TEMA 6 : TÉCNICAS PARA SIMULACIÓN EN TIEMPO REAL

Como hemos comentado al hablar de la animación por ordenador, la principal característica de la simulación en tiempo real es que la aplicación informática debe mostrar las imágenes sintéticas a medida que se van produciendo, de forma que reflejen los cambios producidos por las acciones del usuario sobre el programa (concepto de *interactividad*). Estos cambios suelen responder a la representación de un fenómeno real cuya evolución temporal se intenta replicar (por ejemplo, el vuelo de un avión manejado por un piloto). El principal desafío desde el punto de vista gráfico es, por tanto, optimizar el coste de los cálculos necesarios para realizar la visualización. Para conseguir dibujar varios fotogramas por segundo debe utilizarse casi obligatoriamente el método de procesamiento gráfico de proyección y rasterización que vimos en el tema 3. Como veremos, normalmente las técnicas utilizadas para conseguir la disminución del coste se basan en controlar el número de objetos a visualizar, el número de polígonos que los forman y el coste de rellenar estos polígonos.

6.1 CONCEPTO DE TIEMPO REAL

Este concepto se puede aplicar a distintas ramas de la Informática y proviene de la ingeniería de control. En general la caracterización como 'tiempo real' se refiere a la existencia de determinadas restricciones sobre el comportamiento temporal de nuestro sistema. Dependiendo del tipo de aplicación, esas limitaciones serán de una clase u otra. El concepto de tiempo real, por tanto, puede aplicarse de forma más o menos estricta según lo duras que sean las restricciones impuestas.

El **tiempo de respuesta (T_r)** se define como aquel período de tiempo que transcurre entre la entrada de un dato y la obtención de una salida (ver figura 6.1). En un sistema de simulación visual representaría el tiempo que transcurre desde que el sujeto efectúa una acción (como mover el ratón) hasta que, después de efectuarse los cálculos correspondientes al modelo matemático del proceso y la visualización, se presentan las imágenes resultantes. Una de las condiciones que podemos poner para hablar de tiempo real es limitar este parámetro a un valor máximo permitido (no queremos que haya un retraso mayor de un cierto intervalo), o incluso imponer que el tiempo de respuesta sea constante en todo momento.

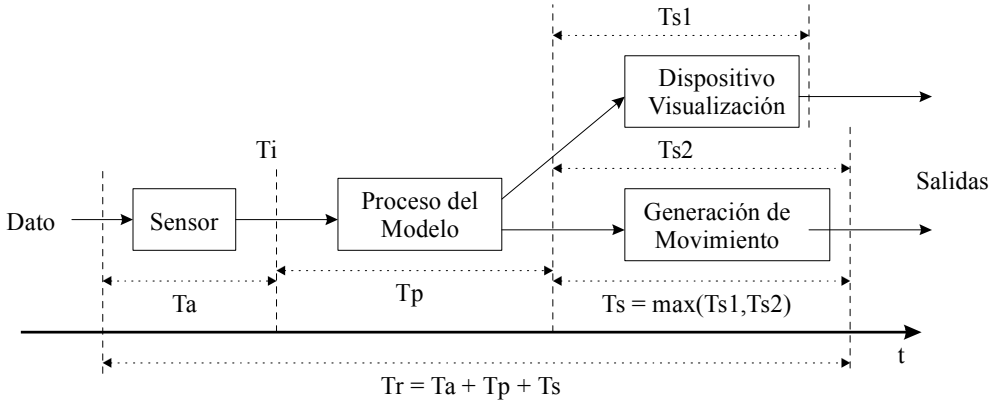

Figura 6.1.: Influencia de las diferentes partes del proceso de simulación en el tiempo de respuesta Tr.

La restricción de este tiempo de respuesta (o 'retraso de transporte') está en la base del concepto de interactividad. En ciertas aplicaciones de simulación la sensación del sujeto de sentirse 'inmerso' en un escenario realista puede depender de este parámetro. Por ejemplo, en un sistema de realidad virtual en el que se usa un casco estereoscópico con localizador de posición, si el retraso observado entre el giro real de la cabeza del sujeto y los cambios correspondientes en la imagen es mayor de una décima de segundo, resulta difícil mantener la ilusión de inmersión.

Aunque pueden estar relacionados, la **frecuencia de salida** es un parámetro diferente al retraso, y depende del intervalo de respuesta entre dos salidas consecutivas del sistema (por ejemplo, si se presenta una nueva imagen cada 50 milisegundos, la frecuencia de refresco de los gráficos es de 20 Herzios, veinte imágenes por segundo). Si el sistema no acepta una nueva entrada de datos hasta que no ha terminado de procesar la anterior, entonces la frecuencia de salida resulta ser precisamente el inverso del tiempo de respuesta. Pero también puede pasar que el sistema vaya procesando entradas de datos con una frecuencia diferente a la que se produce la salida. Por ejemplo, en un simulador de conducción puede leerse la posición del freno cada milisegundo (1000 Hz), pero producirse una salida del sistema de movimiento (plataforma móvil) 120 veces por segundo y una salida gráfica 30 veces por segundo (estos dos subsistemas tendrían diferentes tiempos de respuesta).

En el caso de la representación gráfica, la frecuencia de salida constituye lo que habitualmente se conoce como **frecuencia de refresco** o ***frame rate***. En muchos simuladores es habitual especificar la restricción de que haya una frecuencia de refresco mínima, e incluso constante.

Además de las constricciones relativas al tiempo de respuesta y la frecuencia de salida, existen otras restricciones temporales independientes: la correspondencia entre el **tiempo aparente** de la simulación y el **tiempo físico** del observador (los objetos deben moverse en la simulación con la misma velocidad que lo harían en el mundo real, el tiempo interno utilizado por el modelo de la simulación debe tener la misma escala - o un factor constante y conocido - que el tiempo físico); y también es importante la **sincronización** entre diferentes salidas que corresponden al mismo fenómeno (por ejemplo, el sonido y los gráficos).

6.2 FACTORES DE COSTE EN LA VISUALIZACIÓN

Cada una de las fases en las que se divide la pipeline o proceso de visualización que vimos en el tema 3, tiene un coste asociado. Podemos hablar de tres tipos básicos de costes:

- Coste de las operaciones **por vértice** (proyección, iluminación, clipping): el coste será más o menos proporcional al número de vértices que se han enviado desde la base de datos a la escena, el número de luces y el modelo de iluminación. En principio el número de vértices es constante, pero como veremos, podemos representar a los objetos con diferentes niveles de detalle (con diferente número de vértices y diferentes propiedades visuales) y escoger en cada momento el más conveniente.

- Coste de las operaciones **por pixel** (rellenado $\rightarrow$ comparación z-buffer, interpolación del color, texturas, transparencia): es proporcional al número de pixel a rellenar y depende también de las características del objeto que afecten a las operaciones por pixel, como el modelo de iluminación (plano o Gouraud), si tiene textura, transparencia, etc.

- Coste de las **transferencias de datos** entre diferentes partes de la pipeline, o de los accesos a memoria. En muchos casos este coste no puede despreciarse, especialmente cuando la transferencia de datos se produce a través de un canal compartido por otros recursos del sistema (p.ej. el bus de datos), o cuando el sistema debe hacer transferencias de páginas de memoria (por ejemplo, de texturas).

En los gráficos en tiempo real, el coste final de todas las operaciones depende fuertemente de su implementación, y fundamentalmente del hardware encargado de ejecutarlas (razón por la cual, un sistema gráfico capaz de generar 60 Hz. con escenas complicadas puede valer varias decenas de millones de pesetas). Evidentemente, no todas las fases de la pipeline tardarán lo mismo, de modo que finalmente la frecuencia de salida vendrá determinada por la fase más lenta, produciéndose el efecto de cuello de botella o 'bottleneck'.

6.3 TÉCNICAS DE OPTIMIZACIÓN DEL COSTE

Las técnicas de optimización del coste se basan en el control de las características de la escena relacionadas con el número de vértices y las operaciones por pixel, teniendo en cuenta además la arquitectura del sistema gráfico y sus limitaciones en la transferencia de datos. Por tanto, no es tan importante controlar cómo es la descripción original de la escena, sino más bien cómo son los datos de la escena que se envían al sistema de procesamiento gráfico.

6.3.1 SELECCIÓN DE OBJETOS SEGÚN LA POSICIÓN DEL OBSERVADOR EN UNA PARTICIÓN ESPACIAL

La escena puede contener muchos objetos en zonas alejadas o aisladas del entorno visible para el observador. Un ejemplo típico aparece en visualización de interiores arquitectónicos; hay muchas habitaciones, pero el observador solamente puede ver en cada momento aquellos objetos que están dentro de la sala donde él se encuentra.

Mediante cualquiera de los métodos de partición espacial estudiados podemos organizar los objetos en zonas y enviar al sistema de visualización solamente aquellos que están en las zonas accesibles. Esta forma de selección supone tener información a priori sobre la distribución de los objetos y las relaciones de visibilidad entre las diferentes zonas.

6.3.2 COMPROBACIÓN DE VISIBILIDAD

En general un objeto puede no ser visible desde la posición del observador por dos motivos, bien porque se encuentre fuera del campo de visión, caso que es fácil comprobar sin un gran coste computacional; o bien porque, aún estando dentro del campo de visión, aparece totalmente ocluido por otro objeto. En este segundo caso la comprobación de la condición de visibilidad supone un mayor coste, pues habría que proyectar la silueta del objeto que puede ser ocluido hasta el punto de observación y ver si es completamente tapada o no por otros posibles objetos.

En cualquiera de estos dos casos podríamos llegar a determinar que un objeto no va a ser visible, dejando de enviar sus datos a la pipeline gráfica. Si el sistema de visualización recibiera los datos de estos objetos ‘invisibles’ no llegaría nunca a rellenar los pixels, pero sí necesitaría proyectar sus vértices para saber si caen fuera o dentro de la ventana (en la fase de recorte o *clipping*), y en el caso de un objeto ocluido (que sí cae dentro del campo de visión) también debería hacer la comparación de z-buffer con cada pixel cubierto para poder determinar su no-visibilidad. Por tanto, al dejar de enviar sus datos estamos evitando todas esas operaciones

6.3.3. SELECCIÓN DEL NIVEL DE DETALLE DE LOS OBJETOS

Cuando el coste total de la escena es excesivo, podemos intentar ‘simplificarla’, enviando al sistema de visualización representaciones menos detalladas de los objetos. Normalmente estas diferentes representaciones de los objetos (a veces llamadas "niveles de detalle" LOD: *Level Of Detail*) han sido generadas previamente. Lo lógico será representar con más detalle (mayor número de polígonos, mejor iluminación, texturas de más calidad) aquellos objetos que en cada instante se consideren más importantes para la percepción de la escena. Como veremos, el criterio más empleado para elegir el nivel de detalle de cada objeto es la distancia a la que se encuentra del punto de observación, aunque también se pueden emplear otros como la velocidad transversal del objeto o su tamaño relativo, que pueden dar idea de su importancia dentro de la imagen.

6.4. ESTRUCTURAS DE DATOS JERÁRQUICAS PARA VISUALIZACIÓN EN TIEMPO REAL

Algunas librerías gráficas para tiempo real (como OpenInventor o IRIS Performer de Silicon Graphics) definen estructuras de datos jerárquicas para organizar y optimizar la base de datos con el fin de disminuir el coste de visualización. En general todas estas librerías contemplan mecanismos de selección de objetos como los que hemos mencionado, implementándolas gracias a funciones automáticas asociadas a los nodos de la estructura jerárquica.

Las estructuras de datos jerárquicas empleadas en visualización son grafos acíclicos (similares a un árbol pero permitiendo que dos nodos compartan hijos). El algoritmo de visualización recorre esta estructura de forma recursiva a partir del nodo raíz y selecciona cuáles son los datos enviados al sistema de procesamiento gráfico. La razón para permitir que dos o más nodos puedan compartir hijos es ahorrar espacio de almacenamiento. Si queremos dibujar varias copias del mismo objeto podemos colocarlo como hijo de varios nodos, con lo cual el algoritmo de recorrido llegará hasta él varias veces, por caminos diferentes y se dibujarán varias copias.

Los nodos de la jerarquía de la escena contienen información de diversos tipos, y ejercen diferentes funciones.

6.4.1. NODOS VISUALES

Estos nodos contienen los datos que van a enviarse al sistema de procesamiento gráfico para dibujar los objetos. A su vez pueden dividirse en dos tipos:

a) Nodos de geometría. Contienen datos sobre la forma geométrica del objeto, utilizando primitivas poligonales y sus correspondientes vértices. Cada vértice suele estar caracterizado por la posición y otros parámetros opcionales como el valor del vector normal, el color o las coordenadas de textura.

Cuando el algoritmo de recorrido llega a un nodo de geometría, normalmente los nodos terminales del grafo, sus datos son enviados a la pipeline, y los polígonos correspondientes son dibujados. Los nodos de geometría no suelen tener hijos, y a veces la librería gráfica lo prohibe expresamente.

b) Nodos de propiedades. Describen cómo debe alterarse alguna propiedad u opción visual del contexto gráfico cuando el algoritmo de recorrido llegue a ese nodo. Se utilizan, por tanto, para especificar materiales, texturas, el modelo de iluminación y otras variables del contexto gráfico que determinan la apariencia final de los objetos.

Por ejemplo, cuando el algoritmo de recorrido de la jerarquía llega a un nodo de propiedades que especifica el material, se cambiará el material actual en el contexto gráfico. Lo habitual es que cuando el algoritmo haya recorrido toda la rama que cae bajo este nodo y siga el recorrido en otro lugar del árbol, se restaure en el contexto gráfico el material que hubiera antes. Pero esto no es siempre así. Por ello habrá que tener en cuenta para cada librería gráfica y para cada tipo de nodo si éstos tienen *delimitado su alcance* a sus hijos o su efecto es *acumulativo* (al volver de la recursión no se restaura el estado anterior). En este último caso, más difícil de controlar, hay que tener en cuenta el orden de recorrido del grafo, que es usualmente primero en profundidad (depth-first).

6.4.2. NODOS DE OPERACIONES

Estos nodos no contienen ningún dato sobre los objetos. Su misión es controlar el algoritmo de recorrido, es decir, seleccionar cuáles de sus hijos van a seguir recorriéndose. Son estos nodos los que van a permitir implementar las operaciones de optimización del coste.

a) Nodos de grupo. Todos sus hijos se recorren. Tiene solamente la misión de agrupar varias ramas (objetos) para organizar la base de datos permitiendo definir operaciones o propiedades sobre estas agrupaciones cuando el grupo cuelga bajo los nodos apropiados.

b) Nodos de selección por visibilidad y nodos de partición. El objetivo de estos nodos es obligar al algoritmo de recorrido a hacer una comprobación de visibilidad. En el caso de los nodos de selección

simple, si esa comprobación falla (la parte de la escena no es visible) no se recorrerá ninguno de los hijos del nodo, y si es visible se recorrerán todos. En el caso de los nodos de partición, éstos agrupan a sus hijos por zonas espaciales, y solamente se recorrerán aquellos que pertenezcan a zonas visibles.

Como comentamos, solamente la comprobación de visibilidad dentro del campo de visión suele ser practicable en tiempo real. Para que esta comprobación pueda realizarse, el nodo de visibilidad o partición debe describir el volumen de la zona de la escena cuya visibilidad debe comprobarse. Usualmente estos volúmenes envolventes (*bounding box*) son esféricos o en forma de caja rectangular. Los nodos de visibilidad pueden anidarse, formando una jerarquía de volúmenes envolventes que permite refinar la comprobación con las partes de los objetos, de la misma manera que se hacia para comprobar la intersección con un rayo (ver apartado 4.2).

c) Nodos de nivel de detalle (LOD). Estos nodos se utilizan para implementar la selección de detalle. Un nodo LOD forzará al algoritmo a efectuar una comprobación que escogerá uno solo de sus hijos para ser recorrido. Se supone que cada hijo representa el mismo objeto pero con un diferente nivel de detalle. Dependiendo de la librería se empleará un criterio u otro para la selección. Lo más normal es emplear la distancia al observador.

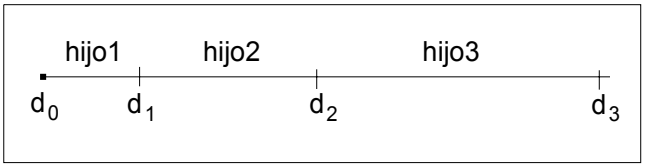

Figura 6.4.2.: Representación de un nodo LOD.

Para efectuar la comprobación de distancia estos nodos necesitan almacenar cierta información adicional:

- **Centro** : el punto desde el cual se mide la distancia del objeto al observador.
- **Rangos de distancia** para cada hijo, es decir, entre qué distancias es visible cada uno de los niveles de detalle. Normalmente se da una lista ordenada de valores de distancia, para los cuales se cambia de un hijo a otro (ver figura 6.4.2.).

El problema que presenta la técnica de selección de detalle es que, al variar la distancia, el observador podrá ver una transición brusca entre dos niveles de detalle. Para solucionarlo se pueden emplear dos técnicas: el fundido o *fading* que realiza una mezcla progresiva de las dos imágenes correspondiente a los dos niveles de detalle vecinos, obteniéndose buenos resultados pero incrementando el coste de la visualización (momentáneamente se dibujan dos de los hijos en lugar de uno); y la técnica de *morphing* que permite cambiar suavemente desde la geometría de un nivel de detalle a la del otro. Para ello hay que describir a los objetos mediante una representación de tipo progresivo (ver apartado 5.5.2).

d) Nodos de control por programa. Cuando se desea realizar un cambio en la escena por otro criterio distinto de los anteriores (por ejemplo, porque la simulación cambia el estado visual de un objeto) podemos utilizar un nodo de este tipo. El nodo tiene un campo que indica el índice del hijo que deberá ser recorrido, y este valor puede cambiarse por programa.

6.4.3. NODOS DE TRANSFORMACIÓN

a) Transformación del sistema de coordenadas. Se indica una operación de transformación afín (rotación, traslación, escalado) que se representa internamente como una matriz 4x4, y que afectará al sistema de coordenadas actual. Podríamos considerarlo, por tanto, como un tipo especial de nodo de propiedades, con el mismo problema de delimitar su alcance. Se emplean a menudo para obtener varias copias visuales de un mismo un objeto modificando su posición y tamaño.

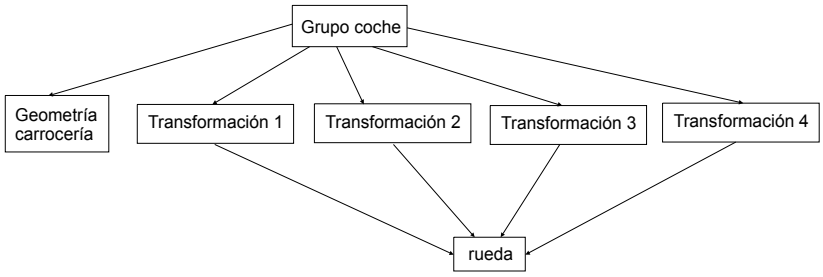

Figura 6.4.3.A.: Ejemplo nodo de transformación de sistema de coordenadas.

b) Nodos de articulación. Estos nodos sirven para implementar las operaciones de transformación ligadas a una estructura articulada. Ya comentamos su funcionamiento en el tema 5. Internamente se comportarán de forma muy similar a los de transformación del sistema de coordenadas.

c) Nodos especiales para animación. Dependiendo de las librerías gráficas podemos encontrar otros nodos que especifican relaciones entre parámetros de otros nodos (sirven para implementar ligaduras) o describen el cambio en el tiempo de un parámetro o de un objeto(generadores de funciones temporales).

d) Manipuladores (*handlers*). Se trata de un tipo especial de nodos de transformación de coordenadas que llevan asociado un objeto visual (aparte de sus correspondientes hijos). Este objeto visual, el manipulador, permite al usuario dar valores de forma interactiva a la transformación. Se emplean en contextos en los que el usuario edita objetos de la escena, como en un programa de modelado (por ejemplo un manipulador de *trackball* en la librería Inventor) .

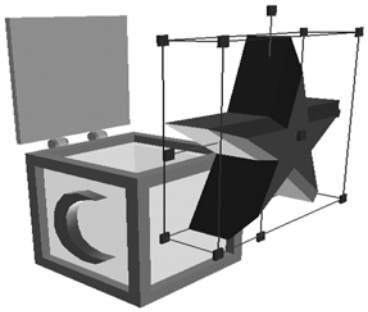

Figura 6.4.3.B.: Edición de un objeto en inventor con un manipulador X asociado.

6.4.4. NODOS DE RETROALIMENTACIÓN

Estos nodos provocarían una cierta acción cuando el algoritmo de recorrido llega hasta ellos.

a) Nodos de feedback de información. Permiten que el programa obtenga información sobre los nodos, que pasan datos a ciertas funciones configurables por el programador. Por ejemplo, un nodo de este tipo podría utilizarse para saber la posición de un cierto objeto cada vez que es dibujado (cuál es el estado del sistema de coordenadas de la escena cuando el algoritmo de recorrido llega al nodo).
b) Nodos de selección. Cuando el usuario del programa seleccionara con el ratón un objeto de la jerarquía que tuviera un nodo de este tipo, se llamaría automáticamente a una función del programa. Una aplicación sencilla es hacer que esta función añada un manipulador a un objeto cuando éste es seleccionado, permitiendo al usuario editarlo.

c) Nodos para enlaces de navegación. Sería un tipo especial de nodo de selección que llamaría automáticamente a un enlace dentro de un sistema de navegación multimedia. El ejemplo más conocido

es el del formato VRML (Virtual Reality Modeling Language) que contiene nodos con llamadas similares a las de HTML (</A HREF…>).

d) Nodos de callback o llamada a funciones. Este sería el tipo de nodo de retroalimentación más general, a partir del cual se pueden implementar todos los tipos anteriores y otros. Estos nodos llaman a una función definida por el programador cuando el algoritmo de recorrido de la escena pasa por él. Permite, por tanto, cambiar totalmente el algoritmo de recorrido usando funciones propias; implementando, por ejemplo, otros sistemas de selección de detalle o de visibilidad que consideremos más apropiados.

6.5. CONTROL DE LA FRECUENCIA DE REFRESCO

Ya hemos comentado que una de las características de los sistemas gráficos en tiempo real es la restricción de la frecuencia de refresco, que debe intentar mantenerse en unos límites adecuados. Así, podemos ver el proceso de generación de imágenes como un ejemplo de problema de control, donde queremos obtener una salida con una cierta característica deseada, en nuestro caso el número de imágenes por segundo. Las posibilidades de control aparecen desde el momento en que podemos variar, para la misma escena, los datos enviados a la pipeline gráfica, y por tanto ajustar el tiempo que se tardará en procesarlos. Usualmente esta modificación se efectúa haciendo variar el nivel de detalle de los objetos.

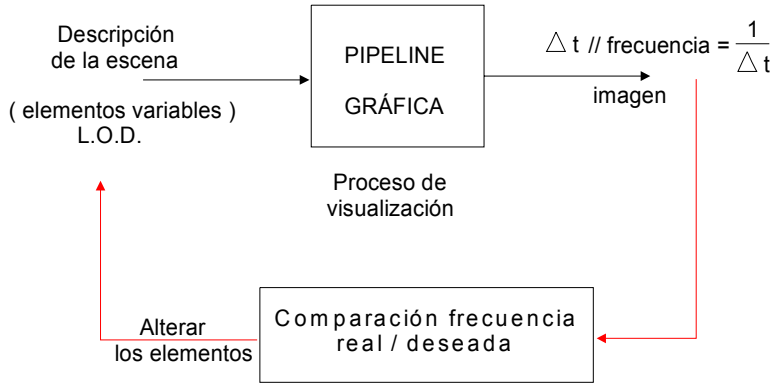

Figura 6.5.: Control de frecuencia de refresco.

Como en todo proceso de control podemos enfocar el problema según dos métodos:

⇒ Control en lazo cerrado clásico aplicado al control de gráficos: Método adaptativo del control de la frecuencia.
⇒ Control en lazo abierto: Método predictivo.

6.5.1. MÉTODO ADAPTATIVO

En lazo cerrado se compara el valor producido por el sistema con el valor que desearíamos obtener, y se intenta hacer las modificaciones adecuadas sobre la entrada del módulo controlado. Como ejemplo, vamos a ver el funcionamiento del control adaptativo tal como se ha implementado en la librería IRIS Performer.

Se define el **factor de estrés** como un número real positivo:

$$f.e. = \frac{f_{deseada}}{f_{real}} \quad \begin{cases} f.e. > 1 & \rightarrow \text{frecuencia real menor que la deseada (sobrecarga)} \\ f.e. < 1 & \rightarrow \text{frecuencia real mayor que la deseada} \end{cases}$$

Para obtener la frecuencia de refresco deseada, el factor de estrés debería ser exactamente uno, pero esto es una situación ideal. Si sólo deseamos poner un valor mínimo a la frecuencia, basta con requerir que el estrés sea menor o igual que uno.

En el bucle de retroalimentación debemos utilizar el valor del estrés para cambiar el nivel de detalle de los objetos. En el caso de Performer lo que se hace es alterar los rangos de distancia de los nodos de LOD, de manera que sea más probable que los objetos se representen con mayor o menor detalle que antes. Los umbrales de distancia aplicados por el algoritmo de recorrido serán:

$$d_i' = \frac{d_i}{f.e.}$$

Cuando haya sobrecarga del sistema gráfico, el factor de estrés será mayor que uno, y las distancias efectivas resultarán más pequeñas (ver figura 6.5.1), con lo que probablemente disminuirá el nivel de detalle de los objetos en la escena, ya que para una misma distancia se escogerán intervalos con índice igual o más alto, y por tanto con menor detalle. A la inversa, si el sistema tiene recursos sobrados y el estrés es menor que uno, aumentarán los rangos de distancia y por tanto, el detalle.

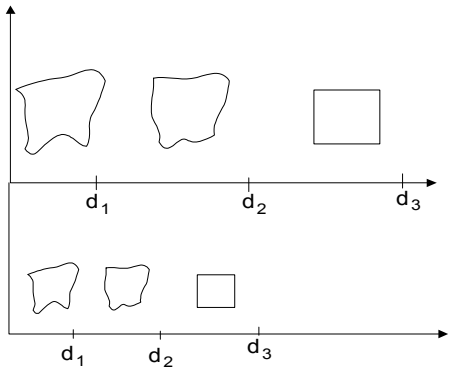

Figura 6.5.1.: Niveles de detalle de un objeto según la variación de los rangos de distancia.

Como vemos, se trata de una técnica relativamente sencilla y fácil de implementar. Su principal ventaja es que no conlleva ningún tipo de dependencia respecto a las características específicas de cada equipo gráfico. Sin embargo, no resulta fácil obtener un control exacto de la frecuencia de refresco, debiendo permitir un equilibrio entre el **retraso** de la adaptación y las **oscilaciones** de la frecuencia, cuya razón explicamos a continuación, y que es común a otros sistemas de control.

Para medir la frecuencia real de refresco debe contarse el tiempo transcurrido desde que se envían los datos gráficos correspondientes a un fotograma hasta que el fotograma es completamente dibujado. Si solamente utilizamos el tiempo transcurrido en el último fotograma, la frecuencia real sufrirá oscilaciones bruscas, ya que el observador va moviéndose, y hay objetos que aparecen, desaparecen o cambian de detalle. El sistema será incapaz de adaptarse a una frecuencia que varía tan rápidamente y sufrirá fácilmente oscilaciones, puesto que se alterará el nivel de detalle, y eso producirá un cambio inmediato en la frecuencia, que puede provocar a continuación el cambio opuesto.

Por otra parte, podemos calcular la frecuencia real haciendo la media de tiempos entre los últimos intervalos de procesamiento. De esa manera evitaremos oscilaciones bruscas, pero tardaremos un cierto tiempo en responder a cambios estables en la frecuencia. Por ejemplo, si el observador entra en una zona de la escena donde aparecen bruscamente muchos objetos, aumentará súbitamente el tiempo requerido para la visualización, pero hasta que no hayan transcurrido algunos fotogramas, ese cambio no hará variar significativamente la media, y por tanto no se adaptará la frecuencia de refresco. Por consiguiente, hay que buscar un valor de compromiso para el número de fotogramas utilizados en el calculo de la frecuencia.

6.5.2. MÉTODO PREDICTIVO

Se trata de un control en lazo abierto, es decir, vamos a intentar obtener la salida deseada sin utilizar su valor real. La manera de conseguir directamente una frecuencia de refresco determinada es alimentar el procesamiento gráfico con datos cuyo coste de computación es conocido y se corresponde con la frecuencia de salida deseada.

Para poder calcular el coste que representa cada nivel de detalle de los objetos y seleccionar los más apropiados, se define:

$$\forall objeto\ i \begin{cases} C_{ij} \rightarrow \text{coste de la visualizacion para cada nivel de detalle j de i} \\ b_{ij} \rightarrow \text{'beneficio' visual que aporta ese LOD j del objeto i} \end{cases}$$

El beneficio visual se corresponde con la 'importancia' del objeto en la visualización y puede evaluarse a partir de su distancia, área aparente y otros criterios. Es más difícil de obtener el coste de visualización, ya que no es tarea sencilla predecir lo que le va a costar a la máquina visualizar un cierto objeto. Para simplificar, este coste se suele calcular como una combinación del coste previsible por vértice y del coste previsible por pixel, proporcional al área aparente del objeto.

$$C_{ij} = \alpha \cdot n^{o}\ vertices + \beta \cdot area\ objeto$$

Donde α y β son constantes de proporcionalidad dependientes del equipo que efectúa el procesamiento y que habría que determinar.

Suponiendo que hemos podido determinar costes y beneficios, debemos entonces elegir qué niveles de detalle utilizar para cada objeto. Se trata de una variante del **problema de la mochila discreto**. El espacio o peso libre en la mochila sería el coste temporal máximo aceptable, pero además cada item que añadamos a la mochila tiene su correspondiente beneficio, y queremos que el total sea máximo. Supondiendo que para cada objeto i escogemos un nivel de detalle j = f(i) el beneficio total sería:

$$b = \sum_{i} b_{i,f(i)}$$

Como sabemos, se trata de un problema en principio no polinómico y que habrá que resolver mediante alguna estrategia heurística.

Este método predictivo resulta en general más preciso que el anterior y permite adaptarse inmediatamente a la situación de la escena sin ningún tipo de retraso, ya que se adelanta al procesamiento del fotograma. Sus desventajas son la dificultad de la implementación, la necesidad de hacer un estudio particular de los factores de coste de cada equipo gráfico y su dependencia de las características del hardware.

6.6. CONTROL DE LA VELOCIDAD APARENTE

Como ya comentamos, otra de las características que suele esperarse de una simulación en tiempo real es que el tiempo debería avanzar igual de rápido para el 'mundo' simulado que para el 'auténtico'. Ello no quiere decir que no podamos hacer que los objetos se muevan más rápido o más lento que en la realidad, sino que en muchas ocasiones queremos utilizar nuestro sentido del tiempo como patrón para juzgar la velocidad de lo que sucede en la simulación. Este requerimiento es esencial cuando intentamos reproducir una situación interactiva que se da en la realidad. Por ejemplo, si en una simulación de conducción queremos representar el movimiento de los vehículos a nuestro alrededor, su velocidad aparente debe ser la misma que el modelo computa interiormente, aunque la frecuencia de refresco de los gráficos pueda variar.

No podemos evitar las variaciones y oscilaciones en el intervalo de tiempo que cuesta redibujar la escena, ya que la distancia temporal entre dos fotogramas consecutivos no será nunca perfectamente constante. Por ello debemos buscar un sistema de compensar estas variaciones a la hora de actualizar las posiciones de los objetos móviles. No dará buen resultado suponer que el intervalo entre fotogramas es fijo (t) y mover a los objetos según una regla incremental del tipo x' = x + v*t.

Una forma de evitar estos problemas es efectuar la simulación en un proceso totalmente independiente y asíncrono con el proceso de visualización, o utilizar un temporizador en nuestro programa que llame a intervalos fijos a la función que actualice las posiciones. La frecuencia de este proceso paralelo deberá en todo caso ser siempre mayor que la frecuencia de refresco de los gráficos.

```
funcion simulacion
{
    t ← 0 ;
    hacer
    {
        t ← leer_ tiempo( ) ;
        para cada objeto i
                x_i ← x_i + v_i(t' − t) ;
        dibujar_ escena( ) ;
    }mientras (t < t_limite) ;
}
```

Pero en el caso de que no podamos, o no nos convenga, establecer este proceso paralelo, y actualicemos las posiciones dentro del mismo bucle en que se generan las imágenes, podemos obtener un resultado aceptable al incrementar las posiciones de los objetos móviles basándonos en la duración del último intervalo de dibujado. Si la frecuencia es lo suficientemente alta y no hay oscilaciones bruscas, este sistema resultará válido.