JÜLICH
FORSCHUNGSZENTRUM

panda

GPU Tracking Activities at Jülich

PANDA Collaboration Meeting 3-2013 (Bochum)

11. September 2013 | Andreas Herten

Outline

- Introduction & Status of Algorithms
  - Hough Transformation
  - GPU Riemann Track Finder
  - GPU Triplet Finder
- Summaries

JÜLICH
FORSCHUNGSZENTRUM

panda

GPUs

A quick reminder

Benefits of GPUs

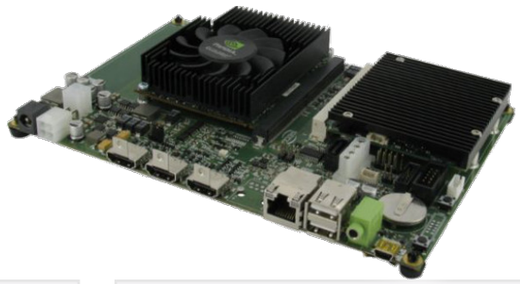
Benefits of GPUs

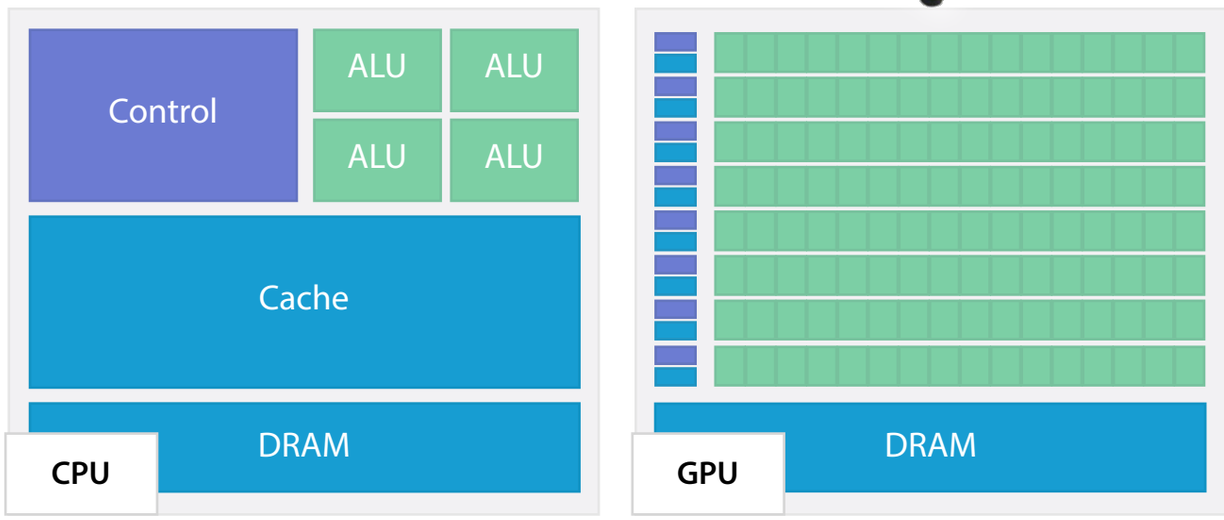
- Extension card to PC

  Although stand-alone GPU boards are coming up

Benefits of GPUs

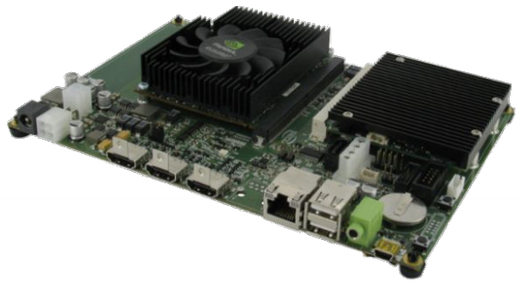
- Extension card to PC
  Although stand-alone GPU boards are coming up
- Many computing cores
  Run in parallel → ***Speed***

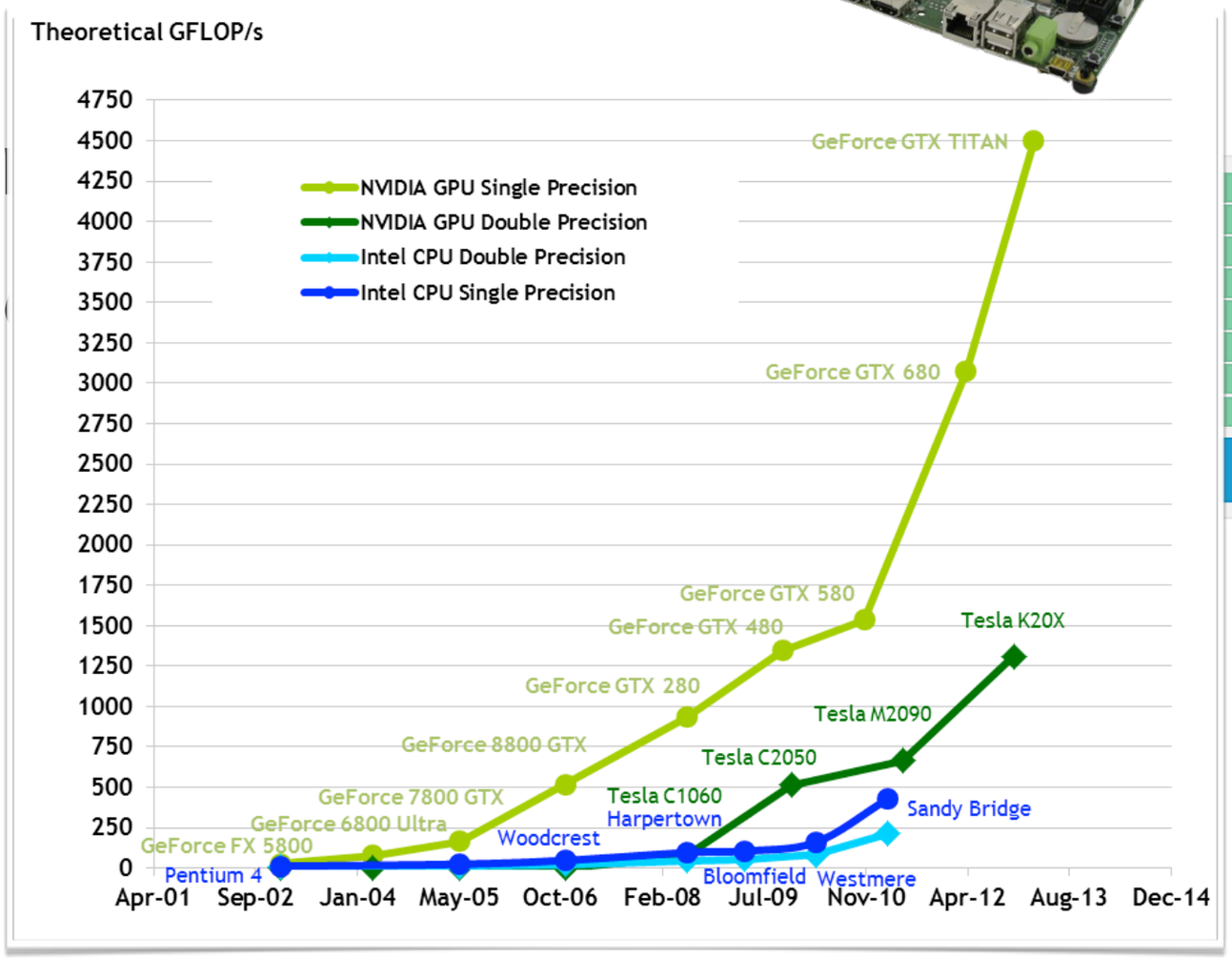




Benefits of GPUs

- Extension card to PC
  Although stand-alone GPU boards are coming up
- Many computing co
  Run in parallel → ***Sp***
- Active hardware
  (& software)
  development



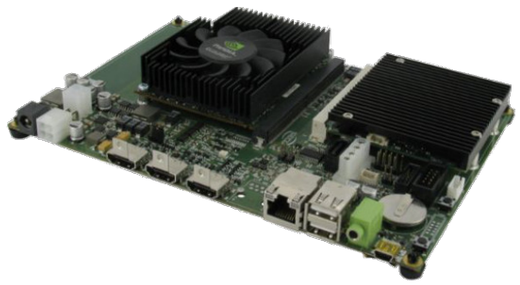


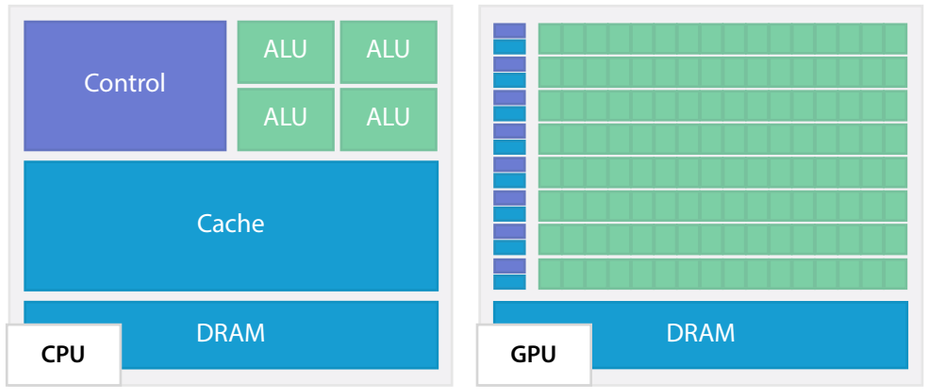
Benefits of GPUs

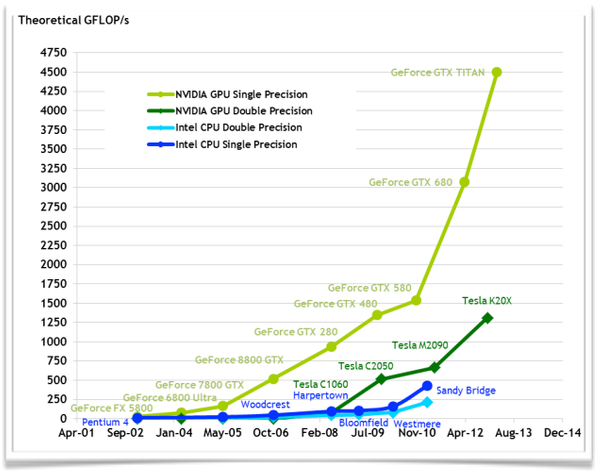
JÜLICH FORSCHUNGSZENTRUM

panda

- Extension card to PC
  Although stand-alone GPU boards are coming up
- Many computing cores
  Run in parallel → ***Speed***
- Active hardware (& software) development
- CUDA C
  - C/C++ extension
  - Mixed host/device code
  - OpenACC 2.0 parallelizer








```
__global__, __device__
```

Thrust, cuBLAS

```
#pragma acc kernels
```

JÜLICH
FORSCHUNGSZENTRUM

panda

ALGORITHMS

Hough Transform

Hough Transform — Algorithm

Hough Transform — Algorithm

1. Conformal Mapping

Hough Transform — Algorithm

1. Conformal Mapping
2. Hit point i (x_i, y_i, ρ_i) → Line parameter r_{ij}

   $\alpha_j \in (0°,360°]$ of desired granularity j

Hough Transform — Algorithm

1. Conformal Mapping
2. Hit point i (x_i, y_i, ρ_i) → Line parameter r_{ij}

   $\alpha_j \in (0°,360°]$ of desired granularity j

$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

Hough Transform — Algorithm

1. Conformal Mapping
2. Hit point i (x_i, y_i, ρ_i) → Line parameter r_{ij}

   $\alpha_j \in (0°,360°]$ of desired granularity j $r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$
3. Fill r_{ij} into histogram

Hough Transform — Algorithm

1. Conformal Mapping
2. Hit point i (x_i, y_i, ρ_i) → Line parameter r_{ij}

   $\alpha_j \in (0°,360°]$ of desired granularity j

$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

3. Fill r_{ij} into histogram
4. Find peak

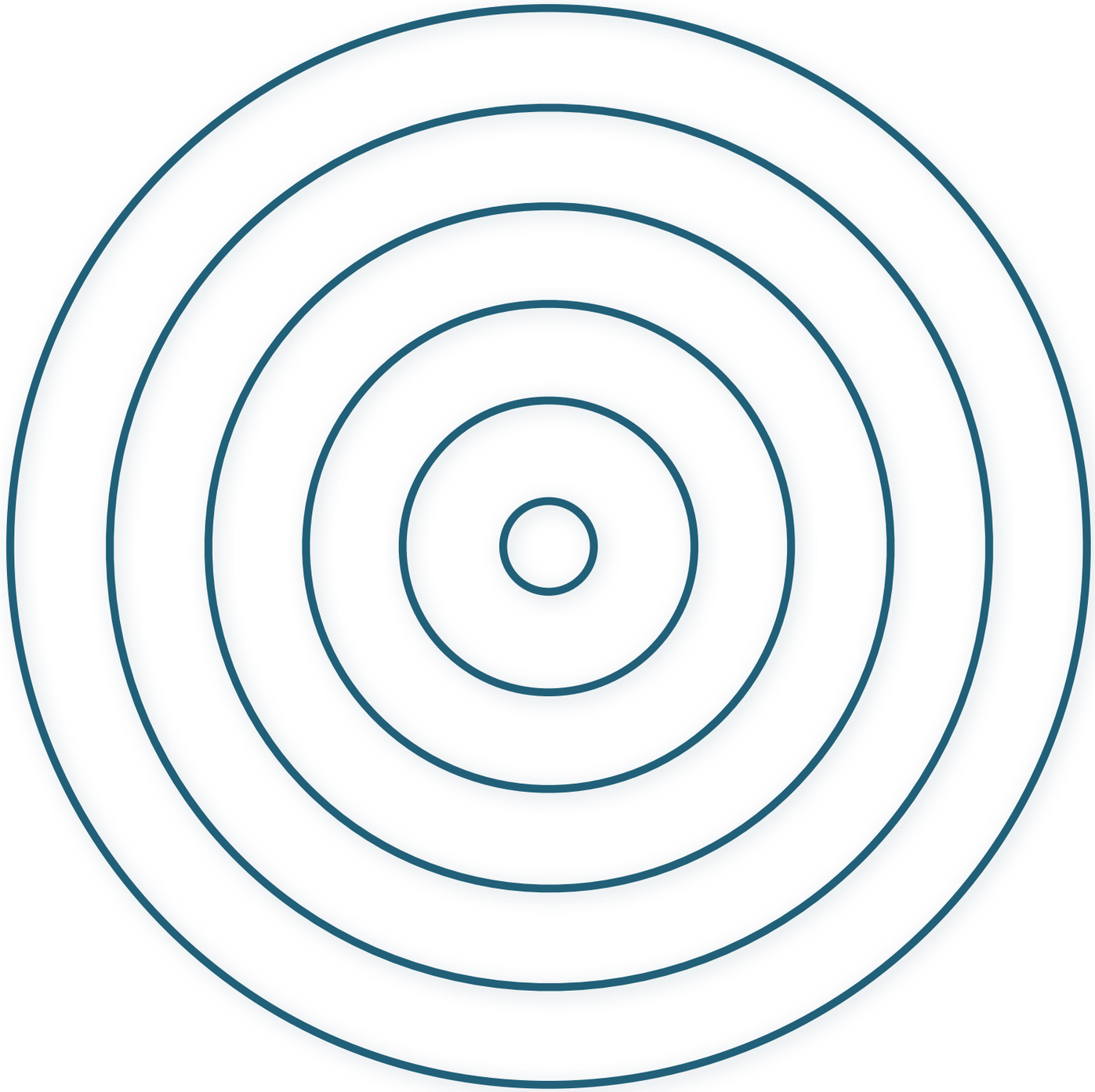
Hough Transform — Algorithm

1. Conformal Mapping
2. Hit point i (x_i, y_i, ρ_i) → Line parameter r_{ij}

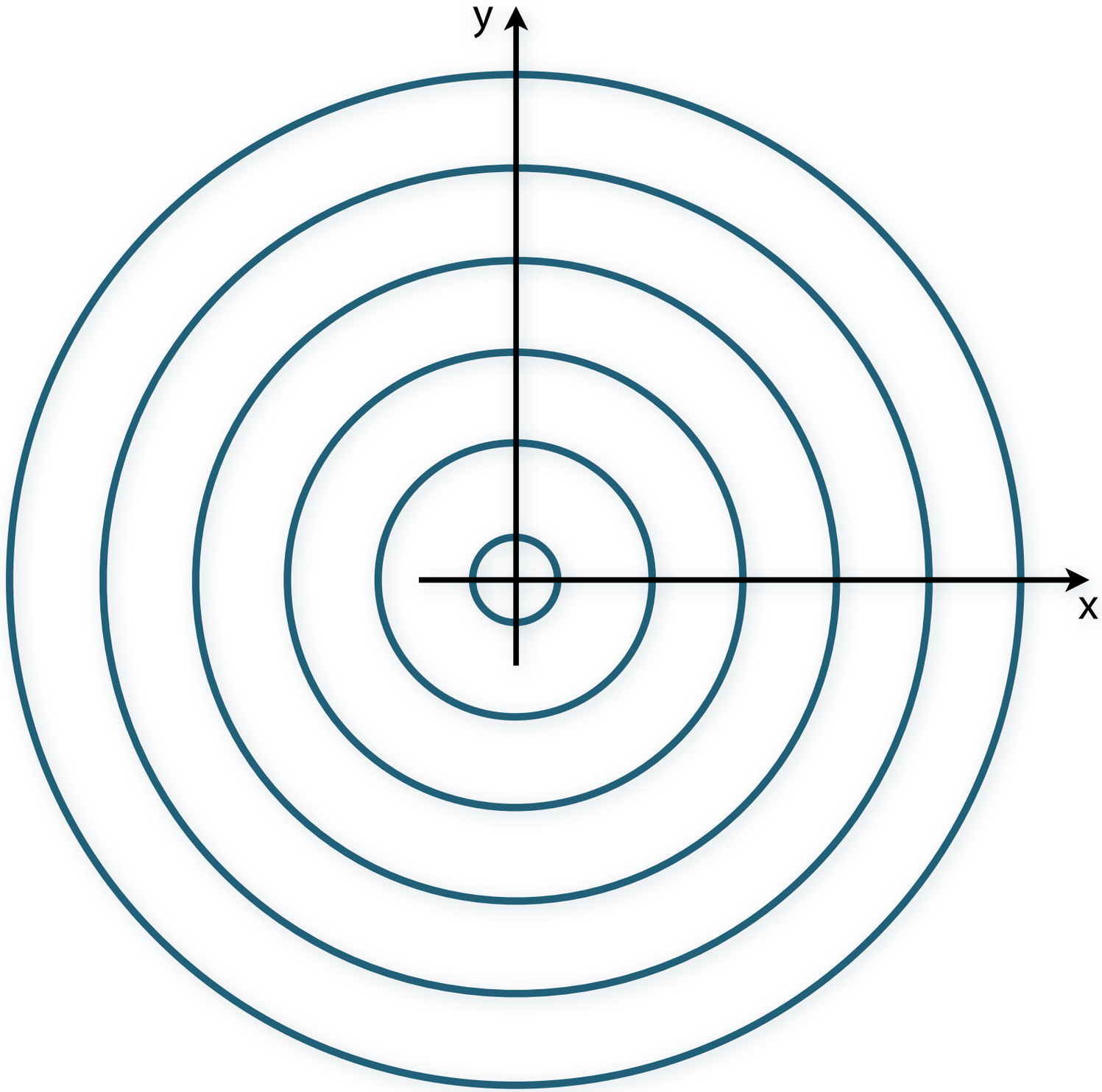
   $\alpha_j \in (0°,360°]$ of desired granularity j

   $$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$
3. Fill r_{ij} into histogram
4. Find peak
5. Extract track parameters (r_j, α_j)

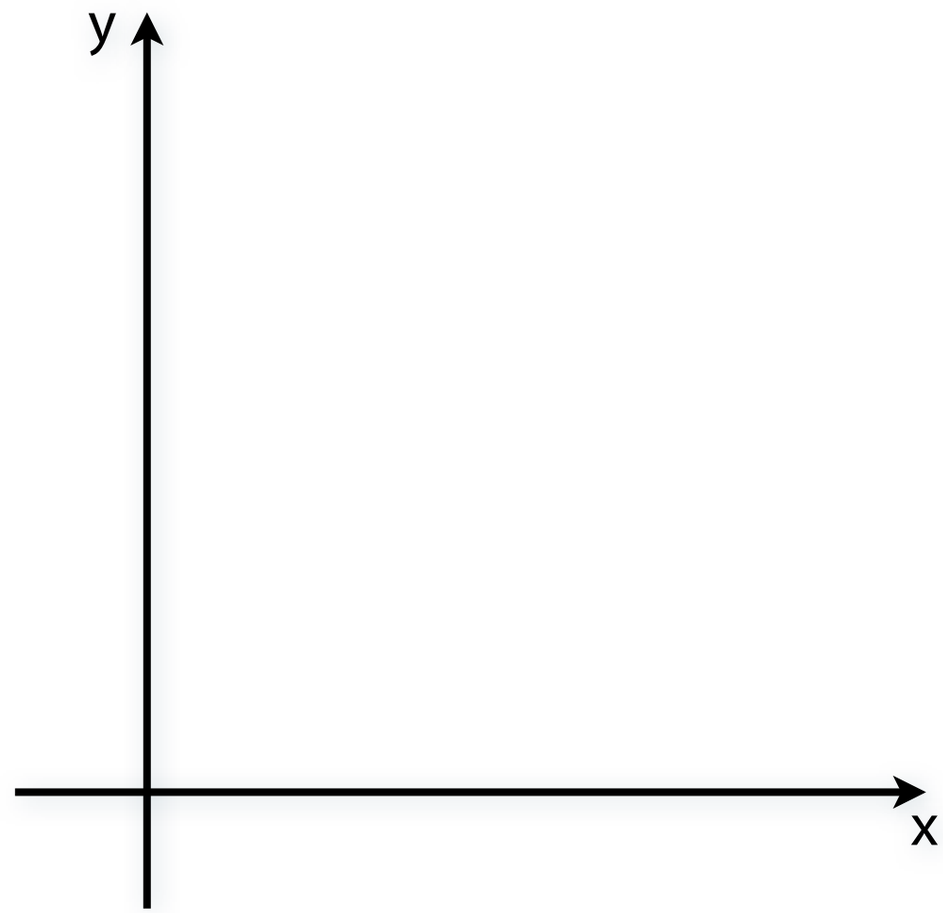
Hough Transform — Principle



Hough Transform — Principle

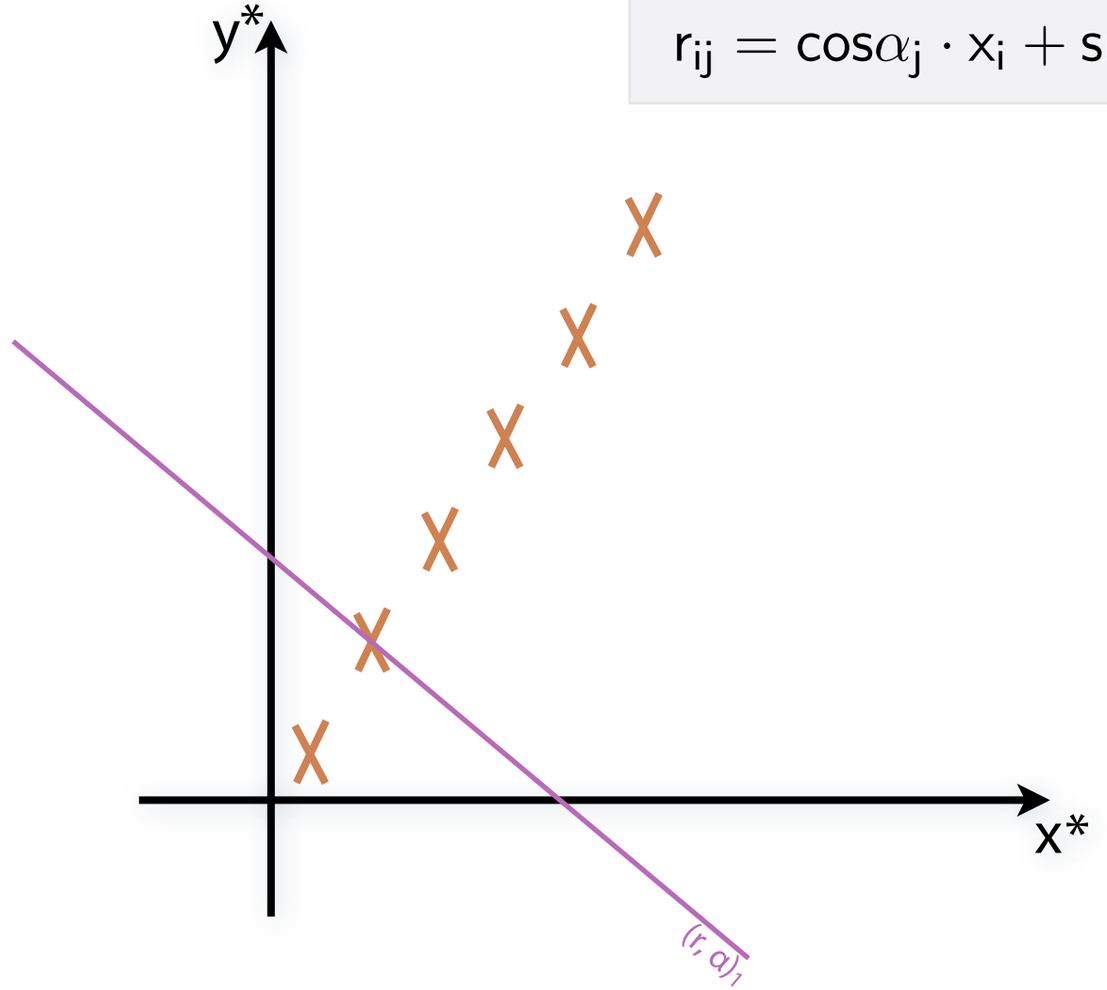


Hough Transform — Principle



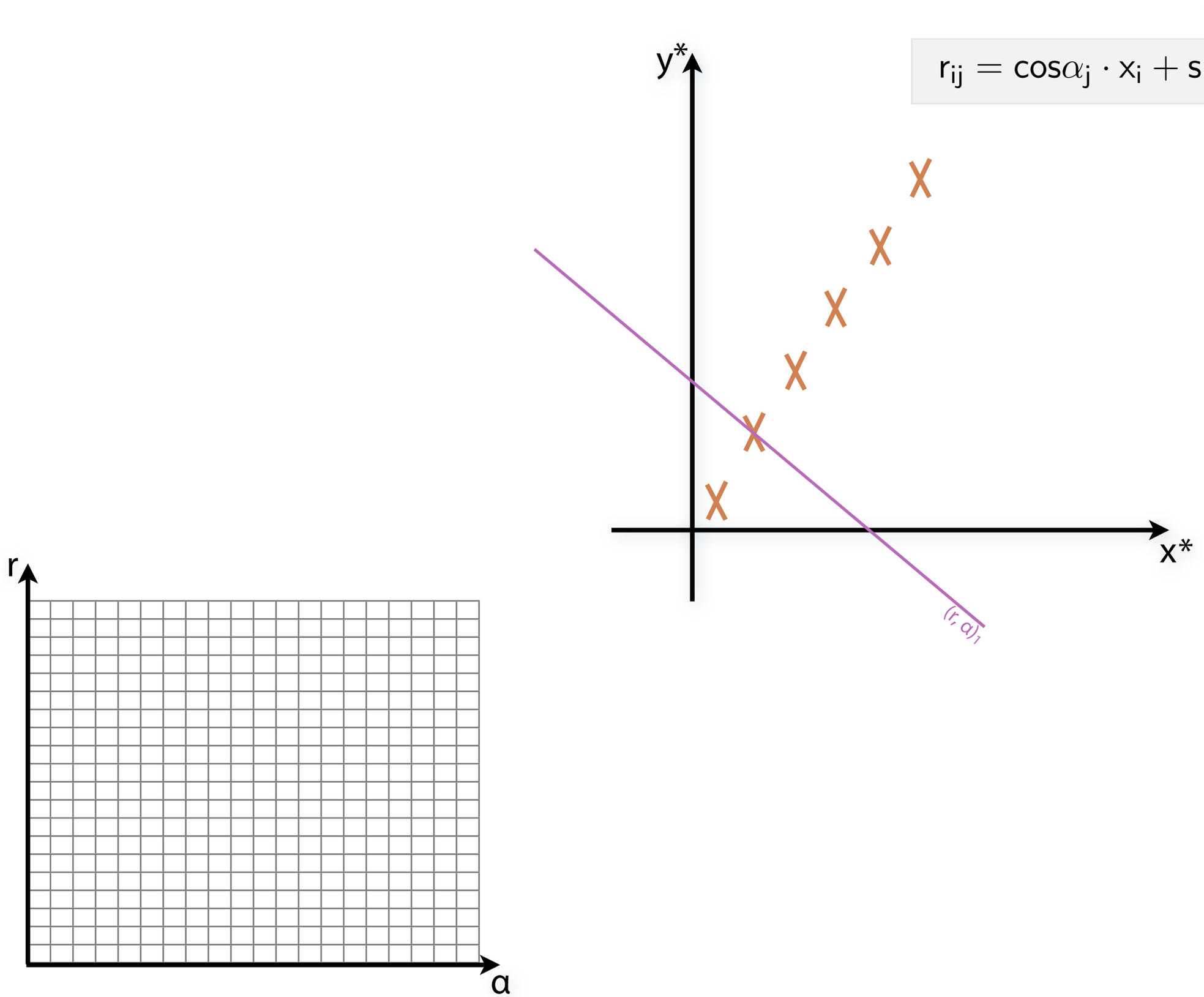
Hough Transform — Principle

$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$



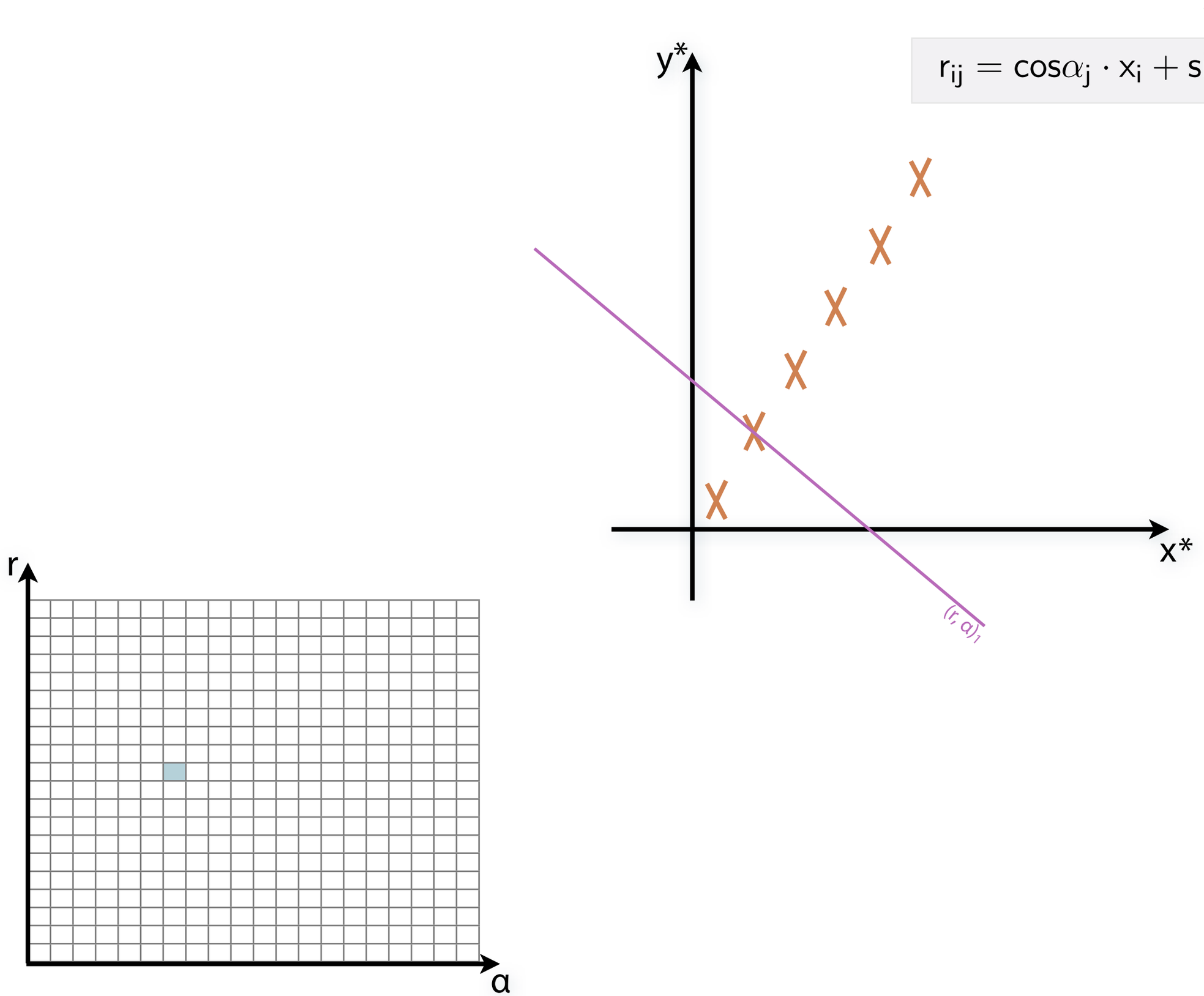
Hough Transform — Principle

$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$



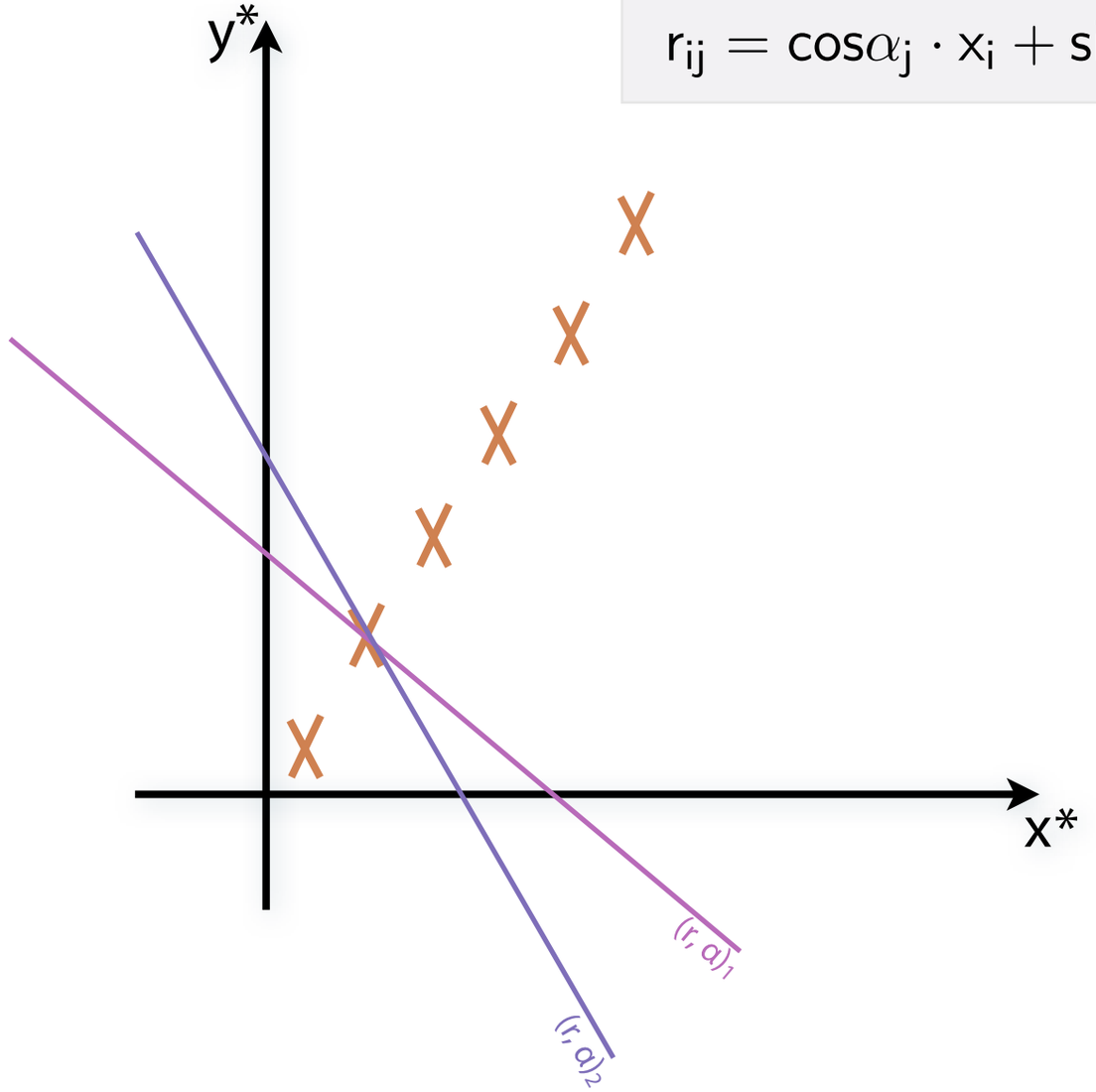
Hough Transform — Principle

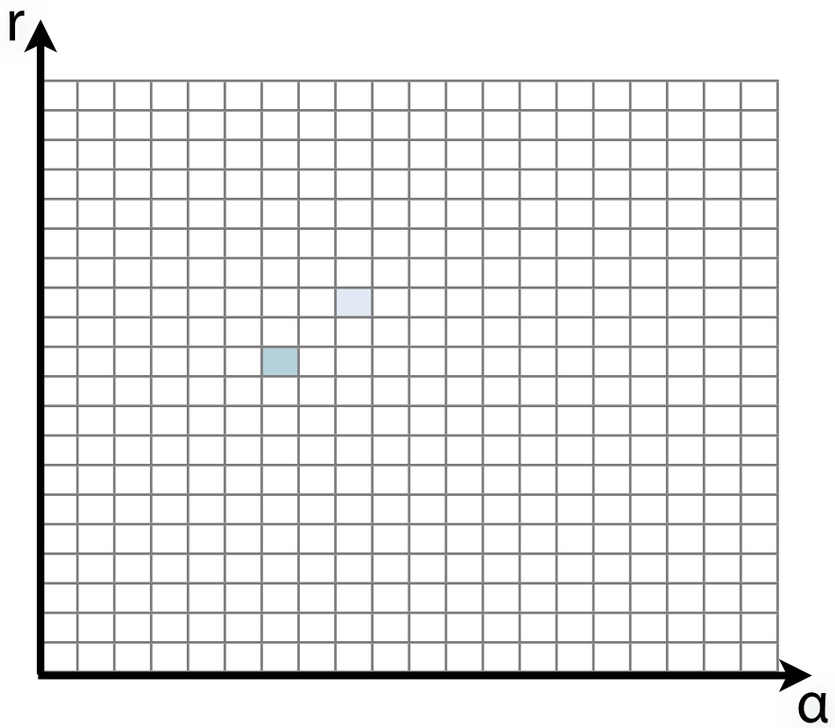
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$



Hough Transform — Principle

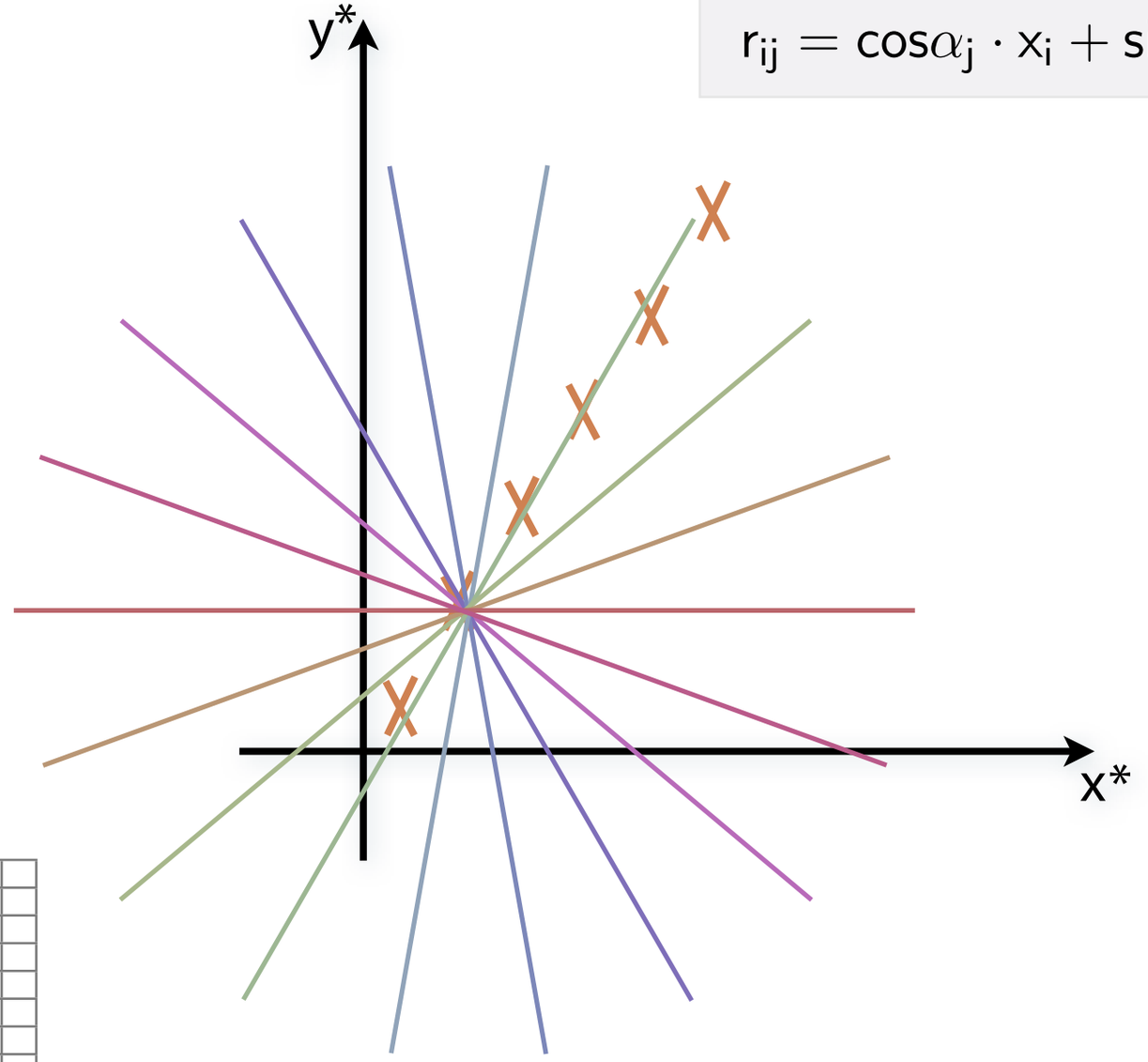
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

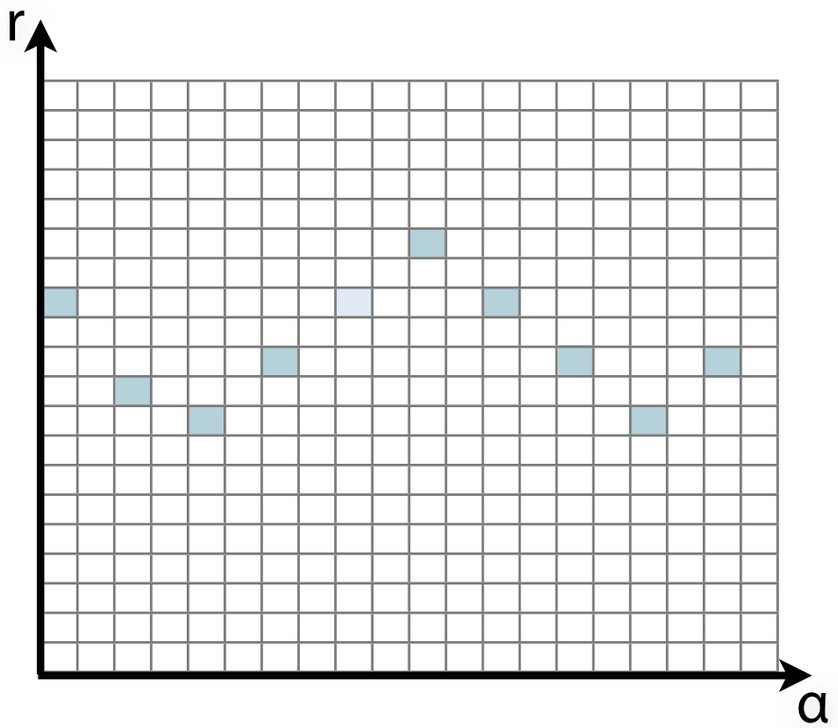




Hough Transform — Principle

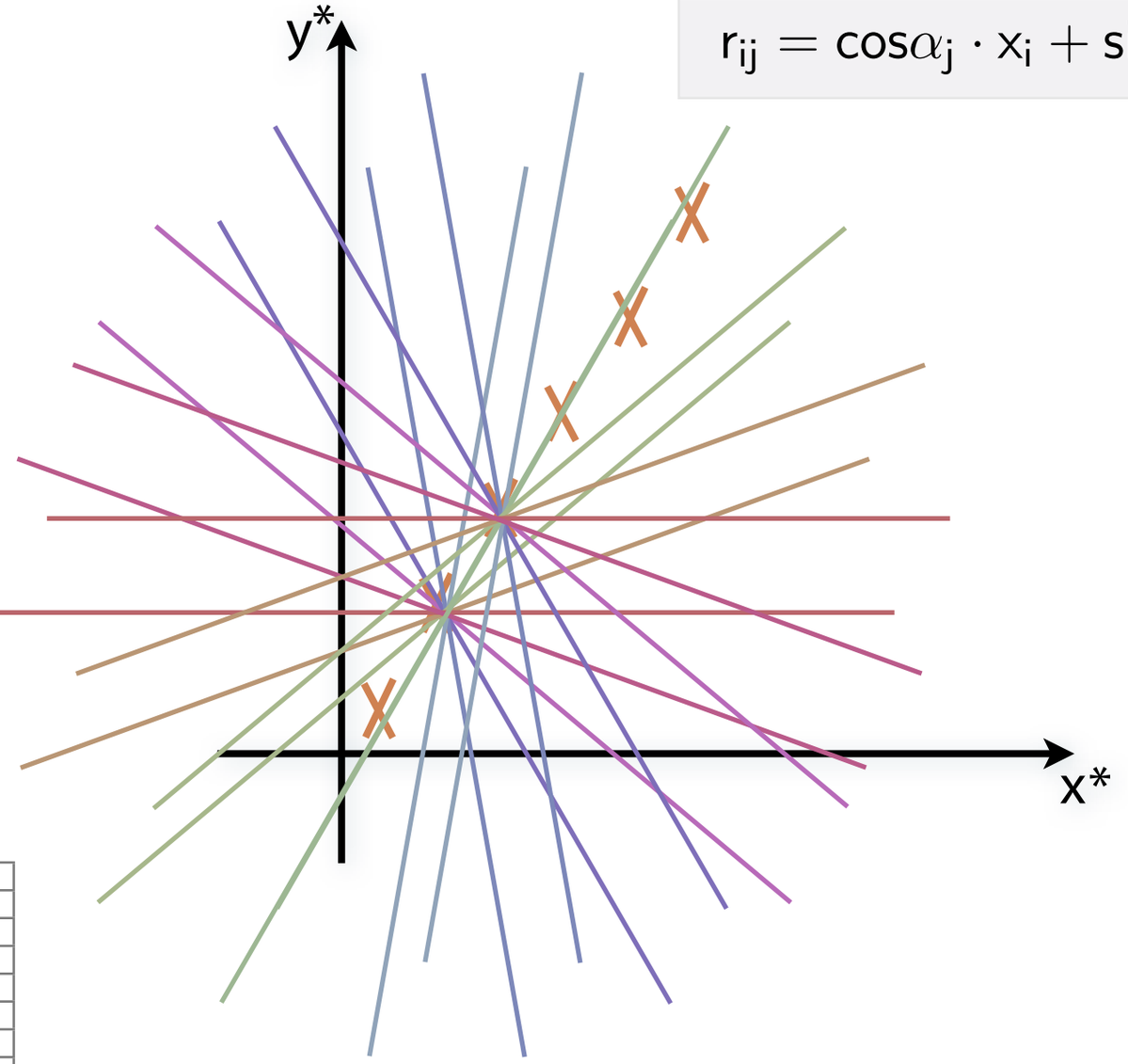
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

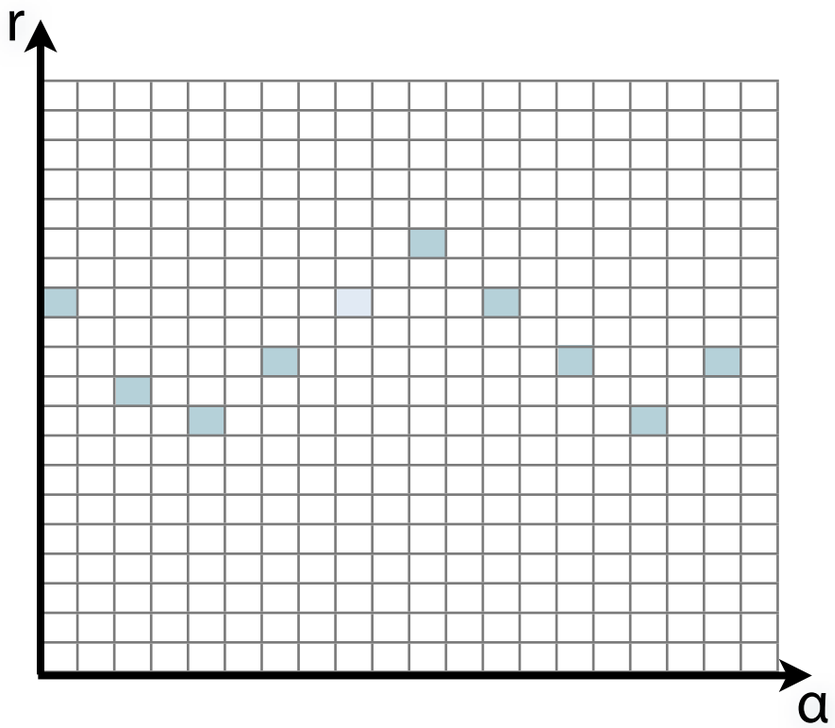




Hough Transform — Principle

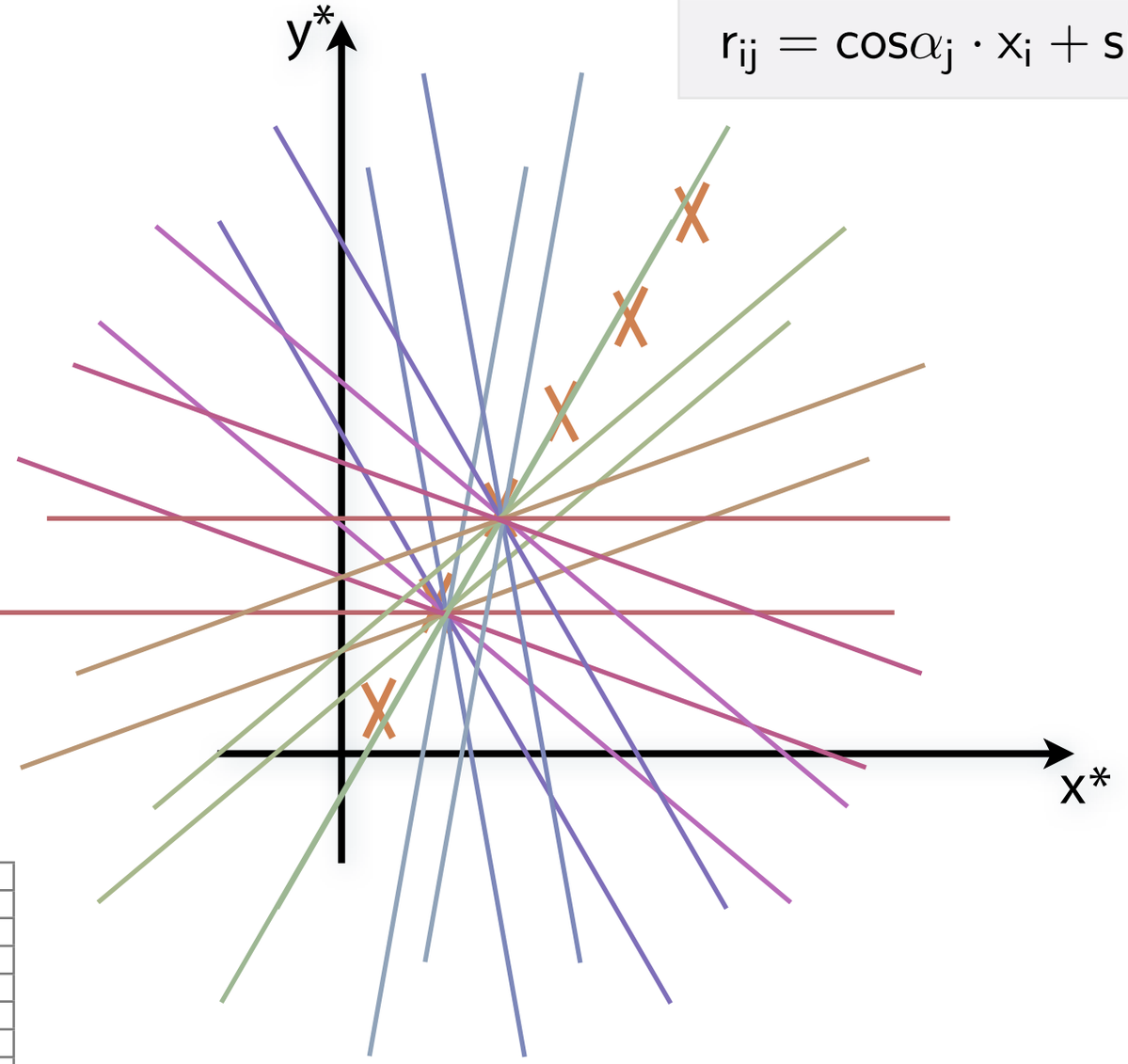
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

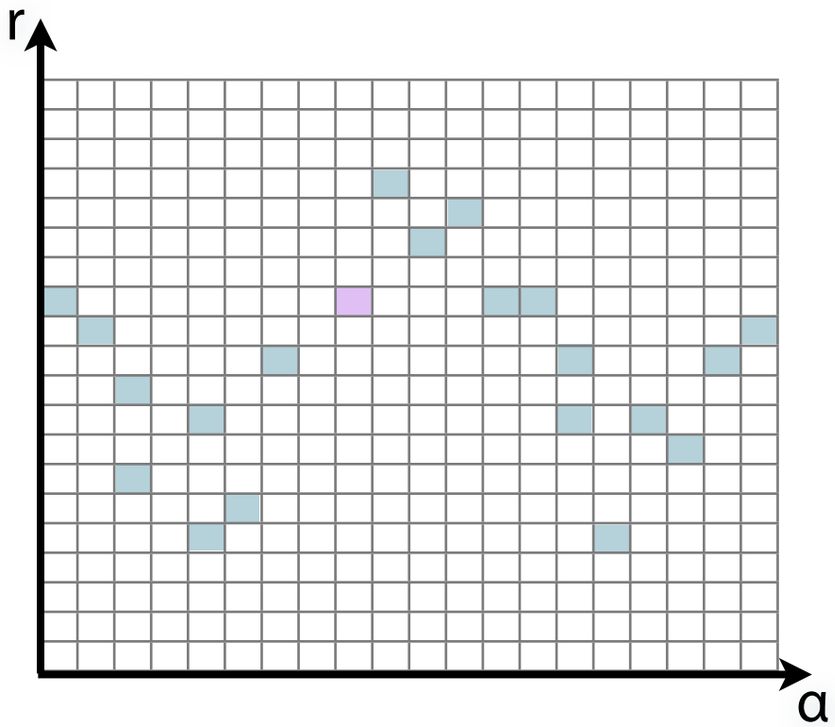




Hough Transform — Principle

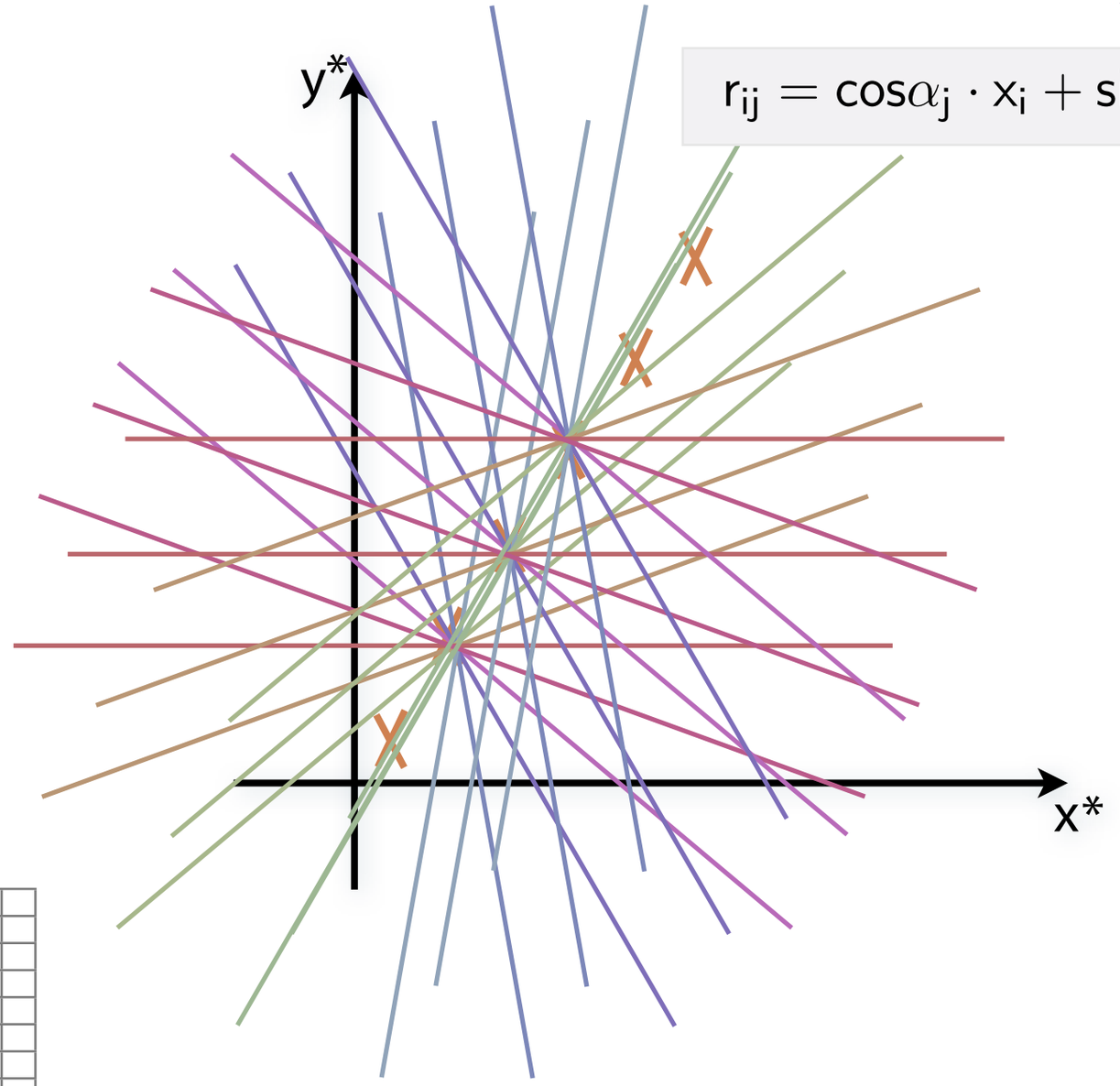
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

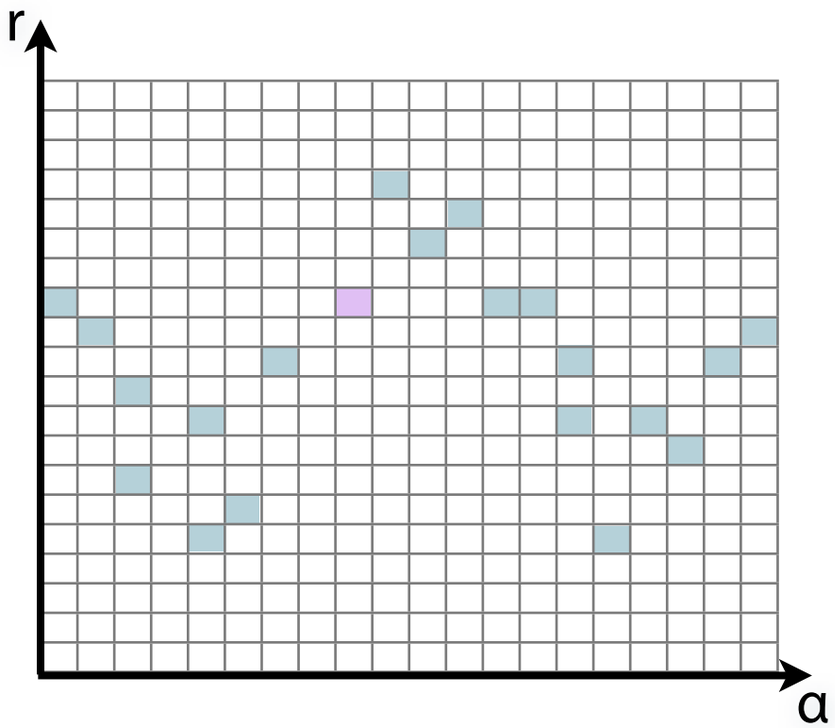




Hough Transform — Principle

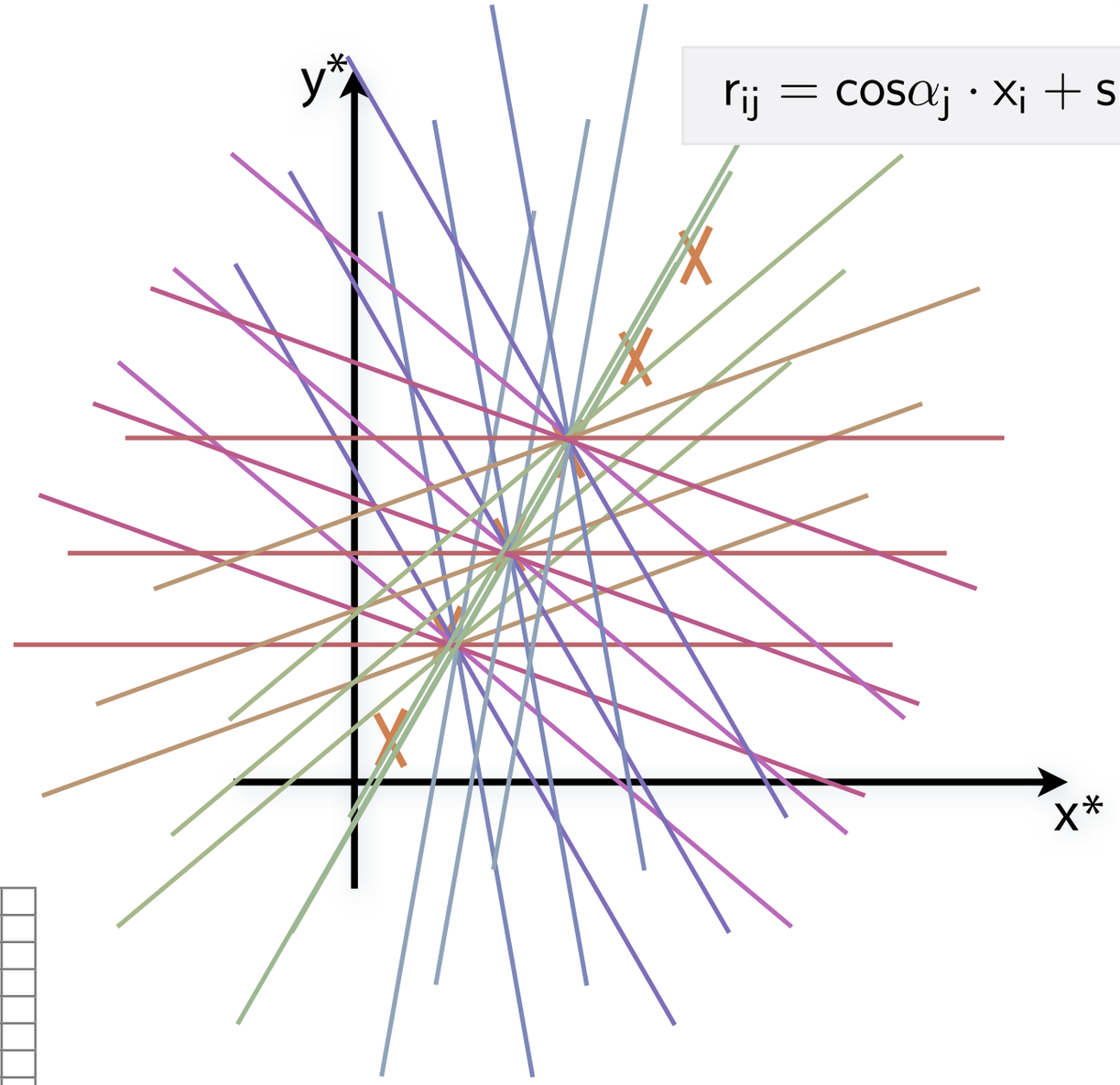
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

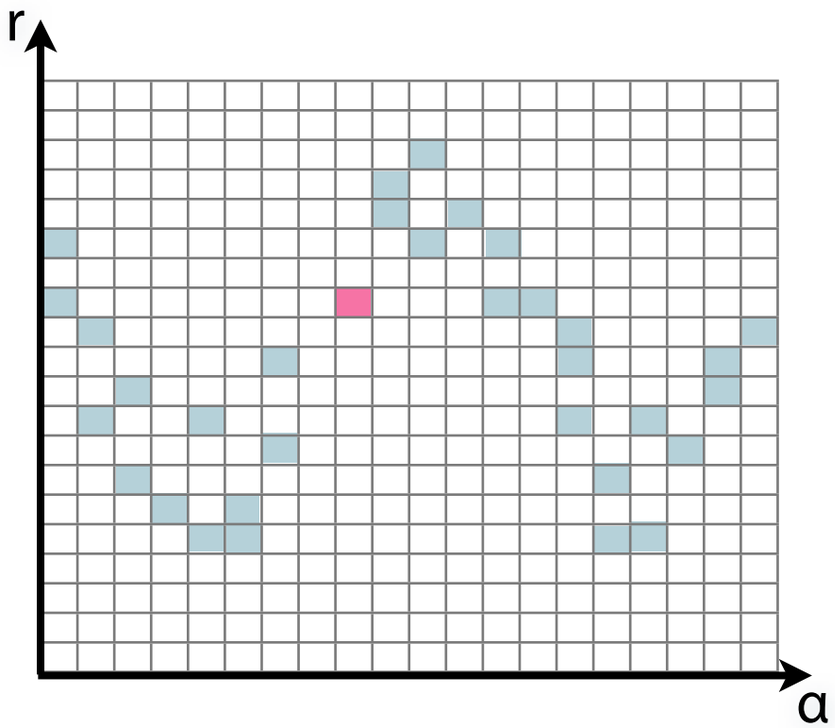




Hough Transform — Principle

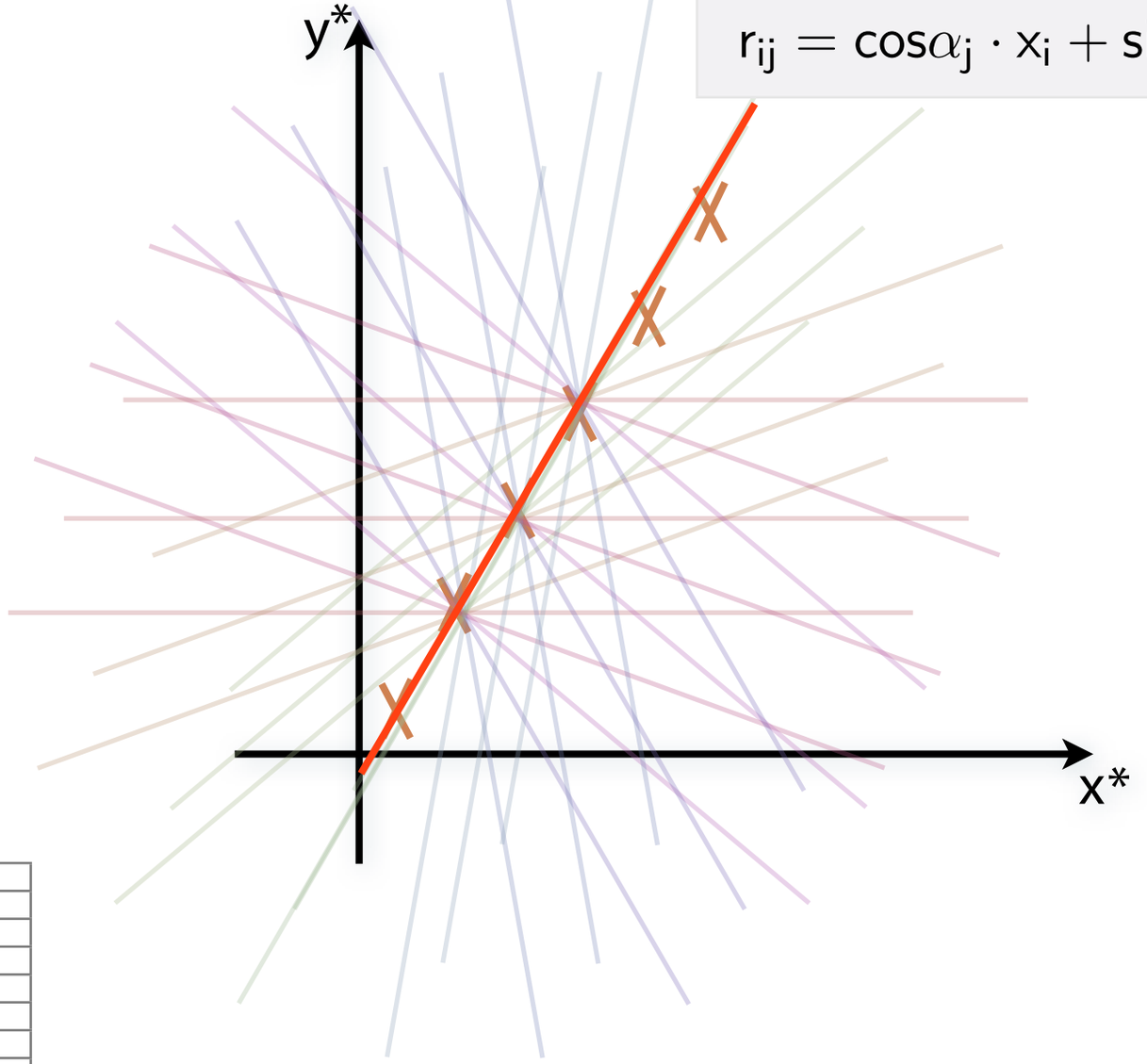
$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

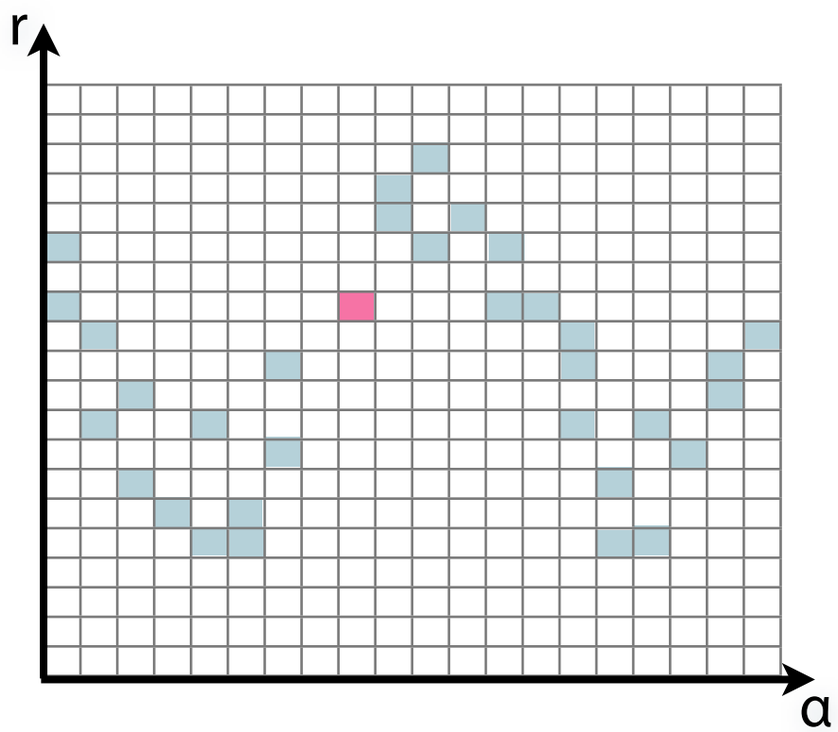




Hough Transform — Principle

$$r_{ij} = \cos\alpha_j \cdot x_i + \sin\alpha_j \cdot y_i + \rho_i$$

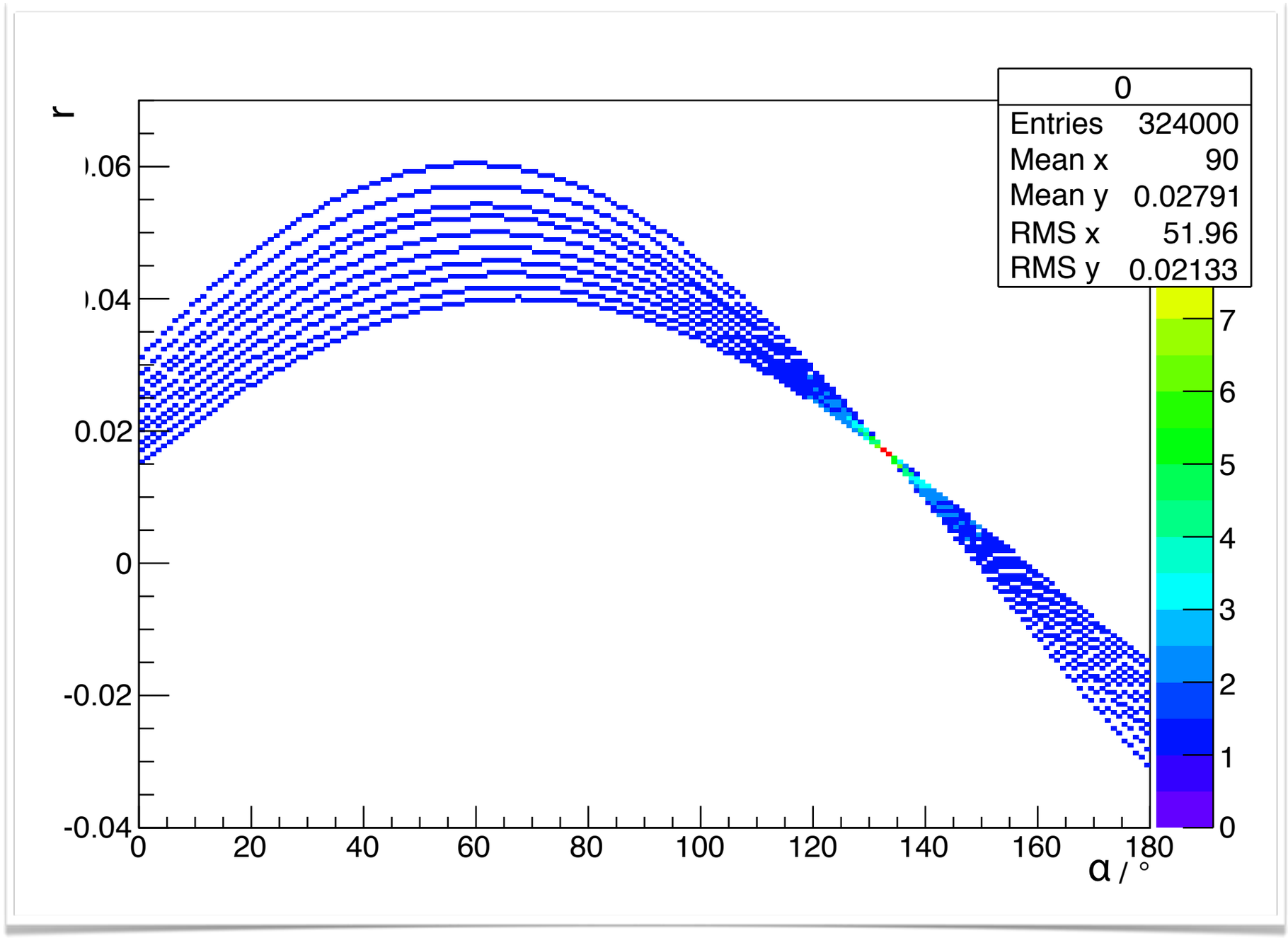




→ **Bin with highest multiplicity gives track parameters**

Hough Transform — Example

- Implementation in Thrust

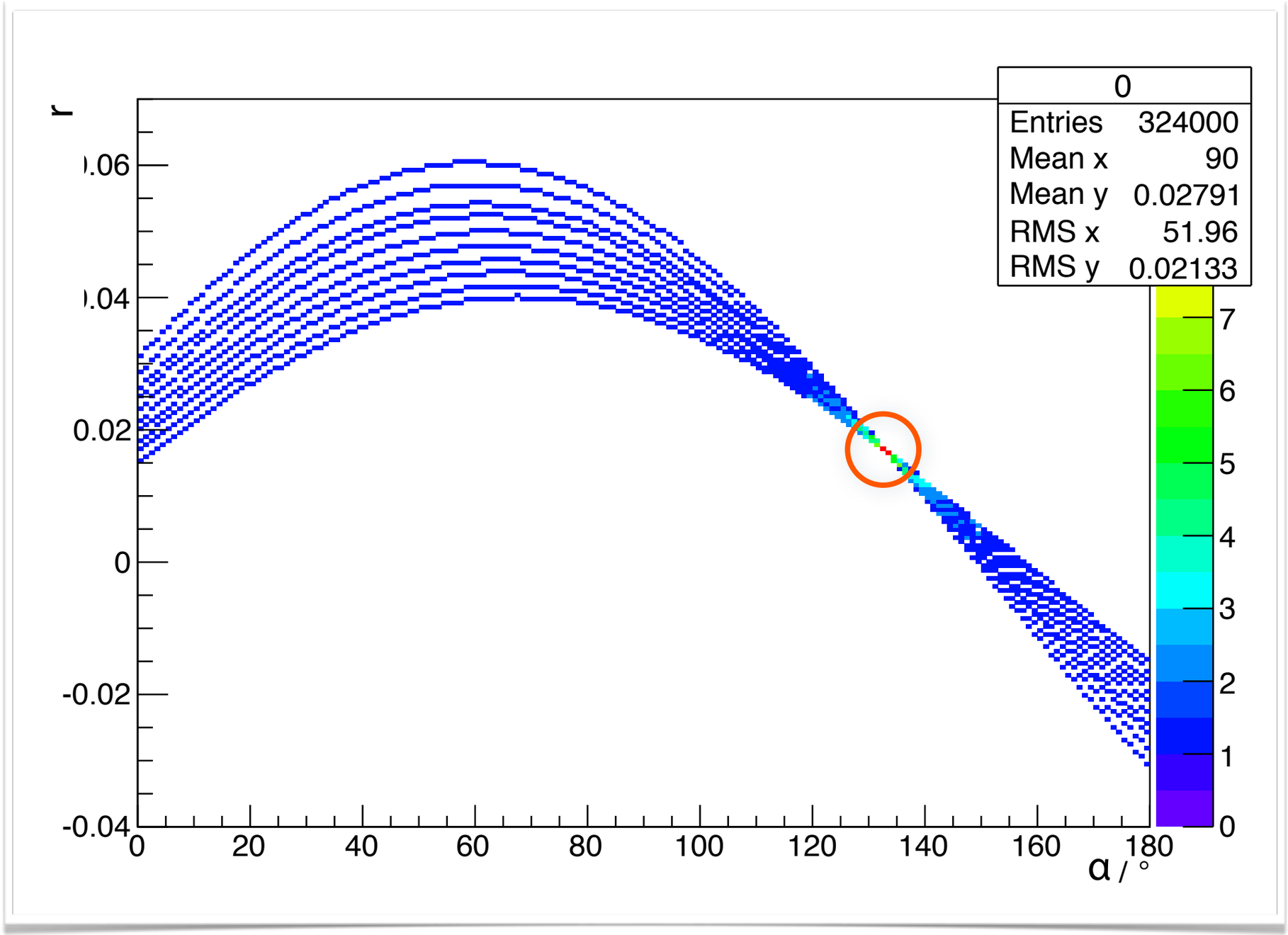


PANDA STT

180 x 180 Grid

Hough Transform — Example

- Implementation in Thrust

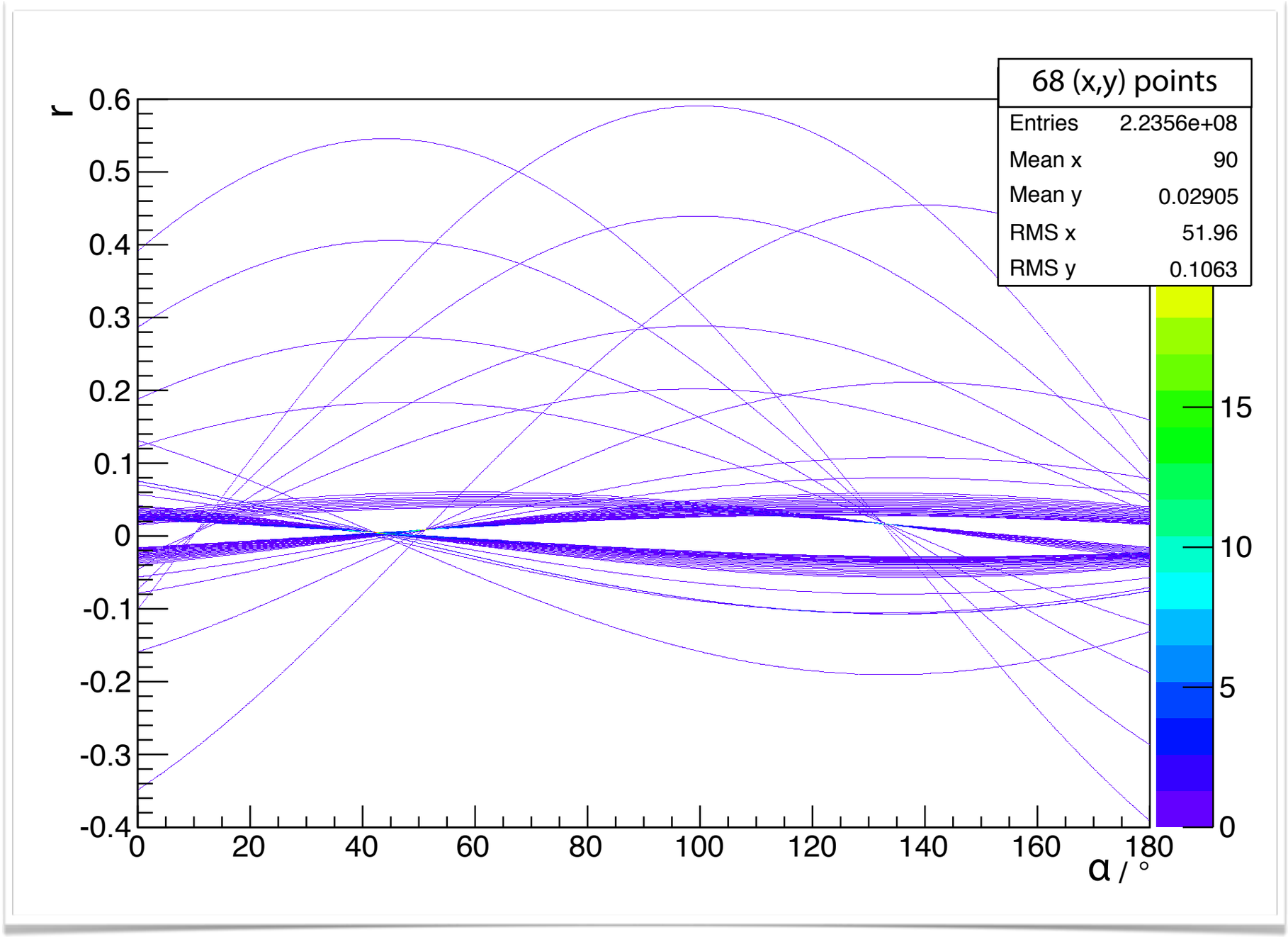


PANDA STT

180 x 180 Grid

Hough Transform — Example

- Implementation in Thrust

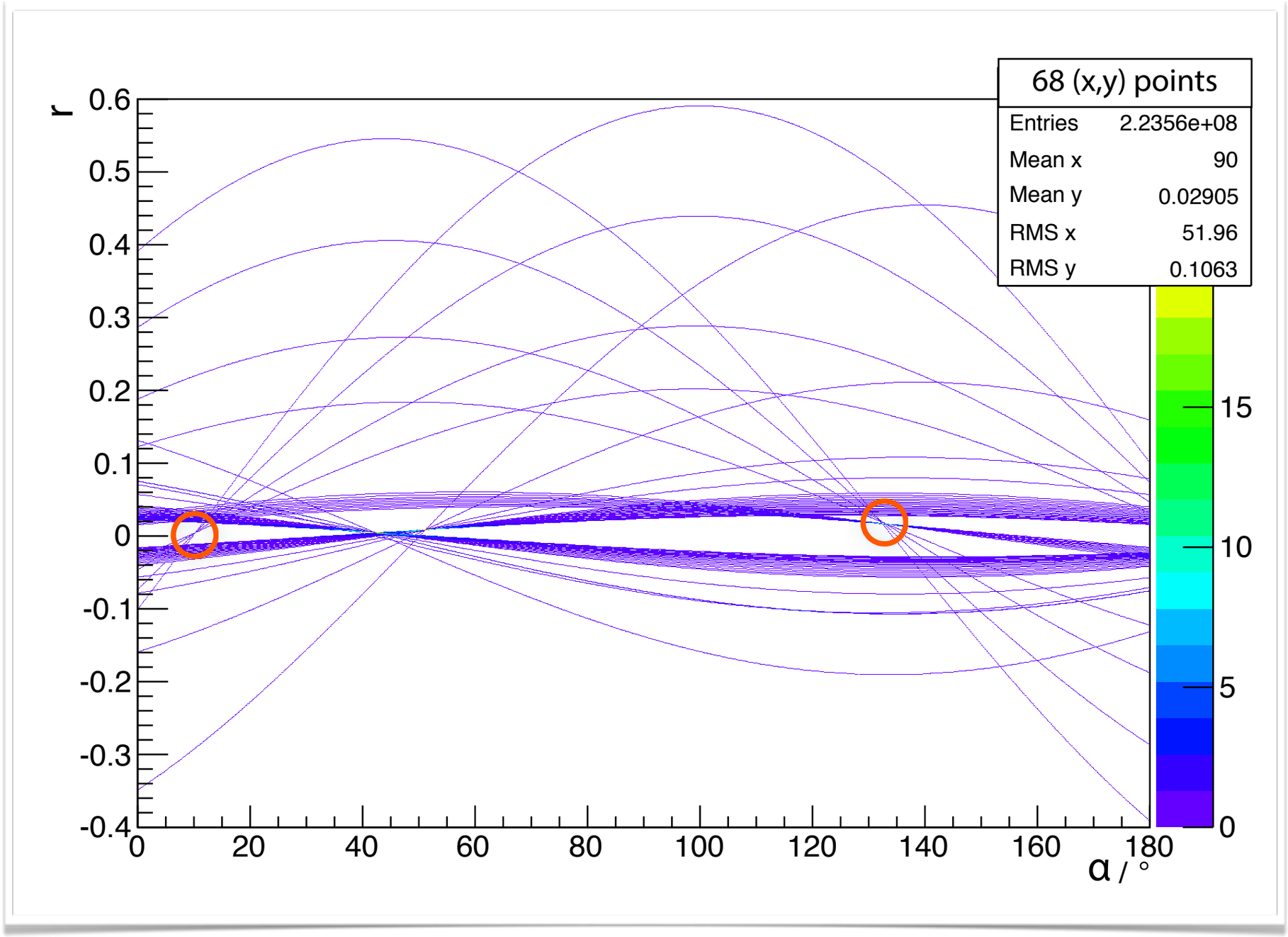


PANDA STT+MVD

1800 x 1800 Grid

Hough Transform — Example

- Implementation in Thrust

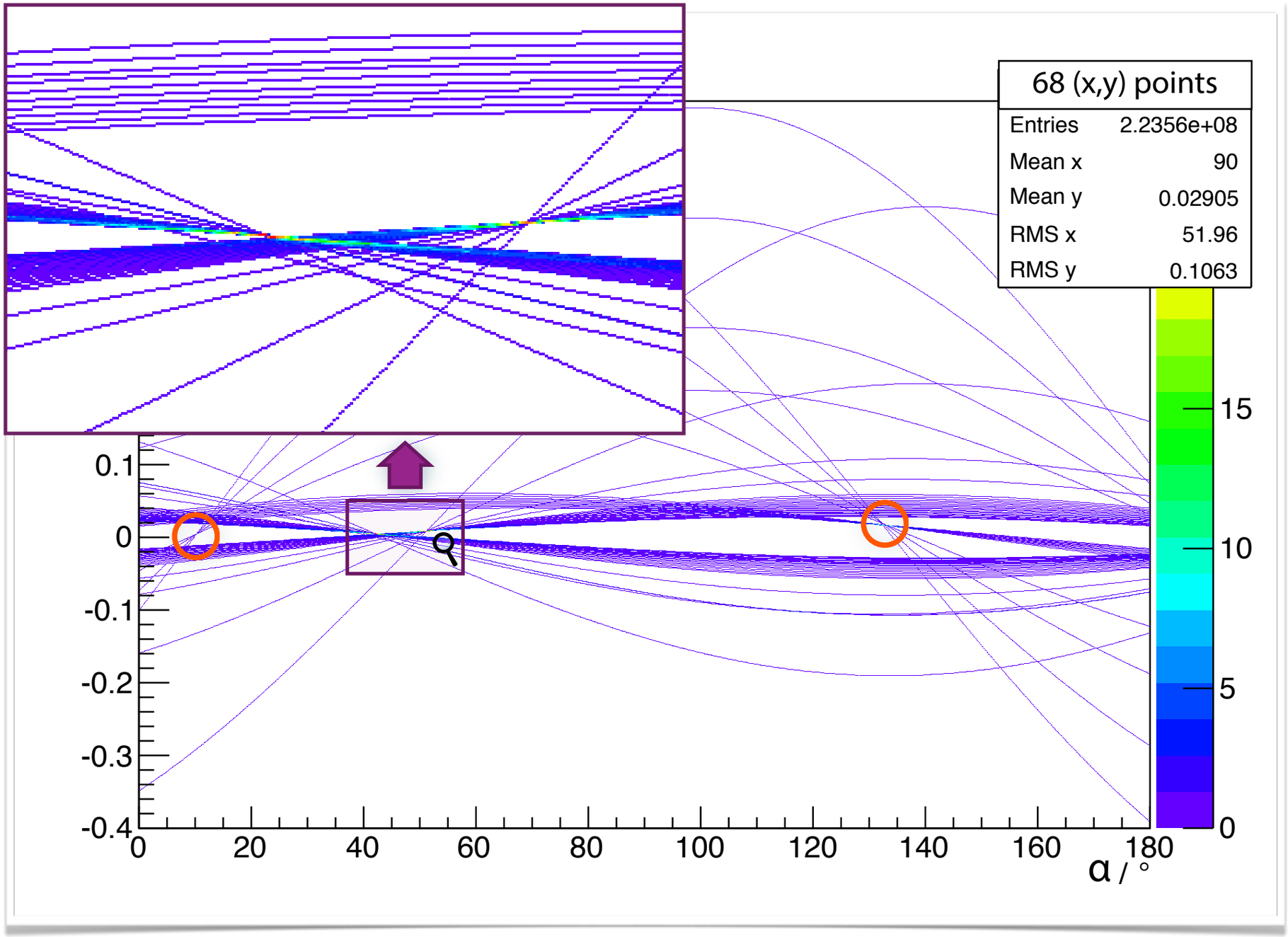


PANDA STT+MVD

1800 x 1800 Grid

Hough Transform — Example

- Implementation in Thrust

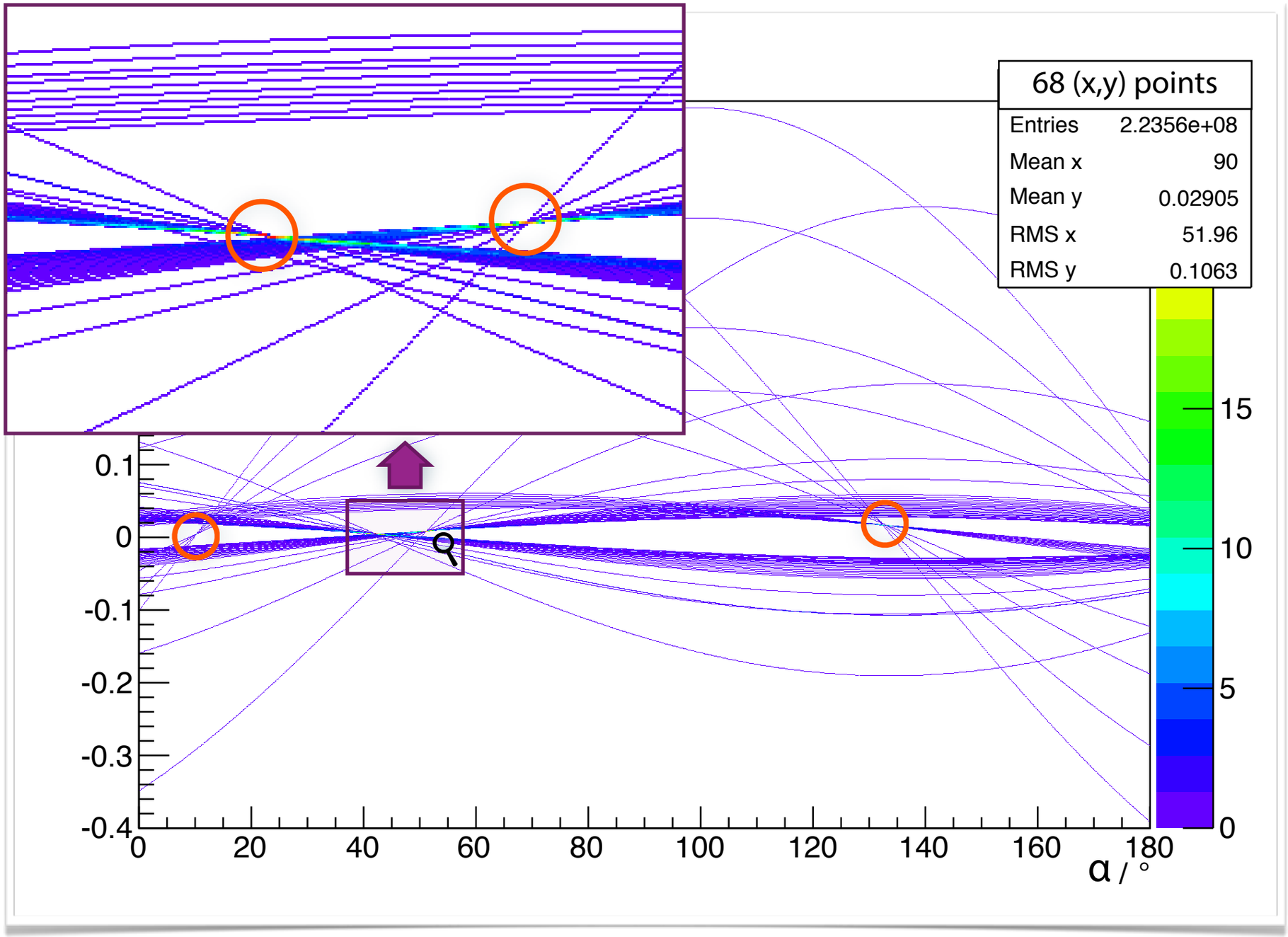


PANDA STT+MVD

1800 x 1800 Grid

Hough Transform — Example

- Implementation in Thrust



PANDA STT+MVD

1800 x 1800 Grid

Hough Transform — Numbers

- **Thrust**: < 3 ms/event
- **Plain CUDA**: 0.5 ms/event

Andrew Adinetz,
NVIDIA Application Lab

Hough Transform — Notes

- **Thrust**: < 3 ms/event
  - Running code
  - Independent of α granularity *(occupancy; see below)*
  - Parallel
    - Parallel for all hits of one event
    - Not parallel for all events
  - Reduced problem(s) to set of **standard routines** *(~STL)*
    - Fast (uses clever pre-made algorithms)
    - Inflexible (has it's limits, hard to customize)
  - No peakfinding included
    - Even possible?
    - Adds to time!

Hough Transform — Notes

- **Thrust**: < 3 ms/event
- **Plain CUDA**: 0.5 ms/event
  - Running code
  - Parallel
    - Fully parallel
    - Coarser grid → faster computing times
  - Built completely from scratch
    - Customizable / fitting to every problem
    - A bit more complicated at parts
  - Simple peakfinder implemented *(threshold)*

Hough Transform — Summary

- Running code from Andrew Adinetz & me
- Big issue: Multipeak finder

Hough Transform — Summary

- Running code from Andrew Adinetz & me
- Big issue: Multipeak finder

	To Dos / Plans	Advantages of HT algorithm	Draw Backs	~~Problems~~ Challenges
Thrust	• Parallelism beyond events • Isochrones (fully) • Integrate to PandaRoot • Time-based	• Easy algorithm • With grid granularity parallelism increases • Flexibility of HT'ed equation	• Stuck in Thrust infrastructure	• Multipeak finder/ Aliasing (Algorithm out of image processing – usually used to detect continuous lines)
Plain CUDA	• Isochrones (fully) • Integrate to PandaRoot • Time-based			

Hough Transform — Summary

- Running code from Andrew Adinetz & me
- Big issue: Multipeak finder

	To Dos / Plans	Advantages of HT algorithm	Draw Backs	~~Problems~~ Challenges
Thrust	• Parallelism beyond events • Isochrones (fully) • Integrate to PandaRoot • Time-based	• Easy algorithm • With grid granularity parallelism increases • Flexibility of HT'ed equation	• Stuck in Thrust infrastructure	• Multipeak finder/ Aliasing (Algorithm out of image processing – usually used to detect continuous lines)
Plain CUDA	• Isochrones (fully) • Integrate to PandaRoot • Time-based			

- Code at:
  - https://subversion.gsi.de/trac/fairroot/browser/pandaroot/development/aherten/GpuHoughTransform
  - https://github.com/AndiH/CUDA/

JÜLICH
FORSCHUNGSZENTRUM

panda

ALGORITHMS

Riemann Track Finder

Riemann Algorithm

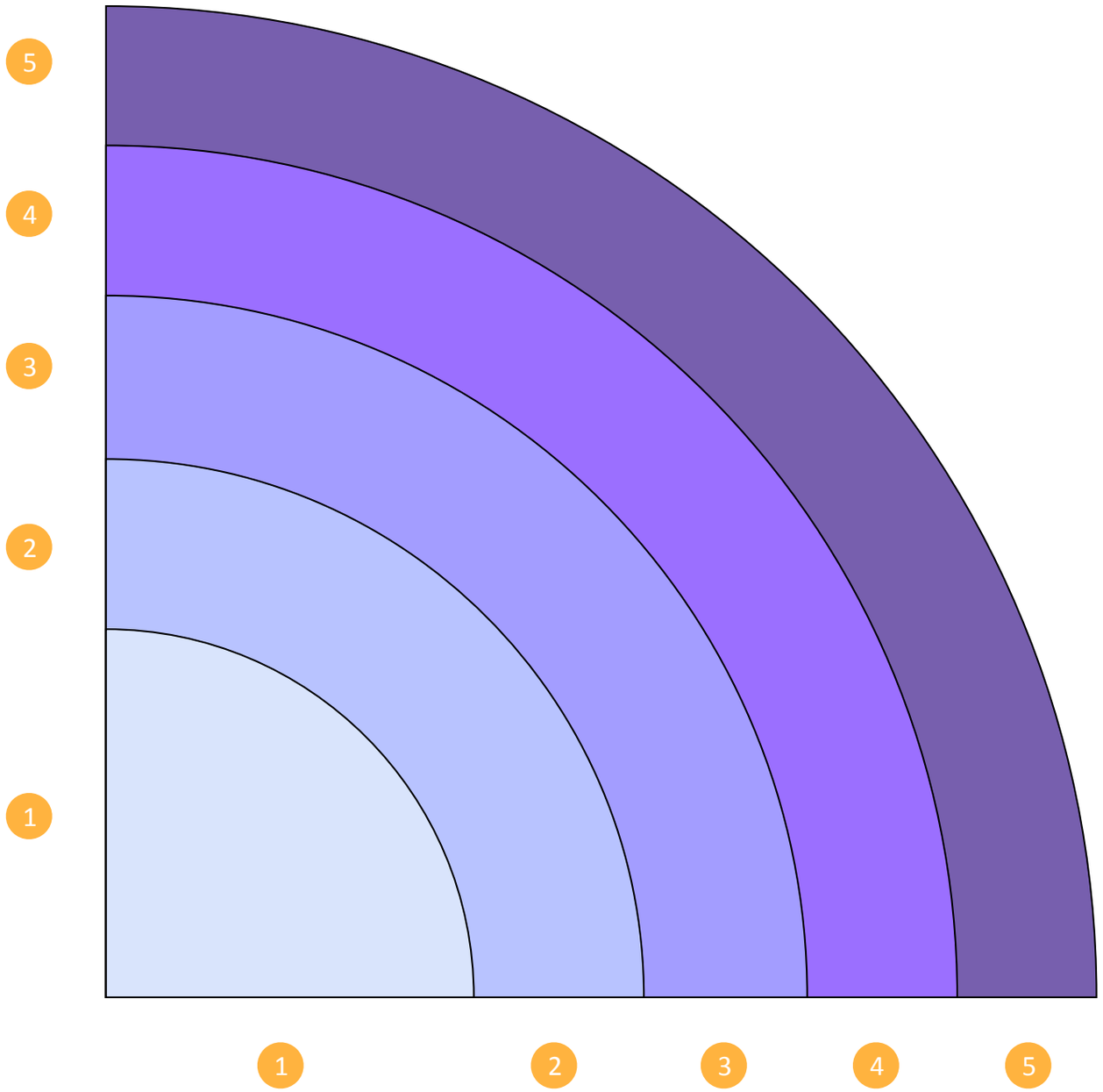
Riemann Algorithm

1. **Create triplet** of MVD hit points
   - All possible three hit combinations need to become triplets

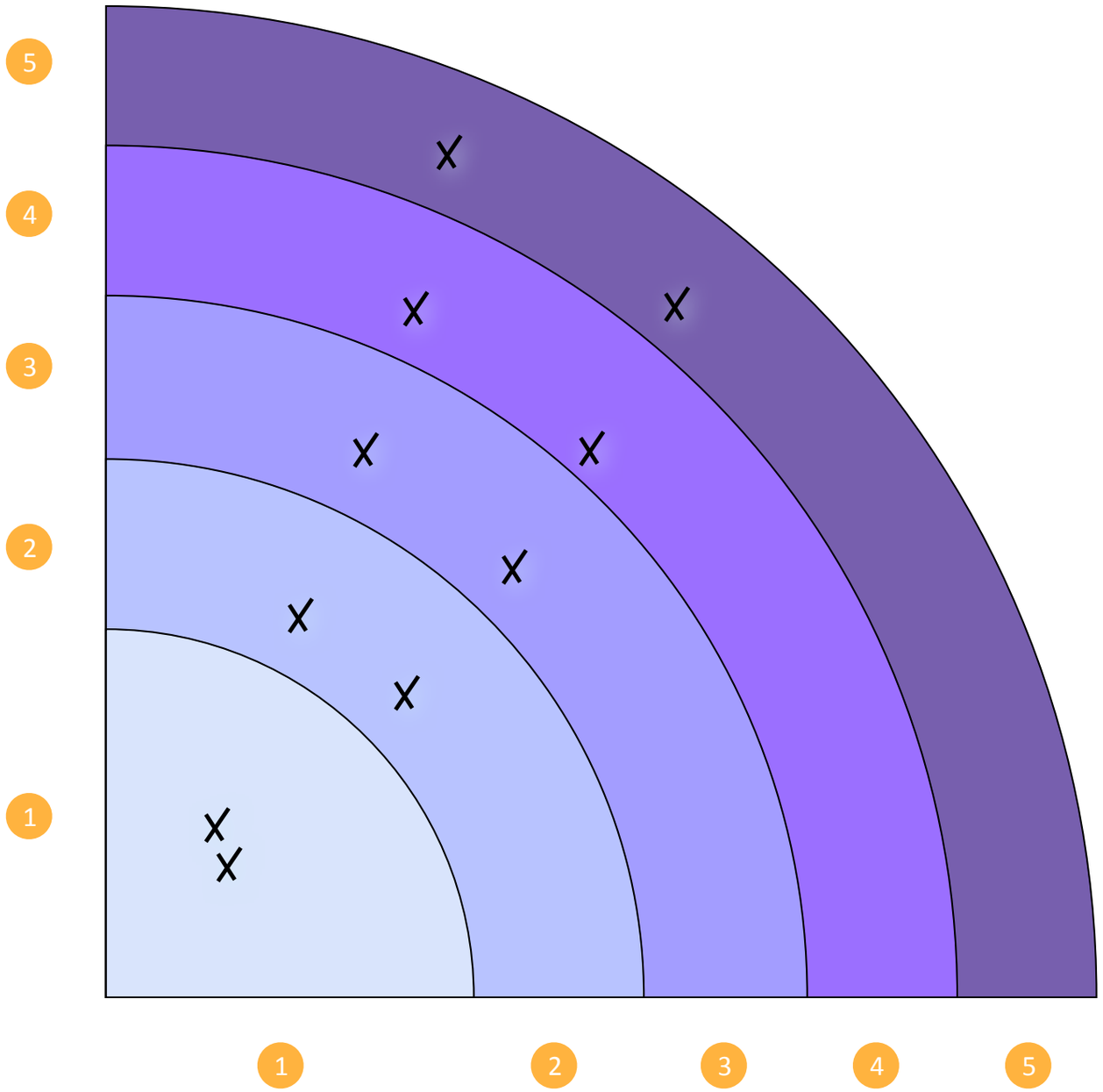
Riemann Algorithm

1. **Create triplet** of MVD hit points
   - All possible three hit combinations need to become triplets
2. **Grow** triplets to **tracks**:
   Continuously test next hit if it fits to triplet track
   - Circle fit in (x,y) plane
     - Project hit points of triplet to Riemann paraboloid
     - Create plane *(∝ track)*
     - Test closeness of new hit: good → add hit; bad → dismiss hit
     - Continue with next hit
   - Helix fit: arc length s vs. z position

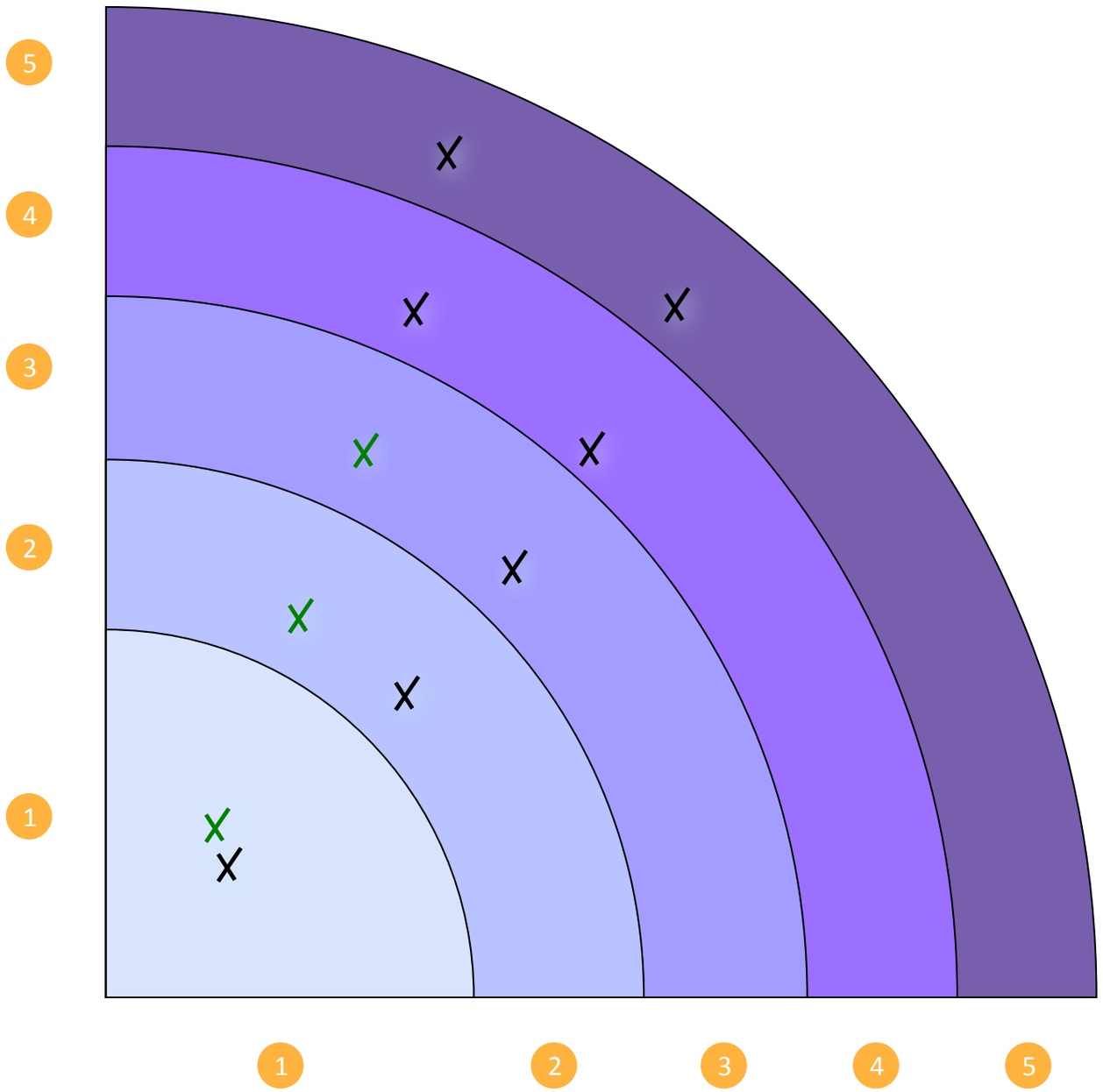
Riemann Algorithm — 1 Triplets



Riemann Algorithm — 1 Triplets

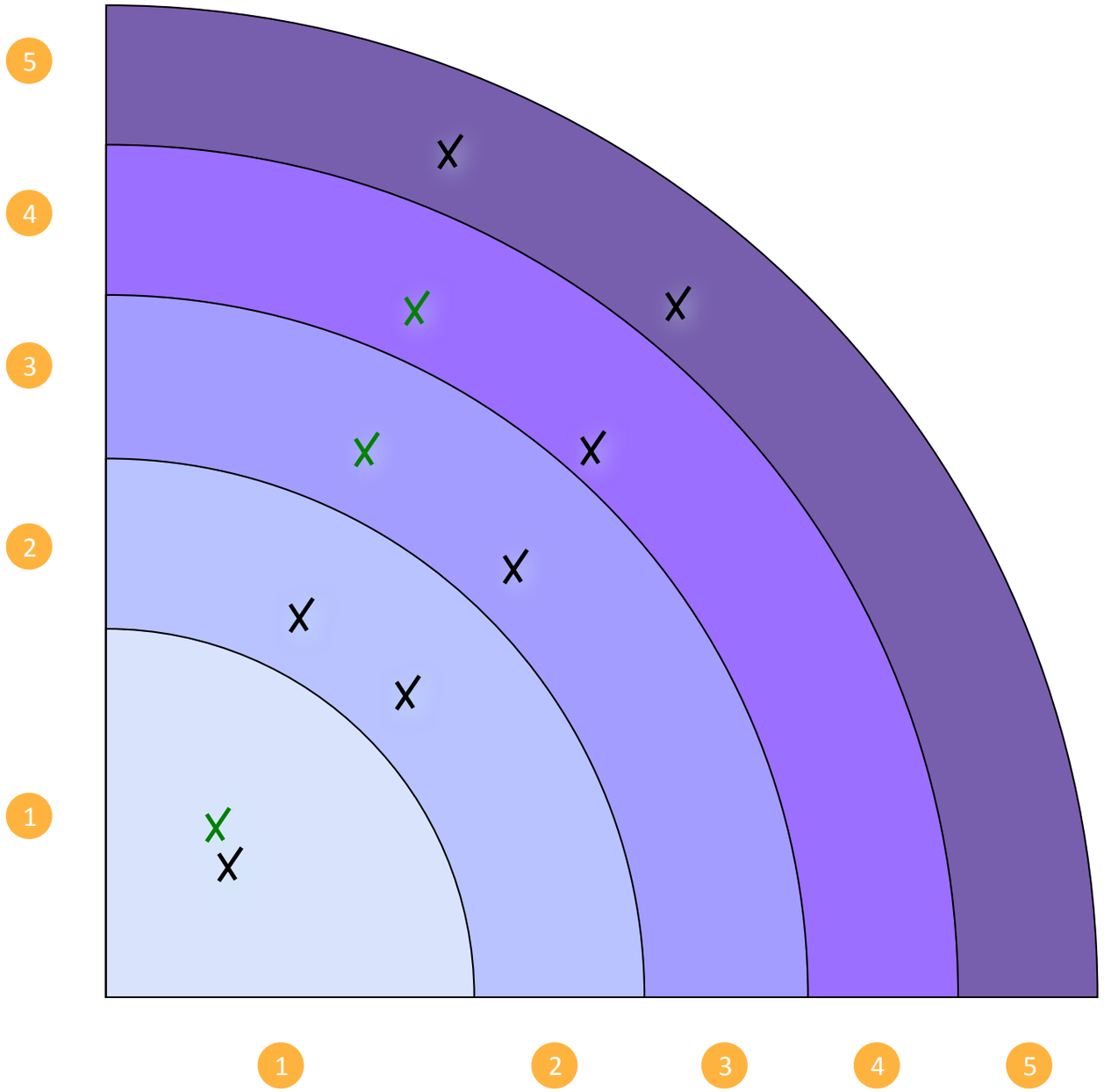


Riemann Algorithm — 1 Triplets



1_1 2_1 3_1

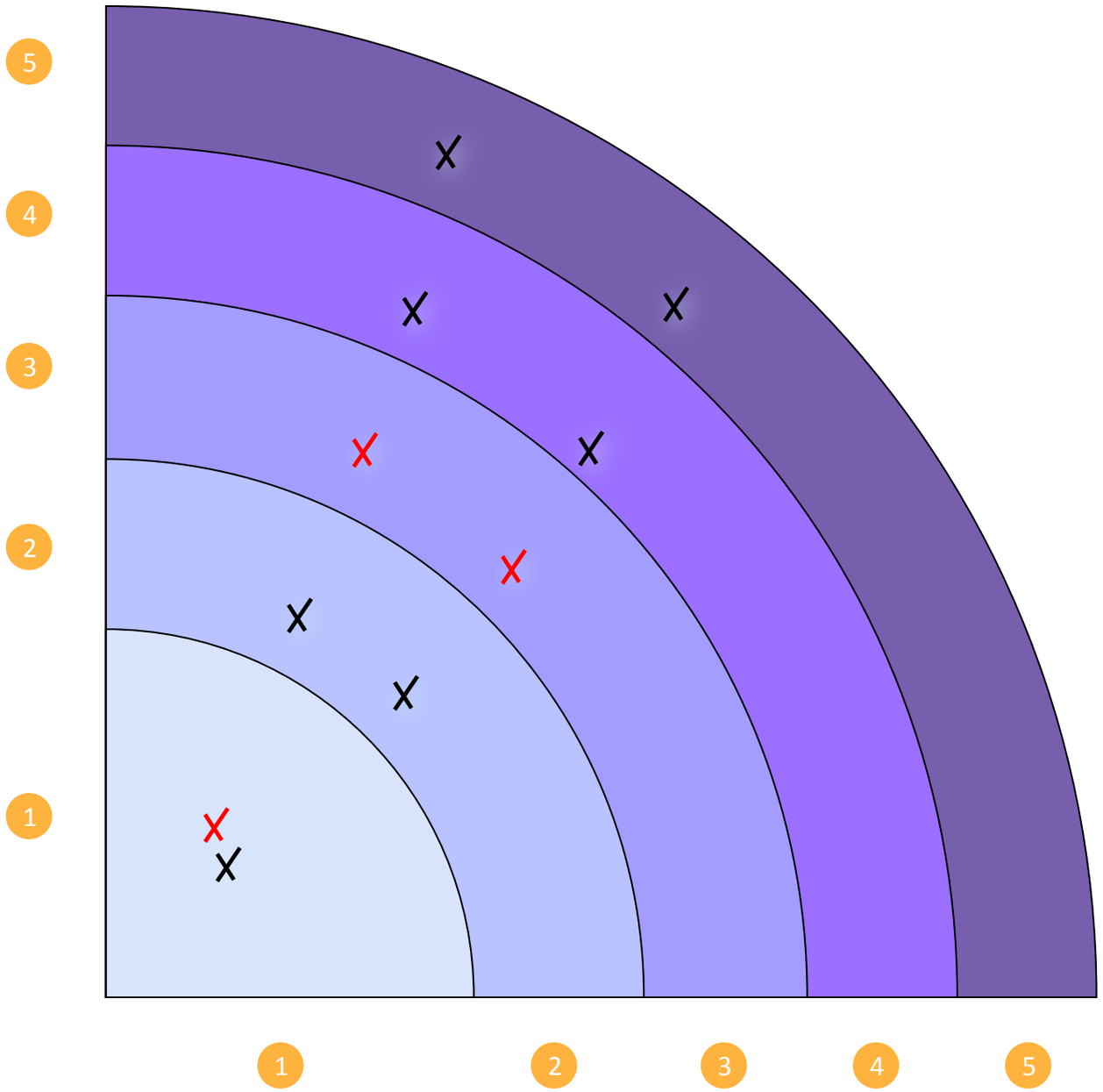
Riemann Algorithm — 1 Triplets



1_1 2_1 3_1

1_1 3_1 4_1

Riemann Algorithm — 1 Triplets

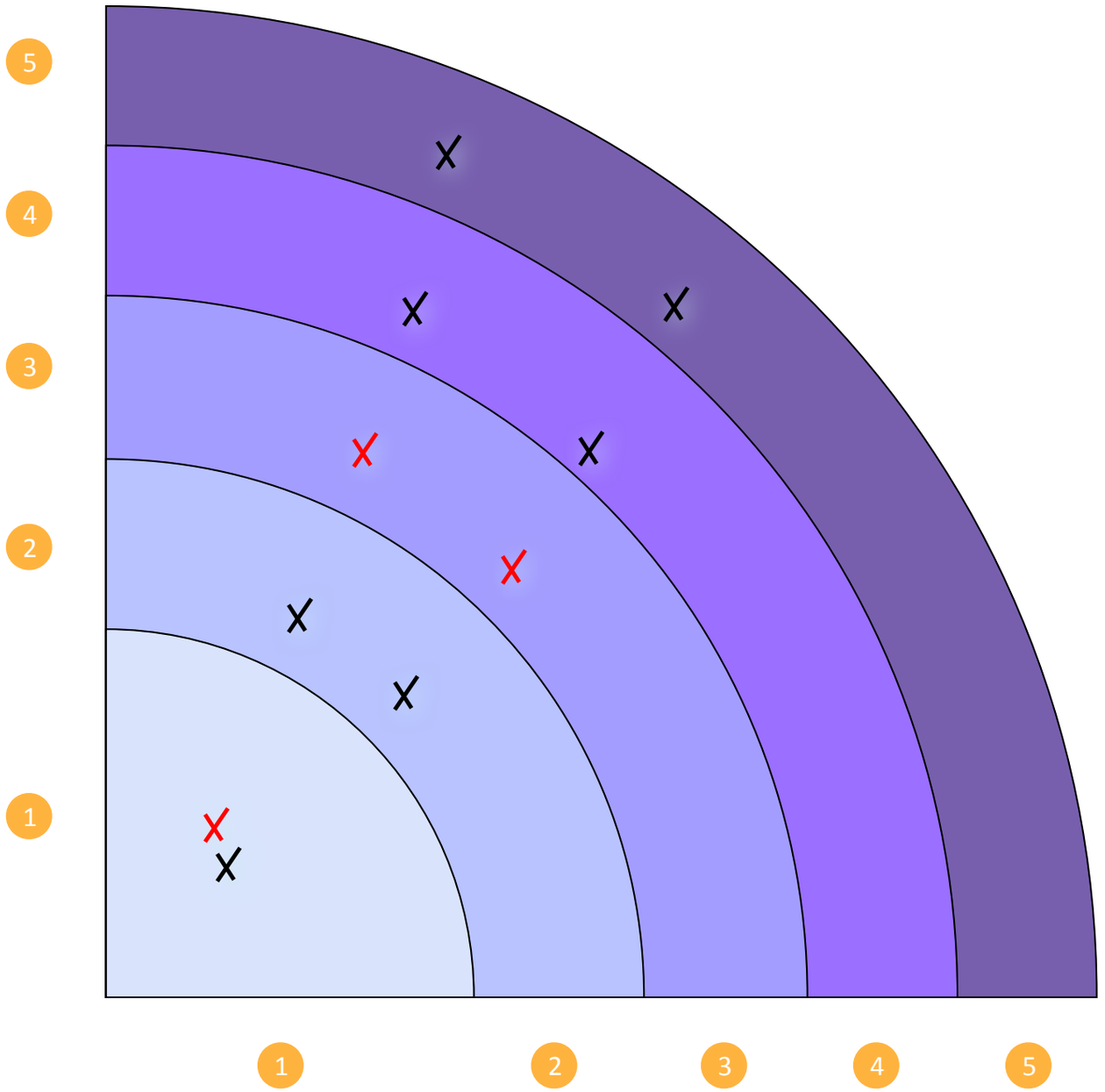


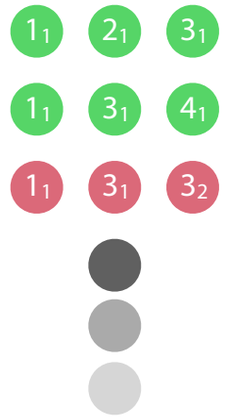
1_1 2_1 3_1

1_1 3_1 4_1

1_1 3_1 3_2

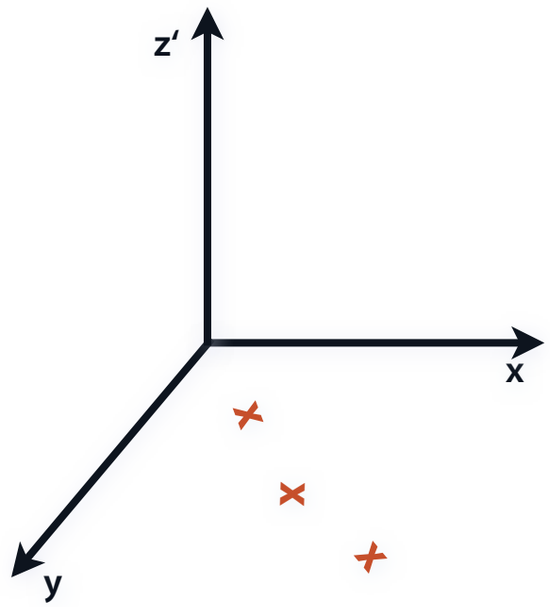
Riemann Algorithm — 1 Triplets





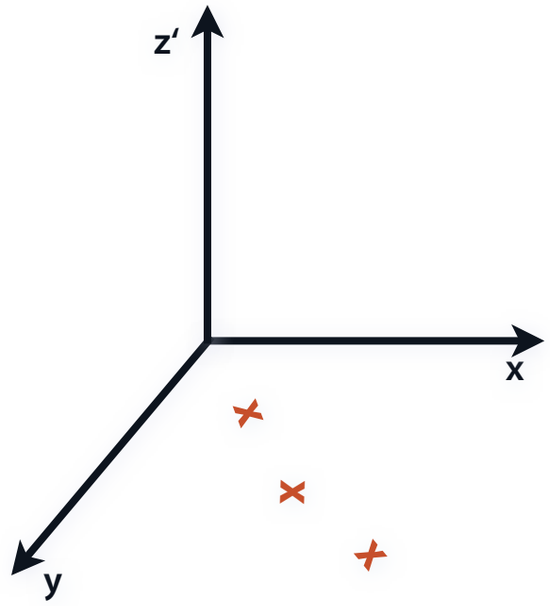
Riemann Algorithm — 2 Expansion

Riemann Algorithm — 2 Expansion

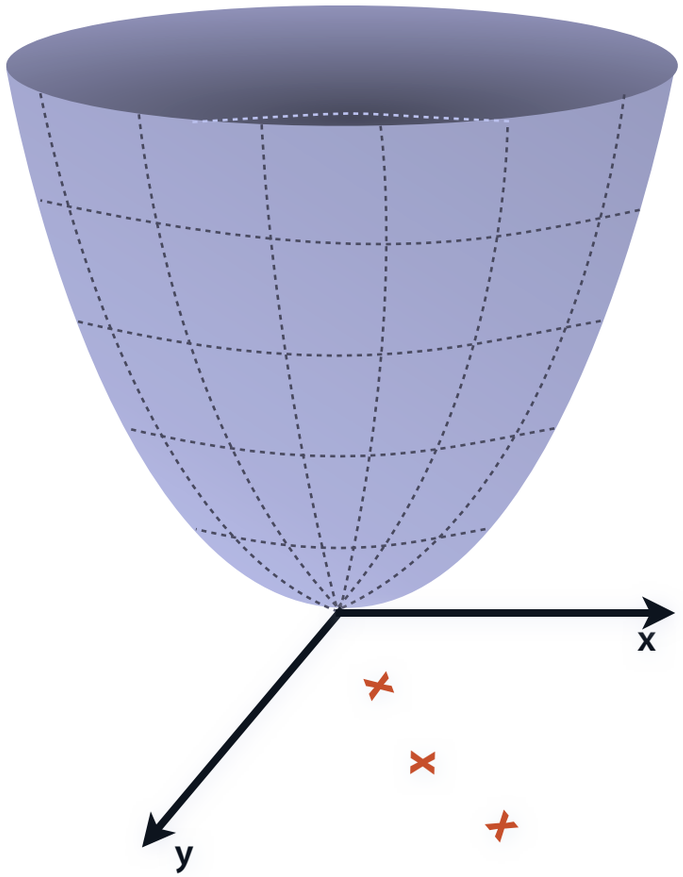


Expand to z‘

Riemann Algorithm — 2 Expansion

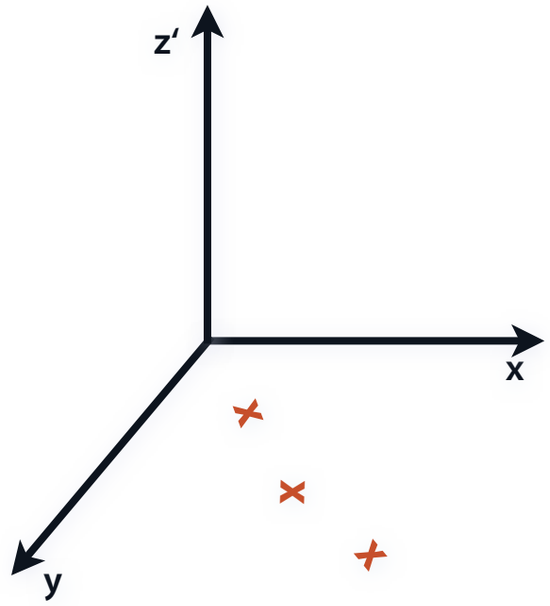


Expand to z‘

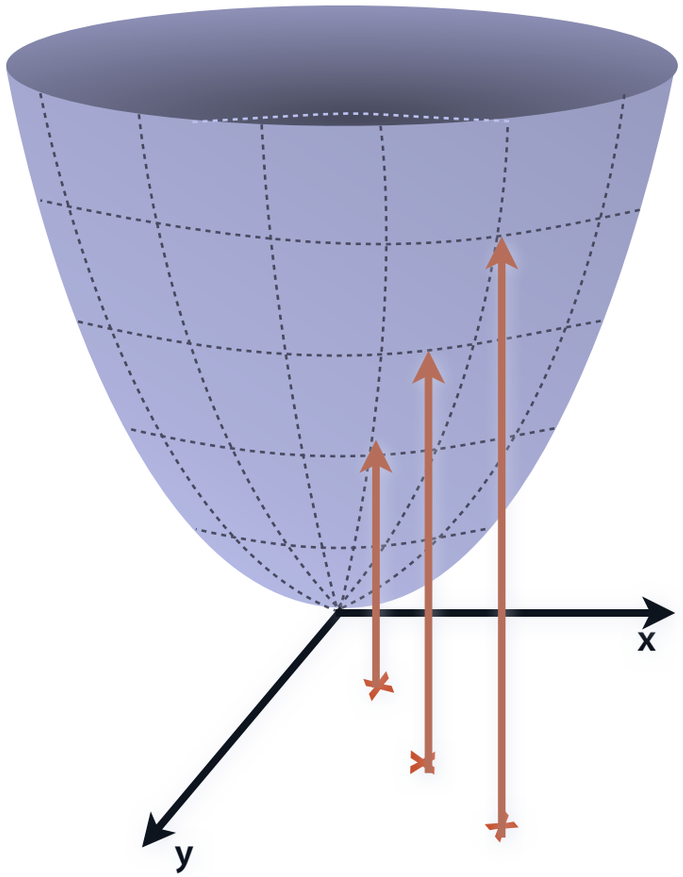


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

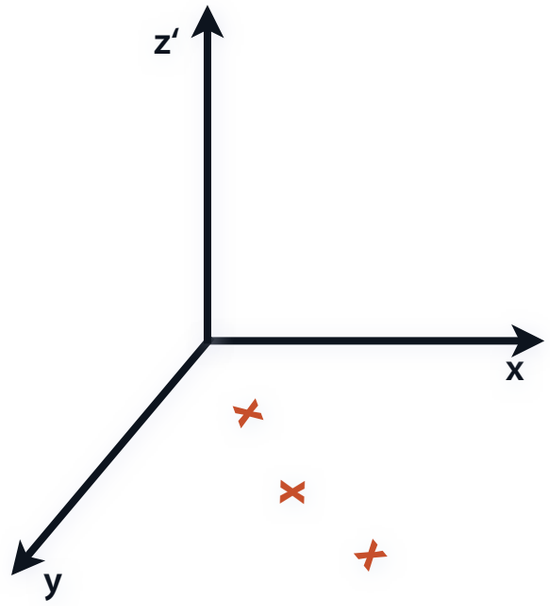


Expand to z‘

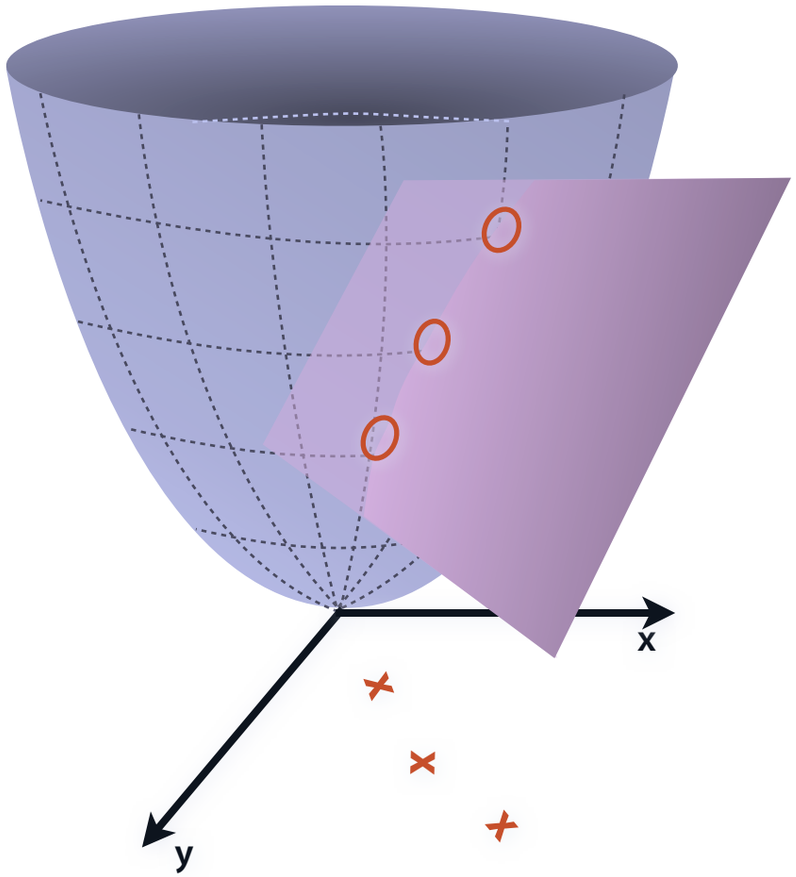


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

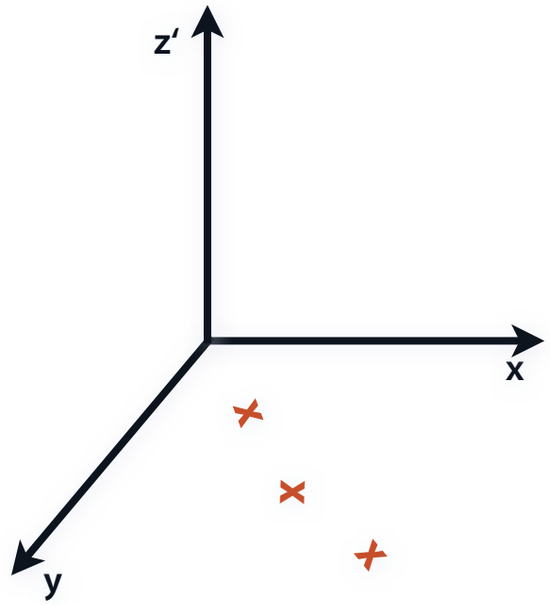


Expand to z‘

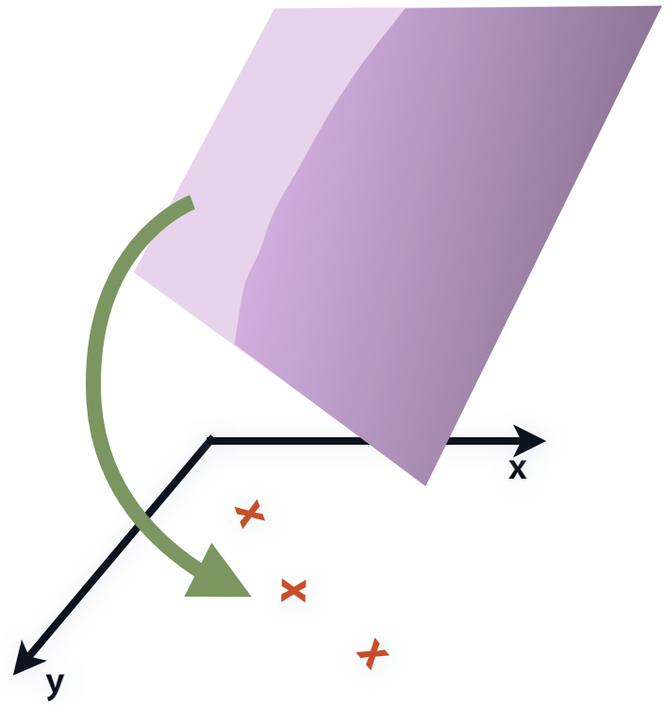


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

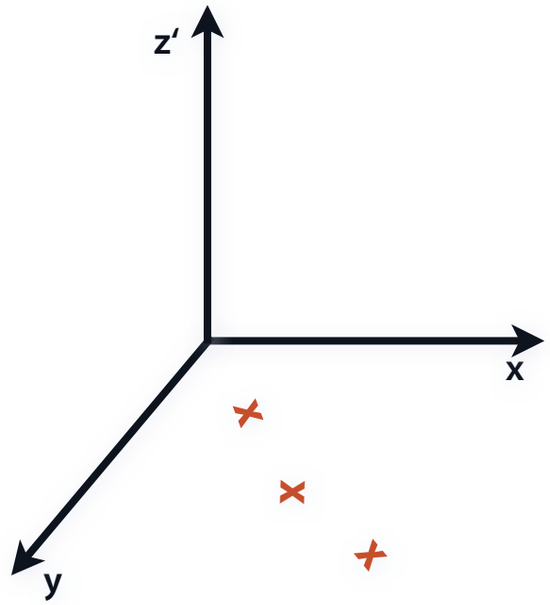


Expand to z‘

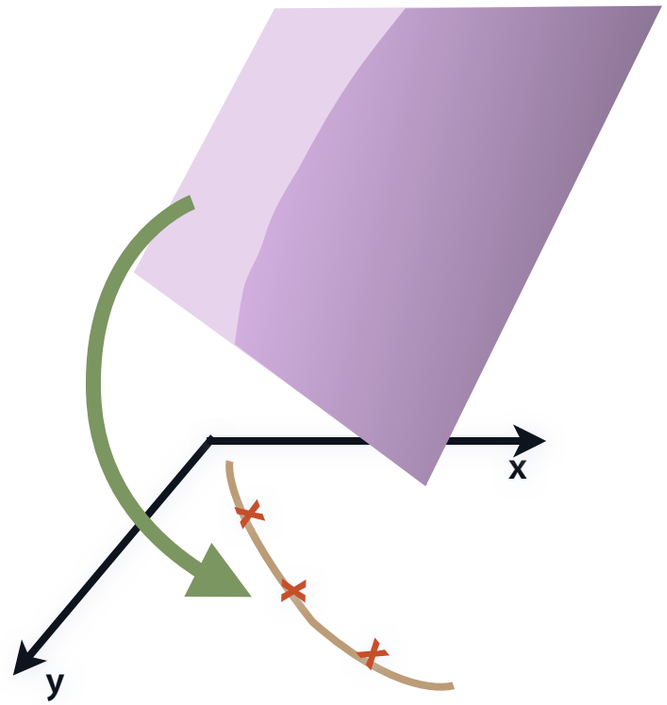


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

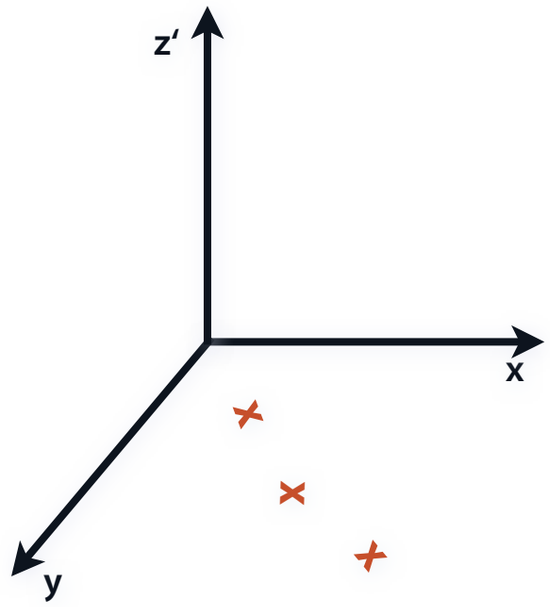


Expand to z‘

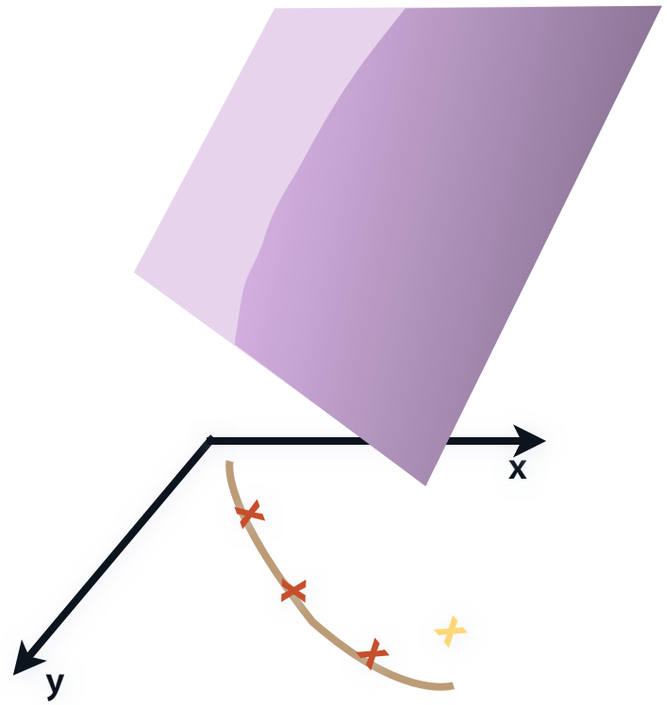


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

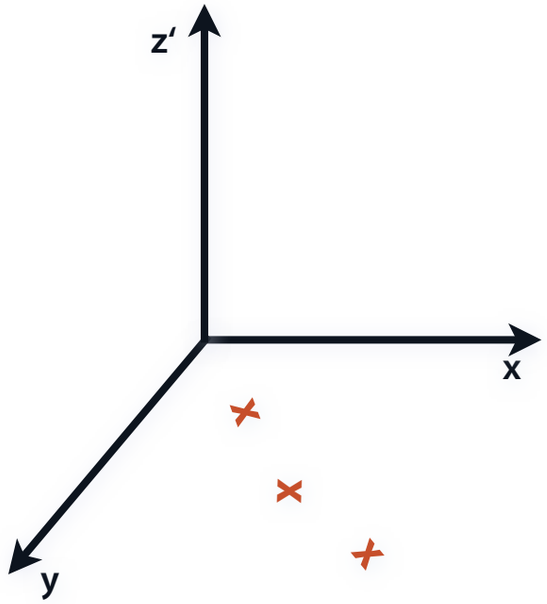


Expand to z‘

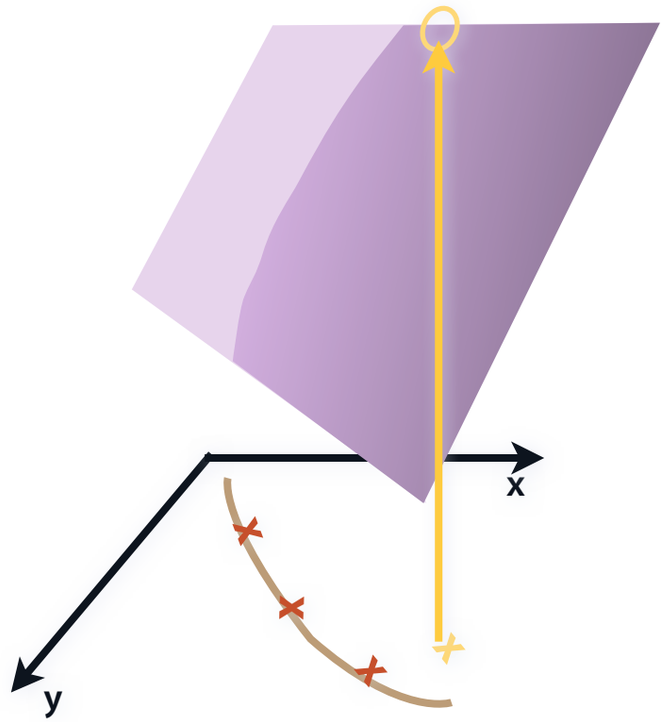


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

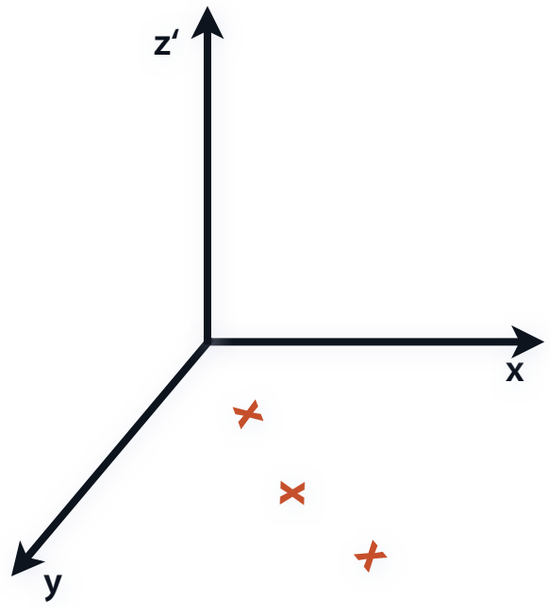


Expand to z‘

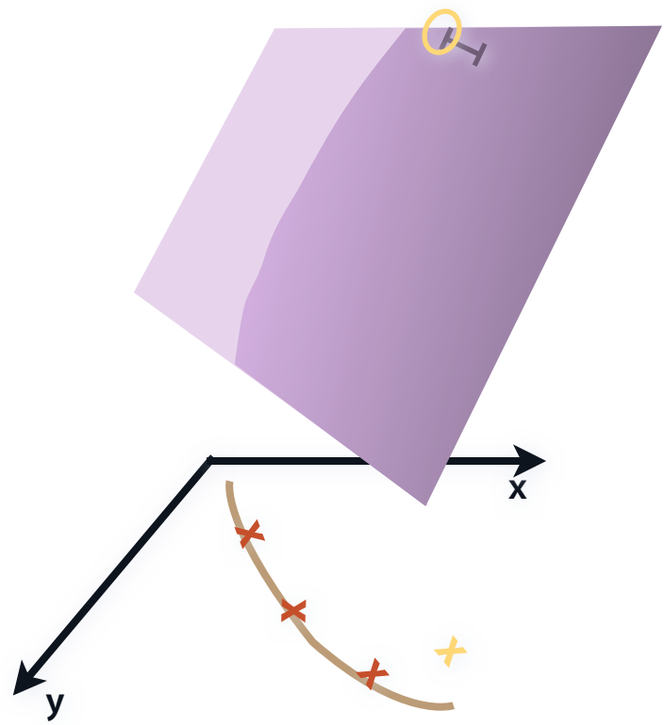


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

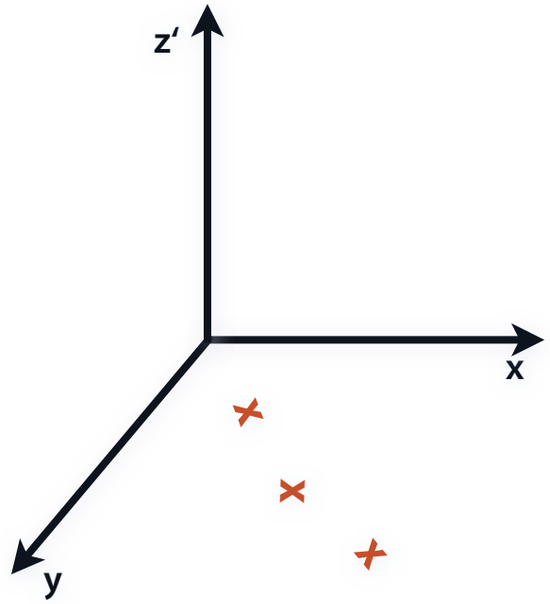


Expand to z‘

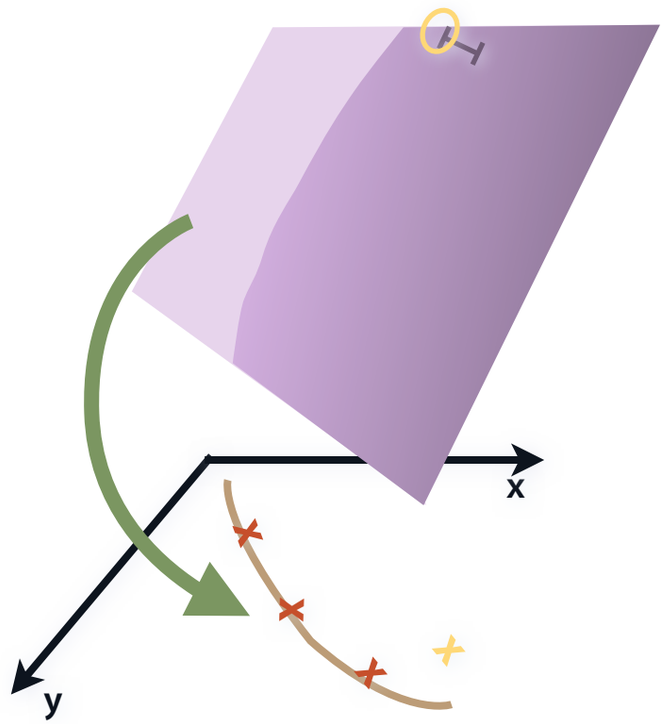


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

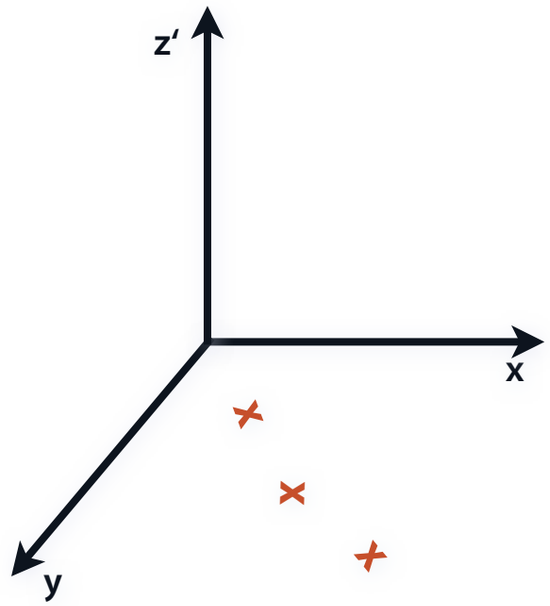


Expand to z‘

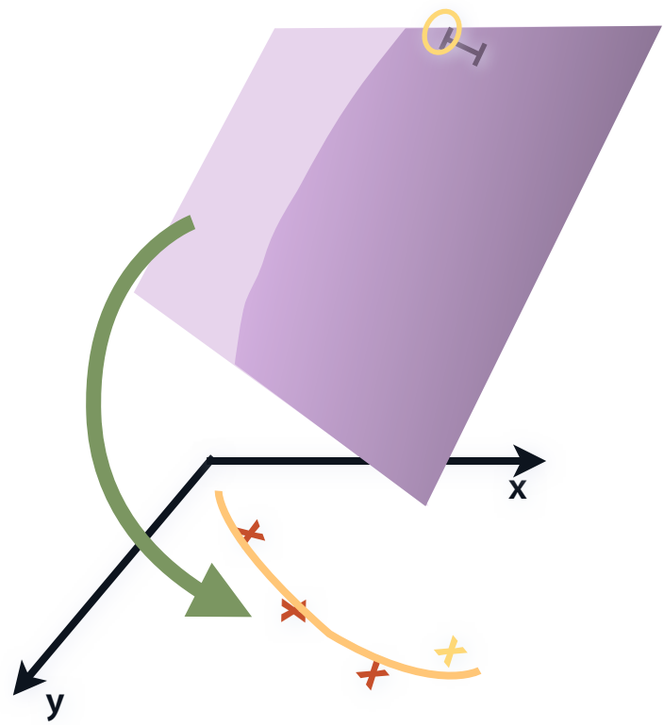


Riemann Surface
(paraboloid)

Riemann Algorithm — 2 Expansion

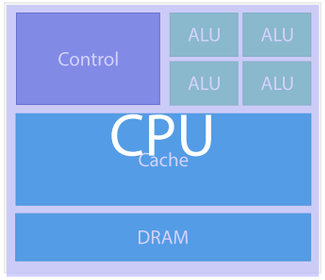


Expand to z‘



Riemann Surface
(paraboloid)

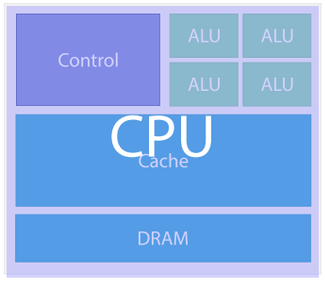
Riemann Algorithm — Triplet Generation

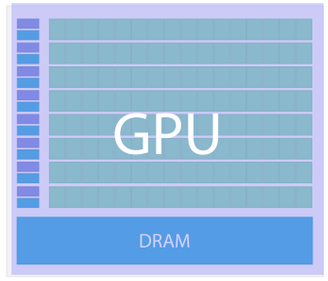


- Three loops to generate triplets serially

```
for (int i = 0; i < hitsInLayerOne.size(); i++) {
  for (int j = 0; j < hitsInLayerTwo.size(); j++) {
    for (int k = 0; k < hitsInLayerThree.size(); k++) {
      /* Triplet Generation */
    }
  }
}
```

Riemann Algorithm — Triplet Generation





- Three loops to generate triplets serially

```
for (int i = 0; i < hitsInLayerOne.size(); i++) {
  for (int j = 0; j < hitsInLayerTwo.size(); j++) {
    for (int k = 0; k < hitsInLayerThree.size(); k++) {
      /* Triplet Generation */
    }
  }
}
```

- Loops are not good parallelizable!
- Needed: Mapping of inherent GPU indexing variable to triplet index

```
int ijk = threadIdx.x + blockIdx.x * blockDim.x;
```

Riemann Algorithm — GPU Version

- Triplet generation: transition via equations

```
for () {for () {for () {}}}  ⟶  int ijk = threadIdx.x + blockIdx.x * blockDim.x;
```

$$n_{\mathrm{Layer}_x} = \frac{1}{2}\left(\sqrt{8x+1}-1\right)$$

$$\mathrm{pos}(n_{\mathrm{Layer}_x}) = \frac{\sqrt[3]{\sqrt{3}\sqrt{243x^2-1}+27x}}{3^{2/3}} + \frac{1}{\sqrt[3]{3}\sqrt[3]{\sqrt{3}\sqrt{243x^2-1}+27x}} - 1$$

- Work by Jonathan Timcheck
  - RISE summer student of Ohio State University at FZJ

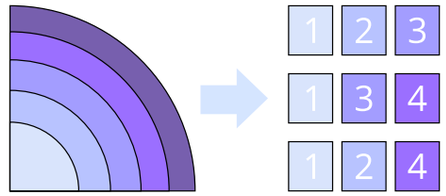
GPU Riemann Algorithm

GPU Riemann Algorithm

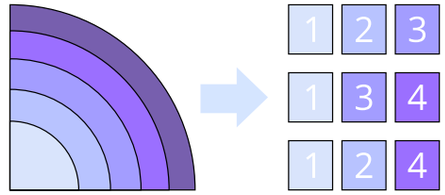
- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*

GPU Riemann Algorithm

- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*



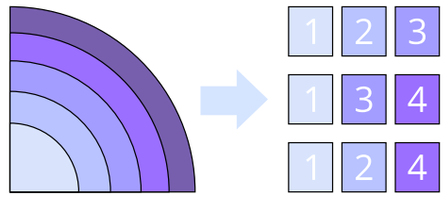
GPU Riemann Algorithm

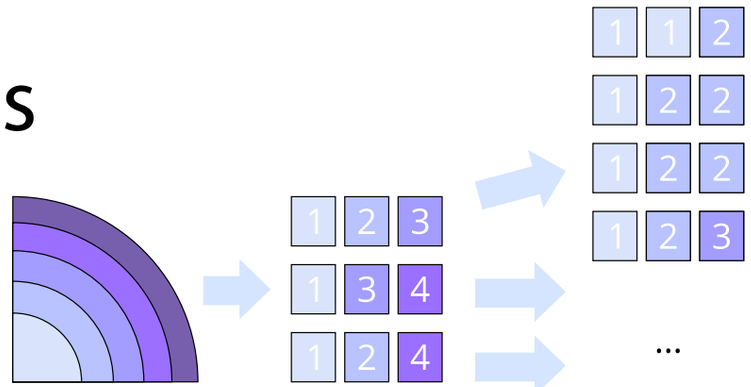


- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

GPU Riemann Algorithm

- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*



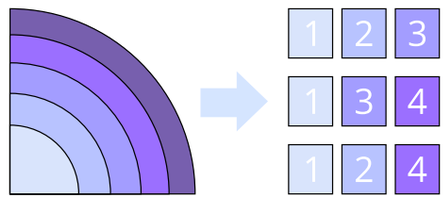


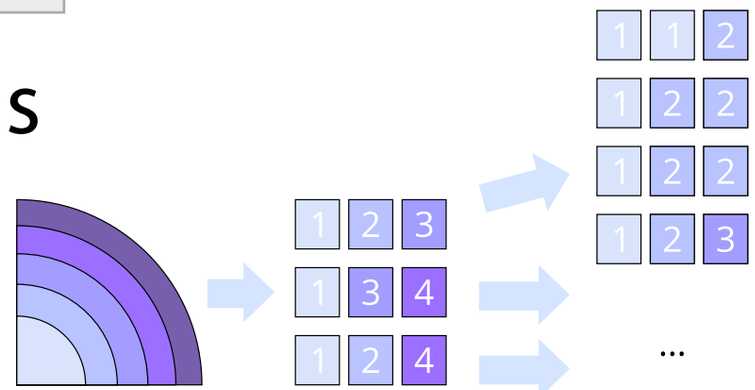
GPU Riemann Algorithm

- Triplet **generation**
  - **Start** with layer triplet: Each creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

Layer triplet:
Combinations of three layers, where each layer has at least one hit in it.

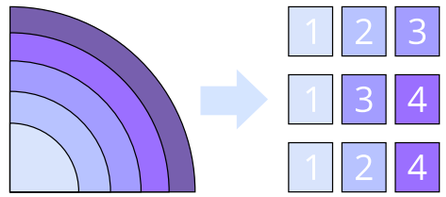
Hit triplet:
Zoom in to layer triplets. All three hit combinations of succeeding layers.

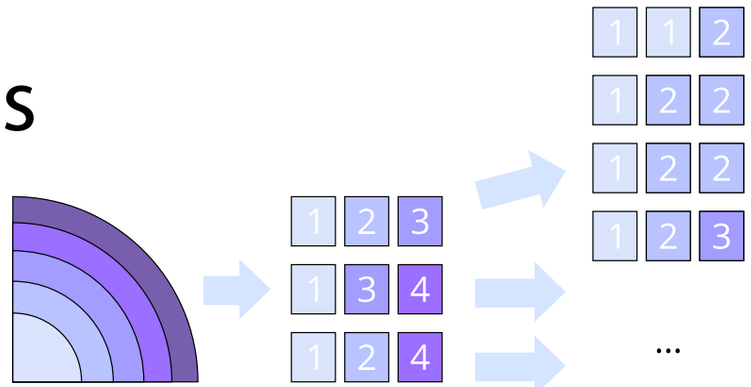




GPU Riemann Algorithm

- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

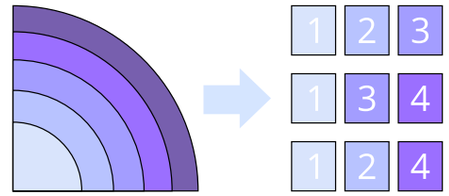


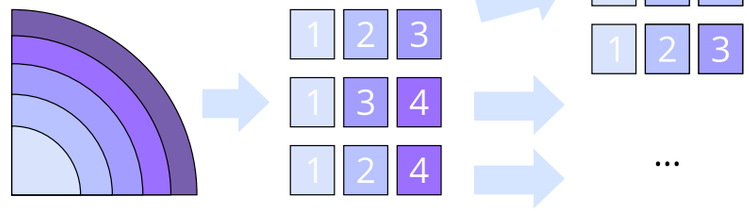


GPU Riemann Algorithm

- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

  → List of hit triplets as seeds

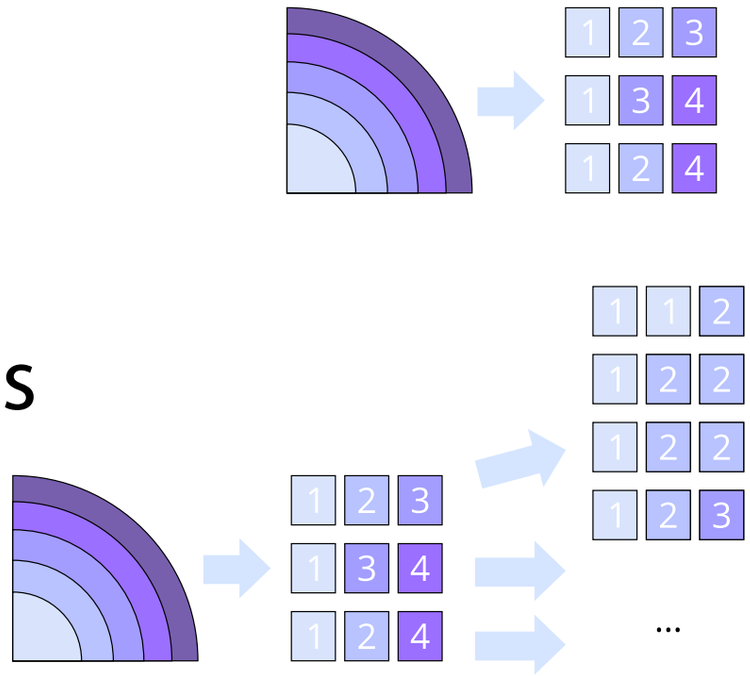




GPU Riemann Algorithm

- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

  → List of hit triplets as seeds

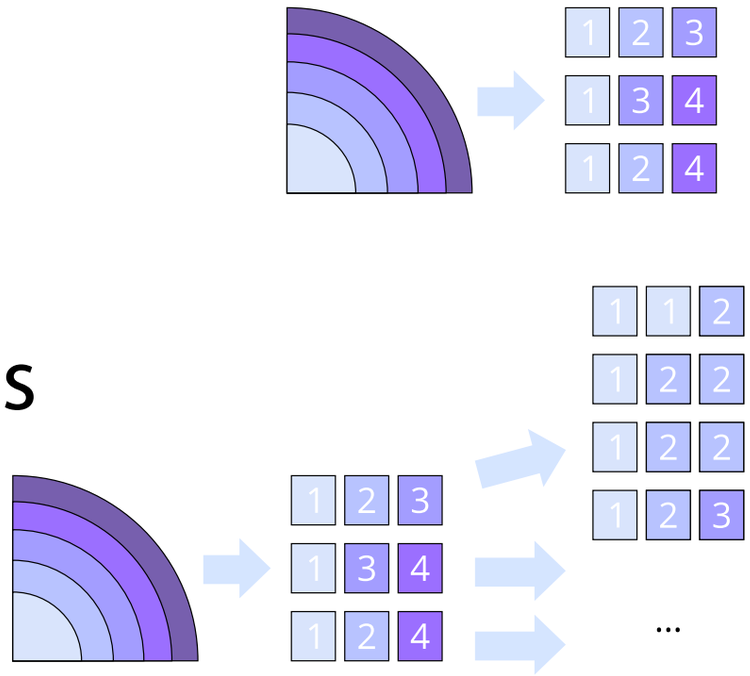


- **Expand** triplets to tracks

GPU Riemann Algorithm

- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

  → List of hit triplets as seeds

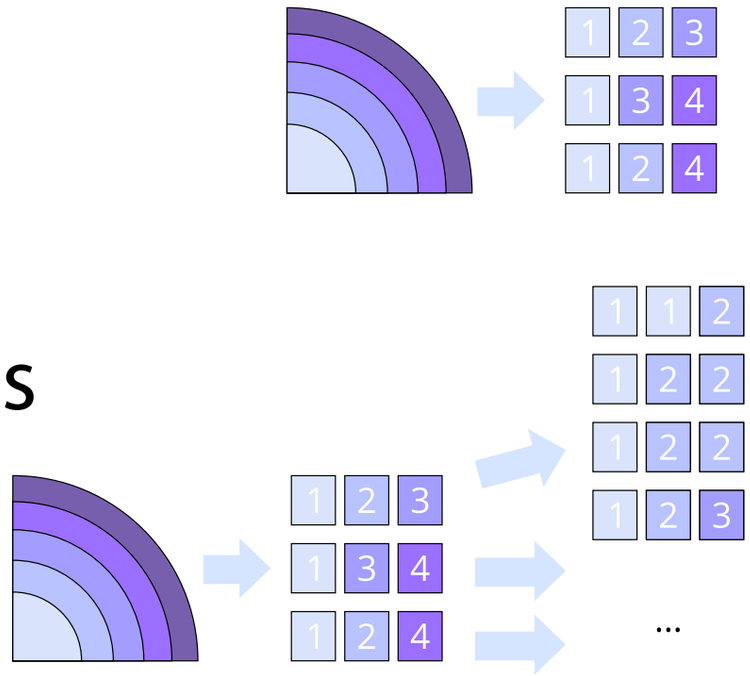


- **Expand** triplets to tracks
  - Add hit one by one if quality criteria are passed
    (from new layer; distance to Riemann plane; s-z quality)

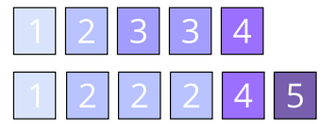
GPU Riemann Algorithm

- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

  → List of hit triplets as seeds



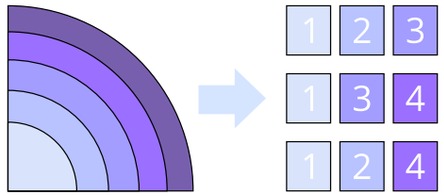
- **Expand** triplets to tracks
  - Add hit one by one if quality criteria are passed
    (from new layer; distance to Riemann plane; s-z quality)

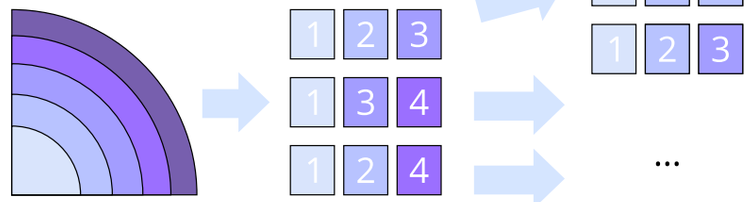


GPU Riemann Algorithm

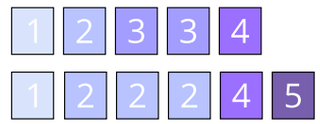
- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

  → List of hit triplets as seeds





- **Expand** triplets to tracks
  - Add hit one by one if quality criteria are passed
    (from new layer; distance to Riemann plane; s-z quality)

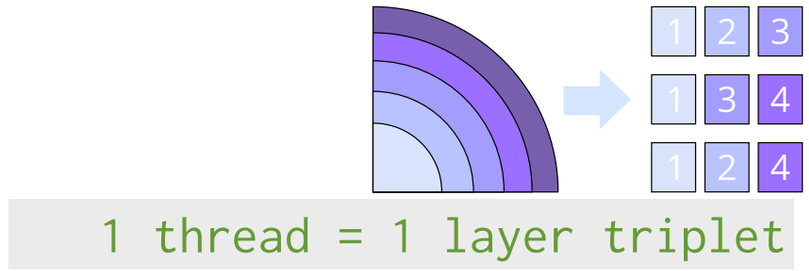


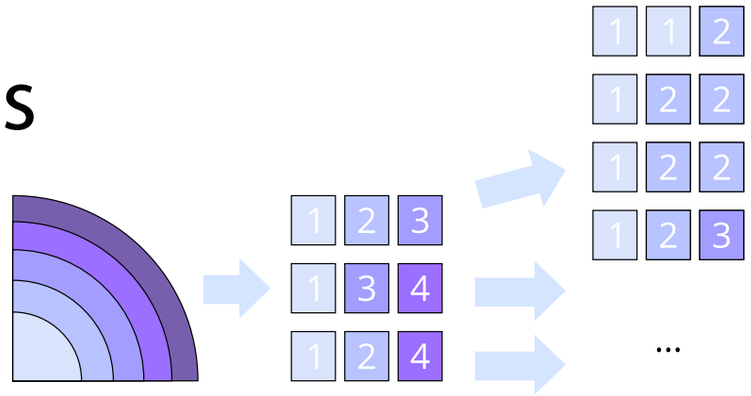
- **Save** successfully found track

GPU Riemann Algorithm

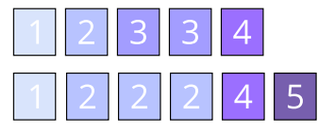
- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

  → List of hit triplets as seeds





- **Expand** triplets to tracks
  - Add hit one by one if quality criteria are passed
    (from new layer; distance to Riemann plane; s-z quality)

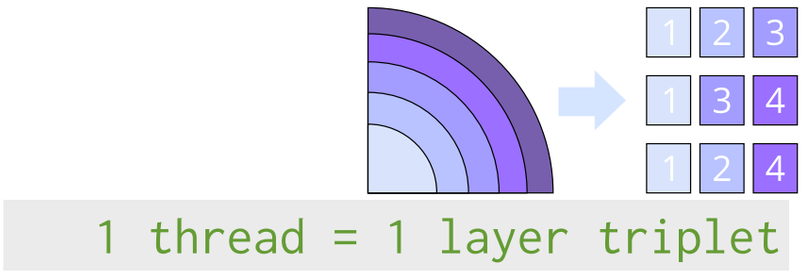


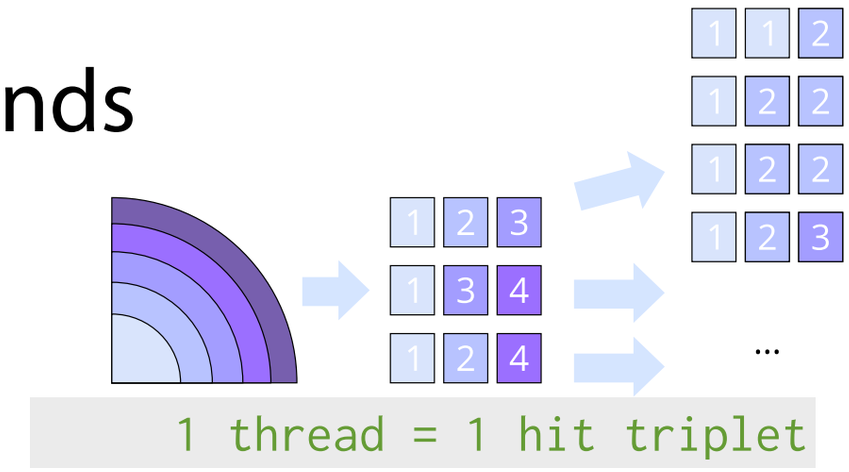
- **Save** successfully found track

GPU Riemann Algorithm

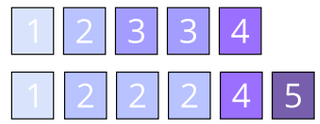
- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

  → List of hit triplets as seeds





- **Expand** triplets to tracks
  - Add hit one by one if quality criteria are passed
    (from new layer; distance to Riemann plane; s-z quality)

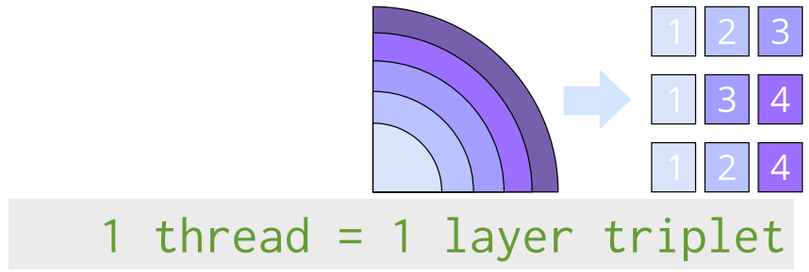


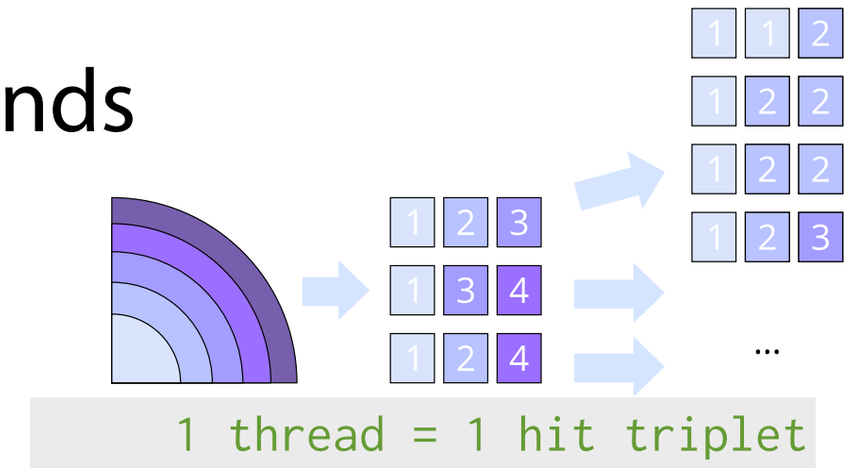
- **Save** successfully found track

GPU Riemann Algorithm

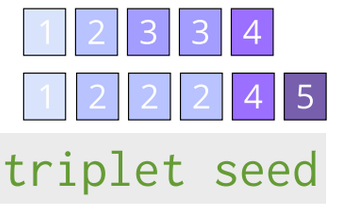
- Triplet **generation**
  - **Start** with layer triplet: Each thread creates unique *layer triplet*
  - **Grow** to hit triplet: Each thread expands *layer triplet* to unique *hit triplet*

  → List of hit triplets as seeds





- **Expand** triplets to tracks
  - Add hit one by one if quality criteria are passed
    (from new layer; distance to Riemann plane; s-z quality)



- **Save** successfully found track

GPU Riemann Algorithm — Performance

- Efficiency & Ghost ratio

	Efficiency | Ghost Ratio			
Dist. Cut Param	0.06	0.575 | 0.205	0.601 | 0.255	0.623 | 0.306
	0.05	0.562 | 0.205	0.589 | 0.254	0.612 | 0.304
	0.04	0.541 | 0.204	0.566 | 0.254	0.588 | 0.303
		0.04	0.05	0.06
	s-z Chi Square Cut Parameter			

CPU Riemann Cuts

4.0	0.877 | 4.652	0.902 | 7.621	0.897 | 10.08
3.0	0.881 | 4.032	0.910 | 6.730	0.913 | 9.030
2.0	0.881 | 3.360	0.914 | 5.708	0.918 | 7.793
	0.5	1.0	1.5

Benchmark Extreme Cuts

GPU Riemann Algorithm — Performance

- Efficiency & Ghost ratio

	Efficiency | Ghost Ratio			
Dist. Cut Param	0.06	0.575 | 0.205	0.601 | 0.255	0.623 | 0.306
	0.05	0.562 | 0.205	0.589 | 0.254	0.612 | 0.304
	0.04	0.541 | 0.204	0.566 | 0.254	0.588 | 0.303
		0.04	0.05	0.06
	s-z Chi Square Cut Parameter			

CPU Riemann Cuts

4.0	0.877 | 4.652	0.902 | 7.621	0.897 | 10.08
3.0	0.881 | 4.032	0.910 | 6.730	0.913 | 9.030
2.0	0.881 | 3.360	0.914 | 5.708	0.918 | 7.793
	0.5	1.0	1.5

Benchmark Extreme Cuts

- Time for one event *(NV Profiler; @Juhydra: Tesla K20X)*

Time(%)	Time	Calls	Avg	Min	Max	Name
75.55%	**439.49us**	1	439.49us	439.49us	439.49us	**extend_cut_hit_triplets_k**
5.96%	34.656us	4	8.6640us	2.3360us	22.432us	[CUDA memcpy DtoH]
4.36%	25.344us	1	25.344us	25.344us	25.344us	cut_hit_triplets_k
4.26%	24.800us	6	4.1330us	3.7760us	5.3440us	[CUDA memset]
2.57%	14.976us	1	14.976us	14.976us	14.976us	generate_hit_triplet
2.44%	14.176us	1	14.176us	14.176us	14.176us	generate_layer_triplets
1.30%	7.5520us	1	7.5520us	7.5520us	7.5520us	void thrust
1.11%	6.4640us	1	6.4640us	6.4640us	6.4640us	void thrust
1.11%	6.4640us	1	6.4640us	6.4640us	6.4640us	void thrust
0.89%	5.1520us	5	1.0300us	928ns	1.3440us	[CUDA memcpy HtoD]
0.45%	2.6240us	1	2.6240us	2.6240us	2.6240us	project_onto_paraboloid_k

GPU Riemann Algorithm — Summary

- Running port of CPU Riemann to GPU (J. Timcheck)
  + Improvements / needed changes wrt to CPU version

GPU Riemann Algorithm — Summary

- Running port of CPU Riemann to GPU (J. Timcheck)

  + Improvements / needed changes wrt to CPU version

To Dos / Plans	**Advantages** of Riemann algorithm	Draw Backs	~~Problems~~ Challenges
• Measurement Uncertainties • Cuts *(Extension: Hit to close, Zero crossing)* • Parallelism *(32 threads per seed, not 1)* • Track merger • Integrate to PandaRoot	• Secondaries • Runs also only with MVD • Basis for more sophisticated algorithms • Uncertainties • Fast track fitter *(track parameters)*	• Combinatorically explosive - Many combinations *(if used bluntly)* - Esp. if used as finder - Pre-steps needed • Jonathan‘s internship is over…	• Include more subdetectors • Make time-based

GPU Riemann Algorithm — Summary

- Running port of CPU Riemann to GPU (J. Timcheck)

  + Improvements / needed changes wrt to CPU version

To Dos / Plans	**Advantages** of Riemann algorithm	Draw Backs	~~Problems~~ Challenges
• Measurement Uncertainties • Cuts *(Extension: Hit to close, Zero crossing)* • Parallelism *(32 threads per seed, not 1)* • Track merger • Integrate to PandaRoot	• Secondaries • Runs also only with MVD • Basis for more sophisticated algorithms • Uncertainties • Fast track fitter *(track parameters)*	• Combinatorically explosive - Many combinations *(if used bluntly)* - Esp. if used as finder - Pre-steps needed • Jonathan's internship is over…	• Include more subdetectors • Make time-based

- Code at: https://subversion.gsi.de/trac/fairroot/browser/pandaroot/development/aherten/GpuRiemann
- Extensive documentation at: http://panda-wiki.gsi.de/cgi-bin/view/Computing/RiemannTrackFinder (+Summary of theory behind Riemann algorithm)

JÜLICH
FORSCHUNGSZENTRUM

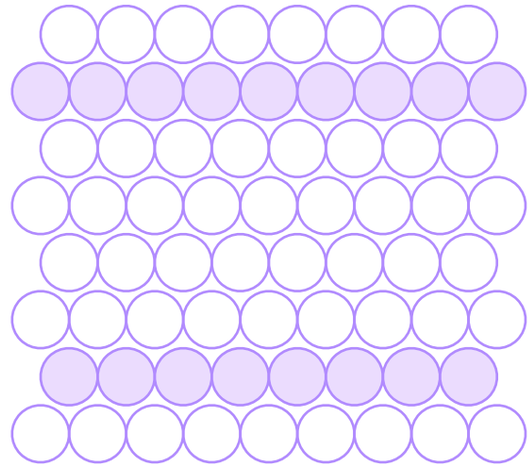
panda

ALGORITHMS

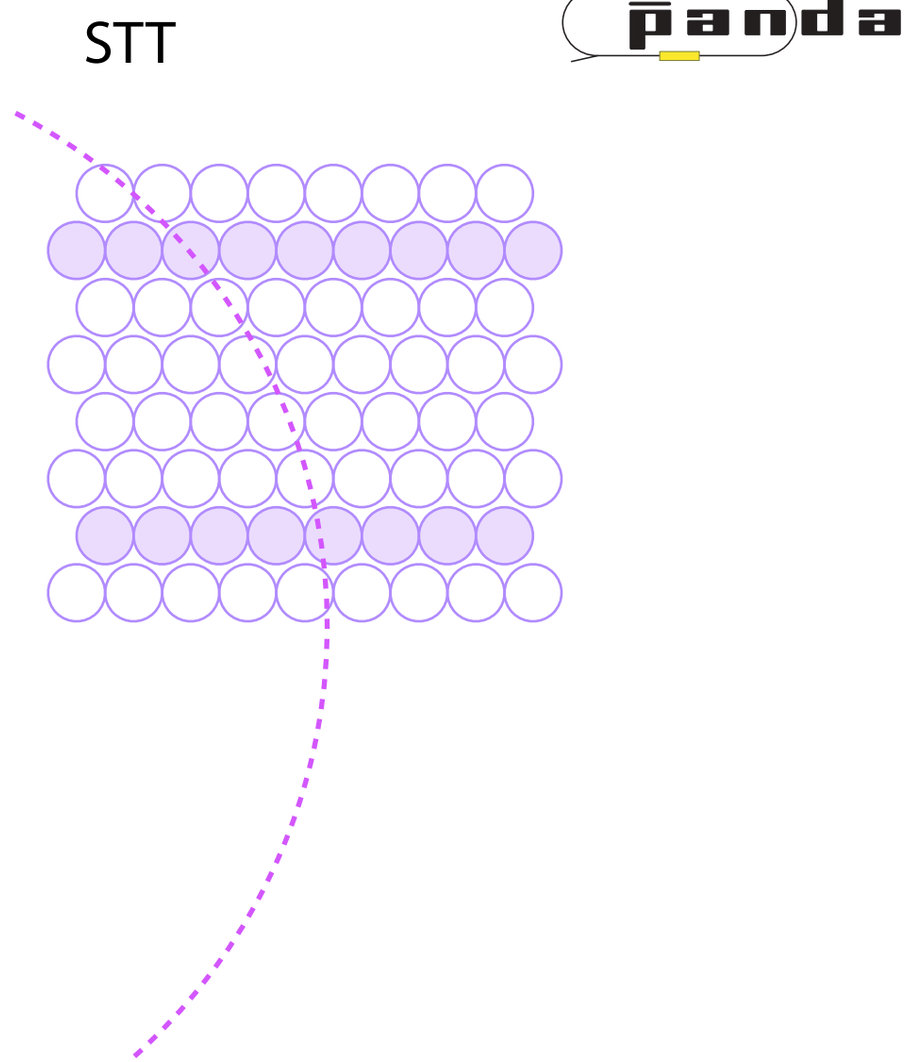
Triplet Finder

Triplet Finder — Method

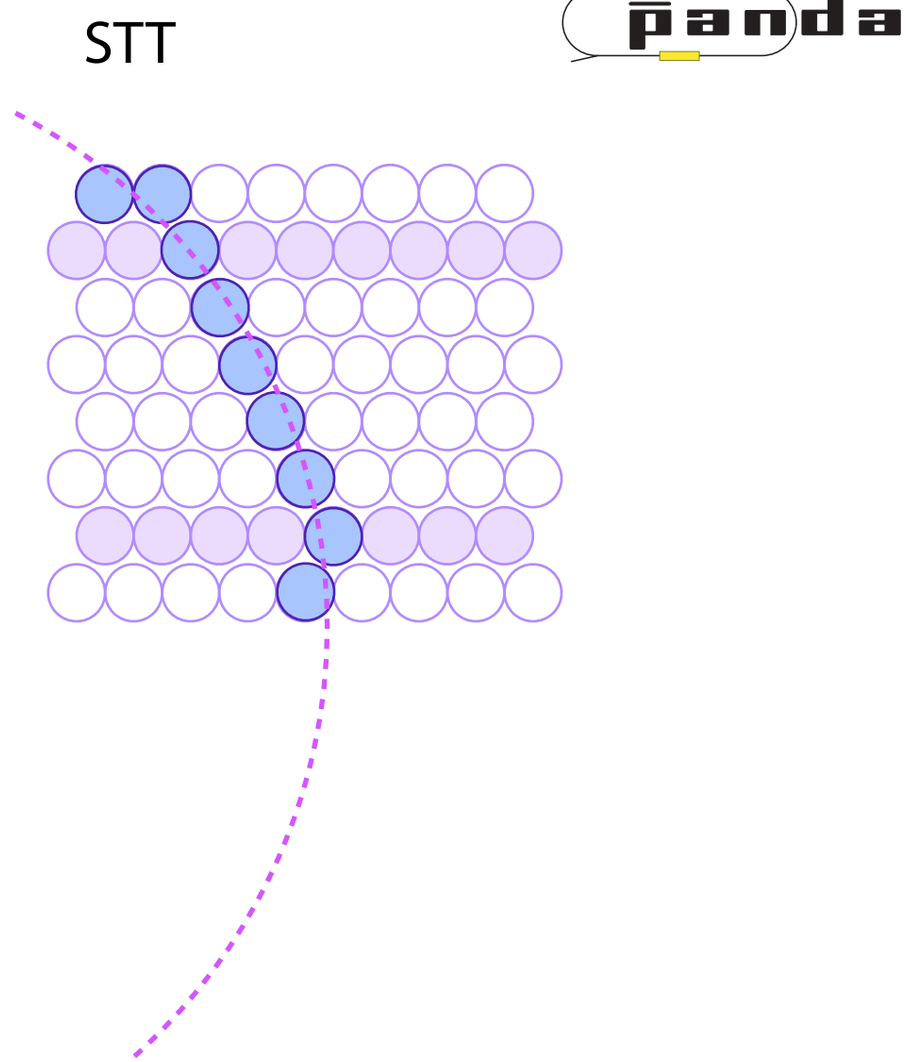
STT



Triplet Finder — Method

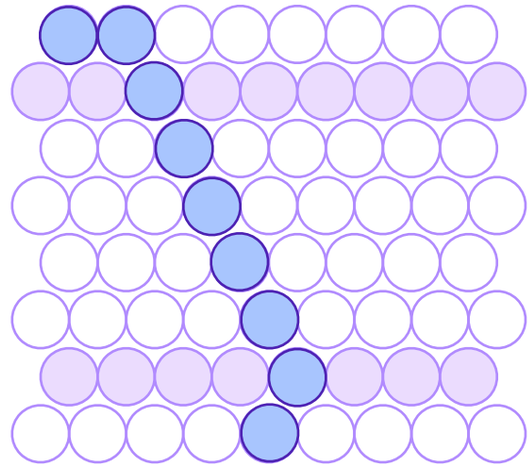


Triplet Finder — Method



Triplet Finder — Method

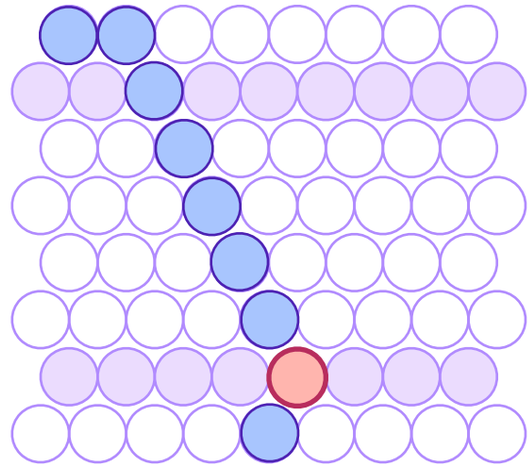
STT



Triplet Finder — Method

- STT hit in **pivot** straw

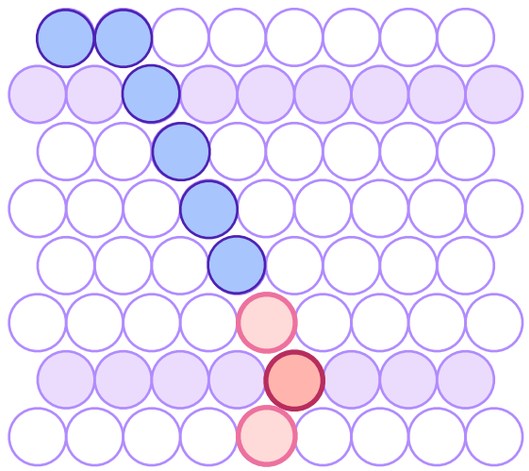
STT



Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

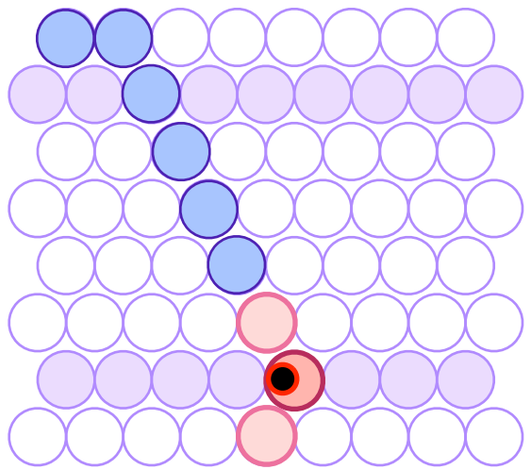
  → Create **virtual hit** *(triplet)* at center of gravity *(**cog**)*



Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity *(**cog**)*
- **Combine** with

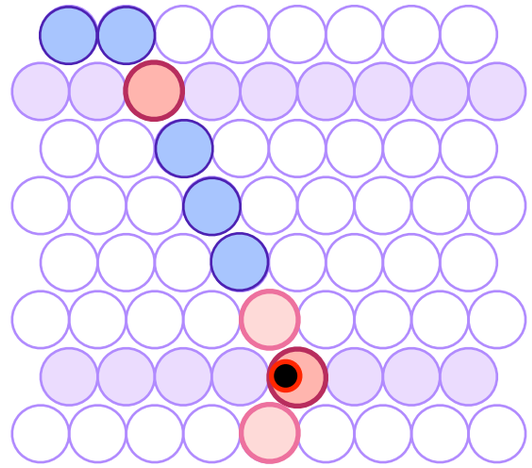


Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

STT



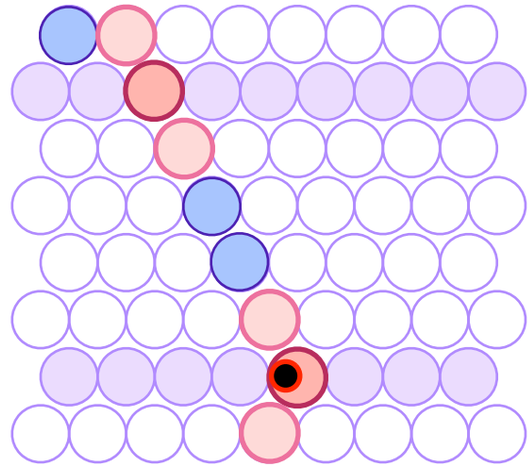
1. Second STT pivot-*cog* virtual hit

Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

STT



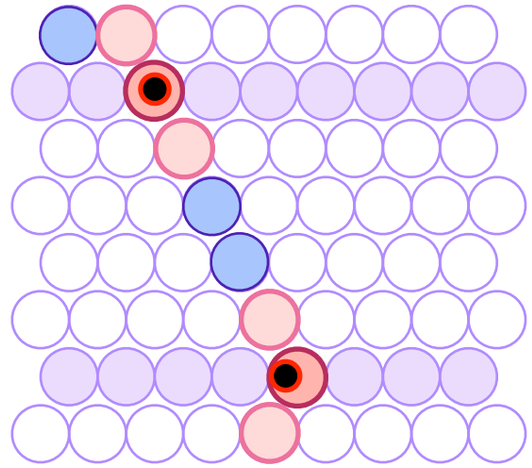
1. Second STT pivot-*cog* virtual hit

Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

STT



1. Second STT pivot-*cog* virtual hit

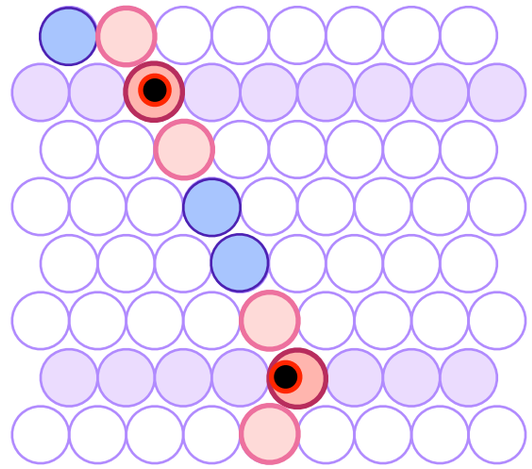
Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with

1. Second STT pivot-*cog* virtual hit
2. Interaction point

STT

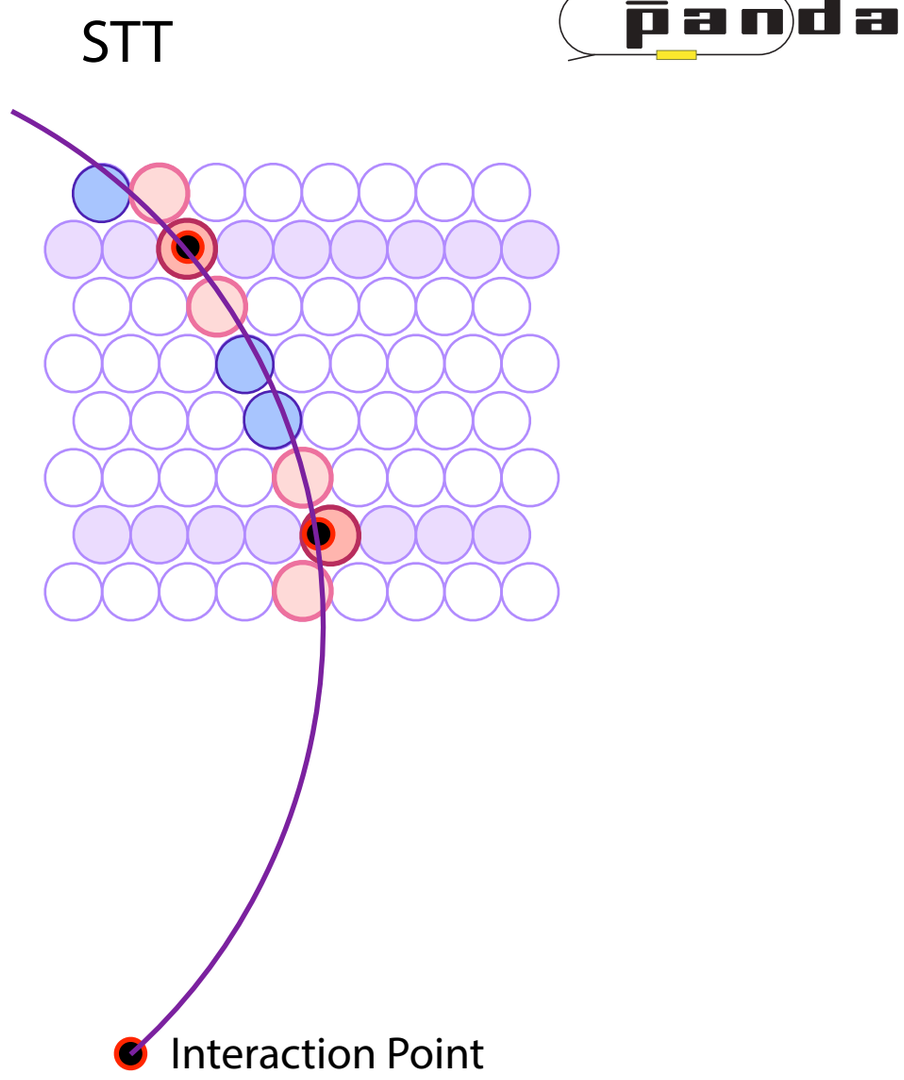


Interaction Point

Triplet Finder — Method

- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with
  1. Second STT pivot-*cog* virtual hit
  2. Interaction point
- Calculate **circle** through three points

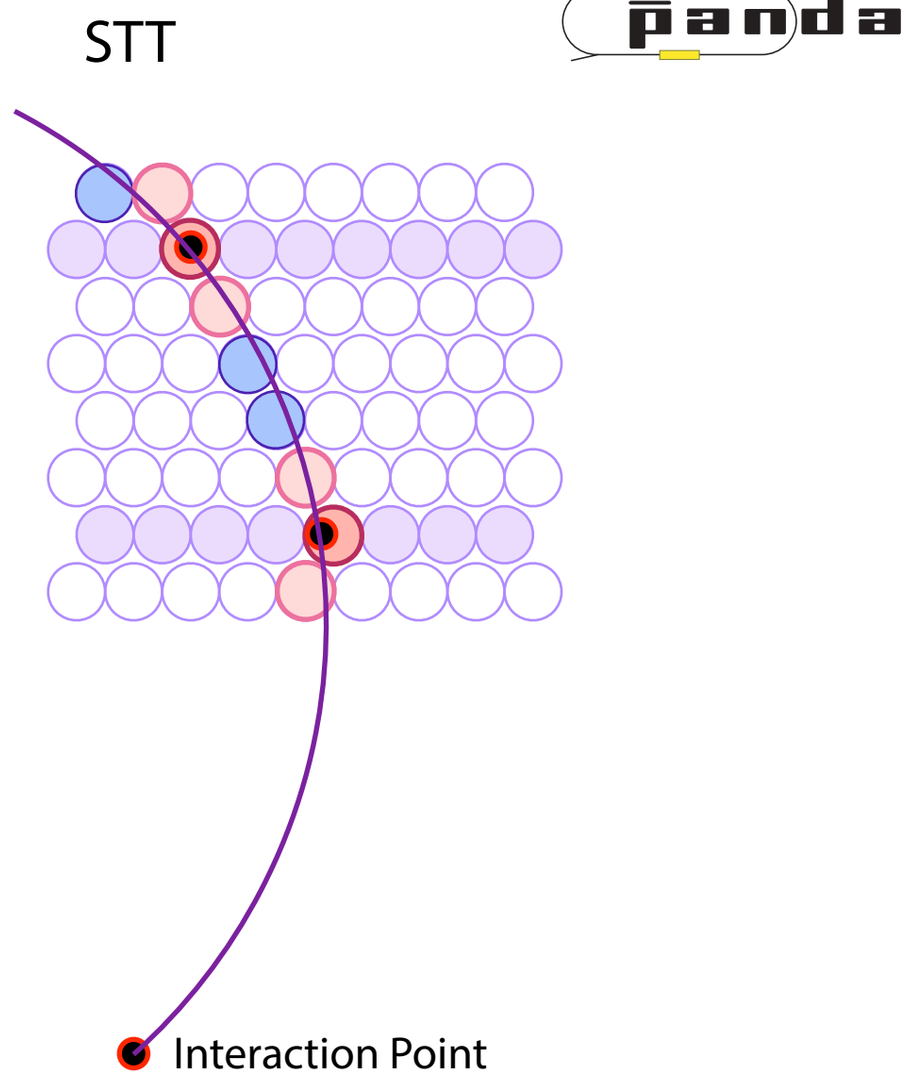


Triplet Finder — Method

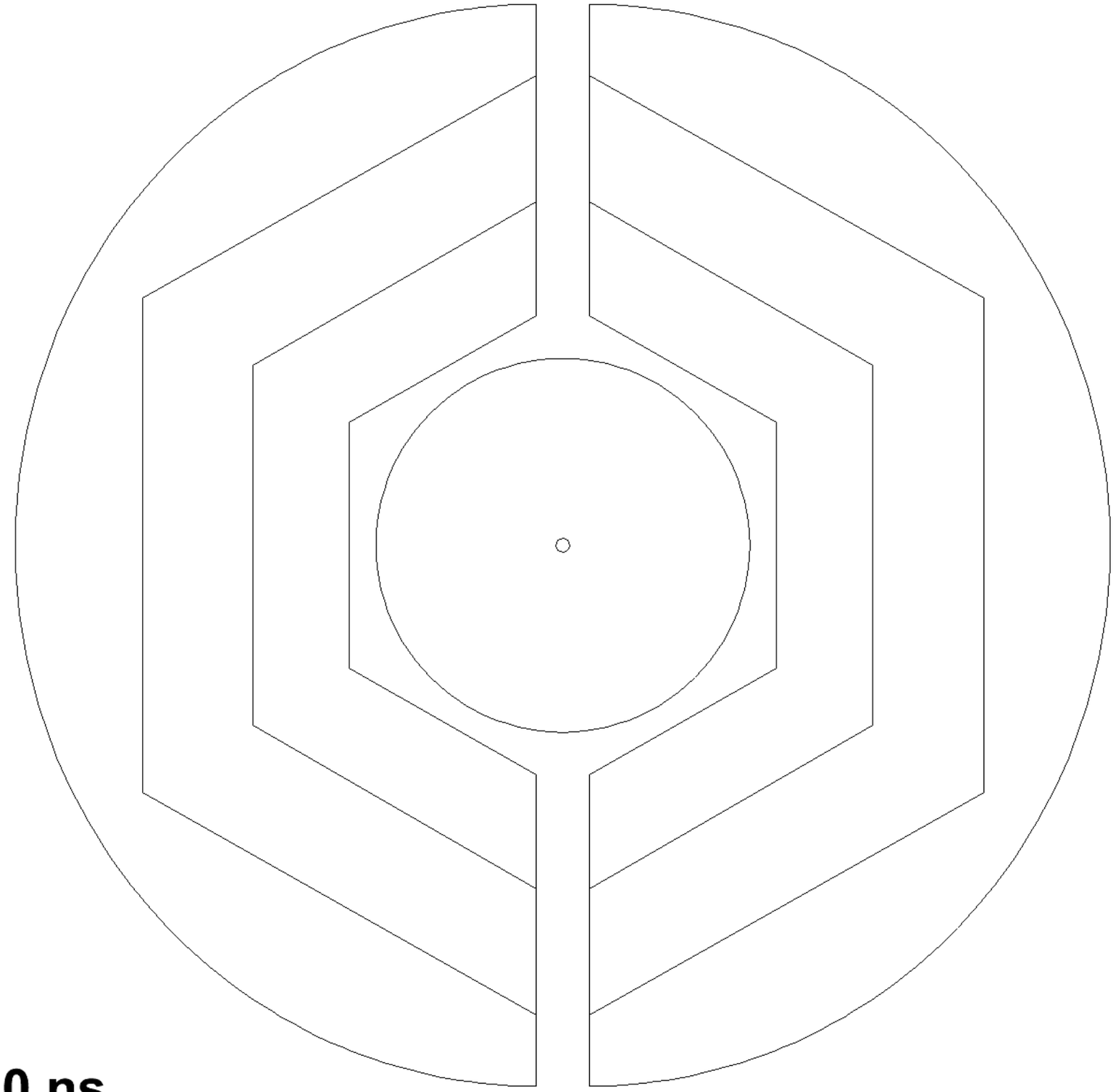
- STT hit in **pivot** straw
- Find surrounding hits

  → Create **virtual hit** *(triplet)* at center of gravity ***(cog)***
- **Combine** with
  1. Second STT pivot-*cog* virtual hit
  2. Interaction point
- Calculate **circle** through three points

→ **Track** Candidate



Triplet Finder — Animation



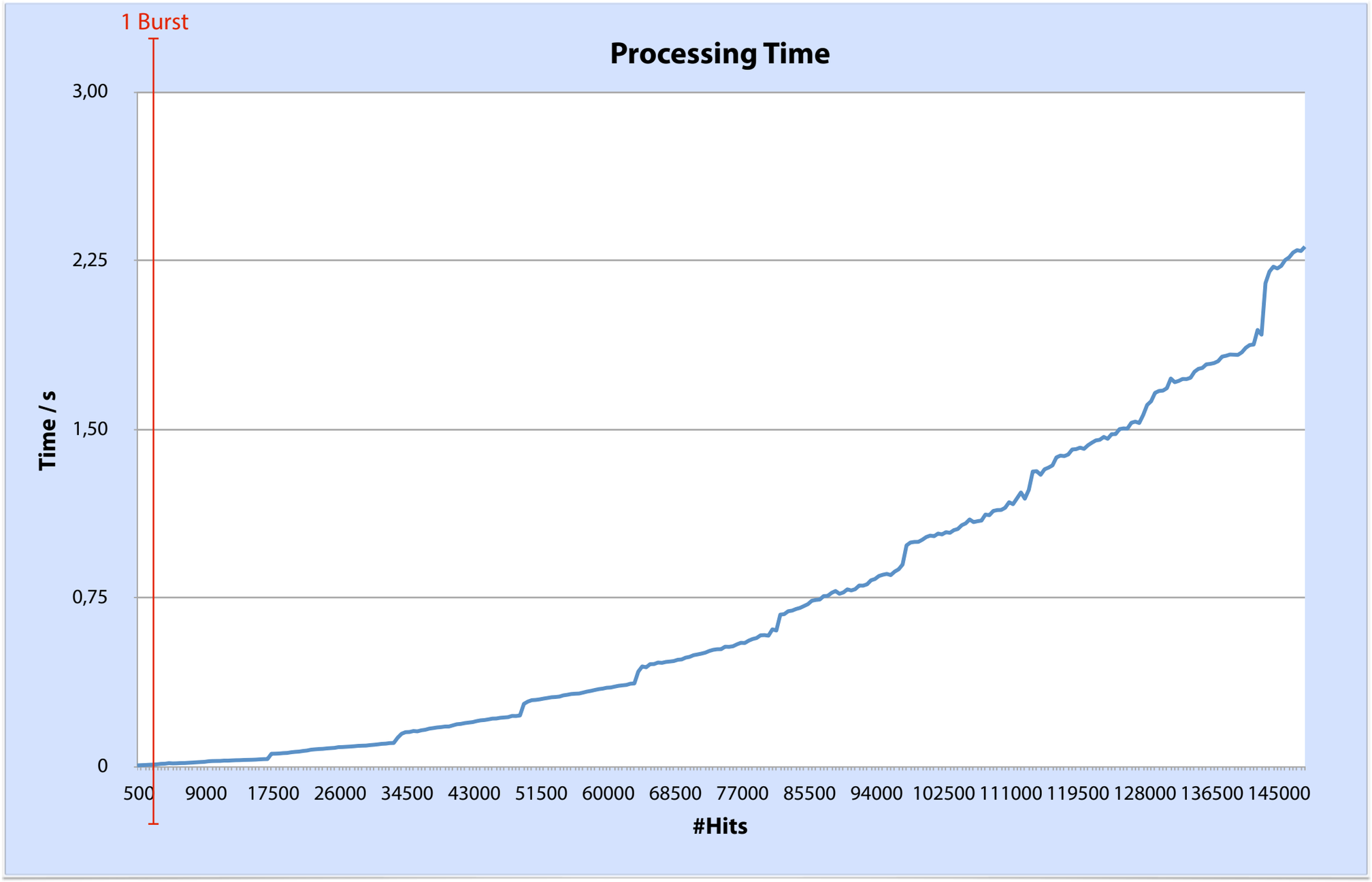
0 ns

- Isochrone *early*
- Isochrone *early* & *skewed*
- Isochrone *close*
- Isochrone *late*
- MVD hit
- **Triplet**
- Track *current*
- Track *timed out*

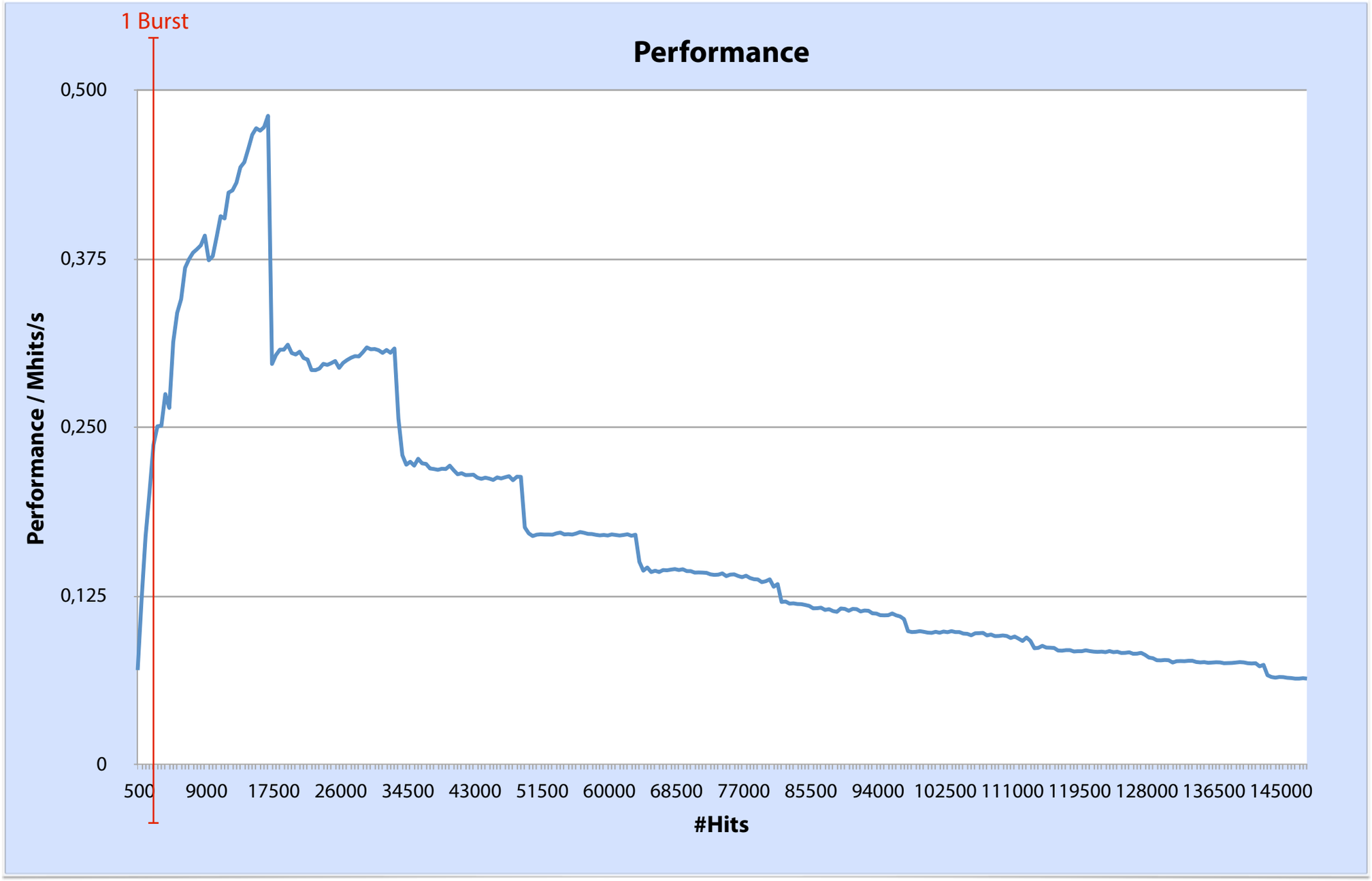
Triplet Finder — GPU Port

- Original algorithm by **Marius C. Mertens** *et al.*
  - Implemented in PandaRoot (also using skewed straws)
  - Reconstruction efficiency & other quality criteria: see Marius' talks on Triplet Finder (eg. last PANDA meeting)
  - Features
    - Fast & robust algorithm
    - No isochrones needed
    - Many tuning possibilities
- Ported to GPU by **Andrew A. Adinetz** (NVIDIA Application Lab)
  - Thrust, CUDA, some dynamic parallelism
  - Supporting skewed straws
  - Quality of results comparable to CPU version *(float vs. double)*

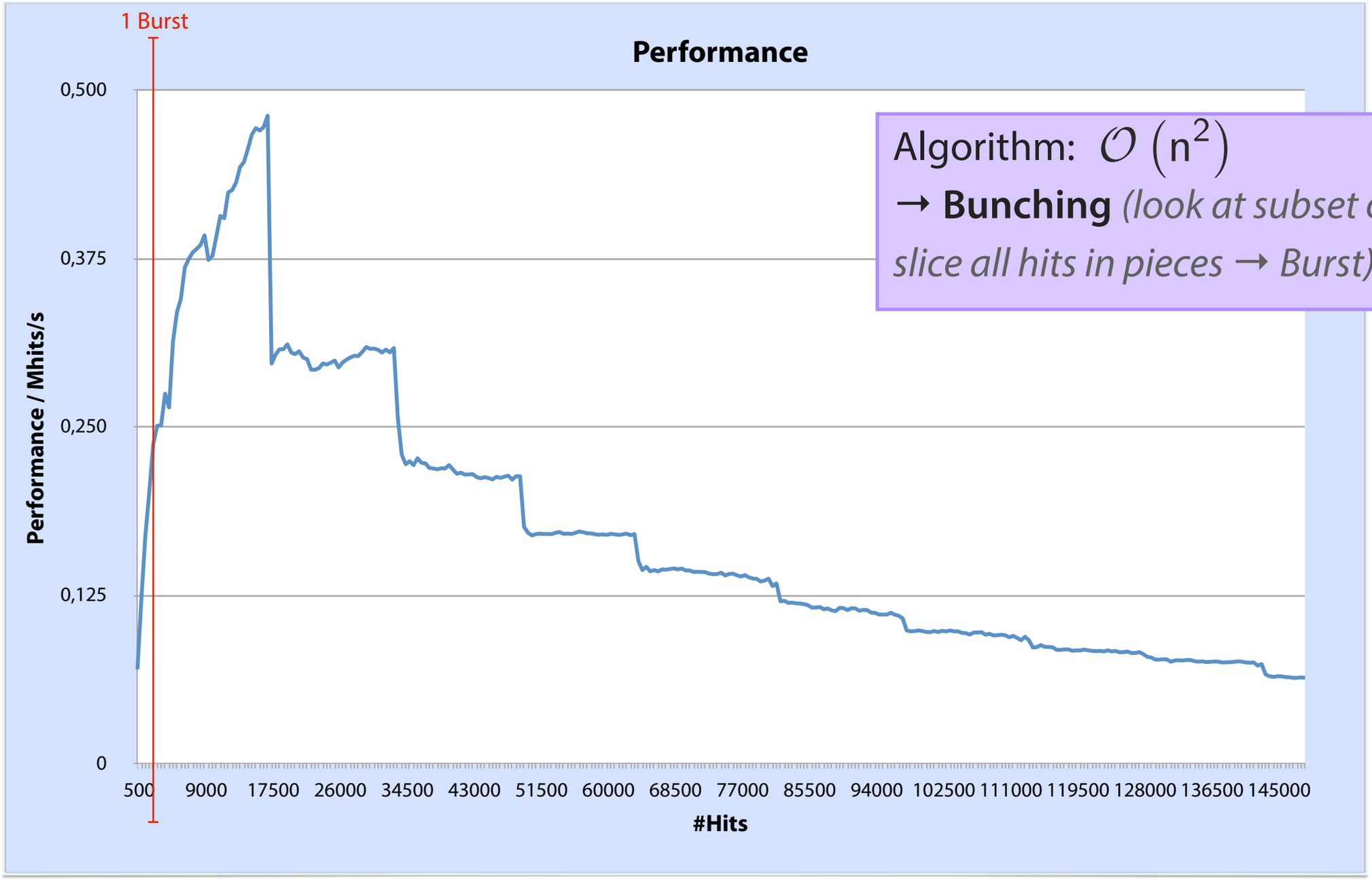
GPU Triplet Finder — Speed



GPU Triplet Finder — Speed



GPU Triplet Finder — Speed



Algorithm: $\mathcal{O}\left(n^2\right)$

→ **Bunching** *(look at subset of hits; slice all hits in pieces → Burst)*

GPU Triplet Finder — Summary

- Running port of CPU Triplet Finder to GPU
  by Andrew Adinetz (NVIDIA Application Lab)

GPU Triplet Finder — Summary

- Running port of CPU Triplet Finder to GPU by Andrew Adinetz (NVIDIA Application Lab)

To Dos / Plans	Advantages	Draw Backs	~~Problems~~ Challenges
• Bunching *(As flexible wrapper, to be used for different algorithms)* • Integrate to PandaRoot	• Fast • No isochrones needed • Already built for time-based hits *(no events)*	• Needs compute capability 3.5	

GPU Triplet Finder — Summary

- Running port of CPU Triplet Finder to GPU by Andrew Adinetz (NVIDIA Application Lab)

To Dos / Plans	Advantages	Draw Backs	~~Problems~~ Challenges
• Bunching *(As flexible wrapper, to be used for different algorithms)* • Integrate to PandaRoot	• Fast • No isochrones needed • Already built for time-based hits *(no events)*	• Needs compute capability 3.5	

- Code at: https://subversion.gsi.de/trac/fairroot/browser/pandaroot/development/aherten/GpuTripletFinder

SUMMARY

Summary — General

- **Three** algorithms under GPU investigation
  - All are fast and GPU-suited
  - Different advantages & disadvantages
- Great help by Andrew Adinetz *& others* of **NVIDIA Application Lab**
- Not yet compared in terms of ε/purity/ghost/etc.
  → Tobias talk @ discussion round
- Not yet looked into GPUs beyond algorithmic stage *(data distribution etc.)*

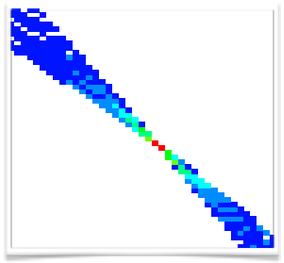
Summary — General

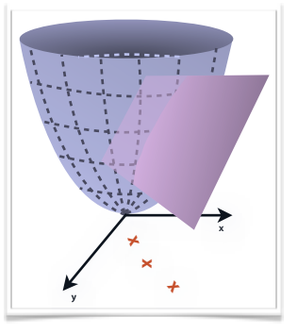
- **Three** algorithms under GPU investigation
  - All are fast and GPU-suited
  - Different advantages & disadvantages
- Great help by Andrew Adinetz *& others* of **NVIDIA Application Lab**
- Not yet compared in terms of ε/purity/ghost/etc.
  → Tobias talk @ discussion round
- Not yet looked into GPUs beyond algorithmic stage *(data distribution etc.)*

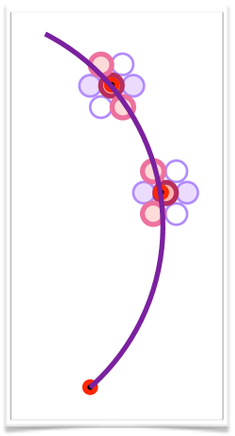
GTLI (General Topics to Look Into)

- Bunching
- Time-based structure
- Track Merger
- Which algorithm for which sub-detector
- Algorithm interplay / hybridization
- Specifically test of sub-parts of the algorithm run faster on CPUs *(CPU-GPU re-distribution)*
- Quality analytics

Summary — Algorithm

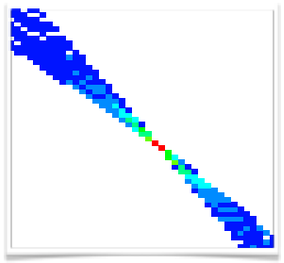


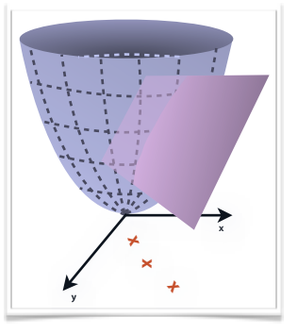


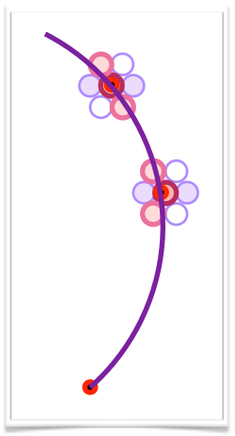


Algorithm	Note	Biggest Challenge	Detectors used
Hough Transform — Thrust	Fast but inflexible	No satisfying multipeak finder	MVD STT
Hough Transform — Plain CUDA *(with 1° of dynamic parallelism)*	Fast		MVD
GPU Riemann Track Finder	Promising initial implementation	Testing, Combinatorics	MVD GEM
GPU Triplet Finder	Fast, robust, scalable	Manpower	STT

Summary — Algorithm

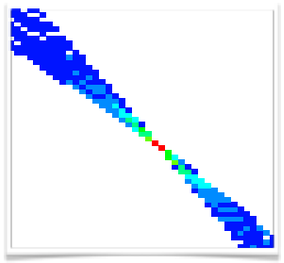


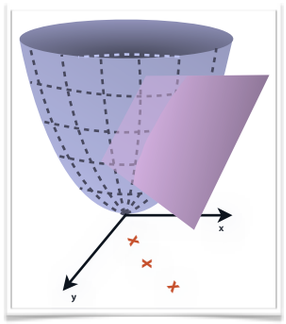


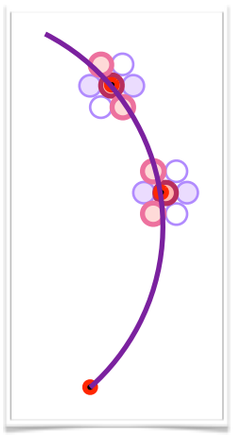


Algorithm	Note	Biggest Challenge	Detectors used
Hough Transform — Thrust	Fast but inflexible	No satisfying multipeak finder	MVD STT
Hough Transform — Plain CUDA *(with 1° of dynamic parallelism)*	Fast		MVD
GPU Riemann Track Finder	Promising initial implementation	Testing, Combinatorics	MVD GEM
GPU Triplet Finder	Fast, robust, scalable	Manpower	STT
GPU STT Cell Finder	→ see next talk by Tobias		

Summary — Algorithm







Algorithm	Note	Biggest Challenge	Detectors used
Hough Transform — Thrust	Fast but inflexible	No satisfying multipeak finder	MVD STT
Hough Transform — Plain CUDA *(with 1° of dynamic parallelism)*	Fast		MVD
GPU Riemann Track Finder	Promising initial implementation	Testing, Combinatorics	MVD GEM
GPU Triplet Finder	Fast, robust, scalable	Manpower	STT
GPU STT Cell Finder	→ see next talk by Tobias		

Thank you!

Andreas Herten
a.herten@fz-juelich.de

APPENDIX

GTLI (General Topics to Look Into)

- Bunching
- Time-based structure
- Track Merger
- Which algorithm for which sub-detector
- Algorithm interplay / hybridization
- Specifically test of sub-parts of the algorithm run faster on CPUs *(CPU-GPU re-distribution)*
- Quality analytics

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

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