


```
1 # Check GPU
2 !nvidia-smi
3
```

🔄 Tue May 20 19:37:55 2025

NVIDIA-SMI 550.54.15				Driver Version: 550.54.15			CUDA Ver
GPU	Name		Persistence-M	Bus-Id	Disp.A	Volati	
Fan	Temp	Perf	Pwr:Usage/Cap		Memory-Usage	GPU-Ut	
=====							
0	Tesla T4		Off	00000000:00:04.0	Off		
N/A	48C	P0	27W / 70W	102MiB / 15360MiB		0	
=====							
Processes:							
GPU	GI	CI	PID	Type	Process name		
	ID	ID					
=====							

```
1 from google.colab import files
2 uploaded = files.upload()
```

🔄 Choose Files  6 files

dog54.jpeg(image/jpeg) - 19075 bytes, last modified: n/a - 100% done

dog53.jpeg(image/jpeg) - 14693 bytes, last modified: n/a - 100% done

dog52.jpeg(image/jpeg) - 18096 bytes, last modified: n/a - 100% done

dog51.jpeg(image/jpeg) - 17665 bytes, last modified: n/a - 100% done

dog50.jpeg(image/jpeg) - 16623 bytes, last modified: n/a - 100% done

dog49.jpeg(image/jpeg) - 18057 bytes, last modified: n/a - 100% done

Saving dog54.jpeg to dog54 (2).jpeg

Saving dog53.jpeg to dog53 (2).jpeg

Saving dog52.jpeg to dog52 (2).jpeg

Saving dog51.jpeg to dog51 (2).jpeg

Saving dog50.jpeg to dog50 (2).jpeg

Saving dog49.jpeg to dog49 (2).jpeg

```
1 import os
2
3 # List all files to confirm it's there
4 print(os.listdir())
5
```

🔄 ['.config', 'dog52.jpeg', 'rotate.cu', 'dog50.jpeg', 'dog36.jpeg', 'dog50 (

```
1 # Download Lena image from OpenCV GitHub
2 !wget https://raw.githubusercontent.com/opencv/opencv/master/samples/data/1
3
```

```
➡ --2025-05-20 19:40:43-- https://raw.githubusercontent.com/opencv/opencv/master/samples/data/1
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199
HTTP request sent, awaiting response... 200 OK
Length: 91814 (90K) [image/jpeg]
Saving to: 'Lena.png'
```

```
Lena.png          100%[=====>] 89.66K  --.-KB/s    in 0.01
```

```
2025-05-20 19:40:44 (8.99 MB/s) - 'Lena.png' saved [91814/91814]
```

```
1 import cv2
2 from matplotlib import pyplot as plt
3
4 # Load the image
5 img = cv2.imread("Lena.png")
6
7 # Check if image is read correctly
8 if img is None:
9     print("Failed to load image.")
10 else:
11     img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
12     plt.imshow(img_rgb)
13     plt.title("Loaded Lena.png")
14     plt.axis('off')
15     plt.show()
16
```



Loaded Lena.png



```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 # Read the uploaded image
6 img = cv2.imread("Lena.png")
7
8 # Check if the image was loaded successfully
9 if img is None:
10     print("Error: Could not load image 'Lena.png'. Please ensure the file i
11 else:
12     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
13     plt.imshow(img)
14     plt.title("Original Image")
15     plt.axis('off')
16     plt.show()
```



Original Image



```

1 %%writefile rotate.cu
2 #include <cuda_runtime.h>
3
4 __global__ void rotateKernel(unsigned char* input, unsigned char* output, i
5     int x = blockIdx.x * blockDim.x + threadIdx.x; // column
6     int y = blockIdx.y * blockDim.y + threadIdx.y; // row
7
8     float radians = angle * 3.14159265 / 180.0;
9     int xc = width / 2;
10    int yc = height / 2;
11
12    if (x < width && y < height) {
13        int tx = x - xc;
14        int ty = y - yc;
15
16        int srcX = cos(radians) * tx + sin(radians) * ty + xc;
17        int srcY = -sin(radians) * tx + cos(radians) * ty + yc;
18
19        for (int c = 0; c < 3; c++) {
20            if (srcX >= 0 && srcX < width && srcY >= 0 && srcY < height) {
21                output[(y * width + x) * 3 + c] = input[(srcY * width + src
22            } else {
23                output[(y * width + x) * 3 + c] = 255; // white background
24            }
25        }
26    }
27 }
28

```

➡ Overwriting rotate.cu

```

1 !nvcc -o rotate rotate.cu
2

```

➡ /usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/11/../../../../x86_64-linux-gnu/Scr
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status

```
1 !pip install pycuda
```

```
⇒ Collecting pycuda
  Downloading pycuda-2025.1.tar.gz (1.7 MB)
    1.7/1.7 MB 39.5 MB/s eta 0:00
    Installing build dependencies ... done
    Getting requirements to build wheel ... done
    Preparing metadata (pyproject.toml) ... done
Collecting pytools>=2011.2 (from pycuda)
  Downloading pytools-2025.1.5-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: platformdirs>=2.2.0 in /usr/local/lib/python
Requirement already satisfied: mako in /usr/lib/python3/dist-packages (from
Collecting siphash24>=1.6 (from pytools>=2011.2->pycuda)
  Downloading siphash24-1.7-cp311-cp311-manylinux_2_17_x86_64.manylinux2014
Requirement already satisfied: typing-extensions>=4.5 in /usr/local/lib/pyt
Downloading pytools-2025.1.5-py3-none-any.whl (93 kB)
    93.6/93.6 kB 9.6 MB/s eta 0:00:
Downloading siphash24-1.7-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x
    105.6/105.6 kB 10.3 MB/s eta 0:
Building wheels for collected packages: pycuda
5
77
6
9
22
77
88
99
99
7
  Building wheel for pycuda (pyproject.toml) ... done
  Created wheel for pycuda: filename=pycuda-2025.1-cp311-cp311-linux_x86_64
  Stored in directory: /root/.cache/pip/wheels/77/7e/6c/d2d1451ea6424cdc3d6
Successfully built pycuda
Installing collected packages: siphash24, pytools, pycuda
Successfully installed pycuda-2025.1 pytools-2025.1.5 siphash24-1.7
```

```
1 # Install pycuda
2 !pip install pycuda
3
4 import pycuda.driver as cuda
5 import pycuda.autoinit
6 import numpy as np
7 import cv2
8 from matplotlib import pyplot as plt
9 from pycuda.compiler import SourceModule
10
11 # Image preprocessing
12 img = cv2.imread("Lena.png")
13
14 # Check if the image was loaded successfully before proceeding
15 if img is None:
16     print("Error: Could not load image 'Lena.png'. Please ensure the file is
17 else:
```

```

18 img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
19 img = cv2.resize(img, (512, 512))
20 height, width, channels = img.shape
21 angle = 45
22
23 # Flatten image to 1D
24 img_flat = img.astype(np.uint8).ravel()
25 output_img = np.zeros_like(img_flat)
26
27 # Allocate device memory
28 input_gpu = cuda.mem_alloc(img_flat.nbytes)
29 output_gpu = cuda.mem_alloc(output_img.nbytes)
30
31 cuda.memcpy_htod(input_gpu, img_flat)
32
33 # Run the compiled binary - Note: This line seems redundant as you are already
34 # compiling and running the kernel through pycuda later. You might want to
35 # remove this unless it serves a specific purpose not evident from the context
36 # !./rotate
37
38 # Define and compile the CUDA kernel
39 cuda_code = """
40 #include <cuda_runtime.h>
41 #include <math.h> // Include math.h for cos and sin
42
43 __global__ void rotateKernel(unsigned char* input, unsigned char* output,
44                             int x = blockIdx.x * blockDim.x + threadIdx.x; // column
45                             int y = blockIdx.y * blockDim.y + threadIdx.y; // row
46
47                             float radians = angle * 3.14159265 / 180.0;
48                             int xc = width / 2;
49                             int yc = height / 2;
50
51                             if (x < width && y < height) {
52                                 int tx = x - xc;
53                                 int ty = y - yc;
54
55                                 // Ensure floating point calculations for rotation
56                                 float srcX_float = cos(radians) * tx + sin(radians) * ty + xc;
57                                 float srcY_float = -sin(radians) * tx + cos(radians) * ty + yc;
58
59                                 // Use nearest neighbor interpolation for simplicity
60                                 int srcX = round(srcX_float);
61                                 int srcY = round(srcY_float);
62
63                                 for (int c = 0; c < 3; c++) {
64                                     if (srcX >= 0 && srcX < width && srcY >= 0 && srcY < height)
65                                         output[(y * width + x) * 3 + c] = input[(srcY * width +
66                                         srcX) * 3 + c];
67                                     else {
68                                         output[(y * width + x) * 3 + c] = 255; // white background
69                                     }
70                                 }
71                             }
72 """
73
74 # Compile the CUDA kernel
75 cuda_compiler = cuda.compile(cuda_code)
76 rotate_kernel = cuda_compiler.get_function('rotateKernel')
77
78 # Run the kernel
79 rotate_kernel.prepare([input_gpu, output_gpu], [img_flat.nbytes, output_img.nbytes])
80 rotate_kernel.execute([1, 1], [width, height])
81
82 # Copy the output back to host memory
83 cuda.memcpy_dtoh(output_img, output_gpu)
84
85 # Reshape the output image
86 output_img = output_img.reshape((height, width, channels))
87
88 # Save the rotated image
89 cv2.imwrite('rotated_image.png', output_img)
90
91 # Print the shape of the rotated image
92 print("Rotated image shape:", output_img.shape)
93
94 # Close the image window
95 cv2.destroyAllWindows()
96
97 # End of script
98

```

```

69         }
70     }
71 }
72 """
73 mod = SourceModule(cuda_code)
74 rotateKernel = mod.get_function("rotateKernel")
75
76 # Launch kernel
77 block = (16, 16, 1)
78 grid = ((width + block[0] - 1) // block[0], (height + block[1] - 1) // b
79 rotateKernel(input_gpu, output_gpu, np.int32(width), np.int32(height), r
80
81 cuda.memcpy_dtoh(output_img, output_gpu)
82 rotated = output_img.reshape((height, width, 3))
83
84 plt.imshow(rotated)
85 plt.title("Rotated Image (CUDA)")
86 plt.axis('off')
87 plt.show()
88
89 # Free device memory
90 input_gpu.free()
91 output_gpu.free()

```


➡ Requirement already satisfied: pycuda in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: pytools>=2011.2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: platformdirs>=2.2.0 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: mako in /usr/lib/python3/dist-packages (from pytools>=2011.2)
Requirement already satisfied: siphash24>=1.6 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: typing-extensions>=4.5 in /usr/local/lib/python3.11/dist-packages

Rotated Image (CUDA)



```
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4 from pycuda.compiler import SourceModule
5
6 # Read the uploaded image
7 img = cv2.imread("Lena.png")
8
9 # Check if the image was loaded successfully
10 if img is None:
11     print("Error: Could not load image 'Lena.png'. Please ensure the file i
12 else:
13     # Rest of your code for processing the image
14     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
15     img = cv2.resize(img, (512, 512))
16     height, width, channels = img.shape
17     angle = 45
18
19     # Flatten image to 1D
20     img_flat = img.astype(np.uint8).ravel()
21     output_img = np.zeros_like(img_flat)
```

```

22
23 # Allocate device memory
24 input_gpu = cuda.mem_alloc(img_flat.nbytes)
25 output_gpu = cuda.mem_alloc(output_img.nbytes)
26
27 cuda.memcpy_htod(input_gpu, img_flat)
28
29 # Define and compile the CUDA kernel
30 cuda_code = """
31 #include <cuda_runtime.h>
32 #include <math.h> // Include math.h for cos and sin
33
34 __global__ void rotateKernel(unsigned char* input, unsigned char* output,
35     int x = blockIdx.x * blockDim.x + threadIdx.x; // column
36     int y = blockIdx.y * blockDim.y + threadIdx.y; // row
37
38     float radians = angle * 3.14159265 / 180.0;
39     int xc = width / 2;
40     int yc = height / 2;
41
42     if (x < width && y < height) {
43         int tx = x - xc;
44         int ty = y - yc;
45
46         // Ensure floating point calculations for rotation
47         float srcX_float = cos(radians) * tx + sin(radians) * ty + xc;
48         float srcY_float = -sin(radians) * tx + cos(radians) * ty + yc;
49
50         // Use nearest neighbor interpolation for simplicity
51         int srcX = round(srcX_float);
52         int srcY = round(srcY_float);
53
54         for (int c = 0; c < 3; c++) {
55             if (srcX >= 0 && srcX < width && srcY >= 0 && srcY < height)
56                 output[(y * width + x) * 3 + c] = input[(srcY * width + srcX) * 3 + c];
57             else {
58                 output[(y * width + x) * 3 + c] = 255; // white background
59             }
60         }
61     }
62 }
63 """
64 mod = SourceModule(cuda_code)
65 rotateKernel = mod.get_function("rotateKernel")
66
67 # Launch kernel
68 block = (16, 16, 1)
69 grid = ((width + block[0] - 1) // block[0], (height + block[1] - 1) // block[1], 1)
70 rotateKernel(input_gpu, output_gpu, np.int32(width), np.int32(height), grid, block)
71
72 cuda.memcpy_dtoh(output_img, output_gpu)

```

```
73     rotated = output_img.reshape((height, width, 3))
74
75     plt.imshow(rotated)
76     plt.title("Rotated Image (CUDA)")
77     plt.axis('off')
78     plt.show()
79
80     # Free device memory
81     input_gpu.free()
82     output_gpu.free()
```

 `/usr/local/lib/python3.11/dist-packages/google/colab/_variable_inspector.py`
`globals().clear()`

Rotated Image (CUDA)



1 Start coding or [generate](#) with AI.

