Universidad Nacional San Agustín de Arequipa
Facultad de Ingeniería de Producción y Servicios
Escuela Profesional de Ciencia de la Computación

UNSA
UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

Laboratorio Cuda

CUDA

Autor: Juan Manuel Soto Begazo
CUI: 20192267
Docente: Dr. Alvaro Henry Mamani Aliaga
Computación Paralela y Distribuida
Grupo A
Arequipa, Peru

1. Actividades

Las tareas a realizar son las siguientes:

- Programas de Escala de Grises y Blur usando CUDA

La implementación se encuentra en el siguiente repositorio de GitHub.

2. Desarrollo

2.1. Gray

La función a continuación implementa una conversión de una imagen en color a escala de grises utilizando el modelo de ponderación de luminancia estándar. Cada hilo en un bloque de hilos de una GPU se encarga de procesar un píxel de la imagen de entrada. El índice del hilo se calcula en base a las dimensiones de la imagen. Para cada píxel, se obtienen los componentes de color rojo, verde y azul, y se aplica la fórmula de conversión ponderada para calcular el nivel de gris correspondiente. El resultado se almacena en la imagen de salida, y se asigna el mismo valor de gris a los componentes de rojo, verde y azul del píxel de salida para producir la imagen en escala de grises. Es importante destacar que este código está diseñado para ejecutarse en paralelo en una GPU, aprovechando la capacidad de procesamiento masivo de hilos para mejorar la eficiencia en la conversión de color a escala de grises.

```
1  void colorToGreyscaleConversion(unsigned char* out, unsigned char* in, int w, int h)
2  {
3      int Col = threadIdx.x + blockIdx.x * blockDim.x;
4      int Row = threadIdx.y + blockIdx.y * blockDim.y;
5      if (Col < w && Row < h)
6      {
7          int greyOffset = Row * w + Col;
8
9          int offset = greyOffset * CHANNELS;
10         unsigned char smr = in[offset];
11         unsigned char smg = in[offset + 1];
12         unsigned char smb = in[offset + 2];
13
14         out[offset] = 0.21f * smr + 0.71f * smg + 0.07f * smb;
15         out[offset+1] = 0.21f * smr + 0.71f * smg + 0.07f * smb;
16         out[offset+2] = 0.21f * smr + 0.71f * smg + 0.07f * smb;
17     }
18 }
```

2.1.1. Resultados de ejecución

Figura 2.1: Imagen Original

Figura 2.2: Imagen con gray

2.2. Blur

La función *colorToBlurConversion* implementa una operación de desenfoque en una imagen en color utilizando una técnica de promedio ponderado. Cada hilo en un bloque de hilos de una GPU procesa un píxel de la imagen de entrada. Los índices de fila y columna se calculan a partir de las dimensiones de la imagen y el bloque de hilos. Dentro de un bucle anidado que itera sobre una ventana de píxeles definida por BLUR_SIZE, se acumulan los valores de los componentes de color (rojo, verde y azul) de los píxeles circundantes al píxel actual. Se lleva un conteo de los píxeles válidos dentro de la ventana. Luego, se calcula el promedio ponderado de los valores acumulados dividiendo por el número de píxeles válidos. El resultado se almacena en la imagen de salida, y este proceso se repite para cada píxel de la imagen original. La implementación aprovecha la paralelización proporcionada por la GPU para realizar eficientemente la operación de desenfoque en cada píxel de la imagen de entrada. Es importante mencionar que la eficacia del desenfoque depende del valor de BLU_SIZE y cómo afecta el área circundante de cada píxel en el cálculo del promedio ponderado.

```
1  void colorToBlurConversion(uc* out, uc* in, int w, int h)
2  {
3      int Col = blockIdx.x * blockDim.x + threadIdx.x;
4      int Row = blockIdx.y * blockDim.y + threadIdx.y;
5
6      if (Col < w && Row < h) {
7          int smr = 0;
8          int smg = 0;
9          int smb = 0;
10         int pixels=0;
11         int Offset = (Row * w + Col) * CHANNELS;
12
13         for (int blurRow = -BLUR_SIZE; blurRow < BLUR_SIZE + 1; ++blurRow)
14         {
15             for (int blurCol = -BLUR_SIZE; blurCol < BLUR_SIZE + 1; ++blurCol)
16             {
17                 int curRow = Row + blurRow;
18                 int curCol = Col + blurCol;
19                 if (curRow > -1 && curRow < h && curCol > -1 && curCol < w)
20                 {
21                     int current = (curRow * w + curCol) * CHANNELS;
22                     smr += in[current];
23                     smg += in[current + 1];
24                     smb += in[current + 2];
25                     pixels++;
26                 }
27             }
28         }
29         out[Offset] = (unsigned char)(smr / pixels);
30         out[Offset + 1] = (unsigned char)(smg / pixels);
31         out[Offset + 2] = (unsigned char)(smb / pixels);
32     }
```

```
33 }
```

2.2.1. Resultados de ejecución

Figura 2.3: Imagen Original

Figura 2.4: Imagen con BLur

3. Conclusiones

- Programación Paralela en CUDA: La realización exitosa de la conversión a escala de grises y el desenfoque utilizando CUDA proporcionaron la oportunidad de aplicar conceptos teóricos aprendidos y adquirir habilidades prácticas en el desarrollo de algoritmos paralelos.
- Eficiencia en el Procesamiento de Imágenes: La implementación de operaciones de procesamiento de imágenes en una GPU permitió experimentar con el rendimiento y la eficiencia de CUDA en comparación con implementaciones en CPU. Las conclusiones obtenidas pueden ser fundamentales para entender las ventajas y desafíos asociados con la aceleración de tareas de visión por computadora mediante GPU.