

TRABAJO GRAFICOS POR COMPUTADOR.

Pedro Jose Trujillo Caballero 901329 Total puntos 32.

Ejes (1 punto)

Introducir en la funcion display() lo siguiente:

```
glColor3f(1.0,0.0,0.0);

    glBegin(GL_LINES);
        glVertex3f(0.0,0.0,0.0);
        glVertex3f(1.0,0.0,0.0);
        glColor3f(0.0,1.0,0.0);
        glVertex3f(0.0,0.0,0.0);
        glVertex3f(0.0,1.0,0.0);
        glColor3f(0.0,0.0,1.0);
        glVertex3f(0.0,0.0,0.0);
        glVertex3f(0.0,0.0,1.0);
    glEnd();
```

Comando LOAD (3 puntos)

Introducir en la funcion parseCommand

```
} else if (!strcmp("load",strToken0)) {
    FILE* file;
    char *filename;

    int i = 0;
    char buf[128];
    filename = strToken1;

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

    /* open the file */
    file = fopen(filename, "r");
    while(fgets(buf, 256, file) != NULL) {
        printf("Line %i: %s", ++i, buf);
        parseCommand(buf);
        glPushMatrix();
        glLoadIdentity();
        glMultMatrixd(actualTurtle->mModel);
    }
}
```

Con esto funciona en el archivo descargado del ADI
Habria que introducir la secuencia "load + NOMBREARCHIVO"

Comando SETCIRCLE (1 punto)

Introducir en la funcion parseCommand

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

Cambiar en la funcion parseCommand en las cuatro rotaciones
Donde pone **val** poner: **val*360/vuelta** asi:

```
    } else if (!strcmp("rt",strToken0)) {
        printf("RIGTH");
        glRotatef(-val*360/vuelta,0.,1.,0.);
    } else if (!strcmp("lt",strToken0)) {
        printf("LEFT");
        glRotatef(val*360/vuelta,0.,1.,0.);
    } else if (!strcmp("up",strToken0)) {
        printf("UP");
        glRotatef(-val*360/vuelta,1.,0.,0.);
    } else if (!strcmp("dn",strToken0)) {
        printf("DOWN");
        glRotatef(val*360/vuelta,1.,0.,0.);
    }
```

Crear la variable global vuelta de la forma:

```
int vuelta= 360;
```

Con esto funciona en el archivo descargado del ADI
Habria que introducir la secuencia "sc + LONGITUDUNAVUELTA"

Comando HIDE TURTLE Y SHOW TURTLE (1 punto)

Introducir en la funcion parseCommand antes de que comience el while
if (!strcmp("ht",strToken0)) {
 ocultar=1;
} else if (!strcmp("st",strToken0)) {
 ocultar=0;
}

Crear una variable global asi:

```
int ocultar=0;
```

La parte del main que llama a la funcion drawSphereTurtle() quedara
asi:

```
If (ocultar == 0) {  
    drawSphereTurtle();  
}
```

Habria que introducir la secuencia "ht" para ocultar y "st" para
mostrarla.

Comando LOG (1 punto)

Declarar como variables globales:

```
int logt=0;  
char filename[10];
```

Introducir en la funcion parseCommand

En las variables:

```
FILE *fichero;
```

```
char *nombre = filename;
```

A continuacion de las variables(en la funcion parseCommand):

```
if (logt==1){

    fichero = fopen( nombre, "a" );
    printf( "Fichero: %s (para escritura) -> ", nombre );
    if( fichero ){
        printf( "creado (ABIERTO)\n" );
    }else{
        printf( "Error (NO ABIERTO)\n" );
    }

    fprintf( fichero, strCommandParse ); fprintf(fichero,"\n");
    if( !fclose(fichero) ){
        printf( "Fichero cerrado\n" );
    }else{
        printf( "Error: fichero NO CERRADO\n" );
    }
}
```

Dentro de la serie de else if que interpretan los comandos(en la funcion parseCommand):

```
} else if (!strcmp("log",strToken0)) {
    logt =1;
    strcpy(filename,strToken1);
    printf("despuse de la copia: ", filename);
} else if (!strcmp("logstop",strToken0)) {
    logt=0;
```

Con esto funciona en el archivo descargado del ADI

Habria que introducir la secuencia "log + NOMBREARCHIVO" para que empezara a guardar y "logstop + NOMBREARCHIVO" para que parara de guardar comandos.

Comando HISTORY (1 punto)

Declarar como variables globales:

```
char history[100][300];
int i;
int twe=0;
```

Introducir en la funcion parseCommand

En las variables:

```
char *p;
char *nombre = filename;
```

A continuacion de las variables y despues de que este asignado strToken0(en la funcion parseCommand):

```

p=history[0];
    k=0;
    strcpy((p+100*twe),strToken0);
    twe=twe+1;
    if (!strcmp(strToken0,"history")){
        while(k<twe){
            printf("Comando %i : %s\n",k,(p+100*k));
            k++;
        }
    }

```

Dentro de la serie de else if que interpretan los comandos(en la funcion parseCommand):

```

} else if (!strcmp("history",strToken0)){
    i=atoi(strToken1);
    if (i>0) {
        k=i-1;
        while (k<twe){
            printf("Comando %i : %s\n",k,(p+100*k));
            k++;
        }
    }else{
        k=twe+i;
        while(k<twe){
            printf("Comando %i : %s\n",k,(p+100*k));
            k++;
        }
    }
}

```

Con esto funciona en el archivo descargado del ADI

Habria que introducir la secuencia "history" para recibir todos los comandos introducidos durante la sesion. La secuencia "history n" para recibir los comandos desde el numero n y la secuencia "history -n" para recibir los ultimos n comandos.

Comando RECOVERY (1 punto)

Declarar como variables globales:

```
char *op;;
```

Dentro de la serie de else if que interpretan los comandos(en la funcion parseCommand):

```

}else if (!strcmp("r",strToken0)){

        op=p+100*atoi(strToken1);

        parseCommand((op));
        glPushMatrix();
        glLoadIdentity();
        glMultMatrixd(actualTurtle->mModel);

```

Con esto funciona en el archivo descargado del ADI

Habria que introducir la secuencia "r n" donde n sera el comando que queramos utilizar. Obvio que todo esto hay que añadirlo despues de añadir todo lo que hay que añadir en la funcion HISTORY.

Comando Patrones de Rastro, Arcoiris, PenUp, Pendown (Total 8 puntos)

(Pongo todos juntos porque es mas facil de entender y separados puede ser bastante lio)

Tenemos que agregar las siguientes variables globales.

```
int trazoelegido=0; AQUÍ SE GUARDA EL TRAZO QUE ELIJAMOS
float a=0; ES NECESARIA PARA UNO DE LOS TRAZOS
int h=0; CLAVE PARA EL PENUP, PENDOWN, SEPARA RASTROS QUE SE DIBUJAN DE LOS QUE NO.
int trace=0; VARIABLE CREADA PARA SABER SI ES EL COMIENZO O NO.
```

La funcion que hay que cambiar es la funcion displayTrace() que queda de la siguiente manera:

```
void displayTrace(turtle* turtlei) {
    int i;
    int j;
    int lightingFlag; DESACTIVAMOS LAS LUCES PARA QUE SE NOTEN LOS COLORES
        lightingFlag = glIsEnabled( GL_LIGHTING );

    if( lightingFlag ) glDisable( GL_LIGHTING );
    if (trace==0){ ESTO PARA EL INICIO, PARA EMPEZAR A CONTAR
        turtlei->np1[0]=0;
    }
    if (rastros==0){ LA VARIABLE RASTRO NOS DEFINE SI PENUP O PENDOWN ESTA ACTIVADO
        actualTurtle->np1[h+1]=actualTurtle->np2[h]-1
        LOS RASTROS SE ESTABLECEN ENTRE NP1 Y NP2 Y SUCESIVAMENTE SE VAN ENLAZANDO,
        CLASIFICANDOSE EN PARES E IMPARES PARA QUE LUEGO EL BUCLE SEPA SI TIENE QUE DIBUJARLO O
        NO. AL FINAL RASTRO =3 PARA QUE ENTRE EN EL SIGUIENTE QUE ES EL QUE VA ACTUALIZANDO LA
        POSICION DE LA TORTUGA EN EL MOMENTO ACTUAL.
        h=h+1;
        actualTurtle->np2[h]=actualTurtle->np;
        rastros=3;
    }else if(rastros==1){
        actualTurtle->np1[2*h]=actualTurtle->np-1;
        h=h*2;
        actualTurtle->np2[h]=actualTurtle->np;
        rastros=3;
    }else{
        actualTurtle->np2[h]=actualTurtle->np;
    }
    for(j=0; j<h+1; j=j+2){ AQUÍ ELIJE QUE RASTRO TIENE QUE DIBUJAR Y CUAL NO
```

```
switch(trazoelegido){
AQUI COMIENZA UN BUCLE PARA ELEGIR TRAZO, SEGUN EL QUE HAYAMOS ELEGIDO CON LA FUNCION
SETTRACE SALDRA UNO U OTRO, TENEMOS 6 POSIBILIDADES, DEL 0 AL 5, DONDE ESTAN INCLUIDAS
LOS 4 RASTROS, EL RASTRO ARCOIRIS Y EL RASTRO INICIAL QUE ES UNA LINEA NEGRA (CASE 0).
```

```
case 3:      AQUI TENEMOS EL RASTRO ARCOIRIS
```

```
glBegin(GL_QUAD_STRIP);
for (i = actualTurtle->np1[j]; i < actualTurtle->np2[j]; i++) {

glColor3f(1.0,0.0,0.0);
glVertex3f(actualTurtle->px2[i],actualTurtle->py2[i],actualTurtle->pz2[i]);
glColor3f(1.0,1.0,0.0);
glVertex3f(actualTurtle->px4[i],actualTurtle->py4[i],actualTurtle->pz4[i]);
}
glEnd();
glBegin(GL_QUAD_STRIP);
for (i = actualTurtle->np1[j]; i < actualTurtle->np2[j]; i++) {
glColor3f(1.0,1.0,0.0);
glVertex3f(actualTurtle->px4[i],actualTurtle->py4[i],actualTurtle->pz4[i]);
glColor3f(0.0,1.0,0.0);
glVertex3f(actualTurtle->px[i],actualTurtle->py[i],actualTurtle->pz[i]);
}
glEnd();
glBegin(GL_QUAD_STRIP);
for (i = actualTurtle->np1[j]; i < actualTurtle->np2[j]; i++) {
glColor3f(0.0,1.0,0.0);
glVertex3f(actualTurtle->px[i],actualTurtle->py[i],actualTurtle->pz[i]);
glColor3f(0.0,1.0,1.0);
glVertex3f(actualTurtle->px5[i],actualTurtle->py5[i],actualTurtle->pz5[i]);
}
glEnd();
glBegin(GL_QUAD_STRIP);
for (i =actualTurtle->np1[j]; i < actualTurtle->np2[j]; i++) {

glColor3f(0.0,1.0,1.0);
glVertex3f(actualTurtle->px5[i],actualTurtle->py5[i],actualTurtle->pz5[i]);
glColor3f(0.0,0.0,1.0);
glVertex3f(actualTurtle->px3[i],actualTurtle->py3[i],actualTurtle->pz3[i]);

}
glEnd();
break;
```

```
case 0:      AQUI EL RASTRO INICIAL QUE ES UNA SIMPLE LINEA NEGRA
```

```
glBegin(GL_QUAD_STRIP);
for (i = actualTurtle->np1[j]; i < actualTurtle->np2[j]; i++) {
glColor3f(0,0,0);
glVertex3f (actualTurtle->px2[i],actualTurtle->py2[i],actualTurtle->pz2[i]);
glColor3f(0,0,0);
glVertex3f (actualTurtle->px4[i],actualTurtle->py4[i],actualTurtle->pz4[i]);
}
glEnd();
break;
```

```
case 1:      RASTRO CON ESFERAS (SIMULANDO NUBES)
```

```
for (i = actualTurtle->np1[j]; i < actualTurtle->np2[j]; i++) {
glPushMatrix();
glColor3f(0,0,1);
glTranslatef(actualTurtle->px[i],actualTurtle->py[i],actualTurtle->pz[i]);

glBegin(GL_POLYGON);

glutSolidSphere(0.1,50,10);
glEnd();
glColor3f(0,0,1);
glPopMatrix();
glPushMatrix();
glTranslatef(actualTurtle->px2[i],actualTurtle->py2[i],actualTurtle->pz2[i]);
glBegin(GL_POLYGON);
glutSolidSphere(0.1,50,10);
glEnd();
glPopMatrix();
```

```

    }
    break;
case 2: ESTE CREA UN RASTRO EN FORMA DE RAMPA CON PENDIENTE ALTERNATIVA (1 O -1)
glBegin(GL_QUAD_STRIP);
for (i = actualTurtle->np1[j]; i < actualTurtle->np2[j]; i++) {

glColor3f(0,0,0);
glVertex3f(actualTurtle->px2[i],actualTurtle->py2[i]+a,actualTurtle->pz2[i]);
glColor3f(0,0,0);
glVertex3f (actualTurtle->px4[i],actualTurtle->py4[i]+a,actualTurtle->pz4[i]);
a=pow(-1,i)*0.1;

}
glEnd();
break;
case 4: CREA UN RASTRO EN ZIG ZAG
glBegin(GL_QUAD_STRIP);
for (i = actualTurtle->np1[j]; i < actualTurtle->np2[j]; i++) {

glColor3f(0,0,0);
glVertex3f(actualTurtle->px2[i]+a,actualTurtle->py2[i],actualTurtle->pz2[i]+a);
glColor3f(0,0,0);
glVertex3f (actualTurtle->px4[i]+a,actualTurtle->py4[i],actualTurtle->pz4[i]+a);
a=pow(-1,i)*0.3;

}
glEnd();
break;
case 5: CREA EL RASTRO TRON, ES DECIR, UNA PARED QUE SIGUE EL MOVIMIENTO
glBegin(GL_QUAD_STRIP);
for (i = actualTurtle->np1[j]; i < actualTurtle->np2[j]; i++) {

glColor3f(0,1,0);
glVertex3f (actualTurtle->px2[i],actualTurtle->py2[i]+0.3,actualTurtle->pz2[i]);
glColor3f(0,1,0);
glVertex3f (actualTurtle->px2[i],actualTurtle->py2[i],actualTurtle->pz2[i]);
a=pow(-1,i)*0.3;

}
glEnd();
break;

}

if( lightingFlag ) glEnable( GL_LIGHTING ); VOLVEMOS A ACTIVAR LAS LUCES

}

```

En la funcion parseCommand tenemos que añadir las funciones penup,pendown y settrace i, esto es lo que habria que añadir:

```

}else if (!strcmp("settrace",strToken0)){ ESTO PARA ELEGIR EL TRAZO,HABRIA QUE
trazoelegido=val; AÑADIRLO EN LOS BUCLES CON ARGUMENTO

}else if(strToken0 != NULL && !strcmp("pu",strToken0)) { ESTO PARA PU,PD Y PONERLO EN
rastros=1; LOS BUCLES SIN ARGUMENTO QUE
}else if(strToken0 != NULL && !strcmp("pd",strToken0)) { ESTAN AL FINAL DEL P.COMMAND
rastros=0;
}else{
rastros=3;
}
}

```

Comando RASTRO DIFUMINADO(3 puntos)

El objetivo es que solo muestre los ultimos puntos de cada rastro, por ejemplo, los ultimos 5 puntos. Para ello habria que agregar un bucle a cada rastro de tal modo que si la tortuga esta en el punto np, solo dibujara los puntos np-5,np-4,np-3,np-2,np-1 que serian las ultimas cinco posiciones.

```
for (z=np-5;z<np;z++){
    //Aqui iria el modelo de rastro de cada uno
}
```

La funcion habria que llamarla desde el ParseCommand asi que incluiriamos en la zona donde utiliza argumento lo siguiente:

```
}else if (!strcmp("settraceblur",strToken0)){
    trazoelegido=val; ESTO PARA QUE ASIGNE RASTRO
    blur=1; ESTO PARA QUE ACTIVE DIFUMINADO
}
```

Habria que crear la variable global int blur, que es la que valdria 1 si esta activado el difuminado y 0 si no esta activado.

Comandos CLEAN, HOME y CLEARSCREEN(1 punto)

La estructura esta basada en que estan diseñado el sistema de rastros como hemos indicado en el punto anterior

```
if (strToken0 != NULL && !strcmp("clear",strToken0)){
    trace=1; CON ESTO BORRA LO QUE HAY DIBUJADO
    actualTurtle->np1[0]=actualTurtle->np-1;
    CON ESTO DEJA LA POSICION GUARDADA EN LA QUE ESTA PARA LUEGO SEGUIR DIBUJANDO RASTRO POR EL MISMO LADO.
} else if (strToken0 != NULL && !strcmp("home",strToken0)) {
    printf("HOME");
    for (i=1;i<10000;i++){ UN BUCLE QUE REINICIA LOS VECTORES QUE GUARDAN LOS PUNTOS
    actualTurtle-> px [i] = 0;
    actualTurtle-> py [i] = 0;
    actualTurtle-> pz [i] = 0;
    actualTurtle-> px2 [i]= 0.0;
    actualTurtle-> py2[i]=0;
    actualTurtle-> pz2[i]=0;
    actualTurtle-> px3 [i]=-0.0;
    actualTurtle-> py3[i]=0;
    actualTurtle-> pz3[i]=0;
    actualTurtle-> px4 [i]=0.0;
    actualTurtle-> py4[i]=0;
    actualTurtle-> pz4[i]=0;
    actualTurtle-> px5[i]=-0;
    actualTurtle-> py5[i]=0;
    actualTurtle-> pz5[i]=0;
    actualTurtle->np=0; TAMBIEN TENEMOS QUE REINICIAR LOS 3 VECTORES QUE UTILIZAMOS
    actualTurtle->np1[i]=0; PARA DIBUJAR RASTROS.
    actualTurtle->np2[i]=0;
```

```

    }
    glGetDoublev (GL_MODELVIEW_MATRIX, actualTurtle->mModel);
    glLoadIdentity();
}

```

EL COMANDO CLEARSCREEN QUE ES LA SUMA DE LOS DOS SIMPLEMENTE PONER QUE EJECUTE LOS DOS.

```

}else if(strToken0 != NULL && !strcmp("cs",strToken0)) {
    parseCommand("clear");
    parseCommand("home");
}

```

Comando BACKGROUND (1 punto)

Introducir en la funcion parseCommand

Variables:

```

int u =0;
double c[3];

```

Justo antes del while

```

if (!strcmp("background",strToken0)){
    //strToken1 = strtok(NULL," ");
    //printf("strtoken1:  ", strToken1);
    while ((strToken1 = strtok(NULL," ")) != NULL) {
printf("Token: %s\n",strToken1);
        c[u]=atof(strToken1)/255;
        u=u+1;
    }

    glClearColor(c[0], c[1], c[2], 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutSwapBuffers();
}

```

Con esto funciona en el archivo descargado del ADI

Habria que introducir la secuencia "background r g b" siendo r,g,b la cantidad de cada color entre 0 y 255.

Comando SUPERPOSEON, SUPERPOSEOFF y CLEARSCENE (1 punto)

En la funcion parseCommand habria que poner en la parte donde se leen los comandos sin argumento

```

}else if(strToken0 != NULL && strcmp(strToken0, "superposeon",11) == 0) {
    super=1;
}else if (strToken0 != NULL && strcmp(strToken0, "superposeoff",11) == 0) {
    super=0;
}else if (strToken0 != NULL && strcmp(strToken0, "clearscene",11) == 0) {
    super=3;
}

```

Hemos declarado la variable global "super".

El encabezado de la funcion display queda como sigue:

```

if (super==0){

    if(autoClear)glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}

```

```

        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
    }else if(super==3){
        if(autoClear)glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
        super=1;

    }else if(super==1){
        if (autoClear) glClear(GL_DEPTH_BUFFER_BIT);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING);
    }
}

```

Lectura Fichero VRML (3 puntos)

Esta manera de leer VRML funciona de la siguiente manera. Tenemos el objeto y desde "Simbolo del sistema" con el archivo cxc.exe convertimos esa imagen en varias funciones y una serie de coordenadas. Esta funcion y la serie de coordenadas las incluimos en el tecnunlogo y despues incluimos el main para que llame a la funcion. Con esto queda conseguido que lea cualquier fichero VRML.

La funcion que lee la imagen es la siguiente (he tomado como ejemplo de una figura no muy complicada ya que en algunos casos de figuras complicadas esta funcion ha llegado a ocupar 1000-2000 lineas.

```

void draw_model(void)
{
    /* Bounding Box */
    /* bboxCenter 0 1.95 0 */
    /* bboxSize 3 4.1 3 */
    glPushMatrix();
        glTranslatef(0, 1.95, 0);
        glColor3fv(v3f_table[8]);          /* <1, 1, 1> */
        glPushClientAttrib(GL_CLIENT_VERTEX_ARRAY_BIT);
        glInterleavedArrays(GL_V3F, 0, v3f_table);
        glDrawElements(GL_LINE_LOOP,      4,      GL_UNSIGNED_BYTE,
indices_table);
        glDrawElements(GL_LINE_LOOP,      4,      GL_UNSIGNED_BYTE,
indices_table + 4);
        glDrawElements(GL_LINE_LOOP,      4,      GL_UNSIGNED_BYTE,
indices_table + 8);
        glDrawElements(GL_LINE_LOOP,      4,      GL_UNSIGNED_BYTE,
indices_table + 12);
        glDrawElements(GL_LINE_LOOP,      4,      GL_UNSIGNED_BYTE,
indices_table + 16);
        glDrawElements(GL_LINE_LOOP,      4,      GL_UNSIGNED_BYTE,
indices_table + 20);
        glPopClientAttrib();
    glPopMatrix();
    /* Transform */
    glPushMatrix();
        /* Shape */
        /* Appearance */
        /* Material */
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, v4f_table[3]);
        /* <0.2, 0.2, 0.1, 1> */

```

```

        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, v4f_table[4]);
/* <1, 1, 0.5, 1> */
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 25.600000);
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, v4f_table[0]);
        /* <0, 0, 0, 1> */
        glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, v4f_table[0]);
        /* <0, 0, 0, 1> */
        glColor4fv(v4f_table[4]);          /* <1, 1, 0.5, 1> */
        glEnable(GL_LIGHTING);
        /* Box */
        glEnable(GL_CULL_FACE);
        glPushClientAttrib(GL_CLIENT_VERTEX_ARRAY_BIT);
        glInterleavedArrays(GL_N3F_V3F, 0, v6f_table);
        glDrawElements(GL_QUADS,          24,          GL_UNSIGNED_BYTE,
indices_table + 24);
        glPopClientAttrib();
        glDisable(GL_CULL_FACE);
        glDisable(GL_LIGHTING);
glPopMatrix();
/* Transform */
glPushMatrix();
        glTranslatef(0, 3, 0);
        /* Shape */
        /* Appearance */
        /* Material */
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, v4f_table[5]);
/* <0.1, 0.1, 0.2, 1> */
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, v4f_table[6]);
/* <0.5, 0.5, 1, 1> */
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 25.600000);
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, v4f_table[0]);
        /* <0, 0, 0, 1> */
        glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, v4f_table[0]);
        /* <0, 0, 0, 1> */
        glColor4fv(v4f_table[6]);          /* <0.5, 0.5, 1, 1>
*/

        glEnable(GL_LIGHTING);
        /* Sphere */
        gluQuadricTexture(qobj, GL_FALSE);
        gluQuadricNormals(qobj, GL_SMOOTH);
        glPushMatrix();
                glRotatef(-90, 1, 0, 0);
                glEnable(GL_CULL_FACE);
                gluSphere(qobj, 1, 10, 10);
                glDisable(GL_CULL_FACE);
        glPopMatrix();
        glDisable(GL_LIGHTING);
glPopMatrix();
glDisable(GL_LIGHT0);
glFlush();

}

```

Las coordenadas que habria que agregar para que dibuje esa figura sencilla serian:

```

GLfloat v3f_table[][3] = {
    {-1.5, -2.05, -1.5}, {-1.5, -2.05, 1.5}, {-1.5, 2.05, 1.5}, {-
1.5, 2.05, -1.5},
    {1.5, -2.05, 1.5}, {1.5, 2.05, 1.5}, {1.5, -2.05, -1.5}, {1.5,
2.05, -1.5}, {1,

```

```

        1, 1}
};

GLfloat v4f_table[][4] = {
    {0, 0, 0, 1}, {1, 1, 1, 1}, {0, 0, 1, 0}, {0.2, 0.2, 0.1, 1},
    {1, 1, 0.5, 1}, {0.1,
    0.1, 0.2, 1}, {0.5, 0.5, 1, 1}
};

GLfloat v6f_table[][6] = {
    {-1, 0, 0, -1.5, -0.1, -1.5}, {-1, 0, 0, -1.5, -0.1, 1.5}, {-1,
    0, 0, -1.5, 0.1,
    1.5}, {-1, 0, 0, -1.5, 0.1, -1.5}, {0, 0, 1, -1.5, -0.1, 1.5},
    {0, 0, 1, 1.5, -0.1,
    1.5}, {0, 0, 1, 1.5, 0.1, 1.5}, {0, 0, 1, -1.5, 0.1, 1.5}, {1,
    0, 0, 1.5, -0.1,
    1.5}, {1, 0, 0, 1.5, 0.1, -1.5}, {1,
    0, 0, 1.5, 0.1,
    1.5}, {0, 0, -1, 1.5, -0.1, -1.5}, {0, 0, -1, -1.5, -0.1, -1.5},
    {0, 0, -1, -1.5,
    0.1, -1.5}, {0, 0, -1, 1.5, 0.1, -1.5}, {0, 1, 0, -1.5, 0.1,
    1.5}, {0, 1, 0, 1.5,
    0.1, 1.5}, {0, 1, 0, 1.5, 0.1, -1.5}, {0, 1, 0, -1.5, 0.1, -
    1.5}, {0, -1, 0, -1.5,
    -0.1, -1.5}, {0, -1, 0, 1.5, -0.1, -1.5}, {0, -1, 0, 1.5, -0.1,
    1.5}, {0, -1, 0,
    -1.5, -0.1, 1.5}
};

GLubyte indices_table[48] = {
    0, 1, 2, 3, 1, 4, 5, 2, 4, 6, 7, 5, 6, 0, 3, 7, 2, 5, 7, 3, 0,
    6, 4, 1, 0, 1, 2,
    3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
    21, 22, 23
};

GLfloat viewpoints[][16] = {

    /* [0] Entry Viewpoint */
    {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, -10, 1}
};

```

Tambien habria que agregar alguna funcion mas. Hay alguna pequeña que no se muy bien su funcion pero que son necesarias, tambien nos la ofreceria el programa cxc.exe cuando transforma el archivo. Hay una importante que es la funcion init(), que es la que llama a Draw_Model () para que dibuje y su sintaxis es la siguiente:

```

void init(void) ESTA ES LA FUNCION QUE LLAMA A DRAW_MODEL PARA
{
    QUE DIBUJE LA FIGURA

    qobj = gluNewQuadric();
    gluQuadricDrawStyle(qobj, GLU_FILL);

    glClearColor(0, 0, 0, 0);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glCullFace(GL_BACK);
    glClearDepth(1);

```

```

    glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);
    glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST);
    /* set perspective correction parameter to GL_FASTEST to
increase texture mapping performance */
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    /* create display list of model */
    compiled_model = glGenLists(1);
    glNewList(compiled_model, GL_COMPILE);
    draw_model();
    glEndList();
}

```

Leer archivos 3DS

Habría que agregar unos pocos archivos y definir algunas variables globales. La cabecera del programa quedaría así:

```

#include <GL/glut.h>
#include "camera.h"
#include "light.h"
#include "primitivas.h"
#include "Vector_tools.h"
#include <stdio.h>

#include "glmObject.h"
#include <GL/glut.h>
#include <stdio.h>
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <lib3ds/file.h>
#include <lib3ds/cameras.h>
#include <lib3ds/mesh.h>
#include <lib3ds/node.h>
#include <lib3ds/material.h>
#include <lib3ds/matrix.h>
#include <lib3ds/vector.h>
#include <lib3ds/lights.h>
#include <string.h>
#include <config.h>
#include <stdlib.h>
#include <math.h>
#ifdef WITH_DMALLOC
#include <dmalloc.h>
#endif

boolean command = FALSE; /* command mode */
char strCommand[256];
static const char* filename=0;
//static const char* camera=0;
static Lib3dsFile *file=0;
static float current_frame=0.0;
static int gl_width;
static int gl_height;
static int camera_menu_id=0;

```

```

static int halt=0;
int gti=0;

/*
 * forward declarations
 */

GLfloat v3f_table[][3]; /* R3 color, normal & vertex data */
GLfloat v4f_table[][4]; /* R4 color & vertex data */
GLfloat v6f_table[][6]; /* N3F_V3F vertex array data */
GLubyte indices_table[]; /* global vertex array indices */
GLfloat viewpoints[][16]; /* view matrix data */
GLUquadricObj * qobj; /* used to render quadric geometry (Sphere,
Cylinder, Cone) */

static camera *LOCAL_MyCamera;
static light **LOCAL_MyLights;
static int current_mode = 0;
static int current_light = -1;
static int spot_move = 0;

static int old_x, old_y;

//boolean command = FALSE; /* command mode */
boolean autoClear = TRUE; /* auto clear mode */
char strCommand[256];

```

La funcion que dibuja la figura es render_node que tendria la siguiente sintaxis:

```

static void render_node(Lib3dsNode *node)
{
    ASSERT(file);

    {
        Lib3dsNode *p;
        for (p=node->childs; p!=0; p=p->next) {
            render_node(p);
        }
    }
    if (node->type==LIB3DS_OBJECT_NODE) {
        if (strcmp(node->name, "$$$DUMMY")==0) {
            return;
        }

        if (!node->user.d) {
            Lib3dsMesh *mesh=lib3ds_file_mesh_by_name(file, node->name);
            ASSERT(mesh);
            if (!mesh) {
                return;
            }

            node->user.d=glGenLists(1);
            glNewList(node->user.d, GL_COMPILE);

            {
                unsigned p;
                Lib3dsVector *normalL=malloc(3*sizeof(Lib3dsVector)*mesh->faces);

                {

```

```

    Lib3dsMatrix M;
    lib3ds_matrix_copy(M, mesh->matrix);
    lib3ds_matrix_inv(M);
    glMultMatrixf(&M[0][0]);
}
lib3ds_mesh_calculate_normals(mesh, normalL);

for (p=0; p<mesh->faces; ++p) {
    Lib3dsFace *f=&mesh->faceL[p];
    Lib3dsMaterial *mat=0;
    if (f->material[0]) {
        mat=lib3ds_file_material_by_name(file, f->material);
    }

    if (mat) {
        static GLfloat a[4]={0,0,0,1};
        float s;
        glMaterialfv(GL_FRONT, GL_AMBIENT, a);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat->diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat->specular);
        s = pow(2, 10.0*mat->shininess);
        if (s>128.0) {
            s=128.0;
        }
        glMaterialf(GL_FRONT, GL_SHININESS, s);
    }
    else {
        Lib3dsRgba a={0.2, 0.2, 0.2, 1.0};
        Lib3dsRgba d={0.8, 0.8, 0.8, 1.0};
        Lib3dsRgba s={0.0, 0.0, 0.0, 1.0};
        glMaterialfv(GL_FRONT, GL_AMBIENT, a);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, d);
        glMaterialfv(GL_FRONT, GL_SPECULAR, s);
    }
    {
        int i;
        glBegin(GL_TRIANGLES);
        glNormal3fv(f->normal);
        for (i=0; i<3; ++i) {
            glNormal3fv(normalL[3*p+i]);
            glVertex3fv(mesh->pointL[f->points[i]].pos);
        }
        glEnd();
    }
}

free(normalL);
}

glEndList();
}

if (node->user.d) {
    Lib3dsObjectData *d;

    glPushMatrix();
    d=&node->data.object;
    glMultMatrixf(&node->matrix[0][0]);
    glTranslatef(-d->pivot[0], -d->pivot[1], -d->pivot[2]);
    glCallList(node->user.d);
    /*glutSolidSphere(50.0, 20,20);*/
}

```

```

        glPopMatrix();
    }
}
}

```

En la funcion display() habria que crear las siguientes variables:

```

Lib3dsNode *p;
Lib3dsNode *c,*t;
Lib3dsMatrix M;

```

y en las lineas que dibujan la tortuga o el *.obj crear una bifurcacion para que no las dibuje en el caso de que leyeramos *.3ds

```

if(gti==0){
    if (turtleVector[i]->object != NULL)

        glCallList(turtleVector[i]->object->model_list);

    else
        drawTurtle();
        glPopMatrix();
}

```

En este caso dibujaria un archivo *.obj o la tortuga simple y en caso de que estuvieramos leyendo un objeto *.3ds la funcion que dibuja es distinta por lo que habria que agregar esto:

```

if (gti==1){
    for (p=file->nodes; p!=0; p=p->next) {
        render_node(p);
    }
}

```

El parte del main donde lee el archivo *.obj habria que cambiarla para que leyera tambien los archivos *.3DS

```

if (argv[1][strlen(argv[1])-1]=='s'){
HACEMOS LA DISTINCION ENTRE OBJ Y 3DS PARA QUE LEA UNO U OTRO
BASANDONOS EN LA ULTIMA LETRA DE LA EXTENSION.EN ESTE CASO SERIA .3DS
    gti=1;
    filename=argv[1];
    file = lib3ds_file_load(filename);
LLAMAMOS A LA FUNCION QUE HACE LA LECTURA DEL ARCHIVO
    actualTurtle->pepe=file;ESTABLECEMOS CUAL ES NUESTRO OBJETO.
}

```

```

lib3ds_file_eval(file,0);
UNA FUNCION QUE COMPRUEBA QUE EL ARCHIVO ESTA OK.
}else{

    actualTurtle->object = createGlmObject(argv[1]);
}CASO DE QUE EL ARCHIVO NO SEA 3DS PUES ES *.OBJ Y LO LEE TAMBIEN

```