TEMA 4. INTERACCIÓN Y ANIMACIÓN

Objetivos:

- *Conocer los distintos métodos de interacción con la escena virtual*
- *Conocer los fundamentos de la animación por ordenador*
- *Conocer cómo implementa OpenGL los mecanismos de interacción y animación*

INTERACCIÓN

Un sistema gráfico interactivo es un sistema que responde de forma directa a las acciones del usuario. La incorporación de la interactividad permite realizar aplicaciones que convierten a los usuarios en elementos activos y creadores del sistema, como en:

- Sistemas CAD
- Videojuegos
- Simuladores (Ilustración 116)

Hoy en día, tan sólo los sistemas gráficos que realizan procesos muy costoso en tiempo de cálculo son no interactivos, encontrándose entre ellos los sistemas de generación de imágenes realistas mediante cálculo de radiosidades, o los motores de animación de las grandes productoras de cine.

Ilustración 116: Sistema interactivo (http://nxr.northwestern.edu/research/surface-haptic-interaction-design)

Sin embargo, la interacción está presente en la Informática Gráfica desde sus comienzos. De hecho, como vimos en el primer tema, los primeros hitos de la Informática Gráfica venían dados por la creación de dispositivos de interacción (el ratón, la tableta digitalizadora, etc.).

Un sistema gráfico interactivo debe contener al menos un dispositivo de entrada y uno de salida, por lo que al menos necesitaremos un teclado y un monitor, aunque hoy en día casi no podemos sobrevivir sin tener un ratón. A estos dispositivos podemos añadirles un diverso conjunto de dispositivos de entrada:

- Joystick
- Pantalla táctil
- Tableta digitalizadora
- Phantom hápticos

Y por supuesto, dispositivos de salida de lo más avanzados:

- Cuevas de realidad virtual
- Cascos de visualización inmersiva (p.ej. oculus Rift®)
- Gafas de Realidad aumentad (p.ej. Google Glass©)
- Pantallas estéreo
- Dispositivos móviles
- Hologramas

Para que un sistema sea realmente interactivo, debe haber un inapreciable espacio de tiempo entre la acción del usuario y la respuesta del sistema. Si se tarda mucho en dar la respuesta, el usuario pierde la noción de causa/efecto, y dejamos de tener un sistema interactivo.

En el caso de las animaciones, deben tener una frecuencia de respuesta inferior a 1/30 de segundo, para que el usuario no perciba saltos. La interacción mediante otros sentidos, como los dispositivos hápticos que permiten percibir respuestas del sistema interactivo mediante el tacto, el tiempo de respuesta ha de ser inferior a 1/1000 de segundo, menos de un microsegundo para calcular si se contacta o no con el objeto para que la interacción sea realista. Estos dispositivos soy muy usados, por ejemplo, en simuladores de cirugía como el de la Ilustración 117.

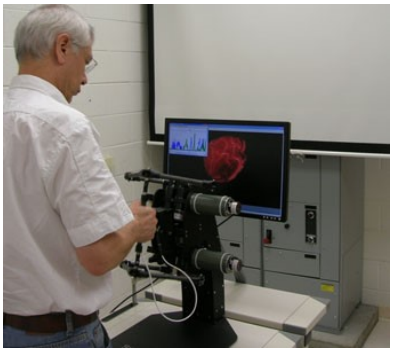


Ilustración 117 Interacción háptica

En general, el proceso de interacción se describe como en la Ilustración 118. El tiempo total del ciclo debe ser inferior a 0.03 segundos en aplicaciones interactivas no hápticas, e inferior a 0.001 segundos en aplicaciones hápticas.

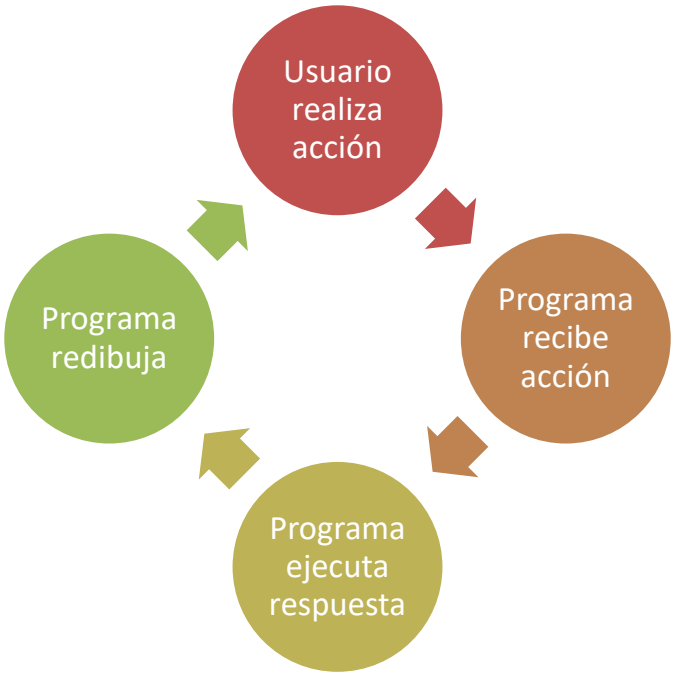


Ilustración 118 Flujo de interacción

A lo largo del proceso de interacción es conveniente proporcionar al usuario un mecanismo de ***realimentación***, que le permita conocer los pasos seguidos por el sistema, lo que el sistema ha entendido con sus gestos interactivos y cómo va a interpretar las acciones a realizar. Un sistema sin *realimentación* es muy poco usable. Entre las técnicas de realimentación encontramos:

- Mostrar el estado del sistema (p.ej. modo de cámara)
- Indicar la herramienta activa
- Resaltar elementos seleccionables
- Cambios de cursor según acción permitida

Todo sistema gráfico que se considere interactivo deberá permitir leer posiciones y seleccionar componentes delo modelo geométrico. La lectura de posiciones permite generar puntos en coordenadas del mundo, mientras que la selección permite identificar elementos de la escena. Por ejemplo:

- Si queremos editar un modelo geométrico, como modificar un perfil de revolución, es necesario por un lado seleccionar los puntos a modificar, y calcular la transformación que hay que realizar sobre esos puntos. Se puede realizar mediante entrada de teclado o de forma interactiva con un ratón, que es mucho más agradable.
- Para enfocar la cámara en un objeto con el ratón, es necesario hacer clic sobre él y saber que se ha pulsado sobre dicho objeto, de forma que sus coordenadas de mundo sean el vector *lookAt* de los parámetros de la cámara.

Toda esta interacción se puede realizar no sólo con los clásicos ratones y teclado, sino con cualquier otro tipo de dispositivos físicos avanzados (tabletas, lápices ópticos, pantallas táctiles, etc.), pero al final, lo que tenemos en cualquiera de los casos es una coordenada (x,y) de la pantalla. Esto no es más que una abstracción del **dispositivo hardware** a un **dispositivo lógico**.

Gracias a la abstracción de ***dispositivo lógico***, las librerías gráficas, y en concreto OpenGL, son independientes del dispositivo concreto de entrada, pues es irrelevante tanto para la aplicación como para el usuario. Hay dos tipos de dispositivos lógicos principales:

- ***Locator***, que lee posiciones (X,Y) de la pantalla. Su sistema de coordenadas es el de la pantalla, por lo que habrá que realizar la transformación inversa de la visualización para obtener la coordenada del mundo a la que se corresponde.
- ***Pick***, que lee identificadores de componentes del modelo geométrico. Es un dispositivo lógico que consulta la escena y devuelve el objeto sobre el que está el cursor.

Gestión de eventos

Los eventos son fundamentales en cualquier sistema interactivo. Los eventos son generados por las acciones del usuario o de forma automática por el sistema. Cada librería de interfaces de usuario dispone de su propio sistema gestor de eventos, podemos destacar:

- *Event Listeners*, en Java
- *Callbacks* en GLUT
- *Signal/Slot* en QT

Tanto los *event listeners* como los *callbacks* son gestores de eventos que se encargan de forma predefinida de unos eventos muy concretos, cada uno del suyo (p.ej. la pulsación de teclado o el movimiento del ratón). El modelo *signal/slot* implementado en QT permite que una misma señal, desencadenada por un evento, sea tratada por distintos gestores (*slot*), incluso de objetos distintos. Un *slot* puede ser cualquier método de un objeto.

Gestión de eventos en GLUT

GLUT gestiona las entradas en modo *evento*, esto es, tiene un buffer donde se van almacenando las peticiones realizadas por el usuario (consideremos *peticiones* al movimiento del ratón, la pulsación de una tecla, etc.). El gestor de eventos de *GLUT* va leyendo ese buffer y, si el evento tiene asociada una función que lo procesa (un *callback*), ejecuta dicha función y reacciona al evento recibido. Si no hay un callback para el evento, se ignora.

Los callback más usados en *glut* son:

- **glutDisplayFunc**: Redibujado.
- **glutMouseFunc**: Pulsación de botones del ratón.
- **glutMotionFunc**: Movimiento del ratón mientras se pulsa un botón.
- **glutPassiveMotionFunc**: Moviviento del ratón sin pulsar botones.

- **glutReshapeFunc**: Cambio de tamaño de la ventana.
- **glutKeyFunc**: Pulsación de tecla.
- **glutIdleFunc**: Ausencia de eventos externos
- **glutTimerFunc**: Temporizador.

Eventos de pulsación de botones del ratón

Si asociamos el callback con la función adecuada, llamada `raton`:

```
glutMouseFunc( raton );
```

siendo

```
void raton( GLint button, GLint state, GLint x, GLint y )
```

podemos conocer mucha información:

- `button` devuelve que botón se accionó:
  - `GLUT_LEFT_BUTTON`,
  - `GLUT_MIDDLE_BUTTON`,
  - `GLUT_RIGHT_BUTTON`
- `state` es si se ha pulsado o soltado:
  - `GLUT_UP`,
  - `GLUT_DOWN`
- `x` e `y` determinan la posición del cursor en coordenadas de pantalla

Una implementación posible sería la siguiente, donde se actualizan el valor de `xm` e `ym`:

```
void clickRaton( int boton, int estado, int x, int y ) {
        if (boton == GLUT_LEFT_BUTTON && estado == GLUT_DOWN) {
              xm= x;
              ym= y;
        glutPostRedisplay();
        }
}
```

EJERCICIOS

78. Documente en sus apuntes cómo se realiza la gestión de eventos de ratón en Java y QT

Combinación de eventos

Una operación muy habitual es pulsar el botón izquierdo del ratón, mantenerlo pulsado y mover el ratón y soltarlo cuando el objeto se ha movido hasta el lugar que queríamos. Esto implica varios eventos:

- pulsación del ratón, gestionado en GLUT con el callback definido en `glutMouseFunc`.
- movimiento del ratón con el botón pulsado, gestionado con `glutMotionFunc`
- liberación del botón pulsado, gestionado con `glutMouseFunc`

¿Cómo se llevan a cabo estas interacciones en la aplicación? Pues hay que tener **una máquina de estados**, que reaccione de forma distinta si se pulsa el botón derecho o el izquierdo, que haga unos

cálculos si se está moviendo el ratón con el botón principal del ratón pulsado o con el secundario, o ninguno.

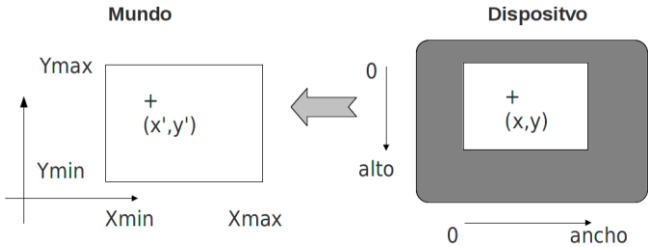
Posicionamiento

La posición que da el dispositivo lógico de posicionamiento en pantalla es una coordenada 2D en el sistema de coordenadas de la pantalla, pero si queremos conocer a qué punto de coordenadas del mundo corresponde, hay que realizar la correspondiente conversión. Para ello se puede:

- invertir la transformación de visualización (si estamos en un entorno 2D)
- restringir el posicionamiento a un plano
- utilizar un cursor 3D
- utilizar vistas con tres proyecciones paralelas
- invertir la transformación de visualización (si el dispositivo de entrada es 3D)

Posicionamiento 2D

Es el caso más fácil, y se da por ejemplo en los sistemas de edición de imágenes (Photoshop©, GIMP, etc.). En este contexto, la proyección es ortogonal y la conversión es casi inmediata:



$$x' = X_{min} + x * (X_{max} - X_{min})/ancho$$

$$y' = Y_{max} - y * (Y_{max} - Y_{min})/alto$$

Los parámetros son los indicados a las funciones glOrtho y glViewport:

```
glOrtho( Xmin,Xmax,Ymin,Ymax,Zmin,Zmax);
glViewport( x0, y0, ancho, alto );
```

Posicionamiento 3D

La introducción de posiciones 3D, utilizando dispositivos de entrada 2D, requiere el uso de técnicas especiales. Una de las estrategias posibles es hacer que el usuario introduzca las tres coordenadas del punto actuando sobre dos o tres proyecciones. La Ilustración 119 muestra la utilización de tres vistas, observándose que las coordenadas de cada vista están ligadas a las de las vistas contiguas. La introducción de un punto se debe descomponer en varios posicionamientos 2D.

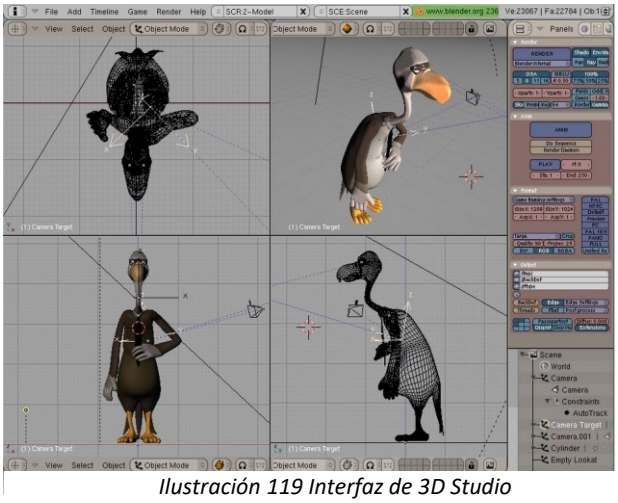


Ilustración 119 Interfaz de 3D Studio

Existen varias técnicas para generar los desplazamientos en las tres direcciones, a partir de los del dispositivo 2D. La más simple es utilizar un botón, o tecla, para conmutar la interpretación del movimiento del dispositivo, del plano X-Y al X-Z.

Entrada de transformaciones

Las transformaciones geométricas se pueden definir a partir de puntos.

- Una traslación se puede definir por el vector que va de la posición original a la posición nueva
- Una rotación por el ángulo formado por el vector que va del punto de referencia a la posición actual con la horizontal

La edición de modelos geométricos requiere la introducción tanto de puntos como de transformaciones geométricas. Las transformaciones geométricas utilizadas en modelado se pueden definir a partir de las coordenadas de puntos de referencia. Concretamente, una traslación, o un escaldo, se pueden definir a partir del vector que une dos puntos. Una rotación 2D se puede especificar a partir de la componente angular de un punto.

Una rotación 3D, respecto a un eje de coordenadas, se puede especificar de la misma forma, usando como referencia la componente angular del punto en el plano perpendicular al eje. Para la introducción de rotaciones 3D se han definido técnicas específicas. Entre las más ellas está la de Evans, Tanner y Wein. Esta técnica simula la rotación de una bola, distinguiendo los movimientos lineales del cursor, de los circulares. Los primeros se interpretan como rotaciones según los ejes X o Y. Los movimientos circulares del cursor se corresponden con rotaciones según el eje Z, perpendicular al plano de movimiento del cursor.

Otra opción es usar un *trackball* virtual, de forma que según la zona sobre la que se sitúe el cursor al hacer clic, se activa la rotación en el eje X, Y o Z, dibujándose en el color el círculo que determina cada trayectoria de rotación, o bien los desplazamientos en cualquiera de las tres direcciones, como se aprecia en la Ilustración 120

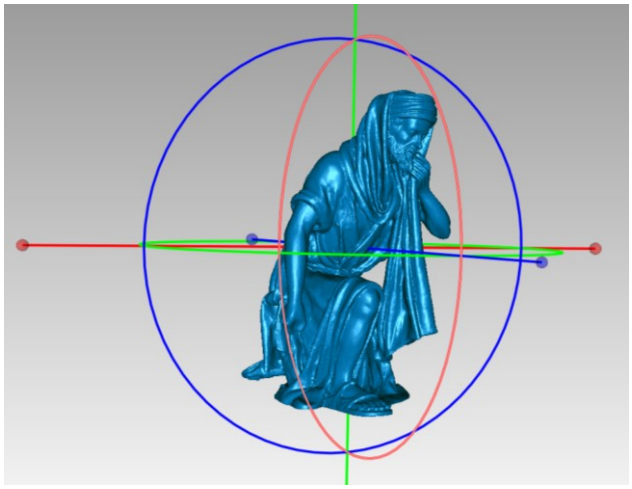


Ilustración 120 Trackball virtual para trasladar y rotar objeto.

Control de cámara

La cámara tiene demasiados parámetros para controlarlos todos interactivamente usando un ratón. Habitualmente se controla de forma interactiva un número reducido de parámetros, seleccionados dependiendo del tipo de aplicación:

- En visualización de modelos se modifica de forma rápida la posición (VRP) e indirectamente el VPN. Estamos en una cámara que mira siempre al mismo punto y gira en torno a él como si estuviéramos recorriendo una esfera. Es lo que se denomina una **cámara orbital**, como la de la Ilustración 121:

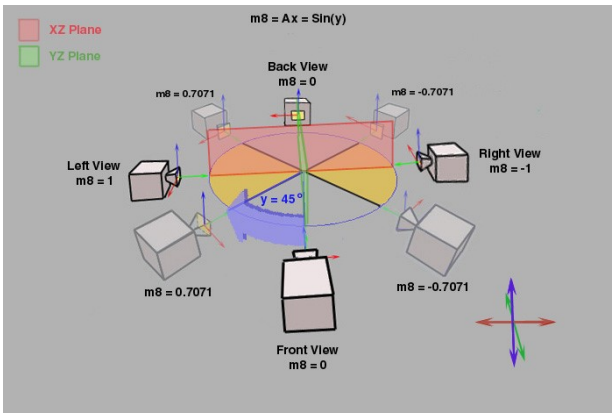


Ilustración 121 Cámara orbitando alrededor de un punto.

- En la exploración de escenarios se modifica de forma rápida la dirección (VPN) e indirectamente el VRP. Es lo que se denomina una cámara **en primera persona**, como la de la Ilustración 122 de forma que la cámara está situada en un punto fijo del sistema de coordenadas de un objeto. Ello no quiere decir que no se mueva la cámara, sino que lo hace de la misma forma que el objeto al que está asociada. El usuario controla la dirección de la cámara (VPN) mediante giros en los ejes X e Y del personaje. Una cámara en **tercera persona** es una cámara que sigue a un objeto, sobre la cual no hay control salvo el del avatar que se mueve, como en la

Ilustración 122 Cámara en primera persona.

Ilustración 123 Cámara en tercera persona.

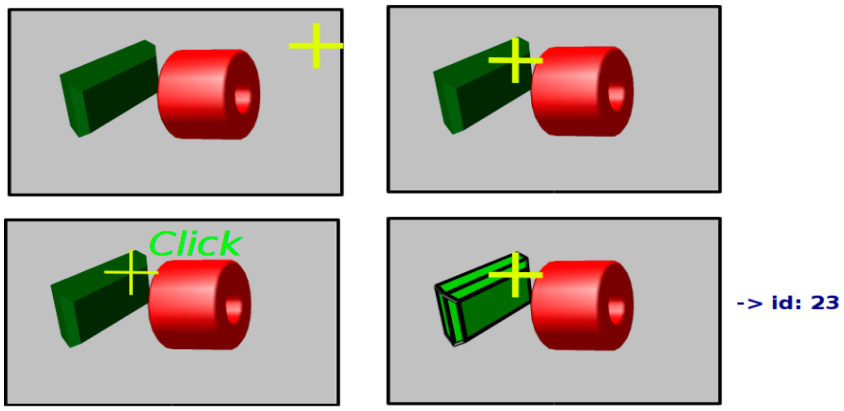


Ilustración 124 El click sobre el objeto verde devuelve su identificador.

La selección permite al usuario referenciar componentes de un modelo geométrico y es esencial en cualquier aplicación gráfica que requiera la edición de un modelo.

El proceso de selección suele realizarse como un posicionamiento y una búsqueda en el modelo geométrico, para lo que es necesario tener identificados a los objetos de la escena de alguna manera.

De hecho, es incluso posible tener una jerarquía de identificadores, para determinar el nivel de detalle de la selección (objeto, primitiva o punto).

En la selección se realiza una búsqueda en el modelo geométrico a partir de la información introducida por el usuario (usualmente una posición en pantalla). La asociación entre posiciones de pantalla y elementos del modelo se puede establecer de varios modos:

- Intersección rayo escena
- Haciendo clipping con subvolumen de visión centrado en la posición dada
- Codificando el Id de objeto como color y leyendo del frame buffer

Selección en OpenGL

OpenGL utiliza de forma clásica como mecanismo de selección el recortado de la escena a un entorno de la posición donde se ha hecho clic, y proporciona la información de las primitivas que se han dibujado. La más cercana al observador será la que se ha querido seleccionar.

Para distinguir un objeto de otro, OpenGL utiliza una pila de enteros como identificadores.

Dos primitivas se considera que corresponden a objetos distintos cuando el contenido de la pila de nombres es distinto.

Para ello proporciona las siguientes funciones:

```
glLoadName(i) / / Sustituye el nombre activo por i
glPushName(i) / / Apila el nombre i
glPopName() / / Desapila un nombre
glInitNames() / / Vacia la pila de nombres
```

Veamos como ejemplo la identificación de las distintas casillas de un tablero de ajedrez:

```
for( i=0 ; i<7 ; i++ ) {
 glPushName(i);
 glPushName(0);
       glTranslatef(1.0,0.0,0.0); // Acumulando Tx por columnas
       glPushMatrix();
       for(j=0;j<7;++j) {
             glLoadName(j);
             glTranslatef(0.0,1.0,0.0); // Acumulando Ty por filas
             DrawCasilla();
       }
 glPopName();
 glPopName();
 glPopMatrix();
}
```

Además de los identificadores, hay que decirle a OpenGL que cuando hacemos una selección no queremos dibujar, sino sólo saber qué objeto hay debajo del cursor. Para ello, OpenGL funciona en tres modos:

- RENDER: modo por omisión. En este modo las primitivas que se envían se dibujan en pantalla
- SELECT: OpenGL devuelve los identificadores de las primitivas que se proyectan en una región.
- FEEDBACK: OpenGL devuelve la información geométrica de las primitivas que se dibujarían

El cambio de modo se realiza con la función glRenderMode:

```
glRenderMode( GL_SELECT );
glRenderMode( GL_RENDER );
glRenderMode( GL_FEEDBACK );
```

Buffer de selección

Al realizar la selección usando el modo GL_SELECT OpenGL devuelve el número de objetos seleccionados como resultado de selección y la información asociada a estos en un buffer de enteros. El buffer se debe de inicializar antes de pasar OpenGL a modo GL_SELECT:

```
GLuint selectBuf[BUFSIZE];
glSelectBuffer (BUFSIZE, selectBuf);
glRenderMode (GL_SELECT);
```

OpenGL devuelve en el buffer de selección la secuencia de objetos pintados en el pequeño volumen de recorte generado. Incluso las que no se vean por efecto del zbuffer.

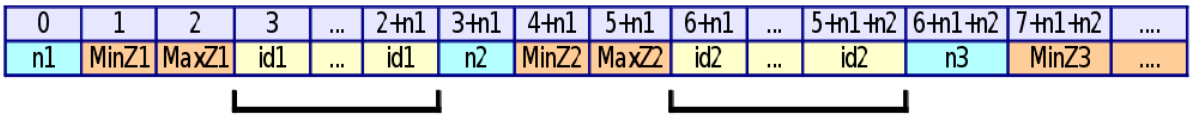
Como mejor se ve es en un ejemplo:

```
GLuint selectBuf[BUFSIZE];
void Escena::pick( int x, int y) {
       GLint hits, viewport[4];
       glGetIntegerv (GL_VIEWPORT, viewport);
       glSelectBuffer (BUFSIZE, selectBuf);
       glRenderMode (GL_SELECT);
             glInitNames();
             glMatrixMode (GL_PROJECTION);
             glLoadIdentity ();
             gluPickMatrix ( x, viewport[3] - y, 5.0, 5.0, viewport[0]);
             glFrustum(-Size_x,Size_x,-Size_y,Size_y,Front_plane,Back_plane);
             escena.dibujar ();
       hits = glRenderMode (GL_RENDER);
       glMatrixMode (GL_PROJECTION); // Volvemos a poner el volumen original
       glLoadIdentity();
       glFrustum(-Size_x,Size_x,-Size_y,Size_y,Front_plane,Back_plane);
       // interpretar el buffer de selección
       if (hits!=0)
             procesarHits(hits,selectBuf);
}
```

A la hora de procesar los elementos seleccionados, hay que saber leer el buffer devuelto por OpenGL. Es el punto clave:

En el buffer devuelve:

- el número de primitivas que contiene la pila
- el intervalo de profundidades de cada primitiva: z_{min} y z_{max}. Los valores se toman del ZBuffer (intervalo [0,1]), y son multiplicados por 2^{32}-1, para dar un entero de grandes dimensiones que permita comparaciones rápidas.
- los nombres asociados a cada primitiva. Es decir, si estamos en un modelo jerárquico, se obtendría el nombre del objeto, y el de la parte o partes en que hayamos querido identificar. Por ejemplo, podemos querer saber que es el brazo derecho de uno de lo 10 androides que tengamos en la escena, y debemos saber qué androide es y el brazo concreto.



Para determinar qué objeto estaba siendo visualizado en ese instante, hay que utilizar la información de profundidad, por ejemplo quedándonos con el objeto con menor Zmin. El código siguiente, adaptado del libro rojo de OpenGL lo saca por pantalla:

```
void Escena::procesarHits (GLint hits, GLuint buffer[])
{
   unsigned int i, j;
   GLuint names, *ptr, minZ,*ptrNames, numberOfNames;

   printf ("Primitivas intersecadas = %d\n", hits);
   ptr = (GLuint *) buffer;
   minZ = 0xffffffff;
   for (i = 0; i < hits; i++) {
      names = *ptr;
        ptr++;
        if (*ptr < minZ) {
                numberOfNames = names;
                minZ = *ptr;
                ptrNames = ptr+2;
        }

        ptr += names+2;
      }
  printf ("Los nombres de la primitiva más cercana son: ");
  ptr = ptrNames;
  for (j = 0; j < numberOfNames; j++,ptr++) {
     printf ("%d ", *ptr);
  }
  printf ("\n");

}
```

Selección en OpenGL con codificación por colores.

Hay un mecanismo más simple aún para determinar qué primitiva ha sido seleccionada. Se trata de usar un código de color para cada objeto seleccionable, como el de la Ilustración 125. Se trata de crear una función de dibujado distinta para cuando queremos seleccionar, y cuando el usuario hace clic, se pinta la escena "para seleccionar" en el buffer trasero y se lee el color del pixel donde el usuario

ha hecho clic. Si no se hace un intercambio de buffers, el usuario jamás verá esa escena "rara", y el programa seguirá su proceso natural.

En resumen, los pasos a seguir son:

- Llamar a la función *dibuja_seleccion()*
- Leer el pixel (x,y) dado por la función gestora del evento de ratón
- Averiguar a qué objeto hemos asignado el color de dicho pixel
- **No intercambiar buffers**

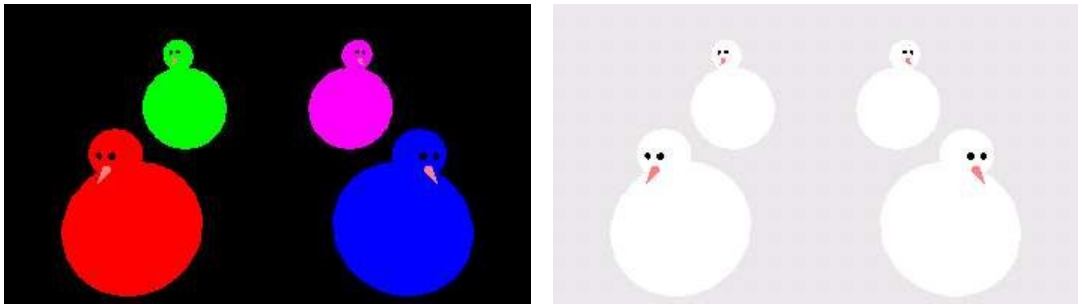


Ilustración 125: Escena visible (dcha) y escena para selección (izda).

```
void Escena::dibuja_seleccion() {

// Dibuja cuatro patos

glDisable(GL_DITHER); // deshabilita el degradado
for(int i = 0; i < 2; i++)
    for(int j = 0; j < 2; j++) {
        glPushMatrix();
        switch (i*2+j) { // Un color para cada pato
                case 0: glColor3ub(255,0,0);break;
                case 1: glColor3ub(0,255,0);break;
                case 2: glColor3ub(0,0,255);break;
                case 3: glColor3ub(250,0,250);break;
        }
        glTranslatef(i*3.0,0,-j * 3.0);
        pato.dibuja();
        glPopMatrix();
    }
glEnable(GL_DITHER);
}
```

Para comprobar el color del pixel, usaremos la función glReadPixels:

```
void glReadPixels(GLint x, GLint y,
                  GLsizei width, GLsizei height,
                  GLenum format, GLenum type, GLvoid *pixels);
```

- `x,y` : la esquina inferior izquierda del cuadrado a leer (en nuestro caso el x,y, del pick)
- `width,height`: ancho y alto del área a leer (1,1 en nuestro caso)
- `format`: Tipo de dato a leer (coincide con el format del buffer, GL_RGB o GL_RGBA).
- `type`: tipo de dato almacenado en cada pixel, según hayamos definido el glColor (p.ej. GL_UNSIGNED_BYTE de 0 a 255, o GL_FLOAT de 0.0 a 1.0)
- `pixels`: El array donde guardaremos los pixels que leamos. Es el resultado de la función.

En el caso del procesamiento del pick, una vez dibujado el buffer, llamaríamos a esta función:

```
void processPick (int x, int y) {
        GLint viewport[4];
        GLubyte pixel[3];

        glGetIntegerv(GL_VIEWPORT,viewport);

        glReadPixels(x,viewport[3]-y,1,1,
                GL_RGB,GL_UNSIGNED_BYTE,(void *)pixel);

        printf("%d %d %d\n",pixel[0],pixel[1],pixel[2]);
        if (pixel[0] == 255)
          printf ("Pato 1");
        else if (pixel[1] == 255)
          printf ("Pato 2");
        else if (pixel[2] == 255)
          printf ("Pato 3");
        else if (pixel[0] == 250)
          printf ("Pato 4");
        else
           printf("Agua!");
  printf ("\n");

}
```

Para que esto funcione, hay varias cosas a tener en cuenta:

- Los colores se han de definir con glColor3ub, es decir, como enteros de 0 a 255
- Es posible que el monitor no esté en modo *trueColor* y no devuelva exactamente el valor que pusimos (hay que tenerlo en cuenta si no funciona bien).
- Hay que **desactivar** el GL_DITHER, GL_LIGHTING, GL_TEXTURE

Si usamos valores reales de objetos, el monitor debe ser muy fiable para renderizar exactamente el color que queremos. En caso contrario no funcionará.

ANIMACIÓN

Dice el Diccionario de la Real Academia Española de la Lengua que **animar**, entre otras acepciones, es "*(6) Dotar de movimiento a cosas inanimadas, (8) Dicho del alma: Vivificar al cuerpo.*", por lo que la animación es "*Acción y efecto de animar o animarse*" y recientemente se ha añadido la acepción "*En las películas de dibujos animados, procedimiento de diseñar los movimientos de los personajes o de los objetos y elementos.*"

En el contexto de la Informática Gráfica podríamos por tanto inferir que la animación es crear la ilusión de que las cosas están vivas, que se producen cambios en ellas. Estos cambios pueden ser por:

- movimiento (cambio de posición / orientación)
- metamorfosis (cambio de forma)
- otras variaciones (cambios de color, de luz, etc.)

La animación añade a los gráficos la dimensión temporal, lo que incrementa enormemente la cantidad de información que puede transmitirse, y para ello se usan muy diversos recursos, la mayoría de ellos heredados de la animación clásica.

La animación por ordenador es posible gracias a que el sistema visual humano interpreta una secuencia de imágenes como una serie continua. Los receptores oculares están muestreando continuamente la cantidad de luz que les llega, pero tienen un "defecto": el tiempo de reacción. Si un

objeto se mueve demasiado rápido, los conos y bastones no serán capaces de responder con tanta celeridad y el cerebro no podrá interpretar los bordes de forma nítida, sino que verá el objeto desenfocado. Por el contrario, gracias a esta propiedad de persistencia, si una secuencia de imágenes se actualiza lo suficientemente rápido, la sensación es de movimiento continuo.

El cine (la palabra cine, viene del griego y significa movimiento) surgió gracias al descubrimiento de esta propiedad de persistencia, aplicado experimentalmente en el **zoopraxiscopio**, precursor del kinetoscopio.

Ilustración 126 Zoopraxiscopio (izda) y secuencia de imágenes usada (dcha)

Las animaciones por ordenador consisten en generar sintéticamente cada uno de los fotogramas, a una frecuencia al menos de 30 fotogramas por segundo, que es la frecuencia a partir de la cual el ojo humano no percibe los saltos.

Si la frecuencia de refresco no alcanza la velocidad necesaria para generar la ilusión de continuidad, se produce el parpadeo. Eso sí, es posible alcanzar una sensación de animación mostrando tan sólo seis imágenes diferentes por segundo (cinco veces cada imagen, para alcanzar los 30fps).

EJERCICIOS

79. *Complete sus apuntes con un resumen del artículo*
    Lassiter, John "Principles of Traditional Animation Applied to 3D Computer Animation," SIGGRAPH '87, pp. 35-44
    descargable de http://www.cs.cmu.edu/~15462/lec_slides/Lesseter.pdf
80. *Vuelva a ver el corto "Las aventuras de André y Wally B" (https://www.youtube.com/watch?v=Taq9LFbcvxE) e identifique las acciones y recursos descritos en el artículo del ejercicio anterior.*

La primera película animada por ordenador se puede considerar que es el corto "Hunger" (1974), visible en http://www.nfb.ca/film/hunger). En ella las imágenes intermedias son generadas automáticamente por interpolación. En 1977 se incluyeron efectos especiales generados por ordenador en *La Guerra de las Galaxias.*

A la hora de entender de qué estamos hablando cuando tratamos la animación, hay que discriminar cómo se generan las imágenes y qué apariencia visual tiene el resultado. No es lo mismo "animación asistida por ordenador", que sería el ejemplo del corto Hunter donde el ordenador se usa como una herramienta para generar imágenes intermedias, que "animación por ordenador", donde la totalidad del proceso se genera en el ordenador.

Tenga usted en cuenta que no estamos hablando de 2D o 3D, sino de cómo se generan las imágenes. Por ejemplo, la famosa serie Los Simpson usa animación asistida por ordenador: los personajes son dibujados a mano y coloreados por ordenador (entrevista a su autor en 2012: http://latimesblogs.latimes.com/showtracker/2012/02/qa-matt-groening-on-the-simpsons-at-500.html), mientras que desde 2013 Disney ya no dibuja a mano ninguno de sus productos (http://www.theguardian.com/film/2013/mar/07/disney-hand-drawn-animation).

En este tema nos vamos a centrar brevemente en la *animación por ordenador,* esto es, en el proceso de generación de secuencias de imágenes íntegramente mediante técnicas digitales. La gran diferencia frente a la animación clásica y a la asistida por ordenador es que, una vez se han definido los modelos y datos que controlan la animación, es el propio ordenador el que gestiona la obtención de las imágenes, sin intervención humana en el proceso.

Lo que para un hombre pueden ser horas de trabajo, el ordenador lo hace rápidamente y sin errores. Además, la flexibilidad que ofrece permite generar mundos virtuales y reglas de animación imposibles de conseguir a mano.

Hemos dicho anteriormente que la animación es el resultado de la modificación de ciertos parámetros de la escena entre dos imágenes consecutivas. Estas modificaciones, ya sean geométricas o de aspecto se pueden realizar usando diversas técnicas:

Animación usando fotogramas clave (keyframe): interpolación.

En la animación por fotogramas clave, el animador define los parámetros de la animación en ciertos instantes concretos (puntos de control o claves), y el ordenador calcula los fotogramas intermedios mediante interpolación.

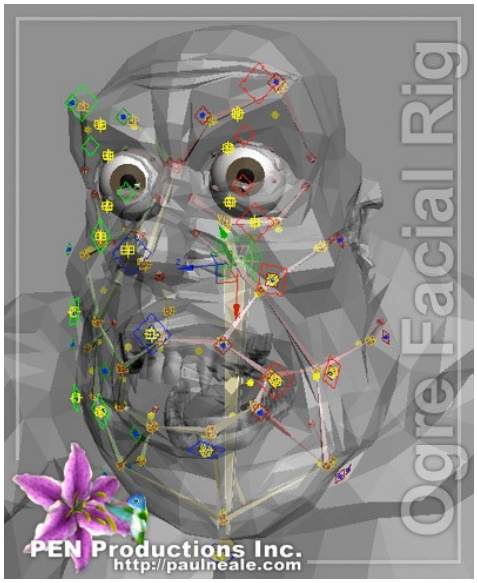


Ilustración 127 Puntos de control en la animación de un ogro.

¿Cuáles son esos parámetros que se definen? Pues depende de la escena. En el caso del ogro de la Ilustración 127, hay docenas de puntos de control y por tanto parámetros de orientación y posición que definir.

Este tipo de animación es la que más se parece a la animación manual clásica, pues ésta también se realiza intercalando fotogramas clave e interpolando (pero a mano).

La interpolación puede realizarse con cualquiera de los interpolantes posibles (lineal, cúbica, paramétrica, etc.), pero lo importante es saber en qué instante de tiempo está cada fotograma clave, como se muestra en la Ilustración 128, donde la posición x del sol en cada momento viene dada por la expresión:

$$x = x_0 + \frac{t - t_0}{t_1 - t_0}(x_1 - x_0)$$

donde t_1 es el instante final de la animación, t_0 el inicial, x_1 la posición final y x_0 la inicial.

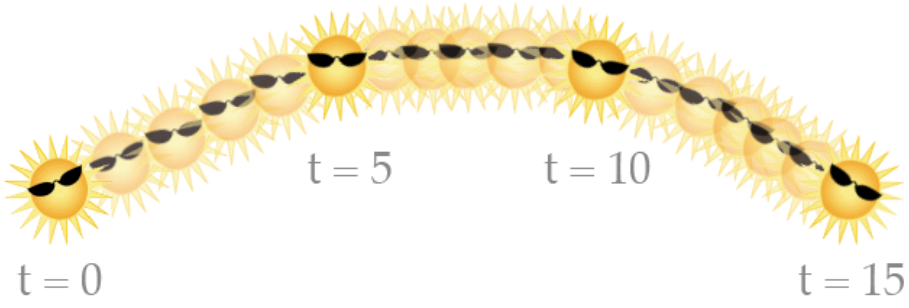


Ilustración 128 Interpolación lineal de la posición y orientación de un Sol con gafas

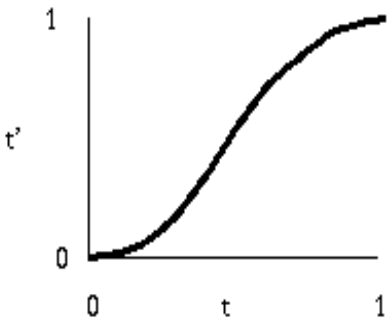


Ilustración 129 Interpolación sinusoidal.

Otro tipo de interpolación podría ser la sinusoidal, es decir, aquella que genera las posiciones intermedias no con una función lineal sino con una función seno. De esta forma, si en lugar de t, usamos t'=sin(t·π/2) para t=0 conservamos t'=0 y para t=1, t'=1 (ver Ilustración 129) pero entre medias el valor de t' sigue un criterio tal que la animación es "lento al salir y lento al llegar", que sería algo parecido a lo que hace un coche al acelerar, tener cierta velocidad constante, y frenar poco a poco, o lo que hace un péndulo como el de la Ilustración 130, que se frena justo antes de llegar a su punto más alto y acelera poco a poco hasta volver a su posición mínima.

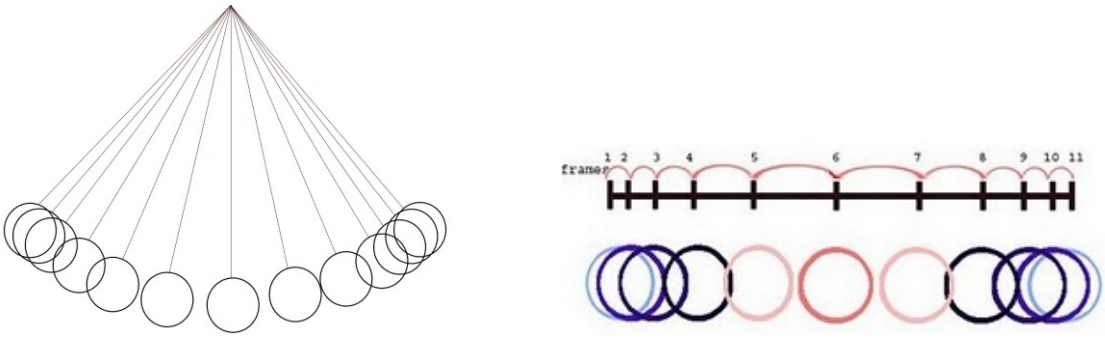


Ilustración 130 Fotogramas interpolados en una animación sinusoidal.

Animación por esqueletos

Para conseguir que la animación de personajes resulte plausible se suelen usar esqueletos. Un esqueleto es un modelo simplificado del personaje, formado por segmentos rígidos unidos por articulaciones. Vinculados a estos segmentos se sitúan las mallas poligonales que representan la superficie del objeto animado, en lo que se denomina la piel. Si nos fijamos, es exactamente lo que ocurre en los modelos reales: tenemos una estructura de huesos con músculos y la piel sobre esta estructura.

Puede usted ver claramente que el esqueleto es un modelo jerárquico, por lo que no le resultará especialmente complejo, a la vista de lo ya aprendido en la asignatura, hacerse una idea de cómo dependen unas articulaciones de otras.

Ahora bien, la animación de un personaje se puede realizar también por la técnica de puntos clave e interpolación del modelo poligonal tradicional, pero supone un gran consumo de memoria y de recursos (hay que calcular las nuevas posiciones para cientos o miles de vértices.

Sin embargo, la animación por esqueleto, tan sólo requiere animar las articulaciones de la estructura y permite incluso definir los movimientos con ciertas funciones paramétricas. Además, los movimientos pueden ser reutilizados de un esqueleto a otro, y cambiar totalmente la piel para que la apariencia sea distinta.

La animación del esqueleto se puede realizar utilizando fotogramas clave e interpolación, pero ojo, interpolamos posición y orientación del esqueleto, no de la piel. Esta interpolación lineal se hace como hemos explicado anteriormente.

Ahora bien, animar un esqueleto no es igual que animar un objeto rígido. Por ejemplo, para hacer que un avatar levante la mano, en realidad se mueven varias articulaciones: el hombro, el codo y la muñeca. ¿Cómo se modela dicha animación? Hay dos aproximaciones:

- cinemática directa. Mediante este proceso, se definen en cada paso los ángulos concretos de rotación para cada articulación.

- cinemática inversa. En este sistema se definiría la posición inicial y final del extremo de la jerarquía, en este caso la mano, y el sistema se encarga de calcular los ángulos de cada articulación en cada frame. Esta forma de animar implica establecer las restricciones de movimiento allí donde las hubiera (p.ej. que el codo no pueda ir hacia atrás), pero sin duda es la preferida por los diseñadores gráficos.

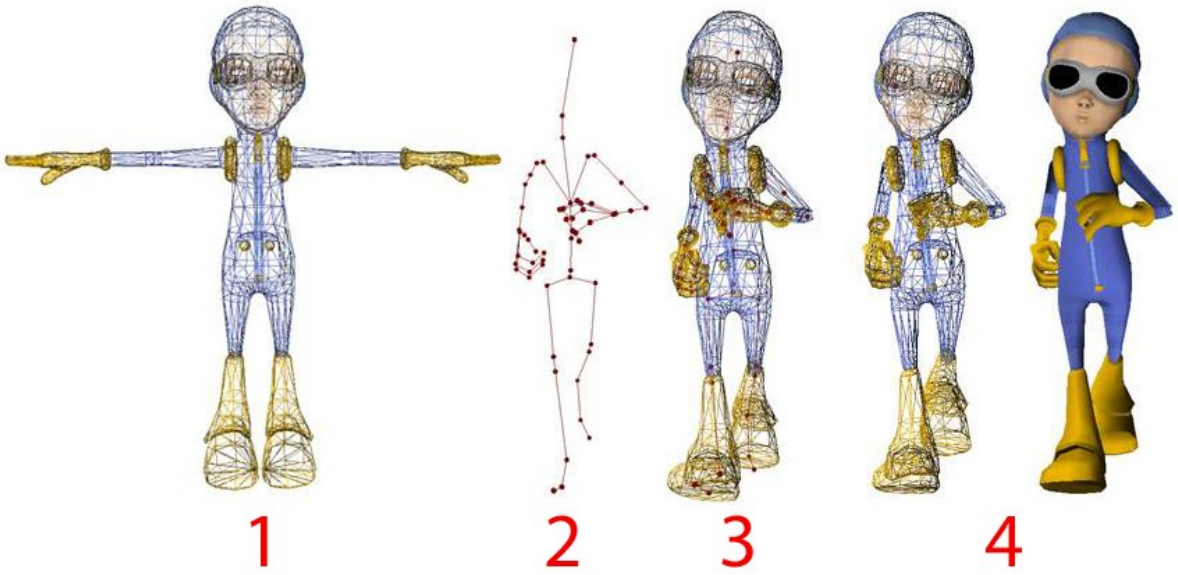


Ilustración 131 1. Malla a animar. 2. Esqueleto de control. 3. Distintas poses. 4. Resultado final.

Animación procedural

En la animación procedural, el comportamiento del objeto se describe mediante un procedimiento o función. Este tipo de animación es útil cuando el comportamiento es fácil de generar pero dificil de simular físicamenete (p.e. la rotura de un vidrio).

Animación por simulación física

Ciertas animaciones de escenas simples se pueden realizar usando las leyes de la mecánica clásica. Por ejemplo, es más fácil dejar que el ordenador calcule los rebotes de una jugada de billar que dibujar los fotogramas clave de cada movimiento e impacto de la bola. Para ello existen diversas librerías, como ODE (Open Dynamic Engine, www.ode.org), Newton (www.**newton**dynamics.com) o Bullet (www.bulletphysics.org) que se encargan de simular detección de colisiones, y dinámica de sólidos rígidos o deformables.

Estas animaciones guíadas por simulación física utilizan modelos simplificados de las leyes e interacciones físicas para alcanzar frecuencias de refresco en tiempo real. Cuando se utilizan para realizar animaciones a partir de generación de imágenes de alto realismo, la influencia y complejidad de la física utilizada aumenta.

Estas simulaciones físicas pueden entrar en la dinámica de sólidos rígidos, deformables e incluso de fluidos, como la mostrada en la Ilustración 132 .

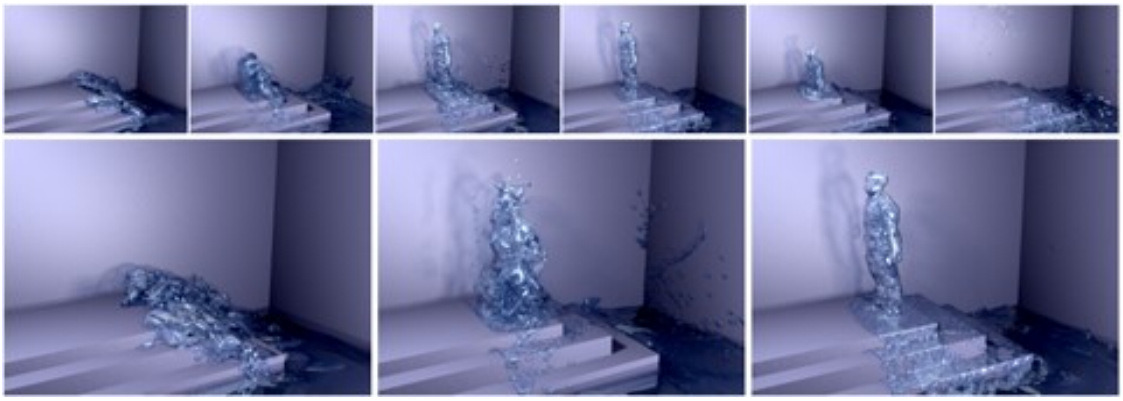


Ilustración 132 Animación de fluidos mediante simulación física.

Sistemas de partículas

Ilustración 133 Sistema de partículas: fuego

Un sistema de partículas es un conjunto de muchas diminutas partículas que juntas representan un objeto difuso. A lo largo de un intervalo de tiempo, las partículas se generan en el sistema, se transforman y mueren. Sistemas de partículas clásicos son el fuego como el de la Ilustración 133, el humo, las explosiones, bandadas de pájaros, etc. Algunos ejemplos se pueden ver en la Ilustración 134.

Una partícula, en singular, es un elemento con una serie de propiedades comunes a todas las partículas de su clase, pero con distintos valores para cada una de la partícula. Entre estas propiedades podemos encontrar la posición, velocidad, aceleración, color, edad, etc. En función de lo que estemos modelando, necesitaremos controlar unas propiedades u otras en el sistema de partículas.

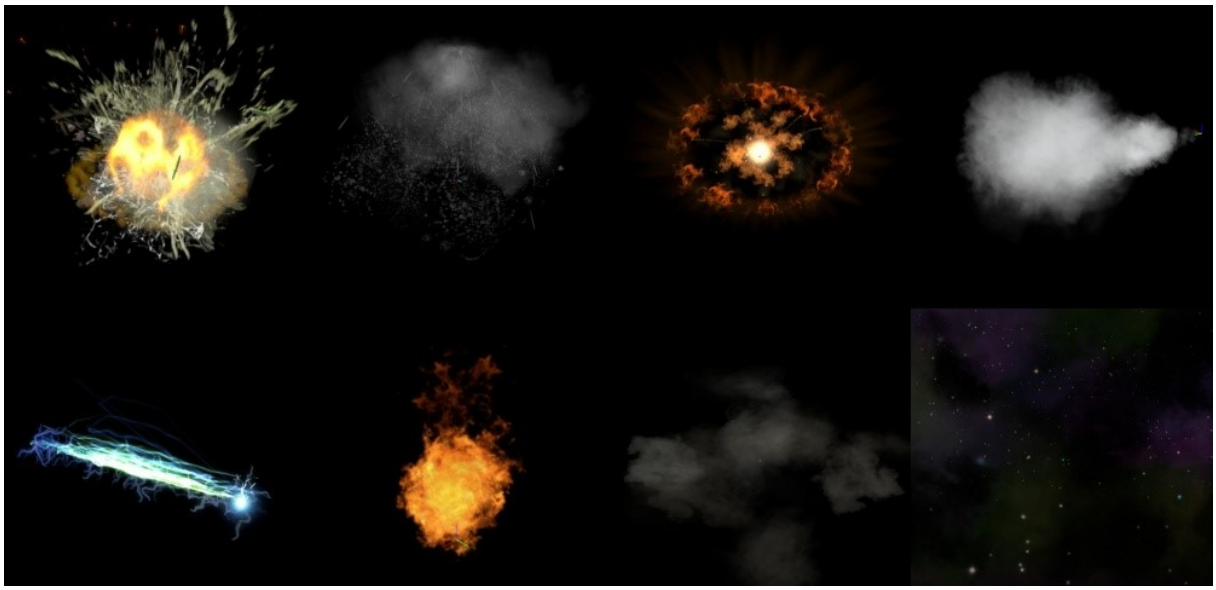


Ilustración 134 Sistemas de partículas diferentes.

Las partículas surgen de una fuente o emisor, es decir, el punto desde el que emanan las partículas al comienzo de su vida útil. Pueden emitirse todas las partículas a la vez (como en el caso de la explosión) o de forma continua (como en el caso de un fuego). En el caso del fuego, por ejemplo, la emisión continua obliga a que las partículas "mueran", pues de lo contrario el sistema se saturaría.

El ciclo de vida de una partícula se puede generalizar de la siguiente forma:

- **Generación**. Las partículas se generan aleatoriamente en el emisor, para el cual se ha definido una forma y posición determinada que puede variar (pensemos p.ej. en un coche que expele humo en movimiento). Los valores iniciales de los parámetros de cada partícula pueden ser definidos en la creación de forma aleatoria o siguiendo algún patrón predefinido.
- **Dinámica de partículas**. Los atributos o propiedades de las partículas cambian en cada frame (p.ej. en una explosión el color de la partícula se oscurece conforme se aleja del origen de la explosión, indicando que se enfría). Algunos atributos se pueden definir en función de otros, en incluso verse influenciados por las propiedades de otras partículas (como en el caso de las colisiones). Esta interacción puede ser controlada por motores físicos más o menos complejos.
- **Extinción**. Cada partícula tiene dos propiedades que controlan su existencia: edad y tiempo máximo de vida. La edad es el número de frames que lleva viva la partícula, y siempre se inicializa a cero. Cuando la partícula llega a una edad igual a su tiempo de vida, ésta muere. Ello no quita que no haya otras causas para una extinción prematura:
    - Cuando se sale del área de visión
    - Cuando golpea el suelo una chispa de fuego
    - Cuando algún atributo alcanza un umbral (p.ej. si el color se torna en negro)

Una partícula es algo diferente a lo que hasta ahora hemos estado habituados en la asignatura ya que un sistema de partículas:

- no se representa con un conjunto de primitivas (polígonos), pero el conjunto de partículas sí forma un volumen
- no es una entidad estática
- Un sistema de partículas se dice que es no determinista, ya que su forma no está completamente definida y se usan procesos estocásticos que cambian la forma y apariencia. Hay ciertos sistemas de partículas, denominados deterministas, donde el comportamiento de éstas se gestiona por una máquina de estados en lugar de por procesos pseudoaleatorios.

BIBLIOGRAFÍA

[Shreiner05] **OpenGL Programming Guide;** Shreiner et al., Addison Wesley, 2005

[Hill01] **Computer Graphics using OpenGL;** F.S. Hill Jr., Prentice Hall, 2001

[Angel08] **Interactive Computer Graphics: A Top Down Approach** (5ª Ed); E. Angel. Addison Wesley, 2008

[Hearn10] **Computer Graphics with Open GL** (4ª Ed) ; D. Hearn, P. Baker, W. Carithers; Prentice Hall, 2010

Recursos online (principalmente cursos en otras universidades):

- http://www.xmission.com/~nate/tutors.html
- http://www.opengl-tutorial.org/
- http://content.gpwiki.org/index.php?title=OpenGL:Tutorials:Basic_Bones_System