**Polarization Angle Computation:**

To compute the polarization angle from a camera with pixels corresponding to polarization orientations of 90 o, 0 o, 45 o, and135 o, we start by computing the Stokes parameters:

S1=I0 - I90

S2 =I45 - I135

Where:

* I0​: Intensity for 0 o polarization.
* I90: Intensity for 90 o polarization.
* I45 ​: Intensity for 45 o polarization.
* I135: Intensity for 135 o polarization.

The polarization angle, θ is then calculated as:

θ = 0.5 \* arctan2(S2, S1)

**Normalize the Polarization Angle:**

Arctan2 (from CUDA math library “math\_function.h”) computes the angle in the correct quadrant, in a range [-π, π]. Since the output image is an 8-bit grayscale, we need to rescale the result before displaying it. The computed θ lies in the range [−π/2,π/2]. To map this range to pixel intensities [0,255] we normalize as follows:

I=(θ+π/2)/π \* 255

* A black pixel corresponds to a polarization angle of −π/2.
* A white pixel corresponds to a polarization angle of π/2.
* Shades of grey represent angles in between.

**Results Interpretation:**

The example demonstrates the computation with two images taken using a polarized camera:

1. **First Image**:
   * The camera is straight, with the laptop's screen polarized at 90 o and the second screen at 0 o.
   * The laptop appears either black (−π/2) or white (π/2).
   * The second screen appears grey (0).
2. **Second Image**:
   * The camera is rotated by 45 o.
   * The laptop's light now results in a π/4 polarization, mapped to light grey.
   * The second screen's light results in a polarization of −π/4 mapped to dark grey.

This illustrates that the code executes as intended.

**Comments:**

* **Image Loading and GPU Processing**:
  + The code loads images from a specified folder and streams them to a GPU kernel for processing.
  + Example images are 2544×2048 in resolution.
* **Limitations**:
  + The dimensions of the images are constrained by GPU memory. For example, larger images may not fit in GPU memory if there isn’t enough space for input and output buffers.
  + The number of images processed simultaneously is limited by the GPU’s memory capacity too.
* **Grid Dimensions**:
  + The grid dimensions for the kernel execution can be easily adjusted in the code to optimize performance for different image resolutions or hardware configurations.
* **Example of code run:**
  + 