A GPU simulation of skyrmion

(Dated: October 11, 2018)

A GPU simulation of skyrmion.

I. LLG

The Landau-Lifshitz-Gilbert-Equation (LLG) can be written as (Eq. (13) of Ref. [1] and Eq. (7) of Ref. [2], NOTE that the sign of the last term is '+' in Eq. (7) of Ref. [2])

$$\dot{\mathbf{n}} = \gamma \mathbf{B}_{\text{eff}} \times \mathbf{n} - \frac{\alpha\gamma}{|n|}\mathbf{n} \times \dot{\mathbf{n}} - \frac{\hbar\gamma}{2e}\left(\mathbf{j}\cdot\nabla\right)\mathbf{n} \tag{1}$$

where $\mathbf{n}$ is the magnetic momentum, $\gamma(\gamma > 0)$ is the gyromagnetic ratio, α represents Gilbert damping, $\mathbf{B}_{\text{eff}}$ is the effective field arising from the spin Hamiltonian, and can be written as (H_S is from Ref. [2], before Eq. (1), also Eq. (4) in Ref. [3])

$$\begin{aligned} \mathbf{B}_{\text{eff}} &\equiv \frac{\delta H_S}{\delta \mathbf{n}} \\ H_S &= \int d^D x \frac{J}{2a}(\nabla \mathbf{n})^2 + \frac{D}{a^2}\mathbf{n}\cdot(\nabla\times\mathbf{n}) - \frac{\mu}{a^3}\mathbf{B}\cdot\mathbf{n} \end{aligned} \tag{2}$$

II. LLG IN 2D LATTICE

In the following, for simplicity, we use $\delta = a(\mathbf{e}_x, \mathbf{e}_y)$, $\delta_x = a\mathbf{e}_x, \delta_y = a\mathbf{e}_y$, where a is the distance between two lattice.

A. Spin torque

Written in lattice, so that

$$\left(\mathbf{j}\cdot\nabla\right)\mathbf{n} = \sum_i j_i \partial_i n_x \mathbf{e}_x + \sum_i j_i \partial_i n_y \mathbf{e}_y + \sum_i j_i \partial_i n_z \mathbf{e}_z \tag{3}$$

In 2D, it is

$$\begin{aligned} &\left(\mathbf{j}\cdot\nabla\right)\mathbf{n} = \sum_{i=x,y} j_i(\partial_i n_x \mathbf{e}_x + \partial_i n_y \mathbf{e}_y + \partial_i n_z \mathbf{e}_z) \\ &= \frac{1}{a}\sum_{i=x,y} j_i\Big(\frac{n_x(\mathbf{r}+\delta_i) - n_x(\mathbf{r}-\delta_i)}{2}\mathbf{e}_x + \frac{n_y(\mathbf{r}+\delta_i) - n_y(\mathbf{r}-\delta_i)}{2}\mathbf{e}_y + \frac{n_z(\mathbf{r}+\delta_i) - n_z(\mathbf{r}-\delta_i)}{2}\mathbf{e}_z\Big) \\ &= \frac{1}{a}\sum_{i=x,y} j_i \frac{\mathbf{n}(\mathbf{r}+\delta_i) - \mathbf{n}(\mathbf{r}-\delta_i)}{2} \end{aligned} \tag{4}$$

Sometimes, it is also written as (see last term in Eq. (8) in Ref. [4], it also says this is the discrete version of the continuous term $(\mathbf{j}\cdot\nabla)\mathbf{n}$ before Eq. (10))

$$\left(\mathbf{j}\cdot\nabla\right)\mathbf{n} = \frac{1}{a}\sum_{i=x,y} j_i \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r}+\delta_i) - \mathbf{n}(\mathbf{r}-\delta_i)}{2} \times \mathbf{n}(\mathbf{r})\right) \tag{5}$$

that is because, for a unit vector, one have

$$\mathbf{n}\cdot\partial_i \mathbf{n} = 0 \tag{6}$$

the discrete version is

$$\mathbf{n}(\mathbf{r})\cdot\frac{\mathbf{n}(\mathbf{r}+\delta_\mathbf{i}) - \mathbf{n}(\mathbf{r}-\delta_\mathbf{i})}{\mathbf{2a}} = \mathbf{0} \tag{7}$$

so that

$$
\begin{aligned}
&\mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r}+\delta_i) - \mathbf{n}(\mathbf{r}-\delta_i)}{2} \times \mathbf{n}(\mathbf{r}) \right) \\
&= \frac{\mathbf{n}(\mathbf{r}+\delta_i) - \mathbf{n}(\mathbf{r}-\delta_i)}{2} (\mathbf{n}(\mathbf{r}) \cdot \mathbf{n}(\mathbf{r})) - \mathbf{n}(\mathbf{r}) \left(\mathbf{n}(\mathbf{r}) \cdot \frac{\mathbf{n}(\mathbf{r}+\delta_i) - \mathbf{n}(\mathbf{r}-\delta_i)}{2} \right) \\
&= \frac{\mathbf{n}(\mathbf{r}+\delta_i) - \mathbf{n}(\mathbf{r}-\delta_i)}{2}
\end{aligned} \tag{8}
$$

to be consist with the references, we use

$$
\left(\mathbf{j} \cdot \nabla\right) \mathbf{n} = \frac{1}{a} \sum_{i=x,y} j_i \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r}+\delta_i) - \mathbf{n}(\mathbf{r}-\delta_i)}{2} \times \mathbf{n}(\mathbf{r}) \right) \tag{9}
$$

In simulation, to simplify, we always take the direction of $\mathbf{j}$ as $\mathbf{x}$-axis, so we only have

$$
\frac{1}{a} j_x \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r}+\delta_x) - \mathbf{n}(\mathbf{r}-\delta_x)}{2} \times \mathbf{n}(\mathbf{r}) \right) \tag{10}
$$

B. J term

$(\nabla \mathbf{n})^2$ is confusing, in 2D, it is in fact $\sum_{i=x,y} (\partial_i \mathbf{n}) \cdot (\partial_i \mathbf{n})$ Consider in 1-dimension, we have

$$
\begin{aligned}
&\mathbf{n}(x) \cdot \partial_i \mathbf{n}(x) = 0 \\
&0 = \sum_{i=x,y} \partial_i \left(\mathbf{n}(x) \cdot \partial_i \mathbf{n}(x)\right) = \sum_{i=x,y} (\partial_i \mathbf{n}) \cdot (\partial_i \mathbf{n}) + \mathbf{n} \cdot (\sum_{i=x,y} (\partial_i)^2) \mathbf{n}
\end{aligned} \tag{11}
$$

In lattice, the derivate can be written as

$$
\frac{d^2}{dx^2} \mathbf{n}(\mathbf{r}) = \frac{1}{a} \left(\mathbf{n}'(\mathbf{r} + \frac{a}{2} \mathbf{e}_x) - \mathbf{n}'(\mathbf{r} - \frac{a}{2} \mathbf{e}_x) \right) \tag{12}
$$

with

$$
\begin{aligned}
\mathbf{n}'(\mathbf{r} + \frac{a}{2} \mathbf{e}_x) &= \frac{1}{a} \left(\mathbf{n}(\mathbf{r} + a\mathbf{e}_x) - \mathbf{n}(\mathbf{r}) \right) \\
\mathbf{n}'(\mathbf{r} - \frac{a}{2} \mathbf{e}_x) &= \frac{1}{a} \left(\mathbf{n}(\mathbf{r}) - \mathbf{n}(\mathbf{r} - a\mathbf{e}_x) \right)
\end{aligned} \tag{13}
$$

so that

$$
(\sum_{i=x,y} (\partial_i)^2) \mathbf{n} = \frac{1}{a^2} \sum_{i=x,y,-x,-y} \mathbf{n}(\mathbf{r} + \delta_i) - 4\mathbf{n}(\mathbf{r}) \tag{14}
$$

and

$$
\mathbf{n} \cdot (\sum_{i=x,y} (\partial_i)^2) \mathbf{n} = \frac{1}{a^2} \sum_{\langle i \rangle} \mathbf{n} \cdot \mathbf{n}_i - 4 \tag{15}
$$

Throw away the constant term

$$
\int d^D x \frac{J}{2a} (\nabla \mathbf{n})^2 = -\frac{J}{2a^3} \sum_{\langle i,j \rangle} \mathbf{n}_i \cdot \mathbf{n}_j \tag{16}
$$

where j are all neighbours of i, $j = i + \delta_x, i - \delta_x, i + \delta_y, i - \delta_y$.

When J is constant, it can also been written as

$$
\int d^D x \frac{J}{2a} (\nabla \mathbf{n})^2 = -\frac{J}{a^3} \sum_{\mathbf{r}} \mathbf{n}(\mathbf{r}) \cdot \left(\mathbf{n}(\mathbf{r} + \delta_x) + \mathbf{n}(\mathbf{r} + \delta_y) \right) \tag{17}
$$

C. D term

In 2D, we find

$$
\begin{aligned}
&\mathbf{n}\cdot(\nabla\times\mathbf{n}) = n_x\partial_y n_z - n_y\partial_x n_z + n_z\partial_x n_y - n_z\partial_y n_x \\
&= \frac{1}{2a}\left[(n_x(\mathbf{r})(n_z(\mathbf{r}+\delta_y) - n_z(\mathbf{r}-\delta_y)) - n_z(\mathbf{r})(n_x(\mathbf{r}+\delta_y) - n_x(\mathbf{r}-\delta_y))\right) \\
&+ (n_z(\mathbf{r})(n_y(\mathbf{r}+\delta_x) - n_y(\mathbf{r}-\delta_x)) - n_y(\mathbf{r})(n_z(\mathbf{r}+\delta_x) - n_z(\mathbf{r}-\delta_x)))] \\
&= \frac{1}{2a}\left[(n_x(\mathbf{r})n_z(\mathbf{r}+\delta_y) - n_z(\mathbf{r})n_x(\mathbf{r}+\delta_y)) - (n_x(\mathbf{r})n_z(\mathbf{r}-\delta_y) - n_z(\mathbf{r})n_x(\mathbf{r}-\delta_y)\right) \\
&+ (n_z(\mathbf{r})n_y(\mathbf{r}+\delta_x) - n_y(\mathbf{r})n_z(\mathbf{r}+\delta_x)) - (n_z(\mathbf{r})n_y(\mathbf{r}-\delta_x) - n_y(\mathbf{r})n_z(\mathbf{r}-\delta_x))] \\
&= -\frac{1}{2a}\left[\mathbf{n}(\mathbf{r})\cdot(\mathbf{n}(\mathbf{r}+\delta_y)\times\mathbf{e}_y) - \mathbf{n}(\mathbf{r})\cdot(\mathbf{n}(\mathbf{r}-\delta_y)\times\mathbf{e}_y\right) \\
&+\mathbf{n}(\mathbf{r})\cdot(\mathbf{n}(\mathbf{r}+\delta_x)\times\mathbf{e}_x) - \mathbf{n}(\mathbf{r})\cdot(\mathbf{n}(\mathbf{r}-\delta_x)\times\mathbf{e}_x)] \\
&= -\frac{1}{2a^2}\left[\mathbf{n}(\mathbf{r})\times\mathbf{n}(\mathbf{r}+\delta_y)\cdot\delta_y - \mathbf{n}(\mathbf{r})\times\mathbf{n}(\mathbf{r}-\delta_y)\cdot\delta_y\right. \\
&+\mathbf{n}(\mathbf{r})\times\mathbf{n}(\mathbf{r}+\delta_x)\cdot\delta_x - \mathbf{n}(\mathbf{r})\times\mathbf{n}(\mathbf{r}-\delta_x)\cdot\delta_x]
\end{aligned}
\tag{18}
$$

so, one have

$$
\int d^D x \frac{D}{a^2}\mathbf{n}\cdot(\nabla\times\mathbf{n}) = -\frac{D}{a^4}\sum_{\mathbf{r}}\left(\mathbf{n}(\mathbf{r})\times\mathbf{n}(\mathbf{r}+\delta_x)\cdot\delta_x + \mathbf{n}(\mathbf{r})\times\mathbf{n}(\mathbf{r}+\delta_y)\cdot\delta_y\right) \tag{19}
$$

It is also

$$
\int d^D x \frac{D}{a^2}\mathbf{n}\cdot(\nabla\times\mathbf{n}) = -\frac{D}{a^4}\sum_{\mathbf{r}}\left(\mathbf{n}(\mathbf{r}+\delta_x)\times\delta_x + \mathbf{n}(\mathbf{r}+\delta_y)\times\delta_y\right)\cdot\mathbf{n}(\mathbf{r}) \tag{20}
$$

D. Effective magnetic field

Using the results, we can write the discrete version of H_s as

$$
H_S = \sum_{\mathbf{r}}\left[-\frac{J}{a^3}(\mathbf{n}(\mathbf{r}+\delta_x)+\mathbf{n}(\mathbf{r}+\delta_y)) - \frac{D}{a^3}(\mathbf{n}(\mathbf{r}+\delta_x)\times\mathbf{e}_x + \mathbf{n}(\mathbf{r}+\delta_y)\times\mathbf{e}_y) - \frac{\mu}{a^3}\mathbf{B}\right]\cdot\mathbf{n}(\mathbf{r}) \tag{21}
$$

which leads to

$$
\begin{aligned}
\mathbf{B}_{\text{eff}}(\mathbf{r}) = \frac{\delta H_S}{\delta\mathbf{n}} &= -\frac{1}{a^3}\sum_{i=x,y}\left[J(\mathbf{r})\mathbf{n}(\mathbf{r}+\delta_i) + J(\mathbf{r}-\delta_i)\mathbf{n}(\mathbf{r}-\delta_i)\right] \\
-\frac{1}{a^3}\sum_{i=x,y}&\left[D(\mathbf{r})\mathbf{n}(\mathbf{r}+\delta_i)\times\mathbf{e}_i - D(\mathbf{r}-\delta_i)\mathbf{n}(\mathbf{r}-\delta_i)\times\mathbf{e}_i\right] - \frac{\mu}{a^3}\mathbf{B}(\mathbf{r})
\end{aligned}
\tag{22}
$$

E. anisotropy

In some material, the effective Hamiltonian can be written with anisotropy terms as (if ignore a, this is as same as Eq. (10) in Ref. [5])

$$
\begin{aligned}
H_S = \sum_{\mathbf{r}}&\left[-\frac{J}{a^3}(\mathbf{n}(\mathbf{r}+\delta_x)+\mathbf{n}(\mathbf{r}+\delta_y)) - \frac{D}{a^3}(\mathbf{n}(\mathbf{r}+\delta_x)\times\mathbf{e}_x + \mathbf{n}(\mathbf{r}+\delta_y)\times\mathbf{e}_y) - \frac{\mu}{a^3}\mathbf{B}\right. \\
&\left.-\mathbf{h}\right]\cdot\mathbf{n}(\mathbf{r}) - K\left(\mathbf{e}_z\cdot\mathbf{n}(\mathbf{r})\right)^2
\end{aligned}
\tag{23}
$$

The contribution of $\mathbf{h}$ can be just put into the applied magnetic field $\mathbf{B}$, we also include the contribution of K.

F. dimensionless LLG

Using $\tau = \gamma t$

$$\frac{d}{d\tau}\mathbf{n} = \mathbf{B}_{\text{eff}} \times \mathbf{n} - \frac{\alpha\gamma}{|n|}\mathbf{n} \times \dot{\mathbf{n}} - \frac{\hbar}{2e}\left(\mathbf{j} \cdot \nabla\right)\mathbf{n} \tag{24}$$

In the following, we use $t \to \tau$, and use dimensionless parameters.
Using dimensionless parameters, the LLG can be written as

$$\begin{aligned}
\dot{\mathbf{n}} &= \mathbf{B}_{\text{eff}} \times \mathbf{n} - \alpha\mathbf{n} \times \dot{\mathbf{n}} - \sum_{i=x,y} j_i \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r}+\delta_i) - \mathbf{n}(\mathbf{r}-\delta_i)}{2} \times \mathbf{n}(\mathbf{r})\right) \\
\mathbf{B}_{\text{eff}}(\mathbf{r}) &= \frac{\delta H_S}{\delta\mathbf{n}} = -\sum_{i=x,y}\left[J(\mathbf{r})\mathbf{n}(\mathbf{r}+\delta_i) + J(\mathbf{r}-\delta_i)\mathbf{n}(\mathbf{r}-\delta_i)\right] \\
&- \sum_{i=x,y}\left[D(\mathbf{r})\mathbf{n}(\mathbf{r}+\delta_i) \times \mathbf{e}_i - D(\mathbf{r}-\delta_i)\mathbf{n}(\mathbf{r}-\delta_i) \times \mathbf{e}_i\right] - \mathbf{B}(\mathbf{r}) - 2K\left(\mathbf{e}_z \cdot \mathbf{n}(\mathbf{r})\right)\mathbf{e}_z
\end{aligned} \tag{25}$$

Ignoring the anisotropy term, this is as same as Eq. (9) in Ref. [4].

III. EVALUATION

Let $\mathbf{N} = \mathbf{B}_{\text{eff}} \times \mathbf{n} - \sum_{i=x,y} j_i \mathbf{n}(\mathbf{r}) \times \left(\frac{\mathbf{n}(\mathbf{r}+\delta_i) - \mathbf{n}(\mathbf{r}-\delta_i)}{2} \times \mathbf{n}(\mathbf{r})\right)$, we have

$$\dot{\mathbf{n}} = \mathbf{N} - \alpha\mathbf{n} \times \dot{\mathbf{n}} \tag{26}$$

This is in fact a combine of 3 equations which can be written as

$$\frac{d\mathbf{n}}{dt} = \frac{1}{1+\alpha^2} \times \left(N_x + \alpha(N_y n_z - n_y N_z) + \alpha^2 n_x \mathbf{n}\cdot\mathbf{N}, N_y + \alpha(n_x N_z - N_x n_z) + \alpha^2 n_y \mathbf{n}\cdot\mathbf{N}, N_z + \alpha(N_x n_y - n_x N_y) + \alpha^2 n_z \mathbf{n}\cdot\mathbf{N}\right) \tag{27}$$

It can be written as

$$\frac{d\mathbf{n}}{dt} = \frac{1}{1+\alpha^2}\left(\mathbf{N} + \alpha\mathbf{N} \times \mathbf{n} + \alpha^2(\mathbf{n}\cdot\mathbf{N})\mathbf{n}\right) \tag{28}$$

Then note that $\mathbf{n} \cdot \frac{d\mathbf{n}}{dt} = 0$ because it is a unit vector, so

$$\mathbf{n} \cdot \frac{d\mathbf{n}}{dt} = 0 = \mathbf{n} \cdot \mathbf{N}. \tag{29}$$

So

$$\frac{d\mathbf{n}}{dt} = \frac{1}{1+\alpha^2}\left(\mathbf{N} + \alpha\mathbf{N} \times \mathbf{n}\right) \tag{30}$$

One can use this to evaluate

$$\mathbf{n}(t+\Delta t) = \mathbf{n}(t) + \Delta t\left(\mathbf{N}(t) + \alpha\mathbf{N}(t) \times \mathbf{n}(t)\right) \tag{31}$$

IV. RUNGECKUTTA

The Eq. (31) can be easily implemented, however, it is better to evaluate using Runge-Kutta method [6]. Using RK4, one need to calculate dn/dt for 4 times, however, a compare between using RK4 and using Eq. (31) with 1/4 time-step (approximately same consumption) shows RK4 is better.

```
In[4]:= DSolve[{y'[t] == Sin[t]^2 y[t], y[0] == 1}, y, t] // StandardForm
Out[4]//StandardForm=
        {{y → Function[{t}, e^(t/2 - 1/4 Sin[2 t])]}}

In[30]:= resNormal1X = Table[i*0.1, {i, 0, 60}];
         resNormal1Y = Table[1, {i, 0, 60}];
         For[i = 1, i < 61, i++,
          resNormal1Y[[i + 1]] = resNormal1Y[[i]] + 0.1* Sin[(i - 1) *0.1]^2 *resNormal1Y[[i]]]

In[33]:= resNormal2X = Table[i*0.025, {i, 0, 240}];
         resNormal2Y = Table[1, {i, 0, 240}];
         For[i = 1, i < 241, i++,
          resNormal2Y[[i + 1]] = resNormal2Y[[i]] + 0.025* Sin[(i - 1) *0.025]^2 *resNormal2Y[[i]]]

In[37]:= resNormal3X = Table[i*0.1, {i, 0, 60}];
         resNormal3Y = Table[1, {i, 0, 60}];
         For[i = 1, i < 61, i++,
          tn = (i - 1) *0.1;
          yn = resNormal3Y[[i]];
          k1 = 0.1*Sin[tn]^2 yn;
          k2 = 0.1*Sin[tn + 0.05]^2 (yn + 0.5*k1);
          k3 = 0.1*Sin[tn + 0.05]^2 (yn + 0.5*k2);
          k4 = 0.1*Sin[tn + 0.1]^2 (yn + k3);
          resNormal3Y[[i + 1]] = yn + (k1 + 2 k2 + 2 k3 + k4)/6
         ]

In[50]:= Show[
          {ListPlot[Transpose[{resNormal3X, resNormal3Y}], PlotMarkers → {"+", Large}, PlotStyle → Red],
           ListPlot[Transpose[{resNormal2X, resNormal2Y}], PlotMarkers → {"*"}, PlotStyle → Black],
           ListPlot[Transpose[{resNormal1X, resNormal1Y}]],
           Plot[e^(t/2 - 1/4 Sin[2 t]), {t, 0, 6}]}, PlotRange → All]

Out[50]=
```

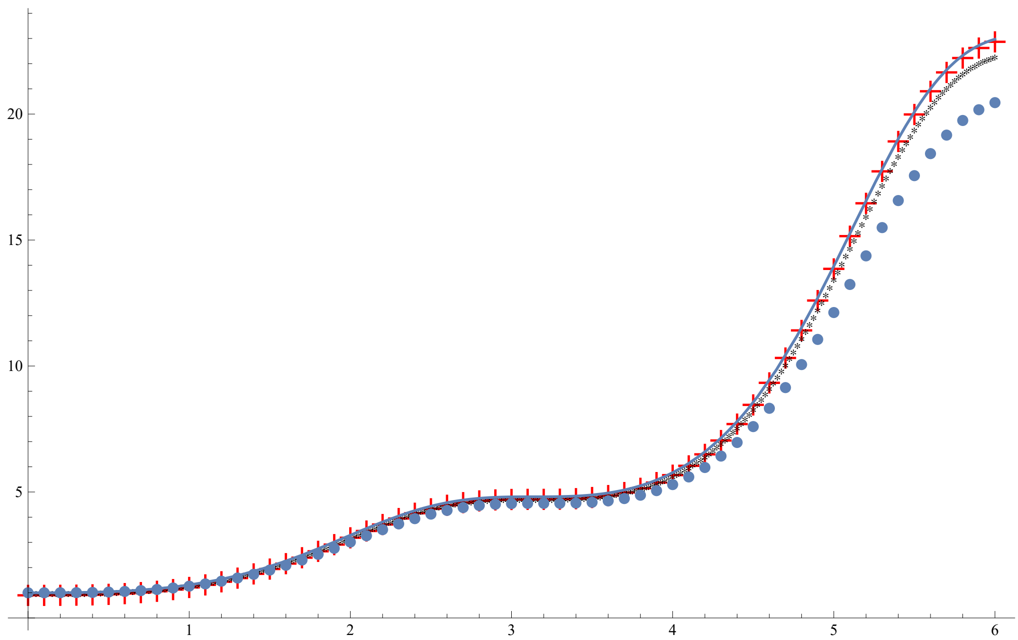

FIG. 1: Compare between RK4 and Eq. (31). The red "+" is numerical result obtained using RK4 which is close to the exact result.

V. SIMULATION

The simulation is running on GPU using the compute shader in Unity3D. The implementation of LLG is the LLG_H.compute, the content is

```
// Each #kernel tells which function to compile; you can have many kernels
#pragma kernel CaclK1

// Create a RenderTexture with enableRandomWrite flag and set it
// with cs.SetTexture
//RWStructuredBuffer<float3> magneticMomentum;
//Using RFloat texture format so we do not need a float4.
RWTexture2D<float> magneticMomentumX;
RWTexture2D<float> magneticMomentumY;
RWTexture2D<float> magneticMomentumZ;

//1024 x 1024
//0,0-512,512 is k1
//512,0-1024,512 is k2
//0,512-512,1024 is k3
RWTexture2D<float> k1x;
RWTexture2D<float> k1y;
RWTexture2D<float> k1z;

Texture2D<float4> boundaryCondition;
Texture2D<float> exchangeStrength;
Texture2D<float> jxPeroidFunction;

uint2 size;
float K;
float D;
float D0;
float B;
float alpha;
float timestep;
uint jxstep;
uint jxperoid;

[numthreads(8, 8, 1)]
void CaclK1(uint3 id : SV_DispatchThreadID)
{
    float3 zero3 = float3(0.0f, 0.0f, 0.0f);

    float3 s = float3(magneticMomentumX[id.xy], magneticMomentumY[id.xy], magneticMomentumZ[id.xy]);

    float3 sleft = id.x > 1 ? float3(magneticMomentumX[id.xy - uint2(1, 0)], magneticMomentumY[id.xy - uint2(1,
        0)], magneticMomentumZ[id.xy - uint2(1, 0)]) : zero3;
    float3 sright = id.x < (size.x - 1) ? float3(magneticMomentumX[id.xy + uint2(1, 0)], magneticMomentumY[id.xy
        + uint2(1, 0)], magneticMomentumZ[id.xy + uint2(1, 0)]) : zero3;
    float3 sdown = id.y > 1 ? float3(magneticMomentumX[id.xy - uint2(0, 1)], magneticMomentumY[id.xy - uint2(0,
        1)], magneticMomentumZ[id.xy - uint2(0, 1)]) : zero3;
    float3 sup = id.y < (size.y - 1) ? float3(magneticMomentumX[id.xy + uint2(0, 1)], magneticMomentumY[id.xy +
        uint2(0, 1)], magneticMomentumZ[id.xy + uint2(0, 1)]) : zero3;

    float j_s = exchangeStrength[id.xy];
    float j_left = id.x > 1 ? exchangeStrength[id.xy - uint2(1, 0)] : 0.0f;
    float j_down = id.y > 1 ? exchangeStrength[id.xy - uint2(0, 1)] : 0.0f;

    float d_s = D0 + D * j_s;
    float d_left = id.x > 1 ? (D0 + D * j_left) : 0.0f;
    float d_down = id.y > 1 ? (D0 + D * j_down) : 0.0f;

    float3 vright = float3(1.0, 0.0, 0.0);
    float3 vup = float3(0.0, 1.0, 0.0);

    float3 beff = (j_left * sleft + j_s * sright + j_down * sdown + j_s * sup)
        + (d_s * cross(sright, vright) - d_left * cross(sleft, vright) + d_s * cross(sup, vup) - d_down * cross(
            sdown, vup))
        + float3(0.0f, 0.0f, B) + 2.0 * K * float3(0.0f, 0.0f, s.z);
```

```

    //t is now
    float jx = jxperoid > 0 ? jxPeroidFunction[uint2(jxstep % jxperoid, 0)] : 0.0f;
    float3 stt = -jx * cross(s, cross((sright - sleft) * 0.5f, s));

    float3 newS = cross(s, beff) + stt;
    newS = (newS - alpha * cross(s, newS)) / (1 + alpha * alpha);

    float3 k1res = timestep * newS;

    k1x[id.xy] = k1res.x;
    k1y[id.xy] = k1res.y;
    k1z[id.xy] = k1res.z;
}

#pragma kernel CaclK2

[numthreads(8, 8, 1)]
void CaclK2(uint3 id : SV_DispatchThreadID)
{
    float3 zero3 = float3(0.0f, 0.0f, 0.0f);

    float3 s = float3(magneticMomentumX[id.xy] + 0.5 * k1x[id.xy],
                      magneticMomentumY[id.xy] + 0.5 * k1y[id.xy],
                      magneticMomentumZ[id.xy] + 0.5 * k1z[id.xy]);

    float3 sleft = id.x > 1 ?
        float3(magneticMomentumX[id.xy - uint2(1, 0)] + 0.5 * k1x[id.xy - uint2(1, 0)],
               magneticMomentumY[id.xy - uint2(1, 0)] + 0.5 * k1y[id.xy - uint2(1, 0)],
               magneticMomentumZ[id.xy - uint2(1, 0)] + 0.5 * k1z[id.xy - uint2(1, 0)]) : zero3;
    float3 sright = id.x < (size.x - 1) ?
        float3(magneticMomentumX[id.xy + uint2(1, 0)] + 0.5 * k1x[id.xy + uint2(1, 0)],
               magneticMomentumY[id.xy + uint2(1, 0)] + 0.5 * k1y[id.xy + uint2(1, 0)],
               magneticMomentumZ[id.xy + uint2(1, 0)] + 0.5 * k1z[id.xy + uint2(1, 0)]) : zero3;
    float3 sdown = id.y > 1 ?
        float3(magneticMomentumX[id.xy - uint2(0, 1)] + 0.5 * k1x[id.xy - uint2(0, 1)],
               magneticMomentumY[id.xy - uint2(0, 1)] + 0.5 * k1y[id.xy - uint2(0, 1)],
               magneticMomentumZ[id.xy - uint2(0, 1)] + 0.5 * k1z[id.xy - uint2(0, 1)]) : zero3;
    float3 sup = id.y < (size.y - 1) ?
        float3(magneticMomentumX[id.xy + uint2(0, 1)] + 0.5 * k1x[id.xy + uint2(0, 1)],
               magneticMomentumY[id.xy + uint2(0, 1)] + 0.5 * k1y[id.xy + uint2(0, 1)],
               magneticMomentumZ[id.xy + uint2(0, 1)] + 0.5 * k1z[id.xy + uint2(0, 1)]) : zero3;

    float j_s = exchangeStrength[id.xy];
    float j_left = id.x > 1 ? exchangeStrength[id.xy - uint2(1, 0)] : 0.0f;
    float j_down = id.y > 1 ? exchangeStrength[id.xy - uint2(0, 1)] : 0.0f;

    float d_s = D0 + D * j_s;
    float d_left = id.x > 1 ? (D0 + D * j_left) : 0.0f;
    float d_down = id.y > 1 ? (D0 + D * j_down) : 0.0f;

    float3 vright = float3(1.0, 0.0, 0.0);
    float3 vup = float3(0.0, 1.0, 0.0);

    float3 beff = (j_left * sleft + j_s * sright + j_down * sdown + j_s * sup)
        + (d_s * cross(sright, vright) - d_left * cross(sleft, vright) + d_s * cross(sup, vup) - d_down * cross(
            sdown, vup))
        + float3(0.0f, 0.0f, B) + 2.0 * K * float3(0.0f, 0.0f, s.z);

    //t is t + 0.5 dt
    float jx = jxperoid > 0 ? jxPeroidFunction[uint2((jxstep + 1) % jxperoid, 0)] : 0.0f;
    float3 stt = -jx * cross(s, cross((sright - sleft) * 0.5f, s));

    float3 newS = cross(s, beff) + stt;
    newS = (newS - alpha * cross(s, newS)) / (1 + alpha * alpha);

    float3 k2res = timestep * newS;

    k1x[id.xy + uint2(512, 0)] = k2res.x;
    k1y[id.xy + uint2(512, 0)] = k2res.y;
```

```
    k1z[id.xy + uint2(512, 0)] = k2res.z;
}

#pragma kernel CaclK3

[numthreads(8, 8, 1)]
void CaclK3(uint3 id : SV_DispatchThreadID)
{
    float3 zero3 = float3(0.0f, 0.0f, 0.0f);

    float3 s = float3(magneticMomentumX[id.xy] + 0.5 * k1x[id.xy + uint2(512, 0)],
                      magneticMomentumY[id.xy] + 0.5 * k1y[id.xy + uint2(512, 0)],
                      magneticMomentumZ[id.xy] + 0.5 * k1z[id.xy + uint2(512, 0)]);

    float3 sleft = id.x > 1 ?
        float3(magneticMomentumX[id.xy - uint2(1, 0)] + 0.5 * k1x[id.xy - uint2(1, 0) + uint2(512, 0)],
               magneticMomentumY[id.xy - uint2(1, 0)] + 0.5 * k1y[id.xy - uint2(1, 0) + uint2(512, 0)],
               magneticMomentumZ[id.xy - uint2(1, 0)] + 0.5 * k1z[id.xy - uint2(1, 0) + uint2(512, 0)]) : zero3;
    float3 sright = id.x < (size.x - 1) ?
        float3(magneticMomentumX[id.xy + uint2(1, 0)] + 0.5 * k1x[id.xy + uint2(1, 0) + uint2(512, 0)],
               magneticMomentumY[id.xy + uint2(1, 0)] + 0.5 * k1y[id.xy + uint2(1, 0) + uint2(512, 0)],
               magneticMomentumZ[id.xy + uint2(1, 0)] + 0.5 * k1z[id.xy + uint2(1, 0) + uint2(512, 0)]) : zero3;
    float3 sdown = id.y > 1 ?
        float3(magneticMomentumX[id.xy - uint2(0, 1)] + 0.5 * k1x[id.xy - uint2(0, 1) + uint2(512, 0)],
               magneticMomentumY[id.xy - uint2(0, 1)] + 0.5 * k1y[id.xy - uint2(0, 1) + uint2(512, 0)],
               magneticMomentumZ[id.xy - uint2(0, 1)] + 0.5 * k1z[id.xy - uint2(0, 1) + uint2(512, 0)]) : zero3;
    float3 sup = id.y < (size.y - 1) ?
        float3(magneticMomentumX[id.xy + uint2(0, 1)] + 0.5 * k1x[id.xy + uint2(0, 1) + uint2(512, 0)],
               magneticMomentumY[id.xy + uint2(0, 1)] + 0.5 * k1y[id.xy + uint2(0, 1) + uint2(512, 0)],
               magneticMomentumZ[id.xy + uint2(0, 1)] + 0.5 * k1z[id.xy + uint2(0, 1) + uint2(512, 0)]) : zero3;

    float j_s = exchangeStrength[id.xy];
    float j_left = id.x > 1 ? exchangeStrength[id.xy - uint2(1, 0)] : 0.0f;
    float j_down = id.y > 1 ? exchangeStrength[id.xy - uint2(0, 1)] : 0.0f;

    float d_s = D0 + D * j_s;
    float d_left = id.x > 1 ? (D0 + D * j_left) : 0.0f;
    float d_down = id.y > 1 ? (D0 + D * j_down) : 0.0f;

    float3 vright = float3(1.0, 0.0, 0.0);
    float3 vup = float3(0.0, 1.0, 0.0);

    float3 beff = (j_left * sleft + j_s * sright + j_down * sdown + j_s * sup)
        + (d_s * cross(sright, vright) - d_left * cross(sleft, vright) + d_s * cross(sup, vup) - d_down * cross(
            sdown, vup))
        + float3(0.0f, 0.0f, B) + 2.0 * K * float3(0.0f, 0.0f, s.z);

    //t is t + 0.5 dt
    float jx = jxperoid > 0 ? jxPeroidFunction[uint2((jxstep + 1) % jxperoid, 0)] : 0.0f;
    float3 stt = -jx * cross(s, cross((sright - sleft) * 0.5f, s));

    float3 newS = cross(s, beff) + stt;
    newS = (newS - alpha * cross(s, newS)) / (1 + alpha * alpha);

    float3 k3res = timestep * newS;

    k1x[id.xy + uint2(0, 512)] = k3res.x;
    k1y[id.xy + uint2(0, 512)] = k3res.y;
    k1z[id.xy + uint2(0, 512)] = k3res.z;
}

#pragma kernel CSMain

//(0,0) is left buttom corner!
//only change the name up to down, the result is unchanged...
[numthreads(8, 8, 1)]
void CSMain (uint3 id : SV_DispatchThreadID)
{
    //Calculate K4 first
    float3 zero3 = float3(0.0f, 0.0f, 0.0f);
```

```

    float3 s = float3(magneticMomentumX[id.xy] + k1x[id.xy + uint2(0, 512)],
                      magneticMomentumY[id.xy] + k1y[id.xy + uint2(0, 512)],
                      magneticMomentumZ[id.xy] + k1z[id.xy + uint2(0, 512)]);

    float3 sleft = id.x > 1 ?
        float3(magneticMomentumX[id.xy - uint2(1, 0)] + k1x[id.xy - uint2(1, 0) + uint2(0, 512)],
               magneticMomentumY[id.xy - uint2(1, 0)] + k1y[id.xy - uint2(1, 0) + uint2(0, 512)],
               magneticMomentumZ[id.xy - uint2(1, 0)] + k1z[id.xy - uint2(1, 0) + uint2(0, 512)]) : zero3;
    float3 sright = id.x < (size.x - 1) ?
        float3(magneticMomentumX[id.xy + uint2(1, 0)] + k1x[id.xy + uint2(1, 0) + uint2(0, 512)],
               magneticMomentumY[id.xy + uint2(1, 0)] + k1y[id.xy + uint2(1, 0) + uint2(0, 512)],
               magneticMomentumZ[id.xy + uint2(1, 0)] + k1z[id.xy + uint2(1, 0) + uint2(0, 512)]) : zero3;
    float3 sdown = id.y > 1 ?
        float3(magneticMomentumX[id.xy - uint2(0, 1)] + k1x[id.xy - uint2(0, 1) + uint2(0, 512)],
               magneticMomentumY[id.xy - uint2(0, 1)] + k1y[id.xy - uint2(0, 1) + uint2(0, 512)],
               magneticMomentumZ[id.xy - uint2(0, 1)] + k1z[id.xy - uint2(0, 1) + uint2(0, 512)]) : zero3;
    float3 sup = id.y < (size.y - 1) ?
        float3(magneticMomentumX[id.xy + uint2(0, 1)] + k1x[id.xy + uint2(0, 1) + uint2(0, 512)],
               magneticMomentumY[id.xy + uint2(0, 1)] + k1y[id.xy + uint2(0, 1) + uint2(0, 512)],
               magneticMomentumZ[id.xy + uint2(0, 1)] + k1z[id.xy + uint2(0, 1) + uint2(0, 512)]) : zero3;

    float j_s = exchangeStrength[id.xy];
    float j_left = id.x > 1 ? exchangeStrength[id.xy - uint2(1, 0)] : 0.0f;
    float j_down = id.y > 1 ? exchangeStrength[id.xy - uint2(0, 1)] : 0.0f;

    float d_s = D0 + D * j_s;
    float d_left = id.x > 1 ? (D0 + D * j_left) : 0.0f;
    float d_down = id.y > 1 ? (D0 + D * j_down) : 0.0f;

    float3 vright = float3(1.0, 0.0, 0.0);
    float3 vup = float3(0.0, 1.0, 0.0);

    float3 beff = (j_left * sleft + j_s * sright + j_down * sdown + j_s * sup)
        + (d_s * cross(sright, vright) - d_left * cross(sleft, vright) + d_s * cross(sup, vup) - d_down * cross(
            sdown, vup))
        + float3(0.0f, 0.0f, B) + 2.0 * K * float3(0.0f, 0.0f, s.z);

    //t is t + dt
    float jx = jxperoid > 0 ? jxPeroidFunction[uint2((jxstep + 2) % jxperoid, 0)] : 0.0f;
    float3 stt = -jx * cross(s, cross((sright - sleft) * 0.5f, s));

    //add time for jx current
    jxstep = jxstep + 2;

    float3 newS = cross(s, beff) + stt;
    newS = (newS - alpha * cross(s, newS)) / (1 + alpha * alpha);

    float3 k4res = timestep * newS;

    k4res = (k4res + float3(k1x[id.xy], k1y[id.xy], k1z[id.xy]) +
        2.0 * float3(k1x[id.xy + uint2(512, 0)], k1y[id.xy + uint2(512, 0)], k1z[id.xy + uint2(512, 0)]) +
        2.0 * float3(k1x[id.xy + uint2(0, 512)], k1y[id.xy + uint2(0, 512)], k1z[id.xy + uint2(0, 512)]))/6.0;

    float edge = boundaryCondition[id.xy].r > 0.5f ? 1.0f : 0.0f;

    s = float3(magneticMomentumX[id.xy], magneticMomentumY[id.xy], magneticMomentumZ[id.xy]);
    float3 retColor = edge < 0.5f ? zero3 : normalize(s + k4res);

    magneticMomentumX[id.xy] = retColor.x * edge;
    magneticMomentumY[id.xy] = retColor.y * edge;
    magneticMomentumZ[id.xy] = retColor.z * edge;
}
```

The magnetic momentum is a 512×512 64-bit ARGB texture, only R, G, B channel used. $\mathbf{n} = (2 \times r - 1, 2 \times g - 1, 2 \times b - 1)$.

The boundary condition is a 512×512 alpha 8-bit texture, only R channel used, when $R < 0.5$, it is a defect.

The exchange strength is a 32-bit RFloat texture, generated from a Lua script. For example, a constant exchange strength can be generated from a lua file as

```
-- Exchange Strength is constant
function GetJValueByLatticeIndex(x, y)
    return 2.0
end

-- Need to register the function
return {
    GetJValueByLatticeIndex = GetJValueByLatticeIndex,
}
```

While a pin with $J = 1 + \exp\left(-0.001\rho^2\right)$ at lattice index $(255, 255)$ can be written as

```
-- Exchange Strength is pin
function GetJValueByLatticeIndex(x, y)
    local j0 = 1
    local j1 = 1
    local j2 = 0.001
    local rho = (x - 255) * (x - 255) + (y - 255) * (y - 255)

    return j0 + j1 * math.exp(-1.0 * j2 * rho)
end

-- Need to register the function
return {
    GetJValueByLatticeIndex = GetJValueByLatticeIndex,
}
```

Manual.pdf is a document introduce how to use the pre-built software.

---

[1] Gen Tatara, Hiroshi Kohno, Junya Shibata, Phys. Rep. 468, 213-301 (2008), 10.1016/j.physrep.2008.07.003, arXiv:0807.2894.
[2] Jiadong Zang, Maxim Mostovoy, Jung Hoon Han, and Naoto Nagaosa, 10.1103/PhysRevLett.107.136804.
[3] Hong Chul Choi, Shi-Zeng Lin, Jian-Xin Zhu, Phys. Rev. B 93, 115112 (2016), 10.1103/PhysRevB.93.115112, arXiv:1601.00933.
[4] Ye-Hua Liu, You-Quan Li, J. Phys.: Condens. Matter 25 076005, 10.1088/0953-8984/25/7/076005, arXiv:1206.5661.
[5] Junichi Iwasaki, Wataru Koshibae, and Naoto Nagaosa, Nano. Lett. 2014, 14, 4432-4437, 10.1021/nl501379k.
[6] https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods