02/09/2016

Efectos

Seminario ISGI (S9)

R. Vivó

R. Vivó

Efectos

Seminario ISGI (S9)

OpenGL nos permite conseguir algunos efectos en nuestra imagen final sin necesidad de recurrir a algoritmos sofisticados. Conseguir objetos traslúcidos, suavizar los bordes de las primitivas o introducir niebla en la imagen aumentan la calidad y realismo de la imagen final. En este seminario se explica cómo mezclar colores, qué hacer para que los bordes de las primitivas no salgan escalonados y cómo hacer presente el color de las partículas suspendidas en el aire.

Mezcla -*Blending*-

La mezcla de colores o *blending* consiste en la combinación del color de un fragmento que se va a dibujar con el que ya existe en el *buffer* de color. Si imaginamos que tenemos un objeto, por ejemplo de color rojo, ya dibujado en el *buffer* de color y queremos dibujar encima otro algo transparente de color azul, por ejemplo, es de esperar que, en los pixeles donde coinciden, el color sea combinación de los anteriores y salga con un tono a magenta.

Este tipo de transparencia es un artificio muy flexible, aunque no tiene que confundirse con la forma en la que los objetos reales dejan pasar la luz, pues realmente se producen efectos de refracción que aquí no se tienen en cuenta.

Para hacer la mezcla se usa el "**canal alfa**", es decir, la cuarta componente de color (A) como índice de opacidad. Por ejemplo, si decidimos que el color RGBA de un objeto es (1.0,0.0,0.0,0.4) podríamos estar queriendo indicar que es transparente en un 60%. Así, cuando dibujamos el objeto, debería combinarse su color -el rojo en este caso- con el color que haya en el *buffer* ya, en la proporción 40% y 60% respectivamente.

Para activar el *blending* en OpenGL usaremos la instrucción:

```
glEnable(GL_BLEND);
```

y la contraria `glDisable` para desactivarlo.

La forma en la que se combina el color nuevo (el del fragmento que vamos a dibujar) con el viejo (el fragmento correspondiente ya dibujado) viene dada por una **función de mezcla**. Si $(RGBA)_N$ es el color nuevo y $(RGBA)_V$ es el viejo, la función de mezcla más simple es:

$$(R,G,B,A) = \propto (R,G,B,A)_N + (1-\propto)(R,G,B,A)_V$$

El factor de peso α se debe indicar mediante la instrucción:

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
```

donde `SRC_ALPHA` significa precisamente A_N, por tanto, el primer parámetro es el peso del color nuevo y el segundo el del viejo.

OpenGL permite indicar otros valores para α, incluso diferentes para cada componente de color, pero el efecto no es tan predecible y son menos usados. Sí se usa el valor de a=1 para que, independientemente de la A, el color se dibuje opaco. Esto se consigue así:

```
glBlendFunc(GL_ONE, GL_ZERO);
```

Ahora imaginemos que tenemos una escena con objetos opacos y traslúcidos. ¡El orden en el que los dibujemos es determinante! Por ejemplo, si un objeto traslúcido está delante de uno opaco y el traslúcido se dibuja primero, no se mezclará correctamente porque, al momento de dibujarlo, el opaco no está todavía en el buffer de color. ¿Cómo podemos subsanar esto si, a priori, no sabemos qué objetos están delante y cuáles detrás? La mejor forma de hacerlo es la siguiente:

1. Dibujar primero todos los objetos opacos con el *buffer* de profundidad en marcha
2. Habilitar el *blending*
3. Proteger el *buffer* de profundidad contra escritura
4. Dibujar los objetos traslúcidos
5. Habilitar la escritura en el *buffer* de profundidad
6. Deshabilitar el *blending*

Para proteger el buffer de profundidad contra escritura se llama a la función:

```
glDepthMask(GL_FALSE)
```

y para volver a habilitar con `GL_TRUE`. Siguiendo estos pasos nos aseguramos de que, cuando se van a dibujar los objetos traslúcidos -en el orden que sea-, el color en el *buffer* de color es el del objeto opaco más cercano y que ningún objeto traslucido dejará de dibujarse por estar detrás de otro.

NOTA: Cuando dibujemos los objetos traslúcidos hemos de tener cuidado en habilitar `GL_CULL_FACE` indicando que queremos eliminar las caras traseras con `glCullFace(GL_BACK)` pues el Z-Buffer está deshabilitado.

Ejercicio S9E01: Dibujar dos esferas gravitando una alrededor de la otra. La que gira tiene un color (0.3,0.3,0.9,0.4) y la fija un color (0.7,0.7,0.2,1.0). Utilizar como pesos en la función de mezcla GL_SRC_ALPHA,GL_ONE.

Medio participante

Cuando generamos imágenes sintéticas aparecen demasiado nítidas pues no tenemos en cuenta que el medio donde se encuentran -el aire por ejemplo- puede tener partículas en suspensión. Las partículas hacen que, conforme el objeto esté más alejado de nosotros, aparezca menos nítido llegando incluso a desaparecer. Es lo que sucede cuando hay niebla, humo o polución en el ambiente.

En OpenGL es muy fácil simular este efecto pues no es ni más ni menos que una mezcla entre el color del medio y el del objeto. Los pesos de la mezcla se ajustan dependiendo de la lejanía del objeto, es decir, el valor en el buffer de profundidad. Para activar este efecto basta con ordenar lo siguiente:

```
glEnable(GL_FOG)
```

También nos interesa decir cuál es el color de medio. Se hace así:

```
glFogfv(GL_FOG_COLOR, color_niebla)
```

donde `color_niebla` es un vector de 4 `GLfloat` que indica el valor RGBA del medio.

Es posible controlar la densidad de la niebla indicando un valor entre 0 (sin niebla) y 1 (muy densa). Para ello usamos la orden:

```
glFogf(GL_FOG_DENSITY, densidad)
```

La figura muestra el porcentaje de color original para diferentes valores de densidad.

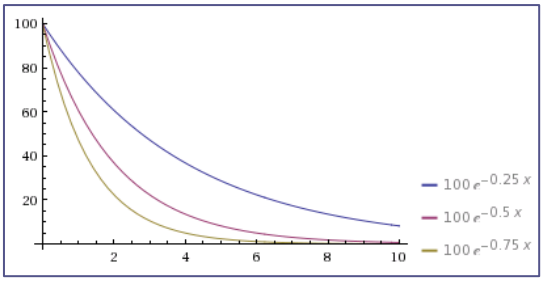


Figura 1. Influencia (%) del color original según la distancia al observador para densidades 0.25, 0.5 y 0.75

Ejercicio S8E02: Dibujar una escena compuesta de objetos en una retícula que se aleje del observador con el efecto de niebla.

Antialiasing

El problema del **aliasing** se da siempre que muestreamos una señal continua reduciéndola a una representación discreta. En gráficos esto sucede porque el área de un pixel no está totalmente cubierta por la proyección del objeto que le da el color, por lo que una parte es del color del objeto y otra no. Sin embargo, cuando decidimos poner el pixel a un determinado color solo se tiene en cuenta el color de su centro por lo que se ignoran áreas del píxel posiblemente a otros colores (en la figura 2 se ignora el área roja asignado al píxel el color azul).

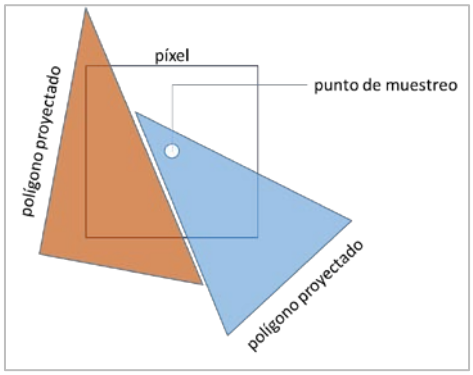


Figura 2. Aliasing dentro del píxel

Las técnicas para evitar el aliasing, o técnicas de **antialiasing**, más sencillas son las que hacen un sobremuestreo. El sobremuestreo consiste en muestrear más de un punto dentro del píxel y promediar los valores de color obtenidos para decidir el color final.

¿Cómo podemos en OpenGL hacer un sobremuestreo? La respuesta es simple: movemos ligeramente los objetos -o la cámara que los mira- para que el centro del píxel caiga cada vez en un sitio diferente. El movimiento tiene que ser tan ligero que el desplazamiento del centro no debe salirse del área inicial del pixel. Con cada desplazamiento generamos una imagen, que nos guardamos, y después las sumamos todas y las dividimos por su número.

Esta sencilla idea es relativamente fácil de aplicar en proyección ortográfica, caso al que nos vamos a ceñir. En primer lugar, ¿cuánto debemos desplazar los objetos? Imaginemos que tenemos una ventana del mundo real de dimensiones *wancho* x *walto* y un marco de pixeles -*viewport*- de dimensiones *ancho* x *alto* con la misma razón de aspecto. La medida del lado de un pixel en el mundo real será:

$$d = \frac{wancho}{ancho} = \frac{walto}{alto}$$

El desplazamiento deberá ser entonces menor que d/2 en cualquiera de las direcciones arriba, abajo, izquierda o derecha. Por ejemplo, si queremos tomar cuatro imágenes, podemos tomar como vector de desplazamiento normalizado *(vdx,vdy)* el siguiente:

```
GLfloat vdx[]={-0.125,-0.375,0.375,0.125};
GLfloat vdy[]={-0.25,0.25,-0.25,0.25};
```

El desplazamiento a aplicar en ***X*** e ***Y*** para la imagen *i* será entonces *(vdx[i]*d,vdy[i]*d)*.

Pero, ¿dónde guardamos las imágenes? OpenGL nos proporciona un *buffer* extra, llamado de **acumulación**, donde poder ir acumulando las imágenes que vamos calculando para después copiarlo al *buffer* de color. Hay que indicarle a OpenGL que vamos a usar este *buffer*:

```
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH|GLUT_ACCUM)
```

Para borrar el *buffer* de acumulación:

```
glClear(GL_ACCUM_BUFFER_BIT);
```

Para acumular una imagen nueva al buffer de acumulación se usa:

```
glAccum(GL_ACCUM,peso)
```

donde `peso` es el peso individual de cada imagen en el resultado final. En nuestro ejemplo, como hay cuatro imágenes, será ¼.

Para copiar la imagen del *buffer* de acumulación al *buffer* frontal -el que vemos en pantalla- usamos la orden:

```
glAccum(GL_RETURN,1)
```

El algoritmo de dibujo será el siguiente:

algoritmo display
    calcular d
    limpiar el buffer de acumulación
    para i=1 hasta numero_imagenes
        limpiar buffers de color y profundidad
        *trasladar (vdx[i]*d,vdy[i]*d)*
        dibujar objetos
        acumular imagen con peso 1/numero_imagenes
    fin para
    cargar imagen acumulada en pantalla

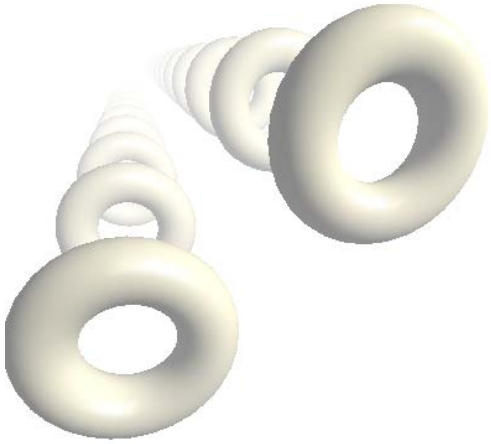
Ejercicio S9E03: Calcular una imagen con y sin antialiasing y comparar los resultados.

Ejercicio S9E01: Dibujar dos esferas gravitando una alrededor de la otra. La que gira tiene un color (0.3,0.3,0.9,0.4) y la fija un color (0.7,0.7,0.2,1.0). Utilizar como pesos en la función de mezcla GL_SRC_ALPHA,GL_ONE.

```
/*************************************************
ISGI::Blending
Roberto Vivo', 2013 (v1.0)

Dibuja dos esferas una de ellas traslucida

Dependencias:
+GLUT +glext +FreeGlut
*************************************************/
#define PROYECTO "ISGI::S9E01::Blending"

#include <iostream>                     // Biblioteca de entrada salida
#include <gl\freeglut.h>                // Biblioteca grafica
#include <GL/glext.h>                   // Biblioteca de extensiones de GL
#include <freeimage/FreeImage.h>        // Biblioteca de gestion de imagenes
using namespace std;

//Variables globales

static int xantes,yantes;               // Valor del pixel anterior
static float girox=0,giroy=0;           // Valor del giro a acumular
static float escalado=1;                // Valor del escalado acumulado
static enum Interaccion {GIRO,ESCALADO,ANIMACION} accion;        // Tipo de acción
static  GLuint tex[1];                  // Ids de texturas
static float orbita(0);

void loadLight()
{
        //Luces
        glEnable(GL_LIGHT0);

        //Materiales
        GLfloat mat_specular[] = {1,1,1,1.0};                           //Ks
        GLfloat mat_shininess[] = {100.0};                              //n
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

        glEnable(GL_COLOR_MATERIAL);
        glColorMaterial(GL_FRONT,GL_DIFFUSE);                           // Define la kd en glColor

        //Habilita la iluminación
        glEnable(GL_LIGHTING);
        glEnable(GL_RESCALE_NORMAL);
}

void init()
// Funcion propia de inicializacion
{
        // Mensajes por consola
        cout << PROYECTO << " running" << endl;                 // Mensaje por consola
        cout << "Version: OpenGL " <<  glGetString(GL_VERSION) << endl;

        cout << "Arrastre con boton izquierdo: Gira la pieza" << endl;
        cout << "Arrastre con boton derecho: Aumenta o disminuye" << endl;
        cout << "S: Captura la ventana en captura.png" << endl;
        cout << "Barra espacio: arranca la animacion" << endl;

        glClearColor(1.0,1.0,1.0,1.0);                          // Color de fondo a blanco
        glEnable(GL_DEPTH_TEST);                                // Habilita visibilidad
        glEnable(GL_CULL_FACE);
        glCullFace(GL_BACK);                                    // Elimina caras traseras

        loadLight();
}

void display()
// Funcion de atencion al dibujo
```

```
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);// Borra la pantalla

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        gluLookAt(0,0,4,0,0,0,0,1,0);                           // Situa la camara

        glRotatef(girox,1,0,0);                                 // Giro en x
        glRotatef(giroy,0,1,0);                                 // Giro en y
        glScalef(escalado,escalado,escalado);                   // Escalado

        // Posicion de la luz
        GLfloat posL[] = {3.0,3.0,3.0,1.0};
        glLightfv(GL_LIGHT0, GL_POSITION, posL);

        // Dibujar de opacos
        glPushMatrix();
        glColor4f(0.7,0.7,0.2,1.0);
        glutSolidSphere(1.5,20,20);
        glPopMatrix();

        // Habilitamos blending
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        // Z-Buffer Readonly
        glDepthMask(GL_FALSE);

        // Dibujar traslucidos
        glPushMatrix();
        glRotatef(orbita,0,1,0);
        glTranslatef(2,0,0);
        glColor4f(0.3,0.3,0.9,0.4);
        glutSolidSphere(0.5,20,20);
        glPopMatrix();

        // Z-Buffer a estado normal
        glDepthMask(GL_TRUE);

        glutSwapBuffers();                                      // Intercambia los buffers
}

void onIdle()
// Funcion de atencion a la animacion
{
        //Gira en y si ANIMACION
        static const float tic2deg=0.1;
        if(accion==ANIMACION){
                orbita+= tic2deg;
                glutPostRedisplay();
        }
}

void onKey(unsigned char tecla, int x, int y)
// Funcion de atencion al teclado
{
        switch(tecla){
        case 'S':                                               // Screenshot
                GLint vport[4];
                glGetIntegerv(GL_VIEWPORT, vport);              // Recupera el viewport corriente
                saveScreenshot("captura.png", vport[2], vport[3]); // CUIDADO. Viewports mayores de
520 dan error en heap
                break;
        case ' ':                                               // one en marcha la animacion
                accion = ANIMACION;
                glutPostRedisplay();
                break;
        case 27:                                                // Puso escape
                exit(0);
        }
}
```

Ejercicio S8E02: Dibujar una escena compuesta de objetos en una retícula que se aleje del observador con el efecto de niebla.



```
/*************************************************
ISGI::Fog
Roberto Vivo', 2013 (v1.0)

Dibuja un array de octaedros con niebla

Dependencias:
+GLUT +glext +FreeGlut
*************************************************/
#define PROYECTO "ISGI::S9E02::Fog"

…
static float giroz=0;           // Valor del giro para animacion

void display()
// Funcion de atencion al dibujo
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);// Borra la pantalla

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(0,0,2,0,0,0,0,1,0);                   // Situa la camara

        glRotatef(girox,1,0,0);                         // Giro en x
        glRotatef(giroy,0,1,0);                         // Giro en y

        // Posicion de la luz
        GLfloat posL[] = {3.0,3.0,3.0,1.0};
        glLightfv(GL_LIGHT0, GL_POSITION, posL);

        glRotatef(giroz,0,0,1);
        glScalef(escalado,escalado,escalado);           // Escalado

        // Habilitamos niebla
        glEnable(GL_FOG);
        static GLfloat cniebla[]={1.0,1.0,1.0,1.0};     // Color de la niebla
        glFogfv(GL_FOG_COLOR, cniebla);
        glFogf(GL_FOG_DENSITY, 0.3);

        glColor3f(0.9,0.86,0.67);
        // Dibuja un array de toros
        for(int i=0; i<2; i++)
                for(int j=0; j<20; j++){
                        glPushMatrix();
                        glTranslatef(2*i-1.0,0,-j*2);
                        glutSolidTorus(0.2,0.5,30,30);
                        glPopMatrix();
                };

        glutSwapBuffers();                              // Intercambia los buffers
}

void reshape(GLint w, GLint h)
// Funcion de atencion al redimensionamiento
{
…
        gluPerspective(60,razon,1,50);
}

void onIdle()
// Funcion de atencion a la animacion
{
        //Gira en y si ANIMACION
        static const float tic2deg=0.1;
        if(accion==ANIMACION){
                giroz+= tic2deg;
                glutPostRedisplay();
        }
}
```

Ejercicio S9E03: Calcular una imagen con y sin antialiasing y comparar los resultados.

```
/*************************************************
ISGI::Antialiasing
Roberto Vivo', 2013 (v1.0)

Dibuja un toro y una tetera con antialiasing

Dependencias:
+GLUT +glext +FreeGlut
*************************************************/
#define PROYECTO "ISGI::S9E03::Antialiasing"

…
static GLfloat d;        // Medida de un pixel en WR

void display()
// Funcion de atencion al dibujo
{
        // Limpia el buffer de acumulacion para acumular las 4 imagenes
        glClear(GL_ACCUM_BUFFER_BIT);

        // Vectores de desplazamiento normalizados para mover la camara
        static const GLfloat vdx[]={-0.125,-0.375,0.375,0.125};
        static const GLfloat vdy[]={-0.25,0.25,-0.25,0.25};

        // Calcula cuatro imagenes diferentes moviendo ligeramente la camara
        for(int i=0; i<4; i++){
                glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);// Borra los buffers

                glMatrixMode(GL_MODELVIEW);
                glLoadIdentity();

                gluLookAt(vdx[i]*d,vdy[i]*d,2,vdx[i]*d,vdy[i]*d,0,0,1,0);        // Situa la camara

                glRotatef(girox,1,0,0);                                          // Giro en x
                glRotatef(giroy,0,1,0);                                          // Giro en y

                // Posicion de la luz
                GLfloat posL[] = {3.0,3.0,3.0,1.0};
                glLightfv(GL_LIGHT0, GL_POSITION, posL);

                glScalef(escalado,escalado,escalado);                            // Escalado

                // Dibuja un toro y una tetera
                glPushMatrix();
                glColor3f(0.9,0.86,0.67);
                glutSolidTorus(0.5,1.4,30,30);
                glColor3f(0.65,0.7,0.4);
                glutSolidTeapot(0.5);
                glPopMatrix();

                // Acumula en buffer de acumulacion
                glAccum(GL_ACCUM,0.25);
        }

        // Carga la imagen del accumulation al front buffe
        glAccum(GL_RETURN,1);

        glFinish();
}

void reshape(GLint w, GLint h)
// Funcion de atencion al redimensionamiento
{
                // Usamos toda el area de dibujo
        glViewport(0,0,w,h);

        // Definimos la camara (matriz de proyeccion)
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
```

```
        float razon = (float) w / h;

        /* CAMARA ORTOGRAFICA */
        // Ajustamos la vista a la dimenson más pequenya del viewport para
        // poder ver la totalidad de la ventana del mundo real (4x4). Tambien
        // calculamos lo que mide un pixel
        if(w<h){
                glOrtho(-2,2,-2/razon,2/razon, 0,100);
                d = 4.0 / w;
        }
        else {
                glOrtho(-2*razon,2*razon,-2,2, 0,100);
                d = 4.0 / h;
        }
}


void main(int argc, char** argv)
// Programa principal
{
…
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH|GLUT_ACCUM);       // Alta de buffers a usar
…
}
```