FACULTY OF SCIENCES

DEPARTMENT OF SOLID STATE SCIENCES

DYNAMAT GROUP

MUMAX3-WORKSHOP SESSION 4

Dr. Jonathan Leliaert,
Dr. Jeroen Mulkers

GHENT UNIVERSITY

IEEE MAGNETICS

NVIDIA

CSPINTRONICS

SCHEDULE

Monday 08/31, 6PM-8PM CET

Session 1: general introduction to micromagnetics in mumax3

Session 2: mumax3 ecosystem, workflow and a first simulation

Monday 09/07, 6PM-7:30PM CET

Session 3: basic examples

>homework

Monday 09/14, 6PM-7:30PM CET

Session 4: advanced features and more extensive examples

GHENT UNIVERSITY

TABLE OF CONTENTS

GHENT UNIVERSITY

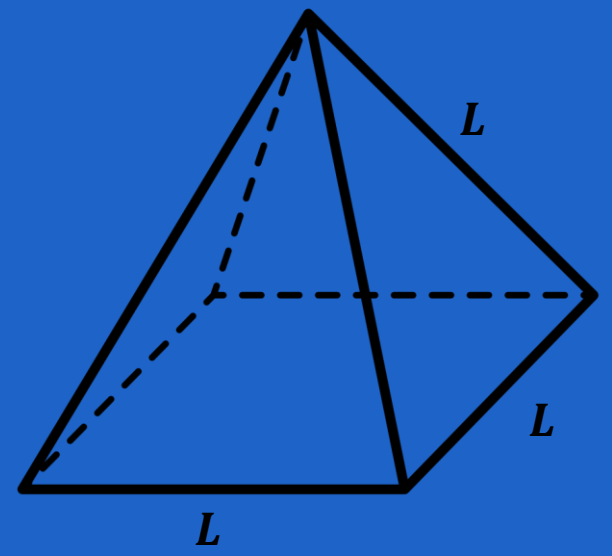
HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID

Consider a ferromagnetic equilateral square pyramid with the following material parameters

- Exchange stiffness $A = 13\ \mathrm{pJ/m}$
- Saturation magnetization $M_{sat} = 800\ \mathrm{kA/m}$

Find the magnetic ground state of this structure for different side lengths L ranging from 50 nm to 100 nm.

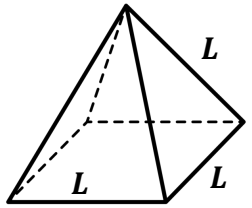


Hint: this is exercise is very similar to standard problem 3

GHENT UNIVERSITY

HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID



Side length: L

Pyramid height: $h = \frac{L}{\sqrt{2}}$

Simulation box size: $L \times L \times h$

Simulation grid size: $N_x \times N_x \times N_z$

Defining the grid size (powers of 2)

```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)
```

Max cell size: $\left(\frac{L_{max}}{N_x}, \frac{L_{max}}{N_x}, \frac{h_{max}}{N_z}\right)$

$= (1.56\ \text{nm}, 1.56\ \text{nm}, 2.21\text{nm})$ ✓

Uniform material parameters

```
Msat = 800e3
Aex  = 13e-12
```

GHENT UNIVERSITY

```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID

Two stable states (similar to ferromagnetic cube):

- Vortex state
- Flower state (quasi uniform)

Don’t know which states are stable? → Relax multiple random configurations

Output scheduling

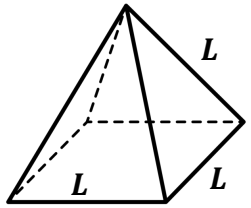
```
Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")
```

```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

GHENT UNIVERSITY

HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID



Side length: L

Pyramid height: $h = \frac{L}{\sqrt{2}}$

Simulation box size: $L \times L \times h$

Simulation grid size: $N_x \times N_x \times N_z$

Simulation cell size: $\frac{L}{N_x} \times \frac{L}{N_x} \times \frac{h}{N_z}$

Dihedral angle: $\arctan(\sqrt{2})$

Gradually increase the pyramid side length L by changing the cell size:

```
for L=50e-9 ; L<=100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx, L/Nx, height/Nz)

    …
}
```

```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

GHENT UNIVERSITY

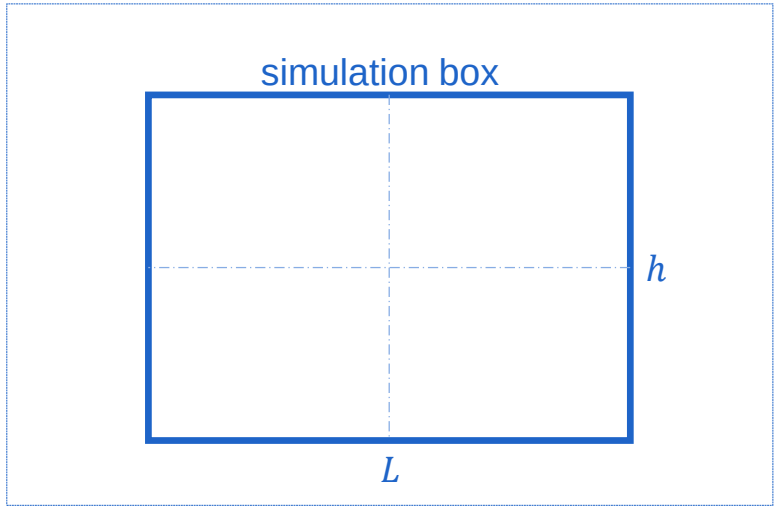
HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID

Creating a pyramid shape (with the top in the center)

```
s1 := zrange(-inf,0).rotY( angle)
s2 := zrange(-inf,0).rotY(-angle)
s3 := zrange(-inf,0).rotX( angle)
s4 := zrange(-inf,0).rotX(-angle)
pyramid := s1.intersect(s2).intersect(s3).intersect(s4)
```

Step-by-step pyramid construction



GHENT UNIVERSITY

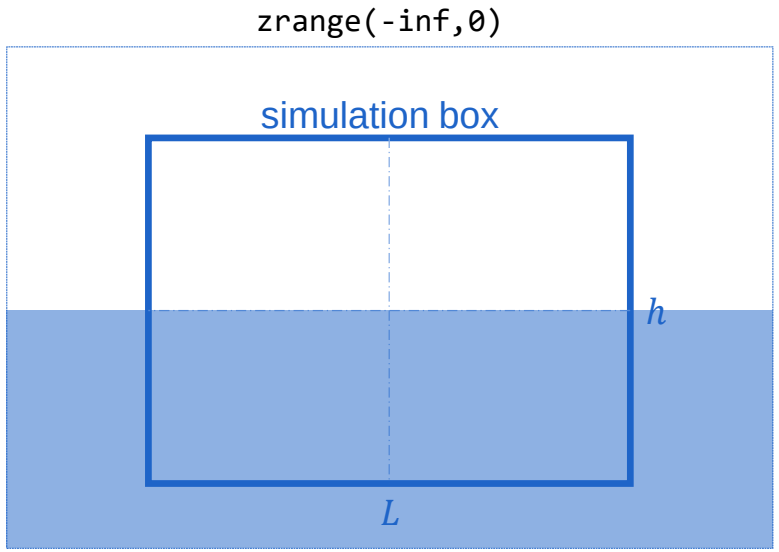
```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID

Creating a pyramid shape (with the top in the center)

```
s1 := zrange(-inf,0).rotY( angle)
s2 := zrange(-inf,0).rotY(-angle)
s3 := zrange(-inf,0).rotX( angle)
s4 := zrange(-inf,0).rotX(-angle)
pyramid := s1.intersect(s2).intersect(s3).intersect(s4)
```



GHENT UNIVERSITY

```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

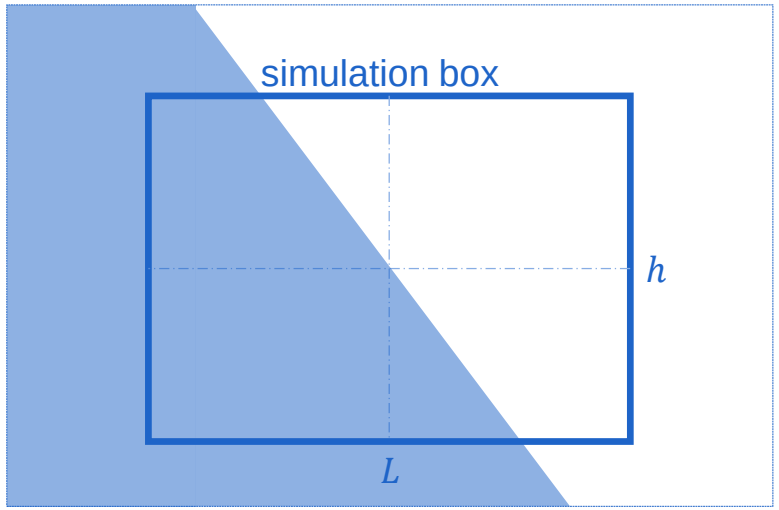
HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID

Creating a pyramid shape (with the top in the center)

```
s1 := zrange(-inf,0).rotY( angle)
s2 := zrange(-inf,0).rotY(-angle)
s3 := zrange(-inf,0).rotX( angle)
s4 := zrange(-inf,0).rotX(-angle)
pyramid := s1.intersect(s2).intersect(s3).intersect(s4)
```

s1 := zrange(-inf,0).rotY(angle)



GHENT UNIVERSITY

```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

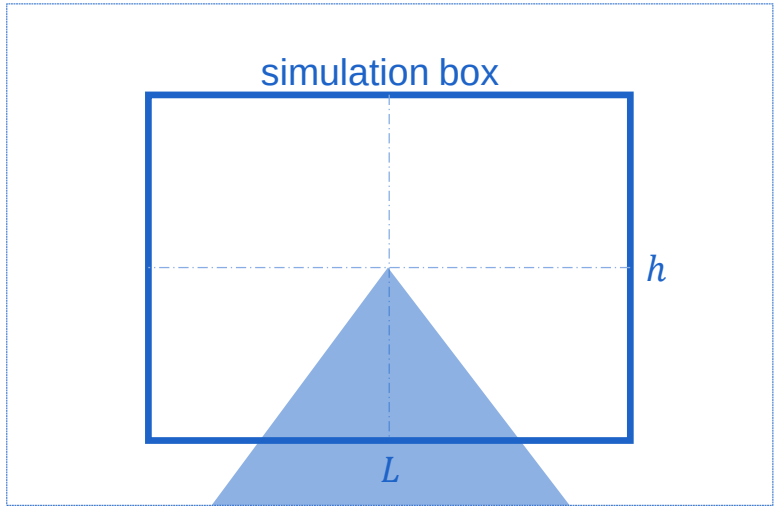
HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID

Creating a pyramid shape (with the top in the center)

```
s1 := zrange(-inf,0).rotY( angle)
s2 := zrange(-inf,0).rotY(-angle)
s3 := zrange(-inf,0).rotX( angle)
s4 := zrange(-inf,0).rotX(-angle)
pyramid := s1.intersect(s2).intersect(s3).intersect(s4)
```

```
s1 := zrange(-inf,0).rotY( angle)
s2 := zrange(-inf,0).rotY(-angle)
s1.intersect(s2)
```



GHENT UNIVERSITY

```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

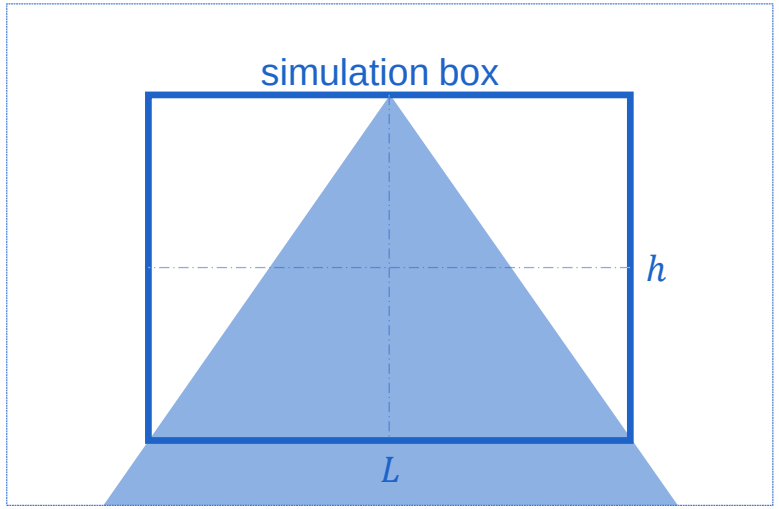
HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID

Moving the pyramid up to fit in the simulation box
(+ a fraction of the cell size to avoid rounding errors)

```
dz := height/2 + 0.1*height/Nz
pyramid = pyramid.transl(0,0,dz)

setgeom(pyramid)
```

```
pyramid = pyramid.transl(0,0,height/2)
```



GHENT UNIVERSITY

```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID

Relax the uniform state and compute the energy

```
m = uniform(1,0,1)
minimize()
Eflower = E_total.get()
```

Relax the vortex state and compute the energy

```
m = vortex(1,-1)
minimize()
Evortex = E_total.get()
```

Write the results to the table

```
tablesave()
```

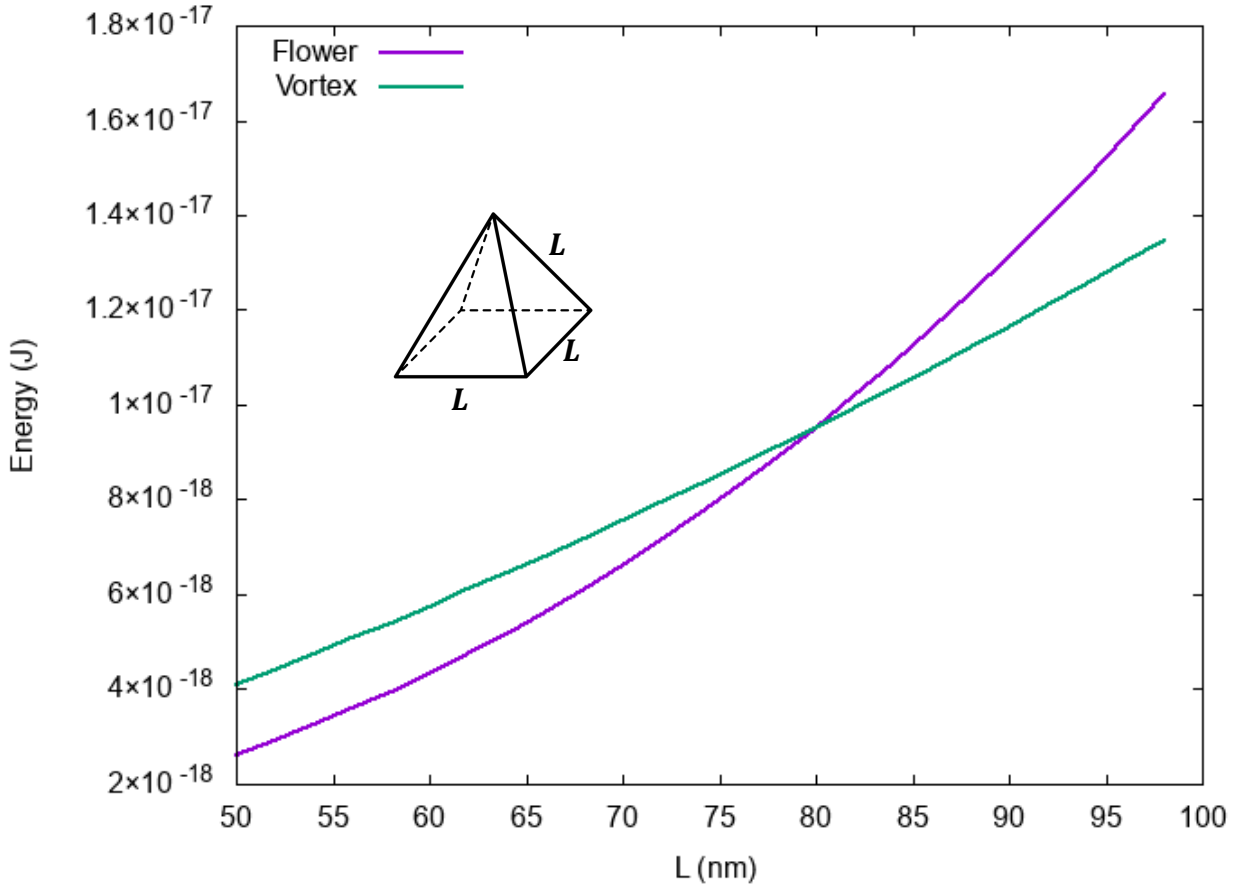
GHENT UNIVERSITY

```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

HOMEWORK ASSIGNMENT 1

THE FERROMAGNETIC PYRAMID

Table.txt:




```
Nx := 64
Nz := 32
setgridsize(Nx,Nx,Nz)

Msat = 800e3
Aex  = 13e-12

Eflower := 0.0
Evortex := 0.0
L := 0.0
tableAddVar(L,"L","m")
tableAddVar(Eflower,"Eflower","J")
tableAddVar(Evortex,"Evortex","J")

for L=50e-9 ; L <= 100e-9; L+=2e-9 {

    height := L/sqrt(2)       // pyramid height
    angle  := atan(sqrt(2))   // dihedral angle

    setcellsize(L/Nx,L/Nx,height/Nz)

    s1 := zrange(-inf,0).rotY( angle)
    s2 := zrange(-inf,0).rotY(-angle)
    s3 := zrange(-inf,0).rotX( angle)
    s4 := zrange(-inf,0).rotX(-angle)
    pyramid := s1.intersect(s2).intersect(s3).intersect(s4)

    dz := height/2 + 0.1*height/Nz
    pyramid = pyramid.transl(0,0,dz)

    setgeom(pyramid)

    m = uniform(1,0,1)
    minimize()
    Eflower = E_total.get()

    m = vortex(1,-1)
    minimize()
    Evortex = E_total.get()

    tablesave()
}
```

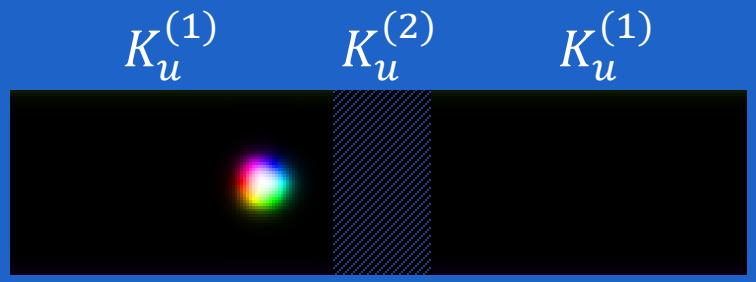
GHENT UNIVERSITY

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

Goal: use a local voltage controlled magnetic anisotropy (VCMA) to create a skyrmion gate on a skyrmion racetrack [1]

Use the input script of the skyrmion racetrack as a starting point
Add a region in the center in which the uniaxial anisotropy is increased by 5% (i.e. the VCMA)



$$K_u^{(2)} = K_u^{(1)} + K_u^{VCMA} = 1.05K_u^{(1)}$$

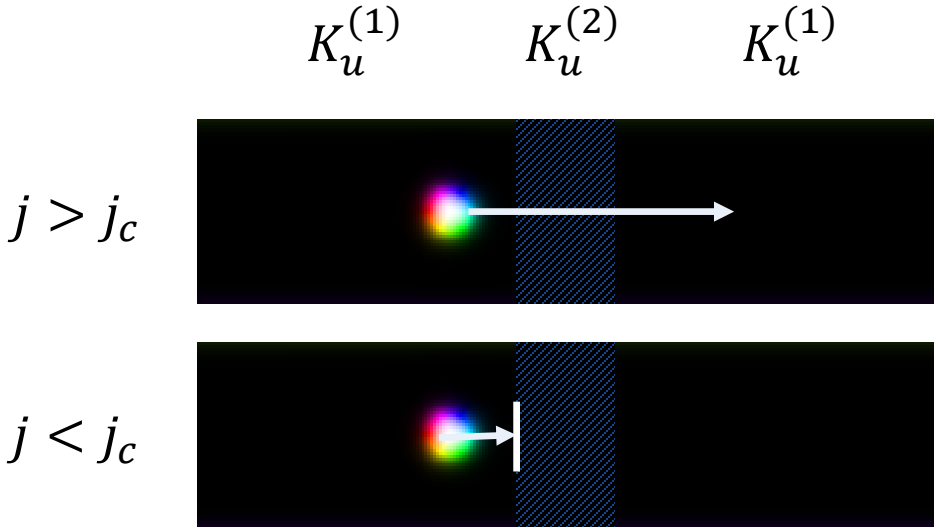
What is the maximal current density for which the region with the VCMA acts as a skyrmion gate?

GHENT UNIVERSITY

[1] **Voltage Controlled Magnetic Skyrmion Motion for Racetrack Memory**
Wang Kang, Yangqi Huang, Chentian Zheng, Weifeng Lv, Na Lei, Youguang Zhang, Xichao Zhang, Yan Zhou & Weisheng Zhao
Scientific reports **volume 6**, pages 23164 (2016)

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE



$$K_u^{(2)} = K_u^{(1)} + K_u^{VCMA} = 1.05K_u^{(1)}$$

Threshold current j_c?

GHENT UNIVERSITY

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

Building the skyrmion racetrack

(see session 3 of this workshop for more information)

```
setgridsize(128,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat  = 580e3
Aex   = 15e-12
Dind  = 3.0e-3
Ku1   = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1
Pol   = 0.4
xi    = 0.2
```

GHENT UNIVERSITY

```
setgridsize(128,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat  = 580e3
Aex   = 15e-12
Dind  = 3.0e-3
Ku1   = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1
Pol   = 0.4
xi    = 0.2

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()
saveas(m,"../m0")
flush()

defRegion(1,rect(50e-9,inf))
Ku1.setRegion(1, 1.05*Ku1.GetRegion(0))

cutofftime := 1e-8

for jx:=2e12; jx>0; jx-=1e11 {

    j = vector(-jx,0,0)
    print("Set current density:", jx, "A/m2")

    m.loadfile("m0.ovf")  // reset m to the initial state
    t = 0                 // reset the time

    runwhile( ext_bubblepos.Get().X()<0 && t<cutofftime)

    if (ext_bubblepos.Get().X() > 0 ) {
        print("Skyrmion got through after", t*1e9 ,"ns")
    } else {
        print("Skyrmion didn't get through!")
        exit()
    }

    cutofftime = 2*t  // adjust cut-off time
}
```

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

Relax a skyrmion and save this state
(we will need this later)

```
m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()
saveas(m,"../m0")
flush()
```

GHENT UNIVERSITY

```
setgridsize(128,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat  = 580e3
Aex   = 15e-12
Dind  = 3.0e-3
Ku1   = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1
Pol   = 0.4
xi    = 0.2

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()
saveas(m,"../m0")
flush()

defRegion(1,rect(50e-9,inf))
Ku1.setRegion(1, 1.05*Ku1.GetRegion(0))

cutofftime := 1e-8

for jx:=2e12; jx>0; jx-=1e11 {

    j = vector(-jx,0,0)
    print("Set current density:", jx, "A/m2")

    m.loadfile("m0.ovf")  // reset m to the initial state
    t = 0                 // reset the time

    runwhile( ext_bubblepos.Get().X()<0 && t<cutofftime)

    if (ext_bubblepos.Get().X() > 0 ) {
        print("Skyrmion got through after", t*1e9 ,"ns")
    } else {
        print("Skyrmion didn't get through!")
        exit()
    }

    cutofftime = 2*t  // adjust cut-off time
}
```

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

Simulate the VCMA effect by increasing the anisotropy strength in the center

```
defRegion(1,rect(50e-9,inf))
Ku1.setRegion(1, 1.05*Ku1.GetRegion(0))
```

Define the cut-off time

If the skyrmion did not get through the gate before the cut-off time, then it probably never will

```
cutofftime := 1e-8
```

GHENT UNIVERSITY

```
setgridsize(128,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat  = 580e3
Aex   = 15e-12
Dind  = 3.0e-3
Ku1   = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1
Pol   = 0.4
xi    = 0.2

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()
saveas(m,"../m0")
flush()

defRegion(1,rect(50e-9,inf))
Ku1.setRegion(1, 1.05*Ku1.GetRegion(0))

cutofftime := 1e-8

for jx:=2e12; jx>0; jx-=1e11 {

    j = vector(-jx,0,0)
    print("Set current density:", jx, "A/m2")

    m.loadfile("m0.ovf")  // reset m to the initial state
    t = 0                 // reset the time

    runwhile( ext_bubblepos.Get().X()<0 && t<cutofftime)

    if (ext_bubblepos.Get().X() > 0 ) {
        print("Skyrmion got through after", t*1e9 ,"ns")
    } else {
        print("Skyrmion didn't get through!")
        exit()
    }

    cutofftime = 2*t  // adjust cut-off time
}
```

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

Gradually decrease the current density

Reset the simulation state in each iteration

```
for jx:=2e12; jx>0; jx-=1e11 {

    j = vector(-jx,0,0)
    print("Set current density:", jx, "A/m2")

    m.loadfile("m0.ovf")  // reset m to the initial state
    t = 0                 // reset the time

    …
}
```

GHENT UNIVERSITY

```
setgridsize(128,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat  = 580e3
Aex   = 15e-12
Dind  = 3.0e-3
Ku1   = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1
Pol   = 0.4
xi    = 0.2

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()
saveas(m,"../m0")
flush()

defRegion(1,rect(50e-9,inf))
Ku1.setRegion(1, 1.05*Ku1.GetRegion(0))

cutofftime := 1e-8

for jx:=2e12; jx>0; jx-=1e11 {

    j = vector(-jx,0,0)
    print("Set current density:", jx, "A/m2")

    m.loadfile("m0.ovf")  // reset m to the initial state
    t = 0                 // reset the time

    runwhile( ext_bubblepos.Get().X()<0 && t<cutofftime)

    if (ext_bubblepos.Get().X() > 0 ) {
        print("Skyrmion got through after", t*1e9 ,"ns")
    } else {
        print("Skyrmion didn't get through!")
        exit()
    }

    cutofftime = 2*t  // adjust cut-off time
}
```

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

Run until the skyrmion got through the gate or if the cut-off time has passed

```
runwhile( ext_bubblepos.Get().X()<0 && t<cutofftime)
```

Check if the skyrmion got through the gate

```
if (ext_bubblepos.Get().X() > 0 ) {
    print("Skyrmion got through after", t*1e9 ,"ns")
} else {
    print("Skyrmion didn't get through!")
    exit()
}

cutofftime = 2*t  // adjust cut-off time
```

GHENT UNIVERSITY

```
setgridsize(128,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat  = 580e3
Aex   = 15e-12
Dind  = 3.0e-3
Ku1   = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1
Pol   = 0.4
xi    = 0.2

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()
saveas(m,"../m0")
flush()

defRegion(1,rect(50e-9,inf))
Ku1.setRegion(1, 1.05*Ku1.GetRegion(0))

cutofftime := 1e-8

for jx:=2e12; jx>0; jx-=1e11 {

    j = vector(-jx,0,0)
    print("Set current density:", jx, "A/m2")

    m.loadfile("m0.ovf")  // reset m to the initial state
    t = 0                 // reset the time

    runwhile( ext_bubblepos.Get().X()<0 && t<cutofftime)

    if (ext_bubblepos.Get().X() > 0 ) {
        print("Skyrmion got through after", t*1e9 ,"ns")
    } else {
        print("Skyrmion didn't get through!")
        exit()
    }

    cutofftime = 2*t  // adjust cut-off time
}
```

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

Snippet out of log.txt:

```
…
//Set current density: 1.2e+12 A/m2
//Skyrmion got through after 0.9582254939672915 ns
//Set current density: 1.1e+12 A/m2
//Skyrmion got through after 1.0764332835966428 ns
//Set current density: 1e+12 A/m2
//Skyrmion got through after 1.2385932085022286 ns
//Set current density: 9e+11 A/m2
//Skyrmion got through after 1.4873588596687457 ns
//Set current density: 8e+11 A/m2
//Skyrmion got through after 1.9984716839327563 ns
//Set current density: 7e+11 A/m2
//Skyrmion didn't get through!
```

⇒Threshold current density between 0.7 A/μm² – 0.8 A/μm²

Repeat procedure for smaller Δj to increase the precision

GHENT UNIVERSITY

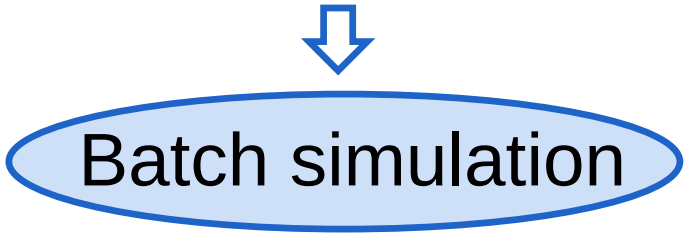
```
setgridsize(128,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat  = 580e3
Aex   = 15e-12
Dind  = 3.0e-3
Ku1   = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1
Pol   = 0.4
xi    = 0.2

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()
saveas(m,"../m0")
flush()

defRegion(1,rect(50e-9,inf))
Ku1.setRegion(1, 1.05*Ku1.GetRegion(0))

cutofftime := 1e-8

for jx:=2e12; jx>0; jx-=1e11 {

    j = vector(-jx,0,0)
    print("Set current density:", jx, "A/m2")

    m.loadfile("m0.ovf")  // reset m to the initial state
    t = 0                 // reset the time

    runwhile( ext_bubblepos.Get().X()<0 && t<cutofftime)

    if (ext_bubblepos.Get().X() > 0 ) {
        print("Skyrmion got through after", t*1e9 ,"ns")
    } else {
        print("Skyrmion didn't get through!")
        exit()
    }

    cutofftime = 2*t  // adjust cut-off time
}
```

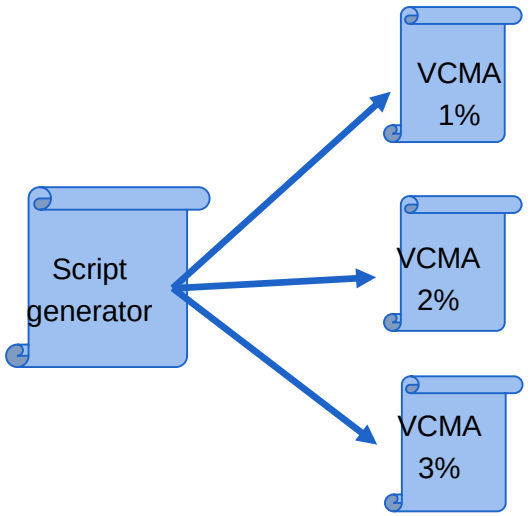
HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

What if we want to make use of multiple GPUs?

What if we want to repeat the simulation for different VCMA values?





GHENT UNIVERSITY

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

homework2_batch_template.txt

```
setgridsize(126,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat  = 580e3
Aex   = 15e-12
Dind  = 3.0e-3
Ku1   = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()

defRegion(1,rect(50e-9,inf))
Ku1.setRegion(1, (1+$VCMA/100.0)*Ku1.GetRegion(0) )

Pol = 0.4
xi  = 0.2
j   = vector(-$JX,0,0)

runwhile( ext_bubblepos.Get().X() < 0 && t < $TCUT )

if ext_bubblepos.Get().X() > 0 {
    print("PASSED!")
}  else {
    print("STOPPED!")
}
```

homework2_batch_setup.py

```
from string import Template
import os

# specify the values for which we want to generate the scripts
vcma_values = [ 1, 2, 3, 4, 5 ]
jx_values   = [ 1e11*i for i in range(1,11) ]
tcut        = 1e-8

# read template script
with open("homework2_batch_template.txt",'r') as f:
    scripttmpl = Template(f.read())

for vcma in vcma_values:

    # create a directory for each vcma value
    dirname = "vcma_simulations/vcma_%02d"%vcma
    os.makedirs(dirname)

    for jx in jx_values:

        # write the script for each combination of VCMA and jx value
        script = scripttmpl.substitute(dict(VCMA=vcma, JX=jx, TCUT=tcut))
        scriptfile = os.path.join(dirname, "jx_%g.mx3"%jx)
        with open(scriptfile,'w') as f:
            f.write(script)
```

HOMEWORK ASSIGNMENT 2

THE VCMA SKYRMION GATE

homework2_batch_post.py

```
import os
import matplotlib.pyplot as plt

# specify the values for which we want to generate the scripts
vcma_values = [ 1, 2, 3, 4, 5 ]
jx_values   = [ 1e11*i for i in range(1,11) ]

results = []
for vcma in vcma_values:
    for jx in jx_values:

        logfile = "vcma_simulations/vcma_%02d/jx_%g.out/log.txt"%(vcma, jx)
        if os.path.exists(logfile):
            with open(logfile, 'r') as f:
                log = f.read()
                if "//PASSED" in log:
                    results.append( (vcma,jx,True) )
                if "//STOPPED" in log:
                    results.append( (vcma,jx,False) )

vcma,jx,passed = zip(*results)
color = ["green" if p else "red" for p in passed]

plt.scatter(vcma,jx,color=color,s=200)
plt.xlabel("VCMA (%)")
plt.ylabel("Current density $j$ (A/m2)")
plt.show()
```

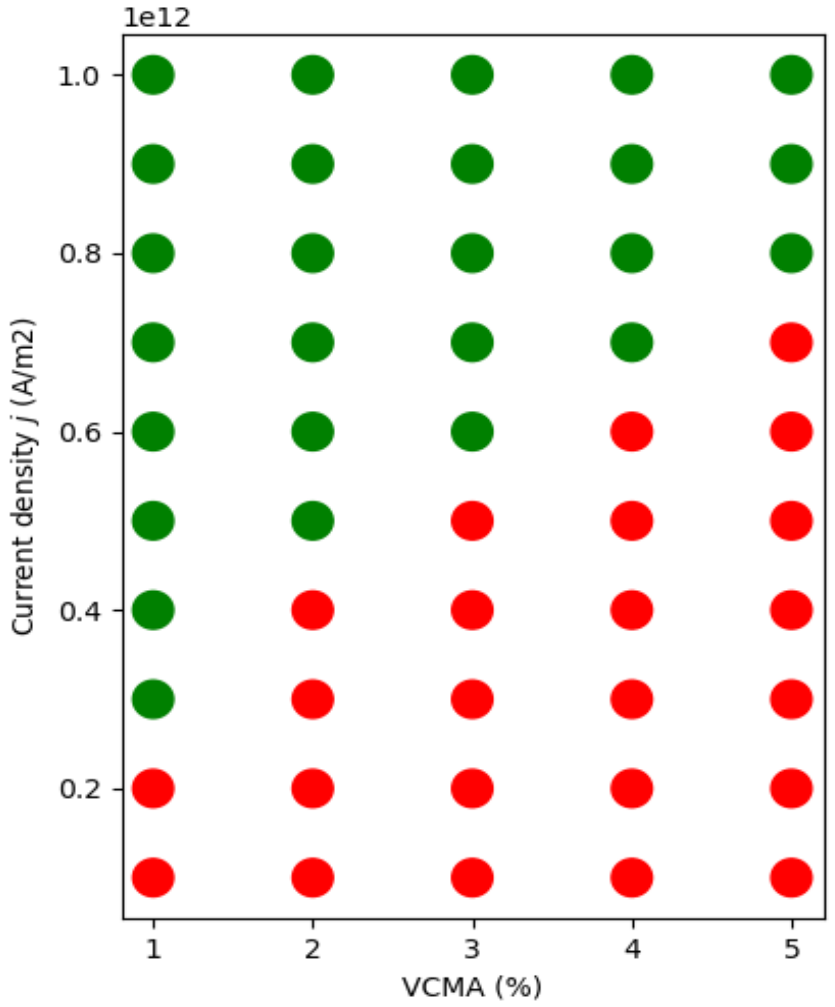


TABLE OF CONTENTS

GHENT UNIVERSITY

DOMAIN WALL CREEP

System

Transverse domain wall in permalloy nanostrip

Task

Calculate the motion of the domain wall

- driven by a spin transfer torque
- at nonzero temperature
- in the presence of material disorder

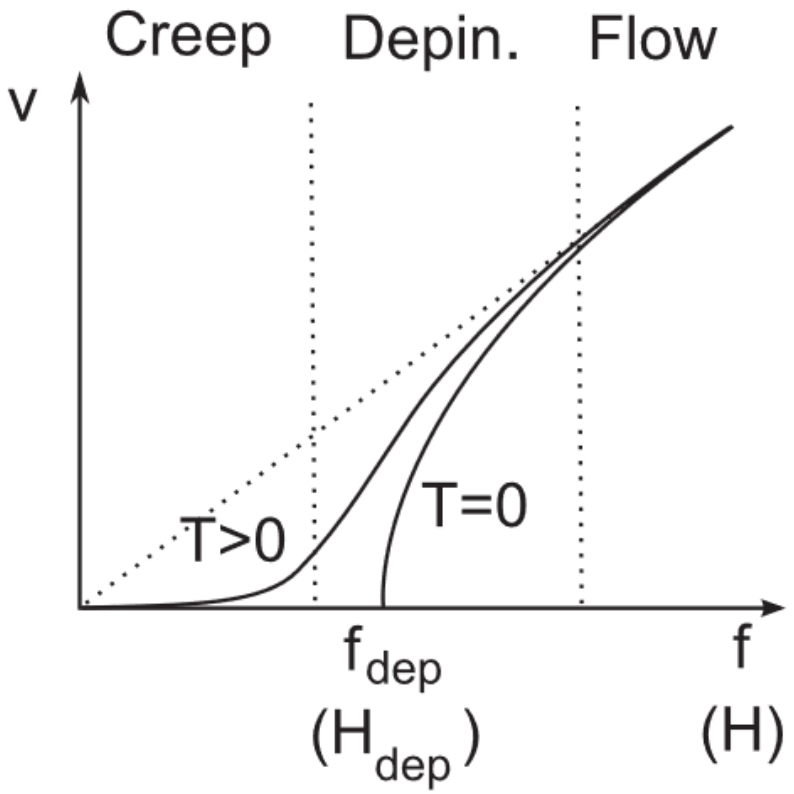
Input file: session4_example1.txt

Titan RTX: 111 min

GHENT UNIVERSITY

"Creep turns linear in narrow ferromagnetic nanostrips"
Scientific Reports, 6, article number 20472 (2016)

CREEP?



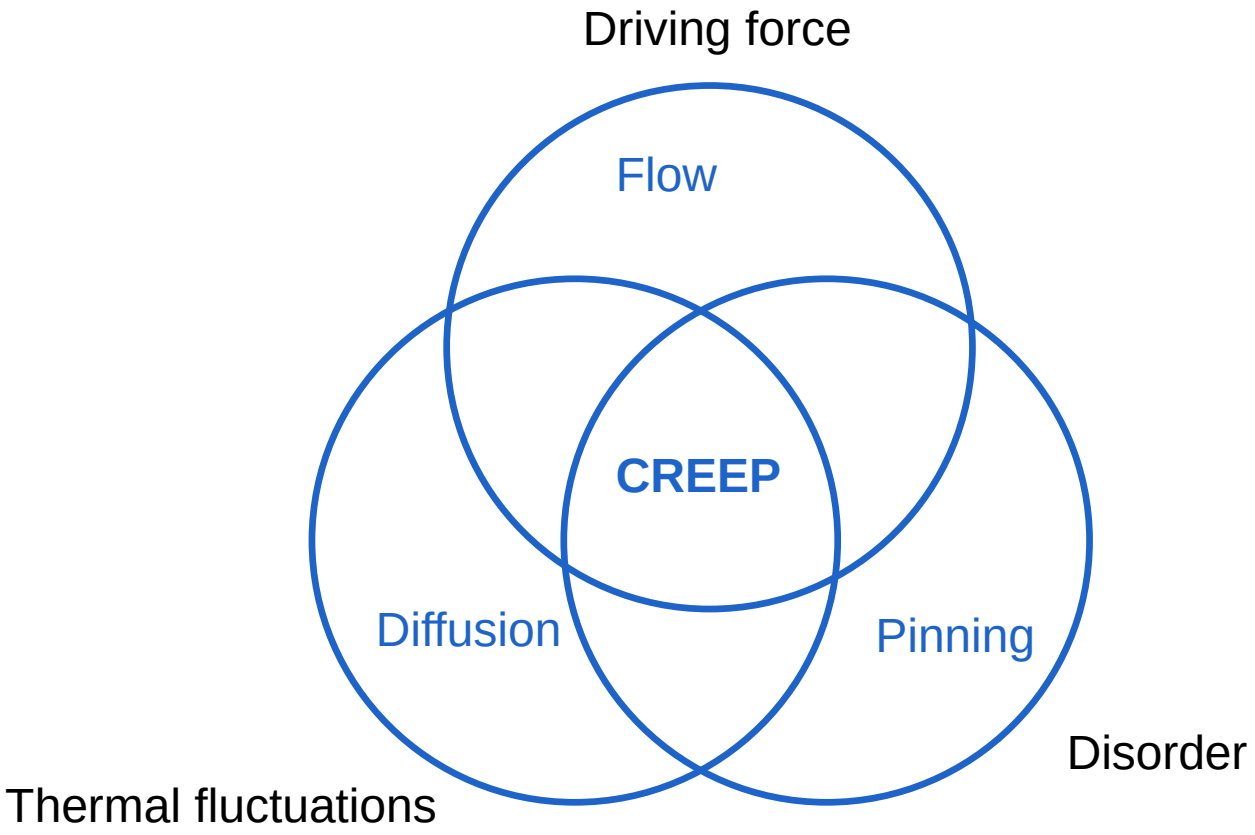


Figure reproduced from P. Metaxas, et al., "Creep and Flow Regimes of Magnetic Domain-Wall Motion in Ultrathin Pt/Co/Pt Films with Perpendicular Anisotropy", Phys. Rev. Lett. **99**, 217208 (2007)

GHENT UNIVERSITY

INPUTFILE

```
SetMesh(128, 32, 1, 3.125e-09, 3.125e-09, 10e-09, 0, 0, 0)
Msat = 800e3
Aex = 1e-11
alpha = 0.1
```

```
m=twodomain(1,0,0,  0,1,0,  -1,0,0)

//remove edge charges
BoundaryRegion := 0
MagLeft        := 1
MagRight       := -1
ext_rmSurfaceCharge(BoundaryRegion, MagLeft, MagRight)

relax()
```

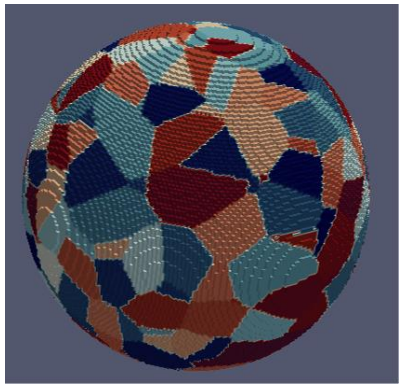
GHENT UNIVERSITY

GRAINS

```
// define grains with region number 0-255
grainSize  := 40e-9
randomSeed := 1234567
maxRegion  := 255
ext_makegrains(grainSize, maxRegion, randomSeed)
```

Also 3D grains are possible

```
ext_make3dgrains(grainSize, StartingRegion, NumberOfRegions, Shape, randomSeed)
```



GHENT UNIVERSITY

More info on grains in mumax3:

"Current-driven domain wall mobility in polycrystalline Permalloy nanowires: A numerical study"
Journal of Applied Physics 115, 233903 (2014)

GRAINS

procedural texture using Voronoi Tessellation

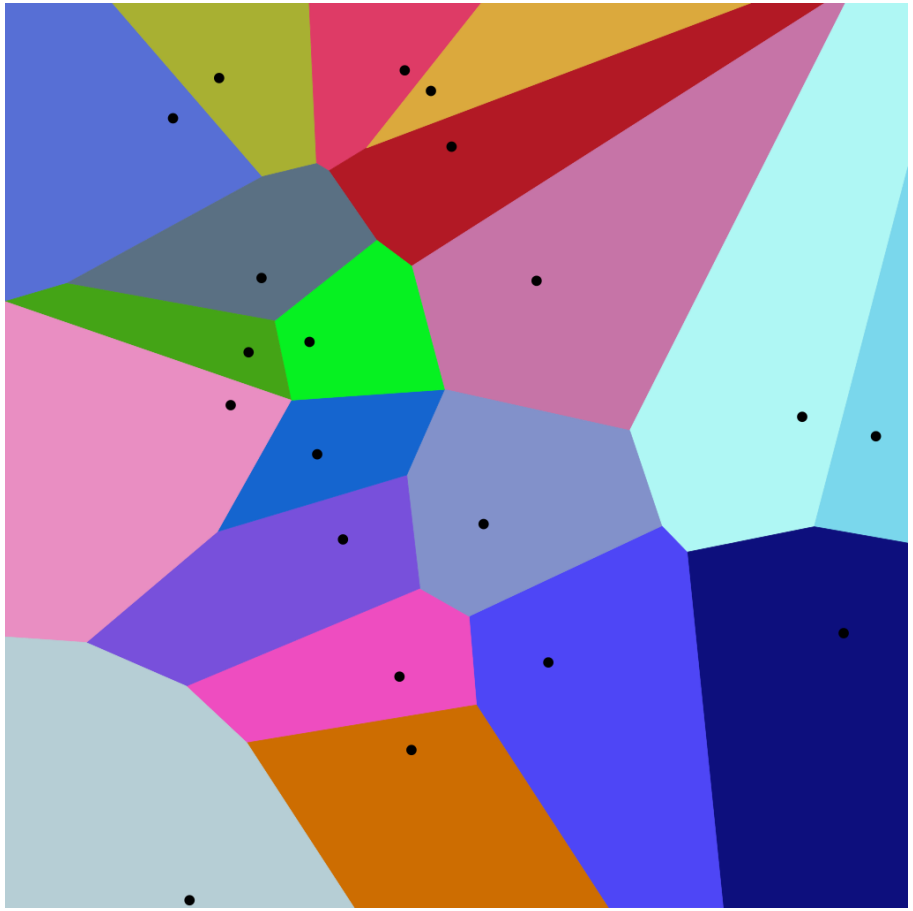


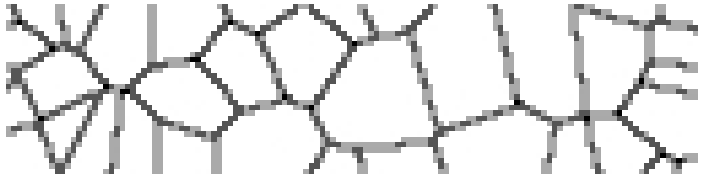
Figure reproduced from wikimedia commons, by Balu Ertl, CC-BY-SA-4.0 license

GHENT UNIVERSITY

MATERIAL DISORDER

```
// set random anisotropy per region
for i:=0; i<maxRegion; i++{
    // random uniaxial anisotropy direction on the unit sphere
    axis  := vector(randNorm(), randNorm(), randNorm())
    // note: axes are normalized by mumax3
    AnisU.SetRegion(i, axis)

    // random 10% anisotropy variation
    K := 1e3
    Ku1.SetRegion(i, K + randNorm() * 0.1 * K)
}
```

```
// reduce exchange coupling between grains by 10%
for i:=0; i<maxRegion; i++{
    for j:=i+1; j<maxRegion; j++{
        ext_ScaleExchange(i, j, 0.9)
    }
}
```



GHENT UNIVERSITY

TEMPERATURE AND DRIVING FORCE

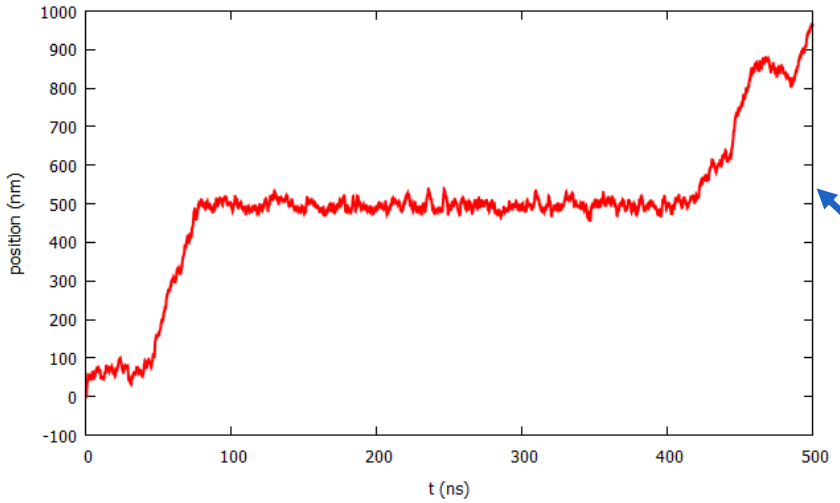
```
//set temperature and solver with adaptive time stepping
temp=300
Thermseed(12345)
setsolver(5)
fixdt=0  //default
```

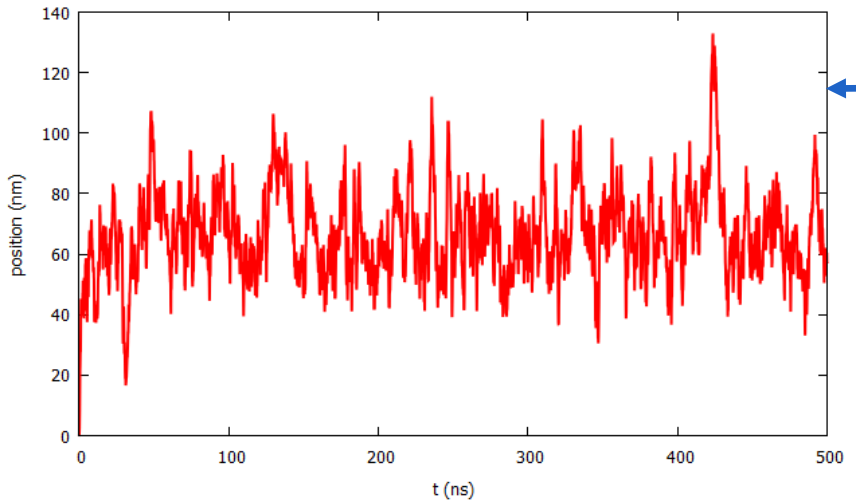
```
//driving force: spin transfer torque
Pol=0.4
xi=0.2
j=vector(-1e12,0,0)
```

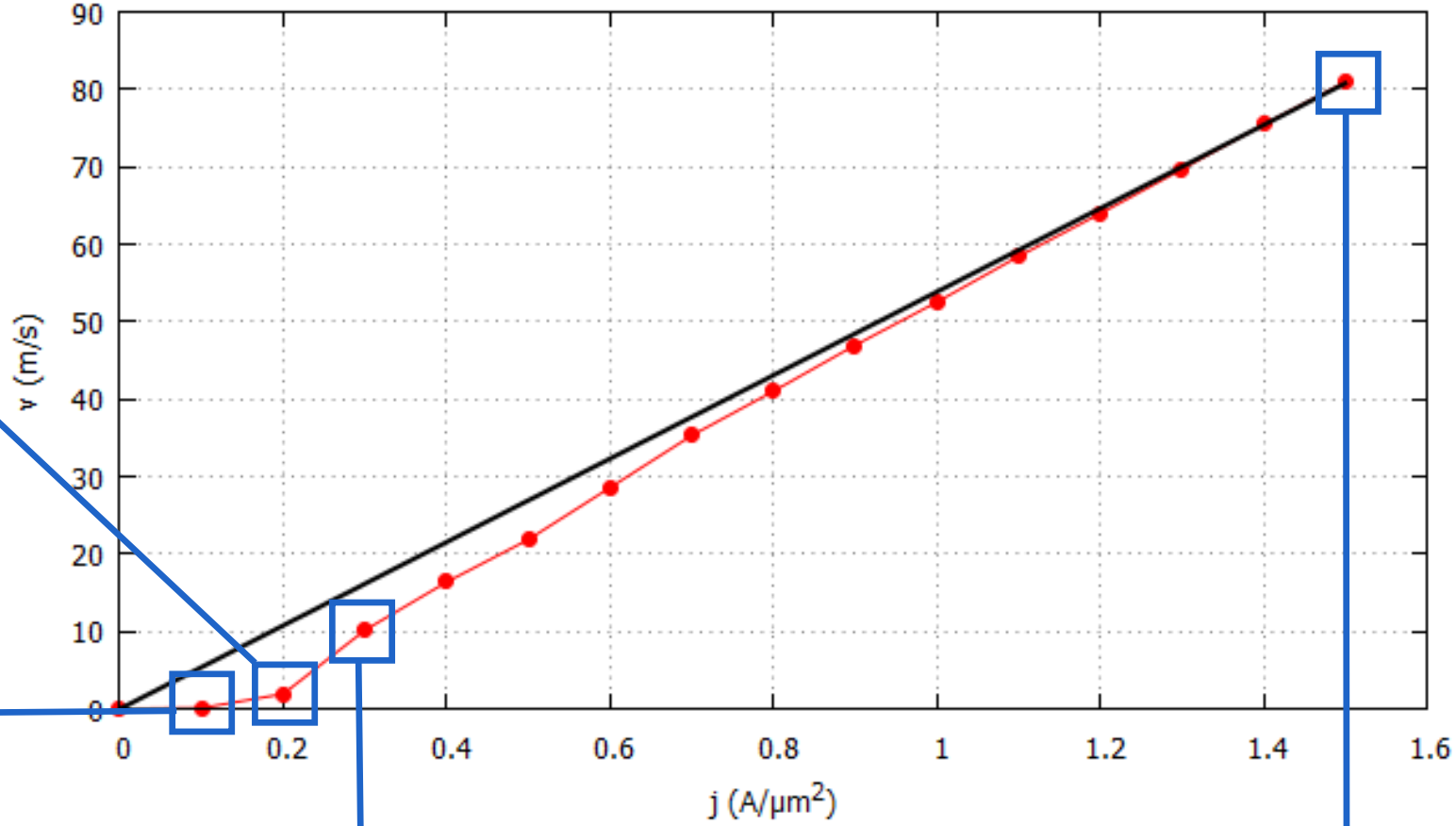
```
//keep wall centered: moving window
ext_centerwall(0)

//schedule output
tableadd(ext_dwxpos)
tableautosave(1e-10)

//run
run(5e-7)
```

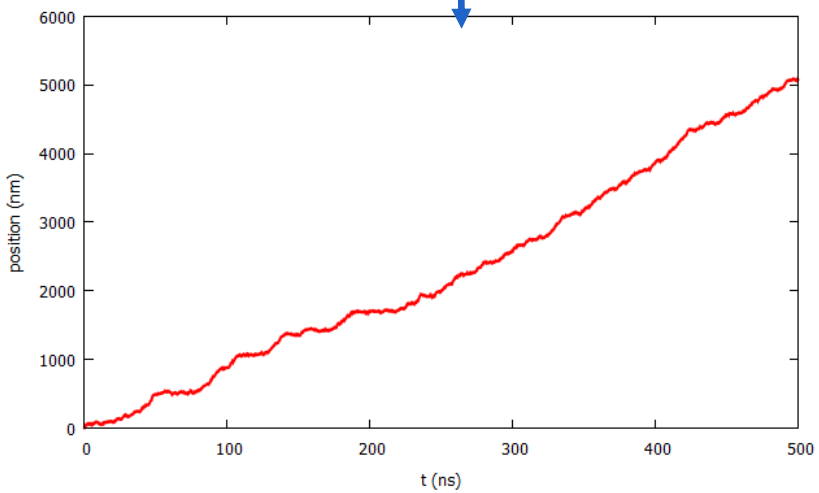
GHENT UNIVERSITY

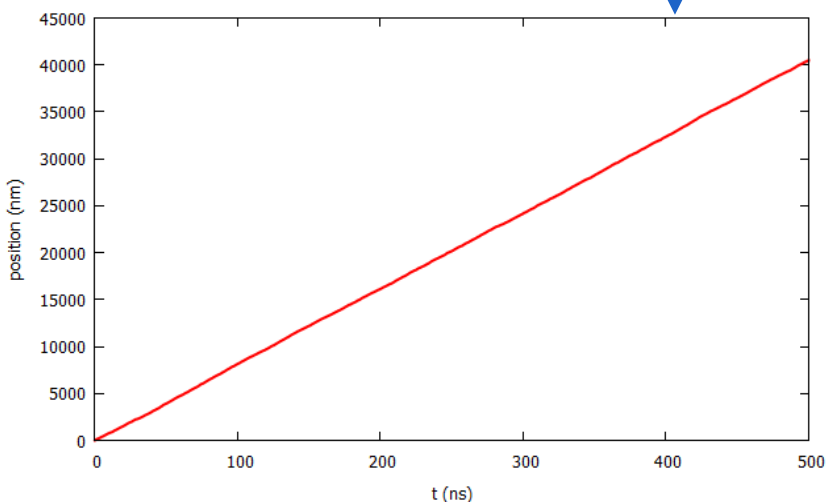
RESULTS





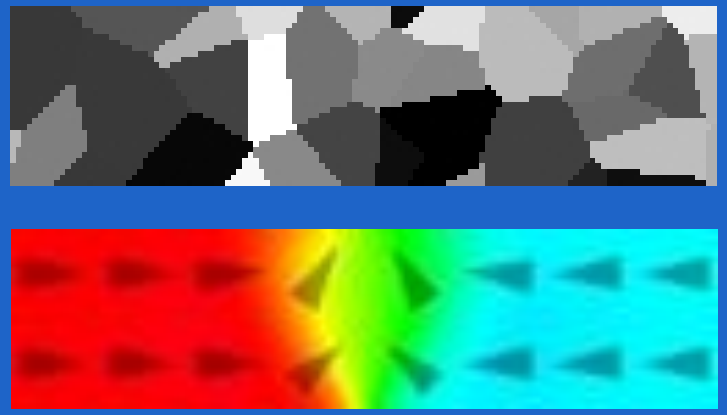






GHENT UNIVERSITY

DOMAIN WALL CREEP



WHAT HAVE WE LEARNED?

- Including disorder in simulations
- Moving window
  - For domain walls: `ext_centerWall()`
  - For skyrmions: `ext_centerBubble()`

"Creep turns linear in narrow ferromagnetic nanostrips"
Scientific Reports, 6, article number 20472 (2016)

GHENT UNIVERSITY

SKYRMION RACETRACK REVISITED

System

Skyrmion driven by **spin-orbit torques (SOT)** in a chiral ferromagnetic strip

Input file: session4_example2a.txt

Titan RTX: 3.5 min

Input file: session4_example2b.txt

Titan RTX: 2 min

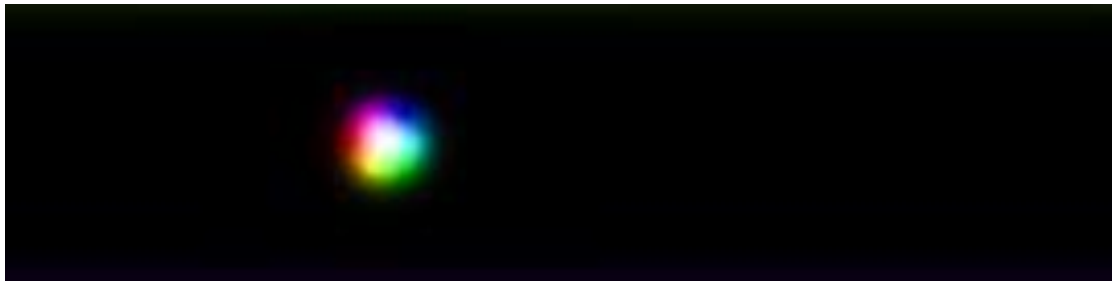
GHENT UNIVERSITY

SKYRMION RACETRACK REVISITED

```
// Skyrmion racetrack

setgridsize(256,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat = 580e3
Aex = 15e-12
Dind = 3.0e-3
Ku1 = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()
```

```
Pol = 0.4
xi = 0.2
j = vector(-1e12,0,0)
```

```
autosave(m,1e-10)
tableAutosave(1e-11)
tableAdd(ext_bubblepos)

run(10e-9)
```



How to use

Spin-Orbit Torques (SOT)

instead of

Spin-Transfer Torques (STT)

?

GHENT UNIVERSITY

SPIN ORBIT TORQUE

$$\dot{\mathbf{m}} = -\gamma \mathbf{m} \times \left(\mathbf{H}_{\text{eff}} + \boxed{a_J(\mathbf{m} \times \mathbf{p}) + b_J \mathbf{p}} \right) + \alpha \mathbf{m} \times \dot{\mathbf{m}}$$

- M. Hayashi, et al., "Quantitative characterization of the spin-orbit torque using harmonic Hall voltage measurements" Physical Review B 89, 144425 (2014)

$$a_J = \| \frac{\hbar}{2 M_{\text{sat}} e} \frac{\alpha_{\text{H}} j}{d} \| \qquad \xi = \frac{b_J}{a_J}$$

$$\mathbf{p} = \text{sign}(\alpha_H) \mathbf{j} \times \mathbf{n}$$

Assuming a material with positive α_H (such as Pt) above the ferromagnet (**n**=-**z)** and a current pulse in –x direction (electrons move along +X)

$$\mathbf{p} = -\mathbf{e}_x \times -\mathbf{e}_z = -\mathbf{e}_y \longrightarrow \mathbf{p} = (0, -1, 0)$$

- F. Büttner, et al. "Field-free deterministic ultrafast creation of magnetic skyrmions by spin–orbit torques" Nature Nanotechnology volume 12, pages1040–1044(2017)

GHENT UNIVERSITY

SPIN ORBIT TORQUE

$$\dot{\mathbf{m}} = -\gamma \mathbf{m} \times \left(\mathbf{H}_{\text{eff}} + \boxed{a_J(\mathbf{m} \times \mathbf{p}) + b_J \mathbf{p}} \right) + \alpha \mathbf{m} \times \dot{\mathbf{m}}$$

- SOT is not explicitly implemented in mumax3
- Solution -> Use the custom fields functionality to add it as an effective field term

GHENT UNIVERSITY

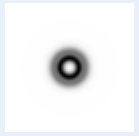
CUSTOM QUANTITIES

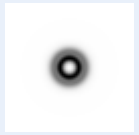
```
//Define constant number or vector
Const(float64) Quantity
ConstVector(float64, float64, float64) Quantity

//pointwise addition, multiplication and division
Add(Quantity, Quantity) Quantity
Mul(Quantity, Quantity) Quantity
Div(Quantity, Quantity) Quantity

//Weighted addition: Madd(Q1,Q2,c1,c2) = c1*Q1 + c2*Q2
Madd(Quantity, Quantity, float64, float64) *mAddition

//vector dot and cross product
Dot(Quantity, Quantity) Quantity
Cross(Quantity, Quantity) Quantity

//shifted quantity
Shifted(Quantity, int, int, int) Quantity
```

EXAMPLE: CUSTOM QUANTITIES

```
cs := 1e-9
setcellsize(cs,cs,cs)
setgridsize(64,64,1)

// Use central finite differences to approximate the spatial derivatives of m
mL := Shifted(m,-1,0,0) // shift left
mR := Shifted(m,1,0,0)  // shift right
mD := Shifted(m,0,-1,0) // shift down
mU := Shifted(m,0,1,0)  // shift up
dmdx := Mul( Const(1/(2*cs)), Madd(mR,mL,1,-1) )
dmdy := Mul( Const(1/(2*cs)), Madd(mU,mD,1,-1) )

// Define the topological charge density
chargeDensity := Mul( Const(1/(4*pi)), Dot(m, Cross(dmdx,dmdy)))

m = neelskyrmion(1,-1)
Save(m)

// Save the topological charge density of a skyrmion
saveas(chargeDensity, "chargeDensity.ovf")

// As reference solution: implementation in mumax3
Saveas(ext_topologicalchargedensity, "reference.ovf"
```

$$\frac{1}{4\pi}\mathbf{m}\cdot\left(\frac{d\mathbf{m}}{dx}\times\frac{d\mathbf{m}}{dy}\right)$$





EXAMPLE: CUSTOM ANISOTROPY FIELD

```
// Define custom quantities
Ms := 1100e3
K  := 0.5e6
u  := ConstVector(1, 0, 0)
anisField := Mul( Const(2*K/Ms)  , Mul( Dot(u, m), u))
anisEdens := Mul( Const(-0.5*Ms) , Dot( anisField, m))

// promote anisField to an effective field term
AddFieldTerm(anisField)

// promote anisEdens to an energy density term
AddEdensTerm(anisEdens)

// Add a column with the energy related to the custom field
tableAdd(E_custom)
```

$$\vec{\mathbf{B}}_{\text{anis}} = \frac{2K_{\text{u1}}}{B_{\text{sat}}} \left(\vec{\mathbf{u}} \cdot \vec{\mathbf{m}} \right) \vec{\mathbf{u}}$$

SPIN-ORBIT TORQUE: CUSTOM FIELDS

```
// Define constants
AlphaH   := 0.15
e        := 1.6021766e-19
d        := 1e-9
Ms       := 580e3
hbar     := 1.0545718e-34
p        := Constvector(0,-1,0)
SOTxi    := -2.0
J_SOT    := abs(-2.e11)
```

$$a_J = \| \frac{\hbar}{2M_{\mathrm{sat}}e} \frac{\alpha_{\mathrm{H}} j}{d} \|$$

$$\xi = \frac{b_J}{a_J}$$

$$\dot{\mathbf{m}} = -\gamma \mathbf{m} \times \left(\mathbf{H}_{\mathrm{eff}} + \boxed{a_J(\mathbf{m} \times \mathbf{p}) + b_J \mathbf{p}} \right) + \alpha \mathbf{m} \times \dot{\mathbf{m}}$$

```
// Define prefactors aj and bj
aj := Const(J_SOT*(hbar/2.*alphaH/e/d/ms))
bj := Mul(aj,Const(SOTxi))
```

```
// Add damping-like SOT term

dampinglike := Mul(aj, Cross(m,p))
AddFieldTerm(dampinglike)
AddEdensTerm(Mul(Const(-0.5),Dot(dampinglike,M_full)))
```

```
// Add field-like SOT term

fieldlike:= Mul(bj,p)
AddFieldTerm(fieldlike)
AddEdensTerm(Mul(Const(-0.5),Dot(fieldlike,M_full)))
```

GHENT UNIVERSITY

SKYRMION RACETRACK

```
// Skyrmion racetrack

setgridsize(256,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat = 580e3
Aex = 15e-12
Dind = 3.0e-3
Ku1 = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()

//Add SOT

autosave(m,1e-10)
tableAutosave(1e-11)
tableAdd(ext_bubblepos)

run(2.5e-9)
```

```
// Define constants
AlphaH  := 0.15
e       := 1.6021766e-19
d       := 1e-9
Ms      := 580e3
hbar    := 1.0545718e-34
p       := Constvector(0,1,0)
SOTxi   := -2.0
J_SOT   := abs(-2.e11)

//Define prefactors aj and bj
aj       := Const(J_SOT*(hbar/2.*alphaH/e/d/ms))
bj       := Mul(aj,Const(SOTxi))

// Add damping-like SOT term
dampinglike := Mul(aj, Cross(m,p))
AddFieldTerm(dampinglike)
AddEdensTerm(Mul(Const(-0.5),Dot(dampinglike,M_full)))

// Add field-like SOT term
fieldlike:= Mul(bj,p))
AddFieldTerm(fieldlike)
AddEdensTerm(Mul(Const(-0.5),Dot(fieldlike,M_full)))
```

GHENT UNIVERSITY

IS THERE A BETTER WAY?

$$\dot{\mathbf{m}} = -\gamma \mathbf{m} \times \left(\mathbf{H}_{\text{eff}} + \boxed{a_J(\mathbf{m} \times \mathbf{p}) + b_J \mathbf{p}} \right) + \alpha \mathbf{m} \times \dot{\mathbf{m}}$$

$$\boxed{\mathbf{m} \times} \dot{\mathbf{m}} = -\gamma \mathbf{m} \times (\mathbf{m} \times \mathbf{H}_{\text{eff}}) - a_J \gamma \mathbf{m} \times (\mathbf{m} \times (\mathbf{m} \times \mathbf{p})) - \gamma b_J \mathbf{m} \times (\mathbf{m} \times \mathbf{p}) + \alpha \mathbf{m} \times (\mathbf{m} \times \dot{\mathbf{m}})$$

$$\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = \mathbf{b}(\mathbf{a} \cdot \mathbf{c}) - \mathbf{c}(\mathbf{a} \cdot \mathbf{b})$$

$$(\mathbf{m} \cdot \dot{\mathbf{m}}) = 0$$

$$(\mathbf{m} \cdot \mathbf{m}) = 1$$

$$\mathbf{m} \times (\mathbf{m} \times (\mathbf{m} \times \mathbf{p})) = -\mathbf{m} \times \mathbf{p}$$

$$\mathbf{m} \times \dot{\mathbf{m}} = -\gamma \mathbf{m} \times (\mathbf{m} \times \mathbf{H}_{\text{eff}})) + a_J \gamma \mathbf{m} \times \mathbf{p} - \gamma b_J \mathbf{m} \times (\mathbf{m} \times \mathbf{p}) - \alpha \dot{\mathbf{m}}$$

GHENT UNIVERSITY

IS THERE A BETTER WAY?

$$\mathbf{m} \times \dot{\mathbf{m}} = -\gamma \mathbf{m} \times (\mathbf{m} \times \mathbf{H}_{\text{eff}})) + a_J \gamma \mathbf{m} \times \mathbf{p} - \gamma b_J \mathbf{m} \times (\mathbf{m} \times \mathbf{p}) - \alpha \dot{\mathbf{m}}$$

$$\dot{\mathbf{m}} = -\gamma \mathbf{m} \times \left(\mathbf{H}_{\text{eff}} + a_J (\mathbf{m} \times \mathbf{p}) + b_J \mathbf{p} \right) + \alpha \mathbf{m} \times \dot{\mathbf{m}}$$

$$\begin{aligned} (1+\alpha^2)\dot{\mathbf{m}} = & -\gamma \mathbf{m} \times \mathbf{H}_{\text{eff}} - \alpha\gamma \mathbf{m} \times (\mathbf{m} \times \mathbf{H}_{\text{eff}}) \\ & - \gamma a_J \mathbf{m} \times (\mathbf{m} \times \mathbf{p}) - \alpha\gamma b_J \mathbf{m} \times (\mathbf{m} \times \mathbf{p}) \\ & - \gamma b_J \mathbf{m} \times \mathbf{p} + \alpha\gamma a_J \mathbf{m} \times \mathbf{p} \end{aligned}$$

Landau-Lifshitz torque

Spin-Orbit torque

$$\tau_{\text{SOT}} = -\frac{\gamma}{(1+\alpha^2)} a_J \left[(1+\xi\alpha)\mathbf{m} \times (\mathbf{m} \times \mathbf{p}) + (\xi - \alpha)(\mathbf{m} \times \mathbf{p}) \right]$$

GHENT UNIVERSITY

IS THERE A BETTER WAY?

$$\boxed{\tau_{\mathrm{SOT}} = -\frac{\gamma}{(1+\alpha^2)} a_J \left[(1+\xi\alpha)\mathbf{m}\times(\mathbf{m}\times\mathbf{p}) + (\xi-\alpha)(\mathbf{m}\times\mathbf{p})\right]}$$

Slonczewski STT in mumax3

$$\begin{cases} \tau_{SL} = \gamma\beta\dfrac{\epsilon-\alpha\epsilon'}{1+\alpha^2}(\mathbf{m}\times(\mathbf{m}_P\times\mathbf{m})) - \gamma\beta\dfrac{\epsilon'-\alpha\epsilon}{1+\alpha^2}\mathbf{m}\times\mathbf{m}_P \\ \beta = \dfrac{j_z\hbar}{M_{\mathrm{sat}}ed} \\ \epsilon = \dfrac{P\Lambda^2}{(\Lambda^2+1)+(\Lambda^2-1)(\mathbf{m}\cdot\mathbf{m}_P)} \end{cases}$$

$$\Lambda = 1 \longrightarrow \epsilon = \frac{P}{2}$$

$$\epsilon' = \xi\epsilon$$

$$\tau_{SL} = -\frac{\beta\gamma P}{2(1+\alpha^2)}\left[(1+\xi\alpha)(\mathbf{m}\times(\mathbf{m}\times\mathbf{m}_P)) + (\xi-\alpha)\mathbf{m}\times\mathbf{m}_P\right]$$

GHENT UNIVERSITY

IS THERE A BETTER WAY?

$$\tau_{\mathrm{SOT}} = -\frac{\gamma}{(1+\alpha^2)} a_J \left[(1+\xi\alpha)\mathbf{m}\times(\mathbf{m}\times\mathbf{p}) + (\xi-\alpha)(\mathbf{m}\times\mathbf{p})\right]$$

We can use the existing Slonczewski spin transfer torque implementation!

$$P = \frac{2a_J}{\beta} = \alpha_H$$

$$\tau_{SL} = -\frac{\beta\gamma P}{2(1+\alpha^2)} \left[(1+\xi\alpha)(\mathbf{m}\times(\mathbf{m}\times\mathbf{m}_P)) + (\xi-\alpha)\mathbf{m}\times\mathbf{m}_P\right]$$

GHENT UNIVERSITY

SKYRMION RACETRACK

```
// Skyrmion racetrack

setgridsize(256,64,1)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat = 580e3
Aex = 15e-12
Dind = 3.0e-3
Ku1 = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1

m = neelskyrmion(-1, 1).transl(-40e-9,0,0)
minimize()

//Add SOT

autosave(m,1e-10)
tableAutosave(1e-11)
tableAdd(ext_bubblepos)

run(2.5e-9)
```

```
//define constants and set slonczewksi parameters
SOTxi          := -2
AlphaH         := 0.15
Pol             = alphaH
Lambda          = 1
Epsilonprime    = alphaH/2 * SOTxi
Fixedlayer      = vector(0,-1,0)        //p

//define current
J               = vector(0,0,abs(-2e11))
```

$P = \frac{2a_J}{\beta} = \alpha_H$

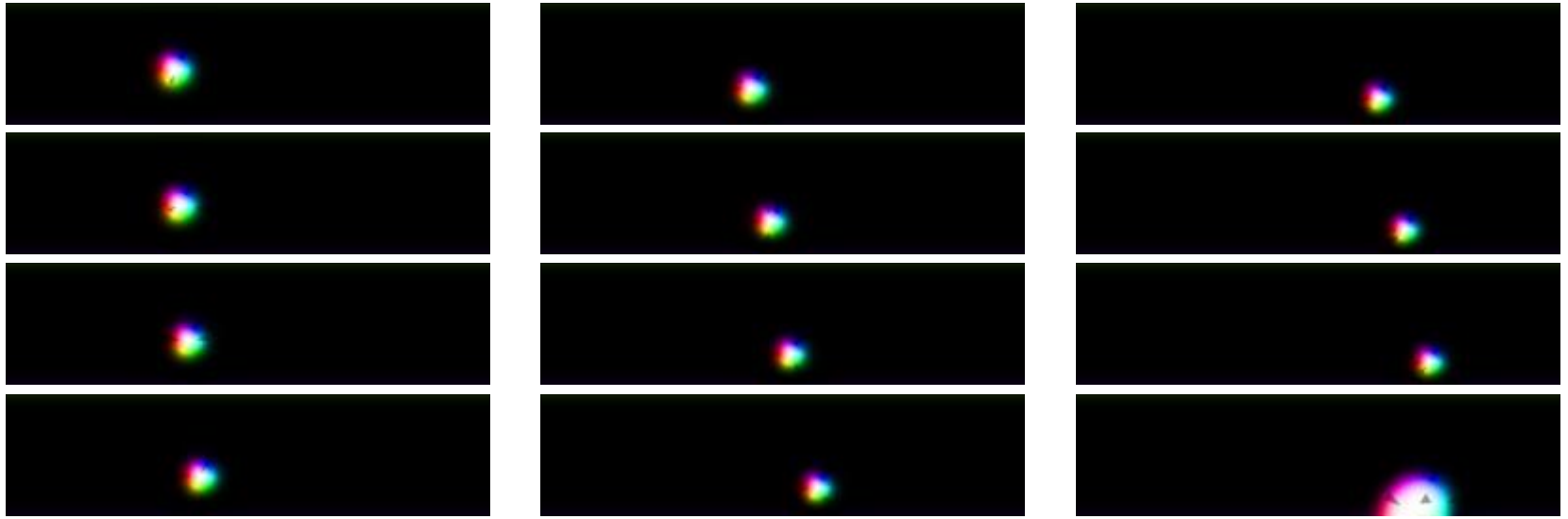
$\Lambda = 1$

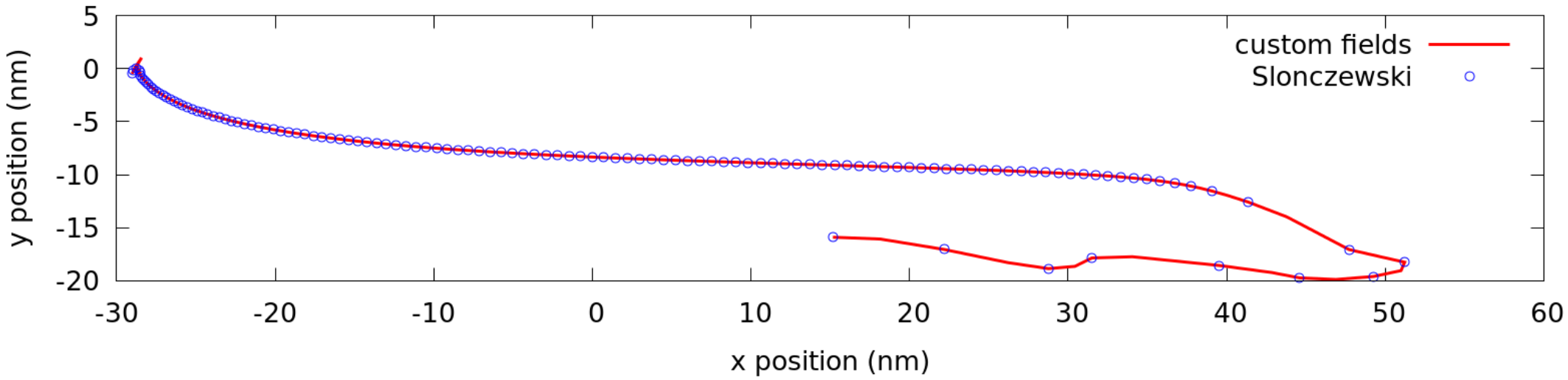
$\epsilon' = \xi\epsilon$

$\epsilon = \frac{P}{2}$

GHENT UNIVERSITY

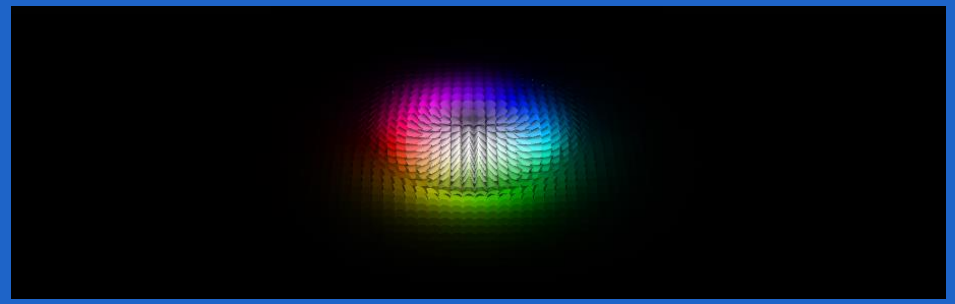
SKYRMION RACETRACK





GHENT UNIVERSITY

SOT–DRIVEN SKYRMION MOTION



WHAT HAVE WE LEARNED?

- The use of Spin-Orbit torques, using
  - Custom fields implementation
  - Slonczewksi STT implementation

Input file: session4_example2a.txt
Titan RTX: 3.5 min

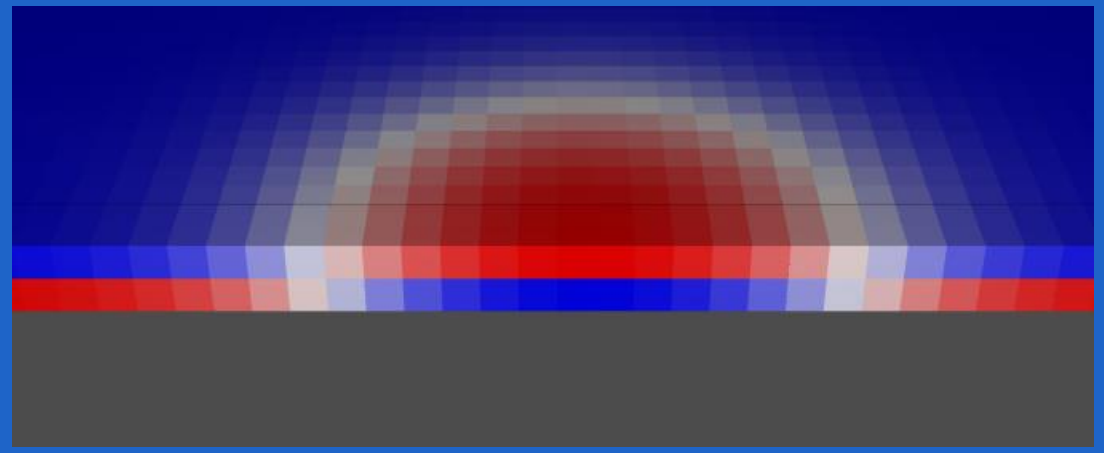
Input file: session4_example2b.txt
Titan RTX: 2 min

GHENT UNIVERSITY

SKYRMION RACETRACK REVISITED 2

System

Skyrmion driven by spin-orbit torques (SOT) in a synthetic antiferromagnet



Input file: session4_example3a.txt

Titan RTX: 86 s

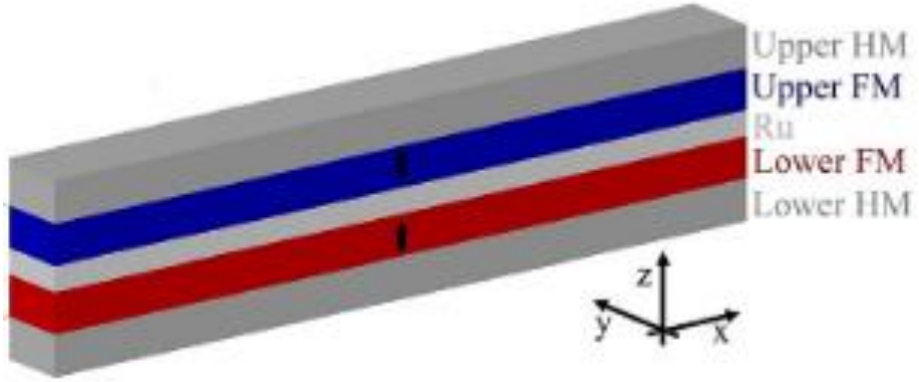
Input file: session4_example3b.txt

Titan RTX: 7 min

GHENT UNIVERSITY

SKYRMION RACETRACK IN SYNTHETIC ANTIFERROMAGNET

```
// Skyrmion racetrack

setgridsize(256,64,2)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat = 580e3
Aex = 15e-12
Dind = 3.0e-3
Ku1 = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1
```



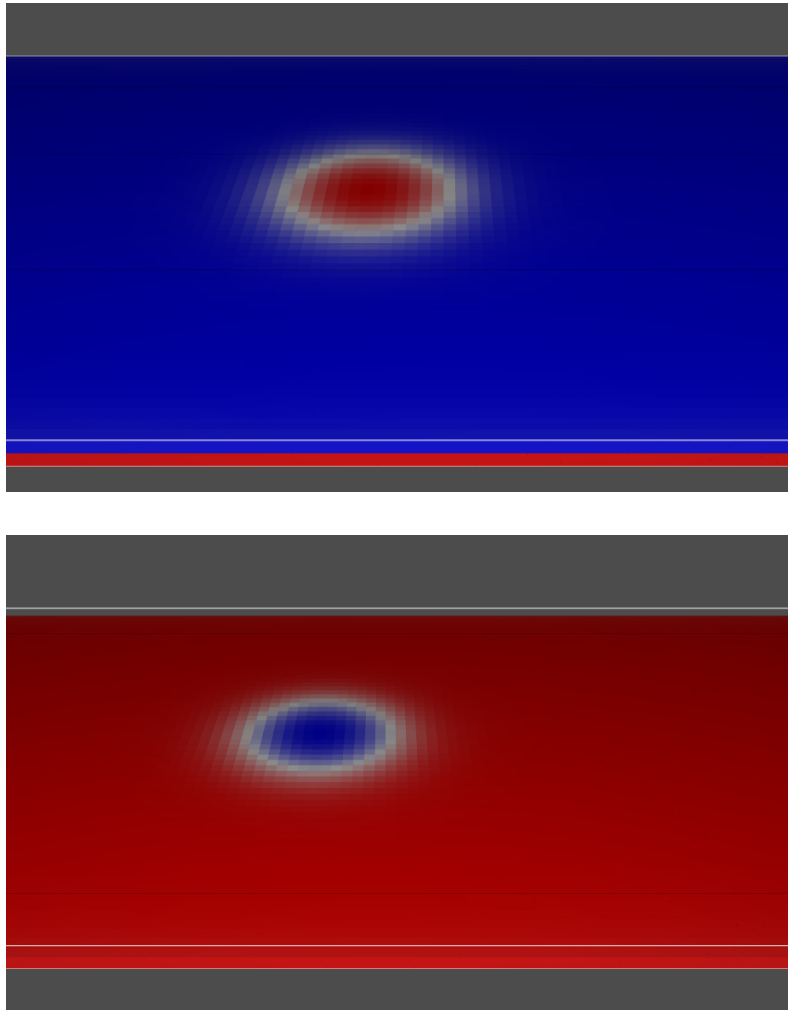

```
// define 2 layers
defregion(1,layer(1))
defregion(2,layer(0))

// set negative interlayer exchange
ext_InterExchange(1, 2, -5e-13)

// define initial magnetization
m.setregion(1,neelskyrmion(-1, 1).transl(-40e-9,0,0))
m.setregion(2,neelskyrmion(1, -1).transl(-40e-9,0,0))

Minimize()
```

GHENT UNIVERSITY

Figure reproduced from "Performance of synthetic antiferromagneticracetrack memory: domain wall versus skyrmion", R. Tomasello, et al. 2017 J. Phys. D: Appl. Phys. 50 325302

SKYRMION RACETRACK IN SYNTHETIC ANTIFERROMAGNET

```
// define constants and set slonczewksi parameters
SOTxi         := -2
AlphaH        := 0.15
Pol            = alphaH
Lambda         = 1
Epsilonprime   = alphaH*SOTxi/2
Fixedlayer     = vector(0,-1,0)

// define current
J  = vector(0,0,2e11)

autosave(m,1e-10)
tableAutosave(1e-11)
tableAdd(ext_bubblepos)

run(2e-9)
```



GHENT UNIVERSITY

SKYRMION RACETRACK IN SYNTHETIC ANTIFERROMAGNET

What about the spacer layer?

- We can't use `ext_InterExchange(1, 2, -5e-13)` because the two FM layers don't have an interface
- CustomFields can provide a solution!

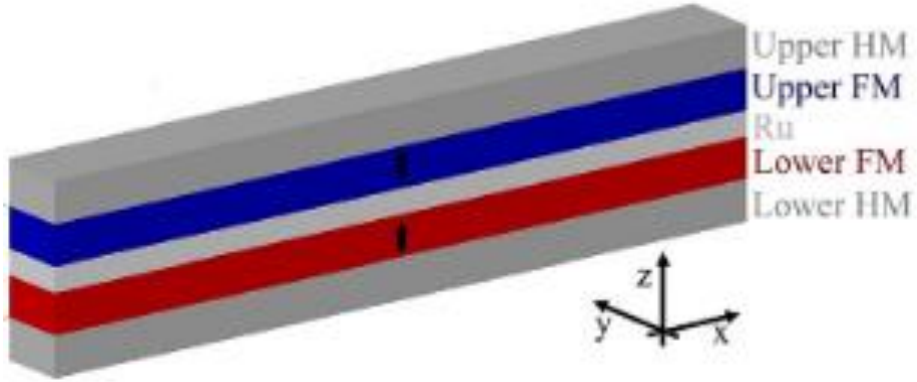
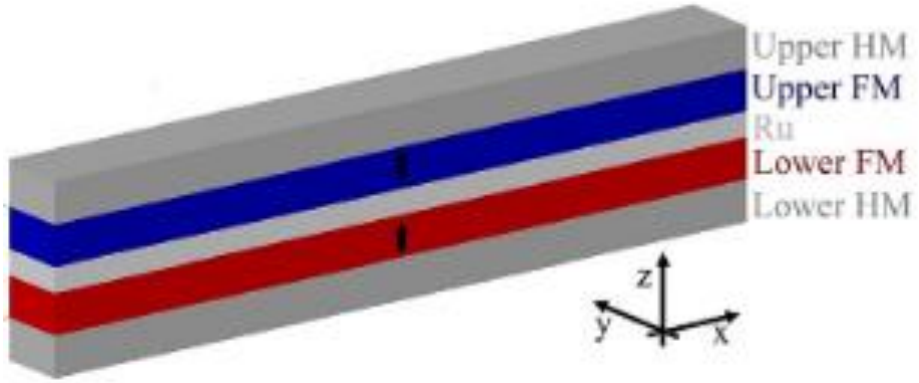


Figure reproduced from "Performance of synthetic antiferromagneticracetrack memory: domain wall versus skyrmion", R. Tomasello, et al. 2017 J. Phys. D: Appl. Phys. 50 325302

SKYRMION RACETRACK IN SYNTHETIC ANTIFERROMAGNET




```
// Skyrmion racetrack

setgridsize(256,64,3)
setcellsize(1e-9,1e-9,1e-9)
setpbc(4,0,0)

Msat = 580e3
Aex = 15e-12
Dind = 3.0e-3
Ku1 = 0.8e6
AnisU = vector(0,0,1)
alpha = 0.1
```

```
// define 2 layers
defregion(1,layer(2))
defregion(2,layer(0))

//set geometry
Setgeom(layer(0).add(layer(2)))

//define initial magnetization
m.setregion(1,neelskyrmion(-1, 1).transl(-40e-9,0,0))
m.setregion(2,neelskyrmion(1, -1).transl(-40e-9,0,0))

Minimize()
```

GHENT UNIVERSITY

Figure reproduced from "Performance of synthetic antiferromagnetic racetrack memory: domain wall versus skyrmion", R. Tomasello, et al. 2017 J. Phys. D: Appl. Phys. 50 325302

SKYRMION RACETRACK IN SYNTHETIC ANTIFERROMAGNET

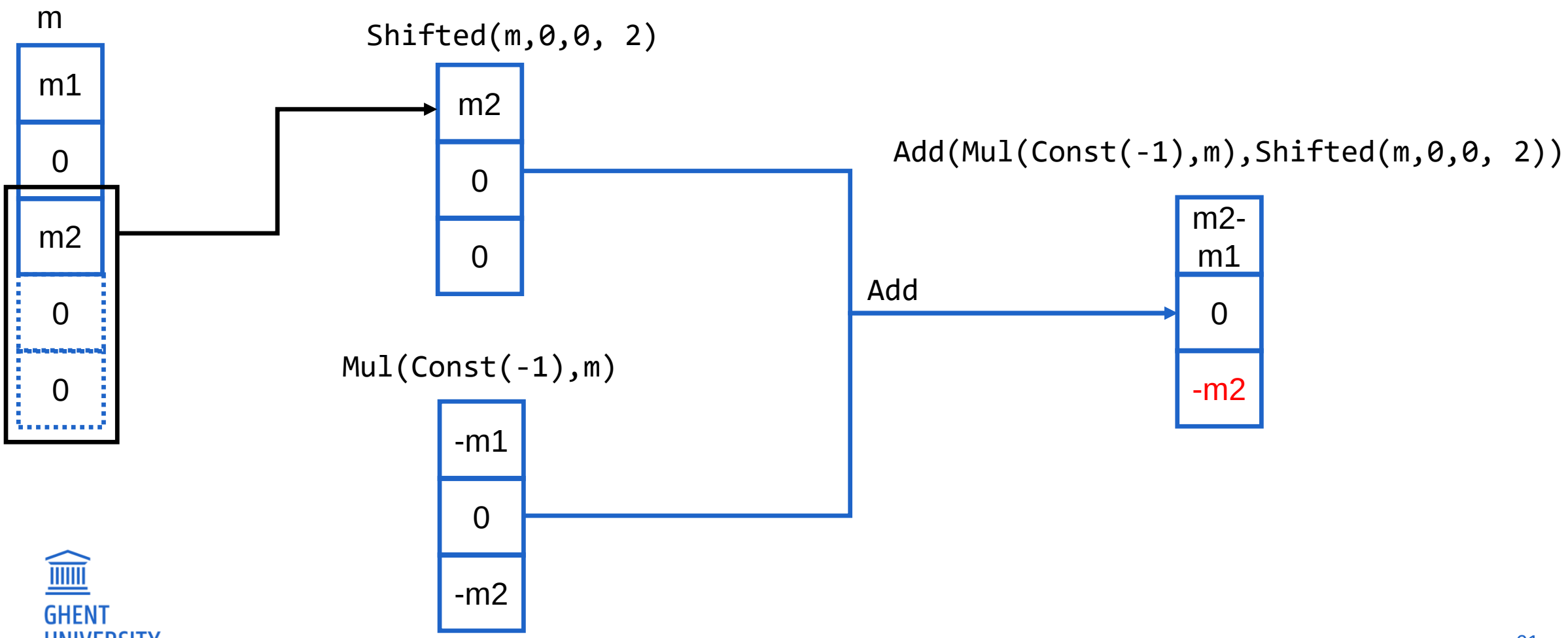
```
//Custom Fields implementation for exchange between the 2 FM layers

cellsize:=1e-9
AFMAex:=-5e-13
Ms:=580e3
prefactorZ := Const( (2 * AFMAex) / (cellsize*cellsize*Ms))

up   :=   Mul(Add(Mul(Const(-1),m),Shifted(m,0,0, 2)),Shifted(Const(1),0,0, 2))
down :=   Mul(Add(Mul(Const(-1),m),Shifted(m,0,0,-2)),Shifted(Const(1),0,0,-2))

Bc :=Mul(prefactorZ,Add(up,down))

AddFieldTerm(Bc)
addEdensTerm(Mul(Const(-0.5),Dot(Bc,M_full)))
```

$$\vec{\mathbf{B}}_{\text{exch}} = 2\frac{A_{\text{ex}}}{M_{\text{sat}}}\sum_i \frac{\left(\vec{\mathbf{m}}_i - \vec{\mathbf{m}}\right)}{\Delta_i^2}$$

GHENT UNIVERSITY

SKYRMION RACETRACK IN SYNTHETIC ANTIFERROMAGNET

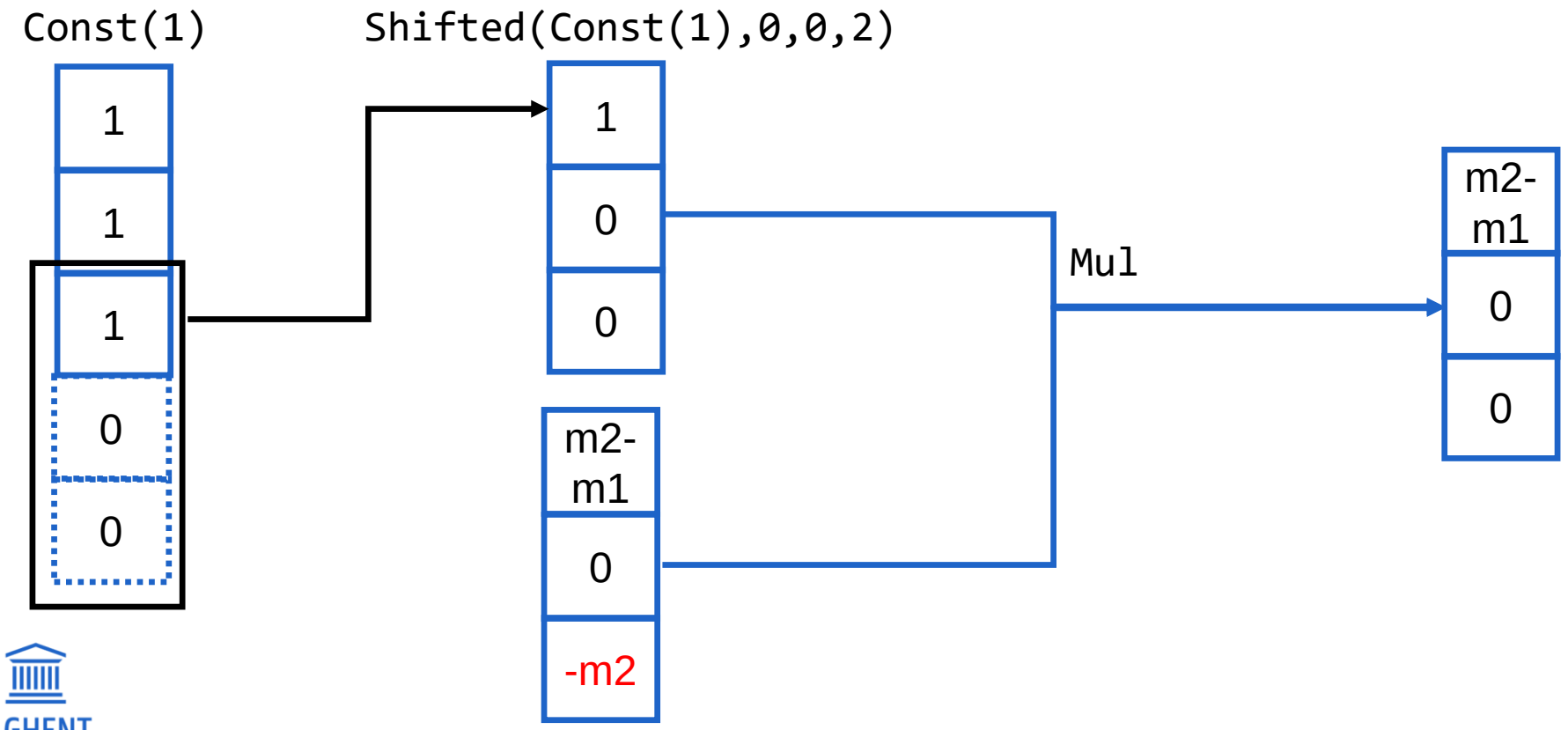
```
//what's happening here?
up :=          Mul(Add(Mul(Const(-1),m),Shifted(m,0,0, 2)),Shifted(Const(1),0,0,2))
```



GHENT UNIVERSITY

SKYRMION RACETRACK IN SYNTHETIC ANTIFERROMAGNET

```
//what's happening here?
up :=        Mul(Add(Mul(Const(-1),m),Shifted(m,0,0, 2)),Shifted(Const(1),0,0,2))
```

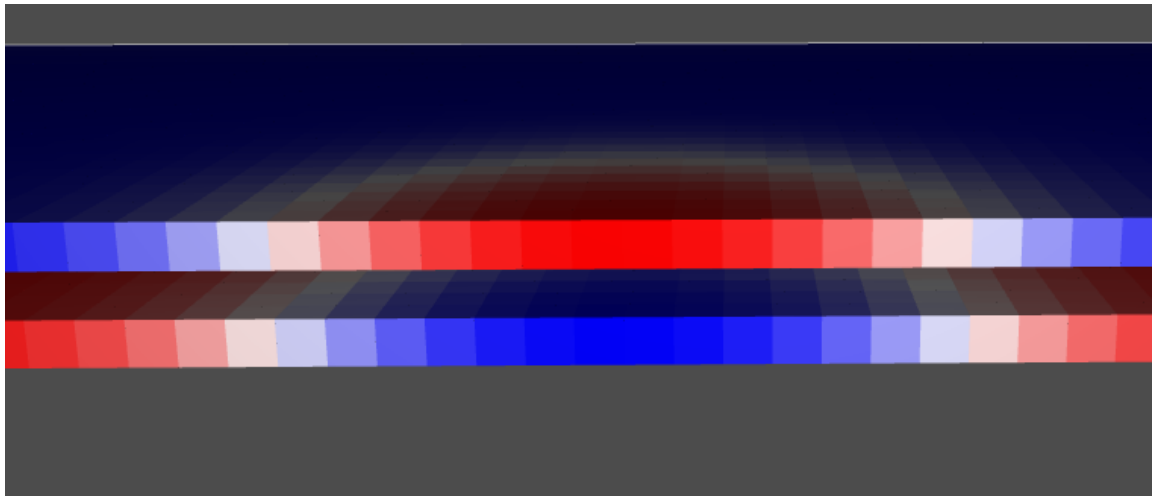


Add(Mul(Const(-1),m),Shifted(m,0,0, 2))

GHENT UNIVERSITY

SKYRMION RACETRACK IN SYNTHETIC ANTIFERROMAGNET

```
// define constants and set slonczewksi parameters
SOTxi                := -2
AlphaH               := 0.15
Pol                   = alphaH
Lambda                = 1
Epsilonprime          = alphaH*SOTxi/2
Fixedlayer            = vector(0,1,0)
FREELAYERTHICKNESS    = 2.0e-9

// define current
J  = vector(0,0,-2e11)


autosave(m,1e-10)
tableAutosave(1e-11)
tableAdd(ext_bubblepos)


run(2e-9)
```



GHENT UNIVERSITY

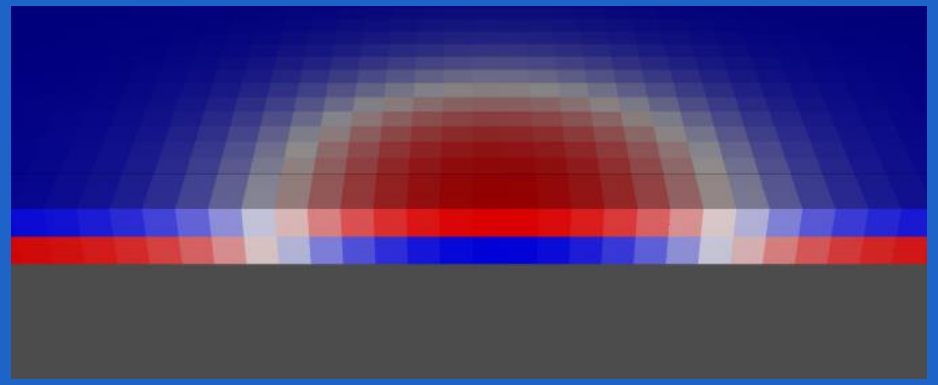
SKYRMION RACETRACK REVISITED

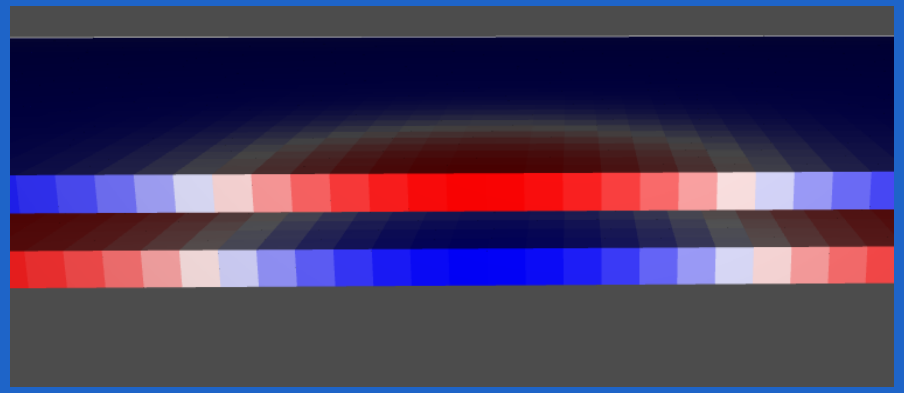
WHAT HAVE WE LEARNED?

- Interlayer exchange coupling
- Custom fields exchange coupling to explicitly include spacer layers

Input file: session4_example3a.txt
Titan RTX: 86 s

Input file: session4_example3b.txt
Titan RTX: 7 min





GHENT UNIVERSITY

POST PROCESSING (IN PYTHON)

This section will be presented in a Jupyter notebook which can be found on https://mumax.ugent.be/mumax3-workshop/

GHENT UNIVERSITY

FACULTY OF SCIENCES

mumax.ugent.be

jonathan.leliaert@ugent.be

jeroen.mulkers@ugent.be

GHENT UNIVERSITY

IEEE MAGNETICS

NVIDIA

CSPINTRONICS