Proyecto No.3:
CUDA

Oliver Graf 17190
Kristen Brandt 171482

Índice

Transformada de Hough

La transformación de Hough es una técnica computacional que se utiliza para poder detectar líneas rectas en una imagen y también para detectar bordes en figuras más complejas. Esta transformada se propuso por Paul Hough en 1962 y empezó a volverse más popular en 1981 cuando se publicó un artículo sobre la transformada (Solberg, 2021).

Para el proyecto la versión transformada de Hough que se estará utilizando es la versión lineal. Esta versión se aplica a imagenes que estan en blanco y negro. Cómo sirve la transformación detectando si un píxel es parte de una línea, esta detección se hace asignándole un peso. Al final de las asignaciones de peso se eligen las líneas que tienen más puntos para ser elegidas para detección.

La transformada de Hough es una técnica robusta que presenta buenos resultados a pesar de ruido y oclusión en la imagen, esta ha sido utilizada en muchas aplicaciones de procesamiento de imágenes, navegación de robots, reconocimiento de objetos e inspección industrial (Braak, 2011).

Implementación en CUDA

Desde el 2006 que salió CUDA se han vuelto más programables y usables para otras aplicaciones que no solo son gráficas de computadoras. El primer paso de la transformada es la detección de bordes, esta se hace por medio de convoluciones. Un ejemplo es la detección Sobel edge que puede ser encontrada en NVIDIA CUDA SDK (Braak, 2011).

Uso de memoria constante en el kernel (segunda entrega)

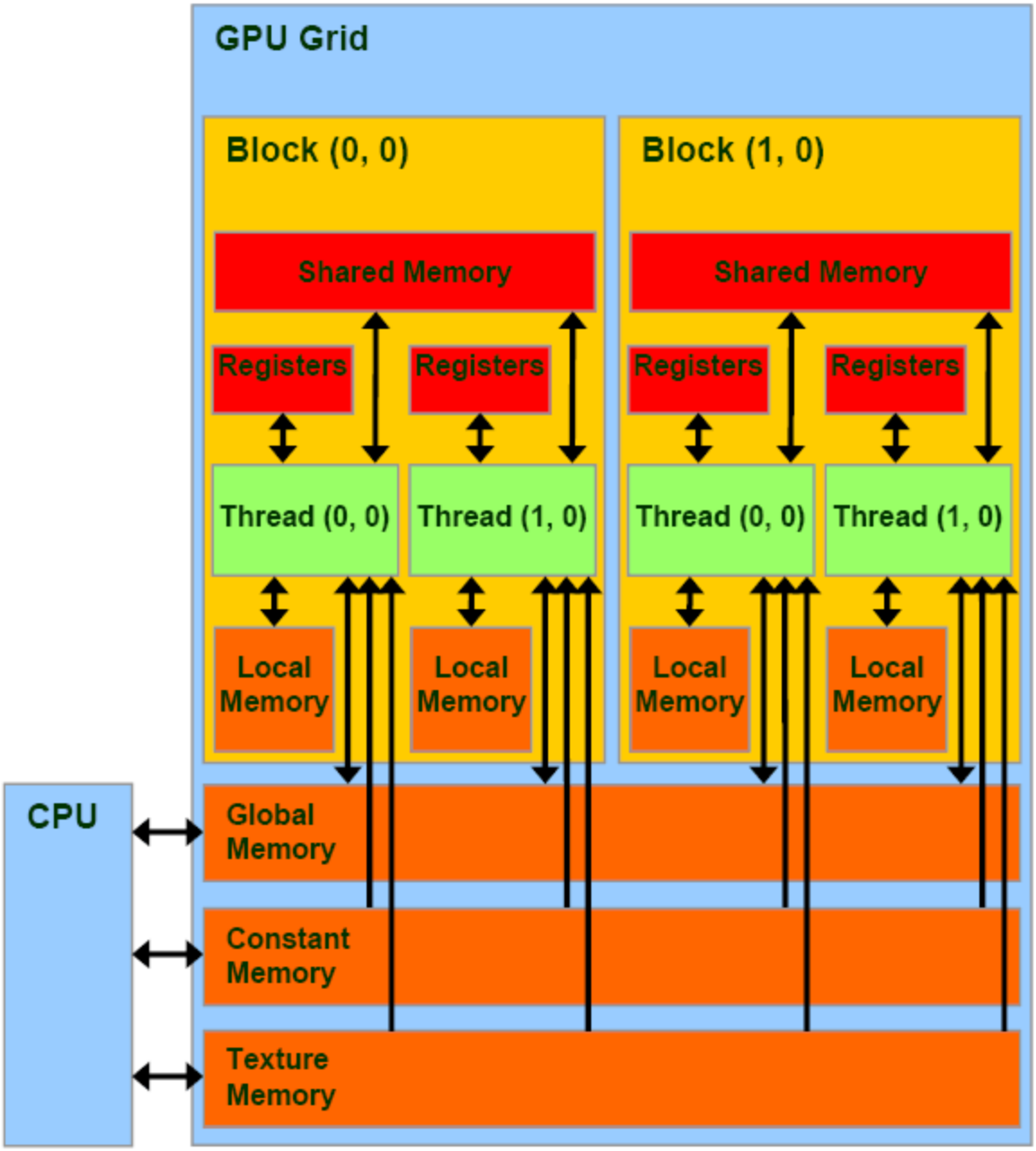
Al adaptar el kernel para que use memoria constante, seguimos utilizando memoria *off-chip*. Pero se trata de una parte de la memoria del device que está dedicada a almacenar datos que permanecen constantes. La manera de utilizar dicha memoria, es declarando los arreglos que van a utilizarse en el kernel con un scope global y con la bandera __constant__.

```
__constant__ float d_Cos [ degreeBins ] ;
__constant__ float d_Sin [ degreeBins ] ;
```

De esta manera, ya no es necesario que el kernel reciba los arreglos como parámetro de entrada, en otras palabras, los arreglos se pueden utilizar dentro del kernel, porque éstos están declarados con un scope global.

```
__global__ void GPU_HoughTran (unsigned char *pic, int w, int h, int *acc, float rMax,
float rScale)
{

  int gloID = blockIdx.x * blockDim.x + threadIdx.x ;
  if (gloID > w * h) return;      // in case of extra threads in block

  int xCent = w / 2;
  int yCent = h / 2;

  //TODO explicar bien bien esta parte. Dibujar un rectangulo a modo de imagen sirve para
  visualizarlo mejor
  int xCoord = gloID % w - xCent;
  int yCoord = yCent - gloID / w;

  //TODO eventualmente usar memoria compartida para el acumulador

  if (pic[gloID] > 0)
    {
      for (int tIdx = 0; tIdx < degreeBins; tIdx++)
        {
          //TODO utilizar memoria constante para senos y cosenos
          //float r = xCoord * cos(tIdx) + yCoord * sin(tIdx); //probar con esto para ver
          diferencia en tiempo
          float r = xCoord * d_Cos[tIdx] + yCoord * d_Sin[tIdx];
          int rIdx = (r + rMax) / rScale;
          //debemos usar atomic, pero que race condition hay si somos un thread por pixel?
          explique
          atomicAdd (acc + (rIdx * degreeBins + tIdx), 1);
        }
    }
```

En el diagrama de abajo podemos ver que, como se mencionó anteriormente, la memoria constante es también parte de la memoria *off-chip*. Por lo tanto, para que el GPU pueda procesar los datos, es necesario trasladarlos desde el host. En este caso, el traslado de la información se realiza mediante la función cudaMemcpyToSymbol()l.



(*What is "Constant memory" in cuda, n.d.)*

```
// TODO eventualmente volver memoria global
cudaMemcpyToSymbol ( d_Cos , pcCos , sizeof ( float ) * degreeBins ) ;
cudaMemcpyToSymbol ( d_Sin , pcSin , sizeof ( float ) * degreeBins ) ;
```

Aplicaciones de las diferentes memorias de GPU

Resultados

```
student20-15@ip-172-31-8-99:~/p3$ ./hough runway.pgm
Elapsed time : 1.58 ms
Calculation mismatch at : 1803 1446 1445
Calculation mismatch at : 1893 1506 1507
Calculation mismatch at : 5931 1653 1654
Calculation mismatch at : 6021 1816 1815
Calculation mismatch at : 6104 1642 1641
Calculation mismatch at : 6194 1586 1587
Done!
double free or corruption (!prev)
```

Tabla No. 1 Tiempos con memoria global

Corrida	Tiempo
1	1.56
2	1.58
3	1.58
4	1.59
5	1.57
6	1.58
7	1.57
8	1.58
9	1.57
10	1.58

Promedio	1.58

```
student20-15@ip-172-31-8-99:~/p3$ ./hough runway.pgm
Elapsed time : 1.61 ms
Calculation mismatch at : 1803 1446 1445
Calculation mismatch at : 1893 1506 1507
Calculation mismatch at : 5931 1653 1654
Calculation mismatch at : 6021 1816 1815
Calculation mismatch at : 6104 1642 1641
Calculation mismatch at : 6194 1586 1587
Done!
double free or corruption (!prev)
```

Tabla No. 2 Tiempos con memoria constante y global

Corrida	Tiempo
1	1.61
2	1.60
3	1.61
4	1.60
5	1.61
6	1.61
7	1.60
8	1.60
9	1.61
10	1.62
Promedio	**1.60**

Tabla No. 3 Tiempos con memoria global, constante y compartida

```
student20-15@ip-172-31-8-99:~/p3$ ./hough runway.pgm
Elapsed time : 1.65 ms
Calculation mismatch at : 1803 1446 1445
Calculation mismatch at : 1893 1506 1507
Calculation mismatch at : 5931 1653 1654
Calculation mismatch at : 6021 1816 1815
Calculation mismatch at : 6104 1642 1641
Calculation mismatch at : 6194 1586 1587
Done!
```

Corrida	Tiempo
1	1.65
2	1.65
3	1.65
4	1.65
5	1.65
6	1.65
7	1.65
8	1.65
9	1.65
10	1.65
Promedio	**1.65**

Conclusiones/Recomendaciones

Literatura Citada

Solberg, A. (2021). *Hough transform*. Universitetet i Oslo. Retrieved June 1, 2022, from https://www.uio.no/studier/emner/matnat/ifi/INF4300/h09/undervisningsmateriale/hough09.pdf

What is "Constant memory" in cuda: Constant memory in Cuda. What is "Constant Memory" in CUDA | Constant Memory in CUDA. (n.d.). Retrieved from http://cuda-programming.blogspot.com/2013/01/what-is-constant-memory-in-cuda.html