Programación de animaciones tridimensionales mediante "Animadores 3D"

Sanchez, A. –Orosco, R.
UNCPBA – INSUC – DCyS
Tandil – Argentina
email: {asanchez,orosco}@exa.unicen.edu.ar

Resumen

La utilidad de las animaciones en las visualizaciones es ampliamente conocida. A pesar de esta importancia, los mecanismos para la especificación de animaciones 3D no son convenientes para los programadores. En la mayoría de los métodos actuales, los programadores de animaciones se ven forzados a considerar varios detalles de implementación de las animaciones, que dificultan la tarea de programación. En este trabajo, se describe el concepto de "restricciones de animación 3D" o " animadores 3D", los cuales son entidades de software específicamente desarrolladas para facilitar la programación de animaciones en visualizaciones. Los animadores 3D combinan las ventajas de los mecanismos de restricciones convencionales con las necesidades para especificar animaciones. Como principales ventajas de este mecanismo, se provee un alto nivel de abstracción para la programación (ocultando varios de los detalles de implementación), y una alta flexibilidad por medio de la parametrización de su comportamiento y la definición de distintos tipos de animadores. En particular, en este trabajo se describe la aplicación de los animadores 3D a las interfaces 3D, considerando las características requeridas por este tipo de aplicaciones.

Palabras clave: Animaciones. Visualización. Visualización de Información. Restricciones.

1. Introducción

Existe una amplia difusión en la utilización de animaciones en presentaciones tridimensionales, en areas tan diversas como juegos, educación, sistemas de control, y el estudio de fenómenos dinámicos [1]. En los ultimos años, esta difusión se ha extendido a las aplicaciones de visualización (tanto científicas como de información [2]), en las cuales han adquirido una gran importancia. La principal razón detrás de esta difusión es la utilidad de las animaciones para facilitar al usuario, o analizador, la comprensión de datos con comportamientos dinámicos [3]. Por ejemplo, en una simulación las animaciones posibilitan al usuario una percepción más sencilla de las repercusiones ante una modificación producida por algún determinado factor.

Usualmente, el programador de una animación 3D utiliza algún lenguaje particular para construir tal animación. Este lenguaje le permite describir los objetos presentes en la visualización (los cuales componen la "escena" o "mundo virtual"), conjuntamente con las capacidades dinámicas e interactivas de cada uno de dichos objetos. La descripción de la escena generalmente establece la posición y orientación de los objetos componentes, el material que los constituye, el tipo de luz utilizado en la iluminación, y finalmente la posición y orientación de la cámara a través de la cual el usuario observa la escena.

Una vez descripta la composición de esta escena, pueden incorporarse aspectos dinámicos a los objetos que componen este mundo virtual, describiendo la variación temporal de uno o más de sus atributos. Esta descripción debe detallar, entre otras cosas: el estado inicial y final de cada

atributo, el tiempo de duración, la cantidad de imágenes (o “frames”) a exhibir durante este tiempo y los estados intermedios que desean resaltarse en cada animación. En general, la especificación de estos aspectos dinámicos no es una tarea sencilla, requiriendo en muchas ocasiones la disponibilidad de un programador especializado en animaciones.

El rango de los mecanismos utilizados actualmente para especificar animaciones es bastante amplio. La mayoría de estos mecanismos consisten en una especificación textual de la composición de la animación; otros, en cambio, intentan determinar la composición de la animación a partir de la observación de determinados fenómenos en la vida real. Sin embargo, estos últimos no son aplicables a aplicaciones de visualización de información, ya que en estos casos se trata con datos de naturaleza abstracta.

A pesar de la diversidad de métodos de especificación existentes, la tarea de construcción de animaciones para visualización de información 3D sigue sin ser sencilla. En la mayoría de los casos [4] [5] [6], la causa de esta dificultad reside en el bajo nivel de abstracción provisto para describir comportamientos dinámicos. Este inconveniente ha sido solucionado parcialmente por otros métodos [7] [8], los cuales proveen un estilo más declarativo para especificar animaciones. Sin embargo, estos métodos han sido aplicados mayormente en interfaces bidimensionales convencionales (tipo WIMP), sin considerar los aspectos propios de las visualizaciones de información 3D.

En este trabajo se describen los “animadores 3D”, los cuales son entidades de software que permiten especificar declarativamente la composición y el comportamiento de una animación en una visualización de información 3D. Estas entidades proveen un alto nivel de abstracción para describir estas animaciones, ocultando al programador todos aquellos detalles de implementación de la animación. Asimismo, estas entidades pueden ser reutilizadas en diferentes situaciones, adaptándolas a las características particulares de cada aplicación desarrollada.

En la siguiente sección se describen los principales métodos de especificación de animaciones disponibles hoy en día, señalando las ventajas y desventajas de cada uno de ellos. En las secciones siguientes se describe el modelo de “animadores 3D” propuesto en este trabajo, señalando sus principales características y su aplicabilidad a una técnica de visualización de información concreta.

2. Métodos de especificación de animaciones

Los métodos de especificación de animaciones difieren en la forma en la cual el programador debe describir la animación. La mayoría de estos métodos consisten en la utilización de algún lenguaje particular para describir la composición de una animación. En otros métodos no es necesario especificar la animación por medio de un lenguaje, ya que se intenta inferir dicha animación a partir de la observación de determinados objetos en el mundo real. Este tipo de animaciones es generalmente utilizado en áreas tales como la realidad virtual (ej. *“JDCAD”* [9]). Sin embargo, no se trata de métodos aplicables a visualización de información, ya que por tratarse de objetos abstractos (es decir, sin una representación física determinada) no es posible observar su comportamiento en el mundo real.

En el caso de los mecanismos de especificación por medio de lenguajes, las diferencias entre ellos están dadas por el nivel de detalle en el que debe especificarse la animación. También existen diferencias en cuanto a la cantidad de código que es necesario introducir para tal especificación.

Los toolkits o bibliotecas 3D de bajo nivel, tales como *OpenGL* [4] y *DirectX* [6], requieren una codificación altamente detallada de todo lo que debe ocurrir dentro del mundo virtual. En general, esta codificación consta de los cambios de escala, traslaciones y rotaciones que debe sufrir cada objeto en el transcurso de la animación.

En otros casos, es posible especificar solamente los principales estados que pueden tomar los objetos durante la animación; posteriormente, pueden calcularse los estados intermedios por medio de técnicas tales como interpolaciones. Dichas técnicas son generalmente implementadas por medio de "interpoladores", los cuales son mecanismos especializados para la construcción de animaciones. *OpenInventor* [5], un toolkit basado en *OpenGL*, provee este tipo de construcciones para implementar animaciones.

El toolkit *ArtKit* [10] utiliza un enfoque similar, a través de la provisión de los denominados "objetos de transición". Estos objetos contienen una referencia al objeto gráfico que será animado, conjuntamente con la trayectoria que debe seguir dicho objeto, y el intervalo de tiempo de la animación. El programador debe codificar tres métodos especiales en cada uno de los objetos animados (*start_transition*, *transition_step*, y *end_transition*). Lamentablemente, este modelo se encuentra disponible solamente para valores numéricos (especialmente posiciones); además requiere la escritura de código específico para las distintas acciones de la animación.

En algunos lenguajes de programación, se han añadido construcciones especiales para determinados efectos de animación. *Self* [11] es uno de estos lenguajes, en el cual se incorporaron primitivas para efectos tales como movimientos o desaparición de objetos de la escena. En este caso, el mayor inconveniente reside en que los efectos fueron introducidos por medio de código específico, y sin guías para que el programador pueda adaptarlos o reusarlos en distintas situaciones.

Uno de los mecanismos más empleados actualmente para la especificación de comportamientos dinámicos en escenas virtuales 3D es la utilización de fórmulas o restricciones. La razón fundamental de su uso es que el nivel de abstracción en el que debe realizarse tal especificación es mucho más elevado.

2.1. Utilización de restricciones para especificar comportamientos dinámicos

Una restricción expresa una relación entre un conjunto de variables por medio de una expresión, y condiciona los valores que puede tomar una variable ante una modificación en el valor de alguna de las otras variables relacionadas. Al modificarse alguno de los valores, un proceso denominado *'resolvedor'* (*"solver"*) buscará un nuevo conjunto de valores posibles para que las restantes variables satisfagan nuevamente la relación expresada por la restricción. Uno de los principales objetivos de este tipo de sistemas es obtener una implementación eficiente que posibilite una resolución rápida de las restricciones expresadas.

Las restricciones son una práctica bastante común en las visualizaciones, ya que permiten especificar claramente las relaciones entre los atributos de los objetos representados. Existen varios *resolvedores* de expresiones para aplicaciones gráficas en dos dimensiones: *"Cassowary"* [7] y *"SkyBlue"* [12] son dos de tales resolvedores. Algunos otros sistemas aplican las restricciones a presentaciones 3D, tales como *"UGA"* [13] y *"VB2"* [14].

A pesar de sus ventajas, la utilización de restricciones puras posee algunos inconvenientes para la implementación de animaciones. Utilizando restricciones puras, la modificación del valor de una variable afectada por una restricción, producirá que el resolvedor inmediatamente determine los nuevos valores para las otras variables involucradas en la restricción. Estas últimas tomaran sus nuevos valores en una forma instantánea, sin poder expresar una transición gradual al nuevo valor (es decir, no existirá una animación). Una posibilidad para evitar tal situación es vincular las restricciones con el tiempo de una ejecución (como en el sistema *Animus* [15]), aunque las animaciones serían bastante complejas de especificar.

La alternativa más conveniente es la provisión de restricciones especiales para efectuar animaciones, que efectúen una transición gradual entre dos valores. Estas *"restricciones de animación"* (o *"animadores"* como se las denomina en este trabajo) encapsulan todos los detalles

de la implementación de la animación, facilitando al programador su especificación. *"Amulet"* [8] utiliza restricciones de este tipo, aunque están restringidas a aplicaciones 2D y a aplicaciones gráficas desarrolladas enteramente en dicho toolkit, en el lenguaje C++.

3. *"Animadores"*

Una restricción de animación, o "animador" encapsula todos los detalles de implementación de la animación de un determinado objeto. Por medio de su uso, la especificación de la animación de una determinada variable consiste en la colocación de un "animador" como valor de dicha variable. De tal forma, cuando se coloque un nuevo valor a esta variable, el animador efectuará una transición gradual entre el valor previo y el nuevo de esta variable. Los animadores son entidades con un alto nivel de abstracción, ya que todos los detalles de la forma en que es implementada esta transición se ocultan al programador, con el fin de facilitar su tarea.

En esta forma, la incorporación de animaciones a objetos gráficos puede efectuarse de una forma sencilla por medio del uso de animadores. Por ejemplo, si se desea rotar un objeto gráfico, efectuando una animación, sólo será necesario colocar un animador como valor de la propiedad *'angle'* de dicho objeto gráfico. Es decir, solamente se deberá efectuar:

```
// crea un objeto gráfico
Cube c = new Cube();
// crea un animador con valor inicial 10
Animator anim = new Animator(10);
// coloca al animador como valor de la propiedad 'angle' del objeto gráfico
c.initValue(c.angle(), anim);
....
```

Posteriormente, cuando se coloque un nuevo valor para la propiedad 'angle':

```
c.setValue(c.angle, 90);
```

el animador efectuará una transición gradual del atributo 'angle' entre el valor inicial (10) y el nuevo valor colocado (90). La modificación gradual de tal atributo producirá una animación de tal objeto, la cual consistirá en una rotación.

Como se ve, la incorporación de un animador en la aplicación resulta simple, ya que la definición de la animación y la acción que da comienzo a ésta son independientes de los objetos gráficos que son animados.

Debido a la independencia de los objetos animados, los animadores pueden ser reutilizados en diferentes situaciones. En muchos casos, los animadores pueden requerir pequeñas modificaciones en su comportamiento, para adaptarse a los requerimientos de una aplicación particular. Para tales situaciones, los animadores proveen un conjunto de parámetros que permiten adaptar este comportamiento, tales como el tiempo de la animación y el tipo de transición a usar.

Para garantizar una mayor generalidad, se proveen distintos tipos de animadores, dependientes del tipo de la variable animada. Por ejemplo, podrán utilizarse animadores diferentes para variables continuas, discretas o booleanas. Estos animadores se encontrarán disponibles en bibliotecas, para que el programador pueda utilizarlos directamente o especializarlos de acuerdo a necesidades particulares.

Actualmente, los "animadores 3D" se encuentran implementados en el lenguaje *Java*, utilizando *Java3D* [16] como toolkit gráfico y al resolvedor *"Cassowary"* para la implementación y resolución de las restricciones convencionales.

3.1. Composición de un animador

Internamente, el animador posee distintos atributos que determinan el tipo de animación a

efectuar. Algunos de ellos son compartidos por todos los tipos de animadores, mientras que otros son característicos de cada tipo de animador particular. Entre los atributos comunes pueden citarse a: atributos propios de la animación (tales como la duración o la velocidad), una descripción de la

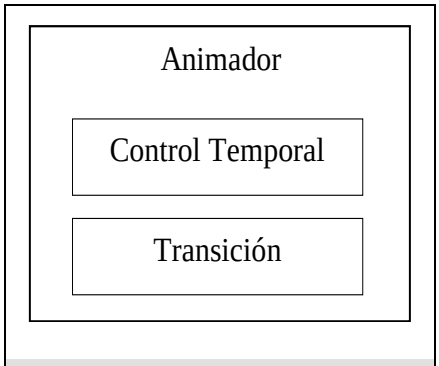

Figura 1 – Composición de un Animador

forma en la que debe efectuarse la transición, la variable animada, y los mecanismos que implementarán efectivamente la animación (tales como interpoladores).

La forma más simple de adaptación del comportamiento de un animador consiste en la especificación de la velocidad a la que debe efectuarse la animación, o indicar la duración de la misma. Ambos parámetros están contenidos dentro de un animador, pudiendo ser modificados por el programador en cualquier momento.

Además de indicar la duración o la velocidad, el programador puede suministrar distintas funciones para calcular la transición entre los dos valores (el inicial y el final). En su forma más simple, los animadores utilizan una interpolación lineal para determinar la transición; sin embargo, el programador puede suministrar una nueva función que determine otro tipo de transición. Por ejemplo, el programador puede suministrar una lista de valores por los cuales tendrá que pasar la animación. De esta forma, se provee flexibilidad a los animadores para especificar distintas animaciones.

De la misma forma que es posible especificar distintas funciones para determinar los valores por los que debe pasar una animación, también existe un control similar sobre los tiempos en los que se llevará a cabo la animación. De esta forma, es posible efectuar animaciones que no se ejecuten a una velocidad constante, sino que puedan tener intervalos con distintas velocidades o aceleraciones. Al igual que lo descripto en el párrafo anterior, el control del tiempo puede efectuarse por medio de la provisión de una función temporal determinada, o la indicación de las distintas velocidades que debe tener la animación en determinados puntos.

De esta forma, el programador posee mecanismos flexibles para especificar y adaptar el comportamiento de su animación. Incluso, estos diferentes comportamientos pueden ser almacenados en bibliotecas, para que puedan estar disponibles en futuras aplicaciones. Es decir, los animadores proveen reusabilidad y flexibilidad para especificar animaciones.

Los mecanismos utilizados para llevar a cabo efectivamente la animación se encuentran encapsulados en el animador. En la implementación actual, los animadores en su mayoría generalmente los interpoladores de Java3D para efectuar la animación. Para ello, además contienen una referencia a la variable animada, la cual es modificada en cada paso de la animación.

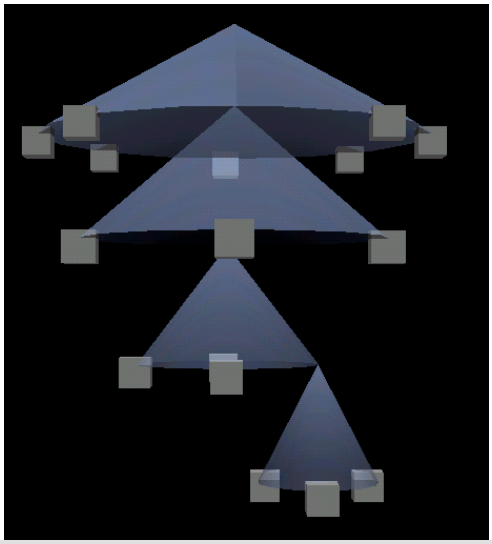

Figura 2 - *ConeTree* implementado con animadores3D (en Java3D)

4. Una Aplicación Ejemplo: *ConeTree*

El *ConeTree* [17] es una de las técnicas de visualización 3D más conocidas, incluida dentro del sistema *InformationVisualizer* [18]. Esta técnica consiste en una representación tridimensional de una estructura jerárquica, con el objetivo de permitir la visualización de grandes cantidades de información aprovechando las ventajas de la tercera dimensión.

Los nodos de la estructura son representados por medio de cubos, mientras que los subarboles son representados por conos semi-transparentes (para evitar la oclusión de los nodos colocados en la parte trasera).

Cuando el usuario selecciona un nodo (generalmente a través de un *picking* o una técnica similar), el subarbol en el cual se encuentra dicho nodo es rotado, para colocar al nodo seleccionado en el frente de la presentación. Los conos superiores en los que se encuentra el subarbol seleccionado también son rotados, para garantizar que todos los ancestros del nodo seleccionado también se encuentren en el frente de la presentación. Todas estas rotaciones son animadas en forma simultánea, para posibilitar una percepción sencilla al usuario de los cambios producidos.

4.1. Implementación del *ConeTree*

Una implementación básica del *ConeTree* debe contener clases para representar los elementos representados en el mismo: nodos y conos (subarboles). Cada uno de estos elementos tendrá una posición que dependerá de los radios de los distintos conos componentes del *ConeTree*. A su vez, estos radios dependerán de la cantidad de nodos contenidos en tal subarbol, tal como se detalla en la especificación original del *ConeTree* [17].

Las relaciones entre las posiciones de los nodos y los radios de los conos pueden ser expresada por medio de un toolkit convencional, como Java3D. Sin embargo, el mantenimiento de estas relaciones puede resultar complejo en el caso de modificarse la estructura del *ConeTree*.

Las restricciones convencionales constituyen un medio apropiado para la especificación de las relaciones entre los nodos y los radios de los conos. Por ejemplo, la localización de un

determinado nodo estaría determinada por una relación de la siguiente forma[1]:

```
x = parent.x + parent.radius * Math.sin(angle);
z = parent.z + parent.radius * Math.cos(angle);
y = parent.y – parent.high;
```

En este caso, estas restricciones especifican que la posición (en sus coordenadas x, z) de un nodo depende de la posición de su nodo padre (*parent*), del radio del cono que lo contiene (*parent.radius*) y el ángulo en el que se encuentra el nodo dentro del cono (*angle*). De una forma similar podrían expresarse otras propiedades de los elementos, tales como el ángulo en el que es colocado cada nodo dentro de un cono y las relaciones entre los radios de los conos.

Las animaciones de las rotaciones de los conos, ante una selección, pueden ser implementadas de diversas formas. En Java3D, una de las formas más sencillas consiste en utilizar "interpoladores", que efectúan todo el proceso de la animación. Una versión simplificada de este proceso consistiría en:

```
rotate() {
       // rota los conos superiores
       parent.rotate();
       // calcula el angulo de rotación
       finalAngle = angle – parent.angle;
       // crea el interpolador que efectuará la animación
       setInterpolator (angle, finalAngle);
       // coloca el valor actualizado del angulo rotado
       angle = finalAngle;
}

setInterpolator (startAngle, finalAngle) {
       interp = new RotationInterpolator(...,duration, ...);
       // coloca valores inicial y final de la interpolación
       interp.setMinimumAngle (startAngle);
       interp.setMaximumAngle (endAngle);
       // coloca el tiempo inicial de la animación
       interp.setStartTime(currentFrameStartTime, ...);
}
```

De esta forma, un interpolador se encarga de efectuar la rotación de cada cono particular. Las relaciones entre las posiciones de los distintos nodos son mantenidas, tal como fueron expresadas en las restricciones descriptas anteriormente.

4.2. Implementación con animadores

Las restricciones no solamente permiten expresar relaciones entre los objetos gráficos que componen una presentación. También pueden ser utilizadas para expresar relaciones con otros objetos componentes de la escena (tales como luces o la cámara).

Por ejemplo, si se desea mantener a un determinado nodo en el frente de la presentación del *ConeTree*, podría hacerse por medio de una restricción que fuerce a mantener dicho nodo en la misma dirección que la cámara:

```
node.parent.angle[2] = camera.angle;
```

[1] La sintaxis utilizada por el resolvedor *Cassowary* (utilizado en este trabajo) para especificar restricciones es mucho más compleja; por razones de legibilidad, aquí solamente se describe una versión simplificada de dichas restricciones.

[2] Se debe especificar el ángulo del nodo padre, ya que éste es quien conoce el ángulo en el que se encuentra rotado el cono que contiene al nodo que queremos mantener al frente de la presentación.

En esta restricción, la modificación de la dirección del campo de visión de la cámara producirá la modificación del ángulo en el que se encuentra localizado el nodo[3]. Sin embargo, la colocación del nodo en su nueva posición se producirá en una forma instantánea, ya que la restricción colocará el valor que la satisface en un solo paso. Es decir, no se efectuará ninguna animación entre la posición anterior y la nueva del nodo. Como consecuencia, el usuario no podrá percibir fácilmente los cambios producidos, produciendo casi seguramente una desorientación en su interacción.

Para animar los cambios producidos, puede utilizarse una restricción de animación, la cual producirá el efecto deseado. Para ello, se deberá colocar un animador como valor de la propiedad *angle*:

```
node.parent.angle = new Animator(...,initialValue,...);
```

En este caso, la colocación de un nuevo valor en el animador producirá una animación:

```
(node.parent).setValue (node.parent.angle(), finalAngle);
```

El animador se encargará internamente de efectuar la animación, a través de interpolar el valor de la variable *angle*, entre el valor previo y *finalAngle*. En la implementación actual, este proceso será efectuado por un interpolador, en una forma similar a la descripta en la sección 4.1.

Sin embargo, la utilización del animador evita la actualización inmediata del valor del ángulo, pero no mantiene la restricción entre dicho ángulo y la dirección de la cámara. Para poder mantener esta relación puede utilizarse un tipo especial de animador, denominado *"animador restringido"*. Estos animadores combinan el uso de las restricciones convencionales, con las restricciones de animación. En el caso de la cámara:

```
node.parent.angle = new ConstrainedAnimator (camera.angle(),
                                ...,initialValue,...);
```

En este caso, cada modificación en la dirección de la cámara producirá una modificación en el valor del ángulo del nodo. Sin embargo, a diferencia de las restricciones convencionales, la actualización de este valor no se efectuará en forma inmediata, sino que se realizará por medio de una interpolación entre los valores inicial y final.

De esta forma, combinando el poder de ambos tipos de restricciones es posible especificar simplemente un comportamiento dinámico como el descripto. El modelo de animadores provee otros tipos de restricciones de animación, para adaptarse a situaciones particulares de cada aplicación. Varios de estos tipos son derivados del estudio de las características de las animaciones tridimensionales, como se describe en las siguientes secciones.

5. Tipos de animadores

Los animadores pueden actuar sobre atributos numéricos, tales como posiciones y tamaños, pero también es posible que actúen sobre otros tipos de atributos. También es posible que efectúen animaciones de atributos conteniendo colores, valores discretos, dibujos, etc. La mayor ventaja en todos estos casos, es que no existe ninguna necesidad de efectuar modificaciones sobre los objetos gráficos a ser animados. Las clases de los atributos sobre los cuales se define la animación, nos permiten establecer una clasificación entre los animadores.

Los animadores numéricos son aquellos que trabajan sobre variables conteniendo valores enteros o reales. En este tipo de animadores en particular, el programador puede especificar el

[3] Debido a que *Cassowary* solamente provee restricciones unidireccionales, la modificación del ángulo de la cámara producirá la modificación del ángulo del nodo, pero la situación inversa no se producirá. Es decir, la modificación de la posición del nodo no tendrá repercusiones en la dirección de la cámara.

incremento deseado en el valor de la variable para cada paso de la animación (cada paso producirá un *"frame"* en la animación). También es posible especificar la cantidad de pasos (o *frames*) que se desea efectuar. El animador mostrado en la sección anterior es un ejemplo de un animador numérico (en este caso, animando la variable entera *'angle'*)

Asimismo, existen animadores que trabajan sobre una lista de valores, dispuestos con un orden dado. En este caso, el animador irá tomando cada uno de los valores de la lista en forma sucesiva. Este tipo de animadores es útil para efectuar animaciones de íconos, en los que los sucesivos estados de un icono (representados por imágenes diferentes) estarán indicados en el valor afectado por el animador. Además de estos estados, el programador puede especificar el tiempo que debe permanecer el animador en cada estado (es decir, el tiempo que se mostrará cada imagen diferente del icono). El programador también puede indicar la acción a tomar al llegar al último valor de la lista (por ejemplo: detenerse, volver al estado inicial, o efectuar la animación en reversa).

También existen animadores particulares para la visibilidad de los objetos y para colores. En los primeros, pueden efectuarse animaciones sobre el atributo 'visible' de un objeto gráfico dado, para lograr efectos tales como 'desaparición', ´difusión´, ´salida de la pantalla´, etc. En el caso de los colores, existen animadores que efectúan la interpolación entre dos colores dados (indicados por sus coordenadas RGB).

De la misma forma que existen animadores de los tipos descriptos, el futuro desarrollo de aplicaciones permitirá determinar la necesidad de contar con nuevos tipos de animadores. Con los prototipos desarrollados actualmente, ha sido posible determinar algunos tipos de animadores y consideraciones que deben ser tratadas en el caso de animaciones 3D. En la siguiente sub-sección se describen algunos de estos aspectos.

5.1. Características de importancia en animaciones 3D

Si bien el concepto de animador es independiente de si se trata de una presentación 2D o 3D, existen varios aspectos requeridos en el caso de 3D que deben ser considerados para poder efectuar animaciones poderosas y eficaces. Varios de los aspectos descriptos a continuación son aplicables en ambos tipos de presentaciones, pero su presencia en el caso tridimensional es de suma importancia.

En las interfaces 3D surgen varios aspectos que no se encuentran presentes en las interfaces 2D. Uno de ellos es la oclusión que puede ocurrir entre los objetos de una presentación. La oclusión debe ser altamente considerada en el diseño de animaciones, ya que si se la ignora puede producir efectos no deseados. Por ejemplo, la utilización de una restricción de animación convencional puede producir que la trayectoria del objeto animado produzca una oclusión en la presentación (un caso típico es un objeto desplazándose entre la cámara y la escena). En este caso, el programador debiera poder especificar un camino alternativo que no produzca la oclusión mencionada. Los animadores permiten que el programador indique un camino particular para la animación del objeto. Este camino podría tener distintas formas: un recorrido en forma de arco, circular, recto, o incluso una forma arbitraria (que puede ser especificada por medio de una lista de puntos).

Sin embargo, la oclusión no es un efecto que pueda ser determinado siempre en el momento de especificar la animación. Como la oclusión depende de varios factores (la posición de los objetos, la cámara, el campo de visión, etc.), en la mayoría de las situaciones no puede ser prevista. En tal caso, el suministro de un camino determinado para la animación no es suficiente. Por tal razón, los animadores 3D proveen la posibilidad de determinar la ocurrencia de determinadas condiciones. Esto permite detectar situaciones de interés, además de la oclusión, tales como las colisiones entre objetos, la detección de proximidades, o la entrada de un objeto en un determinado espacio.

El programador puede indicar al animador las condiciones que desea detectar, así como también las acciones a tomar ante tales condiciones. Estas acciones pueden tener efectos diversos, tales como la toma de un camino alternativo, o la anulación de la animación (restaurando el objeto a su estado original).

De la misma manera que existe flexibilidad para la especificación del recorrido de una animación, también debe ser flexible la especificación de los tiempos de la animación. Los animadores 3D permiten la especificación de una lista de intervalos de la animación, indicando la velocidad o los tiempos en los que debe animarse cada intervalo.

Por último, la cámara es un concepto fundamental en las presentaciones 3D. La provisión de animaciones suaves de las navegaciones efectuadas por el usuario es de primordial importancia para evitar la desorientación del usuario. En numerosas situaciones, el comportamiento de las animaciones depende fuertemente de la posición de la cámara. La oclusión (como se citó anteriormente) es una de estas situaciones. Otra situación típica es mantener un determinado conjunto de objetos siempre visibles. Para ello puede aprovecharse el poder combinado de las restricciones de animación con las restricciones convencionales.

La cámara también suele ser utilizada para efectuar el seguimiento de determinados objetos en una escena. En tal caso, la posición de la cámara debe mantenerse vinculada con dichos objetos. Si estos objetos son animados o pueden ser manipulados interactivamente, la restricción de animación debe efectuar el reposicionamiento de la cámara ante cada movimiento de estos objetos. El ejemplo descripto en la sección 4.2 corresponde a este tipo de animadores.

En 3D, es posible definir algunos tipos de animadores que permitan efectuar algunas animaciones en una forma más eficiente. Por ejemplo, para rotar un objeto (como en el caso del *ConeTree*) es posible utilizar un animador numérico sobre el atributo ´angle´. Sin embargo, también podría implementarse un animador de rotación especial para tal atributo, en el que la rotación sea efectuada más eficientemente. Por ejemplo, Java3D provee interpoladores especiales de rotación (*RotationInterpolators*), los cuales permiten rotar elementos solamente indicando el subgrafo de la escena afectado por la rotación. Un animador de rotación especializado podrá aprovechar las ventajas ofrecidas por tal tipo de interpolador. De la misma manera, pueden definirse animadores especiales para traslaciones y escalados de objetos gráficos.

6. Estado Actual

Como se dijo anteriormente, la implementación de los animadores fue desarrollada en Java, utilizando el API Java3D, y el resolvedor *Cassowary*. Hasta el momento, se han desarrollado algunos prototipos simples de algunas técnicas de visualización 3D, tales como el *ConeTree*. Estos prototipos han permitido efectuar un estudio inicial del concepto de *"animador 3D"*, determinando su aplicabilidad, posibles tipos y aplicaciones, y los requerimientos particulares impuestos por el uso de la tercera dimensión.

Actualmente, se está estudiando la aplicación de este concepto a otras técnicas de visualización, con características dinámicas. En particular, se prevé la aplicación de los animadores a simulaciones interactivas.

7. Conclusiones

En este trabajo, se ha descripto el concepto de *"animador 3D"* o *"restricciones de animación"*. Éstas son restricciones especializadas para efectuar animaciones en presentaciones 3D, que permiten realizar una transición gradual entre dos valores (a diferencia de las restricciones convencionales, que actualizan un valor en forma inmediata).

En particular, la combinación de animadores y restricciones convencionales resultan útiles para especificar comportamientos dinámicos (como el descripto para el caso de la cámara del *ConeTree*). La mayor ventaja de este modelo es que facilita al programador la especificación de animaciones, ya que solamente debe especificarse la variable a animar. Todos los detalles de implementación de una animación están completamente encapsulados dentro de cada animador. El programador puede adaptar parte de este comportamiento, mediante el suministro de distintos parámetros al interactor.

El encapsulado de distintos tipos de animaciones en "animadores" posibilita su reuso y especialización. La tarea del programador de animaciones puede verse facilitada por la disponibilidad de una biblioteca de animadores, que puedan ser utilizados en distintas aplicaciones.

La aplicación de este concepto en visualizaciones 3D ha permitido identificar determinadas características que deben ser provistas por los animadores para este tipo de interacciones. Entre ellas, se destacan la detección de determinadas condiciones (ej. la visibilidad de determinados objetos, o la detección de colisiones), la posibilidad de definir alternativas para la animación efectuada, y la consideración de la cámara como un factor muy importante en estas presentaciones.

8. Referencias

[1] "*Computer Graphics Principles and Practice*", James D. Foley, Andries van Dam, Steven K. Feiner y John F. Hughes. Addison-Wesley Publishing Company. Segunda Edición. 1997.

[2] "*Information Visualization: Using Vision to Think*" Card, S. Mackinlay, J. Shneiderman, B. Morgan Kauffman, 1999.

[3] "*Visualization: Using Computer Graphics to Explore Data and Present Information*" J. Brown, R. Earnshaw, M. Jern, J. Vince. John Wiley, 1995, 290 pp.

[4] "*OpenGL Programming Guide*" J. Neider, T. Davis, M. Woo, Addison-Wesley, 1993, 520 pp.

[5] "*The Inventor Mentor: Programming Object Oriented 3D Graphics with Open Inventor*" Wernecke, J. Addison-Wesley, 1995.

[6] "*Graphics Programming with Direct3D: Techniques and Concepts*". Glidden, R. Addison-Wesley, 1996.

[7] "*The Cassowary Linear Arithmetic Constraint Solving Algorithm: Interface and Implementation*". Badros, G. Borning, A. Washington Univ.

[8] "*Easily Adding Animations o Interfaces Using Constraints*". Myers, B. Miller, R. McDaniel, R. Ferrency, A. Proc. UIST 96, pp. 119-128, 1996.

[9] "*JDCAD: A Highly Interactive 3D modelling system*" Liang, J. Green, M. Computer and Graphics, 18(4), pp. 499-506, 1994.

[10] "*Animation Support in a User Interface Toolkit: Flexible, Robust and Reusable Abstractions*" Proc. UIST 93, 1993. pp 57-67.

[11] "*Animations: from Cartoons to the User Interface*" Proc. UIST 93, 1993, pp. 45-55.

[12] *"SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction"* Sannella, M. Proc. UIST'94, pp. 137-146.

[13] *"An interactive 3D toolkit for constructing 3D widgets"* Zeleznik, R. Herndon, K. Robbins, D. Huang, N. Proc. Siggraph 93, pp 81-84, 1993.

[14] *"An integrated environment to visually construct 3D animations"* Gobbetti, E. Balaguer, J. Proc. Siggraph 95, pp. 395-398, 1995.

[15] *"Constraint-Based Animation: Temporal Constraints in the Animus System"* Duisberg, R. Univ. of Washington, 1995.

[16] *Java3D Specification.* Deering, M. Sowizral, H. Sun Microsystems, 08/1997. java.sun.com/products/java-media/3D/

[17] *"Cone Trees: Animated 3D Visualizations of Hierarchical Information"* Robertson, G. Card, S. Mackinlay, J. Proc. CHI 91, 1991.

[18] *"Information Visualization using 3D Interactive Animation"* Robertson, G. Card, S. Mackinlay, J. Comm. ACM. 4/1993. pp. 57-71.