Analizador Sintáctico

Ariana Bermúdez,Ximena Bolaños, Dylan Rodríguez

Instituto Tecnológico de Costa Rica

May 30, 2017

Análisis Sintáctico

Se hizo un analizador sintáctico que es bottom-up con la ayuda de la herramienta de Bison, para el lenguaje C y que corre en C, este analizador trabaja en conjunto con Flex, para tomar los tokens que este le otorga y revisar con las gramáticas que les sean ingresadas. Bison fue escrito por Robert Corbett y Richard Stallman lo hizo compatible con Yacc. Wilfed Hansen de la Universidad de Carnegie Mellon le adicciono multicaractes de hileras. Bison convierte de una gramática libre de contexto a un analizador sintáctico que emplea las tablas de Parsing LALR(1), siendo:

- L: Look-
- A: Ahead
- L: Left to-
- R: Right
- (1): tiene como lookahead solo un símbolo.

Cabe destacar que Bison es compatible con Yacc. Sirve con C, C++ y Java.

Bison

Si usted quiere descargar bison se puede meter a cualquiera de los siguientes links, el software es completamente gratis:

- http://ftp.gnu.org/gnu/bison/
- ftp://ftp.gnu.org/gnu/bison/

Su documentacion es gratuita online o en los direcctorios locales del sistema en el que haya instalado ya sea /usr/share/bison/, /usr/local/doc/bison/ o directorios similares. Tambien bison contiene listas de correos:

- bug-bison: que sirva para reportar errores en el programa
- help-bison: usado para ayuda en general.
- bison-patches: que es para arreglos del programa.

Código

```
static int Hres ;
static int Vres ;
static int maxAA ;
static int maxReflection ;
static int maxTransparency ;
static long double rays = 0 ;
static long double similar = 0 ;
static long double Xmax ;
static long double Ymax ;
static long double Xmin ;
static long double Ymin ;
static long double Xdif ;
```

Código

```
static long double Ydif ;
static long double Ia ;
static long double e ;
static int debug = 0 ;
static int rec = 0 ;
struct Color {
long double r ;
long double g ;
long double b ;
} ;
struct Point2D {
long double u ;
long double v ;
```

Código

```
} ;
struct Point3D {
long double x ;
long double y ;
long double z ;
} ;
struct Vector {
long double x ;
long double y ;
long double z ;
} ;
struct Light {
long double Xp ;
```

Código

```
long double Yp ;
long double Zp ;
long double c1 ;
long double c2 ;
long double c3 ;
long double Ip ;
} ;
struct Object {
long double Xc ;
long double Yc ;
long double Zc ;
long double other ;
struct Vector directionVector ;
```

Código

```
long double extraD ;
long double Xother ;
long double Yother ;
long double Zother ;
long double Kd ;
long double Ka ;
long double Kn ;
long double Ks ;
long double o1 ;
long double o2 ;
long double o3 ;
long double A ;
long double B ;
```

Código

```
long double C ;
long double D ;
long double E ;
long double F ;
long double G ;
long double H ;
long double I ;
long double J ;
int pointAmount ;
long double D1 ;
long double D2 ;
long double K1 ;
long double K2 ;
```

Código

```
long double height ;
struct Color color ;
struct Point2D * points2D ;
struct Point3D * points3D ;
struct Vector ( * normalVector ) ( ) ;
struct Intersection ( * intersectionFuncion ) ( ) ;
struct Color ( * retrieveTextureColor ) ( ) ;
struct Vector x0y0z0 ;
struct Vector x1y1z1 ;
struct Vector x2y2z2 ;
struct Vector x3y3z3 ;
int numberPlaneCuts ;
int numberTextures ;
```

Código

```
int numberDraftPlanes ;
struct PlaneCut * planeCuts ;
struct Texture * textures ;
struct DraftPlane * draftPlanes ;
} ;
struct PlaneCut {
struct Vector normal ;
struct Vector point ;
long double d ;
} ;
struct Texture {
char * filename ;
struct Color * * textureMap ;
```

Código

```
int vRes ;
int hRes ;
struct Vector greenwich ;
struct Vector north ;
} ;
struct DraftPlane {
char * filename ;
struct Color * * textureMap ;
int vRes ;
int hRes ;
struct Vector greenwich ;
struct Vector north ;
} ;
```

Código

```
struct Intersection {
long double Xi ;
long double Yi ;
long double Zi ;
long double distance ;
struct Object object ;
long double null ;
} ;
static struct Light * Lights ;
static struct Object * Objects ;
static struct Vector eye ;
static struct Color * * Framebuffer ;
static struct Color background ;
```

Código

```
static int numberObjects = 0 ;
static int numberLights = 0 ;
static int lightIndex = 0 ;
static int objectIndex = 0 ;
static char * escenaFile = "escena1.txt" ;
long double min ( long double a , long double b ) {
if ( a < b ) { return a ; }
else { return b ; }
}
long double max ( long double a , long double b ) {
if ( a > b ) { return a ; }
else { return b ; }
}
```

Código

```
int testPlaneCut ( struct PlaneCut plane , long double x , long double y , long double z ) {
int val = ( plane . normal . x * x ) + ( plane . normal . y * y ) + ( plane . normal . z * z ) + plane . d
if ( val > 0 ) {
return 1 ;
} else {
return 0 ;
}
}
int testIntersection ( long double x , long double y , long double z , struct Object object ) {
int k , sign ;
int accept = 1 ;
int amount = object . numberPlaneCuts ;
for ( k = 0 ; k < amount ; k ++ ) {
```

Código

```
sign = testPlaneCut ( object . planeCuts [ k ] , x , y , z ) ;
if ( sign == 1 ) {
accept = 0 ;
}
}
return accept ;
}
long double pointProduct ( struct Vector a , struct Vector b ) {
long double pp = 0 ;
pp += ( a . x * b . x ) ;
pp += ( a . y * b . y ) ;
pp += ( a . z * b . z ) ;
return pp ;
```

Código

```
}
struct Vector crossProduct ( struct Vector a , struct Vector b ) {
struct Vector newVector ;
newVector . x = ( a . y * b . z ) - ( a . z * b . y ) ;
newVector . y = ( a . z * b . x ) - ( a . x * b . z ) ;
newVector . z = ( a . x * b . y ) - ( a . y * b . x ) ;
return newVector ;
}
long double getNorm ( struct Vector vector ) {
long double norm = sqrt ( pow ( vector . x , 2 ) + pow ( vector . y , 2 ) + pow ( vector . z , 2 ) ) ;
return norm ;
}
struct Vector normalize ( struct Vector vector ) {
```

Código

```
long double norm = getNorm ( vector ) ;
struct Vector unitVector ;
if ( norm != 0 ) {
unitVector . x = vector . x / norm ;
unitVector . y = vector . y / norm ;
unitVector . z = vector . z / norm ;
} else {
unitVector . x = vector . x ;
unitVector . y = vector . y ;
unitVector . z = vector . z ;
}
return unitVector ;
}
```

Código

```
void saveFile ( ) {
int i , j ;
FILE * file ;
file = fopen ( "scene.ppm" , "w" ) ;
if ( file == NULL ) {
printf ( "Error creating/opening file!\n" ) ;
exit ( 1 ) ;
}
fprintf ( file , "%s\n" , "P3" ) ;
fprintf ( file , "%i %i\n" , Hres , Vres ) ;
fprintf ( file , "%i\n" , 255 ) ;
for ( i = Vres - 1 ; i >= 0 ; i -- ) {
for ( j = 0 ; j < Hres ; j ++ ) {
```

Código

```
int R = ( int ) 255 * Framebuffer [ i ] [ j ] . r ;
int G = ( int ) 255 * Framebuffer [ i ] [ j ] . g ;
int B = ( int ) 255 * Framebuffer [ i ] [ j ] . b ;
fprintf ( file , "%i %i %i    " , R , G , B ) ;
}
fprintf ( file , "\n" ) ;
}
fclose ( file ) ;
}
long double getAttenuationFactor ( struct Light light , long double distance ) {
long double value = 1 / ( light . c1 + ( light . c2 * distance ) + ( light . c3 * pow ( distance , 2 ) ) )
return min ( 1.0 , value ) ;
}
```

Código

```
struct Color difusseColor ( long double I , struct Color color ) {
struct Color newColor ;
newColor . r = I * color . r ;
newColor . g = I * color . g ;
newColor . b = I * color . b ;
return newColor ;
}
struct Color specularHighlight ( long double E , struct Color color ) {
struct Color newColor ;
newColor . r = color . r + ( E * ( 1 - color . r ) ) ;
newColor . g = color . g + ( E * ( 1 - color . g ) ) ;
newColor . b = color . b + ( E * ( 1 - color . b ) ) ;
return newColor ;
```

Código

```
}
struct Intersection sphereIntersection ( struct Vector anchor , struct Vector direction , struct Object ob
long double t , t1 , t2 ;
long double Xdif = anchor . x - object . Xc ;
long double Ydif = anchor . y - object . Yc ;
long double Zdif = anchor . z - object . Zc ;
struct Intersection tempIntersect ;
tempIntersect . null = 0 ;
long double B = 2 * ( ( direction . x * Xdif ) + ( direction . y * Ydif ) + ( direction . z * Zdif ) ) ;
long double C = pow ( Xdif , 2 ) + pow ( Ydif , 2 ) + pow ( Zdif , 2 ) - pow ( object . other , 2 ) ;
long double discriminant = pow ( B , 2 ) - ( 4 * C ) ;
if ( discriminant >= 0 ) {
long double root = sqrt ( discriminant ) ;
```

Código

```
B *= - 1 ;
t1 = ( B + root ) / 2 ;
t2 = ( B - root ) / 2 ;
if ( t1 > e ) {
if ( t2 > e ) {
t = min ( t1 , t2 ) ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
```

Código

```
t = max ( t1 , t2 ) ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
}
} else {
t = t1 ;
```

Código

```
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
}
} else {
if ( t2 > e ) {
t = t2 ;
```

Código

```
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
} else {
tempIntersect . null = 1 ;
}
}
```

Código

```
return tempIntersect ;
} else {
tempIntersect . null = 1 ;
return tempIntersect ;
}
}
struct Vector sphereNormal ( struct Object object , struct Vector vector ) {
struct Vector normal ;
normal . x = vector . x - object . Xc ;
normal . y = vector . y - object . Yc ;
normal . z = vector . z - object . Zc ;
return normal ;
}
```

Código

```
int getSign ( long double v ) {
if ( v >= 0 ) { return 1 ; }
else { return 0 ; }
}
struct Intersection polygonIntersection ( struct Vector anchor , struct Vector direction , struct Object o
long double denominator = ( direction . x * object . Xc ) + ( direction . y * object . Yc ) + ( direction
struct Intersection tempIntersect ;
tempIntersect . null = 0 ;
if ( denominator == 0 ) {
tempIntersect . null = 1 ;
return tempIntersect ;
} else {
long double numerator = - ( anchor . x * object . Xc + anchor . y * object . Yc + anchor . z * object . Zc
```

Código

```
long double t = numerator / denominator ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
long double maxA_B = max ( fabs ( object . Xc ) , fabs ( object . Yc ) ) ;
long double maxA_B_C = max ( maxA_B , fabs ( object . Zc ) ) ;
long double u , v ;
if ( maxA_B_C == fabs ( object . Xc ) ) {
u = tempIntersect . Zi ;
v = tempIntersect . Yi ;
} else if ( maxA_B_C == fabs ( object . Yc ) ) {
```

Código

```
u = tempIntersect . Xi ;
v = tempIntersect . Zi ;
} else if ( maxA_B_C == fabs ( object . Zc ) ) {
u = tempIntersect . Xi ;
v = tempIntersect . Yi ;
}
int NC = 0 ;
int NV = object . pointAmount ;
struct Point2D * points2DArrayTemp = malloc ( sizeof ( struct Point2D ) * NV ) ;
for ( int i = 0 ; i < NV ; i ++ ) {
points2DArrayTemp [ i ] . u = object . points2D [ i ] . u - u ;
points2DArrayTemp [ i ] . v = object . points2D [ i ] . v - v ;
}
```

Código

```
int SH = getSign ( points2DArrayTemp [ 0 ] . v ) ;
int NSH ;
int a = 0 ;
int b = ( a + 1 ) % NV ;
for ( a = 0 ; a < NV - 1 ; ) {
NSH = getSign ( points2DArrayTemp [ b ] . v ) ;
if ( SH != NSH ) {
if ( points2DArrayTemp [ a ] . u > 0 && points2DArrayTemp [ b ] . u > 0 ) {
NC ++ ;
} else if ( points2DArrayTemp [ a ] . u > 0 || points2DArrayTemp [ b ] . u > 0 ) {
long double N = ( points2DArrayTemp [ b ] . u - points2DArrayTemp [ a ] . u ) ;
long double D = ( points2DArrayTemp [ b ] . v - points2DArrayTemp [ a ] . v ) ;
if ( D != 0 ) {
```

Código

```
if ( points2DArrayTemp [ a ] . u - ( ( points2DArrayTemp [ a ] . v * N ) / D ) > 0 ) {
NC ++ ;
}
}
}
}
SH = NSH ;
a ++ ;
b ++ ;
}
if ( NC % 2 == 0 ) { tempIntersect . null = 1 ; }
else { tempIntersect . null = 0 ; }
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
```

Código

```
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
free ( points2DArrayTemp ) ;
return tempIntersect ;
}
}
struct Vector polygonNormal ( struct Object object ) {
struct Point3D point0 = object . points3D [ 0 ] ;
struct Point3D point1 = object . points3D [ 1 ] ;
struct Point3D point2 = object . points3D [ 2 ] ;
struct Vector vector1 = { point1 . x - point0 . x , point1 . y - point0 . y , point1 . z - point0 . z } ;
struct Vector vector2 = { point2 . x - point1 . x , point2 . y - point1 . y , point2 . z - point1 . z } ;
```

Código

```
struct Vector normal = crossProduct ( vector1 , vector2 ) ;
return normal ;
}
struct Intersection cilinderIntersection ( struct Vector anchor , struct Vector direction , struct Object
struct Intersection tempIntersect ;
tempIntersect . null = 0 ;
long double xo = object . Xc ;
long double yo = object . Yc ;
long double zo = object . Zc ;
long double xq = object . directionVector . x ;
long double yq = object . directionVector . y ;
long double zq = object . directionVector . z ;
long double xd = direction . x ;
```

Código

```
long double yd = direction . y ;
long double zd = direction . z ;
long double xe = anchor . x ;
long double ye = anchor . y ;
long double ze = anchor . z ;
long double radius = object . other ;
long double xdxq = xd * xq ;
long double ydyq = yd * yq ;
long double zdzq = zd * zq ;
long double xexq = xe * xq ;
long double yeyq = ye * yq ;
long double zezq = ze * zq ;
long double xoxq = xo * xq ;
```

Código

```
long double yoyq = yo * yq ;
long double zozq = zo * zq ;
long double coef1 = xdxq * xq + ydyq * xq + zdzq * xq - xd ;
long double coef2 = xdxq * yq + ydyq * yq + zdzq * yq - yd ;
long double coef3 = xdxq * zq + ydyq * zq + zdzq * zq - zd ;
long double coef4 = xo + xexq * xq - xoxq * xq + yeyq * xq - yoyq * xq + zezq * xq - zozq * xq - xe ;
long double coef5 = yo + xexq * yq - xoxq * yq + yeyq * yq - yoyq * yq + zezq * yq - zozq * yq - ye ;
long double coef6 = zo + xexq * zq - xoxq * zq + yeyq * zq - yoyq * zq + zezq * zq - zozq * zq - ze ;
long double A = pow ( coef1 , 2 ) + pow ( coef2 , 2 ) + pow ( coef3 , 2 ) ;
long double B = 2 * ( coef1 * coef4 + coef2 * coef5 + coef3 * coef6 ) ;
long double C = pow ( coef4 , 2 ) +
pow ( coef5 , 2 ) +
pow ( coef6 , 2 ) -
```

Código

```
pow ( radius , 2 ) ;
long double discriminant = pow ( B , 2 ) - ( 4 * A * C ) ;
long double t , t1 , t2 ;
if ( discriminant >= 0 ) {
long double root = sqrt ( discriminant ) ;
B *= - 1 ;
t1 = ( B - root ) / ( 2 * A ) ;
t2 = ( B + root ) / ( 2 * A ) ;
long double Xi ;
long double Yi ;
long double Zi ;
long double d1 = object . D1 ;
long double d2 = object . D2 ;
```

Código

```
if ( t1 > e ) {
if ( t2 > e ) {
t = min ( t1 , t2 ) ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
tempIntersect . Yi = Yi ;
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
```

Código

```
if ( accept == 0 ) {
t = max ( t1 , t2 ) ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
tempIntersect . Yi = Yi ;
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
```

Código

```
tempIntersect . null = 1 ;
}
return tempIntersect ;
} else {
tempIntersect . null = 1 ;
return tempIntersect ;
}
}
return tempIntersect ;
} else {
t = max ( t1 , t2 ) ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
```

Código

```
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
tempIntersect . Yi = Yi ;
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
} else {
tempIntersect . null = 1 ;
}
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
```

Código

```
}
return tempIntersect ;
}
} else {
t = t1 ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
tempIntersect . Yi = Yi ;
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
```

Código

```
tempIntersect . object = object ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
return tempIntersect ;
} else {
tempIntersect . null = 1 ;
return tempIntersect ;
}
}
} else {
if ( t2 > e ) {
```

Código

```
t = t2 ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
tempIntersect . Yi = Yi ;
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
```

Código

```
}
return tempIntersect ;
} else {
tempIntersect . null = 1 ;
return tempIntersect ;
}
} else {
tempIntersect . null = 1 ;
return tempIntersect ;
}
}
} else {
tempIntersect . null = 1 ;
```

Código

```
return tempIntersect ;
}
}
struct Vector cilinderNormal ( struct Object object , struct Vector intersectionPoint ) {
struct Vector normalCilinder ;
long double x = intersectionPoint . x ;
long double y = intersectionPoint . y ;
long double z = intersectionPoint . z ;
long double xo = object . Xc ;
long double yo = object . Yc ;
long double zo = object . Zc ;
long double xq = object . directionVector . x ;
long double yq = object . directionVector . y ;
```

Código

```
long double zq = object . directionVector . z ;
long double xxoxq = ( x - xo ) * xq ;
long double yyoyq = ( y - yo ) * yq ;
long double zzozq = ( z - zo ) * zq ;
long double parenth = ( xxoxq + yyoyq + zzozq ) ;
long double pxq = 2 * ( xo + parenth * xq - x ) ;
long double pyq = 2 * ( yo + parenth * yq - y ) ;
long double pzq = 2 * ( zo + parenth * zq - z ) ;
normalCilinder . x = pxq * ( pow ( xq , 2 ) - 1 ) +
pyq * ( yq * xq ) +
pzq * ( zq * xq ) ;
normalCilinder . y = pxq * ( xq * yq ) +
pyq * ( pow ( yq , 2 ) - 1 ) +
```

Código

```
pzq * ( zq * yq ) ;
normalCilinder . z = pxq * ( xq * zq ) +
pyq * ( yq * zq ) +
pzq * ( pow ( zq , 2 ) - 1 ) ;
normalCilinder = normalize ( normalCilinder ) ;
return normalCilinder ;
}
struct Intersection coneIntersection ( struct Vector anchor , struct Vector direction , struct Object obje
struct Intersection tempIntersect ;
tempIntersect . null = 0 ;
long double xo = object . Xc ;
long double yo = object . Yc ;
long double zo = object . Zc ;
```

Código

```
long double xq = object . directionVector . x ;
long double yq = object . directionVector . y ;
long double zq = object . directionVector . z ;
long double xd = direction . x ;
long double yd = direction . y ;
long double zd = direction . z ;
long double k1 = object . K1 ;
long double k2 = object . K2 ;
long double xe = anchor . x ;
long double ye = anchor . y ;
long double ze = anchor . z ;
long double xdxq = xd * xq ;
long double ydyq = yd * yq ;
```

Código

```
long double zdzq = zd * zq ;
long double xexq = xe * xq ;
long double yeyq = ye * yq ;
long double zezq = ze * zq ;
long double xoxq = xo * xq ;
long double yoyq = yo * yq ;
long double zozq = zo * zq ;
long double coef1 = xdxq * xq + ydyq * xq + zdzq * xq - xd ;
long double coef2 = xdxq * yq + ydyq * yq + zdzq * yq - yd ;
long double coef3 = xdxq * zq + ydyq * zq + zdzq * zq - zd ;
long double coef4 = xo + xexq * xq - xoxq * xq + yeyq * xq - yoyq * xq + zezq * xq - zozq * xq - xe ;
long double coef5 = yo + xexq * yq - xoxq * yq + yeyq * yq - yoyq * yq + zezq * yq - zozq * yq - ye ;
long double coef6 = zo + xexq * zq - xoxq * zq + yeyq * zq - yoyq * zq + zezq * zq - zozq * zq - ze ;
```

Código

```
long double coefk = pow ( k2 / k1 , 2 ) ;
long double coef7 = xdxq + ydyq + zdzq ;
long double coef8 = xexq - xoxq + yeyq - yoyq + zezq - zozq ;
long double A = pow ( coef1 , 2 ) + pow ( coef2 , 2 ) + pow ( coef3 , 2 ) - ( coefk * pow ( coef7 , 2 ) )
long double B = 2 * ( ( coef1 * coef4 + coef2 * coef5 + coef3 * coef6 ) - ( coefk * coef7 * coef8 ) ) ;
long double C = pow ( coef4 , 2 ) + pow ( coef5 , 2 ) + pow ( coef6 , 2 ) - ( coefk * pow ( coef8 , 2 ) )
long double discriminant = pow ( B , 2 ) - ( 4 * A * C ) ;
long double t , t1 , t2 ;
if ( discriminant >= 0 ) {
long double root = sqrt ( discriminant ) ;
B *= - 1 ;
t1 = ( B + root ) / ( 2 * A ) ;
t2 = ( B - root ) / ( 2 * A ) ;
```

Código

```
long double Xi ;
long double Yi ;
long double Zi ;
long double d1 = object . D1 ;
long double d2 = object . D2 ;
if ( t1 > e ) {
if ( t2 > e ) {
t = min ( t1 , t2 ) ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
```

Código

```
tempIntersect . Yi = Yi ;
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
t = max ( t1 , t2 ) ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
tempIntersect . Yi = Yi ;
```

Código

```
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
return tempIntersect ;
} else {
tempIntersect . null = 1 ;
return tempIntersect ;
}
}
```

Código

```
return tempIntersect ;
} else {
t = max ( t1 , t2 ) ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
tempIntersect . Yi = Yi ;
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
```

Código

```
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
return tempIntersect ;
} else {
tempIntersect . null = 1 ;
return tempIntersect ;
}
}
} else {
t = t1 ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
```

Código

```
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
tempIntersect . Yi = Yi ;
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
return tempIntersect ;
} else {
```

Código

```
tempIntersect . null = 1 ;
return tempIntersect ;
}
}
} else {
if ( t2 > e ) {
t = t2 ;
Xi = xe + ( t * xd ) ;
Yi = ye + ( t * yd ) ;
Zi = ze + ( t * zd ) ;
if ( d2 >= ( ( Xi - xo ) * xq + ( Yi - yo ) * yq + ( Zi - zo ) * zq ) && ( ( Xi - xo ) * xq + ( Yi - yo )
tempIntersect . Xi = Xi ;
tempIntersect . Yi = Yi ;
```

Código

```
tempIntersect . Zi = Zi ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
return tempIntersect ;
} else {
tempIntersect . null = 1 ;
return tempIntersect ;
}
} else {
```

Código

```
tempIntersect . null = 1 ;
return tempIntersect ;
}
}
} else {
tempIntersect . null = 1 ;
return tempIntersect ;
}
}
struct Vector coneNormal ( struct Object object , struct Vector intersectionPoint ) {
struct Vector normalCone ;
long double x = intersectionPoint . x ;
long double y = intersectionPoint . y ;
```

Código

```
long double z = intersectionPoint . z ;
long double xo = object . Xc ;
long double yo = object . Yc ;
long double zo = object . Zc ;
long double xq = object . directionVector . x ;
long double yq = object . directionVector . y ;
long double zq = object . directionVector . z ;
long double k1 = object . K1 ;
long double k2 = object . K2 ;
long double xxoxq = ( x - xo ) * xq ;
long double yyoyq = ( y - yo ) * yq ;
long double zzozq = ( z - zo ) * zq ;
long double parenth = ( xxoxq + yyoyq + zzozq ) ;
```

Código

```
long double pxq = 2 * ( xo + parenth * xq - x ) ;
long double pyq = 2 * ( yo + parenth * yq - y ) ;
long double pzq = 2 * ( zo + parenth * zq - z ) ;
long double k1sq = pow ( k1 , 2 ) ;
long double k2sq2 = 2 * pow ( k2 , 2 ) ;
long double lastFactorDerivedX = ( k2sq2 * xq * parenth ) / k1sq ;
long double lastFactorDerivedY = ( k2sq2 * yq * parenth ) / k1sq ;
long double lastFactorDerivedZ = ( k2sq2 * zq * parenth ) / k1sq ;
normalCone . x = pxq * ( pow ( xq , 2 ) - 1 ) +
pyq * ( yq * xq ) +
pzq * ( zq * xq ) - lastFactorDerivedX ;
normalCone . y = pxq * ( xq * yq ) +
pyq * ( pow ( yq , 2 ) - 1 ) +
```

Código

```
pzq * ( zq * yq ) - lastFactorDerivedY ;
normalCone . z = pxq * ( xq * zq ) +
pyq * ( yq * zq ) +
pzq * ( pow ( zq , 2 ) - 1 ) - lastFactorDerivedZ ;
normalCone = normalize ( normalCone ) ;
return normalCone ;
}
long double whatsTheD ( struct Object object ) {
struct Point3D point = object . points3D [ 0 ] ;
long double theD = - ( ( object . Xc * point . x ) + ( object . Yc * point . y ) + ( object . Zc * point
return theD ;
}
long double whatsTheDGeneral ( struct Vector normalNotNormalized , struct Vector point ) {
```

Código

```
long double theD = - ( ( normalNotNormalized . x * point . x ) + ( normalNotNormalized . y * point . y ) -
return theD ;
}
struct Object getABCD ( struct Object object ) {
struct Vector normal = polygonNormal ( object ) ;
object . Xc = normal . x ;
object . Yc = normal . y ;
object . Zc = normal . z ;
object . other = whatsTheD ( object ) ;
long double L = getNorm ( normal ) ;
object . Xc /= L ;
object . Yc /= L ;
object . Zc /= L ;
```

Código

```
object . other /= L ;
return object ;
}
struct Intersection discIntersection ( struct Vector anchor , struct Vector direction , struct Object obje
long double denominator = ( direction . x * object . directionVector . x ) + ( direction . y * object . di
struct Intersection tempIntersect ;
tempIntersect . null = 1 ;
if ( denominator == 0 ) {
tempIntersect . null = 1 ;
return tempIntersect ;
} else {
long double numerator = - ( ( anchor . x * object . directionVector . x ) + ( anchor . y * object . direct
long double t = numerator / denominator ;
```

Código

```
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
long double distanceToCenter = sqrt ( pow ( tempIntersect . Xi - object . Xc , 2 ) +
pow ( tempIntersect . Yi - object . Yc , 2 ) +
pow ( tempIntersect . Zi - object . Zc , 2 ) ) ;
if ( distanceToCenter < object . other ) {
tempIntersect . null = 0 ;
}
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
```

Código

```
tempIntersect . null = 1 ;
}
return tempIntersect ;
}
}
struct Vector discNormal ( struct Object object ) {
return object . directionVector ;
}
struct Intersection elipseIntersection ( struct Vector anchor , struct Vector direction , struct Object ob
long double denominator = ( direction . x * object . directionVector . x ) + ( direction . y * object . di
struct Intersection tempIntersect ;
tempIntersect . null = 1 ;
if ( denominator == 0 ) {
```

Código

```
tempIntersect . null = 1 ;
return tempIntersect ;
} else {
long double numerator = - ( ( anchor . x * object . directionVector . x ) + ( anchor . y * object . direct
long double t = numerator / denominator ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
long double distanceToD1 = sqrt ( pow ( tempIntersect . Xi - object . Xc , 2 ) +
pow ( tempIntersect . Yi - object . Yc , 2 ) +
pow ( tempIntersect . Zi - object . Zc , 2 ) ) ;
```

Código

```
long double distanceToD2 = sqrt ( pow ( tempIntersect . Xi - object . Xother , 2 ) +
pow ( tempIntersect . Yi - object . Yother , 2 ) +
pow ( tempIntersect . Zi - object . Zother , 2 ) ) ;
if ( ( distanceToD1 + distanceToD2 ) < object . other ) {
tempIntersect . null = 0 ;
}
else {
tempIntersect . null = 1 ;
}
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
```

Código

```
return tempIntersect ;
}
}
struct Vector elipseNormal ( struct Object object ) {
return object . directionVector ;
}
struct Intersection quadraticIntersection ( struct Vector anchor , struct Vector direction , struct Object
long double t , t1 , t2 ;
struct Intersection tempIntersect ;
tempIntersect . null = 0 ;
long double a = ( object . A * pow ( direction . x , 2 ) ) + ( object . B * pow ( direction . y , 2 ) ) +
2 * ( ( object . D * direction . x * direction . y ) * ( object . E * direction . y * direction . z ) * (
long double b = 2 * ( ( object . A * anchor . x * direction . x ) + ( object . B * anchor . y * direction
```

Código

```
+ ( object . D * anchor . x * direction . y ) + ( object . D * anchor . y * direction . x )
+ ( object . E * anchor . y * direction . z ) + ( object . E * anchor . z * direction . y )
+ ( object . F * anchor . z * direction . x ) + ( object . F * anchor . x * direction . z )
+ ( object . G * direction . x ) + ( object . H * direction . y ) + ( object . J * direction . z ) ) ;
long double c = ( object . A * pow ( anchor . x , 2 ) ) + ( object . B * pow ( anchor . y , 2 ) ) + ( obje
+ 2 * ( ( object . D * anchor . x * anchor . y ) + ( object . E * anchor . y * anchor . z ) + ( object . I
+ ( object . G * anchor . x ) + ( object . H * anchor . y ) + ( object . J * anchor . z ) ) + object . otl
long double discriminant = pow ( b , 2 ) - ( 4 * a * c ) ;
if ( discriminant >= 0 ) {
long double root = sqrt ( discriminant ) ;
b *= - 1 ;
t1 = ( b + root ) / ( 2 * a ) ;
t2 = ( b - root ) / ( 2 * a ) ;
```

Código

```
if ( t1 > e ) {
if ( t2 > e ) {
t = min ( t1 , t2 ) ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
t = max ( t1 , t2 ) ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
```

Código

```
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
}
} else {
t = t1 ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
```

Código

```
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
}
} else {
if ( t2 > e ) {
t = t2 ;
tempIntersect . distance = t ;
tempIntersect . object = object ;
tempIntersect . Xi = anchor . x + ( t * direction . x ) ;
```

Código

```
tempIntersect . Yi = anchor . y + ( t * direction . y ) ;
tempIntersect . Zi = anchor . z + ( t * direction . z ) ;
int accept = testIntersection ( tempIntersect . Xi , tempIntersect . Yi , tempIntersect . Zi , object ) ;
if ( accept == 0 ) {
tempIntersect . null = 1 ;
}
} else {
tempIntersect . null = 1 ;
}
}
return tempIntersect ;
} else {
tempIntersect . null = 1 ;
```

Código

```
return tempIntersect ;
}
}
struct Vector quadraticNormal ( struct Object object , struct Vector intersectionVector ) {
long double xElement = ( ( object . A * intersectionVector . x ) + ( object . D * intersectionVector . y )
+ ( object . F * intersectionVector . z ) + ( object . G ) ) ;
long double yElement = ( ( object . D * intersectionVector . x ) + ( object . B * intersectionVector . y )
+ ( object . E * intersectionVector . z ) + ( object . H ) ) ;
long double zElement = ( ( object . F * intersectionVector . x ) + ( object . E * intersectionVector . y )
+ ( object . C * intersectionVector . z ) + ( object . C ) ) ;
struct Vector normalNotNormalized = { xElement , yElement , zElement } ;
return normalNotNormalized ;
}
```

Código

```
long double uRectangle ( struct Vector x0y0z0 , struct Vector x1y1z1 , struct Vector xiyizi ) {
struct Vector U ;
U . x = x1y1z1 . x - x0y0z0 . x ;
U . y = x1y1z1 . y - x0y0z0 . y ;
U . z = x1y1z1 . z - x0y0z0 . z ;
struct Vector i0 ;
i0 . x = xiyizi . x - x0y0z0 . x ;
i0 . y = xiyizi . y - x0y0z0 . y ;
i0 . z = xiyizi . z - x0y0z0 . z ;
long double H = getNorm ( U ) ;
U = normalize ( U ) ;
return pointProduct ( i0 , U ) / H ;
}
```

Código

```
long double vRectangle ( struct Vector x0y0z0 , struct Vector x3y3z3 , struct Vector xiyizi ) {
struct Vector V ;
V . x = x3y3z3 . x - x0y0z0 . x ;
V . y = x3y3z3 . y - x0y0z0 . y ;
V . z = x3y3z3 . z - x0y0z0 . z ;
struct Vector i0 ;
i0 . x = xiyizi . x - x0y0z0 . x ;
i0 . y = xiyizi . y - x0y0z0 . y ;
i0 . z = xiyizi . z - x0y0z0 . z ;
long double L = getNorm ( V ) ;
V = normalize ( V ) ;
return pointProduct ( i0 , V ) / L ;
}
```

Código

```
struct Color planeTexture ( struct Intersection in , struct Vector normal ) {
struct Object object = in . object ;
struct Vector ipoint ;
ipoint . x = in . Xi ;
ipoint . y = in . Yi ;
ipoint . z = in . Zi ;
long double u = uRectangle ( object . x0y0z0 , object . x1y1z1 , ipoint ) ;
long double v = vRectangle ( object . x0y0z0 , object . x3y3z3 , ipoint ) ;
struct Color color ;
for ( int i = 0 ; i < object . numberTextures ; i ++ ) {
int xs = object . textures [ i ] . hRes * u ;
int ys = object . textures [ i ] . vRes * v ;
color = object . textures [ i ] . textureMap [ xs ] [ ys ] ;
```

Código

```
}
return color ;
}
long double uCylinder ( struct Vector anchor , struct Vector Q , struct Vector normal , struct Vector gree
long double tempu = acos ( pointProduct ( normal , greenwich ) ) / ( 2 * 3.14159265 ) ;
struct Vector darkSide = crossProduct ( Q , greenwich ) ;
long double d = whatsTheDGeneral ( darkSide , anchor ) ;
long double test = darkSide . x * xiyizi . x + darkSide . y * xiyizi . y + darkSide . z * xiyizi . z + d
if ( test < 0 ) {
tempu = 1 - tempu ;
}
return tempu ;
}
```

Código

```
long double vCylinder ( struct Vector anchor , struct Vector Q , struct Vector xiyizi , long double den )
struct Vector i0 ;
i0 . x = xiyizi . x - anchor . x ;
i0 . y = xiyizi . y - anchor . y ;
i0 . z = xiyizi . z - anchor . z ;
return pointProduct ( Q , i0 ) / den ;
}
struct Color cylinderTexture ( struct Intersection in , struct Vector normal ) {
struct Object object = in . object ;
struct Vector gw = object . textures [ 0 ] . greenwich ;
struct Vector ipoint ;
ipoint . x = in . Xi ;
ipoint . y = in . Yi ;
```

Código

```
ipoint . z = in . Zi ;
struct Vector anchor ;
anchor . x = object . Xc ;
anchor . y = object . Yc ;
anchor . z = object . Zc ;
long double u = uCylinder ( anchor , object . directionVector , normal , gw , ipoint ) ;
long double v = vCylinder ( anchor , object . directionVector , ipoint , object . D2 - object . D1 ) ;
struct Color color ;
for ( int i = 0 ; i < object . numberTextures ; i ++ ) {
int xs = object . textures [ i ] . hRes * u ;
int ys = object . textures [ i ] . vRes * v ;
color = object . textures [ i ] . textureMap [ xs ] [ ys ] ;
}
```

Código

```
return color ;
}
long double uSphere ( struct Vector center , struct Vector north , long double radius , struct Vector xiyi
struct Vector ic ;
ic . x = xiyizi . x - center . x ;
ic . y = xiyizi . y - center . y ;
ic . z = xiyizi . z - center . z ;
long double icnorth = pointProduct ( north , ic ) ;
struct Vector inprime ;
inprime . x = xiyizi . x - north . x * icnorth ;
inprime . y = xiyizi . y - north . y * icnorth ;
inprime . z = xiyizi . z - north . z * icnorth ;
struct Vector nprime ;
```

Código

```
nprime . x = inprime . x - center . x ;
nprime . y = inprime . y - center . y ;
nprime . z = inprime . z - center . z ;
nprime = normalize ( nprime ) ;
long double tempu = acos ( pointProduct ( nprime , greenwich ) ) / ( 2 * 3.14159265 ) ;
struct Vector darkSide = crossProduct ( north , greenwich ) ;
long double d = whatsTheDGeneral ( darkSide , center ) ;
long double test = darkSide . x * xiyizi . x + darkSide . y * xiyizi . y + darkSide . z * xiyizi . z + d ;
if ( test < 0 ) {
tempu = 1 - tempu ;
}
return tempu ;
}
```

Código

```
long double vSphere ( struct Vector center , struct Vector north , long double radius , struct Vector xiyi
struct Vector south ;
south . x = center . x - radius * north . x ;
south . y = center . y - radius * north . y ;
south . z = center . z - radius * north . z ;
struct Vector i0 ;
i0 . x = xiyizi . x - south . x ;
i0 . y = xiyizi . y - south . y ;
i0 . z = xiyizi . z - south . z ;
return pointProduct ( north , i0 ) / ( 2 * radius ) ;
}
struct Color sphereTexture ( struct Intersection in , struct Vector normal ) {
struct Object object = in . object ;
```

Código

```
struct Vector gw = object . textures [ 0 ] . greenwich ;
struct Vector north = object . textures [ 0 ] . north ;
struct Vector ipoint ;
ipoint . x = in . Xi ;
ipoint . y = in . Yi ;
ipoint . z = in . Zi ;
struct Vector center ;
center . x = object . Xc ;
center . y = object . Yc ;
center . z = object . Zc ;
long double u = uSphere ( center , north , object . other , ipoint , gw ) ;
long double v = vSphere ( center , north , object . other , ipoint ) ;
struct Color color ;
```

Código

```
for ( int i = 0 ; i < object . numberTextures ; i ++ ) {
int xs = object . textures [ i ] . hRes * u ;
int ys = object . textures [ i ] . vRes * v ;
color = object . textures [ i ] . textureMap [ xs ] [ ys ] ;
}
return color ;
}
long double uCone ( struct Vector anchor , struct Vector Q , struct Vector normal , struct Vector greenwi
struct Vector aux = crossProduct ( normal , Q ) ;
struct Vector nprime = crossProduct ( aux , Q ) ;
long double tempu = acos ( pointProduct ( nprime , greenwich ) ) / ( 2 * 3.14159265 ) ;
struct Vector darkSide = crossProduct ( Q , greenwich ) ;
long double d = whatsTheDGeneral ( darkSide , anchor ) ;
```

Código

```
long double test = darkSide . x * xiyizi . x + darkSide . y * xiyizi . y + darkSide . z * xiyizi . z + d
if ( test < 0 ) {
tempu = 1 - tempu ;
}
return tempu ;
}
struct Color coneTexture ( struct Intersection in , struct Vector normal ) {
struct Object object = in . object ;
struct Vector gw = object . textures [ 0 ] . greenwich ;
struct Vector ipoint ;
ipoint . x = in . Xi ;
ipoint . y = in . Yi ;
ipoint . z = in . Zi ;
```

Código

```
struct Vector anchor ;
anchor . x = object . Xc ;
anchor . y = object . Yc ;
anchor . z = object . Zc ;
long double u = uCone ( anchor , object . directionVector , normal , gw , ipoint ) ;
long double v = vCylinder ( anchor , object . directionVector , ipoint , object . D2 - object . D1 ) ;
struct Color color ;
for ( int i = 0 ; i < object . numberTextures ; i ++ ) {
int xs = object . textures [ i ] . hRes * u ;
int ys = object . textures [ i ] . vRes * v ;
color = object . textures [ i ] . textureMap [ xs ] [ ys ] ;
}
return color ;
```

Código

```
}
struct Intersection getFirstIntersection ( struct Vector anchor , struct Vector direction ) {
rays += 1 ;
int k ;
int objectsAmount = numberObjects ;
long double tmin ;
struct Intersection intersection ;
struct Intersection tempIntersection ;
intersection . null = 1 ;
tmin = 10000000 ;
tempIntersection . null = 1 ;
for ( k = 0 ; k < objectsAmount ; k ++ ) {
tempIntersection = Objects [ k ] . intersectionFuncion ( anchor , direction , Objects [ k ] ) ;
```

Código

```
if ( tempIntersection . null != 1 && tempIntersection . distance > e && tempIntersection . distance < tmin
tmin = tempIntersection . distance ;
intersection = tempIntersection ;
}
tempIntersection . null = 1 ;
}
return intersection ;
}
struct Color ponderColor ( struct Color baseColor , struct Color reflectionColor , long double o1 , long d
struct Color color ;
color . r = baseColor . r * o1 + reflectionColor . r * o2 ;
color . g = baseColor . g * o1 + reflectionColor . g * o2 ;
color . b = baseColor . b * o1 + reflectionColor . b * o2 ;
```

Código

```
return color ;
}
struct Color getColor ( struct Vector anchor , struct Vector direction , struct Vector V , int rLevel ) {
struct Color color ;
struct Intersection intersection ;
struct Intersection * tempIntersection ;
intersection = getFirstIntersection ( anchor , direction ) ;
if ( intersection . null == 1 ) {
color = background ;
} else {
int k ;
int lightsAmount = numberLights ;
struct Object Q = intersection . object ;
```

Código

```
struct Vector L ;
struct Vector intersectVector = { intersection . Xi , intersection . Yi , intersection . Zi } ;
struct Vector N = normalize ( Q . normalVector ( Q , intersectVector ) ) ;
struct Vector R ;
if ( pointProduct ( N , direction ) > 0 ) {
N . x *= - 1 ;
N . y *= - 1 ;
N . z *= - 1 ;
}
long double Fatt ;
long double I = 0.0 ;
long double E = 0.0 ;
for ( k = 0 ; k < numberLights ; k ++ ) {
```

Código

```
struct Intersection obstacle ;
struct Vector light = { Lights [ k ] . Xp - intersection . Xi , Lights [ k ] . Yp - intersection . Yi , Li
L = light ;
L = normalize ( light ) ;
long double pp = pointProduct ( N , L ) ;
obstacle = getFirstIntersection ( intersectVector , L ) ;
long double distanceToLight = getNorm ( light ) ;
if ( obstacle . null == 1 || ( obstacle . distance > e && obstacle . distance > distanceToLight ) ) {
Fatt = getAttenuationFactor ( Lights [ k ] , distanceToLight ) ;
if ( pp > 0.0 ) {
R . x = ( 2 * N . x * pp ) - L . x ;
R . y = ( 2 * N . y * pp ) - L . y ;
R . z = ( 2 * N . z * pp ) - L . z ;
```

Código

```
R = normalize ( R ) ;
I = I + ( pp * Q . Kd * Fatt * Lights [ k ] . Ip ) ;
}
long double pp2 = pointProduct ( R , V ) ;
if ( pp2 > 0.0 ) {
E = E + ( pow ( pp2 , Q . Kn ) * Q . Ks * Lights [ k ] . Ip * Fatt ) ;
}
}
}
if ( Q . numberTextures != 0 ) { color = Q . retrieveTextureColor ( intersection , N ) ; }
else { color = Q . color ; }
if ( isnan ( color . r ) ) { color = Q . color ; }
I = I + Ia * Q . Ka ;
```

Código

```
I = min ( 1.0 , I ) ;
color = difusseColor ( I , color ) ;
E = min ( 1.0 , E ) ;
color = specularHighlight ( E , color ) ;
if ( rLevel > 0 ) {
long double pNV = pointProduct ( N , V ) ;
R . x = ( 2 * N . x * pNV ) - V . x ;
R . y = ( 2 * N . y * pNV ) - V . y ;
R . z = ( 2 * N . z * pNV ) - V . z ;
R = normalize ( R ) ;
struct Vector otherV = { - R . x , - R . y , - R . z } ;
struct Color reflectionColor = getColor ( intersectVector , R , otherV , rLevel - 1 ) ;
color = ponderColor ( color , reflectionColor , Q . o1 , Q . o2 ) ;
```

Código

```
}
struct Color transparencyColor = background ;
int levelsAllowed = maxTransparency ;
while ( levelsAllowed > 0 && intersection . object . o3 > 0 ) {
transparencyColor = getColor ( intersectVector , direction , V , maxReflection ) ;
levelsAllowed -- ;
if ( transparencyColor . r == background . r && transparencyColor . g == background . g && transparencyCol
break ;
}
}
color = ponderColor ( color , transparencyColor , 1 , intersection . object . o3 ) ;
}
return ( color ) ;
```

Código

```
}
void printPlaneCuts ( struct Object objeto ) {
if ( objeto . planeCuts == NULL ) {
printf ( "\n Este objeto no tiene planos de corte asociados. \n" ) ;
return ;
} else {
int i = 0 ;
struct Vector normal ;
struct Vector punto ;
for ( i = 0 ; i < objeto . numberPlaneCuts ; i ++ ) {
printf ( "Plano %i: \n" , i ) ;
normal = objeto . planeCuts [ i ] . normal ;
punto = objeto . planeCuts [ i ] . point ;
```

Código

```
printf ( "\t Normal unitaria: %LF, %LF, %LF \n" , normal . x , normal . y , normal . z ) ;
printf ( "\t Punto base: %LF, %LF, %LF \n\n" , punto . x , punto . y , punto . z ) ;
}
}
}
void printTextures ( struct Object objeto , int currentTypeObjectReading ) {
if ( objeto . textures == NULL ) {
printf ( "\n Este objeto no tiene texturas asociados. \n" ) ;
return ;
} else {
int i = 0 ;
struct Vector norte ;
struct Vector greenwich ;
```

Código

```
for ( i = 0 ; i < objeto . numberTextures ; i ++ ) {
printf ( "Textura %i: \n" , i ) ;
printf ( "Resolucin Textura: %ix%i \n" , objeto . textures [ i ] . hRes , objeto . textures [ i ] . vRes )
printf ( "Primer texel de la textura: (%LF, %LF, %LF)" , objeto . textures [ i ] . textureMap [ 0 ] [ 0 ]
int lastX = objeto . textures [ i ] . hRes - 1 ;
int lastY = objeto . textures [ i ] . vRes - 1 ;
printf ( "ltimo texel de la textura: (%LF, %LF, %LF)" , objeto . textures [ i ] . textureMap [ lastX ] [ l
for ( int f = 0 ; f < objeto . textures [ i ] . hRes ; f ++ ) {
for ( int j = 0 ; j < objeto . textures [ i ] . vRes ; j ++ ) {
printf ( "Texel [%i][%i]de la textura: (%LF, %LF, %LF) \n" , f , j , objeto . textures [ i ] . textureMap
}
}
if ( currentTypeObjectReading == 2 || currentTypeObjectReading == 4 || currentTypeObjectReading == 5 || cu
```

Código

```
greenwich = objeto . textures [ i ] . greenwich ;
printf ( "\t Greenwich unitario: %LF, %LF, %LF \n" , greenwich . x , greenwich . y , greenwich . z ) ;
if ( currentTypeObjectReading == 2 || currentTypeObjectReading == 8 ) {
norte = objeto . textures [ i ] . north ;
printf ( "\t Norte unitario: %LF, %LF, %LF \n" , norte . x , norte . y , norte . z ) ;
}
}
}
}
}
void printDraftPlanes ( struct Object objeto , int currentTypeObjectReading ) {
if ( objeto . draftPlanes == NULL ) {
printf ( "\n Este objeto no tiene planos de calado  asociados. \n" ) ;
```

Código

```
return ;
} else {
int i = 0 ;
struct Vector norte ;
struct Vector greenwich ;
for ( i = 0 ; i < objeto . numberDraftPlanes ; i ++ ) {
printf ( "Nmero de planos de calado: %i \n" , objeto . numberDraftPlanes ) ;
printf ( "plano de calado %i: \n" , i ) ;
printf ( "Resolucin plano de calado: %ix%i \n" , objeto . draftPlanes [ i ] . hRes , objeto . draftPlanes
printf ( "Primer texel del plano de calado: (%LF, %LF, %LF)" , objeto . draftPlanes [ i ] . textureMap [ (
int lastX = objeto . draftPlanes [ i ] . hRes - 1 ;
int lastY = objeto . draftPlanes [ i ] . vRes - 1 ;
printf ( "ltimo texel del plano de calado: (%LF, %LF, %LF)" , objeto . draftPlanes [ i ] . textureMap [ la
```

Código

```
}
if ( currentTypeObjectReading == 2 || currentTypeObjectReading == 4 || currentTypeObjectReading == 5 || cu
greenwich = objeto . draftPlanes [ i ] . greenwich ;
printf ( "\t Greenwich unitario: %LF, %LF, %LF \n" , greenwich . x , greenwich . y , greenwich . z ) ;
if ( currentTypeObjectReading == 2 || currentTypeObjectReading == 8 ) {
norte = objeto . draftPlanes [ i ] . north ;
printf ( "\t Norte unitario: %LF, %LF, %LF \n" , norte . x , norte . y , norte . z ) ;
}
}
}
}
void createObjectFromData ( long double * data , int whichObjectCreate , int quantityData , struct PlaneCu
switch ( whichObjectCreate ) {
```

Código

```
case 0 : {
if ( debug == 1 ) {
printf ( "Insertando datos de escena \n" ) ;
printf ( "Reflexiones: %LF, Transparencia: %LF, Anti-aliasing: %LF \n" , data [ 0 ] , data [ 1 ] , data [
printf ( "Iluminacin ambiente: %LF \n" , data [ 3 ] ) ;
printf ( "Plano de proyeccin (Xmin, Ymin) (Xmax, Ymax) : (%LF, %LF) (%LF, %LF) \n" , data [ 4 ] , data [ 5
printf ( "Resolucin:  %LFx%LF \n" , data [ 8 ] , data [ 9 ] ) ;
printf ( "Epsilon %LF \n" , data [ 10 ] ) ;
printf ( "Ojo: (%LF, %LF, %LF) \n" , data [ 11 ] , data [ 12 ] , data [ 13 ] ) ;
printf ( "Color background: (%LF, %LF, %LF) \n" , data [ 14 ] , data [ 15 ] , data [ 16 ] ) ;
}
maxAA = data [ 0 ] ;
maxReflection = data [ 1 ] ;
```

Código

```
maxTransparency = data [ 2 ] ;
Ia = data [ 3 ] ;
Xmin = data [ 4 ] ;
Ymin = data [ 5 ] ;
Xmax = data [ 6 ] ;
Ymax = data [ 7 ] ;
Hres = data [ 8 ] ;
Vres = data [ 9 ] ;
e = data [ 10 ] ;
eye . x = data [ 11 ] ;
eye . y = data [ 12 ] ;
eye . z = data [ 13 ] ;
background . r = data [ 14 ] ;
```

Código

```
background . g = data [ 15 ] ;
background . b = data [ 16 ] ;
Framebuffer [ Hres ] [ Vres ] ;
Framebuffer = ( struct Color * * ) malloc ( Vres * sizeof ( struct Color * ) ) ;
for ( int i = 0 ; i < Vres ; i ++ ) {
Framebuffer [ i ] = ( struct Color * ) malloc ( Hres * sizeof ( struct Color ) ) ;
}
return ;
}
case 1 : {
if ( debug == 1 ) {
printf ( "Insertando Luz...\n" ) ;
printf ( "Pos luz (%LF, %LF, %LF) \n" , data [ 0 ] , data [ 1 ] , data [ 2 ] ) ;
```

Código

```
printf ( "c1: %LF, c2: %LF, c3 %LF \n" , data [ 3 ] , data [ 4 ] , data [ 5 ] ) ;
printf ( "Ip luz: %LF \n" , data [ 6 ] ) ;
}
struct Object polygon ;
struct Color colorPolygon ;
struct Light luz ;
luz . Xp = data [ 0 ] ;
luz . Yp = data [ 1 ] ;
luz . Zp = data [ 2 ] ;
luz . c1 = data [ 3 ] ;
luz . c2 = data [ 4 ] ;
luz . c3 = data [ 5 ] ;
luz . Ip = data [ 6 ] ;
```

Código

```
Lights [ lightIndex ] = luz ;
lightIndex ++ ;
return ;
}
case 2 : {
if ( debug == 1 ) {
printf ( "Insertando Esfera..." ) ;
printf ( "Pos esfera (%LF, %LF, %LF) \n" , data [ 0 ] , data [ 1 ] , data [ 2 ] ) ;
printf ( "o1:  %LF, o2: %LF, o3: %LF \n" , data [ 3 ] , data [ 4 ] , data [ 5 ] ) ;
printf ( "Radio esfera: %LF \n" , data [ 6 ] ) ;
printf ( "Esfera Kd: %LF \n" , data [ 6 ] ) ;
printf ( "Esfera Ka: %LF \n" , data [ 8 ] ) ;
printf ( "Esfera Kn: %LF \n" , data [ 9 ] ) ;
```

Código

```
printf ( "Esfera Ks: %LF \n" , data [ 10 ] ) ;
printf ( "Color esfera (%LF, %LF, %LF) \n" , data [ 11 ] , data [ 12 ] , data [ 13 ] ) ;
}
struct Object polygon ;
struct Color colorPolygon ;
struct Object esfera ;
esfera . Xc = data [ 0 ] ;
esfera . Yc = data [ 1 ] ;
esfera . Zc = data [ 2 ] ;
esfera . o1 = data [ 3 ] ;
esfera . o2 = data [ 4 ] ;
esfera . o3 = data [ 5 ] ;
esfera . other = data [ 6 ] ;
```

Código

```
esfera . Kd = data [ 7 ] ;
esfera . Ka = data [ 8 ] ;
esfera . Kn = data [ 9 ] ;
esfera . Ks = data [ 10 ] ;
esfera . normalVector = sphereNormal ;
esfera . intersectionFuncion = sphereIntersection ;
esfera . retrieveTextureColor = sphereTexture ;
struct Color colorSphere ;
colorSphere . r = data [ 11 ] ;
colorSphere . g = data [ 12 ] ;
colorSphere . b = data [ 13 ] ;
esfera . color = colorSphere ;
esfera . planeCuts = planeCutsFound ;
```

Código

```
esfera . numberPlaneCuts = numberPlaneCuts ;
esfera . textures = texturesFound ;
esfera . numberTextures = numberTextures ;
Objects [ objectIndex ] = esfera ;
if ( debug == 1 ) {
printPlaneCuts ( Objects [ objectIndex ] ) ;
printTextures ( Objects [ objectIndex ] , whichObjectCreate ) ;
}
objectIndex ++ ;
return ;
}
case 3 : {
int vertexPolygonIndex = 0 ;
```

Código

```
int numVertexesPolygon = ( quantityData - 10 - 12 ) / 3 + 1 ;
int inicioPlano = 10 + ( numVertexesPolygon - 1 ) * 3 ;
if ( debug == 1 ) {
printf ( "quantityData: %i \n" , quantityData ) ;
printf ( "numVertexesPolygon: %i \n" , numVertexesPolygon ) ;
printf ( "%i inicioPlano \n" , inicioPlano ) ;
printf ( "Insertando polgono..." ) ;
printf ( "Color polgono (%LF, %LF, %LF) \n" , data [ 0 ] , data [ 1 ] , data [ 2 ] ) ;
printf ( "o1:  %LF, o2: %LF, o3: %LF \n" , data [ 3 ] , data [ 4 ] , data [ 5 ] ) ;
printf ( "Poligono Kd: %LF \n" , data [ 6 ] ) ;
printf ( "Poligono Ka: %LF \n" , data [ 7 ] ) ;
printf ( "Poligono Kn: %LF \n" , data [ 8 ] ) ;
printf ( "Poligono Ks: %LF \n" , data [ 9 ] ) ;
```

Código

```
printf ( "Esquina inferior izquierda (%LF, %LF, %LF) \n" , data [ inicioPlano ] , data [ inicioPlano + 1 ]
printf ( "Esquina inferior derecha (%LF, %LF, %LF) \n" , data [ inicioPlano + 3 ] , data [ inicioPlano + 4
printf ( "Esquina superior derecha (%LF, %LF, %LF) \n" , data [ inicioPlano + 6 ] , data [ inicioPlano + 7
printf ( "Esquina superior izquierda (%LF, %LF, %LF) \n" , data [ inicioPlano + 9 ] , data [ inicioPlano +
}
struct Point3D vertex ;
struct Point2D squashedVertex ;
struct Object temp ;
struct Vector x0y0z0 ;
x0y0z0 . x = data [ inicioPlano ] ;
x0y0z0 . y = data [ inicioPlano + 1 ] ;
x0y0z0 . z = data [ inicioPlano + 2 ] ;
struct Vector x1y1z1 ;
```

Código

```
x1y1z1 . x = data [ inicioPlano + 3 ] ;
x1y1z1 . y = data [ inicioPlano + 4 ] ;
x1y1z1 . z = data [ inicioPlano + 5 ] ;
struct Vector x2y2z2 ;
x2y2z2 . x = data [ inicioPlano + 6 ] ;
x2y2z2 . y = data [ inicioPlano + 7 ] ;
x2y2z2 . z = data [ inicioPlano + 8 ] ;
struct Vector x3y3z3 ;
x3y3z3 . x = data [ inicioPlano + 9 ] ;
x3y3z3 . y = data [ inicioPlano + 10 ] ;
x3y3z3 . z = data [ inicioPlano + 11 ] ;
temp . points3D = malloc ( sizeof ( struct Point3D ) * 3 ) ;
for ( int i = 0 ; i + 10 < quantityData - 12 ; ) {
```

Código

```
if ( vertexPolygonIndex == 3 ) {
break ;
}
vertex . x = data [ 10 + i ] ;
i ++ ;
vertex . y = data [ 10 + i ] ;
i ++ ;
vertex . z = data [ 10 + i ] ;
i ++ ;
temp . points3D [ vertexPolygonIndex ] = vertex ;
vertexPolygonIndex ++ ;
}
struct Object polygon ;
```

Código

```
vertexPolygonIndex = 0 ;
polygon = getABCD ( temp ) ;
if ( debug == 1 ) {
printf ( "A del poligono %LF\n" , polygon . Xc ) ;
printf ( "B del poligono %LF\n" , polygon . Yc ) ;
printf ( "C del poligono %LF\n" , polygon . Zc ) ;
printf ( "D del poligono %LF\n" , polygon . other ) ;
}
polygon . points3D = malloc ( sizeof ( struct Point3D ) * numVertexesPolygon ) ;
polygon . points2D = malloc ( sizeof ( struct Point2D ) * numVertexesPolygon ) ;
struct Color colorPolygon ;
colorPolygon . r = data [ 0 ] ;
colorPolygon . g = data [ 1 ] ;
```

Código

```
colorPolygon . b = data [ 2 ] ;
polygon . color = colorPolygon ;
polygon . o1 = data [ 3 ] ;
polygon . o2 = data [ 4 ] ;
polygon . o3 = data [ 5 ] ;
polygon . Kd = data [ 6 ] ;
polygon . Ka = data [ 7 ] ;
polygon . Kn = data [ 8 ] ;
polygon . Ks = data [ 9 ] ;
polygon . pointAmount = numVertexesPolygon ;
polygon . normalVector = polygonNormal ;
polygon . intersectionFuncion = polygonIntersection ;
polygon . retrieveTextureColor = planeTexture ;
```

Código

```
long double u ;
long double v ;
long double maxA_B = max ( fabs ( polygon . Xc ) , fabs ( polygon . Yc ) ) ;
long double maxA_B_C = max ( maxA_B , fabs ( polygon . Zc ) ) ;
int choice = 0 ;
if ( maxA_B_C == fabs ( polygon . Xc ) ) { choice = 0 ; }
else if ( maxA_B_C == fabs ( polygon . Yc ) ) { choice = 1 ; }
else if ( maxA_B_C == fabs ( polygon . Zc ) ) { choice = 2 ; }
for ( int i = 0 ; i + 10 < quantityData - 12 ; ) {
vertex . x = data [ 10 + i ] ;
i ++ ;
vertex . y = data [ 10 + i ] ;
i ++ ;
```

Código

```
vertex . z = data [ 10 + i ] ;
i ++ ;
if ( debug == 1 ) {
printf ( "Vertice: (%LF,%LF,%LF) \n" , vertex . x , vertex . y , vertex . z ) ;
}
if ( choice == 0 ) { u = vertex . z ; v = vertex . y ; }
else if ( choice == 1 ) { u = vertex . x ; v = vertex . z ; }
else if ( choice == 2 ) { u = vertex . x ; v = vertex . y ; }
squashedVertex . u = u ;
squashedVertex . v = v ;
polygon . points3D [ vertexPolygonIndex ] = vertex ;
polygon . points2D [ vertexPolygonIndex ] = squashedVertex ;
vertexPolygonIndex ++ ;
```

Código

```
}
vertex . x = data [ 10 ] ;
vertex . y = data [ 11 ] ;
vertex . z = data [ 12 ] ;
if ( choice == 0 ) { u = vertex . z ; v = vertex . y ; }
else if ( choice == 1 ) { u = vertex . x ; v = vertex . z ; }
else if ( choice == 2 ) { u = vertex . x ; v = vertex . y ; }
squashedVertex . u = u ;
squashedVertex . v = v ;
polygon . points3D [ vertexPolygonIndex ] = vertex ;
polygon . points2D [ vertexPolygonIndex ] = squashedVertex ;
polygon . planeCuts = planeCutsFound ;
polygon . numberPlaneCuts = numberPlaneCuts ;
```

Código

```
polygon . textures = texturesFound ;
polygon . numberTextures = numberTextures ;
polygon . x0y0z0 = x0y0z0 ;
polygon . x1y1z1 = x1y1z1 ;
polygon . x2y2z2 = x2y2z2 ;
polygon . x3y3z3 = x3y3z3 ;
Objects [ objectIndex ] = polygon ;
if ( debug == 1 ) {
printPlaneCuts ( Objects [ objectIndex ] ) ;
printTextures ( Objects [ objectIndex ] , whichObjectCreate ) ;
}
objectIndex ++ ;
return ;
```

Código

```
}
case 4 : {
if ( debug == 1 ) {
printf ( "Insertando cilindro..." ) ;
printf ( "Ancla: (%LF, %LF, %LF) \n" , data [ 0 ] , data [ 1 ] , data [ 2 ] ) ;
printf ( "Vector: (%LF, %LF, %LF) \n" , data [ 3 ] , data [ 4 ] , data [ 5 ] ) ;
printf ( "o1:  %LF, o2: %LF, o3: %LF \n" , data [ 6 ] , data [ 7 ] , data [ 8 ] ) ;
printf ( "Cilindro Radio: %LF \n" , data [ 9 ] ) ;
printf ( "Cilindro d1: %LF Cilindro d2: %LF \n" , data [ 10 ] , data [ 11 ] ) ;
printf ( "Cilindro Kd: %LF \n" , data [ 12 ] ) ;
printf ( "Cilindro Ka: %LF \n" , data [ 13 ] ) ;
printf ( "Cilindro Kn: %LF \n" , data [ 14 ] ) ;
printf ( "Cilindro Ks: %LF \n" , data [ 15 ] ) ;
```

Código

```
printf ( "RGB Cilindro: (%LF, %LF, %LF) \n" , data [ 16 ] , data [ 17 ] , data [ 18 ] ) ;
}
struct Object cilinder ;
cilinder . Xc = data [ 0 ] ;
cilinder . Yc = data [ 1 ] ;
cilinder . Zc = data [ 2 ] ;
struct Vector cilinderVector ;
cilinderVector . x = data [ 3 ] ;
cilinderVector . y = data [ 4 ] ;
cilinderVector . z = data [ 5 ] ;
cilinderVector = normalize ( cilinderVector ) ;
cilinder . directionVector = cilinderVector ;
cilinder . o1 = data [ 6 ] ;
```

Código

```
cilinder . o2 = data [ 7 ] ;
cilinder . o3 = data [ 8 ] ;
cilinder . other = data [ 9 ] ;
cilinder . D1 = data [ 10 ] ;
cilinder . D2 = data [ 11 ] ;
cilinder . Kd = data [ 12 ] ;
cilinder . Ka = data [ 13 ] ;
cilinder . Kn = data [ 14 ] ;
cilinder . Ks = data [ 15 ] ;
cilinder . height = cilinder . D2 - cilinder . D1 ;
cilinder . normalVector = cilinderNormal ;
cilinder . intersectionFuncion = cilinderIntersection ;
cilinder . retrieveTextureColor = cylinderTexture ;
```

Código

```
struct Color cilinderColor ;
cilinderColor . r = data [ 16 ] ;
cilinderColor . g = data [ 17 ] ;
cilinderColor . b = data [ 18 ] ;
cilinder . color = cilinderColor ;
cilinder . planeCuts = planeCutsFound ;
cilinder . numberPlaneCuts = numberPlaneCuts ;
cilinder . textures = texturesFound ;
cilinder . numberTextures = numberTextures ;
Objects [ objectIndex ] = cilinder ;
if ( debug == 1 ) {
printPlaneCuts ( Objects [ objectIndex ] ) ;
printTextures ( Objects [ objectIndex ] , whichObjectCreate ) ;
```

Código

```
}
objectIndex ++ ;
return ;
}
case 5 : {
if ( debug == 1 ) {
printf ( "Insertando cono..." ) ;
printf ( "Ancla: (%LF, %LF, %LF) \n" , data [ 0 ] , data [ 1 ] , data [ 2 ] ) ;
printf ( "Vector: (%LF, %LF, %LF) \n" , data [ 3 ] , data [ 4 ] , data [ 5 ] ) ;
printf ( "o1:  %LF, o2: %LF, o3: %LF \n" , data [ 6 ] , data [ 7 ] , data [ 8 ] ) ;
printf ( "Cono k1: %LF COno k2: %LF \n" , data [ 9 ] , data [ 10 ] ) ;
printf ( "Cono d1: %LF COno d2: %LF \n" , data [ 11 ] , data [ 12 ] ) ;
printf ( "Cono Kd: %LF \n" , data [ 13 ] ) ;
```

Código

```
printf ( "Cono Ka: %LF \n" , data [ 14 ] ) ;
printf ( "Cono Kn: %LF \n" , data [ 15 ] ) ;
printf ( "Cono Ks: %LF \n" , data [ 16 ] ) ;
printf ( "RGB Cono: (%LF, %LF, %LF) \n" , data [ 17 ] , data [ 18 ] , data [ 19 ] ) ;
}
struct Object cone ;
cone . Xc = data [ 0 ] ;
cone . Yc = data [ 1 ] ;
cone . Zc = data [ 2 ] ;
struct Vector coneVector ;
coneVector . x = data [ 3 ] ;
coneVector . y = data [ 4 ] ;
coneVector . z = data [ 5 ] ;
```

Código

```
coneVector = normalize ( coneVector ) ;
cone . directionVector = coneVector ;
cone . o1 = data [ 6 ] ;
cone . o2 = data [ 7 ] ;
cone . o3 = data [ 8 ] ;
cone . K1 = data [ 9 ] ;
cone . K2 = data [ 10 ] ;
cone . D1 = data [ 11 ] ;
cone . D2 = data [ 12 ] ;
cone . height = cone . D2 - cone . D1 ;
cone . Kd = data [ 13 ] ;
cone . Ka = data [ 14 ] ;
cone . Kn = data [ 15 ] ;
```

Código

```
cone . Ks = data [ 16 ] ;
cone . intersectionFuncion = coneIntersection ;
cone . retrieveTextureColor = coneTexture ;
cone . normalVector = coneNormal ;
struct Color coneColor ;
coneColor . r = data [ 17 ] ;
coneColor . g = data [ 18 ] ;
coneColor . b = data [ 19 ] ;
cone . color = coneColor ;
cone . planeCuts = planeCutsFound ;
cone . numberPlaneCuts = numberPlaneCuts ;
cone . textures = texturesFound ;
cone . numberTextures = numberTextures ;
```

Código

```
Objects [ objectIndex ] = cone ;
if ( debug == 1 ) {
printPlaneCuts ( Objects [ objectIndex ] ) ;
printTextures ( Objects [ objectIndex ] , whichObjectCreate ) ;
}
objectIndex ++ ;
return ;
}
case 6 : {
if ( debug == 1 ) {
printf ( "Insertando disco..." ) ;
printf ( "Punto Central: (%LF, %LF, %LF) \n" , data [ 0 ] , data [ 1 ] , data [ 2 ] ) ;
printf ( "Normal: (%LF, %LF, %LF) \n" , data [ 3 ] , data [ 4 ] , data [ 5 ] ) ;
```

Código

```
printf ( "Color: (%LF, %LF, %LF) \n" , data [ 6 ] , data [ 7 ] , data [ 8 ] ) ;
printf ( "Disco Radio: %LF \n" , data [ 9 ] ) ;
printf ( "o1:  %LF, o2: %LF, o3: %LF \n" , data [ 10 ] , data [ 11 ] , data [ 12 ] ) ;
printf ( "DIsco Kd: %LF \n" , data [ 13 ] ) ;
printf ( "DIsco Ka: %LF \n" , data [ 14 ] ) ;
printf ( "DIsco Kn: %LF \n" , data [ 15 ] ) ;
printf ( "Disco Ks: %LF \n" , data [ 16 ] ) ;
printf ( "Esquina inferior izquierda (%LF, %LF, %LF) \n" , data [ 17 ] , data [ 18 ] , data [ 19 ] ) ;
printf ( "Esquina inferior derecha (%LF, %LF, %LF) \n" , data [ 20 ] , data [ 21 ] , data [ 22 ] ) ;
printf ( "Esquina superior derecha (%LF, %LF, %LF) \n" , data [ 23 ] , data [ 24 ] , data [ 25 ] ) ;
printf ( "Esquina superior izquierda (%LF, %LF, %LF) \n" , data [ 26 ] , data [ 27 ] , data [ 28 ] ) ;
}
struct Object disco ;
```

Código

```
struct Vector x0y0z0 ;
x0y0z0 . x = data [ 17 ] ;
x0y0z0 . y = data [ 18 ] ;
x0y0z0 . z = data [ 19 ] ;
struct Vector x1y1z1 ;
x1y1z1 . x = data [ 20 ] ;
x1y1z1 . y = data [ 21 ] ;
x1y1z1 . z = data [ 22 ] ;
struct Vector x2y2z2 ;
x2y2z2 . x = data [ 23 ] ;
x2y2z2 . y = data [ 24 ] ;
x2y2z2 . z = data [ 25 ] ;
struct Vector x3y3z3 ;
```

Código

```
x3y3z3 . x = data [ 26 ] ;
x3y3z3 . y = data [ 27 ] ;
x3y3z3 . z = data [ 28 ] ;
disco . x0y0z0 = x0y0z0 ;
disco . x1y1z1 = x1y1z1 ;
disco . x2y2z2 = x2y2z2 ;
disco . x3y3z3 = x3y3z3 ;
disco . Xc = data [ 0 ] ;
disco . Yc = data [ 1 ] ;
disco . Zc = data [ 2 ] ;
disco . intersectionFuncion = discIntersection ;
disco . normalVector = discNormal ;
disco . retrieveTextureColor = planeTexture ;
```

Código

```
struct Vector puntoCentral ;
puntoCentral . x = data [ 0 ] ;
puntoCentral . y = data [ 1 ] ;
puntoCentral . z = data [ 2 ] ;
struct Vector normalNotNormalized ;
normalNotNormalized . x = data [ 3 ] ;
normalNotNormalized . y = data [ 4 ] ;
normalNotNormalized . z = data [ 5 ] ;
struct Color colorDisco ;
colorDisco . r = data [ 6 ] ;
colorDisco . g = data [ 7 ] ;
colorDisco . b = data [ 8 ] ;
disco . color = colorDisco ;
```

Código

```
long double dPlano = whatsTheDGeneral ( normalNotNormalized , puntoCentral ) ;
dPlano = dPlano / getNorm ( normalNotNormalized ) ;
disco . extraD = dPlano ;
normalNotNormalized = normalize ( normalNotNormalized ) ;
disco . directionVector = normalNotNormalized ;
disco . other = data [ 9 ] ;
disco . o1 = data [ 10 ] ;
disco . o2 = data [ 11 ] ;
disco . o3 = data [ 12 ] ;
disco . Kd = data [ 13 ] ;
disco . Ka = data [ 14 ] ;
disco . Kn = data [ 15 ] ;
disco . Ks = data [ 16 ] ;
```

Código

```
disco . planeCuts = planeCutsFound ;
disco . numberPlaneCuts = numberPlaneCuts ;
disco . textures = texturesFound ;
disco . numberTextures = numberTextures ;
Objects [ objectIndex ] = disco ;
if ( debug == 1 ) {
printPlaneCuts ( Objects [ objectIndex ] ) ;
printTextures ( Objects [ objectIndex ] , whichObjectCreate ) ;
}
objectIndex ++ ;
return ;
}
case 7 : {
```

Código

```
if ( debug == 1 ) {
printf ( "Insertando Elipses..." ) ;
printf ( "Foco 1: (%LF, %LF, %LF) \n" , data [ 0 ] , data [ 1 ] , data [ 2 ] ) ;
printf ( "Foco 2: (%LF, %LF, %LF) \n" , data [ 3 ] , data [ 4 ] , data [ 5 ] ) ;
printf ( "Normal no normalizada: (%LF, %LF, %LF) \n" , data [ 6 ] , data [ 7 ] , data [ 8 ] ) ;
printf ( "Color: (%LF, %LF, %LF) \n" , data [ 9 ] , data [ 8 ] , data [ 9 ] ) ;
printf ( "K del elipse: %LF \n" , data [ 12 ] ) ;
printf ( "o1:  %LF, o2: %LF, o3: %LF \n" , data [ 13 ] , data [ 14 ] , data [ 15 ] ) ;
printf ( "Elipse Kd: %LF \n" , data [ 16 ] ) ;
printf ( "Elipse Ka: %LF \n" , data [ 17 ] ) ;
printf ( "Elipse Kn: %LF \n" , data [ 18 ] ) ;
printf ( "Elipse Ks: %LF \n" , data [ 19 ] ) ;
printf ( "Esquina inferior izquierda (%LF, %LF, %LF) \n" , data [ 20 ] , data [ 21 ] , data [ 22 ] ) ;
```

Código

```
printf ( "Esquina inferior derecha (%LF, %LF, %LF) \n" , data [ 23 ] , data [ 24 ] , data [ 25 ] ) ;
printf ( "Esquina superior derecha (%LF, %LF, %LF) \n" , data [ 26 ] , data [ 27 ] , data [ 28 ] ) ;
printf ( "Esquina superior izquierda (%LF, %LF, %LF) \n" , data [ 29 ] , data [ 30 ] , data [ 31 ] ) ;
}
struct Object elipse ;
struct Vector x0y0z0 ;
x0y0z0 . x = data [ 20 ] ;
x0y0z0 . y = data [ 21 ] ;
x0y0z0 . z = data [ 22 ] ;
struct Vector x1y1z1 ;
x1y1z1 . x = data [ 23 ] ;
x1y1z1 . y = data [ 24 ] ;
x1y1z1 . z = data [ 25 ] ;
```

Código

```
struct Vector x2y2z2 ;
x2y2z2 . x = data [ 26 ] ;
x2y2z2 . y = data [ 27 ] ;
x2y2z2 . z = data [ 28 ] ;
struct Vector x3y3z3 ;
x3y3z3 . x = data [ 29 ] ;
x3y3z3 . y = data [ 30 ] ;
x3y3z3 . z = data [ 31 ] ;
elipse . x0y0z0 = x0y0z0 ;
elipse . x1y1z1 = x1y1z1 ;
elipse . x2y2z2 = x2y2z2 ;
elipse . x3y3z3 = x3y3z3 ;
elipse . intersectionFuncion = elipseIntersection ;
```

Código

```
elipse . normalVector = elipseNormal ;
elipse . retrieveTextureColor = planeTexture ;
struct Vector foco1 ;
foco1 . x = data [ 0 ] ;
foco1 . y = data [ 1 ] ;
foco1 . z = data [ 2 ] ;
elipse . Xc = data [ 0 ] ;
elipse . Yc = data [ 1 ] ;
elipse . Zc = data [ 2 ] ;
elipse . Xother = data [ 3 ] ;
elipse . Yother = data [ 4 ] ;
elipse . Zother = data [ 5 ] ;
struct Vector normalNotNormalized ;
```

Código

```
normalNotNormalized . x = data [ 6 ] ;
normalNotNormalized . y = data [ 7 ] ;
normalNotNormalized . z = data [ 8 ] ;
struct Color colorElipse ;
colorElipse . r = data [ 9 ] ;
colorElipse . g = data [ 10 ] ;
colorElipse . b = data [ 11 ] ;
elipse . color = colorElipse ;
long double dPlano = whatsTheDGeneral ( normalNotNormalized , foco1 ) ;
dPlano = dPlano / getNorm ( normalNotNormalized ) ;
elipse . extraD = dPlano ;
normalNotNormalized = normalize ( normalNotNormalized ) ;
elipse . directionVector = normalNotNormalized ;
```

Código

```
elipse . other = data [ 12 ] ;
elipse . o1 = data [ 13 ] ;
elipse . o2 = data [ 14 ] ;
elipse . o3 = data [ 15 ] ;
elipse . Kd = data [ 16 ] ;
elipse . Ka = data [ 17 ] ;
elipse . Kn = data [ 18 ] ;
elipse . Ks = data [ 19 ] ;
elipse . planeCuts = planeCutsFound ;
elipse . numberPlaneCuts = numberPlaneCuts ;
elipse . textures = texturesFound ;
elipse . numberTextures = numberTextures ;
Objects [ objectIndex ] = elipse ;
```

Código

```
if ( debug == 1 ) {
printPlaneCuts ( Objects [ objectIndex ] ) ;
printTextures ( Objects [ objectIndex ] , whichObjectCreate ) ;
}
objectIndex ++ ;
return ;
}
case 8 : {
if ( debug == 1 ) {
printf ( "Insertando Cuadrtica..." ) ;
printf ( "Coeficientes: \n\t A: %LF \n\t B: %LF \n\t C: %LF\n\t D: %LF\n\t E: %LF \n\t F: %LF\n\t G: %LF \
printf ( "Constante K: %LF \n" , data [ 10 ] ) ;
printf ( "o1:  %LF, o2: %LF, o3: %LF \n" , data [ 11 ] , data [ 12 ] , data [ 13 ] ) ;
```

Código

```
printf ( "Elipse Kd: %LF \n" , data [ 14 ] ) ;
printf ( "Elipse Ka: %LF \n" , data [ 15 ] ) ;
printf ( "Elipse Kn: %LF \n" , data [ 16 ] ) ;
printf ( "Elipse Ks: %LF \n" , data [ 17 ] ) ;
printf ( "Color: (%LF, %LF, %LF) \n" , data [ 18 ] , data [ 19 ] , data [ 20 ] ) ;
}
struct Object cuadratica ;
cuadratica . A = data [ 0 ] ;
cuadratica . B = data [ 1 ] ;
cuadratica . C = data [ 2 ] ;
cuadratica . D = data [ 3 ] ;
cuadratica . E = data [ 4 ] ;
cuadratica . F = data [ 5 ] ;
```

Código

```
cuadratica . G = data [ 6 ] ;
cuadratica . H = data [ 7 ] ;
cuadratica . I = data [ 8 ] ;
cuadratica . J = data [ 9 ] ;
cuadratica . other = data [ 10 ] ;
cuadratica . o1 = data [ 11 ] ;
cuadratica . o2 = data [ 12 ] ;
cuadratica . o3 = data [ 13 ] ;
cuadratica . Kd = data [ 14 ] ;
cuadratica . Ka = data [ 15 ] ;
cuadratica . Kn = data [ 16 ] ;
cuadratica . Ks = data [ 17 ] ;
struct Color colorCuadratica ;
```

Código

```
colorCuadratica . r = data [ 18 ] ;
colorCuadratica . g = data [ 19 ] ;
colorCuadratica . b = data [ 20 ] ;
cuadratica . color = colorCuadratica ;
cuadratica . intersectionFuncion = quadraticIntersection ;
cuadratica . normalVector = quadraticNormal ;
cuadratica . planeCuts = planeCutsFound ;
cuadratica . numberPlaneCuts = numberPlaneCuts ;
cuadratica . textures = texturesFound ;
cuadratica . numberTextures = numberTextures ;
Objects [ objectIndex ] = cuadratica ;
if ( debug == 1 ) {
printPlaneCuts ( Objects [ objectIndex ] ) ;
```

Código

```
printTextures ( Objects [ objectIndex ] , whichObjectCreate ) ;
printf ( "sup" ) ;
}
objectIndex ++ ;
return ;
}
}
}
long double obtainSingleValueFromLine ( char line [ ] ) {
char * token ;
char * search = "=" ;
long double numericValue ;
token = strtok ( line , search ) ;
```

Código

```
token = strtok ( NULL , search ) ;
sscanf ( token , "%LF" , & numericValue ) ;
return numericValue ;
}
long double * obtainPointFromString ( char stringPoint [ ] ) {
char * token ;
char * search = "=" ;
long double numericValue ;
token = strtok ( stringPoint , search ) ;
token = strtok ( NULL , search ) ;
char * pch ;
long double * pointDimensions = malloc ( sizeof ( long double ) * 3 ) ;
int currentDimension = 0 ;
```

Código

```
pch = strtok ( token , "," ) ;
while ( pch != NULL )
{
sscanf ( pch , "%LF" , & pointDimensions [ currentDimension ] ) ;
pch = strtok ( NULL , "," ) ;
currentDimension ++ ;
}
return pointDimensions ;
}
void strip ( char * s ) {
char * p2 = s ;
while ( * s != '\0' ) {
if ( * s != '\t' && * s != '\n' ) {
```

Código

```
* p2 ++ = * s ++ ;
} else {
++ s ;
}
}
* p2 = '\0' ; }
char * obtainFilenameTexture ( char stringLine [ ] ) {
char * token = malloc ( sizeof ( char ) * 200 ) ;
char * search = "=" ;
token = strtok ( stringLine , search ) ;
token = strtok ( NULL , search ) ;
strip ( token ) ;
return token ; }
```

Código

```
struct PlaneCut * readPlaneCuts ( long int pos , int * numberPlanes , long int * posAfterReading ) {
char temporalBuffer [ 300 ] ;
struct PlaneCut * planeCutsFound = NULL ;
long double * datosPlanos ;
int indexPlaneCut = - 1 ;
FILE * file ;
if ( file = fopen ( escenaFile , "r" ) ) {
fseek ( file , pos , SEEK_SET ) ;
while ( fgets ( temporalBuffer , 300 , file ) != NULL ) {
if ( temporalBuffer [ 0 ] == '\n' ) {
continue ;
}
if ( strstr ( temporalBuffer , "#" ) != NULL ) {
```

Código

```
continue ;
}
if ( strstr ( temporalBuffer , "NumberPlanes" ) != NULL ) {
long double numberPlanes = obtainSingleValueFromLine ( temporalBuffer ) ;
planeCutsFound = malloc ( sizeof ( struct PlaneCut ) * numberPlanes ) ;
continue ;
} else if ( strstr ( temporalBuffer , "Plano_" ) != NULL ) {
indexPlaneCut ++ ;
continue ;
} else if ( strstr ( temporalBuffer , "END_Planos" ) != NULL ) {
* numberPlanes = indexPlaneCut + 1 ;
* posAfterReading = ftell ( file ) ;
return planeCutsFound ;
```

Código

```
} else if ( strstr ( temporalBuffer , "Punto" ) != NULL ) {
datosPlanos = obtainPointFromString ( temporalBuffer ) ;
struct Vector temp ;
temp . x = datosPlanos [ 0 ] ;
temp . y = datosPlanos [ 1 ] ;
temp . z = datosPlanos [ 2 ] ;
planeCutsFound [ indexPlaneCut ] . point = temp ;
free ( datosPlanos ) ;
continue ;
} else if ( strstr ( temporalBuffer , "Normal" ) != NULL ) {
datosPlanos = obtainPointFromString ( temporalBuffer ) ;
struct Vector temp ;
temp . x = datosPlanos [ 0 ] ;
```

Código

```
temp . y = datosPlanos [ 1 ] ;
temp . z = datosPlanos [ 2 ] ;
long double dEquation = whatsTheDGeneral ( temp , planeCutsFound [ indexPlaneCut ] . point ) ;
dEquation = dEquation / getNorm ( temp ) ;
temp = normalize ( temp ) ;
planeCutsFound [ indexPlaneCut ] . normal = temp ;
planeCutsFound [ indexPlaneCut ] . d = dEquation ;
free ( datosPlanos ) ;
continue ;
}
continue ;
}
}
```

Código

```
}
struct Color * * getTexels ( char * pFile , int * hRes , int * vRes ) {
int counter , x , y , i , j ;
char dump [ 100 ] ;
time_t t ;
struct Color * * temp ;
srand ( ( unsigned ) time ( & t ) ) ;
FILE * file ;
if ( file = fopen ( pFile , "r" ) ) {
for ( i = 0 ; i < 11 ; ++ i ) {
fscanf ( file , "%s" , & dump [ 0 ] ) ;
if ( i == 8 || i == 9 ) {
if ( i == 8 ) {
```

Código

```
sscanf ( & dump [ 0 ] , "%i" , hRes ) ;
} else {
sscanf ( & dump [ 0 ] , "%i" , vRes ) ;
}
}
}
temp = malloc ( sizeof ( struct Color * ) * ( * hRes ) ) ;
for ( int r = 0 ; r < ( * hRes ) ; r ++ ) {
temp [ r ] = malloc ( sizeof ( struct Color ) * ( * vRes ) ) ;
}
if ( temp == NULL ) {
printf ( "Devolvi NULL \n" ) ;
}
```

Código

```
char temporalBuffer [ 2000 ] ;
int i = 0 , x = 0 , y = 0 , counter = 0 ;
while ( fgets ( temporalBuffer , 200 , file ) != NULL ) {
if ( temporalBuffer [ 0 ] == '\n' ) {
continue ;
}
struct Color texel ;
long double number ;
sscanf ( temporalBuffer , "%LF" , & number ) ;
int xs = * hRes - x - 1 ;
if ( i == 0 ) {
temp [ y ] [ xs ] . r = ( number ) / 255 ;
} else if ( i == 1 ) {
```

Código

```
temp [ y ] [ xs ] . g = ( number ) / 255 ;
} else if ( i == 2 ) {
temp [ y ] [ xs ] . b = ( number ) / 255 ;
}
i = ( i + 1 ) % 3 ;
if ( i == 0 ) {
y = ( y + 1 ) ;
y = y % ( * vRes ) ;
if ( y == 0 ) {
x = ( x + 1 ) ;
x = x % ( * hRes ) ;
if ( x == 0 ) {
break ;
```

Código

```
}
}
}
counter = 1 ;
}
y = 0 ;
x = 0 ;
while ( counter != 5 ) {
counter ++ ;
y ++ ;
}
}
else {
```

Código

```
( * vRes ) = 128 ;
( * hRes ) = 128 ;
temp = malloc ( sizeof ( struct Color * ) * 128 ) ;
for ( int r = 0 ; r < ( * hRes ) ; r ++ ) {
temp [ r ] = malloc ( sizeof ( struct Color ) * 128 ) ;
}
if ( temp == NULL ) {
printf ( "Devolvi NULL \n" ) ;
}
printf ( "La textura de %s no pudo abrirse. Se sustituir por esttica\n" , pFile ) ;
for ( x = 0 ; x < 128 ; x ++ ) {
for ( y = 0 ; y < 128 ; y ++ ) {
struct Color estatica ;
```

Código

```
estatica . r = ( ( long double ) ( rand ( ) % 255 ) ) / 255 ;
estatica . g = ( ( long double ) ( rand ( ) % 255 ) ) / 255 ;
estatica . b = ( ( long double ) ( rand ( ) % 255 ) ) / 255 ;
temp [ x ] [ y ] = estatica ;
}
}
}
return temp ;
}
struct Texture * readTextures ( int currentTypeReading , long int pos , int * numberTextures , long int *
char temporalBuffer [ 300 ] ;
struct Texture * texturesFound = NULL ;
long double * datosTexture ;
```

Código

```
int indexTexture = - 1 ;
FILE * file ;
if ( file = fopen ( escenaFile , "r" ) ) {
fseek ( file , pos , SEEK_SET ) ;
while ( fgets ( temporalBuffer , 300 , file ) != NULL ) {
if ( temporalBuffer [ 0 ] == '\n' ) {
continue ;
}
if ( temporalBuffer [ 0 ] == '\t' ) {
continue ;
}
if ( strstr ( temporalBuffer , "#" ) != NULL ) {
continue ;
```

Código

```
}
if ( strstr ( temporalBuffer , "NumberTextures" ) != NULL || strstr ( temporalBuffer , "NumberTexturas" )
long double numberTextures = obtainSingleValueFromLine ( temporalBuffer ) ;
texturesFound = malloc ( sizeof ( struct Texture ) * numberTextures ) ;
continue ;
} else if ( strstr ( temporalBuffer , "Texture_" ) != NULL || strstr ( temporalBuffer , "Textura_" ) != NU
indexTexture ++ ;
continue ;
} else if ( strstr ( temporalBuffer , "END_Textures" ) != NULL || strstr ( temporalBuffer , "END_Texturas"
* numberTextures = indexTexture + 1 ;
* posAfterReading = ftell ( file ) ;
return texturesFound ;
} else if ( strstr ( temporalBuffer , "Filename" ) != NULL || strstr ( temporalBuffer , "filename" ) != NU
```

Código

```
char * filename = obtainFilenameTexture ( temporalBuffer ) ;
int hRes , vRes ;
texturesFound [ indexTexture ] . filename = filename ;
struct Color * * textureMap = getTexels ( texturesFound [ indexTexture ] . filename , & hRes , & vRes ) ;
texturesFound [ indexTexture ] . textureMap = textureMap ;
texturesFound [ indexTexture ] . hRes = hRes ;
texturesFound [ indexTexture ] . vRes = vRes ;
continue ;
}
if ( currentTypeReading == 2 || currentTypeReading == 4 || currentTypeReading == 5 || currentTypeReading =
if ( strstr ( temporalBuffer , "Greenwich" ) != NULL ) {
datosTexture = obtainPointFromString ( temporalBuffer ) ;
struct Vector temp ;
```

Código

```
temp . x = datosTexture [ 0 ] ;
temp . y = datosTexture [ 1 ] ;
temp . z = datosTexture [ 2 ] ;
temp = normalize ( temp ) ;
texturesFound [ indexTexture ] . greenwich = temp ;
free ( datosTexture ) ;
continue ;
}
if ( currentTypeReading == 2 || currentTypeReading == 8 ) {
if ( strstr ( temporalBuffer , "Norte" ) != NULL || strstr ( temporalBuffer , "North" ) != NULL ) {
datosTexture = obtainPointFromString ( temporalBuffer ) ;
struct Vector temp ;
temp . x = datosTexture [ 0 ] ;
```

Código

```
temp . y = datosTexture [ 1 ] ;
temp . z = datosTexture [ 2 ] ;
temp = normalize ( temp ) ;
texturesFound [ indexTexture ] . north = temp ;
free ( datosTexture ) ;
continue ;
}
}
}
continue ;
}
}
}
```

Código

```
struct DraftPlane * readDraftPlanes ( int currentTypeReading , long int pos , int * numberDraftPlanes , lo
char temporalBuffer [ 300 ] ;
struct DraftPlane * draftPlanesFound = NULL ;
long double * datosDraftPlane ;
int indexDraftPlane = - 1 ;
FILE * file ;
if ( file = fopen ( escenaFile , "r" ) ) {
fseek ( file , pos , SEEK_SET ) ;
while ( fgets ( temporalBuffer , 300 , file ) != NULL ) {
if ( temporalBuffer [ 0 ] == '\n' ) {
continue ;
}
if ( temporalBuffer [ 0 ] == '\t' ) {
```

Código

```
continue ;
}
if ( strstr ( temporalBuffer , "#" ) != NULL ) {
continue ;
}
if ( strstr ( temporalBuffer , "NumberPlanosCalado" ) != NULL || strstr ( temporalBuffer , "NumberDraftPla
long double numberDraftPlanes = obtainSingleValueFromLine ( temporalBuffer ) ;
draftPlanesFound = malloc ( sizeof ( struct DraftPlane ) * numberDraftPlanes ) ;
continue ;
} else if ( strstr ( temporalBuffer , "Plano_Calado_" ) != NULL || strstr ( temporalBuffer , "PlanoCalado_
indexDraftPlane ++ ;
continue ;
} else if ( ( strstr ( temporalBuffer , "END_Planos_Calado" ) != NULL ) || ( strstr ( temporalBuffer , "EN
```

Código

```
* numberDraftPlanes = indexDraftPlane + 1 ;
* posAfterReading = ftell ( file ) ;
return draftPlanesFound ;
} else if ( strstr ( temporalBuffer , "Filename" ) != NULL || strstr ( temporalBuffer , "filename" ) != NU
char * filename = obtainFilenameTexture ( temporalBuffer ) ;
int hRes , vRes ;
draftPlanesFound [ indexDraftPlane ] . filename = filename ;
struct Color * * textureMap = getTexels ( draftPlanesFound [ indexDraftPlane ] . filename , & hRes , & vRe
draftPlanesFound [ indexDraftPlane ] . textureMap = textureMap ;
draftPlanesFound [ indexDraftPlane ] . hRes = hRes ;
draftPlanesFound [ indexDraftPlane ] . vRes = vRes ;
continue ;
}
```

Código

```
if ( currentTypeReading == 2 || currentTypeReading == 4 || currentTypeReading == 5 || currentTypeReading =
if ( strstr ( temporalBuffer , "Greenwich" ) != NULL ) {
datosDraftPlane = obtainPointFromString ( temporalBuffer ) ;
struct Vector temp ;
temp . x = datosDraftPlane [ 0 ] ;
temp . y = datosDraftPlane [ 1 ] ;
temp . z = datosDraftPlane [ 2 ] ;
temp = normalize ( temp ) ;
draftPlanesFound [ indexDraftPlane ] . greenwich = temp ;
free ( datosDraftPlane ) ;
continue ;
}
if ( currentTypeReading == 2 || currentTypeReading == 8 ) {
```

Código

```
if ( strstr ( temporalBuffer , "Norte" ) != NULL || strstr ( temporalBuffer , "North" ) != NULL ) {
datosDraftPlane = obtainPointFromString ( temporalBuffer ) ;
struct Vector temp ;
temp . x = datosDraftPlane [ 0 ] ;
temp . y = datosDraftPlane [ 1 ] ;
temp . z = datosDraftPlane [ 2 ] ;
temp = normalize ( temp ) ;
draftPlanesFound [ indexDraftPlane ] . north = temp ;
free ( datosDraftPlane ) ;
continue ;
}
}
}
```

Código

```
continue ;
}
}
}
long double * readValueFromLine ( int state , int * counterValueSegment , char * lineRead , int * numberVa
long double * values ;
switch ( state ) {
case 0 :
if ( ( * counterValueSegment ) >= 0 && ( * counterValueSegment ) <= 10 ) {
values = malloc ( sizeof ( long double ) ) ;
values [ 0 ] = obtainSingleValueFromLine ( lineRead ) ;
( * counterValueSegment ) ++ ;
* numberValuesRead = 1 ;
```

Código

```
return values ;
} else if ( ( * counterValueSegment ) >= 11 && ( * counterValueSegment ) <= 12 ) {
long double * point = obtainPointFromString ( lineRead ) ;
values = malloc ( sizeof ( long double ) * 3 ) ;
values [ 0 ] = point [ 0 ] ;
values [ 1 ] = point [ 1 ] ;
values [ 2 ] = point [ 2 ] ;
* numberValuesRead = 3 ;
free ( point ) ;
if ( ( * counterValueSegment ) == 12 ) {
( * counterValueSegment ) = 0 ;
} else {
( * counterValueSegment ) ++ ;
```

Código

```
}
return values ;
}
case 1 :
if ( ( * counterValueSegment ) == 0 ) {
long double * positionLight = obtainPointFromString ( lineRead ) ;
values = malloc ( sizeof ( long double ) * 3 ) ;
values [ 0 ] = positionLight [ 0 ] ;
values [ 1 ] = positionLight [ 1 ] ;
values [ 2 ] = positionLight [ 2 ] ;
( * counterValueSegment ) ++ ;
* numberValuesRead = 3 ;
free ( positionLight ) ;
```

Código

```
return values ;
} else if ( ( * counterValueSegment ) >= 1 && ( * counterValueSegment <= 4 ) ) {
values = malloc ( sizeof ( long double ) ) ;
values [ 0 ] = obtainSingleValueFromLine ( lineRead ) ;
if ( ( * counterValueSegment ) == 4 ) {
( * counterValueSegment ) = 0 ;
} else {
( * counterValueSegment ) ++ ;
}
* numberValuesRead = 1 ;
return values ;
}
case 2 :
```

Código

```
if ( ( * counterValueSegment ) == 0 || ( * counterValueSegment ) == 9 ) {
long double * positionSphere = obtainPointFromString ( lineRead ) ;
values = malloc ( sizeof ( long double ) * 3 ) ;
values [ 0 ] = positionSphere [ 0 ] ;
values [ 1 ] = positionSphere [ 1 ] ;
values [ 2 ] = positionSphere [ 2 ] ;
free ( positionSphere ) ;
* numberValuesRead = 3 ;
if ( ( * counterValueSegment ) == 9 ) {
( * counterValueSegment ) = 0 ;
} else {
( * counterValueSegment ) ++ ;
}
```

Código

```
return values ;
} else if ( ( * counterValueSegment ) >= 1 && ( * counterValueSegment ) <= 8 ) {
values = malloc ( sizeof ( long double ) ) ;
values [ 0 ] = obtainSingleValueFromLine ( lineRead ) ;
( * counterValueSegment ) ++ ;
* numberValuesRead = 1 ;
return values ;
}
case 3 :
if ( ( * counterValueSegment ) == 0 ) {
values = malloc ( sizeof ( long double ) * 3 ) ;
long double * rgbColors = obtainPointFromString ( lineRead ) ;
values [ 0 ] = rgbColors [ 0 ] ;
```

Código

```
values [ 1 ] = rgbColors [ 1 ] ;
values [ 2 ] = rgbColors [ 2 ] ;
free ( rgbColors ) ;
* numberValuesRead = 3 ;
( * counterValueSegment ) ++ ;
return values ;
} else if ( ( * counterValueSegment ) >= 1 && ( * counterValueSegment ) <= 7 ) {
values = malloc ( sizeof ( long double ) ) ;
values [ 0 ] = obtainSingleValueFromLine ( lineRead ) ;
( * counterValueSegment ) ++ ;
* numberValuesRead = 1 ;
return values ;
} else if ( ( * counterValueSegment ) == 8 ) {
```

Código

```
if ( strstr ( lineRead , "END_Vertices" ) != NULL ) {
( * counterValueSegment ) ++ ;
return values ;
} else {
values = malloc ( sizeof ( long double ) * 3 ) ;
long double * vertexPolygon = obtainPointFromString ( lineRead ) ;
values [ 0 ] = vertexPolygon [ 0 ] ;
values [ 1 ] = vertexPolygon [ 1 ] ;
values [ 2 ] = vertexPolygon [ 2 ] ;
free ( vertexPolygon ) ;
* numberValuesRead = 3 ;
if ( ( * counterValueSegment ) == 12 ) {
( * counterValueSegment ) = 0 ;
```

Código

```
return values ;
}
if ( ( * counterValueSegment ) != 8 ) {
( * counterValueSegment ) ++ ;
}
return values ;
}
} else if ( ( * counterValueSegment ) >= 9 ) {
values = malloc ( sizeof ( long double ) * 3 ) ;
long double * vertexPolygon = obtainPointFromString ( lineRead ) ;
values [ 0 ] = vertexPolygon [ 0 ] ;
values [ 1 ] = vertexPolygon [ 1 ] ;
values [ 2 ] = vertexPolygon [ 2 ] ;
```

Código

```
free ( vertexPolygon ) ;
* numberValuesRead = 3 ;
if ( ( * counterValueSegment ) == 12 ) {
( * counterValueSegment ) = 0 ;
return values ;
}
( * counterValueSegment ) ++ ;
return values ;
}
case 4 :
if ( * counterValueSegment == 0 || * counterValueSegment == 1 || * counterValueSegment == 12 ) {
long double * positionCilinder = obtainPointFromString ( lineRead ) ;
values = malloc ( sizeof ( long double ) * 3 ) ;
```

Código

```
values [ 0 ] = positionCilinder [ 0 ] ;
values [ 1 ] = positionCilinder [ 1 ] ;
values [ 2 ] = positionCilinder [ 2 ] ;
if ( * counterValueSegment == 12 ) {
( * counterValueSegment ) = 0 ;
} else {
( * counterValueSegment ) ++ ;
}
* numberValuesRead = 3 ;
free ( positionCilinder ) ;
return values ;
} else if ( * counterValueSegment >= 2 && * counterValueSegment <= 11 ) {
values = malloc ( sizeof ( long double ) ) ;
```

Código

```
values [ 0 ] = obtainSingleValueFromLine ( lineRead ) ;
( * counterValueSegment ) ++ ;
* numberValuesRead = 1 ;
return values ;
}
case 5 :
if ( * counterValueSegment == 0 || * counterValueSegment == 1 || * counterValueSegment == 13 ) {
long double * positionCone = obtainPointFromString ( lineRead ) ;
values = malloc ( sizeof ( long double ) * 3 ) ;
values [ 0 ] = positionCone [ 0 ] ;
values [ 1 ] = positionCone [ 1 ] ;
values [ 2 ] = positionCone [ 2 ] ;
if ( * counterValueSegment == 13 ) {
```

Código

```
( * counterValueSegment ) = 0 ;
} else {
( * counterValueSegment ) ++ ;
}
* numberValuesRead = 3 ;
free ( positionCone ) ;
return values ;
} else if ( * counterValueSegment >= 2 && * counterValueSegment <= 12 ) {
values = malloc ( sizeof ( long double ) ) ;
values [ 0 ] = obtainSingleValueFromLine ( lineRead ) ;
( * counterValueSegment ) ++ ;
* numberValuesRead = 1 ;
return values ;
```

Código

```
}
case 6 :
if ( ( * counterValueSegment >= 0 && * counterValueSegment <= 2 ) || ( * counterValueSegment >= 11 && * co
long double * tripleta = obtainPointFromString ( lineRead ) ;
values = malloc ( sizeof ( long double ) * 3 ) ;
values [ 0 ] = tripleta [ 0 ] ;
values [ 1 ] = tripleta [ 1 ] ;
values [ 2 ] = tripleta [ 2 ] ;
if ( * counterValueSegment == 14 ) {
( * counterValueSegment ) = 0 ;
} else {
( * counterValueSegment ) ++ ;
}
```

Código

```
* numberValuesRead = 3 ;
free ( tripleta ) ;
return values ;
} else if ( ( * counterValueSegment >= 3 && * counterValueSegment <= 10 ) ) {
values = malloc ( sizeof ( long double ) ) ;
values [ 0 ] = obtainSingleValueFromLine ( lineRead ) ;
( * counterValueSegment ) ++ ;
* numberValuesRead = 1 ;
return values ;
}
case 7 :
if ( ( * counterValueSegment >= 0 && * counterValueSegment <= 3 ) || ( * counterValueSegment >= 12 && * co
long double * tripleta = obtainPointFromString ( lineRead ) ;
```

Código

```
values = malloc ( sizeof ( long double ) * 3 ) ;
values [ 0 ] = tripleta [ 0 ] ;
values [ 1 ] = tripleta [ 1 ] ;
values [ 2 ] = tripleta [ 2 ] ;
if ( * counterValueSegment == 15 ) {
( * counterValueSegment ) = 0 ;
} else {
( * counterValueSegment ) ++ ;
}
* numberValuesRead = 3 ;
free ( tripleta ) ;
return values ;
} else if ( ( * counterValueSegment >= 4 && * counterValueSegment <= 11 ) ) {
```

Código

```
values = malloc ( sizeof ( long double ) ) ;
values [ 0 ] = obtainSingleValueFromLine ( lineRead ) ;
( * counterValueSegment ) ++ ;
* numberValuesRead = 1 ;
return values ;
}
case 8 :
if ( ( * counterValueSegment ) == 18 ) {
long double * tripleta = obtainPointFromString ( lineRead ) ;
values = malloc ( sizeof ( long double ) * 3 ) ;
values [ 0 ] = tripleta [ 0 ] ;
values [ 1 ] = tripleta [ 1 ] ;
values [ 2 ] = tripleta [ 2 ] ;
```

Código

```
( * counterValueSegment ) = 0 ;
* numberValuesRead = 3 ;
free ( tripleta ) ;
return values ;
} else if ( ( * counterValueSegment >= 0 && * counterValueSegment <= 17 ) ) {
values = malloc ( sizeof ( long double ) ) ;
values [ 0 ] = obtainSingleValueFromLine ( lineRead ) ;
( * counterValueSegment ) ++ ;
* numberValuesRead = 1 ;
return values ;
}
}
}
```

Código

```
int plainCutsFound ( long int pos ) {
char temporalBuffer [ 300 ] ;
FILE * file ;
if ( file = fopen ( escenaFile , "r" ) ) {
fseek ( file , pos , SEEK_SET ) ;
while ( fgets ( temporalBuffer , 300 , file ) != NULL ) {
if ( temporalBuffer [ 0 ] == '\n' ) {
continue ;
}
if ( temporalBuffer [ 0 ] == '\t' ) {
continue ;
}
if ( strstr ( temporalBuffer , "#" ) != NULL ) {
```

Código

```
continue ;
} else if ( strstr ( temporalBuffer , "Planos_Corte:" ) != NULL ) {
return 1 ;
} else if ( strstr ( temporalBuffer , "Texturas:" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Planos_Calado:" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Sphere_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Polygon_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Cylinder_Object" ) != NULL ) {
return 0 ;
```

Código

```
} else if ( strstr ( temporalBuffer , "Cone_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Disc_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Elipse_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Quadratic_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Scene_Data" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Light_Object" ) != NULL ) {
return 0 ;
}
```

Código

```
}
}
return 0 ; }
int texturesFound ( long int pos ) {
char temporalBuffer [ 300 ] ;
FILE * file ;
if ( file = fopen ( escenaFile , "r" ) ) {
fseek ( file , pos , SEEK_SET ) ;
while ( fgets ( temporalBuffer , 300 , file ) != NULL ) {
if ( temporalBuffer [ 0 ] == '\n' ) {
continue ;
}
if ( temporalBuffer [ 0 ] == '\t' ) {
```

Código

```
continue ;
}
if ( strstr ( temporalBuffer , "#" ) != NULL ) {
continue ;
} else if ( strstr ( temporalBuffer , "Texturas:" ) != NULL || strstr ( temporalBuffer , "Textures:" ) !=
return 1 ;
} else if ( strstr ( temporalBuffer , "Planos_Calado:" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Sphere_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Polygon_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Cylinder_Object" ) != NULL ) {
```

Código

```
return 0 ;
} else if ( strstr ( temporalBuffer , "Cone_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Disc_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Elipse_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Quadratic_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Scene_Data" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Light_Object" ) != NULL ) {
return 0 ;
```

Código

```
}
}
}
return 0 ;
}
int draftPlanesFound ( long int pos ) {
char temporalBuffer [ 300 ] ;
FILE * file ;
if ( file = fopen ( escenaFile , "r" ) ) {
fseek ( file , pos , SEEK_SET ) ;
while ( fgets ( temporalBuffer , 300 , file ) != NULL ) {
if ( temporalBuffer [ 0 ] == '\n' ) {
continue ;
```

Código

```
}
if ( temporalBuffer [ 0 ] == '\t' ) {
continue ;
}
if ( strstr ( temporalBuffer , "#" ) != NULL ) {
continue ;
} else if ( ( strstr ( temporalBuffer , "Planos_Calado:" ) != NULL ) || ( strstr ( temporalBuffer , "Plano
return 1 ;
} else if ( strstr ( temporalBuffer , "Sphere_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Polygon_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Cylinder_Object" ) != NULL ) {
```

Código

```
return 0 ;
} else if ( strstr ( temporalBuffer , "Cone_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Disc_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Elipse_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Quadratic_Object" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Scene_Data" ) != NULL ) {
return 0 ;
} else if ( strstr ( temporalBuffer , "Light_Object" ) != NULL ) {
return 0 ;
```

Código

```
}
}
}
return 0 ;
}
void getSceneObjects ( ) {
int i , j , c ;
int state = 0 ;
int counterValueSegment = 0 ;
char temporalBuffer [ 300 ] ;
long double * valuesRead ;
int indexValuesRead = 0 ;
int currentTypeObjectReading = 1 ;
```

Código

```
struct PlaneCut * arrayPlaneCuts = NULL ;
struct Texture * arrayTextures = NULL ;
struct DraftPlane * arrayDraftPlanes = NULL ;
FILE * file ;
if ( file = fopen ( escenaFile , "r" ) ) {
while ( fgets ( temporalBuffer , 300 , file ) != NULL ) {
if ( temporalBuffer [ 0 ] == '\n' ) {
continue ;
}
if ( temporalBuffer [ 0 ] == '\t' ) {
continue ;
}
if ( strstr ( temporalBuffer , "#" ) != NULL ) {
```

Código

```
continue ;
}
if ( strstr ( temporalBuffer , "Scene_Data" ) != NULL ) {
state = 0 ;
counterValueSegment = 0 ;
indexValuesRead = 0 ;
valuesRead = NULL ;
valuesRead = malloc ( sizeof ( long double ) * 22 ) ;
currentTypeObjectReading = 0 ;
continue ;
} else if ( strstr ( temporalBuffer , "Light_Object" ) != NULL ) {
state = 1 ;
counterValueSegment = 0 ;
```

Código

```
indexValuesRead = 0 ;
free ( valuesRead ) ;
valuesRead = malloc ( sizeof ( long double ) * 7 ) ;
currentTypeObjectReading = 1 ;
continue ;
} else if ( strstr ( temporalBuffer , "Sphere_Object" ) != NULL ) {
state = 2 ;
indexValuesRead = 0 ;
free ( valuesRead ) ;
valuesRead = malloc ( sizeof ( long double ) * 22 ) ;
counterValueSegment = 0 ;
currentTypeObjectReading = 2 ;
continue ;
```

Código

```
} else if ( strstr ( temporalBuffer , "Polygon_Object" ) != NULL ) {
state = 3 ;
counterValueSegment = 0 ;
indexValuesRead = 0 ;
free ( valuesRead ) ;
valuesRead = malloc ( sizeof ( long double ) * 2000000 ) ;
currentTypeObjectReading = 3 ;
continue ;
} else if ( strstr ( temporalBuffer , "Cylinder_Object" ) != NULL ) {
state = 4 ;
counterValueSegment = 0 ;
indexValuesRead = 0 ;
free ( valuesRead ) ;
```

Código

```
valuesRead = malloc ( sizeof ( long double ) * 55 ) ;
currentTypeObjectReading = 4 ;
continue ;
} else if ( strstr ( temporalBuffer , "Cone_Object" ) != NULL ) {
state = 5 ;
counterValueSegment = 0 ;
indexValuesRead = 0 ;
free ( valuesRead ) ;
valuesRead = malloc ( sizeof ( long double ) * 55 ) ;
currentTypeObjectReading = 5 ;
continue ;
} else if ( strstr ( temporalBuffer , "Disc_Object" ) != NULL ) {
state = 6 ;
```

Código

```
counterValueSegment = 0 ;
indexValuesRead = 0 ;
free ( valuesRead ) ;
valuesRead = malloc ( sizeof ( long double ) * 55 ) ;
currentTypeObjectReading = 6 ;
continue ;
} else if ( strstr ( temporalBuffer , "Elipse_Object" ) != NULL ) {
state = 7 ;
counterValueSegment = 0 ;
indexValuesRead = 0 ;
free ( valuesRead ) ;
valuesRead = malloc ( sizeof ( long double ) * 55 ) ;
currentTypeObjectReading = 7 ;
```

Código

```
continue ;
} else if ( strstr ( temporalBuffer , "Quadratic_Object" ) != NULL ) {
state = 8 ;
counterValueSegment = 0 ;
indexValuesRead = 0 ;
free ( valuesRead ) ;
valuesRead = malloc ( sizeof ( long double ) * 55 ) ;
currentTypeObjectReading = 8 ;
continue ;
}
int numberValuesRead = 0 ;
long double * valuesReadTemp = readValueFromLine ( state , & counterValueSegment , temporalBuffer , & numb
if ( valuesReadTemp == NULL ) {
```

Código

```
continue ;
}
int i = 0 ;
for ( i = 0 ; i < numberValuesRead ; i ++ ) {
valuesRead [ indexValuesRead + i ] = valuesReadTemp [ i ] ;
}
indexValuesRead += numberValuesRead ;
if ( counterValueSegment == 0 ) {
int numberPlaneCuts = 0 ;
int numberTextures = 0 ;
int numberDraftPlanes = 0 ;
long int posAfterReading ;
long int pos ;
```

Código

```
pos = ftell ( file ) ;
int areTherePlainCuts = plainCutsFound ( pos ) ;
if ( areTherePlainCuts == 1 ) {
pos = ftell ( file ) ;
arrayPlaneCuts = readPlaneCuts ( pos , & numberPlaneCuts , & posAfterReading ) ;
fseek ( file , posAfterReading , SEEK_SET ) ;
}
pos = ftell ( file ) ;
int areThereTextures = texturesFound ( pos ) ;
if ( areThereTextures == 1 ) {
pos = ftell ( file ) ;
arrayTextures = readTextures ( currentTypeObjectReading , pos , & numberTextures , & posAfterReading ) ;
fseek ( file , posAfterReading , SEEK_SET ) ;
```

Código

```
}
int areThereDraftPlanes = draftPlanesFound ( pos ) ;
if ( areThereDraftPlanes == 1 ) {
pos = ftell ( file ) ;
arrayDraftPlanes = readDraftPlanes ( currentTypeObjectReading , pos , & numberDraftPlanes , & posAfterRead
fseek ( file , posAfterReading , SEEK_SET ) ;
}
createObjectFromData ( valuesRead , currentTypeObjectReading , indexValuesRead , arrayPlaneCuts , arrayTe
}
}
free ( valuesRead ) ;
}
fclose ( file ) ;
```

Código

```
}
void howManyObjectsLights ( ) {
char temporalBuffer [ 100 ] ;
FILE * file ;
if ( file = fopen ( escenaFile , "r" ) ) {
while ( fgets ( temporalBuffer , 100 , file ) != NULL ) {
if ( temporalBuffer [ 0 ] == '\n' ) {
continue ;
}
if ( strstr ( temporalBuffer , "#" ) != NULL ) {
continue ;
}
if ( strstr ( temporalBuffer , "Light_Object" ) != NULL ) {
```

Código

```
numberLights ++ ;
continue ;
} else if ( strstr ( temporalBuffer , "Sphere_Object" ) != NULL ) {
numberObjects ++ ;
continue ;
} else if ( strstr ( temporalBuffer , "Polygon_Object" ) != NULL ) {
numberObjects ++ ;
continue ;
} else if ( strstr ( temporalBuffer , "Cone_Object" ) != NULL ) {
numberObjects ++ ;
continue ;
} else if ( strstr ( temporalBuffer , "Cylinder_Object" ) != NULL ) {
numberObjects ++ ;
```

Código

```
continue ;
} else if ( strstr ( temporalBuffer , "Disc_Object" ) != NULL ) {
numberObjects ++ ;
continue ;
} else if ( strstr ( temporalBuffer , "Elipse_Object" ) != NULL ) {
numberObjects ++ ;
continue ;
} else if ( strstr ( temporalBuffer , "Quadratic_Object" ) != NULL ) {
numberObjects ++ ;
continue ;
}
}
}
```

Código

```
fclose ( file ) ;
Objects = malloc ( sizeof ( struct Object ) * numberObjects ) ;
Lights = malloc ( sizeof ( struct Light ) * numberLights ) ;
}
struct Vector throwRay ( long double x , long double y ) {
struct Vector direction ;
long double Xw , Yw ;
Xw = ( long double ) ( ( x ) * Xdif ) / Hres + Xmin ;
Yw = ( long double ) ( ( y ) * Ydif ) / Vres + Ymin ;
direction . x = Xw - eye . x ;
direction . y = Yw - eye . y ;
direction . z = - eye . z ;
direction = normalize ( direction ) ;
```

Código

```
return direction ;
}
struct Color ponderAAcolors ( struct Color c1 , struct Color c2 , struct Color c3 , struct Color c4 ) {
struct Color color ;
color . r = ( c1 . r + c2 . r + c3 . r + c4 . r ) / 4 ;
color . g = ( c1 . g + c2 . g + c3 . g + c4 . g ) / 4 ;
color . b = ( c1 . b + c2 . b + c3 . b + c4 . b ) / 4 ;
return color ;
}
long double getDistanceColors ( struct Color c1 , struct Color c2 ) {
return sqrt ( pow ( c1 . r - c2 . r , 2 ) + pow ( c1 . g - c2 . g , 2 ) + pow ( c1 . b - c2 . b , 2 ) ) ;
}
int theyOK ( struct Color c1 , struct Color c2 , struct Color c3 , struct Color c4 ) {
```

Código

```
long double dist , dist1 , dist2 , dist3 , dist4 , dist5 , dist6 ;
dist1 = getDistanceColors ( c1 , c2 ) ;
dist2 = getDistanceColors ( c1 , c3 ) ;
dist3 = getDistanceColors ( c1 , c4 ) ;
dist4 = getDistanceColors ( c2 , c3 ) ;
dist5 = getDistanceColors ( c2 , c4 ) ;
dist6 = getDistanceColors ( c3 , c4 ) ;
dist = ( dist1 + dist2 + dist3 + dist4 + dist5 + dist6 ) / 6 ;
if ( dist <= similar ) {
return 1 ;
} else {
return 0 ;
}
```

Código

```
}
struct Color getAAColor ( long double x , long double y , int aaLevel ) {
int areOK ;
int level = aaLevel ;
struct Color color , color1 , color2 , color3 , color4 ;
struct Vector direction , V ;
long double sum = 1 / pow ( 2 , level ) ;
level ++ ;
long double nextSum = 1 / pow ( 2 , level ) ;
direction = throwRay ( x , y ) ;
V . x = - direction . x ;
V . y = - direction . y ;
V . z = - direction . z ;
```

Código

```
color1 = getColor ( eye , direction , V , maxReflection ) ;
direction = throwRay ( x , y + sum ) ;
V . x = - direction . x ;
V . y = - direction . y ;
V . z = - direction . z ;
color2 = getColor ( eye , direction , V , maxReflection ) ;
direction = throwRay ( x + sum , y ) ;
V . x = - direction . x ;
V . y = - direction . y ;
V . z = - direction . z ;
color3 = getColor ( eye , direction , V , maxReflection ) ;
direction = throwRay ( x + sum , y + sum ) ;
V . x = - direction . x ;
```

Código

```
V . y = - direction . y ;
V . z = - direction . z ;
color4 = getColor ( eye , direction , V , maxReflection ) ;
areOK = theyOK ( color1 , color2 , color3 , color4 ) ;
if ( areOK == 0 && level <= maxAA ) {
color1 = getAAColor ( x , y , level ) ;
color2 = getAAColor ( x , y + nextSum , level ) ;
color3 = getAAColor ( x + nextSum , y , level ) ;
color4 = getAAColor ( x + nextSum , y + nextSum , level ) ;
color = ponderAAcolors ( color1 , color2 , color3 , color4 ) ;
} else {
color = ponderAAcolors ( color1 , color2 , color3 , color4 ) ;
}
```

Código

```
return color ;
}
void * runRT ( void * x ) {
int start , i , j ;
struct Color color ;
i = * ( ( int * ) x ) ;
printf ( "%i\n" , i ) ;
for ( i ; i < Vres ; i += 4 ) {
for ( j = 0 ; j < Hres ; j ++ ) {
color = getAAColor ( ( long double ) j , ( long double ) i , 0 ) ;
Framebuffer [ i ] [ j ] = color ;
}
}
```

Código

```
return NULL ;
}
int main ( int argc , char * arcgv [ ] ) {
howManyObjectsLights ( ) ;
printf ( "Lights: %i \n" , numberLights ) ;
printf ( "Objects: %i \n" , numberObjects ) ;
getSceneObjects ( ) ;
int i ;
pthread_t threads [ 4 ] ;
Xdif = Xmax - Xmin ;
Ydif = Ymax - Ymin ;
similar = sqrt ( 3 ) / 32 ;
printf ( "\nRay Tracing\n...\n...\n" ) ;
```

Código

```
for ( i = 0 ; i < 4 ; i ++ ) {
pthread_create ( & ( threads [ i ] ) , NULL , & runRT , & i ) ;
sleep ( 1 ) ;
}
for ( i = 0 ; i < 4 ; i ++ ) {
pthread_join ( threads [ i ] , NULL ) ;
}
printf ( "Rays: %LF\n" , rays ) ;
saveFile ( ) ;
free ( Objects ) ;
free ( Lights ) ;
free ( Framebuffer ) ;
printf ( "\nDONE.\n" ) ;
```

Código