


```

}
// if is the first point
if (np == 0) { // add the first point
    px [0] = 0;
    py [0] = 0;
    pz [0] = 0;
    np++;
}
px [np] = m[0] * px [0] + m[4] * py [0] + m[8] * pz [0] + m[12];
py [np] = m[1] * px [0] + m[5] * py [0] + m[9] * pz [0] + m[13];
pz [np] = m[2] * px [0] + m[6] * py [0] + m[10] * pz [0] + m[14];
printf ("Point %i: %f %t%f %t%f \n",
np, px[np],py[np],pz[np]);
np++;
}

```

Función que dibuja el camino: (Primer patrón, esferas tipo humo)

```

void displayTrace() {

    int i;
    int slices = 40;
    int stacks = 40;
    double dist;
    double modulo;
    double dir[3];
    glColor3f(0.2,0.2,0.2) ;
    for (i = 0; i < np; i++) {
        modulo= sqrt((px[i]-px[i-1])*(px[i]-px[i-1])+(py[i]-py[i-1])*(py[i]-py[i-1])+(pz[i]-
pz[i-1])*(pz[i]-pz[i-1]));
        dir[0]= (px[i]-px[i-1])/modulo;
        dir[1]= (py[i]-py[i-1])/modulo;
        dir[2]= (pz[i]-pz[i-1])/modulo;
        dist=0;
        if (px[i-1]!=px[i] || py[i-1]!=py[i] || pz[i-1]!=pz[i]){
            while (dist<modulo){
                glPushMatrix();
                glTranslatef(px[i-1]+dist*dir[0],py[i-1]+dist*dir[1],pz[i-
1]+dist*dir[2]);

                glutSolidSphere(.08,slices,stacks);
                glPopMatrix();
                dist +=0.15;
            }
        }
    }
}

```

Función que dibuja el camino: (Segundo patrón, conos naranjas)

```

void displayTrace() {

    int i;
    int slices = 40;
    int stacks = 40;

    double dist;
    double modulo;

```

```

double dir[3];
glColor3f(1.0,0.5,0.0) ;
for (i = 0; i < np; i++) {
    modulo= sqrt((px[i]-px[i-1])*(px[i]-px[i-1])+(py[i]-py[i-1])*(py[i]-py[i-1])+(pz[i]-
pz[i-1])*(pz[i]-pz[i-1]));
    dir[0]= (px[i]-px[i-1])/modulo;
    dir[1]= (py[i]-py[i-1])/modulo;
    dir[2]= (pz[i]-pz[i-1])/modulo;
    dist=0;
    if (px[i-1]!=px[i] || py[i-1]!=py[i] || pz[i-1]!=pz[i]){
        while (dist<modulo){
            glPushMatrix();
            glTranslatef(px[i-1]+dist*dir[0],py[i-1]+dist*dir[1],pz[i-
1]+dist*dir[2]);

            glRotatef(90.0,-90.0,1.0,0.0);
            glutSolidCone(0.07,0.2,28,28); //Radio base, altura
            glPopMatrix();
            dist +=0.13;
        }
    }
}
}
}

```

Tercer patrón, camino en zig-zag

Guardando el camino, de esta forma:

```

void addPointToTrace() {
    int i;
    GLdouble m[16];
    glGetDoublev (GL_MODELVIEW_MATRIX, m);
    // print the matrix
    printf ("\nMatrix:\n");
    for (i = 0; i < 4; i++) {
        printf ("Row %i: %f \t%f \t%f \t%f \n",
            i+1, m[i+0],m[i+4],m[i+8],m[i+12]);
    }
    // if is the first point
    if (np>2){
        np++;
        np--;
    }

    if (np == 0) { // add the first point
        px [0] = -0.4f;
        py [0] = 0;
        pz [0] = 0;
        np++;

        px[np]= 0.4f;
        py[np]= 0;
        pz[0] = 0;
        np++;
    }
    px [np] = m[0] * px [0] + m[4] * py [0] + m[8] * pz [0] + m[12];
    py [np] = m[1] * px [0] + m[5] * py [0] + m[9] * pz [0] + m[13];
    pz [np] = m[2] * px [0] + m[6] * py [0] + m[10] * pz [0] + m[14];
    //printf ("Point %i: %f \t%f \t%f \n",

```

```

//np, px[np],py[np],pz[np]);
np++;
px [np] = m[0] * px [1] + m[4] * py [1] + m[8] * pz [1] + m[12];
py [np] = m[1] * px [1] + m[5] * py [1] + m[9] * pz [1] + m[13];
pz [np] = m[2] * px [1] + m[6] * py [1] + m[10] * pz [1] + m[14];
np++;
}

```

Función para dibujar el camino:

```

void displayTrace() {
    int i, q=0;
    int slices = 40;
    int stacks = 40;
    double dist;
    double modulo;
    double dir[3];
    glColor3f(1.0,1.0,1.0) ;

    glBegin(GL_LINE_STRIP);
    for (i = 0; i < np-3; i+=2) {
        modulo= sqrt((px[i]-px[i+2])*(px[i]-px[i+2])+(py[i]-
py[i+2])*(py[i]-py[i+2])+(pz[i]-pz[i+2])*(pz[i]-pz[i+2]));
        //vector unitario
        dir[0]= (px[i+2]-px[i])/modulo;
        dir[1]= (py[i+2]-py[i])/modulo;
        dir[2]= (pz[i+2]-pz[i])/modulo;
        dist=0;
        while (dist<modulo){
            if (q==0){
                glVertex3f
(px[i]+dir[0]*dist,py[i]+dir[1]*dist,pz[i]+dist*dir[2]);
                dist +=0.3;
                glVertex3f
(px[i+1]+dir[0]*dist,py[i+1]+dir[1]*dist,pz[i+1]+dist*dir[2]);
                q=1;
            } else if (q==1){
                glVertex3f (px[i+1]+dir[0]*dist,py[i+1]+dist
*dir[1] ,pz[i+1]+dist*dir[2]);
                dist+=0.3;
                glVertex3f
(px[i]+dir[0]*dist,py[i]+dist*dir[1],pz[i]+dist*dir[2]);
                q=0;
            }
        }
    }
    glEnd();
}

```

Comando History

Variables globales: char *lista[30], *tmp;
 int num_lista;

Función Hystory:

```
void history(int n){
    if (n>0){
        printf ("\n*****Comando History*****\n");
        printf ("\nComandos introducidos:");
        for (i=n-1; i<num_lista; i++){
            printf( "\n %d .- %s", i+1, lista[i] );
        }
        printf ("\n\n*****\n");
    }else if (n<0){
        printf ("\n*****Comando History*****\n");
        printf ("\nUltimos %d comandos introducidos:", n);
        for (i=num_lista+n+1; i<num_lista; i++){
            printf( "\n %d .- %s", i+1, lista[i] );
        }
        printf ("\n\n*****\n");
    }
}
```

Implementamos la función parsecommand con los siguientes comandos despues de los "if" para girar y desplazar la tortuga

```
    } else if (!strcmp("hi",strToken0)) { //comando history
        history(val);
    }

    tmp = (char *)malloc(20);
    strcpy(tmp, strToken0);
    strcat(tmp, " ");
    strcat(tmp, strToken1);
    //strcat(tmp, 0);
    lista[num_lista] = (char *) malloc(strlen(tmp)+1);
    strcpy(lista[num_lista], tmp);
    free (tmp);
    num_lista++;
```

Comando RECOVERY

Manteniendo todo lo necesario para la función HISTORY añadimos:

Declaraciones globales:

```
    boolean cargar=TRUE;
    char *tmp2;
    void history(int n);
    void recovery(int val);
```

Incluimos en la funcion ParseCommand:

```
    } else if (!strcmp("rn",strToken0)) {
        val2=val;
        tmp2 = (char *)malloc(20);
        strcpy(tmp2, "Hola");
        strcpy(tmp2, lista[val2-1]);
        printf( "\n %s ", lista[val2-1] );
        parseCommand(lista[val2-1]);
        cargar=FALSE;
        strcpy(lista[val2-1], tmp2);
```

```

        free (tmp2);
    }

    tmp = (char *)malloc(20);
    strcpy(tmp, strToken0);
    strcat(tmp, " ");
    strcat(tmp, strToken1);
    //strcat(tmp, 0);
    lista[num_lista] = (char *) malloc(strlen(tmp)+1);
    strcpy(lista[num_lista], tmp);
    free (tmp);
    num_lista++;
    agregar=FALSE;

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

    if (cargar==FALSE)    cargar =TRUE;
    else{
        glGetDoublev
(GL_MODELVIEW_MATRIX, mModel);
    }
    glPopMatrix();

```

Comando HIDETURTLE y SHOWTURTLE

Variable global: boolean ocultar_tortuga=FALSE;

Modificamos la función display:

```

    glPushMatrix();
    glMultMatrixd(mModel);
    glColor3f(1.0,1.0,0.0) ;

    if (ocultar_tortuga==FALSE){
        drawTurtle();
        dibujar_ejes();
    }

    glPopMatrix();
    displayTrace();
    dibujar_ejes();

```

El comienzo de la función parsecommand queda de la siguiente manera:

```

void parseCommand(char* strCommandParse) {
    char *strToken0;
    char *strToken1;
    double val;
    strToken0 = strtok(strCommandParse, " ");

    while (1){
        if (strToken0 == NULL) break;
        glPushMatrix();
        glLoadIdentity();

```

```

glMultMatrixd(mModel);

if (!strcmp("ht",strToken0)) {
    ocultar_tortuga=TRUE;
} else if (!strcmp("st",strToken0)) {
    ocultar_tortuga=FALSE;
} else if ((strToken1 = strtok(NULL, " ")) != NULL) {
    val = atof(strToken1);

    if (!strcmp("fd",strToken0)) { // FORWARD
        glTranslatef(0.0, 0.0, val);
        addPointToTrace();
    } else if (!strcmp("bk",strToken0)) { // BACK
        glTranslatef(0.0, 0.0, -val);
        addPointToTrace();
    } else if (!strcmp("ht",strToken0)) { // RIGHT
        ocultar_tortuga=TRUE;
    } else if (!strcmp("lt",strToken0)) { // LEFT
        glRotatef(val,0.,1.,0.);
    } else if (!strcmp("up",strToken0)) { // UP
        glRotatef(val,1.,0.,0.);
    } else if (!strcmp("dn",strToken0)) { // DOWN
        glRotatef(-val,1.,0.,0.);
    }
}
}

```

Ejes globales y de la tortuga

Codigo para la función display:

```

drawTurtle();
dibujar_ejes();
glPushMatrix(); //cargamos una matriz nueva
glLoadIdentity(); //cargamos la identidad
dibujar_ejes(); //Dibujamos en los ejes globales
glPopMatrix(); //Volvemos a dejar la matriz anterior

```

Comando PENUP y PENDOWN

Variables globales:

boolean dibujar_rastro=TRUE;

Función e dibujar rastro en la función display: if (dibujar_rastro==TRUE)displayTrace();

Implementamos la función parsecommand con el siguiente codigo:

```

while (1){
    if (strToken0 == NULL) break;
    glPushMatrix();
    glLoadIdentity();
    glMultMatrixd(mModel);

    if (!strcmp("pd",strToken0)) { //rastro false
        dibujar_rastro=FALSE;
    } else if (!strcmp("pu",strToken0)) { // rastro ok
        if (dibujar_rastro==FALSE){
            dibujar_rastro=TRUE;
        }
    }
}

```

```

        np=0;
        addPointToTrace();
        px [0]=px[1];
        py [0]=py[1];
        pz [0]=pz[1];
    }
} else if ((strToken1 = strtok(NULL, " ") != NULL) {
    val = atof(strToken1);

    if (!strcmp("fd",strToken0)) { // FORWARD
        glTranslatef(0.0, 0.0, val);
        addPointToTrace();

        ....
        ....
    }
}
}

```

La función addPointToTrace quedará de la siguiente forma:

```

void addPointToTrace() {
    int i;

    GLdouble m[16];
    glGetDoublev (GL_MODELVIEW_MATRIX, m);
    // print the matrix
    printf ("\nMatrix:\n");
    for (i = 0; i < 4; i++) {
        printf ("Row %i: %f \t%f \t%f \t%f \n",
            i+1, m[i+0],m[i+4],m[i+8],m[i+12]);
    }
    // if is the first point
    if (np == 0) { // add the first point
        px [0] = 0;
        py [0] = 0;
        pz [0] = 0;
        np++;
    }
    px [np] = m[0] * 0 + m[4] * 0 + m[8] * 0 + m[12];
    py [np] = m[1] * 0 + m[5] * 0 + m[9] * 0 + m[13];
    pz [np] = m[2] * 0 + m[6] * 0 + m[10] * 0 + m[14];
    printf ("Point %i: %f \t%f \t%f \n",
        np, px[np],py[np],pz[np]);
    np++;
}

```

RASTRO DIFUMINADO

Creamos las variables gloabales: double m,q;
 Inicializamos en main: m=0;
 Implementamos la función parseCommand: else if (!strcmp("rastros",strToken0)) {
 if (val==0) m=1;
 else m=1.0/(val);
 }

La función DisplayTrace queda de la siguiente forma:

```

void displayTrace() {

```

```

int i;
if (np>0){
    q=1;
    glBegin(GL_QUAD_STRIP);
    for (i = np-1; i >=0; i-) {

        glColor3f(q*1.0,q*1.0,q*1.0);
        glVertex3f (.....);

        glColor3f(q*1.0,q*1.0,q*1.0);
        glVertex3f
        (.....);
        q=q-m;

        glColor3f(q*1.0,q*1.0,q*1.0);
        glVertex3f
        (.....);

        glColor3f(q*1.0,q*1.0,q*1.0);
        glVertex3f (.....);
    }
    glEnd();
}
}

```

Comandos CLEAN, HOME, CLEARSCREEN

Completamos la función ParseCommand con los siguientes comandos:

```

if (strToken0 != NULL && strncmp(strToken0, "exit", 4) == 0) {
    command = FALSE;
    // HOME
} else if (strToken0 != NULL && !strcmp("cs",strToken0)) {
    glVertex3f (.....); //Cargamos una
matriz nueva en la modelview
glLoadIdentity(); //La sustituimos por la
identidad
glGetDoublev (GL_MODELVIEW_MATRIX,
mModel);
addPointToTrace();
glPopMatrix();
np=0;
glPushMatrix(); //Cargamos una
matriz nueva en la modelview
glLoadIdentity(); //La sustituimos por la
identidad
glMultMatrixd(mModel);
addPointToTrace();
px [0] = px[1];
py [0] = py[1];
pz [0] = pz[1];
glPopMatrix();
display();
}
}

```

```

} else if (strToken0 != NULL && !strcmp("home",strToken0)) {
    glPushMatrix();           //Cargamos una
matriz nueva en la modelview
                                glLoadIdentity();           //La sustituimos por la
identidad
                                glGetDoublev (GL_MODELVIEW_MATRIX,
mModel);
                                addPointToTrace();
                                glPopMatrix();
                                display();
} else if (strToken0 != NULL && !strcmp("clear",strToken0)) {
    np=0;
matriz nueva en la modelview
                                glPushMatrix();           //Cargamos una
identidad
                                glLoadIdentity();           //La sustituimos por la

                                glMultMatrixd(mModel);
                                addPointToTrace();
                                px [0] = px[1];
                                py [0] = py[1];
                                pz [0] = pz[1];
                                glPopMatrix();
                                display();
}
}

```

Comando Load

Incluimos en la función main:

```

FILE* file;
char buf[128];
int i = 0;
.....
file = fopen("comandos.txt", "r");           //Comandos.txt

archivo a cargar
encuentre el archivo
filename);

if (!file) {                               //Caso de q no se

    fprintf(stderr, "open failed: can't open file \"%s\".\n",
    exit(0);
}

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

```

Comando LOG

Variables globales: FILE* file;
char filename[] = " " ;
boolean escribir=FALSE;

Dejamos el final de la función ParseCommand de la siguiente manera:

```

LOG
} else if (!strcmp("log",strToken0)) { // Comando
    if (escribir==FALSE){
        escribir=TRUE;
        strcpy(filename, strToken1);
        file = fopen( filename, "a" );
    }
}

if (escribir==TRUE){
    fprintf(file, "Comando: %s %s\n ",
strToken0, strToken1 );
}

strToken0 = strtok(NULL, " ");
display();
glGetDoublev (GL_MODELVIEW_MATRIX,
mModel);
glPopMatrix();
}
if (strToken0 != NULL && strcmp(strToken0, "exit", 4) ==
0) {
    command = FALSE;
} else if (strToken0 != NULL &&
!strcmp("logstop",strToken0)) { // Comando Logstop
    escribir=FALSE;
    fclose( file);
}
}

```

Comando LOG fileName n m

Manteniendo lo necesario para las funciones LOG y HISTORY:

Creamos las siguientes variables en la funcion parseCommand: char *numero, *numero2;
double num, num2;

Sustituimos el codigo de la funcion LOG incluido en ParseCommand por:

```

Comando LOG
} else if (!strcmp("log",strToken0)) { // Comando LOG
    if (escribir==FALSE){
        escribir=TRUE;
        strcpy(filename, strToken1);
        file = fopen( filename, "w" );

        numero= strtok(NULL, " ");
        numero2= strtok(NULL, " ");
        //Comprobamos si despues del
nombre del archivo se han introducido 2 numeros
if (!strcmp("0",numero) ||
!strcmp("1",numero) || !strcmp("2",numero) || !strcmp("3",numero) ||
!strcmp("4",numero) ||

```



```

{
buf);

//printf("Line %i: %s", ++i,
xx = strtok( buf, ", " );
x = atof(xx);
x=atof(xx);
p2x[num_lineas]=x;
xx = strtok(NULL, ",");
x=atof(xx);
p2y[num_lineas]=x;
xx = strtok(NULL, ",");
x=atof(xx);
p2z[num_lineas]=x;
num_lineas++;
}
}
}

```

Añadimos a la función DisplayTrace:

```

if (rastros==TRUE){
glColor3f(1.0,1.0,0.0) ;
glBegin(GL_LINE_STRIP);
for (i = 0; i < num_lineas; i++) {
glVertex3f (p2x[i],p2y[i],p2z[i]);
}
glEnd();
}
}

```

Salvar el rastro de un objeto en formato Wavefront

Variables globales: FILE* file;
char filename[] = "file.obj";

Añadimos a la función parseCommand:

```

SAVETRACE } else if (!strcmp("savetrace",strToken0)) { //
strcpy(filename , strToken1);
file = fopen(filename, "w");
for (i=0; i<np; i++){
fputs("v ",file);
fprintf(file , "%f", px[i]);
fprintf(file , " ");
fprintf(file , "%f", py[i]);
fprintf(file , " ");
fprintf(file , "%f\n", pz[i]);
}
for (i = 1; i <= np-3; i=i+2) {
fputs("f ",file);
fprintf(file , "%i", i);
fprintf(file , " ");
fprintf(file , "%i", i+1);
fprintf(file , " ");
}
}

```

```

        fprintf(file , "%i", i+3);
        fprintf(file , " ");
        fprintf(file , "%\n", i+2);
    }
    fclose( file);
}

```

Comando BACKGROUND

Variables globales: double r=0.0, g=0.0, b=0.0;
 Añadimos a la función display: glClearColor(r, g, b, 0.0);

Añadimos a la función ParseCommand:

```

else if (!strcmp("rgb",strToken0)) {
    //*****RGB*****
        r= val;
        if ((strToken1 = strtok(NULL," ")) != NULL) {
            val = atof(strToken1);
            g= val;
            if ((strToken1 = strtok(NULL," ")) != NULL) {
                val = atof(strToken1);
                b= val;
            }
        }
    }
}

```

Comandos SUPERPOSEON, SUPERPOSEOFF, CLEARSCENE

Variables globales: boolean dibujar=FALSE;
 boolean guardar=FALSE;
 GLdouble estatico[16];

Añadimos a la función main: glGetDoublev (GL_MODELVIEW_MATRIX, estatico);

Añadimos a la función display:

```

if (dibujar==TRUE){
    glPushMatrix();
    glMultMatrixd(estatico);
    glColor3f(1.0,1.0,0.0) ;
    drawTurtle();
    glPopMatrix();
    glutSwapBuffers();
}

```

Añadimos a la función ParseCommand:

Al inicio de la función: if (guardar==TRUE){

```

    for (i=0; i<16; i++){
        estatico[i]=mModel[i];
    }
}

```

Al final de la función: } else if (strToken0 != NULL && !strcmp("superposeon",strToken0))

```

{
    //Modo SuperposeON

```

```

        guardar= TRUE;
        dibujar =TRUE;
        for (i=0; i<16; i++){
            estatico[i]=mModel[i];
        }
    } else if (strToken0 != NULL && !strcmp("superposeoff",strToken0))
{
        guardar= FALSE;
    } else if (strToken0 != NULL && !strcmp("clearscene",strToken0)) {
        dibujar=FALSE;
        guardar= FALSE;
    }
}

```

Rastro con los colores del arco iris

Asi queda la función que guarda los puntos del camino:

```

void addPointToTrace() {
    int i;

    GLdouble m[16];
    glGetDoublev (GL_MODELVIEW_MATRIX, m);
    // print the matrix
    printf ("\nMatrix:\n");
    for (i = 0; i < 4; i++) {
        printf ("Row %i: %f \t%f \t%f \t%f \n",
            i+1, m[i+0],m[i+4],m[i+8],m[i+12]);
    }
    // if is the first point
    if (np>2){
        np++;
        np--;
    }

    if (np == 0) { // add the first point
        px [0] = 0;
        py [0] = 0;
        pz [0] = 0;
        np++;

        px[np]= -0.8f;
        py[np]= 0;
        pz[0] = 0;
        np++;
    }
    px [np] = m[0] * px [0] + m[4] * py [0] + m[8] * pz [0] + m[12];
    py [np] = m[1] * px [0] + m[5] * py [0] + m[9] * pz [0] + m[13];
    pz [np] = m[2] * px [0] + m[6] * py [0] + m[10] * pz [0] + m[14];
    //printf ("Point %i: %f \t%f \t%f \n",
    //np, px[np],py[np],pz[np]);
    np++;
    px [np] = m[0] * px [1] + m[4] * py [1] + m[8] * pz [1] + m[12];
    py [np] = m[1] * px [1] + m[5] * py [1] + m[9] * pz [1] + m[13];
    pz [np] = m[2] * px [1] + m[6] * py [1] + m[10] * pz [1] + m[14];
    np++;
}

```

La función DisplayTrace que dibuja el camino queda de la siguiente forma:

```
void displayTrace() {
    int i;
    double nx, ny, nz;
    glColor3f(1.0,1.0,1.0) ;
    //glBegin(GL_QUADS);
    //glBegin(GL_QUAD_STRIP);
    for (i = 0; i < np-3; i+=2) {
        nx=(px[i]-px[i+1])/4;
        ny=(py[i]-py[i+1])/4;
        nz=(pz[i]-pz[i+1])/4;

        glBegin(GL_QUAD_STRIP);
        glColor3f(1.0,0.0,0.0) ;
        glVertex3f (px[i], py[i],pz[i]);
        glVertex3f (px[i+2], py[i+2],pz[i+2]);
        glColor3f(1.0,1.0,0.0) ;
        glVertex3f (px[i]-nx, py[i]-ny,pz[i]-nz);
        glVertex3f (px[i+2]-nx, py[i+2]-ny,pz[i+2]-nz);
        glEnd();

        glBegin(GL_QUAD_STRIP);
        glColor3f(1.0,1.0,0.0) ;
        glVertex3f (px[i]-nx, py[i]-ny,pz[i]-nz);
        glVertex3f (px[i+2]-nx, py[i+2]-ny,pz[i+2]-nz);
        glColor3f(0.0,1.0,0.0) ;
        glVertex3f (px[i]-2*nx, py[i]-2*ny, pz[i]-2*nz);
        glVertex3f (px[i+2]-2*nx, py[i+2]-2*ny,pz[i+2]-2*nz);
        glEnd();

        glBegin(GL_QUAD_STRIP);
        glColor3f(0.0,1.0,0.0) ;
        glVertex3f (px[i]-2*nx, py[i]-2*ny, pz[i]-2*nz);
        glVertex3f (px[i+2]-2*nx, py[i+2]-2*ny,pz[i+2]-2*nz);
        glColor3f(0.0,1.0,1.0) ;
        glVertex3f (px[i]-3*nx, py[i]-3*ny, pz[i]-3*nz);
        glVertex3f (px[i+2]-3*nx, py[i+2]-3*ny,pz[i+2]-3*nz);
        glEnd();

        glBegin(GL_QUAD_STRIP);
        glColor3f(0.0,1.0,1.0) ;
        glVertex3f (px[i]-3*nx, py[i]-3*ny, pz[i]-3*nz);
        glVertex3f (px[i+2]-3*nx, py[i+2]-3*ny,pz[i+2]-3*nz);
        glColor3f(0.0,0.0,1.0) ;
        glVertex3f (px[i]-4*nx, py[i]-4*ny,pz[i]-4*nz);
        glVertex3f (px[i+2]-4*nx, py[i+2]-4*ny,pz[i+2]-4*nz);
        glEnd();
    }
}
```

Rastro con objetos

Variables globales: turtle *camino;
 boolean dibujar_camino=FALSE;

Asi queda la función addPointToTrace:

```
void addPointToTrace() {
    int i;
    GLdouble *m;
    m = actualTurtle->mModel;
    if (actualTurtle->np == 0) {
        actualTurtle->px [0] = 0;
        actualTurtle->py [0] = 0;
        actualTurtle->pz [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->np++;
}
}
```

Asi queda la función displayTrace:

```
void displayTrace(turtle* turtlei) {
    int i;
    double dist;
    double modulo;
    double dir[3], angulo[3];

    if (dibujar_camino==TRUE){
        for (i = 1; i < turtlei->np; i++) {
            modulo= sqrt((turtlei->px[i]-turtlei->px[i-1])*(turtlei-
>px[i]-turtlei->px[i-1])+(turtlei->py[i]-turtlei->py[i-1])*
(turtlei->py[i]-turtlei->py[i-
1])+(turtlei->pz[i]-turtlei->pz[i-1])*(turtlei->pz[i]-turtlei->pz[i-1]));
            dir[0]= (turtlei->px[i]-turtlei->px[i-1])/modulo;
            dir[1]= (turtlei->py[i]-turtlei->py[i-1])/modulo;
            dir[2]= (turtlei->pz[i]-turtlei->pz[i-1])/modulo;
            //Producto escalar
            angulo[0]=acos(dir[0]);
            angulo[0]=angulo[0]*180/3,14159265358;
            angulo[1]=acos(dir[1]);
            angulo[1]=angulo[1]*180/3,14159265358;

            dist=0;
            if (turtlei->px[i-1]!=turtlei->px[i] || turtlei->py[i-
1]!=turtlei->py[i] || turtlei->pz[i-1]!=turtlei->pz[i]){
                while (dist<modulo){
                    glPushMatrix();
                    glTranslatef(turtlei->px[i-
```

```

1]+dist*dir[0],turtlei->py[i-1]+dist*dir[1],turtlei->pz[i-1]+dist*dir[2]);
                                                                    glScalef(0.2f, 0.2f, 0.2f);

                                                                    glRotatef(-(angulo[0])+90 , 0.0 ,
1.0 , 0.0 );
                                                                    glRotatef( angulo[1]-90 ,1.0 , 0.0 ,
0.0 );

                                                                    glCallList(camino->object-
>model_list);
                                                                    glutSolidCone(0.07,0.2,28,28);

//Radio base, altura
                                                                    glPopMatrix();
                                                                    dist +=0.4;
                                                                    }
                                                                    }
} else {
    glBegin(GL_LINE_STRIP);
    glColor3f(1.0,1.0,1.0) ;
    for (i = 0; i < turtlei->np; i++) {
        glVertex3f (turtlei->px[i],turtlei->py[i],turtlei->pz[i]);
    }
    glEnd();
}
}
}

```

Añadimos a la función ParseCommand:

```

} else if (!strcmp("SETTRACEOBJECT",strToken0)) {
    camino = (turtle*) malloc(sizeof(turtle));
    camino = (turtle*) malloc(sizeof(turtle));
    camino->object = createGlmObject(strToken1);
    dibujar_camino=TRUE;
}

```

Ventana con vistas de perfil, alzado, planta y cámara.

Declaraciones globales:

```

static camera *Camera[3];
int camara;
int w, h;

```

Función para escribir el texto:

```

void display3(){
    if (command) {
        glColor3f(1.0,1.,0.0) ;
        text(5, 5, 20, "->%s", strCommand);
    }
}

```

Función display2:

```

void display2(){
    float At[3];
    float Direction[3];

```

```

        glEnable(GL_LIGHTING);

        if (camara==0) SetGLCamera( LOCAL_MyCamera );
        else if (camara==1) SetGLCamera( Camera[0]);
        else if (camara==2) SetGLCamera( Camera[1]);
        else if (camara==3) SetGLCamera( Camera[2]);

        SetLight( LOCAL_MyLights[0] );
        SetLight( LOCAL_MyLights[1] );
        SetLight( LOCAL_MyLights[2] );

        .....(como estaba la función display)
        .....

        dibujar_ejes();
        displayTrace();
    }

```

Función display:

```

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);

    glViewport(0, h/2, w/2, h/2);
    camara=0;
    display2();

    glViewport(w/2, h/2, w/2, h/2);
    camara=1;
    display2();

    glViewport(0, 0, w/2, h/2);
    camara=2;
    display2();

    glViewport(w/2, 0, w/2, h/2);
    camara=3;
    display2();

    glViewport(0, 0, w, h);
    display3();

    glutSwapBuffers();
}

```

Incluimos en la función main:

```

LOCAL_MyCamera = CreatePositionCamera(4.0f, 1.0f, -3.0f);
Camera[0] = CreatePositionCamera(0.0f, 0.0f, -4.0f);
Camera[1] = CreatePositionCamera(0.0f, 4.0f, -0.0f);
Camera[2]= CreatePositionCamera(4.0f, 0.0f, 0.0f);

```