

tecnunLOGO

TRABAJO DE PROGRAMACIÓN EN OpenGL | Fran Pastor

ÍNDICE

1. INTRODUCCIÓN A OPENGL: DIBUJANDO UNA TORTUGA CON OPENGL	4
MODIFICACIÓN DE GL VIEWPORT PARA QUE NO SE DEFORME.....	4
DIBUJAR LOS EJES DEL SISTEMA DE COORDENADAS DE LA VENTANA UTILIZANDO LOS COLORES ROJO, VERDE Y AZUL (RGB) PARA LOS EJES X, Y, Z CORRESPONDIENTEMENTE.....	4
DIBUJAR EN LA VENTANA LAS DIFERENTES PRIMITIVAS DE GLUT	4
INTRODUCIR UN COMANDO NUEVO DE MANERA QUE AL APRETAR LA TECLA 'A' (AXIS), SE MUESTREN LOS EJES DE LA FIGURA SI ESTÁN DESACTIVADOS, O SE DESACTIVEN SI ESTÁN ACTIVADOS.	4
REALIZAR LA FUNCIÓN DRAW TURTLE() QUE DIBUJA POR MEDIO DE LA FUNCIÓN GLBEGIN() UNA TORTUGA.....	5
REALIZAR LA FUNCIÓN DRAW SPHERE TURTLE() QUE DIBUJA UNA TORTUGA MEDIANTE LA PRIMITIVA DE LA FUNCIÓN GLUT WIRE SPHERE (). ESTA FUNCIÓN TIENE COMO ARGUMENTOS EL RADIO Y LA PRECISIÓN EN LATITUD Y LONGITUD.....	5
2. TRANSFORMACIONES: DANDO ÓRDENES A LA TORTUGA	5
IMPLEMENTAR LAS INSTRUCCIONES RIGHTROLL Y LEFTROLL CON LAS ABREVIATURAS RR Y LR, QUE REALIZAN UN GIRO ALREDEDOR DEL EJE "Z".....	5
INCLUIR EN LA APLICACIÓN LAS INSTRUCCIONES PARA CAMBIAR EL TAMAÑO DEL OBJETO: SCALEX, SCALEY Y SCALEZ, CON LAS ABREVIATURAS SX, SY, SZ. EL ARGUMENTO QUE SE PROPORCIONA ES EL FACTOR DE ESCALA. PARA DISMINUIR EL TAMAÑO BASTA EMPLEAR UN FACTOR MENOR A LA UNIDAD.....	5
DIBUJAR UNOS EJES DEL MUNDO Y OTROS EN EL SISTEMA DE LA TORTUGA.....	6
UTILIZANDO LOS COMANDOS DE LOGO DAR LAS INSTRUCCIONES PARA QUE LA TORTUGA SE DESPLAZE EN UNA CIRCUNFERENCIA.....	6
LEER COMANDOS DESDE UN FICHERO, LA INSTRUCCIÓN SE DEFINE "LOAD NOMBRE FICHERO", EL FICHERO CONTIENE 1 O VARIAS LÍNEAS DE SENTENCIAS DE LOGO.	6
DEFINIR EL SISTEMA DE MEDIDA DE ÁNGULOS, DE FORMA QUE EN VEZ DE UTILIZAR EL SISTEMA SEXAGESIMAL DE 0 A 360°, SE PUEDA UTILIZAR CUALQUIER OTRO; COMO RADIANES DE 0 A 2 π ; GRADOS	

CENTESIMALES, DE 0 A 400 O CUALQUIER OTRO COMO DE 0 A 12 Ó 0 A 60 COMO LAS HORAS O MINUTOS DE UN RELOJ. UTILIZAR PARA ELLO EL COMANDO SETCIRCLE NN, SIENDO NN EL NÚMERO CORRESPONDIENTE A LA MEDIDA DE UN GIRO COMPLETO. POR DEFECTO EL VALOR ES 360. LA ABREVIATURA DEL COMANDO ES SC.	6
IMPLEMENTAR LOS COMANDOS HIDETURTLE Y SHOWTURTLE, CON LAS ABREVIATURAS HT Y ST QUE MUESTRAN U OCULTAN LA TORTUGA.	7
3. OPERACIONES CON MATRICES: DIBUJANDO EL CAMINO.....	7
DIBUJAR UN RASTRO QUE CONSISTA EN UNAS SUPERFICIE EN LUGAR DE UNA LÍNEA. PARA ELLO SE PUEDE UTILIZAR GLBEGIN(GL_QUAD_STRIP) QUE DIBUJA UNA SUCESIÓN DE RECTÁNGULOS PARA CADA PAREJA DE PUNTOS QUE RECIBE.	7
UTILIZANDO LOS COMANDOS DE LOGO REPRESENTAR UNA ESFERA COMPUESTA POR UN CONJUNTO DE CIRCUNFERENCIAS EN EL ESPACIO.....	7
UTILIZANDO LOS COMANDOS DE LOGO REALIZAR LA REPRESENTACIÓN DE UNA HELICOIDAL.....	7
4. VIENDO LA ESCENA DES DE OTRO PUNTO DE VISTA: CÁMARAS	7
DOTAR AL PROGRAMA DE UNA TECLA QUE PERMITA CAMBIAR EL MODO DE PROYECCIÓN ENTRE ORTOGONAL Y PERSPECTIVA.....	7
PROGRAMAR OTROS MODOS DE MOVIMIENTO DE CÁMARA COMO SON EL MODO PAN O EL MODO TRÍPODE.	8
PROGRAMAR OTRO MODO DE MOVIMIENTO DE LA TORTUGA COMO ES EL MODO CAR. PULSANDO EL BOTÓN IZQUIERDO DEL RATÓN LOGRAREMOS EL MOVIMIENTO DESEADO. SI LO DESPLAZAMOS HACIA DELANTE CONDUciremos EN LÍNEA RECTA. EN CAMBIO, SI MOVEMOS EL PUNTERO HACIA LOS LADOS, HAREMOS QUE LA TORTUGA GIRE COMO LO HARÍA UN COCHE.....	9
OPCIÓN DE ELEGIR EL LUGAR DE LA CÁMARA. LA ORDEN “CAM OBJ” Y “CAM ZERO” COLOCARÁN LA CÁMARA DETRÁS DEL OBJETO O EN EL ORIGEN RESPECTIVAMENTE.....	10
PROGRAMAR OTRO MODO DE MOVIMIENTO PARA NUESTRO OBJETO COMO ES EL MODO FLY. AL APRETAR EL BOTÓN IZQUIERDO DEL RATÓN COMIENZA EL MOVIMIENTO EN LÍNEA RECTA Y AL MOVER EL CURSOR HACIA LOS LADOS PRODUCE UN GIRO. AL MOVER EL RATÓN HACIA DELANTE EL OBJETO BAJA Y VICEVERSA COMO OCURRE EN UNA AVIONETA.....	10
PROGRAMAR OTRO MODO DE CÁMARA COMO ES EL MODO FLY EN EL QUE LA CÁMARA VUELA POR EL ESCENARIO. NAVEGAMOS PULSANDO EL BOTÓN IZQUIERDO DEL RATÓN Y GIRAMOS SEGÚN LOS DESPLAZAMIENTOS DE ESTE.	11
MEJORA EN LA POSICIÓN DE LA CÁMARA.....	12
OPERACIÓN DE ATERRIZAJE DEL OBJETO.....	13
5. ILUMINANDO LA ESCENA.....	14
PROPORCIONAR ALGUNA FORMA AMIGABLE DE MODIFICAR EL COLOR DE LA LUZ. POR EJEMPLO MEDIANTE EL TECLADO CON LAS LETRAS (R,G,B PARA DISMINUIR Y R,G,B PARA AUMENTAR).....	14
AÑADIR LA ORDEN “FUNKYCOLORS” QUE GENERA UN CONTROL ALEATORIO EN EL COLOR DE LAS LUCES.....	15
AÑADIR DOS FOCOS AL OBJETO IMITANDO LOS FAROS DE UN VEHÍCULO. ALUMBRARÁN LA ESCENA CUANDO LOS ACTIVEMOS MEDIANTE LA ORDEN “LIGHTS”.....	15
6. MODO JUEGO	17
PROGRAMAR UN MODO LLAMADO GAME DONDE SE HAGA USO DEL TIMER Y HAYA UN CONTROL DE POSICIÓN DE VARIOS OBJETOS AL MISMO TIEMPO EN MOVIMIENTO DENTRO DE LA ESCENA, ASÍ COMO DE LA CÁMARA QUE PERSEGUIRÁ A NUESTRA ‘TORTUGA’	17
7. VISIÓN ESTEREOSCÓPICA.....	20
COMO FINAL A ESTAS PRÁCTICAS, INTENTAR IMPLEMENTAR UNA ORDEN QUE GENERE UNA IMAGEN DE VISIÓN ESTEREOSCÓPICA MEDIANTE EL COMANDO “STEREO”.	20

1. INTRODUCCIÓN A OPENGL: DIBUJANDO UNA TORTUGA CON OPENGL

Modificación de `glViewport` para que no se deforme

Cambiar en la función `glViewport()` la línea:

```
glViewport(0, 0, width, height);
```

por:

```
if(width<=height)
    glViewport(0, 0, width, width);
else
    glViewport(0, 0, height, height);
```

y la línea:

```
gluPerspective(60.0, (GLfloat)height/(GLfloat)width, 1.0, 128.0);
```

por:

```
gluPerspective(60.0, 1.0, 1.0, 128.0);
```

Dibujar los ejes del sistema de coordenadas de la ventana utilizando los colores rojo, verde y azul (RGB) para los ejes x, y, z correspondientemente.

Llamada a una nueva función que dibuja los ejes desde la función `display`:

```
dibujaEjes();
```

Función incorporada:

```
void dibujaEjes(void){
    glBegin(GL_LINES);
        glVertex3f(0.0,0.0,0.0);
        glVertex3d(0.0,0.0,0.05);
        glColor3f(1.0,0.0,0.0);
        glVertex3f(0.0,0.0,0.0);
        glVertex3d(0.05,0.0,0.0);
        glColor3f(0.0,1.0,0.0);
        glVertex3f(0.0,0.0,0.0);
        glVertex3d(0.0,0.05,0.0);
    glEnd();
}
```

Dibujar en la ventana las diferentes primitivas de GLUT

Diferentes pruebas colocando vértices entre dos instrucciones que definen el tipo de geometría que obtendremos en base a la primitiva que elijamos:

```
glBegin(GL_LINE_LOOP);
    glVertex3f(0.0,0.0,0.0);
    glVertex3f(1.0,0.0,0.0);
    glVertex3f(1.0,0.0,1.0);
glEnd();
```

Introducir un comando nuevo de manera que al apretar la tecla 'a' (axis), se muestren los ejes de la figura si están desactivados, o se desactiven si están activados.

Creamos un booleano llamado `ejes` para controlar el estado visible o no de los mismos.

```
GLboolean ejes = TRUE;
```

En la función `keyboard()` se hace la captura del teclado. En caso de que la tecla pulsada sea la 'a' cambiamos el valor del booleano:

```
case 'a':
    if(ejes) ejes = FALSE; else ejes = TRUE;
    break;
```

Finalmente, en la función `display()` llamamos a `dibujaEjes()` según el valor del booleano:

```
if(ejes) dibujaEjes();
```

Realizar la función `drawTurtle()` que dibuja por medio de la función `glBegin()` una tortuga.

```
void drawTurtle(void){

    GLdouble x[] = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 5.0, 4.0, 3.0, 2.0, 3.0, 3.0,
3.0, 2.0, 3.0, 4.0, 5.0, 5.0, 4.0, 3.0, 2.0, 1.0, 2.0, 2.0, 2.0, 2.0,
1.0, 0.0};
    GLdouble z[] = {7.0, 4.0, 4.0, 5.0, 5.0, 4.0, 3.0, 2.0, 2.0, 2.0, 1.0, 0.0, -
1.0, -2.0, -2.0, -2.0, -3.0, -4.0, -5.0, -5.0, -4.0, -4.0, -5.0, -6.0, -7.0, -8.0,
-9.0, -9.0};
    GLint i;
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_LINE_LOOP);
        for (i=0;i<28;i++){
            glVertex3f(x[i]*0.1,0.0,-.1*z[i]);
        }
        for (i=27;i>-1;i--){
            glVertex3f(-.1*x[i],0.0,-.1*z[i]);
        }
    glEnd();
}
```

Realizar la función `drawSphereTurtle()` que dibuja una tortuga mediante la primitiva de la función `glutWireSphere()`. Esta función tiene como argumentos el radio y la precisión en latitud y longitud.

Desde `display()` lanzamos una nueva función llamada `drawSphereTurtle()` que se encarga de la tarea:

```
void drawSphereTurtle(Void){

    GLdouble valx[] = { 0.0, 4.0, -8.0, 0.0, 8.0,-4.0};
    GLdouble valz[] = { -6.0, 2.0, 0.0, 8.0, 0.0,-4.0};
    GLdouble r[] = { 6.0, 3.0, 2.0, 2.0, 2.0, 2.0};
    GLint i;
    glColor3f(0.0,0.0,1.0);
    for (i=0;i<6;i++){
        glutWireSphere(r[i], 10.0, 10.0);
        glTranslatef(valx[i],0.0,valz[i]);
    }
}
```

2. TRANSFORMACIONES: DANDO ÓRDENES A LA TORTUGA

Implementar las instrucciones `RIGHTROLL` y `LEFTROLL` con las abreviaturas `rr` y `lr`, que realizan un giro alrededor del eje "Z".

Añadimos al `if` del `parseCommand` las siguientes instrucciones:

```
else if (!strcmp("lr",strToken0)) { // LEFTROLL
    glRotatef(-val,0.,0.,1.);
} else if (!strcmp("rr",strToken0)) { // RIGTHROLL
    glRotatef(val,0.,0.,1.);
}
```

Incluir en la aplicación las instrucciones para cambiar el tamaño del objeto: `SCALEX`, `SCALEY` y `SCALEZ`, con las abreviaturas `sx`, `sy`, `sz`. El argumento que se proporciona es el factor de escala. Para disminuir el tamaño basta emplear un factor menor a la unidad.

Añadimos al `if` del `parseCommand` las siguientes instrucciones:

```

else if (!strcmp("sx",strToken0)) { // SCALE X
    glScalef(val,1.,1.);
} else if (!strcmp("sy",strToken0)) { // SCALE Y
    glScalef(1.,val,1.);
} else if (!strcmp("sz",strToken0)) { // SCALE Z
    glScalef(1.,1.,val);
}

```

Dibujar unos ejes del mundo y otros en el sistema de la tortuga

Conseguimos el efecto deseado dibujando los ejes antes y después del `glPopMatrix()`:

```

glPushMatrix();
dibujaEjes();
glPopMatrix();
dibujaEjes();

```

Utilizando los comandos de logo dar las instrucciones para que la tortuga se desplace en una circunferencia.

```
Repeat 360 [fd 1 lt 1]
```

Leer comandos desde un fichero, la instrucción se define "LOAD nombreFichero", el fichero contiene 1 o varias líneas de sentencias de logo.

Introducimos la función `readFile(char* nameArchivo)` que ejecutará las ordenes del archivo que nosotros le indiquemos por consola:

```

void readFile(char* nameArchivo){
    FILE* file;
    char* filename;
    char buf[128];
    int i = 0;

    filename = nameArchivo;
    printf("File: %s\n", filename);

    /* open the file */
    file = fopen(filename, "r");
    if (!file) {
        fprintf(stderr, "open failed: can't open file \"%s\".\n", filename);
        exit(1);
    }

    while(fgets(buf, 256, file) != NULL) {
        parseCommand(buf);
    }
}

```

Para ello introducimos el siguiente código dentro del `while` de la función `parseCommand()` en el `if` donde comparamos la entrada con las ordenes de la tortuga que conocemos:

```

} else if (!strcmp("load",strToken0)) { // LOAD
    readFile(strToken1);
    break;
}

```

Definir el sistema de medida de ángulos, de forma que en vez de utilizar el sistema sexagesimal de 0 a 360°, se pueda utilizar cualquier otro; como radianes de 0 a 2 π ; grados centesimales, de 0 a 400 o cualquier otro como de 0 a 12 ó 0 a 60 como las horas o minutos de un reloj. Utilizar para ello el comando `SETCIRCLE nn`, siendo `nn` el número correspondiente a la medida de un giro completo. Por defecto el valor es 360. La abreviatura del comando es `sc`.

Definimos una nueva variable llamada `giroCompleto` de valor inicial 360

```
GLdouble giroCompleto=360;
```

La utilizaremos como factor corrector de los valores que introduzcamos por ventana.

```
glRotatef(val*360/giroCompleto,0.,1.,0.);
```

Añadimos la correspondiente lectura del comando “sc” en la función parseCommand()

```
else if (!strcmp("sc",strToken0)) { // SETCIRCLE
    giroCompleto=val;
}
```

Implementar los comandos HIDE TURTLE y SHOW TURTLE, con las abreviaturas ht y st que muestran u ocultan la tortuga.

Añadimos lo siguiente en parseCommand() dentro del if de comprobación de comandos que no llevan argumento:

```
else if (strToken0 != NULL && !strcmp("st",strToken0)) { // SHOWTURTLE
    turtle = TRUE;
} else if (strToken0 != NULL && !strcmp("ht",strToken0)) { // HIDE TURTLE
    turtle = FALSE;
}
```

3. OPERACIONES CON MATRICES: DIBUJANDO EL CAMINO

Dibujar un rastro que consista en una superficie en lugar de una línea. Para ello se puede utilizar glBegin(GL_QUAD_STRIP) que dibuja una sucesión de rectángulos para cada pareja de puntos que recibe.

Guardamos una nueva sucesión de puntos ligeramente desplazados en las variables ppx[], ppy[] y ppz[] dentro de la función addPointToTrace().

```
// if is the first point
if (np == 0) { // add the first point
    px [0] = 0;
    py [0] = 0;
    pz [0] = 0;
    ppx [0] = 0.1;
    ppy [0] = 0;
    ppz [0] = 0;
    np++;
}
px [np] = m[0] * px [0] + m[4] * py [0] + m[8] * pz [0] + m[12];
py [np] = m[1] * px [0] + m[5] * py [0] + m[9] * pz [0] + m[13];
pz [np] = m[2] * px [0] + m[6] * py [0] + m[10] * pz [0] + m[14];

ppx [np] = m[0] * ppx [0] + m[4] * ppy [0] + m[8] * ppz [0] + m[12];
ppy [np] = m[1] * ppx [0] + m[5] * ppy [0] + m[9] * ppz [0] + m[13];
ppz [np] = m[2] * ppx [0] + m[6] * ppy [0] + m[10] * ppz [0] + m[14];
```

Utilizando los comandos de logo representar una esfera compuesta por un conjunto de circunferencias en el espacio.

```
repeat 4 [repeat 360 [fd .2 up 1] rt 45]
```

Utilizando los comandos de logo realizar la representación de una helicoidal.

```
repeat 720 [fd .2 up 90 fd .01 dn 90 rt 1]
```

4. VIENDO LA ESCENA DESDE OTRO PUNTO DE VISTA: CÁMARAS

Dotar al programa de una tecla que permita cambiar el modo de proyección entre ORTOGONAL y PERSPECTIVA.

Utilizaremos la tecla F4 para cambiar de modo añadiendo este pedazo de código en el switch de la función SpecialKey(key, x, y). Al estar programado no son necesarias

grandes operaciones, sólo hay que definir el campo de visión de la cámara en el modo cónico. El funcionamiento es de tipo toggle.

```
case GLUT_KEY_F4:
    if(LOCAL_MyCamera->camProjection == CAM_CONIC){
        LOCAL_MyCamera->x1=-3;
        LOCAL_MyCamera->x2=3;
        LOCAL_MyCamera->y1=-3;
        LOCAL_MyCamera->y2=3;
        LOCAL_MyCamera->z1=-5;
        LOCAL_MyCamera->z2=5;
        LOCAL_MyCamera->camProjection = CAM_PARALLEL;
    } else LOCAL_MyCamera->camProjection = CAM_CONIC;
    break;
```

Programar otros modos de movimiento de cámara como son el MODO PAN o el MODO TRÍPODE.

Para lograr el MODO TRÍPODE he definido una nueva función en el header camera.h

```
void PitchCamera( camera *thisCamera, float angle ); // local axis Z camera
```

El código nuevo que permite este movimiento se encuentra en camera.c:

```
void PitchCamera( camera *thisCamera, float angle )
{
    float vIn[3];

    vIn[0]= thisCamera->camAtX - thisCamera->camViewX;
    vIn[1]= thisCamera->camAtY - thisCamera->camViewY;
    vIn[2]= thisCamera->camAtZ - thisCamera->camViewZ;

    VectorRotXZ( vIn, angle, TRUE );

    thisCamera->camAtX = thisCamera->camViewX + vIn[0];
    thisCamera->camAtY = thisCamera->camViewY + vIn[1];
    thisCamera->camAtZ = thisCamera->camViewZ + vIn[2];

    SetDependentParametersCamera( thisCamera );
}
```

En tecunlogo.c añadimos al SpecialKey(key, x, y) la captura de la tecla F5:

```
case GLUT_KEY_F5:
    if (current_mode != 0) break;
    current_mode = 3;
    LOCAL_MyCamera->camMovimiento = CAM_TRIPODE;
    break;
```

desde la función mouse(button, state, x, y) dentro del switch

```
switch(LOCAL_MyCamera->camMovimiento){
```

captamos el estado CAM_TRIPODE

```
case CAM_TRIPODE:
    if (state == GLUT_DOWN) glutMotionFunc(Tripode);
    if (state == GLUT_UP) glutMotionFunc(NULL);
    break;
```

que llama a la nueva función Trípode(x, y):

```
void Tripode(int x, int y){

    float rotacion_x, rotacion_y;

    rotacion_x = (float)(old_x - x) * DEGREE_TO_RAD / 5;
    rotacion_y = (float)(old_y - y) * DEGREE_TO_RAD / 5;
    YawCamera( LOCAL_MyCamera, rotacion_x );
    PitchCamera( LOCAL_MyCamera, rotacion_y );

    old_y = y;
    old_x = x;

    glutPostRedisplay();
}
```

Para lograr el MODO PAN también creamos una nueva función que declaramos en camera.h

```
void PanCamera( camera *thisCamera, float stepX, float stepY );
```

Esta función que se encuentra en camera.c recibe los valores de la posición del ratón y calcula el desplazamiento de la cámara como sigue:

```
void PanCamera( camera *thisCamera, float stepX, float stepY )
{
    float    vaX, vaY, vaZ;

    vaX= stepX * thisCamera->camIX + stepY * thisCamera->camJX;
    vaY= stepX * thisCamera->camIY + stepY * thisCamera->camJY;
    vaZ= stepX * thisCamera->camIZ + stepY * thisCamera->camJZ;

    // Set V & A
    thisCamera->camViewX = thisCamera->camViewX + vaX;
    thisCamera->camViewY = thisCamera->camViewY + vaY;
    thisCamera->camViewZ = thisCamera->camViewZ + vaZ;
    thisCamera->camAtX   = thisCamera->camAtX + vaX;
    thisCamera->camAtY   = thisCamera->camAtY + vaY;
    thisCamera->camAtZ   = thisCamera->camAtZ + vaZ;

    SetDependentParametersCamera( thisCamera );
}

```

Al igual que antes activamos el modo PAN desde la función mouse(button, state, x, y) dentro del switch

```
case CAM_PAN:
    if (state == GLUT_DOWN) glutMotionFunc(Pan);
    if (state == GLUT_UP) glutMotionFunc(NULL);
    break;

```

El control lo tiene la función Pan(x, y):

```
void Pan(int x, int y){

    float avance_x, avance_y;

    avance_x = (float)(old_x - x) / 10;
    avance_y = (float)(y - old_y) / 10;
    PanCamera( LOCAL_MyCamera, avance_x, avance_y);

    old_y = y;
    old_x = x;

    glutPostRedisplay();
}

```

Programar otro modo de movimiento de la tortuga como es el MODO CAR. Pulsando el botón izquierdo del ratón lograremos el movimiento deseado. Si lo desplazamos hacia delante conduciremos en línea recta. En cambio, si movemos el puntero hacia los lados, haremos que la tortuga gire como lo haría un coche.

Este modo lo activamos desde el modo interprete mediante la instrucción “car” y lo podemos detener o bien con la instrucción “stop” o bien pulsando F1. Dotamos al programa de un 5º modo que no permitirá el movimiento de la cámara mientras se mantenga activo.

En commandParse() encontramos esta captura:

```
else if (strToken0 != NULL && !strcmp("car",strToken0)) {
    enfoque = 1;
    current_mode = 5;
} else if (strToken0 != NULL && !strcmp("stop",strToken0)) {
    current_mode = 0;
}

```

En mouse(button, state, x, y) dirigimos el control del modo

```
if (current_mode == 5){
    if (state == GLUT_DOWN) glutMotionFunc(Car);
}

```

```

    if (state == GLUT_UP) glutMotionFunc(NULL);
    break;
}

```

hacia la función Car(x, y)

```

void Car(int x, int y){

    float avance_x, avance_y;

    avance_x = (float)(x - old_x) / 10;
    avance_y = (float)(old_y - y) / 10;

    if(avance_y > 0){
        strcat(strCommand, "fd .3 ");
        parseCommand(strCommand);
        strcpy(strCommand, "");
    }
    if(avance_x > 0){
        strcat(strCommand, "rt 5 fd .3 ");
        parseCommand(strCommand);
        strcpy(strCommand, "");
    } else if(avance_x < 0){
        strcat(strCommand, "lt 5 fd .3 ");
        parseCommand(strCommand);
        strcpy(strCommand, "");
    }

    old_y = y;
    old_x = x;

    glutPostRedisplay();
}

```

Opción de elegir el lugar de la cámara. La orden "cam obj" y "cam zero" colocarán la cámara detrás del objeto o en el origen respectivamente.

La función EnfocaObjeto() se encarga de colocar la cámara según sea necesario. El valor de la variable enfoque será 0 para enfocar el objeto y 1 para enfocar el origen.

```

void EnfocaObjeto(void){

    if (enfoque){
        LOCAL_MyCamera->camViewX = 0;
        LOCAL_MyCamera->camViewY = 1;
        LOCAL_MyCamera->camViewZ = -3;
        LOCAL_MyCamera->camAtX = 0;
        LOCAL_MyCamera->camAtY = 0;
        LOCAL_MyCamera->camAtZ = 0;
    } else {
        LOCAL_MyCamera->camViewX = mModel[0]*0+mModel[4]*2-mModel[8]*5+mModel[12];
        LOCAL_MyCamera->camViewY = mModel[1]*0+mModel[5]*2-mModel[9]*5+mModel[13];
        LOCAL_MyCamera->camViewZ = mModel[2]*0+mModel[6]*2-mModel[10]*5+mModel[14];
        LOCAL_MyCamera->camAtX = mModel[0]*0+mModel[4]*0+mModel[8]*15+mModel[12];
        LOCAL_MyCamera->camAtY = mModel[1]*0+mModel[5]*0+mModel[9]*15+mModel[13];
        LOCAL_MyCamera->camAtZ = mModel[2]*0+mModel[6]*0+mModel[10]*15+mModel[14];
    }
    SetDependentParametersCamera( LOCAL_MyCamera );
}

```

Introducimos la captura de las ordenes en el while de parseCommand()

```

while ((strToken1 = strtok(NULL, " ")) != NULL ) {

    if(!strcmp("cam",strToken0)){
        if (!strcmp("obj",strToken1)) enfoque = 0;
        else if (!strcmp("zero",strToken1)) enfoque = 1;
        EnfocaObjeto();
        return;
    }
}

```

Programar otro modo de movimiento para nuestro objeto como es el MODO FLY. Al apretar el botón izquierdo del ratón comienza el movimiento en línea recta y al mover el cursor hacia los lados

produce un giro. Al mover el ratón hacia delante el objeto baja y viceversa como ocurre en una avioneta.

Funciona análogamente al MODO CAR y parará con la orden "stop" o pulsando F1. La orden se recibe en parseCommand() como de costumbre

```
else if (strToken0 != NULL && !strcmp("fly",strToken0)) {
    enfoque = 0;
    current_mode = 6;
}
```

En la función del ratón introducimos el cambio al nuevo y 6º modo. Hacemos uso de la orden glutIdleFunc() para llamar a la función FlyStep el rato que pasamos sin mover el raton pero sí mantenemos pulsado el botón izquierdo:

```
if (current_mode == 6){
    if (state == GLUT_DOWN) {glutMotionFunc(Fly);glutIdleFunc(FlyStep);}
    if (state == GLUT_UP) {glutMotionFunc(NULL);glutIdleFunc(NULL);}
    break;
}
```

El control de la aplicación se divide en dos funciones en caso de que tengamos pulsado el ratón. Fly() para los giros a la hora de mover el ratón y FlyStep() para el rato durante el cual no movemos el ratón pero nos interesa que siga avanzando nuestro objeto.

```
void FlyStep(void){
    strcat(strCommand, "fd .2 ");
    parseCommand(strCommand);
    strcpy(strCommand, "");
    EnfocaObjeto();
}

void Fly(int x, int y){
    float avance_x, avance_y;

    avance_x = (float)(x - old_x);
    avance_y = (float)(old_y - y);

    if(avance_y < 0){
        strcat(strCommand, "up 1 ");
        parseCommand(strCommand);
        strcpy(strCommand, "");
    } else if(avance_y > 0){
        strcat(strCommand, "dn 1 ");
        parseCommand(strCommand);
        strcpy(strCommand, "");
    }
    if(avance_x > 0){
        strcat(strCommand, "rt 3 fd .2 ");
        parseCommand(strCommand);
        strcpy(strCommand, "");
    } else if(avance_x < 0){
        strcat(strCommand, "lt 3 fd .2 ");
        parseCommand(strCommand);
        strcpy(strCommand, "");
    }

    EnfocaObjeto();

    old_y = y;
    old_x = x;
}
```

Programar otro modo de cámara como es el MODO FLY en el que la cámara vuela por el escenario. Navegamos pulsando el botón izquierdo del ratón y giramos según los desplazamientos de este.

Pulsando la tecla F7 activamos el modo dándole valor CAM_FLY a camMovimiento de nuestra cámara.

```

case GLUT_KEY_F7:
    if (current_mode != 0) break;
    current_mode = 8;
    LOCAL_MyCamera->camMovimiento = CAM_FLY;
    break;

```

Dentro de la función mouse(int button, int state, int x, int y), en el switch donde comprobamos el valor de camMovimiento, nos interesa captar cuándo es igual a CAM_FLY

```

case CAM_FLY:
    if (state == GLUT_DOWN){glutMotionFunc(CamFly); glutIdleFunc(CamFlyStep);}
    if (state == GLUT_UP) {glutMotionFunc(NULL);glutIdleFunc(NULL);}
    break;

```

Esto hace que cuando pulsemos el ratón y lo mantengamos pulsado llamemos a CamFlyStep() y cuando teniéndolo pulsado lo movamos llamemos a CamFly(). Estas dos funciones se encargarán del desplazamiento de la cámara por la escena.

```

void CamFly(int x, int y){

    float rotacion_x, rotacion_y;

    rotacion_x = (float)(old_x - x) * DEGREE_TO_RAD / 5;
    rotacion_y = (float)(old_y - y) * DEGREE_TO_RAD / 5;
    YawCamera( LOCAL_MyCamera, rotacion_x );
    PitchCamera( LOCAL_MyCamera, rotacion_y );
    AvanceFreeCamera( LOCAL_MyCamera, -.05f);

    old_y = y;
    old_x = x;

    glutPostRedisplay();
}

void CamFlyStep(void){

    AvanceFreeCamera( LOCAL_MyCamera, -.05f);

    glutPostRedisplay();
}

```

Mejora en la posición de la cámara.

Para poder respetar la posición de la cámara cuando la desplazemos manualmente desactivamos el modo de enfoque poniendo a FALSE la variable enfoca. La pondremos a TRUE cuando elijamos los modos “cam obj” o “cam zero”. Esto añade pequeños cambios en muchas funciones del código que no voy a redactar por completo por ser repetición de lo ya expuesto con anterioridad.

```

. . .
static int current_light = -1;

GLboolean ejes = TRUE;
GLboolean command = FALSE; /* command mode */
GLboolean turtle = TRUE; // La tortuga
GLboolean enfoca = TRUE;
char strCommand[256];
GLdouble giroCompleto=360;
int enfoca=1; // 0 enfoque en objeto. 1 enfoque en origen

int np = 0;
float px [10000];
. . .

void parseCommand(char* strCommandParse) {

    char *repeatCommand;
    char *nextCommand;
    . . .
    while ((strToken1 = strtok(NULL," ")) != NULL ) {

        if(!strcmp("cam",strToken0)){

```

```

        if (!strcmp("obj",strToken1)) {
            if(current_mode != 5 && current_mode != 6)current_mode=0;
            enfoque = 0; enfoca = TRUE;
        }
        else
            if (!strcmp("zero",strToken1)) {
                if(current_mode != 5 && current_mode != 6)current_mode=0;
                enfoque = 1; enfoca = TRUE;
            }
        EnfocaObjeto();
        return;
    }
    val = atof(strToken1);
. . .
} else if (strToken0 != NULL && !strcmp("car",strToken0)) { // MODO CAR
    current_mode = 5;
} else if (strToken0 != NULL && !strcmp("stop",strToken0)) { //STOP
    current_mode = 0;
} else if (strToken0 != NULL && !strcmp("fly",strToken0)) { // MODO FLY
    current_mode = 6;
}
}

static void SpecialKey ( int key, int x, int y ){
    switch(key) {
. . .

        case GLUT_KEY_F2:
            . . .
            enfoca = FALSE;
            break;

```

Ídem con F3, F5, F6 y F7.

```

        case GLUT_KEY_HOME: //Reset Camera
            enfoque = 1; enfoca = TRUE; EnfocaObjeto();
            SetDependentParametersCamera( LOCAL_MyCamera );
            break;
        default:
            printf("key %d %c X %d Y %d\n", key, key, x, y );
    }
    glutPostRedisplay();
}

void EnfocaObjeto(void){

    if (enfoca)
        if (enfoca){
            LOCAL_MyCamera->camViewX = 0
. . .
}

```

Operación de aterrizaje del objeto

Después de navegar en modo fly podemos pedir que aterrice mediante la orden "land". Para ello captamos la orden dentro de la función parseCommand y hacemos los cálculos necesarios para el correcto aterrizaje. Cargamos la identidad y trasladamos el objeto al punto de final del vuelo. Después calculamos el giro que llevaba la trayectoria para mantener una cierta continuidad. Giramos el objeto 45° hacia arriba o hacia abajo según convenga, dependiendo de si hemos volado o nos hemos sumergido, y calculamos los pasos necesarios a una velocidad de avance de 0.1 para que llegue de vuelta al plano inicial de coordenada y cero. Después corregimos el objeto con un giro de 45° otra vez para que se coloque paralelo al plano.

```

. . .
} else if (strToken0 != NULL && !strcmp("land",strToken0) && current_mode ==6) {

    a[0] = asin(mModel[8])*RAD_TO_DEGREE;//Giro sufrido por el objeto medido según
eje z
    a[2] = 0; // determinará si aterrizamos por encima o por debajo de z=0

    glPushMatrix();
    glLoadIdentity();

```

```

    glTranslatef(mModel[12], mModel[13], mModel[14]);
    printf("a: %f m: %f ", a[0], mModel[10]);
    if(a[0]<0.01 && mModel[10]<0)
        glRotatef(180,0.,1.,0.);
    else
        glRotatef(a[0],0.,1.,0.);
    glGetDoublev (GL_MODELVIEW_MATRIX, mModel);
    glPopMatrix();

    a[1]=10*2*fabs(mModel[13])/sqrt(2);//Vueltas que necesita el for para
    aterrizar.
    strcpy(strCommand, "");
    if(mModel[13]>0) {strcat(strCommand, "dn 45");a[2]=1;}//Modelo por encima de
    tierra?
    else strcat(strCommand, "up 45");
    parseCommand(strCommand);
    strcpy(strCommand, "");
    for(i=0;i<a[1];i++){
        strcat(strCommand, "fd .1 ");
        parseCommand(strCommand);
        strcpy(strCommand, "");
        EnfocaObjeto();
    }
    if(a[2]==1) strcat(strCommand, "up 45");
    else strcat(strCommand, "dn 45");
    parseCommand(strCommand);
    strcpy(strCommand, "");
    EnfocaObjeto();
}

```

5. ILUMINANDO LA ESCENA

Proporcionar alguna forma amigable de modificar el color de la luz. Por ejemplo mediante el teclado con las letras (r,g,b para disminuir y R,G,B para aumentar)

Lo controlaremos desde la función keyboard(unsigned char key, int x, int y). Modificaremos el valor de la componente de luz difusa.

```

void keyboard(unsigned char key, int x, int y) {

    if (command) {
        . . .
    }

    if (current_mode==7){

        switch (key) {

            case 'R':
                if(LOCAL_MyLights[current_light]->diffuse[0]<1){
                    LOCAL_MyLights[current_light]->diffuse[0] = LOCAL_MyLights[current_light]-
>diffuse[0] + 0.1f;
                    LOCAL_MyLights[current_light]->needsUpdate = TRUE;
                }
                break;
            case 'G':
                if(LOCAL_MyLights[current_light]->diffuse[1]<1){
                    LOCAL_MyLights[current_light]->diffuse[1] = LOCAL_MyLights[current_light]-
>diffuse[1] + 0.1f;
                    LOCAL_MyLights[current_light]->needsUpdate = TRUE;
                }
                break;
            case 'B':
                if(LOCAL_MyLights[current_light]->diffuse[2]<1){
                    LOCAL_MyLights[current_light]->diffuse[2] = LOCAL_MyLights[current_light]-
>diffuse[2] + 0.1f;
                    LOCAL_MyLights[current_light]->needsUpdate = TRUE;
                }
                break;
            case 'r':

```

```

        if(LOCAL_MyLights[current_light]->diffuse[0]>0){
            LOCAL_MyLights[current_light]->diffuse[0] = LOCAL_MyLights[current_light]-
>diffuse[0] - 0.1f;
            LOCAL_MyLights[current_light]->needsUpdate = TRUE;
        }
        break;
    case 'g':
        if(LOCAL_MyLights[current_light]->diffuse[1]>0){
            LOCAL_MyLights[current_light]->diffuse[1] = LOCAL_MyLights[current_light]-
>diffuse[1] - 0.1f;
            LOCAL_MyLights[current_light]->needsUpdate = TRUE;
        }
        break;
    case 'b':
        if(LOCAL_MyLights[current_light]->diffuse[2]>0){
            LOCAL_MyLights[current_light]->diffuse[2] = LOCAL_MyLights[current_light]-
>diffuse[2] - 0.1f;
            LOCAL_MyLights[current_light]->needsUpdate = TRUE;
        }
        break;
    }
}

glutPostRedisplay();
}

```

Añadir la orden "funkycolors" que genera un control aleatorio en el color de las luces

La orden activa desde `parseCommand()` un booleano llamado `funky`:

```

} else if (strToken0 != NULL && !strcmp("funkycolors",strToken0)) { // funkyColors
    if(funky) funky = FALSE; else funky = TRUE;
}

```

La función en cuestión es la siguiente:

```

void funkyColors(void){
    int i;
    for(i=0;i<3;i++){
        LOCAL_MyLights[i]->diffuse[0]=(float)rand()/32767;
        LOCAL_MyLights[i]->diffuse[1]=(float)rand()/32767;
        LOCAL_MyLights[i]->diffuse[2]=(float)rand()/32767;
        LOCAL_MyLights[i]->needsUpdate = TRUE;
    }
}

```

Que será llamada desde `EnfocaObjeto()` cada vez que nos movamos.

```

if(funky) funkyColors();

```

Añadir dos focos al objeto imitando los faros de un vehículo. Alumbrarán la escena cuando los activemos mediante la orden "lights".

Necesitaremos 2 luces más en la escena. Reservamos memoria y las inicializamos en el `main()`.

```

. . .
LOCAL_MyLights = (light **) malloc( 5 * sizeof(light *));
. . .
//FAROS
LOCAL_MyLights[3] = CreateDefaultLight();
LOCAL_MyLights[3]->type = AGA_SPOT;
LOCAL_MyLights[3]->id = GL_LIGHT3;
LOCAL_MyLights[3]->position[0] = -.7f;
LOCAL_MyLights[3]->position[1] = 0.0f;
LOCAL_MyLights[3]->position[2] = 1.2f;
LOCAL_MyLights[3]->spotDirection[0] = 0.0f;
LOCAL_MyLights[3]->spotDirection[1] = 0.0f;
LOCAL_MyLights[3]->spotDirection[2] = 1.0f;
LOCAL_MyLights[3]->spotCutOff=15;

LOCAL_MyLights[4] = CreateDefaultLight();
LOCAL_MyLights[4]->type = AGA_SPOT;
LOCAL_MyLights[4]->id = GL_LIGHT4;
LOCAL_MyLights[4]->position[0] = .7f;
LOCAL_MyLights[4]->position[1] = 0.0f;

```

```

LOCAL_MyLights[4]->position[2] = 1.2f;
LOCAL_MyLights[4]->spotDirection[0] = 0.0f;
LOCAL_MyLights[4]->spotDirection[1] = 0.0f;
LOCAL_MyLights[4]->spotDirection[2] = 1.0f;
LOCAL_MyLights[4]->spotCutOff=15;

```

Como viene siendo habitual añadimos la lectura en parseCommand(). Encendemos o apagamos las luces según sea necesario y de la misma manera activamos o desactivamos el booleano faros para los cálculos necesarios después en el display.

```

} else if(strToken0 != NULL && !strcmp("lights",strToken0)){
    if ( LOCAL_MyLights[3]->switched ){
        SwitchLight( LOCAL_MyLights[3], FALSE);
        SwitchLight( LOCAL_MyLights[4], FALSE);
        faros = FALSE;
    } else {
        SwitchLight( LOCAL_MyLights[3], TRUE);
        SwitchLight( LOCAL_MyLights[4], TRUE);
        faros = TRUE;
    }
}
}

```

Los cálculos los realizamos desde aquí:

```

void colocaFaros(void){

    LOCAL_MyLights[3]->position[0] = mModel[0]*(-.7) + mModel[4]*0 + mModel[8]*1.2
+ mModel[12];
    LOCAL_MyLights[3]->position[1] = mModel[1]*(-.7) + mModel[5]*0 + mModel[9]*1.2
+ mModel[13];
    LOCAL_MyLights[3]->position[2]= mModel[2]*(-.7) + mModel[6]*0 + mModel[10]*1.2
+ mModel[14];

    LOCAL_MyLights[4]->position[0] = mModel[0]*.7 + mModel[4]*0 + mModel[8]*1.2 +
mModel[12];
    LOCAL_MyLights[4]->position[1] = mModel[1]*.7 + mModel[5]*0 + mModel[9]*1.2 +
mModel[13];
    LOCAL_MyLights[4]->position[2] = mModel[2]*.7 + mModel[6]*0 + mModel[10]*1.2 +
mModel[14];

    LOCAL_MyLights[3]->spotDirection[0]=mModel[8];
    LOCAL_MyLights[3]->spotDirection[1]=mModel[9];
    LOCAL_MyLights[3]->spotDirection[2]=mModel[10];

    LOCAL_MyLights[4]->spotDirection[0]=mModel[8];
    LOCAL_MyLights[4]->spotDirection[1]=mModel[9];
    LOCAL_MyLights[4]->spotDirection[2]=mModel[10];

    LOCAL_MyLights[3]->needsUpdate = TRUE;
    LOCAL_MyLights[4]->needsUpdate = TRUE;

}

```

Cuando lo pedimos desde display()

```

. . .
SetLight( LOCAL_MyLights[3] );
SetLight( LOCAL_MyLights[4] );

. . .

if(faros) colocaFaros();

. . .

```

Y muere en el Esc de la función keyboard()

```

. . .
DestroyLight( LOCAL_MyLights[3] );
DestroyLight( LOCAL_MyLights[4] );
free (LOCAL_MyLights);
exit(0);
break;

```

6. MODO JUEGO

Programar un modo llamado game donde se haga uso del timer y haya un control de posición de varios objetos al mismo tiempo en movimiento dentro de la escena, así como de la cámara que perseguirá a nuestra 'tortuga'.

La solución viene dada por un sencillo juego donde nos aparece un muro desde donde salen bloques dispuestos a derribarnos. Estos bloques nos perseguirán e intentarán alcanzar la línea de meta que es nuestro punto de partida. La línea de z cero. Nuestro objetivo es obtener buena puntuación alcanzando los bloques con el lanzamiento de tiro parabólico de nuestra cabeza. Si derribamos los suficientes ganamos y si nos alcanzan o superan en sucesivas ocasiones la línea de partida perdemos.

Comencemos por estudiar las variables de control de los objetos de la escena.

```
GLdouble poszSprite[2];
GLdouble posxSprite[2];
GLdouble posxSpriteCabeza[2];
GLdouble posySpriteCabeza[2];
GLdouble poszSpriteCabeza[2];
GLdouble timerSpriteCabeza[2];
GLdouble funcSpriteCabeza[2];
GLint game = 0;
GLint puntos=0;
```

Las posiciones de los llamados Sprite a secas corresponden a los bloques. Estos no sufren desplazamientos en y. Los SpriteCabeza corresponden a la esfera de la cabeza de la tortuga que va a ser nuestro proyectil. La variable timerSpriteCabeza llevará la cuenta del tiempo que lleva en acción la cabeza para hacer correctamente el cálculo del tiro parabólico. Y funcSpriteCabeza confirma o desmiente el estado del proyectil. Valdrá 1 para decir que la hemos lanzado. El valor del entero game servirá para saber si no estamos jugando(0); si estamos en modo juego(1); si hemos perdido(2) o si hemos ganado(3). Finalmente la variable puntos es la que lleva la cuenta de los bloques que hemos destruido o de los que han cruzado la línea.

Veamos ahora la llamada al modo desde la función parseCommand()

```
} else if (strToken0 != NULL && !strcmp("game",strToken0)) { // GAME
    command = FALSE;
    enfoque=0;
    enfoca=TRUE;
    EnfocaObjeto();
    LOCAL_MyLights[1]->diffuse[0]=1.0f;
    LOCAL_MyLights[1]->diffuse[1]=0.9f;
    LOCAL_MyLights[1]->diffuse[2]=0.7f;
    LOCAL_MyLights[1]->position[0] = 5.0f;
    LOCAL_MyLights[1]->position[1] = 30.0f;
    LOCAL_MyLights[1]->position[2] = 5.0f;
    LOCAL_MyLights[1]->position[3] = 1.0f;
    if ( !LOCAL_MyLights[1]->switched )
        SwitchLight( LOCAL_MyLights[1], TRUE);

    poszSprite[0]=100;
    posxSprite[0]=aleatorio()*10-5;
    poszSprite[1]=100;
    posxSprite[1]=aleatorio()*10-5;
    funcSpriteCabeza[0]=0;
    puntos=0;

    game=1;
    Timer();
}
```

Desactivamos el modo comando y colocamos la cámara detrás del objeto. Elegimos la iluminación que nos interesa. Colocamos los bloques en su línea de salida en $z=100$ y con un valor en x aleatorio. Activamos el modo juego y llamamos al timer. La función `aleatorio()` la hice en un intento de conseguir un valor más diferente cada vez porque noté que se repetía bastante el juego.

```
float aleatorio(void){
    int i;
    for(i=0;i<(int)(rand()/32767*100);i++){
    }
    return (float)rand()/32767;
}
```

La función `timer` que encontré en la `glut` me lanzaba una sola vez esta función y si utilizaba el `glutIdleFunc()` con el `Timer` no conseguía periodicidad así que utilicé la función `Timer()`

```
void Timer(void){
    glutTimerFunc(50, sprite, 1);
}
```

para llamar a otra, `sprite()`, que haría los cálculos y al terminar volvería a llamar al timer. Observé que esto proporcionaba la periodicidad que yo le indicaba al timer.

```
void sprite(int value){

    int i;
    //Avance del cubo
    poszSprite[0]=poszSprite[0]-0.7;
    poszSprite[1]=poszSprite[1]-1.3;

    for(i=0;i<2;i++){//2 Vueltas para los 2 cubos

        if(poszSprite[i]>mModel[14])
            if(posxSprite[i]<mModel[12]+0.3)
                posxSprite[i]=posxSprite[i]+0.5;
            else
                posxSprite[0]=posxSprite[0]-0.5;
        if(funcSpriteCabeza[0]==1) { //Cabeza en marcha
            if( posxSpriteCabeza[0]<posxSprite[i]+3 && posxSpriteCabeza[0]>posxSprite[i]-
3 &&
                posySpriteCabeza[0]<5 && posySpriteCabeza[0]>=0 &&
                poszSpriteCabeza[0]<poszSprite[i]+3 && poszSpriteCabeza[0]>poszSprite[i]-3)
                { //HIT CABEZA-CUBO
                    funcSpriteCabeza[0]=0;
                    posxSpriteCabeza[0]=0;
                    posySpriteCabeza[0]=0;
                    poszSpriteCabeza[0]=0;
                    poszSprite[i]=100;
                    puntos=puntos+100;
                } else if ( posySpriteCabeza[0]<0 || poszSpriteCabeza[0]>100){ //Fin
trayectoria cabeza
                    funcSpriteCabeza[0]=0;
                    timerSpriteCabeza[0]=0;
                    posxSpriteCabeza[0]=0;
                    posySpriteCabeza[0]=0;
                    poszSpriteCabeza[0]=0;
                }
                //Cálculo coordenadas en el avance
                //Tiro parabólico de la cabeza
                posySpriteCabeza[0]=15*timerSpriteCabeza[0]/10*sin(45*DEGREE_TO_RAD)-
0.5*9.8*(timerSpriteCabeza[0]/10)*(timerSpriteCabeza[0]/10);
                poszSpriteCabeza[0]=poszSpriteCabeza[0]+2;
                timerSpriteCabeza[0]=timerSpriteCabeza[0]+2;
            }
        if(mModel[12]<posxSprite[i]+3 && mModel[12]>posxSprite[i]-3 &&
mModel[14]<poszSprite[i]+3 && mModel[14]>poszSprite[i]-3){ //HIT TORTU-CUBO
            game=2; //GAME OVER
        } else if(poszSprite[i]<=0) { //FIN RECORRIDO CUBO
            poszSprite[i]=100;
            posxSprite[i]=aleatorio()*20-5;
            puntos=puntos-50;
        }
    }
}
```

```

    if (puntos <= -150) game=2;
    else if(puntos >= 500) game=3;
    EnfocaObjeto();
    glutPostRedisplay();if(game==1)Timer();
}

```

El control del movimiento se hace con las teclas 4, 5, 6 y 8 para movernos hacia la izquierda, atrás, derecha y adelante respectivamente. Y la barra espaciadora nos sirve para el lanzamiento. El control se hace desde la función keyboard()

```

void keyboard(unsigned char key, int x, int y) {
    if(game==1){
        switch (key){
            case '4'://IZQ
                mModel[12]=mModel[12]+1;
                EnfocaObjeto();
                break;
            case '5'://ATRÁS
                mModel[14]=mModel[14]-1;
                EnfocaObjeto();
                break;
            case '6'://DCHA
                mModel[12]=mModel[12]-1;
                EnfocaObjeto();
                break;
            case '8'://ADELANTE
                mModel[14]=mModel[14]+1;
                EnfocaObjeto();
                break;
            case ' '://DISPARO
                if(funcSpriteCabeza[0]==0){
                    posXSpriteCabeza[0]=mModel[12];
                    poszSpriteCabeza[0]=mModel[14];
                    timerSpriteCabeza[0]=0;
                    funcSpriteCabeza[0]=1;
                }
            default:
                break;
        }
    }
    if (command) {
        . . .
    }
}

```

Y finalmente, desde el display se dibujan todos los objetos que participan en la acción o el texto de “GAME OVER” o “YOU WIN” del final de juego.

```

. . .
switch(game){
case 1:

    //Bloque de fondo *****ESTÁTICO*****
    glColor3f(0.5,0.5,1.0);
    glPushMatrix();
    glTranslatef(0,5,100);
    glScalef(5.0f,1.0f,1.0f);
    glutSolidCube(10);
    glPopMatrix();
    /*******MOVIMIENTO*****
    //Bloques
    glColor3f(1.0,1.0,1.0);
    for(i=0;i<2;i++){
        glPushMatrix();
        glTranslatef(posxSprite[i],2.5f,poszSprite[i]);
        glutSolidCube(5);
        glPopMatrix();
    }
    //Cabeza
    if(funcSpriteCabeza[0]==1){
        glColor3f(1.0,0.0,0.0);
        glPushMatrix();
        glScalef(1.0f,.6f,1.0f);
        glTranslatef(posxSpriteCabeza[0],posySpriteCabeza[0],1.2f+poszSpriteCabeza[0]);
        glutSolidSphere(.4,40,40);
        glPopMatrix();
    }
    /*******

```

```

        break;
    case 2:
        glColor3f(1.0,0.0,0.0);
        text((glutGet(GLUT_WINDOW_WIDTH)-260)/2,glutGet(GLUT_WINDOW_HEIGHT)-60,50,"GAME
OVER");
        break;
    case 3:
        glColor3f(1.0,1.0,0.0);
        text((glutGet(GLUT_WINDOW_WIDTH)-260)/2,glutGet(GLUT_WINDOW_HEIGHT)-60,50,"YOU
WIN");
        break;
    default:
        break;
}

```

7. VISIÓN ESTEREOSCÓPICA

Como final a estas prácticas, intentar implementar una orden que genere una imagen de visión estereoscópica mediante el comando "stereo".

El efecto logrado resulta de mezclar dos imágenes, una roja y otra azul, captadas por dos cámaras separadas una distancia equivalente a la de los ojos. Al utilizar gluPerspective() a la hora de proyectar las imágenes captadas por la cámara no podremos conseguir que parezca que el objeto salga de la pantalla y es un método que cansa la vista por tener un pequeño desajuste en vertical. Para corregir estos errores habría que utilizar la función glFrustum() y hacer ciertos cálculos de trigonometría. Por otro lado comentar que se ha alternado la cámara que imprime el display, lo cual genera un parpadeo que podría arreglarse imprimiendo en la misma pantalla y masqueando los colores rojo y azul para que el uno no tape al otro. Esto se consigue utilizando glColorMask(R, G, B, ALPHA) poniendo a GL_TRUE los que necesitamos.

Vamos a ir en orden analizando el código. Primero, definición de variables.

```

static camera *LOCAL_MyCamera2;

GLboolean stereo=FALSE;
GLint noCamara=1;

```

Tenemos una nueva cámara que creamos en el main()

```

LOCAL_MyCamera2 = CreatePositionCamera(-0.1f, 1.0f, -3.0f);

```

En parseCommand recibimos la orden e inicializamos las variables.

```

} else if(strToken0 != NULL && !strcmp("stereo",strToken0)){
    if (!stereo){

        LOCAL_MyLights[1]->diffuse[0]=0.0f;
        LOCAL_MyLights[1]->diffuse[1]=0.0f;
        LOCAL_MyLights[1]->diffuse[2]=0.0f;
        LOCAL_MyLights[1]->position[0] = 5.0f;
        LOCAL_MyLights[1]->position[1] = 30.0f;
        LOCAL_MyLights[1]->position[2] = 0.0f;
        LOCAL_MyLights[1]->position[3] = 1.0f;
        LOCAL_MyLights[1]->ambient[0]=0.0f;
        LOCAL_MyLights[1]->ambient[1]=0.0f;
        LOCAL_MyLights[1]->ambient[2]=0.0f;

        if ( !LOCAL_MyLights[1]->switched )
            SwitchLight( LOCAL_MyLights[1], TRUE);

        LOCAL_MyCamera->camViewX = -0.03f;
        LOCAL_MyCamera->camViewY = 1;
        LOCAL_MyCamera->camViewZ = -3;
        LOCAL_MyCamera->camAtX = 0;//-0.15f;
        LOCAL_MyCamera->camAtY = 0;
        LOCAL_MyCamera->camAtZ = -1.0f;
    }
}

```

```

LOCAL_MyCamera2->camViewX = 0.03f;//0.03
LOCAL_MyCamera2->camViewY = 1;
LOCAL_MyCamera2->camViewZ = -3;
LOCAL_MyCamera2->camAtX = 0;//0.15f;
LOCAL_MyCamera2->camAtY = 0;
LOCAL_MyCamera2->camAtZ = -1.0f;

stereo = TRUE;
stereoTimer();
} else stereo = FALSE;
}

```

Ponemos a cero las luces y colocamos las cámaras. Activamos la variable stereo y llamamos al timer stereoTimer()

```

void stereoTimer(void){
    glutTimerFunc(2,stereoVision,1);
}

```

que a su vez llama a la función stereoVision que se encarga de cambiar la iluminación de rojo a azul y viceversa así como de cambiar de cámara.

```

void stereoVision(int value){

    if(noCamara==1){
        LOCAL_MyLights[1]->ambient[0]=0.3f;
        LOCAL_MyLights[1]->ambient[2]=0.0f;
        noCamara=2;
    } else {
        LOCAL_MyLights[1]->ambient[0]=0.0f;
        LOCAL_MyLights[1]->ambient[2]=0.3f;
        noCamara=1;
    }
    LOCAL_MyLights[1]->needsUpdate = TRUE;

    glutPostRedisplay();
    if(stereo) stereoTimer();
}

```

Importante restablecer bien el tamaño de la cámara 2 en la función reshape()

```

void reshape(int width, int height) {

    glViewport(0, 0, width, height);
    SetGLAspectRatioCamera( LOCAL_MyCamera );
    SetGLAspectRatioCamera( LOCAL_MyCamera2 );

}

```

Estos son los cambios introducidos en la función display(). El fondo lo ponemos gris porque conseguimos un rojo y azul menos llamativos que funcionan mejor con las gafas.

```

void display(void) {

    float At[3];
    float Direction[3];
    int i;

    if(stereo) glClearColor(0.2f,0.2f,0.2f,0.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);

    if (stereo){
        if (noCamara==1)
            SetGLCamera( LOCAL_MyCamera );
        else
            SetGLCamera( LOCAL_MyCamera2 );
    } else SetGLCamera( LOCAL_MyCamera );
    . . .

    glPushMatrix();
    glMultMatrixd(mModel);
    glColor3f(1.0,1.0,1.0);
    if (tortu) if(stereo) drawWireSphereTurtle(); else drawSphereTurtle();
}

```

En el archivo primitivas.c y primitivas.h incorporamos una nueva función llamada drawWireSphereTurtle() que dibuja la misma tortuga pero empleando la orden glutWireSphere(). Con esto obtenemos la visualización de la tortuga con esferas hechas a base de alambres. El control de la cámara también lo he incorporado, en la función keyboard(), para comprobar mejores puntos de visión si así se desea.

```

. . .
    if (command) {
        //STEREO
        if (stereo)
            switch (key){
                case '0'://View ABAJO
                    LOCAL_MyCamera2->camViewY = LOCAL_MyCamera2->camViewY-0.1;
                    LOCAL_MyCamera->camViewY = LOCAL_MyCamera->camViewY-0.1;
                    break;
                case '1'://View ARRIBA
                    LOCAL_MyCamera2->camViewY = LOCAL_MyCamera2->camViewY+0.1;
                    LOCAL_MyCamera->camViewY = LOCAL_MyCamera->camViewY+0.1;
                    break;
                case '4'://View ATRÁS
                    LOCAL_MyCamera2->camViewZ = LOCAL_MyCamera2->camViewZ-0.1;
                    LOCAL_MyCamera->camViewZ = LOCAL_MyCamera->camViewZ-0.1;
                    break;
                case '7'://View ADELANTE
                    LOCAL_MyCamera->camViewZ = LOCAL_MyCamera->camViewZ+0.1;
                    LOCAL_MyCamera2->camViewZ = LOCAL_MyCamera2->camViewZ+0.1;
                    break;
                case '5'://At ATRÁS
                    LOCAL_MyCamera->camAtZ = LOCAL_MyCamera->camAtZ-0.1;
                    LOCAL_MyCamera2->camAtZ = LOCAL_MyCamera2->camAtZ-0.1;
                    break;
                case '8'://At ADELANTE
                    LOCAL_MyCamera->camAtZ = LOCAL_MyCamera->camAtZ+0.1;
                    LOCAL_MyCamera2->camAtZ = LOCAL_MyCamera2->camAtZ+0.1;
                    break;
                case '6'://Acercar ojos
                    LOCAL_MyCamera2->camViewX = LOCAL_MyCamera2->camViewX+0.0005;
                    LOCAL_MyCamera->camViewX = LOCAL_MyCamera->camViewX-0.0005;
                    break;
                case '9'://Alejarlos
                    LOCAL_MyCamera2->camViewX = LOCAL_MyCamera2->camViewX-0.0005;
                    LOCAL_MyCamera->camViewX = LOCAL_MyCamera->camViewX+0.0005;
                    break;
                default:
                    break;
            }
    }
. . .

```