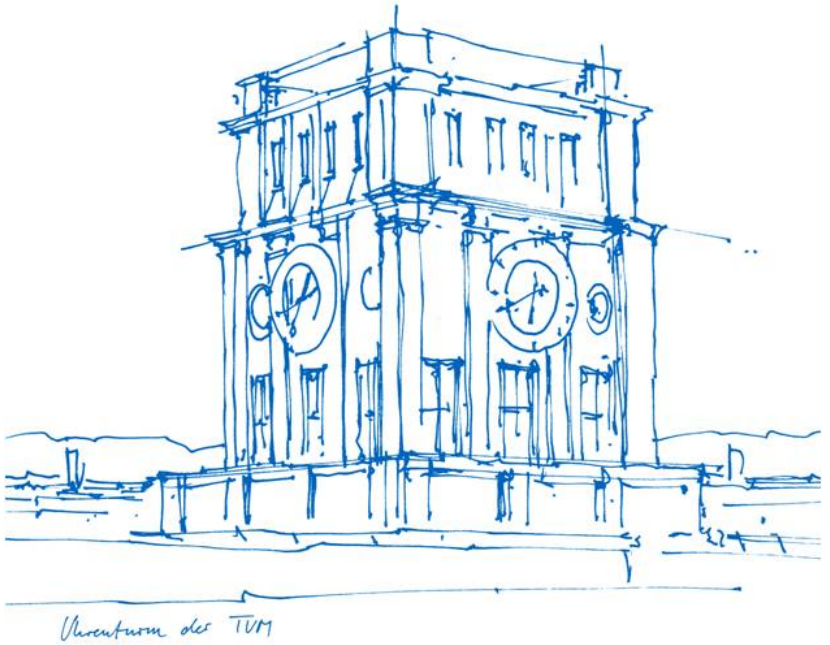
MAPS

MEDICAL
AUTONOMY AND
PRECISION SURGERY

Utilising Potential Fields for Trajectory Planning in Medical Robotics

Vincent Clifford

Clinical Application Project Final Presentation



Wednesday, 5. June 2024

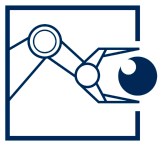
Outline

- Motivation
- Modeling the Physical Forces
- Path Planning based upon Physical Forces
- Performance Increase & Real-time Processing Capabilities
- Examples
- Questions

Motivation

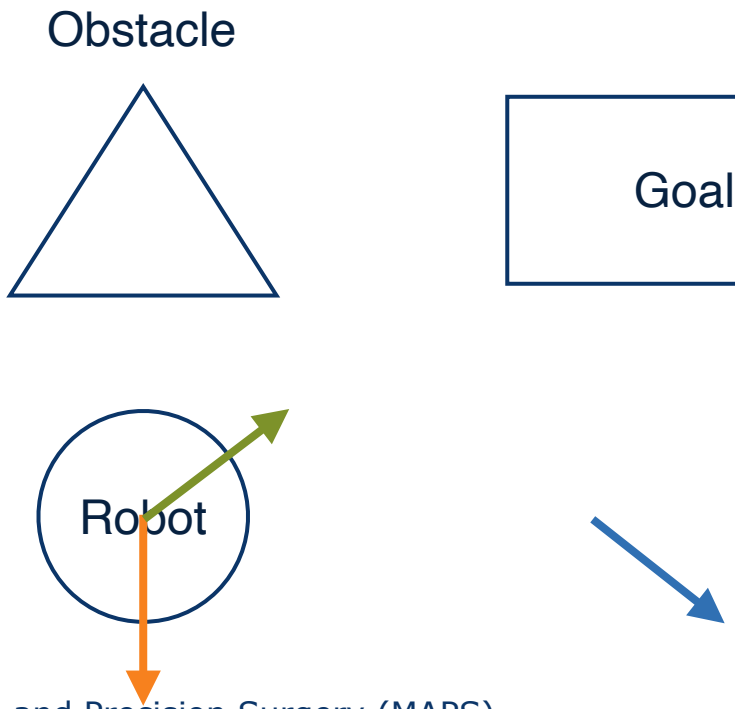
The problem

Guide the robot to the patient in the operating room



Key Idea

- Each obstacle emits a repulsive force on the robot
- The goal emits an attraction force on the robot
- The robot will follow the sum of force vectors



Benefits

- Simplicity and Intuitiveness
- Real-time Performance
- Smooth and Continuous Paths/Trajectories
- Coordination in Multi-Agent Systems
- Scalability

Modelling the Physical Forces

Wednesday, 5. June 2024

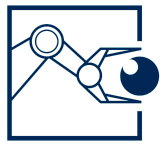
Modelling single repulsive forces

- Inspired by Wu et. al. [1] in a paper
- Their proposal:

$$\overrightarrow{F_{repi_x}(q)} = \begin{cases} \eta\left(\frac{1}{\rho_i(q, q_{obst})} - \frac{1}{\rho_0}\right)\frac{1}{\rho_i^2(q, q_{obst})}\frac{x - x_{obst}}{\rho_i(x, x_{obst})}, & \rho_i(q, q_{obst}) \le \rho_0 \\ 0, & \rho_i(q, q_{obst}) > \rho_0 \end{cases} \tag{8}$$

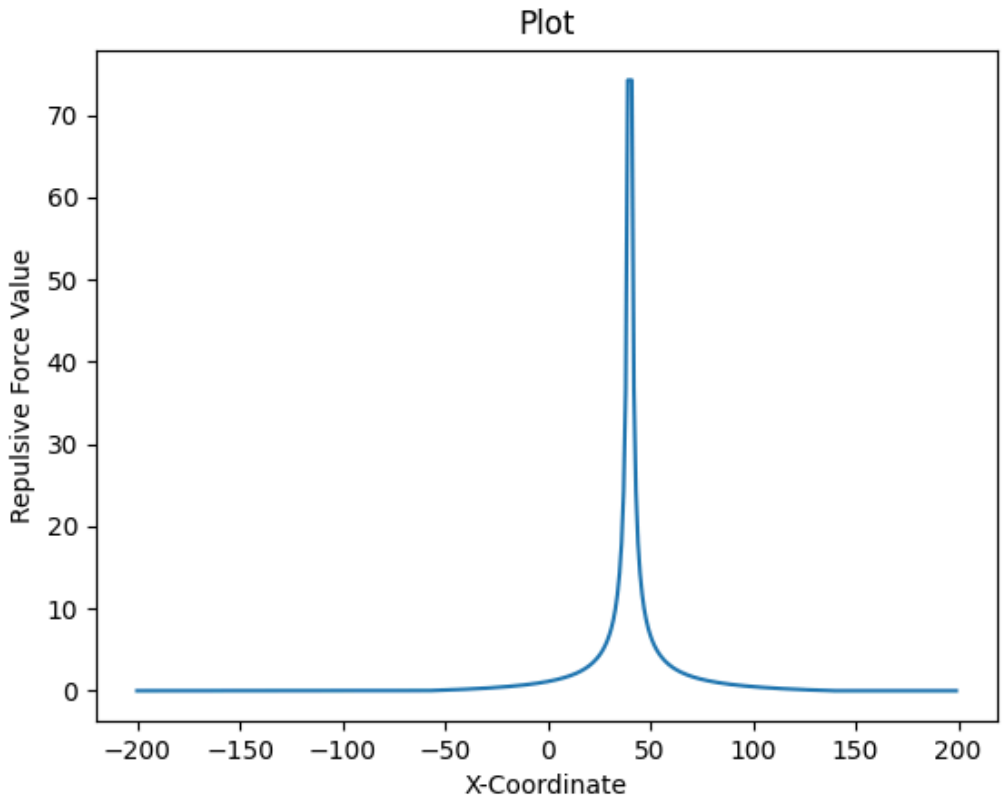
$$\overrightarrow{F_{repi_y}(q)} = \begin{cases} \eta\left(\frac{1}{\rho_i(q, q_{obst})} - \frac{1}{\rho_0}\right)\frac{1}{\rho_i^2(q, q_{obst})}\frac{y - y_{obst}}{\rho_i(y, y_{obst})}, & \rho_i(q, q_{obst}) \le \rho_0 \\ 0, & \rho_i(q, q_{obst}) > \rho_0 \end{cases}. \tag{9}$$

- Takeaway: Function based on inverse-square law and uses a case distinction

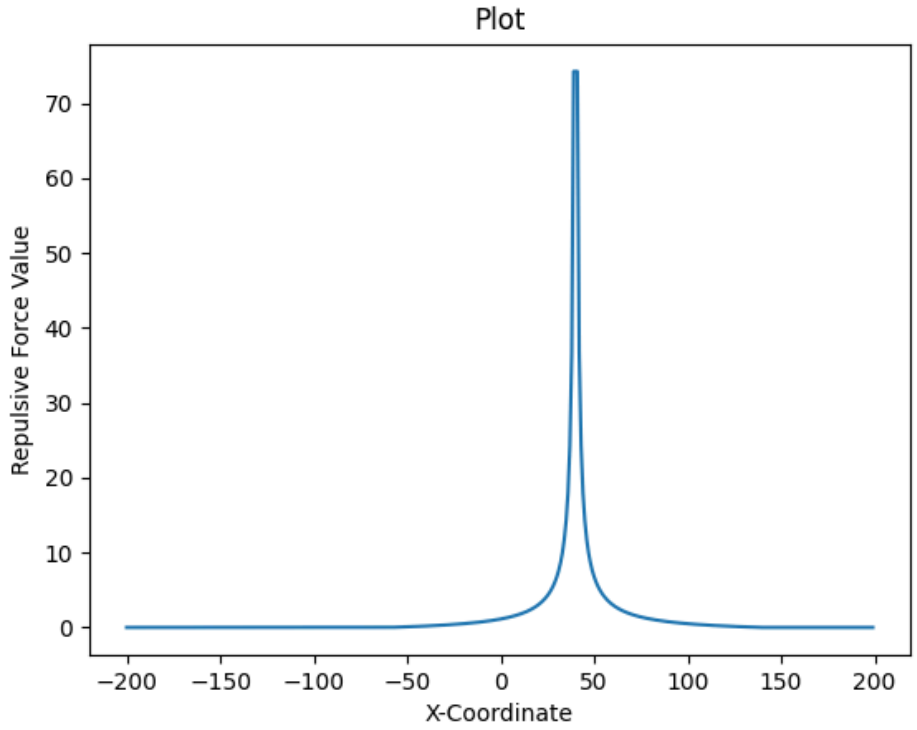


Example repulsive function

- Example repulsive 1D repulsive force function for environment with obstacles at x = 40



Example repulsive function

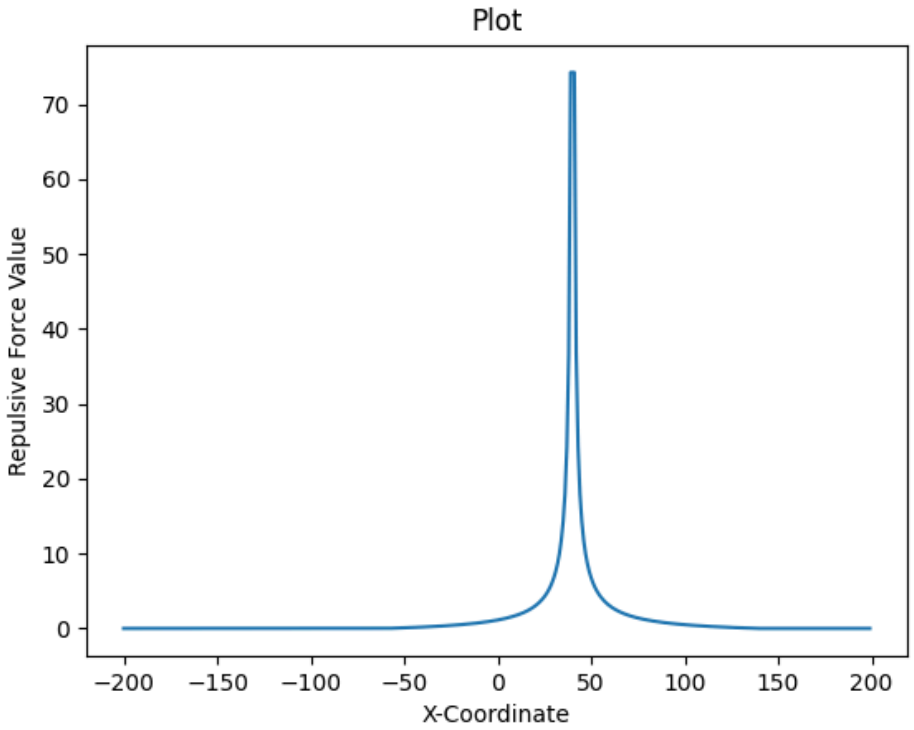


Problem:

- Very narrow spikes lead the robot to realise an obstacle extremely late

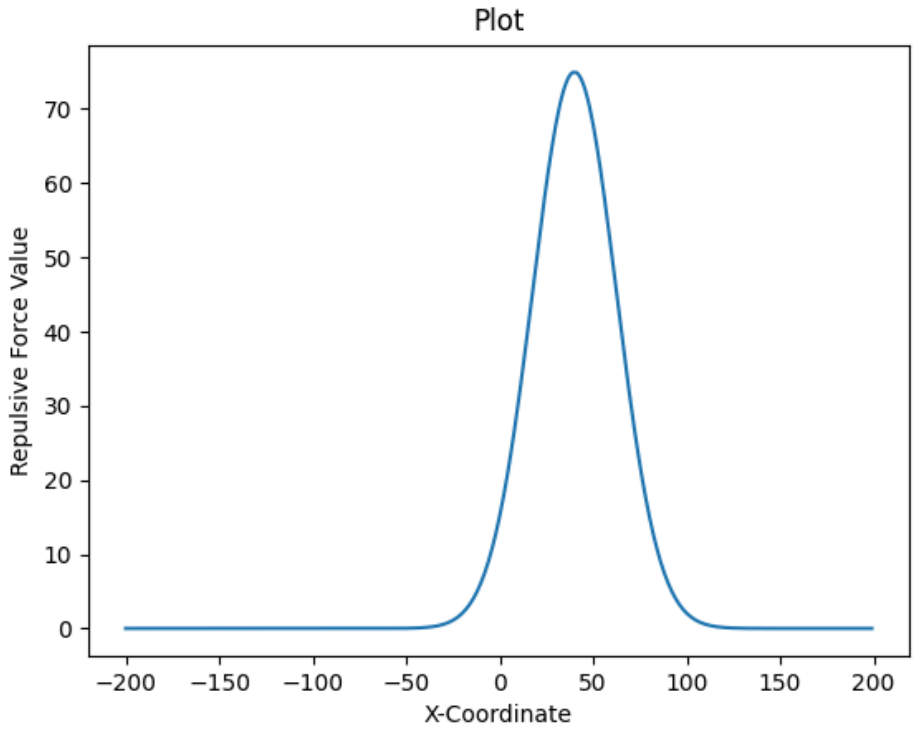
Our contributions

Wu et. al.’s proposed function:



$F_{repulsive}(x) = c/distance(x, x_{obstacle})^2$

Our proposed function:



$F_{repulsive}(x) = c' * e^{(-c * distance(x, x_{obstacle}))}$

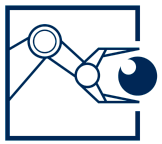
Key differences

- Function based upon inverse-square law vs “normal distribution” like function

- Function dependent on two constants: c’ and c’’
  ⇨ Allows more flexibility and adaptability

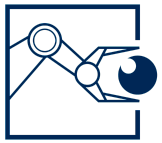
- No case distinction is needed for our function
  ⇨ Allows for future parallelisation of the algorithm

- Robot “notices” the obstacle earlier



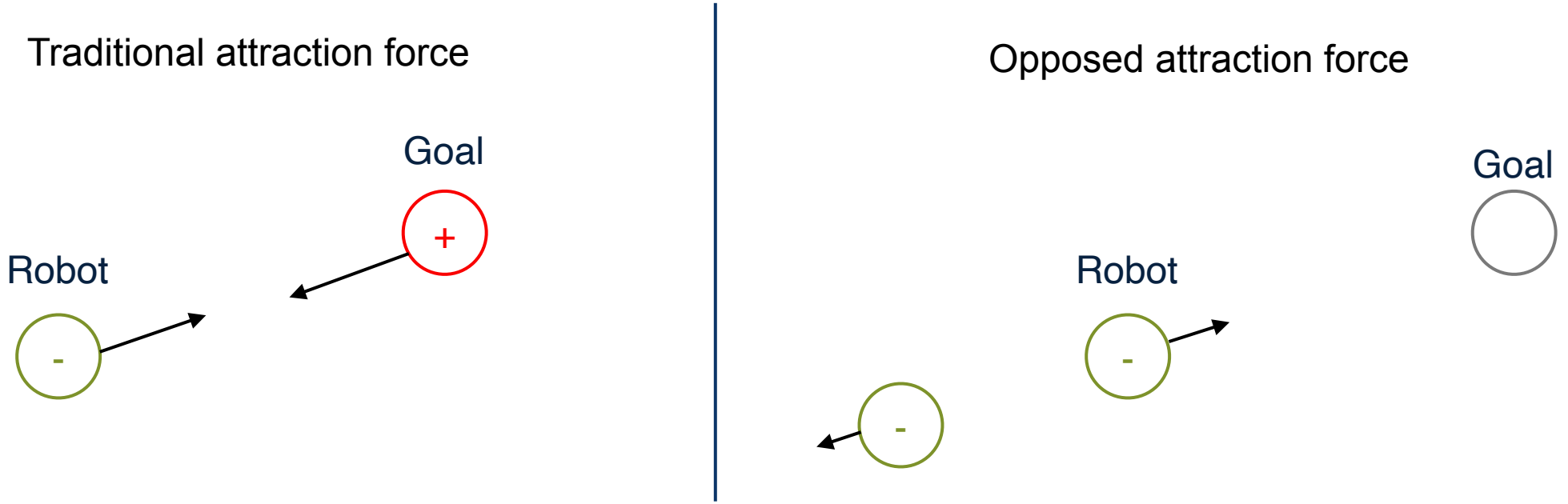
Modelling total repulsive force

$$F_{\text{TotalRepulsive}}\left(x_{\text{robot}}, y_{\text{robot}}\right) = \sum_{i=1}^{n} F_{\text{SingleRepulsive, i}}\left(x_{\text{robot}}, y_{\text{robot}}\right) \quad (3)$$



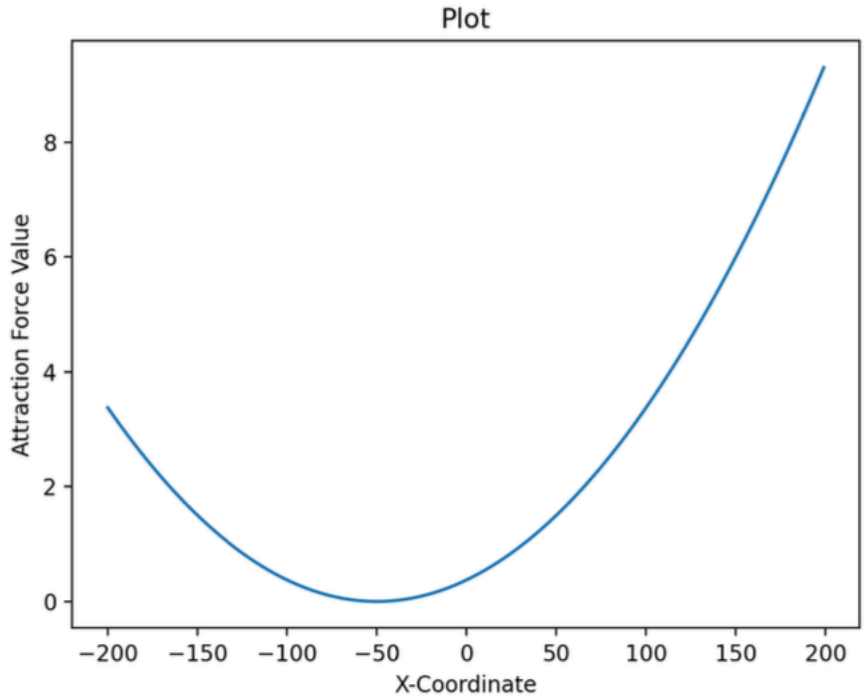
Modelling attraction force

We model the attraction force as a repulsive force that pushes the robot to the goal



Example attraction function

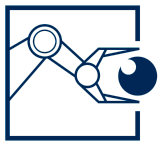
Example force value plot for a 1D environment with the target/ goal located at at x = -50



Modelling attraction force

We propose to model the attraction force function as follows:

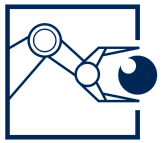
$$F_{\text{Attraction}}(x_{\text{robot}}, y_{\text{robot}}) = c' * ((x_{robot} - x_{obstacle})^2 + (y_{robot} - y_{obstacle})^2) \quad (4)$$



Total force value

Now, the total force value can be expressed by:

$$F_{\text{Total}}(x_{\text{robot}}, y_{\text{robot}}) = F_{Attraction}(x_{robot}, y_{robot}) + F_{TotalRepulsive}(x_{robot}, y_{robot}) \quad (5)$$



My documentation includes more formal mathematics

trajectories.

3.1 Repulsive Force

One challenge arose from the rate at which the repulsive force from obstacles decreased with distance. In Wu et al.'s proposal, the rapid repulsive force falloff potentially caused the robot to "realize" an impending collision too late for safe course correction. This resulted in jerky movements as the robot attempted abrupt adjustments to avoid obstacles within the confined and delicate environment of the operating room.

Hence, a new repulsive force function had to be defined. We came up with a new similar function that mitigates this issue. In our trajectory planning algorithm, we use the following function to calculate the repulsive force for a single obstacle, with index i, at position $(x_{\text{obstacle}}, y_{\text{obstacle}})$:

$$F_{\text{singleRepulsive, i}}(x_{\text{robot}}, y_{\text{robot}}) = c' * e^{-c'' * \sqrt{(x_{\text{robot}} - x_{\text{obstacle}})^2 - (y_{\text{robot}} - y_{\text{obstacle}})^2}} \quad (1)$$

Parameters of an obstacle:

- $c' > 0$: Attraction strength of obstacle. The bigger this constant, the bigger the magnitude of the repulsive force.
- $c'' > 0$: Distance of influence of obstacle. The higher this constant is, the faster the drop-off of the repulsive force.

A key distinction between our proposed function and the one defined by Wu et al. lies in how it handles the repulsive force based on the distance to the obstacle. Unlike Wu et al.'s approach, our function eliminates the need for case-based calculations depending on the robot-obstacle distance. This simplification unlocks the power of mathematical objects called *tensors*, enabling parallel computation of the potential field value. To achieve real-time planning in the dynamic environment of the operating room, we prioritize parallelization of the distance calculation algorithm. This optimization is crucial for meeting the performance requirements of this application. We will discuss the further performance gains offered by utilizing tensors for computation in Section 6. Here, we first establish the core functionality of the algorithm before delving into real-time optimization strategies.

Wu et al. makes a case distinction under the following condition and returns the repulsive force value of 0 iff.

$$\sqrt{(x_{\text{robot}} - x_{\text{obstacle}})^2 - (y_{\text{robot}} - y_{\text{obstacle}})^2} > \textit{distance of influence} \quad (2)$$

Recall, *distance of influence* is a parameter of each individual obstacle. By design, my proposed function, constructed without the need for case distinctions and relying on an exponential function, never returns a repulsive force value of precisely 0. However, in scenarios where Wu et al.'s function would yield 0, our offers a value infinitesimally close to 0. This infinitesimally small value is neglectable in path planning and enables parallel computation of the repulsive force value. Note that this property only holds for rather large *distances of influence*. This is sufficient because most obstacles in the operating room don't have small distances of influences.

Unlike the inverse-square law observed in physics and used by Wu et al., which leads to undesirable jerkiness in trajectory planning, our proposed repulsive function leverages an exponential distribution. This choice avoids the abrupt dropoff of a function following the inverse-square law while maintaining a smoother decay compared to a linear function. As a result, the exponential function balances responsiveness to nearby obstacles with computational efficiency for path planning.

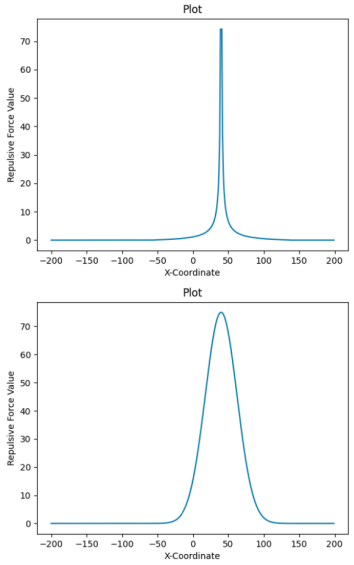


Figure 1: Comparison between Wu et al.'s and our function.

This can be seen best by an example in Figure 1. Consider the one-dimensional environment depicted in Figure 1, where the only obstacle is at $x = 40$. This figure compares two repulsive value functions used for robot navigation. The upper image shows the function proposed by Wu et al. Here, the repulsive force value is highest directly at the obstacle ($x = 40$) and rapidly decreases as the agent moves away. In contrast, the lower image depicts our proposed function. While it also assigns the maximum force at the obstacle, the repulsive effect extends over a greater distance. This smoother falloff helps trajectory planning algorithms generate smoother, less jerky paths to navigate around the obstacle.

Within the operating room environment, numerous obstacles are present. We can sum each i-th object's individual repulsive force, $F_{\text{SingleRepulsive, i}}$, to efficiently compute the entire repulsive force, $F_{\text{TotalRepulsive}}$, at a specific location. Assuming we have got n objects, the total repulsive force can be expressed mathematically as:

$$F_{\text{TotalRepulsive}}(x_{\text{robot}}, y_{\text{robot}}) = \sum_{i=1}^{n} F_{\text{SingleRepulsive, i}}(x_{\text{robot}}, y_{\text{robot}}) \quad (3)$$

3.2 Attraction Force

Wu et al. propose a unique solution to model the attraction force. They essentially flip the attraction force and model it using a repulsive function.

This can be explained best with an analogy between a magnet and the robot: Imagine a magnet at the target and one on the robot with flipped poles. In this setting, the magnet would pull the robot towards the goal. This is how an attraction force is typically thought of. Now, let's draw a line between the robot and the target. Imagine placing a new magnet directly on this line, a bit behind the robot. Fictively, the magnet is "chasing" the robot to the goal. The imaginary magnet has the same pole orientation as the one on the robot. Due to the same poles of the robot and the robot behind the magnet, the robot is pushed to the goal, resulting in the same movement of the robot.

Wu et al. model the attraction force as a force that pushes the robot toward the goal, as described above. An example of this can be seen in Fig. 2, a one-dimensional environment with only a target at $x = -50$.

Our project uses the core concept of the attraction force function proposed by Wu et al. However, we've introduced a simplification to enhance efficiency. In the original function, a distinction was made in the calculations based on the robot's distance to the goal. While this can provide more nuanced results, we found that the impact on the overall trajectory path was minimal. By removing this case distinction, we can streamline the calculation process. This allows us to compute the attraction force value for any location much faster, making our implementation more efficient without significantly compromising the outcome. In this project, the following function was used for calculating the attraction force of a robot given a certain position. c' being the attraction strength of the target.

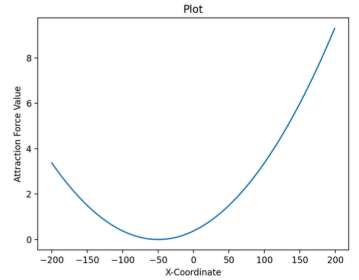


Figure 2: Attraction force value plot for a 1D environment with the target located at $x = -50$.

$$F_{\text{Attraction}}(x_{\text{robot}}, y_{\text{robot}}) = c' * ((x_{robot} - x_{obstacle})^2 + (y_{robot} - y_{obstacle})^2) \quad (4)$$

Given that the attractive force is effectively negated and hereby transformed into a repulsive force, the total force acting on the robot at a specific location can be obtained by simply summing the repulsive and attractive force contributions. Mathematically, this is expressed as:

$$F_{\text{Total}}(x_{\text{robot}}, y_{\text{robot}}) = F_{Attraction}(x_{robot}, y_{robot}) + F_{TotalRepulsive}(x_{robot}, y_{robot}) \quad (5)$$

This summation is visualized in Figure 3, which depicts the total force acting on the robot as the combined effect of the previously defined attractive force (Figure 1) and repulsive force (Figure 2).

4 Distance Calculations

In path planning simulations, calculating distances between the robot and surrounding elements is crucial for both attractive and repulsive force functions. While we typically model the robot as a circle for simplicity, real-world operating rooms contain obstacles with various shapes. To account for this complexity, we model obstacles as either circles or polygons.

https://github.com/vinceclifford/CAP/blob/cap_submission/report/documentation.pdf

Path Planning on Physical Forces

Wednesday, 5. June 2024

Summary (so far)

- Constructed function f: $R^2 \longrightarrow R$
- Input of function are the coordinates of the robot
- Output is the total force value acting on robot
- Finding goal is equivalent to finding global minimum of function

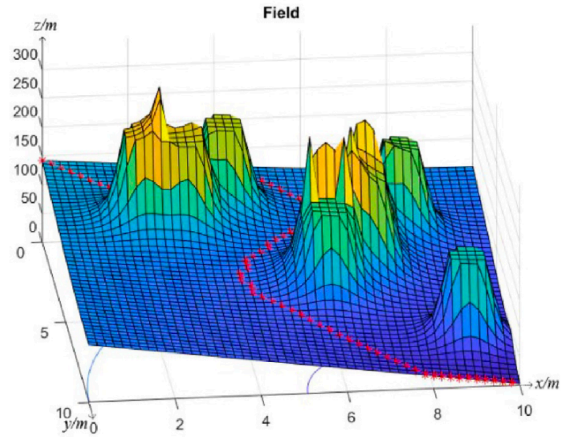
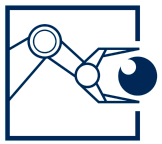


Fig. 2. Path planning in APF.

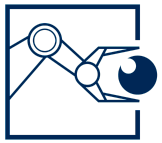
The question

How do we navigate this function/this world?



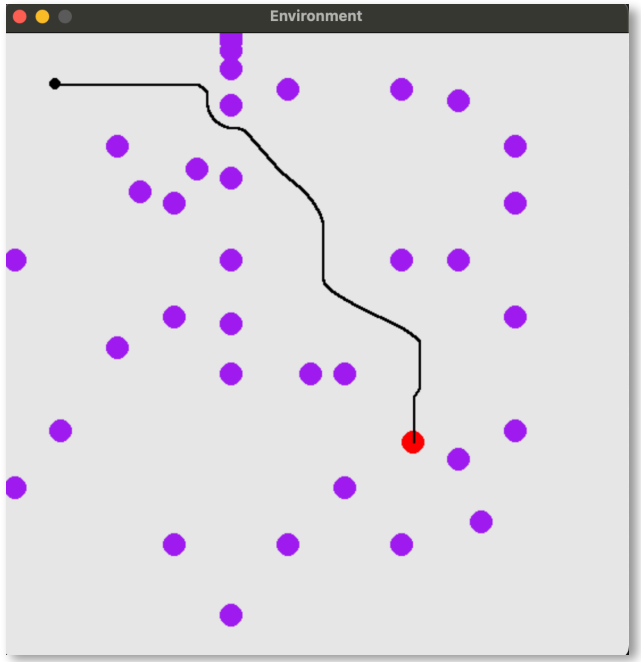
The answer

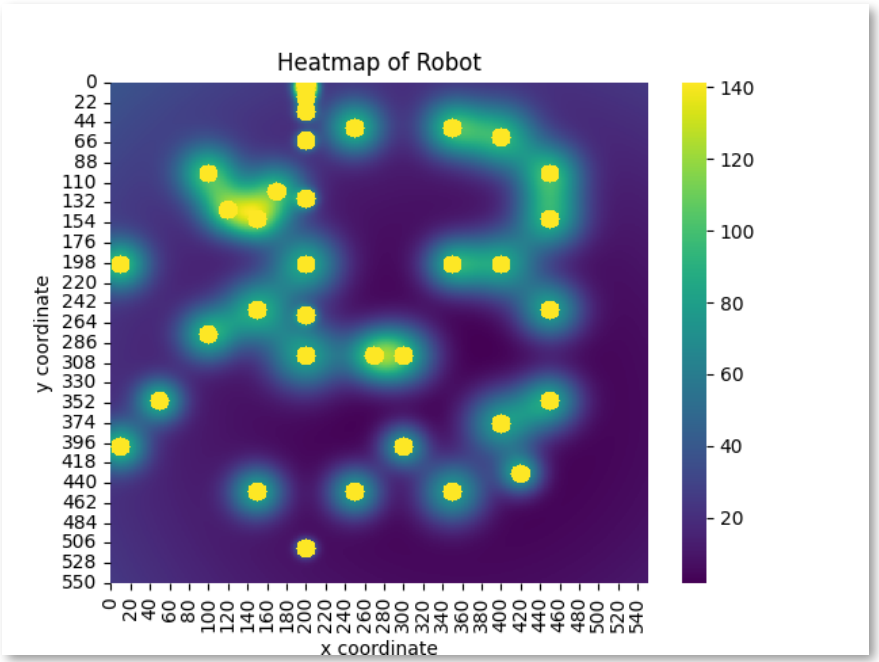
We minimise the total accumulated repulsive force values along the entire path



Results

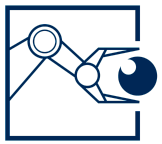
- There are many already existing algorithms for this minimisation - we use Dijkstra for simplicity





Two remaining problems

1. How do we guarantee safety
2. Slow runtime/performance

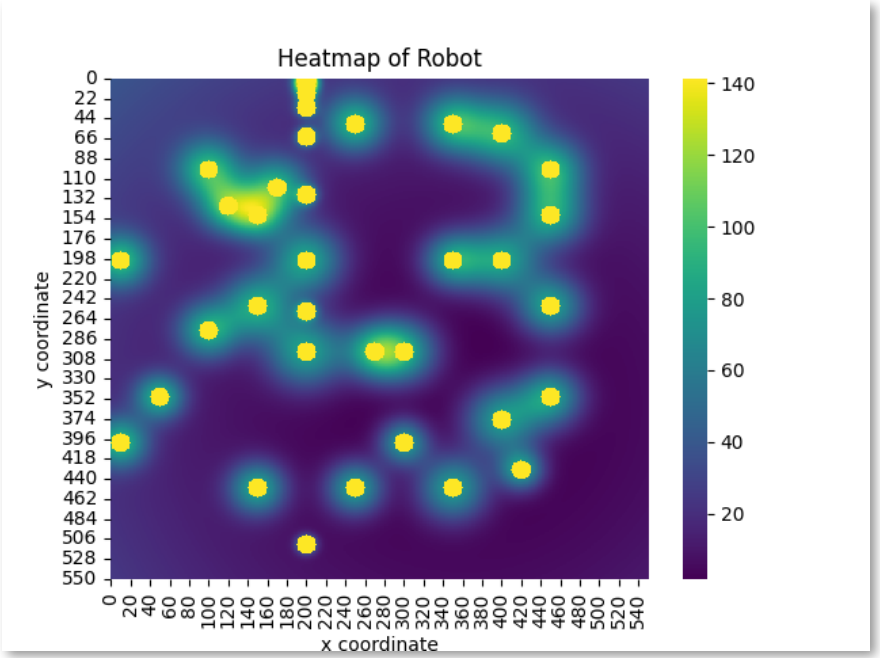


Performance Increase & Real-time Processing Capabilities

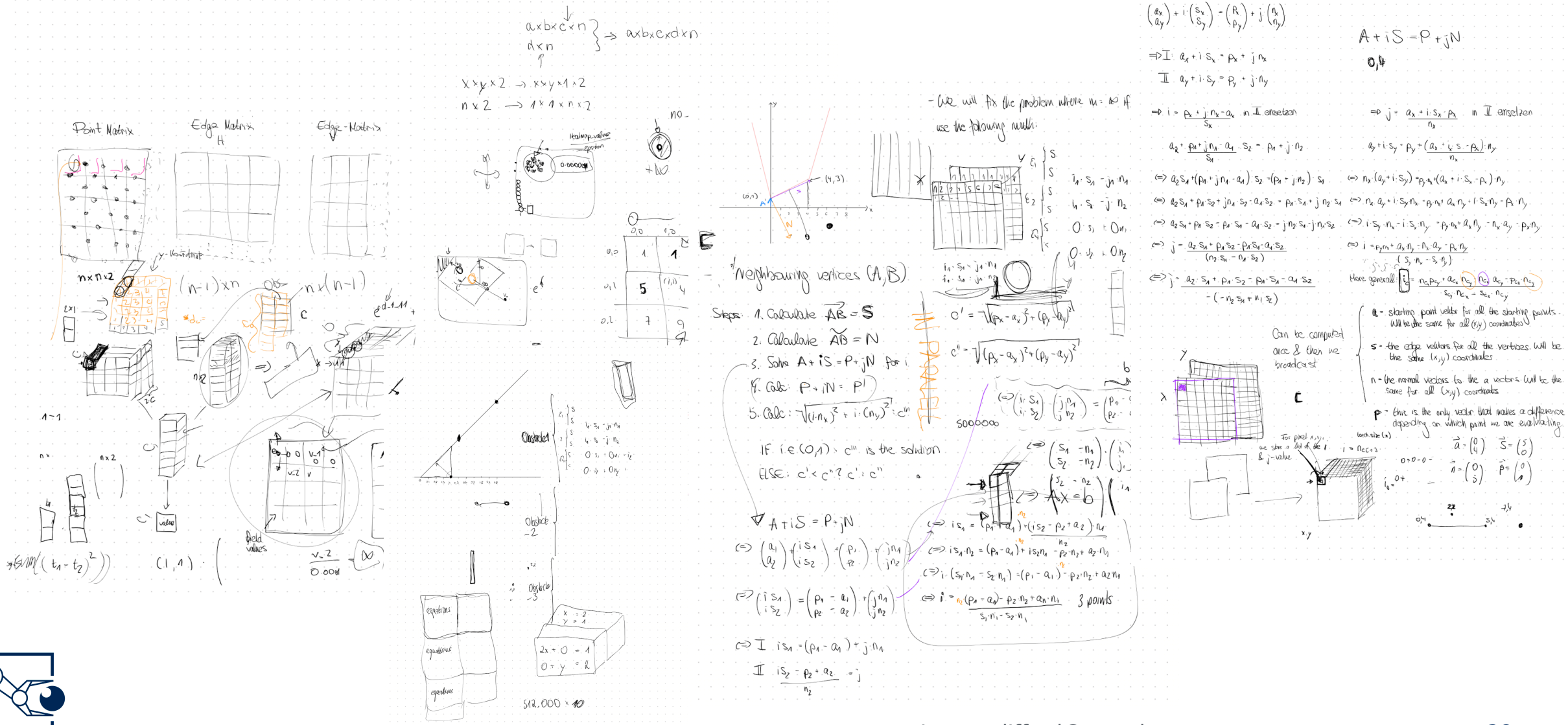
Wednesday, 5. June 2024

Guaranteeing Safety

Concept: Assign an infinite repulsive force to specific coordinates to ensure collision avoidance.



Improving Performance

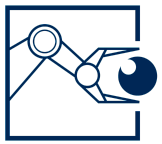


Improving Performance

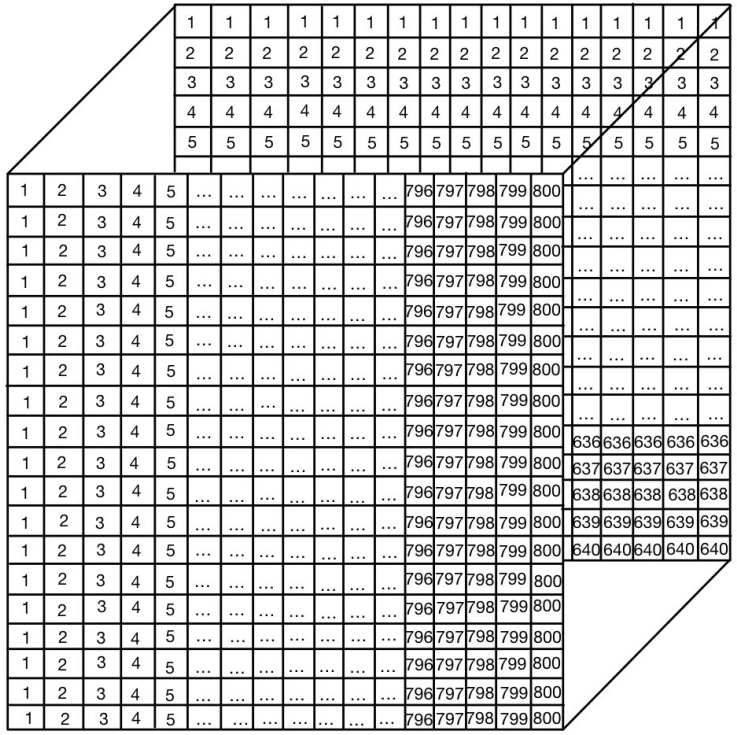
- Compute potential field values in parallel rather than serially
- Multithreading?

⇨ Overhead & Python's GIL (Global Interpreter Lock)

Solution: Express all math equations as tensor operation and let libraries take care of optimisations.



Tensor Math for Repulsive Force of Circles



“Base tensor”

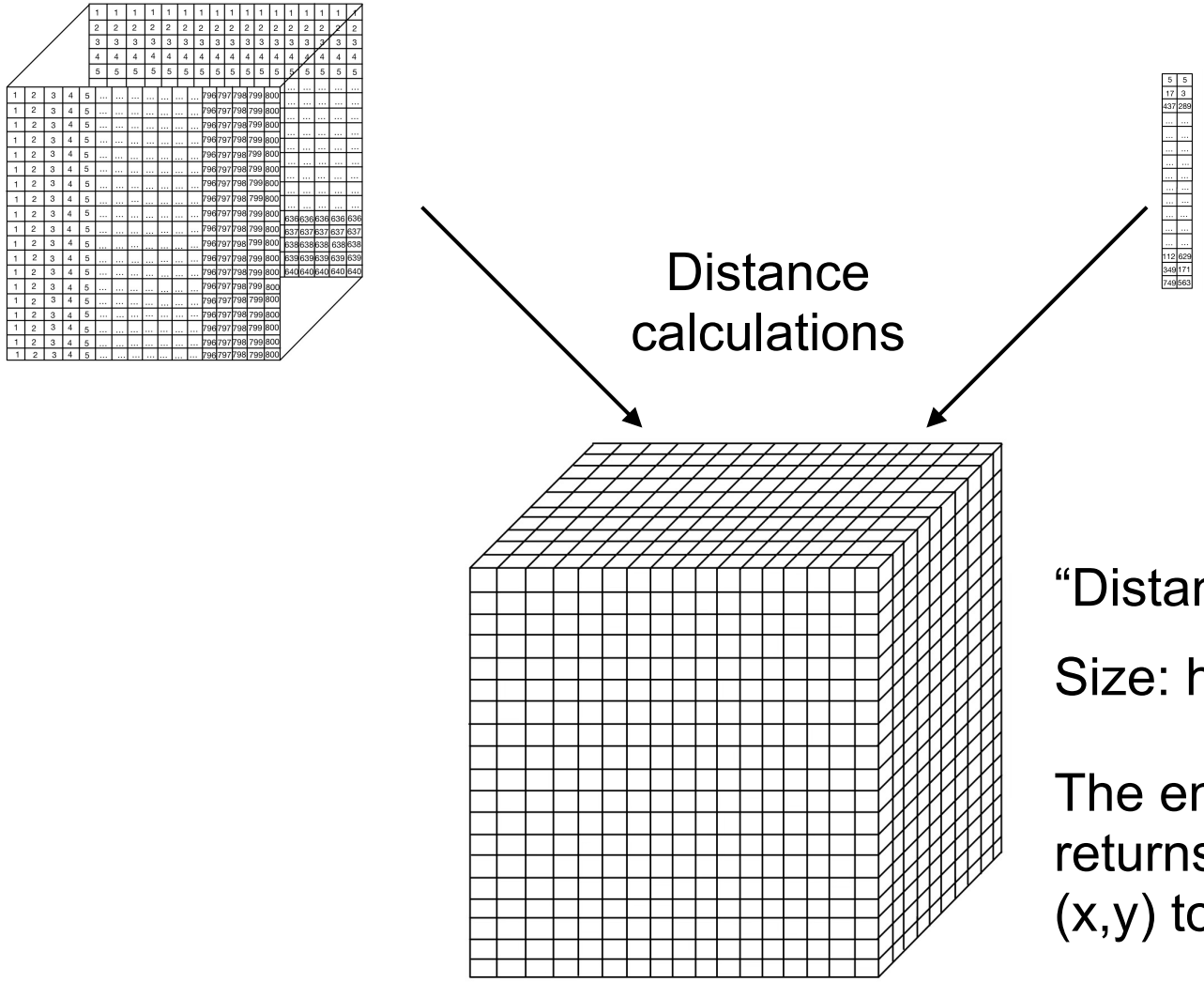
Size: height x width x 2

5	5
17	3
437	289
…	…
…	…
…	…
…	…
…	…
…	…
…	…
…	…
…	…
…	…
…	…
112	629
349	171
749	563

“Obstacle position tensor”

Size: #obstacles x 2

Tensor Math for Repulsive Force of Circles



"Distance tensor"

Size: height x width x #obstacles

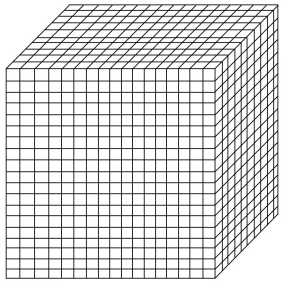
The entry distance_tensor[y, x, i] returns the distance from position (x,y) to the i-th obstacle

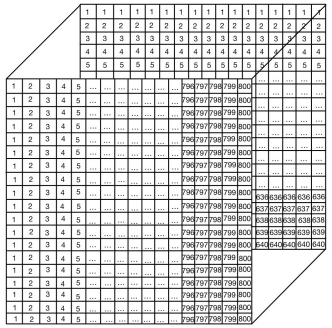
Tensor Math for Repulsive Force of Circles

“Obstacle distance of influence tensor”
Size: #obstacles x 2

“Obstacle attraction tensor”
Size: #obstacles x 2

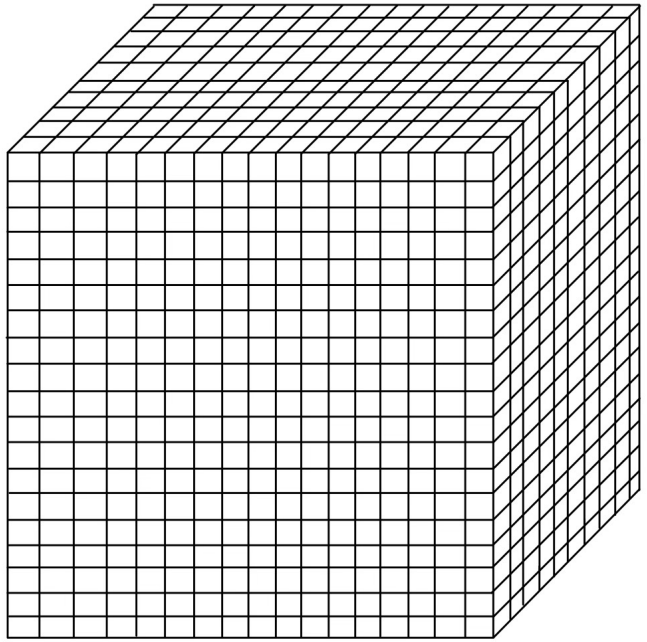
Distance calculations





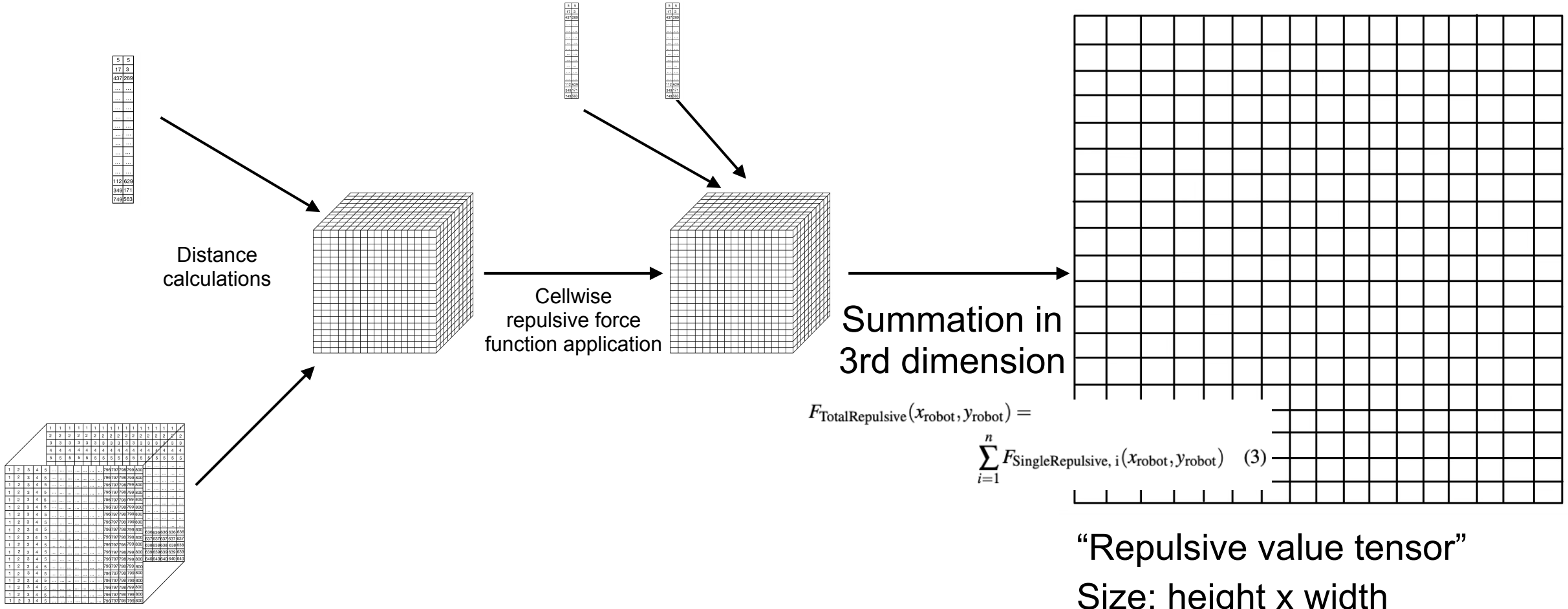
Cellwise repulsive force function application

$$F_{\text{singleRepulsive, i}}(x_{\text{robot}}, y_{\text{robot}}) = c' * e^{-c'' * \sqrt{(x_{\text{robot}} - x_{\text{obstacle}})^2 - (y_{\text{robot}} - y_{\text{obstacle}})^2}} \quad (1)$$



“Single repulsive tensor”
Size:
height x width x #obstacles

Tensor Math for Repulsive Force of Circles

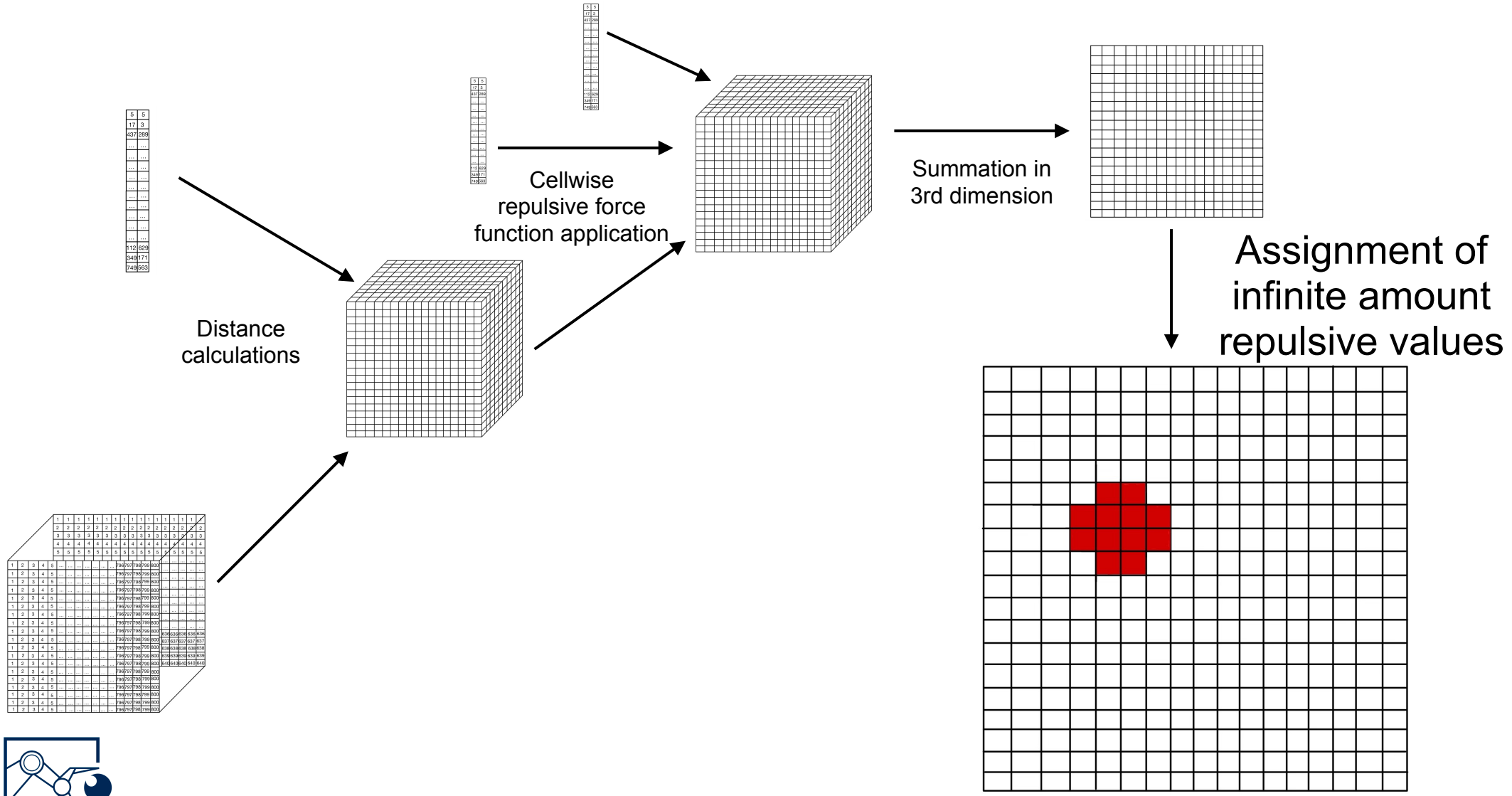


$$F_{\text{TotalRepulsive}}(x_{\text{robot}}, y_{\text{robot}}) = \sum_{i=1}^{n} F_{\text{SingleRepulsive, i}}(x_{\text{robot}}, y_{\text{robot}}) \quad (3)$$

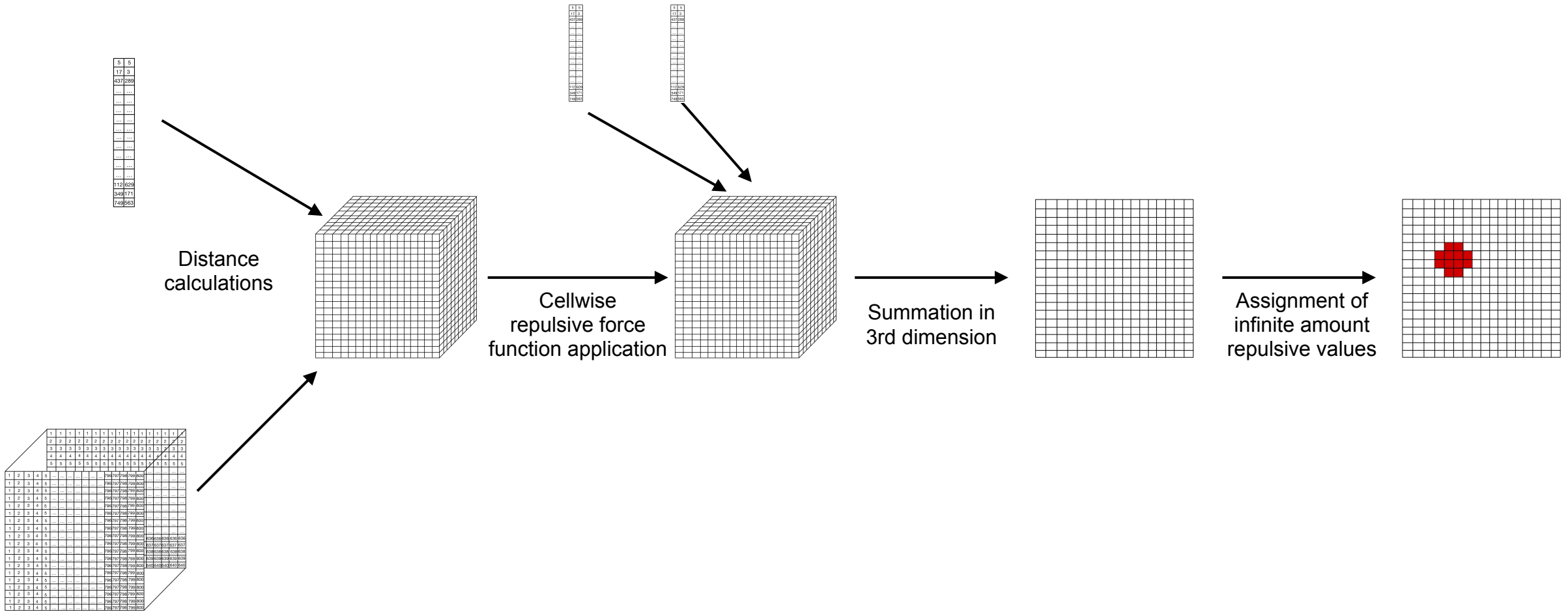
"Repulsive value tensor"

Size: height x width

Tensor Math for Repulsive Force of Circles



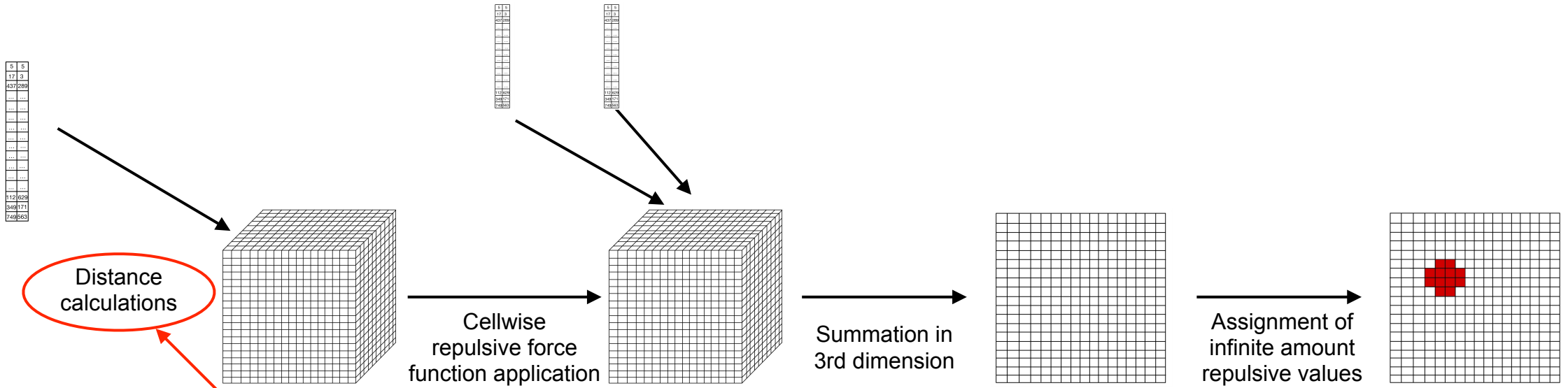
Tensor Math for Repulsive Force of Circles

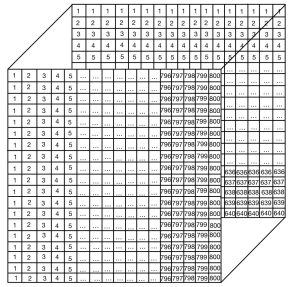


Distance Calculations for Polygons

Wednesday, 5. June 2024

Distance Calculations for Polygons





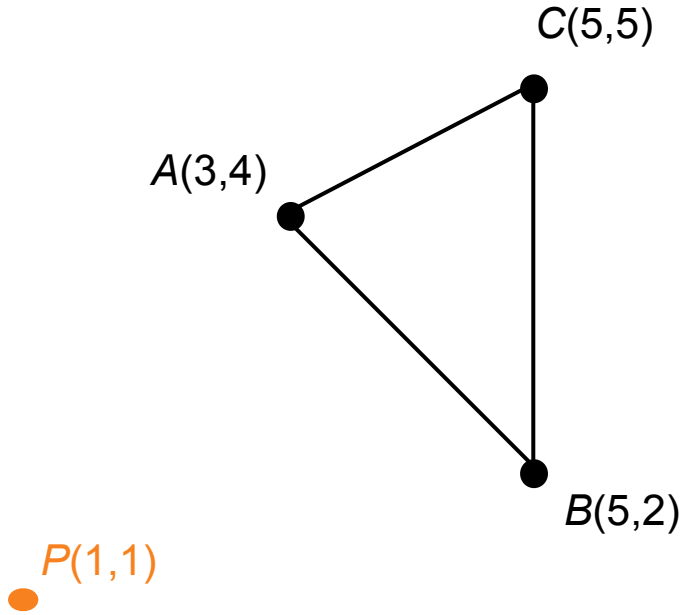
Fairly easy for circle obstacles with euclidean norm:

$$d(x_r, y_r, x_o, y_o,) \quad = \quad \sqrt{(x_r - x_o)^2 + (y_r - y_o)^2} \quad (6)$$

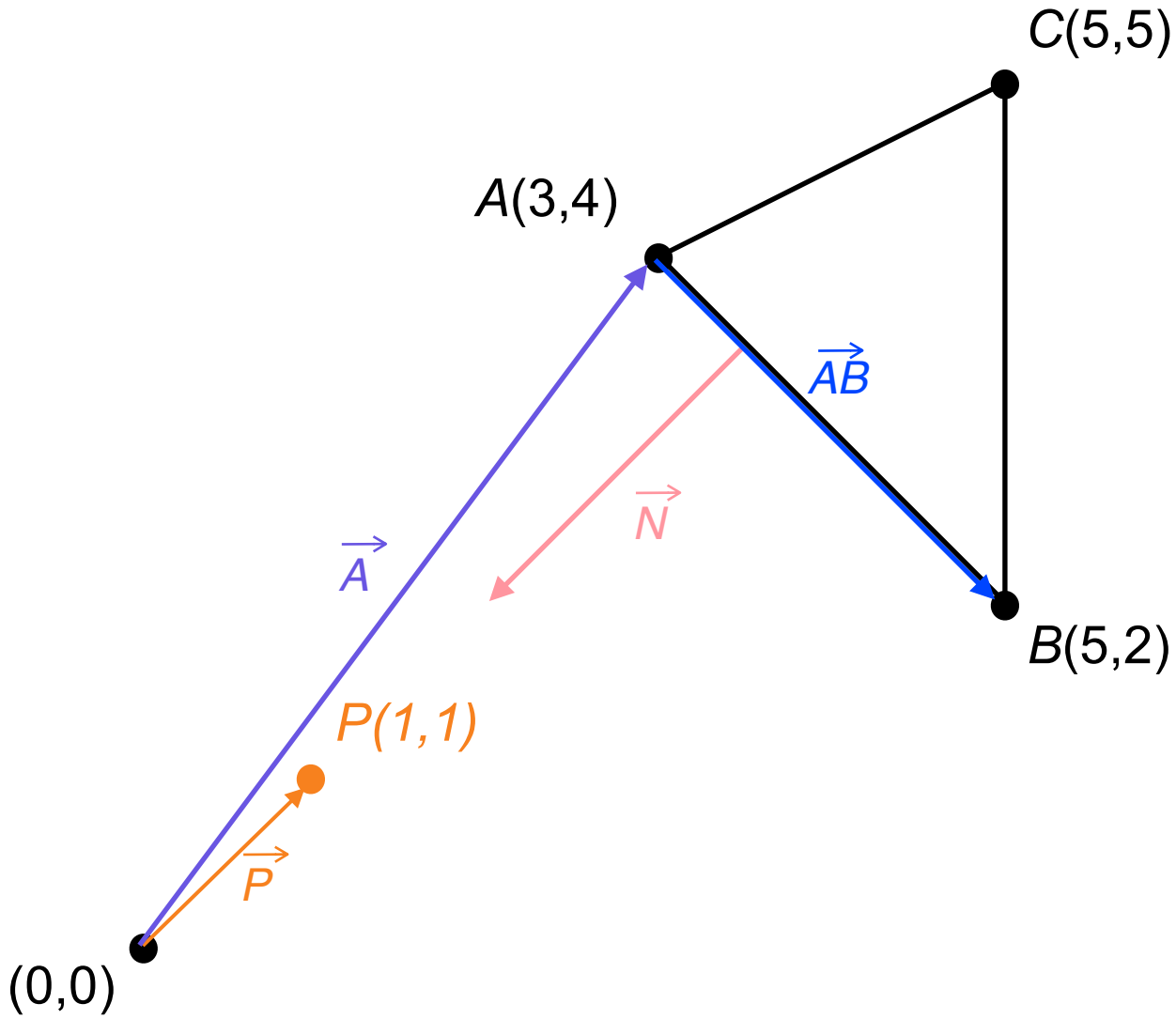
Distance Calculations for Polygons

Prevailing question: How de we compute the distance of a point (x,y) to an obstacle?

Solution: Determine the distance to all edges and select the shortest distance.



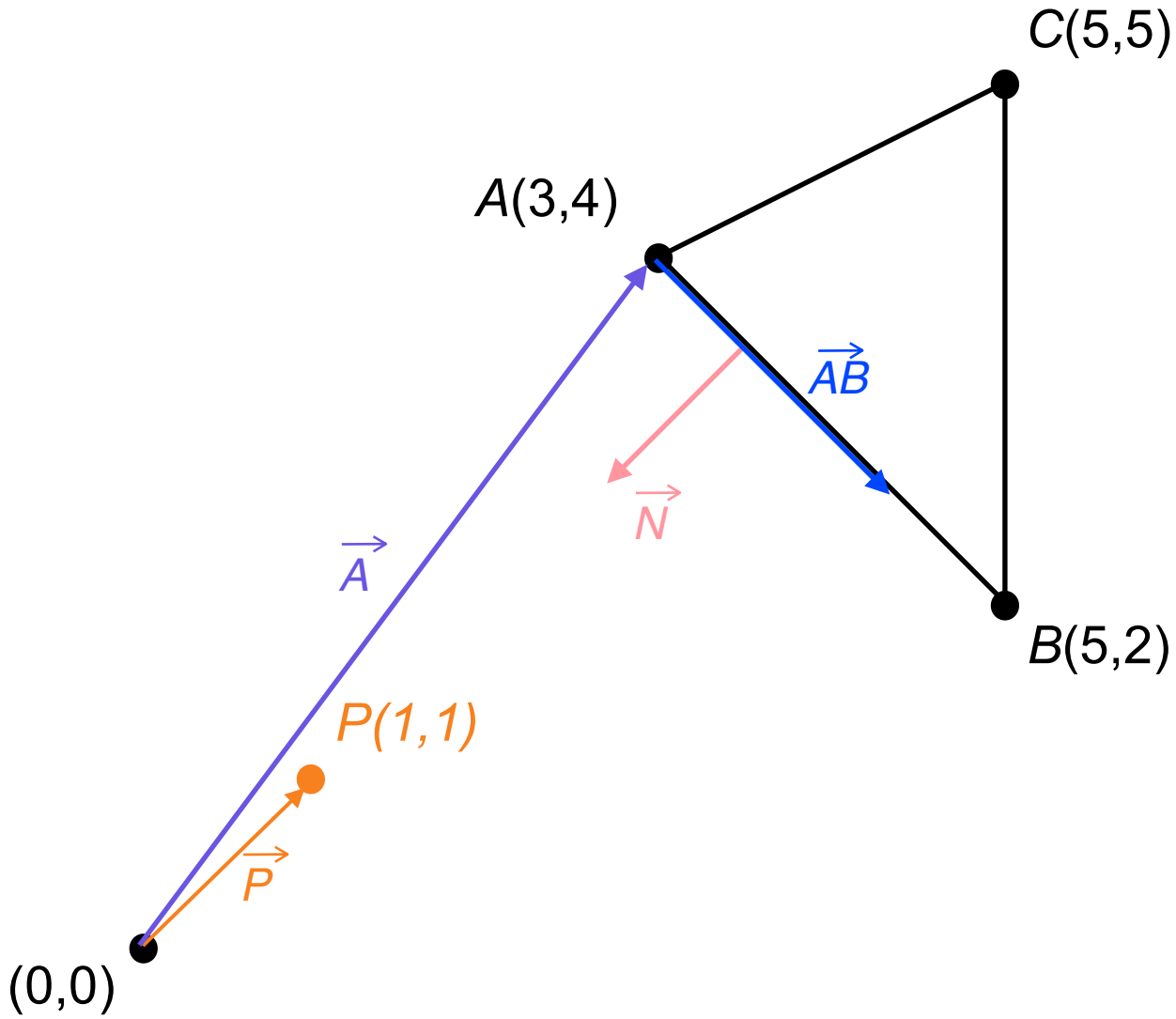
Distance Calculations for Polygons



Solve:

$$\overrightarrow{A} + i \cdot \overrightarrow{AB} = \overrightarrow{P} + j \cdot \overrightarrow{N}$$

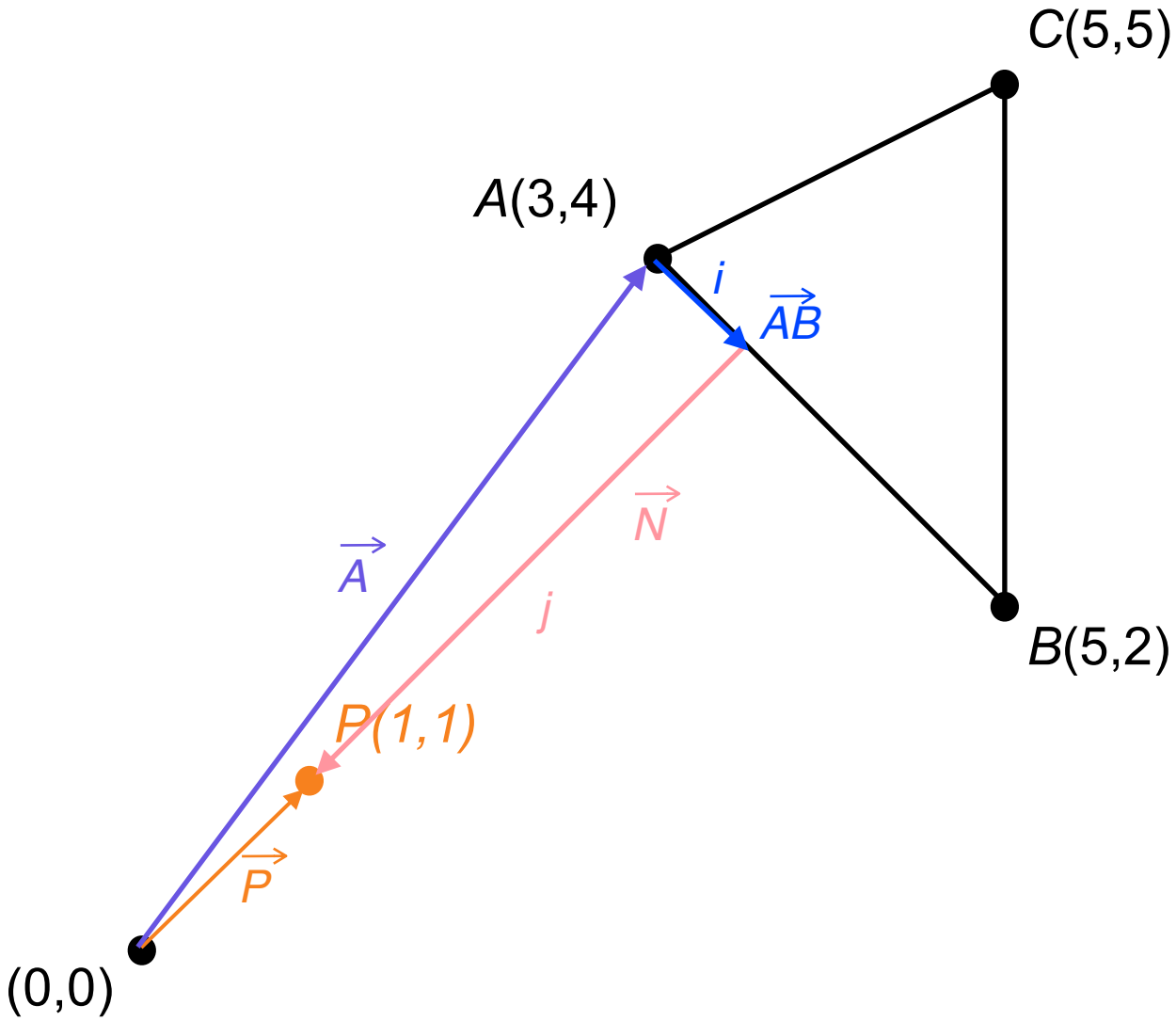
Distance Calculations for Polygons



Solve:

$$\overrightarrow{A} + i \cdot \overrightarrow{AB} = \overrightarrow{P} + j \cdot \overrightarrow{N}$$

Distance Calculations for Polygons



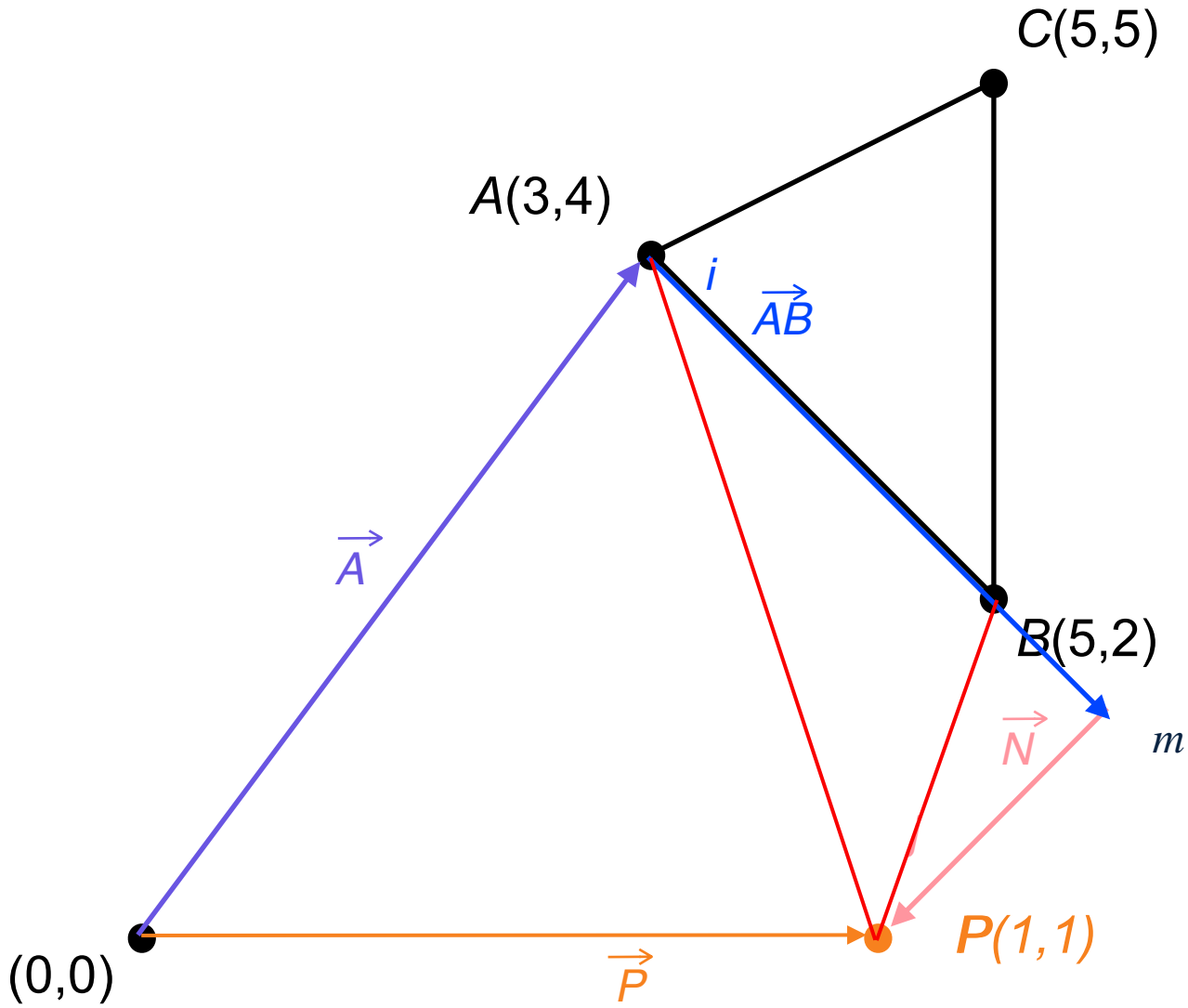
Solve:

$$\overrightarrow{A} + i \cdot \overrightarrow{AB} = \overrightarrow{P} + j \cdot \overrightarrow{N}$$

Then the distance is:

$$|j\overrightarrow{N}|$$

Distance Calculations for Polygons



Solve:

$$\overrightarrow{A} + i \cdot \overrightarrow{AB} = \overrightarrow{P} + j \cdot \overrightarrow{N}$$

Then the distance is:

$|j\overrightarrow{N}|$ *iff.* $i \in [0,1]$

else

$$min\{\sqrt{(p_x - a_x)^2 + (p_y - a_y)^2}, \sqrt{(p_x - b_x)^2 + (p_y - b_y)^2}\}$$

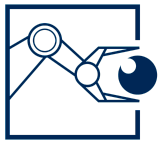
Distance Calculations for Polygons

$$\overrightarrow{A} + i \cdot \overrightarrow{AB} = \overrightarrow{P} + j \cdot \overrightarrow{N}$$

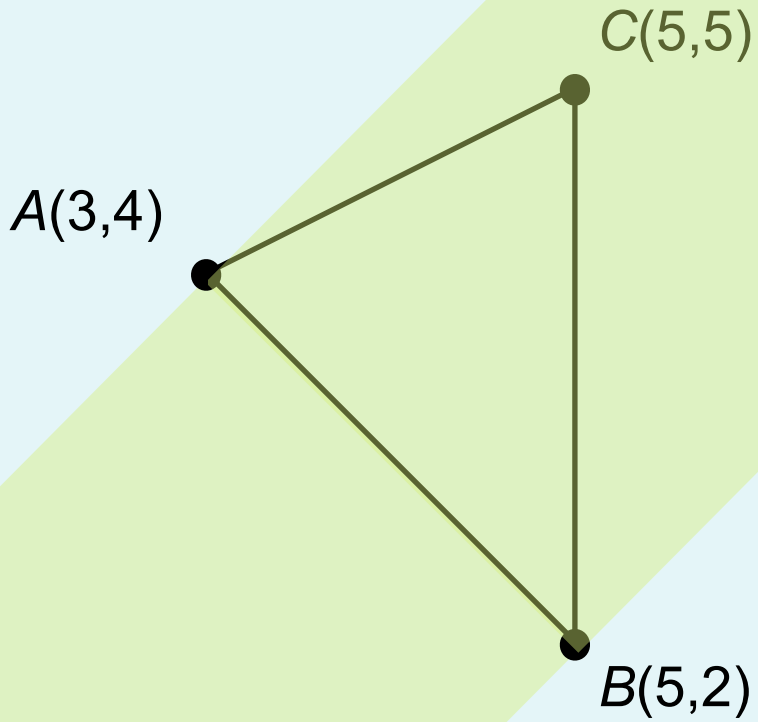
$$\Leftrightarrow \begin{bmatrix} a_x \\ a_y \end{bmatrix} + i \begin{bmatrix} ba_x \\ ba_y \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} + j \begin{bmatrix} n_x \\ n_y \end{bmatrix}$$

$$\Leftrightarrow i \begin{bmatrix} ba_x \\ ba_y \end{bmatrix} - j \begin{bmatrix} n_x \\ n_y \end{bmatrix} = \begin{bmatrix} p_x - a_x \\ p_y - a_x \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} ba_x - n_x \\ ba_y - n_y \end{bmatrix} * \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} p_x - a_x \\ p_y - a_x \end{bmatrix}$$



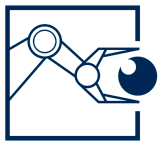
Distance Calculations for Polygons



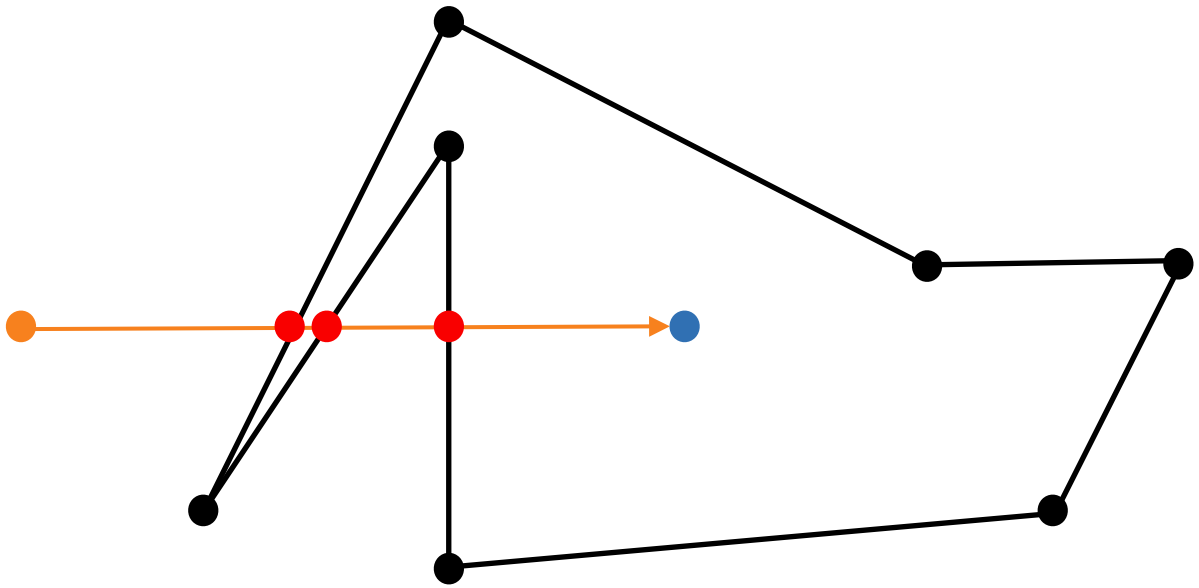
Last Distance Calculation of Polygon

Question: How can we determine whether a point (x,y) resides inside the polygon?

Solution: Ray-casting algorithm.



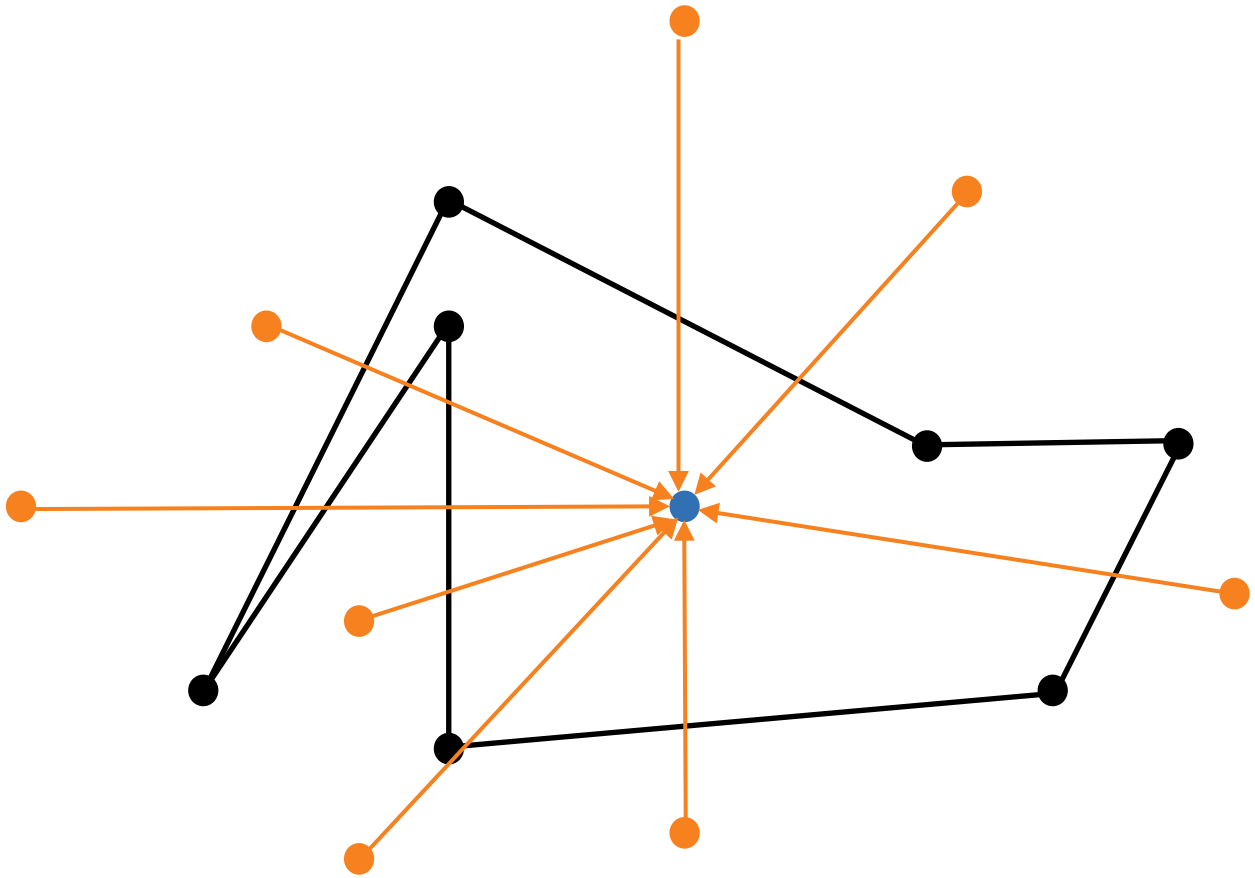
Ray Casting Algorithm



If number of intersections points is even ⇨ outside.

If number of intersections points is odd ⇨ inside.

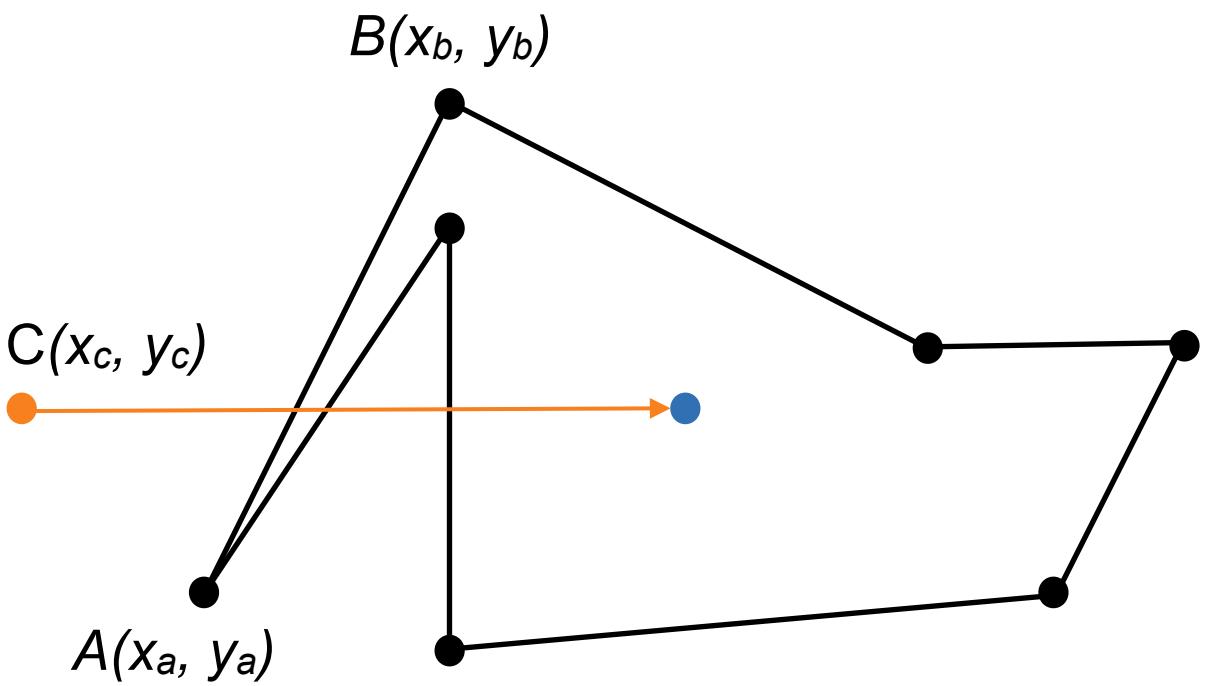
Choose your Casting Point wisely



Any point can be chosen in theory.

Some are computationally better than others.

Ray Casting Algorithm



Choose any point with the same x-coordinate as the point in question that is outside of the polygon.

Then, the casted ray intersects an edge bounded by the vertices A & B if the following conditions are met:

- $y_c <= max\{y_a, y_b\}$
- $y_c >= min\{y_a, y_b\}$
- $x_{start} < x_a + \frac{y_{start} - y_a}{y_b - y_a} * (x_b - x_a)$

Solving Polygon Equations at Scale

Remember: For every point (x,y) we want so solve the following equation for every edge in each obstacle:

$$\overrightarrow{A} + i \cdot \overrightarrow{AB} = \overrightarrow{P} + j \cdot \overrightarrow{N}$$

$$\Leftrightarrow \begin{bmatrix} a_x \\ a_y \end{bmatrix} + i \begin{bmatrix} ba_x \\ ba_y \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} + j \begin{bmatrix} n_x \\ n_y \end{bmatrix}$$

$$\Leftrightarrow i \begin{bmatrix} ba_x \\ ba_y \end{bmatrix} - j \begin{bmatrix} n_x \\ n_y \end{bmatrix} = \begin{bmatrix} p_x - a_x \\ p_y - a_x \end{bmatrix}$$

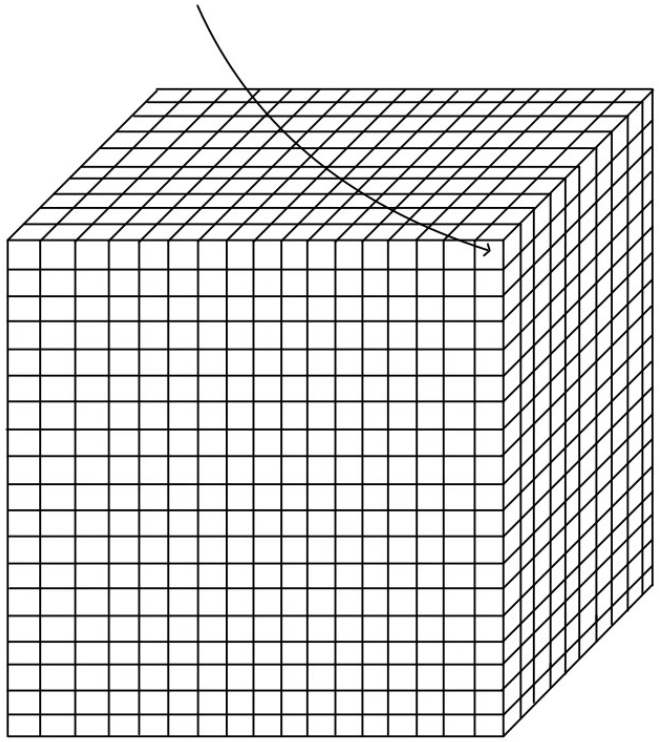
$$\Leftrightarrow \begin{bmatrix} ba_x - n_x \\ ba_y - n_y \end{bmatrix} * \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} p_x - a_x \\ p_y - a_x \end{bmatrix}$$

Solving Polygon Equations at Scale

4-th dimension:

$$\left[\begin{bmatrix} ba_{1_x} & -n_{1_x} \\ ba_{1_y} & -n_{1_y} \end{bmatrix} * \begin{bmatrix} i_1 \\ j_1 \end{bmatrix} = \begin{bmatrix} p_x - a_{1_x} \\ p_y - a_x \end{bmatrix} / \begin{bmatrix} ba_{2_x} & -n_{2_x} \\ ba_{2_y} & -n_{2_y} \end{bmatrix} * \begin{bmatrix} i_2 \\ j_2 \end{bmatrix} = \begin{bmatrix} p_x - a_{2_x} \\ p_y - a_{2_x} \end{bmatrix} / \ \ldots \right.$$

$$\left. \ldots / \ \ldots / \begin{bmatrix} ba_{m_x} & -n_{m_x} \\ ba_{m_y} & -n_{m_y} \end{bmatrix} * \begin{bmatrix} i_m \\ j_m \end{bmatrix} = \begin{bmatrix} p_x - a_{m_x} \\ p_y - a_{m_x} \end{bmatrix} \right]$$

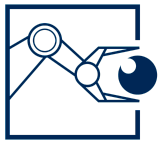


Solving Polygon Equations at Scale

PyTorch unfortunately does not offer this type of complexity while solving linear equations at scale.

Reformulation as a single matrix:

$$\begin{pmatrix} ba_{1_x} & -n_{1_x} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ ba_{1_y} & -n_{1_y} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & ba_{2_x} & -n_{2_x} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & ba_{2_y} & -n_{2_y} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & ba_{3_x} & -n_{3_x} & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & ba_{3_y} & -n_{3_y} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & ba_{m_x} & -n_{m_x} \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & ba_{m_y} & -n_{m_y} \end{pmatrix} \begin{pmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \\ \vdots \\ i_m \\ j_m \end{pmatrix} = \begin{pmatrix} p_{1_x} - a_{1_x} \\ p_{1_y} - a_{1_y} \\ p_{2_x} - a_{2_x} \\ p_{2_y} - a_{2_y} \\ p_{3_x} - a_{3_x} \\ p_{3_y} - a_{3_y} \\ \vdots \\ p_{m_x} - a_{m_x} \\ p_{m_y} - a_{m_y} \end{pmatrix}$$

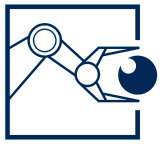


Solving Polygon Equations at Scale

Creating such a single matrix would result (in an environment of 800 x 640 of 10 regular polygons) in a matrix of size: 50.000.000 x 50.000.000.

This matrix does not fit into memory.

$$\begin{pmatrix} ba_{1_x} & -n_{1_x} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ ba_{1_y} & -n_{1_y} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & ba_{2_x} & -n_{2_x} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & ba_{2_y} & -n_{2_y} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & ba_{3_x} & -n_{3_x} & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & ba_{3_y} & -n_{3_y} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & ba_{m_x} & -n_{m_x} \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & ba_{m_y} & -n_{m_y} \end{pmatrix} \begin{pmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \\ \vdots \\ i_m \\ j_m \end{pmatrix} = \begin{pmatrix} p_{1_x} - a_{1_x} \\ p_{1_y} - a_{1_y} \\ p_{2_x} - a_{2_x} \\ p_{2_y} - a_{2_y} \\ p_{3_x} - a_{3_x} \\ p_{3_y} - a_{3_y} \\ \vdots \\ p_{m_x} - a_{m_x} \\ p_{m_y} - a_{m_y} \end{pmatrix}$$



Solving Polygon Equations at Scale

Reformulate the problem again and take advantage of the structure of the matrix: 2 x 2 block diagonal matrix

$$\begin{pmatrix} ba_{1_x} & -n_{1_x} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ ba_{1_y} & -n_{1_y} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & ba_{2_x} & -n_{2_x} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & ba_{2_y} & -n_{2_y} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & ba_{3_x} & -n_{3_x} & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & ba_{3_y} & -n_{3_y} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & ba_{m_x} & -n_{m_x} \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & ba_{m_y} & -n_{m_y} \end{pmatrix} \begin{pmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \\ \vdots \\ i_m \\ j_m \end{pmatrix} = \begin{pmatrix} p_{1_x} - a_{1_x} \\ p_{1_y} - a_{1_y} \\ p_{2_x} - a_{2_x} \\ p_{2_y} - a_{2_y} \\ p_{3_x} - a_{3_x} \\ p_{3_y} - a_{3_y} \\ \vdots \\ p_{m_x} - a_{m_x} \\ p_{m_y} - a_{m_y} \end{pmatrix}$$

Math we won't go into…

$$a_{m_y} + i_m \cdot ba_{m_y} = p_{m_y} + j_m \cdot n_{m_y}$$

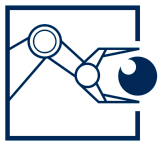
$$\Leftrightarrow\ a_{m_y} + i_m \cdot ba_{m_y} = p_{m_y} + \frac{a_{m_x} + i_m \cdot ba_{m_x} - p_{m_x}}{n_{m_x}} \cdot n_{m_y}$$

$$\Leftrightarrow\ n_{m_x} \cdot (a_{m_y} + i_m \cdot ba_{m_y}) = p_{m_y} \cdot n_{m_x} + (a_{m_x} + i_m \cdot ba_{m_x} - p_{m_x}) \cdot n_{m_y}$$

$$\Leftrightarrow\ n_{m_x} \cdot a_{m_y} + i_m \cdot ba_{m_y} \cdot n_{m_x} = p_{m_y} \cdot n_{m_x} + a_{m_x} \cdot ba_{m_y} + i_m \cdot ba_{m_x} \cdot n_{m_y} - p_{m_x} \cdot n_{m_y}$$

$$\Leftrightarrow\ i_m \cdot ba_{m_y} \cdot n_{m_x} - i_m \cdot ba_{m_x} \cdot n_{m_y} = p_{m_y} \cdot n_{m_x} + a_{m_x} \cdot n_{m_y} - n_{m_x} \cdot a_{m_y} - p_{m_x} \cdot n_{m_y}$$

$$\Leftrightarrow\ i_m = \frac{p_{m_y} \cdot n_{m_x} + a_{m_x} \cdot n_{m_y} - n_{m_x} \cdot a_{m_y} - p_{m_x} \cdot n_{m_y}}{ba_{m_y} \cdot n_{m_x} - ba_{m_x} \cdot n_{m_y}}$$



More Math we won't go into…

$$a_{m_y} + i_m \cdot ba_{m_y} = p_{m_y} + j_m \cdot n_{m_y}$$

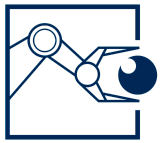
$$\Leftrightarrow \; a_{m_y} + \frac{p_{m_x} + j_m \cdot n_{m_x} - a_{m_x}}{ba_{m_x}} \cdot ba_{m_y} = p_{m_y} + j_m \cdot n_{y_m}$$

$$\Leftrightarrow \; a_{m_y} \cdot ba_{m_x} + (p_{m_x} + j_m \cdot n_{m_x} - a_{m_x}) \cdot ba_{m_y} = (p_{m_y} + j_m \cdot n_{m_y}) \cdot ba_{m_x}$$

$$\Leftrightarrow \; a_{m_y} \cdot ba_{m_x} + p_{m_x} \cdot ba_{m_y} + j_m \cdot n_{m_x} \cdot ba_{m_y} - a_{m_x} \cdot ba_{m_y} = p_{m_y} \cdot ba_{m_x} + j \cdot n_{m_y} \cdot ba_{m_x}$$

$$\Leftrightarrow \; a_{m_y} \cdot ba_{m_x} + p_{m_x} \cdot ba_{m_y} - a_{m_x} \cdot ba_{m_y} - p_{m_y} \cdot ba_{m_x} = j_m \cdot n_{m_y} \cdot ba_{m_x} - j_m \cdot n_{m_x} \cdot ba_{m_y}$$

$$\Leftrightarrow \; j_m = \frac{a_{m_y} \cdot ba_{m_x} + p_{m_x} \cdot ba_{m_y} - a_{m_x} \cdot ba_{m_y} - p_{m_y} \cdot ba_{m_x}}{n_{m_y} \cdot ba_{m_x} - n_{m_x} \cdot ba_{m_y}}$$



Math Takeaways

$$\begin{pmatrix} ba_{1_x} & -n_{1_x} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ ba_{1_y} & -n_{1_y} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & ba_{2_x} & -n_{2_x} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & ba_{2_y} & -n_{2_y} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & ba_{3_x} & -n_{3_x} & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & ba_{3_y} & -n_{3_y} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & ba_{n_x} & -n_{n_x} \\ 0 & 0 & 0 & 0 & 0 & 0 & \ldots & ba_{n_y} & -n_{n_y} \end{pmatrix} \begin{pmatrix} i_1 \\ j_1 \\ i_2 \\ j_2 \\ i_3 \\ j_3 \\ \vdots \\ i_{(n)} \\ j_{(n)} \end{pmatrix} = \begin{pmatrix} p_{1_x} - a_{1_x} \\ p_{1_y} - a_{1_y} \\ p_{2_x} - a_{2_x} \\ p_{2_y} - a_{2_y} \\ p_{3_x} - a_{3_x} \\ p_{3_y} - a_{3_y} \\ \vdots \\ p_{n_x} - a_{n_x} \\ p_{n_y} - a_{n_y} \end{pmatrix}$$

$$i_m = \frac{p_{m_y} \cdot n_{m_x} + a_{m_x} \cdot n_{m_y} - n_{m_x} \cdot a_{m_y} - p_{m_x} \cdot n_{m_y}}{ba_{m_y} \cdot n_{m_x} - ba_{m_x} \cdot n_{m_y}}$$

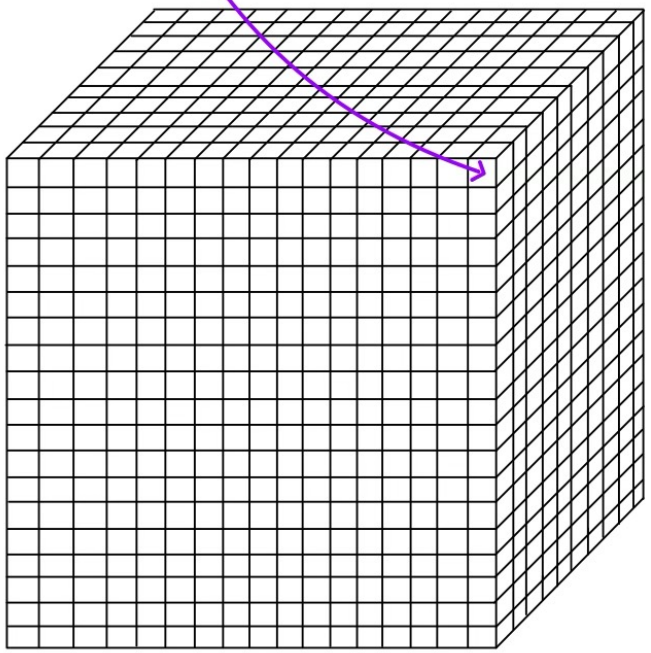
$$j_m = \frac{a_{m_y} \cdot ba_{m_x} + p_{m_x} \cdot ba_{m_y} - a_{m_x} \cdot ba_{m_y} - p_{m_y} \cdot ba_{m_x}}{n_{m_y} \cdot ba_{m_x} - n_{m_x} \cdot ba_{m_y}}$$

Recall

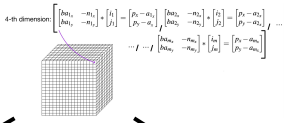
4-th dimension:

$$\left[\begin{bmatrix} ba_{1_x} & -n_{1_x} \\ ba_{1_y} & -n_{1_y} \end{bmatrix} * \begin{bmatrix} i_1 \\ j_1 \end{bmatrix} = \begin{bmatrix} p_x - a_{1_x} \\ p_y - a_x \end{bmatrix} / \begin{bmatrix} ba_{2_x} & -n_{2_x} \\ ba_{2_y} & -n_{2_y} \end{bmatrix} * \begin{bmatrix} i_2 \\ j_2 \end{bmatrix} = \begin{bmatrix} p_x - a_{2_x} \\ p_y - a_{2_x} \end{bmatrix} / \ \ldots \right.$$

$$\left. \cdots / \ \cdots / \begin{bmatrix} ba_{m_x} & -n_{m_x} \\ ba_{m_y} & -n_{m_y} \end{bmatrix} * \begin{bmatrix} i_m \\ j_m \end{bmatrix} = \begin{bmatrix} p_x - a_{m_x} \\ p_y - a_{m_x} \end{bmatrix} \right]$$



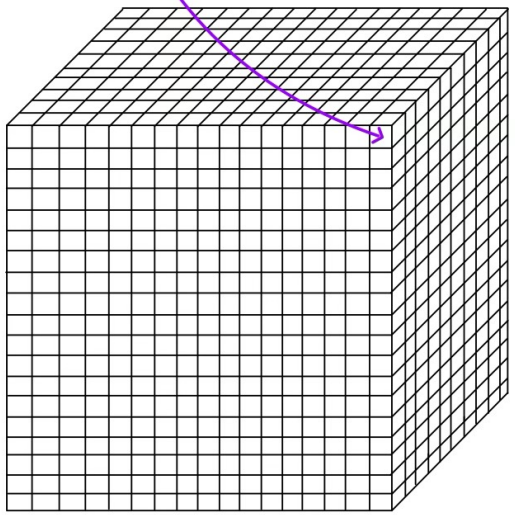
Tensor Math for Repulsive Force of Polygons



Split

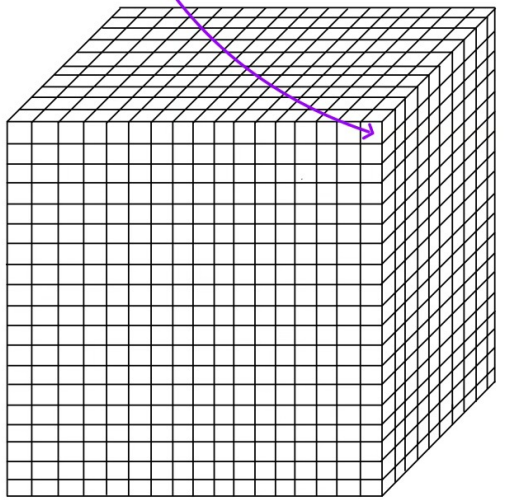
4-th dimension: $\left[i_1 \, , \, i_2 \, , \cdots , \cdots , \, i_m \right]$

4-th dimension: $\left[j_1 \, , \, j_2 \, , \cdots , \cdots , \, j_m \right]$



$$i_m = \frac{p_{m_y} \cdot n_{m_x} + a_{m_x} \cdot n_{m_y} - n_{m_x} \cdot a_{m_y} - p_{m_x} \cdot n_{m_y}}{ba_{m_y} \cdot n_{m_x} - ba_{m_x} \cdot n_{m_y}}$$

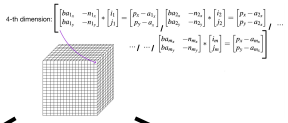
Create tensors and do math?



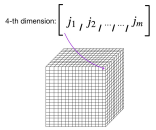
“i-value tensor”

“j-value tensor”

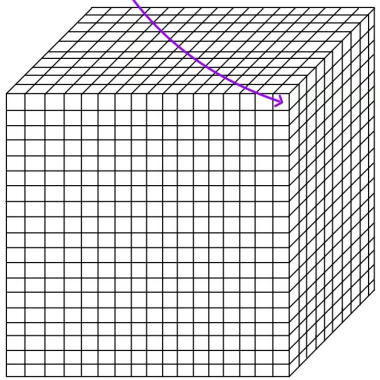
Tensor Math for Repulsive Force of Polygons



Split

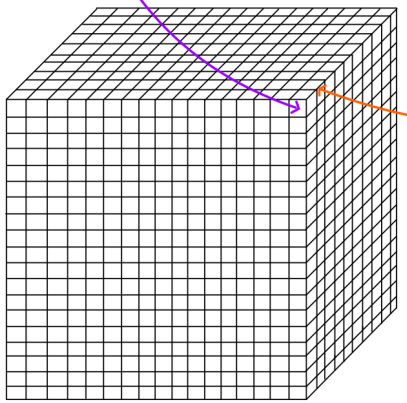


4-th dimension: $\left[i_1 , i_2 , \cdots , \cdots , i_m \right]$



“i-value tensor”

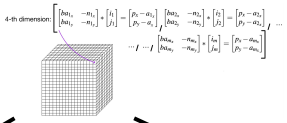
4-th dimension: $\left[i_1 , i_2 , \cdots , \cdots , i_m \right]$



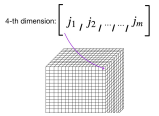
4-th dimension: $\left[i_1 , i_2 , \cdots , \cdots , i_k \right]$

Size: height x width x #polygons x ?

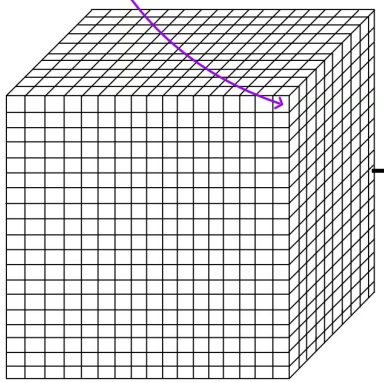
Tensor Math for Repulsive Force of Polygons



Split

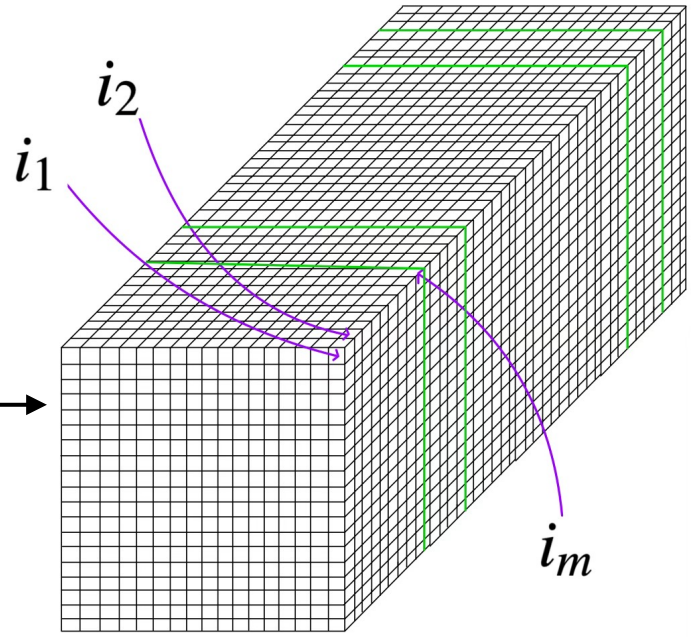


4-th dimension: $\left[i_1 \, , \, i_2 \, , \cdots , \cdots , \, i_m \right]$



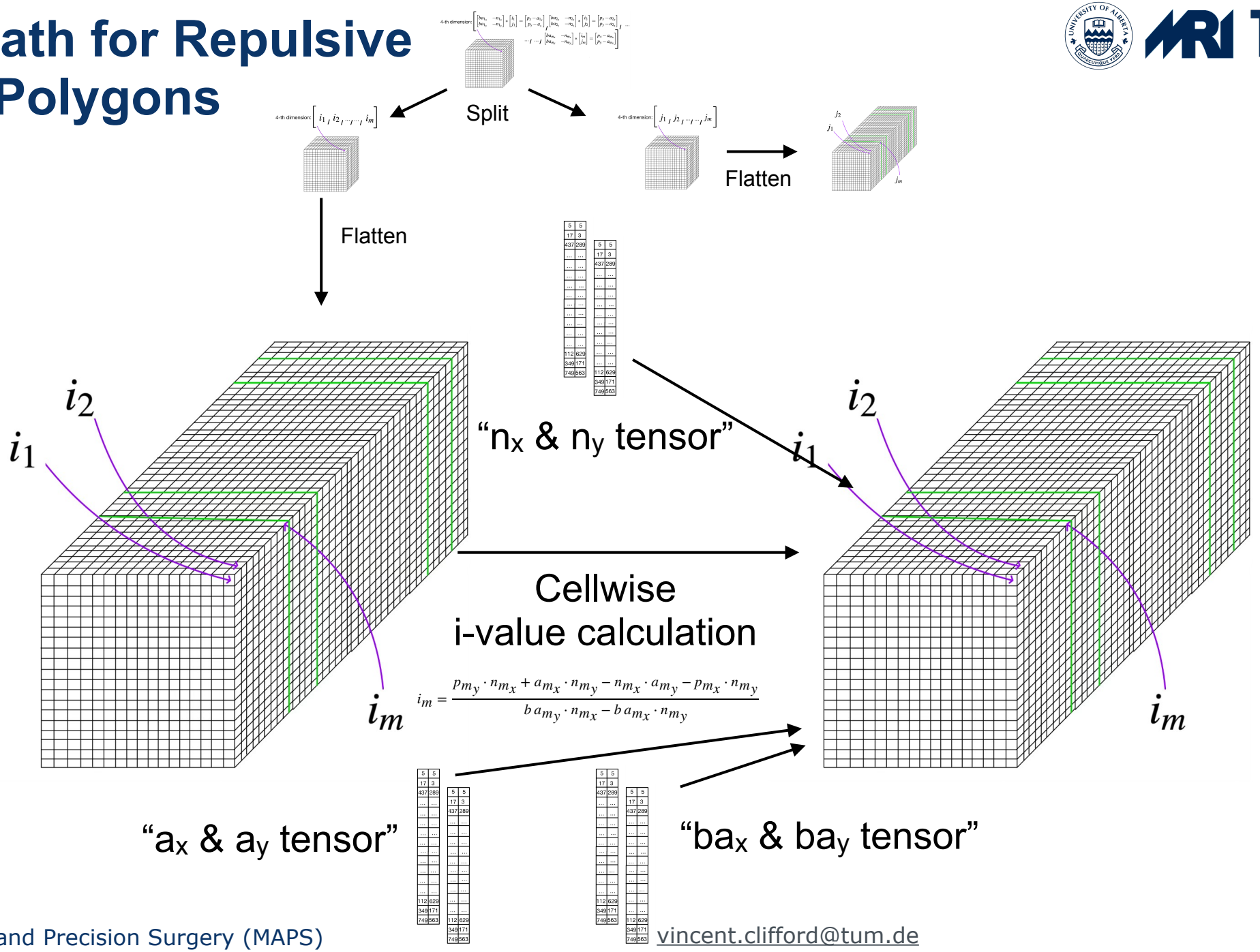
“i-value tensor”

Flatten

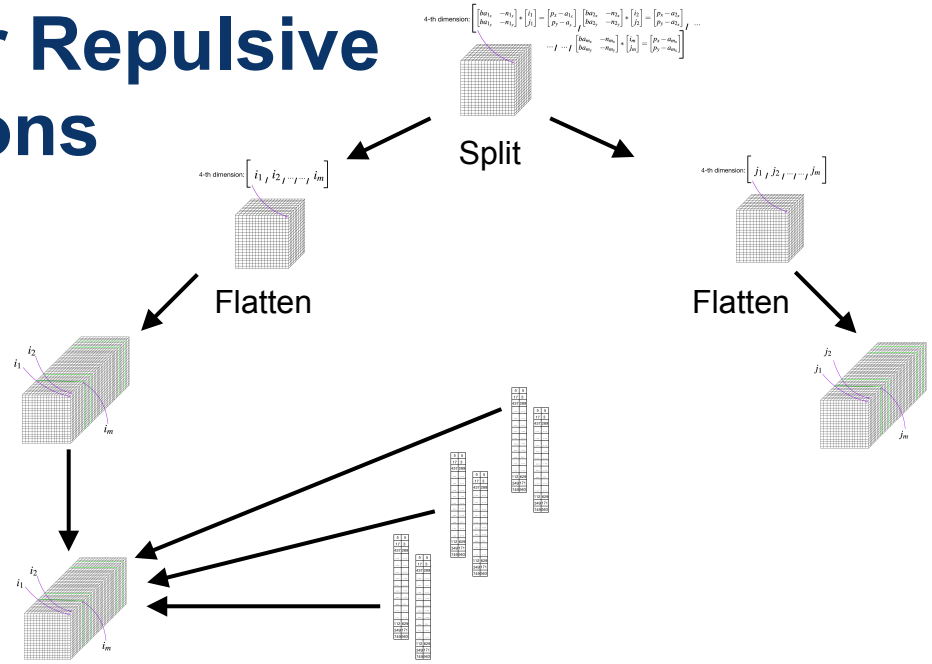


Size: height x width x $\sum_{\text{polygon}} \#edges_{polygon}$

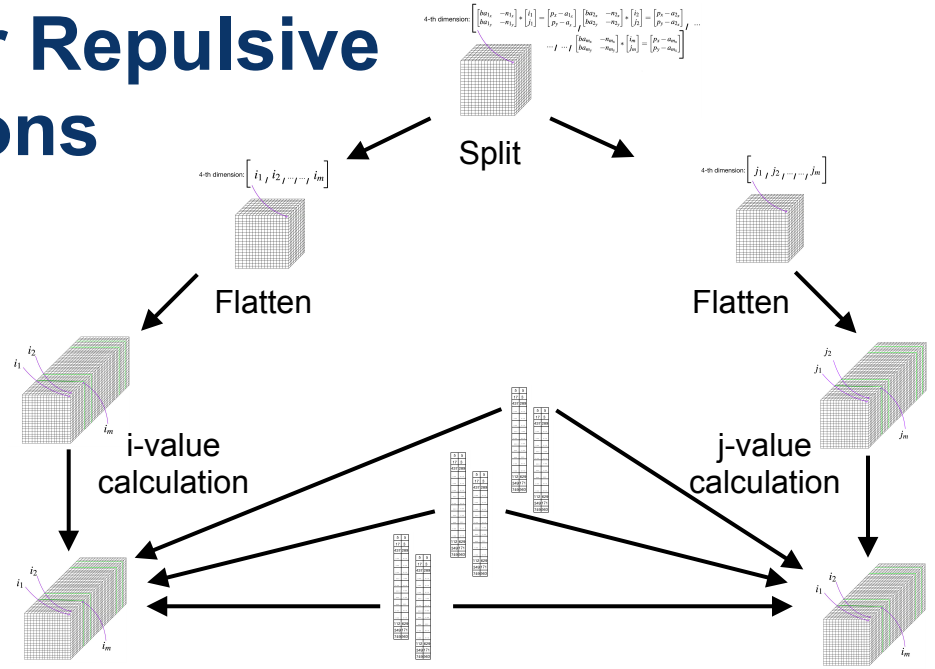
Tensor Math for Repulsive Force of Polygons



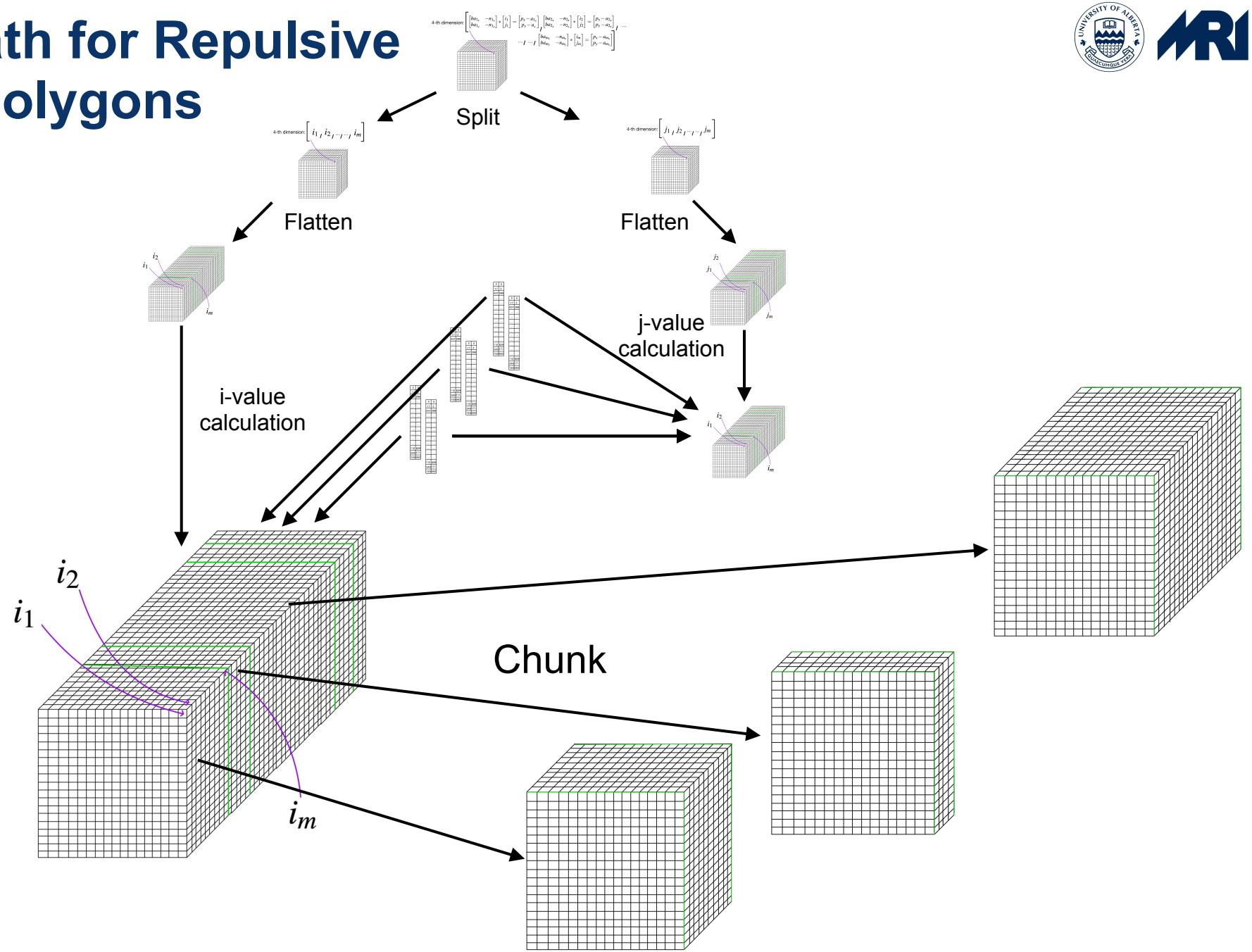
Tensor Math for Repulsive Force of Polygons



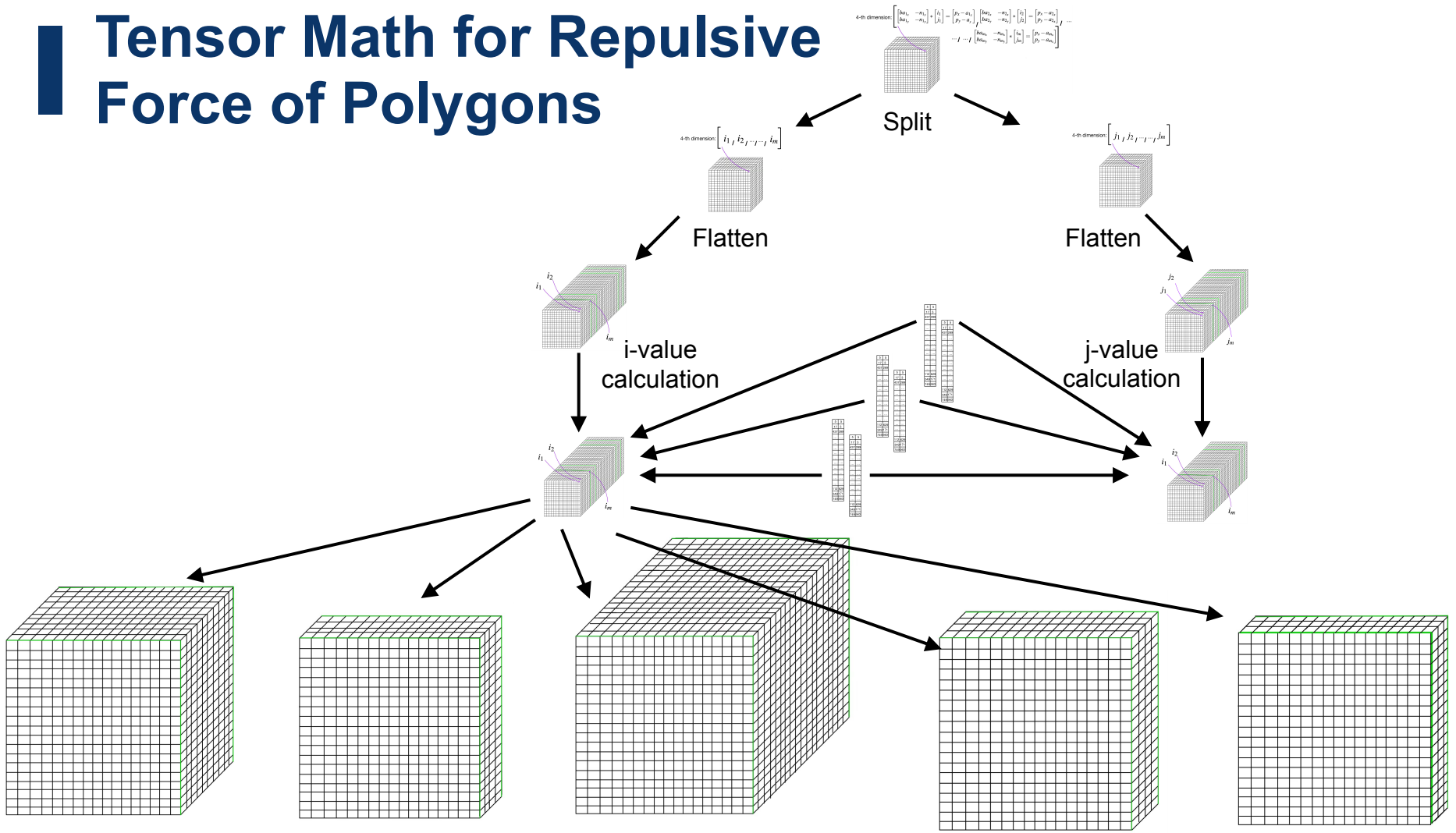
Tensor Math for Repulsive Force of Polygons



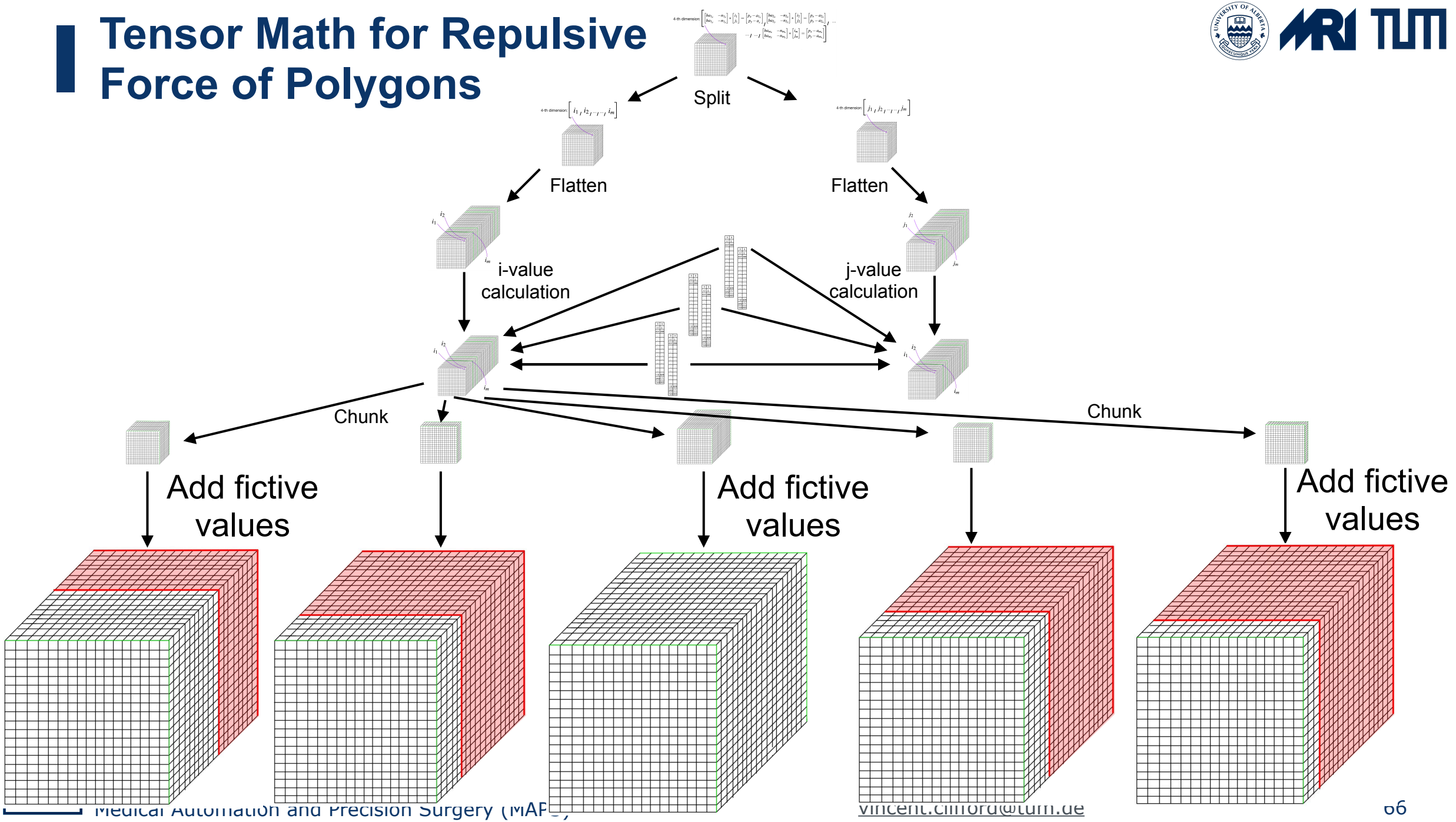
Tensor Math for Repulsive Force of Polygons



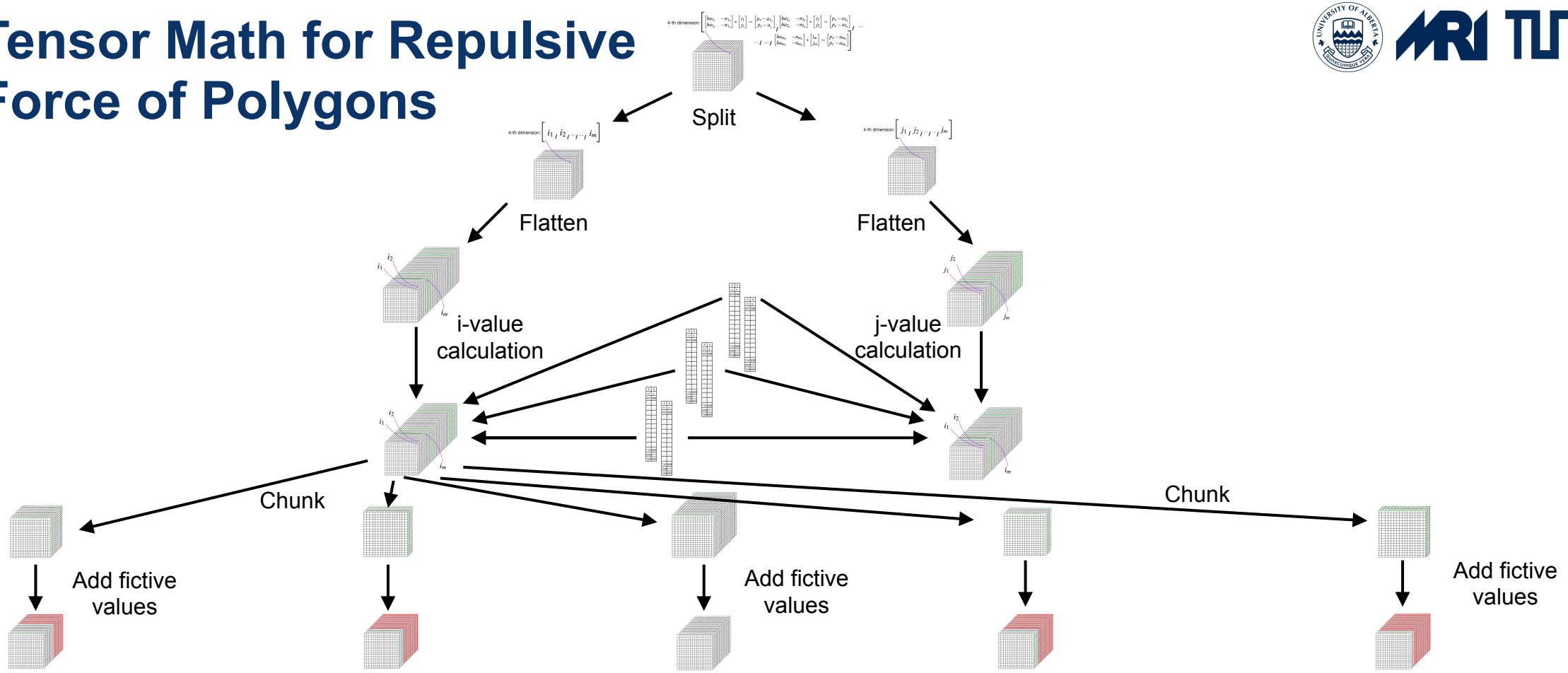
Tensor Math for Repulsive Force of Polygons



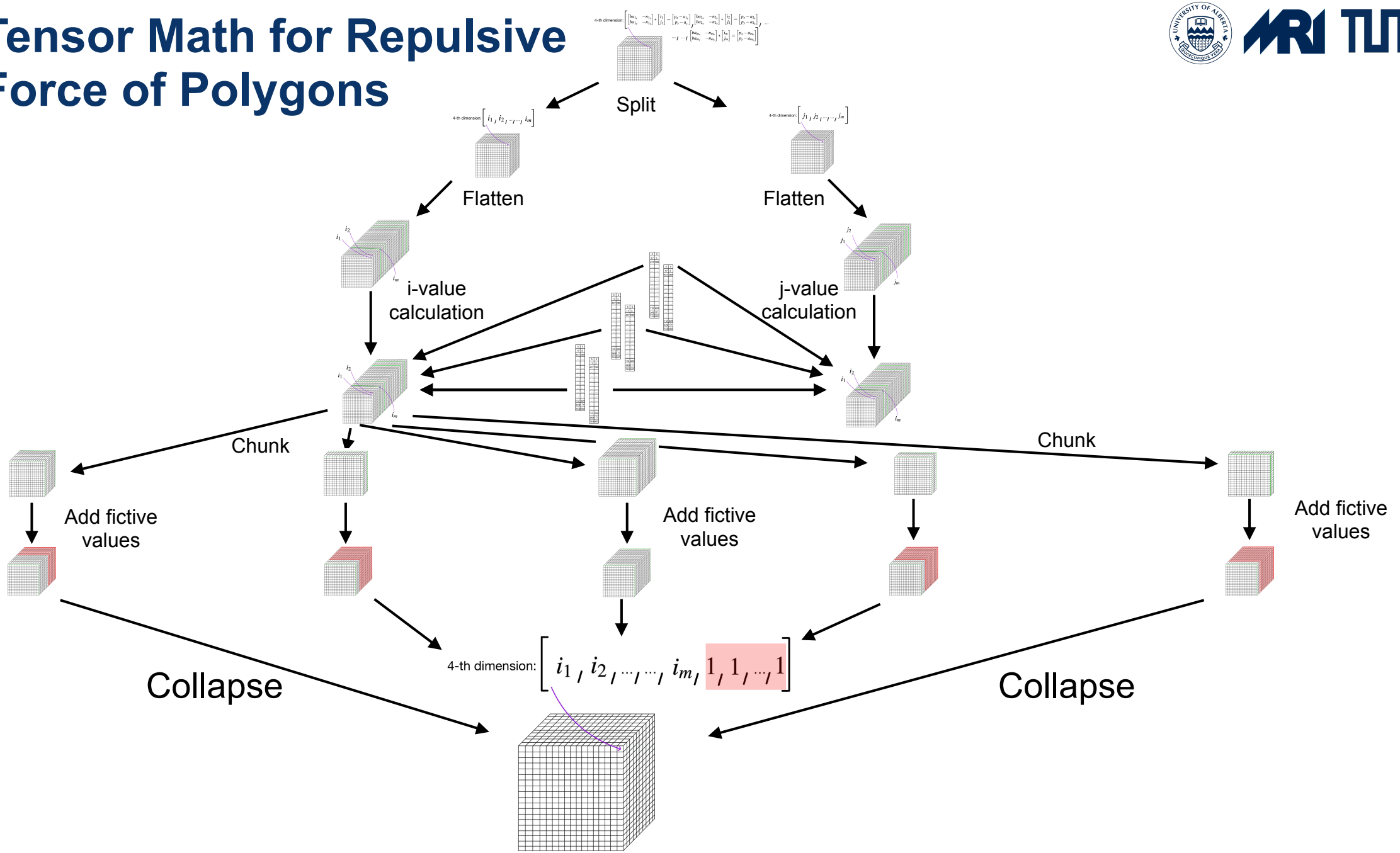
Tensor Math for Repulsive Force of Polygons



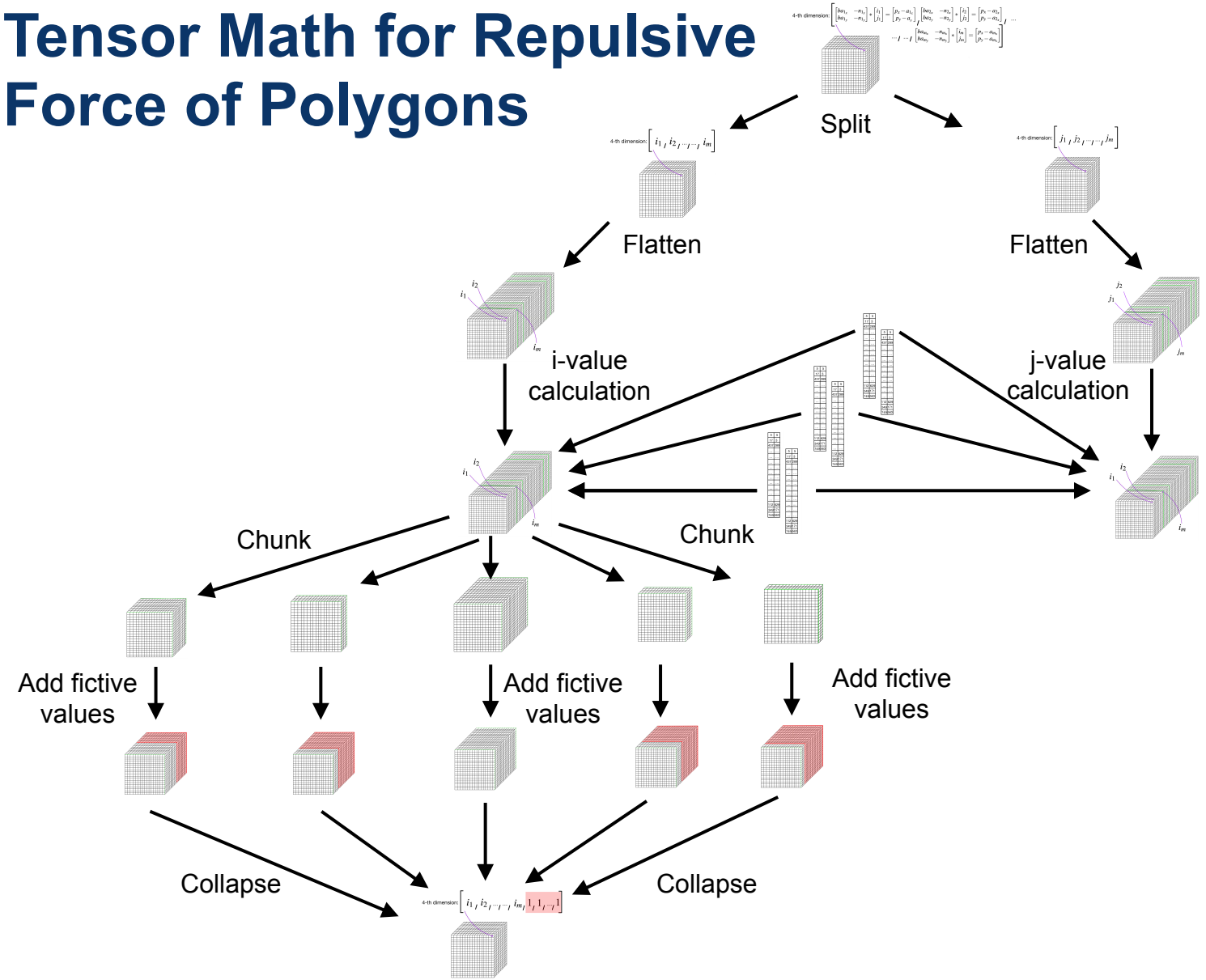
Tensor Math for Repulsive Force of Polygons



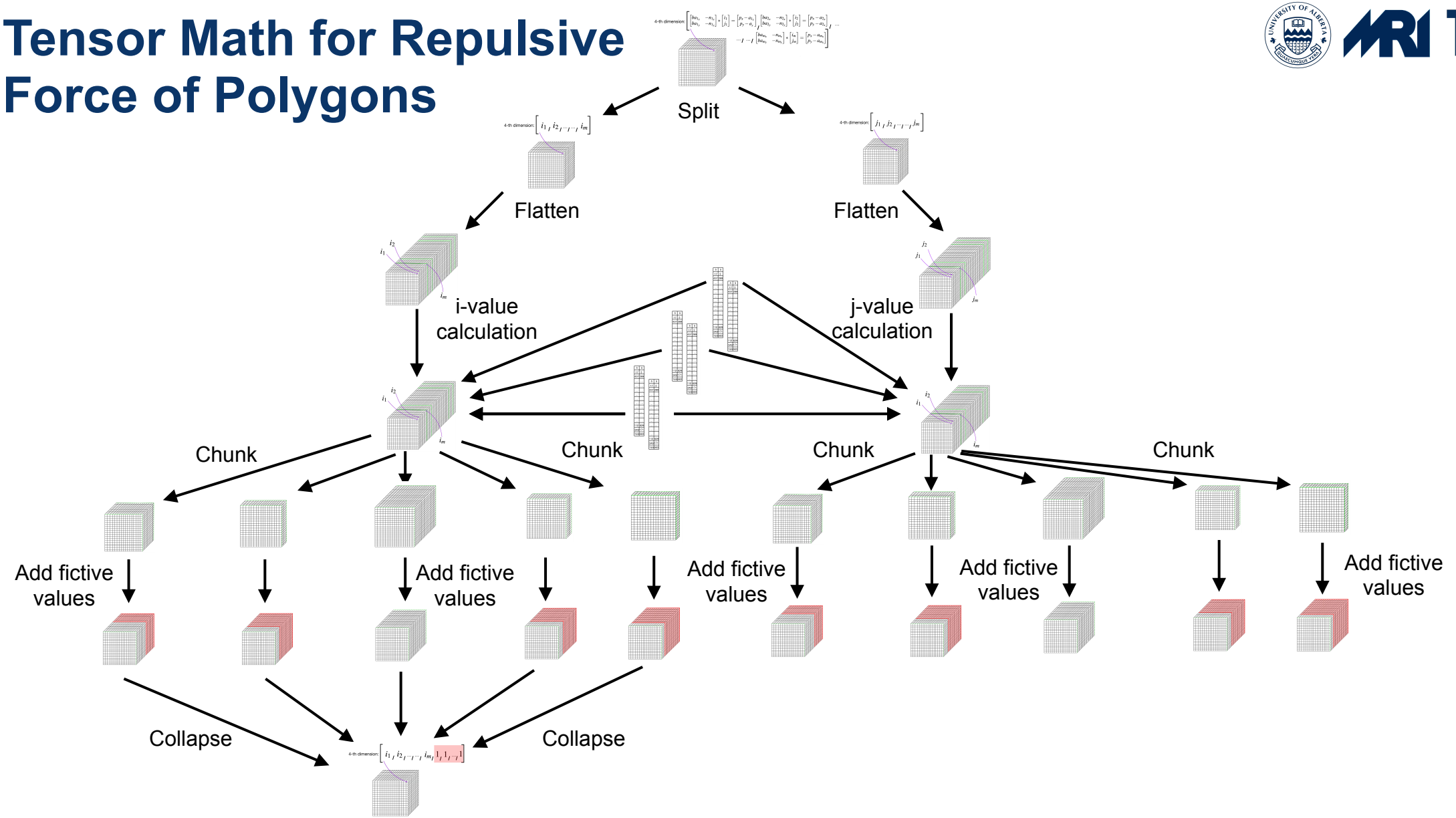
Tensor Math for Repulsive Force of Polygons



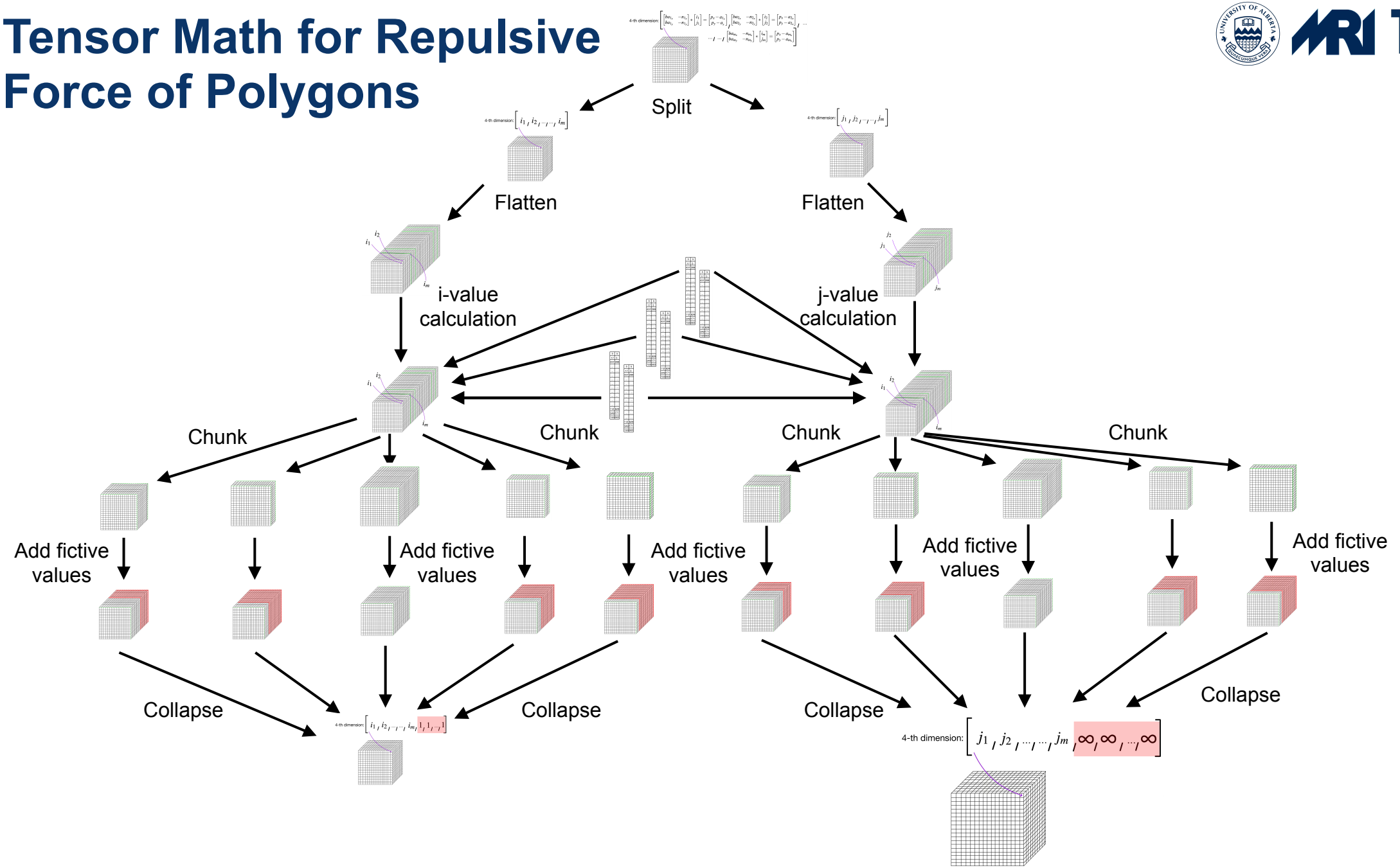
Tensor Math for Repulsive Force of Polygons

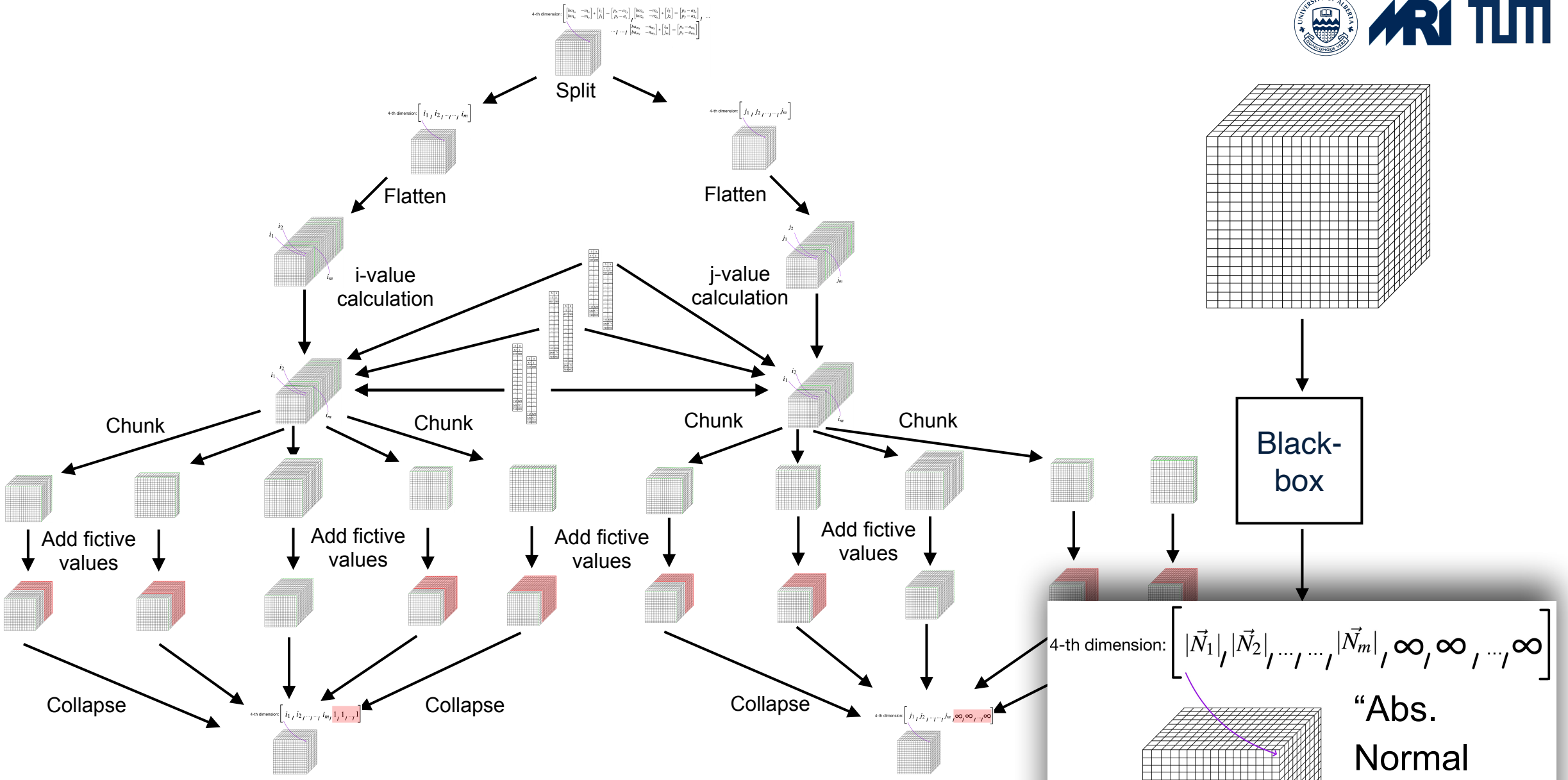


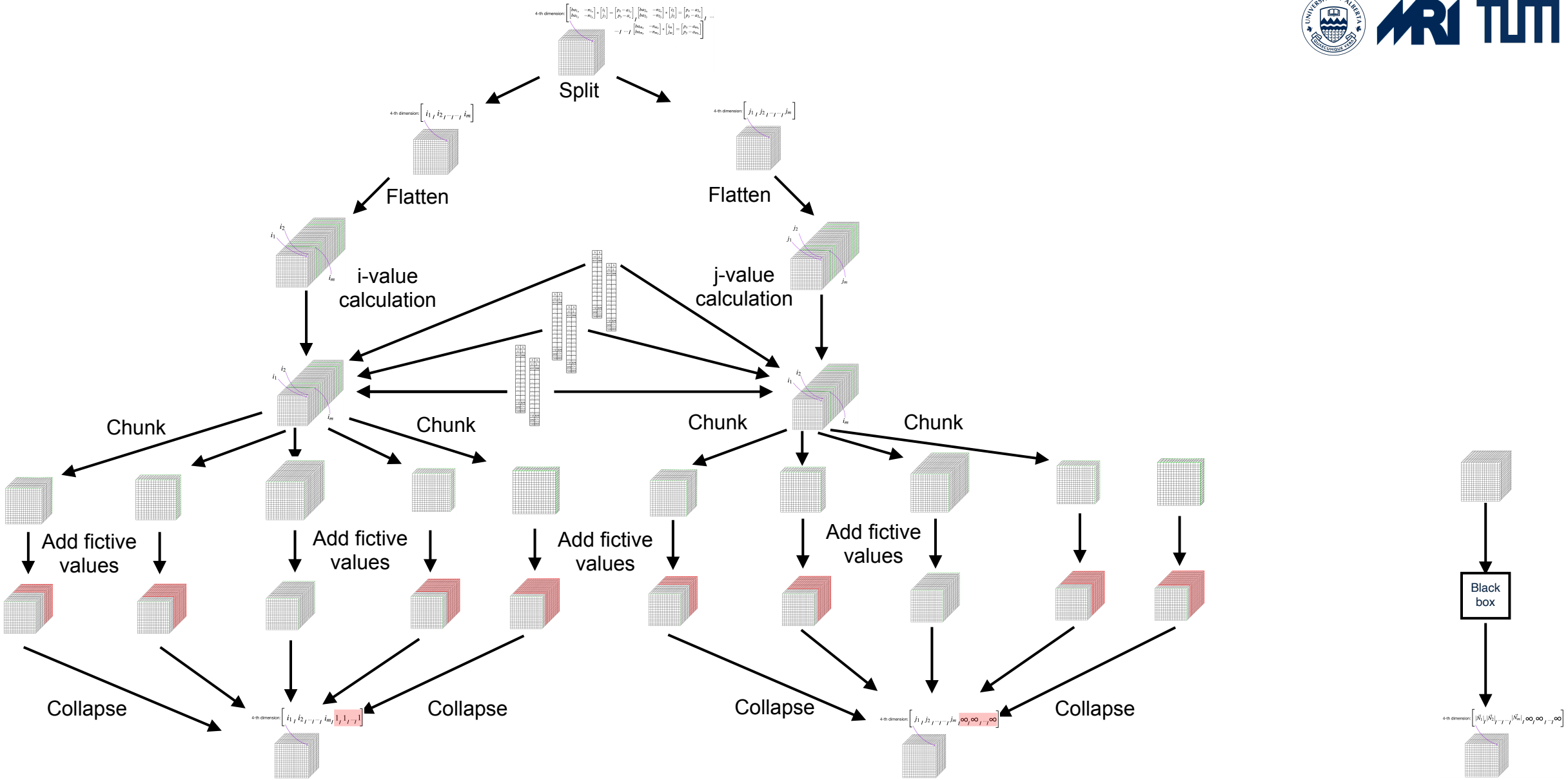
Tensor Math for Repulsive Force of Polygons

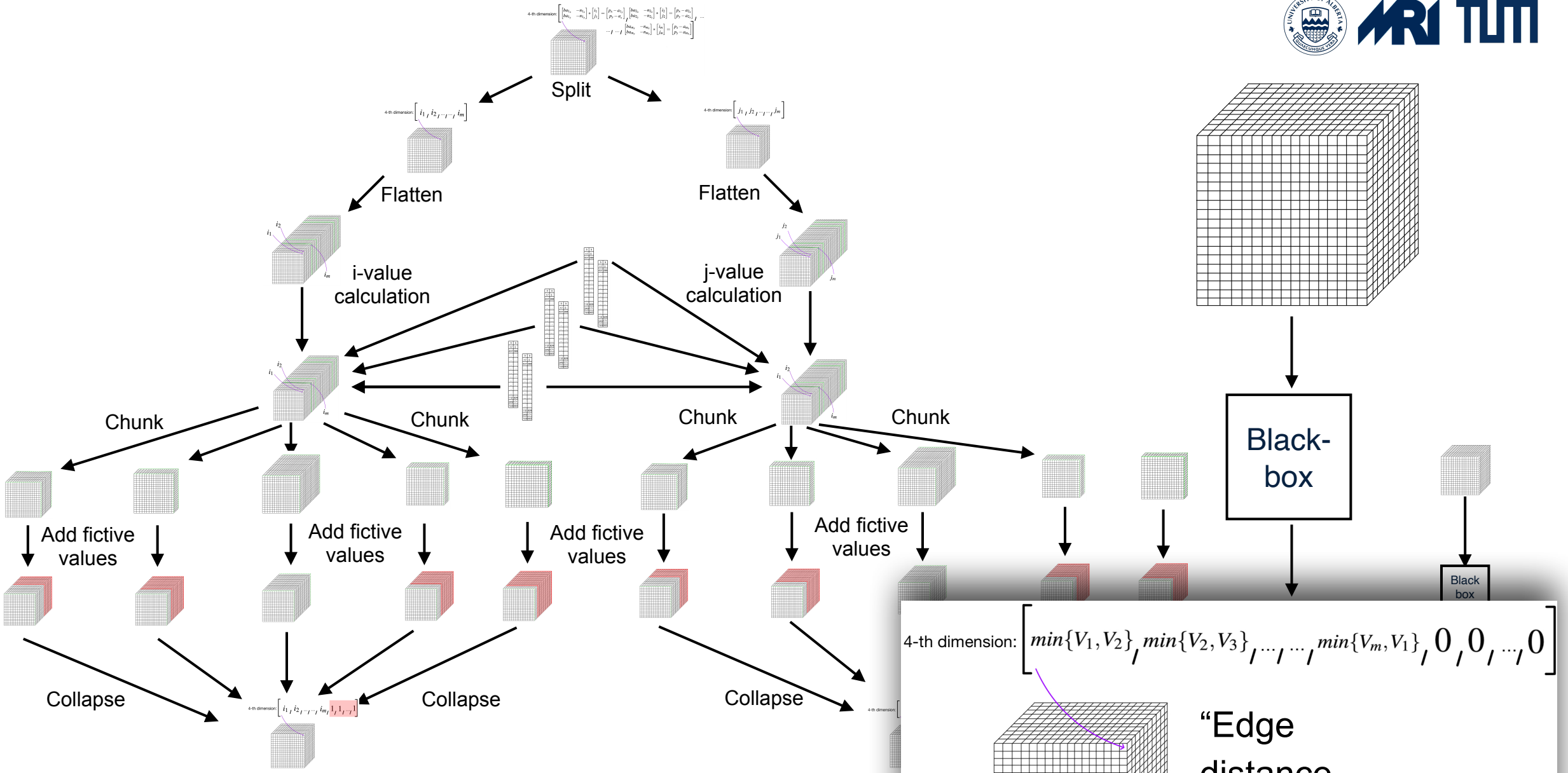


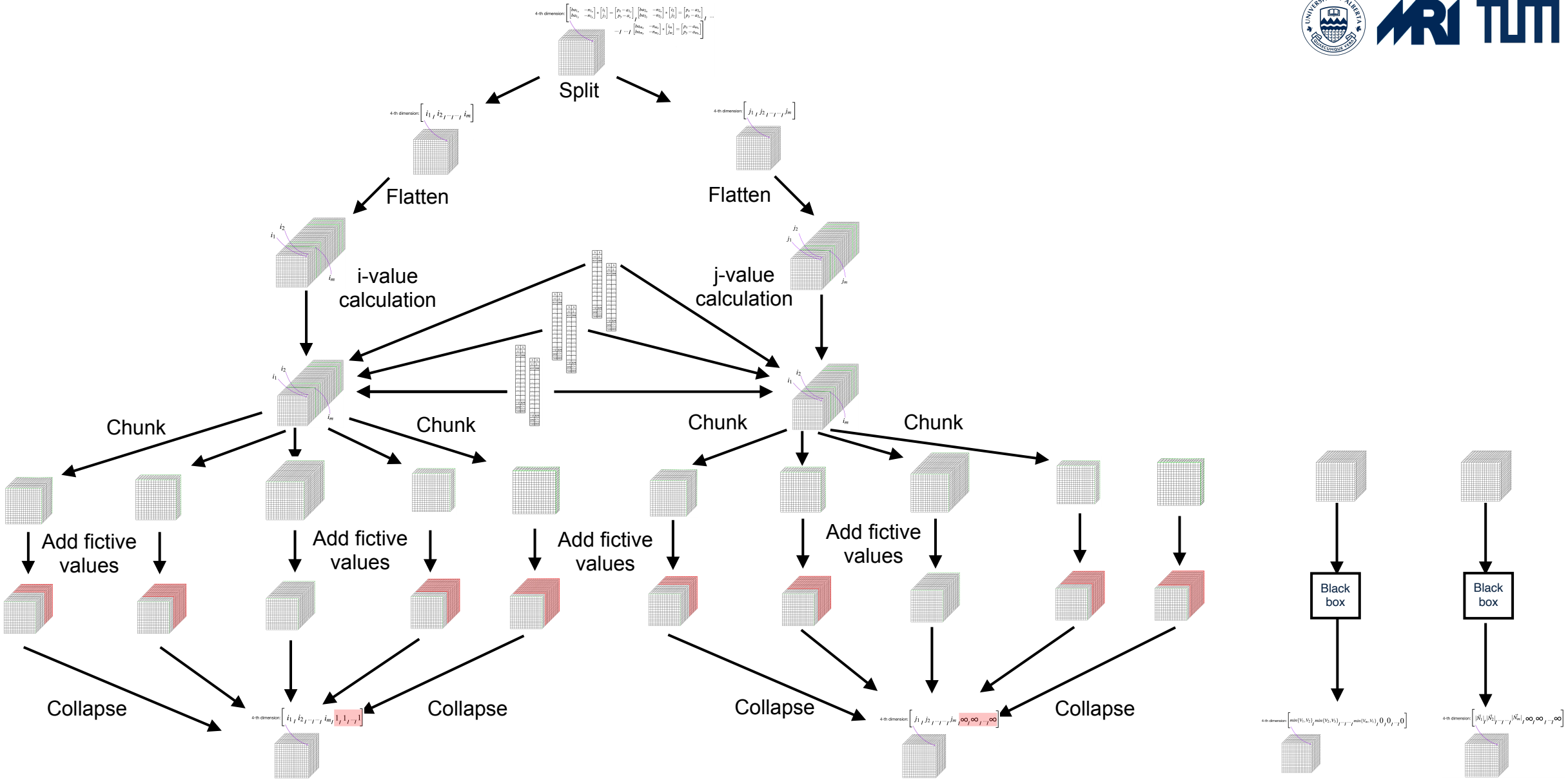
Tensor Math for Repulsive Force of Polygons



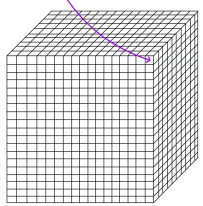

Split
Flatten
Flatten
i-value calculation
j-value calculation
Chunk
Chunk
Chunk
Chunk
Add fictive values
Add fictive values
Add fictive values
Add fictive values
Collapse
Collapse
Collapse
Black-box
4-th dimension:
"Abs. Normal distance tensor"


Split
Flatten
Flatten
i-value calculation
j-value calculation
Chunk
Chunk
Chunk
Chunk
Add fictive values
Add fictive values
Add fictive values
Add fictive values
Collapse
Collapse
Collapse
Collapse
Black box

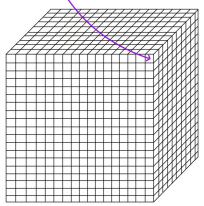

Split
Flatten
Flatten
i-value calculation
j-value calculation
Chunk
Chunk
Chunk
Chunk
Add fictive values
Add fictive values
Add fictive values
Add fictive values
Collapse
Collapse
Collapse
Black-box
Black box
4-th dimension: $\left[min\{V_1,V_2\}, min\{V_2,V_3\}, \ldots, \ldots, min\{V_m,V_1\}, 0, 0, \ldots, 0\right]$
"Edge distance tensor"


Split
Flatten
Flatten
i-value calculation
j-value calculation
Chunk
Chunk
Chunk
Chunk
Add fictive values
Add fictive values
Add fictive values
Add fictive values
Collapse
Collapse
Collapse
Collapse
Black box
Black box

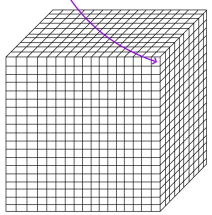
4-th dimension: $\left[i_1, i_2, \cdots, \cdots, i_m, 1, 1, \cdots, 1 \right]$



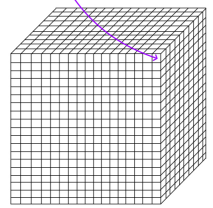
4-th dimension: $\left[j_1, j_2, \cdots, \cdots, j_m, \infty, \infty, \cdots, \infty \right]$



4-th dimension: $\left[|\vec{N}_1|, |\vec{N}_2|, \cdots, \cdots, |\vec{N}_m|, \infty, \infty, \cdots, \infty \right]$

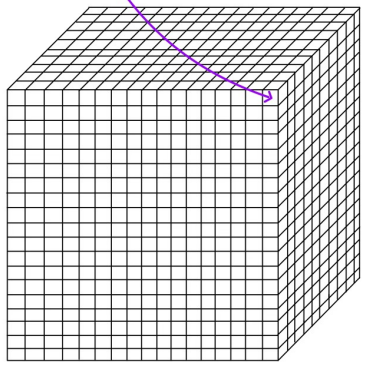


4-th dimension: $\left[min\{V_1, V_2\}, min\{V_2, V_3\}, \cdots, \cdots, min\{V_m, V_1\}, 0, 0, \cdots, 0 \right]$

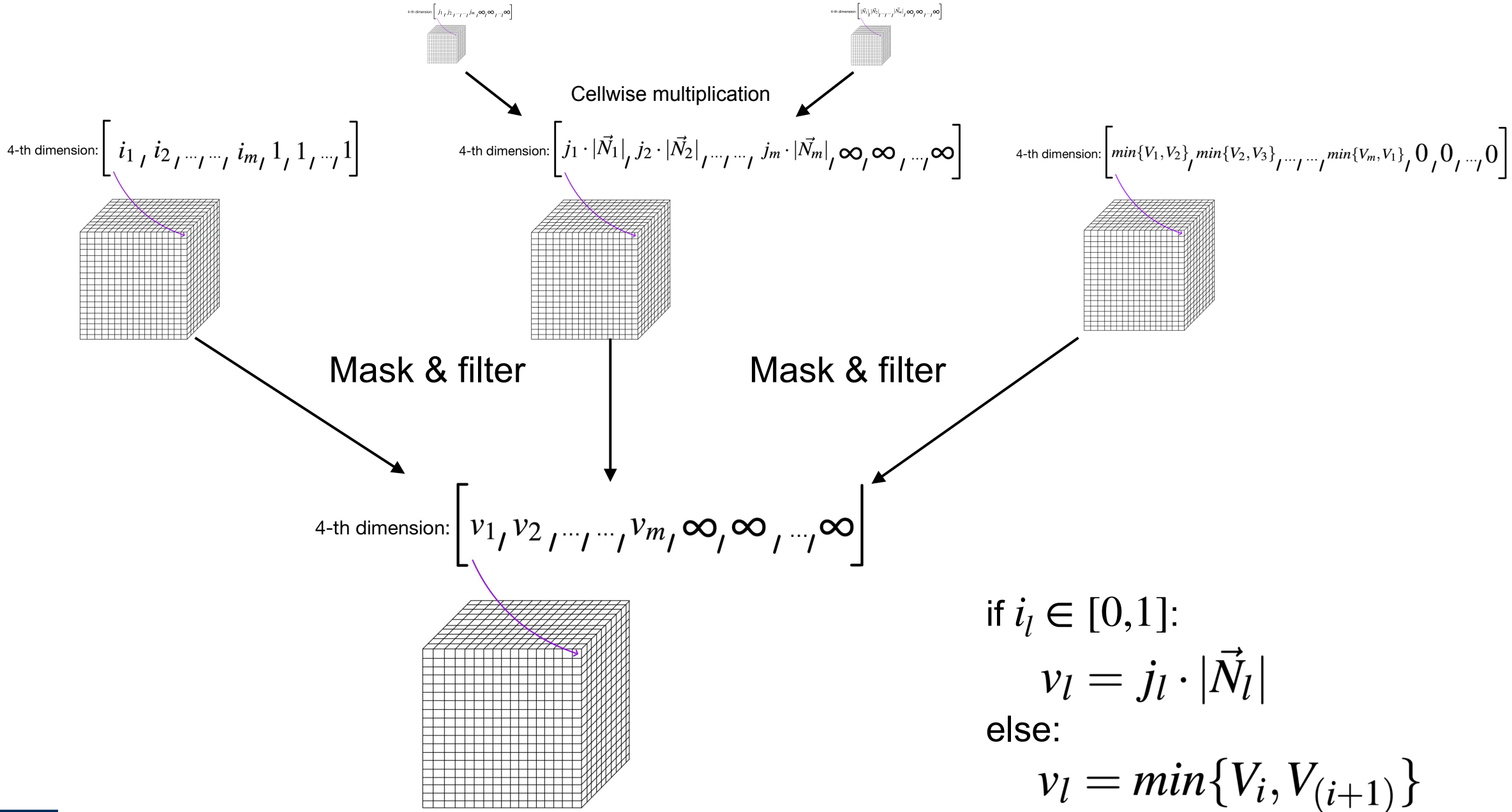


Cellwise multiplication

4-th dimension: $\left[j_1 \cdot |\vec{N}_1|, j_2 \cdot |\vec{N}_2|, \cdots, \cdots, j_m \cdot |\vec{N}_m|, \infty, \infty, \cdots, \infty \right]$



“Scaled distance tensor”

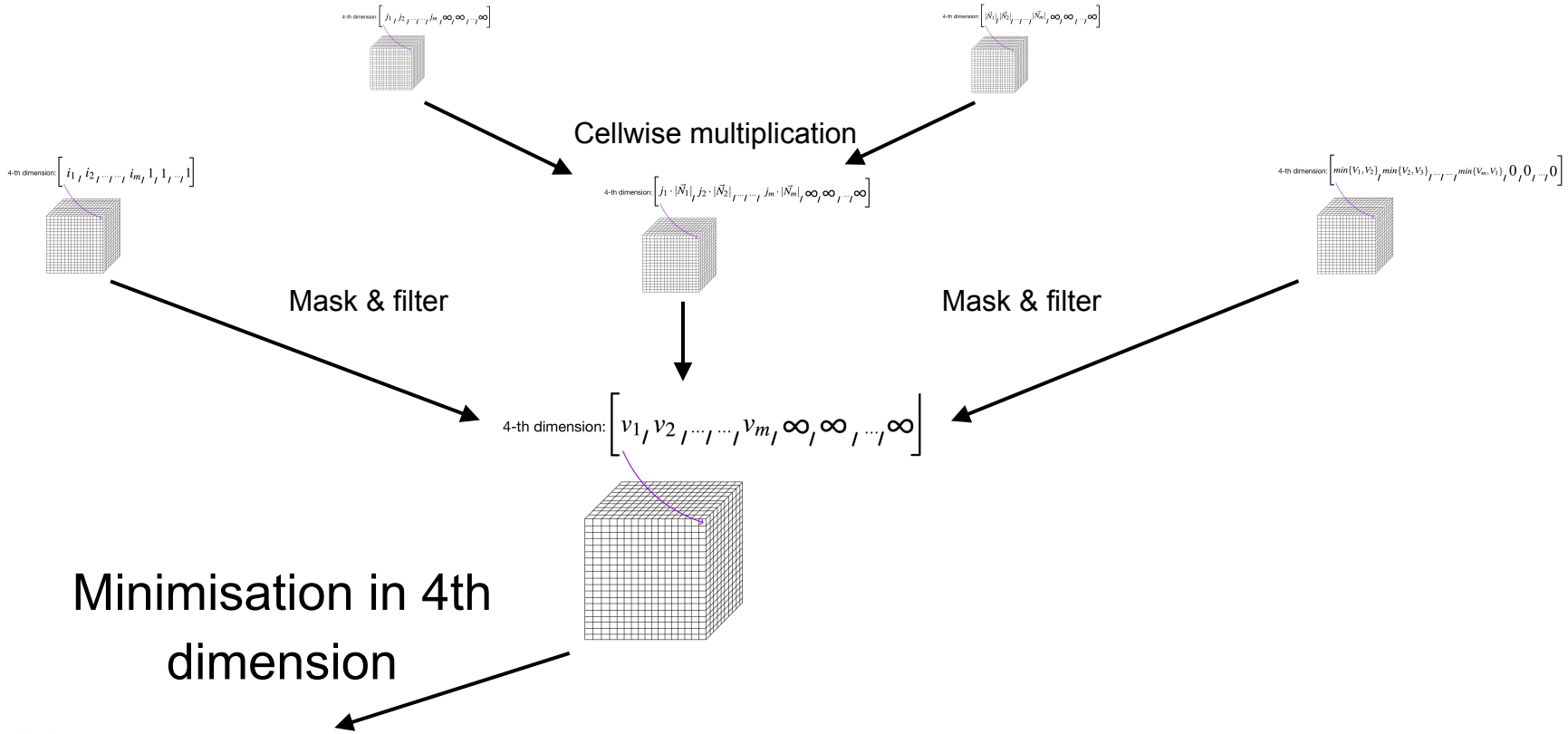


if $i_l \in [0,1]$:

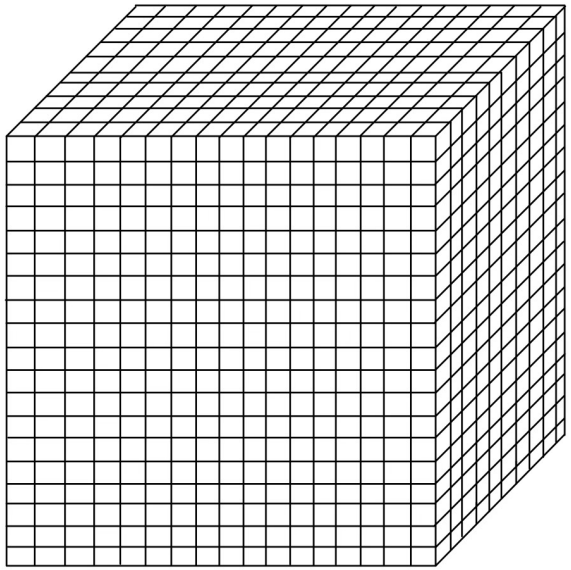
$$v_l = j_l \cdot |\vec{N}_l|$$

else:

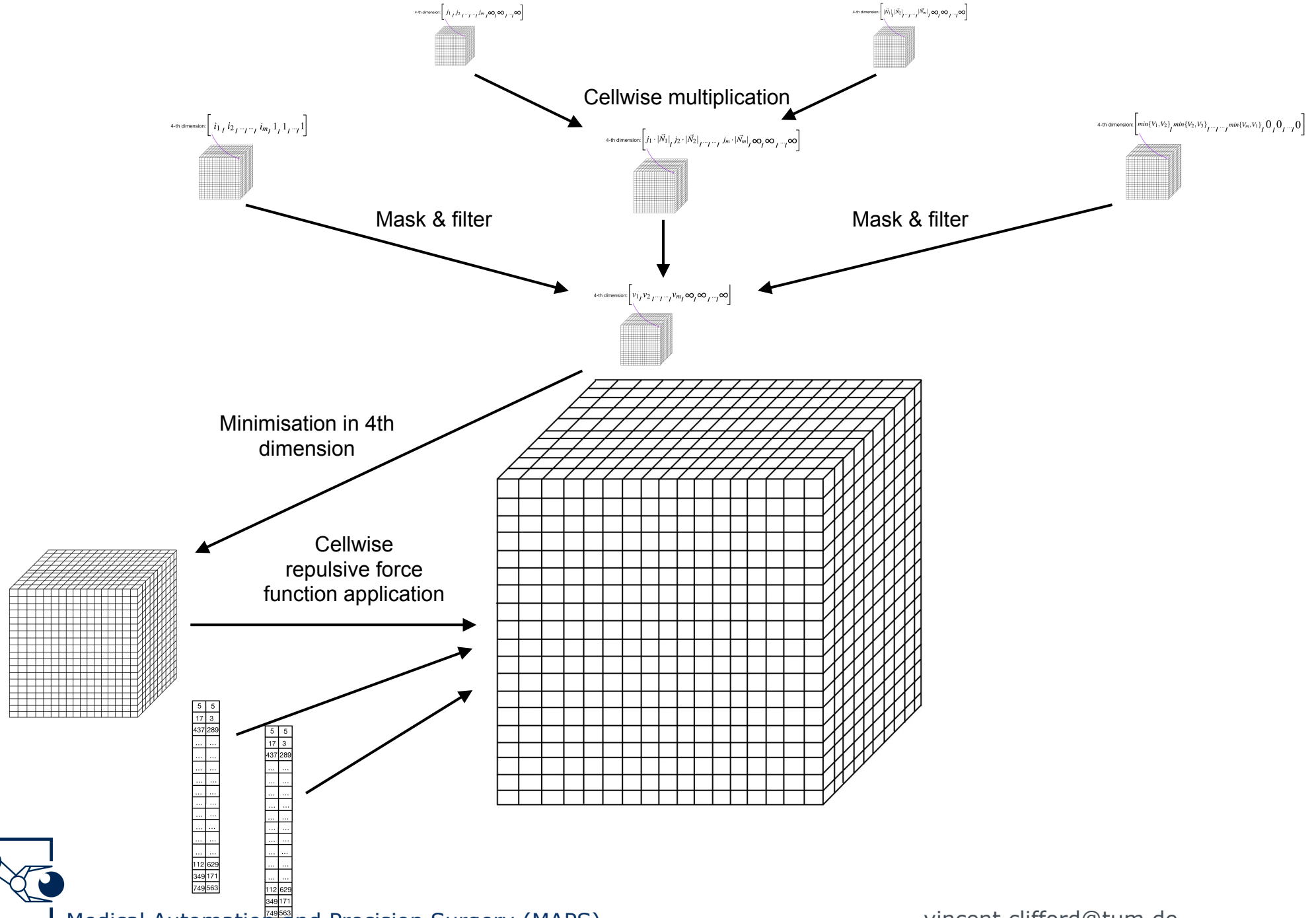
$$v_l = min\{V_i, V_{(i+1)}\}$$

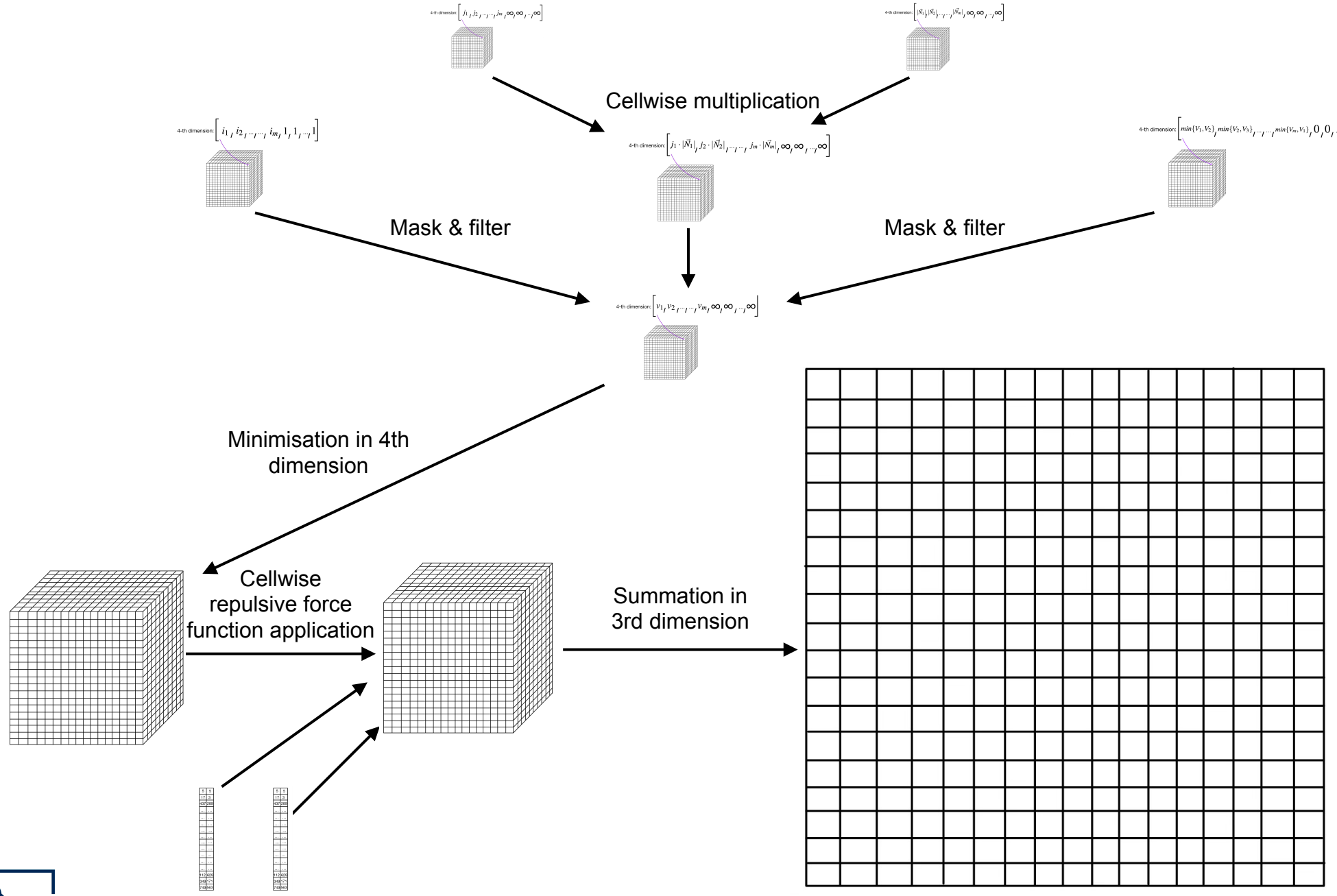


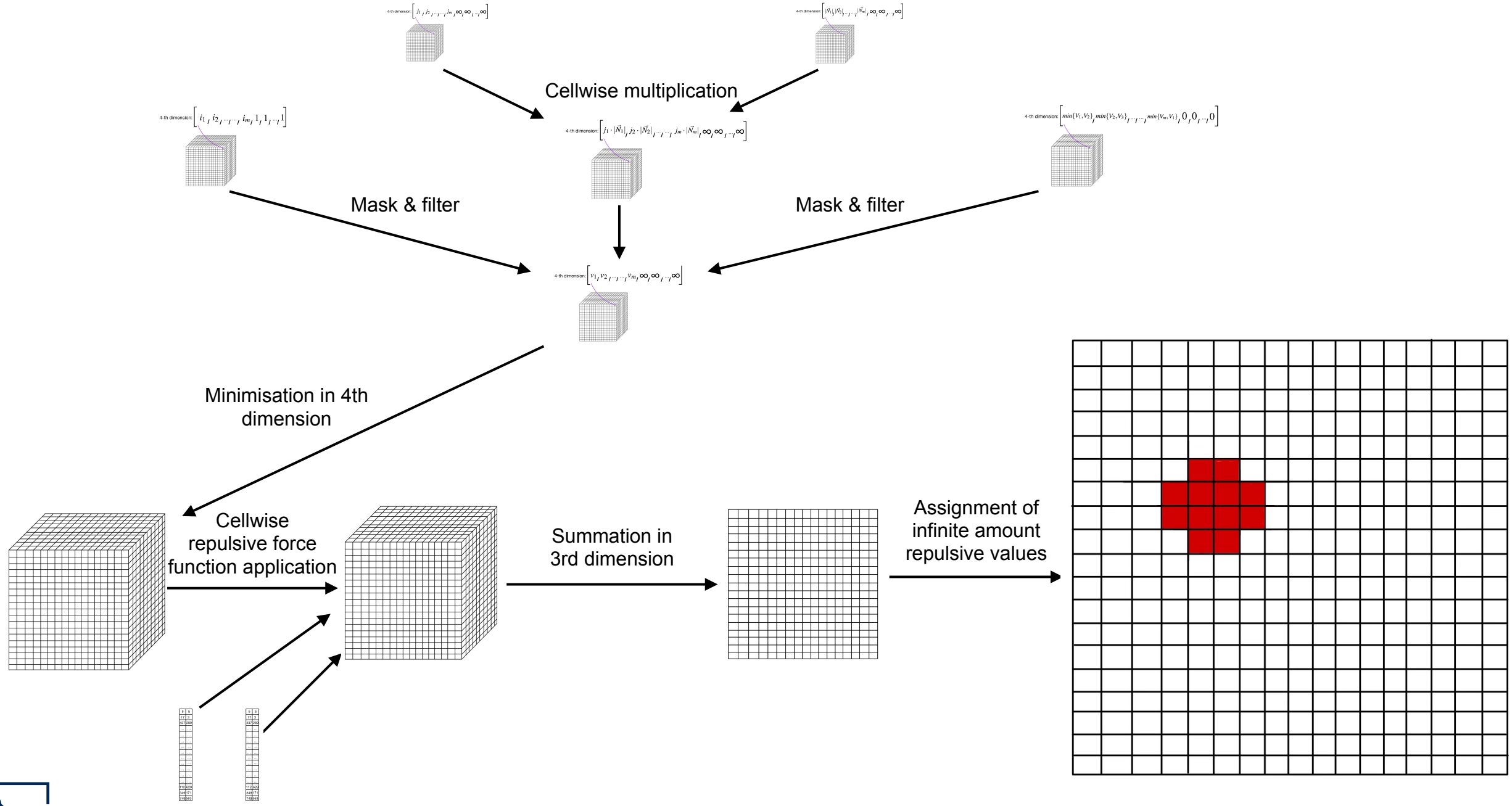
Minimisation in 4th dimension

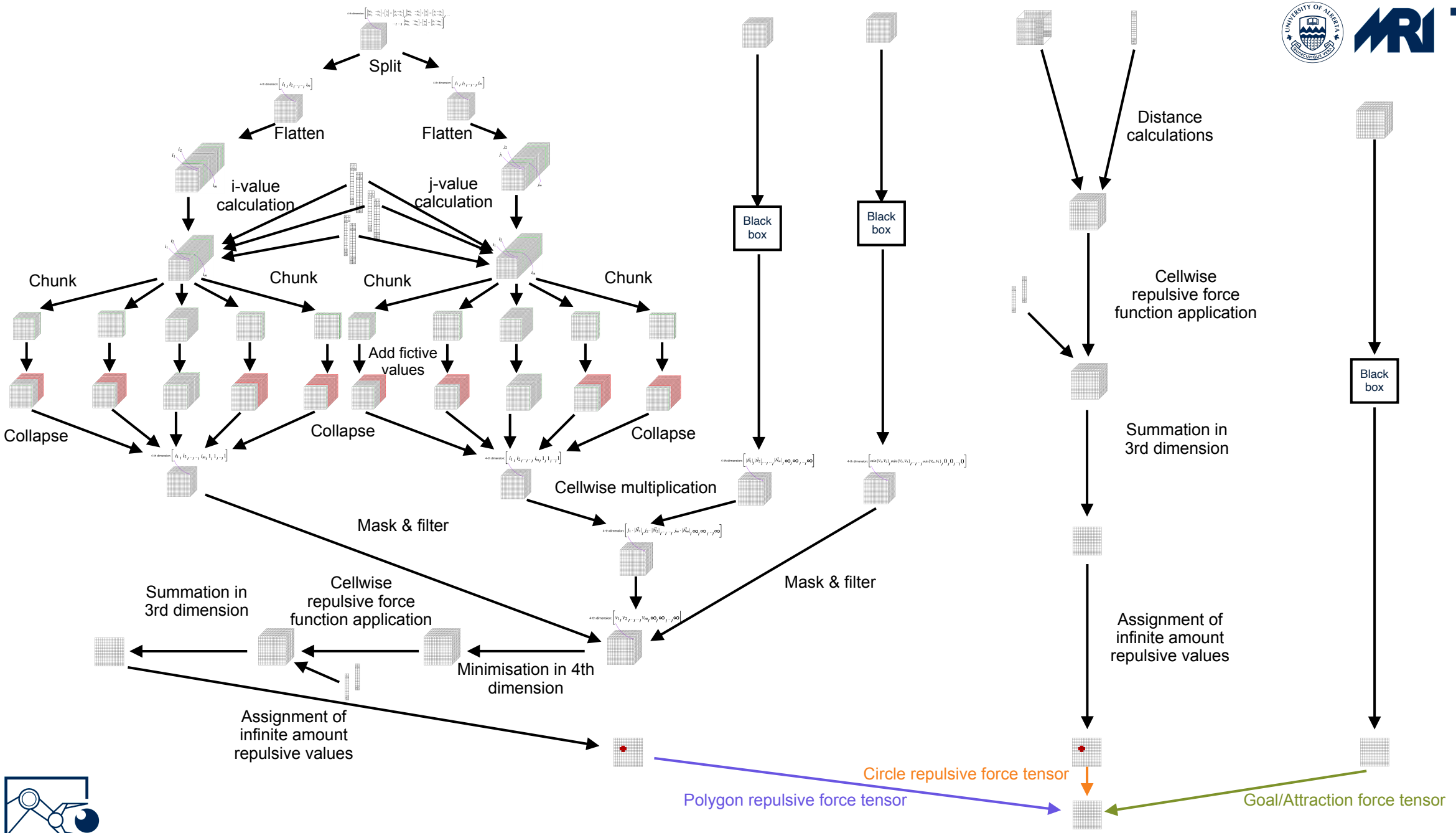


“Distance tensor for Polygons”


Cellwise multiplication
Mask & filter
Mask & filter
Minimisation in 4th dimension
Cellwise repulsive force function application

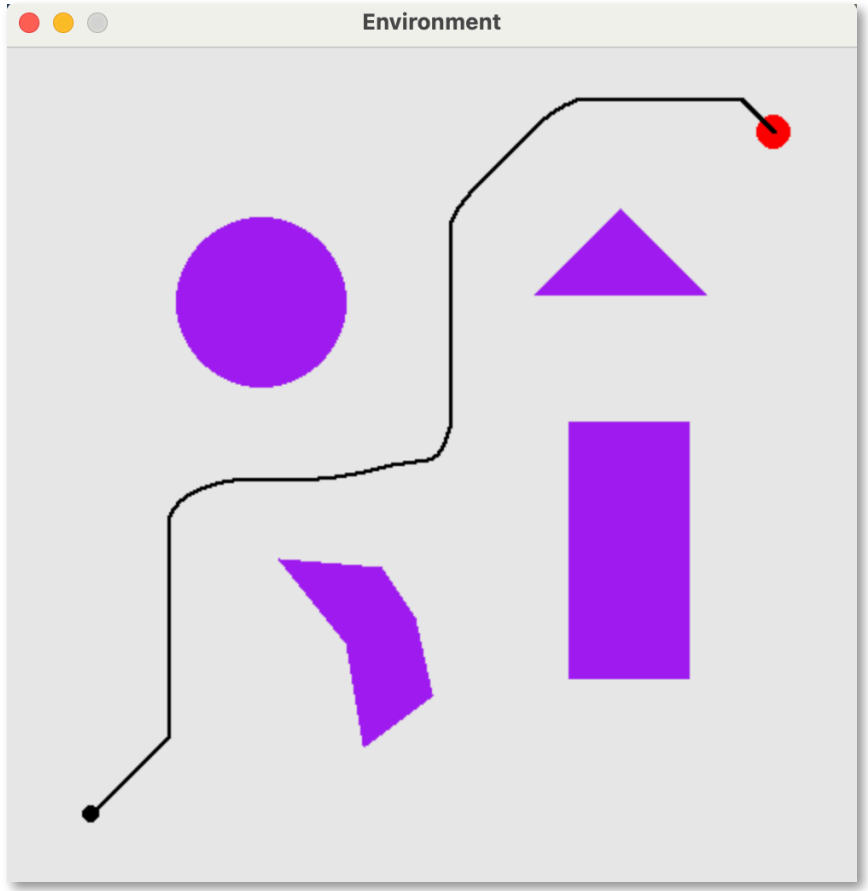

Cellwise multiplication
Mask & filter
Mask & filter
Minimisation in 4th dimension
Cellwise repulsive force function application
Summation in 3rd dimension

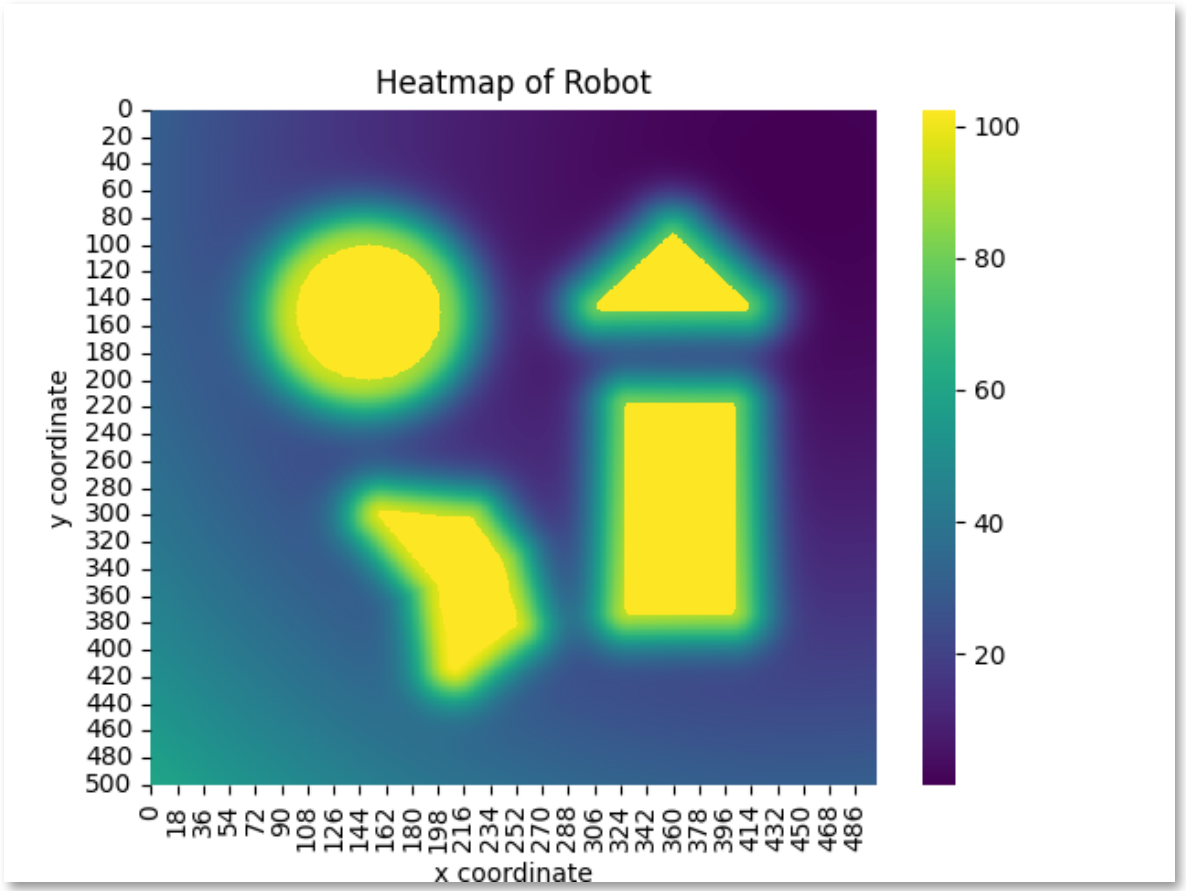

Cellwise multiplication
Mask & filter
Mask & filter
Minimisation in 4th dimension
Cellwise repulsive force function application
Summation in 3rd dimension
Assignment of infinite amount repulsive values


Split
Flatten
Flatten
i-value calculation
j-value calculation
Chunk
Chunk
Chunk
Chunk
Add fictive values
Collapse
Collapse
Collapse
Black box
Black box
Distance calculations
Cellwise repulsive force function application
Summation in 3rd dimension
Assignment of infinite amount repulsive values
Black box
Cellwise multiplication
Mask & filter
Mask & filter
Summation in 3rd dimension
Cellwise repulsive force function application
Minimisation in 4th dimension
Assignment of infinite amount repulsive values
Circle repulsive force tensor
Polygon repulsive force tensor
Goal/Attraction force tensor

Examples

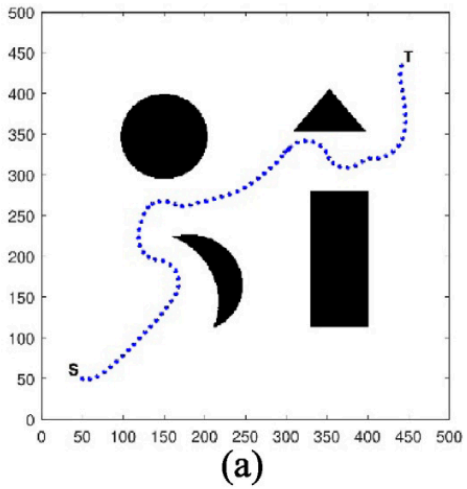
Examples

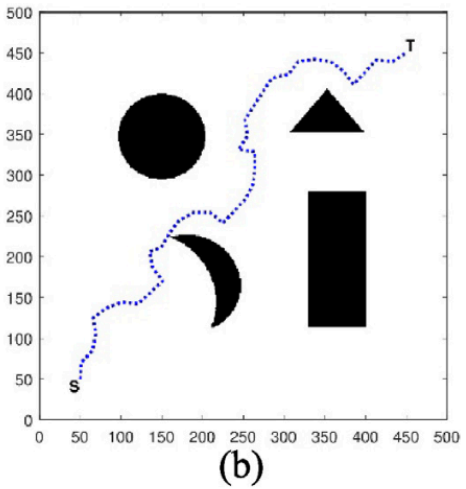


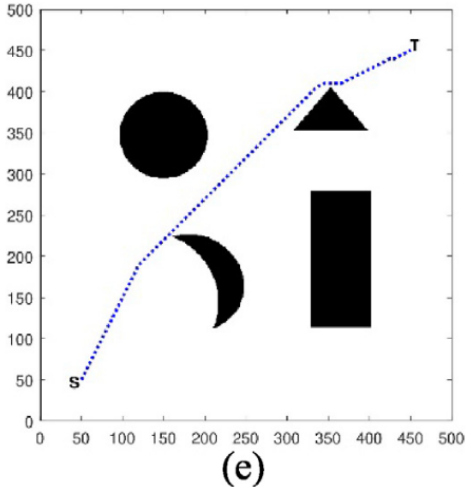


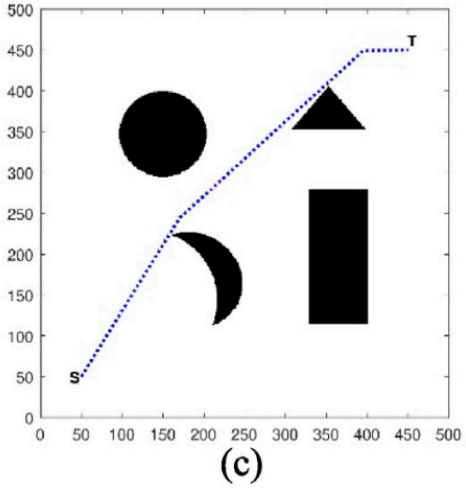
Computation time: ≈0.87s

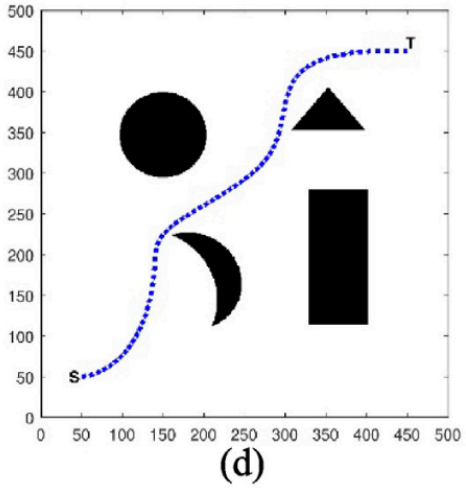
Examples

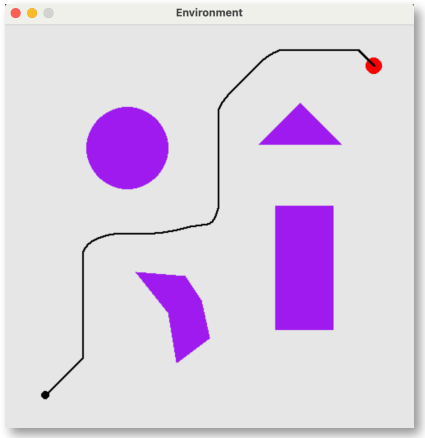






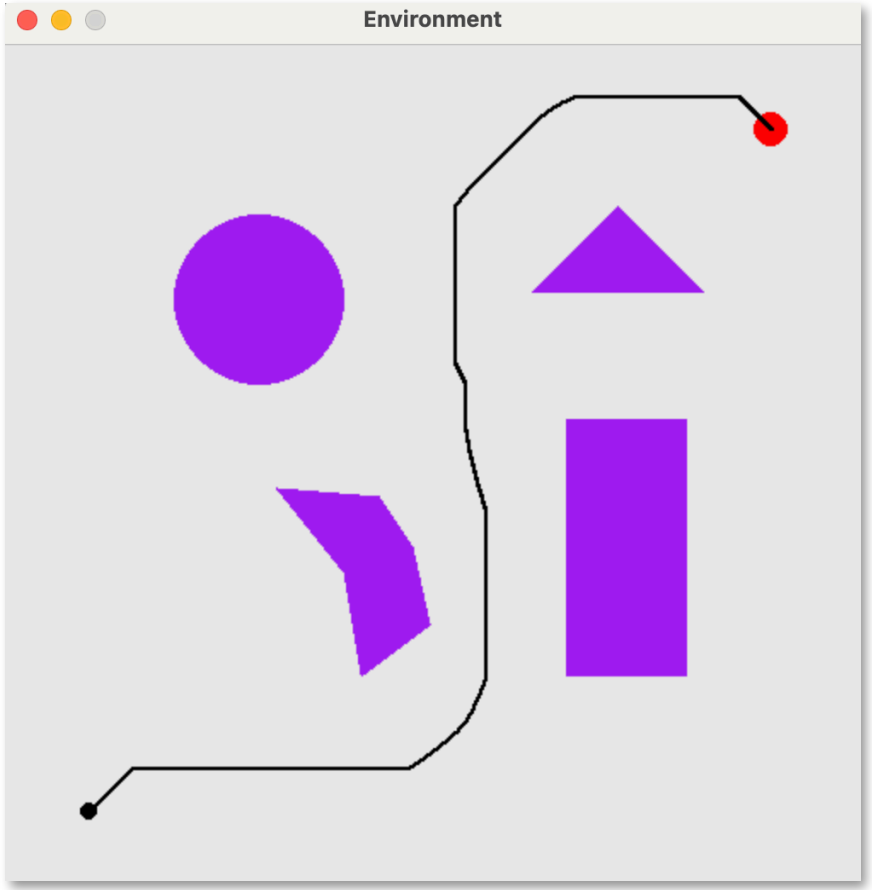






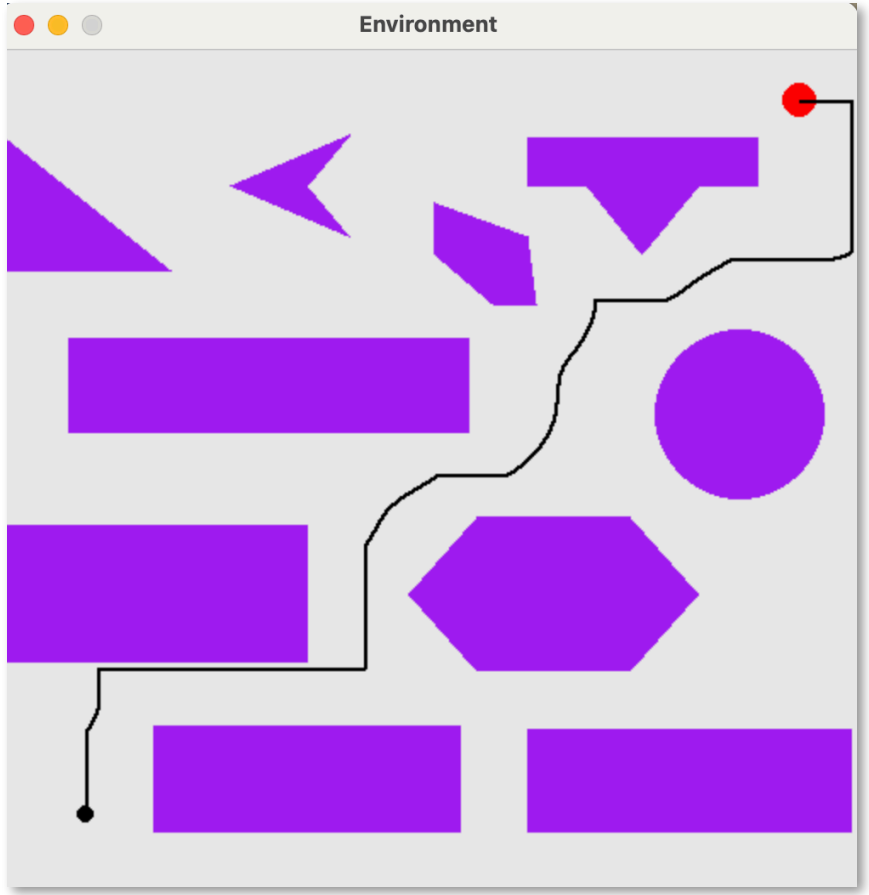
A (DA-APF)	1.90s
B (B-RRT)	2.89s
C (PRM)	4.56s
D (GA)	1.50s
E (5.98)	5.98s
Ours	0.87s

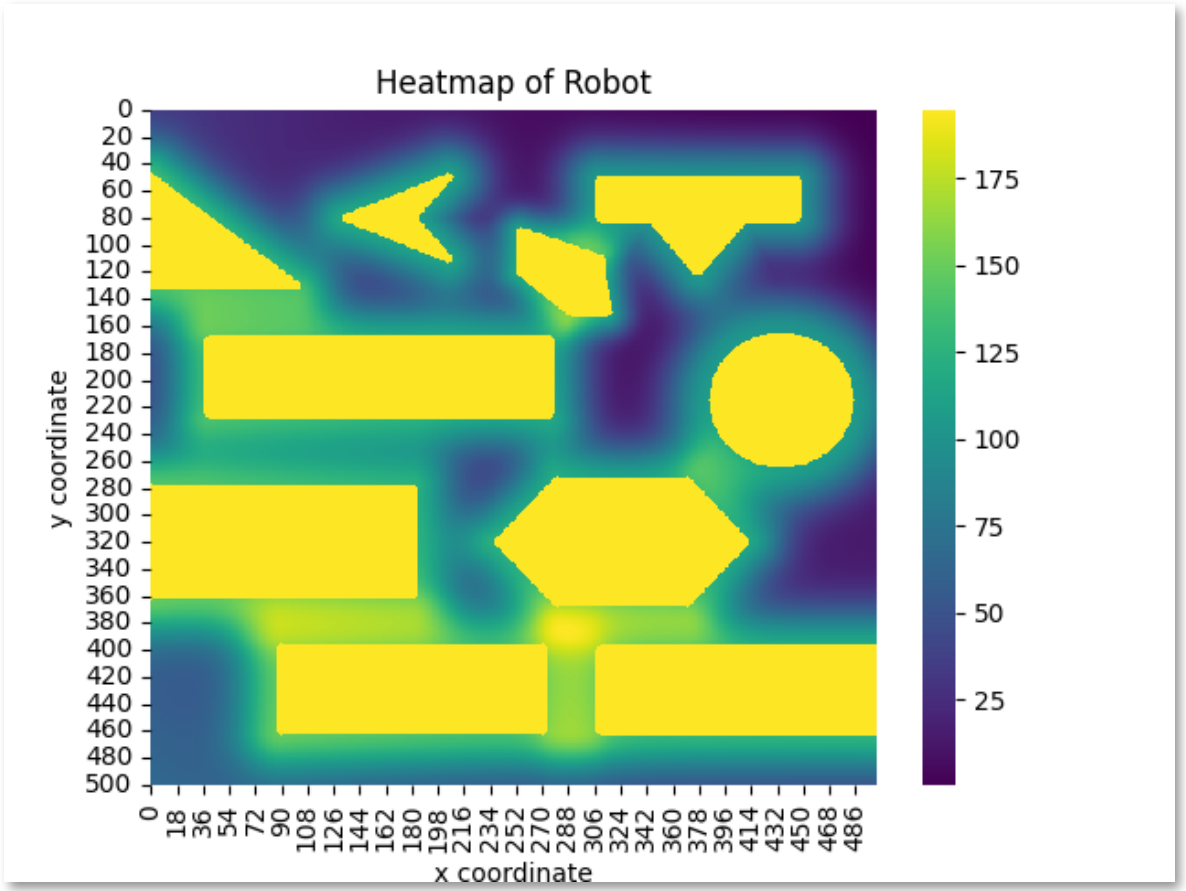
Examples



Computation time: ≈0.88s

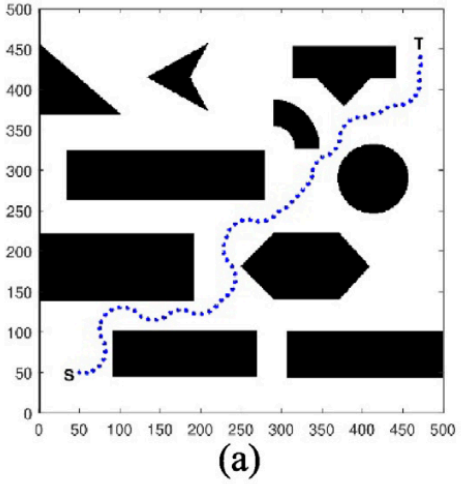
Examples

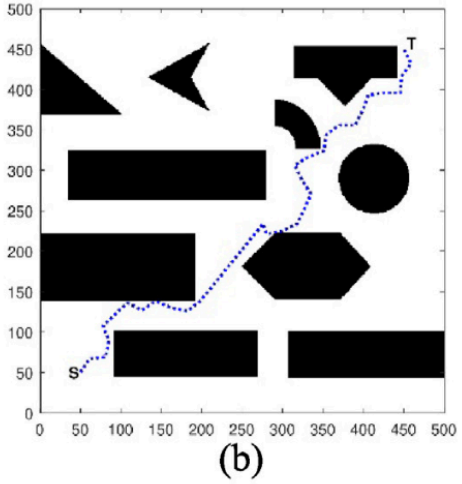


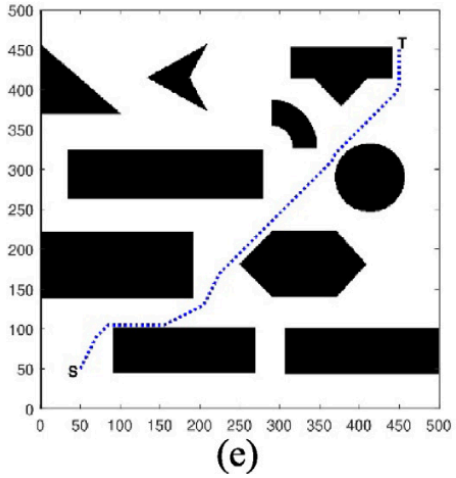


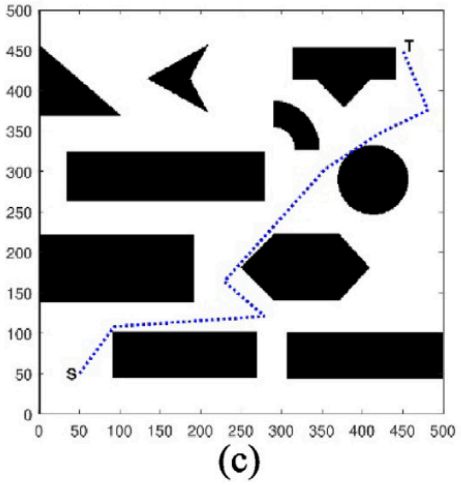
Computation time: ≈1.26s

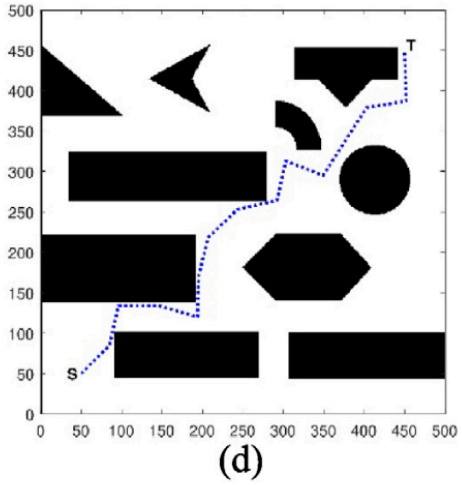
Examples

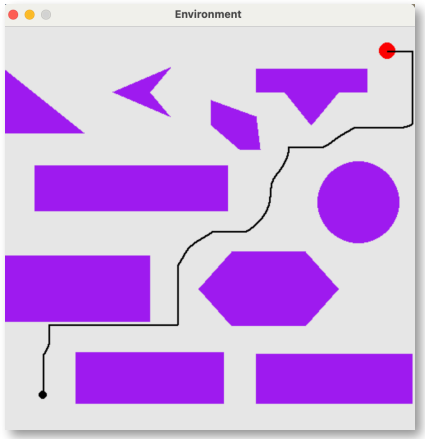






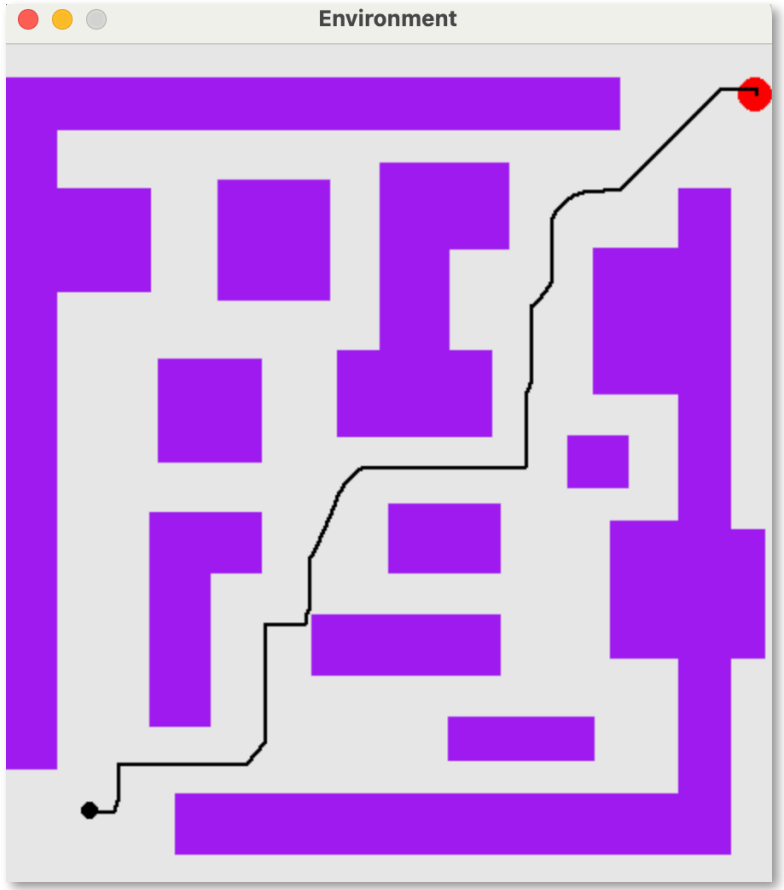


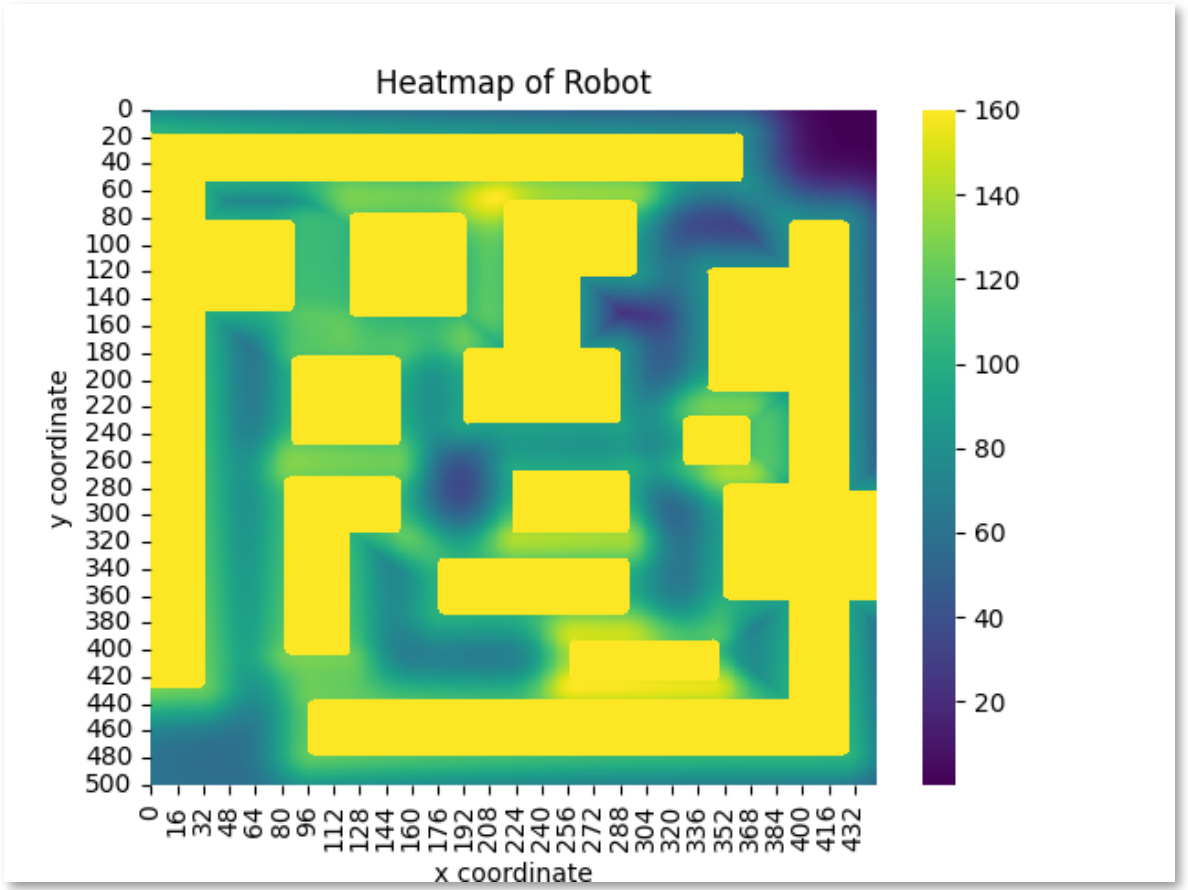




A (DA-APF)	1.75s
B (B-RRT)	2.58s
C (PRM)	1.88s
D (GA)	1.54s
E (5.98)	4.56s
Ours	1.26s

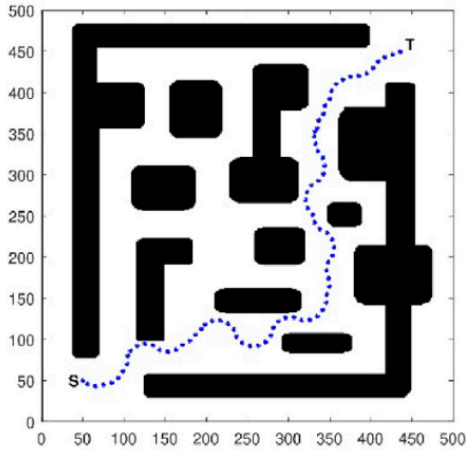
Examples

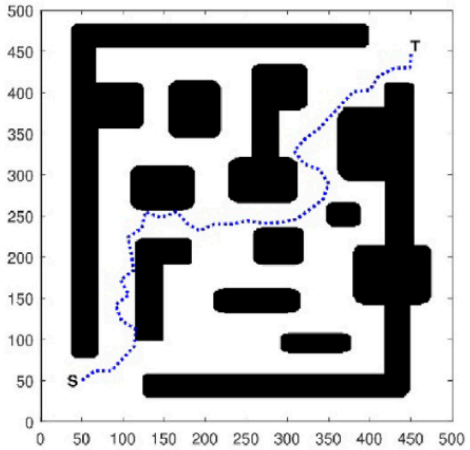


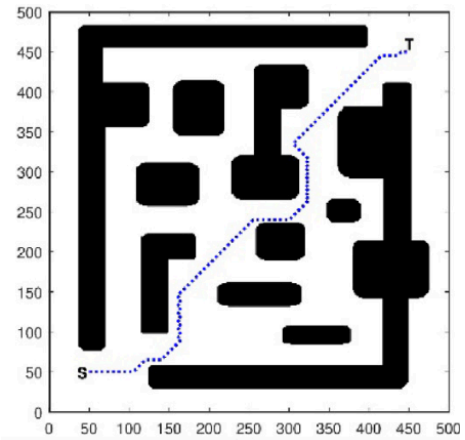


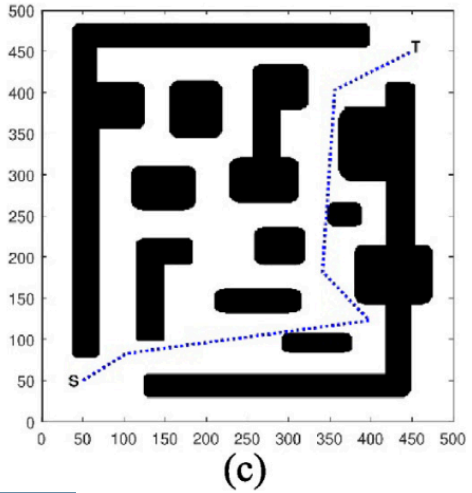
Computation time: ≈2.47s

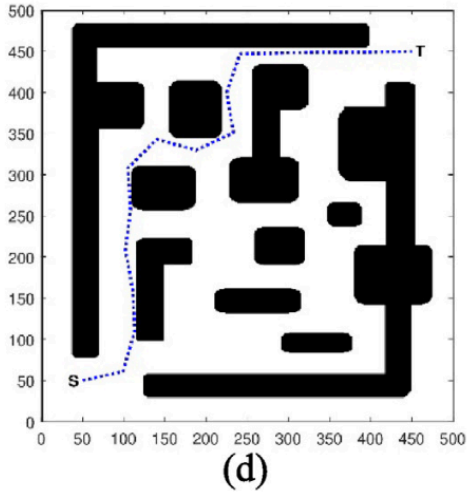
Examples

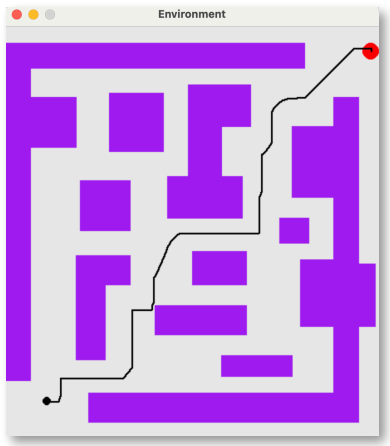












A (DA-APF)	1.90s
B (B-RRT)	2.80s
C (PRM)	2.61s
D (GA)	2.13s
E (5.98)	2.98s
Ours	2.47s

Questions?

References

[1]: Baoping Jiang Zhengtian Wua, Jinyu Dai and Hamid Reza Karimi. Robot path planning based on artificial potential field with deterministic annealing. 2022.