

Gráficos por Computador y Multimedia

TecnunLogo: Documentación sobre ejercicios

Iker Belloso Loidi
900484

Comando HI DETURTLE y SHOWTURTLE

Declarar al inicio del código una variable global booleana que indica si se debe dibujar o no la tortuga:

```
BOOL HideTurtle=1; ///'1' show turtle - '0' hide turtle
```

Añadir en la función ParseCommand tras el último if en el que se compara strToken0 con home:

```
else if (strToken0 != NULL && !strcmp("ht",strToken0)) { // DOWN
    HideTurtle=0;
    strToken0 = strtok(NULL, " ");
    display();
} else if (strToken0 != NULL && !strcmp("st",strToken0)) { // DOWN
    HideTurtle=1;
    strToken0 = strtok(NULL, " ");
    display();
}
```

Por último, en la función display, cuando se llama a la función que dibuja la tortuga, incluir

```
if (HideTurtle){
    if(TurtleType) drawTurtle();
    else drawSphereTurtle();
}
```

En caso de que la tortuga sea un objeto, el comando if se incluirá antes de llamada a la lista del objeto.

De modo que solamente cuando HideTurtle sea igual a 1 dibujará la tortuga.

El if que determina si se llama a drawTurtle() o a drawSphereTurtle() permite cambiar el tipo de tortuga.

Se define una variable global TurtleType:

```
BOOL TurtleType=1; ///'1' plane turtle - '0' sphere turtle
```

Y después en la función keyboard he añadido la identificación de la letra 'k', haciendo variar el estado actual de la variable TurtleType. (dentro del switch(key))

```
case 't':
case 'T':
    TurtleType=!TurtleType;
    break;
```

Comando SETCIRCLE:

Primero declarar una variable global que contendrá el valor del giro completo:

```
float CircleCicle= 360.0;
```

(se pone como valor por defecto 360-Grados) después se añade en la función ParseCommand las líneas que permitirán identificar el comando, en el if que identifica todos los comandos con argumento añadidos

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

Por último en todas las funciones glRotatef() sustituiremos val por val*360/CircleCicle de ese modo se hace un cambio de la cantidad de giro en la escala definida a grados.

Ejes del mundo y de la tortuga:

Añadir la siguiente función cuyos argumentos son las coordenadas del centro del eje de coordenadas:

```
void AxisDrawing(float X, float Y, float Z) {

    glBegin(GL_LINES);
        //Axis X
        glColor3f(1.0 ,0.0 ,0.0);
        glVertex3f(X, Y , Z);
        glVertex3f(X+1.0, Y , Z);
        //Axis Y
        glColor3f(0.0 ,1.0 ,0.0);
        glVertex3f(X, Y , Z);
        glVertex3f(X, Y+1.0 , Z);
        //Axis Z
        glColor3f(0.0, 0.0 , 1.0);
        glVertex3f(X, Y , Z);
        glVertex3f(X,Y , Z+1.0);

    glEnd();
}
```

```

//Draw x a
glColor3f(1.0,0.0,0.0);
glTranslatef(X+1,0.0,0.0);
glRotatef(90.0,0.0,1.0,0.0);
glutSolidCone(0.03,0.1,100,100);
glRotatef(-90.0,0.0,1.0,0.0);
glTranslatef(-(X+1),0.0,0.0);
//Eje Y
glColor3f(0.0,1.0,0.0);
glTranslatef(0.0,Y+1,0.0);
glRotatef(-90.0,1.0,0.0,0.0);
glutSolidCone(0.03,0.1,100,100);
glRotatef(90.0,1.0,0.0,0.0);
glTranslatef(0.0,-(Y+1),0.0);
//Eje Z
glColor3f(0.0,0.0,1.0);
glTranslatef(0.0,0.0,Z+1);
glutSolidCone(0.03,0.1,100,100);
glTranslatef(0.0,0.0,-(Z+1));
}

```

Para dibujar unos ejes en el origen de la tortuga, realizaremos una llamada a la función dentro de la función display después de haber hecho pushMatrix y para dibujar el eje global, llamaremos a la función después de la llamada a popMatrix. Yo también he añadido una variable global BOOL AxisState=1; que define si se deben dibujar o no los ejes, por ello también aparecen unos if que controlan la llamada o no a la función AxisDrawing.

```

glMultMatrixd(mModel);
if(AxisState) AxisDrawing(0.0,0.0,0.0);
...
...
glPopMatrix();
if(AxisState)AxisDrawing(0.0,0.0,0.0);
...
...

```

El cambio en la variable AxisState se hace mediante la lectura de teclado, en la función keyboard he añadido la identificación de la tecla 'a' para cambiar el estado actual de la variable: (dentro del switch(key))

```

case 'a':
case 'A':
    AxisState =! AxisState;
    break;

```

Posibilidad de modificar dinámicamente el tamaño de los ejes al objeto

Además, para hacer que estos ejes tengan un tamaño acuerdo con el tamaño de la tortuga en cada caso, se puede hacer que los tamaños de estos ejes sean variables:

Para empezar, en la definición de la función AxisDrawing se debe añadir tras los argumentos

```
float X, float Y, float Z
```

```
el argumento
```

```
,float size
```

A continuación, en cada vértice que se define, así como en las características de los los conos, se debe sustituir 1 por la variable size, esto es, sustituir

```
X+1
```

```
Y+1
```

```
Z+1
```

por

```
X+size
```

```
Y+size
```

```
Z+size
```

respectivamente.

Se debe definir una variable global que almacene el tamaño de los ejes e inicializarla a un valor por defecto:

```
FLOAT ASize=1.5;
```

Finalmente, en la función Keyboard se debe añadir la lectura de la tecla + y – que harán aumentar y disminuir el tamaño de los ejes. Dentro del switch(Key) introducimos:

```
case '+':
```

```
    ASize=ASize+0.1;
```

```
    break;
```

```
case '-':
```

```
    ASize=ASize-0.1;
```

```
    break;
```

Comando LOAD:

Declarar una variable global en la que se almacena el Path del archivo que contiene los comandos:

```
const char filename[50]="C:\\commands.txt";
```

Declarar variables necesarias para abrir y leer el archivo dentro de la función parseCommand:

```
int NumRead; // Número de bytes leídos en la llamada a gets
char ReadedCommand[256]; //Vector en el que se almacenan los
caracteres leídos
BOOL read=1; // Variable booleana que controla el while y pasa a 0 cuando
FILE *stream; //handle al file
```

Después añadimos un nuevo else if en la función parseCommand después del if para el home

```
else if (strToken0 != NULL && !strcmp("load",strToken0)) { // SETCIRCLE
//Opening the instruction file
if( (stream = fopen(filename, "r" )) == NULL ) printf( "The file 'data' was not
opened\n" );
else {
printf( "The file 'data' was opened\n" );
while(read){
for(i=0; i<sizeof(ReadedCommand);i++)ReadedCommand[i]=0;
NumRead=fgets( ReadedCommand, 256, stream );
if(NumRead==NULL){
read =0;
}
else{
repeatCommand=ReadedCommand;
nextCommand = repeatCommand +
strlen(repeatCommand) + 1;
strcpy (parseCommandInit, repeatCommand);
parseCommand(parseCommandInit);
strToken0 = strtok(nextCommand, " ");
if (strToken0 == NULL) continue;
}
}
fclose( stream );
```

```
}  
}
```

Mediante este comando se ejecutarán los comandos escritos en el archivo commands.txt situado en el path definido anteriormente.

Definir distintos patrones de rastro:

Declaramos una variable int dentro de la estructura turtle que controlara el tipo de rastro a dibujar

```
int LineType=1;
```

Después declaramos tres variables que contendrán puntos paralelos a la línea de desplazamiento. Se incluirán dentro de la estructura tortuga.

```
float px2[10000], px3[10000];  
float py2[10000], py3[10000];  
float pz2[10000], pz3[10000];
```

Introducimos en la función addPointToTrace en el If (np=0) :

Permite dibujar tanto el rastro de líneas paralelas como el de cinta:

```
actualTurtle->px2 [0] = (float)0.05;  
actualTurtle->py2 [0] = (float)0;  
actualTurtle->pz2 [0] = (float)0;
```

Permite dibujar el rastro como el de las motos de la película Tron:

```
actualTurtle->px3 [0] =(float)0;  
actualTurtle->py3 [0] =(float)0.4;  
actualTurtle->pz3 [0] =(float)0;
```

En el else if añadimos:

```
actualTurtle->px2 [actualTurtle->np] = m[0] * actualTurtle->px2 [0] + m[4] *  
actualTurtle->py2 [0] + m[8] * actualTurtle->pz2 [0] + m[12];  
actualTurtle->py2 [actualTurtle->np] = m[1] * actualTurtle->px2 [0] +  
m[5] * actualTurtle->py2 [0] + m[9] * actualTurtle->pz2 [0] + m[13];  
actualTurtle->pz2 [actualTurtle->np] = m[2] * actualTurtle->px2 [0] +  
m[6] * actualTurtle->py2 [0] + m[10] * actualTurtle->pz2 [0] + m[14];  
actualTurtle->px3 [actualTurtle->np] = m[0] * actualTurtle->px3 [0] +  
m[4] * actualTurtle->py3 [0] + m[8] * actualTurtle->pz3 [0] + m[12];  
actualTurtle->py3 [actualTurtle->np] = m[1] * actualTurtle->px3 [0] +  
m[5] * actualTurtle->py3 [0] + m[9] * actualTurtle->pz3 [0] + m[13];  
actualTurtle->pz3 [actualTurtle->np] = m[2] * actualTurtle->px3 [0] +  
m[6] * actualTurtle->py3 [0] + m[10] * actualTurtle->pz3 [0] + m[14];
```

En la función `parseCommand` el comando de cambio de tipo de rastro, al `if` de comparación de comandos con argumento le añadimos:

```
    } else if (!strcmp("slt",strToken0)) { // SETTYPE
        actualTurtle->LineType=val;
    }
```

Si no se incluyen, se deben incluir llamadas a la función `addPointToTrace` en los comandos de giro.

Finalmente sustituimos la función `displayTrace` por la siguiente:

```
void displayTrace() {
    int i;
    glColor3f(1.0,1.0,1.0) ;

    switch(LineType){
    case(1):
        glColor3f(1.0,1.0,1.0) ;
        glBegin(GL_LINE_STRIP);
        for (i = 0; i < np; i++) {
            glVertex3f (px[i],py[i],pz[i]);
        }
        glEnd();
        glBegin(GL_LINE_STRIP);
        for (i = 0; i < np; i++) {
            glVertex3f (px2[i],py2[i],pz2[i]);
        }
        glEnd();
        break;
    case(2):
        glColor3f(0.5,1.0,0.5) ;
        glBegin(GL_QUAD_STRIP);
        for (i = 0; i < np; i++) {
            glVertex3f (px[i],py[i],pz[i]);
            glVertex3f (px2[i],py2[i],pz2[i]);
        }
        glEnd();
        break;
    case(3):
        glColor3f(1.0,0.0,0.0) ;
        glBegin(GL_QUAD_STRIP);
        for (i = 0; i < np; i++) {
            glVertex3f (px[i],py[i],pz[i]);
```

```

        glVertex3f (px3[i],py3[i],pz3[i]);
    }
    glEnd();
    break;

case(4):
    glColor3f(0.3,0.8,0.2) ;
    for (i = 0; i < np; i++) {
        glTranslatef(px[i],py[i],pz[i]);
        glutSolidSphere(0.05, 10, 10);
        glTranslatef(-px[i],-py[i],-pz[i]);
    }
    break;
}
}

```

Solución al problema de ejes sin colores:

Dado que el sistema de luces está activado cuando se dibuja el eje de la tortuga, estos aparecen sin sus colores característicos a no ser que las luces estén configuradas adecuadamente. Para solucionar este problema basta con añadir:

Al inicio de la función AxisDrawing:

```

int lightingFlag = glEnable( GL_LIGHTING );//variable en la que se
                                                almacena el estado de las
                                                luces
if( lightingFlag ) glDisable( GL_LIGHTING );//en caso de estar activadas se
                                                desactivan las luces

```

Al final de la función, volvemos a dejar el sistema de luces como estaba:

```

if( lightingFlag ) glEnable(GL_LIGHTING);

```

Problema de los trazos al añadir las diferentes tortugas:

Al añadir las diferentes tortugas guardadas en archivos .obj los trazos dejan de verse con su color original y solamente se ven desde una de sus caras. Para solucionarlo, se debe hacer lo siguiente. Añadir al comienzo de la función DisplayTrace:

```

int lightingFlag = gllsEnabled( GL_LIGHTING );//variable para almacenar el
                                estado de las luces
int Face= gllsEnabled(GL_CULL_FACE);//Variable para almacenar el
                                estado de si se dibujan ambas caras de
                                los objetos o no.
if( lightingFlag ) glDisable( GL_LIGHTING );
if(Face)glDisable(GL_CULL_FACE);

```

Al final de la función se restituye el estado inicial de las luces y el dibujo de caras.

```

if(Face) glEnable(GL_CULL_FACE);
if( lightingFlag ) glEnable(GL_LIGHTING);

```

Comando PENUP y PENDOWN

Primero declaramos una variable global de control y un vector dentro de la estructura turtle que controla que tramos son dibujados y cuales no:

```

int Pen=1;

int PenD[10000];

```

En la función addPointTo trace añadimos en los if-s

```

1 Turtlei->PenD[0]=Pen;
2 Turtlei->PenD[turtlei->np]=Pen;

```

En la función ParseCommand, tras el if que compara strToken0 con home añadimos:

```

else if (strToken0 != NULL && !strcmp("pu",strToken0)) { //PENUP
    Pen=0;
    strToken0 = strtok(NULL, " ");
    display();
} else if (strToken0 != NULL && !strcmp("pd",strToken0)) { // PENDOWN
    Pen=1;
    strToken0 = strtok(NULL, " ");
    display();
}

```

Sustituimos el switch dentro de la función displaytrace por:

```
switch(turtle->LineType){
    case(1):
        glColor3f(1.0,1.0,1.0) ;

        glBegin(GL_LINE_STRIP);
        for (i = 0; i < turtle->np-1; i++) {

            if(turtle->PenD[i+1]){
                glColor3f(1.0,1.0,1.0);
                glVertex3f (turtle->px[i],turtle->py[i],turtle->pz[i]);
                glVertex3f (turtle->px[i+1],turtle->py[i+1],turtle-
>pz[i+1]);
            }
            else{
                glEnd();
                glBegin(GL_LINE_STRIP);
            }
        }
        glEnd();
        glBegin(GL_LINE_STRIP);

        for (i = 0; i < turtle->np-1; i++) {
            if(turtle->PenD[i+1]){
                glColor3f(1.0,1.0,1.0);
                glVertex3f (turtle->px2[i],turtle->py2[i],turtle->pz2[i]);
                glVertex3f (turtle->px2[i+1],turtle->py2[i+1],turtle-
>pz2[i+1]);
            }
            else{
                glEnd();
                glBegin(GL_LINE_STRIP);
            }
        }
        glEnd();
        break;
    case(2):
        glColor3f(0.5,1.0,0.5) ;
        glBegin(GL_QUAD_STRIP);
        for (i = 0; i < turtle->np-1; i++) {
            if(turtle->PenD[i+1]){
                glColor3f(0.5,1.0,0.5);
                glVertex3f (turtle->px[i],turtle->py[i],turtle->pz[i]);
                glVertex3f (turtle->px2[i],turtle->py2[i],turtle->pz2[i]);
                glVertex3f (turtle->px[i+1],turtle->py[i+1],turtle-
>pz[i+1]);
```

```

        glVertex3f (turtlei->px2[i+1],turtlei->py2[i+1],turtlei-
>pz2[i+1]);
    }
    else{
        glEnd();
        glBegin(GL_QUAD_STRIP);
    }
}
glEnd();
break;

case(3):
    glColor3f(1.0,0.0,0.0) ;
    glBegin(GL_QUAD_STRIP);
    for (i = 0; i < turtlei->np-1; i++) {
        if(turtlei->PenD[i+1]){
            glColor3f(1.0,0.0,0.0) ;
            glVertex3f (turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
            glVertex3f (turtlei->px3[i],turtlei->py3[i],turtlei->pz3[i]);
            glVertex3f (turtlei->px[i+1],turtlei->py[i+1],turtlei-
>pz[i+1]);
            glVertex3f (turtlei->px3[i+1],turtlei->py3[i+1],turtlei-
>pz3[i+1]);
        }
        else{
            glEnd();
            glBegin(GL_QUAD_STRIP);
        }
    }
    glEnd();
    break;

case(4):
    glColor3f(0.3f,0.8f,0.2f) ;
    for (i = 0; i < turtlei->np; i++) {
        if(turtlei->PenD[i]){
            glColor3f(0.3f,0.8f,0.2f);
            glTranslatef(turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
            glutSolidSphere(0.05, 10, 10);
            glTranslatef(-turtlei->px[i],-turtlei->py[i],-turtlei->pz[i]);
        }
    }
    break;

```

Rastro con los colores del arco iris

Declaramos variables globales dentro de la estructura turtle:

```
float pxa1[10000], pxa2[10000],pxa3[10000],pxa4[10000];  
float pya1[10000], pya2[10000],pya3[10000],pya4[10000];  
float pza1[10000], pza2[10000],pza3[10000],pza4[10000];
```

En la función addPointToTrace añadimos en cada uno de las opciones del if respectivamente:

```
actualTurtle->pxa1 [0] = (float)0.01;  
actualTurtle->pya1 [0] = (float)0;  
actualTurtle->pza1 [0] = (float)0;
```

```
actualTurtle->pxa2 [0] = (float)0.02;  
actualTurtle->pya2 [0] = (float)0;  
actualTurtle->pza2 [0] = (float)0;
```

```
actualTurtle->pxa3 [0] =(float)0.03;  
actualTurtle->pya3 [0] =(float)0;  
actualTurtle->pza3 [0] =(float)0;
```

```
actualTurtle->pxa4 [0] =(float)0.04;  
actualTurtle->pya4 [0] =(float)0;  
actualTurtle->pza4 [0] =(float)0;
```

y

```
actualTurtle->pxa1 [actualTurtle->np] = m[0] * actualTurtle->pxa1 [0] + m[4] *  
actualTurtle->pya1 [0] + m[8] * actualTurtle->pza1 [0] + m[12];  
actualTurtle->pya1 [actualTurtle->np] = m[1] * actualTurtle->pxa1 [0] + m[5]  
* actualTurtle->pya1 [0] + m[9] * actualTurtle->pza1 [0] + m[13];  
actualTurtle->pza1 [actualTurtle->np] = m[2] * actualTurtle->pxa1 [0] + m[6]  
* actualTurtle->pya1 [0] + m[10] * actualTurtle->pza1 [0] + m[14];  
actualTurtle->pxa2 [actualTurtle->np] = m[0] * actualTurtle->pxa2 [0] + m[4]  
* actualTurtle->pya2 [0] + m[8] * actualTurtle->pza2 [0] + m[12];  
actualTurtle->pya2 [actualTurtle->np] = m[1] * actualTurtle->pxa2 [0] + m[5]  
* actualTurtle->pya2 [0] + m[9] * actualTurtle->pza2 [0] + m[13];  
actualTurtle->pza2 [actualTurtle->np] = m[2] * actualTurtle->pxa2 [0] + m[6]  
* actualTurtle->pya2 [0] + m[10] * actualTurtle->pza2 [0] + m[14];
```

```

    actualTurtle->pxa3 [actualTurtle->np] = m[0] * actualTurtle->pxa3 [0] + m[4]
* actualTurtle->pya3 [0] + m[8] * actualTurtle->pza3 [0] + m[12];
    actualTurtle->pya3 [actualTurtle->np] = m[1] * actualTurtle->pxa3 [0] + m[5]
* actualTurtle->pya3 [0] + m[9] * actualTurtle->pza3 [0] + m[13];
    actualTurtle->pza3 [actualTurtle->np] = m[2] * actualTurtle->pxa3 [0] + m[6]
* actualTurtle->pya3 [0] + m[10] * actualTurtle->pza3 [0] + m[14];
    actualTurtle->pxa4 [actualTurtle->np] = m[0] * actualTurtle->pxa4 [0] + m[4]
* actualTurtle->pya4 [0] + m[8] * actualTurtle->pza4 [0] + m[12];
    actualTurtle->pya4 [actualTurtle->np] = m[1] * actualTurtle->pxa4 [0] + m[5]
* actualTurtle->pya4 [0] + m[9] * actualTurtle->pza4 [0] + m[13];
    actualTurtle->pza4 [actualTurtle->np] = m[2] * actualTurtle->pxa4 [0] + m[6]
* actualTurtle->pya4 [0] + m[10] * actualTurtle->pza4 [0] + m[14];

```

Después añadimos una nueva opción en el switch de la función displayTrace:

```

    case(5):
        glBegin(GL_QUAD_STRIP);
        for (i = 0; i < turtlei->np-1; i++) {
            if(turtlei->PenD[i+1]){
                glBegin(GL_QUAD_STRIP);
                glColor3f(1.0f,0.0f,0.0f) ;
                glVertex3f (turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
                glVertex3f (turtlei->pxa1[i],turtlei->pya1[i],turtlei->pza1[i]);
                glVertex3f (turtlei->px[i+1],turtlei->py[i+1],turtlei->pz[i+1]);
                glVertex3f (turtlei->pxa1[i+1],turtlei->pya1[i+1],turtlei-
>pza1[i+1]);
                glColor3f(1.0,1.0,0.0) ;
                glVertex3f (turtlei->pxa1[i],turtlei->pya1[i],turtlei->pza1[i]);
                glVertex3f (turtlei->pxa2[i],turtlei->pya2[i],turtlei->pza2[i]);
                glVertex3f (turtlei->pxa1[i+1],turtlei->pya1[i+1],turtlei-
>pza1[i+1]);
                glVertex3f (turtlei->pxa2[i+1],turtlei->pya2[i+1],turtlei-
>pza2[i+1]);
                glColor3f(0.0,1.0,0.0) ;
                glVertex3f (turtlei->pxa2[i],turtlei->pya2[i],turtlei->pza2[i]);
                glVertex3f (turtlei->pxa3[i],turtlei->pya3[i],turtlei->pza3[i]);
                glVertex3f (turtlei->pxa2[i+1],turtlei->pya2[i+1],turtlei-
>pza2[i+1]);
                glVertex3f (turtlei->pxa3[i+1],turtlei->pya3[i+1],turtlei-
>pza3[i+1]);
                glColor3f(0.0,1.0,1.0) ;
                glVertex3f (turtlei->pxa3[i],turtlei->pya3[i],turtlei->pza3[i]);
                glVertex3f (turtlei->pxa4[i],turtlei->pya4[i],turtlei->pza4[i]);

```

```

>pza3[i+1]);          glVertex3f (turtlei->pxa3[i+1],turtlei->pya3[i+1],turtlei-
>pza4[i+1]);          glVertex3f (turtlei->pxa4[i+1],turtlei->pya4[i+1],turtlei-

                        glColor3f(0.0,0.0,1.0) ;
                        glVertex3f (turtlei->pxa4[i],turtlei->pya4[i],turtlei->pza4[i]);
                        glVertex3f (turtlei->px2[i],turtlei->py2[i],turtlei-
>pz2[i]);
                        glVertex3f (turtlei->pxa4[i+1],turtlei->pya4[i+1],turtlei-
>pza4[i+1]);          glVertex3f (turtlei->px2[i+1],turtlei->py2[i+1],turtlei->pz2[i+1]);
                        }
else{
                        glEnd();
                        glBegin(GL_QUAD_STRIP);
                        }
}
glEnd();
break;

```

Rastro Difuminado:

Primero definimos una variable booleana global que nos permitirá definir por teclado si el rastro debe ser difuminado o no.

```

BOOL diffuse=1;          //'1' not diffused trace
Int diffusePoints=20;

```

Después añadimos en la función parseCommand el if que nos cambie el estado de la variable diffuse al teclear diffuse en el teclado estando en modo comando. Añadimos el elseif tras los ifs de lectura de comandos con argumento, el argumento en este caso será el número de puntos que se representan:

```

else if ( !strcmp("diffuse",strToken0)) { // DIFUSE TRACE
    diffusePoints=( int) val;
    diffuse=0;
}

```

Tras el if de lectura de comandos sin argumento y para desactivar el trazo difuso:

```
        else if (strToken0 != NULL && !strcmp("diffuse",strToken0)) { // diffuse
TRACE
                diffuse=1;
                strToken0 = strtok(NULL, " ");
                display();
        }
```

Finalmente en la función DisplayTrace realizamos las siguientes operaciones.

Primero definimos una variable en la que se almacenan el numero de punto del trazo desde el que debemos comenzar a dibujar.

```
int points;
```

Después controlamos la variable diffuse para darle un valor a points, le daremos un valor fijo de 0 en caso de que la variable diffuse este a cero, y en caso de que este a 1 le daremos el valor np-20 de modo que se comenzará a dibujar desde el punto 20 anterior al actual.

```
if (diffuse)points =0;
else {
        if(turtlei->np > diffusePoints)points= turtlei->np-diffusePoints;
        else points =0;
}
```

Finalmente a la hora de dibujar cada tramo del trazo, en el bucle for que va dibujando cada tramo, sustituimos 0 por la variable points de modo que nos quede que:

```
for (i = points; i < np; i++) {
    ...
}
```

Comando Log:

Primero definimos el path del archivo log y una variable booleana global que determinara si se deben almacenar los comandos o no. Además también debemos definir un Handle al archivo.

```
BOOL Log=0;
FILE *Logstream;
const char LogFilename[50]="C:\\Logs.txt";
```

Después implementamos la lectura de los comandos en la función parseCommand tras el if de llamada a exit y home:

```

else if (strToken0 != NULL && !strcmp("log",strToken0)) { // STARTLOG
if( (Logstream = fopen(LogFilename, "a" )) == NULL )      printf( "The Log file
was not opened\n" );
    else printf( "The Log file was opened\n" );
    Log=1;
    strToken0 = strtok(NULL, " ");
    display();
} else if (strToken0 != NULL && !strcmp("logstop",strToken0)) { // STOPLOG
    fclose(Logstream);
    Log=0;
    strToken0 = strtok(NULL, " ");
    display();
}

```

Finalmente, y también en la función parseCommand, añadimos la escritura de los comandos en caso de que la variable Log este a 1, añadimos justo antes del if que compara strToken0 con repeat:

```

    if (strcmp("savetrace",strToken0))val = atof(strToken1);
        if(Log){
            if(strcmp("repeat",strToken0)){
                if(fputs((const)strToken0, Logstream ))printf( "The command
was not successfully written" );
                if(fputs(" ", Logstream ))printf( "The command was not
successfully written" );
                if(fputs((const)strToken1, Logstream ))printf( "The command
was not successfully written" );
                if(fputs(" ", Logstream ))printf( "The command was not
successfully written" );
            }
        }

```

Crear un modo amigable de cambiar los colores de la luz:

Primero declaramos tres variables globales que almacenaran los niveles de cada color en el momento actual, y a continuación tres vectores que almacenaran el color de cada una de las 3 luces:

```

float Rcolor;
float Gcolor;
float Bcolor;

float RScolor[3];
float GScolor[3];

```

```
float BScolor[3];
```

Estas variables se inicializan al inicio del main:

```
RScolor[0]=0.8;
RScolor[1]=0.8;
RScolor[2]=0.8;
GScolor[0]=0.8;
GScolor[1]=0.8;
GScolor[2]=0.8;
BScolor[0]=0.8;
BScolor[1]=0.8;
BScolor[2]=0.8;
```

A continuación definimos dos nuevas funciones, una primera TomarColoresLuz, que lee los valores de color de la luz actual desde los vectores de colores, y una DevolverColoresLuz que reinserta los nuevos valores de color a los vectores de colores:

```
/******
*****
DEVOLVERCOLORESLUZ FUNCTION:
*****
*****/
void DevolverColoresLuz(void){
    if(current_light>-1){
        RScolor[current_light]=Rcolor;
        GScolor[current_light]=Gcolor;
        BScolor[current_light]=Bcolor;
    }
}

/******
*****
TOMARCOLORESLUZ FUNCTION:
*****
*****/
void TomarColoresLuz(void){
    if(current_light>-1){
        Rcolor=RScolor[current_light];
        Gcolor=GScolor[current_light];
        Bcolor=BScolor[current_light];
    }
}
```

Después definimos una nueva función de la clase light que nos permita dar esos colores a luz. Primero definimos la función en el fichero light.h:

```
void Color_Luces( light *thisLight,float Rcolor, float Gcolor,float Bcolor);
```

Después la añadimos al fichero light.c:

```
void Color_Luces( light *thisLight,float Rcolor, float Gcolor,float Bcolor){

    /*thisLight->ambient[0]          = Rcolor;
    thisLight->ambient[1]            = Gcolor;
    thisLight->ambient[2]            = Bcolor;
    thisLight->ambient[3]            = 1.0f;*/

    thisLight->diffuse[0]            = Rcolor;
    thisLight->diffuse[1]            = Gcolor;
    thisLight->diffuse[2]            = Bcolor;
    thisLight->diffuse[3]            = 1.0f;

    /*thisLight->specular[0]          = Rcolor;
    thisLight->specular[1]            = Gcolor;
    thisLight->specular[2]            = Bcolor;
    thisLight->specular[3]            = 1.0f;*/
    thisLight->needsUpdate = TRUE;

}
```

*Las luces ambiente y specular están comentadas ya que en el ejercicio trabajamos con la luz difusa.

Finalmente en la función Keyboard, añadimos unas nuevas opciones al switch(key):

```
//////////cambiar colores de la luz//////////
case 'r':
    if(current_light>-1){
        TomarColoresLuz();
        if(Rcolor >0){
            Rcolor=Rcolor-0.1;
        }
        Color_Luces(LOCAL_MyLights[current_light],Rcolor,Gcolor,Bcolor);
        DevolverColoresLuz();
    }
    break;
case 'R':
    if(current_light>-1){
        TomarColoresLuz();
        if(Rcolor <1){
            Rcolor=Rcolor+0.1;
        }
    }
}
```

```

    }
    Color_Luces(LOCAL_MyLights[current_light],Rcolor,Gcolor,Bcolor);
    DevolverColoresLuz();
    }
    break;
    case 'g':
        if(current_light>-1){
            TomarColoresLuz();
            if(Gcolor >0){
                Gcolor=Gcolor-0.1;
            }
        }
        Color_Luces(LOCAL_MyLights[current_light],Rcolor,Gcolor,Bcolor);
        DevolverColoresLuz();
    }
    break;
    case 'G':
        if(current_light>-1){
            TomarColoresLuz();
            if(Gcolor <1){
                Gcolor=Gcolor+0.1;
            }
        }
        Color_Luces(LOCAL_MyLights[current_light],Rcolor,Gcolor,Bcolor);
        DevolverColoresLuz();
    }
    break;
    case 'b':
        if(current_light>-1){
            TomarColoresLuz();
            if(Bcolor >0){
                Bcolor=Bcolor-0.1;
            }
        }
        Color_Luces(LOCAL_MyLights[current_light],Rcolor,Gcolor,Bcolor);
        DevolverColoresLuz();
    }
    break;
    case 'B':
        if(current_light>-1){
            TomarColoresLuz();
            if(Bcolor <1){
                Bcolor=Bcolor+0.1;
            }
        }
        Color_Luces(LOCAL_MyLights[current_light],Rcolor,Gcolor,Bcolor);
        DevolverColoresLuz();
    }
    break;

```

Ejercicio realización de los modos de cámara Ortogonal y el modo Cónico:

Dado que las opciones cónico y paralelo ya existen en la clase cámara, solamente, deberemos asignar una de las teclas especiales para pasar de cámara cónica a cámara paralela. Además se deben determinar los parámetros de la cámara en modo paralelo, determinado la posición de los dos extremos opuestos de la visualización.

Así mismo, para poder volver al modo de visualización en perspectiva, deberemos asignar una nueva tecla para el paso a cónico. Dado que ya se definen los parámetros de la cámara en modo cónico al inicializarla, no será necesario asignarle estos valores cuando se pulse la tecla de paso a modo cónico.

```
case GLUT_KEY_F6:
    if (current_mode != 0) break;
    current_mode = 5;
    LOCAL_MyCamera->x1 = -10;
    LOCAL_MyCamera->y1 = -10;
    LOCAL_MyCamera->z1 = -3;
    LOCAL_MyCamera->x2 = 10;
    LOCAL_MyCamera->y2 = 10;
    LOCAL_MyCamera->z2 = 10;
    LOCAL_MyCamera->camProjection= CAM_PARALLEL;
    SetDependentParametersCamera( LOCAL_MyCamera );
    break;
```

```
case GLUT_KEY_F7:
    if (current_mode != 0) break;
    current_mode = 6;
    LOCAL_MyCamera->camProjection= CAM_CONIC;
    SetDependentParametersCamera( LOCAL_MyCamera );
    break;
```

Ejercicio de realización del modo PAN y el modo TRÍPODE de cámara:

Primero, en el archivo camara.h se deben definir dos nuevos modos de cámara así como una nueva función miembro de la clase

```
#define CAM_TRIPODE 3
#define CAM_PAN 4

void PanCamera( camera *thisCamera, float stepx, float stepy );
```

En el archivo camara.c se debe añadir la implementación de dicha función:

```
void PanCamera( camera *thisCamera, float valx, float valy )
{

    thisCamera->camViewX=thisCamera->camViewX + valx;
    thisCamera->camViewY =thisCamera->camViewY + valy;
    thisCamera->camAtX = thisCamera->camViewX + valx;
    thisCamera->camAtY = thisCamera->camViewY + valy;

    SetDependentParametersCamera( thisCamera );

}
```

Una vez tenemos la función definida, debemos incluir en el archivo tecnunlogo.c las siguientes funciones:

```
/******
*****
*****
*****/
TRIPODE FUNCTION:
*****/
void Tripode(int x,int y){
    float rot_x;
    rot_x = (float)( x - old_x) * DEGREE_TO_RAD / 5;
    YawCamera( LOCAL_MyCamera, rot_x );
    old_y = y;
    old_x = x;
    glutPostRedisplay();
}
/******
*****
*****
*****/
PAN FUNCTION:
*****/
void Pan(int x,int y){
    float mov_x;
```

```

float mov_y;
mov_x = (float)(old_x - x)/100 ;
mov_y = (float)(old_y - y)/100;
PanCamera( LOCAL_MyCamera, mov_x, mov_y );
old_y = y;
old_x = x;
glutPostRedisplay();
}

```

Se debe incluir las teclas de funciones especiales para activar los modos TRÍPODE y modo PAN:

```

case GLUT_KEY_F4:
    if (current_mode != 0) break;
    current_mode = 3;
    glutPassiveMotionFunc(MouseMotion);
    LOCAL_MyCamera->camMovimiento= CAM_TRIPODE;
    SetDependentParametersCamera( LOCAL_MyCamera );
    break;

case GLUT_KEY_F5:
    if (current_mode != 0) break;
    current_mode = 4;
    glutPassiveMotionFunc(MouseMotion);
    LOCAL_MyCamera->camMovimiento= CAM_PAN;
    SetDependentParametersCamera( LOCAL_MyCamera );
    break;

```

Finalmente en la función mouse, dentro del switch de control de camMovimiento se deben añadir los casos de los modos PAN y TRÍPODE.

```

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

```

Ejercicio de cambio del color de fondo mediante la función Background:

Se debe declarar la variable global:

```
float BackgroundColor[3];
```

Al inicio de la función main se deben inicializar los valores del vector por

```
BackgroundColor[0]=0.0;  
BackgroundColor[1]=0.0;  
BackgroundColor[2]=0.0;
```

Que definen el color negro. En la función display se debe sustituir el comando glColor por:

```
glClearColor(BackgroundColor[0], BackgroundColor[1], BackgroundColor[2], 0.0);
```

En la función parseCommand se deben declarar las siguientes variables:

```
char *strToken2;  
char *strToken3;  
double val2,val3;
```

En el if donde se reconocen los comandos con argumento, se debe añadir:

```
else if(!strcmp("background",strToken0))    {  
    if(((strToken2 = strtok(NULL, " ")) != NULL) &&((strToken3 = strtok(NULL, " "))  
    != NULL)){  
        val2 = atof(strToken2);  
        val3 = atof(strToken3);  
        BackgroundColor[0]=val;  
        BackgroundColor[1]=val2;  
        BackgroundColor[2]=val3;  
    }  
}
```

Ejercicio de almacenamiento del trazo de la tortuga en un archivo wavefront .obj mediante el comando savetrace Nombre, que almacenara el trazo como NombreTrace.obj:

Se debe incluir la siguiente función

```

/*****
*****
SAVE TRACE FUNCTION:
*****
*****/

void SaveTrace(char *Name){
    char Trace[100]="C:\\";
    FILE *stream;
    char Numx[8],Numy[8],Numz[8];
    int i=0;
    strcat(Trace,(const)Name);
    strcat(Trace, "Trace.obj");
    stream= fopen(Trace,"w");
    if(!stream)printf("Error opening the Trace file");
    if(stream){
        printf("The Trace file was opened");
        for (i=0; i<actualTurtle->np;i++)fprintf(stream,"%s%f%s%f%s%f\n", "v
",actualTurtle->px[i], " ",actualTurtle->py[i], " ",actualTurtle->pz[i]);
        for (i=0; i<actualTurtle->np;i++)fprintf(stream,"%s%f%s%f%s%f\n", "v
",actualTurtle->px2[i], " ",actualTurtle->py2[i], " ",actualTurtle->pz2[i]);

        for (i=1;i<actualTurtle->np;i++){
            fprintf(stream,"%s%d%s\n", "g ",i,"Tramo");
            fprintf(stream,"%s%d%s%d%s%d%s%d\n", "f ",i, " ",actualTurtle-
>np+i, " ",actualTurtle->np+i+1, " ",i+1);

        }
        printf("Data saved");
        fclose(stream);
    }
}

```

Tras el if de identificación de los comandos con argumento, se debe incluir:

```

else if (!strcmp("savetrace",strToken0)){
    SaveTrace(strToken1);
}

```

Se debe tener en cuenta que los datos contenidos en strToken1 son texto para este caso, por tanto, se debe sustituir previamente en la función parseCommand

```

val = atof(strToken1);

```

por

```
if (strcmp("savetrace",strToken0))val = atof(strToken1);
```

insertándolo justo tras la función while.

Ejercicio de Rastro con objetos:

Dibujar un rastro que esté formado por un objeto, el mismo que se está desplazando u otro distinto. Se implementa con el comando SETTRACEOBJECT fileName.

Inicialmente se debe declarar una variable global que determina si el rastro de objetos está activo o no.

```
BOOL TraceObject=0;
```

También se debe declarar una estructura que nos permita almacenar las matrices mModel en cada una de las posiciones por las que pasa la tortuga. Además se deberá declarar un vector de estas estructuras en el interior de la estructura de la tortuga así como un objeto que contenga el objeto de los trazos:

```
typedef struct _turtletraces
{
    GLdouble mModel[16];
}turtletraces;
```

En la estructura turtle:

```
turtletraces *turtletracesVector[10000];
glmObject *Traceobject;
```

A continuación se incluye la lectura del comando SETTRACEOBJECT. Para ello se debe incluir el siguiente comando tras los else if de lectura de comandos con un solo argumento:

```
else if (!strcmp("settraceobject",strToken0)){ ///SETTRACEOBJECT

    actualTurtle->Traceobject= createGlmObject(strToken1);
    TraceObject=!TraceObject;

}
```

Se deberá incluir en la función addPoitnToTrace , primero en el interior del if que se ejecuta únicamente para el primer punto:

```
    actualTurtle->turtletracesVector[actualTurtle->np] = (turtletraces*)
    malloc(sizeof(turtletraces));

    glGetDoublev (GL_MODELVIEW_MATRIX,actualTurtle-
->turtletracesVector[actualTurtle->np]->mModel);
```

Después, en la misma función se incluirá el comando mediante el cual se copiará la matriz mModel en cada una de las posiciones:

```
    actualTurtle->turtletracesVector[actualTurtle->np] = (turtletraces*)
    malloc(sizeof(turtletraces));
    glGetDoublev (GL_MODELVIEW_MATRIX,actualTurtle-
->turtletracesVector[actualTurtle->np]->mModel);
```

Por último se debe añadir el siguiente if en el interior de la función displayTrace justo después de que se vuelvan a activar las luces y las caras de los objetos:

```
    if (TraceObject){
        for(i = points; i < turtlei->np-1; i++){
            glPushMatrix();
            glLoadIdentity();
            glMultMatrixd(turtlei->turtletracesVector[i]->mModel);
            glCallList(turtlei->Traceobject->model_list);
            glPopMatrix();
        }
    }
```

sustituyendo points por 0 si no se ha implementado la opción de trazo difuso

Ejercicio de desplazamiento de la Tortuga mediante el uso del ratón:

En este ejercicio se realizan las modificaciones necesarias para que se pueda desplazar la tortuga mediante el uso del ratón. Mediante el movimiento horizontal del ratón la tortuga realizara giros sobre su plano. Mediante el movimiento vertical, la tortuga realizará giros sobre su plano vertical.

Si se mantiene el botón Izquierdo pulsado, y mediante movimiento vertical, se logra desplazamiento de la tortuga hacia delante y hacia atrás.

Primero se debe declarar una variable global que determina si nos encontramos en el modo de manejo mediante ratón o no.

```
    BOOL Avance=0;
```

A continuación se debe incluir en la función specialkeys incluir el nuevo caso:

```
case GLUT_KEY_F11:
```

```
current_mode=8;
glutPassiveMotionFunc(MouseRot);
Avance=1;
break;
```

Además en el caso F1 se debe incluir:

```
Avance=0;
```

Después se deben declarar las siguientes funciones:

```
/******
*****
MOUSE_AVANCE FUNCTION:
*****/
void MouseAvance(int x, int y){
    float rot_x, rot_y;
    int i,Pasos;
    char fd[7]="fd 0.5";
    char bk[7]="bk 0.5";
    rot_y = (long)(old_y - y);

    Pasos=(int) abs(rot_y)/2;
    if(rot_y<0){
        for(i=0; i<Pasos; i++){
            parseCommand(bk);
            //MovFCamPlain(LOCAL_MyCamera);
        }
    }
    else{
        for(i=0; i<Pasos; i++){
            parseCommand(fd);
            //MovBCamPlain(LOCAL_MyCamera);
        }
    }
    old_y = y;
    old_x = x;
    glutPostRedisplay();
}
/******
*****
MOUSE_AVANCE_PLAIN FUNCTION:
*****/
void MouseRot(int x, int y){
```

```

float rot_x, rot_y;
int i,Pasos;

char rt[7]="rt 5";
char lt[7]="lt 5";
char up[7]="up 5";
char dn[7]="dn 5";

rot_x = (long)(old_x - x );
rot_y = (long)(old_y - y);
if(abs(rot_x)>abs(rot_y)){
    Pasos=(int) abs(rot_x)/5;
    if(rot_x<0){
        for(i=0; i<Pasos; i++){
            parseCommand(rt);
            //RotRCamPlain(LOCAL_MyCamera);
        }
    }
    else {
        for(i=0; i<Pasos; i++){
            parseCommand(lt);
            //RotLCamPlain(LOCAL_MyCamera);
        }
    }
}else{
    Pasos=(int) abs(rot_y)/5;
    if(rot_y<0){
        for(i=0; i<Pasos; i++){
            parseCommand(dn);
        }
    }
    else {
        for(i=0; i<Pasos; i++){
            parseCommand(up);
        }
    }
}

old_y = y;
old_x = x;
glutPostRedisplay();
}

```

Por ultimo en la función mouse y entre el else del if que controla el valor de currentlight y el switch de control del modo de movimiento de cámara.

```

if(Avance){
    if (state == GLUT_DOWN) glutMotionFunc(MouseAvance);
    if (state == GLUT_UP)glutMotionFunc(NULL);
}
else{
    ...

```

Tras todo el switch de control de cámara se deberán cerrar las llaves.

Ejercicio de modo racing:

Al iniciar el modo racing la tortuga avanzará automáticamente hacia delante. Mientras avanza, se podrá controlar la tortuga mediante el ratón, o el teclado. Mediante el movimiento vertical del ratón se inclinara hacia arriba o hacia abajo la tortuga, mientras que con el movimiento horizontal, los giros serán a izquierda y a derecha.

Estos mismos movimientos se podrán realizar mediante el teclado:

‘o’ giro hacia arriba
‘l’ giro hacia abajo
‘k’ giro a la izquierda
‘ñ’ giro a la derecha.

Primero se debe incluir en el switch de lectura de tecla en la función SpecialKey la tecla que activará el modo racing:

```

case GLUT_KEY_F12:
    current_mode=9;
    glutKeyboardFunc(Giro_Vel);
    glutPassiveMotionFunc(MouseRot);
    glutIdleFunc(Racing);
    break;

```

Dado que se modifica las funciones de control de teclas y de Idle, cuando se llama a F1 que detiene los modos de operación, se deben restaurar las funciones tal y como estaban. Por lo tanto tras el case GLUT_KEY_F1 y antes de el break correspondiente se debe incluir:

```

glutIdleFunc(NULL);
glutKeyboardFunc(keyboard);

```

Además se deberán incluir las dos funciones nuevas a las que se llaman desde la función SpecialKeys:

```

void Giro_Vel(char key, int x, int y){
    char rt[7]="rt 5";

```

```

char lt[7]="lt 5";
char up[7]="up 5";
char dn[7]="dn 5";
switch(key){
    case 'o':
        parseCommand(up);
        break;
    case 'l':
        parseCommand(dn);
        break;
    case 'k':
        parseCommand(lt);
        break;
    case 'ñ':
        parseCommand(rt);
        break;
}
}

void Racing(void){
    char avance[7]="fd 0.05";
    parseCommand(avance);
}

```

Si se quisiera aumentar la velocidad de avance o de giro, simplemente habría que modificar los valores de avance y de giro en estas dos funciones.

Si no se ha incluido, también se debe incluir la función mouseRot que se encuentra en la sección anterior.

Ejercicio de cámara sobre objeto en todo momento:

Esta función que se llama mediante comando, permite mantener la cámara siempre sobre el objeto.

Primero se deben declarar las siguientes variables globales

```

BOOL Cameraonthefront=0;

```

Que servirá para determinar si nos encontramos en el modo de cámara en persecución. Además, dentro de la estructura de turtle, se deberán incluir las siguientes variables, que determinarán la posición de la cámara y la de la verticalidad:

```

float followcamx,followcamy,followcamz;

```

```
float Verticalx,Verticaly,Verticalz;
```

También habrá que incluir la lectura del comando camerastatus en la función parseCommand, para ello se deberá incluir el siguiente comando dentro del if en el que se comparan los comandos sin argumento:

```
else if (strToken0 != NULL && !strcmp("camerastatus",strToken0))
    {
    //showturtle
    if(Cameraonthefront==1)SetCamera( LOCAL_MyCamera,
        0.0f, 1.0 , -3.0f,
        0.0f, 0.0f , 0.0f,
        0.0f, 1.0f , 0.0f);

        Cameraonthefront=!Cameraonthefront;
        strToken0 = strtok(NULL, " ");
        display();

    }
}
```

También en la función parseCommand, en el else if del comando 'st' se deben inicializar las variables incluidas en la estructura turtle, para las nuevas tortugas:

```
turtleVector[ival]->followcamx=0.0f;
turtleVector[ival]->followcamy=1.0f;
turtleVector[ival]->followcamz=-3.0f;
turtleVector[ival]->Verticalx=0.0f;
turtleVector[ival]->Verticaly=1.0f;
turtleVector[ival]->Verticalz=0.0f;
```

En el main, cuando se inicializa el primer elemento de el vector de tortugas, también se deberán inicializar estas variables, por ello, tras inicializar el valor de np, se debe incluir:

```
turtleVector[0]->Verticalx=0.0f;
turtleVector[0]->Verticaly=1.0f;
turtleVector[0]->Verticalz=0.0f;
turtleVector[0]->followcamx=0.0f;
turtleVector[0]->followcamy=1.0f;
turtleVector[0]->followcamz=0.0f;
```

En la función parseCommand en el comando home, también se reiniciara el valor de las posiciones de cámara al mismo tiempo que la posición de el objeto:

```
actualTurtle->followcamx=0.0f;
actualTurtle->followcamy=1.0f;
actualTurtle->followcamz=-3.0f;
actualTurtle->Verticalx=0.0f;
```

```

actualTurtle->Verticaly=1.0f;
actualTurtle->Verticalz=0.0f;
glGetDoublev (GL_MODELVIEW_MATRIX, actualTurtle->mModel);
actualTurtle->np = 0;

```

Finalmente en la función addPointToTrace, tras el if que se ejecuta en el primero de los puntos, se debe incluir:

```

actualTurtle->followcamx = m[0] * 0.0 + m[4] * 1.0 + m[8] * -3.0 + m[12];
actualTurtle->followcamy = m[1] * 0.0 + m[5] * 1.0 + m[9] * -3.0 + m[13];
actualTurtle->followcamz = m[2] * 0.0 + m[6] * 1.0 + m[10] * -3.0 + m[14];
actualTurtle->Verticalx = m[0] * 0.0 + m[4] * 1.0 + m[8] * 0.0 + m[12];
actualTurtle->Verticaly = m[1] * 0.0 + m[5] * 1.0 + m[9] * 0.0 + m[13];
actualTurtle->Verticalz = m[2] * 0.0 + m[6] * 1.0 + m[10] * 0.0 + m[14];

```

```

if(Cameraonthefront)SetCamera( LOCAL_MyCamera,
    actualTurtle->followcamx, actualTurtle->followcamy, actualTurtle-
>followcamz,
    actualTurtle->px[actualTurtle->np], actualTurtle->py[actualTurtle-
>np],actualTurtle->pz[actualTurtle->np],
    actualTurtle->Verticalx, actualTurtle->Verticaly, actualTurtle-
>Verticalz);

```