(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 804 149**

(51) Int. CI.:

G06T 15/00 (2011.01)
G06F 9/50 (2006.01)
G06T 1/20 (2006.01)
G06T 1/60 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **19.01.2016 PCT/US2016/013932**

(87) Fecha y número de publicación internacional: **18.08.2016 WO16130286**

(96) Fecha de presentación y número de la solicitud europea: **19.01.2016 E 16705351 (1)**

(97) Fecha y número de publicación de la concesión europea: **01.04.2020 EP 3257024**

(54) Título: **Renderización híbrida en procesamiento de gráficos**

(30) Prioridad:

10.02.2015 US 201514618463

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
04.02.2021

(73) Titular/es:

QUALCOMM INCORPORATED (100.0%)
International IP Administration, 5775 Morehouse Drive
San Diego, CA 92121-1714, US

(72) Inventor/es:

WANG, TAO

(74) Agente/Representante:

FORTEA LAGUNA, Juan José

DESCRIPCIÓN

Renderización híbrida en procesamiento de gráficos

CAMPO TÉCNICO

[0001] Esta divulgación se refiere a técnicas para el procesamiento de gráficos, y, más específicamente, a técnicas para el modo de renderización híbrida en el procesamiento de gráficos.

ANTECEDENTES

[0002] El contenido visual para visualización, tal como contenido para interfaces gráficas de usuario y videojuegos, se puede generar mediante una unidad de procesamiento de gráficos (Graphics Processing Unit, GPU). Una GPU puede convertir objetos bidimensionales o tridimensionales (3D) en una representación de píxeles bidimensional (2D) que se puede visualizar. La conversión de información sobre objetos 3D en un mapa de bits que se puede visualizar se conoce como renderización de píxeles, y requiere una memoria y potencia de procesamiento considerables. En el pasado, la capacidad de gráficos 3D solo estaba disponible en estaciones de trabajo potentes. Sin embargo, en la actualidad, los aceleradores de gráficos 3D se encuentran habitualmente en ordenadores personales (PC), así como en dispositivos embutidos, tales como teléfonos inteligentes, tabletas, reproductores multimedia portátiles, consolas de videojuegos portátiles y similares. Típicamente, los dispositivos embutidos tienen menos potencia de cálculo y capacidad de memoria en comparación con los PC convencionales. Así pues, la complejidad incrementada en las técnicas de renderización de gráficos 3D presenta dificultades al implementar dichas técnicas en un sistema embutido.

[0003] La publicación de la solicitud de patente US 2013/0135322 A1 divulga un sistema de renderización en el que un modo de renderización se puede cambiar entre un modo de renderización directa y un modo de renderización de fragmentos dependiendo de la cantidad de superposición en un fragmento. En el modo de renderización de fragmentos, los datos de color y los datos de profundidad se escriben en la memoria de gráficos rápida, mientras que en el modo de renderización directa, se usa una memoria del sistema más lenta.

BREVE EXPLICACIÓN

[0004] En general, esta divulgación describe técnicas para un modo de renderización híbrida en el procesamiento de gráficos. En particular, en algunos ejemplos, esta divulgación describe un modo de renderización híbrida que usa un pase de fragmento (binning pass) para determinar qué primitivas de una trama pueden afectar o de otro modo contribuir a un fragmento de una escena. La renderización se realiza entonces en el fragmento usando solo los triángulos que se determina que afectan al fragmento mediante el pase de fragmento. Durante la renderización del fragmento, los valores en un búfer de profundidad se almacenan en la memoria de gráficos rápida en chip (GMEM). Una vez que se ha renderizado el fragmento, los valores en el búfer de profundidad se resuelven en la memoria del sistema. Sin embargo, en lugar de almacenar también valores en un búfer de color (es decir, el búfer que almacena los valores de color de los píxeles) en la GMEM, y resolver a continuación el búfer de color en la memoria del sistema cuando se completa la renderización de un fragmento, las técnicas de renderización híbrida de esta divulgación pueden escribir valores de píxeles directamente en un búfer de color almacenada en la memoria del sistema sin usar la GMEM. Escribir valores del búfer de color directamente en la memoria del sistema puede ser beneficioso en situaciones en las que el búfer de color se actualiza con poca frecuencia, o solo una vez, durante la renderización de un fragmento.

[0005] En un ejemplo de la divulgación, se proporciona un procedimiento de procesamiento de gráficos como se define en la reivindicación 1.

[0006] En otro ejemplo de la divulgación, se proporciona un aparato configurado para el procesamiento de gráficos como se define en la reivindicación 4.

[0007] En otro ejemplo, esta divulgación describe un medio de almacenamiento no transitorio legible por ordenador como se define en la reivindicación 8.

[0008] Las técnicas de esta divulgación también se describen en términos de un aparato y un medio de almacenamiento legible por ordenador que almacena instrucciones para provocar que un procesador realice las técnicas. Los detalles de uno o más ejemplos se exponen en los dibujos adjuntos y en la siguiente descripción. Otras características, objetivos y ventajas resultarán evidentes a partir de la descripción y de los dibujos, y a partir de las reivindicaciones.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

[0009]

La FIG. 1 es un diagrama de bloques que muestra un dispositivo informático de ejemplo configurado para usar

las técnicas de esta divulgación.

La FIG. 2 es un diagrama de bloques que muestra unidades de procesamiento de ejemplo configuradas para usar las técnicas de esta divulgación.

La FIG. 3 es un diagrama conceptual que ilustra fragmentos de una trama.

La FIG. 4 es un diagrama conceptual que ilustra fragmentos de una trama con más detalle.

La FIG. 5 es un diagrama conceptual que ilustra memorias intermedias de comandos para un modo de renderización híbrida que usa fragmentación de "software" de acuerdo con las técnicas de la divulgación.

La FIG. 6 es un diagrama conceptual que ilustra memorias intermedias de comandos para un modo de renderización híbrida que usa la fragmentación de "hardware" de acuerdo con las técnicas de la divulgación.

La FIG. 7 es un diagrama de flujo que ilustra un procedimiento de acuerdo con un ejemplo de la divulgación.

DESCRIPCIÓN DETALLADA

[0010] Esta divulgación se refiere a técnicas para el procesamiento de gráficos, y, más específicamente, a técnicas para un modo de renderización híbrida. Los sistemas de renderización de gráficos actuales típicamente usan uno de un modo de renderización de fragmentos (a veces denominada renderización basada en mosaicos) o un modo de renderización directa para renderizar una escena. En muchos ejemplos, una unidad de procesamiento de gráficos (GPU) se puede configurar para realizar selectivamente tanto la renderización de fragmentos como la renderización basada en mosaicos. En la renderización de fragmentos, se renderiza una trama de una escena 2D o 3D dividiendo la trama en partes más pequeñas (por ejemplo, fragmentos rectangulares o mosaicos) y renderizando cada uno de estos fragmentos por separado. La renderización de fragmentos es útil para aplicaciones en las que se dispone de poca memoria de gráficos rápida dedicada (GMEM), tal como para aplicaciones móviles. El tamaño de los fragmentos se puede configurar para representar la cantidad de almacenamiento que está disponible en la GMEM. Por ejemplo, si la GMEM puede almacenar 512 kB, el tamaño de un fragmento se puede configurar de modo que los datos de píxeles contenidos en ese fragmento sean menores o iguales a 512 kB.

[0011] Los datos de píxeles que se pueden almacenar en la GMEM pueden incluir valores de profundidad de los píxeles y valores de color de los píxeles. Los valores de profundidad para los píxeles en el fragmento se pueden almacenar en un búfer de profundidad en la GMEM. Del mismo modo, los valores de color para los píxeles en el fragmento se pueden almacenar en un búfer de color en la GMEM. Cuando finaliza la renderización de un fragmento, tanto el búfer de profundidad como el búfer de color se pueden resolver en la memoria del sistema (por ejemplo, en la memoria de acceso aleatorio dinámica (DRAM)). Es decir, resolver los datos de un búfer a la memoria del sistema incluye escribir los valores almacenados en el búfer (por ejemplo, tanto el búfer de profundidad como el búfer de color) de la GMEM en la memoria del sistema. De esta manera, los valores de profundidad y los valores de color para el siguiente fragmento se pueden almacenar en el búfer de profundidad y el búfer de color en la GMEM.

[0012] El procesamiento de gráficos en un modo de renderización directa, por otra parte, no divide una trama en partes más pequeñas (por ejemplo, fragmentos o mosaicos). En cambio, la totalidad de una trama se renderiza de una vez. En algunos sistemas de procesamiento de gráficos (por ejemplo, un sistema de procesamiento de gráficos en un dispositivo móvil), no hay suficiente GMEM para contener una trama completo de datos de píxeles. En cambio, para un modo de renderización directa, se usa una memoria del sistema más lenta para renderizar la trama. Así pues, los valores de profundidad y los valores de color se escriben en un búfer de profundidad y un búfer de color, respectivamente, directamente en una memoria del sistema.

[0013] Dado que las técnicas de renderización directa usan la memoria del sistema para almacenar el búfer de color y el búfer de profundidad, las técnicas de renderización directa típicamente experimentan más tráfico de memoria desde la memoria del sistema. El tráfico de memoria entre la GPU y la memoria del sistema típicamente es lento y, por tanto, limita el rendimiento de la GPU. Además, el tráfico de memoria entre la GPU y la memoria del sistema consume más energía que el tráfico de memoria entre la GPU y la memoria de gráficos en chip.

[0014] Normalmente, las técnicas de renderización de fragmentos experimentan menos tráfico de memoria total entre la GPU y la memoria del sistema que la renderización directa, ya que las técnicas de renderización de fragmentos usan la GMEM para almacenar el búfer de color y el búfer de profundidad. Sin embargo, cada vez más aplicaciones y arquitecturas de GPU usan pruebas de profundidad, incluyendo pruebas de profundidad avanzadas, para reducir la complejidad de la lectura/escritura del color. Es decir, realizar operaciones de gráficos, tales como el sombreado, en base a los valores en el búfer de profundidad hace que dicho sombreado solo se realice en los píxeles "más cercanos" al usuario. Así pues, se necesitan menos escrituras de valores de píxeles en el búfer de color. Si la mayoría de los píxeles en un fragmento se escriben varias veces (o solo una vez) en el búfer de color, escribir píxeles del búfer de color en la GMEM y, a continuación, resolver el búfer de color en la memoria del sistema puede dar como resultado un menor rendimiento de memoria que solo escribir píxeles en el búfer de color directamente en la memoria del

sistema. Es decir, el uso de la GMEM proporciona las mayores ventajas cuando los valores de color y/o profundidad se escriben en el búfer de color y el búfer de profundidad en la GMEM muchas veces. Escribir valores de color en la GMEM solo unas pocas veces por fragmento, y, a continuación, resolver dichos valores en el sistema, puede dar como resultado un menor rendimiento de memoria y energía que sencillamente escribir los valores en la memoria del sistema directamente.

[0015] En vista de estos inconvenientes, esta divulgación propone un sistema de renderización híbrida en el que los valores de píxeles se escriben en un búfer de color directamente en la memoria del sistema (por ejemplo, DRAM), y los valores de profundidad se escriben en un búfer de profundidad en la GMEM. Las técnicas de renderización híbrida de esta divulgación se pueden usar con o sin técnicas de fragmentación. Debido a que la complejidad de la profundidad (es decir, el número de lecturas/escrituras de profundidad) es, en general, mayor que la complejidad del color (es decir, el número de lecturas/escrituras de valores de píxeles), las técnicas de esta divulgación pueden ahorrar resoluciones innecesarias del búfer de color de la GMEM en la DRAM. Es decir, las técnicas de renderización híbrida de esta divulgación proporcionan una mayor eficiencia de uso de memoria para situaciones en las que es probable que los valores de píxeles se escriban en la GMEM varias veces, o solo una vez, por fragmento.

[0016] La FIG. 1 es un diagrama de bloques que ilustra un dispositivo informático de ejemplo 2 que se puede usar para implementar las técnicas de esta divulgación para realizar un modo de renderización híbrida. El dispositivo informático 2 puede comprender, por ejemplo, un ordenador personal, un ordenador de escritorio, un ordenador portátil, una tableta, una estación de trabajo de ordenador, una plataforma o consola de videojuegos, un teléfono móvil, tal como, por ejemplo, un teléfono celular o satelital, un teléfono fijo, un teléfono de Internet, un dispositivo de mano tal como un dispositivo portátil de videojuegos o un asistente digital personal (PDA), un reproductor de música personal, un reproductor de vídeo, un dispositivo de pantalla, una televisión, un descodificador de televisión, un servidor, un dispositivo de red intermedio, un ordenador central, cualquier dispositivo móvil o cualquier otro tipo de dispositivo que procesa y/o visualiza datos gráficos.

[0017] Como se ilustra en el ejemplo de la FIG. 1, el dispositivo informático 2 puede incluir una interfaz de entrada de usuario 4, una unidad de procesamiento central (CPU) 6, un controlador de memoria 8, una memoria del sistema 10, una unidad de procesamiento de gráficos (GPU) 12, una memoria de gráficos (GMEM) 14, una interfaz de visualización 16, una pantalla 18 y buses 20 y 22. Se debe observar que, en algunos ejemplos, la GMEM 14 puede estar "en el chip" con la GPU 12. En algunos casos, todos los elementos de hardware mostrados en la FIG. 1 pueden estar en el chip, por ejemplo, en un diseño de sistema en un chip (System on a Chip, SoC). La interfaz de entrada de usuario 4, la CPU 6, el controlador de memoria 8, la GPU 12 y la interfaz de visualización 16 se pueden comunicar entre sí usando el bus 20. El controlador de memoria 8 y la memoria del sistema 10 también se pueden comunicar entre sí usando el bus 22. Los buses 20, 22 pueden ser cualquiera de una variedad de estructuras de bus, tales como un bus de tercera generación (por ejemplo, un bus Hyper Transport o un bus InfiniBand), un bus de segunda generación (por ejemplo, un bus Advanced Graphics Port, un bus Peripheral Component Interconnect (PCI) Express, o un bus Advanced exentisible Interface (AXI)) u otro tipo de bus o dispositivo de interconexión. Cabe señalar que la configuración específica de los buses y las interfaces de comunicación entre los diferentes componentes mostrados en la FIG. 1 es meramente a modo de ejemplo, y se pueden usar otras configuraciones de dispositivos informáticos y/u otros sistemas de procesamiento de gráficos con los mismos o diferentes componentes para implementar las técnicas de esta divulgación.

[0018] La CPU 6 puede comprender un procesador de propósito general o de propósito especial que controle el funcionamiento del dispositivo informático 2. Un usuario puede proporcionar datos de entrada al dispositivo informático 2 para hacer que la CPU 6 ejecute una o más aplicaciones de software. Las aplicaciones de software que se ejecutan en la CPU 6 pueden incluir, por ejemplo, un sistema operativo, una aplicación de procesador de textos, una aplicación de correo electrónico, una aplicación de hoja de cálculo, una aplicación de reproductor multimedia, una aplicación de videojuegos, una aplicación de interfaz gráfica de usuario u otro programa. Adicionalmente, la CPU 6 puede ejecutar un controlador de GPU 7 para controlar el funcionamiento de la GPU 12. El usuario puede proporcionar entrada al dispositivo informático 2 por medio de uno o más dispositivos de entrada (no mostrados) tal como un teclado, un ratón, un micrófono, un panel táctil u otro dispositivo de entrada que esté acoplado al dispositivo informático 2 por medio de la interfaz de usuario 4.

[0019] Las aplicaciones de software que se ejecutan en la CPU 6 pueden incluir una o más instrucciones de renderización de gráficos que indiquen al procesador 6 que inicie la renderización de datos de gráficos en el dispositivo de visualización 18. En algunos ejemplos, las instrucciones de software se pueden ajustar a una interfaz de programación de aplicaciones (API) de gráficos, tal como, por ejemplo, una API de Librería Abierta de Gráficos (OpenGL®), una API de Sistemas Integrados de Librería Abierta de Gráficos (OpenGL ES), una API Direct3D, una API X3D, una API RenderMan, una API WebGL o cualquier otra API gráfica estándar pública o propietaria. Para procesar las instrucciones de renderización de gráficos, la CPU 6 puede enviar uno o más comandos de renderización de gráficos a la GPU 12 (por ejemplo, a través del controlador de GPU 7) para provocar que la GPU 12 realice una parte o la totalidad de la renderización de los datos de gráficos. En algunos ejemplos, los datos de gráficos a renderizar pueden incluir una lista de primitivas de gráficos, por ejemplo, puntos, líneas, triángulos, cuadriláteros, tiras de triángulos, etc.

[0020] El controlador de memoria 8 facilita la transferencia de los datos que entran y salen de la memoria del sistema 10. Por ejemplo, el controlador de memoria 8 puede recibir comandos de lectura y escritura de memoria, y atender dichos comandos con respecto a la memoria del sistema 10 para proporcionar servicios de memoria para los componentes en el dispositivo informático 2. El controlador de memoria 8 se acopla de forma comunicativa a la memoria del sistema 10 por medio del bus de memoria 22. Aunque el controlador de memoria 8 se ilustra en la FIG. 1 como un módulo de procesamiento que está separado tanto de la CPU 6 como de la memoria 10, en otros ejemplos, parte o toda la funcionalidad del controlador de memoria 8 se puede implementar en una o ambas de la CPU 6 y la memoria del sistema 10.

[0021] La memoria del sistema 10 puede almacenar módulos de programa y/o instrucciones que son accesibles para su ejecución mediante la CPU 6 y/o datos para su uso mediante los programas que se ejecutan en la CPU 6. Por ejemplo, la memoria del sistema 10 puede almacenar una aplicación de gestor de ventanas que usa la CPU 6 para presentar una interfaz gráfica de usuario (GUI) en la pantalla 18. Además, la memoria del sistema 10 puede almacenar aplicaciones de usuario y datos de superficie de aplicación asociados con las aplicaciones. La memoria del sistema 10 puede almacenar adicionalmente información para su uso por y/o generada por otros componentes del dispositivo informático 2. Por ejemplo, la memoria del sistema 10 puede actuar como una memoria de dispositivo para la GPU 12 y puede almacenar datos que puede usar la GPU 12, así como datos resultantes de operaciones realizadas por la GPU 12. Por ejemplo, la memoria del sistema 10 puede almacenar cualquier combinación de memorias intermedias de texturas, memorias intermedias de profundidad, memorias intermedias de plantillas, memorias intermedias de vértices, memorias intermedias de tramas, o similares. La memoria del sistema 10 puede incluir una o más memorias o dispositivos de almacenamiento volátiles o no volátiles, tales como, por ejemplo, memoria de acceso aleatorio (RAM), RAM estática (SRAM), RAM dinámica (DRAM), memoria de solo lectura (ROM), ROM programable borrable (EPROM), ROM programable borrable eléctricamente (EEPROM), memoria Flash, medios de datos magnéticos o medios de almacenamiento óptico.

[0022] La GPU 12 se puede configurar para realizar operaciones de gráficos para renderizar una o más primitivas de gráficos en la pantalla 18. Por tanto, cuando una de las aplicaciones de software que se ejecuta en la CPU 6 requiere procesamiento de gráficos, la CPU 6 puede proporcionar comandos de gráficos y datos de gráficos a la GPU 12 para su renderización en la pantalla 18. Los datos de gráficos pueden incluir, por ejemplo, comandos de dibujo, información de estado, información de primitivas, información de textura, etc. La GPU 12 se puede construir, en algunos casos, con una estructura altamente paralela que proporcione un procesamiento más eficaz de operaciones complejas relacionadas con gráficos que la CPU 6. Por ejemplo, la GPU 12 puede incluir una pluralidad de elementos de procesamiento que están configurados para funcionar en múltiples vértices o píxeles de manera paralela. La naturaleza altamente paralela de la GPU 12 puede, en algunos casos, permitir que la GPU 12 dibuje imágenes de gráficos (por ejemplo, GUI y escenas de gráficos bidimensionales (2D) y/o tridimensionales (3D)) en la pantalla 18 más rápidamente que representando las escenas directamente en la pantalla 18 usando la CPU 6.

[0023] La GPU 12, en algunos casos, se puede integrar en una placa base del dispositivo informático 2. En otros casos, la GPU 12 puede estar presente en una tarjeta de gráficos que esté instalada en un puerto en la placa base del dispositivo informático 2 o se puede incorporar de otro modo dentro de un dispositivo periférico configurado para funcionar conjuntamente con el dispositivo informático 2. La GPU 12 puede incluir uno o más procesadores, tales como uno o más microprocesadores, circuitos integrados específicos de la aplicación (ASIC), matrices de puertas programables por campo (FPGA), procesadores de señales digitales (DSP) u otro circuito lógico integrado o discreto equivalente.

[0024] La GPU 12 se puede acoplar directamente a la GMEM 14. Por tanto, la GPU 12 puede leer y escribir datos en la GMEM 14 sin usar el bus 20. En otras palabras, la GPU 12 puede procesar datos localmente usando un almacenamiento local, en lugar de la memoria fuera del chip. Esto permite que la GPU 12 funcione de una manera más eficaz eliminando la necesidad de que la GPU 12 lea y escriba datos por medio de un bus 20, que puede experimentar un tráfico pesado. Sin embargo, en algunos ejemplos, la GPU 12 puede no incluir una memoria independiente, sino usar en su lugar la memoria 10 por medio del bus 20. La GMEM 14 puede incluir una o más memorias o dispositivos de almacenamiento volátiles o no volátiles, tales como, por ejemplo, memoria de acceso aleatorio (RAM), RAM estática (SRAM), RAM dinámica (DRAM), ROM programable borrable (EPROM), ROM programable borrable eléctricamente (EEPROM), memoria Flash, un medio de datos magnéticos o un medio de almacenamiento óptico.

[0025] La CPU 6 y/o la GPU 12 pueden almacenar datos de imagen renderizados en un búfer de tramas 15. El búfer de tramas 15 puede ser una memoria independiente o se puede asignar dentro de la memoria del sistema 10. La interfaz de pantalla 16 puede recuperar los datos del búfer de tramas 15 y configurar la pantalla 18 para mostrar la imagen representada mediante los datos de imagen renderizados. En algunos ejemplos, la interfaz de visualización 16 puede incluir un convertidor de digital a analógico (DAC) que está configurado para convertir los valores digitales recuperados del búfer de tramas en una señal analógica consumible por la pantalla 18. En otros ejemplos, la interfaz de visualización 16 puede pasar los valores digitales directamente a la pantalla 18 para su procesamiento. La pantalla 18 puede incluir un monitor, un televisor, un dispositivo de proyección, una pantalla de cristal líquido (LCD), un panel de pantalla de plasma, una matriz de diodos emisores de luz (LED), tal como un LED orgánico (OLED), una pantalla de tubo de rayos catódicos (CRT), papel electrónico, una pantalla de emisión de electrones de conducción de

superficie (SED), una pantalla de televisión láser, una pantalla de nanocristales u otro tipo de unidad de pantalla. La pantalla 18 puede estar integrada dentro del dispositivo informático 2. Por ejemplo, la pantalla 18 puede ser una pantalla de un teléfono móvil. De forma alternativa, la pantalla 18 puede ser un dispositivo independiente acoplado al dispositivo informático 2 a través de un enlace de comunicaciones alámbrico o inalámbrico. Por ejemplo, la pantalla 18 puede ser un monitor de ordenador o una pantalla plana conectada a un ordenador personal a través de un cable o un enlace inalámbrico.

[0026] De acuerdo con un ejemplo de la divulgación, la GPU 12 se puede configurar para renderizar una o más porciones de la trama usando una o más operaciones de gráficos, y escribir datos de color directamente en el búfer de color en la memoria del sistema 10 de acuerdo con la una o más operaciones de gráficos. La GPU 12 se puede configurar además para escribir datos de profundidad en un búfer de profundidad en la GMEM 14 de acuerdo con una o más operaciones de gráficos, y resolver el búfer de profundidad en la GMEM 14 en la memoria del sistema 10 cuando se completa la renderización de la una o más porciones de la trama.

[0027] La FIG. 2 es un diagrama de bloques que ilustra con mayor detalle implementaciones de ejemplo de la CPU 6, de la GPU 12 y de la memoria de sistema 10 de la FIG. 1. La CPU 6 puede incluir al menos una aplicación de software 24, una API de gráficos 26 y un controlador de GPU 7, cada uno de los cuales puede ser una o más aplicaciones de software o servicios que se ejecutan en la CPU 6. La GPU 12 puede incluir una tubería de procesamiento de gráficos 30 que incluye una pluralidad de etapas de procesamiento de gráficos que funcionan en conjunto para ejecutar comandos de procesamiento de gráficos. La GPU 12 se puede configurar para ejecutar la tubería de procesamiento de gráficos 30 en un modo de renderización híbrida, de acuerdo con las técnicas de esta divulgación. Como se muestra en la FIG. 2, la tubería de procesamiento de gráficos 30 puede incluir un motor de comandos 32, una etapa de procesamiento de geometría 34, una etapa de rasterización 36 y una tubería de procesamiento de píxeles 38. Cada uno de los componentes en la tubería de procesamiento de gráficos 30 se puede implementar como componentes de función fija, componentes programables (por ejemplo, como parte de un programa de sombreado que se ejecuta en una unidad de sombreador programable), o como una combinación de componentes de función fija y programables. La memoria disponible para la CPU 6 y la GPU 12 puede incluir la memoria de sistema 10 y el búfer de tramas 15. El búfer de tramas 15 puede formar parte de la memoria de sistema 10 o puede estar separada de la memoria de sistema 10. El búfer de tramas 15 puede almacenar una trama de datos de imagen renderizados.

[0028] Como se explicará con más detalle a continuación, de acuerdo con las técnicas de la divulgación, la GPU 12 se puede configurar para almacenar valores de color en el búfer de profundidad 13A en la GMEM 14. La GPU 12 se puede configurar además para resolver el búfer de profundidad 13A en el búfer de profundidad 13B en la memoria del sistema 10 cuando se completa la renderización de una o más porciones de una trama (por ejemplo, un fragmento). La GPU 12 se puede configurar además para almacenar valores de color en el búfer de color 11 directamente en la memoria del sistema 10 (es decir, sin almacenar valores de color en la GMEM 14).

[0029] La aplicación de software 24 puede ser cualquier aplicación que use la funcionalidad de la GPU 12. Por ejemplo, la aplicación de software 24 puede ser una aplicación de GUI, un sistema operativo, una aplicación de correlación portátil, un programa de diseño asistido por ordenador para aplicaciones de ingeniería o artísticas, una aplicación de videojuegos u otro tipo de aplicación de software que use gráficos 2D o 3D.

[0030] La aplicación de software 24 puede incluir una o más instrucciones de dibujo que indiquen a la GPU 12 que renderice una interfaz gráfica de usuario (GUI) y/o una escena gráfica. Por ejemplo, las instrucciones de dibujo pueden incluir instrucciones que definan un conjunto de una o más primitivas de gráficos a renderizar mediante la GPU 12. En algunos ejemplos, las instrucciones de dibujo pueden, conjuntamente, definir todo o parte de una pluralidad de superficies de ventanas usadas en una GUI. En ejemplos adicionales, las instrucciones de dibujo pueden, conjuntamente, definir todo o parte de una escena de gráficos que incluya uno o más objetos de gráficos dentro de un espacio modelo o espacio mundial definido por la aplicación.

[0031] La aplicación de software 24 puede invocar al controlador de GPU 7, por medio de la API de gráficos 26, para enviar uno o más comandos a la GPU 12 para renderizar una o más primitivas de gráficos en imágenes de gráficos visualizables. Por ejemplo, la aplicación de software 24 puede invocar al controlador de GPU 7, por medio de la API de gráficos 26, para proporcionar definiciones de primitivas a la GPU 12. En algunos casos, las definiciones de primitivas se pueden proporcionar a la GPU 12 en forma de una lista de primitivas de dibujo, por ejemplo, triángulos, rectángulos, abanicos de triángulos, tiras de triángulos, etc. Las definiciones de primitivas pueden incluir especificaciones de vértices que especifiquen uno o más vértices asociados a las primitivas que se vayan a renderizar. Las especificaciones de vértices pueden incluir coordenadas de posición para cada vértice y, en algunos casos, otros atributos asociados al vértice, tales como, por ejemplo, coordenadas de color, vectores normales y coordenadas de textura. Las definiciones de primitivas también pueden incluir información de tipo de primitiva (por ejemplo, triángulo, rectángulo, abanico de triángulos, tira de triángulos, etc.), información de escalado, información de rotación y similares. En base a las instrucciones emitidas por la aplicación de software 24 para el controlador de la GPU 7, el controlador de la GPU 7 puede formular uno o más comandos que especifiquen una o más operaciones de gráficos para que la GPU 12 las realice para renderizar la primitiva. Cuando la GPU 12 recibe un comando de la CPU 6, la tubería de procesamiento de gráficos 30 descodifica el comando y configura uno o más elementos de procesamiento dentro de

la tubería de procesamiento de gráficos 30 para realizar una o más operaciones de gráficos especificadas en el comando. Después de realizar las operaciones de gráficos especificadas, la tubería de procesamiento de gráficos 30 envía los datos renderizados al búfer de tramas 15 asociada con un dispositivo de visualización.

[0032] De acuerdo con las técnicas de esta divulgación, la GPU 12 se puede configurar para renderizar una o más porciones de una trama usando operaciones de procesamiento de píxeles y una o más pruebas de profundidad. Las operaciones de procesamiento de píxeles pueden incluir operaciones de gráficos que usan, cambian, actualizan y/o, en general, manipulan los valores de color para un píxel. Los valores de color de un píxel pueden estar en un formato RGB (rojo, azul verde), formato YUV (luma (Y) y dos valores de croma (UV)), o en cualquier otro formato. La GPU 12 también puede ejecutar pruebas de profundidad (u otras operaciones de gráficos) para usar, cambiar, actualizar y/o, en general, manipular los valores de profundidad. Los valores de profundidad pueden indicar la proximidad de píxel a un espectador con respecto a otros píxeles. Los valores de profundidad se pueden usar para determinar qué primitivas son visibles en la escena renderizada final.

[0033] De acuerdo con las técnicas de esta divulgación, la GPU 12 se configura para almacenar los valores de color directamente en el búfer de color 11 en la memoria del sistema 10. Es decir, la GPU 12 actualiza el búfer de color 11 sin escribir en ningún búfer de color en la GMEM 14. La GPU 12 también se puede configurar para almacenar valores de profundidad en el búfer de profundidad 13A en la GMEM 14 para una o más porciones de una trama (por ejemplo, un fragmento de una trama). Cuando la GPU 12 ha terminado de renderizar una o más porciones de una trama (por ejemplo, un fragmento de una trama), la GPU 12 puede resolver el búfer de profundidad 13A en la GMEM 14 en el búfer de profundidad 13B en la memoria del sistema 10.

[0034] El controlador de la GPU 7 se puede configurar además para compilar uno o más programas de sombreado, y para descargar los programas de sombreado compilados en una o más unidades de sombreado programables contenidas dentro de la GPU 12. Los programas de sombreado se pueden escribir en un lenguaje de sombreado de alto nivel, tal como, por ejemplo, un Lenguaje de Sombreado OpenGL (GLSL), un Lenguaje de Sombreado de Alto Nivel (HLSL), un lenguaje de sombreado de C para Gráficos (Cg), etc. Los programas de sombreado compilados pueden incluir una o más instrucciones que controlan el funcionamiento de una unidad de sombreado programable dentro de la GPU 12. Por ejemplo, los programas de sombreado pueden incluir programas de sombreado de vértices y/o programas de sombreado de píxeles. Un programa de sombreado de vértices puede controlar la ejecución de una unidad de sombreado de vértices programable o de una unidad de sombreado unificada e incluir instrucciones que especifiquen una o más operaciones por vértice. Un programa de sombreado de píxeles puede incluir programas de sombreado de píxeles que controlan la ejecución de una unidad de sombreado de píxeles programable o de una unidad de sombreado unificada, e incluir instrucciones que especifican una o más operaciones por píxel. De acuerdo con algunos ejemplos de esta divulgación, un programa de sombreado de píxeles también puede incluir instrucciones que provocan que se recuperen selectivamente los valores de textura para los píxeles de origen en base a los valores alfa de destino correspondientes para los píxeles de origen. La GPU 12 puede ejecutar sombreadores de vértices y sombreadores de píxeles de acuerdo con las técnicas de esta divulgación. Es decir, cualquier operación de gráficos realizada por un sombreador de píxeles o un sombreador de vértices que actualiza los valores de color provocará que la GPU 12 actualice el búfer de color 11 en la memoria del sistema 10. Cualquier operación de gráficos realizada por un sombreador de píxeles o un sombreador de vértices que actualice los valores de profundidad provocará que la GPU 12 actualice el búfer de profundidad 13A en la GMEM. De nuevo, la GPU 12 se configura para resolver el búfer de profundidad 13A en el búfer de profundidad 13B en la memoria del sistema 10 en la porción de la trama que está siendo renderizado por la GPU 12 que está completa.

[0035] La tubería de procesamiento de gráficos 30 se puede configurar para recibir uno o más comandos de procesamiento de gráficos de la CPU 6, por medio del controlador de gráficos 28, y para ejecutar los comandos de procesamiento de gráficos para generar imágenes de gráficos visualizables. Como se analiza anteriormente, la tubería de procesamiento de gráficos 30 incluye una pluralidad de etapas que funcionan en conjunto para ejecutar comandos de procesamiento de gráficos. Cabe señalar, sin embargo, que dichas etapas no necesariamente se deben implementar en bloques de hardware separados. Por ejemplo, porciones de la etapa de procesamiento de geometría 34 y la tubería de procesamiento de píxeles 38 se pueden implementar como parte de una unidad de sombreado unificada. De nuevo, la tubería de procesamiento de gráficos 30 se puede configurar para ejecutarse en uno de una pluralidad de modos de renderización diferentes, incluyendo un modo de renderización de fragmentos y un modo de renderización directa.

[0036] El motor de comandos 32 puede recibir comandos de procesamiento de gráficos y configurar las etapas de procesamiento restantes dentro de la tubería de procesamiento de gráficos 30 para realizar diversas operaciones para llevar a cabo los comandos de procesamiento de gráficos. Los comandos de procesamiento de gráficos pueden incluir, por ejemplo, comandos de dibujo y comandos de estado de gráficos. Los comandos de dibujo pueden incluir comandos de especificación de vértices que especifican las coordenadas de posición para uno o más vértices y, en algunos casos, otros valores de atributos asociados con cada uno de los vértices, tales como, por ejemplo, coordenadas de color, vectores normales, coordenadas de textura y coordenadas de niebla. Los comandos de estado de gráficos pueden incluir comandos de tipo de primitiva, comandos de transformación, comandos de iluminación, etc. Los comandos de tipo de primitiva pueden especificar el tipo de primitiva que se va a renderizar y/o cómo se combinan los vértices para formar una primitiva. Los comandos de transformación pueden especificar los tipos de

transformaciones a realizar en los vértices. Los comandos de iluminación pueden especificar el tipo, la dirección y/o la ubicación de diferentes luces dentro de una escena de gráficos. El motor de comandos 32 puede provocar que la etapa de procesamiento de geometría 34 realice el procesamiento de geometría con respecto a los vértices y/o las primitivas asociadas con uno o más comandos recibidos.

[0037] La etapa de procesamiento de geometría 34 puede realizar operaciones por vértice y/u operaciones de configuración de primitivas en uno o más vértices para generar datos de primitivas para la etapa de rasterización 36. Cada vértice se puede asociar con un conjunto de atributos, tales como, por ejemplo, coordenadas de posición, valores de color, un vector normal y coordenadas de textura. La etapa de procesamiento de geometría 34 modifica uno o más de estos atributos de acuerdo con diversas operaciones por vértice. Por ejemplo, la etapa de procesamiento de geometría 34 puede realizar una o más transformaciones en las coordenadas de posición de vértices para producir coordenadas de posición de vértices modificadas. La etapa de procesamiento de geometría 34 puede, por ejemplo, aplicar una o más de una transformación de modelado, una transformación de visualización, una transformación de proyección, una transformación de modelo-vista, una transformación de modelo-vista-proyección, una transformación de ventana gráfica y una transformación de escalado de rango de profundidad a las coordenadas de posición de vértices para generar las coordenadas de posición de vértices modificadas.

[0038] En algunos casos, las coordenadas de posición de vértices pueden ser coordenadas del espacio del modelo, y las coordenadas de posición de vértices modificadas pueden ser coordenadas del espacio de la pantalla. Las coordenadas del espacio de la pantalla se pueden obtener después de la aplicación de las transformaciones de modelado, visualización, proyección y ventana gráfica. En algunos casos, la etapa de procesamiento de geometría 34 también puede realizar operaciones de iluminación por vértice en los vértices para generar coordenadas de color modificadas para los vértices. La etapa de procesamiento de geometría 34 también puede realizar otras operaciones que incluyen, por ejemplo, transformaciones normales, operaciones de normalización normales, recorte de volumen de vista, división homogénea y/u operaciones de eliminación de la parte posterior.

[0039] La etapa de procesamiento de geometría 34 puede producir datos de primitivas que incluyen un conjunto de uno o más vértices modificados que definen una primitiva a rasterizar, así como datos que especifican cómo se combinan los vértices para formar una primitiva. Cada uno de los vértices modificados puede incluir, por ejemplo, coordenadas de posición de vértice modificadas y valores de atributos de vértice procesados asociados con el vértice. Los datos de primitivas pueden corresponder colectivamente a una primitiva a rasterizar mediante otras etapas de la tubería de procesamiento de gráficos 30. Conceptualmente, cada vértice puede corresponder a una esquina de una primitiva donde se encuentran dos bordes de la primitiva. La etapa de procesamiento de geometría 34 puede proporcionar los datos de la primitiva a la etapa de rasterización 36 para su procesamiento adicional.

[0040] En algunos ejemplos, la totalidad o parte de la etapa de procesamiento de geometría 34 se puede implementar mediante uno o más programas de sombreado que se ejecutan en una o más unidades de sombreado. Por ejemplo, la etapa de procesamiento de geometría 34 se puede implementar, en dichos ejemplos, mediante un sombreador de vértices, un sombreador de geometría o cualquier combinación de los mismos. En otros ejemplos, la etapa de procesamiento de geometría 34 se puede implementar como una tubería de procesamiento de hardware de función fija o como una combinación de hardware de función fija y uno o más programas de sombreado que se ejecutan en una o más unidades de sombreado.

[0041] La etapa de rasterización 36 se configura para recibir, de la etapa de procesamiento de geometría 34, datos de primitiva que representan una primitiva a rasterizar, y para rasterizar la primitiva para generar una pluralidad de píxeles de origen que corresponden a la primitiva rasterizada. En algunos ejemplos, la etapa de rasterización 36 puede determinar qué localizaciones de píxeles de pantalla están cubiertas por la primitiva a rasterizar, y generar un píxel de origen para cada localización de píxel de pantalla que se determina que debe cubrir la primitiva. La etapa de rasterización 36 puede determinar qué localizaciones de píxeles de pantalla están cubiertas por una primitiva usando técnicas conocidas por los expertos en la técnica, tales como, por ejemplo, una técnica de selección a través de bordes, que evalúa las ecuaciones en los bordes, etc. La etapa de rasterización 36 puede proporcionar los píxeles de origen resultantes a la tubería de procesamiento de píxeles 38 para su procesamiento adicional.

[0042] Los píxeles de origen generados por la etapa de rasterización 36 pueden corresponder a una localización de píxel de pantalla, por ejemplo, un píxel de destino, y estar asociados con uno o más atributos de color. Se puede decir que todos los píxeles de origen generados para una primitiva rasterizada específica están asociados con la primitiva rasterizada. Los píxeles que la etapa 36 de rasterización determina que están cubiertos por una primitiva pueden incluir conceptualmente píxeles que representan los vértices de la primitiva, píxeles que representan los bordes de la primitiva y píxeles que representan el interior de la primitiva.

[0043] La tubería de procesamiento de píxeles 38 se configura para recibir un píxel de origen asociado con una primitiva rasterizada, y para realizar una o más operaciones por píxel en el píxel de origen. Las operaciones por píxel que puede realizar la tubería de procesamiento de píxeles 38 incluyen, por ejemplo, prueba alfa, asignación de texturas, cálculo de color, sombreado de píxeles, iluminación por píxel, procesamiento de niebla, mezcla, un texto de propiedad de píxel, una prueba alfa de origen, una prueba de plantilla, una prueba de profundidad, una prueba de tijeras y/u operaciones de punteado. Además, la tubería de procesamiento de píxeles 38 puede ejecutar uno o más

programas de sombreados de píxeles para realizar una o más operaciones por píxel. Los datos resultantes producidos por la tubería de procesamiento de píxeles 38 pueden denominarse en el presente documento datos de píxeles de destino y almacenarse en el búfer de tramas 15. Los datos de píxeles de destino pueden estar asociados con un píxel de destino en el búfer de tramas 15 que tiene la misma localización en pantalla que el píxel de origen que se procesó. Los datos de píxeles de destino pueden incluir datos tales como, por ejemplo, valores de color en el búfer de color 11, valores alfa de destino, valores de profundidad en el búfer de profundidad 13B, etc.

[0044] El búfer de tramas 15 almacena píxeles de destino para la GPU 12. Cada píxel de destino puede estar asociado con una localización única de píxel en pantalla. En algunos ejemplos, el búfer de tramas 15 puede almacenar componentes de color y un valor alfa de destino para cada píxel de destino. Por ejemplo, el búfer de tramas 15 puede almacenar componentes Rojo, Verde, Azul, Alfa (RG-BA) para cada píxel, donde los componentes "RGB" corresponden a valores de color y el componente "A" corresponde a un valor alfa de destino. Aunque el búfer de tramas 15 y la memoria de sistema 10 se ilustran como unidades de memoria separadas, en otros ejemplos, el búfer de tramas 15 puede formar parte de la memoria de sistema 10.

[0045] Como se analiza anteriormente, la GPU 12 se puede configurar para renderizar una o más porciones de una trama a la vez. En algunos ejemplos de la divulgación, la GPU 12 se puede configurar para renderizar una trama completa a la vez. En este ejemplo, la GPU 12 resolvería el búfer de profundidad 13A en la GMEM 14 cuando se renderice toda la trama. En otros ejemplos de la divulgación, la GPU 12 se puede configurar para renderizar solo una porción de una trama a la vez. Es decir, la GPU 12 se puede configurar para dividir una trama en regiones denominadas fragmentos o mosaicos, y renderizar cada uno de los fragmentos por separado. De acuerdo con las técnicas de esta divulgación, la GPU 12 se puede configurar para dividir una trama en fragmentos y almacenar cualquier valor de profundidad asociado con la renderización del fragmento en el búfer de profundidad 13A en la GMEM 14. Cuando se completa la renderización del fragmento, la GPU 12 se configura para resolver el búfer de profundidad 13A en el búfer de profundidad 13B en la memoria del sistema 10. Sin embargo, la GPU 12 se configura para almacenar directamente cualquier valor de color relacionado con la renderización del fragmento en el búfer de color 11 en la memoria del sistema 10.

[0046] A este respecto, dado que el búfer de color 11 se puede usar para cada fragmento en una trama, el búfer de color 11 se puede configurar con un tamaño de memoria de modo que todos los valores de color de una trama se ajusten al búfer de color 11. Sin embargo, el búfer de profundidad 13A solo necesita configurarse con un tamaño de modo que todos los valores de profundidad de un fragmento se ajusten al búfer de profundidad 13A. El búfer de profundidad 13B se puede configurar con un tamaño de modo que todos los valores de profundidad de una trama se ajusten al búfer de profundidad 13B. A continuación se analizarán detalles adicionales con respecto a la fragmentación.

[0047] La FIG. 3 es un diagrama conceptual que ilustra una trama dividido en fragmentos para el modo de renderización de fragmentos. La trama 40 se puede dividir en una pluralidad de fragmentos, tal como el fragmento 42. Típicamente, el hardware de gráficos contendrá memoria rápida (por ejemplo, la GMEM 14 de la FIG. 2) que es de un tamaño suficiente para contener al menos un fragmento de datos. De acuerdo con los ejemplos de esta divulgación, la GMEM 14 se puede configurar con un tamaño para contener un valor de fragmento de los datos de profundidad almacenados en el búfer de profundidad 13A.

[0048] Como parte de una única pasada de renderización para un fragmento particular de la trama, la tubería de procesamiento de gráficos 30 puede renderizar la totalidad o un subconjunto del lote de primitivas con respecto a un subconjunto particular de los píxeles de destino (por ejemplo, un fragmento particular de los píxeles de destino) de la trama. Después de realizar una primera pasada de renderización con respecto a un primer fragmento, la tubería de procesamiento de gráficos 30 puede realizar una segunda pasada de renderización con respecto a un segundo fragmento, etc. La tubería de procesamiento de gráficos 30 puede atravesar incrementalmente los fragmentos hasta que se hayan renderizado las primitivas asociadas con todos los fragmentos.

[0049] La FIG. 4 es un diagrama conceptual que muestra fragmentos usados en un modo de renderización híbrida, de acuerdo con las técnicas de esta divulgación, con más detalle. Los fragmentos 44, 46, 48 y 50 se renderizan/rasterizan para contener múltiples píxeles 52. Una o más primitivas de gráficos pueden ser visibles en cada fragmento. Por ejemplo, porciones del triángulo A (Tri A) son visibles tanto en el fragmento 44 como en el fragmento 48. Porciones del triángulo B (Tri B) son visibles en cada uno de los fragmentos 44, 46, 48 y 50. El triángulo C (Tri C) solo es visible en el fragmento 46. La GPU 12 realiza una etapa adicional antes de la renderización para determinar qué triángulos en el fragmento son realmente visibles en la escena renderizada final (esto a veces se denomina fragmentación de hardware). Por ejemplo, algunos triángulos pueden estar detrás de uno o más de otros triángulos y no serán visibles en la escena renderizada final. De esta manera, no es necesario renderizar los triángulos que no son visibles para ese fragmento.

[0050] De acuerdo con las técnicas de esta divulgación, mientras se realiza una pasada de renderización particular, los datos de profundidad para el fragmento asociado con esa pasada de renderización particular se pueden almacenar en el búfer de profundidad 13A en la GMEM 14. Sin embargo, los datos de píxeles para el fragmento asociado con esa pasada de renderización particular se pueden almacenar en el búfer de color 11 en la memoria del sistema 10.

Después de realizar la pasada de renderización, la GPU 12 puede transferir (es decir, resolver) el contenido del búfer de profundidad 13A en la GMEM 14 en el búfer de profundidad 13B en la memoria del sistema 10. Después de transferir el contenido del búfer de profundidad 13A en la GMEM 14 al búfer de profundidad 13B en la memoria del sistema 10, la GPU 12 puede inicializar el búfer de profundidad 13A en la GMEM 14 a los valores predeterminados y comenzar una pasada de renderización subsiguiente con respecto a un fragmento diferente.

[0051] La FIG. 5 es un diagrama conceptual que muestra una estructura de comandos de ejemplo para renderizar una escena usando el modo de renderización híbrida de esta divulgación junto con técnicas de fragmentación de "software". El búfer indirecto de nivel 1 (IB1) 60 contiene una serie de comandos de ejecución para dirigir la GPU 12 para que realice las diversas etapas de la tubería de procesamiento de gráficos 30 (es decir, para que ejecute una o más operaciones de gráficos). Cada comando de ejecución en la IB1 60 es esencialmente un puntero a una o más memorias intermedias indirectas de nivel 2 (IB2) que contienen comandos para diversos aspectos de la tubería de renderización. De esta forma, se establece una estructura de dos o más niveles para ejecutar la tubería de renderización de gráficos. La GPU 12 puede pasar secuencialmente a través de cada comando de ejecución en la IB1 60, donde cada ejecución en la IB1 60 apunta a una pila específica de comandos almacenados en una IB2. La IBI y la IB2 pueden ser memorias que están en la GPU 12 integrada o pueden ser memorias externas a la GPU 12, tal como la memoria del sistema 10.

[0052] El comando de ejecución de preámbulo en la IB1 60 apunta a una IB2 de preámbulo 62 que contiene comandos de preámbulo que puede ejecutar la GPU 12. Por ejemplo, la IB2 de preámbulo 62 puede incluir comandos que inicializan ese estado estático de la GPU 12 y establecen el estado de renderización inicial de la GPU 12. El estado estático de la GPU 12 incluye configuraciones que no cambian en base a la aplicación en particular. El estado de renderización, por otra parte, incluye configuraciones de GPU que pueden cambiar en base a la aplicación particular (por ejemplo, una aplicación OpenGL frente a una aplicación Direct X). Una vez que se completan los comandos de la IB2 de preámbulo, el control vuelve a la IB1 60 para realizar el siguiente comando de ejecución.

[0053] El siguiente comando de ejecución en la IB1 60 configura la pasada de renderización para el modo de renderización que se está empleando. De nuevo, en el ejemplo de la FIG. 5, el modo de renderización es el modo de renderización híbrida de esta divulgación que usa la fragmentación de software. A continuación, el comando de ejecución Cargar fragmento en la IB1 60 apunta a los comandos en la IB2 de carga 66. Para la fragmentación de software, los datos de profundidad para un fragmento particular se cargan en el búfer de profundidad 13A en la GMEM 14. Además, los datos de color para el fragmento particular se cargan en el búfer de color 11 en la memoria del sistema 10. A continuación, el control vuelve a la IB1 60 y el comando de ejecución Renderizar fragmento apunta a los comandos en la IB2 de renderización.

[0054] La IB2 de renderización 68 consiste en una serie de comandos de estado y comandos de dibujo para representar los triángulos en el fragmento cargado. Cada comando de dibujo indica a la GPU 12 que dibuje el triángulo de acuerdo con una tubería de procesamiento de gráficos 30 (por ejemplo, incluyendo una etapa de procesamiento de geometría 34, una etapa de rasterización 36 y/o una tubería de procesamiento de píxeles 38) establecida por los comandos y/o el hardware de la GPU. Como se muestra en la IB2 de renderización 68, cada uno de los comandos de dibujo indica que no se usa ningún flujo de visibilidad para determinar si los triángulos específicos son realmente visibles en el fragmento. Los flujos de visibilidad se generan en un modo de renderización de fragmentos que usa la fragmentación de "hardware", y se analizará con más detalle con referencia a la FIG. 6.

[0055] Los comandos de estado en la IB2 de renderización 68 afectan al comportamiento de la tubería de procesamiento de gráficos ejecutada por la GPU 12. Por ejemplo, los comandos de estado pueden cambiar el color, el modo de polígono (por ejemplo, puntos en lugar de sólidos o líneas), la mezcla (activar/desactivar), la prueba de profundidad (activar/desactivar), la texturización (activar/desactivar), la eliminación, el recorte y otras operaciones lógicas. Como se muestra en la IB2 de renderización 68, los comandos de estado se pueden emitir en una base por triángulo (o por primitiva). Es decir, el comando "Estado Tri A" puede afectar al comportamiento de la GPU 12 al dibujar el triángulo A, mientras que los comandos "Estado Tri B1" y "Estado Tri B2" pueden afectar al comportamiento de la GPU 12 al dibujar el triángulo B. Los comandos "Estado Tri B1" y "Estado Tri B2 "simplemente indican que se pueden ejecutar múltiples comandos de estado para cada triángulo. Cualquier comando de renderización en la IB2 de renderización 68 que actualice los valores de color para los píxeles en el fragmento que se está renderizando provoca que la GPU 12 actualice dichos valores de color en el búfer de color 11 en la memoria del sistema 10. Cualquier comando de renderización en la IB2 de renderización 68 que actualice los valores de profundidad para los píxeles en el fragmento que se está renderizando provoca que la GPU 12 actualice el búfer de profundidad 13A en la GMEM 14.

[0056] Después de que se hayan ejecutado todos los comandos en la IB2 de renderización 68 (por ejemplo, después de que se han dibujado todos los triángulos), el control vuelve a la IB1 60. El comando de ejecución Almacenar fragmento puede incluir un puntero a una IB2 de almacenamiento 70 que incluye un comando para almacenar (es decir, resolver) los valores de profundidad en el búfer de profundidad 13A en la GMEM 14 en el búfer de profundidad 13B en la memoria del sistema 10. La pasada de renderización (por ejemplo, los comandos de ejecución desde Configurar pasada de renderización hasta Almacenar fragmento como se muestra en la IB1 60) se repite para cada fragmento 72 para uno o más tramas.

[0057] La FIG. 6 es un diagrama conceptual que muestra una estructura de comandos de ejemplo para renderizar una escena usando un modo de renderización de fragmentos usando la fragmentación de "hardware". Los comandos de ejecución en la IB1 61 son similares a los de la IB1 60 de la FIG. 6 a excepción de los comandos relacionados con un pase de fragmento. Un pase de fragmento se usa para generar un flujo de visibilidad que indica si los triángulos específicos en el fragmento son realmente visibles en la escena renderizada final o no. Por ejemplo, algunos triángulos pueden estar detrás de otro triángulo en la escena y no serán visibles en algunos escenarios (por ejemplo, cuando el triángulo de delante es opaco o cuando no se usa ninguna mezcla). Antes de renderizar los fragmentos 72, la IB1 61 puede incluir un comando de ejecución Pase de fragmento que apunta a comandos en la IB2 de fragmentación 74. La IB2 de fragmentación 74 incluye comandos que provocan que la GPU 12 realice una versión simplificada de una tubería de gráficos (por ejemplo, una versión simplificada de la IB2 de renderización 69), pero añade la etapa de actualizar un flujo de visibilidad para cada triángulo en el fragmento en base a una prueba de profundidad (prueba Z) que determina si el triángulo es visible o no en la escena renderizada final. La IB2 de fragmentación 74 puede generar valores de profundidad al realizar el pase de fragmento, y puede provocar que la GPU 12 actualice el búfer de profundidad 13A en la GMEM 14.

[0058] El objetivo del pase de fragmento es identificar triángulos que se cruzan con el fragmento actual. Como tal, solo es necesario determinar la posición de los vértices del triángulo para identificar si un triángulo se cruza con un fragmento en particular. El pase de fragmento usa un sombreador de vértices simplificado que solo incluye instrucciones que afectan a la posición de los vértices. Por ejemplo, las instrucciones de color, las coordenadas de textura y otras instrucciones que no afectan a la posición del vértice del triángulo se pueden eliminar del sombreador de vértices simplificado usado para el pase de fragmento. El pase de fragmento también usa una rasterización gruesa, en lugar de una rasterización fina, para determinar una profundidad aproximada de cada triángulo. La rasterización gruesa calcula un valor de profundidad con una precisión menor (por ejemplo, usando un número de bits menor) que la rasterización fina. Solo se necesitan valores de profundidad aproximados para determinar si un triángulo es visible en el fragmento. Los sombreadores de píxeles no se usan en el pase de fragmento.

[0059] A continuación, el pase de fragmento usa una prueba de profundidad en los valores de profundidad aproximados para determinar si un triángulo es visible en el fragmento con respecto a otros triángulos en el fragmento. En base a esta prueba de profundidad, se actualiza un flujo de visibilidad. El flujo de visibilidad puede ser una cadena de bits que indica si un triángulo específico en el fragmento renderizado es visible (por ejemplo, 1 indica que un triángulo es visible, 0 indica que un triángulo no es visible).

[0060] Los comandos en la IB2 de renderización 69 son similares a los de la IB de renderización 68 en la FIG. 5, pero para el uso del flujo de visibilidad. Los comandos de dibujo (por ejemplo, Dibujar Tri A, Dibujar Tri B, Dibujar Tri C, etc.) en la IB2 de renderización 69 pueden usar el flujo de visibilidad generado por el pase de fragmento para determinar si es necesario o no dibujar un triángulo específico. Por ejemplo, el dibujo se puede omitir para los triángulos indicados como no visibles por la secuencia de visibilidad.

[0061] La FIG. 7 es un diagrama de flujo que ilustra un procedimiento de acuerdo con un ejemplo de la divulgación. El procedimiento de la FIG. 13 se puede realizar mediante una o más unidades de hardware de la GPU 12. En un ejemplo de la divulgación, la GPU 12 se puede configurar para renderizar una o más porciones de la trama usando una o más operaciones de gráficos (702). La GPU 12 se puede configurar además para escribir datos de color directamente en el búfer de color 11 en la memoria del sistema 10 de acuerdo con una o más operaciones de gráficos (704). La GPU 12 se puede configurar además para escribir datos de profundidad en el búfer de profundidad 13A en la GMEM 14 de acuerdo con una o más operaciones de gráficos (706), y resolver el búfer de profundidad 13A en la GMEM 14 en el búfer de profundidad 13B en la memoria del sistema 10 cuando se completa la renderización de la una o más porciones de la trama. En un ejemplo de la divulgación, la memoria del sistema 10 puede ser DRAM, y la GMEM 14 puede estar en un mismo circuito integrado que la GPU 12.

[0062] En otro ejemplo de la divulgación, la GPU 12 se puede configurar para realizar un pase de fragmento en la una o más porciones de la trama. La una o más porciones de la trama pueden ser un fragmento de la trama. En un ejemplo, para realizar el pase de fragmento, la GPU 12 se configura además para realizar un pase de fragmento de hardware que identifica las primitivas que son visibles en el fragmento. En este ejemplo, la GPU 12 se puede configurar además para representar el fragmento de la trama usando una o más operaciones de gráficos en las primitivas identificadas como visibles en el fragmento. En otro ejemplo, la GPU 12 se configura para realizar un pase de fragmento de software que identifica las primitivas que están dentro del fragmento, y para renderizar el fragmento de la trama usando la una o más operaciones de gráficos en las primitivas que se identifica que están dentro del fragmento.

[0063] En uno o más ejemplos, las funciones descritas anteriormente se pueden implementar en hardware, software, firmware o en cualquier combinación de los mismos. Si se implementa en software, las funciones se pueden almacenar como una o más instrucciones o códigos en un artículo de fabricación que comprende un medio no transitorio legible por ordenador. Los medios legibles por ordenador pueden incluir medios de almacenamiento de datos informáticos. Los medios de almacenamiento de datos pueden ser cualquier medio disponible al que se pueda acceder desde uno o más ordenadores o uno o más procesadores para recuperar instrucciones, código y/o estructuras de datos para la implementación de las técnicas descritas en la presente divulgación. A modo de ejemplo, y no de limitación, dichos

medios legibles por ordenador pueden comprender RAM, ROM, EEPROM, CD-ROM u otro almacenamiento de disco óptico, almacenamiento de disco magnético u otros dispositivos de almacenamiento magnético, memoria flash o cualquier otro medio que pueda usarse para transportar o almacenar un código de programa deseado en forma de instrucciones o estructuras de datos y al que pueda accederse mediante un ordenador. Los discos, como se usan en el presente documento, incluyen el disco compacto (CD), el disco láser, el disco óptico, el disco versátil digital (DVD), el disco flexible y el disco Blu-ray, donde algunos discos reproducen normalmente los datos magnéticamente, mientras que otros discos reproducen los datos ópticamente con láseres. Las combinaciones de los anteriores también se deben incluir dentro del alcance de los medios legibles por ordenador.

[0064] El código se puede ejecutar mediante uno o más procesadores, tales como uno o más DSP, microprocesadores de uso general, ASIC, FPGA u otros circuitos lógicos integrados o discretos equivalentes. Además, en algunos aspectos, la funcionalidad descrita en el presente documento se puede proporcionar dentro de módulos de software y/o módulos de hardware dedicados. Además, las técnicas se podrían implementar por completo en uno o más circuitos o elementos lógicos.

[0065] Las técnicas de la presente divulgación se pueden implementar en una amplia variedad de dispositivos o aparatos, incluyendo un teléfono inalámbrico, un circuito integrado (IC) o un conjunto de IC (por ejemplo, un conjunto de chips). En la presente divulgación se describen diversos componentes, módulos o unidades para destacar aspectos funcionales de dispositivos configurados para realizar las técnicas divulgadas, pero no se requiere necesariamente su realización mediante diferentes unidades de hardware. En su lugar, como se describe anteriormente, diversas unidades se pueden combinar en una unidad de hardware de códec o proporcionar mediante un grupo de unidades de hardware interoperativas, que incluya uno o más procesadores como se describe anteriormente, junto con software y/o firmware adecuados.

REIVINDICACIONES

1. Un procedimiento de procesamiento de gráficos, comprendiendo el procedimiento:

realizar, con una unidad de procesamiento de gráficos (GPU), un pase de fragmento de hardware en uno o más fragmentos de una trama, para identificar, para cada uno del uno o más fragmentos, primitivas que son visibles en el fragmento; renderizar, con la GPU, el uno o más fragmentos usando una o más operaciones de gráficos en las primitivas identificadas como visibles en el fragmento respectivo;
escribir, con la GPU, datos de color directamente en un búfer de color en la memoria del sistema de acuerdo con la una o más operaciones de gráficos; escribir, con la GPU, datos de profundidad en un búfer de profundidad en una memoria de gráficos de acuerdo con la una o más operaciones de gráficos, en el que la memoria de gráficos está en el mismo circuito integrado que la GPU; y
transferir, con la GPU, el búfer de profundidad en la memoria de gráficos a la memoria del sistema cuando se completa la renderización del uno o más fragmentos.

2. El procedimiento de la reivindicación 1, en el que renderizar la una o más porciones de la trama usando la una o más operaciones de gráficos comprende renderizar la una o más porciones de la trama usando operaciones de procesamiento de píxeles y una o más pruebas de profundidad, y en el que escribir los datos de color directamente en el búfer de color en la memoria del sistema de acuerdo con la una o más operaciones de gráficos comprende escribir los datos de color directamente en el búfer de color en la memoria del sistema de acuerdo con las operaciones de procesamiento de píxeles, y

en el que escribir los datos de profundidad en el búfer de profundidad en la memoria de gráficos de acuerdo con la una o más operaciones de gráficos comprende escribir los datos de profundidad en el búfer de profundidad en la memoria de gráficos de acuerdo con la una o más pruebas de profundidad.

3. El procedimiento de la reivindicación 1, en el que la memoria del sistema es memoria de acceso aleatorio dinámica (DRAM).

4. Un aparato configurado para el procesamiento de gráficos, comprendiendo el aparato:

una memoria de gráficos configurada para almacenar los datos de gráficos; y
una unidad de procesamiento de gráficos (GPU) en comunicación con la memoria del sistema y la memoria de gráficos, estando configurada la GPU para:

realizar un pase de fragmento de hardware en uno o más fragmentos de una trama, para identificar, para cada uno de los fragmentos, las primitivas que son visibles en el fragmento;
renderizar el uno o más fragmentos usando una o más operaciones de gráficos en la primitiva identificada como visible en el fragmento respectivo;
escribir datos de color directamente en un búfer de color en la memoria del sistema de acuerdo con la una o más operaciones de gráficos; escribir datos de profundidad en un búfer de profundidad en la memoria de gráficos de acuerdo con

la una o más operaciones de gráficos, en el que la memoria de gráficos está en el mismo circuito integrado que la GPU; y transferir el búfer de profundidad en la memoria de gráficos a la memoria del sistema cuando se completa la renderización del uno o más fragmentos.

5. El aparato de la reivindicación 4 que comprende además la memoria del sistema.

6. El aparato de la reivindicación 4, que comprende además una pantalla configurada para mostrar la trama.

7. El aparato de la reivindicación 4, en el que la GPU y la memoria de gráficos son parte de un ordenador personal, un ordenador de escritorio, un ordenador portátil, una tableta, una estación de trabajo de ordenador, una plataforma o consola de videojuegos, un teléfono móvil, un dispositivo de videojuegos portátil, un reproductor de música personal, un reproductor de vídeo, un dispositivo de visualización, un televisor, un descodificador de televisión o un servidor.

8. Un medio de almacenamiento no transitorio legible por ordenador que almacena instrucciones que, cuando se ejecutan, provocan que uno o más procesadores de un dispositivo de acuerdo con la reivindicación 5 realicen el procedimiento de cualquiera de las reivindicaciones 1-3.

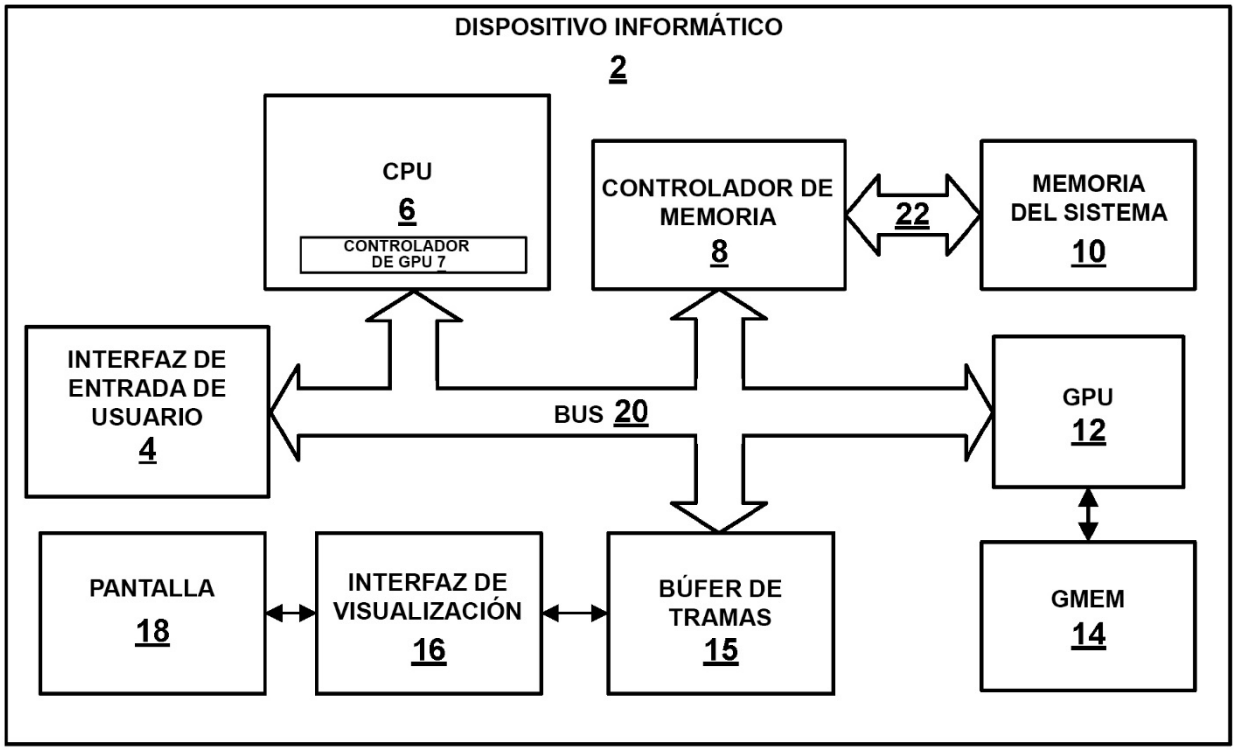
DISPOSITIVO INFORMÁTICO
2
CPU
6
CONTROLADOR DE GPU 7
CONTROLADOR DE MEMORIA
8
22
MEMORIA DEL SISTEMA
10
INTERFAZ DE ENTRADA DE USUARIO
4
BUS 20
GPU
12
PANTALLA
18
INTERFAZ DE VISUALIZACIÓN
16
BÚFER DE TRAMAS
15
GMEM
14

FIG. 1

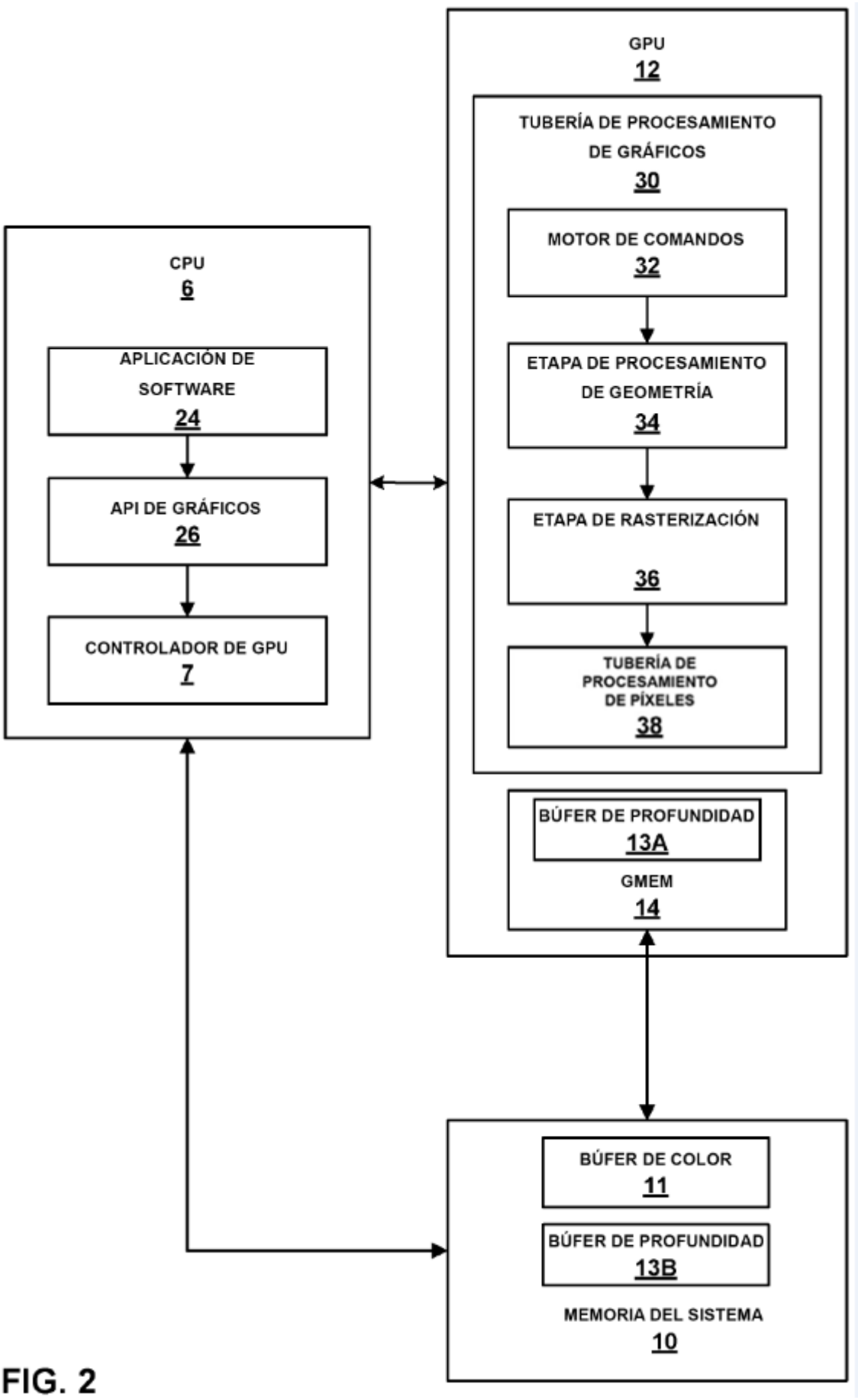
GPU
12
TUBERÍA DE PROCESAMIENTO DE GRÁFICOS
30
MOTOR DE COMANDOS
32
ETAPA DE PROCESAMIENTO DE GEOMETRÍA
34
ETAPA DE RASTERIZACIÓN
36
TUBERÍA DE PROCESAMIENTO DE PÍXELES
38
BÚFER DE PROFUNDIDAD
13A
GMEM
14
CPU
6
APLICACIÓN DE SOFTWARE
24
API DE GRÁFICOS
26
CONTROLADOR DE GPU
7
BÚFER DE COLOR
11
BÚFER DE PROFUNDIDAD
13B
MEMORIA DEL SISTEMA
10

FIG. 2

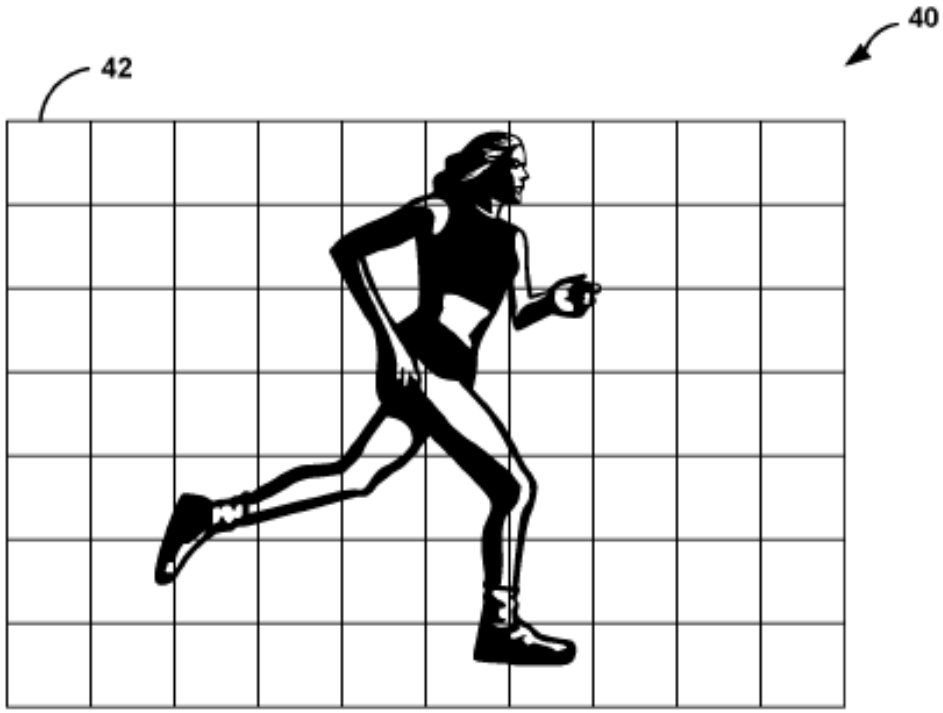

FIG. 3

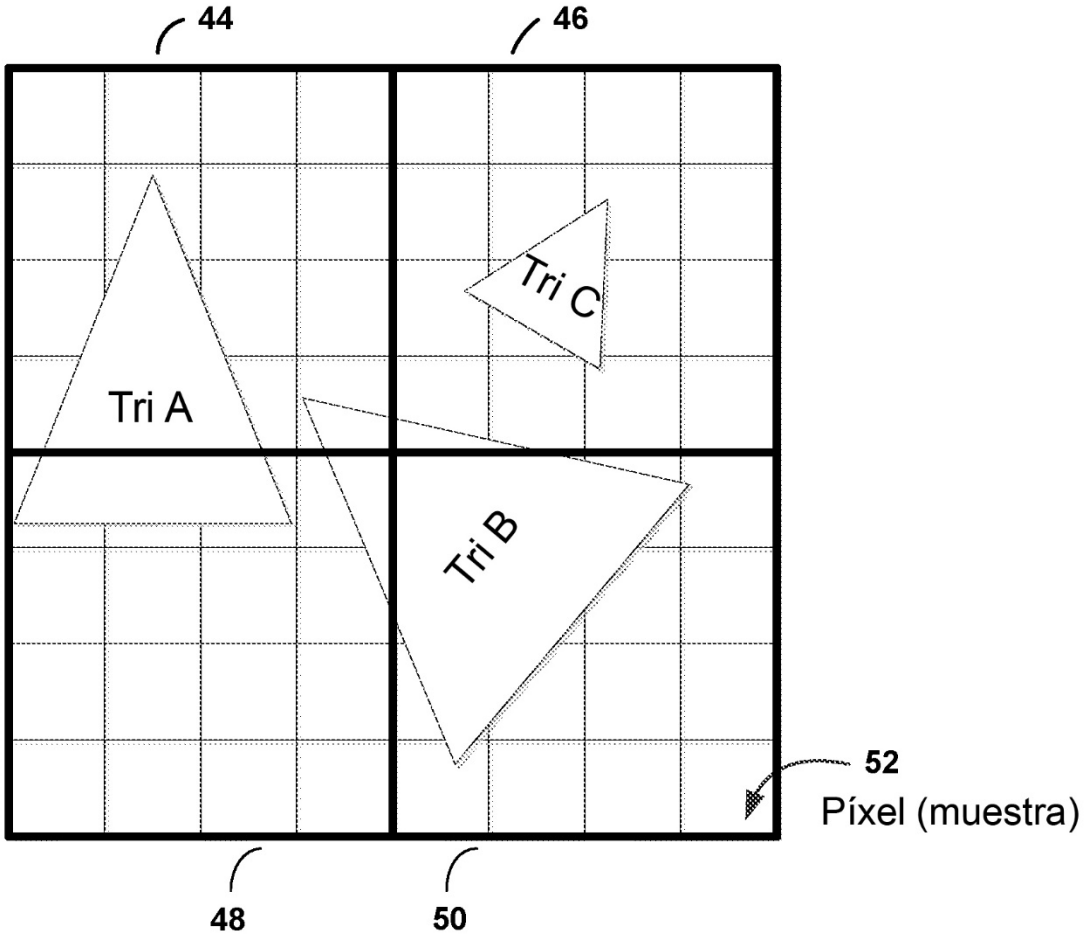

FIG. 4

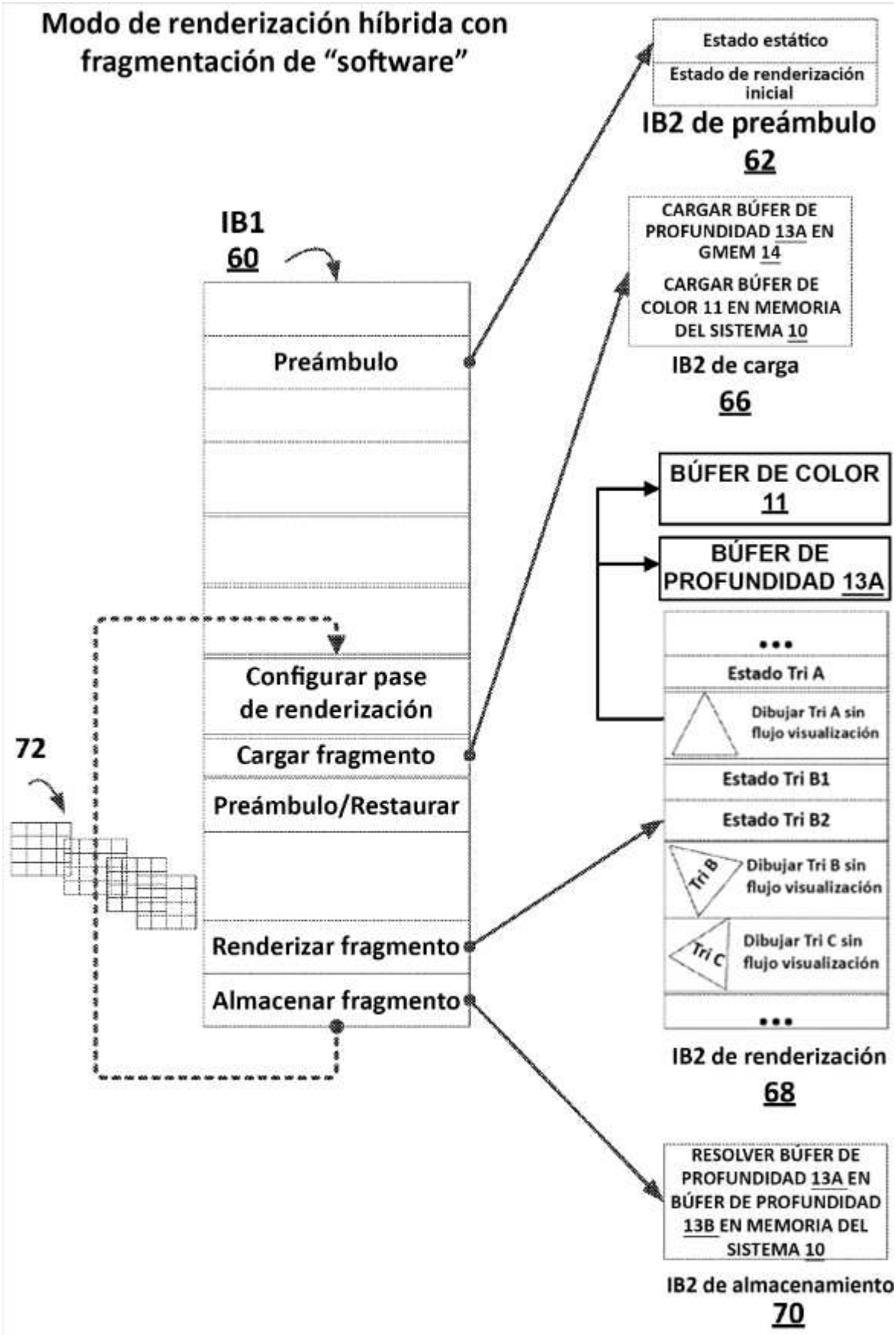
Modo de renderización híbrida con fragmentación de "software"
IB1
60
Preámbulo
Configurar pase de renderización
Cargar fragmento
Preámbulo/Restaurar
Renderizar fragmento
Almacenar fragmento
72
Estado estático
Estado de renderización inicial
IB2 de preámbulo
62
CARGAR BÚFER DE PROFUNDIDAD 13A EN GMEM 14
CARGAR BÚFER DE COLOR 11 EN MEMORIA DEL SISTEMA 10
IB2 de carga
66
BÚFER DE COLOR 11
BÚFER DE PROFUNDIDAD 13A
Estado Tri A
Dibujar Tri A sin flujo visualización
Estado Tri B1
Estado Tri B2
Tri B
Dibujar Tri B sin flujo visualización
Tri C
Dibujar Tri C sin flujo visualización
IB2 de renderización
68
RESOLVER BÚFER DE PROFUNDIDAD 13A EN BÚFER DE PROFUNDIDAD 13B EN MEMORIA DEL SISTEMA 10
IB2 de almacenamiento
70

FIG. 5

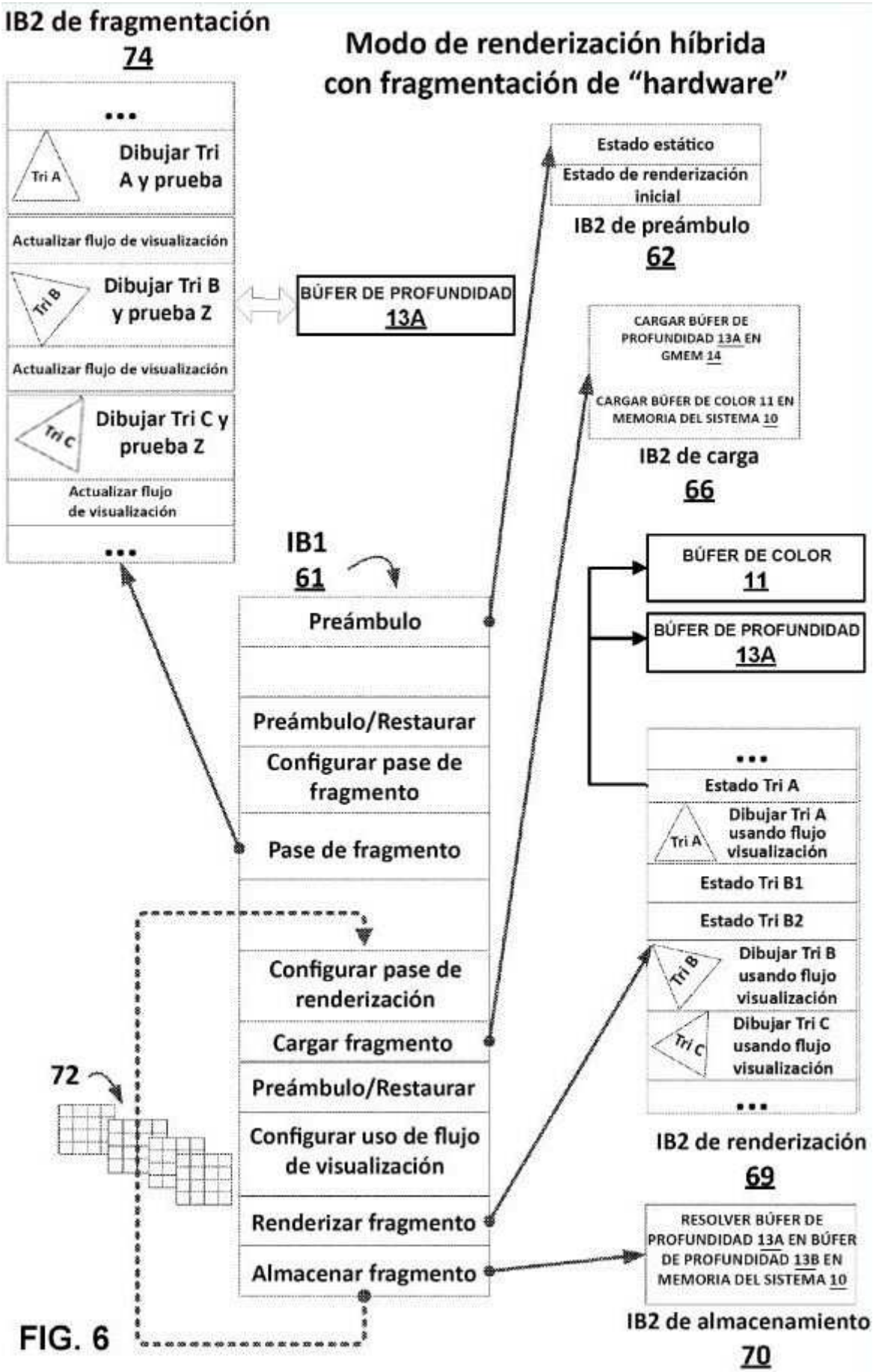
IB2 de fragmentación
74
Modo de renderización híbrida con fragmentación de "hardware"
...
Tri A
Dibujar Tri A y prueba
Actualizar flujo de visualización
Tri B
Dibujar Tri B y prueba Z
Actualizar flujo de visualización
Tri C
Dibujar Tri C y prueba Z
Actualizar flujo de visualización
...
BÚFER DE PROFUNDIDAD 13A
Estado estático
Estado de renderización inicial
IB2 de preámbulo
62
CARGAR BÚFER DE PROFUNDIDAD 13A EN GMEM 14
CARGAR BÚFER DE COLOR 11 EN MEMORIA DEL SISTEMA 10
IB2 de carga
66
IB1
61
Preámbulo
Preámbulo/Restaurar
Configurar pase de fragmento
Pase de fragmento
Configurar pase de renderización
Cargar fragmento
Preámbulo/Restaurar
Configurar uso de flujo de visualización
Renderizar fragmento
Almacenar fragmento
72
BÚFER DE COLOR 11
BÚFER DE PROFUNDIDAD 13A
...
Estado Tri A
Tri A
Dibujar Tri A usando flujo visualización
Estado Tri B1
Estado Tri B2
Tri B
Dibujar Tri B usando flujo visualización
Tri C
Dibujar Tri C usando flujo visualización
...
IB2 de renderización
69
RESOLVER BÚFER DE PROFUNDIDAD 13A EN BÚFER DE PROFUNDIDAD 13B EN MEMORIA DEL SISTEMA 10
IB2 de almacenamiento
70

FIG. 6

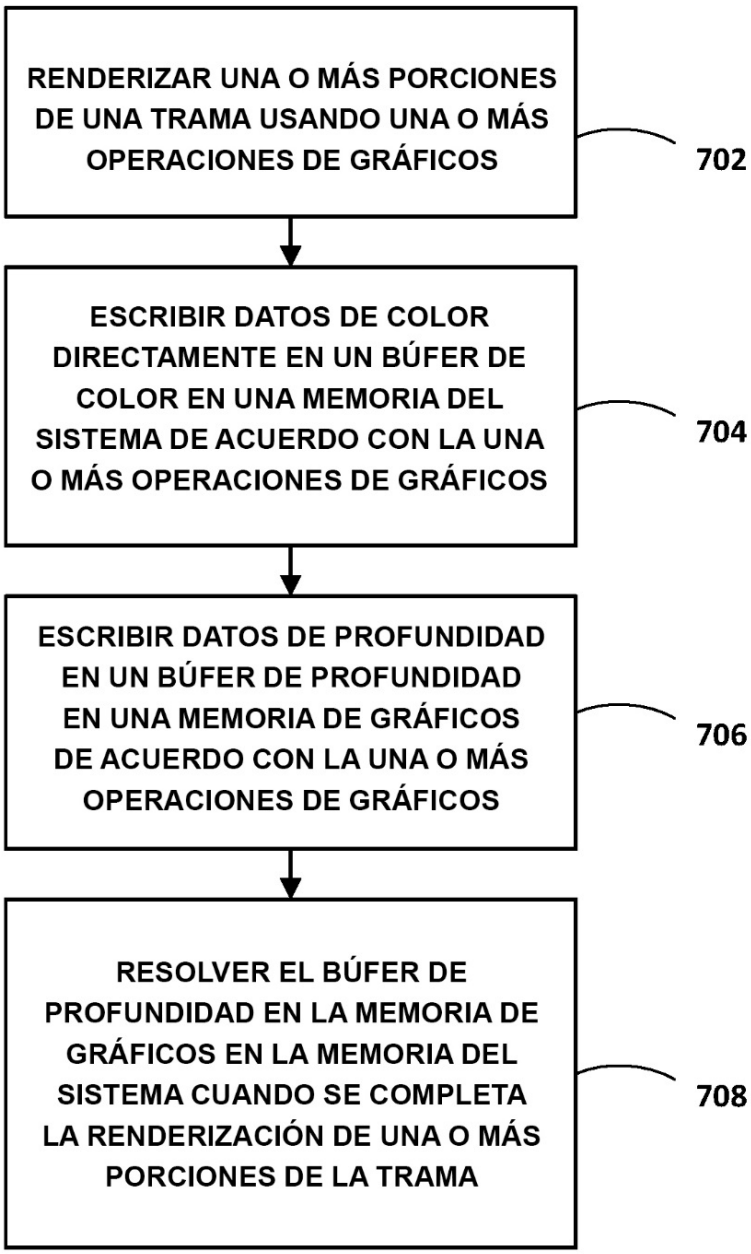
RENDERIZAR UNA O MÁS PORCIONES DE UNA TRAMA USANDO UNA O MÁS OPERACIONES DE GRÁFICOS
702
ESCRIBIR DATOS DE COLOR DIRECTAMENTE EN UN BÚFER DE COLOR EN UNA MEMORIA DEL SISTEMA DE ACUERDO CON LA UNA O MÁS OPERACIONES DE GRÁFICOS
704
ESCRIBIR DATOS DE PROFUNDIDAD EN UN BÚFER DE PROFUNDIDAD EN UNA MEMORIA DE GRÁFICOS DE ACUERDO CON LA UNA O MÁS OPERACIONES DE GRÁFICOS
706
RESOLVER EL BÚFER DE PROFUNDIDAD EN LA MEMORIA DE GRÁFICOS EN LA MEMORIA DEL SISTEMA CUANDO SE COMPLETA LA RENDERIZACIÓN DE UNA O MÁS PORCIONES DE LA TRAMA
708

FIG. 7