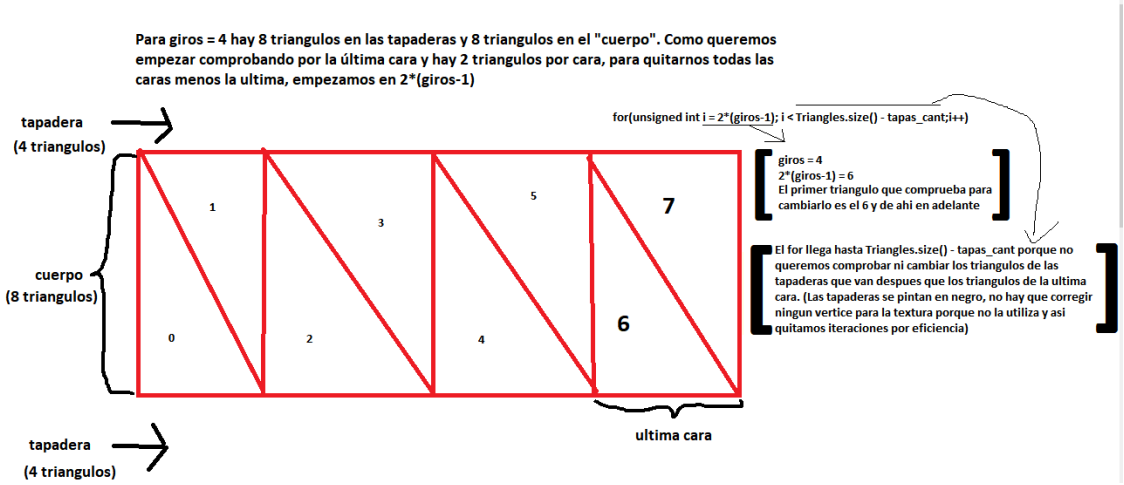
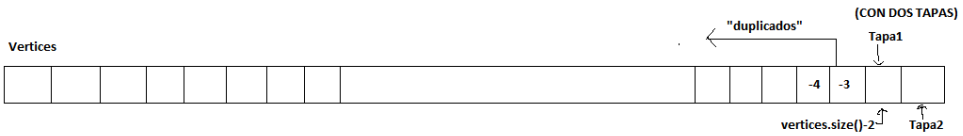
CILINDRO.CC

CILINDRO.CC (LINEA 75)

Para evitar que los triángulos de la última cara (sin contar las tapas) utilicen los primeros vértices (0, 1...), cambiamos estos por los nuevos vértices que son diferentes, pero tienen las mismas coordenadas. Esto es porque no puedes hacer que el vértice 0 sea el comienzo de la imagen en el vector Texturas y a la vez sea el final de la imagen. Los vértices que son diferentes, pero tienen las mismas coordenadas se introducen en la función de circular sweeping justo antes de meter las tapaderas de nuevo al vector de vértices. Esto duplica todos los puntos que haya en el perfil original (sin contar las tapas).

El for comienza en i=2*(giros-1) porque queremos evitar que compruebe los triángulos iniciales (porque estos también usan los vértices 0, 1, etc pero esos no los queremos cambiar, tiene sentido que utilicen el comienzo de la imagen).



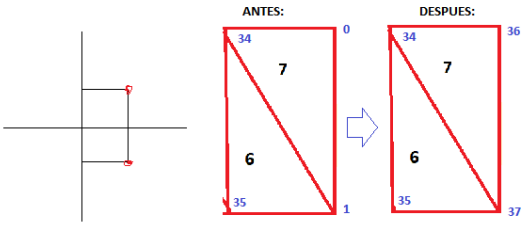


dentro del for anterior:

```
if(Triangles[i]._0 == 0) Triangles[i]._0 = Vertices.size()-4;
if(Triangles[i]._0 == 1) Triangles[i]._0 = Vertices.size()-3;

if(Triangles[i]._1 == 0) Triangles[i]._1 = Vertices.size()-4;
if(Triangles[i]._1 == 1) Triangles[i]._1 = Vertices.size()-3;

if(Triangles[i]._2 == 0) Triangles[i]._2 = Vertices.size()-4;
if(Triangles[i]._2 == 1) Triangles[i]._2 = Vertices.size()-3;
```

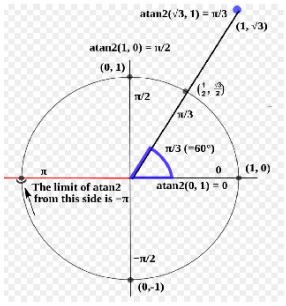


Como sabemos que tiene dos tapas que se incluyen al final del vector Vertices dentro de circular_sweeping, los vertices "duplicados" (aquellos que tienen las mismas coordenadas que los originales pero con otro numero para darle otra parte de la foto) estaran situados a partir de Vertices.size() - 2 hacia el origen. Si solo hubiese una tapa, estarian a partir de vertices.size() - 1 hacia el origen

Como el cilindro, una vez quitadas las tapas, solo tiene 2 puntos en el perfil (los rojos), en este caso los vertices "duplicados" estaran unicamente en size - 3 (el que tiene las mismas coordenadas que 1) y size - 4 (el que tiene las mismas coordenadas que 0). Esto es porque se meten al final de Vertices con el mismo orden en el que estan al comienzo

Esto es asi por tener 2 tapaderas, pero si tenemos solamente 1 tapadera, en circular_sweeping solo se añade un elemento (la tapa) al final de vertices, por lo que los "duplicados" ya no estan en -3 y -4, sino que estan en -2 y -3. De la misma manera, si no hay tapas, los "duplicados" estaran en -1 y -2.

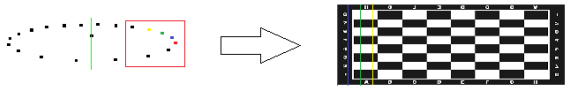
CILINDRO.CC (LINEA 96)



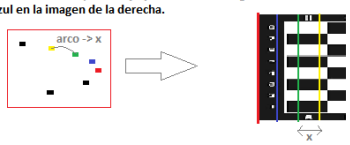

```
float alpha = atan2(Vertices[i]._2, Vertices[i]._0);
```

Para poner la imagen sobre el cilindro, a cada vertice debemos darle una coordenada de la imagen para que cuando se pinte en draw_textura y se llame al vector Textura, coincida correctamente.

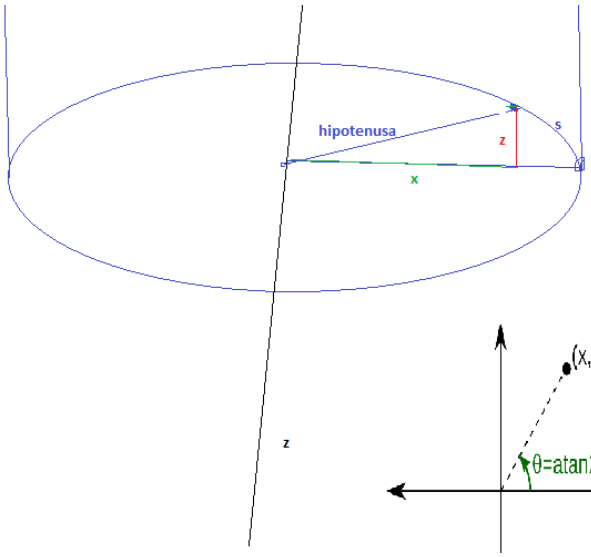
Debemos hacer coincidir la rotacion de los puntos con la imagen plana como se ve a continuación:



Para ello, necesitamos saber cuál es el arco entre los vertices para buscar su equivalente en la imagen. Por ejemplo, el arco entre el punto rojo y el azul en la imagen anterior mide lo mismo que la distancia entre la linea roja y la azul en la imagen de la derecha.



Como sabemos, el arco es equivalente al angulo y la formula del angulo es: alpha = atan2(z, x)



s = arco = radio · angulo

radio = hipotenusa = raiz(x^2 + z^2)

x = cos (angulo) z = sen (angulo)

radio = hipotenusa = raiz(cos(angulo)^2 + sen(angulo)^2)

cos (angulo) ^ 2 + sen (angulo) ^ 2 = 1

radio = hipotenusa = raiz(1) = 1

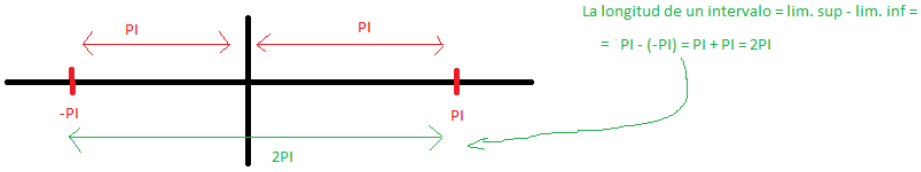
s = arco = radio · angulo = 1 · angulo = angulo

angulo = atan2(z, x)

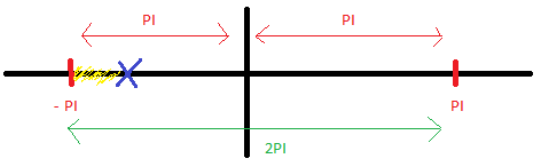
CILINDRO.CC (LINEA 100)

Como los valores de atan2 se encuentran en el intervalo [-PI, PI] y para la imagen necesitamos que esten en el [0,1] hacemos las siguientes transformaciones:

Si queremos que nuestro limite inferior (actualmente -PI) sea el limite inferior del nuevo intervalo (0) planteamos la siguiente situacion:



Si suponemos que 2PI es el total del intervalo, podemos considerarlo como un 1 sobre 1. Para pasar cualquier número de ese intervalo, a uno de [0,1] vemos qué distancia hay hasta el limite inferior y hacemos una regla de 3. Por ejemplo:



El punto azul puede estar situado en cualquier punto, pero la parte de la barra que está de color amarillo mide X - (-PI) , que es lo que ocupa del total (2PI). Si hacemos una regla de 3 dando por hecho que ahora el valor de 2PI es 1, queda asi:

X- (- PI) ----------> 2PI
?----------->1

Si despejamos ? = (X+ PI) / 2PI

Es decir, como hemos visto, para pasar cualquier punto del intervalo [- PI , PI] al [0,1], tenemos que hacer la siguiente operacion:

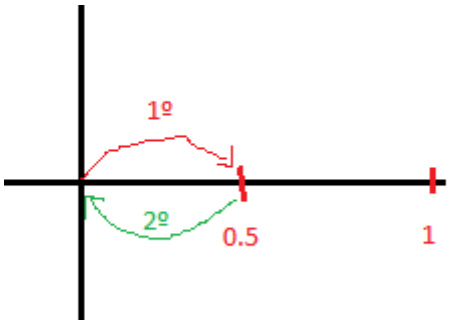
$$(X+ PI) / 2PI$$

CILINDRO.CC (LINEA 104)

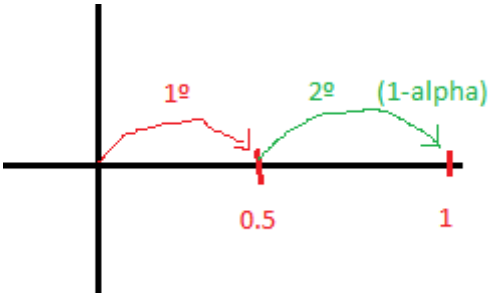
Originalmente los valores de Alpha van desde 0.5 hacia 0 y al llegar a este desde 1 hasta 0.5.

Por eso le hacemos 0.5 – Alpha, que hará que vaya de 0 a 0.5 el primer tramo y, al llegar al 2º que empieza en 1 y en decremento, sería 0.5 – 1 = -0.5 en adelante:

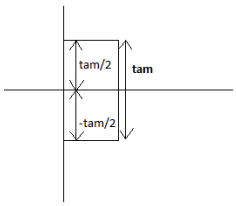
Y entonces, al hacer el valor absoluto, todos los valores quedan entre 0.5, la primera mitad creciente, y la segunda decreciente:



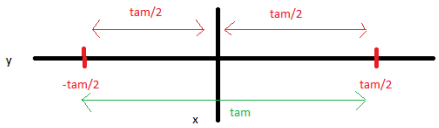
Por eso hay una condición que hace que a partir de la mitad de los puntos (que empiezan a decrecer), el valor empieza a ser 1-alpha, provocando que continúe la cadena creciente a partir de 0.5 como iba en la primera mitad:



CILINDRO.CC (LINEA 113)



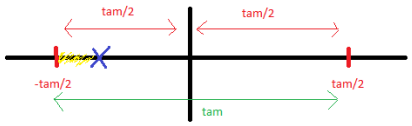
Si queremos que nuestro limite inferior (actualmente -tam/2) sea el limite inferior del nuevo intervalo (0) planteamos la siguiente situacion:



La longitud de un intervalo = lim. sup - lim. inf =

= tam/2 - (-tam/2) = tam/2 + tam/2 = tam

Si suponemos que tam es el total del intervalo, podemos considerarlo como un 1 sobre 1. Para pasar cualquier número de ese intervalo, a uno de [0,1] vemos qué distancia hay hasta el limite inferior y hacemos una regla de 3. Por ejemplo:



El punto azul puede estar situado en cualquier punto, pero la parte de la barra que está de color amarillo mide X - (-tam/2), que es lo que ocupa del total (tam). Si hacemos una regla de 3 dando por hecho que ahora el valor de tam es 1, queda asi:

X- (-tam/2) ----------> tam

?----------->1

Si despejamos ? = (X+tam/2)/tam

Es decir, como hemos visto, para pasar cualquier punto del intervalo [-tam/2, tam/2] al [0,1], tenemos que hacer la siguiente operacion:

(X+tam/2)/tam

CUBE.CC

CONSTRUCTOR PARA QUE SE APLIQUEN LAS TEXTURAS:

```
//  Vertices.resize(24);

//  Vertices[0]= _vertex3f(-0.5,-0.5,0.5);
//  Vertices[8]= _vertex3f(-0.5,-0.5,0.5);
//  Vertices[16]= _vertex3f(-0.5,-0.5,0.5);

//  Vertices[1]= _vertex3f(0.5,-0.5,0.5);
//  Vertices[9]= _vertex3f(0.5,-0.5,0.5);
//  Vertices[17]= _vertex3f(0.5,-0.5,0.5);

//  Vertices[2]= _vertex3f(0.5,0.5,0.5);
//  Vertices[10]= _vertex3f(0.5,0.5,0.5);
//  Vertices[18]= _vertex3f(0.5,0.5,0.5);

//  Vertices[3]= _vertex3f(-0.5,0.5,0.5);
//  Vertices[11]= _vertex3f(-0.5,0.5,0.5);
//  Vertices[19]= _vertex3f(-0.5,0.5,0.5);

//  Vertices[4]= _vertex3f(0.5,0.5,-0.5);
//  Vertices[12]= _vertex3f(0.5,0.5,-0.5);
//  Vertices[20]= _vertex3f(0.5,0.5,-0.5);

//  Vertices[5]= _vertex3f(-0.5,0.5,-0.5);
//  Vertices[13]= _vertex3f(-0.5,0.5,-0.5);
//  Vertices[21]= _vertex3f(-0.5,0.5,-0.5);

//  Vertices[6]= _vertex3f(-0.5,-0.5,-0.5);
//  Vertices[14]= _vertex3f(-0.5,-0.5,-0.5);
//  Vertices[22]= _vertex3f(-0.5,-0.5,-0.5);

//  Vertices[7]= _vertex3f(0.5,-0.5,-0.5);
//  Vertices[15]= _vertex3f(0.5,-0.5,-0.5);
//  Vertices[23]= _vertex3f(0.5,-0.5,-0.5);




//  Triangles.resize(12);

//  Triangles[0]=_vertex3ui(0,1,3);
//  Triangles[1]=_vertex3ui(1,2,3);

//  Triangles[2]=_vertex3ui(9,7,10);
//  Triangles[3]=_vertex3ui(7,4,10);

//  Triangles[4]=_vertex3ui(15,6,12);
//  Triangles[5]=_vertex3ui(6,5,12);

//  Triangles[6]=_vertex3ui(14,8,13);
//  Triangles[7]=_vertex3ui(8,11,13);

//  Triangles[8]=_vertex3ui(18,20,21);
//  Triangles[9]=_vertex3ui(18,21,19);

//  Triangles[10]=_vertex3ui(16,22,17);
//  Triangles[11]=_vertex3ui(17,22,23);

//  Textura.resize(24);
//  Textura[0]=_vertex2f(0.0f, 0.0f);
```

```
//  Textura[9]=_vertex2f(0.0f, 0.0f);
//  Textura[15]=_vertex2f(0.0f, 0.0f);
//  Textura[14]=_vertex2f(0.0f, 0.0f);
//  Textura[19]=_vertex2f(0.0f, 0.0f);
//  Textura[22]=_vertex2f(0.0f, 0.0f);

//  Textura[1]=_vertex2f(1.0f, 0.0f);
//  Textura[7]=_vertex2f(1.0f, 0.0f);
//  Textura[6]=_vertex2f(1.0f, 0.0f);
//  Textura[8]=_vertex2f(1.0f, 0.0f);
//  Textura[18]=_vertex2f(1.0f, 0.0f);
//  Textura[23]=_vertex2f(1.0f, 0.0f);

//  Textura[3]=_vertex2f(0.0f, 1.0f);
//  Textura[10]=_vertex2f(0.0f, 1.0f);
//  Textura[12]=_vertex2f(0.0f, 1.0f);
//  Textura[13]=_vertex2f(0.0f, 1.0f);
//  Textura[21]=_vertex2f(0.0f, 1.0f);
//  Textura[16]=_vertex2f(0.0f, 1.0f);

//  Textura[2]=_vertex2f(1.0f, 1.0f);
//  Textura[4]=_vertex2f(1.0f, 1.0f);
//  Textura[5]=_vertex2f(1.0f, 1.0f);
//  Textura[11]=_vertex2f(1.0f, 1.0f);
//  Textura[20]=_vertex2f(1.0f, 1.0f);
//   Textura[17]=_vertex2f(1.0f, 1.0f);
```

ESFERA.CC

ESFERA.CC (LINEA 59)

```
for(unsigned int i = 2*(num_div-tapas_totales); i < Triangles.size()-tapas_cant;i++)
```

Empieza en 2*(num_div-tapas) para evitar comprobar los primeros triangulos que usan los vértices del perfil correctamente. En la primera cara siembre hay num_div – 2 cuadrados, y cada cuadrado está formado por dos triángulos que utilizan los vértices del perfil. Para evitar comprobar ninguno de estos, empezamos justo por el siguiente a la primera cara.

Va hasta Triangles.size- tapas_cant para que no haga las iteraciones de las tapas, pues estas van en negro y no necesitan textura

```
if((int)Triangles[i]._0 < (num_div+1-tapaderas)) Triangles[i]._0 = Vertices.size() - tapas_totales - (num_div+1-tapaderas-Triangles[i]._0);

    if((int)Triangles[i]._1 < (num_div+1-tapaderas)) Triangles[i]._1 = Vertices.size() - tapas_totales - (num_div+1-tapaderas-Triangles[i]._1);

    if((int)Triangles[i]._2 < (num_div+1-tapaderas)) Triangles[i]._2 = Vertices.size() - tapas_totales - (num_div+1-tapaderas-Triangles[i]._2);
```

Se comprueba con los 3 if si alguno de los 3 vertices de los triángulos a comprobar pertenece al perfil. En este caso, significaría que los triángulos de la ultima cara (los cuales deben tener el final de la imagen) están unidos a vértices que contienen el inicio de la imagen, por lo que no se vería bien. De ser asi, el vertice que pertenece al perfil se cambia por otro diferente que contenga el final de la imagen y las mismas coordenadas que aquel que sustituye. Estos vértices nuevos se incluyen en el final del vector dentro de circular_sweeping (justo antes de las tapas). El primer vertice “duplicado” estará en Vertices.size() – num_tapas – (num_div-1) pero para cada punto hay que restarle los necesarios para encontrar su equivalente.

```
Vertices.size() - tapas_totales - (num_div+1-tapaderas-Triangles[i]._2)
```

Tapaderas -> tapaderas reales que tiene la imagen, independientemente de las que pida el usuario para mostrar

Tapas_totales -> tapaderas que quiere mostrar el usuario, independientemente de las que tenga la figura

Triangles[i]._2 -> Se resta el vector concreto para que cada vertice se cambie por su equivalente en posicion

ESFERA.CC (LINEA 94)

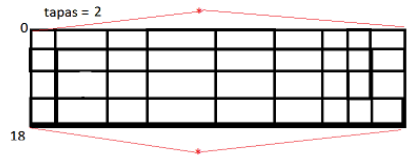
if((int)i>(((giros+1)*(num_div-1))/2)-num_div/2)

if((int)i > (((giros+1) * (num_div+1-tapaderas)) /2) - (num_div-tapaderas)/2)

- (giros+1) es la cantidad de lineas verticales de la figura
- (num_div+1-tapaderas) es la cantidad de lineas horizontales de la figura (mirar siguiente foto***)

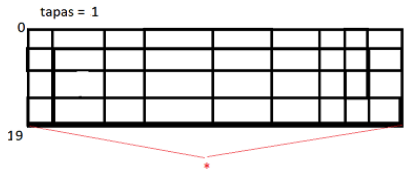
} su multiplicacion es la cantidad de puntos totales que vamos a calcular

- Como a partir de la mitad de los valores empieza a decrementar su valor y hacemos 1-coordx para que siga la continuidad, dividimos entre 2 para localizar el punto de la mitad (punto rojo figura derecha)
- Como cada linea vertical tiene la misma x para cualquier altura, necesitamos cambiar el punto desde el primero de la linea (arriba del todo, flecha roja en la figura derecha). Por ello, como el punto está a media altura, para situarnos en el primero subimos 1 punto más para situarnos en la flecha azul y así empezar por el que queremos (la roja) ---------> (num_div -tapaderas)/2



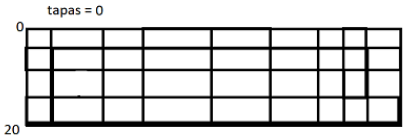
num_div=20
vertices_totales = 21
lineas_horizontales = 19
puntos = 95

giros = 4
lineas_verti = giros + 1 = 5
(21 totales - 2 que son tapas *)



num_div=20
vertices_totales = 21
lineas_horizontales = 20
puntos = 100

giros = 4
lineas_verti = giros + 1 = 5
(21 vertices totales - 1 tapadera *)



num_div=20
vertices_totales = 21
lineas_horizontales = 21
puntos = 105

giros = 4
lineas_verti = giros + 1 = 5
(21 totales - 0 tapas)

Como vemos, lineas_horizontales = vertices_totales - tapas = num_div + 1 - tapas

REV.CC

Rev.cc (línea 172)

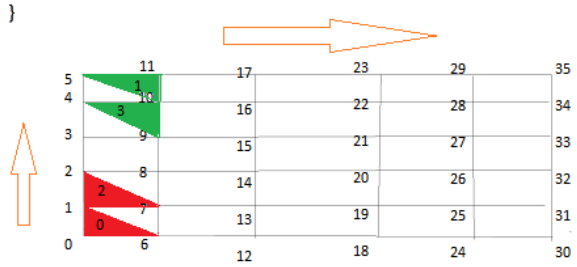
```
for( int i = 0; i < n; i++){

    for(int j = verti.size() - 1 ; j > 0 ; j--){

        Triangles.push_back(_vertex3ui(matriz[j][i],matriz[j][(i+1)%n],matriz[j-1][i]));

        Triangles.push_back(_vertex3ui(matriz[(verti.size()-1)-j][i],matriz[(verti.size()-1)-j][(i+1)%n],matriz[verti.size()-j][(i+1)%n]));

    }
}
```



Primera iteracion:
i = 0 j = 5

vertices del perfil -> 8 tapas -> 2
vertices para matriz (verti.size()) -> 6
rotaciones (n) -> 5

//CODIGO PARA PARTIR LA ESFERA POR LA MITAD Y CERRARLA(NO COMO CUENCO, SINO DE LADO)

```
Vertices.push_back(_vertex3f(0,0,0));


  for( unsigned int i = 0; i < verti.size()-1; i++){

    Triangles.push_back(_vertex3ui(Vertices.size()-1, matriz[i][0], matriz[(i+1)][0]));

    Triangles.push_back(_vertex3ui(Vertices.size()-1, matriz[i][n/2], matriz[i+1][n/2]));

  }


  Triangles.push_back(_vertex3ui(Vertices.size()-3, matriz[0][0], Vertices.size()-1));

  Triangles.push_back(_vertex3ui(Vertices.size()-3, matriz[0][n/2], Vertices.size()-1));

  Triangles.push_back(_vertex3ui(Vertices.size()-2, matriz[verti.size()-1][0], Vertices.size()-1));

  Triangles.push_back(_vertex3ui(Vertices.size()-2, matriz[verti.size()-1][n/2], Vertices.size()-
1));
```

OBJECT3D.CC

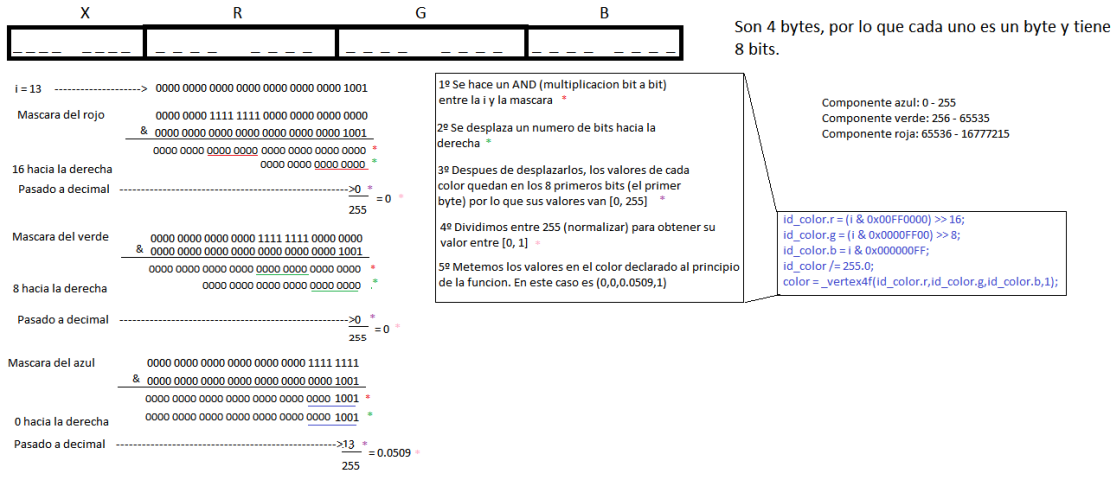
Object3d.cc (Linea 584 funcion draw_selection)

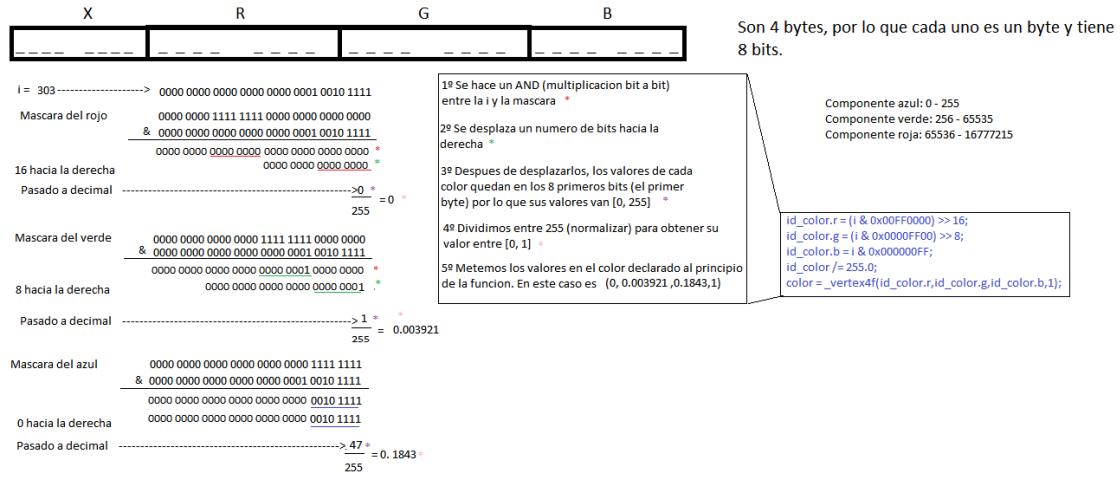
Los identificadores posibles serán del 0 al 16777215(16777216 pero quitado el blanco, como representante de ninguna selección)
debido a los 24 bits para representar las tonalidades RGB.

Componente azul: 0 - 255
Componente verde: 256 - 65535
Componente roja: 65536 - 16777215

Para convertir cada identificador a RGB bastaria con: Dividir el id entre 65535 siendo el cociente la componente roja,
el resto lo dividimos entre 256 quedando como cociente la componente verde y el resto la azul.

Tras esa parte solo queda normalizar el rgb obtenido.





Producto vectorial:

$$\vec{u} \times \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix} = \begin{vmatrix} u_y & u_z \\ v_y & v_z \end{vmatrix} \vec{i} - \begin{vmatrix} u_x & u_z \\ v_x & v_z \end{vmatrix} \vec{j} + \begin{vmatrix} u_x & u_y \\ v_x & v_y \end{vmatrix} \vec{k}$$

Un vector normal es aquel que tiene misma dirección y sentido y cuyo módulo es igual a la unidad. Si queremos normalizar un vector, basta con dividir cada componente por su módulo.

Es decir, dado el vector $\vec{u} = (u_1, u_2, u_3)$,

el vector $\vec{v}_{\vec{u}} = \frac{1}{|\vec{u}|} \cdot \vec{u}$ es un vector con mismo sentido y dirección.

$$|\vec{u}_{\vec{v}}| = \left(\frac{u_1}{|\vec{u}|}, \frac{u_2}{|\vec{u}|}, \frac{u_3}{|\vec{u}|} \right) = \left(\frac{u_1}{\sqrt{u_1^2 + u_2^2 + u_3^2}}, \frac{u_2}{\sqrt{u_1^2 + u_2^2 + u_3^2}}, \frac{u_3}{\sqrt{u_1^2 + u_2^2 + u_3^2}} \right)$$

es un vector normalizado en la dirección de $\vec{u}$.

GLWIDGET.CC

GLWIDGET.CC (LINEA 947)

Son 4 bytes, por lo que cada uno es un byte y tiene 8 bits.

Componente azul: 0 - 255
Componente verde: 256 - 65535
Componente roja: 65536 - 16777215

EJEMPLO LITTLE ENDIAN:
0x12345678 -> 0x78563412

(1º paso) Cuando la i = 303, lo que se pasa a Color era (0,0.003921,0.1843,0). Como cada valor ha sido dividido entre 255 antes, ahora lo multiplicamos por 255: (0, 1, 47, 0). Luego, cada uno a binario:

0 -> 0000 0000
1 -> 0000 0001
47 -> 0010 1111
0 -> 0000 0000

Color = 0000 0000 0000 0001 0010 1111 0000 0000 cuando i = 303

PERO COMO SE ALMACENA EN LITTLE ENDIAN, LO LEERA DE LA MEMORIA ASI:

(1ºpaso) 0000 0000 0010 1111 0000 0001 0000 0000

1º Se pasa color a numero binario (Con el little endian)
2º Hacer AND entre el color y mascara correspondiente
3º Hacer los desplazamientos necesarios para obtener el numero que queremos. *

4º Pasamos a decimal cada uno
5º Los sumamos

```
uint B = uint((Color & 0x00FF0000) >> 16);
uint G = uint((Color & 0x0000FF00);
uint R = uint((Color & 0x000000FF) << 16);

Selected_triangle = R + G + B;
```

Mascara del azul

0000 0000 1111 1111 0000 0000 0000 0000
& 0000 0000 0010 1111 0000 0001 0000 0000

(2ºpaso) 0000 0000 0010 1111 0000 0000 0000 0000
(3ºpaso) 0000 0000 0010 1111
(4ºpaso) 47

Mascara del verde

0000 0000 0000 0000 1111 1111 0000 0000
& 0000 0000 0010 1111 0000 0001 0000 0000

(2ºpaso) 0000 0000 0000 0000 0000 0001 0000 0000
(3ºpaso) 0000 0000 0000 0000 0000 0001 0000 0000
(4ºpaso) 256

Mascara del rojo

0000 0000 0000 0000 0000 0000 1111 1111
& 0000 0000 0010 1111 0000 0001 0000 0000

(2ºpaso) 0000 0000 0000 0000 0000 0000 0000 0000
(3ºpaso) 0000 0000 0000 0000 0000 0000 0000 0000
(4ºpaso) 0

(5ºpaso) Selected_triangle = R + G + B = 0 + 256 + 47 = 303

*
COMO LO QUEREMOS SEGUN EL GUION: (X, R, G, B)
COMO SE LO PASAMOS POR EL TEMA DEL COLOR: (R, G, B, X)
COMO LO LEEMOS POR EL LITTLE ENDIAN: (X, B, G, R)
PARA PASARLO DE LO QUE LEEMOS A LO QUE QUEREMOS:
(X, B, G, R)

>> 16 (DE LA COMPONENTE AZUL)
<< 16 (DE LA COMPONENTE ROJA)
NO DESPLAZAMOS PARA LA COMPONENTE VERDE