Estimación de la Función de Iluminación No Uniforme para la Restauración de Imágenes en un Procesador Gráfico

Kenia Picos Espinoza[a], Víctor Hugo Díaz Ramírez[a], Juan José Tapia Armenta[a]

[a] *Instituto Politécnico Nacional - CITEDI*
Ave. del Parque 1310, Mesa de Otay, Tijuana, B.C., 22510, México
kpicos@citedi.mx vdiazr@ipn.mx, jtapiaa@ipn.mx

2013 Published by $\mathcal{DIFU}_{100}ci@$ http://www2.uaz.edu.mx/web/www/publicaciones
Selection and peer-review under responsibility of the Organizing Committee of the CICOMP-2013, www.cicomp.org

Resumen

Un sistema en tiempo real es propuesto para restaurar imágenes degradadas por iluminación y ruido mediante los parámetros estadísticos de la escena de entrada. Se diseña un sistema de restauración de imágenes mediante el diseño de filtros, implementado en una unidad de procesamiento gráfico. El sistema es capaz de restaurar la escena de entrada que consiste en una imagen afectada por degradaciones en iluminación no uniforme y ruido aditivo con distribución normal. El filtro diseñado es capaz de adaptarse a las condiciones de la escena de entrada para estimar los parámetros estadísticos de la función de la fuente de iluminación. Los resultados obtenidos en simulación en CPU/GPU con el sistema propuesto se presentan en términos de desempeño de restauración de imágenes, complejidad computacional y tolerancia a iluminación de la escena de entrada y ruido aditivo.

Palabras clave: Cómputo paralelo, Iluminación no uniforme, Procesador gráfico, Procesamiento de imágenes.

1. Introducción

En años recientes, el reconocimiento de objetos ha recibido mayor atención debido a la necesidad de desempeñar sistemas de procesamiento de imágenes que mejoren las actividades de vigilancia, navegación en vehículos, y seguimiento de objetos, entre otros. El reconocimiento de objetos consiste básicamente de dos características: (a) identificación del objeto dentro de una escena observada, y (b) estimación de las coordenadas exactas del objeto. El reconocimiento de objetos basado en filtros por correlación ha probado ser una técnica efectiva en la detección y localización de objetos [1]. En este enfoque, la detección se lleva a cabo buscando los mejores niveles de la correlación en el sistema de salida, brindando una localización mediante la estimación de la posición de un objeto dentro de una escena de entrada. Sin embargo, estos sistemas son muy sensibles a distorsiones y ruido [1]. La respuesta de los sensores (por ejemplo, cámaras o el ojo

humano) es proporcional a la radiación incidente sobre ellos, donde la constante de proporcionalidad depende de la geometría del sensor [2]. Los algoritmos de iluminación global calculan la cantidad de radiación estimada que incide sobre un objeto. La elección de un algoritmo adecuado es importante debido a que este afectará la exactitud de la restauración resultante de la imagen. Dichos algoritmos pueden presentar distintos enfoques tales como: (a) orientados por pixel u orientados por escena, (b) superficie difusa o superficie especular, (c) integración determinística o integración por Monte Carlo, entre otras [2]. Las distorsiones a la respuesta de los sensores pueden afectar la escena de entrada por degradaciones debido a iluminación no uniforme y ruido aditivo. Dado a que se asume que estas degradaciones son procesos no estacionarios, es necesario mejorar el proceso de filtrado con el fin de adaptar sus parámetros para estimar ruido aditivo y condiciones de iluminación mediante los parámetros estadísticos de la escena de entrada.

En el área de procesamiento de imágenes, la necesidad de aplicar los algoritmos en un sistema de tiempo real se muestra cada vez más evidente. Un sistema de tiempo real, requiere precisión en sus cálculos y que se produzcan dentro de un periodo de tiempo especificado.

Se ha vuelto indispensable implementar algoritmos de procesamiento de imágenes con un enfoque en paralelo, debido a la creciente popularidad del diseño de los procesadores modernos con las estructuras jerárquicas de memoria. Se ha demostrado que esta técnica de procesamiento de imágenes puede ser ejecutada para plataformas con estructuras de memoria compartida, produciendo ventajas de rendimiento significativas [3]. El trabajo actual se centra en mejorar el rendimiento de la estimación de ruido e iluminación, particularmente la porción no estacionaria de la aplicación GPU.

En este proyecto, los resultados son obtenidos en simulación por computadora empleando un procesador gráfico (GPU) para el sistema propuesto. La aplicación de un sistema CPU/GPU es utilizado para obtener un desempeño alto y eficiente en la implementación, obteniendo un cumplimiento de las restricciones temporales para el tiempo real. Estos resultados se presentan y discuten en términos de desempeño de restauración de imágenes complejidad computacional, tolerancia al ruido aditivo y tolerancia a iluminación no uniforme.

El trabajo se presenta con la siguiente organización. En la sección 2 se presenta el estudio del modelo de iluminación no uniforme. En la sección 3 se presenta el análisis propuesto para la estimación de los coeficientes de normalización de la función de iluminación. En la sección 4 se presenta la metodología realizada para su implementación. Por último, en las secciones 5 y 6, se presentan los resultados obtenidos, y las conclusiones, respectivamente.

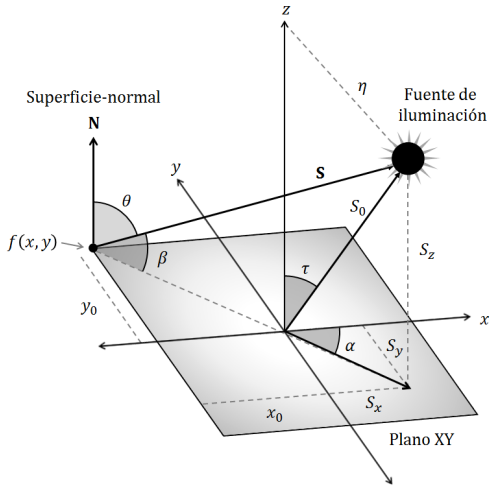


Figura 1. Geometría del modelo de iluminación en un plano $x - y$

2. Modelo de iluminación no uniforme

Los modelos de reflectancia describen la relación que existe entre la dirección de iluminación y la forma de la superficie. Las superficies pueden ser Lambertiana, especular o híbrida [1].

La Fig. 1 muestra un modelo geométrico de una superficie que está iluminada por una fuente de luz, en el cual, el eje z es el punto de observación.

En el modelo Lambertiano, la superficie refleja la luz en todas las direcciones con amplitudes iguales a $d_L = \cos(\theta)$, donde θ es el ángulo incidente, por ejemplo, el ángulo entre el vector $\mathbf{N}$ normal a la superficie y el vector de la dirección de iluminación $\mathbf{S}$.

Las superficies del tipo especular incluyen las reflexiones de espejo, refracciones, y cáustica [4]. Esta superficie puede describirse como $d_S = \delta(\theta_S - \theta)$, donde $\delta(x)$ es la función delta de Dirac, $\theta_S = \theta_N - \theta_v)$, θ_N es el ángulo entre $\mathbf{N}$ y el eje z, y $\theta_v)$ es el ángulo entre $\mathbf{N}$ y la dirección del observador.

Las superficies híbridas son combinaciones de componentes de superficies de tipo Lambertianas y especulares expresadas como $d_H = k_1 d_L + k_2 d_s$, donde $k_1 + k_2 = 1$.

En la Fig. 1 se observa que las direcciones de la iluminación se determinan por los parámetros τ, α y ρ, los cuales indican, el ángulo de rotación, el ángulo de inclinación y la magnitud del vector s_0, respectivamente.

Estos parámetros definen la posición de la fuente de iluminación con respecto a su superficie de origen [1].

Para una superficie Lambertiana, la luz reflejada desde un punto $\mathbf{x} = (x, y)$ está dada por:

$$d(x, y) = \cos(\theta) = \cos\left(\frac{\pi}{2} - \beta\right) \tag{1}$$

donde β [1] es:

$$\beta = \arctan\left[\frac{s_z}{\left[(s_x - x)^2 + (s_y - y)^2\right]^{1/2}}\right] \tag{2}$$

Sustituyendo la Ec. (2) en la Ec. (1), la luz reflejada está dada por:

$$d(x, y) = \cos\left(\frac{\pi}{2} - \arctan\left[\frac{s_z}{\left[(s_x - x)^2 + (s_y - y)^2\right]^{1/2}}\right]\right) \tag{3}$$

donde los términos s_x, s_y y s_z se definen por:

$$s_x = \rho \tan\tau \cos\alpha \tag{4}$$

$$s_y = \rho \tan\tau \operatorname{sen}\alpha \tag{5}$$

$$s_z = \frac{\rho}{\cos\tau} \tag{6}$$

Notemos que $d(x, y)$ depende de los parámetros de la dirección de la iluminación τ, α y ρ, indicando el ángulo de inclinación, el ángulo de rotación, y la magnitud del vector s_0, respectivamente. Estos parámetros pueden conocerse mediante un proceso de estimación para aplicaciones con escenas reales.

La función de iluminación de la Ec. (3) puede degradar la escena de entrada mediante el siguiente modelo de señal [5]:

$$f(\mathbf{x}) = s(\mathbf{x})d(\mathbf{x}) + n\mathbf{x} \tag{7}$$

donde $d(\mathbf{x})$ es el modelo de es el modelo de iluminación construido en el dominio espacial $\mathbf{x}$ que representa dos dimensiones.

3. Estimación de coeficientes de la función de iluminación no uniforme

De manera general, analizar imágenes capturada por un sensor optodigital, presenta sensibilidad en la respuesta cuando las escenas de entrada contienen iluminación no uniforme. Para ello, se implementa un análisis de aproximación de la función de iluminación que degrada la escena.

Consideremos que la región de soporte $w_t(x)$ es suficientemente pequeña, y que en ella se encuentra una porción de la señal de entrada tal que se puede considerar uniformemente iluminada. Ya que la iluminación de la escena de entrada es desconocida, consideremos el siguiente modelo de señal [6]:

$$\hat{f}(\mathbf{x}) = a_1(\mathbf{x})f(\mathbf{x}) + a_0\mathbf{x} \tag{8}$$

donde donde a_0 y a_1 son coeficientes de normalización, los cuales son considerados para aproximar la función de iluminación desconocida de la escena de entrada dentro de la región de soporte de la imagen del objeto w_t alrededor del k-ésimo pixel. Estos coeficientes se pueden obtener resolviendo un sistema de dos ecuaciones y dos incógnitas.

Minimizando el error cuadrático promedio (MSE) se obtiene:

$$MSE(k) = \sum_{\mathbf{x}\in w_t} [a_1(k)\ s(k + \mathbf{x}) + a_0(k) - t(\mathbf{x})]^2 \tag{9}$$

Entonces, resolviendo para $\frac{\partial}{\partial a_1} MSE(k) = 0$, la expresión se obtiene como:

$$a_1(k) = \frac{\sum_{\mathbf{x}\in w_t} t(\mathbf{x})\ s(k + \mathbf{x}) - a_0(k)}{\sum_{\mathbf{x}\in w_t} s^2(k + \mathbf{x})} \tag{10}$$

De igual manera, resolviendo la expresión mediante $\frac{\partial}{\partial a_0} MSE(k) = 0$, obtenemos:

$$a_0(k) = \frac{1}{|w_t|}\sum_{\mathbf{x}\in w_t} t(\mathbf{x}) - a_1(k)\frac{1}{|w_t|}\sum_{\mathbf{x}\in w_t} s(k + \mathbf{x}) \tag{11}$$

donde $|w_t|$ es la suma de los elementos de $s(\mathbf{x})$.

Sustituyendo la Ec. (11) en (10), el coeficiente $a_1(k)$ se expresa como:

$$a_1(k) = \frac{\sum_{\mathbf{x}\in w_t} t(\mathbf{x})\ s(k + \mathbf{x}) - \bar{t}\sum_{\mathbf{x}\in w_t} s(k + \mathbf{x})}{\sum_{\mathbf{x}\in w_t} s^2(k + \mathbf{x}) - \bar{s}(k)\sum_{\mathbf{x}\in w_t} t(\mathbf{x})\ s(k + \mathbf{x})} \tag{12}$$

Simplificando las Ec. (11) y (12), los coeficientes se pueden estimar explícitamente mediante [6]:

$$a_1(k) = \frac{\sum_{\mathbf{x}\in w_t} t(\mathbf{x})\ s(k + \mathbf{x}) - |w_t|\ \bar{t}\ \bar{s}(k)}{\sum_{\mathbf{x}\in w_t} [s(k + \mathbf{x})]^2 - |w_t|\,[\bar{s}(k)]^2} \tag{13}$$

$$a_0(k) = \bar{t} - a_1(k)\ \bar{s}(k) \tag{14}$$

donde $\bar{t}$ y $\bar{s}(k)$ son los promedios del objeto y la ventana local sobre w_t en la k-ésima posición de la ventana, respectivamente.

4. Implementación

El procesamiento de imágenes en tiempo real y el cálculo de gráficos de alta definición requieren una gran cantidad de recursos de cómputo. Para satisfacer esta demanda de cómputo se usa una unidad de procesamiento gráfico (GPU) ya que presenta ventajas para la implementación por la capacidad de ejecución multiproceso altamente en paralelo, con gran poder de cómputo y ancho de banda de memoria muy alta [7].

Con la creciente capacidad de programación de los GPU, los cuales son capaces de realizar cálculos mediante operaciones en paralelo para grandes cantidades de datos, específicamente utilizados para gráficos, para los cuales han sido diseñados. En la actualidad hay procesadores potentes, que alcanzan una alta velocidad que pueden procesar distintas aplicaciones que se pueden expresar como en el cálculo de datos paralelos [8].

La arquitectura CUDA, por sus siglas en inglés, *Compute Unified Device Architecture*, es aplicada en hardware y software para la emisión y administración de operaciones en GPU como un dispositivo de computación de datos en paralelo sin la necesidad de asignar a una API de gráficos. La API de CUDA comprende una extensión al lenguaje de programación C, por lo tanto, con la ayuda de CUDA, programar un GPU es mucho más accesible [8]-[9].

Se implementa un algoritmo para la reconstrucción de una escena degradada por iluminación. La implementación se ha desarrollado en una computadora CPU con sistema operativo Lynux (Debian 6.0.5). Este cuenta con una unidad de procesamiento gráfico (GPU) GeForce GTX 580 de NVIDIA, el cual se basa en lenguaje C con CUDA 5.0.

Se utilizan bibliotecas de funciones (APIs) de CUDA, tales como CURAND y CUDART. El manejo de imágenes se ha utilizado con OpenCV y C++.

Los pasos de operación se describen en la Tabla 1 (Algoritmo 1).

Tabla 1. **Algoritmo 1**. Procedimiento para restauración de imágenes

Entrada:
$s(\mathbf{x})$ (Escena de entrada)
$vart$ (Varianza del ruido con distribución normal)
mt (Valor esperado del ruido con distribución normal)
τ (Ángulo de inclinación de la fuente de iluminación)
α (Ángulo de rotación de la fuente de iluminación)
ρ (Magnitud del vector de la fuente de iluminación)

Salida:
$f(\mathbf{x})$ (Escena restaurada)

Pasos del Algoritmo:
Paso 1. Inicialización. Capturar escena de entrada $s(\mathbf{x})$.
Paso 2. Definición de parámetros de iluminación. Establecer valores constantes de los parámetros que definen la función de iluminación τ, α y ρ.
Paso 3. Definición de parámetros de ruido. Establecer valores constantes de los parámetros de varianza y valor esperado, $vart$ y mt, respectivamente, de la función del ruido aditivo con distribución normal.
Paso 4. Degradación de la señal. Calcular el modelo de señal degradada por la función de iluminación no uniforme y ruido aditivo $f(\mathbf{x}) = s(\mathbf{x})d(\mathbf{x}) + n(\mathbf{x})$.
Paso 5. Normalización. Calcular los coeficientes de normalización $a_0(\mathbf{x})$ y $a_1(\mathbf{x})$.
Paso 6. Restauración. Estimar la restauración de la imagen mediante los coeficientes de normalización y el modelo de señal $f(\mathbf{x}) = a_1(\mathbf{x})f(\mathbf{x}) + a_0(\mathbf{x})$.

1. Inicialización. Se captura una escena de entrada de tamaño 800×608 pixeles mostrada en la Fig. 2(a). Se utiliza OpenCV para cargar las imágenes y dimensionarlas al tamaño original [10]-[11].

2. Definición de parámetros de iluminación La escena de entrada se degrada con una función de iluminación no homogénea, la cual contiene ruido aditivo, tal como se muestra en la Fig. 2(b) y 2(c), respectivamente. La función de iluminación se genera mediante un *kernel* (función en paralelo), en donde cada hebra procesa un pixel de la imagen.

3. Definición de parámetros de ruido. La función del ruido aditivo se implementa con la biblioteca CURAND, en la que se utiliza un generador de números aleatorios con distribución normal [12], tal como se muestra en el Algoritmo 2, asumiendo que el ruido aditivo contiene una respuesta Gaussiana. La implementación de CURAND se realiza en dos etapas. La primera etapa consiste en generar números aleatorios en el *device* (GPU), llamándolos con la librería `/include/curand.h` que ocurre dentro del *host* (CPU). La segunda etapa consiste en definir las funciones dentro del device para generar secuencias de números aleatorios, mediante la librería `/include/curand_kernel.h` esto permite que los números generados se puedan utilizar directamente en las funciones kernel, sin requerir la escritura y luego la lectura desde la memoria global [12].

4. Degradación de la señal. La degradación de la escena de entrada se obtiene utilizando el modelo multiplicativo y aditivo de la Ec. (7), tomando en cuenta las funciones resultantes de la iluminación y ruido, obtenidas en el Paso 2 y Paso 3.

5. Normalización. Se asume que la función de iluminación es desconocida, por lo cual este sistema

Tabla 2. **Algoritmo 2**. Generación de la función de ruido con distribución normal.

Paso 1. Inicialización.
Inicializar la variable del generador.
Paso 2. Crear el generador aleatorio.
Especificar tipo de generación de valores aleatorios.
Paso 3. Crear semilla.
Determinar la semilla del generador que sea dependiente del reloj.
Paso 4. Reservar memoria.
Utilizar variable de device del tamaño de la señal de entrada.
Paso 5. Generación ruido Gaussiano.
Utilizar la función de generación de números aleatorios con distribución normal, utilizando los parámetros de valor esperado y varianza $vart$ y mt, respectivamente.

restaura la imagen de entrada que ha sido degradada, mediante el cálculo de coeficientes a_0 y a_1 , representados en las Ecs. (13) y (14). Para la obtención de los coeficientes se realiza el cálculo de $\bar{s}(\mathbf{x})$, que es la imagen promedio de la escena de entrada en la k-ésima posición dentro de la región de soporte w_t. Este cálculo incluye un algoritmo de reducción [13] en una función de memoria global [9], tal como se muestra en la Fig. 3. La implementación del algoritmo de reducción se detalla en el Algoritmo 3.

6. Restauración. La restauración de la imagen se obtiene mediante el ajuste de los coeficientes de normalización, dada por la expresión de la Ec. (8). La calidad de restauración de la imagen se obtiene mediante el error cuadrático promedio (MSE) y el pico máximo de la relación señal a ruido (PSNR), obtenidas a continuación:

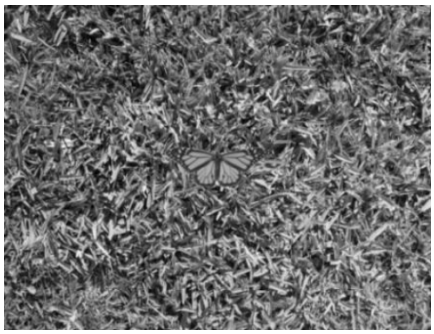
$$MSE = \sum_{x,y} \left(s(x,y) - \hat{f}(x,y) \right)^2 \quad (15)$$

$$PSNR_{dB} = 10 \log_{10} \left(\frac{\max \left(s(x,y)^2 \right)}{MSE} \right) \quad (16)$$

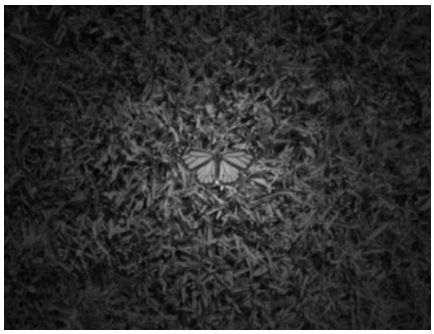
donde $s(x,y)$ es la escena sin degradación, y $\hat{f}(x,y)$ es la imagen estimada de la restauración.

5. Resultados

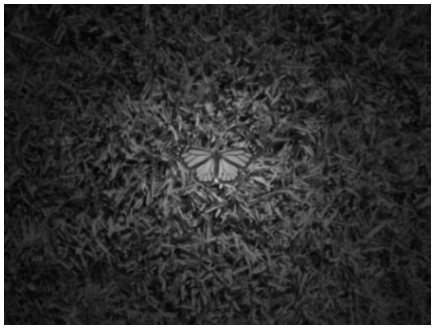
En esta sección se presentan los resultados en cuestiones de calidad de restauración y tiempos de ejecución. Se ha implementado el algoritmo de restauración de imágenes en paralelo, en donde las tareas de cada hilo se distribuyen dependiendo del número de elementos y el número de hebras. En esta técnica de procesamiento de imágenes, la operación ejecutada en cada pixel es



(a) Escena de entrada.



(b) Degradación de la imagen de entrada mediante una función de iluminación.



(c) Imagen iluminada que contiene ruido aditivo.

Figura 2. Degradación de imagen

Tabla 3. **Algoritmo 3**. Sistema de reducción para el cálculo de imagen promedio.

Paso 1. Inicialización de kernel.
Determinar índices para hebras en dos dimensiones.
Paso 2. Inicializar bloque.
Especificar que el valor del bloque k inicialmente valdrá la mitad de las dimensiones de la imagen de entrada.
Paso 3. Realizar reducción. 3.1 Mientras el bloque sea divisible entre dos, sumar los valores de cada bloque en orden con sus índices respectivos.
3.2 Sincronización de las hebras.
3.3 Reducir el bloque a la mitad.
Paso 4. Suma resultante.
Si el índice del bloque es el primero, la operación resultante se obtiene sumando los primeros índices de cada bloque.

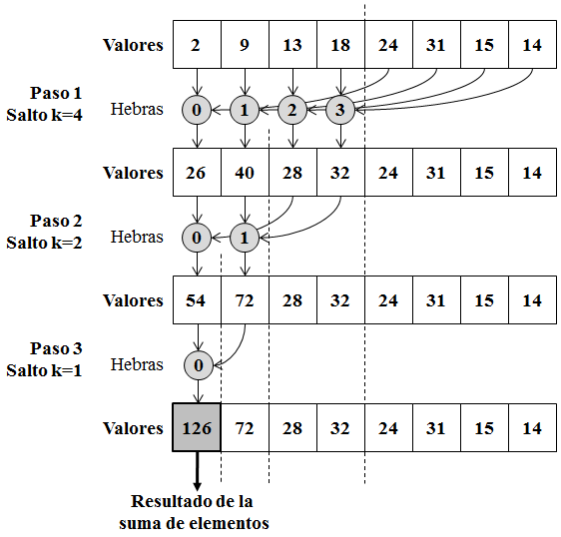


Figura 3. Desarrollo del algoritmo de reducción para la suma de elementos de un vector.

la misma y es independiente de la operación en otros pixeles, además su resultado no depende del resultado de la operación de otro pixel, por lo tanto se puede realizar en toda la imagen de manera simultánea [14].

Por ejemplo, para un CPU de 4 núcleos el número máximo de operaciones que pueden ocurrir en paralelo es 4. El uso del GPU está orientado para este tipo de aplicaciones de paralelización masiva.

Por ejemplo, el GPU utilizado en este trabajo NVIDIA GTX 580 tiene 512 multiprocesadores (16 por 32 núcleos), cada uno de los cuales soportan 1536 hilos de ejecución en paralelo. Combinados, estos multiprocesadores permiten que el GPU pueda operar más de 700.000 ejecuciones simultáneas de instrucciones individuales. Así, para un algoritmo que tiene un alto grado de paralelismo, una aplicación GPU es mucho más rápida que una implementación en CPU.

En la Tabla 4 se muestran los tiempos de ejecución resultantes con respecto a la utilización del algoritmo con distintos tipos de memoria. Se puede observar que la memoria compartida presenta ventajas ante la memoria global en la facilidad del acceso y procesamiento de la información. Se realizan comparaciones con respecto a distintas dimensiones de la imagen de entrada, las cuales presentan las mismas condiciones de degradación.

En la Fig. 4 se muestra la curva resultante del speedup para distintas dimensiones de la imagen de entrada, tomando en cuenta que es una imagen cuadrada, donde N representa el valor de los renglones o columnas.

En la Tabla 5 se muestra la evaluación de la calidad

Tabla 4. Tiempos de ejecución resultante con respecto al uso de memoria para distintas dimensiones de la imagen de entrada.

Tamaño	Tiempo de ejecución de GPU (ms)		Speedup
	Memoria compartida	Memoria Global	
32	0.190176	0.196992	1.04
64	0.20272	0.22	1.09
128	0.490624	0.622624	1.27
256	0.433856	1.208224	2.78
512	0.915136	4.4688	4.88
1024	1.876384	17.944576	9.56

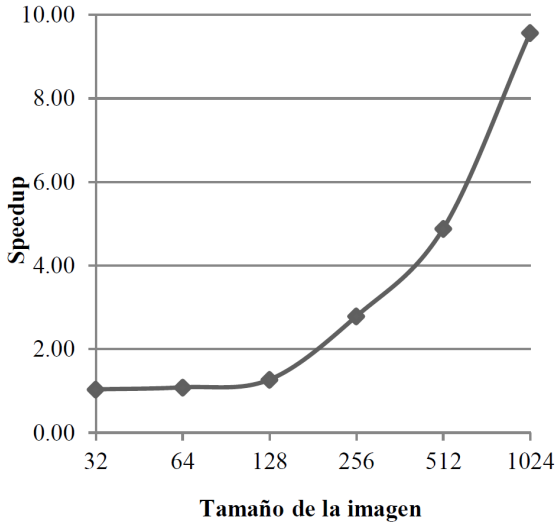


Figura 4. Speedup resultante con respecto a la dimensión de la imagen de entrada.

de restauración de la imagen de entrada mediante las métricas de MSE y PSNR de las Ecs. (16) y (16). La imagen utilizada es de dimensiones 800×608 pixeles, contiene un objeto dentro de un fondo, y es degradada utilizando distintos valores de los parámetros de iluminación, con valor de 100 dB de SNR de ruido aditivo.

En la Tabla 6 se muestran los resultados de las métricas de calidad con respecto a distintos niveles de SNR de ruido aditivo. Se utiliza una imagen que contiene un objeto sin fondo. Esta imagen es cuadrada que contiene dimensiones de 512×512 pixeles para todas las muestras, y con parámetros de iluminación de $\rho = 150$, $\alpha = 45$ y $\tau = 0$.

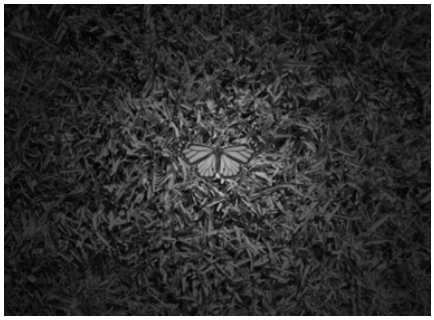
En las Figs. 5(a) y 5(b) se muestran las imágenes resultantes de la iluminación y restauración de la escena original sin degradación y libre de ruido aditivo presentada en la Fig. 2(a).

Tabla 5. Resultados de la calidad de restauración de imagen que contiene un objeto dentro de un fondo para diferentes parámetros de iluminación con dimensiones 800×608.

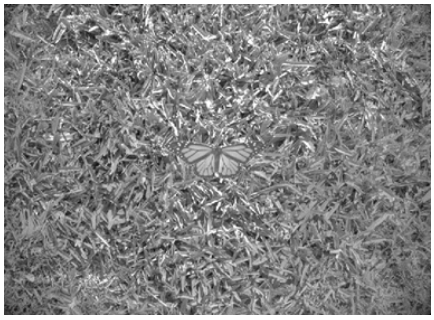
Parámetros de iluminación	MSE (%)	PSNR (dB)
$\rho = 40, \alpha = 45, \tau = 0$	15.1358	34.2206
$\rho = 80, \alpha = 45, \tau = 0$	14.1379	35.9241
$\rho = 160, \alpha = 45, \tau = 0$	14.1237	36.2839

Tabla 6. Evaluación de restauración para distintos niveles de ruido aditivo para una imagen que contiene un objeto sin fondo de dimensiones 512×512.

Nivel de SNR de ruido aditivo (dB)	MSE (%)	PSNR (dB)
100	15.5137	8.09183
50	15.5176	8.09176
20	15.5722	8.07651
10	16.9016	7.72071
0	27.7875	5.56151



(a) Escena de entrada.Imagen resultante de la degradación por iluminación con niveles $\rho = 80, \alpha = 45, \tau = 0$ y 40 dB de SNR de ruido aditivo.



(b) Imagen resultante de la restauración.

Figura 5. Restauración de imagen.

6. Conclusiones

Mediante el desarrollo de este trabajo, se puede demostrar que el desempeño de restauración de imágenes puede incrementarse significativamente en condiciones cambiantes de iluminación y ruido, al utilizar algoritmos implementados en cómputo paralelo. Estos contribuyen a estimar los parámetros de la función de degradación en tiempo real.

Se encuentra un área de oportunidad en trabajos futuros, con el fin de obtener mayores valores de speedup al usar algoritmos de reducción en funciones atómicas, además de aprovechar de manera más eficiente los recursos de memoria compartida y de textura en la generación de la función de iluminación. El uso de bibliotecas de funciones APIs de CUDA podrá facilitar la programación de funciones, e incrementar la eficiencia en la ejecución.

En este trabajo se han presentado las mejoras de procesamiento de imagen aplicadas en procesador gráfico. Al realizar el procesamiento de imágenes en un GPU, se obtienen mejoras significativas con respecto al tiempo de ejecución. Además, proporciona una base de para poner en práctica trabajos futuros en algoritmos de procesamiento con mayor complejidad computacional, tales como el filtrado por correlación en el dominio de la frecuencia, que implica el uso de diferentes funciones para obtener resultados óptimos con respecto a la calidad de restauración de la imagen y respuesta en tiempo real. Los trabajos futuros realizados con este enfoque serán usados en migrar la aplicación de reconocimiento de patrones en paralelo utilizando de manera óptima los recursos computacionales de la tarjeta GPU.

Agradecimientos

Se agradece el apoyo recibido de los proyectos SIP-20130489 y SIP-20131209.

Referencias

[1] V. H. Diaz-Ramirez, y V. Kober, “Target recognition under nonuniform illumination conditions,” *Appl. Opt.*, Vol. 48, pp. 1408-1418, 2009.

[2] P. Dutré, K. Bala, y P. Bekaert, *Advanced global illumination*, 2a. edición, Wellesley: A K Peters, Ltd., 2006.

[3] M. Sherry, y A. Shearer, “IMPAIR: massively parallel deconvolution on the GPU,” *Proc. SPIE, Image Processing: Algorithms and Systems XI*, Vol. 8655, Febrero, 2013.

[4] L. Szirmay-Kalos, L. Szécsi, y M. Sbert. *GPU-based techniques for global illumination effects*, Hungary: Morgan & Claypool, 2008.

[5] V. H. Diaz-Ramirez, y V. Kober, “Illumination invariant adaptive joint transform correlator,” *Proc. SPIE, In Optics and Photonics*

for Information Processing, Vol. 6695, pp. 66951B1-66951B7, 2007.

[6] S. Martinez-Diaz, y V. Kober. "Nonlinear synthetic discriminant function filters for illumination-invariant pattern recognition," *Optical Engineering*, Vol. 47(6), pp. 067201-067201-9, 2008.

[7] NVIDIA Corporation, "NVIDIA CUDA C Programming Guide," PG-02829-001 v5.5, Julio, 2013.

[8] C. Jiang, P. Li, y Q. Luo, "High speed parallel processing of biomedical optics data with PC graphic hardware," *Proc. SPIE, Optical Sensors and Biophotonics*, Vol. 7634, Nov., 2009.

[9] NVIDIA Corporation, "CUDA C best practices guide, Design Guide," DG-05603-001 v5.5, Julio, 2013.

[10] G. Bradski, y A. Kaebler. *Learning OpenCV, Computer Vision with the OpenCV Library*, Sebastopol: O'Reilly, 2008.

[11] R. Laganière, *OpenCV 2 Computer Vision Application Programming Cookbook*, Birmingham: Packt Publishing, 2011.

[12] NVIDIA Corporation, "NVIDIA CUDA Toolkit V5.5 RN-06722-001 V.5.5," Mayo, 2013.

[13] J. Sanders, y E. Kandrot. *CUDA by example, An Introduction to General-Purpose GPU Programming*, Addison-Wesley, 2011.

[14] S. N. Swetadri Vasan *et al*, "Graphics processing unit (GPU) implementation of image processing algorithms to improve system performance of the control acquisition, processing, and image display system of the micro-angiographic fluoroscope," *Proc. SPIE 8313, Medical Imaging 2012: Physics of Medical Imaging*, Vol. 8313, Febrero, 2012.