SHARP

A shape recognition system and its parallel implementation

Agenda

- Theoretical aspects of Hough transformation
- Algorithm description
- Implementation considerations
- Expected results

Theoretical aspects of Hough Transformation

Shape recognition problem

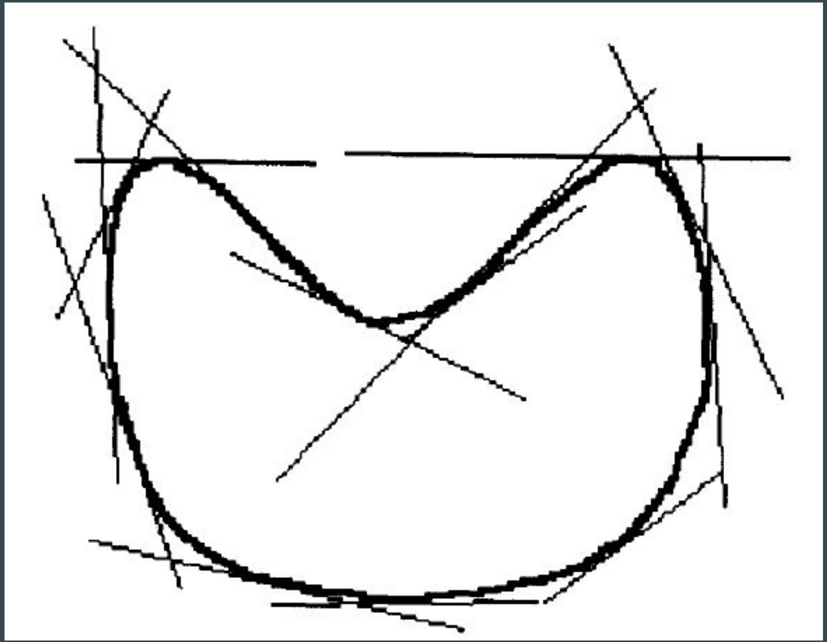
- Fundamental aspect problem of computer vision.
- Defined as the problem of determining whether the test image contains one of the available reference shapes or not.
- SHARP algorithm takes into account the problem of shape recognition in **binary images.**

Shape representation: Hough transformation (1)

- An arbitrary shape can be considered as composed of small tangent straight line segments.
- In cartesian coordinates, a line is commonly represented by the equation

$$y = mx + q$$

- The Hough transformation represents lines in polar coordinates.



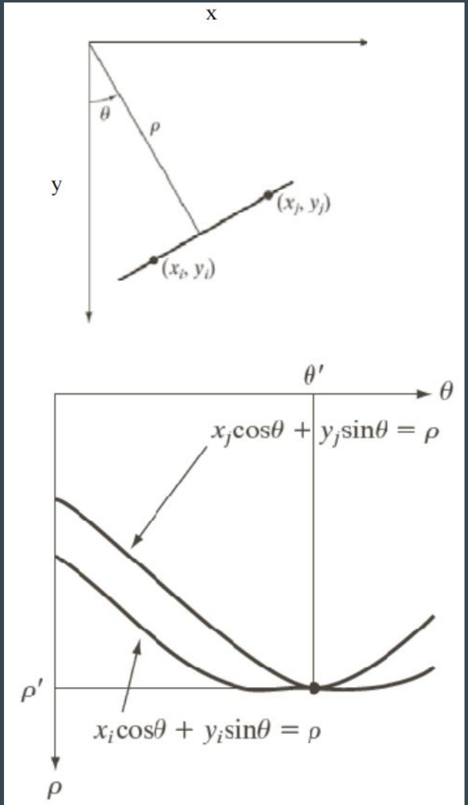
Shape representation: Hough transformation (2)

- The θ dimension is given by the angle of the normal of the line.
- The r dimension is the distance of the line from the origin.

$$r = x\cos\theta + y\sin\theta$$

- Value of θ is restricted to the interval $[0, \pi]$ and r is restricted to the interval

$$[\overline{-n}(\cos 45 + \sin 45), \overline{n}(\cos 45 + \sin 45)]$$



Shape representation: Hough transformation (3)

- Representing a line back in the cartesian plane, after the considerations made:

$$y = -\frac{\cos\theta}{\sin\theta}x + \frac{r}{\sin\theta}$$

- In general, for each point ***(x_0, y_0)***, we can define a family of lines that goes through that point as:

$$r = x_0 \cos\theta + y_0 \sin\theta$$

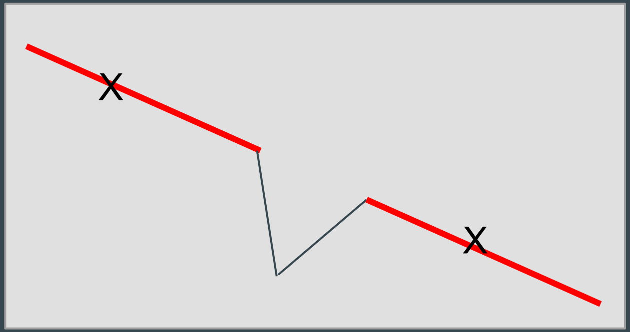
- This means that each pair *(θ, r)* represents each line that passes through *(x_0, y_0)*

Key point

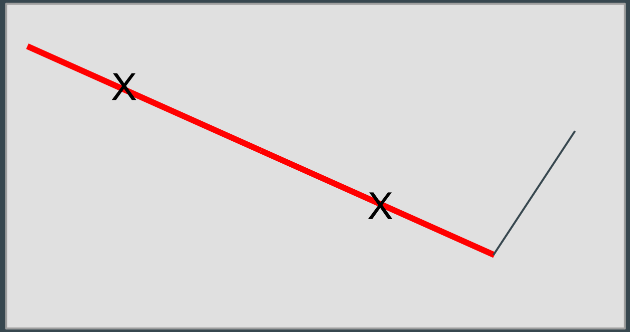
Two curves intersecting in the Hough space determine **a (θ, r) pair** , i.e. a straight line, on which **the two points** that generated those curves **lie on**.

Hough space (1)

- **Discretized** and represented as a two-dimensional accumulator array.
- In general, Hough transform implementations accumulate in each array cell the number of pixels lying on the same line in the cartesian plane.
- This does not preserve the information of which pixel belongs to a particular line.



Same as



Hough space (2)

- SHARP prefers to adopt a modified version known as **Straight-line Hough Transform (SLHT)**.
- According to this variant, pixels mapped to the same *(θ, r)* are grouped in *lines* of adjacent points.

- This way, we can identify **dominant segments** based on their **length** instead of the number of points that eventually lie on the same line.

Algorithm description

Algorithm outline

In order to recognize a shape, three macro-steps are required:

1. Computation of SLHT for the test image.
2. Computation of the STIRS signature of the test image.
3. Matching the test signature with that of the reference shapes.

STIRS Signature

- Distances between pairs of parallel tangential lines to a curve C.
- Basically a **feature** of a shape.

Has the following properties:

- It is invariant to the translation of the shape.
- Rotation of the shape corresponds to a circular-shift of its signature in the (θ, r) space.
- If the shape is scaled by a factor S, then the signature is also scaled by the same factor.

Therefore named: **scalable translation-invariant rotation-to-shifting** (STIRS) signature.

Parallelization scheme (1)

- Assuming a distributed-memory, multiple instruction, multiple data (MIMD) computational model.
- The SLHT array is divided over the θ space into $\boldsymbol{p}$ partitions, number of processors.
- Each processor $\boldsymbol{i}$ computes the SLHT and the (partial) STIRS signature for angles in range

$$A_i = [i\frac{m_\theta}{p}, (i+1)\frac{m_\theta}{p} - 1]$$

Parallelization scheme (2)

- Each processor ***i*** also applies the matching algorithm to the test image and the reference image for the angles in range A_i for all the m_θ orientations of the reference shape.
- The matching procedure produces a ***matching score***, which is *partial* for the range of angles taken into account.
- A merging step is carried out by adding the partial scores, using a **binary-tree reduction procedure**

SHARP Algorithm in details (2)

```
procedure partial_slht (i)
/* i is the processor id */
begin
        Θ^i_min = i*δ_Θ*m_Θ/p
        Θ^i_max = (i+1)*δ_Θ*m_Θ/p - 1
        for x = 0 to n-1 do
                for y = 0 to n-1 do
                        if pixel[x][y] = 1 then
                                for Θ = Θ^i_min to Θ^i_max step δ_Θ do
                                begin
                                        t = (Θ - Θ^i_min) / δ_Θ
                                        r = x * cos Θ + y * sin Θ
                                        update/append slht^i[t][r].lines
                                end
end
```

Figure 3 *pixel* is an $n \times n$ array containing the test shape. $slht^i$ contains line segments

Complexity

$O(n^2 l * m_\theta / p)$

SHARP Algorithm in details (3)

```
procedure partial_signature (i)
/* i is the processor id */
begin
        for Θ = 0 to m_Θ/p - 1  do
        begin
                for r = 0 to m_r - 1 do
                        for each line in slht^i[Θ][r] do
                                if length of line > threshold then
                                        A^i[Θ][r] = 1
                for r = 0 to m_r - 1 do
                        if A^i[Θ][r] = 1 then
                                for r_1 = r + 1 to m_r - 1 do
                                        if A^i[Θ][r_1] = 1 then
                                                D^i[Θ][r_1-r] = 1
        end
end
```

Figure 4 Computing the signature. *Threshold* is the line length threshold

Complexity

$O(m_r^2 * m_\theta/p)$

SHARP Algorithm in details (4)

```
procedure partial_match(i)
/* i is the processor id */
begin
        for Θ_1 = 0 to m_Θ - 1 do
        begin
                match = approx = miss = 0
                for Θ_2 = 0 to m_Θ/p - 1 do
                begin
                        t = (Θ_1 + Θ_2) mod m_Θ
                        for r = 0 to m_r - 1 do
                                if D_r[t][r] = 1 then
                                        if D^i_t[Θ_2][r] = 1 then
                                                match = match + 1
                                        else if D^i_t[Θ_2][r ± 1] = 1 then
                                                approx = approx + 1
                                        else miss = miss + 1
                end
                score^i[Θ_1] = match + approx/2 - miss
        end
end
```

Figure 5 The array *score*^{i} contains the matching score for the *i*th processor

Complexity

$O(m_r * m_\theta^2/p)$

SHARP Algorithm in details (5)

```
procedure participate_in_add(i)
/* i is the processor id */
begin
        for k = 0 to log_2 p - 1 do
                if i ≥ 2^k - 1 then
                        if (i mod 2^(k+1) = 2^k - 1) then
                                send Score^i to processor i + 2^k
                        else if (i mod 2^(k+1) = 2^(k+1) - 1) then
                        begin
                                receive Score from processor i - 2^k
                                update local Score^i
                        end
end
```

Figure 6 Procedure to add partial scores available in each processor

Complexity

$$O(m_\theta * \log_2 p + t_{comm})$$

SHARP Algorithm in details (1)

```
procedure SHARP (p, i, test_shape, reference_shapes)
/* p is the number of processors and
   i is the processor id */
begin
        read the test shape into pixel array;
        compute partial_slht;
        compute partial_signature;
        for each reference_shape do
        begin
                read reference signature;
                perform partial_match;
                participate_in_add;   (* Add partial scores *)
                if i = p - 1 then
                        find peak in matching score;
                synchronize;
        end
end
```

Figure 2 The SHARP algorithm

Implementation considerations

Input data

Generating input shapes (both test and reference)

- Manually drawn
- Perform **edge detection** on samples (e.g. with Canny algorithm) to build a binary image. We could exploit OpenCV library.

Communication among threads

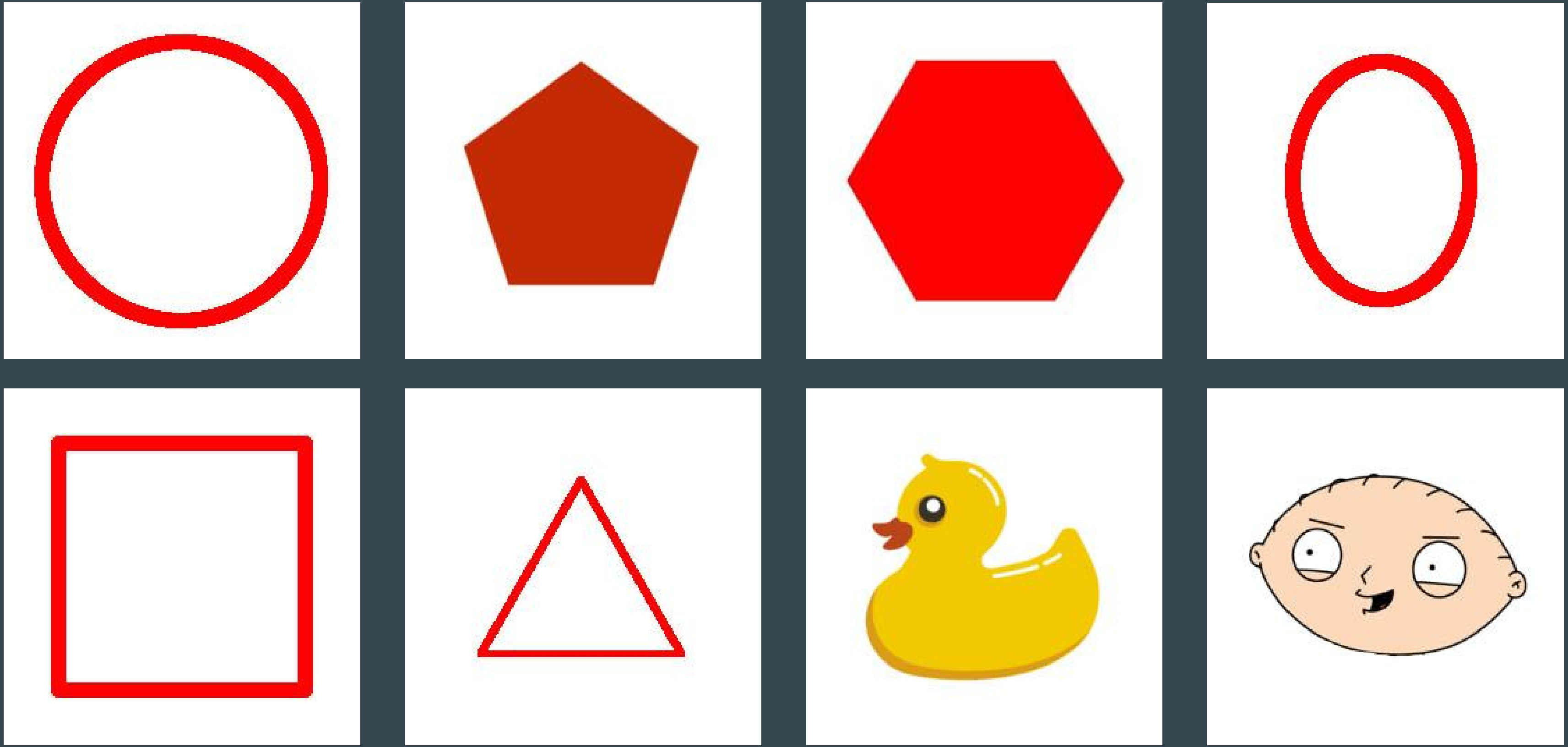
- Given the adoption of OpenMP framework, we can exploit locking APIs on a mutex for each processor, as well as a pointer to the *local* data structure to merge for each processor. Like a std::unique_ptr so that we enforce use of efficient move semantics.

Parameters

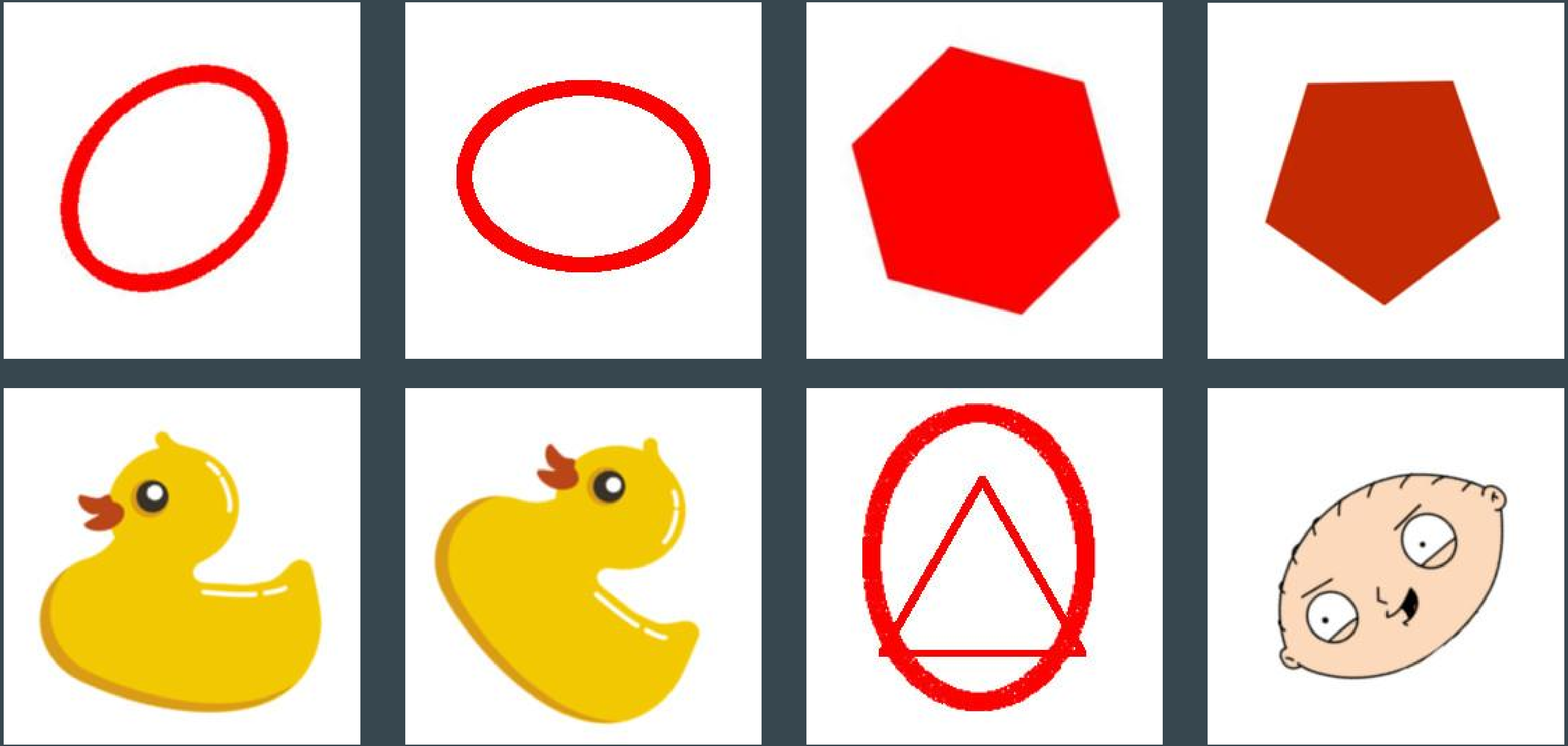
The SHARP paper suggests the following parameters, as they are those showed in their results:

1. **Shape size**: n = 256
2. $0 \leq \boldsymbol{\theta} \leq \pi$
3. $-363 \leq \boldsymbol{r} \leq 363$
4. $\boldsymbol{\delta_\theta} = 5°$ i.e. $\boldsymbol{m_\theta} = 37$
5. $\boldsymbol{\delta_r} = 1$ i.e. $\boldsymbol{m_r} = 727$
6. **Line length threshold** = 2.0

Reference shapes



Test shapes



Expected results

Ideal speedup of SHARP algorithm

Parallel time

$$T_{SHARP} = O\left(\frac{n^2 l m_\theta}{p}\right) + O\left(\frac{m_r^2 m_\theta}{p}\right) + O\left(\frac{N m_r l m_\theta^2}{p}\right) + O\left(m_\theta log_2 p\right) + t_{comm}$$

Sequential time

$$T_{SEQ} = O\left(n^2 l m_\theta\right) + O\left(m_r^2 m_\theta\right) + O\left(N m_r l m_\theta^2\right)$$

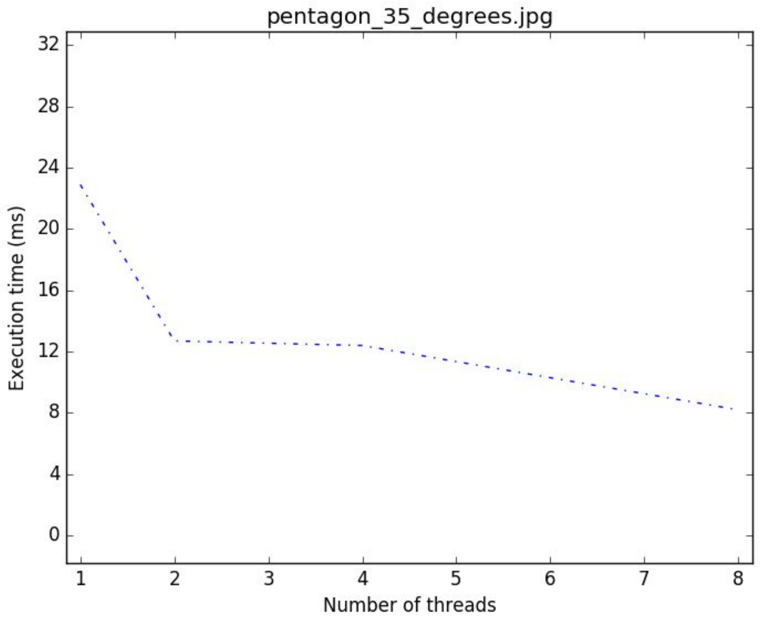
Speedup

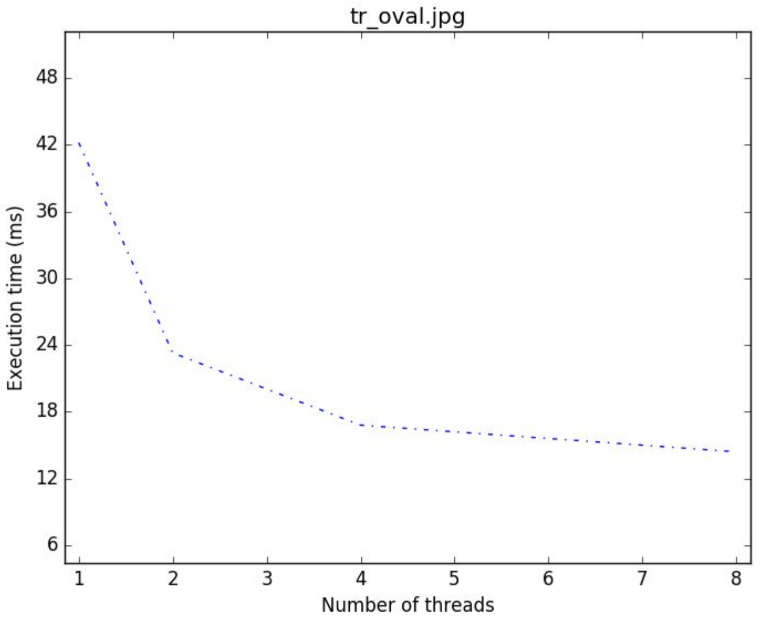
$$Speedup = \frac{T_{SEQ}}{T_{SHARP}} \approx p$$

Experimental results

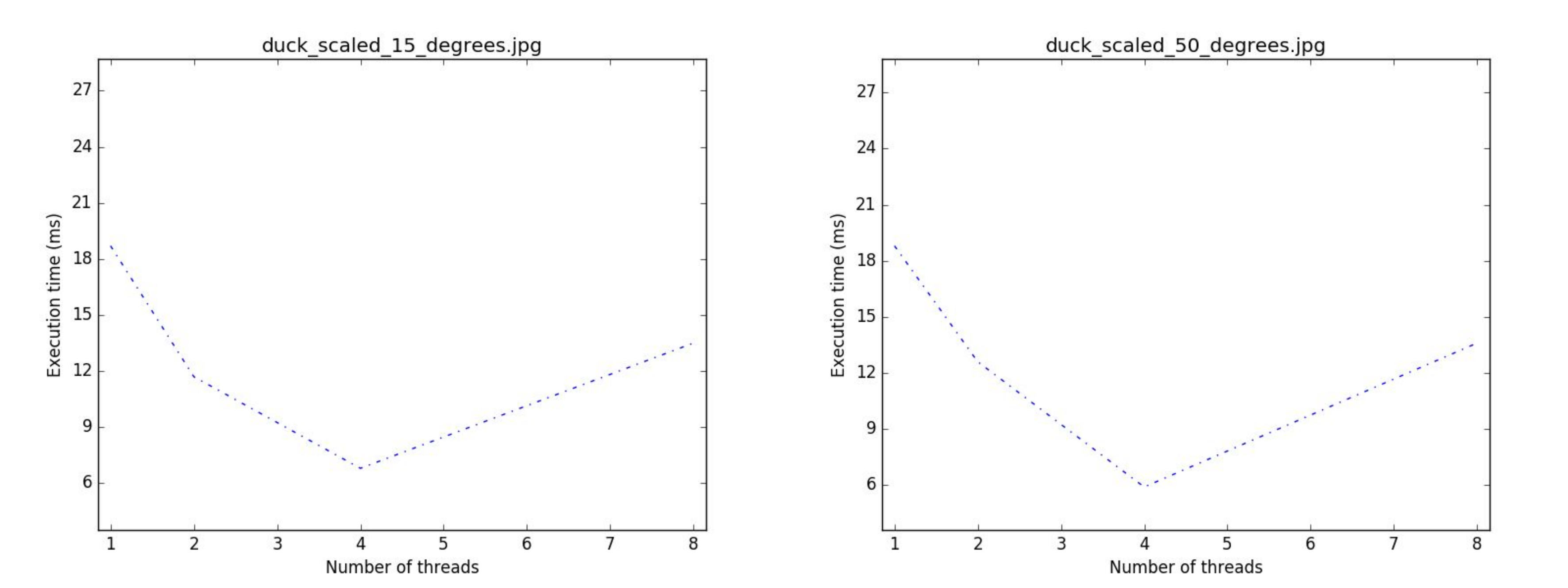
Run time [ms] with # of threads:	1	2	4	8
pentagon_35_degrees.jpg	22.90	12.70	12.40	8.20
tr_oval.jpg	42.20	23.30	16.80	14.40
duck_scaled_15_degrees.jpg	18.70	11.70	6.80	13.50
duck_scaled_50_degrees.jpg	18.80	12.60	5.90	13.60
oval_45_degrees.jpg	28.00	15.20	10.90	9.80
oval_90_degrees.jpg	24.20	12.80	10.40	8.20
stewie_135_degrees.jpg	49.60	27.70	27.50	16.20
hexagon_135_degrees.jpg	23.10	13.90	11.60	10.70

Experimental results

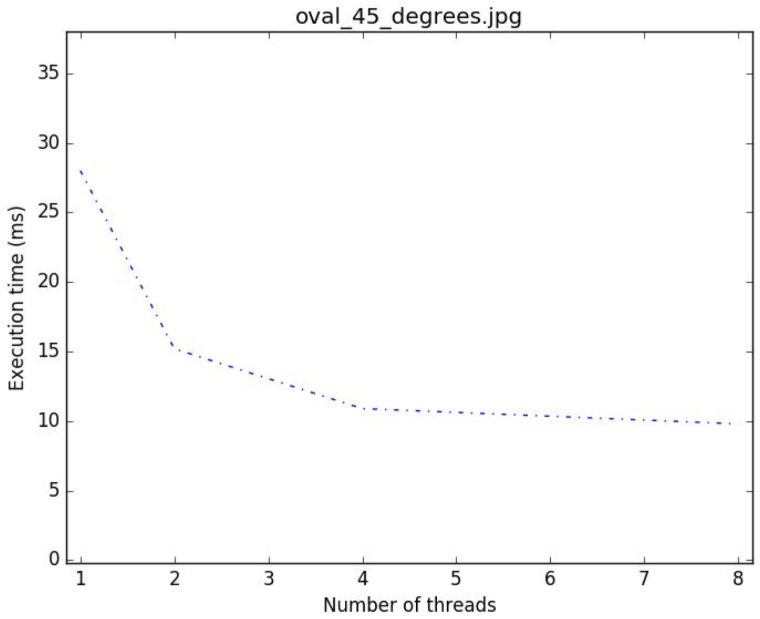


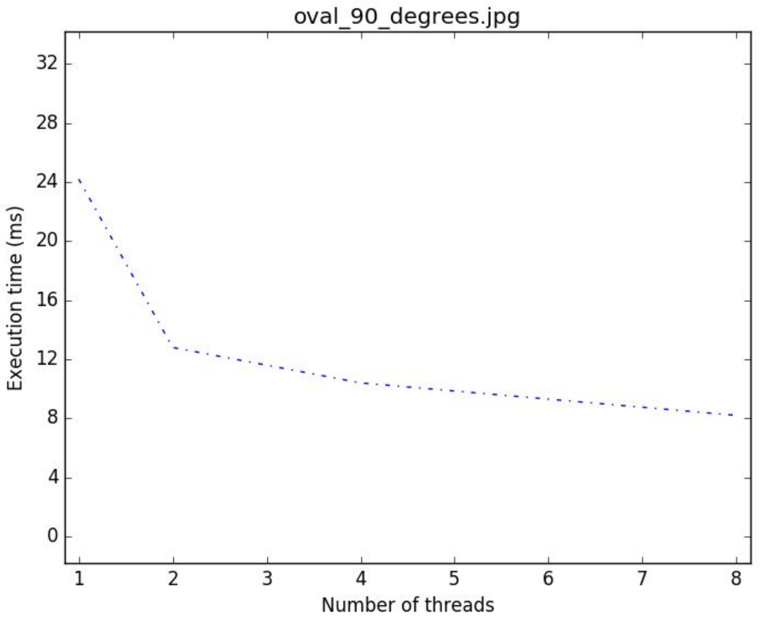


Experimental results



Experimental results





Experimental results