VISUALIZATION OF CIRCLE COLLISIONS USING QUAD- AND OCTA-TREES

HPC-PROJECT

 \mathbf{O}

BY ELIAH AND EVGENI

PRESENTATION OUTLINE

 \bigcap



 $\int O$

MOTIVATION

- Physics Simulations
- Swarm Robotics
 - Drones, Cars
- Computer Games
- It's fun

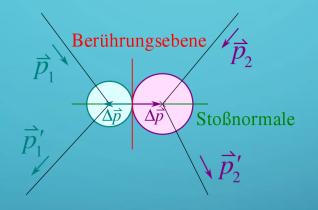


PROBLEM DESCRIPTION



2D INSTEAD OF 3D

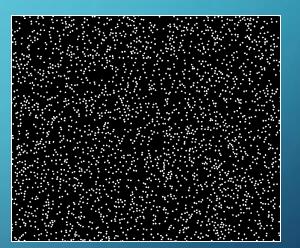
We reduced the dimensions to 2



COLLISIONS

Check the collision between circles and with the wall

All circles have the same size and same mass



THE AMOUNT OF CIRCLES

The amount of cirlces increases the amount of collision checks drastically

APPROACHES

 \mathcal{O}

 \bigcirc

Ó

 \bigcirc

0

 \bigcirc

Q

 \bigcirc

0

 \square

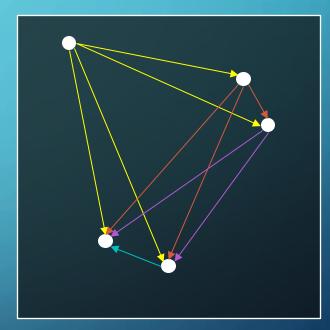
0

NAÏVE IMPLEMENTATION

Main Loop

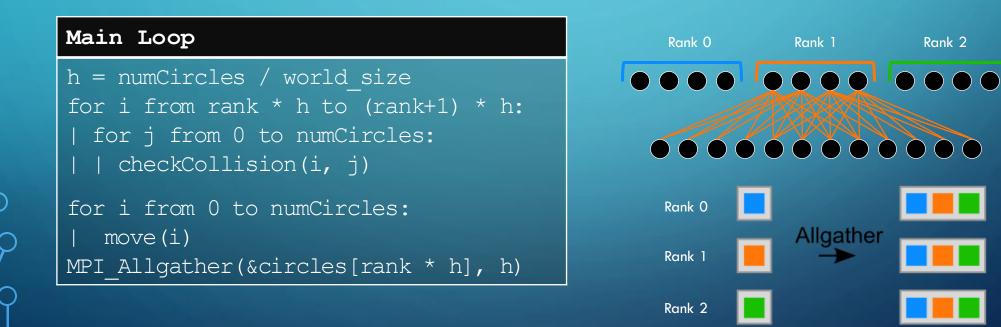
Check Collision

if distance(circle1, circle2) < 2r:
calcVelocities(circle1, circle2)
resolveOverlap(circle1, circle2)</pre>



NAÏVE – WITH MPI

- Parallelize the outer main loop
- Each process only calculates the collisions for its own circles



NAÏVE – WORKLOAD DISTRIBUTION

	0.0s	0.1s	0.2s	0.3s		0.4s		C).5s			0.6s			0.7s		
		4	1	E		1	_		1			4			1		
Master thread:0	MPI_Init				1		1	0	0	0	0	0	0	0	0	0	۲
Master thread:1	MPI_Init				9 9	8	1	0	٢	•	Ø	٢	0	0	٥	٥.	۰.
Master thread:2	MPI_Init				0.0		4	0	0	0	œh	ecoCo	lisions	0	0	۲	۲
Master thread:3	MPI_Init				• •	1	4	0	۲	0	Ó.	0	۲	0	0	0	\diamond
		8	1	E .					1			3.					-01

					nmary		
Accumulated	Exclusive Tim	e per Function	1				
35s	30s	25s	20s	15s	10s	5s	Os
37.384s							checkCollisions
						1.401	Ls MPI_Init
						1.0	5s main
						0.6	09s move
						0.5	22s MPI_Allgather
						0.3	383s checkPosition
						47.21	.8 ms TRACE BUFFER FLUSH
						40.8	2 ms MPI_Bcast
						7.07	1 ms random_double
						24.6	13 μs MPI_Comm_size
						9.7	16 µs MPI_Comm_rank

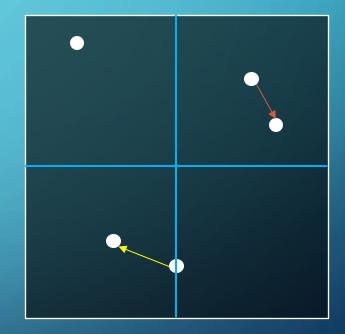
• Most of the calculation is in the checkCollisions function

 Communication is only a small overhead

THE TREE

- Split field in cells
- Assign circles to cells
- Check collisions in every cell

Benefits	Drawbacks
Less collision checks	Tree creation



STATIC TREE

Add Circle To Tree (circle, cell)

```
if cell.isLeaf:
```

```
if cell.numCirclesInCell < 2:
```

else:

```
| split(cell)
```

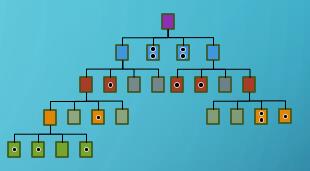
```
addCircleToTree(circle, cell)
```

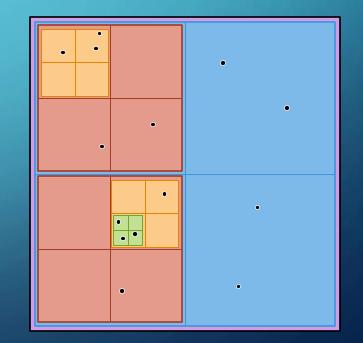
else:

```
for i from 0 to 4:
```

```
if circleOverCell(circle, subcell[i])
```

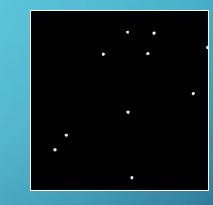
```
addCircleToTree(circle, subcell[i])
```

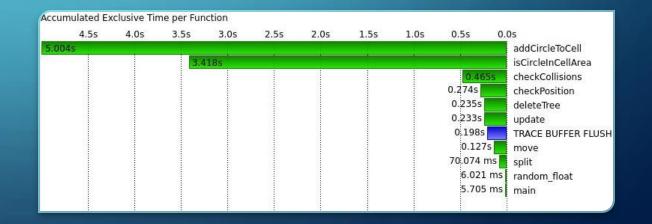




STATIC TREE - WORKLOAD DISTRIBUTION

- Main workload on tree building
- checkCollision needs much less time
- Wasteful to delete the tree and rebuild it for every frame





DYNAMIC TREE

The idea:

- Don't reconstruct the tree
- \rightarrow Reuse and update: split/collapse
- Don't "waste" perfectly fine subtrees

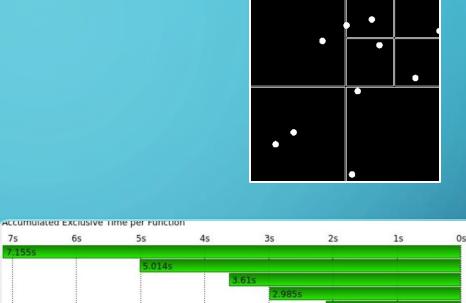
Problems:

• Implementation

- Recursive functions are harder to debug
- Exception thrown multiple iterations after error occured

DYNAMIC TREE -WORKLOAD DISTRIBUTION

- The dynamic functions are costing a lot of time
- Multiple tree traversals per frame
- There might be a better implementation we haven't found yet



5s

6s

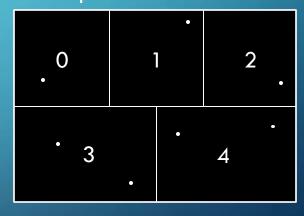
7s

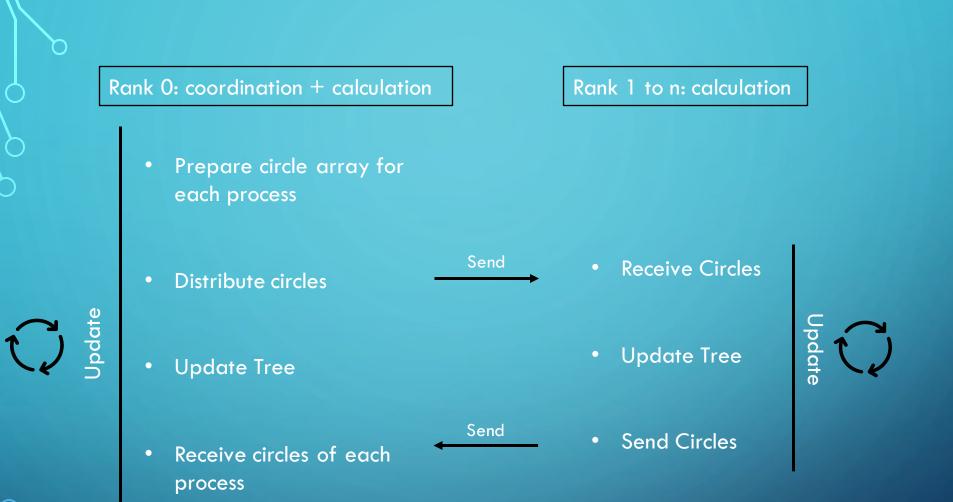
0s cellContainsCircle isCircleCloseToCellArea isCircleOverlappingCellArea addCircleToCell deleteCircle addCircleToParentCell isCircleFullInsideCellArea 772 updateCell 0.516s updateTree 0.505s checkCollisions 0.395 MPI Init 0.333s checkPosition 0.2995 split 0.294s move 0.2645 isCircleOverlappingArea 0.23s update 0.211s main 0.182s random double 84.208 ms setupQuadtree 55.921 ms collapse

MPI WITH TREES

- Split the window in multiple subfields
- Assign one to each process
- Every process manages a tree with dimensions of its subfield
- Process 0 coordinates circle distribution

Window subfields with process numbers





• Draw all circles

 \bigcirc

0

Ó

 \bigcirc

 \cap

PERFORMANCE COMPARISON

 \mathcal{O}

 \cap

Ó

 \mathbf{O}

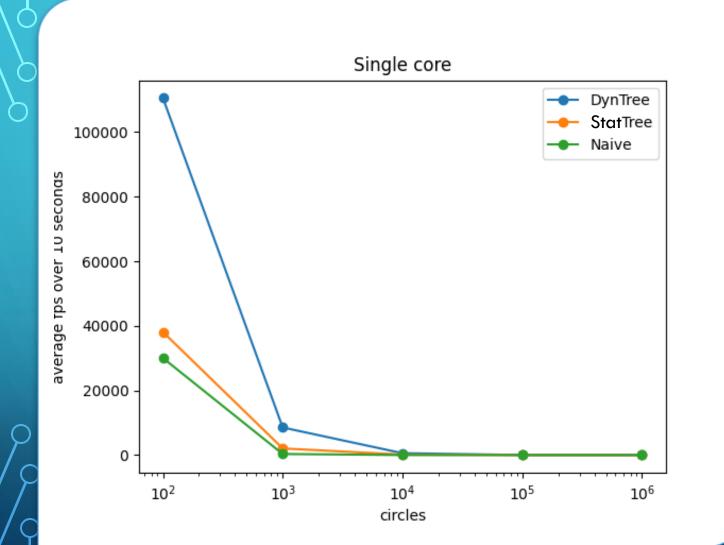
Ó

 \bigcap

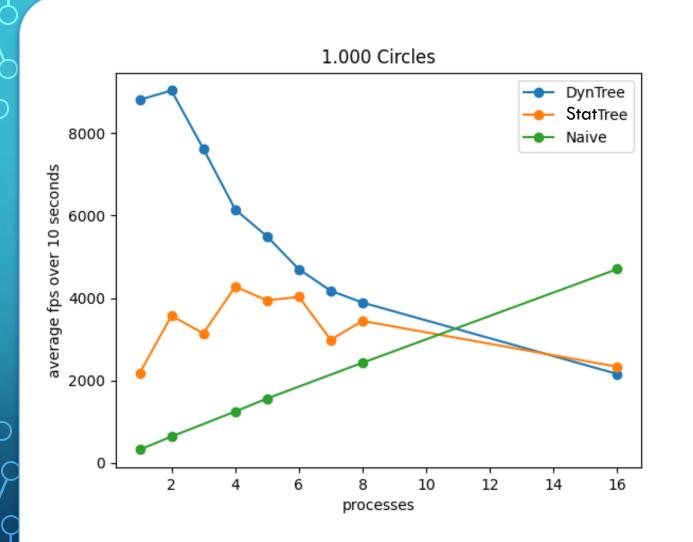
 \cap

 \bigcirc

 \bigcirc



- Naïve works slower than both tree approaches
- With more circles the fps improvement gets smaller
- Dynamic tree shows best results



- The naïve implementation scales nearly optimal
- Both trees do not scale
- The tree solutions get worse with more processes

THE PROBLEM (STATIC TREE)

Timeline															
	0 <u>,</u> s	1,s	2 <u>,</u> s	3,s	4 <u>,</u> s	5 <u>,</u> s	6 <u>,</u> s	7 <u>,</u> s	8 <u>,</u> s	9 <u>,</u> s	10s	11s	12s	13s	14s
	A REAL PROPERTY.	1			1		1		1		1	0	1		1
Master thread:0	MPI_Init	•				۲		۲		۲		0		۲	\odot
Master thread:1	MPI_Init	MPI_Recv		MPI_Re		♦● MPI_Recv		MPI_Recv		MPI_Recv		MPI_Recv		MPI_Recv	۲
Master thread:2	MPI_Init	MPI_Recv		MPI_Re	ev	♦●MPI_Recv	15	♦● MPI_Recv		MPI_Recv	1	OCMPI_Recv		♦●MPI_Recv	۲
Master thread:3	MPI_Init	MPI_Recv		OMPI_Re	cv	♦●MPI_Recv		♦● MPI_Recv		MPI_Recv		MPI_Recv		♦●MPI_Recv	۲
Master thread:4	MPI_Init	MPI_Recv		MPI_Re	CV	OMPI_Recv	14	◇ MPI_Recv		MPI_Recv	14	♦ MPI_Recv		MPI_Recv	۲
Master thread:5	MPI_Init	MPI_Recv		OMPI_Re	ev	OMPI_Recv		MPI_Recv		MPI_Recv		MPI_Recv		♦● MPI_Recv	۲
Master thread:6	MPI_Init	MPI_Recv	_	MPI_Re	cv	MPI_Recv		NPI_Recv		MPI_Recv	1	MPI_Recv		◇ MPI_Recv	۲
Master thread:7	MPI_Init	MPI_Recv		OMPI_Re	CV	♦ MPI_Recv	- North	OMPI_Recv		MPI_Recv		MPI_Recv	-	♦● MPI_Recv	۲
Master thread:8	MPI_Init	MPI_Recv	_	MPI_Re	cv	♦ MPI_Recv		oo MPI_Recv		MPI_Recv	-	OOMPI_Recv	_	OMPI_Recv	۲
Master thread:9	MPI_Init	MPI_Recv	_	OMPI_Re	CV	♦ OMPI_Recv		♦● MPI_Recv	1	MPI_Recv		OCMPI_Recv		♦●MPI_Recv	۲
Master thread:10	MPI_Init	MPI_Recv		OMPI_Re	CV	♦●MPI_Recv		♦ MPI_Recv		MPI_Recv		MPI_Recv		♦● MPI_Recv	۲
Master thread:11	MPI_Init	MPI_Recv		MPI_Re	cv	OMPI_Recv		MPI_Recv		MPI_Recv		OMPI_Recv		MPI_Recv	٠
Master thread:12	MPI_Init	MPI_Recv	_	OMPI_Re	CV	MPI_Recv		MPI_Recv	-	MPI_Recv	-	OOMPI_Recv	_	MPI_Recv	۲
Master thread:13	MPI_Init	MPI_Recv	-	•MPI_Re	¢ν	♦ MPI_Recv		ONPI_Recv		MPI_Recv	1	MPI_Recv		●● MPI_Recv	۲
Master thread:14	MPI_Init	MPI_Recv		OMPI_Re	cv	OMPI_Recv		ONPI_Recv		MPI_Recv		OOMPI_Recv		O MPI_Recv	۲
Master thread:15	MPI_Init	MPI_Recv		OMPI_Re	cv	♦ MPI_Recv		OMPI_Recv		MPI_Recv		OOMPI_Recv		OMPI_Recv	۲

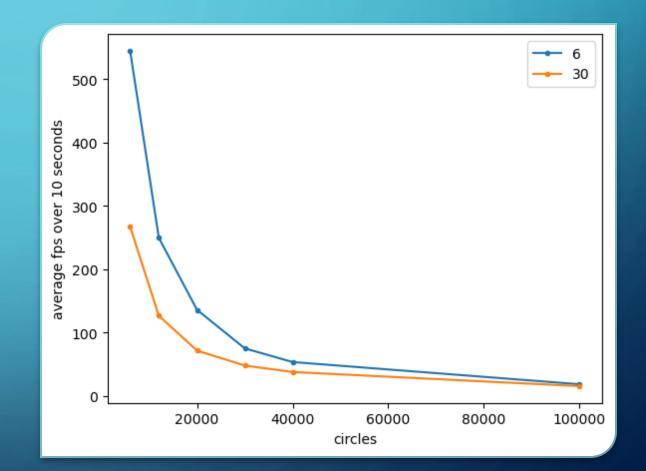
t sector contractor		100 M (100 M (10		Function S	Summary		
ccumulated Ex	clusive Time	per Function					
175s	150s	125s	100s	75s	50s	25s	Os
190.389s							MPI_Recv
						13.422s	MPI_Init
						6.413s	isCircleOverlappingArea
						6.022s	distributeCircles
						5.576s	addCircleToCell
						4.583s	isCircleOverlappingCellArea
						0.62	
						0.468	s checkCollision
						0.394	s cellContainsCircle
						0.315	

- All processes are waiting until process 0 distributes updated circles
- The overhead for process 0 is bigger than the time advantage we get with less collision checks

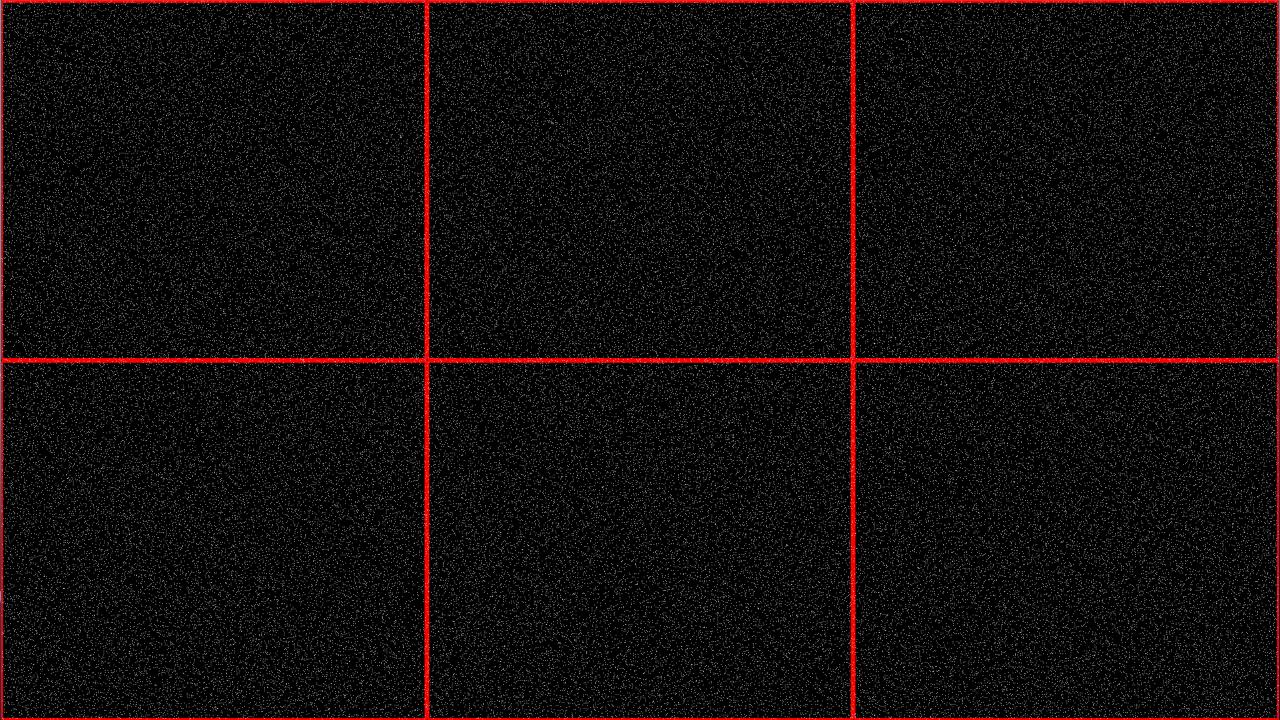
• Only if number of circles is low

STATIC TREE

- static tree approach works better with less processes
- Even if we increase the amount of circles
- At around 100k circles we get a turn point
- The collision check gets more complex compared to the work process 0 gets



Performance comparison



CONCLUSION

Naïve:

- Gets slow with growing circle amounts
- Scales nearly optimal with increasing process size

Trees:

- Trees work way better in single process applications
- MPI is harder to implement and scales worse

 Should be used if you have a high amount of processes Should be used in single or with low amount of processes

OUTLOOK

- Different circle sizes and differente masses
- Find a better dynamic tree implmentation
- Find a better way of circles coordination (direct messaging)
- Find a better implementation where processes get a circle range instead of subfields
 - Reduces message, comparable with the naïve messaging
 - Scales better if circles are not equally distributed (e.g. gravity)