

TRABAJOGRAFICOS POR COMPUTADOR
Samuel Marín Calvo 900719

1. EJES	2
2. Función SETCIRCLE.....	2
3. Función LOAD	3
4. Función HI	4
5. SETTRACE i.....	5
6. Rastro con colores del arco iris.....	11
7. Rastro con objetos SETTRACEOBJECT filename	13
8. COMANDO LOG	15
9. RASTRO DIFUMINADO	16
10. MODO COCHE Y CAMARA SEGUIDORA.....	18

1. EJES

Crear variable global ejes tipo boolean.

Para los ejes crear una función Ejes() con el siguiente código

```
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();
```

para que aparezcan los ejes cargar esta función antes del pushmatrix() en la función display

y para los móviles entre esta y la popmatrix().

2. Función SETCIRCLE

Creo una variable global double grados e inicializarla a 360 o lo que se quiera, añadido a parsecommand()

```
else if (!strcmp("setcircle",strToken0)) { // Scalez
    grados=val;
}
```

y la fórmula de giro es ahora: $val*360/grados$ que convierte a grados de nuevo en las funciones de rotar.

3. Función LOAD

El siguiente código justo después del if para la función repeat:

```
else if (!strcmp("load",strToken0)) {
FILE* file;
char buf[128];
int i = 0;
/* open the file */
file = fopen(strToken1, "r");
if (!file) {
    fprintf(stderr, "open failed: can't open file \"%s\".\n",strToken1);
}
else{
    while(fgets(buf, 256, file) != NULL) {
        printf("Line %i: %s", ++i, buf);
    }
    parseCommand(buf);
}
}
```

4. Función HI

Meter dentro de la estructura de tortuga una variable booleana
boolean hi;

Dentro de la función display() debería quedar así dentro del for donde se cargan las tortugas justo después de la llamada a la función displaytrace()

```
if (turtleVector[i]->hi==FALSE){
    glPushMatrix();
    glMultMatrixd(turtleVector[i]->mModel);
    DrawAxis();
//    Ejes_World(); // ejes locales
//    tortuga_simple();
//    tortuga_simple_3D();
    glColor3f(1.0,1.0,0.0) ;
    if (turtleVector[i]->object != NULL)
        glCallList(turtleVector[i]->object->model_list);
    else
        drawTurtle();
//    glutWireTorus(0.25, 0.75, 28, 28);
    glPopMatrix();
}
```

En la función parsecommand() donde se inicializan los valores de una nueva tortuga hay que poner

```
turtleVector[ival]->hi=FALSE;
```

También hay que poner esto al final del main() cuando se inicializa la primera tortuga
turtleVector[0]->hi = FALSE;

y por último esto al final del parsecommand()

```
else if (strToken0 != NULL && !strcmp("hi",strToken0)) {
    if(actualTurtle->hi==FALSE){
        actualTurtle->hi=TRUE;
    }else{
        actualTurtle->hi=FALSE;
    }
}
```

5. SETTRACE i

Primero creamos una variable en la clase turtle int selecciontrazo; para saber que trazo se va a hacer en cada tortuga hay que inicializarla a cero cuando se cree cualquier objeto de clase tortuga.

Se añade a parsecommand() esto para poder seleccionar el trazo

```
else if (!strcmp("settrace",strToken0)) {
    printf("SETTRACE");
    actualTurtle->selecciontrazo=val;

}
```

En displaytrace() se mete un switch para elegir el trazo que se quiera
int lightingFlag;

```
lightingFlag = glIsEnabled( GL_LIGHTING );
```

```
if( lightingFlag ) glDisable( GL_LIGHTING );
switch (turtlei->selecciontrazo) {
case 0:

    glBegin(GL_QUAD_STRIP);
    glColor3f(1.0,0.0,1.0) ;
    for (i = 0; i < turtlei->np; i++) {

        glVertex3f (turtlei->px2[i],turtlei->py2[i],turtlei->pz2[i]);
        glVertex3f (turtlei->px3[i],turtlei->py3[i],turtlei->pz3[i]);
    }
    glEnd();

    glBegin(GL_QUAD_STRIP);
    for (i = 0; i < turtlei->np; i++) {

        glVertex3f (turtlei->px3[i],turtlei->py3[i],turtlei->pz3[i]);
        glVertex3f (turtlei->px2[i],turtlei->py2[i],turtlei->pz2[i]);
    }
    glEnd();

case 1:

    trazoarcoiris(turtlei);
```

```

        break;
    case 2:
        glBegin(GL_QUAD_STRIP);
        glColor3f(0.0,1.0,1.0) ;

        for (i = 0; i < turtlei->np; i++) {

            glVertex3f (turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
            glVertex3f (turtlei->px6[i],turtlei->py6[i],turtlei->pz6[i]);
        }
        glEnd();
        glBegin(GL_QUAD_STRIP);
        for (i = 0; i < turtlei->np; i++) {
            glVertex3f (turtlei->px6[i],turtlei->py6[i],turtlei->pz6[i]);

            glVertex3f (turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
        }
        glEnd();

        break;
    case 3:
        nubes(turtlei);
        break;
    case 4:
        triangulo(turtlei);
        break;

    case 5:
        trazoobjeto(turtlei);
        break;
if( lightingFlag ) glEnable( GL_LIGHTING );

}

```

En mi caso el 0 es el trazo normal, el 1 es el trazo con colores de arcoiris, el 2 es el trazo de Tron, el 3 es un trazo de esferas, el 4 un prisma triangular y el 5 un trazo hecho con objetos wavefront.

Para hacer los distintos trazos he metido en la clase tortuga

```

float px [10000];
float py [10000];
float pz [10000];
float px2 [10000];
float py2 [10000];
float pz2 [10000];
float py3 [10000];
float pz3 [10000];
float px3 [10000];
float py4 [10000];
float pz4 [10000];
float px4 [10000];

```

```
float py5 [10000];
float pz5 [10000];
float px5 [10000];
float py6 [10000];
float pz6 [10000];
float px6 [10000];
```

```
y en addtrace()
```

```
if (actualTurtle->np == 0) { // add the first point
    //for(z=0;z<16;z++){
    //actualTurtle->Fulin[0][z]=actualTurtle->mModel[z];
    //}
    actualTurtle->px [0] = 0;
    actualTurtle->py [0] = 0;
    actualTurtle->pz [0] = 0;

// the second point
    actualTurtle->px2 [0] = 0.05f;
    actualTurtle->py2 [0] = 0;
    actualTurtle->pz2 [0] = 0;
    actualTurtle->px3 [0] = -0.05f;
    actualTurtle->py3 [0] = 0;
    actualTurtle->pz3 [0] = 0;
    actualTurtle->px4 [0] = 0.025f;
    actualTurtle->py4 [0] = 0;
    actualTurtle->pz4 [0] = 0;
    actualTurtle->px5 [0] = -0.025f;
    actualTurtle->py5 [0] = 0;
    actualTurtle->pz5 [0] = 0;
    actualTurtle->px6 [0] = 0;
    actualTurtle->py6 [0] = 0.05f;
    actualTurtle->pz6 [0] = 0;

    actualTurtle->np++;

}
actualTurtle->np;
actualTurtle->px [actualTurtle->np] = m[0] * actualTurtle->px [0] + m[4] *
actualTurtle->py [0] + m[8] * actualTurtle->pz [0] + m[12];
actualTurtle->py [actualTurtle->np] = m[1] * actualTurtle->px [0] + m[5] *
actualTurtle->py [0] + m[9] * actualTurtle->pz [0] + m[13];
actualTurtle->pz [actualTurtle->np] = m[2] * actualTurtle->px [0] + m[6] *
actualTurtle->py [0] + m[10] * actualTurtle->pz [0] + m[14];
printf ("Point %i: %f \t%f \t%f \n",
        actualTurtle->np, actualTurtle->px[actualTurtle->np],actualTurtle-
>py[actualTurtle->np],actualTurtle->pz[actualTurtle->np]);
```

```
    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];
```

```
    actualTurtle->px4 [actualTurtle->np] = m[0] * actualTurtle->px4 [0] + m[4] *
actualTurtle->py4 [0] + m[8] * actualTurtle->pz4 [0] + m[12];
```

```
    actualTurtle->py4 [actualTurtle->np] = m[1] * actualTurtle->px4 [0] + m[5] *
actualTurtle->py4 [0] + m[9] * actualTurtle->pz4 [0] + m[13];
```

```
    actualTurtle->pz4 [actualTurtle->np] = m[2] * actualTurtle->px4 [0] + m[6] *
actualTurtle->py4 [0] + m[10] * actualTurtle->pz4 [0] + m[14];
```

```
    actualTurtle->px5 [actualTurtle->np] = m[0] * actualTurtle->px5 [0] + m[4] *
actualTurtle->py5 [0] + m[8] * actualTurtle->pz5 [0] + m[12];
```

```
    actualTurtle->py5 [actualTurtle->np] = m[1] * actualTurtle->px5 [0] + m[5] *
actualTurtle->py5 [0] + m[9] * actualTurtle->pz5 [0] + m[13];
```

```
    actualTurtle->pz5 [actualTurtle->np] = m[2] * actualTurtle->px5 [0] + m[6] *
actualTurtle->py5 [0] + m[10] * actualTurtle->pz5 [0] + m[14];
```

```
    actualTurtle->px6 [actualTurtle->np] = m[0] * actualTurtle->px6 [0] + m[4] *
actualTurtle->py6 [0] + m[8] * actualTurtle->pz6 [0] + m[12];
```

```
    actualTurtle->py6 [actualTurtle->np] = m[1] * actualTurtle->px6 [0] + m[5] *
actualTurtle->py6 [0] + m[9] * actualTurtle->pz6 [0] + m[13];
```

```
    actualTurtle->pz6 [actualTurtle->np] = m[2] * actualTurtle->px6 [0] + m[6] *
actualTurtle->py6 [0] + m[10] * actualTurtle->pz6 [0] + m[14];
```

```
    printf ("Point %i: %f \t%f \t%f \n",
            actualTurtle->np, actualTurtle->px[actualTurtle->np],actualTurtle-
>py[actualTurtle->np],actualTurtle->pz[actualTurtle->np]);
    actualTurtle->np++;
```

y el código para el trazo de nubes es el siguiente

```
void nubes(turtle *turtlei){
    int i;
    for (i = 0; i < turtlei->np; i++) {
        glPushMatrix();
        glColor3f(0.1f,0.0,0.0);
        glTranslatef(turtlei->px [i],turtlei->py[i],turtlei->pz[i]);
```

```

        glBegin(GL_POLYGON);

        glutSolidSphere(0.05,50,10);
        glEnd();
        glColor3f(0.0,0.0,2);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(turtle->px2[i],turtle->py2[i],turtle->pz2[i]);
        glBegin(GL_POLYGON);
        glutSolidSphere(0.05,50,10);
        glEnd();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(turtle->px3[i],turtle->py3[i],turtle->pz3[i]);
        glBegin(GL_POLYGON);
        glutSolidSphere(0.06,100,100);
        glEnd();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(turtle->px4[i],turtle->py4[i],turtle->pz4[i]);
        glBegin(GL_POLYGON);
        glutSolidSphere(0.07,50,10);
        glEnd();
        glPopMatrix();
    }
}

```

el código para el trazo de prisma triangular es el siguiente

```

void triangulo(turtle *turtle){
    int i;
    glBegin(GL_QUAD_STRIP);
    for (i = 0; i < turtle->np; i++) {
        glColor3f(1.0,0.0,0.0);
        glVertex3f (turtle->px6[i],turtle->py6[i],turtle->pz6[i]);
        glColor3f(1.0,1.0,0.0);
        glVertex3f (turtle->px2[i],turtle->py2[i],turtle->pz2[i]);
    }
    glEnd();
    glBegin(GL_QUAD_STRIP);
    for (i = 0; i < turtle->np; i++) {

        glColor3f(1.0,1.0,0.0);
        glVertex3f (turtle->px2[i],turtle->py2[i],turtle->pz2[i]);
        glColor3f(1.0,0.0,0.0);
        glVertex3f (turtle->px6[i],turtle->py6[i],turtle->pz6[i]);
    }
    glEnd();
    glBegin(GL_QUAD_STRIP);

```

```

for (i = 0; i < turtlei->np; i++) {
    glColor3f(1.0,1.0,0.0);
    glVertex3f (turtlei->px2[i],turtlei->py2[i],turtlei->pz2[i]);
    glColor3f(0.0,1.0,0.0);
    glVertex3f (turtlei->px3[i],turtlei->py3[i],turtlei->pz3[i]);
}
glEnd();
glBegin(GL_QUAD_STRIP);
for (i = 0; i < turtlei->np; i++) {

    glColor3f(0.0,1.0,0.0);
    glVertex3f (turtlei->px3[i],turtlei->py3[i],turtlei->pz3[i]);
    glColor3f(1.0,1.0,0.0);
    glVertex3f (turtlei->px2[i],turtlei->py2[i],turtlei->pz2[i]);
}
glEnd();
glBegin(GL_QUAD_STRIP);
for (i = 0; i < turtlei-> np; i++) {
    glColor3f(0.0,1.0,0.0);
    glVertex3f (turtlei->px3[i],turtlei->py3[i],turtlei->pz3[i]);
    glColor3f(0.0,1.0,1.0);
    glVertex3f (turtlei->px6[i],turtlei->py6[i],turtlei->pz6[i]);
}
glEnd();
glBegin(GL_QUAD_STRIP);
for (i = 0; i < turtlei-> np; i++) {
    glColor3f(0.0,1.0,1.0);
    glVertex3f (turtlei->px6[i],turtlei->py6[i],turtlei->pz6[i]);
    glColor3f(0.0,1.0,0.0);
    glVertex3f (turtlei->px3[i],turtle i->py3[i],turtlei->pz3[i]);
}
glEnd();

```

```

}

```

6. Rastro con colores del arco iris

En el caso anterior ya se ve que esta incluido pero el código para el trazo de arcoiris es el siguiente

```
void trazoarcoiris(turtle *turtlei){
    int i;
    glBegin(GL_QUAD_STRIP);
    for (i = 0; i < turtlei->np; i++) {
        glColor3f(1.0,0.0,0.0);
        glVertex3f (turtlei->px2[i],turtlei->py2[i],turtlei->pz2[i]);
        glColor3f(1.0,1.0,0.0);
        glVertex3f (turtlei->px4[i],turtlei->py4[i],turtlei->pz4[i]);
    }
    glEnd();
    glBegin(GL_QUAD_STRIP);
    for (i = 0; i < turtlei->np; i++) {

        glColor3f(1.0,1.0,0.0);
        glVertex3f (turtlei->px4[i],turtlei->py4[i],turtlei->pz4[i]);
        glColor3f(1.0,0.0,0.0);
        glVertex3f (turtlei->px2[i],turtlei->py2[i],turtlei->pz2[i]);
    }
    glEnd();

    glBegin(GL_QUAD_STRIP);
    for (i = 0; i < turtlei->np; i++) {
        glColor3f(1.0,1.0,0.0);
        glVertex3f (turtlei->px4[i],turtlei->py4[i],turtlei->pz4[i]);
        glColor3f(0.0,1.0,0.0);
        glVertex3f (turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
    }
    glEnd();
    glBegin(GL_QUAD_STRIP);
    for (i = 0; i < turtlei->np; i++) {
        glColor3f(0.0,1.0,0.0);
        glVertex3f (turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
        glColor3f(1.0,1.0,0.0);
        glVertex3f (turtlei->px4[i],turtlei->py4[i],turtlei->pz4[i]);
    }

    glEnd();
    glBegin(GL_QUAD_STRIP);
```

```

for (i = 0; i <turtlei-> np; i++) {
    glColor3f(0.0,1.0,0.0);
    glVertex3f (turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
    glColor3f(0.0,1.0,1.0);
    glVertex3f (turtlei->px5[i],turtlei->py5[i],turtlei->pz5[i]);
}
glEnd();
glBegin(GL_QUAD_STRIP);
for (i = 0; i <turtlei-> np; i++) {

    glColor3f(0.0,1.0,1.0);
    glVertex3f (turtlei->px5[i],turtlei->py5[i],turtlei->pz5[i]);
    glColor3f(0.0,1.0,0.0);
    glVertex3f (turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
}
glEnd();
glBegin(GL_QUAD_STRIP);
for (i = 0; i < turtlei->np; i++) {
    glColor3f(0.0,1.0,1.0);
    glVertex3f (turtlei->px5[i],turtlei->py5[i],turtlei->pz5[i]);
    glColor3f(0.0,0.0,1.0);
    glVertex3f (turtlei->px3[i],turtlei->py3[i],turtlei->pz3[i]);
}
glEnd();
glBegin(GL_QUAD_STRIP);
for (i = 0; i < turtlei->np; i++) {
    glColor3f(0.0,0.0,1.0);
    glVertex3f (turtlei->px3[i],turtlei->py3[i],turtlei->pz3[i]);
    glColor3f(0.0,1.0,1.0);
    glVertex3f (turtlei->px5[i],turtlei->py5[i],turtlei->pz5[i]);
}
glEnd();
}
}

```

7. Rastro con objetos SETRACEOBJECT filename

Primero dentro de la clase de tortuga hay que crear una variable `int nMatrizmModel`; que lleva la cuenta de matrices que he guardado también hay que crear `double MatrizmModel[10000][16]`; y por último una variable tipo `glmObject *trazo`; que almacena el objeto a dibujar.

Cada vez que se cree un objeto de la clase tortuga hay que inicializar las variables `turtleVector[ival]->nMatrizmModel = 0`;
`turtleVector[ival]->trazo=NULL`;
en `addpointToTrace()` hay que poner

```
for(z=0;z<16;z++){
    actualTurtle->MatrizmModel[actualTurtle-
>nMatrizmModel][z]=actualTurtle->mModel[z];
}
```

```
actualTurtle->nMatrizmModel++;
```

para guardar las matrices de transformación y por último el código que dibujaría el rastro es este y se inicializa en el código de `displaytrace()` que ya he puesto arriba

```
void trazoobjeto(turtle *turtlei){
    if (turtlei->trazo!=NULL){
        glPushMatrix();

        glColor3f(1.0,1.0,0.0);
        int i;

        for( i=0; i<turtlei->nfulin; i++){
            glColor3f(1.0,1.0,0.0) ;
            if (turtlei->hi==FALSE){
                glPushMatrix();
                GLdouble temp[16];
                for(int z=0;z<16;z++){
```


8. COMANDO LOG

Creo dos variables globales
boolean logear=FALSE;
char logfile[25];

En la funcion parsecommand() creo la variable
FILE *logfichero;
Y justo al principio pongo el siguiente codigo
if (logear==TRUE){

```
    logfichero = fopen( logfile, "a" );
```

```
    fprintf( logfichero, strCommandParse );  
    fprintf(logfichero, "\n");
```

```
    }
```

Y por ultimo donde están todos el resto de comandos en parsecommand()

```
else if (!strcmp("log",strToken0)) {  
    logear =TRUE;  
    strcpy(logfile,strToken1);  
}  
else if (strToken0 != NULL && !strcmp("logstop",strToken0)) {  
    logear=FALSE;  
}
```

9. RASTRO DIFUMINADO

Poniendo `settraceblurr` i hago que a la tortuga actual (y cualquier otra que tenga una variable boolean en `TRUE`) se le dibujen la ultimas i posiciones
Primero hay que crear dos variable en turtle

```
boolean traceblurr;  
int ultimaspos;
```

Hay que poner esto en el `parsecommand`

```
else if (!strcmp("settraceblurr",strToken0)) {  
  
    ival=val;  
    actualTurtle->traceblurr=TRUE;  
    actualTurtle->ultimaspos=ival;  
  
}
```

por ultimo delante de todos los tipos de trazo hay que cambiar igual que el que voy a cambiar aquí

```
case 0:  
    if (turtlei->traceblurr==TRUE && turtlei->np>turtlei-  
>ultimaspos){  
        z=turtlei->np-turtlei->ultimaspos;  
    }  
    else{  
        z=0;  
    }  
    glBegin(GL_QUAD_STRIP);  
    glColor3f(1.0,0.0,1.0) ;  
    // glNormal3f(0.0,1.0,0.0) ;  
  
    for (i=z; i < turtlei->np; i++) {  
  
        glVertex3f (turtlei->px2[i],turtlei->py2[i],turtlei-  
>pz2[i]);  
        glVertex3f (turtlei->px3[i],turtlei->py3[i],turtlei-  
>pz3[i]);  
    }  
    glEnd();  
    if (turtlei->traceblurr==TRUE && turtlei-  
>np>turtlei->ultimaspos){  
        z=turtlei->np-turtlei->ultimaspos;  
    }  
    else{  
        z=0;
```

```
    }  
    glBegin(GL_QUAD_STRIP);  
    for (i = z; i < turtle->np; i++) {  
        glVertex3f (turtle->px3[i],turtle->py3[i],turtle-  
>pz3[i]);  
        glVertex3f (turtle->px2[i],turtle->py2[i],turtle-  
>pz2[i]);  
    }  
    glEnd();  
  
    break;
```

10. MODO COCHE Y CAMARA SEGUIDORA

Aquí voy a explicar como he hecho para poder controlar los objetos con los cursores dándoles velocidad y con dos tipos de cámara.

Primero hay que crear las siguientes variables globales

```
boolean giroi=FALSE;
boolean girod=FALSE;
boolean seguidora=TRUE;
double velocidad=0;
double angulo=0;
double anguloc=100*PI/180;
```

En poscomando especiales hay que añadir el siguiente código

```
case GLUT_KEY_F4:
    if (current_mode != 0) break;
    current_mode = 4;
    andar=TRUE;

    idle();
    break;
case GLUT_KEY_F5:
    if (seguidora==TRUE)
        seguidora=FALSE;
    else
        seguidora=TRUE;

    break;
```

Ahora hay que poner en el main() lo siguiente para identificar ala función callback tipo idle()

```
glutIdleFunc(idle);
```

y por ultimo hay que definir la función idle() que tiene todo el funcionamiento

```
void idle(void){

    if (andar==TRUE){

        glPushMatrix();
        glLoadIdentity();
        glMultMatrixd(actualTurtle->mModel);
        if (giroi==TRUE){
            glRotatef(anguloc ,0.,1.,0.);
            giroi=FALSE;
        }
        if (girod==TRUE){
```

```

        glRotatef(-anguloc ,0.,1.,0.);
        girod=FALSE;
    }
    glTranslatef(0.0, 0.0, velocidad);

    glGetDoublev (GL_MODELVIEW_MATRIX, actualTurtle->mModel);
    addPointToTrace();
    glPopMatrix();
    if(andar==TRUE){
        GLdouble *m;

        m = actualTurtle->mModel;
        if (seguidora==TRUE){
            LOCAL_MyCamera->camAtX= m[0] * 0 + m[4] * 0 + m[8] * 0 +
m[12];
            LOCAL_MyCamera-> camAtY= m[1] * 0 + m[5] * 0 + m[9] * 0 +
m[13];
            LOCAL_MyCamera->camAtZ = m[2] * 0 + m[6] * 0 + m[10] * 0 +
m[14];

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

            LOCAL_MyCamera->camViewX= 0.0;
            LOCAL_MyCamera->camViewY= 2.0;
            LOCAL_MyCamera->camViewZ = -3.0;
        }
    }
}

```

REPRESENTACIÓN DE UN OBJETO EN FORMATO VRML

Vrml (Virtual Reality Model Lenguaje) es un modo de representar modelos en tres dimensiones en mundos virtuales.

<http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-IS-VRML97WithAmendment1/part1/introduction.html>

Para leer este formato se va a usar la librería cyberVRML97

<http://www.cybergarage.org/vrml/cv97/cv97cc/>

Los archivos que se necesitan para hacer esta práctica son los siguientes y se pueden obtener a través del link de download de la pagina.

flexbison.zip
cv97r110.zip

Dentro de estos zips se encuentran una serie de archivos que son necesarios añadir a nuestro proyecto.
Primero hay que poner un include

```
#include "CyberVRML97.h"
```

Hay que corregir un error que da que se corrige simplemente cambiando los siguientes nombres de funciones
En el archivo node.h y node .cpp cambiar
Crear una función igual que la que hay pero cambiando el DEF por Def están la final de los archivos correspondientes
Esto en el punto h
DEFNode *createDEFNode();
DEFNode *createDefNode();
Y esto en el punto cpp

```
DEFNode *Node::createDEFNode()  
{  
    DEFNode *defNode = new DEFNode();  
  
    Node *refNode = this;  
    while (refNode->isInstanceNode() == true)  
        refNode = refNode->getReferenceNode();  
    defNode->setAsInstanceNode(refNode);  
}
```

```

        return defNode;
    }
    DEFNode *Node::createDefNode()
    {
        DEFNode *defNode = new DEFNode();

        Node *refNode = this;
        while (refNode->isInstanceNode() == true)
            refNode = refNode->getReferenceNode();
        defNode->setAsInstanceNode(refNode);

        return defNode;
    }

```

Para que funcione hay que ponerle unas FLAGS en el menú
 Projects → settings → C/C++ → Preprocessor Definitions
 Hay que poner

,SUPPORT_OPENGL,SUPPORT_STL,SUPPORT_GLUT

Los siguiente es en la clase turtle hay que crear una variable tipo
 SceneGraph que contendrá el modelo

```
SceneGraph *sg;
```

Al final del main cuando se crea la primera tortuga hay que poner

```
turtleVector[0]->sg=NULL;
```

Para inicializarla y ponerla en NULL, lo mismo hay que hacer en
 parsecommand en el comando st

```
turtleVector[ival]->sg=NULL;
```

Ahora en el parsecommand justo de bajo del else if que hace
 funcionar el comando ol ponemos lo siguiente

```

else if (!strcmp("vrml",strToken0)) {
    printf("VRML");
    actualTurtle->sg=new SceneGraph();
    actualTurtle->sg->load(strToken1);
}

```

En la función display() hay que cambiar esto para que dibuje u modelo aunque tiene preferencia el wavefront sobre el vrml

```

if (turtleVector[i]->object != NULL){
    glCallList(turtleVector[i]->object->model_list);
}
else if(turtleVector[i]->sg != NULL){
    DrawSceneGraph(turtleVector[i]->sceneGraph);
}
else{
    drawTurtle();
}

```

Ahora vamos a crear la función SceneGraph() y otras funciones necesarias estas funciones lo que hacen es ir pasando a través d los nodos de la SceneGraph y si detectan que son nodos geométricos los dibujan.

```

void DrawShapeNode(SceneGraph *sg, ShapeNode *shape)
{

```

```

    float color[4];
    color[3] = 1.0f;

    AppearanceNode *appearance = shape-
>getAppearanceNodes();
    MaterialNode *material = NULL;

    bool bEnableTexture = false;
    if (appearance) {

```

```

        material = appearance->getMaterialNodes();
        if (material) {
            float ambientIntensity = material-
>getAmbientIntensity();

            material->getDiffuseColor(color);
            glMaterialfv(GL_FRONT_AND_BACK,
GL_DIFFUSE, color);

            material->getSpecularColor(color);
            glMaterialfv(GL_FRONT_AND_BACK,
GL_SPECULAR, color);

            material->getEmissiveColor(color);
            glMaterialfv(GL_FRONT_AND_BACK,
GL_EMISSION, color);

            material->getDiffuseColor(color);
            color[0] *= ambientIntensity;
            color[1] *= ambientIntensity;
            color[2] *= ambientIntensity;
            glMaterialfv(GL_FRONT_AND_BACK,
GL_AMBIENT, color);

            glMateriali(GL_FRONT, GL_SHININESS,
(int)(material->getShininess()*128.0));
        }
    }

    if (!appearance || !material) {
        color[0] = 0.8f; color[1] = 0.8f; color[2] = 0.8f;
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,
color);
        color[0] = 0.0f; color[1] = 0.0f; color[2] = 0.0f;
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,
color);
        glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION,
color);
        color[0] = 0.8f*0.2f; color[1] = 0.8f*0.2f; color[2] =
0.8f*0.2f;
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,
color);
    }
}

```

```

        glMateriali(GL_FRONT_AND_BACK, GL_SHININESS,
(int)(0.2*128.0));
    }

```

```

float m4[4][4];
shape->getTransformMatrix(m4);
glMultMatrixf((float *)m4);

```

```

glColor3f(1.0f, 1.0f, 1.0f);

```

```

GeometryNode *gnode = shape->getGeometry();
if (gnode) {
    if (0 < gnode->getDisplayList())
        gnode->draw();
}

```

```

}

```

```

void DrawSceneGraph(SceneGraph *sceneGraph, Node *firstNode)
{

```

```

    Node*node;

```

```

    for (node = firstNode; node; node=node->next()) {

```

```

        if (node->isShapeNode()) {

```

```

            DrawShapeNode(sceneGraph, (ShapeNode
*)node);

```

```

        } else{
            DrawNode(sceneGraph, node->getChildNodes());

```

```

        }

```

```

    }

```

```
}
```

Si al linkar da un error de que no puede porque falta la función `getDisplayList()` hay que ir al archivo `geometrynode.cpp` y compilarlo.

No he sido capaz de lograr que lea todos los archivos `vrml` pero archivos simples si que los lee. Parece ser que es debido al tamaño de los archivos complicados y a que uso un puntero en el vector `turtle` pero debido a falta de tiempo no puedo comprobarlo.