

7. LEYENDO OBJETOS: OTRAS TORTUGAS Y EL ESCENARIO

Hasta ahora, en la aplicación tecnunLogo se han utilizado para representar a la tortuga primitivas de OpenGL u objetos generados en el propio código del programa. Si se desea representar objetos con más detalles de modelado lo más lógico es generarlos con aplicaciones de modelado y leerlos en la aplicación. En este capítulo se explica el modo de leer y representar objetos en formato Wavefront. Posteriormente se generaliza el control y representación de una tortuga a un número variable de las mismas.

7.1 REPRESENTACIÓN DE UN OBJETO EN FORMATO WAVEFRONT

Existen multitud de formatos para representar un objeto en tres dimensiones. Uno de ellos es el formato Wavefront.

<http://www.tnt.uni-hannover.de/js/soft/compgraph/fileformats/docs/OBJ.format.txt>

Para leer dichos formatos se va a utilizar la librería glm.

http://www.opengl.org/developers/code/examples/more_samples/smooth.zip

Se deben añadir los ficheros glm.c y glm.h al directorio del proyecto y añadirlos al proyecto (Project / Add to Project / Files ...).

Además de la librería se han creado unas funciones para facilitar el uso de dicha librería, estas funciones se encuentran en los ficheros glmObject.c y glmObject.h que también se deben copiar al directorio del proyecto y añadirlos al proyecto.

Incluir en tecnunLogo el include de glmObject.h, no es necesario el de glm.h puesto que lo incluye glmObject.h:

```
#include "glmObject.h"
```

Incluir el puntero al objeto que se va a leer:

```
static glmObject *object;
```

Incluir en la función main la comprobación de que existe el argumento número 1 que es el que va a proporcionar el nombre del fichero en el que se encuentra el objeto a representar:

```
if (!argv[1]) {  
    fprintf(stderr, "usage: tecnunLogo model_file.obj\n");  
    exit(1);  
}
```

Incluir en la función main la llamada a la función createGlmObject() que realiza la lectura del fichero, esta función se puede leer en glmObject.c:

```
object = createGlmObject(argv[1]);
```

Substituir el dibujo actual de la tortuga:

```
drawSphereTurtle();
```

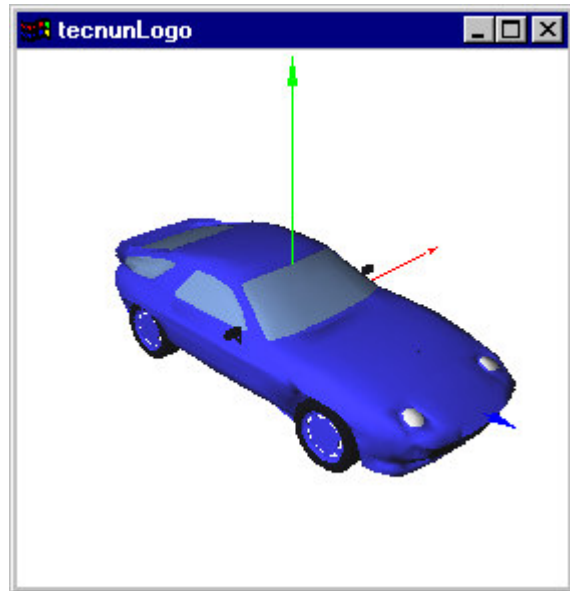
por la representación del objeto que se ha leído:

```
glCallList(object->model_list);
```

En modo de debugger, para indicar el nombre del primer argumento, se debe incluir en la configuración del proyecto (Project /Settings / Debug/ Program arguments:). Proporcionar por ejemplo el valor:

```
../data/porsche.obj
```

En la figura 1 se puede apreciar el aspecto de la aplicación con el objeto representado e iluminado:



7.2 VARIAS TORTUGAS

Se va a implementar a continuación el comando SETTURTLE *i*, que activa un número de objeto *i*, pudiendo ir *i* de 0 a *n*. Si la tortuga *i* no está creada se crea en ese momento. Para asociar un objeto a la tortuga se va a implementar el comando OBJECTLOAD *fileName*.

Para manejar varias tortugas se crea la estructura *turtle*, que contiene la información de cada una de las posibles tortugas en el escenario. La declaración se introduce con el resto de declaraciones en el fichero *tecnunLogo.c*

```
typedef struct _turtle
{
    int np;
    float px [10000];
    float py [10000];
    float pz [10000];
    glmObject *object;
    GLdouble mModel[16];
} turtle;
```

Se crean variables para almacenar el número de tortugas existentes, un vector para contener todas las tortugas posibles (256 en este caso) y un puntero a la tortuga actual, la designada por SETTURTLE *i*.

```
int nturtles;
turtle *turtleVector[256];
turtle *actualTurtle;
```

En la función *main* se inicializa el vector de tortugas y la primera tortuga (posición 0) con el valor del primer argumento, como se ha hecho en el primer apartado. En este caso se inicializa también la matriz *mModel* de la primera tortuga:

```
turtleVector[0] = (turtle*) malloc(sizeof(turtle));
turtleVector[0]->np = 0;
actualTurtle = turtleVector[0];
nturtles = 1;
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glLoadIdentity();
```

```

glGetDoublev (GL_MODELVIEW_MATRIX, actualTurtle->mModel);
glPopMatrix();
actualTurtle->object = createGlmObject(argv[1]);

```

En display se debe realizar un bucle para todas las tortugas existentes:

```

Ejes_World(); // ejes globales
for (i=0; i < nturtles; i++){
    if (turtleVector[i] != NULL) {
        glColor3f(1.0,1.0,0.0) ;
        displayTrace(turtleVector[i]);
        glPushMatrix();
        glMultMatrixd(turtleVector[i]->mModel);
        Ejes_World(); // ejes locales
        glColor3f(1.0,1.0,0.0) ;
        if (turtleVector[i]->object != NULL)
            glCallList(turtleVector[i]->object->model_list);
        else
            drawTurtle();
        glPopMatrix();
    }
}

```

En la interpretación de los comandos se deben añadir los comandos SETTURTLE y OBJECTLOAD, abreviados por st y ol:

```

} else if (!strcmp("st",strToken0)) {
    printf("SETTURTLE");
    ival = val;
    actualTurtle = turtleVector[ival];
    if (actualTurtle == NULL) {
        turtleVector[ival] = (turtle*) malloc(sizeof(turtle));
        turtleVector[ival]->np = 0;
        turtleVector[ival]->object = NULL;
        actualTurtle = turtleVector[ival];
        if ((ival + 1) > nturtles) nturtles = ival + 1;
        glLoadIdentity();
        glGetDoublev (GL_MODELVIEW_MATRIX, actualTurtle->mModel);
    }
    glLoadIdentity();
    glMultMatrixd(actualTurtle->mModel);
} else if (!strcmp("ol",strToken0)) {
    printf("OBJECTLOAD");
    actualTurtle->object = createGlmObject(strToken1);
}

```

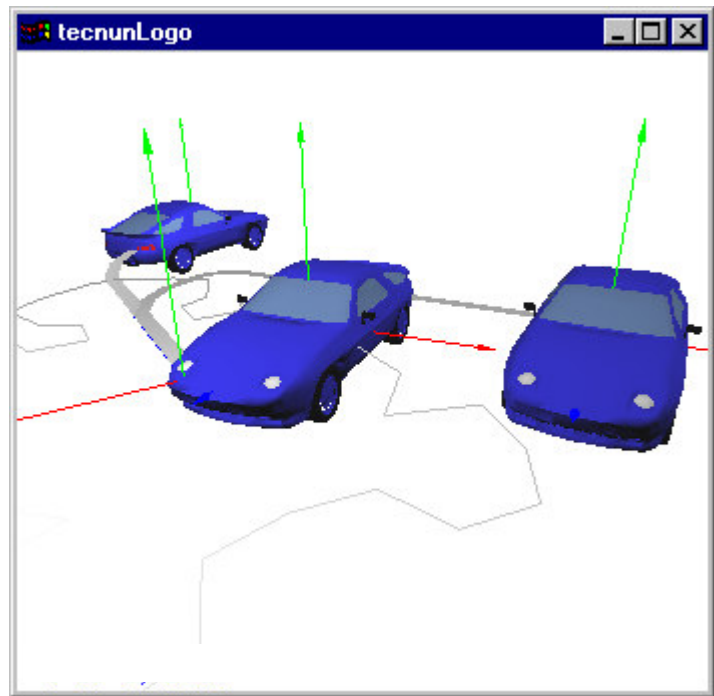
Se deben modificar además las funciones addPointToTrace(), displayTrace() y parseCommand().

En addPointToTrace() los puntos se añaden a la tortuga actual, ya que la variable np y los vectores px, py y pz pertenecen ahora a la estructura turtle.

La función displayTrace(turtle* turtlei) dibuja el rastro de la tortuga que se le pasa como parámetro, que es la que contiene la información de los puntos (np, px, py y pz).

En parseCommand(char* command), se manipula la matriz de la tortuga actual (actualTurtle->mModel).

El resultado se muestra en la figura 2, en la que se han introducido 4 tortugas, 3 de ellas con el objeto de un vehículo.



7.3 TRABAJOS PROPUESTOS

- *Definir un modo (MOVEBOX ON / MOVEBOX OFF), en el que al mover la escena, los objetos se representen por cajas.*
- *Actualmente los objetos se escalan para que tengan una dimensión de uno. Realizar esta operación si el modo autoescala está en ON y no realizarlo cuando está en OFF. El comando es AUTOSCALE ON / AUTOSCALE OFF.*
- *Permitir tener n luces, (estará limitado a las 8 que permite OpenGL) cada una de las cuales puede ser de uno de los tres tipos posibles (direccional, posicional o spotlight). Para ello se debe utilizar una estructura similar al vector de tortugas utilizado en este capítulo.*