TFG Noita

Nicolás Rosa Caballero
Jonathan Andrade Gordillo

February 26, 2024

Índice

1 Introduction 4
2 Autómatas Celulares y simuladores de arena 5
2.1 Simuladores de arena 8
2.1.1 Introducción a la simulacion de partículas 8
2.1.2 Importancia y aplicación en diversos campos 9
2.1.3 Simuladores de arena como videojuegos 10
3 Programación paralela 12
3.1 Introduccion 12
3.2 Arquitectura 13
3.3 Usos 14
4 Plug-ins y lenguaje de script 16
5 Simulador en CPU 16
6 Simulador en GPU 16
7 Comparación y pruebas 16
8 Conclusiones y trabajo futuro 16
Bibliografía 16

No sé si esta es la mejor forma de crear una tabla de figuras pero es lo que encontré por ahora. Es posible crear una tabla usando una query (figure.where) y dándole el formato que nosotros queramos.

Índice de Figuras

Figura 1: Ejemplo de autómata celular 6
Figura 2: Ejemplo de autómata de contacto 6
Figura 3: Ejemplo autómata de Wolfram 7
Figura 4: Ejemplo de la hormiga de Langton 8
Figura 5: Ejemplo del Juego de la Vida 8
Figura 6: Imagen gameplay de Noita 10
Figura 7: Geforce 256, la primera tarjeta gráfica independiente 12
Figura 8: Comparativa arquitectura de un chip de CPU y de GPU 13
Figura 9: Streaming Multiprocessor 14
Figura 10: Jerarquía de ejecución 14

1 Introduction

2 Autómatas Celulares y simuladores de arena

Fuente del siguiente párrafo: https://en.wikipedia.org/wiki/Cellular_automaton No consigo averiguar de donde sacó wikipedia la información, las fuentes que cita son innacessibles o bien no contienen la información que necesito.

John von Neumann, matemático húngaroestadounidense, investigaba el problema de sistemas auto replicantes en la década de 1940. Su diseño se basaba en la idea de un robot que construye a otro. Esto le resultaba complejo debido a la dificultad de tener que proveer a dicho robot de un gran conjunto de piezas que pudiera usar para replicarse. Tras la publicación de un papel científico en 1948 titulado «The general and logical theory of automata», un colega suyo, Stanislaw Ulam, matemático polaco; sugirió usar un sistema discreto para crear un modelo reduccionista de auto replicación.

Más tarde, en 1950, Ulam y von Neumann crearon un método para calcular el movimiento de los líquidos. El concepto impulsor de dicho método consistía en considerar un líquido como un grupo de unidades discretas y calcular el movimiento de ellas basándose en los comportamientos de sus vecinas. Así nació el primer autómata celular No he encontrado uno anterior ni en wikipedia ni fuera de ella. No parece haber mucha información sobre los orígenes de los autómatas celualres

Bueno, al menos el siguiente párrafo tiene una fuente más sólida, aunque parte de la información del párrafo proviene de la wiki de Gardner

Los descubrimientos de von Neumann y Ulam propiciaron que otros matemáticos e investigadoras contribuyeran al desarrollo de los automátas celulares, sin embargo, no fue hasta 1970 que se popularizaron de cara al público general. En este año, Martin Gardner, divulgador de ciencia y matemáticas estadounidense; escribió un artículo [1] en la revista Scientific American sobre un automáta celular específico: El juego de la vida de Conway.

Los autómatas celulares [2] son un modelo matemático que consiste en una matriz n-dimensional de celdas. Cada celda puede estar en un estado de un conjunto de estados finitos. Estos estados cambian en función de los estados de sus celdas vecinas cada unidad discreta de tiempo llamada generación [3]. Cada celda tiene 8 vecinos, laterales y esquinas inclusive. Al procesar un automáta celular, una celda y sus vecinos se consideran como una sola, esto es llamado "vencidad de Moore"[4]. Si solo se consideran los vecinos sin las esquinas entonces es llamado "vencidad de Neumann"[4]. Por último, para definir un autómata celular necesitamos una "regla" que aplicar a cada celda, esta transformará el sistema de celdas en el siguiente estado en cada generación.

A continuación se mostrará un ejemplo de automáta celular para comprender mejor su concepto y propiedades. Este sistema consiste en una matriz bidimensional infinita cuyas celdas pueden tener cuatro estados posibles que serán llamados `agua`, `lava`, `roca` y `vacío`. Se presupone un sistema de coordenadas (X,Y). La coordenada X representa la posición horizontal de la celda de izquiera a derecha. La coordenada Y representa la posición vertical de la celda de abajo a arriba. Para simplificar la explicación de este sistema se definirá la operación mover. Se entiende como moverse a la transformación matemática por el cual se toman dos coordenadas, `inicio` y `final`, en la que la que el estado de la celda final se sobreescribe con el de la celda inicio. La celda inicio pasa a tener como estado `vacío`. (no sé si debería escribir esto en notación matemática, tampoco estoy muy seguro de como sería dicha notación). Una celda en estado agua puede moverse hacia debajo si la celda inferior contiene estado `vacío`. Si la celda inferior contiene el

estado `lava`, la celda inferio pasará a ser estado `roca` y la celda actual pasará de estado `agua` a `vacío`. Una celda en estado `roca` no sufre alteraciones de ningún tipo. Una celda en estado `lava` se comporta similar a una celda con estado `agua`. Puede moverse hacia abajo si la celda inferior está vacía y se transforma en roca si la celda inferior es `agua`. Para representar el estado de forma visual, se asigna el color rojo a la celda en estado lava, azul al estado agua, negro al estado roca y blanco al estado vacío:

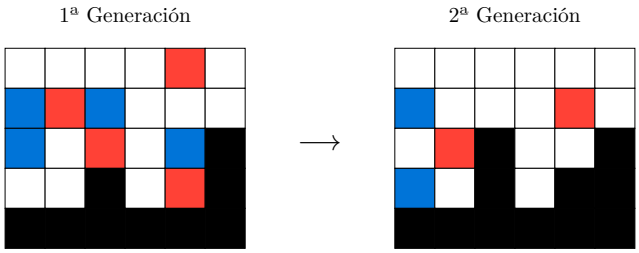

Figura 1: Ejemplo de autómata celular

Los automátas celulares han interesado a la comunidad científica y matemática desde su establecimiento por von Neumann. Debido a esto existen diversos autómatas celulares desarrolladores por distintos investigadores. A continuación se muestran una serie de autómatas celulares relevantes.

- Autómata de Contacto [4]

Un autómata de contacto puede considerarse como uno de los modelos más simples para la propagación de una enfermedad infecciosa. Imagina una cuadrícula cuadrada de celdas. Supongamos que todas las celdas son «blancas» (o «no infectadas») excepto un número finito de celdas «negras» (o «infectadas»). Para una celda dada, definimos los vecinos como las ocho celdas inmediatamente adyacentes y la celda misma. Una versión determinista de un proceso de contacto funciona en pasos discretos de tiempo de la siguiente manera: una celda negra permanece negra, una celda blanca que es vecina de una celda negra se vuelve negra. Este sistema puede denominarse un proceso de contacto determinista. Si comenzamos con una sola celda negra, entonces el número de celdas negras aumenta en 1, 9, 25, etc. La forma general sigue siendo cuadrática. Se usa en la investigación de propagación de fenómenos epidemiológicos.

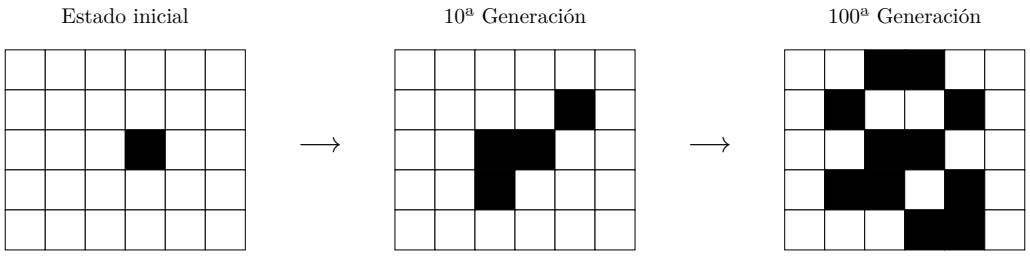

Figura 2: Ejemplo de autómata de contacto

- Autómatas de Wolfram [4]

Los Autómatas de Wolfram, propuestos por el físico y matemático Stephen Wolfram, son un conjunto de reglas para autómatas celulares unidimensionales de estados binarios. Cada regla define cómo evolucionan las células en función de su estado y el estado de sus vecinos. Cada

generación en un automáta de wolfram es una fila más qu ese añade a las anteriores. Numeradas del 0 al 255, estas reglas proporcionan un marco teórico para explorar la complejidad emergente y la computación universal en sistemas celulares simples.

A continuación se muestra la regla 30 [5] de Wolfram así como el autómata derivado de ejecutar 15 iteraciones de este:

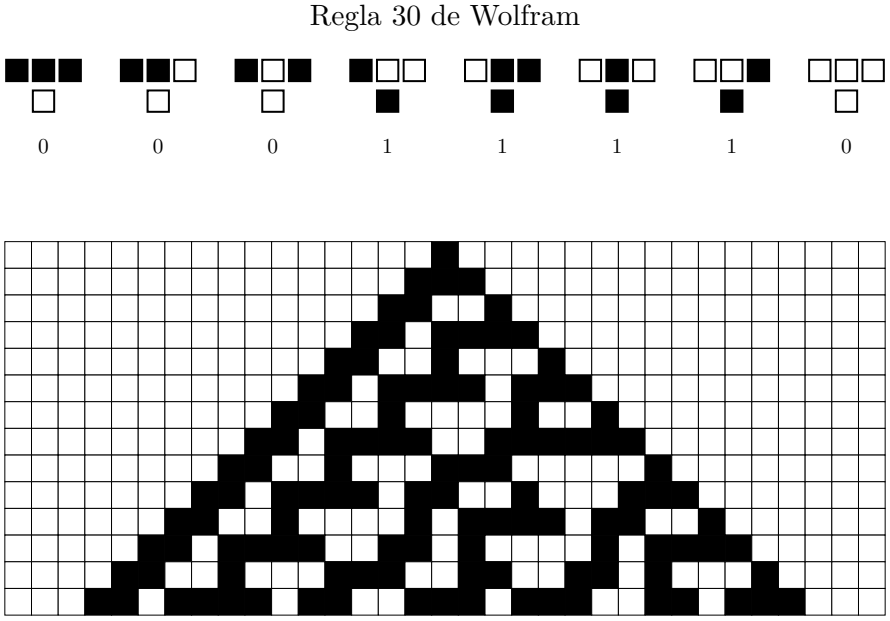

Figura 3: Ejemplo autómata de Wolfram

- Autómata de Greenberg-Hastings [4]

Los autómatas de Greenberg-Hastings son bidimensionales y están compuestos de células que pueden estar en 3 estados: reposo, excitado y refractario. La evolución de las células se basa en reglas locales que determinan la activación y desactivación de las células en función de su estado actual y el estado de sus vecinos. Debido a esteo este modelo ha sido utilizado para simular patrones de propagación de la actividad eléctrica en tejidos cardiacos. Destacando por su capacidad para modelar fenómenos de propagación y ondas, este autómata sigue reglas específicas para la activación y desactivación de células, lo que lo convierte en una herramienta valiosa en la investigación biomédica.

TODO: Añadir ejemplo de Greenberg-Hastings (no hay ninguno en el libro ni en internet, no esoty muy seguro de como crearlo pero el concepto de este autómata es intereante, ¿Podría dejarlo sin ejeplo o tendría que quitarlo por ser el único sin ejepmlo?).

- Hormiga de Langton [4]

La hormiga de Langton es un autómata bidimensional de 18 estados. Cada celda puede ser blanca o negra, además de que puede contener o no a la hormiga (solo hay una). La hormiga está orientada a hacia una de las 4 direcciones cardinales y solo se mueve una sola vez de acorde a las siguientes reglas: Si le hormiga está en una celda negra, cambia su orientación 90 grados a la derecha. Si está en una celda blanca, cambia su orientación 90 grados a la izquierda. Finalmente, cada vez que la hormiga abandona una celda, esta cambia de color. Este sistema se vuelve periódico tras algo más de mil iteraciones, creando una estructura con forma de camino

con periodo 104. A continuación se mostrará un ejemplo de las primeras iteraciones. La hormiga se representará con el color rojo si está en una celda negra y azul si está en una celda blanca. Se asume que en el estado inicial, la hormiga está orientada hacia arriba y en una celda blanca.

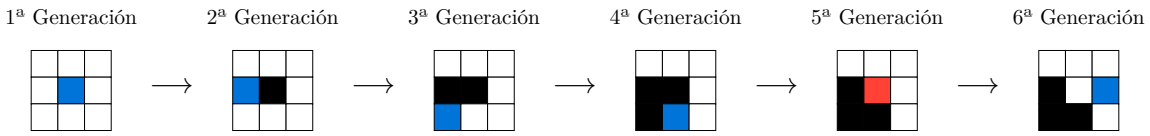

Figura 4: Ejemplo de la hormiga de Langton

- Juego de la vida

El Juego de la Vida, propuesto por el matemático John Conway, es un autómata celular bidimensional que se desarrolla en una cuadrícula infinita de células, cada una de las cuales puede estar en uno de dos estados: viva o muerta. La evolución del juego está determinada por reglas simples basadas en el número de vecinos vivos alrededor de cada célula. Este modelo ha demostrado ser notable debido a su capacidad para generar patrones complejos y estructuras dinámicas a partir de reglas simples de transición de estados. El Juego de la Vida ha sido ampliamente estudiado en el campo de la teoría de la complejidad y ha sido utilizado como un modelo para explorar la autoorganización y la emergencia de la complejidad en sistemas dinámicos.

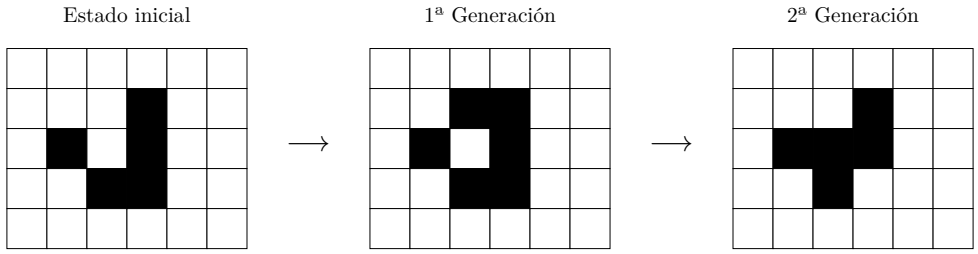

Figura 5: Ejemplo del Juego de la Vida

Con el tiempo, esos sistema llamaron la atención de un público menos científico debido a su cualidad recreacional, pues no era necesario entender los fundamentos matemáticos de estos para poder disfrutar sus interacciones. De esta forma, y con un enfoque orientado a lo lúdico, es como surgieron los simuladores de arena.

2.1 Simuladores de arena

En este capítulo, se hablará de manera resumida acerca de qué son los simuladores de partículas de manera general así como sus múltiples aplicaciones a diferentes sectores. Mas tarde, se presentan una serie de antecedentes que se han tomado de base para el desarrollo del proyecto.

2.1.1 Introducción a la simulacion de partículas

Los simuladores de partículas son un subgénero destacado tanto en el sector de los videojuegos como en el de los autómatas celulares [6], haciendose uso de ellos para herramientas y experiencias capaces de representar interacciones dinámicas presentes en el mundo real.

Los simuladores de partículas cumplen las características básicas que permiten considerarlos autómatas celulares, ya que: el «mapa» de la simulación está formado por un conjunto de celdas

dentro de un número finito de dimensiones, cada una de estas celdas se encuentra, en cada paso discreto de la simulación, en un estado concreto dentro de un número finito de estados en los que puede encontrarse y el estado de cada celda es determinado mediante interacciones locales, es decir, está determinado por condiciones relacionadas con sus celdas adyacentes.

Estos abarcan una gran diversidad de enfoques destinados a proporcionar diferentes funcionalidades y experiencias dependiendo del enfoque del proyecto. Algunos de los tipos mas reseñables son:

- Simulador de fluidos [7]: Enfoque dirigido a la replicación del comportamiento de sustancias como pueden ser el agua y diferentes clases de fluidos, lo que implica la simulación de fenómenos como flujos y olas y de propiedades físico químicas como la viscosidad y densidad del fluido.
- Simulador de efectos [8]: Categoría que apunta a la simulación detallada de efectos de partículas,como pueden ser las chispas, humo, polvo, o cualquier elemento minúsculo que contribuya a la creación y composición de elementos visuales para la generación de ambientes inmersivos.
- Simuladores de arena [9]: Conjunto de simuladores cuyo objetivo es la replicación exacta de interacciones entre elementos físicos como granos de arena u otros materiales granulares. Esto implica la modelización de interacciones entre particulas individuales, como pueden ser colisiones o desplazamientos, buscando recrear interacciones reales de terrenos.

2.1.2 Importancia y aplicación en diversos campos

La versatilidad de los simuladores de partículas ha llevado a su aplicación y adopción en una vasta gama de disciplinas mas allá de los videojuegos, debido a la facilitación que ofrecen de trabajar con fenómenos físicos de manera digital.

Algunos — aunque no todos — de los campos que han hecho uso de simuladores de particulas, así como una pequeña explicación acerca de su aplicación son:

- Física y ciencia de materiales

Los simuladores de partículas han permitido la experimentación virtual de propiedades físicas de diferentes elementos, como pueden ser la dinámica de partículas en sólidos o la simulación de materiales.

- Ingeniería y construcción

Se emplean simuladores con el objetivo de prever y comprender el funcionamiento de diferentes estructuras y materiales en el ámbito de construcción antes de su edificación, lo que permite predecir elementos básicos como la distribución de fuerzas y tensiones así como el comportamiento ante distintos fenómenos como pueden ser terremotos

- Medicina y biología

Los simuladores de partículas en el ámbito de la medicina permite modelar comportamientos biológicos así como, por ejemplo, imitar la interacción y propagación de sustancias en fluidos corporales, ayudando al desarrollo de tratamientos médicos.

2.1.3 Simuladores de arena como videojuegos

Dentro de la industria de los videojuegos, se han utilizado simuladores de partículas con diferentes fines, como pueden ser: proporcionarle libertad al jugador, mejorar la calidad visual o aportarle variabilidad al diseño y jugabilidad del propio videojuego.

Este proyecto toma como principal referencia a «Noita», un videojuego indie roguelike que utiliza la simulación de partículas como núcleo principal de su jugabilidad. En «Noita», cada píxel en pantalla representa un material y está simulado físicamente según reglas físicas y químicas específicas de ese material. Esto permite que los diferentes materiales físicos, líquidos y gaseosos se comporten de manera realista de acuerdo a sus propiedades. El jugador tiene la capacidad de provocar reacciones en este entorno, como destruirlo o hacer que interactúen entre sí.

Figura 6: Imagen gameplay de Noita

Por supuesto, Noita no es el primer videojuego que hace uso de los simuladores de partículas. A continuación, voy a enumerar algunos de los títulos, tanto videojuegos como sandbox, más notables de simuladores de los cuales hemos tomado inspiración durante el desarrollo.

- Falling Sand Game [10]

Probablemente el primer videojuego comercial de este amplio subgénero. A diferencia de Noita, este videojuego busca proporcionarle al jugador la capacidad de experimentar con diferentes partículas físicas así como fluidos y gases, ofreciendo la posibilidad de ver como interaccionan tanto en un apartado físico como químicas. Este videojuego establecería una base que luego tomarían videojuegos más adelante.

- Powder Toy [11]

Actualmente el sandbox basado en partículas más completo y complejo del mercado. Este no solo proporciona interacciones ya existentes en sus predecesores, como Falling Sand Game, sino que añade otros elementos físicos de gran complejidad como pueden ser: temperatura, presión, gravedad, fricción, conductividad, densidad, viento etc.

- Sandspiel [12]

Este proyecto utiliza la misma base de sus sucesores, proporcionando al jugador libertad de hacer interaccionar partículas a su gusto. Además, añade elementos presentes en Powder Toy como el viento, aunque la escala de este proyecto es mas limitada que la de proyectos anteriores. De Sandspiel, nace otro proyecto llamado Sandspiel Club [13], el cual utiliza como base Sandspiel, pero, en esta versión, el creador porporciona a cualquier usuario de este proyecto la capacidad de crear partículas propias mediante un sistema de scripting visual haciendo uso de la librería

blockly [14] de Google. Además, similar a otros títulos menos relevantes como Powder Game (No confundir con Powder Toy), es posible guardar el estado de la simulación y compartirla con otros usuarios. A cambio de esta funcionalidad, en Sandspiel Club no es posible hacer uso del viento, elemento sí presente en Sandspiel.

3 Programación paralela

En este capítulo, se desea introducir al lector el funcionamiento y programación de GPUs, hablando acerca de su arquitectura, es decir, cómo están fabricadas,su origen y sus usos.

3.1 Introduccion

Las Unidades de Procesamiento Gráfico, comúnmente conocidas como GPUs (por sus siglas en inglés, Graphics Processing Units), son hoy en dia un componente clave de la informática moderna presente en todos los ordenadores, destinado principalmente al procesamiento gráfico.

El origen del concepto de lo que sería una GPU hoy en día, se remonta a 1980, con un proyecto en el cual con el objetivo de generar imagenes 3D para el estudio de la estructura protéica, en la Universidad de Carolina del Norte (UNC) desarrollaron un sistema llamado Pixel Planes, el cual tuvo la innovadora aplicación de asignar un procesador por pixel mostrado, lo que significaba que muchas partes de las imágenes en la pantalla se generaban simultáneamente, mejorando enormemente la velocidad en la cual se generaban los gráficos.[15] Este trabajo continuó hasta 1997 donde se desarrolló la última iteración del proyecto.

En 1982, Nippon Electric Company introdujo el primer controlador gráfico a gran escala, el chipset gráfico NEC µPD7220, lanzado originalmente para la PC-98. Este controlador introdujo aceleración hardware para dibujar lineas, círculos y otras figuras geométricas en una pantalla de píxeles.

Hasta ese momento, se utilizaban terminales caligráficas para mostrar dibujos e imagenes mediante trazos vectoriales en lugar de píxeles individuales, ya que la mayoría de microcomputadores carecían de la potencia necesaria para hacer uso de pantallas rasterizadas, que utilizan una cuadrícula de píxeles para representar imágenes en pantalla.

Figura 7: Geforce 256, la primera tarjeta gráfica independiente

El término GPU no se popularizaría hasta el año 1999, donde NVIDIA sacó y comercializó la GeForce 256 [16] como la primera unidad de procesamiento gráfico completamente integrada que ofrecía transformaciónes de vértices, iluminación, configuración y recorte de triángulos y motores de renderizado en un único procesador de chip integrado. El chip contaba con numerosas características avanzadas, incluyendo cuatro pipelines de renderizado independientes. Esto permitía que la GPU alcanzara una tasa de relleno (cantidad de píxeles que una GPU escribe sobre la memoria de vídeo para crear el fotograma sobre esta) de 480 Megapíxeles por segundo. La salida de video era VGA. Además, el chip incluía mezcla de alfa por hardware y cumplía con los estándares de televisión de alta definición (HDTV). Lo que de verdad diferenció a la

GeForce 256 de la competencia fue la integración de iluminación que lo diferenció de modelos pasados y de la competencia, que hacían uso de la CPU para ejecutar este tipo de funciones, mejorando el rendimiento y abaratando costes para el consumidor.

3.2 Arquitectura

El propósito principal de una GPU es priorizar el procesamiento rápido de instrucciones simples mediante una estrategia de «divide y vencerás». Esto implica dividir la tarea en componentes más pequeños que pueden ser ejecutados por los numerosos núcleos presentes en una GPU. A diferencia de los núcleos de la CPU, los núcleos de la GPU son más lentos y menos versátiles, con un conjunto de instrucciones más limitado. Sin embargo, la compensación radica en el mayor número de núcleos disponibles para ejecutar instrucciones en paralelo. [17]

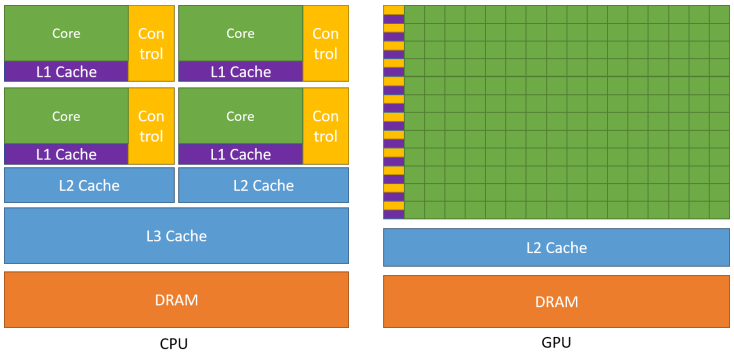

Figura 8: Comparativa arquitectura de un chip de CPU y de GPU

Se hará uso de CUDA para explicar el funcionamiento de la ejecución de programas en GPU.[18]

La GPU está optimizada para realizar tareas de manera paralela, mientras que la CPU está diseñada para ejecutar tareas de forma secuencial. Esta diferencia se refleja en la distribución del espacio dentro del chip. En una GPU, la mayor parte del espacio se dedica a tener muchos núcleos pequeños, mientras que en una CPU, la estructura está más orientada hacia una jerarquía de memoria, con múltiples niveles de caché y núcleos más grandes, potentes y capaces, pero limitados en términos de paralelización.

La arquitectura de la GPU se compone de múltiples SM (Streaming Multiprocessors), que son procesadores de propósito general con baja velocidad de reloj y una caché pequeña. La tarea principal de un SM es controlar la ejecución de múltiples bloques de hilos, liberándolos una vez que han terminado y ejecutando nuevos bloques para reemplazar los finalizados. Cada SM está formado por núcleos de ejecución, una jerarquía de memoria que incluye: caché L1, memoria compartida, caché constante y caché de texturas, un programador de tareas y un número considerable de registros.

Figura 9: Streaming Multiprocessor

Desde el punto de vista de software, la ejecución de programas se divide en kernels, grids de bloques, bloques de hilos e hilos.

Se le conoce como kernel a aquellas funciones que están destinadas a ser ejecutadas en la GPU desde la CPU o Host. Estas funciones se subdividen en un grid de una, dos o hasta 3 dimensiones de bloques de hilos mediante los que se ejecutará el kernel. El número de bloques totales que se crean viene dictado por el volumen de datos a procesar. Es totalmente necesario que estos bloques de hilos puedan ser ejcutados de manera totalmente independiente y sin dependencias entre ellos, ya que a partir de aquí el programador de tareas es el que decide que y cuando se ejecuta. Los hilos dentro del bloque pueden cooperar compartiendo datos y sincronizando su ejecución para coordinar los accesos a datos, teniendo una memoria compartida a nivel de bloque. El número de hilos dentro de un bloque esta limitado por el tamaño del SM, ya que todos los hilos necesitan residir en el mismo SM para poder compartir recursos de memoria.

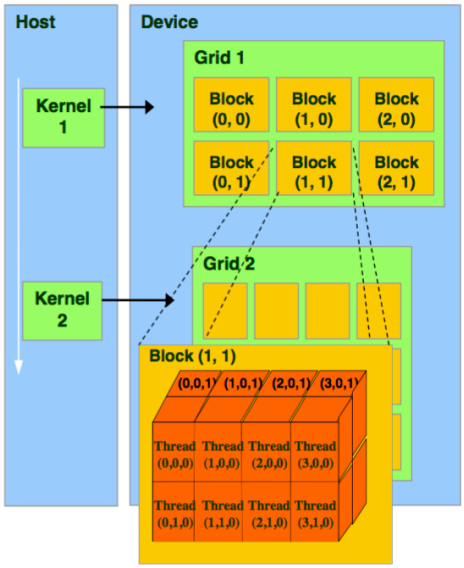

Figura 10: Jerarquía de ejecución

3.3 Usos

La función inicial y principal de las GPU era la de procesar la imagen que iba a ser mostrada al usuario. Para facilitar esto, se crearon los shaders, pequeños programas gráficos destinados

a ejecutarse en la GPU como parte de la pipeline principal de renderizado, cuya base es transformar inputs en outputs que finalmente formarán la imagen a mostrar.

El pipeline principal de renderizado esta compuesto de manera general por las siguientes etapas: El shader de vértices recibe el input directamente de la CPU, y su trabajo principal es calcular cual será la posición final de cada vertice en la escena. Le sigue la etapa de teselación, la cual es opcional, y cuya labor es la de transformar las superficies formadas por los vértices en primitivas mas pequeñas, aumentando el nivel de detalle de la malla a mostrar. El shader de geometría, también opcional, genera una o más primitivas como output a partir de una primitiva recibida como input. Le siguen las etapas de post-procesado, donde se descarta el area de volúmenes fuera del volumen visible, el ensamblado de primitivas, que ordena la geometría de la escena en una secuencia de primitivas simples (lineas, puntos o triangulos), la rasterización, donde se transforma las primitivas recibidas en fragmentos a traves de los cuales podemos determinar el estado final de cada pixel, y por último el shader de fragmentos, que determina el color final de cada «fragmento» o pixel de la escena. Opcionalmente existen ciertos tests que se ejcutan al terminar el shader de fragmentos.[19]

Durante muchos años, los programadores utilizaban los shaders modificables , es decir, el shader de vertices,de geometría y de fragmentos para realizar ciertas operaciones que no implicaban de manera directa al pipeline gráfico, para así aprovechar la potencia de cómputo que ofrece la GPU. Por este motivo, se introdujeron los compute shaders, shaders que pueden ser ejecutados fuera del pipeline gráfico.[20]

La manera de trabajar con ellos funciona de manera similar a lo explicado anteriormente, cada compute shader ejecuta una funcion que se subdivide en work groups de una, dos o tres dimensiones que a su vez contienen un número de hilos que puede estar subdividido también en hasta 3 dimensiones.

4 Plug-ins y lenguaje de script

5 Simulador en CPU

6 Simulador en GPU

7 Comparación y pruebas

8 Conclusiones y trabajo futuro

Bibliografía

[1] «MATHEMATICAL GAMES. The fantastic combinations of John Conway\'s new solitaire game 'life'». [En línea]. Disponible en: https://www.ibiblio.org/lifepatterns/october1970.html

[2] Thomas M. Li, *Cellular Automata*. 2010.

[3] Andrew Adamatzky, *Game of Life Cellular Automata*, vol. 1. 2010.

[4] Karl-Peter Hadeler y Johannes Müller, *Cellular Automata':' Analysis and Applications*, vol. 1. 2017.

[5] «Rule 30 of Wolfram Automata». [En línea]. Disponible en: https://mathworld.wolfram.com/Rule30.html

[6] Wikipedia, «Autómata Celular». [En línea]. Disponible en: https://es.wikipedia.org/wiki/Aut%C3%B3mata_celular

[7] Wikipedia, «Computational fluid dynamics». [En línea]. Disponible en: https://en.wikipedia.org/wiki/Computational_fluid_dynamics

[8] Wikipedia, «Particle System». [En línea]. Disponible en: https://en.wikipedia.org/wiki/Particle_system

[9] Wikipedia, «Sand Simulator». [En línea]. Disponible en: https://en.wikipedia.org/wiki/Falling-sand_game

[10] Web Archive, «Falling Sand Game». [En línea]. Disponible en: https://web.archive.org/web/20090423105358/http://fallingsandgame.com/overview/index.html

[11] Powder Toy, «Powder Toy». [En línea]. Disponible en: https://powdertoy.co.uk/

[12] Max Bittker, «Sandspiel». [En línea]. Disponible en: https://maxbittker.com/making-sandspiel

[13] Max Bittker, «Sandspiel Club». [En línea]. Disponible en: https://sandspiel.club/

[14] Google, «Blockly». [En línea]. Disponible en: https://developers.google.com/blockly

[15] Jon Peddie, *The History of the GPU - Steps to Invention*, vol. 1. 2023.

[16] «Nvidias Geforce 256». [En línea]. Disponible en: https://www.computer.org/publications/tech-news/chasing-pixels/nvidias-geforce-256

[17] Jon Peddie, *Professional CUDA C Programming*, vol. 1. 2014. [En línea]. Disponible en: https://www.cs.utexas.edu/~rossbach/cs380p/papers/cuda-programming.pdf

[18] «Cuda C++ Programming Guide». [En línea]. Disponible en: https://docs.nvidia.com/cuda/cuda-c-programming-guide/

[19] «Rendering Pipeline Overview». [En línea]. Disponible en: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

[20] «Compute Shaders Introduction». [En línea]. Disponible en: https://learnopengl.com/Guest-Articles/2022/Compute-Shaders/Introduction