Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Laboratorio de Bio-Robótica

Detección y Reconocimiento de objetos utilizando técnicas de visión en GPU

T E S I S

que para obtener el grado de:

Maestro en Ciencias de la Computación

presenta:

Jaime Alan Márquez Montes

directores de tesis:

Dr. Jesús Savage Carmona

Dr. Jose David Flores Peñaloza

Ciudad de México, D. F. Junio, 2015

Detección y Reconocimiento de objetos utilizando técnicas de visión en GPU

por

Jaime Alan Márquez Montes

Tesis presentada para obtener el grado de

Maestro en Ciencias de la Computación

en el

Posgrado en Ciencia e Ingeniería de la Computación

Universidad Nacional Autónoma de México

Ciudad de México, D. F.. Junio, 2015

Dedicatorias.....

.....

.....

AGRADECIMIENTOS

Agradecimientos.....
.....
.....

TABLA DE CONTENIDO

CAPÍTULO 1

INTRODUCCIÓN

1.1. Contexto

Mas allá de la compleja circuiteria y mecánica que conforman la parte física, los robots móviles rigen su comportamiento en base a el software, dependiendo de cuanta interacción tenga el robot con su entorno, podría llegar a ser mas complejo que todo el hardware que lo conforma. Una parte importante de este software, es la forma en la que el robot puede obtener datos e interpretarlos.

El sistema de visión humano, es al que más recurre para obtener información de su entorno. Por ello no es de extrañarse que la visión computacional, en la robótica, tenga una participación muy importante,por que estas maquinas empiezan a ser utilizadas, para tareas que antes solo los humanos realizaban. Entonces las deben realizar de una manera adecuada y en tiempo.

El tiempo es preciado en cualquier rama y esta no es la excepción, la forma de ganar tiempo que se a venido sesgando por hardware es el paralelismo, y no solo hablo de procesadores multinúcleo, las tarjetas gráficas se pueden programar para realizar tareas de propósito general.

1.2. Problema a resolver

Los algoritmos que se utilizan, en visión computacional muchas veces son muy confiables, pero consumen mucho tiempo de procesador, por esto se ha tratado de hacer mas eficientes estos algoritmos, pero provoca que la confiabilidad de estos disminuya. El tiempo en el cual se adquieren y procesa la información, es crucial en la actividad de un robot, de esto depende que decisión tomara.

Con lo anteriormente dicho, lo importante es el tiempo en que procesemos los datos, para tomar una decisión, pero igual de importante es que la información obtenida sea congruente.

En muchos casos, el software que funciona en paralelo es mas rápido que el secuencial, podemos ver que los algoritmos que se manejan en visión computacional son siempre secuenciales. Otro punto importante son los recursos, como procesador y memoria de la computadora del robot, siempre estarán siendo demandados por otros módulos del robot.

1.3. Hipótesis

1.3.1. Alcance

1.4. Estructura de la tesis

CAPÍTULO 2

ALGORITMOS DE EXTRACCIÓN Y DESCRIPCIÓN DE CARACTERÍSTICAS

Las características de los objetos son cualidades que nos servirán para identificarlos dentro de una imagen donde pueden existir objetos similares. Para poder discernir de los objetos que estén en la imagen nos basamos en las características encontradas, que serán encapsuladas en un descriptor. Un descriptor de un objeto es la representación, de una manera reducida, de todas las características que se pueden obtener de toda la información del objeto, esto facilitara la comparación entre los diferentes objetos que existan en una imagen. Para extraer las características existen diferentes formas, dependerá de que algoritmo se utilice. Y para generar un descriptor, es la misma situación, dependerá del algoritmo. En este capítulo se explicarán un algoritmo para extraer características y generar su descriptor.

2.1. Scale-Invariant Feature Transform SIFT

El algoritmo de SIFT, propuesto por Lowe en [1], provee un método robusto para la extracción de puntos característicos que se utilizan para generar el descriptor. Los puntos que se encuentran son invariantes a diferentes transformaciones como traslación, escalamiento y rotación. Han mostrado tener un amplio rango de tolerancia a transformaciones afines, adición de ruido y cambios de iluminación. A continuación se describirán los pasos del algoritmo para la generación del conjunto de puntos característicos:

2.1.1. Detección de puntos extremos en el Espacio-Escala

Se realiza una búsqueda en las imágenes en todo el espacio escala, para localizar puntos extremos se debe identificar su ubicación y escala, para volver a encontrarlos no importando la vista o tamaño del mismo objeto.

El espacio escala es un conjunto de imágenes, que se forman a partir de suavizar la imagen original a diferentes niveles de detalles, los cuales son definidos por un parámetro σ. Está representado por la función $L(x, y; \sigma)$ la cual se forma por la convolución con $G(x, y; \sigma)$ y la imagen original $I(x, y)$:

$$L(x, y; \sigma) = G(x, y; \sigma) * I(x, y)$$

Donde $*$ es el operador convolución en x y y, y

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

Para la detección de puntos extremos estables se aplicará el espacio escala, usando diferencias

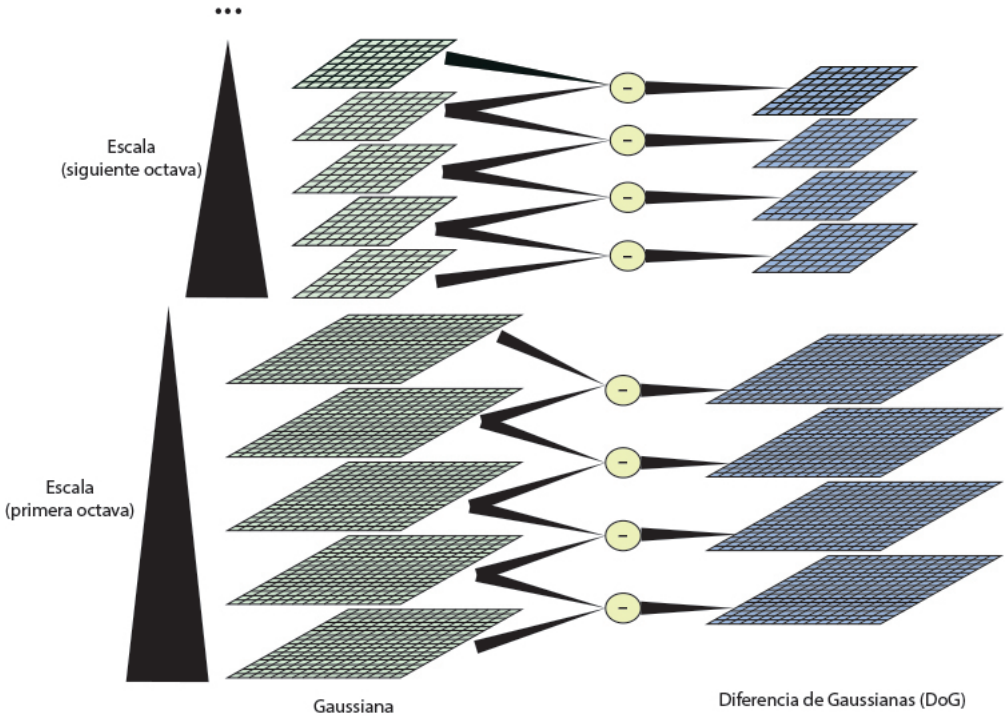

Figura 2-1: Espacio Escala de Diferencia de Gaussianas

de gaussianas convolucionadas con una imagen, en lugar de solo un filtro gaussiano, $D(x, y; \sigma)$ que podremos calcular por la diferencia de dos escalas cercanas separadas por un factor k multiplicativo:

$$D(x, y; \sigma) = (G(x, y; k\sigma) - G(x, y; \sigma)) * I(x, y)$$

$$= L(x, y; k\sigma) - L(x, y; \sigma)$$

La diferencia de gaussianas es una aproximación muy cercana a el laplaciano de gaussiana (LoG) normalizado en escala, $\sigma^2\nabla^2 G$. La normalización hecha con el factor σ^2 es necesaria para poder asegurar que el algoritmo sera invariante a los cambios en tamaño. La relación entre D y $\sigma^2\nabla^2 G$ es una ecuación en derivadas parciales:

$$\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G$$

Podemos ver que $\nabla^2 G$ se puede calcular con una aproximación de diferencias finitas de $\frac{\partial G}{\partial \sigma}$, usando diferencias de escalas próximas de $k\sigma$ y σ:

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, k\sigma)}{k\sigma - \sigma}$$

y por lo tanto,

$$G(x, y, k\sigma) - G(x, y, k\sigma) \approx (k - 1)\sigma^2\nabla^2 G$$

En la figura 2-1 se puede ver la construcción de $D(x, y, \sigma)$. La imagen inicial se convoluciona con diferentes mascaras gaussianas, para producir imágenes separadas por un factor constante k en el espacio escala. Se divide cada octava del espacio escala entre un numero entero, s, de intervalos entonces $k = 2^{\frac{1}{s}}$. Se producen $s + 3$ imágenes emborronadas en la pila, por octava. Para extraer las

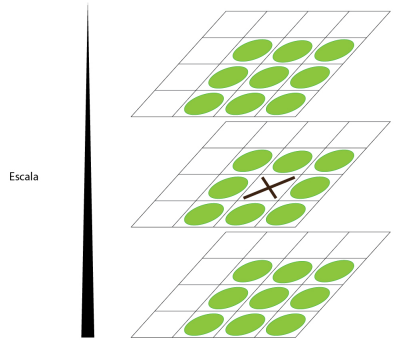

Figura 2-2: Espacio Escala de Diferencia de Gaussianas

ubicaciones máximas y mínimas (puntos extremos) en $D(x, y, \sigma)$, cada punto es comparado con sus ocho vecinos en la misma imagen y con sus otros dieciocho vecinos de escala, nueve en la imagen de arriba y nueve en la imagen de abajo (Figura 2-2). Solo se selecciona el punto si es el más grande o el más pequeño de entre todos sus vecinos.

2.1.2. Localización de puntos característicos

Una vez que se seleccionaron los puntos extremos, se aplica una medida de estabilidad sobre todos para descartar aquellos que no sean adecuados, para obtener puntos característicos de forma precisa. Existen dos casos donde los puntos extremos anteriormente seleccionados tendrían que ser eliminados:

1. El punto tiene un contraste muy bajo.

2. El punto está localizado sobre un borde.

Para eliminar los puntos del caso uno, primero debemos obtener la serie de Taylor del espacio escala $D(x, y, \sigma)$:

$$D(X) = D + \frac{\partial D}{\partial X}^T X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X$$

donde la D y su derivada son evaluadas en el punto $X = (x, y, \sigma)^T$ cuando se deriva esta función respecto a X y se iguala a cero podemos encontrar los valores extremos:

$$\hat{X} = -\frac{\partial^2 D}{\partial X^2}^{-1} \frac{\partial D}{\partial X}$$

La función que evaluara al punto extremo sera, $D(\hat{X})$, la cual rechazara al punto si es de muy bajo contraste, la cual se obtiene de sustituir $\hat{X}$ en $D(X)$:

$$D(\hat{X}) = D + \frac{1}{2} \frac{\partial D}{\partial X}^T \hat{X}$$

En el trabajo de Lowe [1], se puede ver que encontraron experimentalmente que cualquier valor extremo menor de 0.03 es descartado:

$$|D(\hat{X})| < 0.03$$

Para el segundo caso, se utiliza una matriz Hessiana de 2×2, H, la cual se calcula en la escala y lugar del punto extremo:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Los valores propios de H son proporcionales a las curvaturas de D. Se toma prestado el criterio que se usa para la detección de esquinas usando el algoritmo de Harris [2], se puede evitar el calculo de los valores propios ya que solo nos interesa su relación. Sea α el valor propio de mayor magnitud y β el de menor. Entonces podemos calcular la suma de los valores propios de la diagonal de H y su producto por medio del determinante:

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta$$

Sea r la razón de la magnitud que existe entre α y β, $\alpha = r\beta$. Entonces:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

el cual solo depende de la razón de los valores propios y no de los valores individuales. El valor de $\frac{(r+1)^2}{r}$, es mas pequeño cuando los valores propios son iguales e incrementa con r.Entonces para cerciorar que la razón de las curvas principales es menor que cierto umbral, r, solo se necesita:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

En la publicación de Lowe [1] se encontró un valor experimental para $r = 10$, que elimina los puntos extremos que tengan la razón entre las dos curvas mayor que 10.

2.1.3. Asignación de orientación

Por medio de la asignación de una orientación a cada punto característico, basado en propiedades locales de la imagen, el descriptor que encontremos sera invariante a la rotación. La ubicación en el espacio escala del punto característico, es usada para seleccionar la imagen suavizada por una mascara gaussiana, L, esto provocara que sea invariante a la escala. Para cada muestra de la

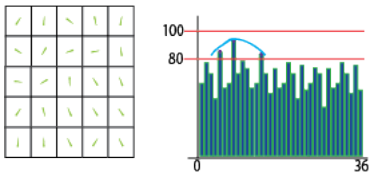

Figura 2-3: Histograma de Orientación TEMPORAL

imagen, $L(x, y)$, la magnitud del gradiente, $m(x, y)$, y la orientación ,$\theta(x, y)$, son precalculadas por medio de diferencias de gaussianas:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

Se formara un histograma de orientaciones que tendrá la orientación de los gradientes calculados en una región, al rededor del punto característico, el tamaño de esta muestra dependerá de la ubicación en el espacio escala en la que se encuentre el punto característico. El histograma de orientaciones tendrá 36 divisiones cubriendo los 360 grados. Para cada muestra agregada se ponderada por la magnitud de su gradiente y por una mascara circular gaussiana ponderada con σ, que es 1.5 veces que de la ubicación del espacio escala donde reside el punto característico. Los picos en el histograma de orientación corresponden a las direcciones dominantes de los gradientes locales. Se encuentra el pico mas grande y cualquier otro pico que se encuentre en el rango de $100\,\% - 80\,\%$, del pico más grande, se utiliza para hacer que el punto característico tenga una orientación. Para ubicaciones con varios picos de magnitudes similares, se generaran puntos característicos con la misma ubicación y escala pero con diferentes orientaciones. Solo el $15\,\%$ de los puntos se les asignan múltiples orientaciones, pero aun así esto contribuye mucho al momento de emparejar. Finalmente se obtiene una parábola usando como puntos tres picos cercanos entre si, para interpolar la posición del pico con mas precisión.

2.1.4. Descriptor de puntos característicos

Hasta este momento tenemos una colección de puntos característicos, los cuales están formados por una ubicación, una escala y una orientación. Ahora debemos formar un descriptor que sea lo suficientemente distintivo. Para esto tenemos que tomar una muestra de la imagen, al rededor del

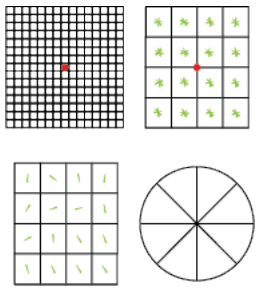

Figura 2-4: Descriptor TEMPORAL

punto característico de 16×16 pixeles y se dividirá en una región de 4×4. Se generará un histograma de orientación de los gradientes de cada región, a diferencia del histograma de orientación explicado anteriormente, el histograma solo tiene 8 divisiones con las cuales se cubrirán los 360 grados, igualmente se usara una ponderación gaussiana para la asignación de la magnitud al histograma. Al final el descriptor de cada punto característico estará formado por un vector, que tiene las ocho orientaciones de los 4×4 histogramas. Por lo tanto el tamaño del vector sera de $4 \times 4 \times 8 = 128$ elementos.

CAPÍTULO 3

COMPUTO EN GPU'S

Hasta hace 12 años la velocidad a la que crecían cada generación de procesadores era increíble, los programas eran tan rápidos como cada nueva generación de procesadores. Este crecimiento entre cada generación se detuvo, el problema es el consumo de energía y la disipación de calor, no permiten aumentar la frecuencia del reloj del procesador y el nivel de actividades por ciclo, en una sola unidad de procesamiento (CPU). Todos los productores de procesadores migraron a un nuevo modelo, los procesadores multinúcleo incrementaron el poder de procesamiento.

Este cambio en los procesadores tuvo un gran impacto a los programadores, la mayoría de las aplicaciones son escritas de forma secuencial, por que la ejecución de estas son comprensibles paso a paso, mediante el código. Pero un programa secuencial ejecutándose en un solo núcleo del procesador, no sera más rápido. Entonces los programadores ya no pueden agregar cualidades y capacidades a sus programas.

Llega el momento de cambiar, si se desea que la calidad de los programas siga escalando con cada generación de procesadores, se deben crear programas que trabajen con múltiples hilos, cooperando todos para completar un trabajo mas rápido. Existen dos corrientes principales en cuanto a los procesadores multi-núcleo, el primero, es donde se pretende mantener la velocidad de los programas secuenciales, mientras se mueven entre múltiples núcleos; la segunda, se centra mas en la ejecución de aplicaciones en paralelo, tiene un gran numero de núcleos pequeños que va creciendo con cada generación. Es esta rama en la que entran las unidades de procesamiento gráfico o por sus siglas en ingles GPU.[3]

3.1. GP-GPUs Nvidia

"Las GPU han evolucionado al punto que muchas aplicaciones del mundo real se están implementando fácilmente en ellas y se ejecutan muchísimo más rápido que en sistemas con múltiples núcleos. Las arquitecturas de computación del futuro serán sistemas híbridos con GPU de núcleos paralelos trabajando en tándem con CPU de múltiples núcleos".[4]

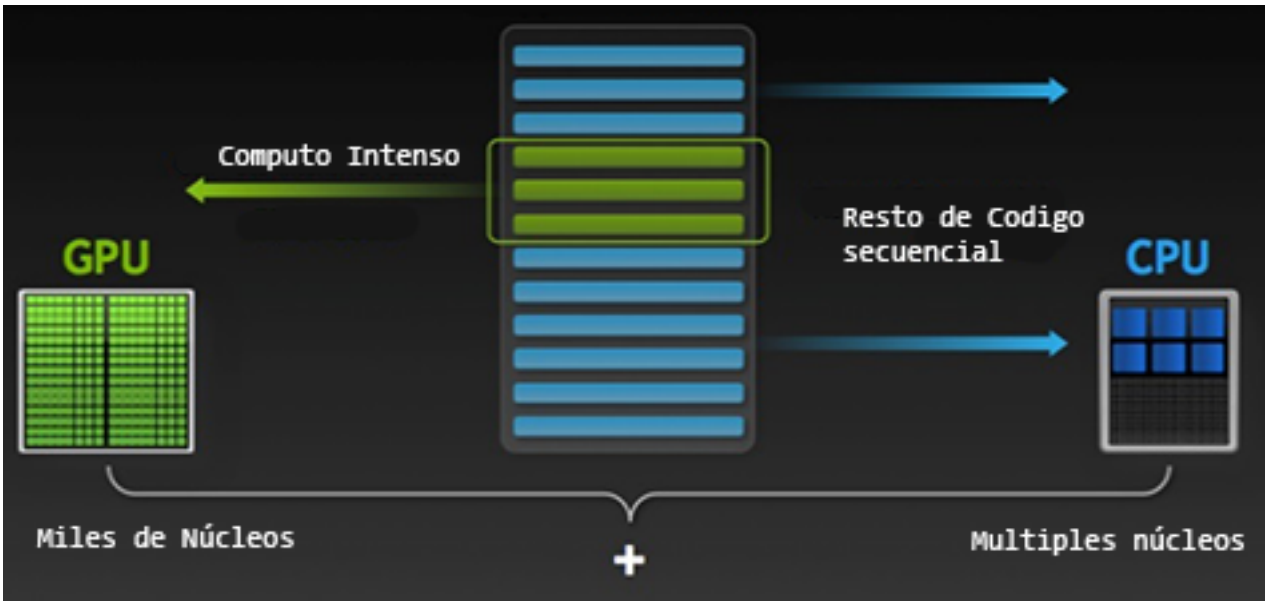

Figura 3-1: Sistema Híbrido

3.1.1. Breve Historia

La necesidad de mejores gráficos para los vídeo juegos, provocaron un gran avance en el hardware que se diseñaría. Desde principios de 1980 hasta finales de 1990 las tarjetas dedicadas a gráficos, no eran más que pipelines fijos que despliegan las formas geométricas calculadas por el CPU, por medio del hardware de acceso directo a memoria, por sus siglas en ingles DMA, esto les daba un funcionamiento fijo y apenas se podía configurar, con principalmente dos API, OpenGL de *Silicon Graphics* y Direct3D de *Microsoft*. Un ejemplo de estas tarjetas gráficas es, a la que se le acuño el nombre de GPU, la GeForce 256[5] lanzada al mercado en 1999, aporta una capacidad visual sin precedentes, capaz de realizar las funciones de transformación, iluminación, organización y rendering, con la capacidad de procesar 15 millones de triángulos por segundo y un rendimiento de 480 millones de píxeles por segundo. Su motor de rendering 256 bits muestra una mejora en cuanto a la complejidad visual.

Toda esta tecnología tan revolucionaria llamo la atención de otros profesionales, que se integraron a el trabajo de los artistas y desarrolladores de vídeo juegos, utilizando el gran rendimiento de punto flotante que tenían los GPU para otros objetivos. De esta forma surge el movimiento de la GPU para fines generales(GP-GPU).

Pero en ese momento, la GP-GPU era muy difícil de manipular, solo aquellos que tenían amplios conocimientos en lenguajes de programación de gráficos, desarrollaban para esta plataforma. Pero aun que memorizaras el API entera se enfrentaba un reto, donde los cálculos para resolver problemas generales debían ser representados por triángulos o polígonos.

Fue hasta 2001, en la Universidad de Stanford un equipo, liderado por Ian Buck, que se propuso ver el GPU como un *procesador de flujos*. Este equipo desarrollaría *Brook* [6], un lenguaje de programación diseñado para ser igual a la sintaxis de C, con algunas características adicionales. El lenguaje se desarrolla con el objetivo de minimizar el complejo trabajo de análisis, que se requería para generar aplicaciones paralelas. Introducirían conceptos como los flujos(streams), kernels y los operadores de reducción. Todo esto le dio un gran impulso a los GPU como procesadores de propósitos generales, ya que el lenguaje era mas fácil de manejar, ya que era de más alto nivel, y lo mas importante los programas escritos en *Brook* eran hasta 7 veces mas rápido que códigos similares existentes.

La compañía NVIDIA se dio cuenta que tenia un hardware muy poderoso en las manos, pero debía complementarlo con herramientas de hardware y software intuitivas, con ello le hicieron la invitación a Ian Buck para colaborar con ellos, el objetivo sería ejecutar C a la perfección en una GPU. NVIDIA alcanza este objetivo en 2006 con el lanzamiento de CUDA, la cual seria la primera solución para las GP-GPU, y aunado a esta solución, lanza la GeForce 8800 la cual fue diseñada, para ser usada en cómputo de propósito general, y su arquitectura fue pensada en la de CUDA.

3.2. CUDA

Compute Unified Device Architecture (CUDA) es una plataforma para computo paralelo y un modelo de programación, que NVIDIA lanzo en noviembre de 2006, permitiendo obtener aumentos en los rendimientos del computo, esto es gracias a la ayuda que la unidad de procesamiento de gráficos, le proporciona al CPU.

Los dispositivos CUDA aceleran la ejecución de los programas que tienen una gran cantidad de datos a procesar, ya que la arquitectura de esta plataforma, de la cual se hablara adelante, es

como un procesador tradicional como el que las computadoras tienen, solo que tienen la cualidad de que los procesadores son masivamente paralelos equipados con una gran cantidad de unidades aritméticas. En las cuales se ejecutara la misma instrucción en todas, respecto a la taxonomía de Flynn, la categoría seria de *una instrucción, multiples datos* (SIMD).

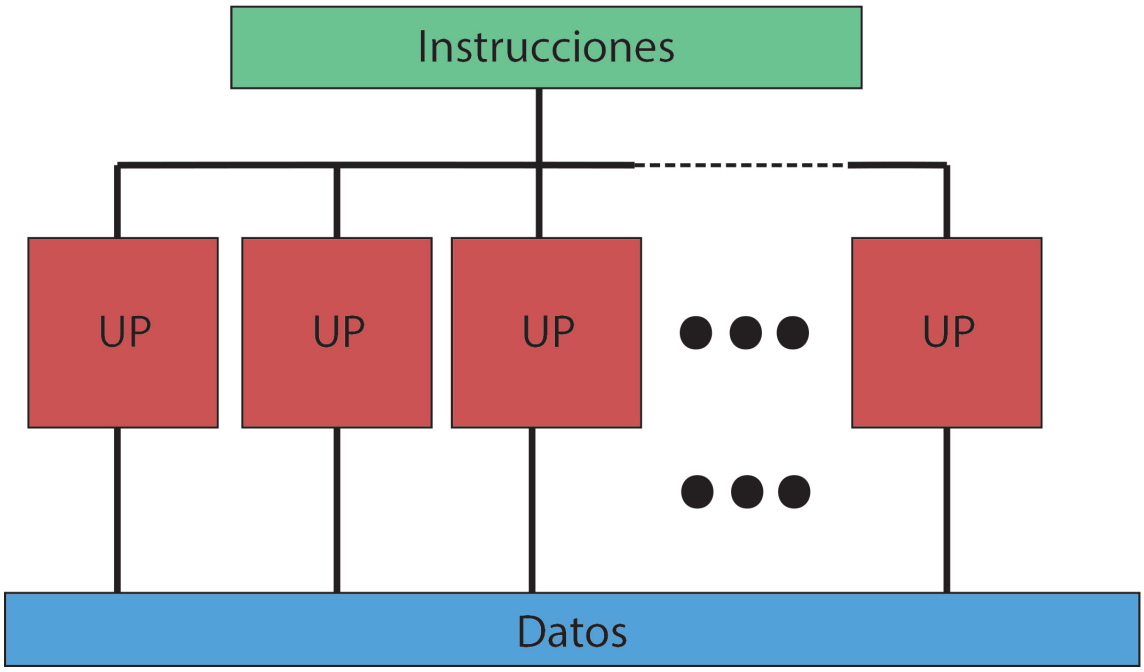

Figura 3-2: SIMD

Respecto al modelo de programación para desarrollar los programas para las GPU, es gracias a una extensión del lenguaje C, conocida como CUDA C. Existen alternativas a esta extensión, se pueden utilizar lenguajes como FORTRAN, Python, .NET combinando CUDA con Microsoft's F# o alguna API como OpenCL u OpenACC[7].

3.2.1. Arquitecturas

La arquitectura de CUDA fue diseñada, para que la GPU pudiera ser utilizada en aplicaciones de propósito general. En la cual se tiene un arreglo de procesadores con múltiples unidades aritmético lógica, por sus siglas en ingles ALU, las cuales para alcanzar este objetivo, fueron diseñadas para poder realizar operaciones de punto flotante, cumpliendo los requisitos del Instituto de Ingeniería Eléctrica y Electrónica (IEEE). Aparte de esto las ALU debían tener acceso a diferentes tipos de memoria, como la compartida entre unidades y la memoria de la tarjeta gráfica.

Estas ALU tan particulares, en la arquitectura de CUDA las conoceremos como *CUDA cores*,

conforman gran parte de los Streaming Multiprocessor (SM). Los SM son procesadores que tienen la tarea de ejecutar los hilos concurrentemente, aparte de los CUDA cores tienen, están formados por una memoria cache(shared memory), registros y algunas unidades de funciones especiales.

Fermi

Los GPU basados en la arquitectura Fermi, están formados por 512 CUDA cores. Los CUDA cores ejecutan operaciones de punto flotantes o enteras por ciclo de reloj, y por cada uno de los hilos. Los 512 CUDA cores están organizados en 16 SM de 32 cores cada uno. El GPU tiene seis particiones de memoria de 64-bits, capacidad para leer 384-bits de la memoria simultáneamente y con una capacidad de hasta 6GB de memoria DRAM categoría DDR5. El sistema de conexión entre el GPU y el CPU es vía PCI-Express. La forma en que se hace la programación de el trabajo a realizar en cada bloque es asignado por un modulo llamado *GigaThread*, este pasa las tareas a cada SM para que el haga la asignación de trabajo a cada hilo.

3.2.2. Modelo de Programación

3.2.3. Rendimiento

CAPÍTULO 4

SIFT EN GPU

4.1. Diseño

4.2. Implementación

4.3. Resultados

CAPÍTULO 5

CONCLUSIONES Y TRABAJO A FUTURO

BIBLIOGRAFÍA

[1] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[2] C. Harris and M. Stephens. A Combined Corner and Edge Detector. *Procedings of the Alvey Vision Conference 1988*, pages 147–151, 1988.

[3] David Kirk and Wen-mei Hwu. *Programming massively parallel processors : a hands-on approach*. Morgan Kaufmann, San Francisco, CA, USA, 2010.

[4] Jack Dongarral. Lo que la gente dice[en linea]. `http://www.nvidia.com.mx/object/cuda_home_new_la.html`, Accedido: [22/08/2015].

[5] NVIDIA. The world's first gpu geforce 256 [en linea]. `http://www.nvidia.es/page/geforce256.html`, Accedido: [25/08/2015].

[6] I Buck. Brook : A Streaming Programming Language. *Communication*, (October):1–12, 2001.

[7] NVIDIA. Language solutions [en linea]. `https://developer.nvidia.com/language-solutions`, Accedido: [25/08/2015].

ÍNDICE DE FIGURAS