Goals & Motivations

- The iris pattern is considered to be amongst the most reliable for high confidence identification
  - Identification in enormous (national/global) databases
  - Requires 'many-to-many' comparisons; makes even the simplest distance metric computationally expensive
  - Fortunately: “embarrassingly parallel”
  - Distance matrix elements independent

- Standard solution: large CPU-based clusters
  - CPUs designed for general purpose, sequential tasks
  - Non-optimal w.r.t. power, cooling, footprint size & cost

- Better solution: acceleration with off-the-shelf GPUs
  - Market demand for realistic 3D games has evolved the GPU into a highly parallel, multithreaded, many-core processor of tremendous power

Compute Unified Device Architecture

- Prior to introduction of CUDA (API/architecture) in 2006 General-Purpose Computation on Graphics Hardware (GPGPU) was hard work:
  - Express problems in terms of graphics primitives
- CUDA enables expression of programs in C, C++, Fortran and other high level languages
- Heterogeneous: execute “kernels” on GPU
  - Section of device code that executes in parallel on GPU
  - Unique thread ID, program counter, registers and private local memory
  - Shared global device memory for communication
  - Hierarchy of memory types
  - SIMT model v. SIMD model

How Iris Recognition Works

The most widely employed procedure for feature extraction, pioneered by John Daugman, uses the phase response of 2D Gabor wavelets.

In addition to being a component of Daugman’s algorithm, the template matching process (select the minimum fractional HD over a range of bitwise horizontal circular shifts) is a common final step of other iris recognition routines.

Implementation Overview

- Naïve implementation: exploit fine grain parallelism
  - Kernel below computes pairwise HD between a probe template and a gallery template determined by a unique two dimensional thread ID

- Optimizations: kernel is extremely memory bound
  - Exploit memory hierarchy ➔ shared memory and texture cache reduce global memory bandwidth usage/latency
  - Rotation invariance:
    - Perform shifts in shared memory and reduce within thread block to find minimal HD for a given probe

Results

- Achieved rates of 44 million iris template matches/s without rotation invariance. With tolerance to head tilt, 4.2 million matches/s (template size 2048 bits)
- Show a 14X speedup over optimized CPU implementation
- In contrast to other published work, our parallel implementation incorporates shifting for rotation invariance