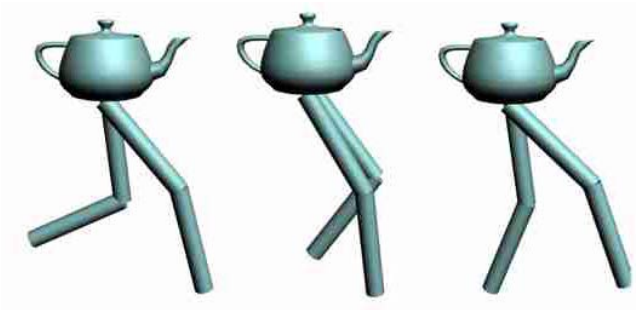
02/09/2016

Animación

Seminario ISGI (S5)



R. Vivó

R. Vivó

Animación

Seminario ISGI (S5)

En este seminario se introduce la variable tiempo. Los objetos y la cámara pueden variar su posición y características de un *frame* a otro produciendo una imagen animada. El seminario explica cómo hacer las variables dependientes del tiempo, cómo controlar los *frames* por segundo y cómo hacer la animación independiente de la velocidad de proceso del computador.

El evento idle

El método que utiliza la librería GLUT para conectar OpenGL con los eventos (de usuario y de sistema) es el registro de funciones callbacks que se disparan cuando llega el turno de atender al evento para el que fueron registradas. Los eventos que atiende la GLUT son los de redibujo, redimensión de la ventana, eventos de teclado y ratón y, el que usaremos en este seminario, el evento *idle*.

El evento *idle* se produce cuando la cola de eventos está vacía, es decir, cuando se han procesado todos los eventos pendientes y el gestor no tiene nada que hacer. Esto sucede, por ejemplo, cuando se ha completado la fase de dibujo (evento de *display*). Podemos aprovechar este suceso para:

1. Capturar el evento *idle* y lanzar la *callback* de atención
2. Variar algunos parámetros del dibujo en la *callback* como, por ejemplo, la posición de la cámara
3. Encolar un evento de redibujo para que se renueve el dibujo con los nuevos valores

Esta manera de proceder produce animaciones sin la intervención del usuario, es decir, sin necesidad de interacción. Veamos como ejecutar cada paso.

Para capturar el evento *idle* necesitamos registrar una *callback* de atención al inicio de nuestra aplicación. Para ello usaremos la función de GLUT:

```
glutIdleFunc(onIndle)
```

donde `onIdle` es el nombre de la *callback* de atención que debemos construir. La función `onIdle()` no lleva parámetros así que la declararemos así:

```
void onIdle(){…}
```

En el cuerpo de la función variaremos a voluntad cualquier característica del dibujo cuyo cambio será evidente en el próximo *frame* de la animación. Podemos variar, por ejemplo, la posición de la cámara -matriz de la vista-, la posición y orientación de los objetos -matriz del modelo-, el color de dibujo, etc.

La última orden dentro del cuerpo de la función `onIdle()` debe ser la de encolar un evento de redibujo para que se borre lo que hay en pantalla y se dibuje otra vez con los valores nuevos. Para ello existe una función en GLUT:

```
glutPostRedisplay()
```

En evento encolado se procesará inmediatamente después de que la función `onIdle` devuelva el control al gestor de la cola.

Ejercicio S5E01: Dibujar un cuadrado que se vaya moviendo y rebote cuando llegue a los límites de la ventana.

Doble buffer

Cuando la animación es muy rápida, hay muchos objetos a dibujar o la tarjeta gráfica es lenta pueden aparecer efectos de parpadeo en la pantalla. Este efecto desagradable se produce porque se dibuja sobre el mismo buffer que se está viendo en pantalla.

Para solucionar este problema se utiliza la técnica del doble buffer. Consiste en dibujar en una zona de memoria que representa una pantalla que no se ve y, cuando el dibujo ha concluido, copiar a la pantalla que vemos el dibujo como una imagen de pixeles. Así, el tiempo de redibujo subjetivo -el que nosotros vemos-, es constante y muy rápido como corresponde a una transferencia de bits en memoria o al cambio de dirección de un puntero. OpenGL mantiene, entre otros, dos *buffers* de dibujo: el *buffer* delantero `GL_FRONT` y el *buffer* trasero `GL_BACK`. El trasero es donde se dibuja y el delantero es el que se muestra, cuando está activada la técnica del doble buffer.

Para activar el doble *buffer* hay que informar de ello en la orden de GLUT:

```
glutInitDisplayMode(GL_DOUBLE|…)
```

`GL_DOUBLE` substituye a la constante `GL_SINGLE` habitual. Con ello, cuando dibujemos lo haremos sobre el *buffer* trasero, no sobre el que estamos viendo. Por ello si queremos ver el dibujo hay que pasar el contenido del *buffer* trasero al delantero justo cuando hayamos acabado de dibujar. Usaremos la orden de GLUT:

```
glutSwapBuffers()
```

Esta función debe ser la última orden de la función que atiende al evento de dibujo -*display*-. Internamente hace un `glFlush()` por lo que no es necesario hacerlo en la función de *display*.

Control de la velocidad de la animación

La velocidad de la animación es la relación entre el movimiento de los objetos o la cámara en la escena y el tiempo real transcurrido. La velocidad de la animación no se controla en las condiciones descritas hasta aquí, pues no se ha tenido en cuenta la variable tiempo. El ciclo dibujo-actualización-dibujo irá más deprisa o más lento según la velocidad de nuestro computador y calidad de la tarjeta gráfica.

La forma de mantener una referencia de tiempo es fijando un origen y conocer en cada momento los milisegundos transcurridos desde entonces, lo mismo que se hace con un cronómetro. La librería GLUT permite conocer los milisegundos transcurridos desde la puesta en marcha -llamada a `glutInit`- de nuestra aplicación mediante la orden:

```
glutGet(GLUT_ELAPSED_TIME)
```

Por tanto sabemos el tiempo transcurrido entre dos actualizaciones de la escena en la función `onIdle`:

```
tiempo_transcurrido = hora_actual - hora_anterior;
hora_anterior = hora_actual;
…
```

Lo que buscamos es que se mantenga la velocidad de la animación. Por tanto, el avance de la magnitud que estamos variando debe ser:

```
avance = velocidad_animacion * tiempo_transcurrido;
```

y si la variable que estamos actualizando tenía un valor `valor_anterior`, después de la actualización valdrá:

```
valor_actual = valor_anterior + avance;
valor_anterior = valor_actual;
```

Ejercicio S5E02: Hacer un travelling de la escena moviendo la cámara en una circunferencia con centro el origen sobre el plano XZ con una velocidad de 24°/sg.

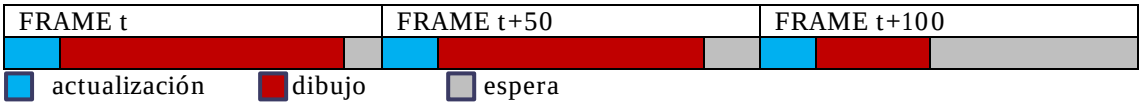
Control de FPS

Los FPS o fotogramas por segundo son las veces que se ejecuta `display()` en un segundo.

Debido a la persistencia de la visión, el ojo no es capaz de discriminar como fotogramas independientes dos que suceden en un intervalo menor que, aproximadamente, 30 msg. Es decir, con un ratio mayor de 50 FPS (20 msg/fotograma) es suficiente para percibir una sucesión de fotogramas como una animación continua. Este es el fundamento del cine o la televisión para ofrecer imágenes en movimiento.

Es evidente que cuanto mayor sean los FPS mayor es la carga a la que sometemos al computador. ¿Podemos entonces controlar los FPS para que se mantengan a una tasa constante de, digamos por ejemplo, 60? La respuesta es sí, siempre que el tiempo entre dos ejecuciones de `display`, sin hacer el control, sea menor que 16,7 msg. Es decir, el control de FPS sólo sirve para aquellas máquinas potentes que dan una tasa de FPS mayor que la queremos fijar.

El control de FPS significa pues que el ciclo actualización-dibujo se da a una tasa de FPS constante. Si los FPS son 50 por ejemplo, podemos representar los ciclos en una barra de tiempo:



La fase de actualización incluye la captura y gestión de los eventos. Esta fase suele mantener una duración más o menos constante. La fase de dibujo, sin embargo, es muy dependiente de la cantidad de objetos a dibujar que depende de, primero, el número de polígonos de la escena y, segundo, de cuántos ve la cámara en cada momento. Por eso se ha dibujado con diferente duración para cada fotograma. El resto del tiempo hasta que comience el nuevo ciclo se queda en espera.

La forma de despertar un nuevo ciclo es poniendo una alarma -*timer*-. La librería GLUT permite el registro de una *callback* que atienda a un evento -*timer*- periódico:

```
glutTimerFunc(milisegundos, onTimer, valor)
```

donde `milisegundos` es un entero sin signo que da cuenta del periodo, `onTimer` es el nombre de la *callback* que atenderá el evento y `valor` es un número entero que se pasará a la *callback* cuando se la llame. Por tanto, la *callback* debe definirse así:

```
void onTimer(int valor)
```

GLUT intentará llamar a la *callback* tan pronto como el periodo se haya cumplido pero si, por ejemplo, el dibujo dura mucho, no se cumplirá para ese ciclo la restricción de tiempo. Con este método podemos asegurar, por tanto, que la tasa de FPS no subirá por encima del valor prefijado, pero no al revés.

Es posible registrar más de una función de atención a *timers* con el mismo o diferente periodo.

La función `onTimer()` sustituye a la función `onIdle()`, pues el comienzo del ciclo ya no se produce cuando acaba la fase de dibujo sino a un periodo preestablecido.

Ejercicio S5E03: Repetir el ejercicio anterior manteniendo una tasa de 40 FPS. Mostrar en ambos casos los FPS en la barra de título de la ventana.

Ejercicio S5E01: Dibujar un cuadrado que se vaya moviendo y rebote cuando llegue a los límites de la ventana.

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

Anima un cuadrado en pantalla que rebota al llegar a
los limites de la ventana sin control de tiempos

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S5E01::Cuadrado rebotante"

#include <iostream>                             // Biblioteca de entrada salida
#include <cmath>                                // Biblioteca matematica de C
#include <gl\freeglut.h>                        // Biblioteca grafica
using namespace std;

static float posicionx(0.0), posiciony(0.0);    // Posicion del cuadrado
static GLuint cuadrado;                         // Id del cuadrado
static const GLfloat semilado(0.1);             // Tamanyo del cuadrado
static GLfloat semiancho, semialto;             // Tamaño actual de la ventana del MR


void init()
// Funcion propia de inicializacion
{
        cout << "Version: OpenGL " <<  glGetString(GL_VERSION) << endl;

        glClearColor(1.0,1.0,1.0,1.0);          // Color de fondo a blanco
        glColor3f(0,0,0);                       // Color de dibujo a negro

        // Crea un cuadrado de lado x lado
        cuadrado = glGenLists(1);
        glNewList(cuadrado,GL_COMPILE);
        glBegin(GL_POLYGON);
          glVertex3f(-semilado,-semilado,0.0);
          glVertex3f(semilado,-semilado,0.0);
          glVertex3f(semilado,semilado,0.0);
          glVertex3f(-semilado,semilado,0.0);
        glEnd();
        glEndList();
}

void display()
// Funcion de atencion al dibujo
{
  glClear(GL_COLOR_BUFFER_BIT);                 // Borra la pantalla
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  gluLookAt(0,0,5,0,0,0,0,1,0);                 // Situa la camara

  glTranslatef(posicionx, posiciony, 0);        // Situa al cuadrado
  glCallList(cuadrado);                         // y lo dibuja

  glFlush();                                    // Finaliza el dibujo
}

void reshape(GLint w, GLint h)
// Funcion de atencion al redimensionamiento
{
        // Tamanyo objetivo de la ventana del mundo real
        static const GLfloat ANCHO(2.0),ALTO(2.0);

        // 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 (2x2)
        if(w<h){
                semiancho = ANCHO/2;
                semialto = ALTO/2/razon;
        }
        else{
                semiancho = ANCHO*razon/2;
                semialto = ALTO/2;
        }
        glOrtho(-semiancho,semiancho, -semialto,semialto, 0,10);
}

void onIdle()
// Funcion de atencion al evento idle
{
        // Vector velocidad del cuadrado en unidades/frame en ambas direcciones
        static float uperframe[] = {0.001,0.001};

        // posicion = posicion anterior + velocidad x (1 frame)
        posicionx+= uperframe[0];
        posiciony+= uperframe[1];

        // La ventana mide ancho x alto
        // Detección de colisiones
        if(posicionx+semilado>=semiancho || posicionx-semilado<=-semiancho) uperframe[0] *= -1;
        if(posiciony+semilado>=semialto  || posiciony-semilado<=-semialto)  uperframe[1] *= -1;

        // Puede suceder que al variar el marco, la posicion actual quede fuera de la ventana
        // con lo que el cuadrado desaparece.
        // Se puede mejorar, pero por simplicidad escogemos hacer un reset al centro

        if(abs(posicionx)>semiancho || abs(posiciony)>semialto){
                posicionx = 0.0;
                posiciony = 0.0;
        };

        glutPostRedisplay();
}
void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                        // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);    // Alta de buffers a usar
  glutInitWindowSize(400,400);                  // Tamanyo inicial de la ventana
  glutCreateWindow(PROYECTO);                   // Creacion de la ventana con su titulo
  scout << PROYECTO << " running" << endl;      // Mensaje por consola
  glutDisplayFunc(display);                     // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                     // Alta de la funcion de atencion a reshape
  glutIdleFunc(onIdle);                         // Alta de la funcion de atencion a idle
  init();                                       // Inicializacion propia
  glutMainLoop();                               // Puesta en marcha del programa
}
```

Ejercicio S5E02: Hacer un travelling de la escena moviendo la cámara en una circunferencia con centro el origen sobre el plano XZ con una velocidad de 24º/sg.

```
/*************************************************
ISGI::Control del tiempo
Roberto Vivo', 2013 (v1.0)

Anima la camara en un travelling circular controlando
el tiempo entre frames para dar velocidad constante

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S5E02::Control del tiempo"

#include <iostream>                             // Biblioteca de entrada salida
#include <cmath>                                // Biblioteca matematica de C
#include <gl\freeglut.h>                        // Biblioteca grafica
using namespace std;

static const float radio = 5.0;                 // Radio de giro de la camara
static float angulo = 0.0;                      // Angulo de travelling inicial de la camara
static float ojo[] = {0,0,radio};               // Posicion inicial de la camara
static const float velocidad = 24.0*3.1415926/180;       // radianes/segundo

void init()
// Funcion propia de inicializacion
{
        cout << "Version: OpenGL " <<  glGetString(GL_VERSION) << endl;

        glClearColor(1.0,1.0,1.0,1.0);          // Color de fondo a blanco

        glEnable(GL_DEPTH_TEST);                // Habilita visibilidad
}

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

  gluLookAt(ojo[0],ojo[1],ojo[2],0,0,0,0,1,0);           // Situa la camara

  glColor3f(0,0,1);                                      // Color de dibujo a azul
  glutSolidTeapot(1.0);                                  // Dibuja una tetera en el origen

  glPushMatrix();
  glTranslatef(0,0,-2);
  glColor3f(1,0,0);                                      // Color de dibujo a rojo
  glutSolidTeapot(0.5);                                  // Dibuja una tetera detras
  glPopMatrix();

  glutSwapBuffers();                                     // Intercambia los buffers
}

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 PERSPECTIVA */
        gluPerspective(60,razon,1,10);
}
```

```
void onIdle()
// Funcion de atencion al evento idle
{
        //Calculamos el tiempo transcurrido desde la última vez
        static int antes=0;
        int ahora, tiempo_transcurrido;

        ahora= glutGet(GLUT_ELAPSED_TIME);              //Tiempo transcurrido desde el inicio
        tiempo_transcurrido= ahora - antes;             //Tiempo transcurrido desde antes en msg.

        // angulo = angulo anterior + velocidad x tiempo
        angulo += velocidad*tiempo_transcurrido/1000.0;
        ojo[0] = radio * sin(angulo);
        ojo[2] = radio * cos(angulo);

        antes = ahora;

        glutPostRedisplay();
}
void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH); // Alta de buffers a usar
  glutInitWindowSize(400,400);                          // Tamanyo inicial de la ventana
  glutCreateWindow(PROYECTO);                           // Creacion de la ventana con su titulo
  std::cout << PROYECTO << " running" << std::endl;     // Mensaje por consola
  glutDisplayFunc(display);                             // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                             // Alta de la funcion de atencion a reshape
  glutIdleFunc(onIdle);                                 // Alta de la funcion de atencion a idle
  init();                                               // Inicializacion propia
  glutMainLoop();                                       // Puesta en marcha del programa
}
```

Ejercicio S5E03: Repetir el ejercicio anterior manteniendo una tasa de 40 FPS. Mostrar en ambos casos los FPS en la barra de título de la ventana.

```
/*************************************************
ISGI::Control de los FPS
Roberto Vivo', 2013 (v1.0)

Anima la camara en un travelling circular controlando
la frecuencia de dibujo para mantener los FPS constantes
y manteneindo la velocidad de la animacion constante

Dependencias:
+GLUT
*************************************************/
#define PROYECTO "ISGI::S5E02::Control del tiempo"

#include <iostream>                                     // Biblioteca de entrada salida
#include <sstream>                                      // Biblioteca de manejo de strings
#include <cmath>                                        // Biblioteca matematica de C
#include <gl\freeglut.h>                                // Biblioteca grafica
using namespace std;

static const float radio = 5.0;                         // Radio de giro de la camara
static float angulo = 0.0;                              // Angulo de travelling inicial de la camara
static float ojo[] = {0,0,radio};                       // Posicion inicial de la camara
static const float velocidad = 24.0*3.1415926/180;      // radianes/segundo
static const int tasaFPS = 40;                          // tasa que se quiere como maximo

void muestraFPS()
// Calcula la frecuencia y la muestra en el título de la ventana
// cada segundo
{
        int ahora, tiempo_transcurrido;
        static int antes = 0;
        static int FPS = 0;
        stringstream titulo;

        //Cuenta de llamadas a esta función en el último segundo
        FPS++;
        ahora = glutGet(GLUT_ELAPSED_TIME);                     //Tiempo transcurrido desde el inicio
        tiempo_transcurrido = ahora - antes;                    //Tiempo transcurrido desde antes
        if(tiempo_transcurrido>1000){                           //Acaba de rebasar el segundo
                titulo << "FPS: " << FPS;                       //Se muestra una vez por segundo
                glutSetWindowTitle(titulo.str().c_str());
                antes = ahora;                                  //Ahora ya es antes
                FPS = 0;                                        //Reset del conteo
        }
}

void init()
// Funcion propia de inicializacion
{
        cout << "Version: OpenGL " <<  glGetString(GL_VERSION) << endl;
        glClearColor(1.0,1.0,1.0,1.0);                          // Color de fondo a blanco
        glEnable(GL_DEPTH_TEST);                                // Habilita visibilidad
}

void display()
// Funcion de atencion al dibujo
{
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);             // Borra la pantalla
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(ojo[0],ojo[1],ojo[2],0,0,0,0,1,0);                  // Situa la camara
  glColor3f(0,0,1);                                             // Color de dibujo a azul
  glutSolidTeapot(1.0);                                         // Dibuja una tetera en el origen
  glPushMatrix();
  glTranslatef(0,0,-2);
  glColor3f(1,0,0);                                             // Color de dibujo a rojo
  glutSolidTeapot(0.5);                                         // Dibuja una tetera detras
  glPopMatrix();
```

```
  muestraFPS();                                          // Hemos dibujado un frame

  glutSwapBuffers();                                     // Intercambia los buffers
}

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 PERSPECTIVA */
        gluPerspective(60,razon,1,10);
}

void onTimer(int valor)
// Funcion de atencion al timer periodico
{
        //Calculamos el tiempo transcurrido desde la última vez
        static int antes=0;
        int ahora, tiempo_transcurrido;

        ahora= glutGet(GLUT_ELAPSED_TIME);                    //Tiempo transcurrido desde el inicio
        tiempo_transcurrido= ahora - antes;                   //Tiempo transcurrido desde antes msg

        // angulo = angulo anterior + velocidad x tiempo
        angulo += velocidad*tiempo_transcurrido/1000.0;
        ojo[0] = radio * sin(angulo);
        ojo[2] = radio * cos(angulo);

        antes = ahora;

        glutTimerFunc(1000/tasaFPS,onTimer,tasaFPS);          // Se encola un nuevo timer

        glutPostRedisplay();                                  // Se manda el dibujo
}
void main(int argc, char** argv)
// Programa principal
{
  glutInit(&argc, argv);                                // Inicializacion de GLUT
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH); // Alta de buffers a usar
  glutInitWindowSize(400,400);                          // Tamanyo inicial de la ventana
  glutCreateWindow(PROYECTO);                           // Creacion de la ventana con su titulo
  std::cout << PROYECTO << " running" << std::endl;     // Mensaje por consola
  glutDisplayFunc(display);                             // Alta de la funcion de atencion a display
  glutReshapeFunc(reshape);                             // Alta de la funcion de atencion a reshape
  glutTimerFunc(1000/tasaFPS,onTimer,tasaFPS);          // Alta de la funcion de atencion al timer
  init();                                               // Inicializacion propia
  glutMainLoop();                                       // Puesta en marcha del programa
}
```