Hardware y Software

Dispositivos gráficos de salida: Monitores

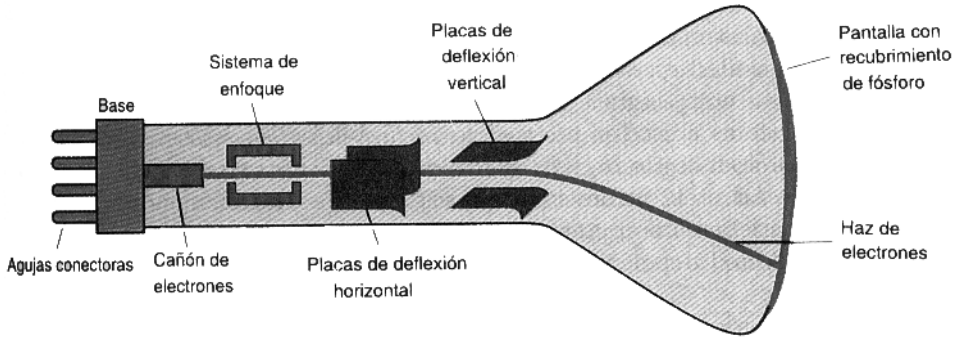
El tubo de rayos catódicos.

En el campo de la informática gráfica se pueden encontrar diferentes dispositivos de salida; desde los que permiten obtener representaciones en soporte físico (impresoras, p.e.) hasta sofisticados sistemas de "inmersión" capaces de generar todo un entorno de realidad virtual alrededor del usuario.

Por el momento nos concentraremos en los dispositivos de salida a vídeo, por ser los mas comunes y por constituir la base de otros muchos sistemas de visualización.

La tecnología en la que aún están basados la mayor parte de los monitores es el tubo de rayos catódicos (CRT); aunque parece estar siendo desplazada por la tecnología de cuarzo líquido (LCD).

Un CRT básicamente consiste en un tubo de vacío en cuyo interior un cátodo de metal calentado mediante un filamento por el que circula corriente eléctrica. El calor propicia el desprendimiento de electrones del cátodo; estos electrones, cargados negativamente, atraviesan diferentes dispositivos de enfoque y aceleración, y mediante los adecuados sistemas de control de deflexión son dirigidos a diferentes puntos del otro externo del tubo. En este extremo se encuentra la pantalla, recubierta internamente de fósforo, fósforo que al recibir el impacto de los electrones absorbe su energía, transformándola parcialmente en calor y utilizando el resto para elevar sus propios electrones a niveles superiores de excitación. Poco tiempo después estos electrones "excitados" vuelven a su estado original, desprendiendo el exceso de energía en forma de fotones; que es lo que produce el punto de la imagen visible.



Así pues, este punto de luz desaparece en el momento en que el fósforo vuelve a su estado de reposo; el tiempo en que esto tarda en ocurrir define una de las propiedades mas importantes para diferenciar entre dos fósforos: la persistencia, que se define como el tiempo que tarda en disminuir la luz emitida por la pantalla a un décimo de su intensidad inicial. Valores típicos son de 10 a 60 microsegundos.

Por lo tanto, si deseamos mantener una imagen visible en el CRT deberemos re-generarla a una velocidad superior a la que tarda en desaparecer de la pantalla. Los fósforos de menor

persistencia serán mas adecuados para animación; mientras que para mostrar imágenes estáticas puede ser suficiente un monitor más "lento".

Otra propiedad importante de los tubos es su resolución, generalmente definida simplemente como el máximo número de puntos luminosos que se pueden diferenciar. Dado que la luminosidad de un punto decae desde el centro hacia su periferia, se considera que dos puntos son diferentes cuando su separación es mayor del diámetro en el que el punto tiene aproximadamente el 60% de la luminosidad del centro del mismo. Es fácil encontrar monitores que soportan hasta 1280x1024; a partir de esta resolución se suele hablar de sistemas de alta definición.

También caracteriza a un tubo su "relación de aspecto", o relación entre la resolución horizontal y la vertical. La mas frecuente es de 4:3, aunque empiezan a aparecer relaciones "panorámicas" o 16:9, más apropiadas p.e. para cine en alta definición.

Monitores de barrido.

En el tipo de monitor con CRT más utilizado el haz de electrones recorre la punto por punto en dirección longitudinal y línea por línea en dirección vertical, variando su intensidad en cada punto de acuerdo con la luminosidad correspondiente al punto de la imagen correspondiente que se pretende representar. La intensidad de cada uno de estos puntos de imagen, conocidos como "pixels", debe estar almacenada en la memoria del ordenador (a la memoria encargada de contener esta información se le denomina búfer de pantalla o frame buffer); si p.e. para representar la intensidad de un pixel se utilizan 24 bits, para una pantalla de 1024 x 1024 se necesitan 3 Mb de memoria.

La velocidad a que se realiza este barrido (evidentemente, relacionada con la persistencia del fósforo) suele darse como característica del monitor; tanto en la forma de frecuencia de barrido vertical (para toda la imagen), que suele oscilar entre los 50 y los 80 Hz y la frecuencia de barrido horizontal (para cada línea) que suele ser del orden de los 40 KHz. Por lo general, una frecuencia mas alta, por lo general, es conveniente para disminuir la fatiga visual, y necesaria cuando se desea generar imágenes a gran velocidad (p.e. pares de imágenes estereoscópicas en tiempo real a velocidades de refresco tales que el ojo no distinga "parpadeos").

En algunos monitores realiza el barrido vertical en dos fases, en la primera se recorren las líneas impares, y en la segunda las pares. De esta forma se consigue una frecuencia aparente doble de la real (aparente en el sentido de que cada imagen se genera en la mitad de tiempo, aunque, evidentemente la imagen contenga la mitad de la información de la real). Este sistema se utiliza en los televisores domésticos, lo que puede aprovecharse también para conseguir una mayor suavidad en las animaciones generadas por ordenador: Efectivamente, en lugar de generar una imagen cada p.e. 1/60 segundo, se genera la mitad de la imagen cada 1/120 segundo, con lo que el tiempo de cálculo es el mismo y el movimiento más continuo.

Monitores de trazado aleatorio o vectoriales

En este sistema, actualmente menos utilizado, el haz de electrones en lugar de recorrer la pantalla a base de pixels, traza una serie de segmentos correspondientes a las líneas de la imagen que se pretende representar.

Así, la imagen está definida en memoria como una sucesión de instrucciones de trazado y movimiento aleatorio; y esta serie de instrucciones son ejecutadas por el haz de electrones un

mínimo de 30 veces por segundo. Los sistemas vectoriales de alta calidad pueden llegar a manejar imágenes con 100.000 líneas; cuando la imagen tiene muy pocas líneas se introduce un retraso entre redibujados, ya que de otra forma se podría llegar a quemar el fósforo.

Monitores CRT en color

Los monitores CRT en color utilizan una combinación de fósforos que emiten luz con colores distintos, colores básicos que combinados producen la sensación de los colores del espectro que se deseen.

Estos monitores utiliza principalmente la técnica de la "máscara de sombra". Partiendo de un tubo con tres fósforos que emiten, para cada punto, luz roja, verde y azul, y con tres cañones de electrones, los tres haces de electrones se enfocan y se hacen coincidir sobre la máscara, que contiene una serie de orificios alineados con los patrones de punto de fósforo. Cuando los tres haces pasan por un orificio activan el triángulo correspondiente, que se ve como un punto de color.

La intensidad de cada haz se controla independientemente, y por lo general se considera que con una resolución de 8 bits para cada uno (24 en total, por lo tanto), es decir, 256 niveles de voltaje por cañón para un total de 256 al cubo (16 Millones) de colores, se obtiene "color real" (haciendo referencia a la incapacidad del ojo humano de distinguir diferencias al aumentar la cantidad de niveles por componente).

Monitores de cuarzo líquido (LCD)

Los monitores de cuarzo líquido utilizan dos placas de cristal que contienen cada una un polarizador de luz girado 90 grados una respecto a la otra. Estas placas prensan el cuarzo liquido (con estructura cristalina, pero con moléculas capaces de fluir como en estado líquido). En una de las placas se crean líneas verticales de conductores transparentes, y en la otra, horizontales. La intersección de las líneas define los pixels, de forma que cuando existe voltaje ambos conductores en ambos conductores las moléculas se alinean y la luz que atraviesa el cristal no gira y por lo tanto no atraviesa el segundo cristal.

Cuando la luz se genera mediante un transistor en cada pixel, se habla de pantallas LCD de matriz activa, siendo esta la tecnología más utilizada en la actualidad.

Aparte de eso, la representación de la información es similar a la de los monitores CRT de barrido: también se utiliza el bufer de pantalla con la imagen formada por pixels.

Software de gráficos.

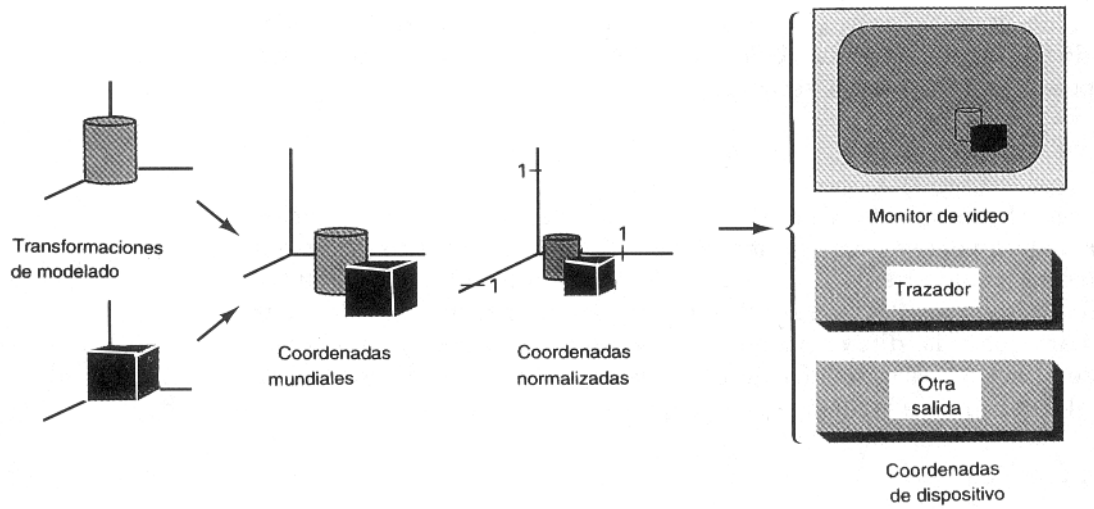
Introducción

Según se ha visto, por lo general vamos a necesitar una representación gráfica consistente en un conjunto de pixels a partir de una idea de más alto nivel; como pueda ser la descripción de la gráfica en términos de líneas, arcos, colores etc. o incluso en términos de objetos tridimensionales, puntos de vista e iluminación.

El como llegar de estas descripciones de alto nivel al conjunto de pixels final es algo de lo que las diferentes partes del sistema se deberán encargar; por lo general el programador dispone

de una serie de librerías de programación gráfica que le permiten escribir aplicaciones sin tener que llegar a conocer en detalle el hardware sobre el que se ejecutará su código, y sin tener que reescribir de cero miles de procedimientos que, además, distan de ser triviales. Ejemplos de estas librerías podrían ser OpenGL de SGI y Direct3D de Microsoft.

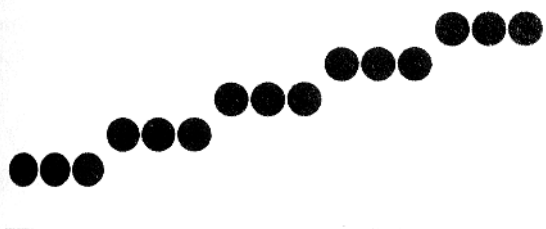
Por lo general, estas librerías permiten trabajar creando estructuras en un sistema de coordenadas local, integrar estas estructuras en una escena más compleja que utiliza un sistema de coordenadas global o "de mundo". De algún modo, el software transformará estas coordenadas a unas coordenadas de dispositivo normalizado (independiente de las características físicas del dispositivo real) y en un último paso estas se ajustarán al rango de salida del dispositivo final.



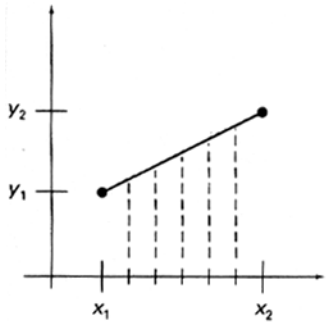
Los bloques de construcción básicos que ofrece una librería se conocen como "primitivas" y pueden incluir desde un mínimo de líneas, círculos, caracteres, etc. en dos dimensiones hasta mallas de polígonos tridimensionales, definiciones de luces, etc.

Trazado de líneas

En general, para obtener la representación como mapa de pixels a partir de una línea en dos dimensiones es necesario seleccionar las posiciones de la pantalla que pertenecen al trazado de la línea. Dado que estas posiciones se corresponden a pixels localizados en posiciones fijas enteras lo máximo que se podrá conseguir es una aproximación a los valores redondeados de las coordenadas de la línea. Esta aproximación por lo general producirá un efecto de "escalonamiento"; efecto que puede amortiguarse ajustando la intensidad de los pixels a lo largo de la línea, según ser verá.



Para el caso de una línea recta, por ejemplo, definida mediante



$y = m\,x + b$

donde

$\Delta y = (y_2 - y_1)$

$\Delta x = (x_2 - x_1)$

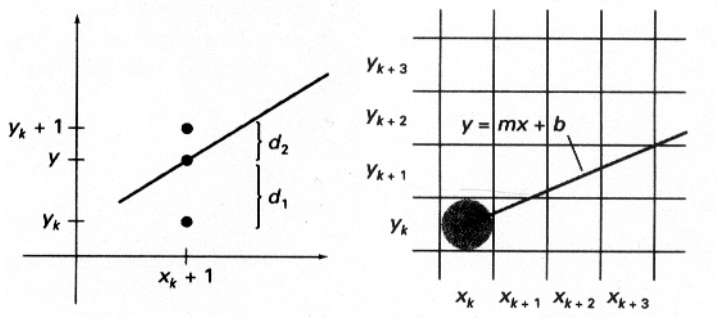
$m = \Delta y / \Delta x$

$b = y_1 - m\,x_1$

el muestrear el segmento en los valores de x correspondientes a los pixels (aumentos unitarios de x, p.e., para valores de pendiente menor o igual que uno) basta con incrementar y en la pendiente a cada paso.

Un problema matemático tan simple se complica toda vez que deseamos evitar las operaciones en coma flotante, computacionalmente mucho mas lentas que las operaciones con enteros. Tanto para segmentos rectilineos como para otras primitivas Bresenham desarrolló un algoritmo que permite determinar las coordenadas de los pixels utilizando sólo cálculos incrementales con enteros.

Si partimos de un punto intermedio (x_k , y_k), la coordenada y siguiente correspondiente a x_{k+1} podrá corresponder o bien a y_k o a y_{k+1}



Las distancias de ambas opciones a la línea real d1 y d2 vendrán dadas por

$d_1 = y - y_k = m\,(x_k + 1) + b - y_k$

$d_2 = y_k + 1 - y = y_k + 1 - m\,(x_k + 1) - b$

y

$d_1 - d_2 = 2\ m\ (x_k + 1) - 2\ y_k + 2\ b - 1$

sustituyendo m por su valor definiendo pk como:

$p_k = \Delta_x\ (\ d_1 - d_2\ ) = 2\ \Delta_y\ x_k - 2\ \Delta_x\ y_k + \underline{2\ \Delta_y + \Delta_x\ (\ 2b-1\ )}$

$p_k = \Delta_x\ (\ d_1 - d_2\ ) = 2\ \Delta_y\ x_k - 2\ \Delta_x\ y_k + c$

donde hemos sustituido $\underline{2\ \Delta_y + \Delta_x\ (\ 2b-1\ )}$ por c como constante.

Aquí ya se ve que el signo de pk indica si la distancia d_1 es mayor o menor que d_2, y por tanto, si se habrá de elegir y_{k+1} o y_k.

Desarrollando, se llega a la expresión de pk+1 como

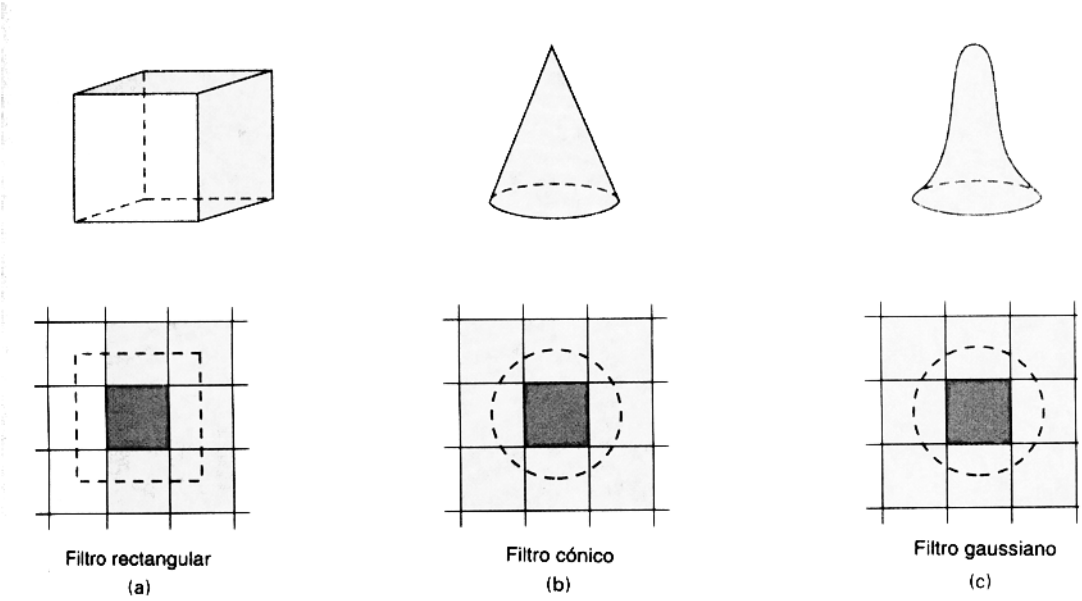
$p_{k+1} = p_k + 2\ \Delta_y - 2\ \Delta_x\ (y_{k+1} - y_k)$

donde $(y_{k+1} - y_k)$ solo puede valer 0 o 1 dependiendo del signo de p_k; con lo que tenemos que el incremento simplemente toma un valor constante u otro (y por lo tanto, se elige un valor de y u otro) en función de la decisión anterior, partiendo de

$p_0 = 2\ \Delta_y - \Delta_x$

Antialiasing

Para reducir el efecto de escalonamiento se suele recurrir, como ya se ha comentado, a variaciones en la intensidad de los pixels a lo largo de la línea. La forma de determinar esa intensidad, por lo general, se basa en un sobre-muestreo; es decir, se supone cada pixel formado por una malla de subpixels, y la línea, se supone de grosor finito de un pixel (por oposición a la línea ideal sin anchura). De esta forma se puede calcular el grado de "cobertura" de cada pixel por parte de la línea. Este cálculo se puede hacer por simple enumeración de subpixels por los que pasa la línea, o bien dando más peso a los subpixels mas próximos al centro del pixel. La forma de realizar esto último suele consistir en la aplicación de una función de filtro simétrica tipo rectángulo, cono o campana de Gauss.



Otras primitivas.

Aparte del trazado de líneas, otras primitivas importantes como conceptos pueden ser:

- Caracteres
- Transformaciones (giros, escalas, distorsiones)
- Atributos de las primitivas (tipo de línea, color)
- Llenado de áreas, es decir, el considerar un área cerrada de un color sólido. Conviene notar que algo que a primera vista puede parecer sencillo, como el llenar un polígono de determinado color, en la practica resulta también complicado, recurriéndose sobre todo a dos tipos de algoritmos:

  1: Recorrido de todas las líneas de pixels de la pantalla buscando intersecciones con los límites del polígono.

  Esto suele conllevar abundantes cálculos.

  2: Procedimientos recursivos a partir de un punto. Se comienza en un punto y se repite a partir de todos sus vecinos a no ser que se encuentre una condición de frontera.
- Algoritmos de recorte de líneas: Por lo general, una vez transformadas las coordenadas de las líneas será necesario determinar que partes de las líneas son visibles dada una ventana de visualización.

Primitivas 3D

Veamos algunas primitivas típicas, en este caso de OpenGL, con objeto de tener una idea mas clara del tipo de programación necesaria para trabajar con librerías gráficas.

Atributos de primitivas:

```
Tamaño de puntos - void glPointSize(GLfloat size)
Anchura de líneas - void glLineWidth(GLfloat width)
Modo de Polígonos -
glPolygonMode(GL_FRONT, GL_FILL)
glPolygonMode(GL_BACK, GL_LINE)
define que los polígonos seran "llenos" vistos por delante y
formados por líneas ("huecos") por detrás.
```

Vértices:

```
Posición - void glVertex3dv( const GLdouble *v )
Normales - void glNormal3fv( const GLfloat *v )
```

Polígonos:

```
Por lo general se definen mediante secuencias de vértices, aunque
lo mas utilizado con diferencia son simplemente los triángulos. De
hecho por lo general cualquier primitiva 3D compleja es
descompuesta en triángulos de forma transparente al programador.

Ejemplo de polígono:
```

```
glBegin (GL_POLYGON);
   glNormal3fv(n0);
   glVertex3fv(v0);
   glNormal3fv(n1);
   glVertex3fv(v1);
   glNormal3fv(n2);
   glVertex3fv(v2);
   glNormal3fv(n3);
   glVertex3fv(v3);
glEnd();
```

Aunque como hemos dicho la descomposición puede ser realizada por el sistema, por lo general es mas recomendable, desde el punto de vista de la eficiencia, que sea el programador el que efectúe manualmente dicha descomposición; ya que por lo general llegara a mejores resultados que una descomposición automática.

```
Transformaciones geométricas:
```

Como es sabido, todas las operaciones de posicionado de primitivas en una escena, modificación de las mismas y proyección sobre el plano se lleva a cabo mediante operaciones con matrices; por lo que todas las librerías gráficas ofrecen facilidades para trabajar con este tipo de datos. Conviene notar que todos los productos de matrices se llevan a cabo internamente en la librería (o en el sistema gráfico, en el caso más general), y por lo tanto los resultados numéricos no tienen por que estar disponibles para el programador.

Ejemplos de estas transformaciones son:

```
Matriz de rotación –
void glRotatef( GLfloat angle, GLfloat x, GLfloat y, GLfloat z )
```

```
Matriz de traslación –
void void glTranslatef( GLfloat x, GLfloat y, GLfloat z )

Matriz de escalado –
void glScalef( GLfloat x, GLfloat y, GLfloat z )

Matriz de proyección (perspectiva) –
void glFrustum( GLdouble left,
                    GLdouble right,
                    GLdouble bottom,
                    GLdouble top,
                    GLdouble near,
                    GLdouble far )
```

Otro tipo de primitivas necesarias son las fuentes de luz:

```
void glLightfv(GLenum light, GLenum pname, TYPEparam);
```

donde se dispone de las típicas fuentes ambientales (GL_AMBIENT), difusas (GL_DIFFUSE) y especulares (GL_SPECULAR), así como luces tipo "spot" definidas por su cono luminoso y las funciones de atenuación

con la distancia.

Controladores Gráficos y Aceleradoras 3D

Conceptos

Si se piensa en la cantidad de bits necesarios para describir la imagen de pixels, y que se debe acceder a estos del orden de 60 veces por segundo, se comprende que el uso de memoria convencional no es suficiente en cuanto a velocidad. Por esta razón se utilizan dispositivos especiales con memoria dedicada al sistema de vídeo, y, por lo general, con coprocesadores específicos dedicados a manejar estos datos; descargando en lo posible a la unidad central de todo el proceso gráfico.

Por ejemplo, para el caso de OpenGL, la librería puede estar implementada de forma que el propio código de la librería (ejecutado en la CPU) realice todos los cálculos necesarios, o simplemente puede consistir en una capa de enlace con los coprocesadores de OpenGL capaces de interpretar directamente estas instrucciones y realizar las operaciones en los procesadores diseñados a medida a tal fin.

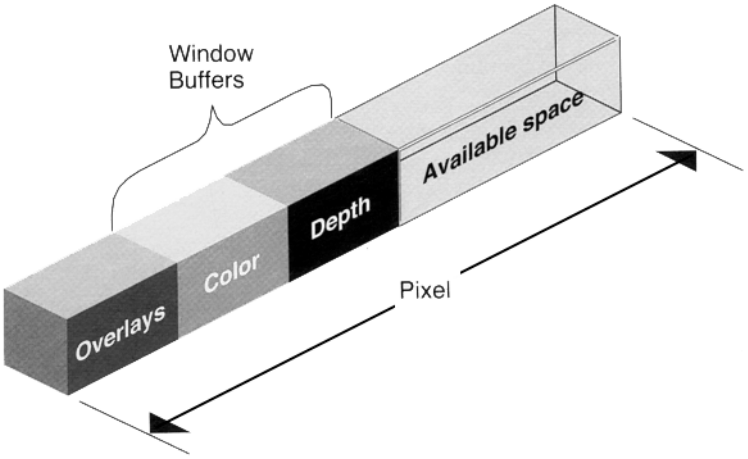
Curiosamente, con el incremento de la capacidad de cálculo de CPU de los sistemas de hoy en día; en algunos casos, sobre todo dependiendo del tamaño de los datos a manejar (es decir, para volúmenes pequeños de datos) algunos sistemas sin coprocesadores gráficos específicos puede resultar más rápida que otra que los tenga. Esto es debido a que por lo general la CPU tiene que alimentar de comandos OpenGL (simplificando, la geometría de la escena) al subsistema gráfico a través de una "tubería", si la CPU tiene más capacidad de enviar información que el acelerador de procesarla y así vaciar la tubería, la CPU pasa a modo de espera, lo que causa que el proceso que esta en la CPU tenga que estar continuamente parando y re-arrancando.

Hardware y Software:

Veamos algunos conceptos que necesarios para comprender como funcionan estos "coprocesadores" o aceleradores gráficos:

Bufer de pantalla

Como se ha mencionado anteriormente la información de cada imagen se almacena en memoria en forma de matriz de pixels RGB. Esto significa que cada pixel tiene asignada en esa matriz una cantidad de bits que en buena medida definirán las capacidades del hardware gráfico.



Un sistema básico típico de hoy en día asignará 8 bits para cada componente de color, dando así la impresión de "color real" como ya se ha dicho anteriormente.

Ahora bien, en sistemas con un mayor numero de bits por pixels, el sistema dispone de espacio adicional, espacio que es posible utilizar para por ejemplo almacenar una segunda imagen (casos de visión estereoscópica o animación por doble bufer), mapas de profundidad o "Z bufer" o capas auxiliares (para detalles auxiliares, como p.e. cursores)

Z Bufer

Es fácil encontrarse, sobre todo en revistas sobre juegos, descripciones del Z Bufer como algo que sigue la pista a cada polígono de la escena y es capaz de realizar comparaciones sobre que polígonos están más próximos al observador, permitiendo que no se dibujen los que están ocultos por otros.

Esta descripción es incorrecta, ya que en realidad todo lo que hace el Z Bufer es comparaciones de pixels. Para cada pixel de la pantalla podemos imaginar que trazamos un rayo hacia la escena. Este rayo puede llegar al fondo o bien chocar con algún polígono; en este caso el color del polígono será el que determine lo que se ve en la pantalla.

El sistema gráfico convierte cada polígono a medida que es definido a puntos; el punto se compara con el valor correspondiente almacenado en el Z bufer, si resulta estar más próximo su color pasa a ser el elegido, y el valor del Z Bufer se reemplaza con su profundidad.

Conviene tener en cuenta que las comparaciones serán más precisas cuantos mas bits se puedan utilizar para codificar los valores de profundidad, por ejemplo, para escenas definidas con

precisión de milímetros en las que existen objetos distantes p.e. un kilometro, no es suficiente con un Z Bufer de 16 bits, debiéndose recurrir al menos a 32 bits.

Otro uso frecuente del Z Bufer es el cálculo de mapas de profundidad a partir de una fuente luminosa local con objeto de determinar que objetos quedan oscurecidos por la sombra arrojada por objetos más próximos a la fuente luminosa.

Mapeado de texturas

Una forma muy común de mejora la apariencia de una imagen sintética consiste en la aplicación de una imagen real (fotográfica) sobre la superficie que se pretende representar. Podemos pensar por ejemplo en la definición de una fachada de un edificio. El modelo real debería incluir millones de polígonos para reflejar no solo la complejidad del la geometría básica (portales, ventanas, etc.) si no todo tipo de imperfecciones, abolladuras, etc.; por no mencionar la descripción precisa de cada uno de los materiales. El reemplazar esto por un simple plano con la foto de un edificio real pegada simplifica enormemente la escena, y los resultados son por lo general muy buenos.

La forma de realizar esta aplicación consiste en hacer corresponder a cada punto del polígono un punto de la textura.

Hay varios puntos a tener en cuenta: Este mapeado deberá realizarse de forma que se mantenga la continuidad entre polígonos adyacentes. De otra forma se pierde la sensación de realismo.

Al aplicar la textura, no basta una simple aplicación lineal, ya que debe tenerse en cuenta la corrección debida a la perspectiva.

El tamaño de la textura será un compromiso entre la calidad de imagen deseada y la memoria disponible para almacenarla. Dado que el acceso a los datos de la(s) texturas es continuo, estas deberán estar almacenadas por lo general, en memoria rápida especialmente destinada a tal fin.

Sombreado (Plano, Gouraud, Phong)

Como es bien sabido, en aplicaciones de tiempo real la iluminación se calcula en los vértices de cada polígono, debiendo determinarse la iluminación correspondiente a los puntos del interior del mismo mediante técnicas de interpolación más o menos complejas.

El caso de la no interpolación (sombreado plano) se corresponde a las hipótesis de que la luz está a distancia infinita, el observador también, y que los polígonos pretenden ser realmente polígonos planos, no una aproximación a superficies mas complejas. Esto por lo general da unos resultados bastante pobres.

Gouraud mejora la cuestión utilizando las normales, que deben estar definidas en cada vértice, para determinar la intensidad de la iluminación en cada vértice, y esta se interpola a lo largo de las aristas, y luego a lo largo de cada línea representada del polígono. Ofrece resultados visualmente convincentes, pero no estrictamente correctos.

Phong interpola las normales, con lo que los resultados son mucho mejores, pero mucho mas costosos en cuanto a cálculo también. Por otra parte, el uso de texturas disminuye la necesidad de tanta precisión, por lo que no es frecuente que el acelerador lo incluya.

Antialiasing (en 3D)

La idea es la misma que la vista para líneas en 2D: evitar que las aristas proyectadas presenten "escalones". El efecto de aliasing es incluso peor en animaciones, dado que mínimos cambios de posición pueden determinar efectos de aliasing totalmente distintos, lo que produce la sensación de que "algo se mueve" en los bordes de los objetos.

Un método primitivo consiste en procesar la imagen final ya proyectada y examinar los pixels, suavizando las diferencias entre pixels adyacentes. Es simple y rápido.

En un método mucho más costoso se tomarán varias muestras (el concepto de "sub-pixels" otra vez) para calcular el color final del pixel.

Y por último otro método que puede resultar más costoso consiste en hacer "vibrar" la imagen (mediante cambios mínimos de posición del punto de vista) y realizar una media entre las imágenes calculadas. Es probablemente el que da mejores resultados, pero también el más costoso.

Interpolación Mipmap de texturas

Un problema de las texturas es que, cuando se utiliza una resolución de textura demasiado baja, que resultaría adecuada p.e. para determinada distancia al observador, al ser observada más de cerca, ofrece un aspecto totalmente irreconocible, ya que cada pixel de textura se mapea a mas de un pixel de pantalla.

Para solucionar esto, una posible técnica consiste en almacenar la textura a diferentes resoluciones (por lo general, utilizando múltiplos de dos) y en función de la distancia elegir dos de estas resoluciones e interpolar entre ellas. El tipo de interpolación (bilineal, tri-lineal, quad-lineal) define la calidad de la imagen obtenida.

Efectos de profundidad

Por lo general en la vida real, los efectos atmosféricos causan que los objetos lejanos se vean de forma diferente. Se puede pensar por ejemplo en una situación con niebla en la que los objetos se van viendo cada vez mas ocultos hasta perderse en un fondo blanco.

Por lo general este tipo de efectos se puede conseguir utilizando la información almacenada en el Z Bufer para añadir color en función de la distancia; con lo que se consiguen efectos de gran realismo.

Transparencia

Anteriormente se ha hablado de 3 componentes de color (RGB). Se puede considerar una cuarta (Alpha) que determina la transparencia del punto.

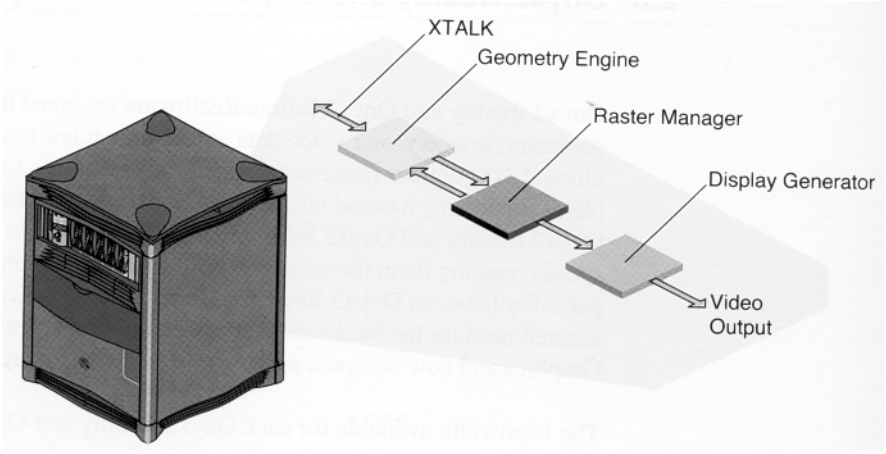
De esta forma cada vértice representado puede llevar un valor de transparencia asociado, valor que se utiliza de forma que objetos con diferente nivel de transparencia puedan representarse uno delante del otro, permitiendo que ambos sean parcialmente visibles.

Por lo general es conveniente que el programador defina los polígonos transparentes de forma que estén ya ordenados en cuanto a profundidad respecto al observador. (Nótese por la descripción del Z Bufer que no resulta posible en general utilizarlo para evitar esta ordenación previa)

Ejemplos de hardware

Veremos a continuación dos ejemplos de hardware gráfico:

Onyx2 Infinite Reality



Como vemos, el sistema se compone de tres partes diferenciadas:

La "Geometry Engine" (que se correspondería con los coprocesadores gráficos a los que antes nos referíamos) se encarga de realizar transformaciones de vértices, iluminación, recorte, proyección, etc., así como de realizar operaciones sobre pixels como histogramas, convoluciones, escalado, etc., operaciones que se aplican tanto a los pixels normales, como a las texturas o a fuentes de vídeo.

El "Raster Manager" recibe todos los datos de triángulos, puntos y líneas ya proyectados por la GE; estos datos deben ser recorridos línea por línea para generar la información de pixels. Por su elevado grado de paralelismo, el RM puede realizar todo tipo de operaciones de antialiasing (mediante sobre muestreo), mapeado de texturas, etc. sin encontrar cuellos de botella.

El mapeado de texturas se realiza por medio de la ya comentada técnica de MIPmapping; así mismo esta unidad realiza los cálculos necesarios sobre la transparencia y los efectos de profundidad.

El "Display Generator" básicamente contiene los conversores Digital-Analógico, por lo general generando mas de un canal de salida de vídeo; permitiendo el re-muestreo de la señal de salida para adaptarlo a todo tipo de dispositivos de vídeo.

La Onyx2 IR standard permite manejar 2.62 millones de pixels, para obtener p.e. una resolución de 1920 x 1200 o dos salidas a 1280 x 1024.

La cantidad mínima de bits por pixel en el bufer de pantalla es de 256 bits, con lo que se puede llegar a utilizar salidas estereoscópicas en cuádruple bufer (dos por canal) con 12 bits para color y 23 de Z Bufer.

Otros efectos que permite conseguir pueden ser:

- Mapas de entorno
- Texturas volumétricas
- Antialiasing
- Luces locales (hasta 8)

- Sombras utilizando mapas de profundidad
- Texturas proyectadas (diapositivas) - Vídeo como textura

nvidia GeForce2 GTS

Esta tarjeta gráfica, adecuada para ordenadores personales, orientada especialmente al mercado de juegos, plantea una aproximación diferente, al englobar toda su funcionalidad en un solo chip.

Su nombre significa GigaTexel Shader (GTS), haciendo referencia a la capacidad de procesado de texturas, ya que en cada ciclo de reloj las dos unidades de textura calculan cuatro pixels, lo que nos da 1.6 Gtexels por segundo

Sus principales características son:

- Geometría (transformación e iluminación): 25 Millones pol./segundo

- Pixel shading:

Como se ha mencionado, la interpolación de normales resulta muy costosa. Aquí la solución adoptada consiste en la utilización de una textura suministrada por el programador que contiene los valores de la normal para cada punto. Esto produce imágenes muy realistas; y es posible utilizar esto también para aplicar las conocidas técnicas de "bump-mapping", en las que por perturbaciones de la normal se obtiene una apariencia de rugosidad en la superficie.

- Antialiasing por Hardware. Se observa caída en las prestaciones a resoluciones superiores a 1024 x 768 debido a la influencia de la limitación de velocidad de acceso a memoria.

- Transferencia de texturas y geometría por AGP - Compresión de texturas.

Con objeto de aprovechar mejor la memoria destinada a texturas a costa de potencia de cálculo se utiliza un sistema de compresión de las texturas en memoria capaz de reducirlas hasta aproximadamente 1/8 de tamaño. Con los sistemas actuales, el ahorro de memoria y de ancho de banda justifica esta medida.

- Bufer de pantalla: Se usan 32 bits para el color y otros 32 para el Z Bufer con una configuración standard, aunque es posible aumentar la cantidad de memoria del bufer de los 32 Mb hasta los 128 Mb

- Optimizado para OpenGL y DirectX

Indice