# GPU Online Tracking

*PANDA Meeting Goa*

12 March 2013, Andreas Herten, IKP, Forschungszentrum Jülich

# Intro & Outline

- ***GPGPU*** *= General-Purpose Graphics Processing Units*
  - *CPU → GPU*

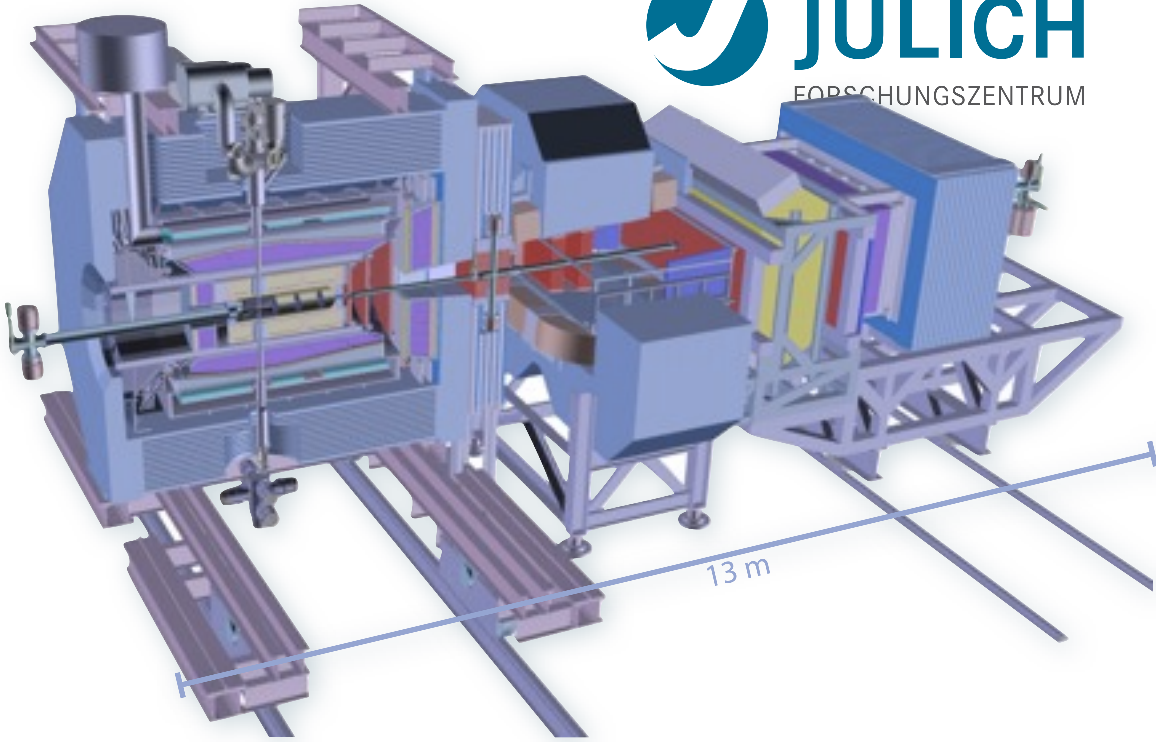
# Intro & Outline

- ***GPGPU*** *= General-Purpose Graphics Processing Units*
  - *CPU → GPU*

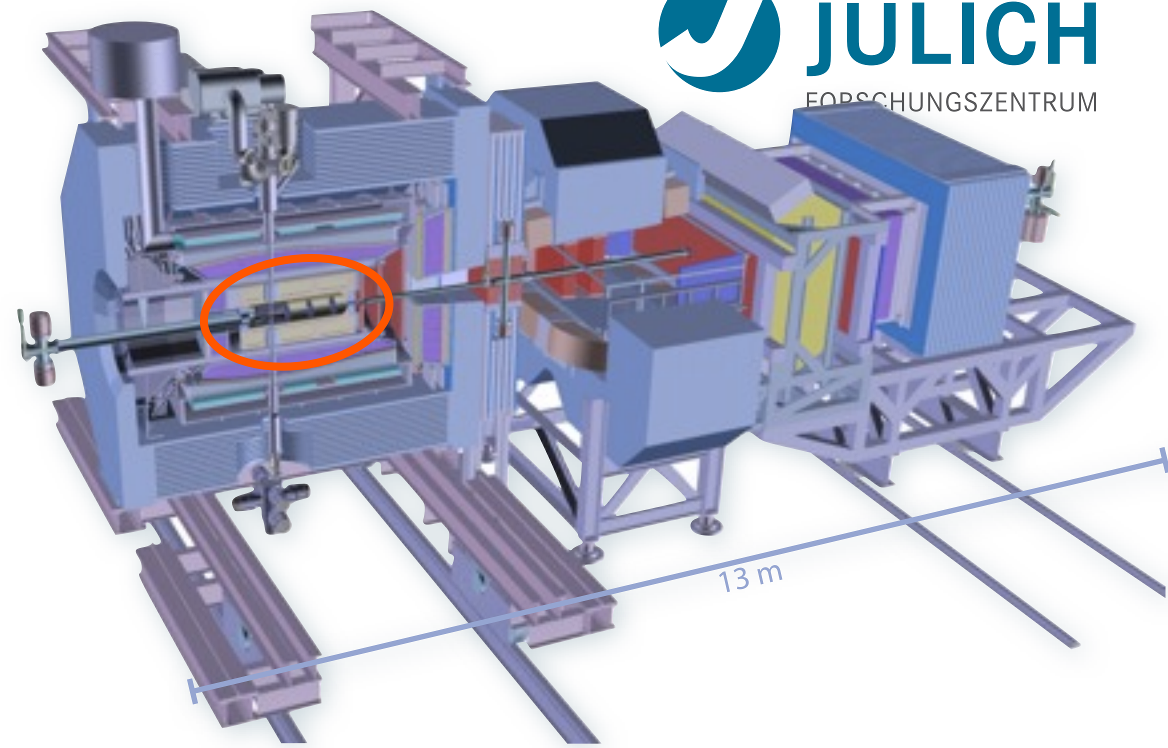
- PANDA
- GPUs: CUDA / Thrust
- Tracking: Hough Transform & Conformal Mapping
- Status & Outlook

# Motivation

- PANDA: **Triggerless** read out
  - Many benchmark channels
  - Background & signal similar
- Event Rate: $2 \cdot 10^7/s$



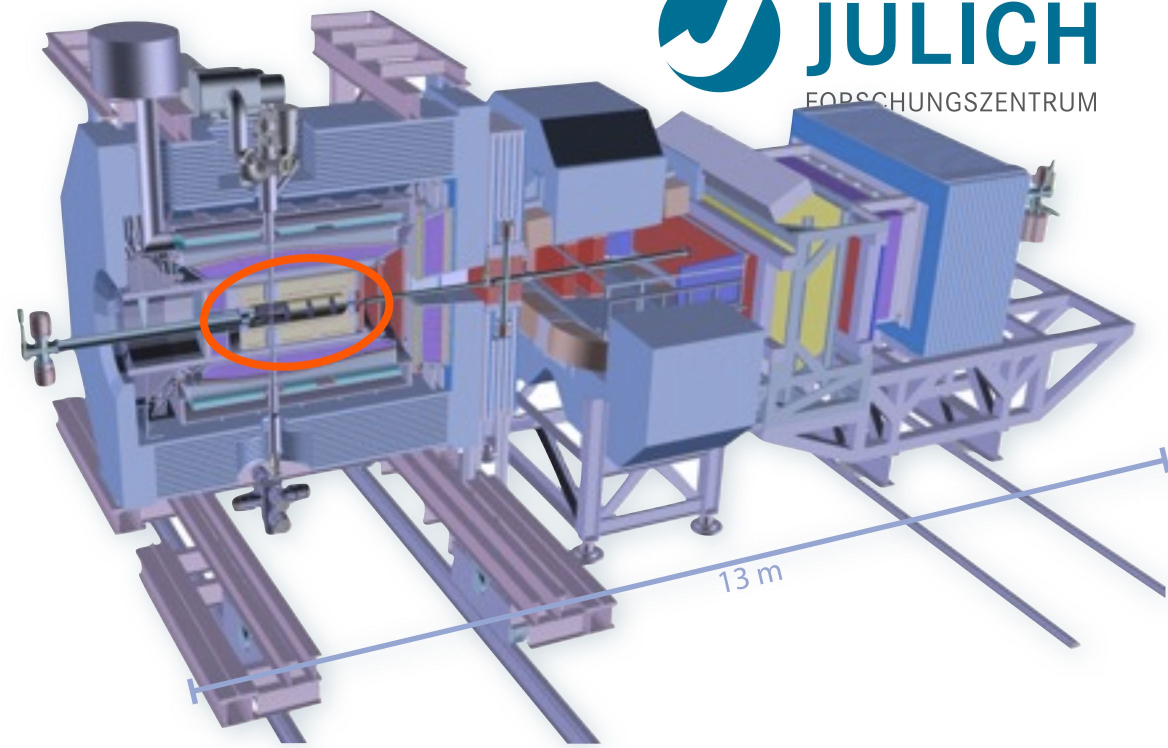
# Motivation

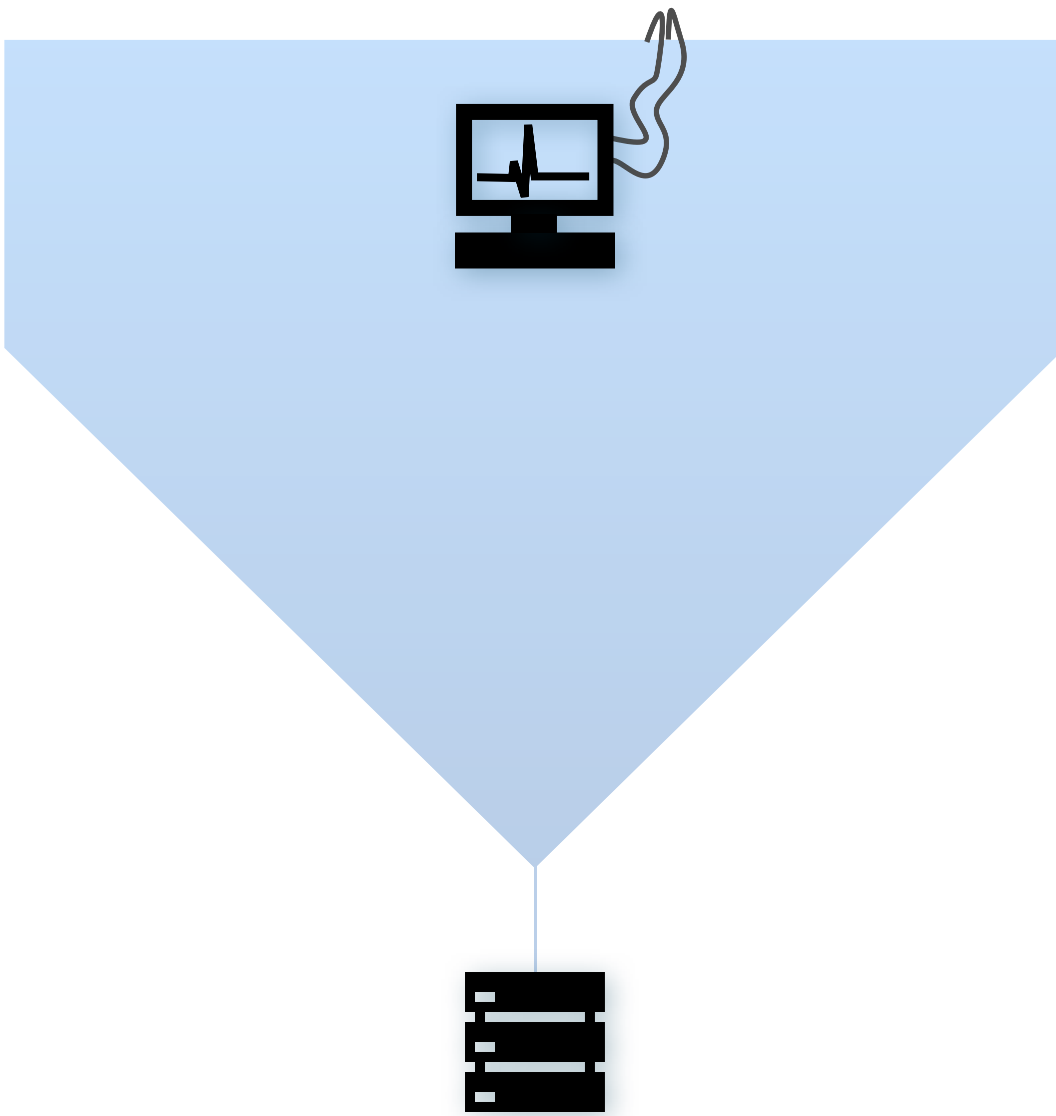



- PANDA: **Triggerless** read out
  - Many benchmark channels
  - Background & signal similar
- Event Rate: $2 \cdot 10^7/s$

# Motivation

- PANDA: **Triggerless** read out
  - Many benchmark channels
  - Background & signal similar
- Event Rate: $2 \cdot 10^7$/s



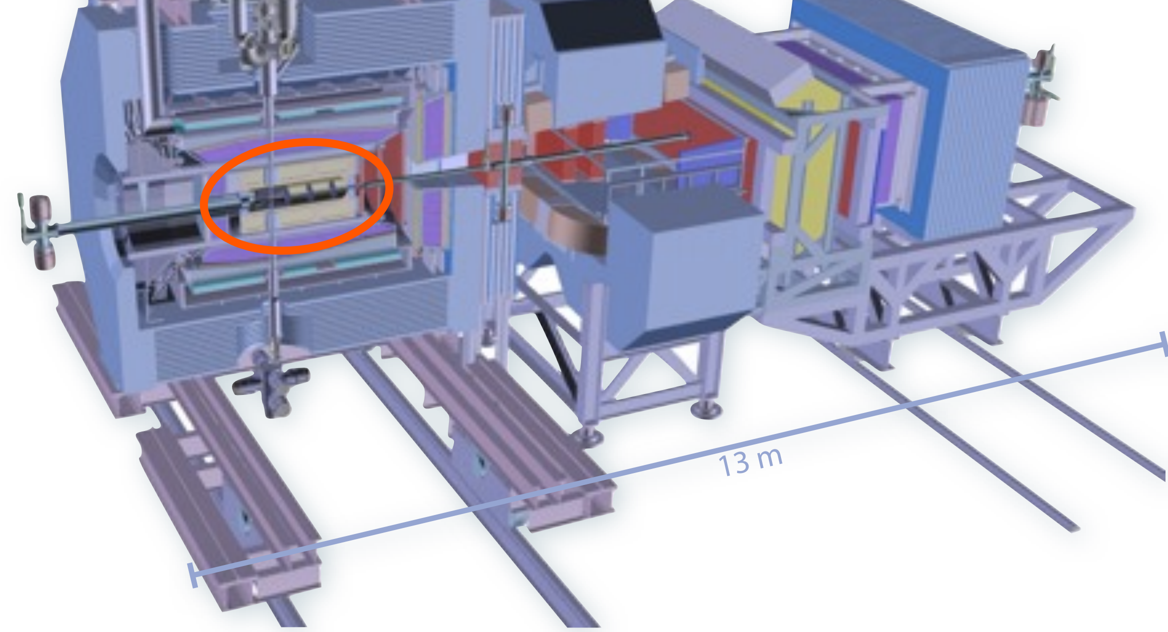



Raw Data Rate:
**200 GB/s**

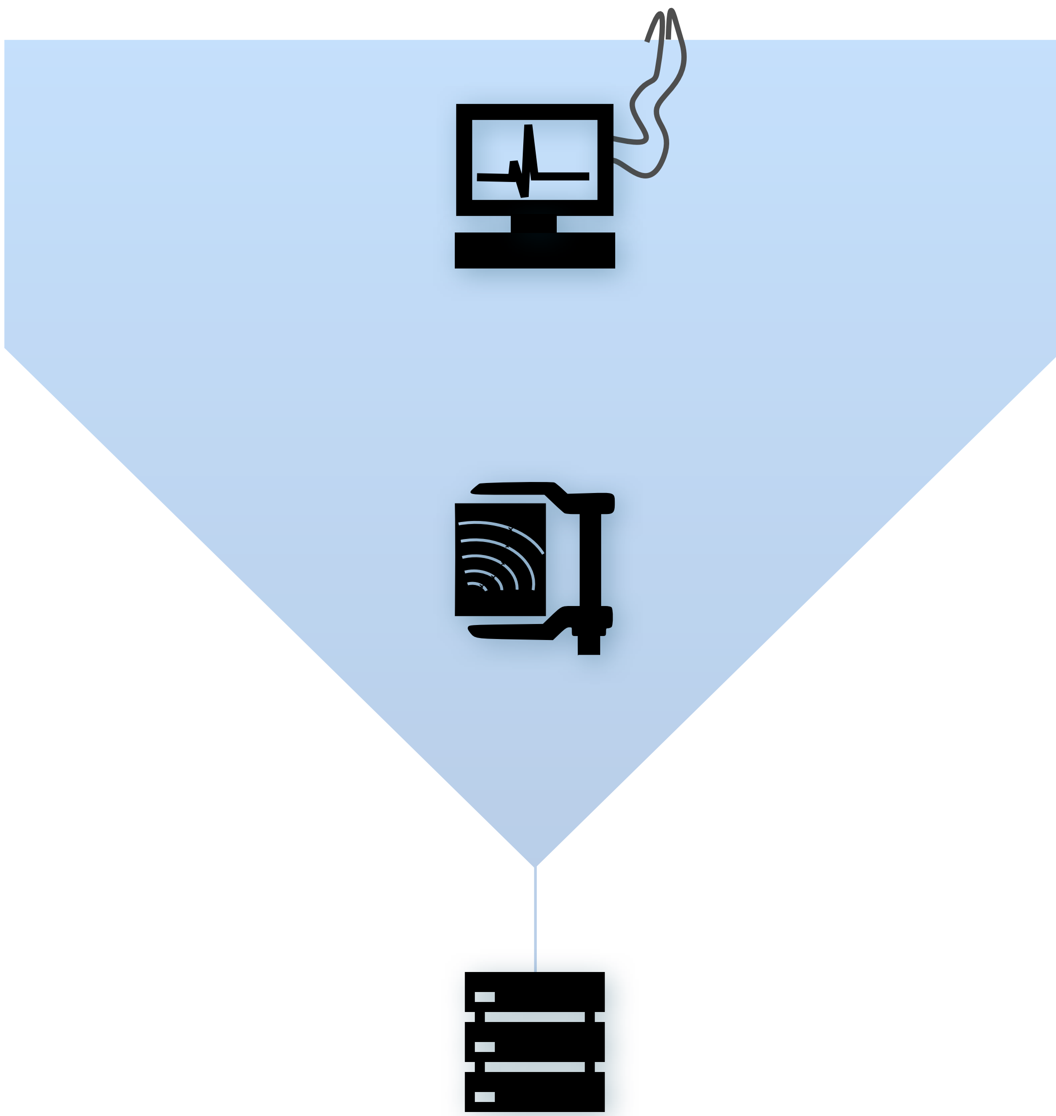
Disk Storage Space for Offline Analysis: ~10 PB/y

# Motivation

JÜLICH
FORSCHUNGSZENTRUM




- PANDA: **Triggerless** read out
  - Many benchmark channels
  - Background & signal similar
- Event Rate: $2 \cdot 10^7$/s



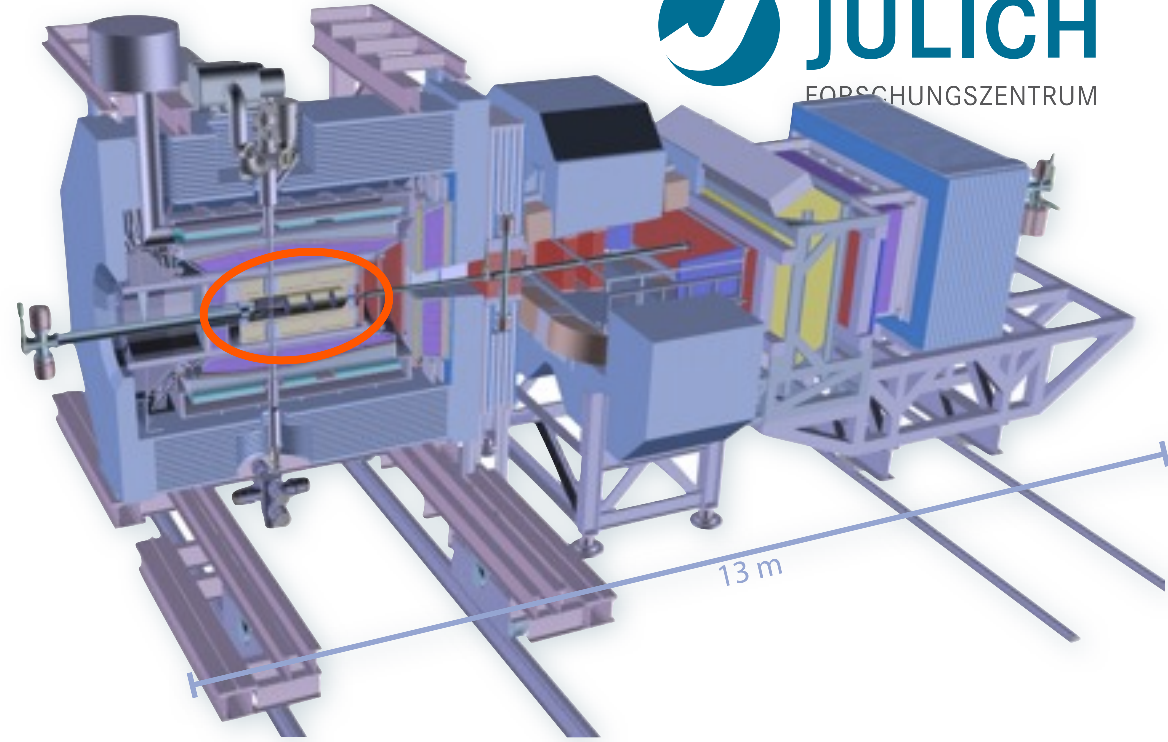
Raw Data Rate:
**200 GB/s**

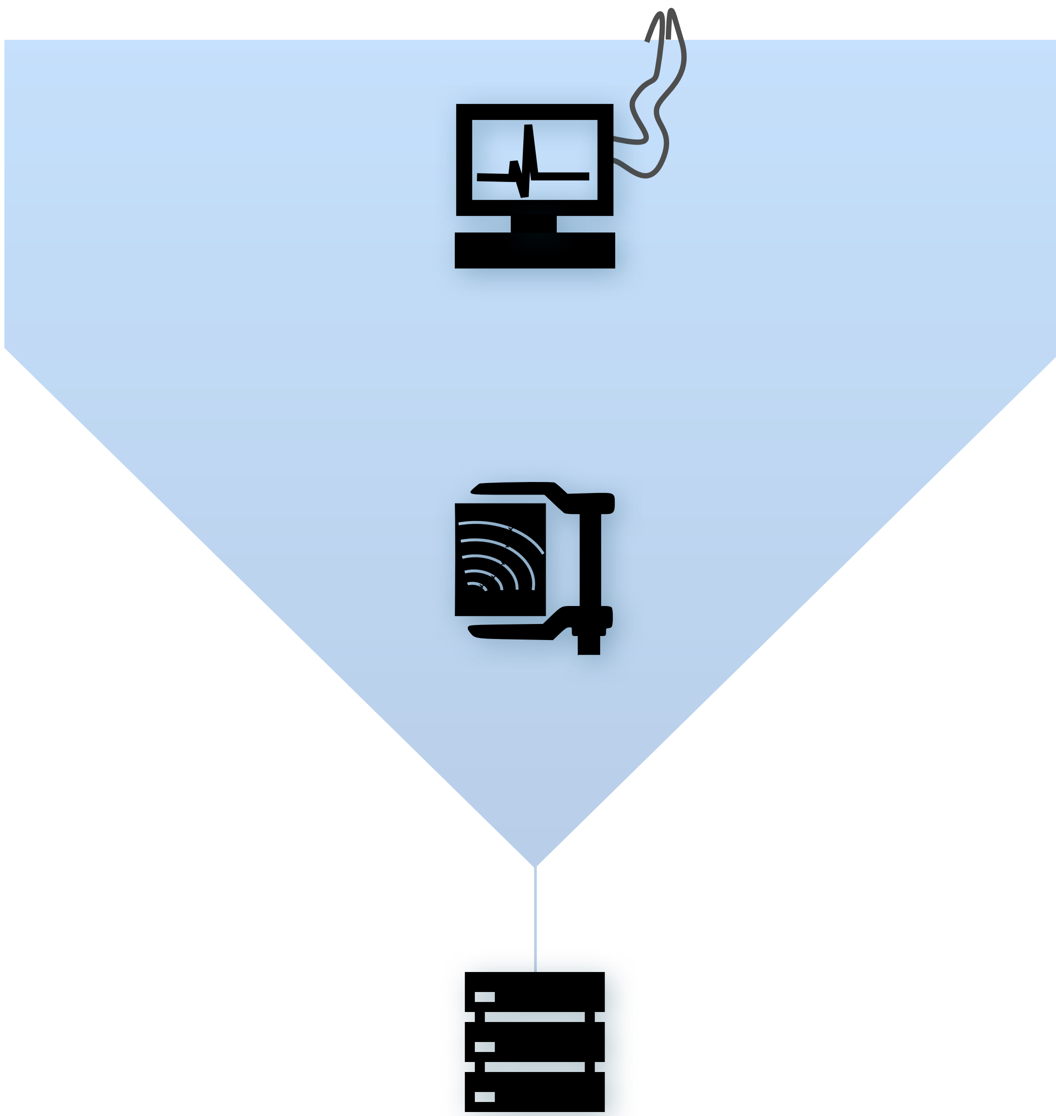
Reduce by
**~1/1000**
(Reject background events,
save interesting physics events)

Disk Storage Space for
Offline Analysis: ~10 PB/y

# Motivation

- PANDA: **Triggerless** read out
  - Many benchmark channels
  - Background & signal similar
- Event Rate: $2 \cdot 10^7$/s



Raw Data Rate:
**200 GB/s**

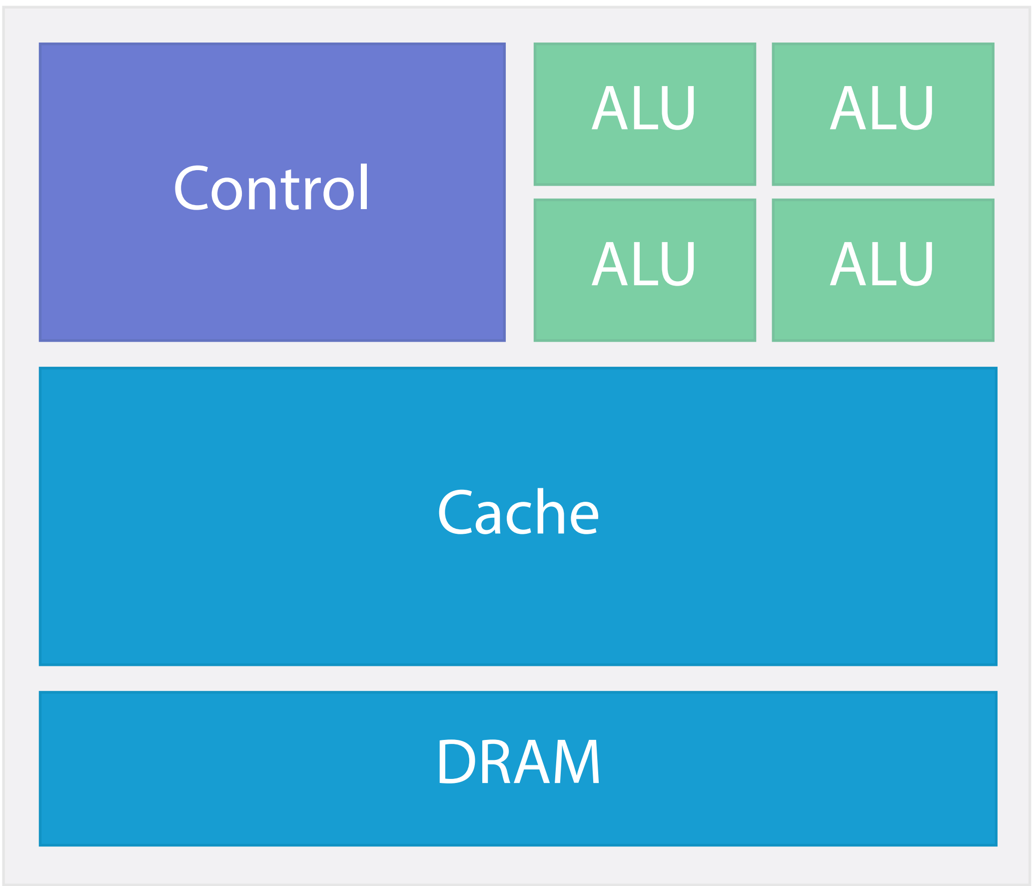
Reduce by
**~1/1000**

*Online Filtering*

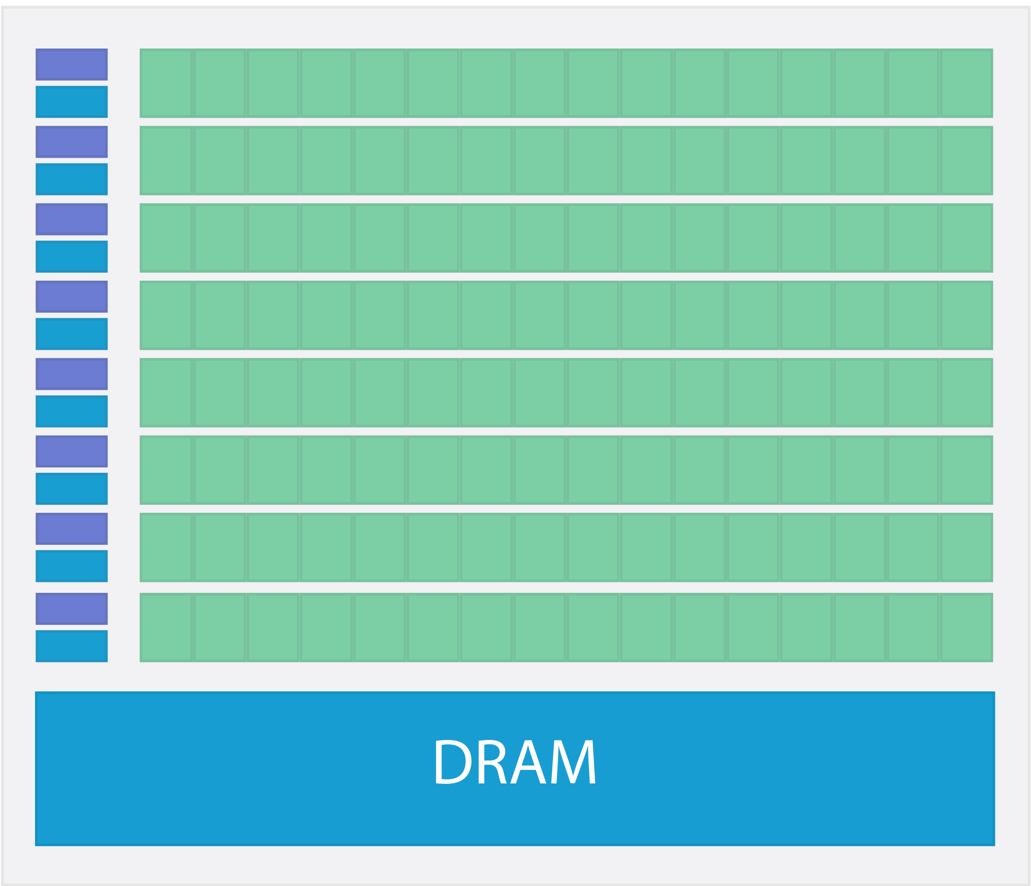
(Reject background event ,
save interesting physics events)

Disk Storage Space for
Offline Analysis: ~10 PB/y

# GPU




**CPU**




**GPU**

# GPU


PCI Express 3.0 Host Interface
GigaThread Engine
SMX
Memory Controller
L2 Cache

# GPU


SMX
Instruction Cache
Warp Scheduler
Dispatch
Register File (65,536 x 32-bit)
Core
DP Unit
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
48 KB Read-Only Data Cache
Tex
Memory Controller

# GPU

- Concept: Many small multiprocessors working in parallel
- My card at IKP: **NVIDIA GeForce GTX 580**
  - *Fermi* architecture
    - Card released three years ago

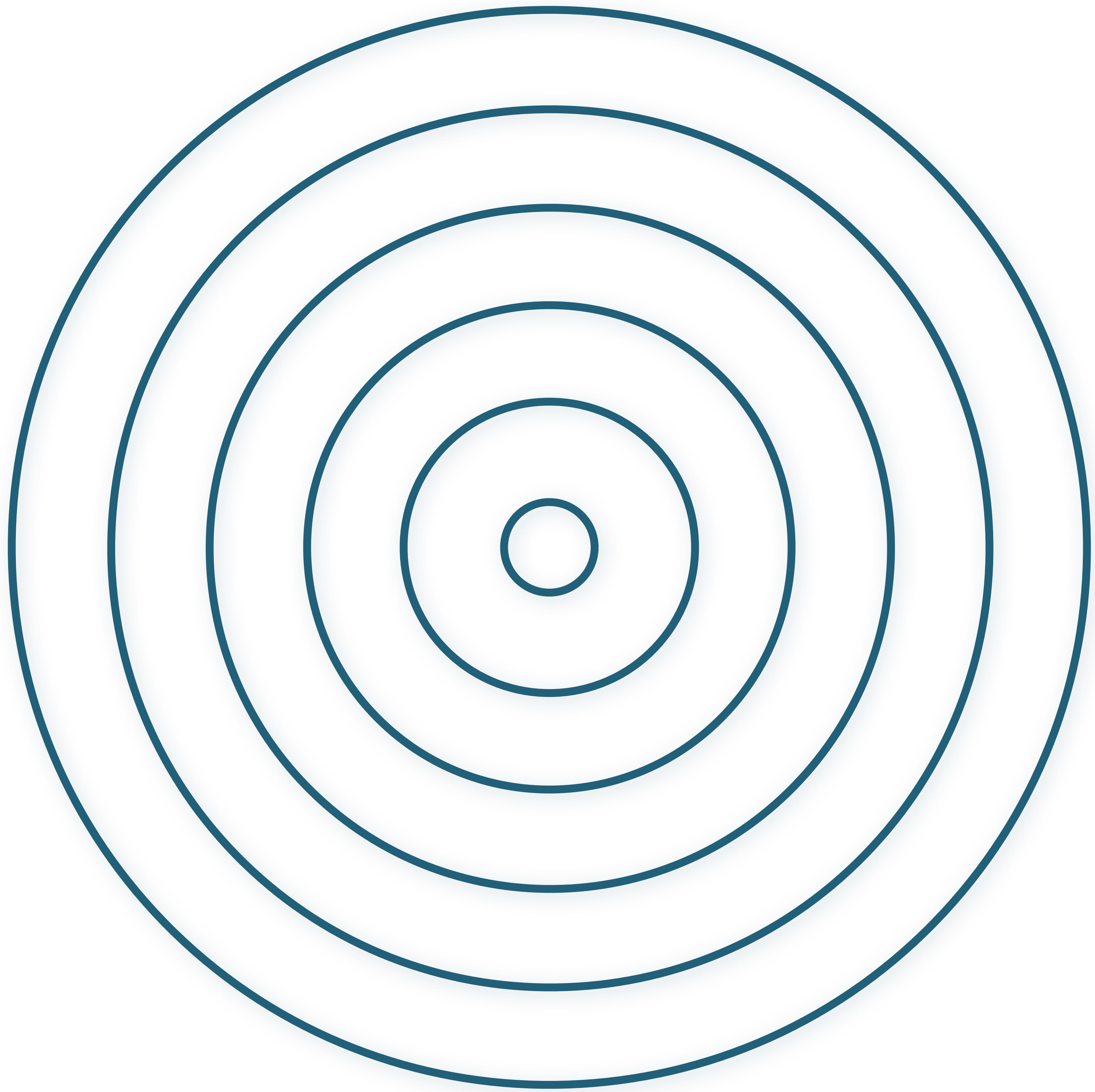
| Feature | **GTX 580 (2010, Fermi)** | GTX 680 (2012, Kepler) | Tesla K20X (2013, Kepler) |
|---|---|---|---|
| Cores | 512 (16 MP x 32 CUDA Cores) | 1536 (8 MP x 192 CUDA Cores) | 2688 (14 MP x 192 CUDA Cores) |
| Global Memory | 1,5 GB | 2 GB | 6 GB |
| GFLOPS | 1580 | 3000 | 3950 |

# CUDA

- **CUDA C/C++**: Interface C ↔ GPU
  - Mixed-device code (parts run on host, parts on GPU)
  - Extensions: `__global__`, `__device__`
  - Compiler: nvcc; Debugger, IDE
  - Libraries/APIs: cuFFT, cuBLAS, cusp, Thrust
  - Thrust: $\frac{\text{Thrust}}{\text{CUDA}} = \frac{\text{STL}}{\text{C++}}$
    - Collection of template-oriented functions
    - Easier handling of common tasks, extendable for complex tasks

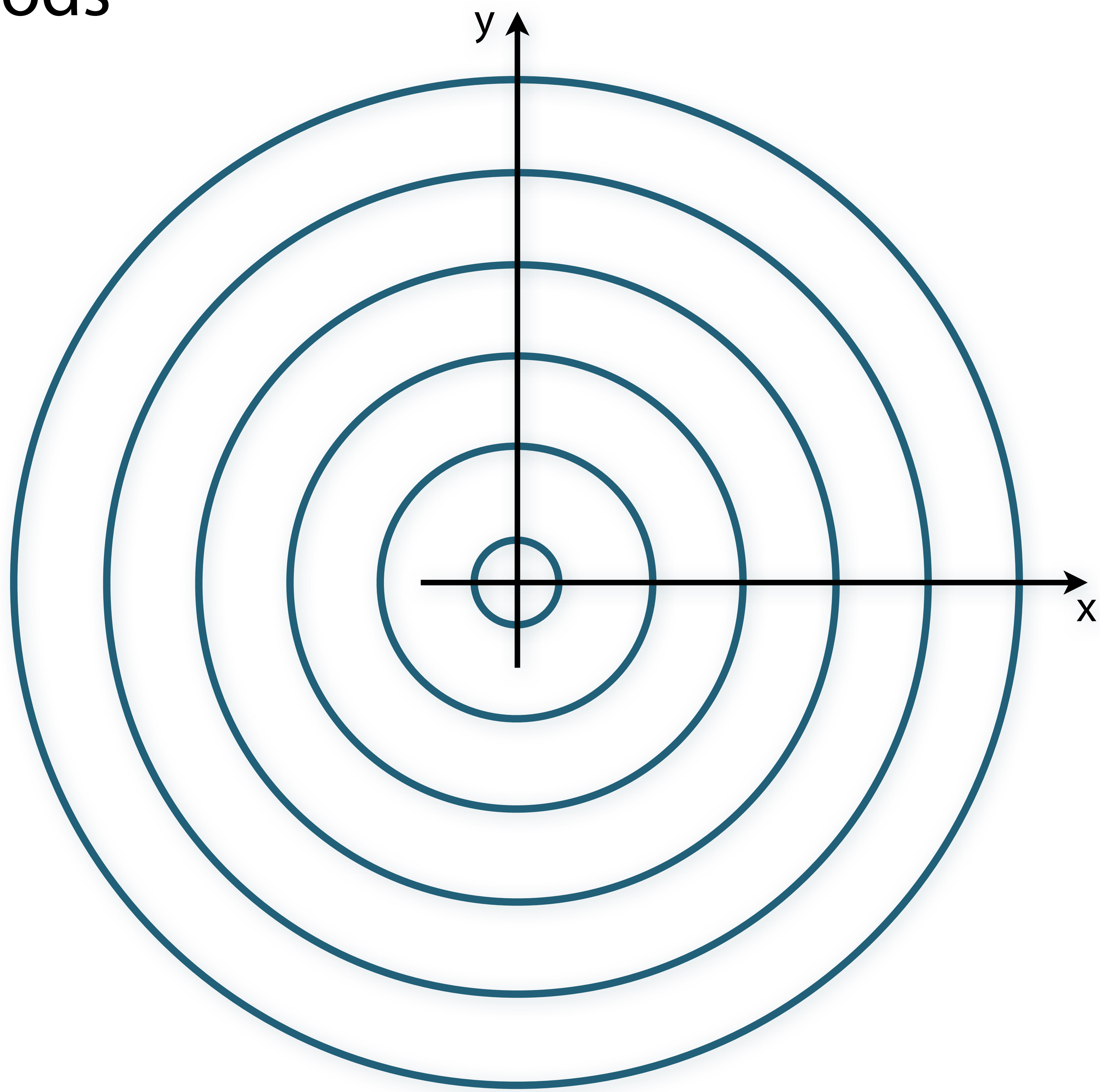
# Hough Transformation

- The Methods



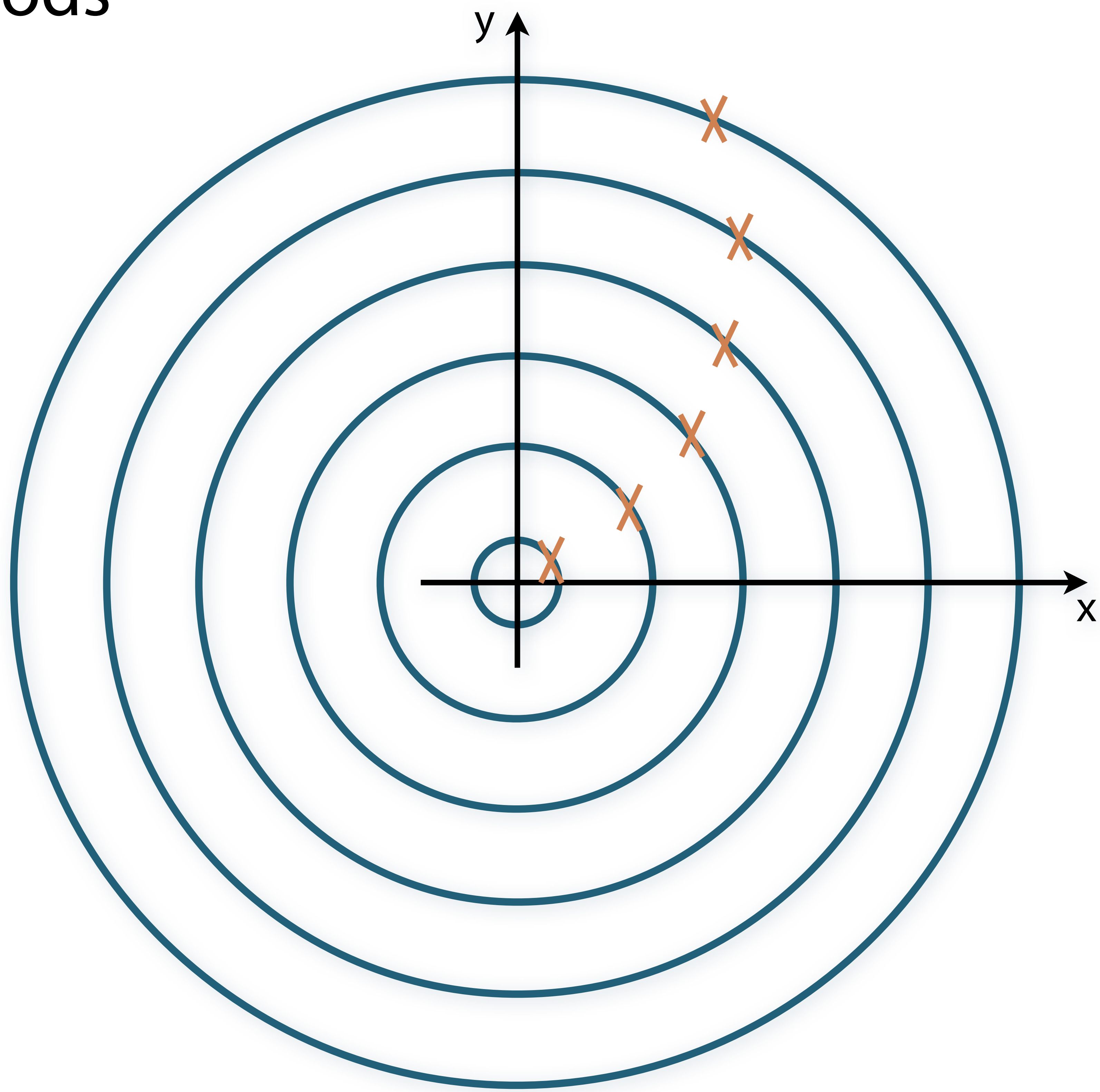
# Hough Transformation

- The Methods



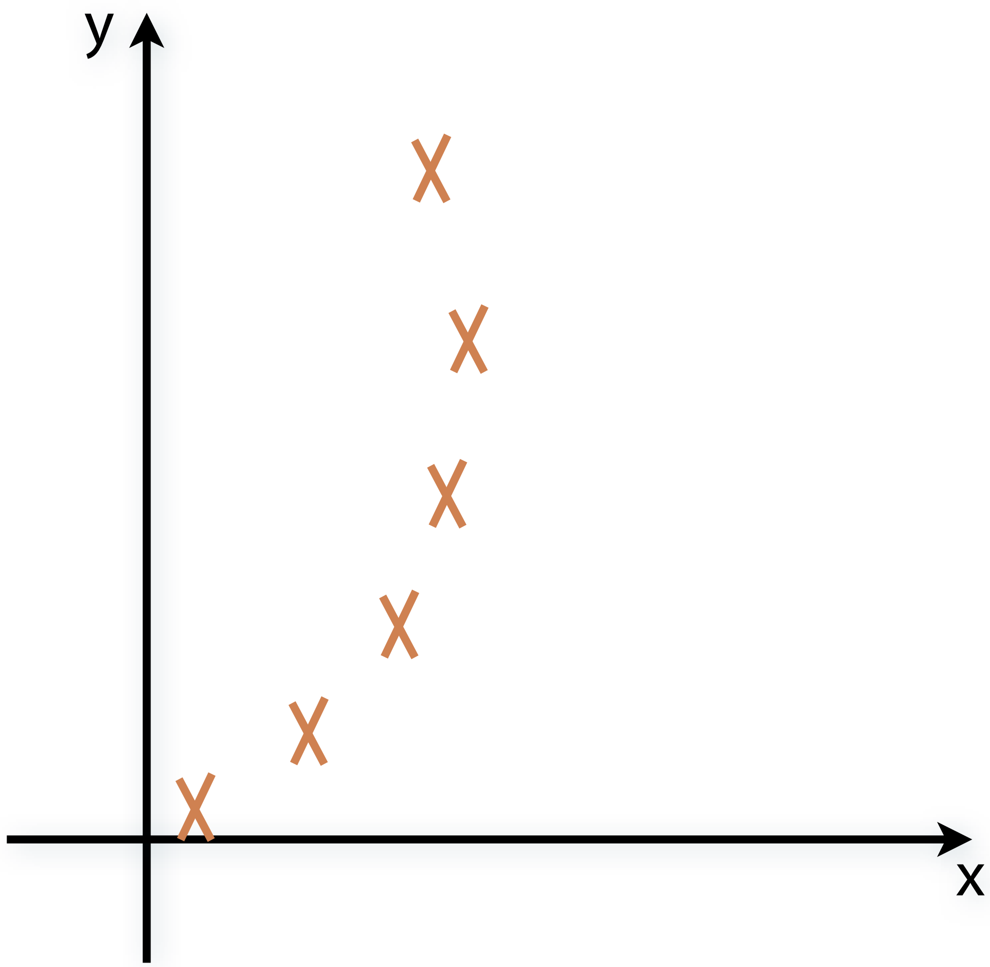
# Hough Transformation

- The Methods



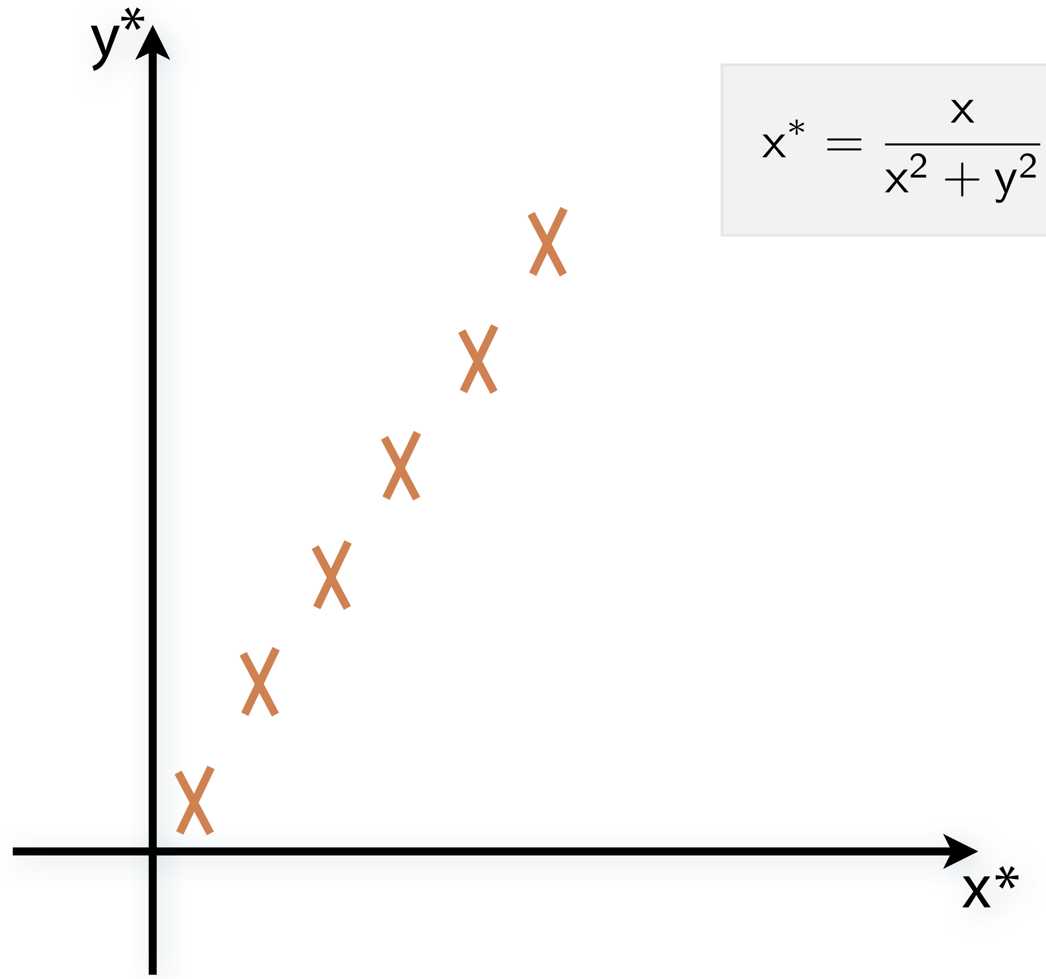
# Hough Transformation

- The Methods



# Hough Transformation

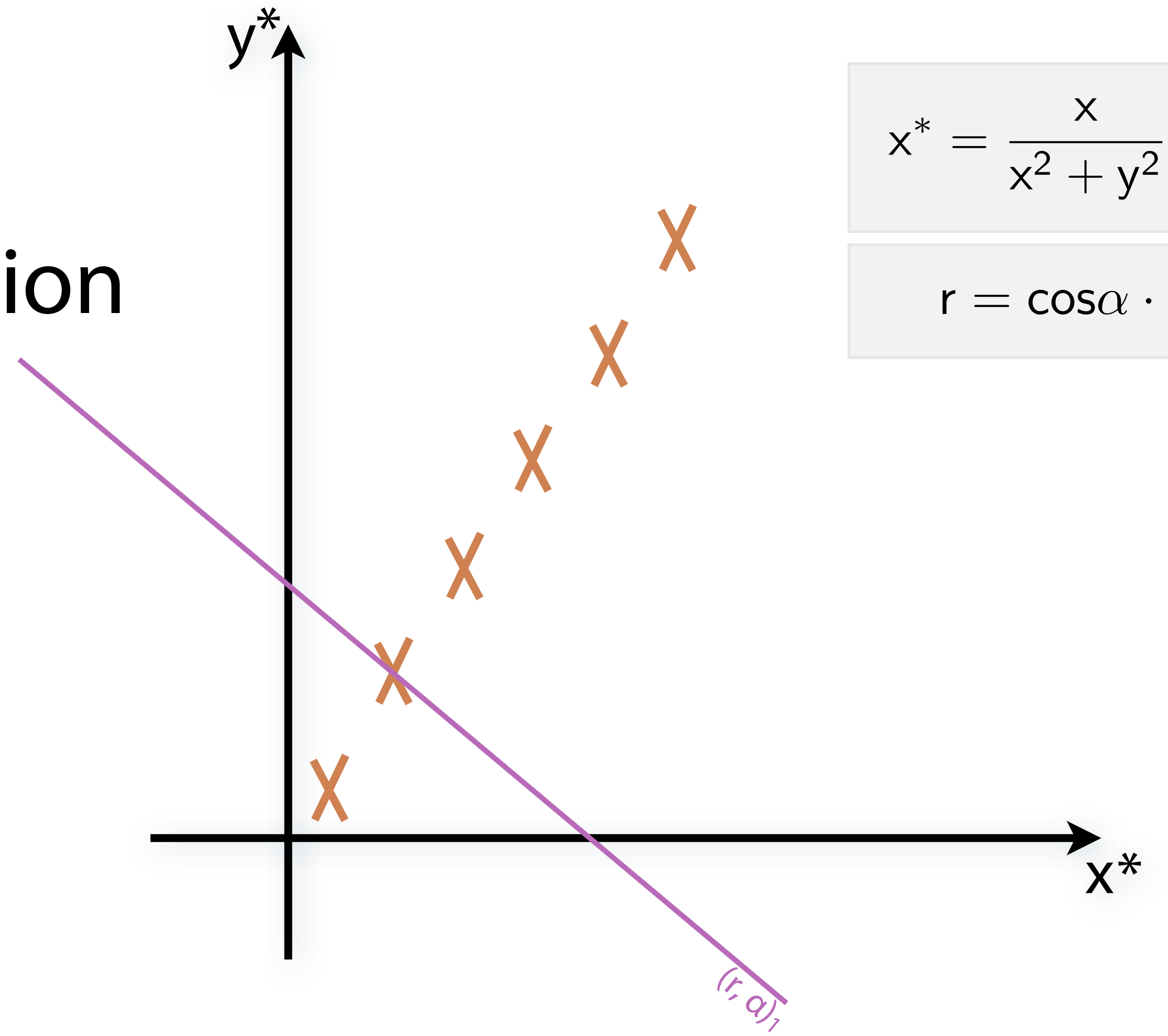
- The Methods
  - Conformal Map




$$x^* = \frac{x}{x^2+y^2}, y^* = \frac{-y}{x^2+y^2}$$

# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation




$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

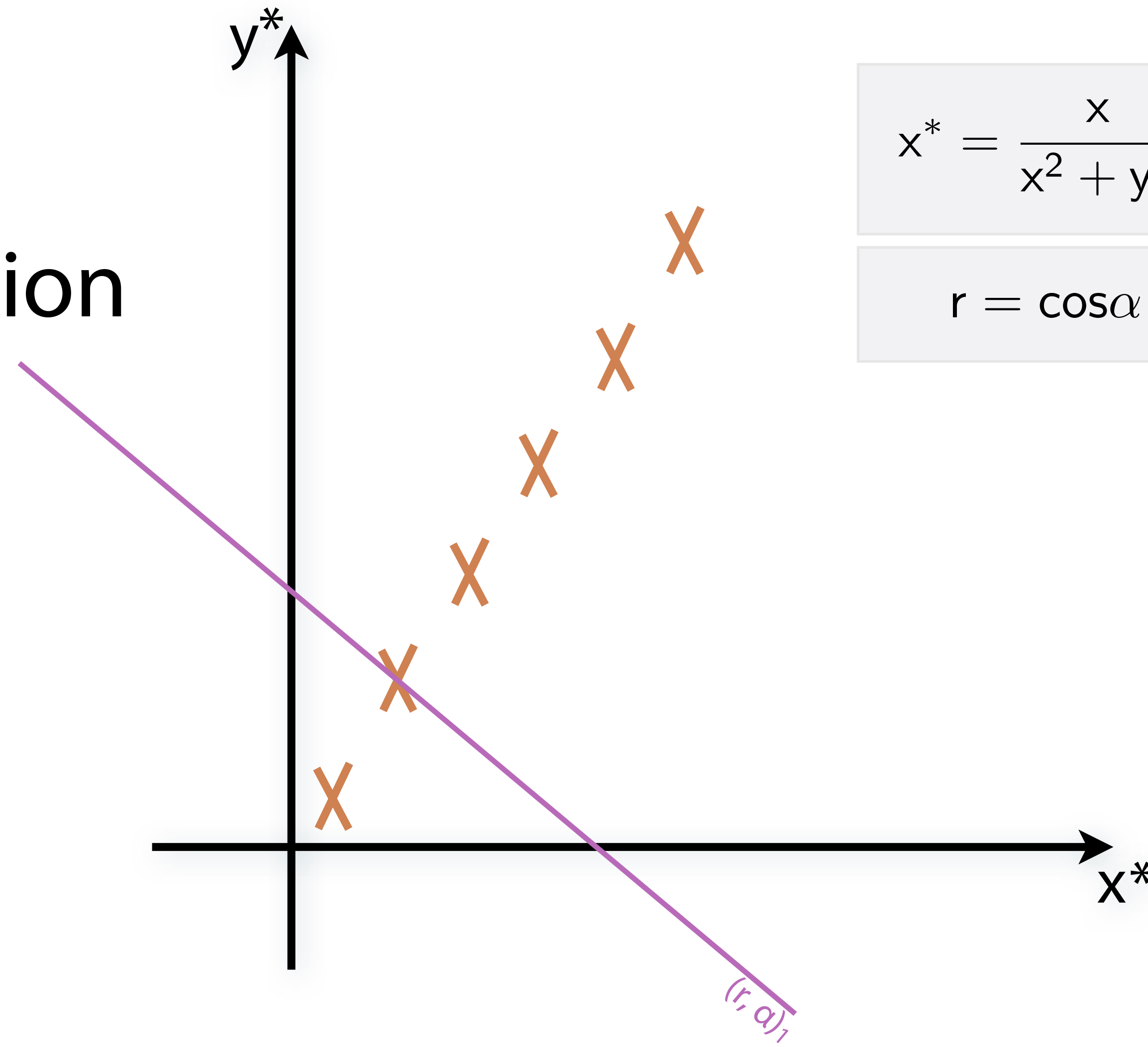
$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

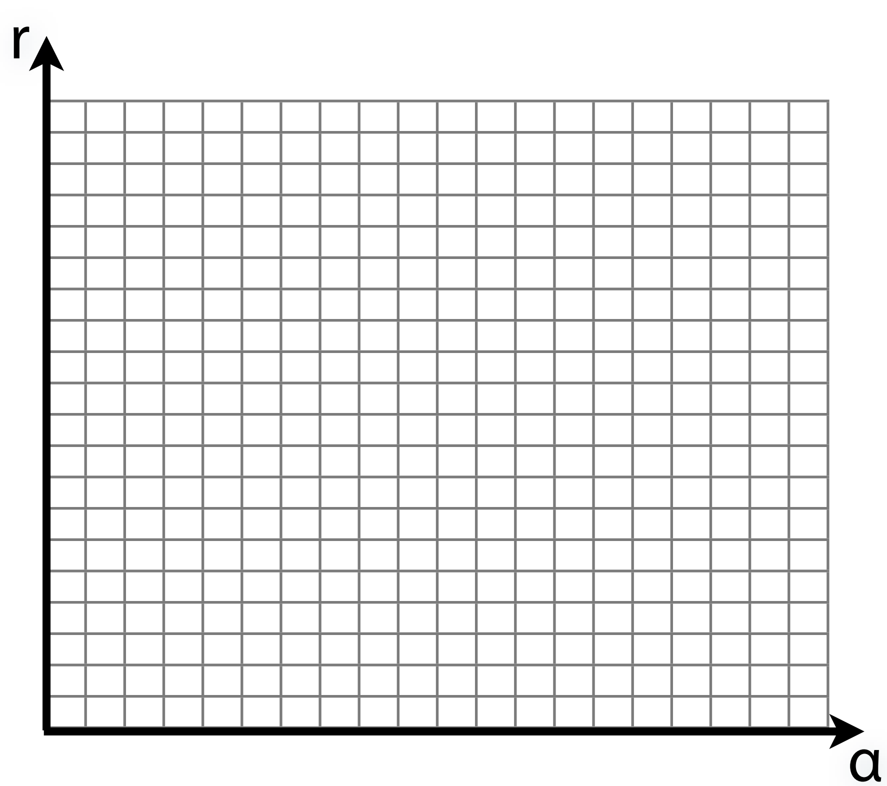
# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$



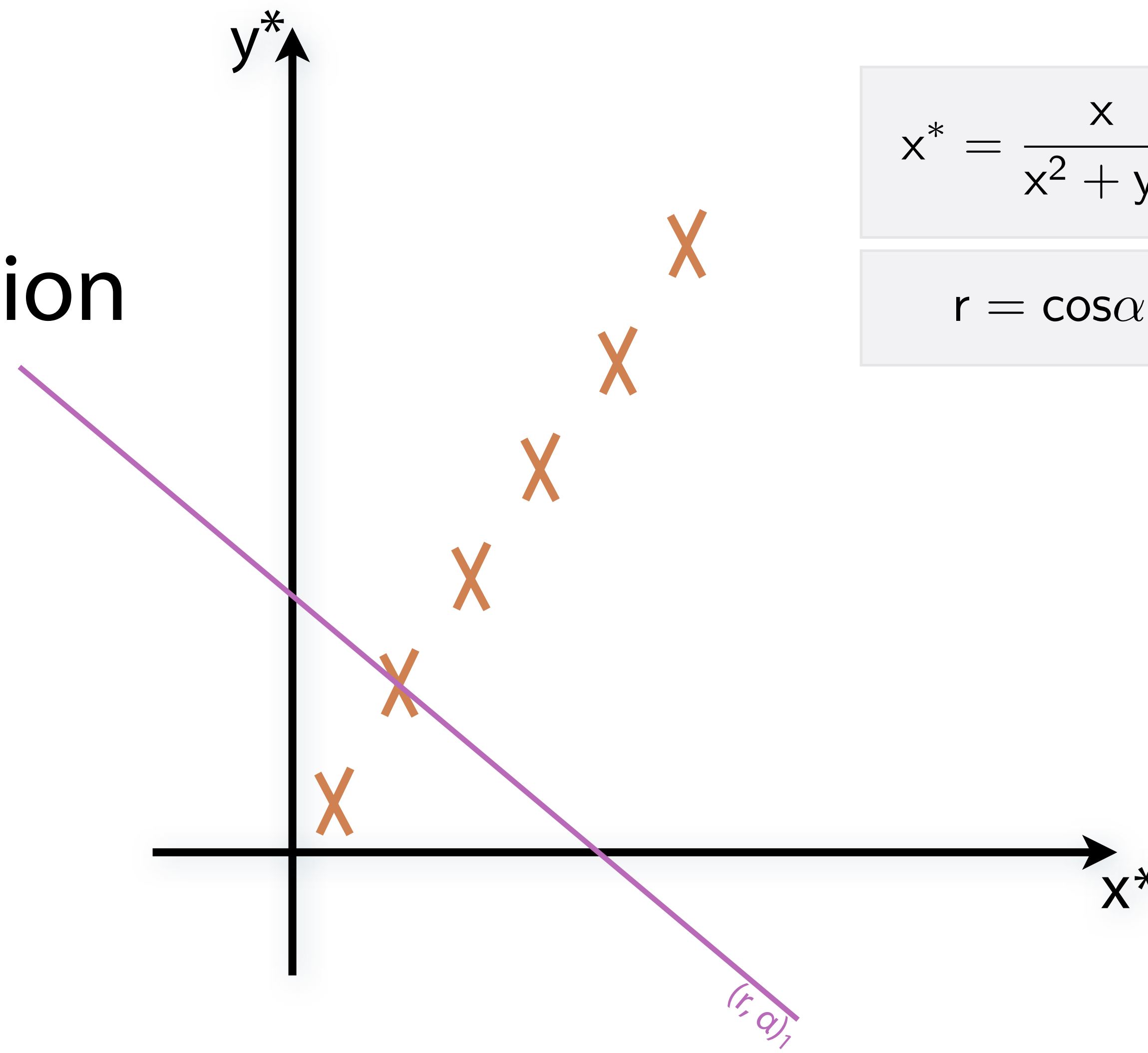



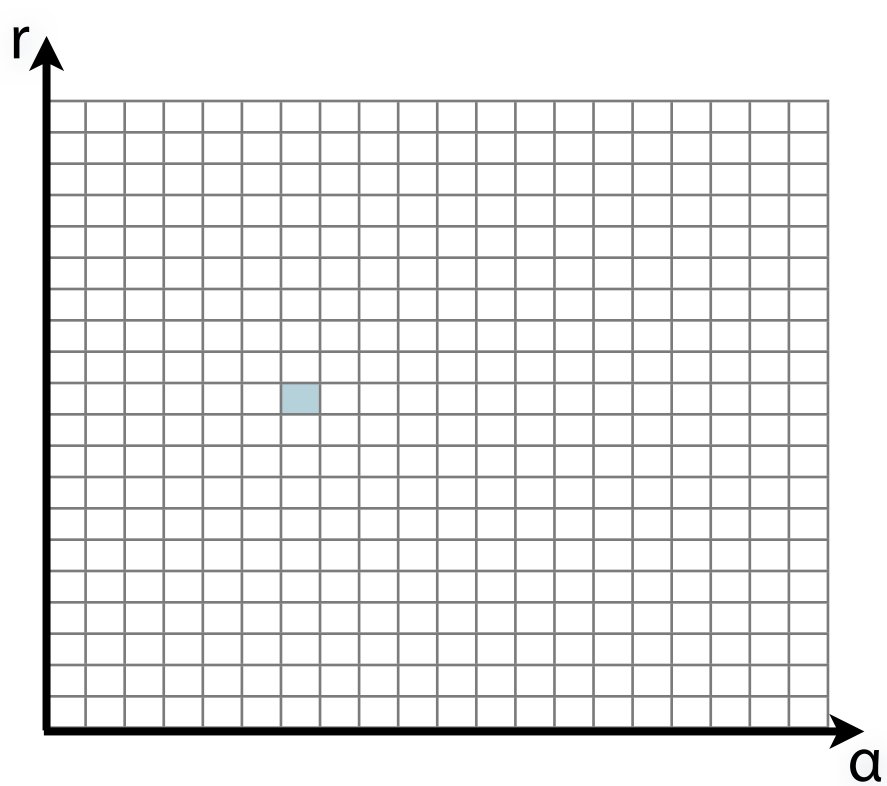
# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

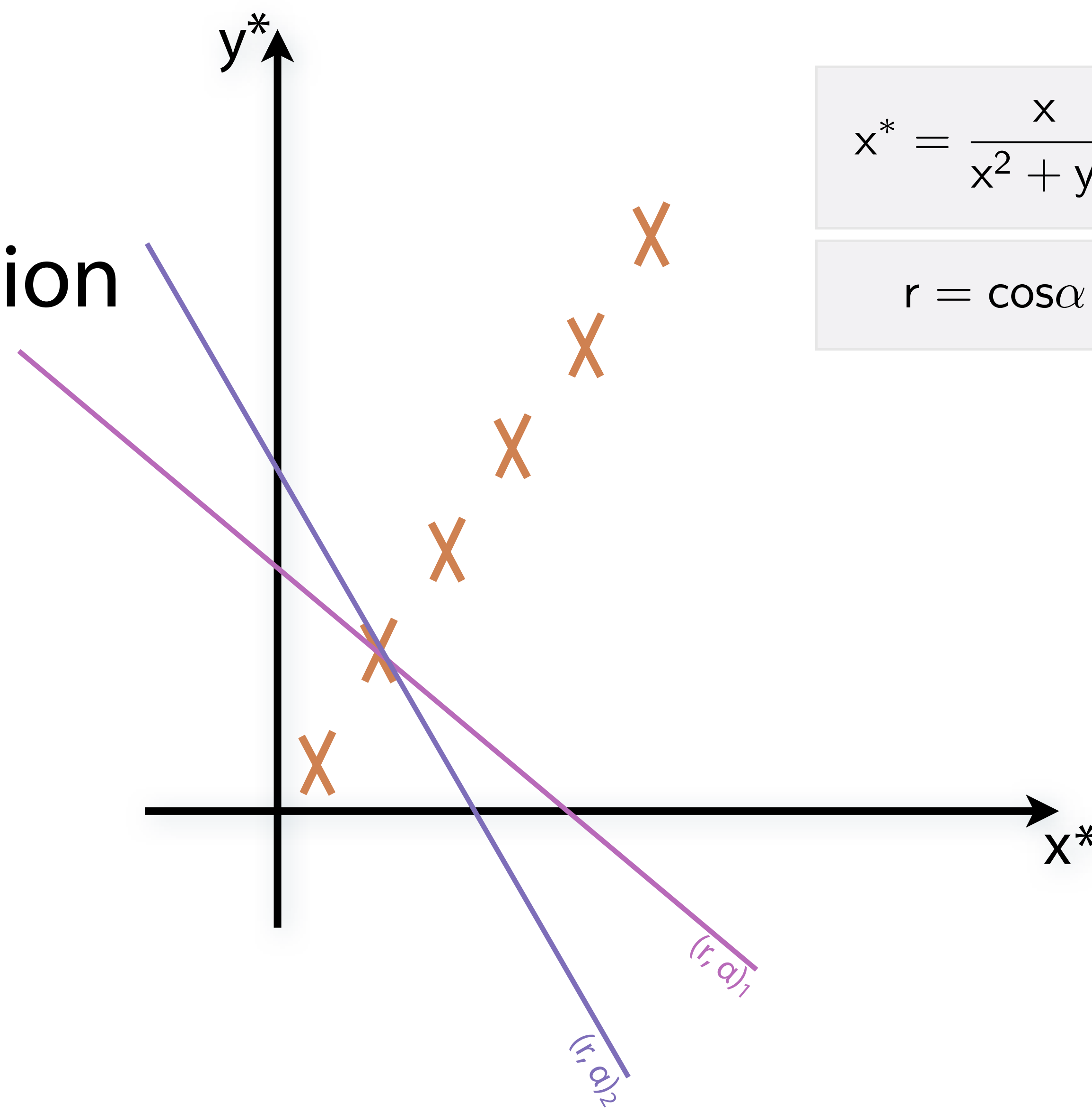
$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$



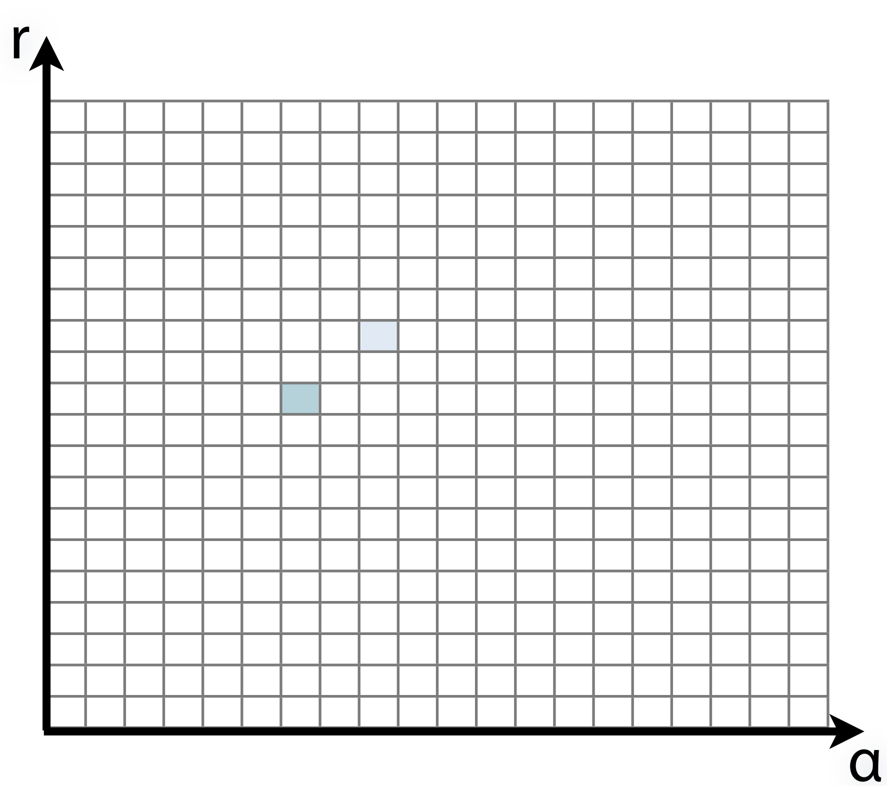
# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation




$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

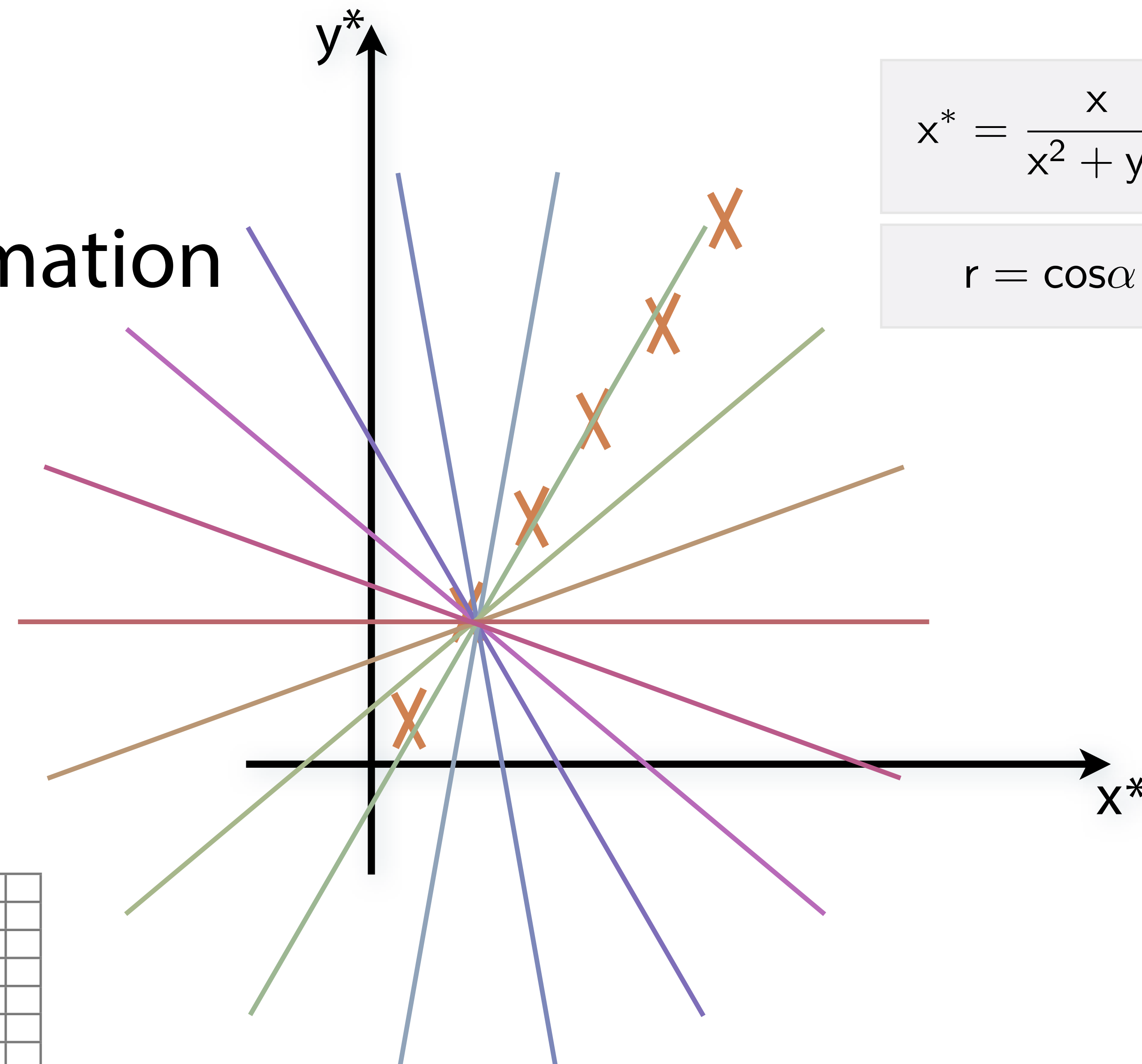


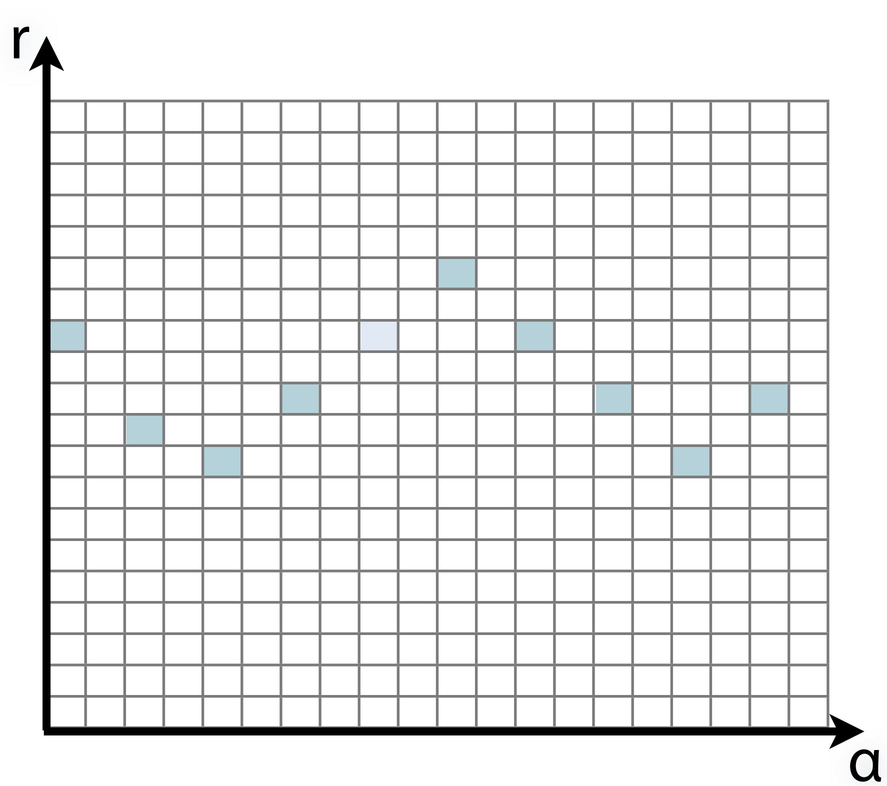
# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation

$$x^* = \frac{x}{x^2 + y^2},\ y^* = \frac{-y}{x^2 + y^2}$$

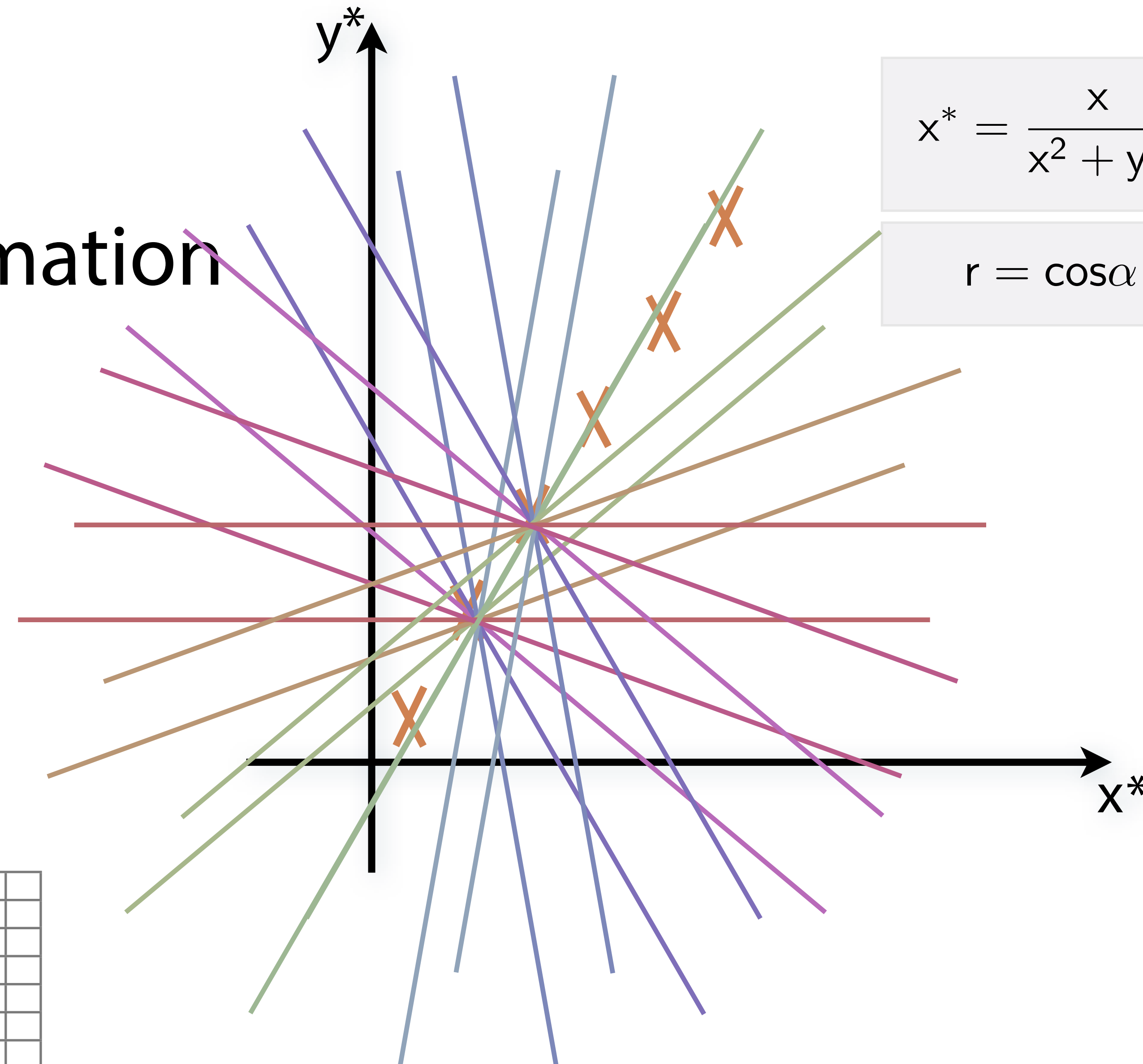
$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$



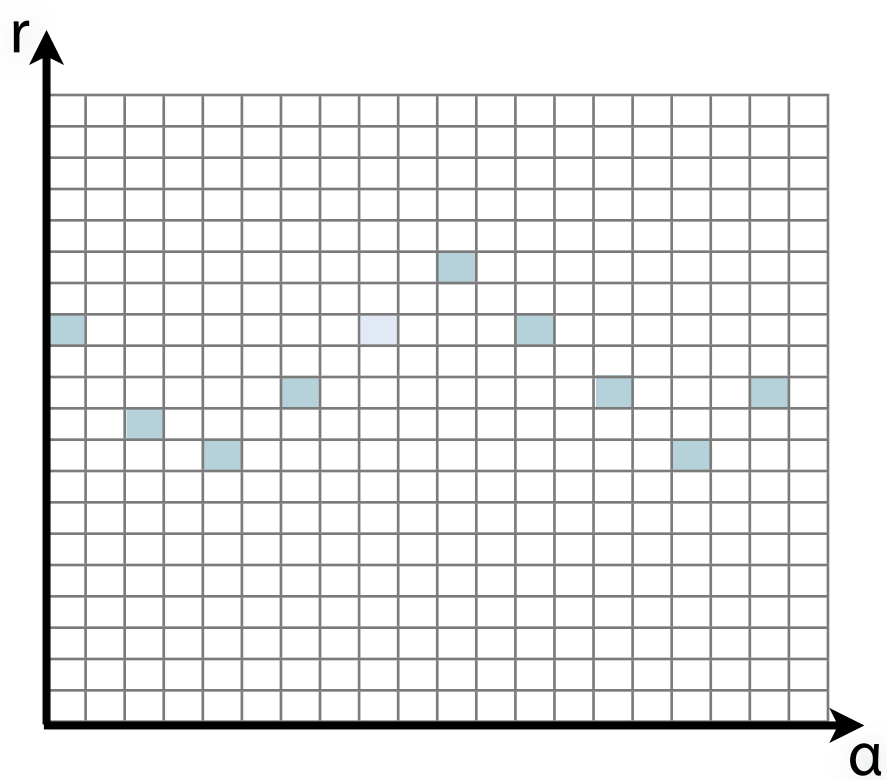
# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation




$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

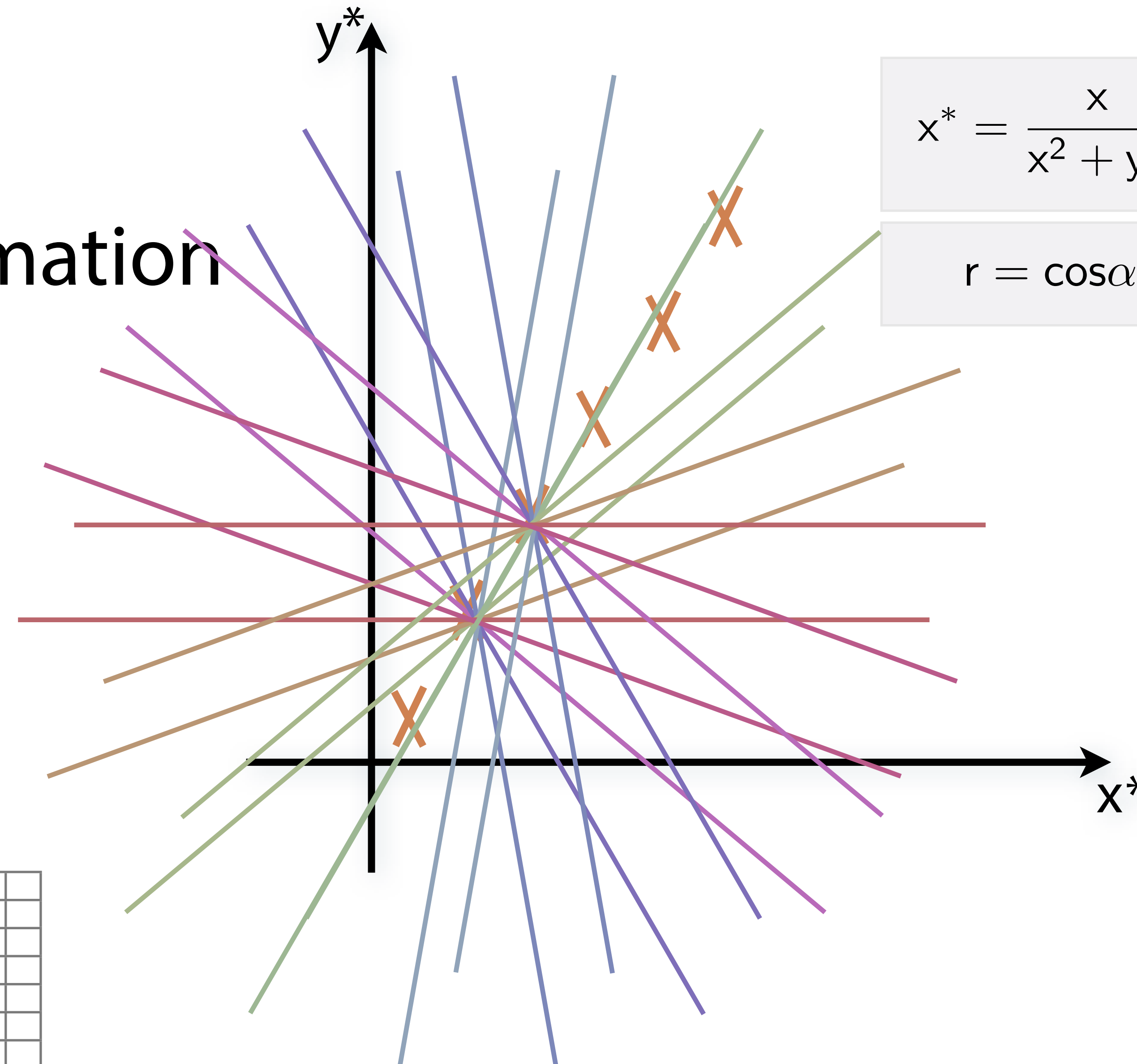


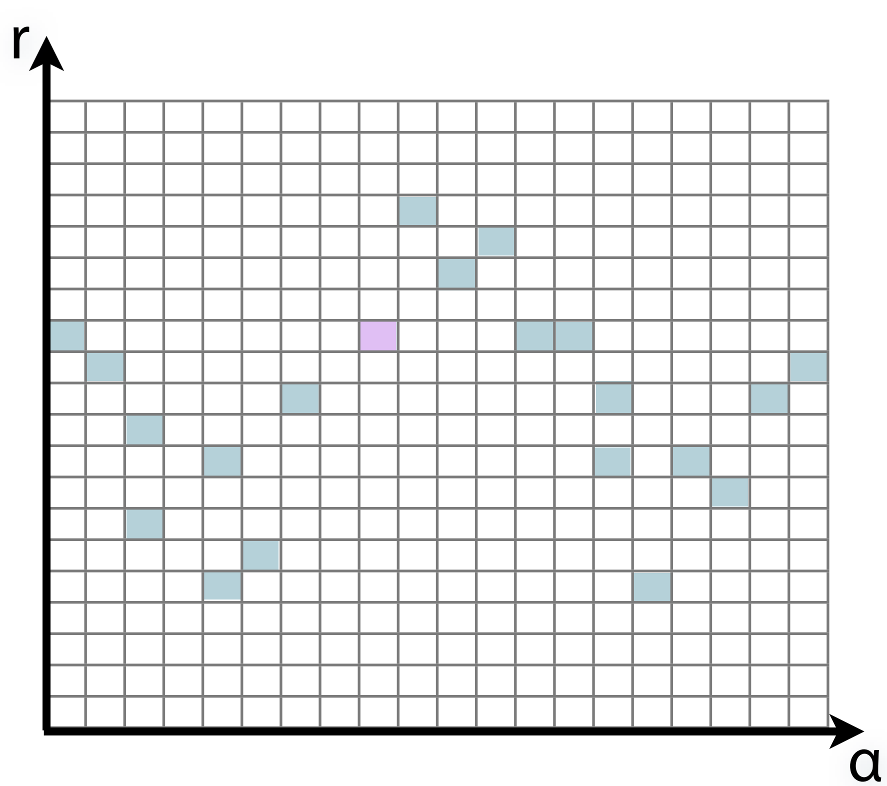
# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation

$$x^* = \frac{x}{x^2+y^2}, y^* = \frac{-y}{x^2+y^2}$$

$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$



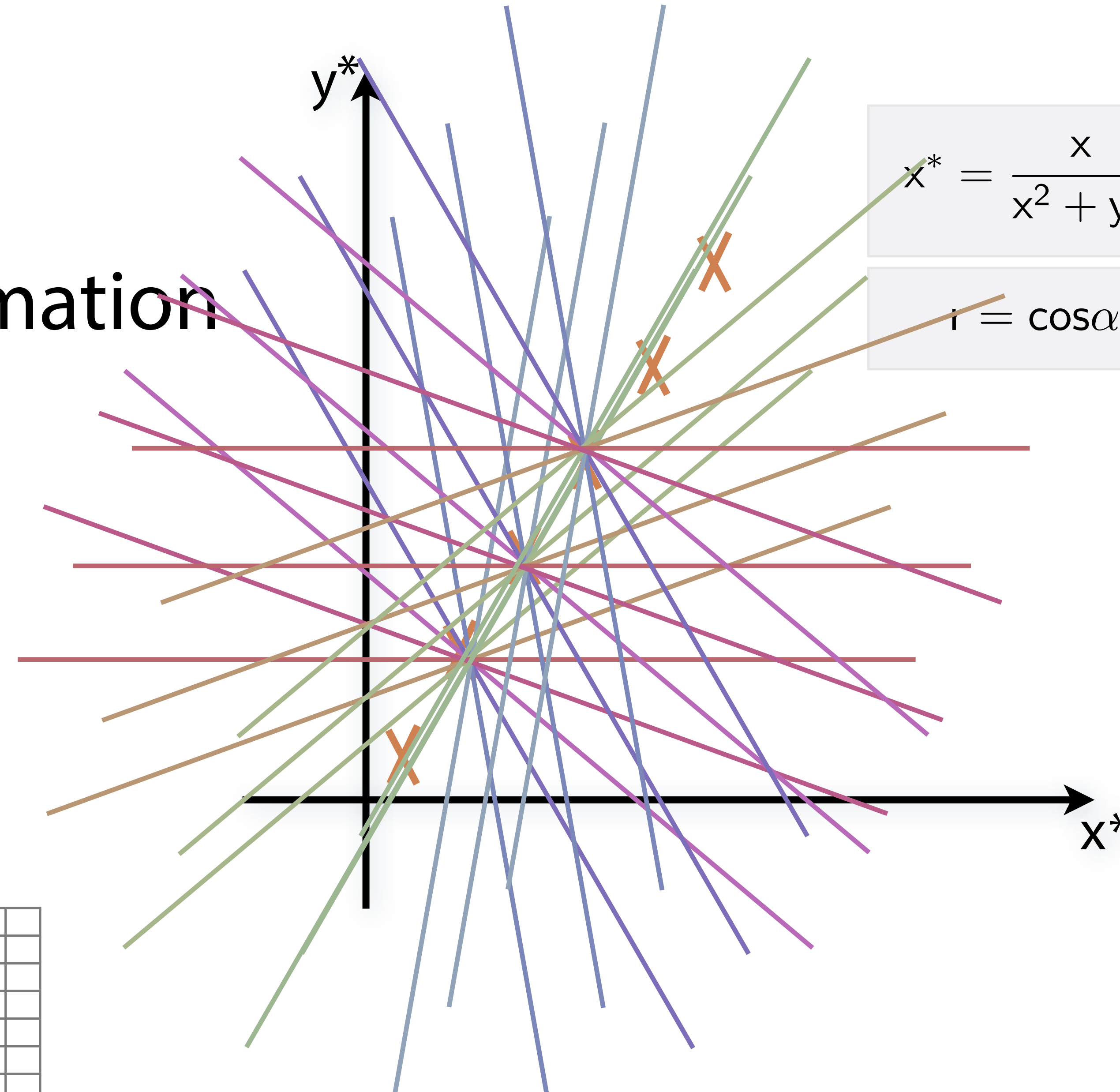



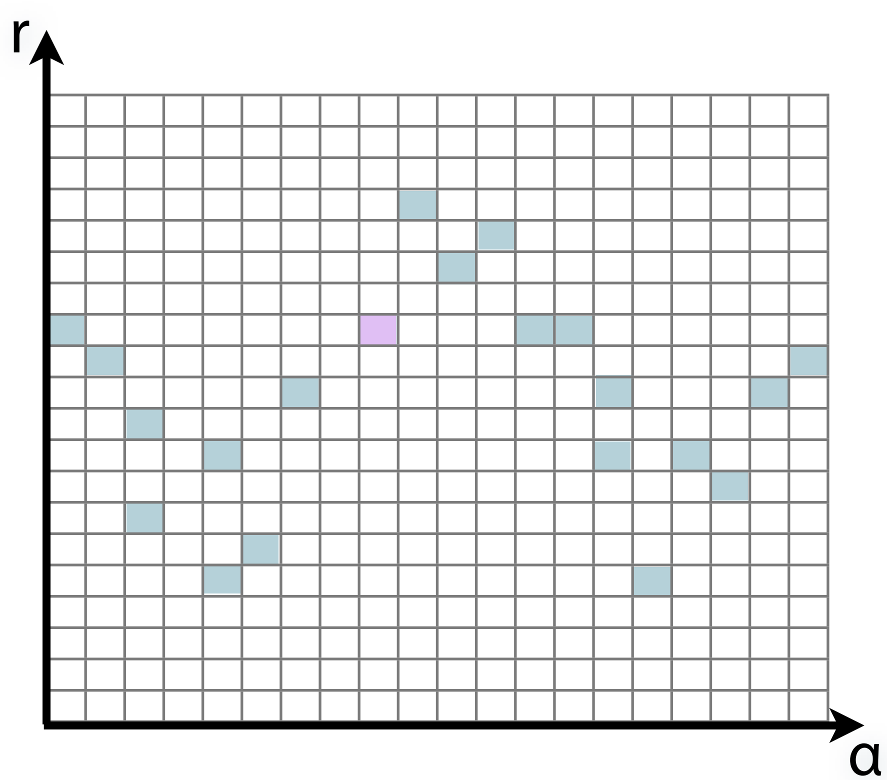
# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation

$$x^* = \frac{x}{x^2 + y^2}, \; y^* = \frac{-y}{x^2 + y^2}$$

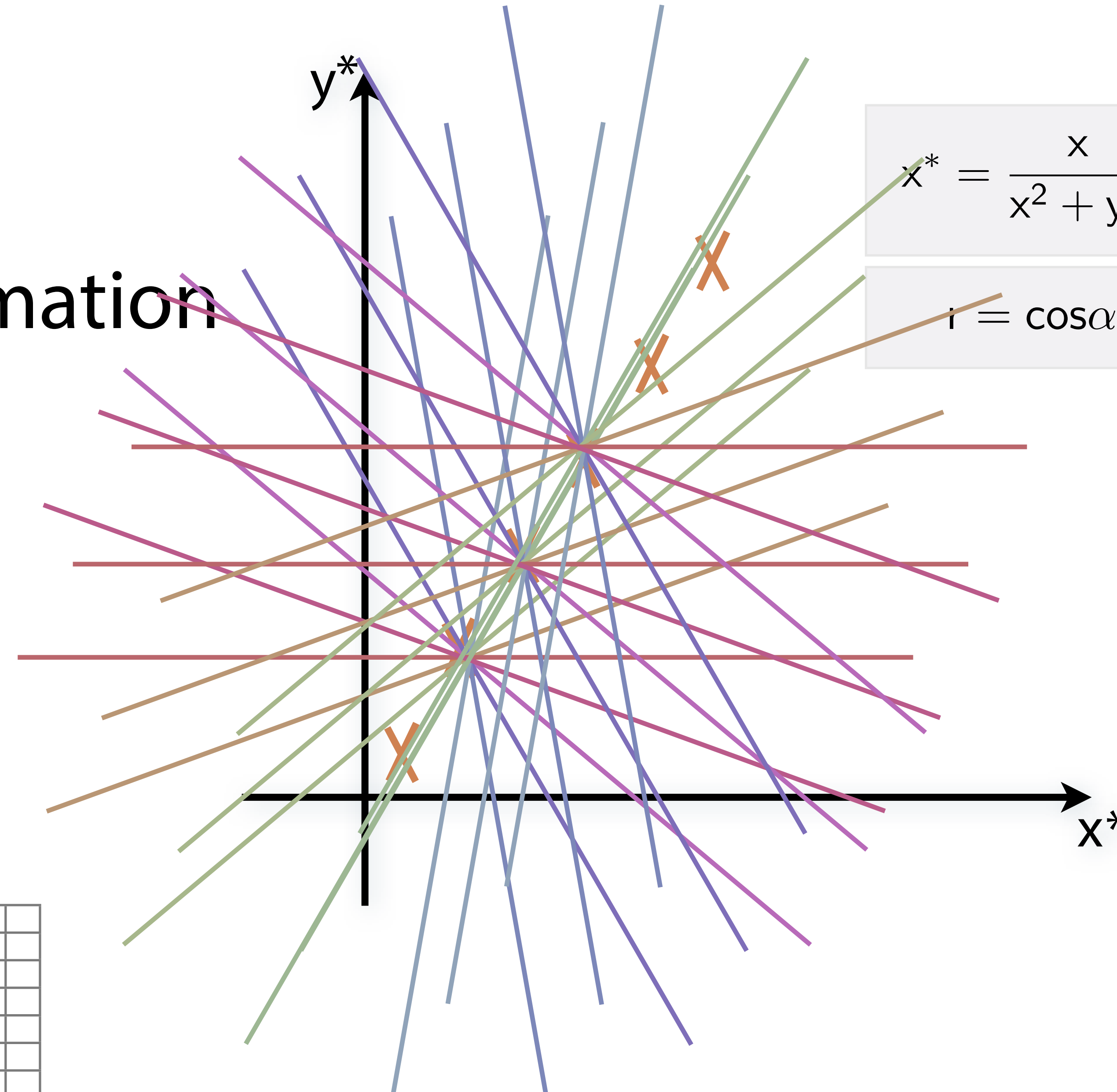
$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$



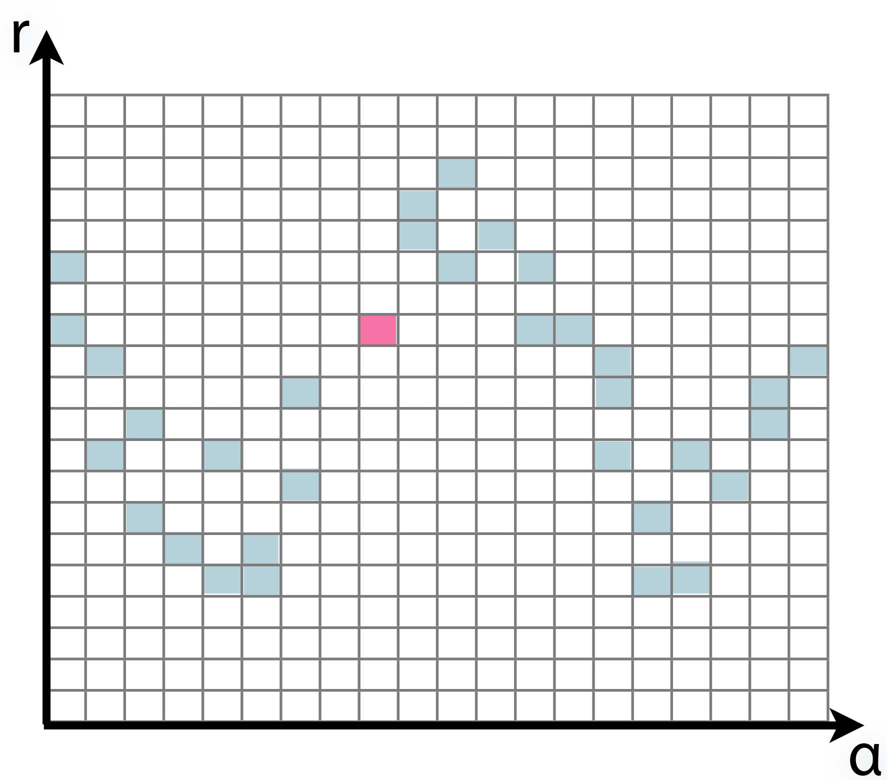
# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation



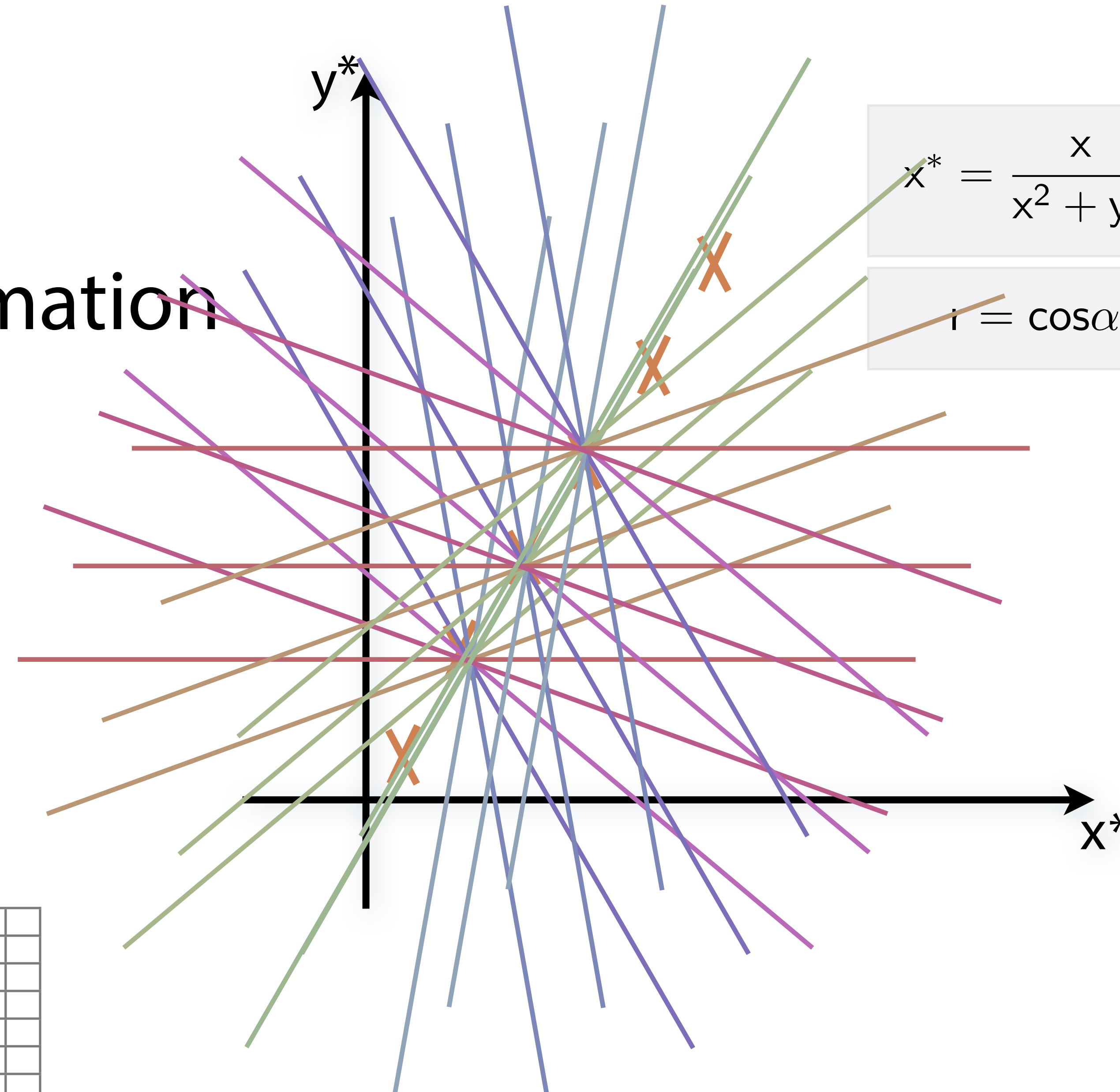

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$



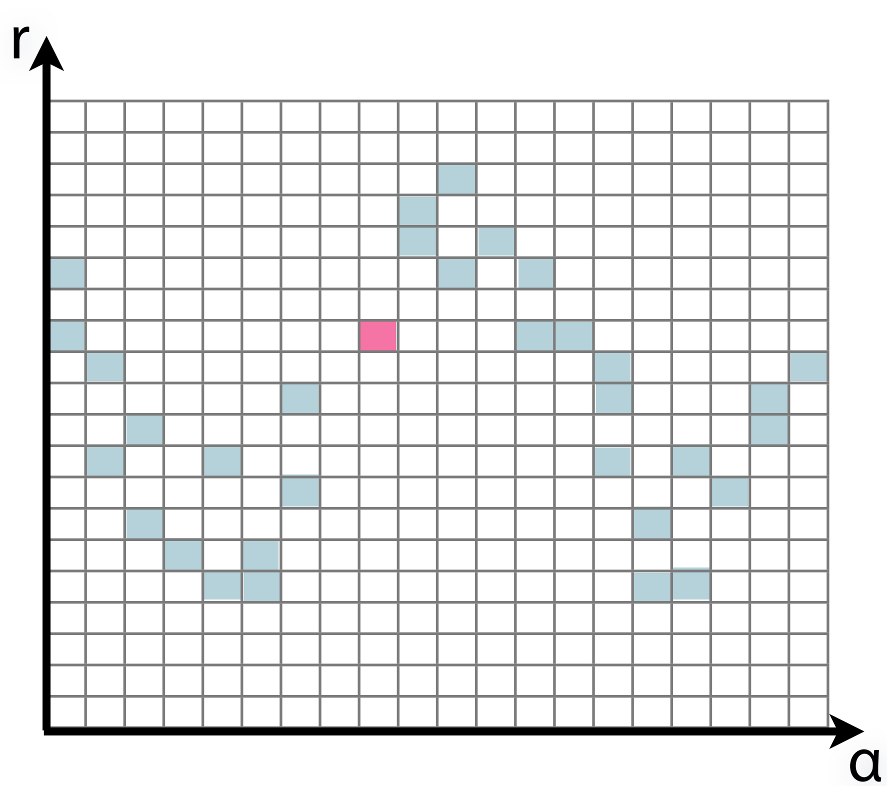
# Hough Transformation

- The Methods
  - Conformal Map
  - Hough Transformation




$$x^* = \frac{x}{x^2 + y^2}, \; y^* = \frac{-y}{x^2 + y^2}$$

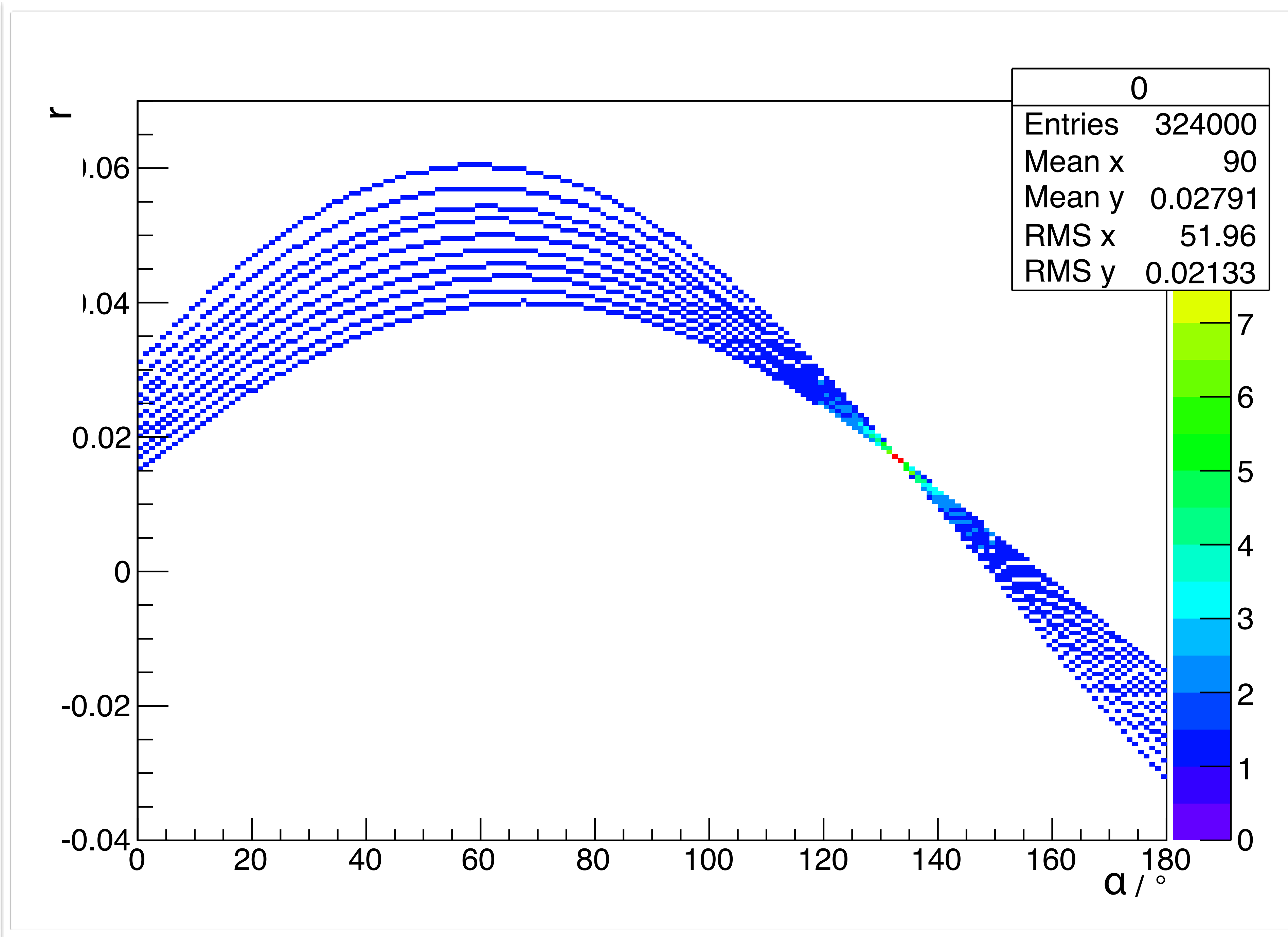
$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$




→ **Bin with highest multiplicity gives track parameters**

# Tracking – Visualization

- Implementation in Thrust

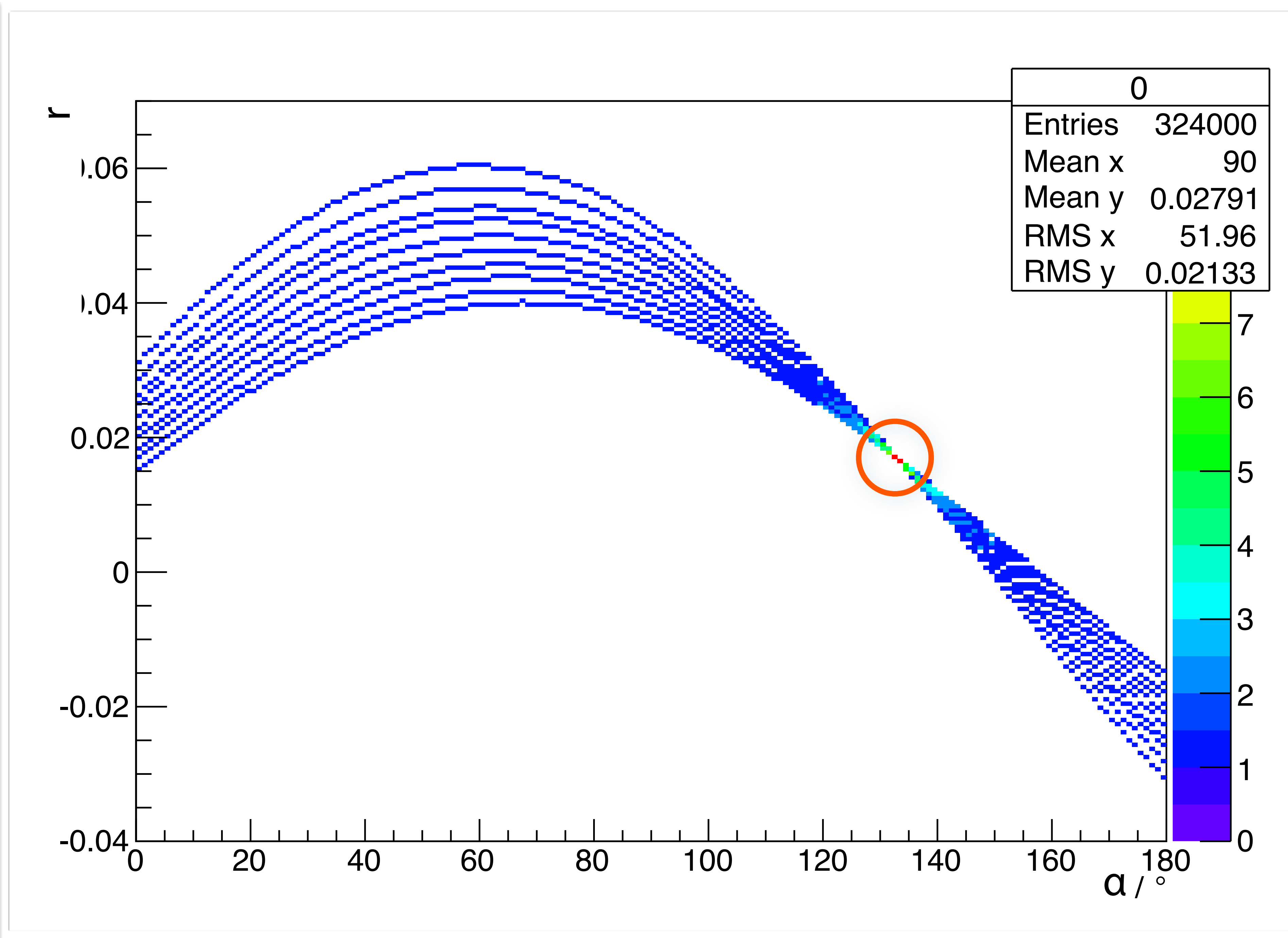



PANDA STT

180 x 180 Grid

# Tracking – Visualization

- Implementation in Thrust

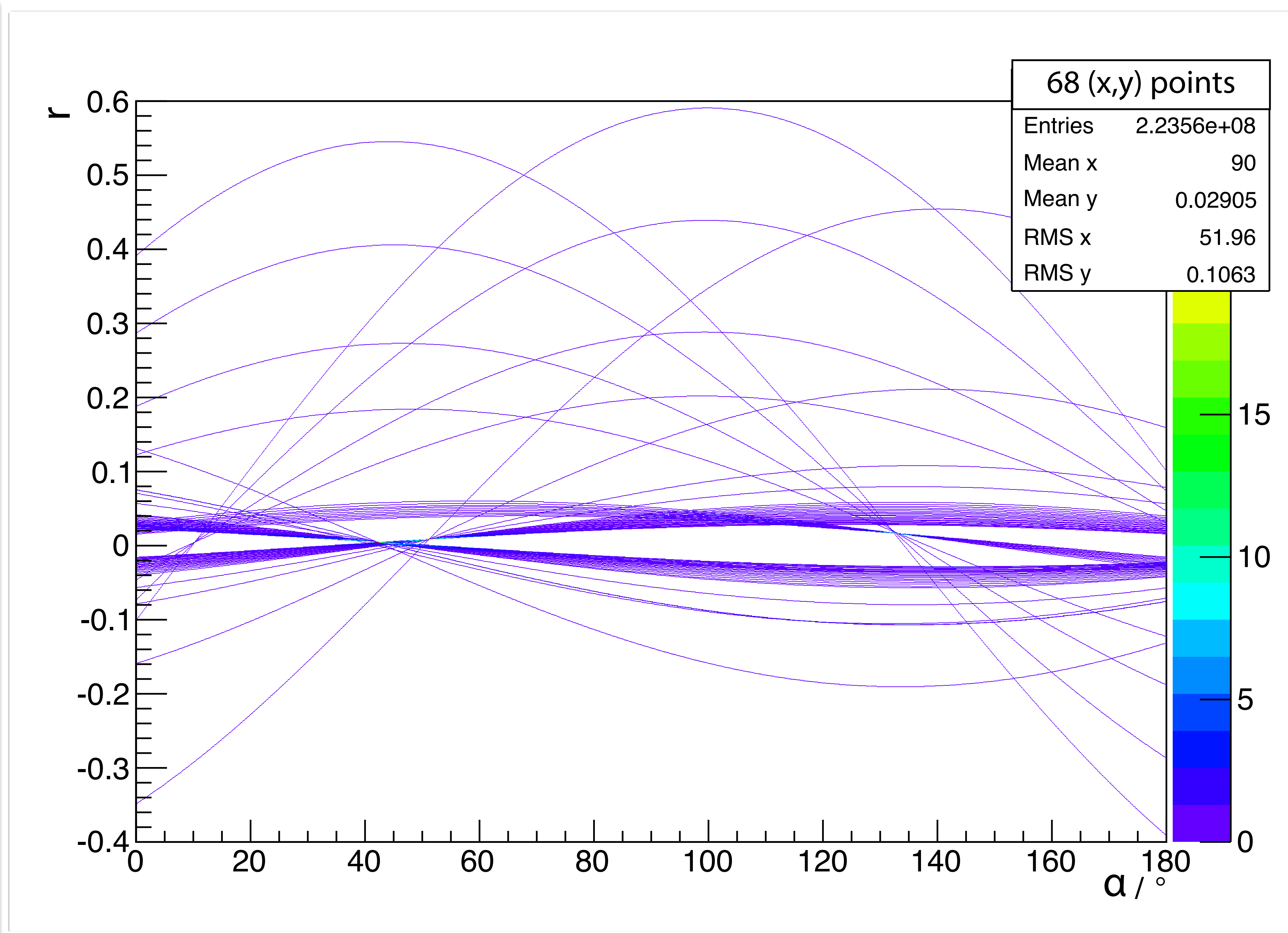



PANDA STT

180 x 180 Grid

# Tracking – Visualization

- Implementation in Thrust

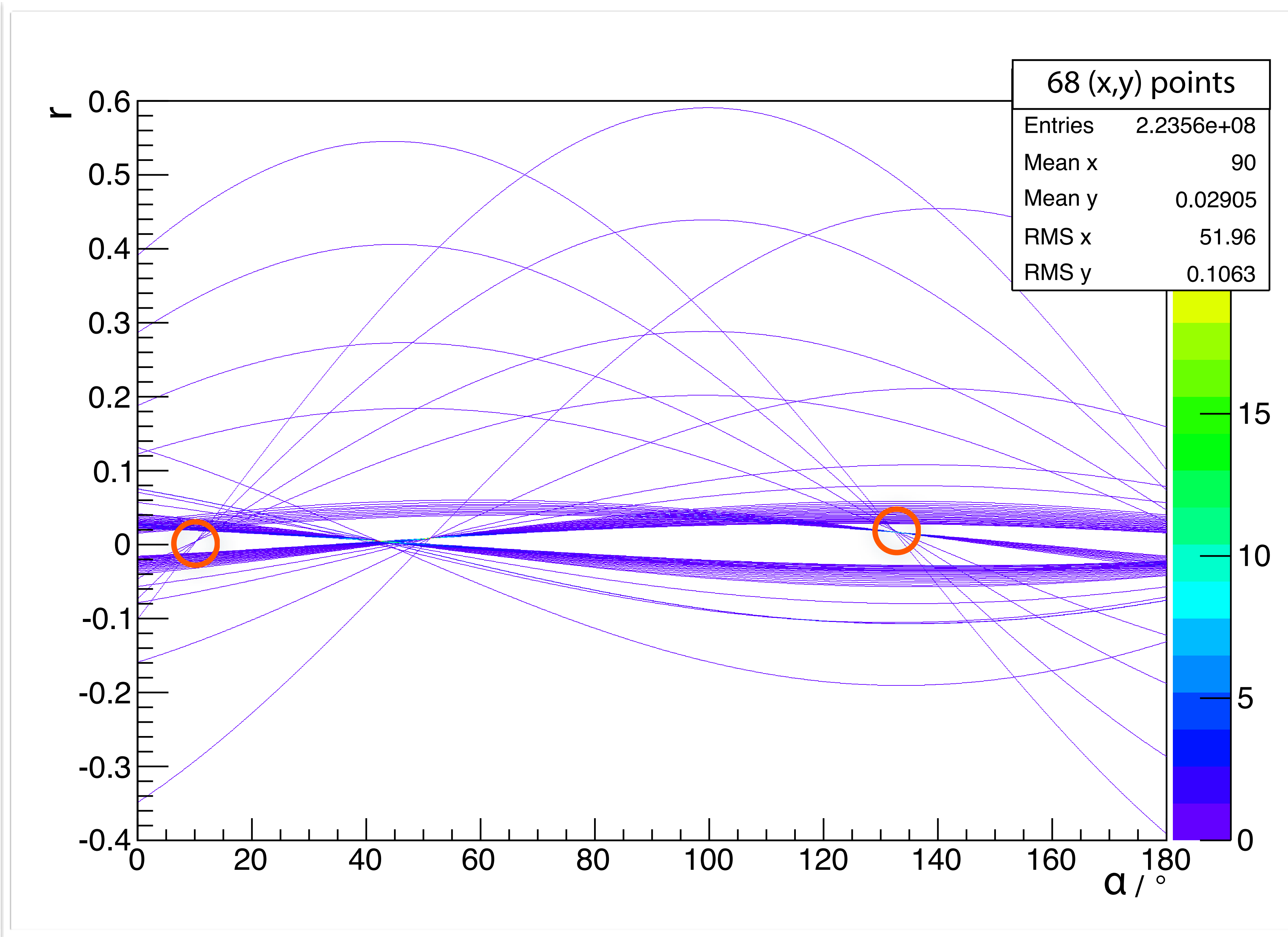



PANDA STT+MVD

1800 x 1800 Grid

# Tracking – Visualization

- Implementation in Thrust

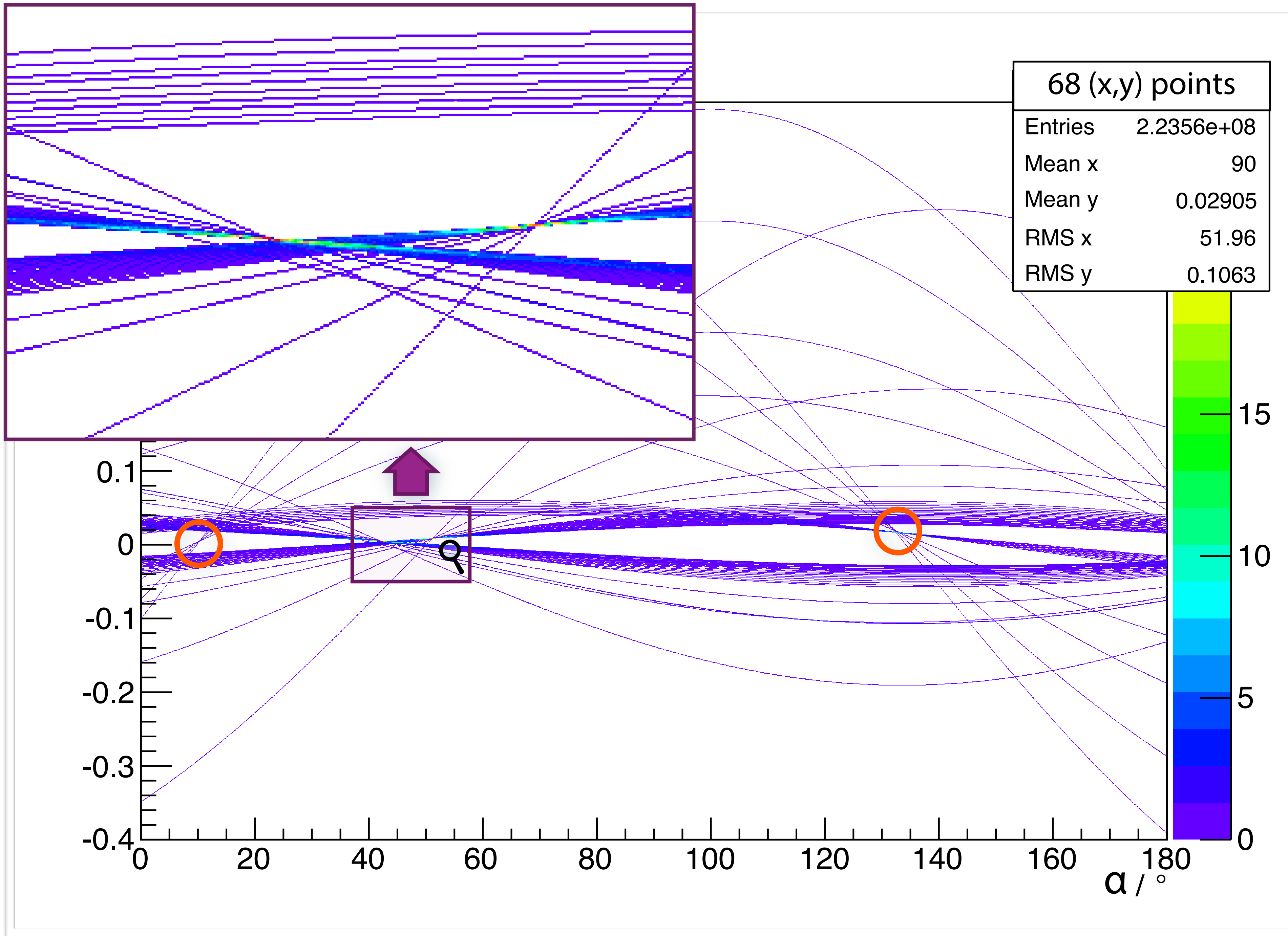



PANDA STT+MVD

1800 x 1800 Grid

# Tracking – Visualization

- Implementation in Thrust

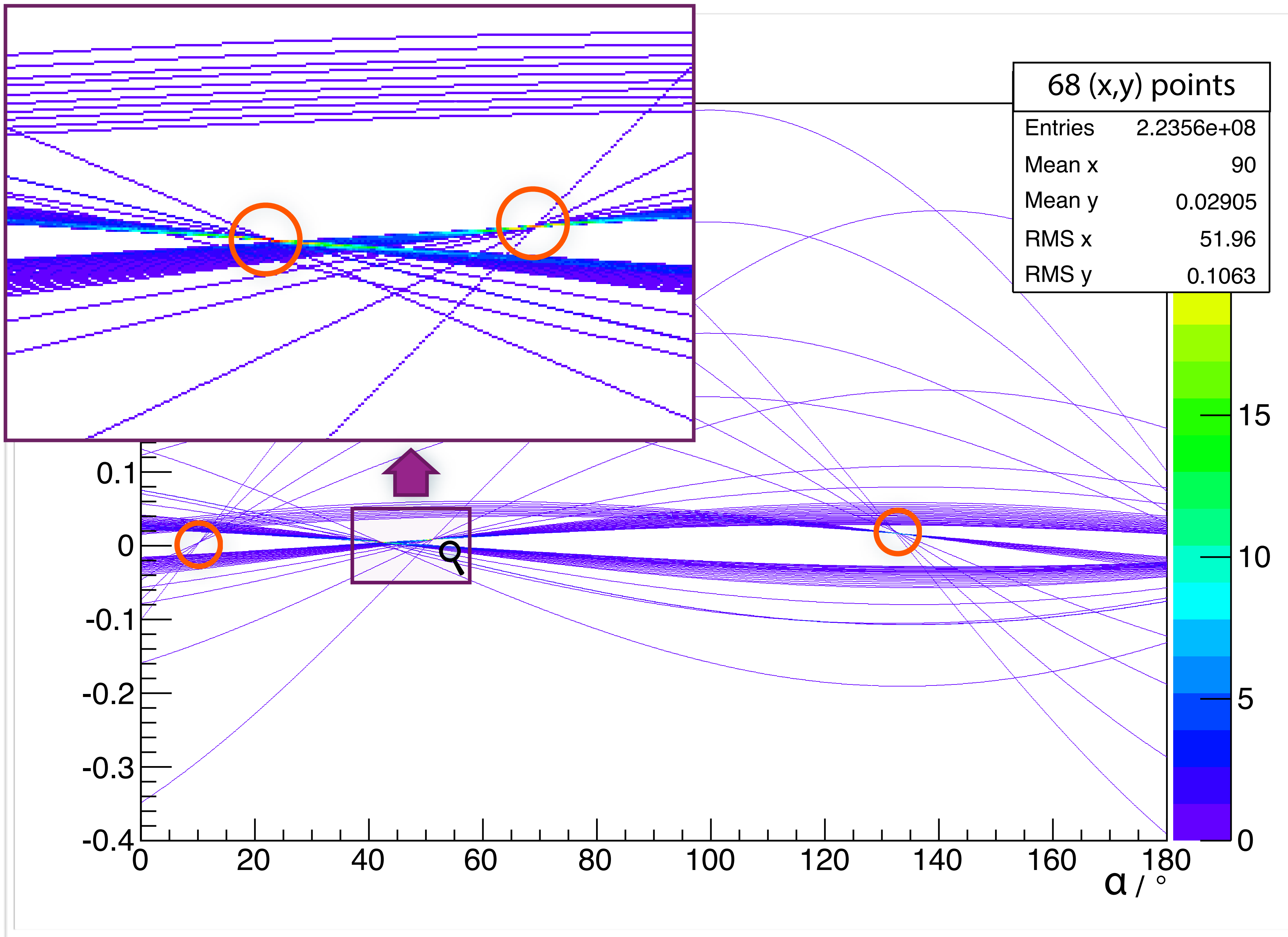



PANDA STT+MVD

1800 x 1800 Grid

# Tracking – Visualization

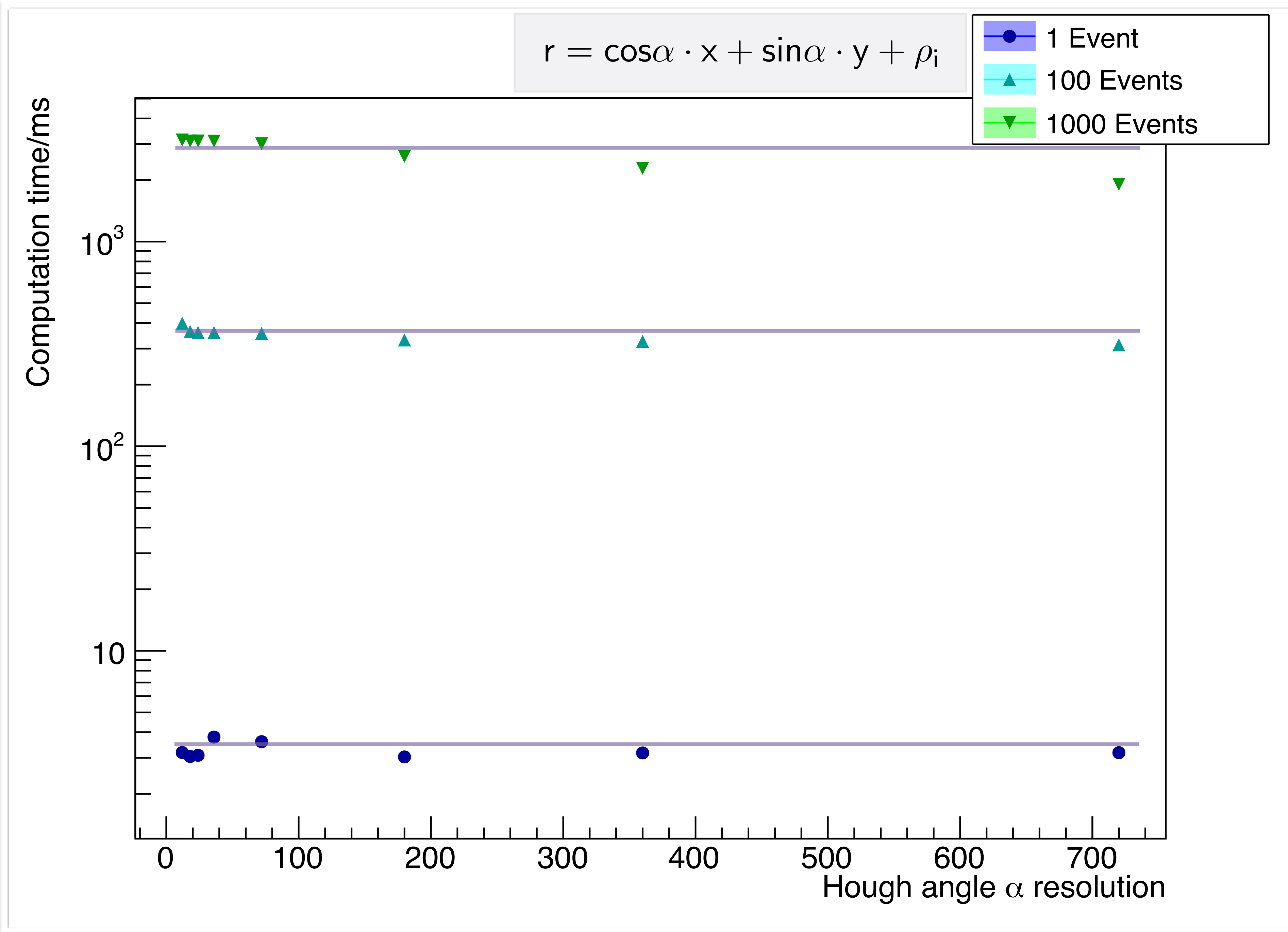
- Implementation in Thrust




PANDA STT+MVD

1800 x 1800 Grid

# Tracking – Computation Time




CM+HT times for STT events (float)

<3 ms/event — ~const in α — 90% HT, 10% CM

# Other Works

- Hough Transform in plain CUDA
  - From GSI/Gießen
- JSC NVIDIA Application Lab
  - Cluster Access (Tesla K20X)
  - Programming Knowledge
  - Direct line to NVIDIA
  - Evaluating different HT implementations

# Other Works — NVIDIA App Lab

### Hough transform (r >= 0)

| HT size (n x n) | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|---|
| threshold | 14 | 14 | 13 | 11 | 10 | 8 |
| # rec. tracks | 6616 | 8218 | 11109 | 15246 | 16930 | 19088 |
| # failed tracks | 13440 | 11838 | 8947 | 4810 | 3126 | 968 |
| # false positives | 5496 | 2925 | 1444 | 698 | 115 | 33 |
| % succ | 33,0 % | 41,0 % | 55,4 % | 76,0 % | 84,4 % | 95,2 % |
| % fail | 67,0 % | 59,0 % | 44,6 % | 24,0 % | 15,6 % | 4,8 % |
| % false pos | 27,4 % | 14,6 % | 7,2 % | 3,5 % | 0,6 % | 0,2 % |
| time, s | 0,047084 | 0,141719 | 0,491206 | 1,706892 | 6,018243 | 22,202198 |
| time/event, s | 9,4168E-06 | 0,00002834 | 0,00009824 | 0,00034138 | 0,00120365 | 0,00444044 |

### Hough transform in shared memory

| HT size (n x n) | 256 | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|---|
| # threads/block | 256 | 256 | 256 | 512 | 1024 |
| threshold | 14 | 14 | 13 | 11 | 10 |
| # rec. tracks | 6616 | 8218 | 11109 | 15246 | 16930 |
| # failed tracks | 13440 | 11838 | 8947 | 4810 | 3126 |
| # false positives | 5496 | 2925 | 1444 | 698 | 115 |
| % succ | 33,0 % | 41,0 % | 55,4 % | 76,0 % | 84,4 % |
| % fail | 67,0 % | 59,0 % | 44,6 % | 24,0 % | 15,6 % |
| % false pos | 27,4 % | 14,6 % | 7,2 % | 3,5 % | 0,6 % |
| time, s | 0,030202 | 0,064211 | 0,150491 | 0,585119 | 2,536495 |
| time/event, s | 6,0404E-06 | 0,00001284 | 0,0000301 | 0,00011702 | 0,0005073 |

# Other Works — NVIDIA App Lab

JÜLICH FORSCHUNGSZENTRUM

panda

**Hough transform (r >= 0)**

| HT size (n x n) | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|---|
| threshold | 14 | 14 | 13 | 11 | 10 | 8 |
| # rec. tracks | 6616 | 8218 | 11109 | 15246 | 16930 | 19088 |
| # failed tracks | 13440 | 11838 | 8947 | 4810 | 3126 | 968 |
| # false positives | 5496 | 2925 | 1444 | 698 | 115 | 33 |
| % succ | 33,0 % | 41,0 % | 55,4 % | 76,0 % | 84,4 % | 95,2 % |
| % fail | 67,0 % | 59,0 % | 44,6 % | 24,0 % | 15,6 % | 4,8 % |
| % false pos | 27,4 % | 14,6 % | 7,2 % | 3,5 % | 0,6 % | 0,2 % |
| time, s | 0,047084 | 0,141719 | 0,491206 | 1,706892 | 6,018243 | 22,202198 |
| time/event, s | 9,4168E-06 | 0,00002834 | 0,00009824 | 0,00034138 | 0,00120365 | 0,00444044 |

**Hough transform in shared memory**

| HT size (n x n) | 256 | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|---|
| # threads/block | 256 | 256 | 256 | 512 | 1024 |
| threshold | 14 | 14 | 13 | 11 | 10 |
| # rec. tracks | 6616 | 8218 | 11109 | 15246 | 16930 |
| # failed tracks | 13440 | 11838 | 8947 | 4810 | 3126 |
| # false positives | 5496 | 2925 | 1444 | 698 | 115 |
| % succ | 33,0 % | 41,0 % | 55,4 % | 76,0 % | 84,4 % |
| % fail | 67,0 % | 59,0 % | 44,6 % | 24,0 % | 15,6 % |
| % false pos | 27,4 % | 14,6 % | 7,2 % | 3,5 % | 0,6 % |
| time, s | 0,030202 | 0,064211 | 0,150491 | 0,585119 | 2,536495 |
| time/event, s | 6,0404E-06 | 0,00001284 | 0,0000301 | 0,00011702 | 0,0005073 |

# Other Works — NVIDIA App Lab

JÜLICH FORSCHUNGSZENTRUM

panda

**Hough transform (r >= 0)**

| HT size (n x n) | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
|---|---|---|---|---|---|---|
| threshold | 14 | 14 | 13 | 11 | 10 | 8 |
| # rec. tracks | 6616 | 8218 | 11109 | 15246 | 16930 | 19088 |
| # failed tracks | 13440 | 11838 | 8947 | 4810 | 3126 | 968 |
| # false positives | 5496 | 2925 | 1444 | 698 | 115 | 33 |
| % succ | 33,0 % | 41,0 % | 55,4 % | 76,0 % | 84,4 % | 95,2 % |
| % fail | 67,0 % | 59,0 % | 44,6 % | 24,0 % | 15,6 % | 4,8 % |
| % false pos | 27,4 % | 14,6 % | 7,2 % | 3,5 % | 0,6 % | 0,2 % |
| time, s | 0,047084 | 0,141719 | 0,491206 | 1,706892 | 6,018243 | 22,202198 |
| time/event, s | 9,4168E-06 | 0,00002834 | 0,00009824 | 0,00034138 | 0,00120365 | 0,00444044 |

**Hough transform in shared memory**

| HT size (n x n) | 256 | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|---|
| # threads/block | 256 | 256 | 256 | 512 | 1024 |
| threshold | 14 | 14 | 13 | 11 | 10 |
| # rec. tracks | 6616 | 8218 | 11109 | 15246 | 16930 |
| # failed tracks | 13440 | 11838 | 8947 | 4810 | 3126 |
| # false positives | 5496 | 2925 | 1444 | 698 | 115 |
| % succ | 33,0 % | 41,0 % | 55,4 % | 76,0 % | 84,4 % |
| % fail | 67,0 % | 59,0 % | 44,6 % | 24,0 % | 15,6 % |
| % false pos | 27,4 % | 14,6 % | 7,2 % | 3,5 % | 0,6 % |
| time, s | 0,030202 | 0,064211 | 0,150491 | 0,585119 | 2,536495 |
| time/event, s | 6,0404E-06 | 0,00001284 | 0,0000301 | 0,00011702 | 0,0005073 |

# Future

- Hough Transform:
  - STT isochrones
  - Adaptive Hough
  - Peakfinder
  - Comparison to other implementations (FPGA)
- Other Algorithms:
  - Triplet Finder
  - Riemann Track Finder
- NVIDIA Application Lab

# Summary

- Hough Transform in CUDA Thrust
  - Computation time independent of α granularity
- More GPU efforts at PANDA

# Summary

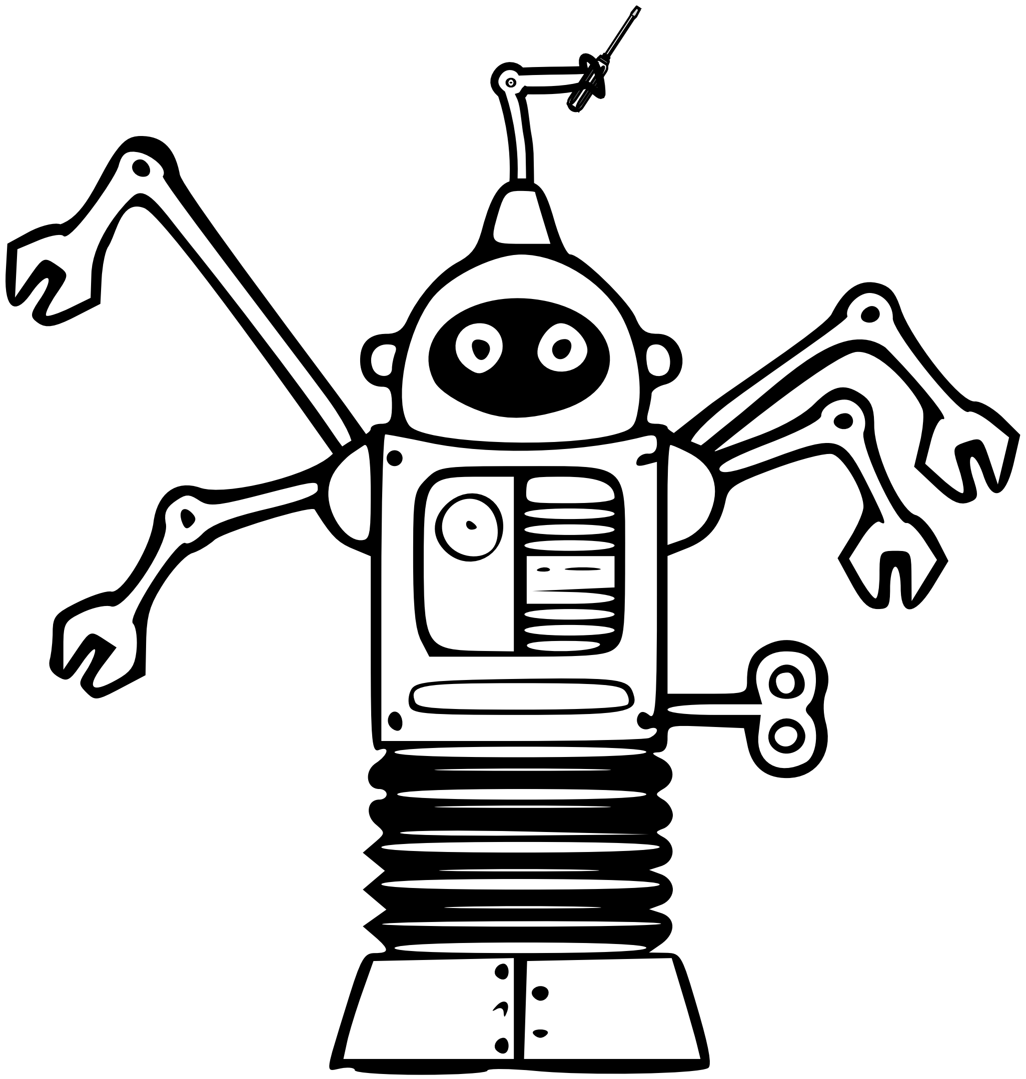
- Hough Transform in CUDA Thrust
  - Computation time independent of α granularity
- More GPU efforts at PANDA

**Thank you!**

Andreas Herten

# Appendix

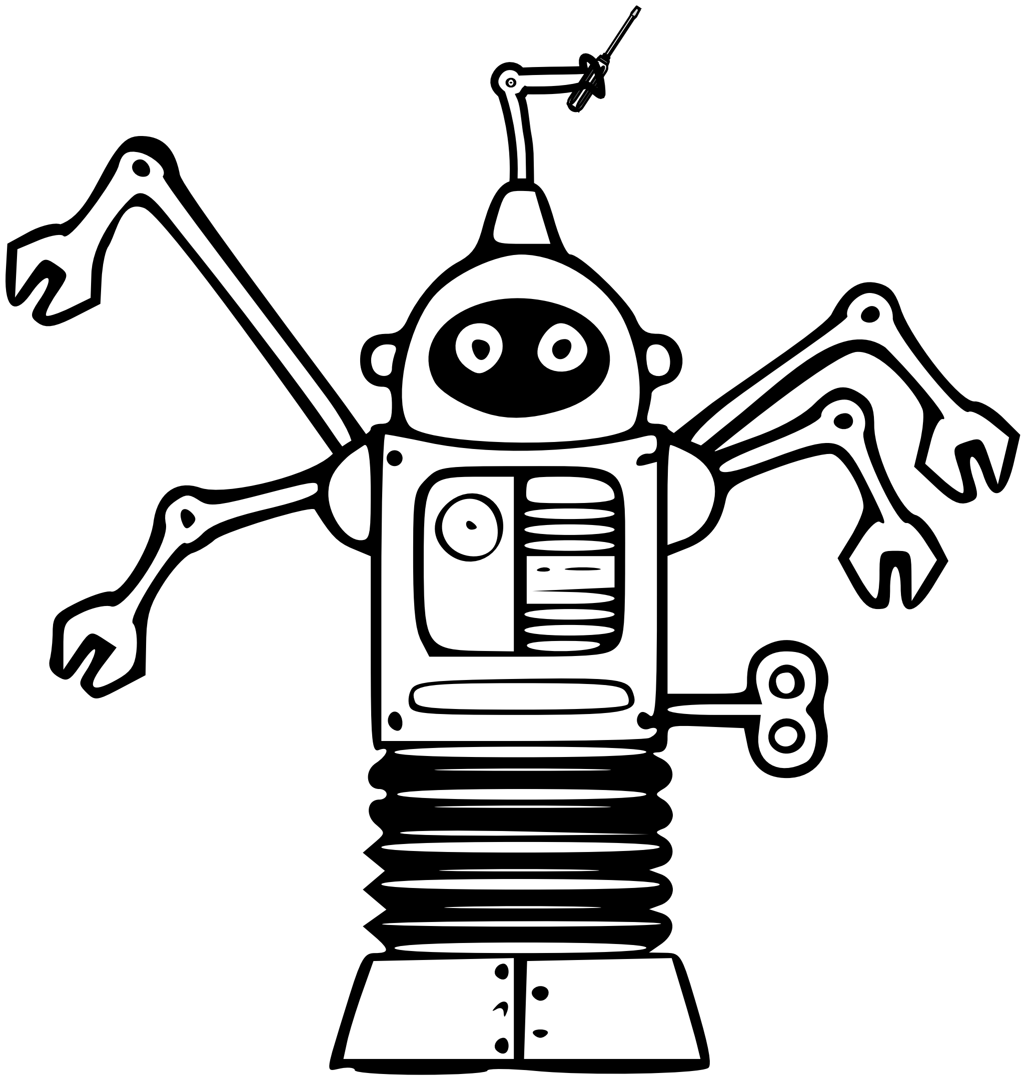
# CPU & GPU Cores



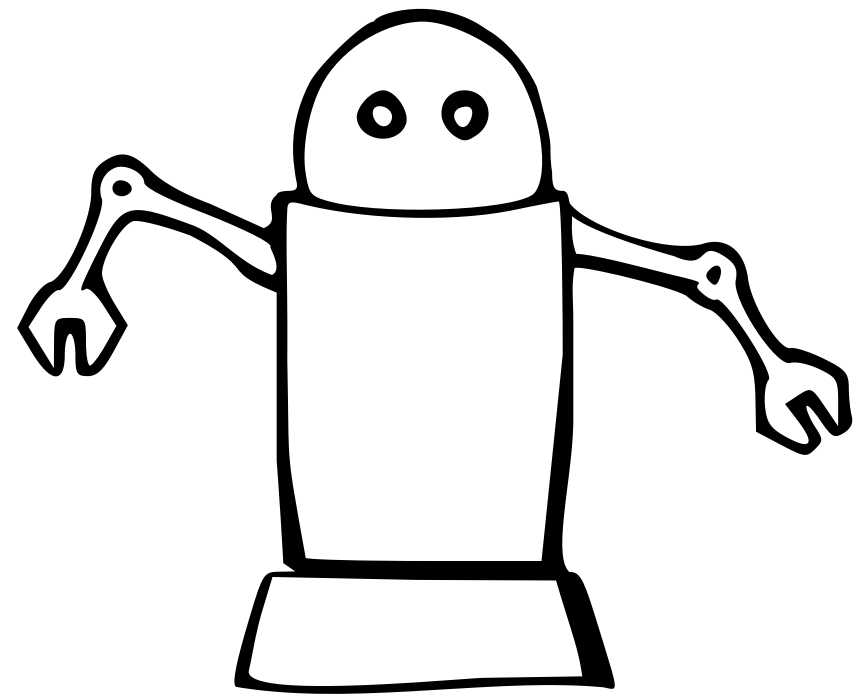
## CPU

- Powerful
- Flexible

# CPU & GPU Cores

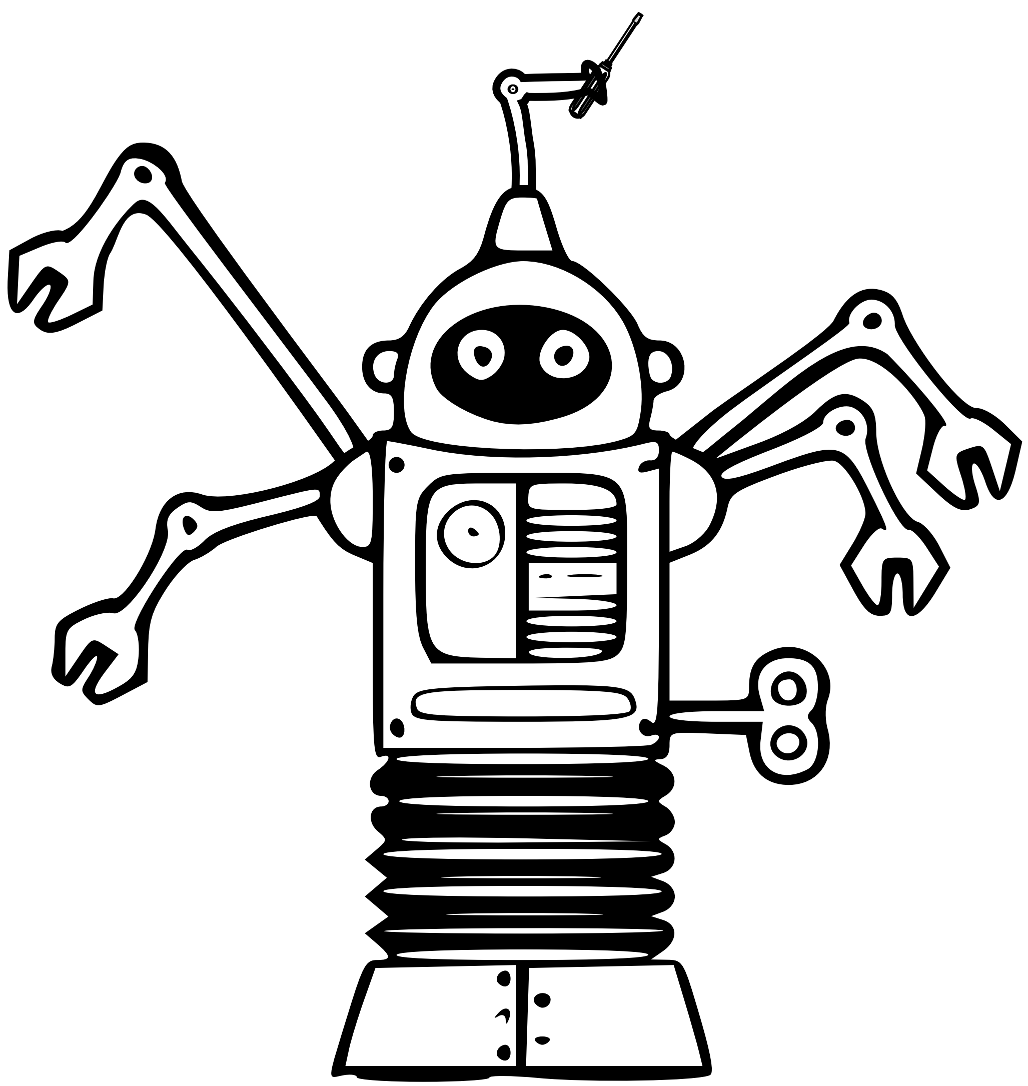


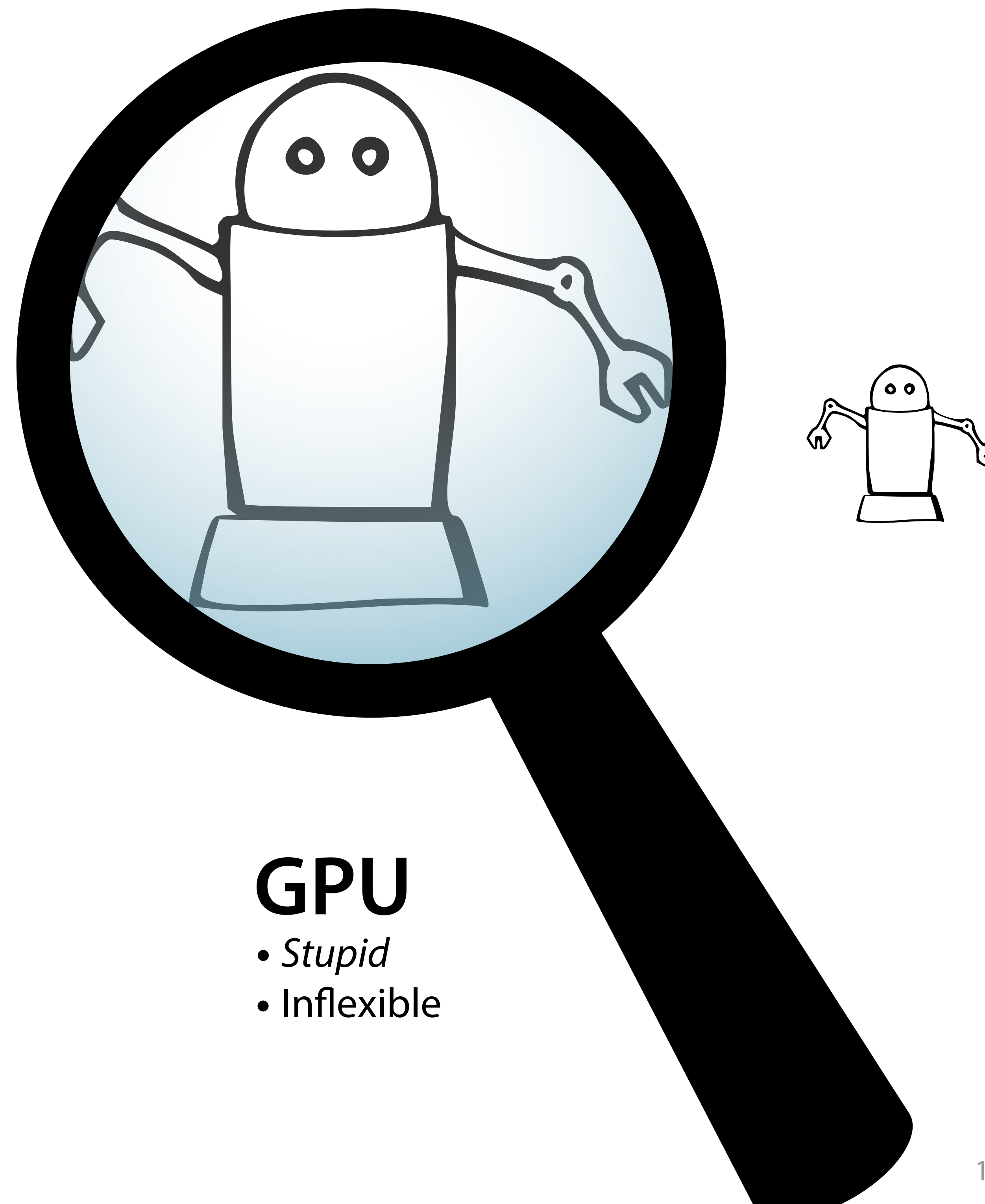
**CPU**
- Powerful
- Flexible



**GPU**
- *Stupid*
- Inflexible

# CPU & GPU Cores

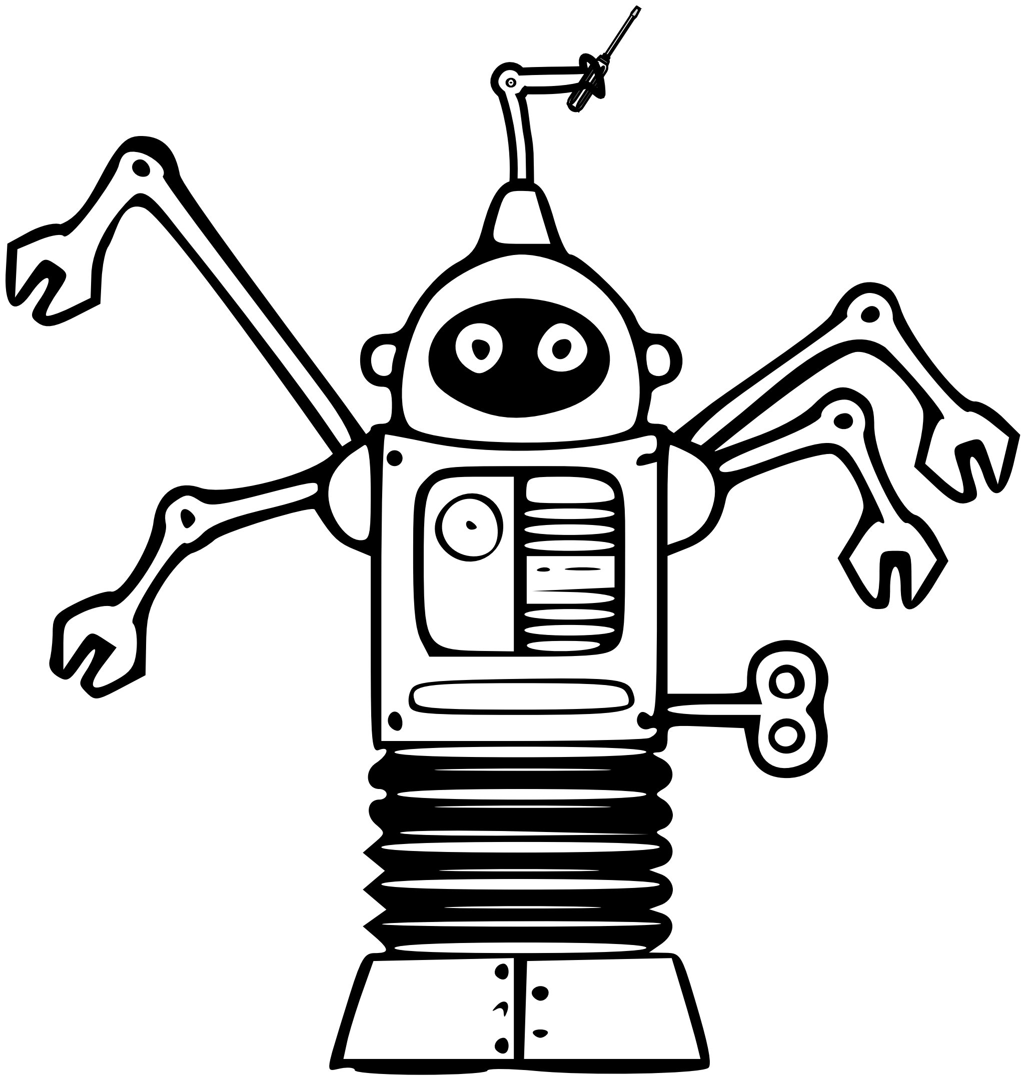




## CPU
- Powerful
- Flexible

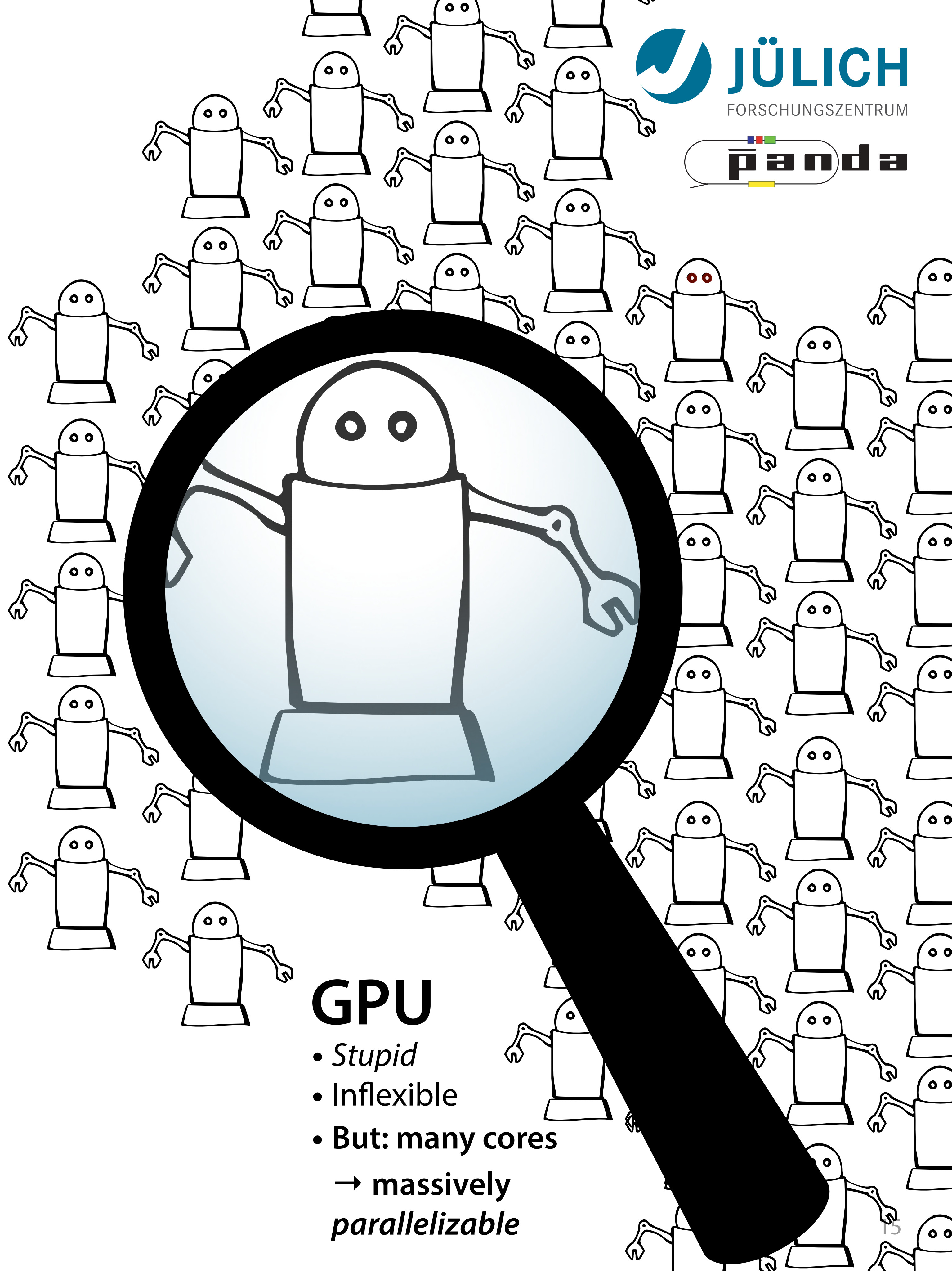
## GPU
- *Stupid*
- Inflexible

# CPU & GPU Cores



## CPU

- Powerful
- Flexible



## GPU

- *Stupid*
- Inflexible
- **But: many cores**
  **→ massively *parallelizable***

# Tracking – Methods

**Goal:** Fit circles

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

## Conformal Mapping

Curved tracks → straight lines

$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

## Hough Transformation

Straight line finding

# Tracking – Methods

**Goal:** Fit circles

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

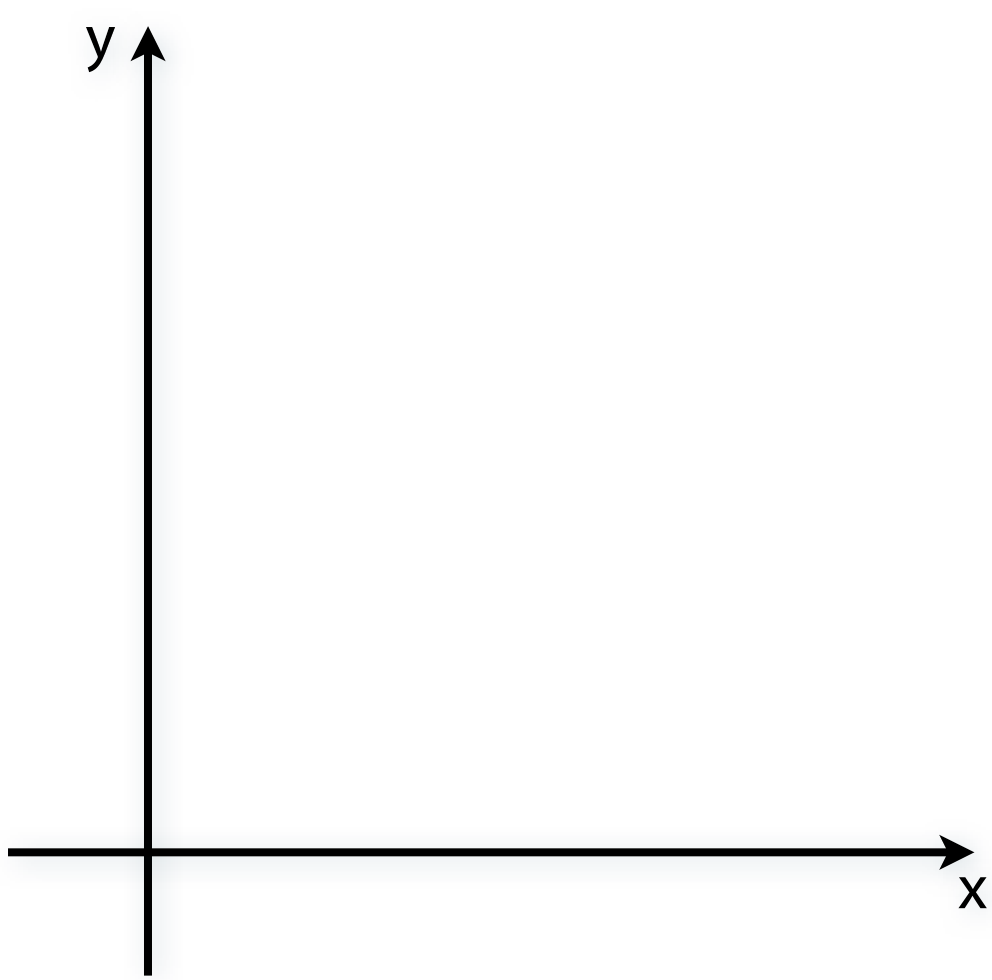
## Conformal Mapping

Curved tracks → straight lines

$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

## Hough Transformation

Straight line finding



# Tracking – Methods

**Goal:** Fit circles

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

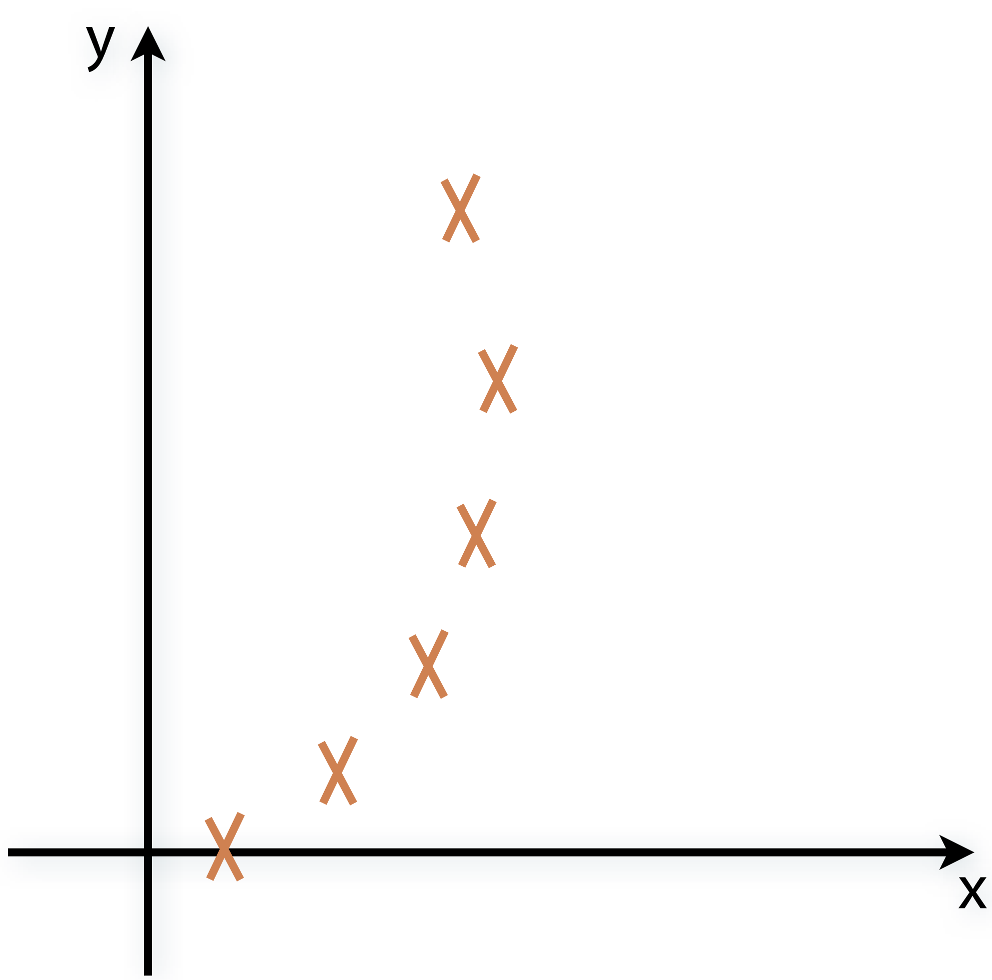
## Conformal Mapping

Curved tracks → straight lines

$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

## Hough Transformation

Straight line finding



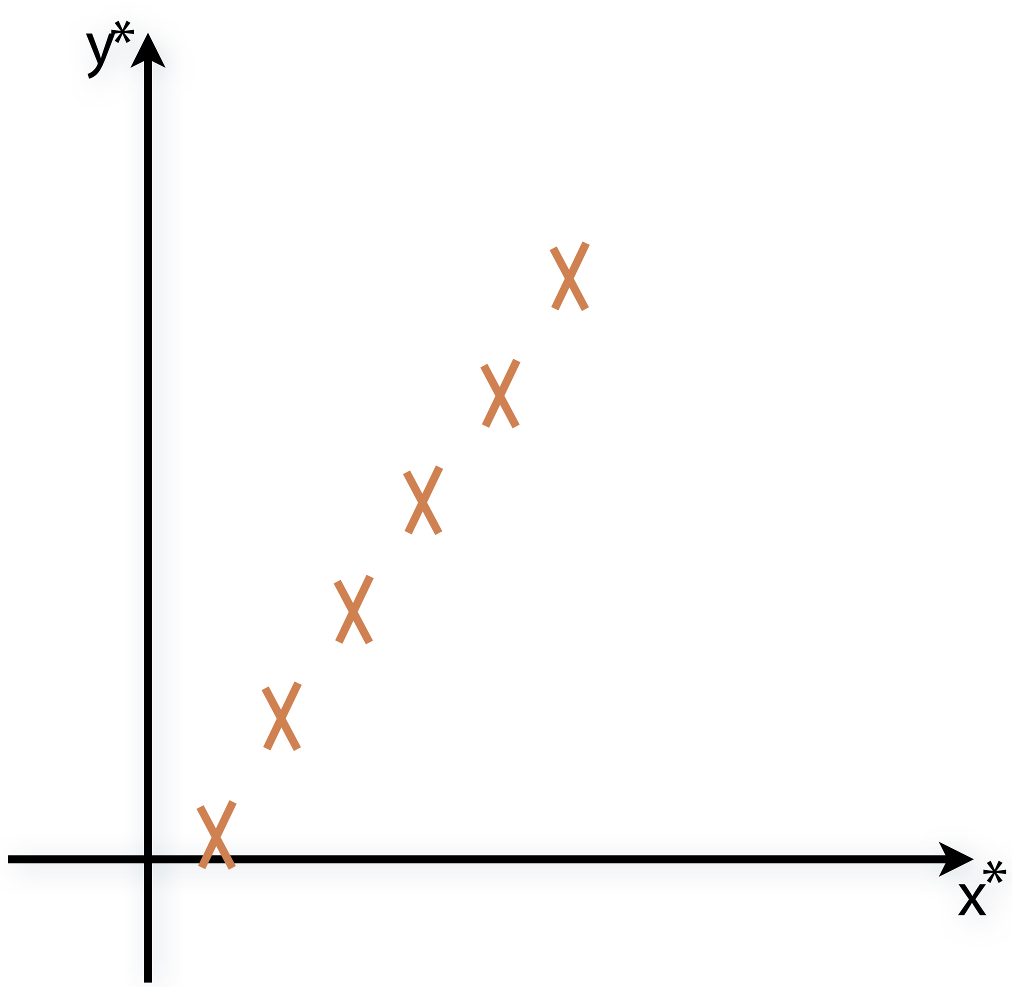
# Tracking – Methods

**Goal:** Fit circles

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

## Conformal Mapping

Curved tracks → straight lines




$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

## Hough Transformation

Straight line finding

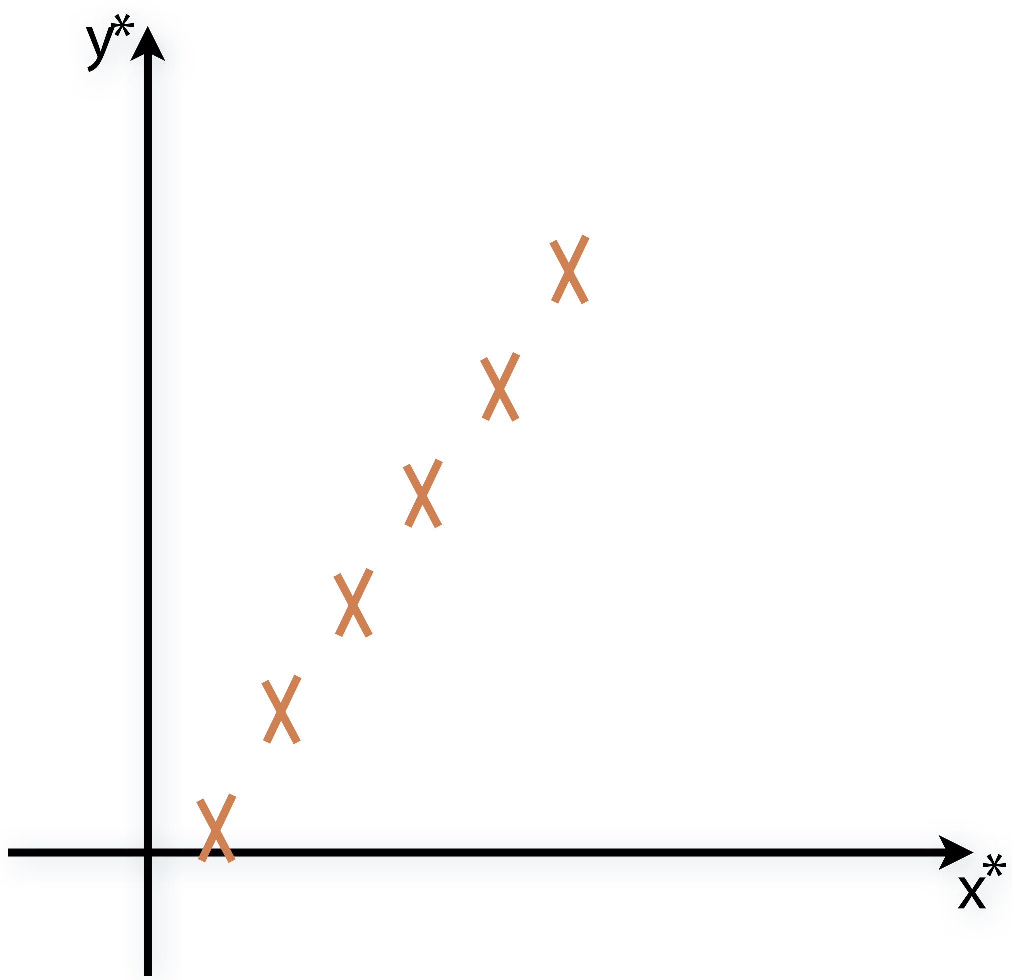
# Tracking – Methods

**Goal:** Fit circles

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

## Conformal Mapping

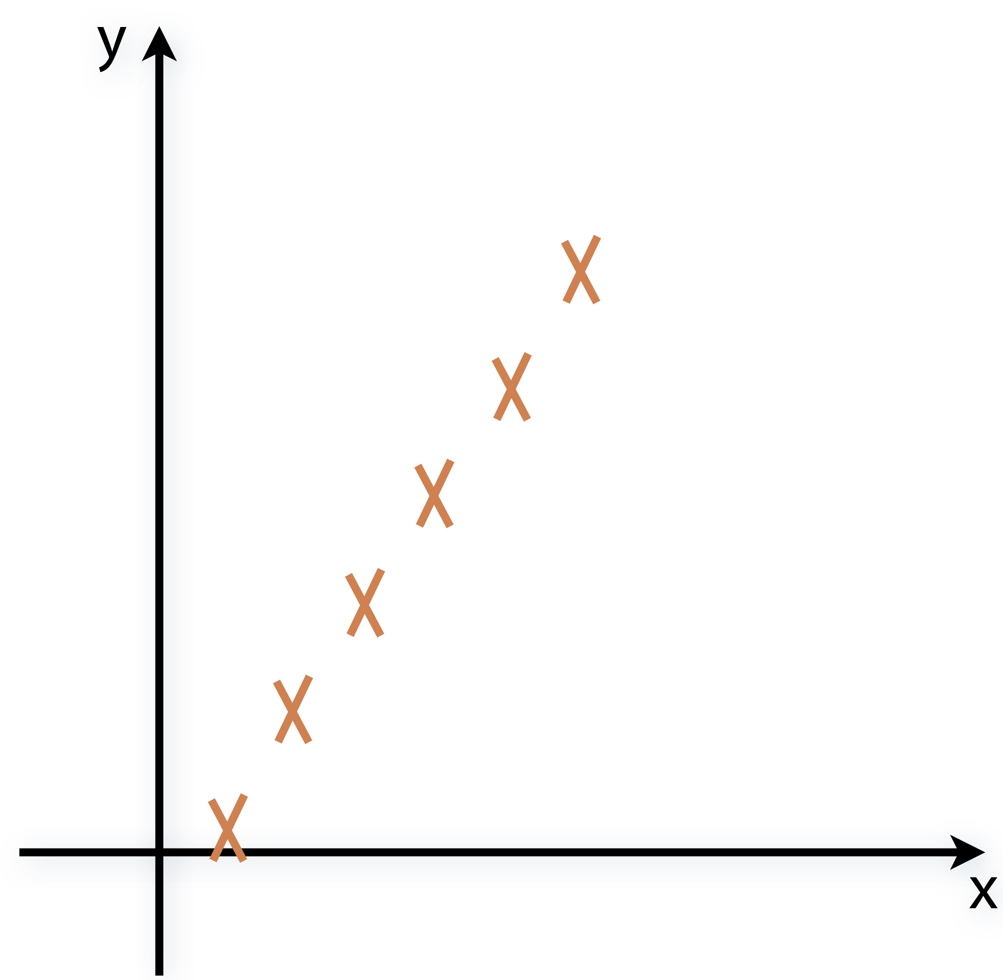
Curved tracks → straight lines




$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

## Hough Transformation

Straight line finding



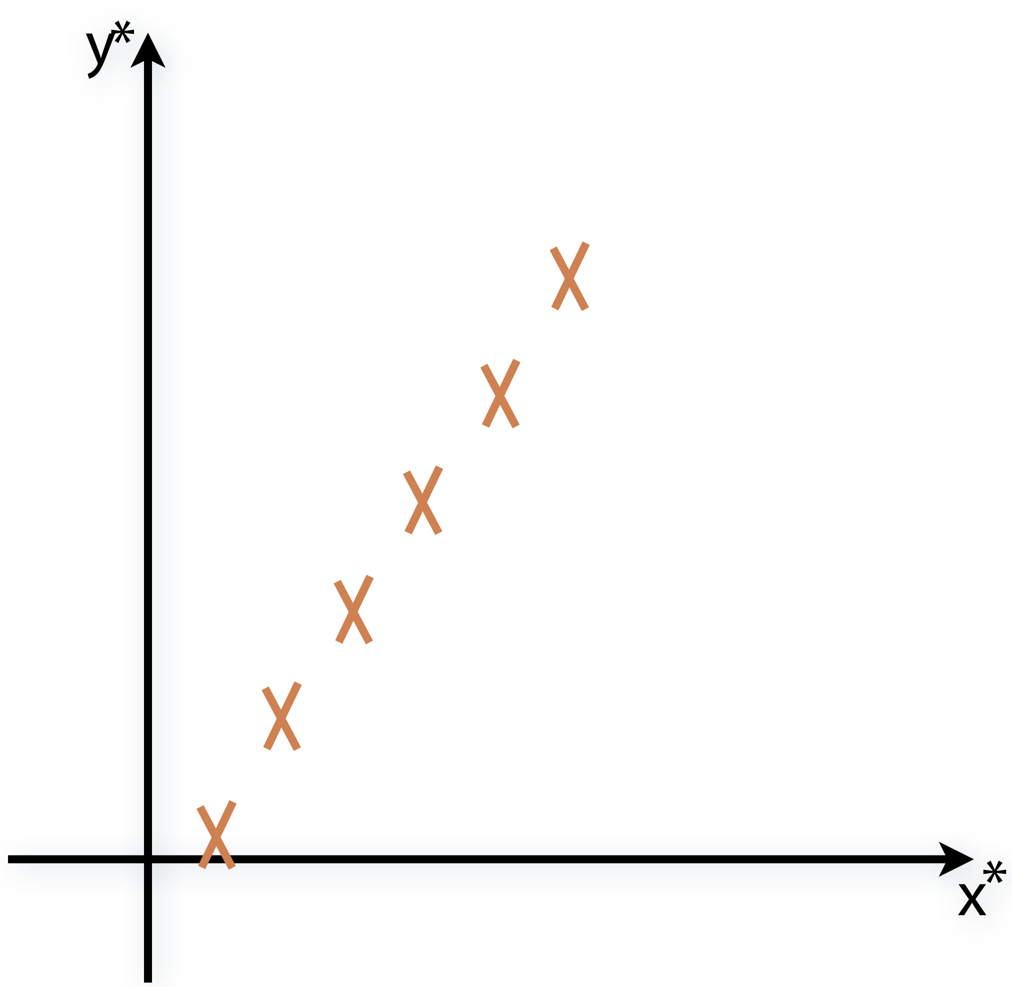
# Tracking – Methods

**Goal:** Fit circles

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

## Conformal Mapping

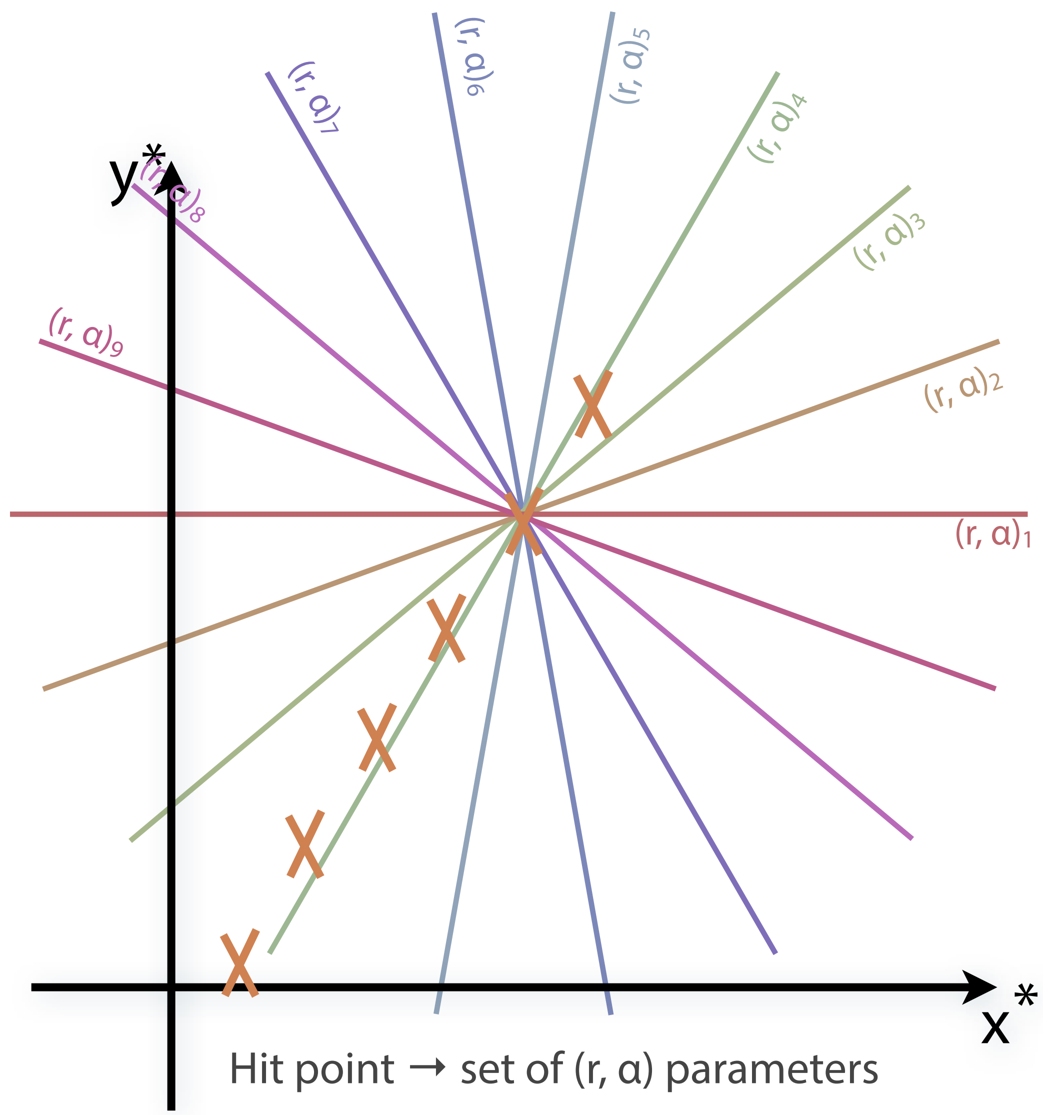
Curved tracks → straight lines




$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

## Hough Transformation

Straight line finding




Hit point → set of (r, α) parameters

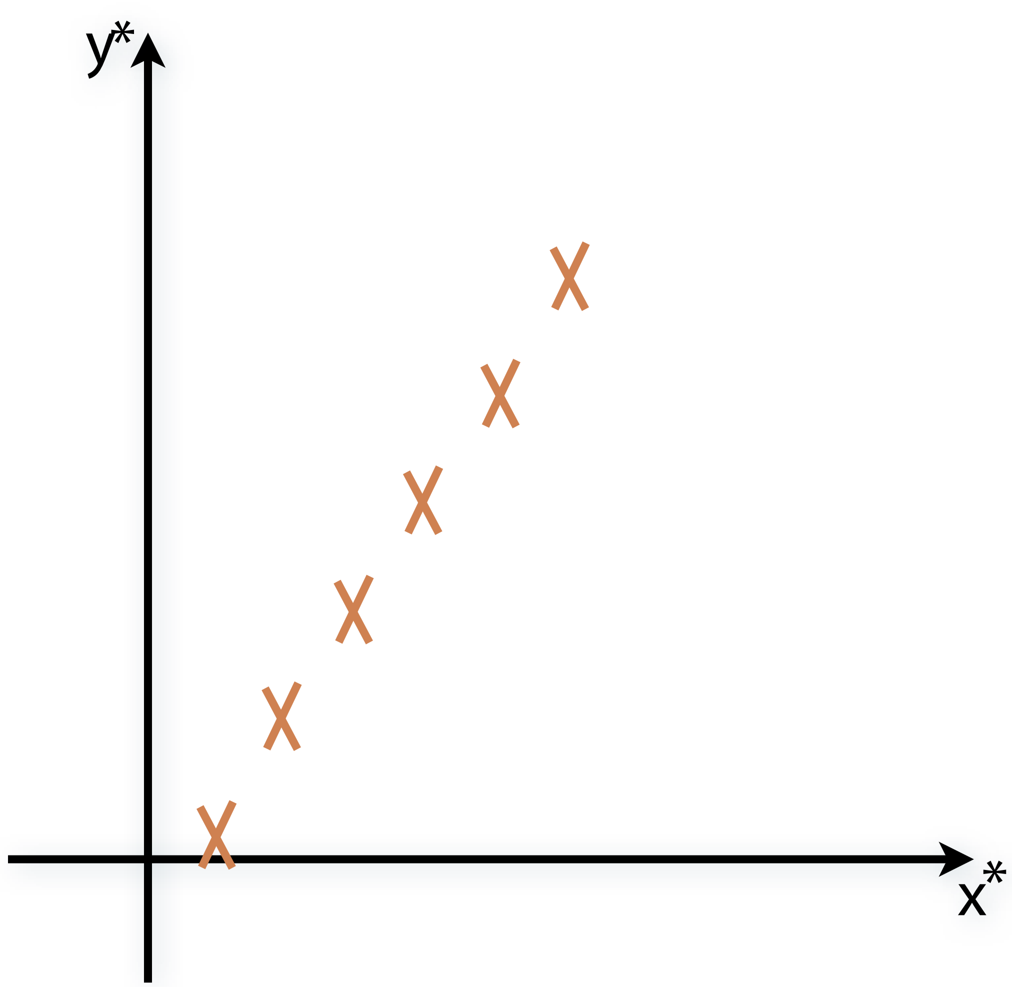
# Tracking – Methods

**Goal:** Fit circles

$$x^* = \frac{x}{x^2 + y^2}, y^* = \frac{-y}{x^2 + y^2}$$

## Conformal Mapping

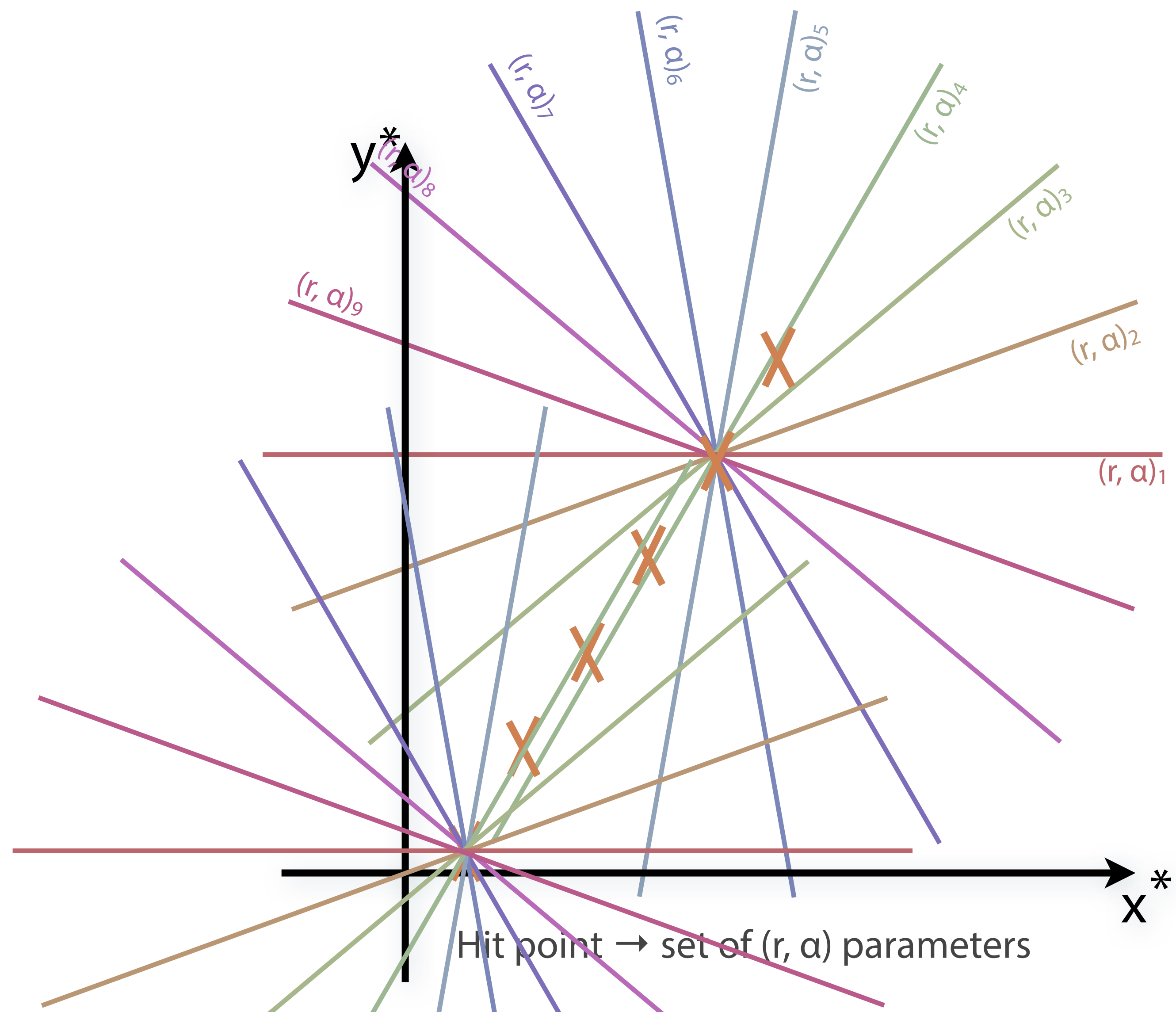
Curved tracks → straight lines



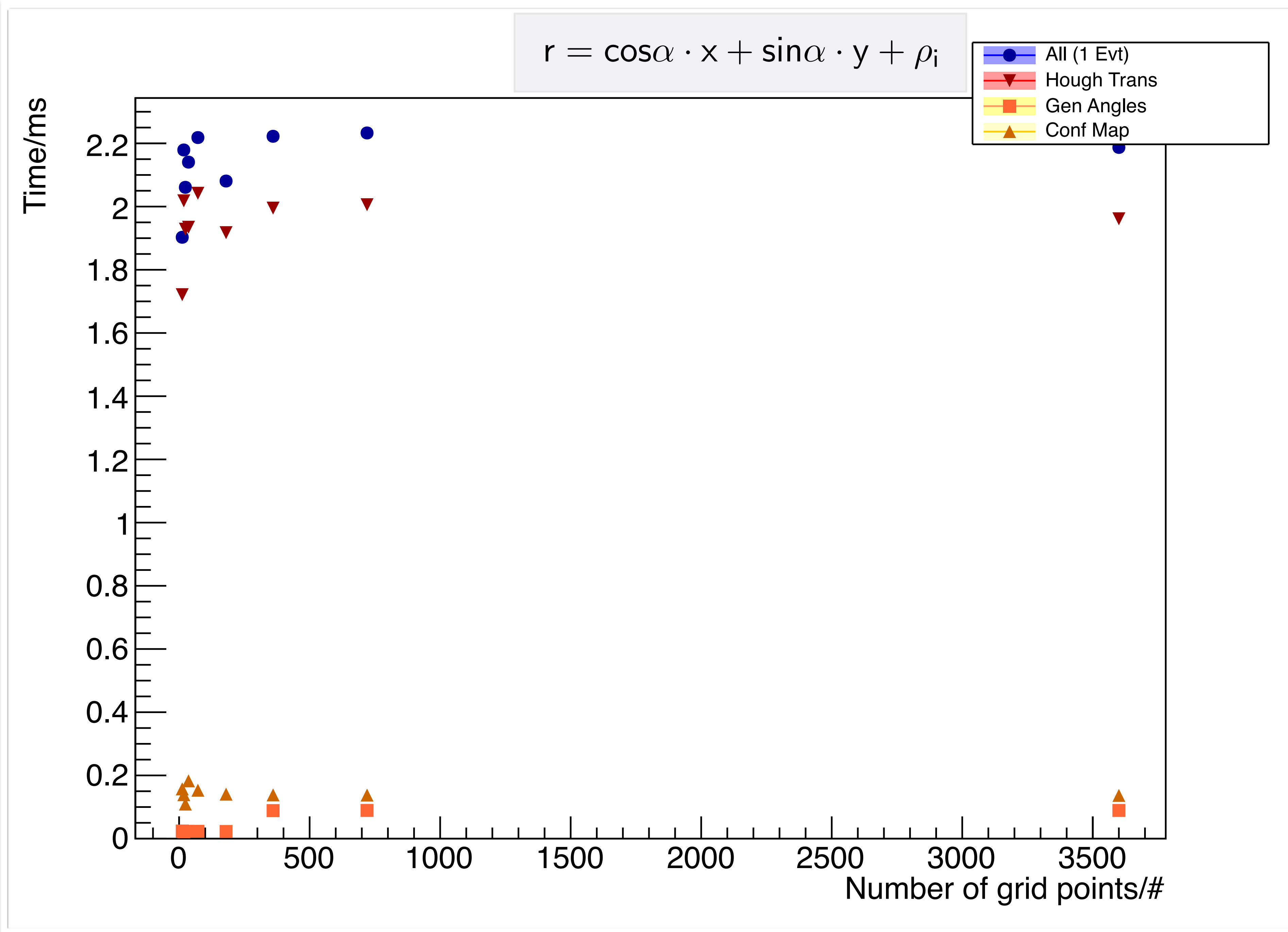

$$r = \cos\alpha \cdot x + \sin\alpha \cdot y$$

## Hough Transformation

Straight line finding



# Tracking – Computation Time




Time consumption for CM+HT (float)

# CUDA – Sample Code

**C**

```
void saxpy_serial(int n,
                  float a,
                  float* x,
                  float* y) {
    for (int i = 0; i < n; ++i)
        y[i] = a * x[i] + y[i];
}

saxpy_serial(4096*256, 2.0, x, y);
```

# CUDA – Sample Code

**C**

```
void saxpy_serial(int n,
                  float a,
                  float* x,
                  float* y) {
    for (int i = 0; i < n; ++i)
        y[i] = a * x[i] + y[i];
}


saxpy_serial(4096*256, 2.0, x, y);
```

**CUDA C**

```
__device__
void saxpy_parallel(int n,
                    float a,
                    float* x,
                    float* y) {
    int i = threadIdx.x + blockIdx.x *
    blockDim.x;
    if (i < n) y[i] = a * x[i] + y[i];
}

saxpy_parallel<<<4096,256>>>(n, 2.0, x, y);
```

# CUDA – Sample Code

### C

```c
void saxpy_serial(int n,
                  float a,
                  float* x,
                  float* y) {
    for (int i = 0; i < n; ++i)
        y[i] = a * x[i] + y[i];
}

saxpy_serial(4096*256, 2.0, x, y);
```

### CUDA C

```c
cudaMalloc( (void**)&y, N * sizeof(float) );
cudaMemcpy(y, y_host, N * sizeof(float),
cudaMemcpyHostToDevice);
__device__
void saxpy_parallel(int n,
                     float a,
                     float* x,
                     float* y) {
    int i = threadIdx.x + blockIdx.x *
    blockDim.x;
    if (i < n) y[i] = a * x[i] + y[i];
}

saxpy_parallel<<<4096,256>>>(n, 2.0, x, y);
```

# CUDA – Sample Code

### C

```
void saxpy_serial(int n,
                  float a,
                  float* x,
                  float* y) {
    for (int i = 0; i < n; ++i)
        y[i] = a * x[i] + y[i];
}


saxpy_serial(4096*256, 2.0, x, y);
```

### CUDA C

```
cudaMalloc( (void**)&y, N * sizeof(float) );
cudaMemcpy(y, y_host, N * sizeof(float),
cudaMemcpyHostToDevice);
__device__
void saxpy_parallel(int n,
                     float a,
                     float* x,
                     float* y) {
    int i = threadIdx.x + blockIdx.x *
    blockDim.x;
    if (i < n) y[i] = a * x[i] + y[i];
}

saxpy_parallel<<<4096,256>>>(n, 2.0, x, y);
```

### CUDA Thrust

```
thrust::host_vector<int> h_vec(100);
std::generate(h_vec.begin(), h_vec.end(), rand);

thrust::device_vector<int> d_vec = h_vec;

int x = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::plus<int>());

thrust::transform(X.begin(), X.end(), Y.begin(), Y.begin(), saxpy_functor(A));
```