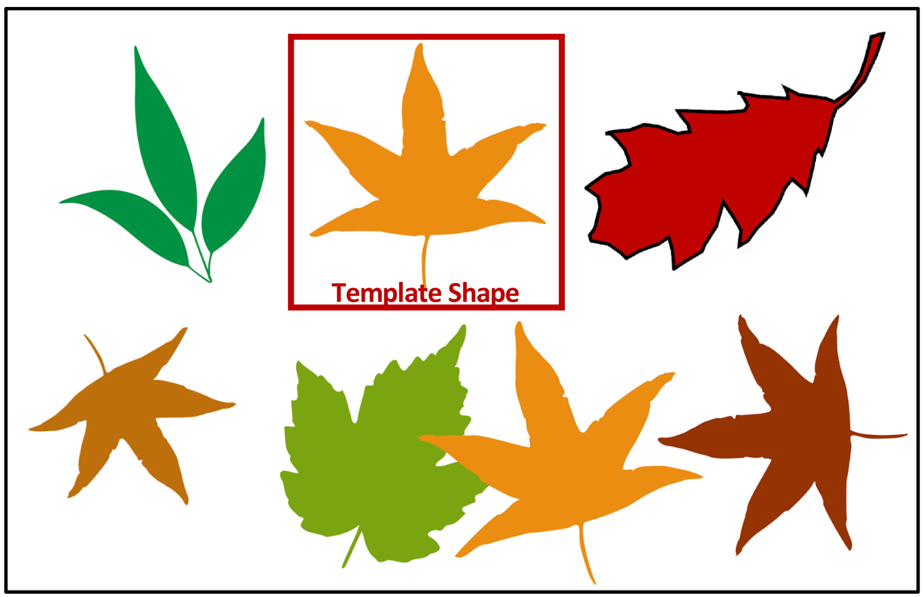
Generalized Hough Transform on GPU

Yijie Chen (yijieche), Yiliu Xu (yiliux)

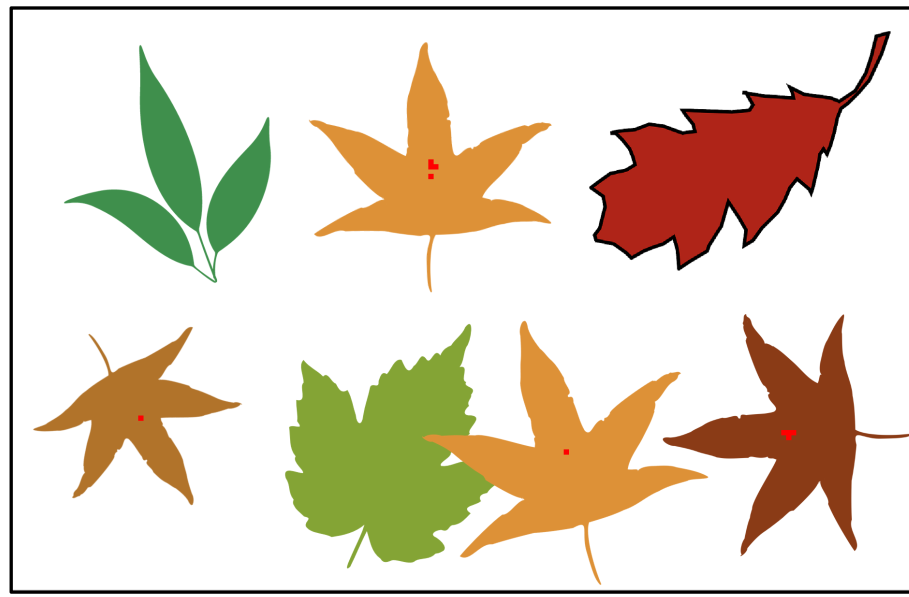
We implemented **Generalized Hough Transform** on **GPU** using **CUDA** and **Thrust**

Feature: Detect **arbitrary** target shape under **rotation** and **scaling**

Acceleration Result: **433x** overall speedup on Intel i7-9700 CPU, RTX 2080 GPU



Sample Input

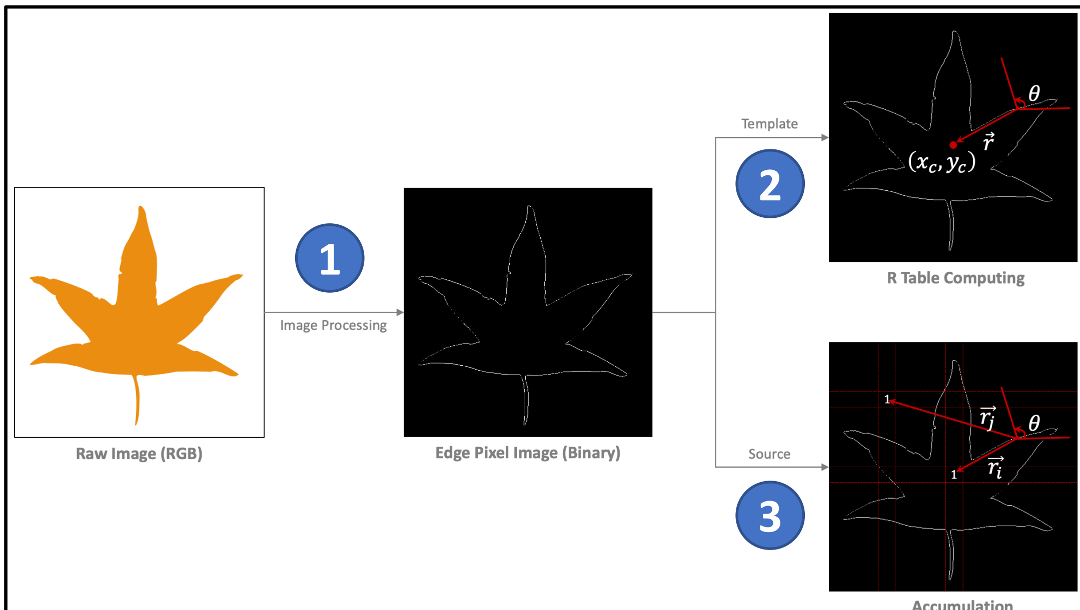


Sample Output

General Hough Transform Pipeline

Carnegie Mellon University

1. Apply derivative filter to **detect edge pixels**
2. (Template) Encode all edge points in **R Table**
3. (Source) Accumulate vote of each edge pixel into a **4D** accumulator matrix (x_c, y_c, s, θ)
4. Find the shape centers with high votes



Equations for accumulation (2D):

$$x_c = x + r * \cos(\alpha)$$
$$y_c = y + r * \sin(\alpha)$$

Algorithm 1 Calculating Accumulator Loop

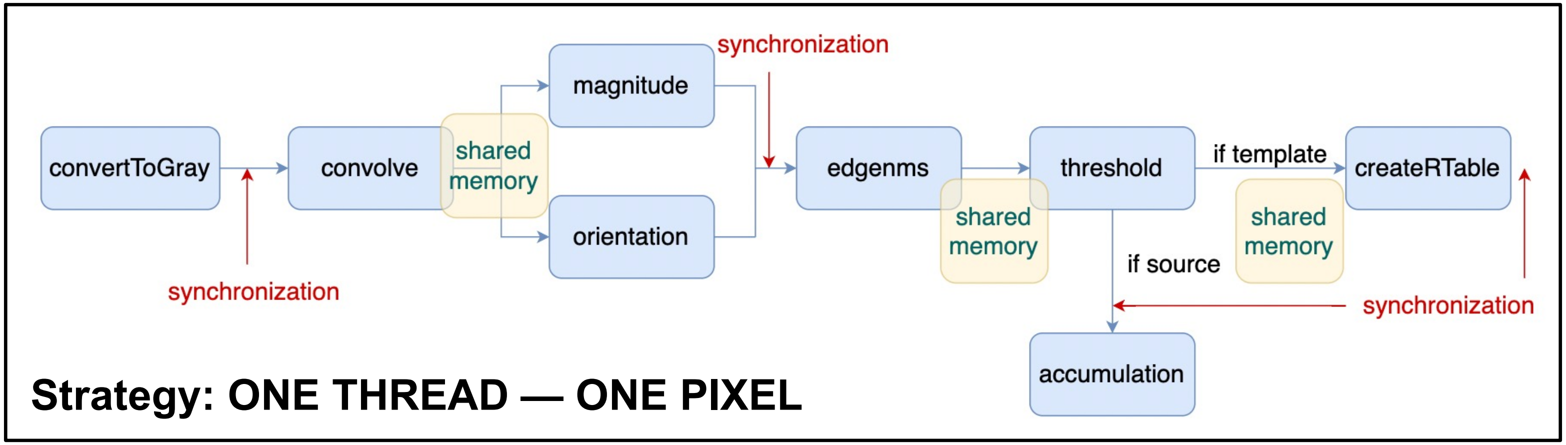
for pixel (x, y) in image **do**
  if (x, y) is an edge pixel **then**
    ϕ = orientation(x, y)
    for θ in $[0, 2\pi)$ **do**
      entries = RTable$(\phi - \theta)$
      for entry in entries **do**
        r = entry.r, α = entry.α
        for s in [minScale, maxScale] **do**

Accumulation is expected to benefit the most from CUDA parallelization.

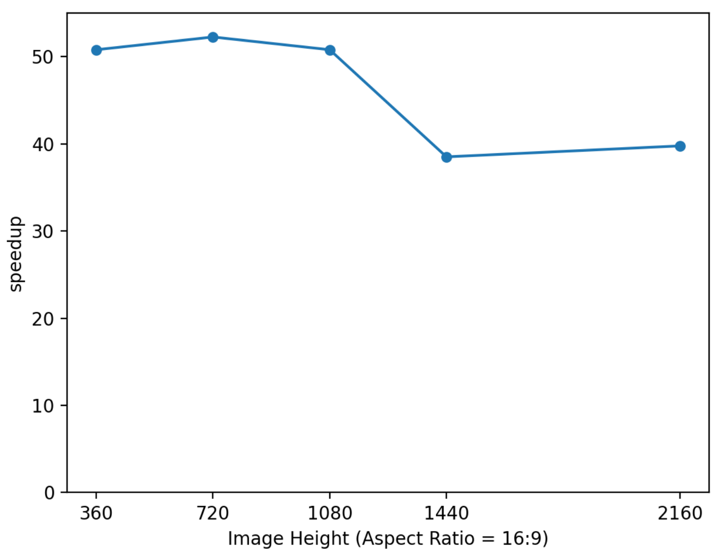
Parallel **Image Processing**

Carnegie Mellon University

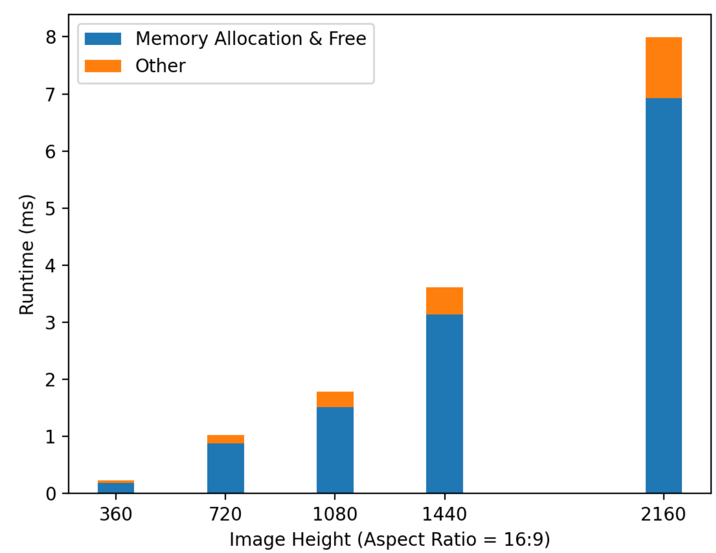
Parallel Image Processing Pipeline



40 - 50x Speedup



Overhead in Memory Allocation & Free



Parallel **R Table Processing**

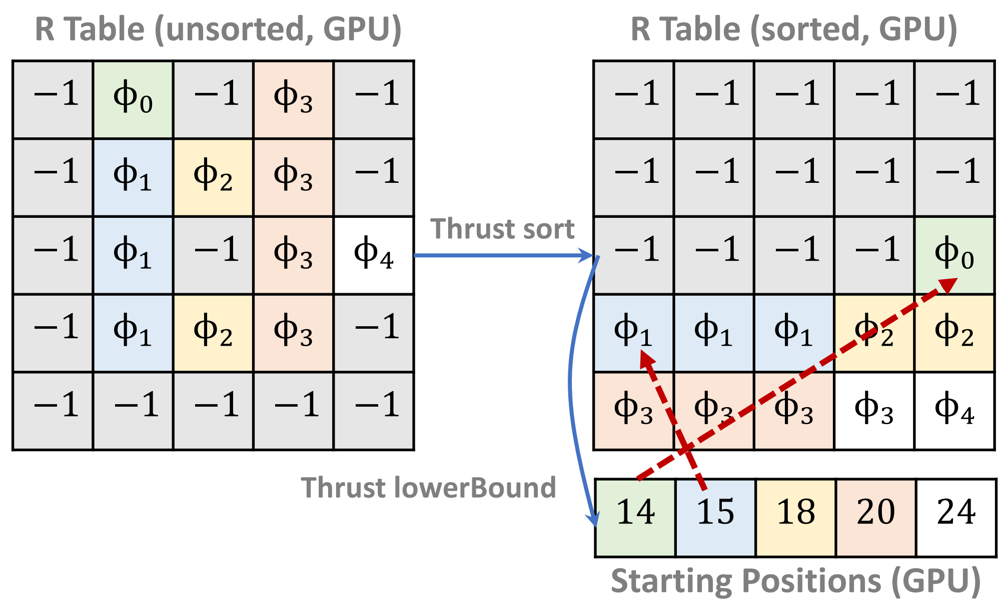
Carnegie Mellon University

R Table in **CPU** memory

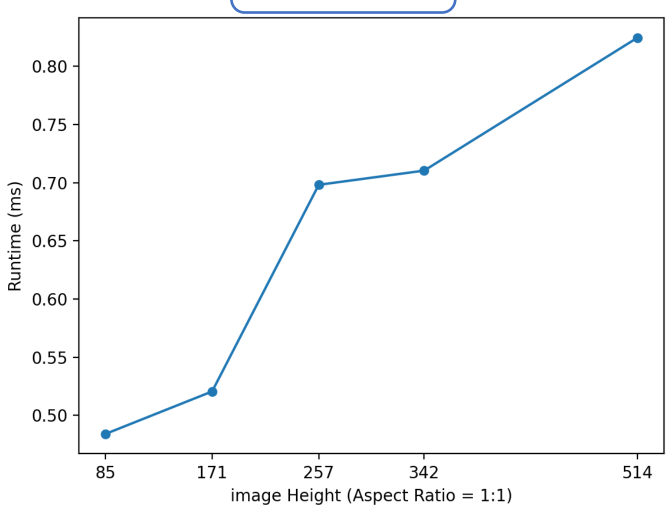
i	ϕ_i	R_{ϕ_i}
1	0	$(r_{11}, \alpha_{11}), \ldots (r_{1n}, \alpha_{1n})$
2	$\Delta\phi$	$(r_{21}, \alpha_{21}), \ldots (r_{2n}, \alpha_{2n})$
3	$2\Delta\phi$	$(r_{31}, \alpha_{31}), \ldots (r_{3n}, \alpha_{3n})$
…	…	…

- Each slice is with dynamic size
- Troublesome for GPU implementation

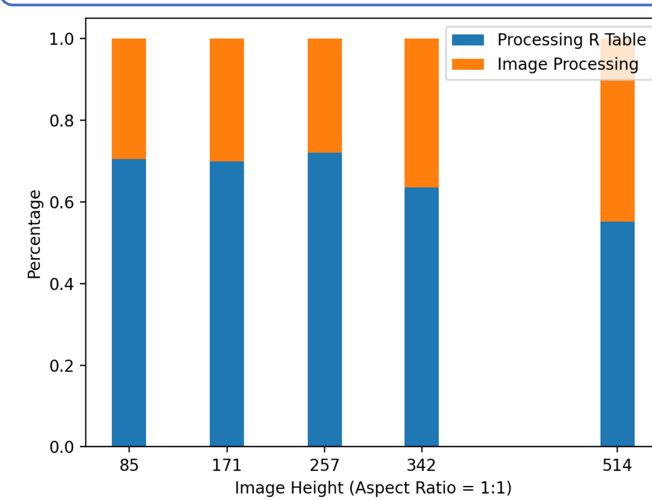
R Table in **GPU** memory



Runtime



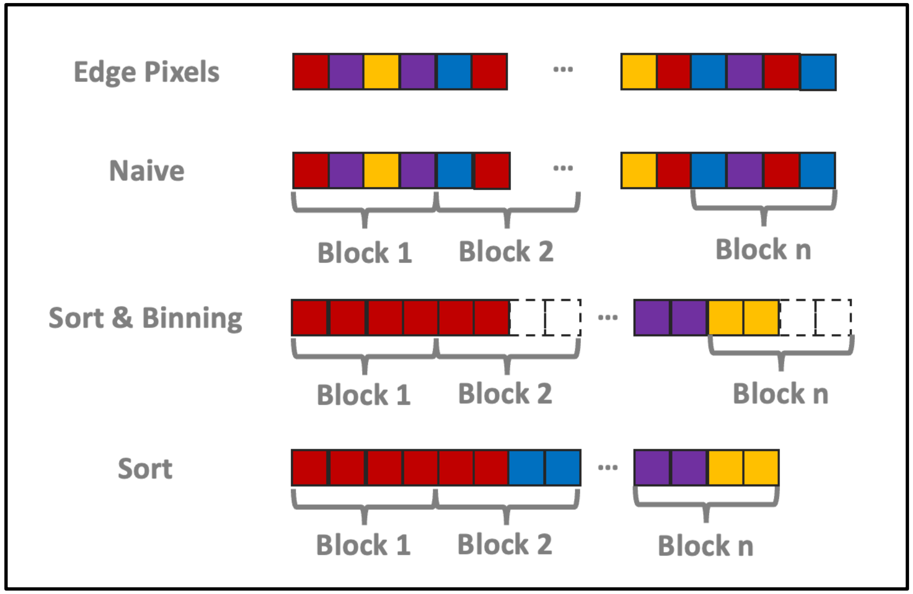
Compared with Image Processing



Parallel **Accumulation**

Carnegie Mellon University

Design Alternative 1: Map Edge Pixels to CUDA Blocks



Design Alternative 2: Kernel Dimension

1D Kernel

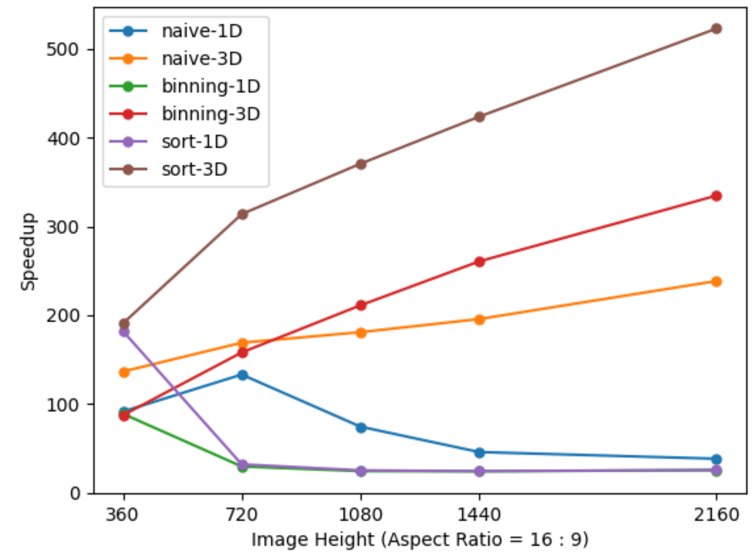
*for each **theta** {*
*for each **s** {*
*for each **rEntry**{…}}}*

3D Kernel

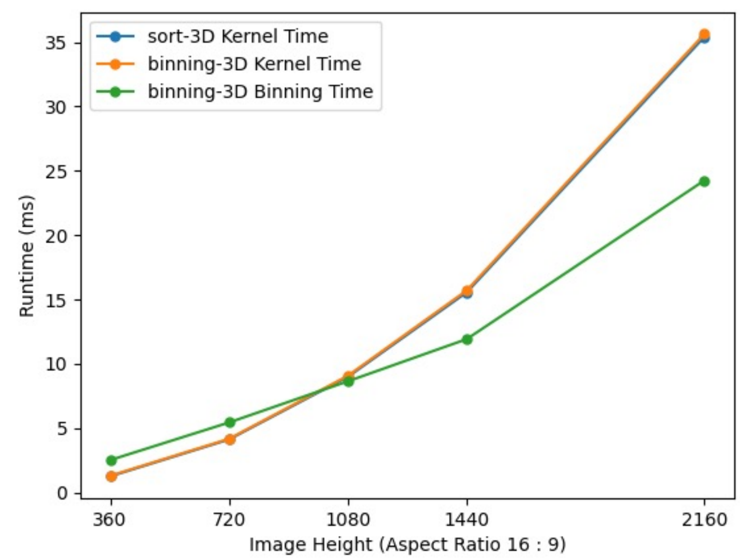
(One block share **theta** and **s**)

*for each **rEntry** {…}*

Speedup



Binning is expensive



Parallel **Accumulation**

Carnegie Mellon University

Design Alternative 3: Shared Memory

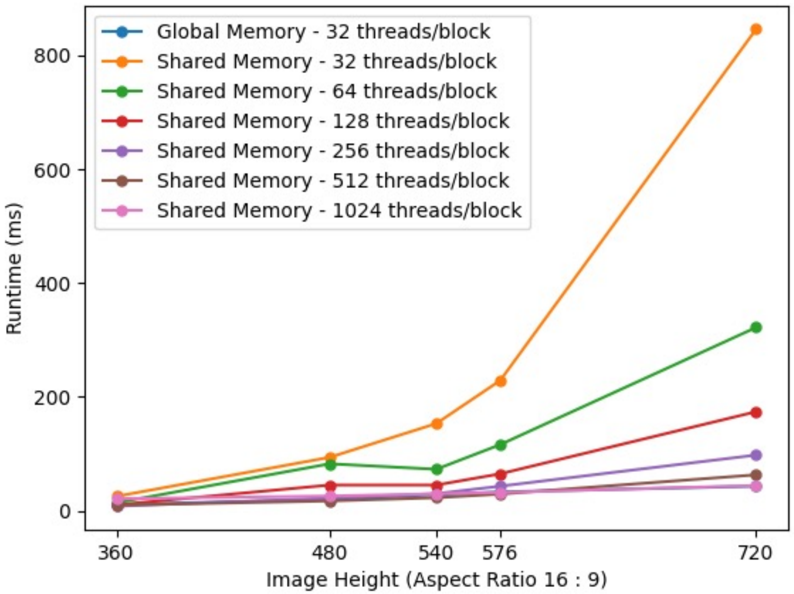
Global Memory

- Vote directly to **4D** accumulator (AtomicAdd)

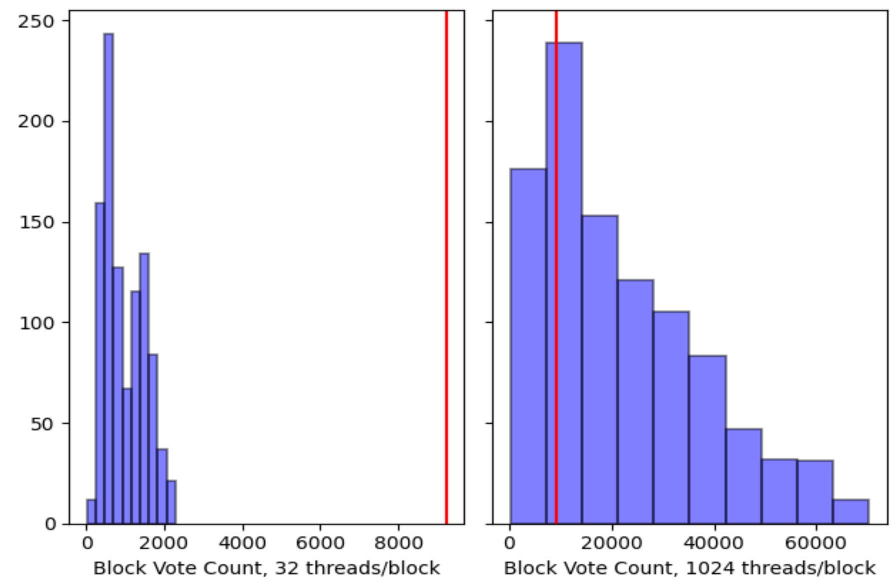
Shared Memory

- Create a shared **2D** accumulator slice / block, first vote **within** block, then write back

Performance



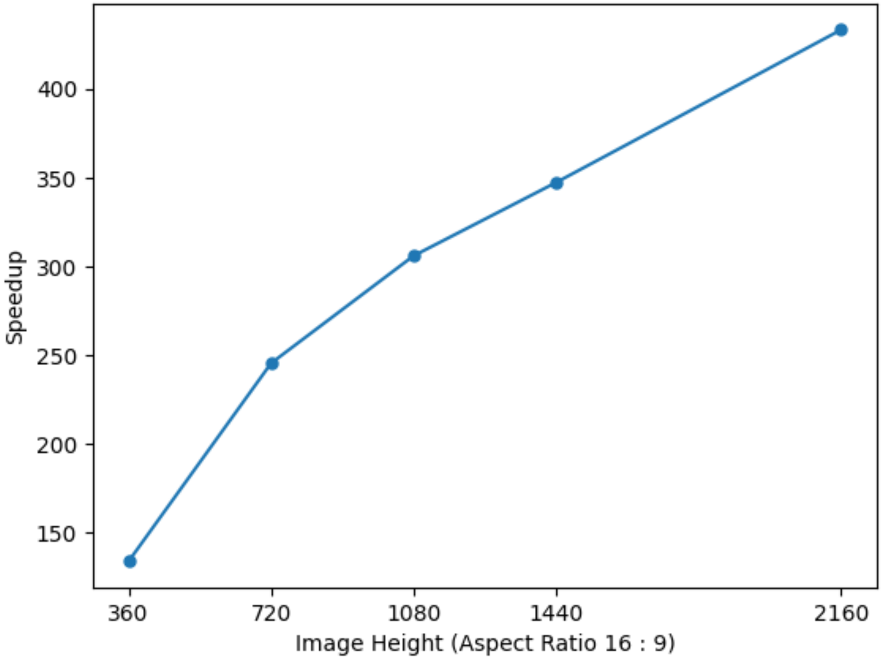
Overhead Analysis



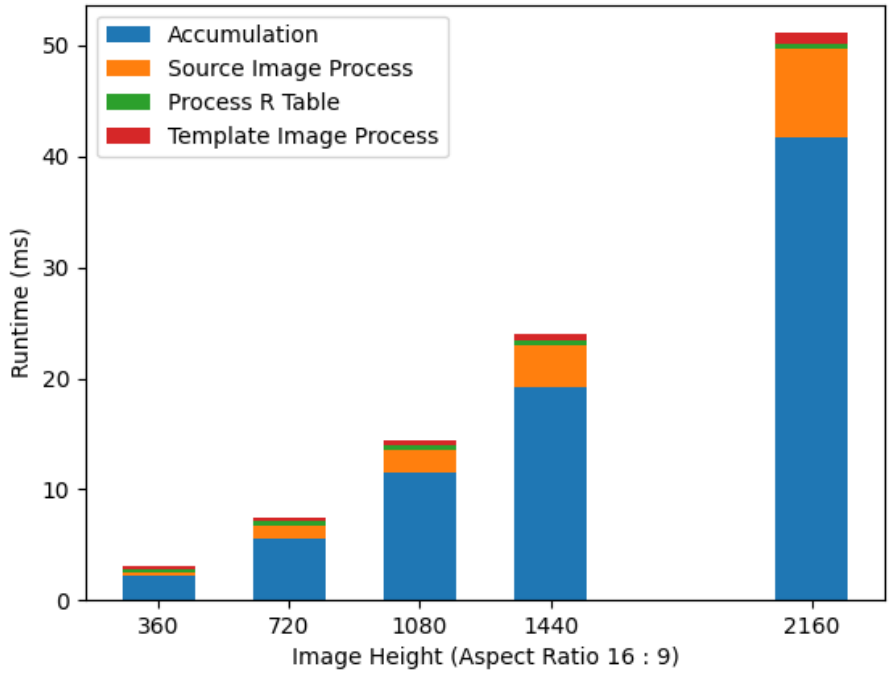
Challenges using shared memory:
1. Bounded by shared memory allocation limit (48KB)
2. Need to more efficiently store and update voting data in memory

Overall Speedup

Speedup w.r.t Image Size



Runtime Break Down



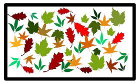
Accumulation is the most critical step to achieve a high speedup.

As the image size grows, the algorithm benefits more from parallelization.

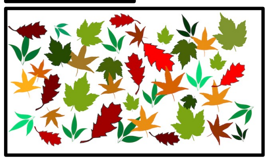
Sample Test Images

Different image sizes

360p

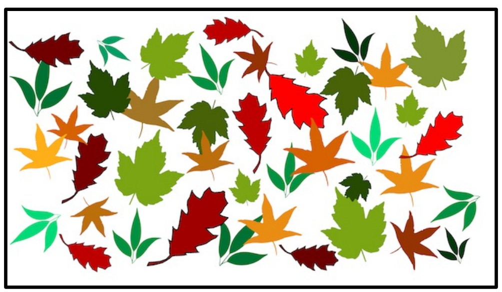


720p

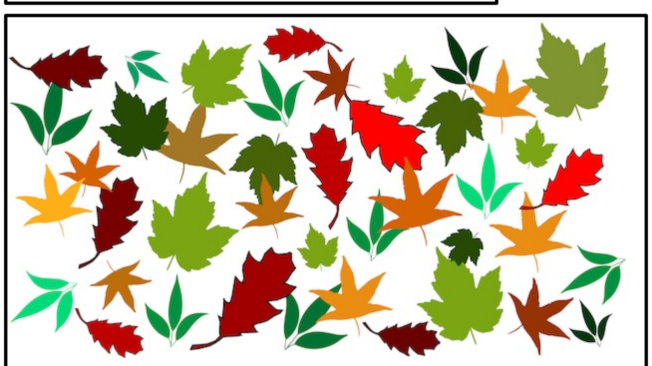


1080p

2K



4K



Different shape density