   //dibujamos el radar
   glColor3f(0,0,1);
   DrawBlueCircle();     // usando GL_POLYGON
   glColor3f(0,1,0);
   DrawAros();           // usando GL_LINE_LOOP
   DrawPalito();         // usando GL_LINES
   glColor3f(1,0,0);
   DrawPuntoRojo();      // usando GL_POLYGON

   glFlush();
}
```



```
void DrawPuntoRojo()
{
   GLfloat nCenterX, nCenterY;
   double distancia, xDelta, zDelta, angulo;
   struct OBJETO elotro;

   // miramos quién debe ser el punto rojo
   if (viewer == tanque) elotro=robot; else elotro=tanque;

   // calculamos dónde está el otro y a qué distancia
   xDelta = elotro->x - viewer->x;
   zDelta = elotro->z - viewer->z;
   distancia = sqrt((xDelta*xDelta) + (zDelta*zDelta))/10.0;

   // calculamos la posición exacta del otro en el radar
   angulo = atan2(zDelta,xDelta) + viewer->ori-(PI/2.0);
   nCenterX = (GLfloat) (cos(angulo)*distancia)/150;
   nCenterY = (GLfloat) (sin(dAngle)*distancia)/150;

   // pintamos el punto
   glPushMatrix();
   glTranslatef (nCenterX, nCenterY, 0);
   auxSolidIcosahedron(0.1);
   glPopMatrix();
}
```