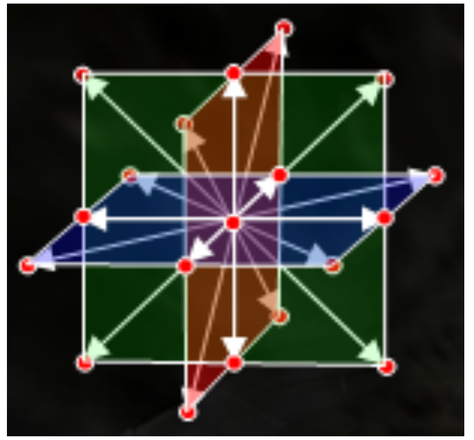
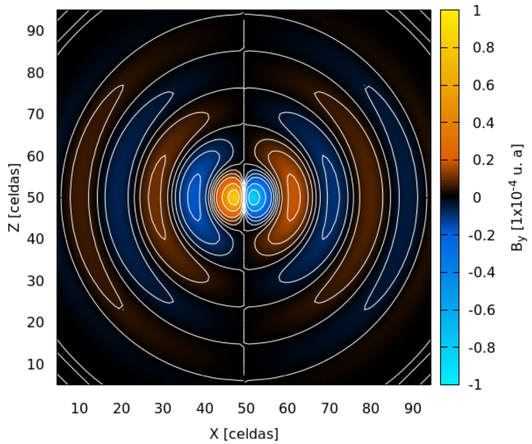
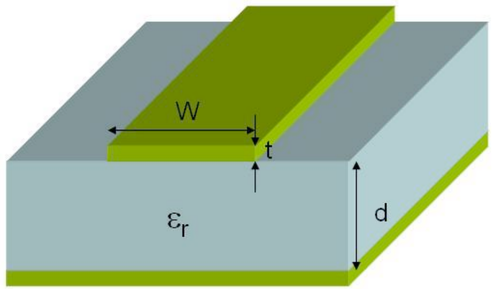
Lattice - Boltzmann Crash Course
Session 4a - Electrodynamics

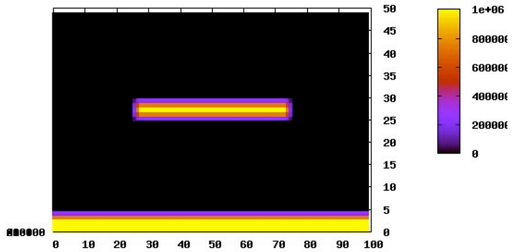


Prof. José Daniel Muñoz

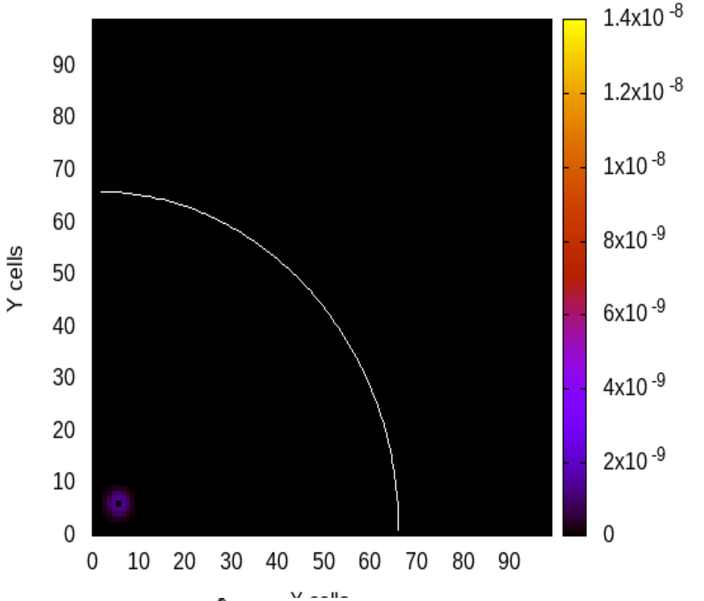
Simulation of Physical Systems Group,
Department of Physics, Universidad Nacional de Colombia, Bogota
Crr 30 # 45-03, Ed. 404, Of. 348, Bogotá D.C. 111321, Colombia







Electric field intensity time evolution for Corner Reflector antenna 3rd order



https://drive.google.com/drive/folders/1zzzTctPTXmEG4nqfR90LQIm1gyTPR3fx?usp=sharing

Website

Helmholtz Institute Erlangen-Nürnberg

El conocimiento es de todos

Minciencias

UNIVERSIDAD NACIONAL DE COLOMBIA

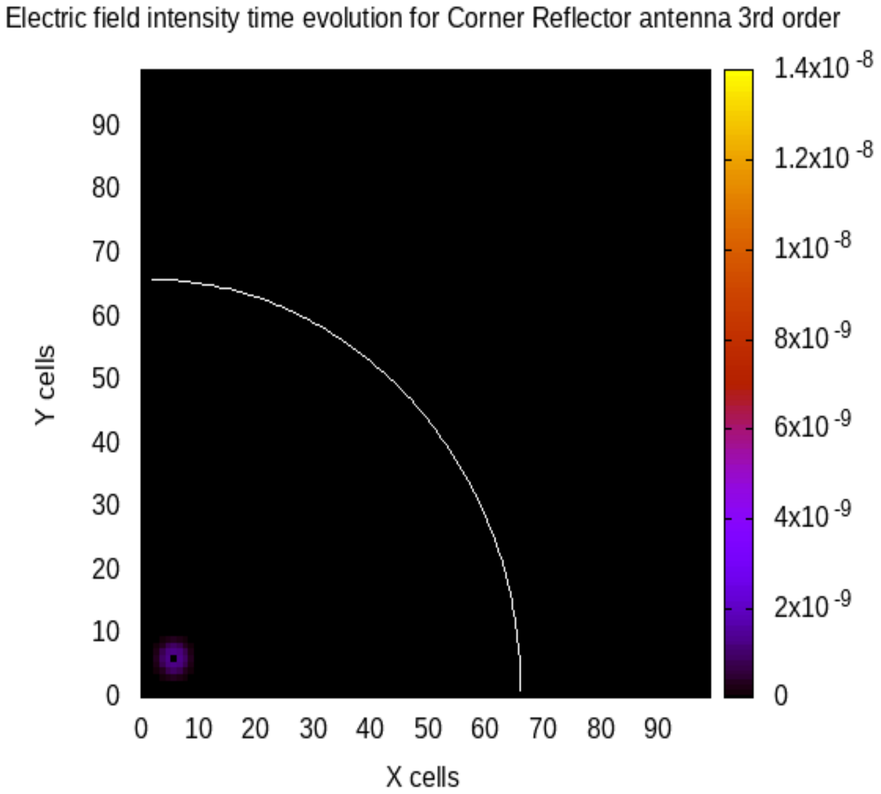
Lattice Boltzmann for Electrodynamics

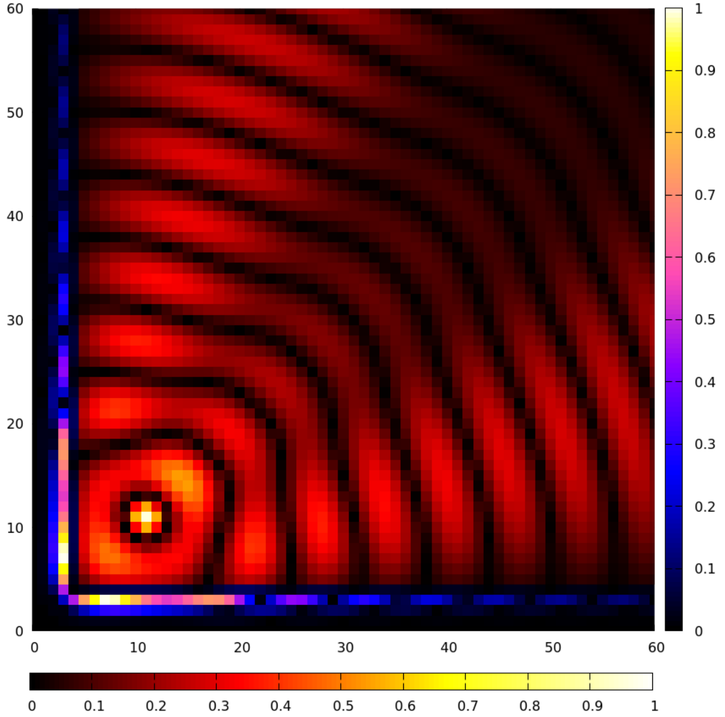
Mendoza M. , J. D. Muñoz, 2010, *Three dimensional Lattice-Boltzmann model for Electrodynamics.* Phys. Rev. E 82, 056708), M.Sc. thesis.

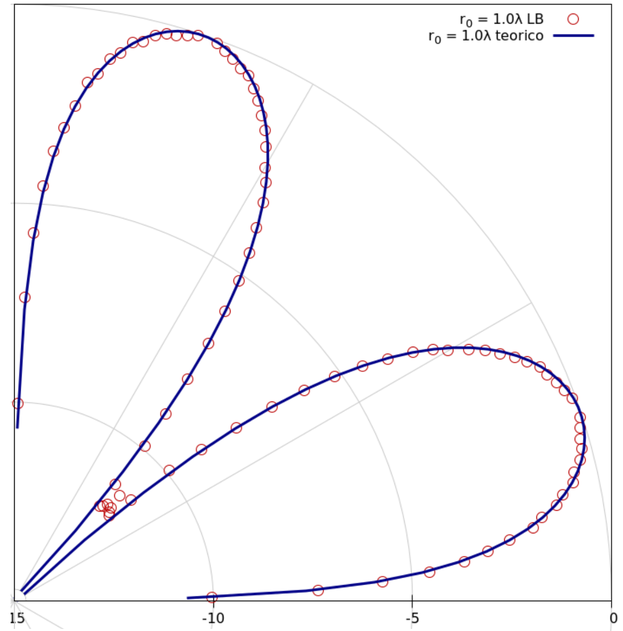
$$\nabla \times \vec{E}' = -\frac{\partial \vec{B}}{\partial t} \qquad \nabla \times \vec{H} = \mu_0 \vec{J}' + \frac{1}{c^2}\frac{\partial \vec{D}'}{\partial t} \qquad \frac{\partial \rho_c}{\partial t} + \nabla \cdot \vec{J}' = 0$$

$$\frac{\partial}{\partial t}\left(\nabla \cdot \vec{D}' - \frac{\rho_c}{\epsilon_0}\right) = 0 \qquad \frac{\partial}{\partial t}\left(\nabla \cdot \vec{B}\right) = 0$$

Esteban Castro Ávila (undergraduate thesis)

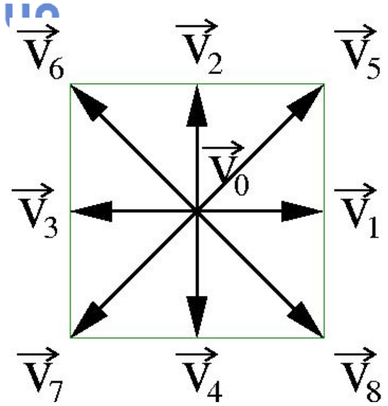






What is a Lattice-Boltzmann BGK?

(McNamara-Zanetti,1988, Higuera-Jiménez-Succi,1989, Benzi-Succi-Vergassola,1992)



$+ f_i$

Macroscopic fields

$$\rho = \sum_i f_i \quad , \quad \rho \vec{U} = \sum_i \vec{v}_i f_i$$

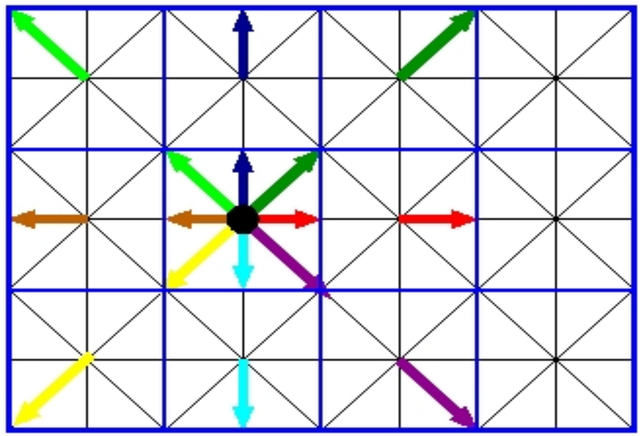
Equilibrium Functions

$$f_i^{(\mathrm{eq})} = f_i^{(\mathrm{eq})}(\rho, \vec{U})$$

Discrete Boltzmann's transport equation

$$f_i(\vec{x} + \delta t \vec{v}_i, t + \delta t) - f_i(\vec{x}, t) = -\frac{1}{\tau}\left[f_i(\vec{x}, t) - f_i^{(\mathrm{eq})}(\vec{x}, t)\right]$$

(Example: BGK (Bhatnagar-Gross-Krook) Evolution rule)



Advection

Choosing the right f(eq) gives us the desired partial differential equation

Start from the evolution rule,

$$f_i(\vec{x}+\delta t\vec{v}_i, t+\delta t) - f_i(\vec{x},t) = -\frac{1}{\tau}[f_i(\vec{x},t) - f_i^{(eq)}(\vec{x},t)]$$

and perform a Chapman-Enskog expansion,

•A Taylor series

$$f_i(\vec{x}+\delta t\,\vec{v}_i, t+\delta t) \quad - \quad f_i(\vec{x},t) =$$

$$\delta t\left[\frac{\partial}{\partial t}+\vec{v}_i\cdot\vec{\nabla}\right]f_i + \frac{\delta t^2}{2}\left[\frac{\partial}{\partial t}+\vec{v}_i\cdot\vec{\nabla}\right]^2 f_i$$

•A perturbative expansion in $\epsilon = \delta x/x$

$$\frac{\partial}{\partial t} = \epsilon\frac{\partial}{\partial t_1} + \epsilon^2\frac{\partial}{\partial t_2} \qquad \frac{\partial}{\partial x} = \epsilon\frac{\partial}{\partial x_1}$$

$$f_i = f_i^{(0)} + \epsilon f_i^{(1)} + \epsilon^2 f_i^{(2)}$$

•Just the order 0 gives the fields

$$\rho = \sum_i f_i^{(0)} \quad ,$$

$$\vec{J} = \sum_i \vec{v}_i f_i$$

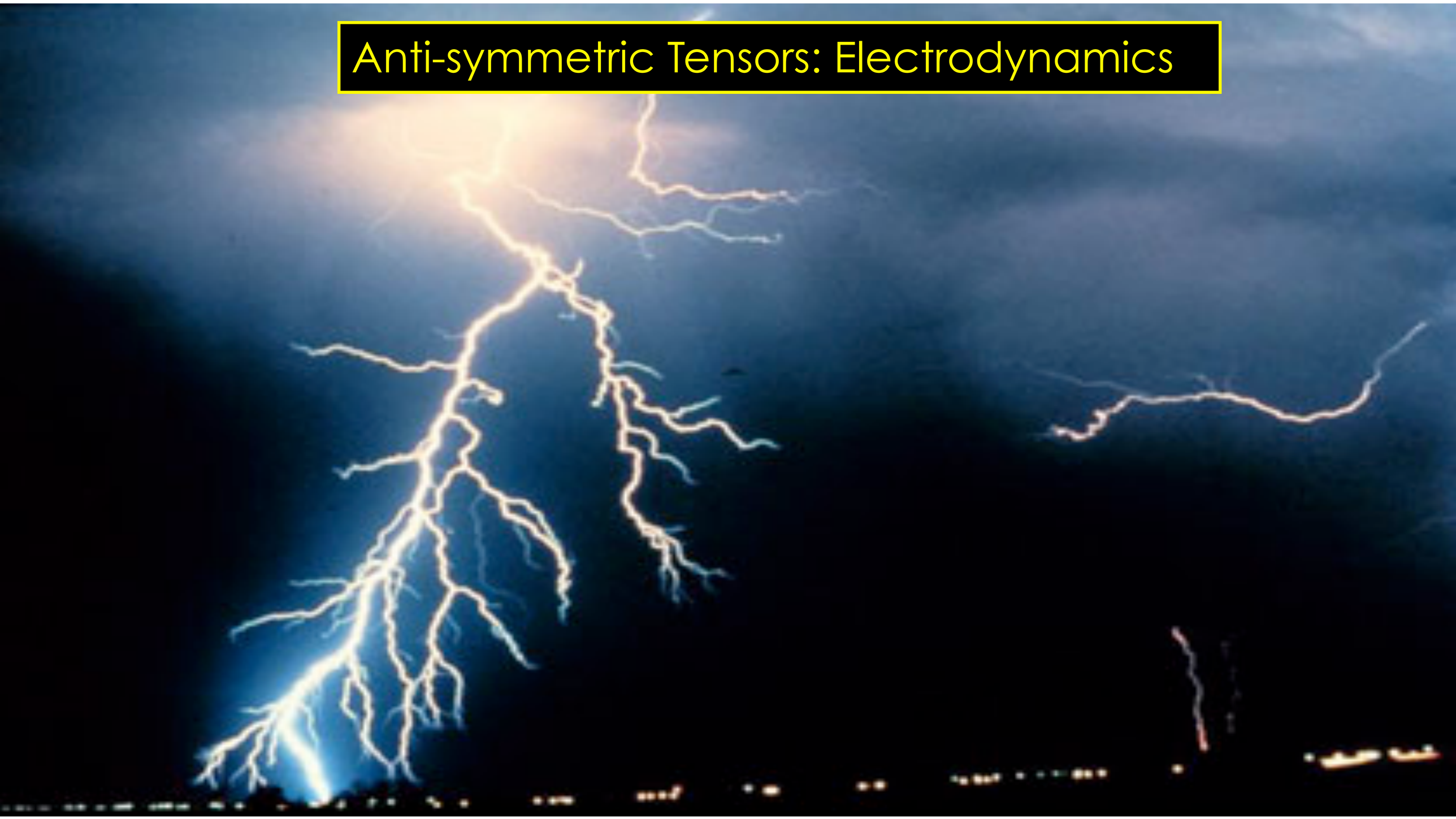
By solving order by order and joining together, you obtain ($\tau = 1/2$)

$$-\frac{1}{\tau\delta t}\left(\epsilon f_i^{(1)} + \epsilon^2 f_i^{(2)}\right) = \frac{\partial}{\partial t} f_i^{(0)} + \vec{\nabla}\cdot\left(\vec{v}_i f_i^{(0)}\right)$$

By multiplying by and taking $\epsilon \to 0$, you will obtain conservative laws

$$\frac{\partial\rho}{\partial t} = -\vec{\nabla}\cdot\vec{J} \qquad \frac{\partial\vec{J}}{\partial t} = -\vec{\nabla}\cdot\Pi^{(0)} \qquad \text{with} \qquad \Pi^{(0)} = \sum_i f_i^{(0)}\vec{v}_i\otimes\vec{v}_i$$

Anti-symmetric Tensors: Electrodynamics

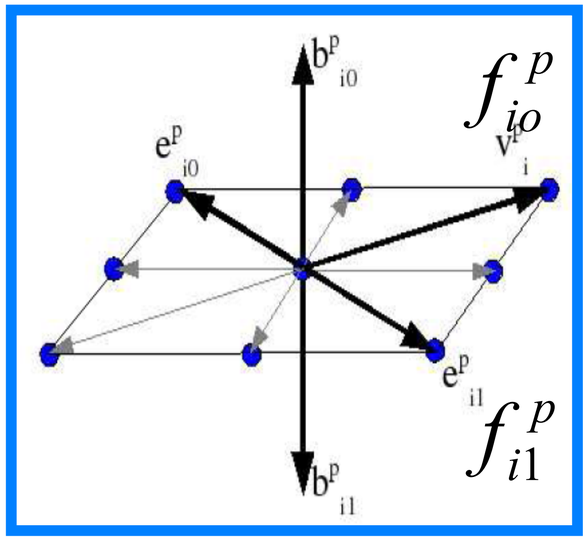


Anti-symmetric tensors: Faraday's Law

$$\frac{\partial \vec{B}}{\partial t} = -\vec{\nabla} \times \vec{E} = -\vec{\nabla} \cdot \begin{pmatrix} 0 & -E_z & E_y \\ E_z & 0 & -E_x \\ -E_y & E_x & 0 \end{pmatrix}$$

Miller Mendoza

Let us define two electrical $\vec{e}^{\,p}_{i0}, \vec{e}^{\,p}_{i1}$ and two magnetic $\vec{b}^{\,p}_{i0}, \vec{b}^{\,p}_{i1}$ auxiliary vectors for each $\vec{v}^{\,p}_{i}$



Define two distribution functions f^p_{i0}, f^p_{i1} travelling with $\vec{v}^{\,p}_{i}$

$$\vec{E} = \sum_{i,j,p} \vec{e_{ij}}^{\,p} f^p_{ij} \qquad \vec{B} = \sum_{i,j,p} \vec{b_{ij}}^{\,p} f^p_{ij}$$

We choose the following equilibrium distribution

$$f^{p(eq)}_{ij} = \frac{1}{4}\left(\vec{e}^{\,p}_{ij} \cdot \vec{E}\right) + \frac{1}{8}\left(\vec{b}^{\,p}_{ij} \cdot \vec{B}\right)$$

$$\sum_{i,p} v^p_{i\alpha} e^p_{ij\beta} b^p_{ij\gamma} = 4\varepsilon_{\alpha\beta\gamma}$$

From the Chapman-Enskog expansion,

$$-\frac{1}{\tau\delta t}\left(\epsilon f_i^{(1)}+\epsilon^2 f_i^{(2)}\right)=\frac{\partial}{\partial t}f_i^{(0)}+\vec{\nabla}\cdot\left(\vec{v}_i f_i^{(0)}\right)$$

Multiplying by $\overrightarrow{b}_{ij}^{p}$ and adding on i,j,p

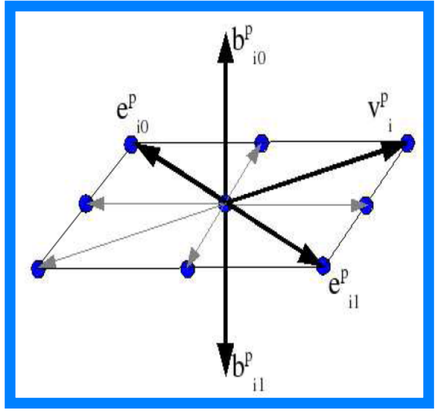
$$0=\frac{\partial\vec{B}}{\partial t}+\vec{\nabla}\cdot\Lambda^{(0)} \qquad \Lambda_{\alpha\beta}^{(0)}=\sum_{i,j,p} f_{ij}^{p(eq)} v_{i\,\alpha}^{p} b_{ij\,\beta}^{p}$$

With the equilibrium function

$$f_{ij}^{p(eq)}=\frac{1}{4}\left(\vec{e}_{ij}^{\,p}\cdot\vec{E}\right)+\frac{1}{8}\left(\vec{b}_{ij}^{\,p}\cdot\vec{B}\right)$$

, we obtain

$$\Lambda_{\alpha\beta}^{(0)}=\frac{1}{4}\sum_{\gamma}E_\gamma\left(\sum_{i,j,p} v_{i\alpha}^{p} e_{ij\beta}^{p} b_{ij\gamma}^{p}\right)=\sum_{\gamma}E_\gamma\varepsilon_{\alpha\beta\gamma}$$

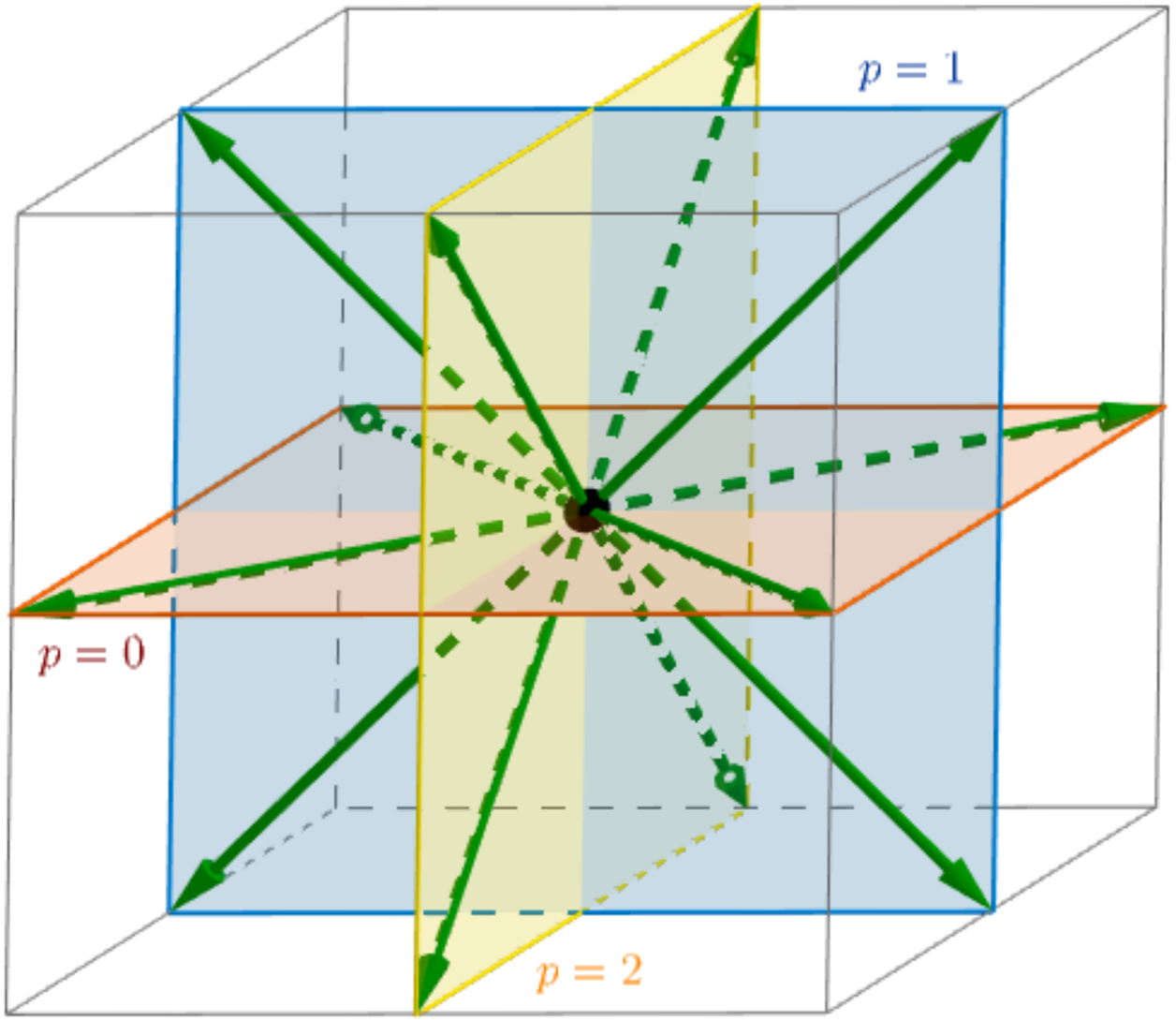


$$\sum_{i,p} v_{i\alpha}^{p} e_{ij\beta}^{p} b_{ij\gamma}^{p} = 4\varepsilon_{\alpha\beta\gamma}$$

$$\vec{E}=\sum_{i,j,p}\vec{e_{ij}}^{\,p} f_{ij}^{p}$$

$$\vec{B}=\sum_{i,j,p}\vec{b_{ij}}^{\,p} f_{ij}^{p}$$

$$\Lambda^{(0)}=\begin{pmatrix} 0 & -E_z & E_y \\ E_z & 0 & -E_x \\ -E_y & E_x & 0 \end{pmatrix}$$

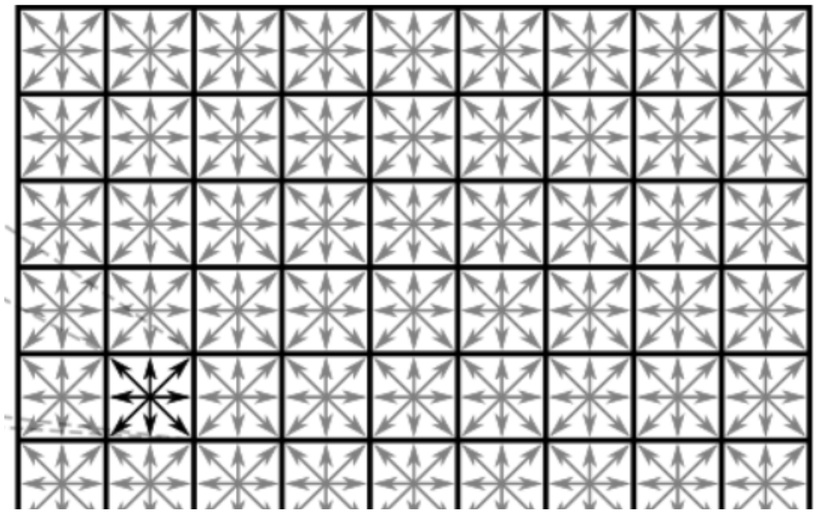

$p = 1$
$p = 0$
$p = 2$

LBM for Electrodynamics (EMLBM)

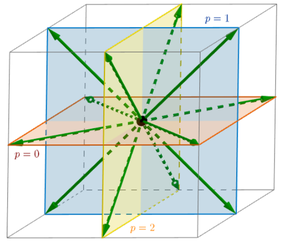
(Mendoza M. , J. D. Muñoz, 2010, *Three dimensional Lattice-Boltzmann model for Electrodynamics.* Phys. Rev. E 82, 056708)

Miller Mendoza

1 Cubic cells

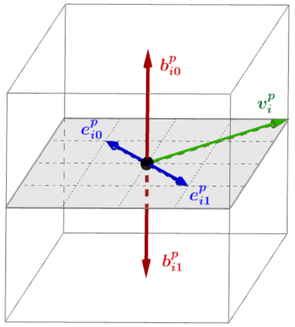


D3Q13



auxiliary vectors

$\vec{e}_{ij}^{\,p}$ y $\vec{b}_{ij}^{\,p}$



Four functions for each $\vec{v}_i^{\,p}$

$$f_{i0}^{p(0)}, f_{i1}^{p(0)}, f_{i0}^{p(1)}, f_{i1}^{p(1)}$$

$$\sum_{i,p} v_{i\alpha}^p e_{ij\beta}^p b_{ij\gamma}^p = 4\varepsilon_{\alpha\beta\gamma}$$

2 Macroscopic fields

$$\vec{D} = \sum_{i=1}^{4}\sum_{p=0}^{2}\sum_{j=0}^{1} f_{ij}^{p(0)}\vec{e}_{ij}^{\,p}$$

$$\vec{B} = \sum_{i=1}^{4}\sum_{p=0}^{2}\sum_{j=0}^{1} f_{ij}^{p(1)}\vec{b}_{ij}^{\,p}$$

$$\rho_c = f_0^{(0)} + \sum_{i=1}^{4}\sum_{p=0}^{2}\sum_{j=0}^{1} f_{ij}^{p(0)}$$

$$\vec{E} = \frac{\vec{D}}{\epsilon_r} \qquad \vec{H} = \frac{\vec{B}}{\mu_r}$$

$$\vec{J} = \sigma\vec{E}$$

$$\vec{J}' = \frac{\sigma}{1+\frac{\mu_0\sigma}{4\epsilon_r}}\vec{E} \qquad \vec{E}' = \vec{E} - \frac{\mu_0}{4\epsilon_r}\vec{J}'$$

LBM for Electrodynamics (EMLBM)

Miller Mendoza

3 Corrected equilibrium distributions

$$f_{ij}^{p(0)\mathrm{eq}} = \frac{1}{16}(\vec{v}_i^p \cdot \vec{J}') + \frac{\epsilon_{\mathbf{r}}}{4}(\vec{e}_{ij}^p \cdot \vec{E}') + \frac{1}{8\mu_r}(\vec{b}_{ij}^p \cdot \vec{B})$$

$$f_{ij}^{p(1)\mathrm{eq}} = \frac{1}{16}(\vec{v}_i^p \cdot \vec{J}') + \frac{1}{4}(\vec{e}_{ij}^p \cdot \vec{E}') + \frac{1}{8}(\vec{b}_{ij}^p \cdot \vec{B})$$

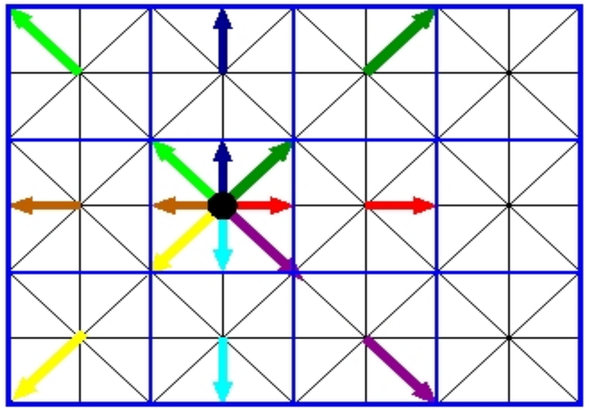
4 BGK evolution rule with $\tau = 1/2$

$$f_{ij}^{p(r)}(\vec{x} + \vec{v}_i^p \delta t, t + \delta t) = f_{ij}^{p(r)}(\vec{x}, t) - \frac{1}{\tau}\left[f_{ij}^{p(r)}(\vec{x}, t) - f_{ij}^{p(r)\mathrm{eq}}(\vec{x}, t)\right]$$

5 Impose boundary conditions

$$f_{ij}^{p(r)}(\vec{x}, t) = f_{ij}^{p(r)\mathrm{eq}}(\vec{E}_o, \vec{B}_o, \vec{J}_0, \rho_c)$$

6 Advection



7 Maxwell's Equations

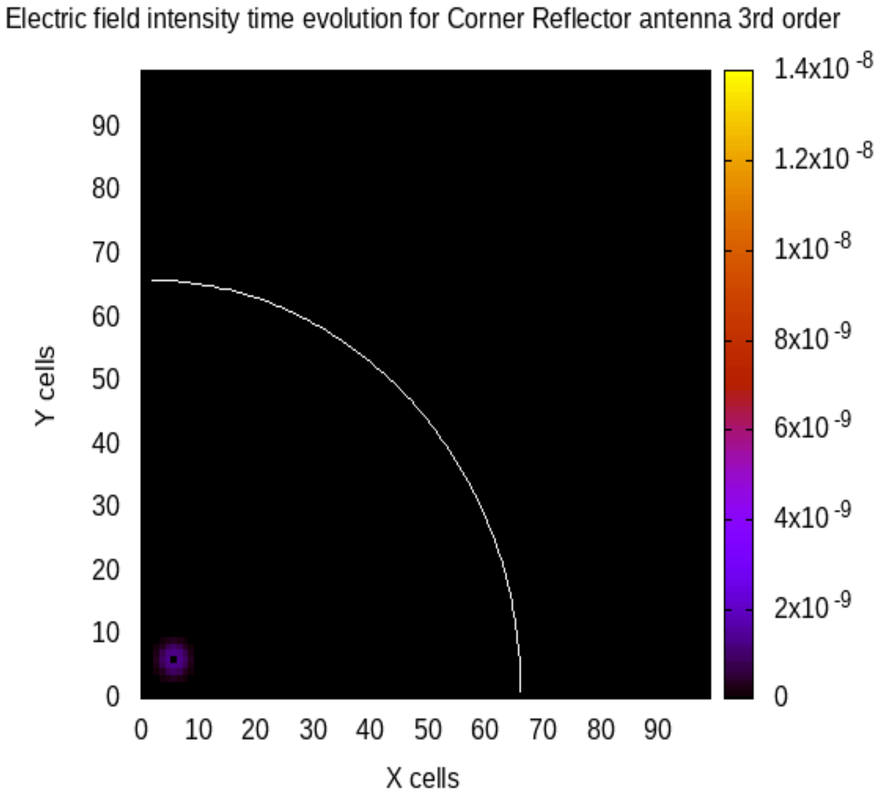
$$\nabla \times \vec{E}' = -\frac{\partial \vec{B}}{\partial t} \qquad \nabla \times \vec{H} = \mu_0 \vec{J}' + \frac{1}{c^2}\frac{\partial \vec{D}'}{\partial t} \qquad \frac{\partial \rho_c}{\partial t} + \nabla \cdot \vec{J}' = 0$$

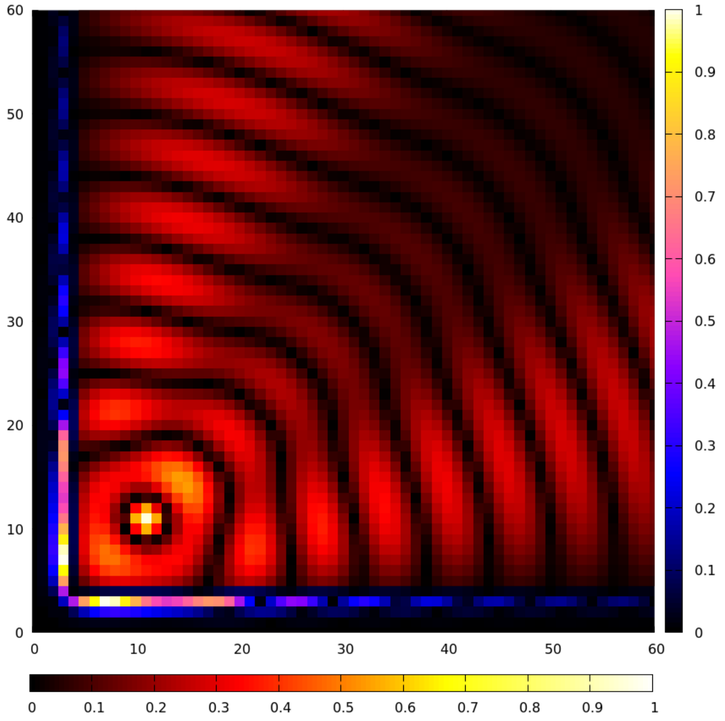
$$\frac{\partial}{\partial t}\left(\nabla \cdot \vec{D}' - \frac{\rho_c}{\epsilon_0}\right) = 0 \qquad \frac{\partial}{\partial t}\left(\nabla \cdot \vec{B}\right) = 0$$

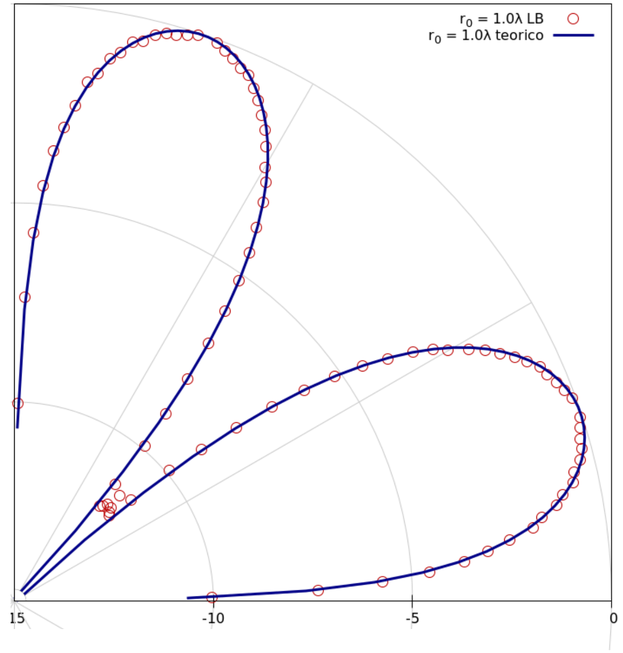
Initial conditions must satisfy Gauss Laws

A Corner- Reflector Antenna

Esteban Castro Ávila (undergraduate thesis)



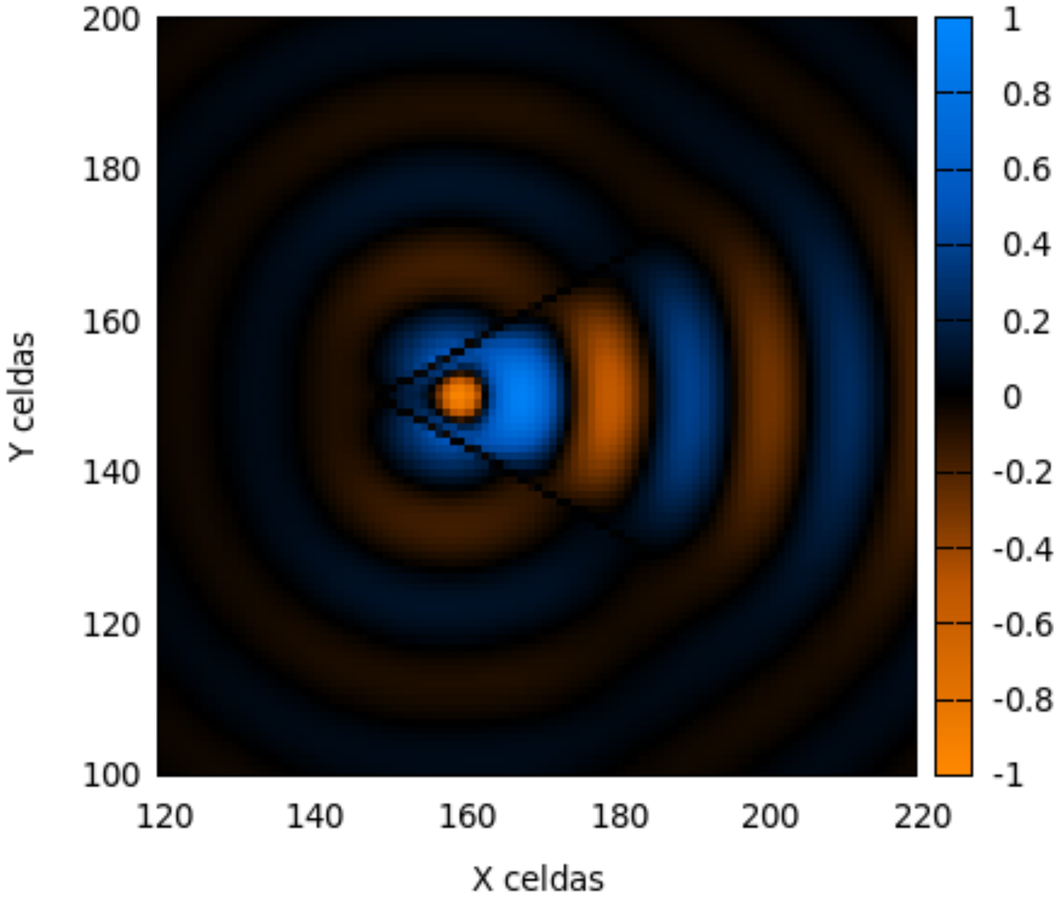




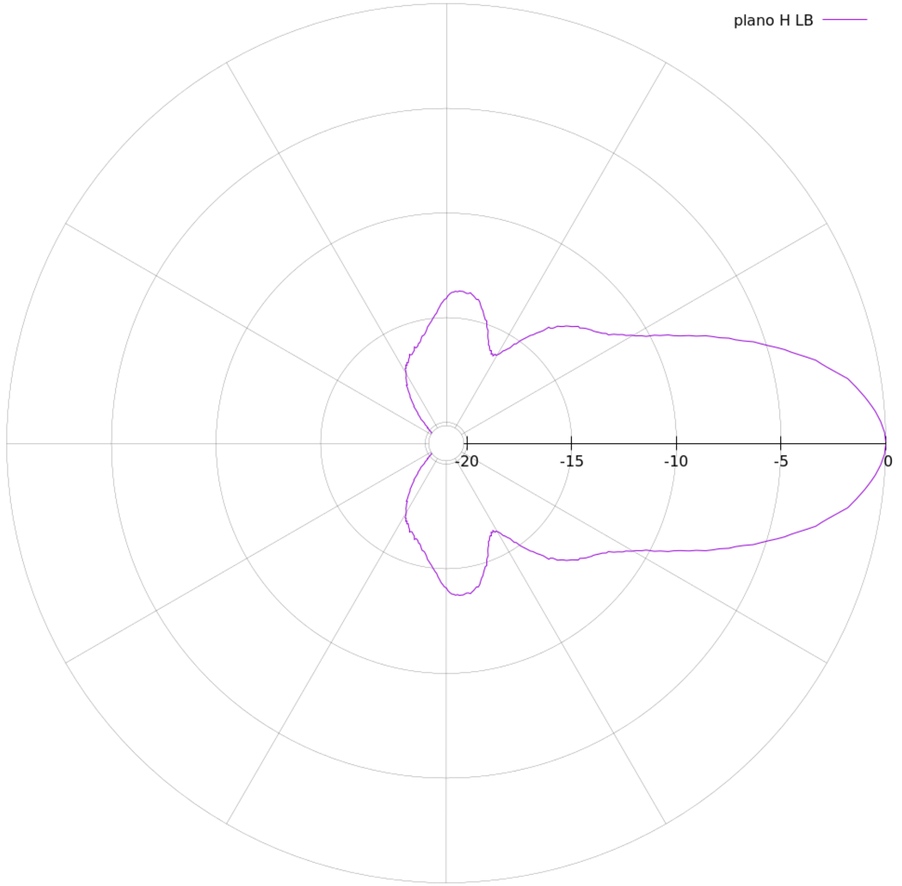
Precision around 3%

Corner-Reflector Antenna - 3D simulation on GPU (CUDA)

10X faster and better resolution

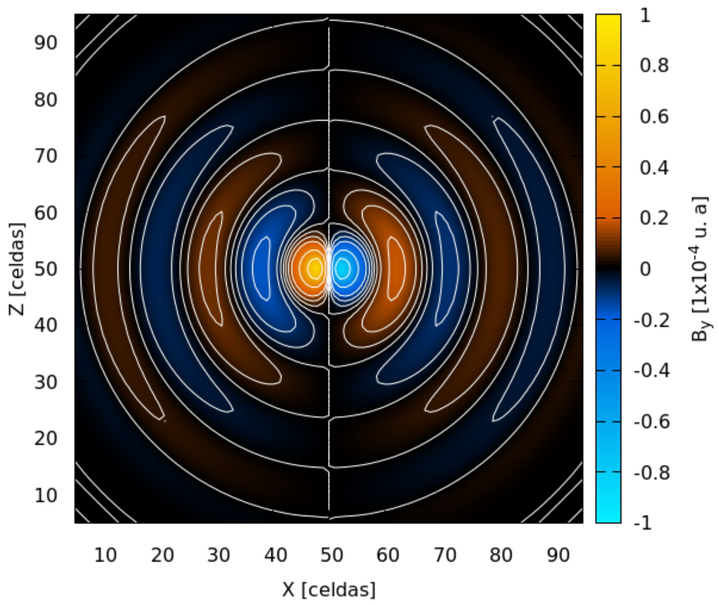


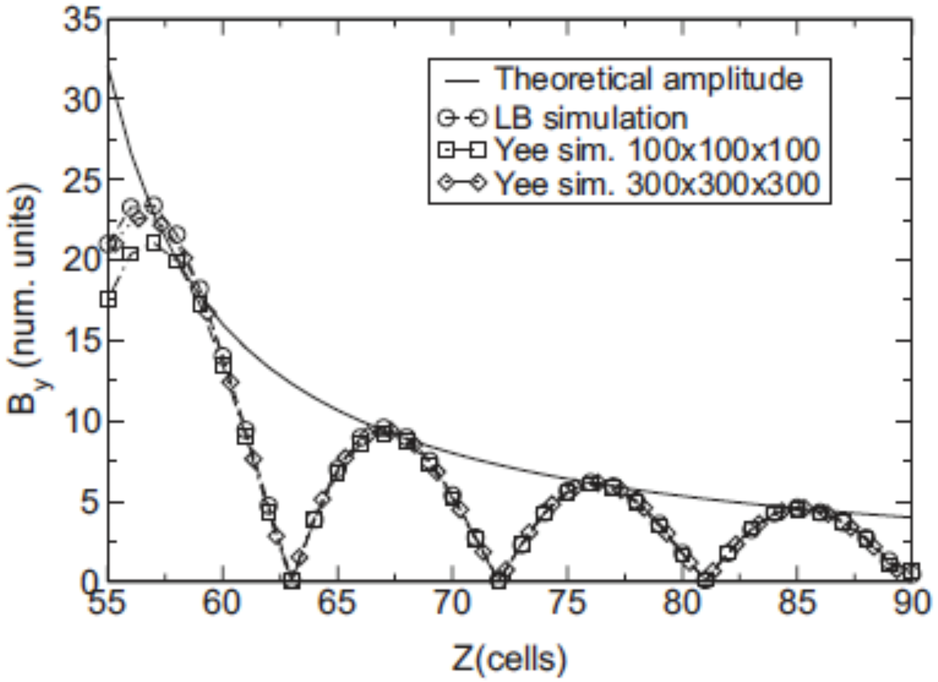
3D Electric field by a Corner-Reflector antenna with finite plates



Radiation Pattern

Electric dipole



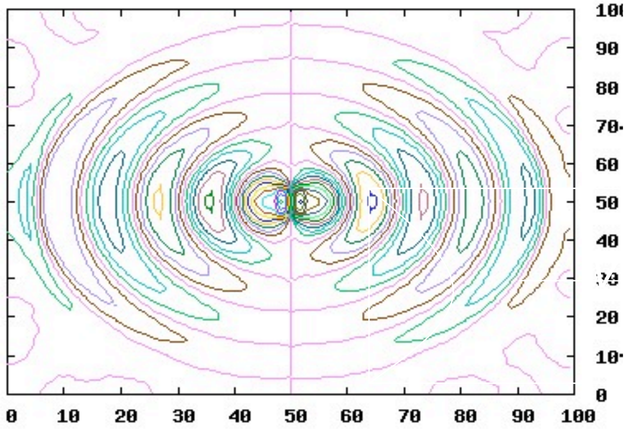


Accuracy	LB(L=100)	Yee(L=100)	Yee(L=200)	Yee(L=300)
1st peak	1.4%	8.7%	3.6%	1.6%
2nd peak	1.7%	1.7%	0.1%	0.1%
3rd peak	0.8%	0.3%	0.3%	1.3%
4th peak	0.7%	0.9%	1.1%	0.7%
Total Variance	2.5%	8.9%	4.0%	2.2%
CPU time	23s	5s	27s	336s
N Variables	50M	6M	48M	162M

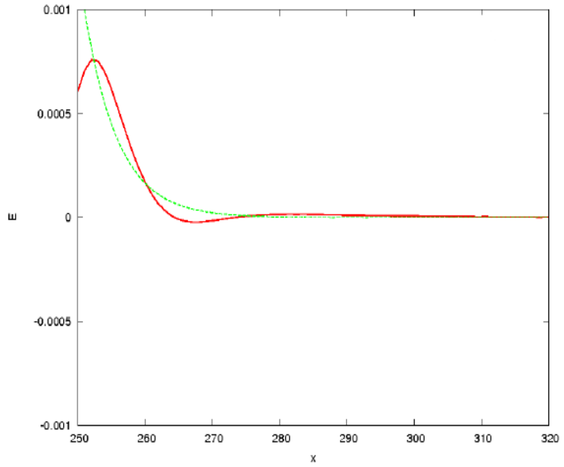
LB=10x faster and 3x less memory than FDTD

Some results

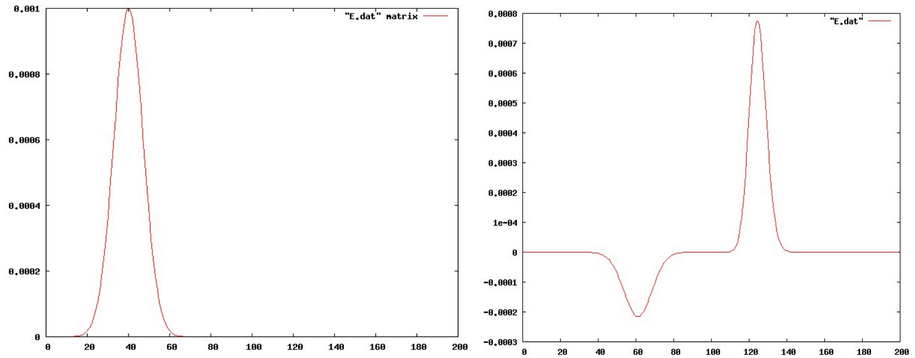
Oscillating dipole



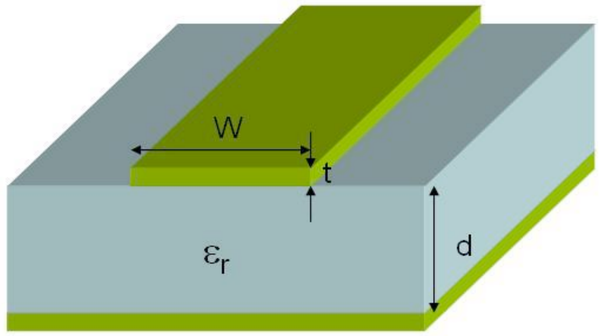
Skin Effect

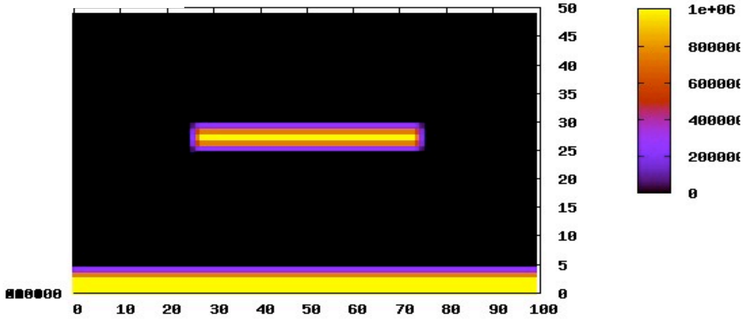


Pulse Reflection



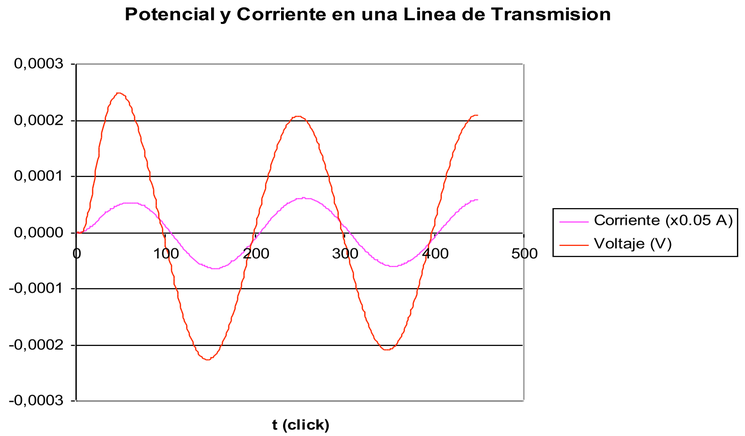
Microstrip





Zo=70.73Ω, error = 3%

Voltage and Current



A Non-Linear medium

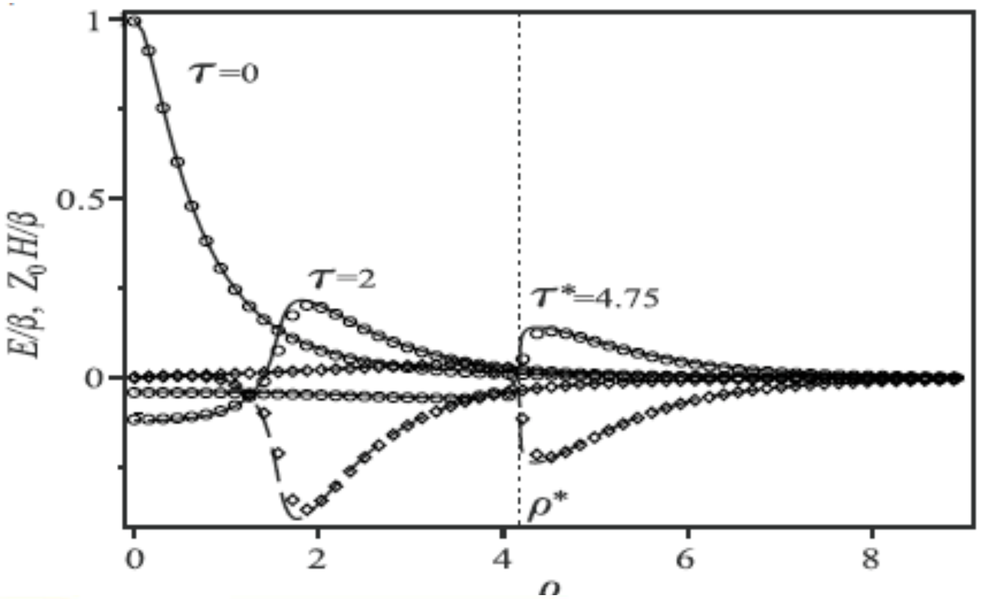
M. Mendoza, J.D. Muñoz, *PIERS Proceedings, Marrakesh MOROCCO* 1632 (2011)

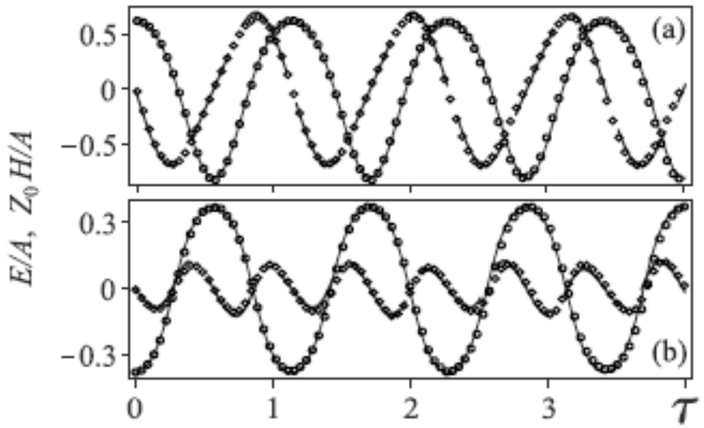
A dielectric medium with $\vec{E} = (0, 0, E) \quad \vec{D} = (0, 0, D)$

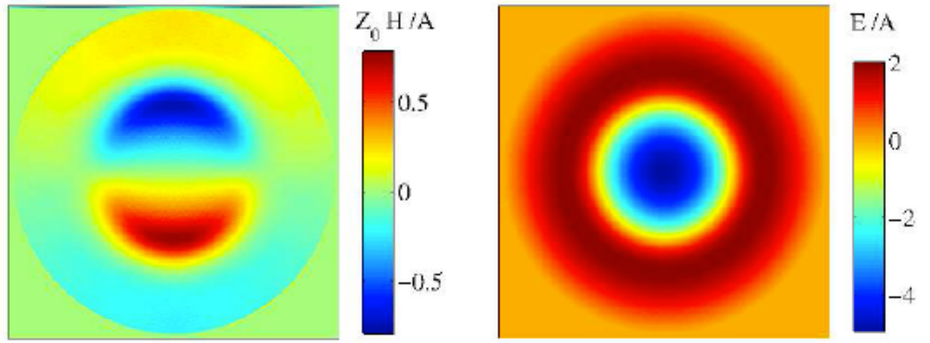
$$D = \epsilon_0 \epsilon_1 \left[E + \alpha \frac{E^2}{2} \right] \qquad E = \frac{\sqrt{\epsilon_0 \epsilon_1 (2\alpha D + \epsilon_1 \epsilon_0)} - \epsilon_0 \epsilon_1}{\alpha \epsilon_0 \epsilon_1}$$

(Exact solution by L.Y. Petrov, A.V. Kudrin, *Phys. Rev. Lett.* **104**, 190404, 2010)

Free propagation of a cylindrical wave

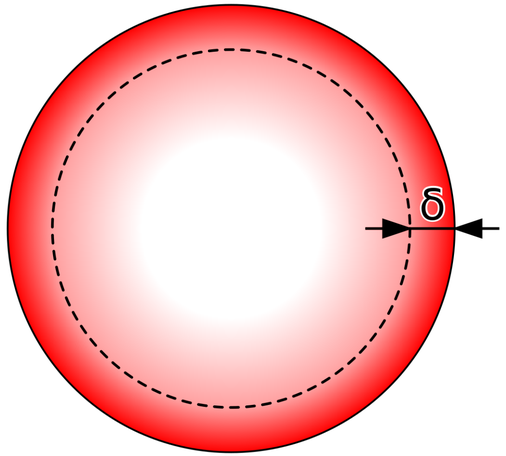




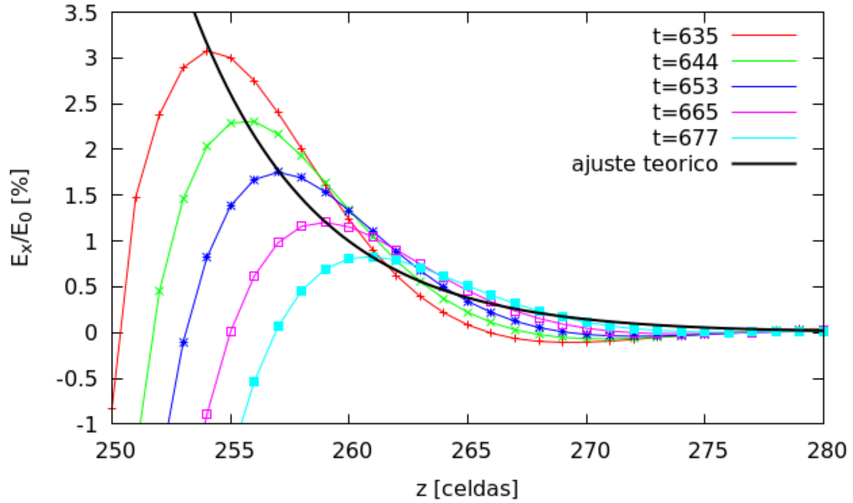


2nd mode of a cylindrical resonant cavity

Skin effect

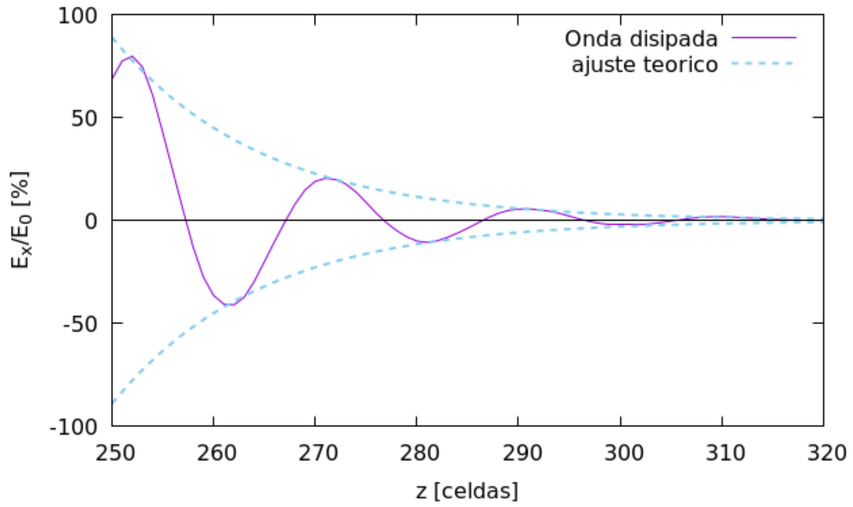


$$\delta = \sqrt{\frac{2}{\sigma \mu \omega}}$$



Good conductor

Use $\sigma \sim 10^{-1}$ in automaton units.



Bad conductor

EMLBM_Skin.cpp

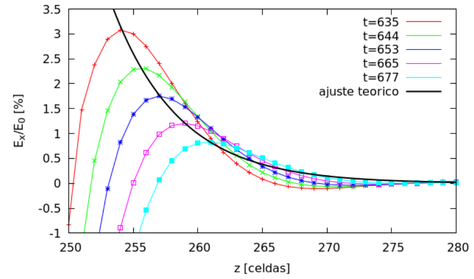
```cpp
//Skin Effect in 1D
#include<iostream>
#include<cmath>
#include "Vector.h"
using namespace std;

//--------CONSTANTS---------------
const int Lx = 1;    //
const int Ly = 1;    //
const int Lz = 1000; //
//--------------------
const double Tau = 0.5;
const double UTau = 1/Tau;
const double UmUTau=1-1/Tau;
//--------------------
const double Epsilon0=1, Mu0=2;
const double Sigma0=0.0125;
const double C=1.0/sqrt(2.0);

const double E00=0.001,B00=E00/C;
```

Vector.h

```cpp
//---------------------- class vector3D -------------------
class vector3D{
private:
  double v[3];
 public:
  void   cargue(double x0, double y0, double z0);
  void   show(void);
  // Get functions
  double x(void){return v[0];};
  double y(void){return v[1];};
  double z(void){return v[2];};
  //Reading the elements as an array
  double & operator[](int i){return v[i];};

  // Vector operations
  vector3D    operator= (vector3D v2);
  vector3D    operator+ (vector3D v2);
  vector3D    operator+=(vector3D v2);
  vector3D    operator- (vector3D v2);
  vector3D    operator-=(vector3D v2);
  // Multiplying by a number
  vector3D    operator* (double a);
  vector3D    operator*=(double a);
  friend  vector3D    operator* (double a,vector3D v1);
  // Dividing by a number
  vector3D    operator/ (double a);
  // Cross product
  vector3D    operator^ (vector3D v2);
  // Dot product
  double operator* (vector3D v2);
  // Norm
  friend  double norma2(vector3D v1);
  friend  double norma(vector3D v1);
};
```

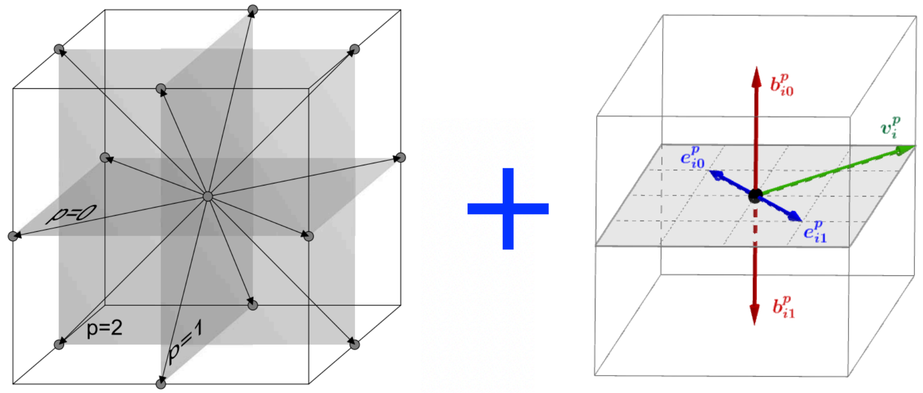
Electromagnetic parameters for the media

```
//---Electromagnetic Constants for the Media---
double mur(int ix,int iy,int iz){
  return 1.0;
}
double epsilonr(int ix,int iy,int iz){
  return 1.0;
}
double sigma(int ix,int iy,int iz){
  return Sigma0*(tanh(iz-100)-tanh(iz-900))/2;
}
```



Class LatticeBoltzmann (data)

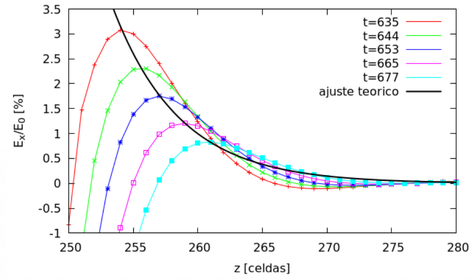
D3Q13




```
//---------------------- class LatticeBoltzmann ------------
class LatticeBoltzmann{
private:
  int V[3][3][4], V0[3]; /*V[xyz][p][i]*/  vector3D v[3][4],v0;
  vector3D e[3][4][2], e0; //e[p][i][j]
  vector3D b[3][4][2], b0; //b[p][i][j]
  double f[Lx][Ly][Lz][2][3][4][2],fnew[Lx][Ly][Lz][2][3][4][2];//f[ix][iy][iz][r][p][i][j]
  double f0[Lx][Ly][Lz],f0new[Lx][Ly][Lz];//f0[ix][iy][iz] (r=0)
```

```
LatticeBoltzmann::LatticeBoltzmann(void){
  int ix,iy,iz,alpha,r,p,i,j;
  //Velocity vectors V[p][i]=V^p_i (in components)
  V0[0]=V0[1]=V0[2]=0;

  V[0][0][0]=V[0][1][0]=V[1][2][0]=1;
  V[1][0][0]=V[2][1][0]=V[2][2][0]=1;
  V[2][0][0]=V[1][1][0]=V[0][2][0]=0;

  V[0][0][1]=V[0][1][1]=V[1][2][1]=-1;
  V[1][0][1]=V[2][1][1]=V[2][2][1]=1;
  V[2][0][1]=V[1][1][1]=V[0][2][1]=0;

  V[0][0][2]=V[0][1][2]=V[1][2][2]=-1;
  V[1][0][2]=V[2][1][2]=V[2][2][2]=-1;
  V[2][0][2]=V[1][1][2]=V[0][2][2]=0;

  V[0][0][3]=V[0][1][3]=V[1][2][3]=1;
  V[1][0][3]=V[2][1][3]=V[2][2][3]=-1;
  V[2][0][3]=V[1][1][3]=V[0][2][3]=0;

  //Velocity vectors V[p][i]=V^p_i (as vectors)
  v0.cargue(V0[0],V0[1],V0[2]); //cargue= load (in Spanish)
  for(p=0;p<3;p++)
    for(i=0;i<4;i++){
      v[p][i].cargue(V[0][p][i],V[1][p][i],V[2][p][i]);
  }
  //Electric vectors e[p][i][j]=e^p_{ij}
  e0.cargue(0,0,0);
  for(p=0;p<3;p++)
    for(i=0;i<4;i++){
      e[p][i][0]=v[p][(i+1)%4]*0.5;
      e[p][i][1]=v[p][(i+3)%4]*0.5;
  }
  //Magnetic vectors b[p][i][j]=b^p_{ij}=v^p_i x e^p_{ij}
  b0.cargue(0,0,0);
  for(p=0;p<3;p++)
    for(i=0;i<4;i++)
      for(j=0;j<2;j++)
        b[p][i][j]=(v[p][i]^e[p][i][j]);
}
```

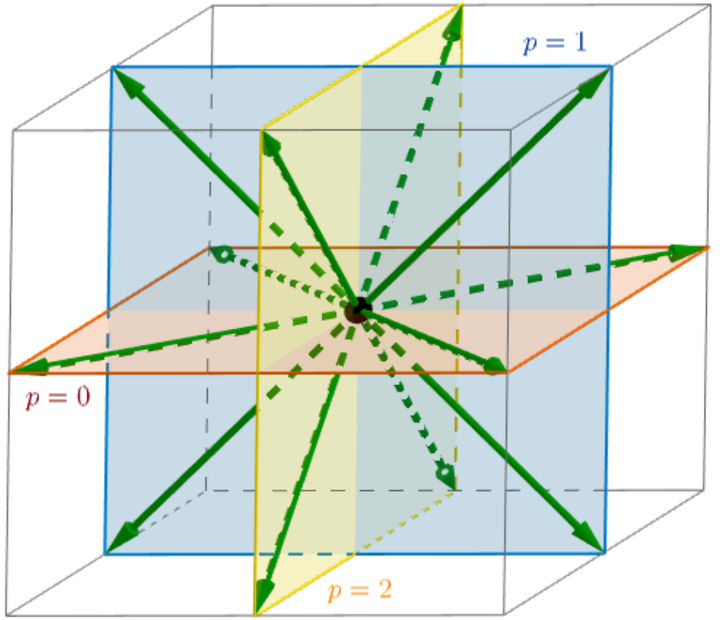
Class LatticeBoltzmann (functions)




```
public:
  LatticeBoltzmann(void);
  //Auxiliary variables
  double prefactor(double epsilonr0,double sigma0){return sigma0/(1+(sigma0*Mu0)/(4*epsilonr0));};
  //Fields from direct sums
  double rhoc(int ix,int iy,int iz,bool UseNew);
  vector3D D(int ix,int iy,int iz,bool UseNew);
  vector3D B(int ix,int iy,int iz,bool UseNew);
  //Fields deduced from the first ones through electromagnetic constants
  vector3D E(vector3D & D0,double Epsilonr);
  vector3D H(vector3D & B0,double Mur);
  //Forced (Actual) Macroscopic Fields (inline)
  vector3D Jprima(vector3D & E0,double prefactor0){return E0*prefactor0;};
  vector3D Eprima(vector3D & E0,vector3D & Jprima0,double epsilonr0){return E0-Jprima0*(Mu0/(4*epsilonr0));};
  vector3D Dprima(vector3D & Eprima0,double epsilonr0){return Eprima0/epsilonr0;};
  //Equilibrium Functions
  double feq(vector3D & Jprima0,vector3D & Eprima0,vector3D & B0,double Epsilonr,double Mur,int r,int p,int i,int j);
  double feq0(double rhoc0);
  //Simulation Functions
  void Start(void);
  void Collision(void);
  void ImposeFields(int t);
  void Advection(void);
  void Print(void);
};
```

Functions (some examples)

```
//Fields from direct sums

    vector3D LatticeBoltzmann::B(int ix,int iy,int iz,bool UseNew){
      int p,i,j; vector3D sum; sum.cargue(0,0,0);
      for(p=0;p<3;p++)
        for(i=0;i<4;i++)
          for(j=0;j<2;j++)
            if(UseNew)
              sum+=b[p][i][j]*fnew[ix][iy][iz][1][p][i][j];
            else
              sum+=b[p][i][j]*f[ix][iy][iz][1][p][i][j];
      return sum;
    }

  //Fields deduced from the first ones through electromagnetic constants

      vector3D LatticeBoltzmann::H(vector3D & B0,double Mur){
        return B0*(1.0/Mur);
      }

   //Forced (Actual) Macroscopic Fields (inline)

vector3D Eprima(vector3D & E0,vector3D & Jprima0,double epsilonr0){return E0-Jprima0*(Mu0/(4*epsilonr0));};
```



Equilibrium Functions

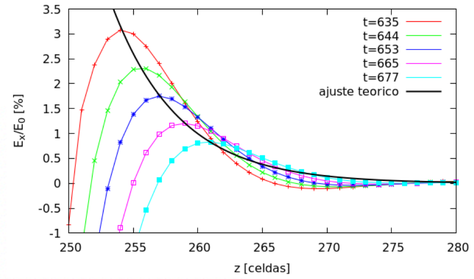
```
double LatticeBoltzmann::feq(vector3D & Jprima0,vector3D & Eprima0,vector3D & B0,
                             double epsilonr0,double mur0,
                             int r,int p,int i,int j){
  double VdotJp=(v[p][i]*Jprima0),Epdote=(e[p][i][j]*Eprima0),Bdotb=(b[p][i][j]*B0),aux;
  if(r==0)
    aux=0.25*(0.25*VdotJp+epsilonr0*Epdote+0.5/mur0*Bdotb);
  if(r==1)
    aux=0.25*(0.25*VdotJp+Epdote+0.5*Bdotb);
  return aux;
}
double LatticeBoltzmann::feq0(double rhoc0){
  return rhoc0;
}
```

3 **Corrected equilibrium distributions**

$$f_{ij}^{p(0)\mathrm{eq}} = \frac{1}{16}(\vec{v}_i^{\,p}\cdot\vec{J}') + \frac{\epsilon_{\mathbf{r}}}{4}(\vec{e}_{ij}^{\,p}\cdot\vec{E}') + \frac{1}{8\mu_r}(\vec{b}_{ij}^{\,p}\cdot\vec{B})$$

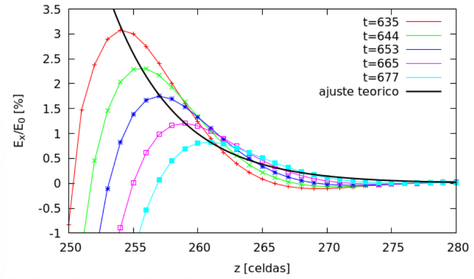
$$f_{ij}^{p(1)\mathrm{eq}} = \frac{1}{16}(\vec{v}_i^{\,p}\cdot\vec{J}') + \frac{1}{4}(\vec{e}_{ij}^{\,p}\cdot\vec{E}') + \frac{1}{8}(\vec{b}_{ij}^{\,p}\cdot\vec{B})$$

Simulation functions: Start



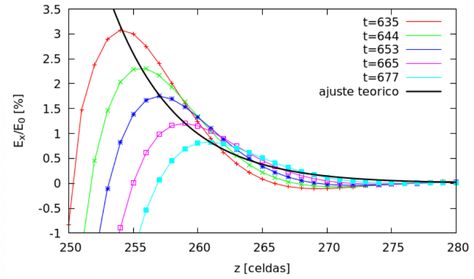

```
void LatticeBoltzmann::Start(void){
  int ix,iy,iz,r,p,i,j; double sigma0,mur0,epsilonr0,prefactor0;
  double rhoc0; vector3D D0,B0,E0,H0,Jprima0,Eprima0;
  for(ix=0;ix<Lx;ix++) //para cada celda
    for(iy=0;iy<Ly;iy++)
      for(iz=0;iz<Lz;iz++){
        //Compute the constants
        sigma0=sigma(ix,iy,iz); mur0=mur(ix,iy,iz); epsilonr0=epsilonr(ix,iy,iz);
        prefactor0=prefactor(epsilonr0,sigma0);
        //Impose the fields
        rhoc0=0; D0.cargue(0,0,0); B0.cargue(0,0,0);
        E0=E(D0,epsilonr0); H0=H(B0,mur0);
        Jprima0=Jprima(E0,prefactor0); Eprima0=Eprima(E0,Jprima0,epsilonr0);
        //Impose f=fnew=feq with the desired fields
        f0new[ix][iy][iz]=f0[ix][iy][iz]=feq0(rhoc0);
        for(r=0;r<2;r++)
          for(p=0;p<3;p++)
            for(i=0;i<4;i++)
              for(j=0;j<2;j++)
                fnew[ix][iy][iz][r][p][i][j]=f[ix][iy][iz][r][p][i][j]=
                  feq(Jprima0,Eprima0,B0,epsilonr0,mur0,r,p,i,j);
      }
}
```

Simulation functions: Collision




```
void LatticeBoltzmann::Collision(void){
  int ix,iy,iz,r,p,i,j; double sigma0,mur0,epsilonr0,prefactor0;
  double rhoc0; vector3D D0,B0,E0,H0,Jprima0,Eprima0;
  for(ix=0;ix<Lx;ix++) //para cada celda
    for(iy=0;iy<Ly;iy++)
      for(iz=0;iz<Lz;iz++){
        //Compute the constants
        sigma0=sigma(ix,iy,iz); mur0=mur(ix,iy,iz); epsilonr0=epsilonr(ix,iy,iz);
        prefactor0=prefactor(epsilonr0,sigma0);
        //Compute the fields
        rhoc0=rhoc(ix,iy,iz,false); D0=D(ix,iy,iz,false); B0=B(ix,iy,iz,false);
        E0=E(D0,epsilonr0); H0=H(B0,mur0);
        Jprima0=Jprima(E0,prefactor0); Eprima0=Eprima(E0,Jprima0,epsilonr0);
        //BGK evolution rule
        f0new[ix][iy][iz]=UmUTau*f0[ix][iy][iz]+UTau*feq0(rhoc0);
        for(r=0;r<2;r++)
          for(p=0;p<3;p++)
            for(i=0;i<4;i++)
              for(j=0;j<2;j++)
                fnew[ix][iy][iz][r][p][i][j]=UmUTau*f[ix][iy][iz][r][p][i][j]
                  +UTau*feq(Jprima0,Eprima0,B0,epsilonr0,mur0,r,p,i,j);
      }
}
```

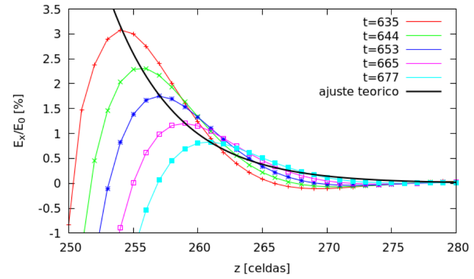
Simulation functions: ImposeFields




```
void LatticeBoltzmann::ImposeFields(int t){
  int ix,iy,iz,r,p,i,j; double sigma0,mur0,epsilonr0,prefactor0;
  double rhoc0; vector3D D0,B0,E0,H0,Jprima0,Eprima0;
  double T=25.0,omega=2*M_PI/T;//25<T<50
  iz=0; //On the whole incident plane
  for(ix=0;ix<Lx;ix++) //for each cell
    for(iy=0;iy<Ly;iy++){
      //Compute the constants
      sigma0=sigma(ix,iy,iz); mur0=mur(ix,iy,iz); epsilonr0=epsilonr(ix,iy,iz);
      prefactor0=prefactor(epsilonr0,sigma0);
      //Compute the fields
      //Primary fields (i.e. those from the sums)
      rhoc0=rhoc(ix,iy,iz,false);
      D0.cargue(E00*sin(omega*t)*epsilonr0,0,0);
      B0.cargue(0,B00*sin(omega*t),0);
      //Secundary fields (i.e. computed from the primary fields)
      E0=E(D0,epsilonr0); H0=H(B0,mur0);
      Jprima0=Jprima(E0,prefactor0); Eprima0=Eprima(E0,Jprima0,epsilonr0);
      //Impose fnew=feq with the desired fields
      for(r=0;r<2;r++)
        for(p=0;p<3;p++)
          for(i=0;i<4;i++)
            for(j=0;j<2;j++){
              fnew[ix][iy][iz][r][p][i][j]=feq(Jprima0,Eprima0,B0,epsilonr0,mur0,r,p,i,j);
              f0new[ix][iy][iz]=feq0(rhoc0);
            }
    }
}
```

A sinusoidal oscillation on the plane iz=0

Simulation functions: Print the results



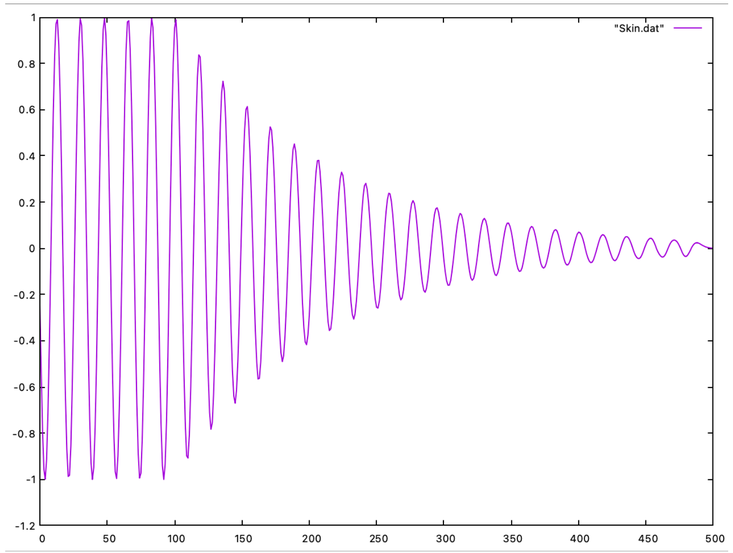

```
void LatticeBoltzmann::Print(void){
  int ix=0,iy=0,iz,r,p,i,j; double sigma0,mur0,epsilonr0,prefactor0;
  double rhoc0; vector3D D0,B0,E0,H0,Jprime0,Eprime0; double E2,B2;
  for(iz=0;iz<Lz/2;iz++){
    //Compute the electromagnetic constants
    sigma0=sigma(ix,iy,iz); mur0=mur(ix,iy,iz); epsilonr0=epsilonr(ix,iy,iz);
    prefactor0=prefactor(epsilonr0,sigma0);
    //Compute the Fields
    rhoc0=rhoc(ix,iy,iz,true); D0=D(ix,iy,iz,true); B0=B(ix,iy,iz,true);
    E0=E(D0,epsilonr0); H0=H(B0,mur0);
    Jprime0=Jprime(E0,prefactor0); Eprime0=Eprime(E0,Jprime0,epsilonr0);
    //Print
    cout<<iz<<" "<<E0.x()/E00<<endl;
  }
}
```

main ()

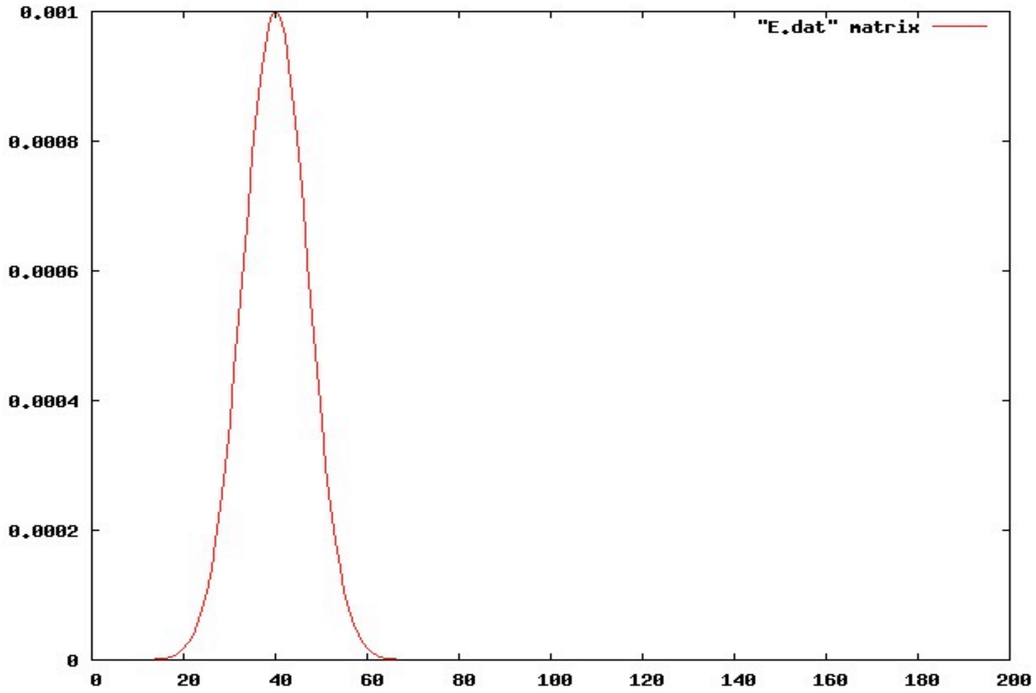
```
int main(){
  LatticeBoltzmann SkinWaves;
  int t, tmax=700;

  SkinWaves.Start();
  SkinWaves.ImposeFields(0);

  for(t=0;t<tmax;t++){
    SkinWaves.Collision();
    SkinWaves.ImposeFields(t);
    SkinWaves.Advection();
  }

  SkinWaves.Print();

  return 0;
}
```

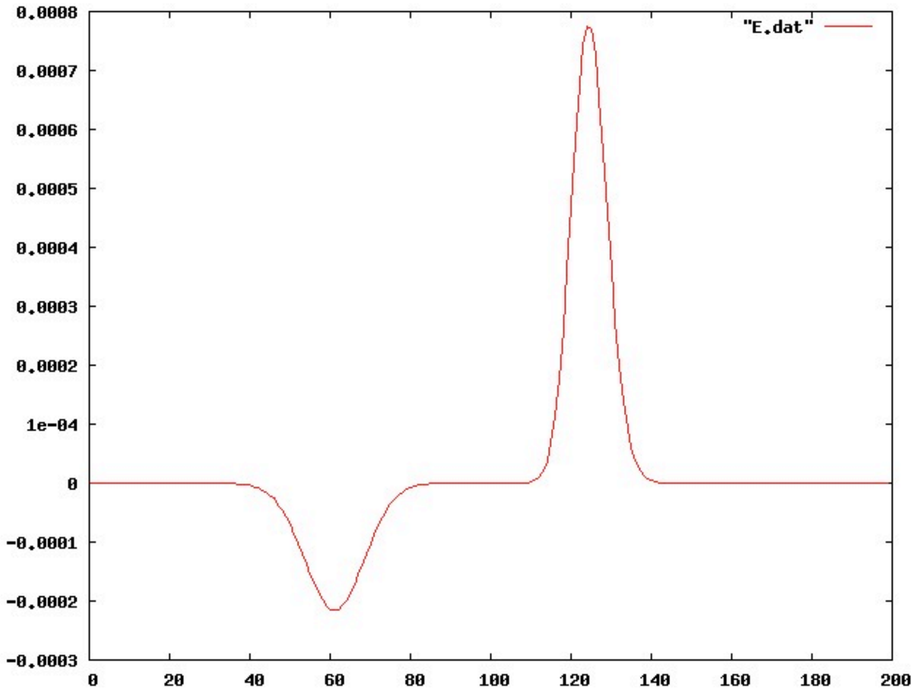
```
> emacs EMLBM_Skin.cpp &
> g++ EMLBM_Skin.cpp
> ./a.out > Skin.dat
```

```
> gnuplot
> plot “Skin.dat” w l
> exit
```



Pulse Reflection



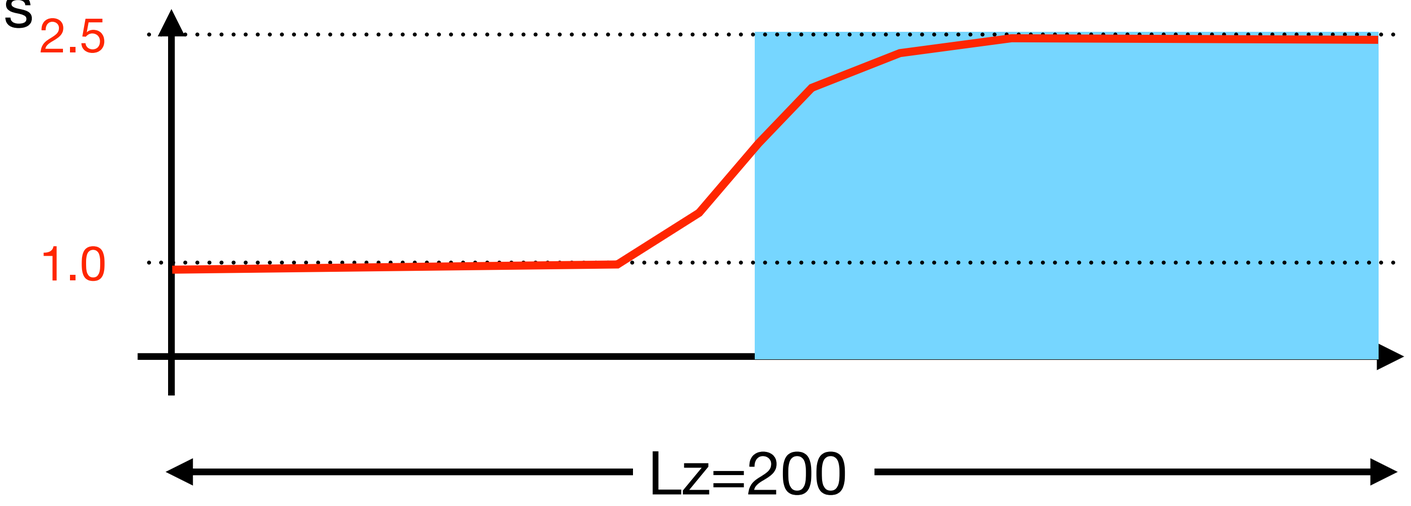


D2Q9 and feq

1. Make a copy of EMLBM_Skin.cpp and change the name to EMLBM_Dielectric.cpp

2. Change constants and functions

```
const int Lx = 1;   //
const int Ly = 1;   //
const int Lz = 200; //
//--------------------
const double Tau = 0.5;
const double UTau = 1/Tau;
const double UmUTau=1-1/Tau;
//--------------------
const double Epsilon0=1, Mu0=2;
const double Sigma0=0.0;
const double C=1.0/sqrt(2.0);

const double E00=1.0,B00=E00/C;
```




```
double mur(int ix,int iy,int iz){
  return 1.0;
}
double epsilonr(int ix,int iy,int iz){
  return 1.75+0.75*tanh((double)(iz-(Lz/2)));
}
double sigma(int ix,int iy,int iz){
  return 0.0;
}
```

3 Change the initial condition for a Gaussian pulse

```
void LatticeBoltzmann::Start(void){
  int ix,iy,iz,r,p,i,j; double sigma0,mur0,epsilonr0,prefactor0;
  double rhoc0; vector3D D0,B0,E0,H0,Jprime0,Eprime0;
  double alpha0=5.0,iz0=40;
  for(ix=0;ix<Lx;ix++) //para cada celda
    for(iy=0;iy<Ly;iy++)
      for(iz=0;iz<Lz;iz++){
        //Compute the constants
        sigma0=sigma(ix,iy,iz); mur0=mur(ix,iy,iz); epsilonr0=epsilonr(ix,iy,iz);
        prefactor0=prefactor(epsilonr0,sigma0);
        //Impose the fields
        rhoc0=0; Jprime0.cargue(0,0,0);
        B0.cargue(0,B00*exp(-0.25*(iz-iz0)*(iz-iz0)/(alpha0*alpha0)),0);
        Eprime0.cargue(E00*exp(-0.25*(iz-iz0)*(iz-iz0)/(alpha0*alpha0)),0,0);
        D0=E0*epsilonr0; H0=H(B0,mur0);
```

A gaussian pulse

4 Get rid of imposing fields

```
void LatticeBoltzmann::ImposeFields(int t){
}
```

5 Print

```
void LatticeBoltzmann::Print(void){
  int ix=0,iy=0,iz,r,p,i,j; double sigma0,mur0,epsilonr0,prefactor0;
  double rhoc0; vector3D D0,B0,E0,H0,Jprime0,Eprime0; double E2,B2;
  for(iz=0;iz<Lz;iz++){
    //Compute the electromagnetic constants
    sigma0=sigma(ix,iy,iz); mur0=mur(ix,iy,iz); epsilonr0=epsilonr(ix,iy,iz);
    prefactor0=prefactor(epsilonr0,sigma0);
    //Compute the Fields
    rhoc0=rhoc(ix,iy,iz,true); D0=D(ix,iy,iz,true); B0=B(ix,iy,iz,true);
    E0=E(D0,epsilonr0); H0=H(B0,mur0);
    Jprime0=Jprime(E0,prefactor0); Eprime0=Eprime(E0,Jprime0,epsilonr0);
    //Print
    E2=norma2(Eprime0); B2=norma2(B0);
    cout<<iz<<" "<<0.5*(Epsilon0*epsilonr0*E2+B2/(Mu0*mur0))<<endl;
  }
}
```

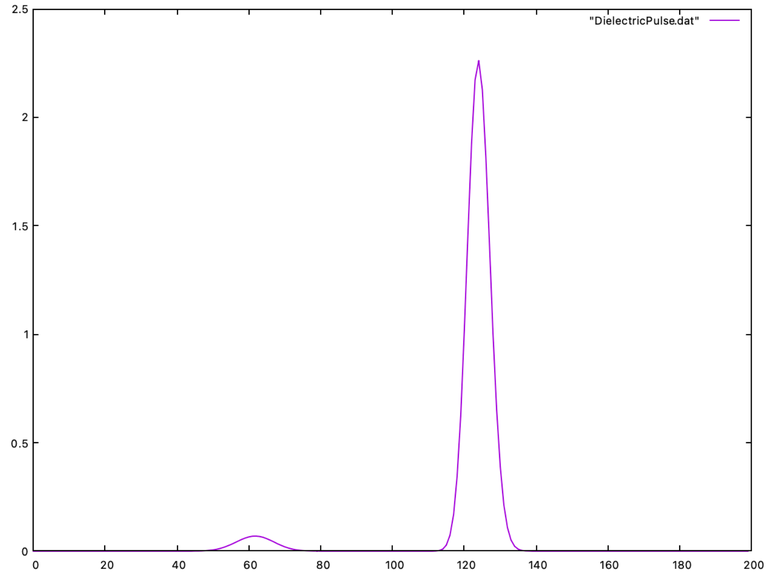
Energy & iz

Modify the main ()

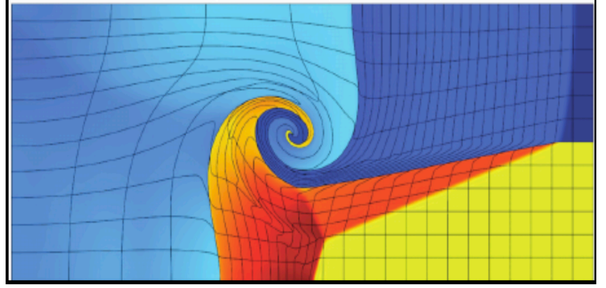
9
```
int main(){
  LatticeBoltzmann DielectricPulse;
  int t, tmax=140;

  DielectricPulse.Start();
  DielectricPulse.ImposeFields(0);

  for(t=0;t<tmax;t++){
    DielectricPulse.Collision();
    DielectricPulse.ImposeFields(t);
    DielectricPulse.Advection();
  }

  DielectricPulse.Print();

  return 0;
```

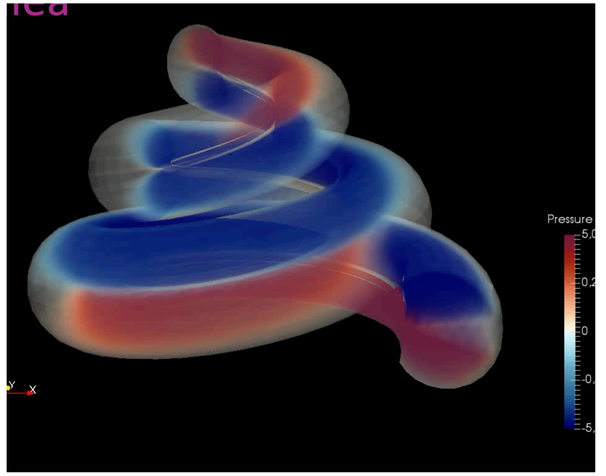
```
> emacs EMLBM_Dielectric.cpp &
> g++ EMLBM_Dielectric.cpp
> ./a.out > DielectricPulse.dat
```

```
> gnuplot
> plot “DielectricPulse.dat” w l
> exit
```



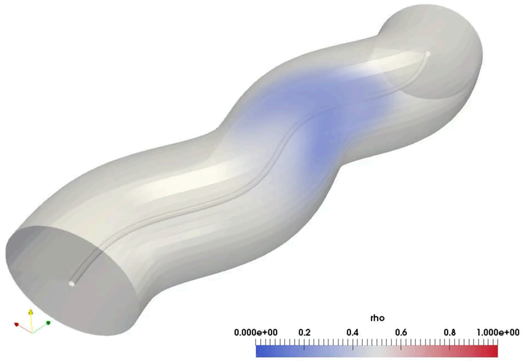
Lattice - Boltzmann Crash Course
Session 4b - Curved coordinates

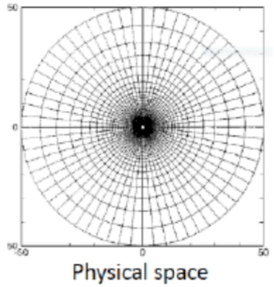




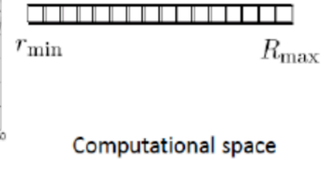
Prof. José Daniel Muñoz

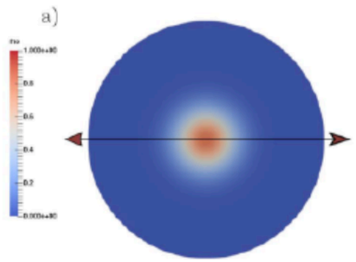
Simulation of Physical Systems Group,
Department of Physics, Universidad Nacional de Colombia, Bogota
Crr 30 # 45-03, Ed. 404, Of. 348, Bogotá D.C. 111321, Colombia

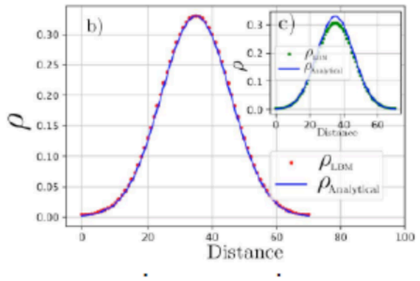


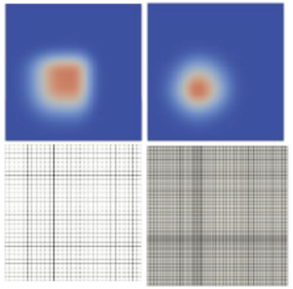


$$x = r\cos\theta \qquad y = r\sin\theta \qquad g = \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix} \qquad \Gamma^{r}_{\theta\theta} = -r \qquad \Gamma^{\theta}_{r\theta} = \Gamma^{\theta}_{\theta r} = \frac{1}{r}$$









https://drive.google.com/drive/folders/1zzzTctPTXmEG4nqfR90LQlm1gyTPR3fx?usp=sharing

Website

HERN Helmholtz Institute Erlangen-Nürnberg

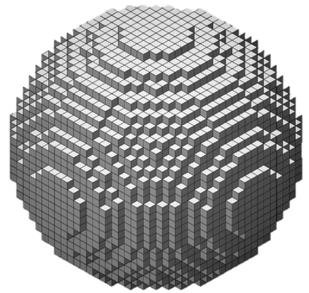
El conocimiento es de todos

Minciencias

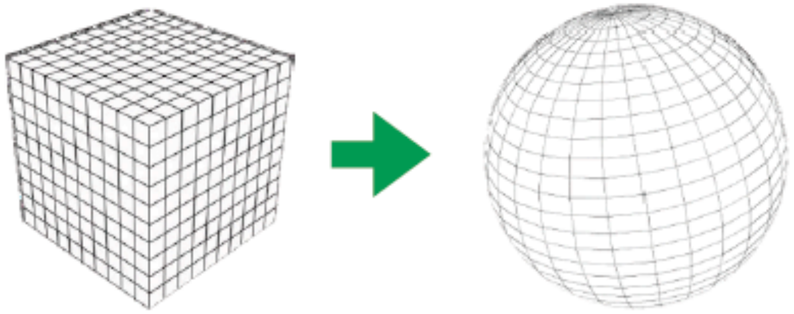
UNIVERSIDAD NACIONAL DE COLOMBIA

Why generalized coordinates?

- To avoid staircase approximations.
- To handle complex geometries, even time-dependent ones.
- To simplify boundary conditions on those geometries.
- To take advantage of symmetries.
- To increase mesh resolution, just where you need.

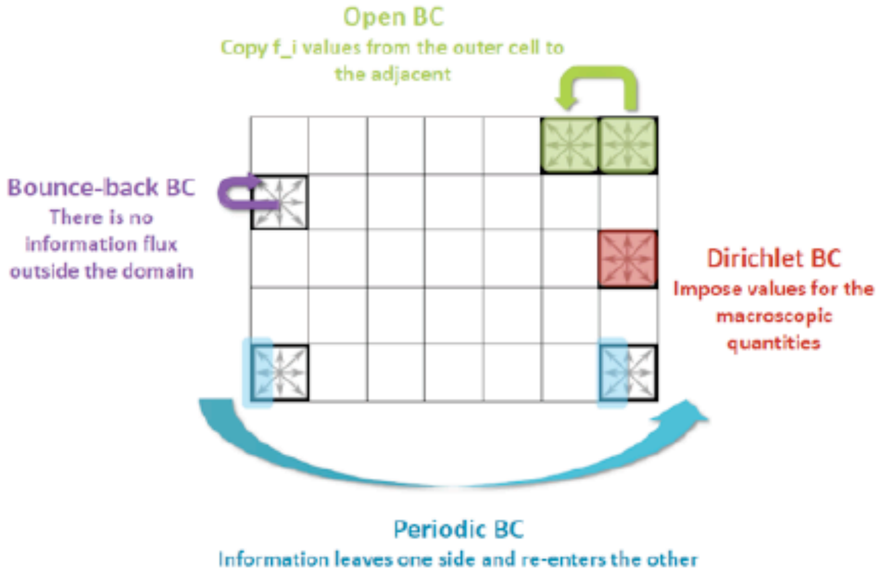


Strategy: to simulate on the computer a cubic array of cells that adds the geometrical terms corresponding to the desired equation in generalized coordinates



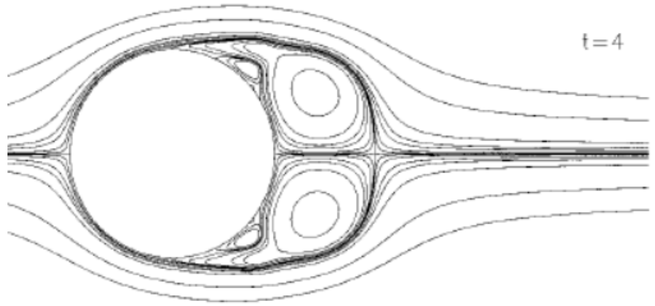
$$\frac{\partial^2 p}{\partial t^2} - \frac{c^2}{\sqrt{g}} \frac{\partial}{\partial x^i} \left(\sqrt{g} g^{ik} \frac{\partial p}{\partial x^k} \right) = 0,$$

$$\frac{\partial^2 p}{\partial t^2} - c^2 \nabla^2 p = 0.$$



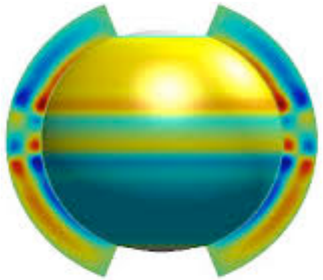
Previous works on fluids

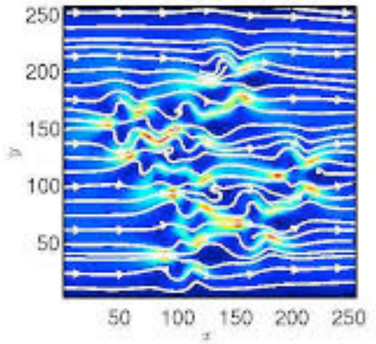
- Flow around a Circular Cylinder. He and Doolen 1997



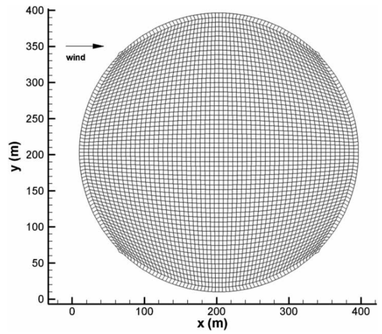
Miller Mendoza
Ph.D. dissertation in ETH
(Prof. H. Herrmann)

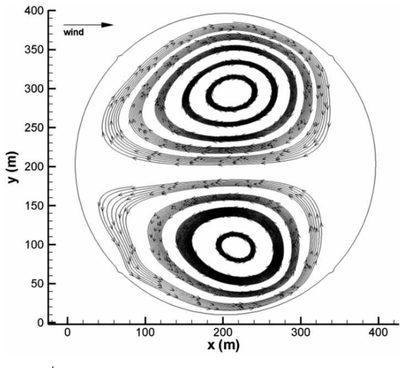
- Couette Flows and campylotic media, M.Mendoza PhD Thesis 2012





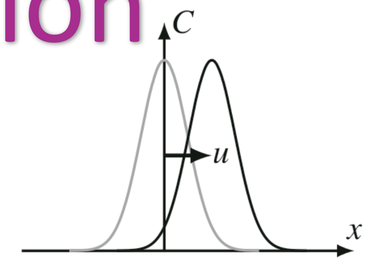
- Lattice Boltzmann method for 2D flows in curvilinear coordinates. Budinsky 2012



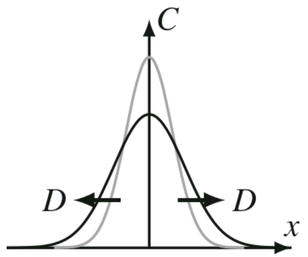


Advection-Diffusion

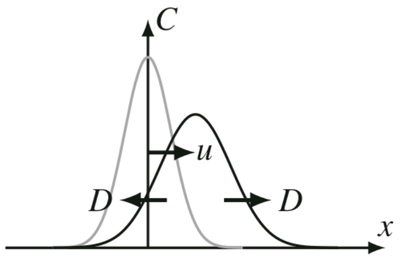
$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = D \nabla^2 \rho$$



Pure advection

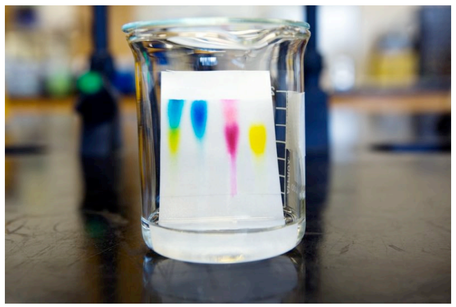


Pure diffusion

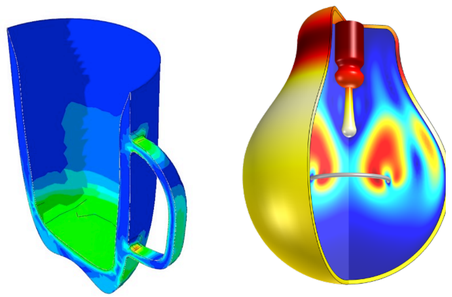


Advection and diffusion

Sediment transport



Chromatography



Heat transfer

Advection-Diffusion

1. On Cartesian coordinates

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = D\nabla^2 \rho + \vec{S}(\vec{x}, t)$$

Source term

New BGK evolution rule*:

$$f_\lambda(\vec{x} + \vec{c}_\lambda \Delta t, t + \Delta t) - f_\lambda(\vec{x}, t) = -\frac{1}{\tau}[f_\lambda(\vec{x}, t) - f_\lambda^{eq}(\vec{x}, t)] + \underbrace{\Delta t S_\lambda + \frac{\Delta t^2}{2} \bar{D}_\lambda S_\lambda}_{\text{new terms}}$$

with

$$S_\lambda = \omega_\lambda S \left(1 + \frac{\tau - \frac{1}{2}}{\tau - \frac{\theta}{2}} \frac{\vec{c}_\lambda \cdot \vec{u}}{c_s^2}\right) \qquad D_\lambda S_\lambda = \frac{S_\lambda(\vec{x}, t) - S_\lambda(\vec{x} - \vec{c}_\lambda \Delta t, t - \Delta t)}{\Delta t}$$

Discrete forward scheme

The source term fulfills:

$$\sum_i S_\lambda = S \quad , \quad \sum_i \vec{c}_\lambda S_\lambda = \frac{\tau - \frac{1}{2}}{\tau - \frac{\theta}{2}} S\vec{u}$$

Through a Chapman-Enskog expansion, the ADE with source is recovered in the continuous limit.

Juliana García

Miller Mendoza

* B. Shi et al., Computers and Mathematics with Applications 55 (2008) 1568 -1575

LBM for the ADE in generalized coordinates

Juliana García

Miller Mendoza

Advection-diffusion equation **with** source:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = D\nabla^2 \rho + \vec{S}(\vec{x}, t)$$

Differential operators in generalized coordinates:

$$\nabla_i(\rho u^i) = \frac{\partial}{\partial \eta_i}(\rho u^i) + \Gamma^i_{ij}\rho u^j$$

Divergence

$$\nabla^2 \rho = \frac{1}{\sqrt{g}} \frac{\partial}{\partial \eta_i}(\sqrt{g} g^{ij} \frac{\partial \rho}{\partial \eta_j})$$

Laplace-Beltrami operator

Written in terms of:

Metric tensor:

$$g_{ij} = \sum_k \frac{\partial x_k}{\partial \eta_i} \frac{\partial x_k}{\partial \eta_j}$$

Christoffel symbols:

$$\Gamma^i_{jk} = \frac{1}{2} g^{il} \left(\frac{\partial g_{kl}}{\partial \eta_j} + \frac{\partial g_{jl}}{\partial \eta_k} - \frac{\partial g_{jk}}{\partial \eta_l} \right)$$

Now, we need to re-write our equation!

LBM for the ADE in generalized coordinates. Our model

Juliana García

Miller Mendoza

$$\partial_t \rho + \partial_i(\rho u^i) = D\partial_i^2 \rho + S_{\text{phys}}$$

$$\nabla_i(\rho u^i) = \frac{\partial}{\partial \eta_i}(\rho u^i) + \Gamma^i_{ij}\rho u^j$$

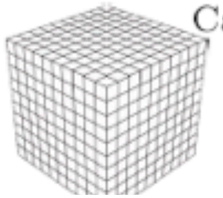
$$\nabla^2 \rho = \frac{1}{\sqrt{g}} \frac{\partial}{\partial \eta_i}(\sqrt{g} g^{ij} \frac{\partial \rho}{\partial \eta_j})$$

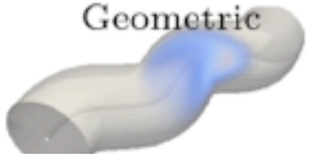
$$\partial_t \rho + \partial_i(\rho u^i) + \Gamma^i_{ij}\rho u^j = \frac{D}{\sqrt{g}}\partial_i(\sqrt{g} g^{ij}\partial_j \rho) + S_{\text{phys}}$$

$$\partial_t \rho + \partial_i(\rho u^i) = \frac{D}{\sqrt{g}}\partial_i(\sqrt{g} g^{ij}\partial_j \rho) - \Gamma^i_{ij}\rho u^j + S_{\text{phys}}$$

Strategy: subtract the old (Cartesian) and add the new

$$\partial_t \rho + \partial_i(\rho u^i) = \underbrace{D\partial_i^2 \rho}_{\text{Cartesian}} \underbrace{- D\partial_i^2 \rho + \frac{D}{\sqrt{g}}\partial_i(\sqrt{g} g^{ij}\partial_j \rho) - \Gamma^i_{ij}\rho u^j}_{\text{Geometric}} + S_{\text{phys}}$$





Geometric information as a new source term S.

LBM for the ADE in generalized coordinates. Our model

Juliana García

Miller Mendoza

We introduce the terms associated with the geometry into a new source term S, and a diffusive Cartesian contribution that needs to be removed:

$$\partial_t \rho + \partial_i(\rho u^i) = \underbrace{D\partial_i^2\rho - D\partial_i^2\rho}_{\text{Cartesian contribution}} + \underbrace{\frac{D}{\sqrt{g}}\partial_i(\sqrt{g}g^{ij}\partial_j\rho) - \Gamma^i_{ij}\rho u^j}_{\text{Geometry information}} + S_{\text{phys}}$$

$$\partial_t \rho + \partial_i(\rho u^i) = D\partial_i^2\rho + S$$

With the new source term as:

$$S = -D\partial_i^2\rho + \frac{D}{\sqrt{g}}\partial_i(\sqrt{g}g^{ij}\partial_j\rho) - \Gamma^i_{ij}\rho u^j + S_{phys}$$

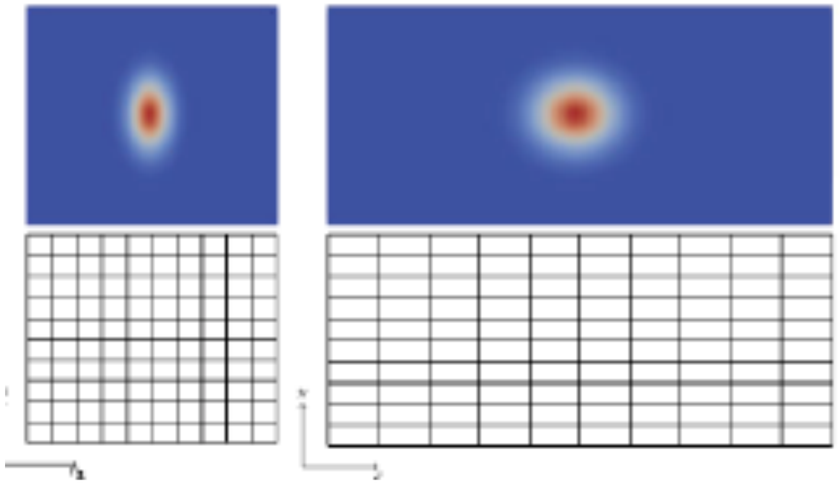
Our strategy consists in introducing the geometry information as a source term.

We need to compute some derivatives. Γ^i_{ij} $\partial\rho$

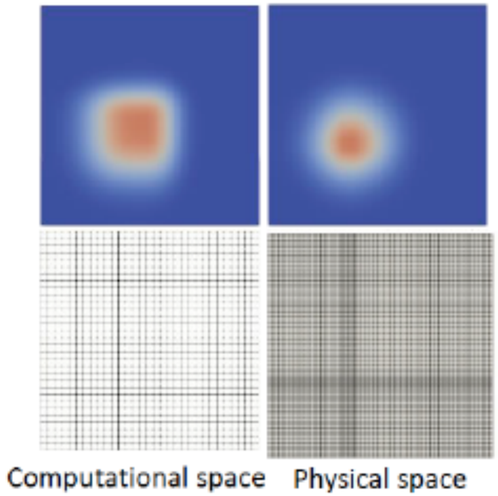
$$\partial_k\phi(\vec{x}) = \frac{1}{c_s^2\Delta t}\sum_{\lambda}^{m}\omega_\lambda c_\lambda^k \phi(\vec{x} + c_\lambda\Delta t)$$

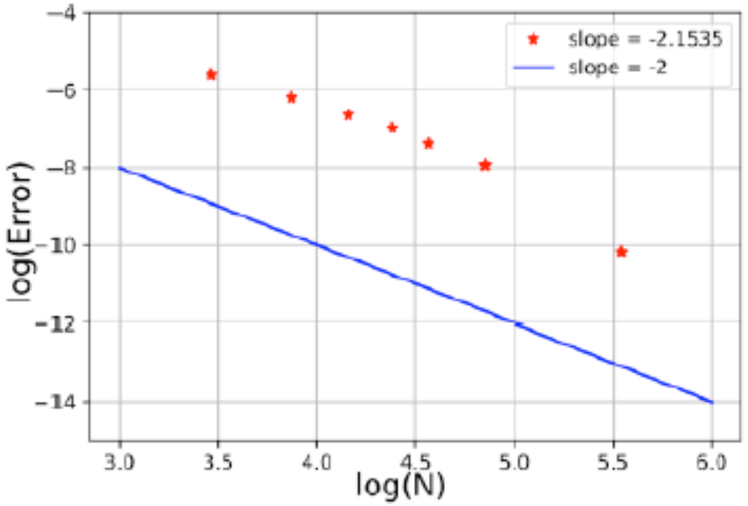
S. P. Thampi et al.. *Isotropic discrete laplacian operators from lattice hydrodynamics.* Journal of Computational Physics, 234(1):1–7, 2013.

Remains isotropic



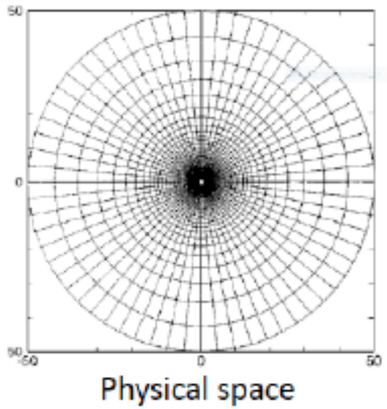
Allows to increase resolution in some places



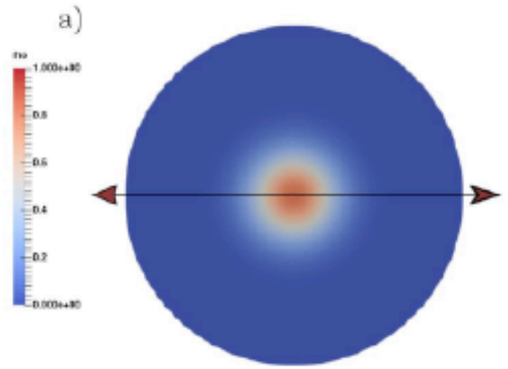


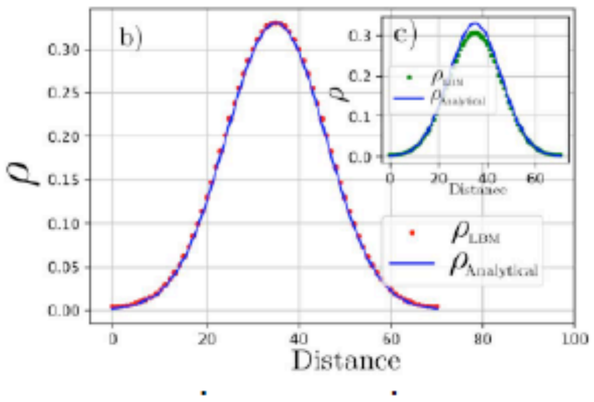
Allows to take advantage of symmetries

$$x = r\cos\theta \quad , \quad y = r\sin\theta \quad , \quad g = \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix} \quad , \quad \Gamma^r_{\theta\theta} = -r \quad , \quad \Gamma^\theta_{r\theta} = \Gamma^\theta_{\theta r} = \frac{1}{r} \quad ,$$

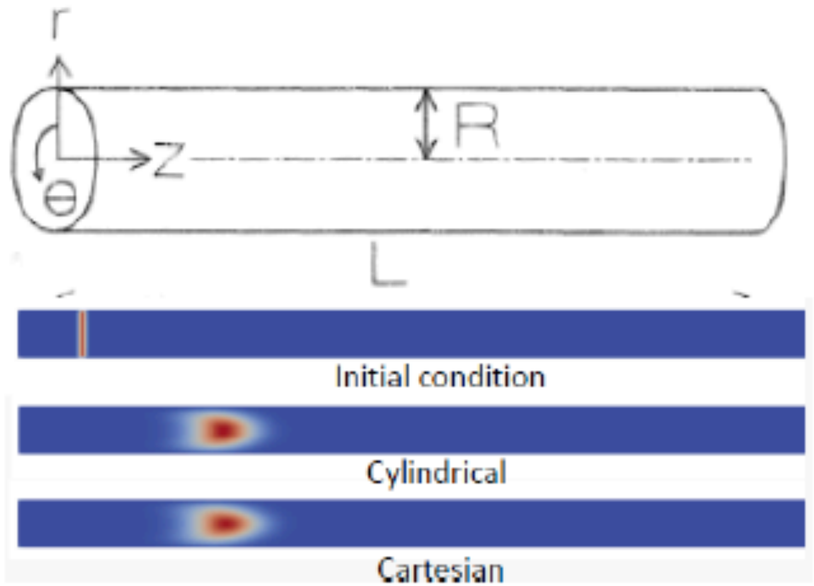


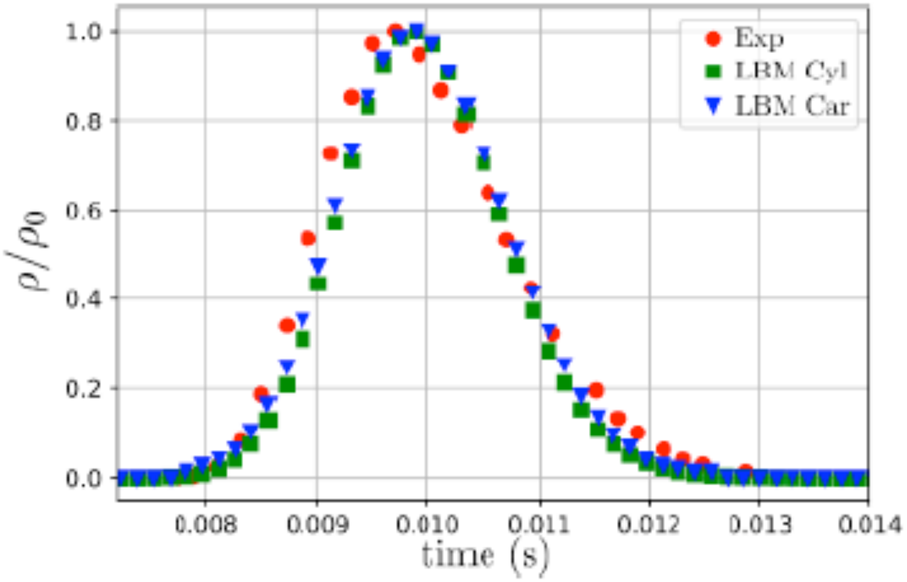
Computational space





Reproduces experimental data

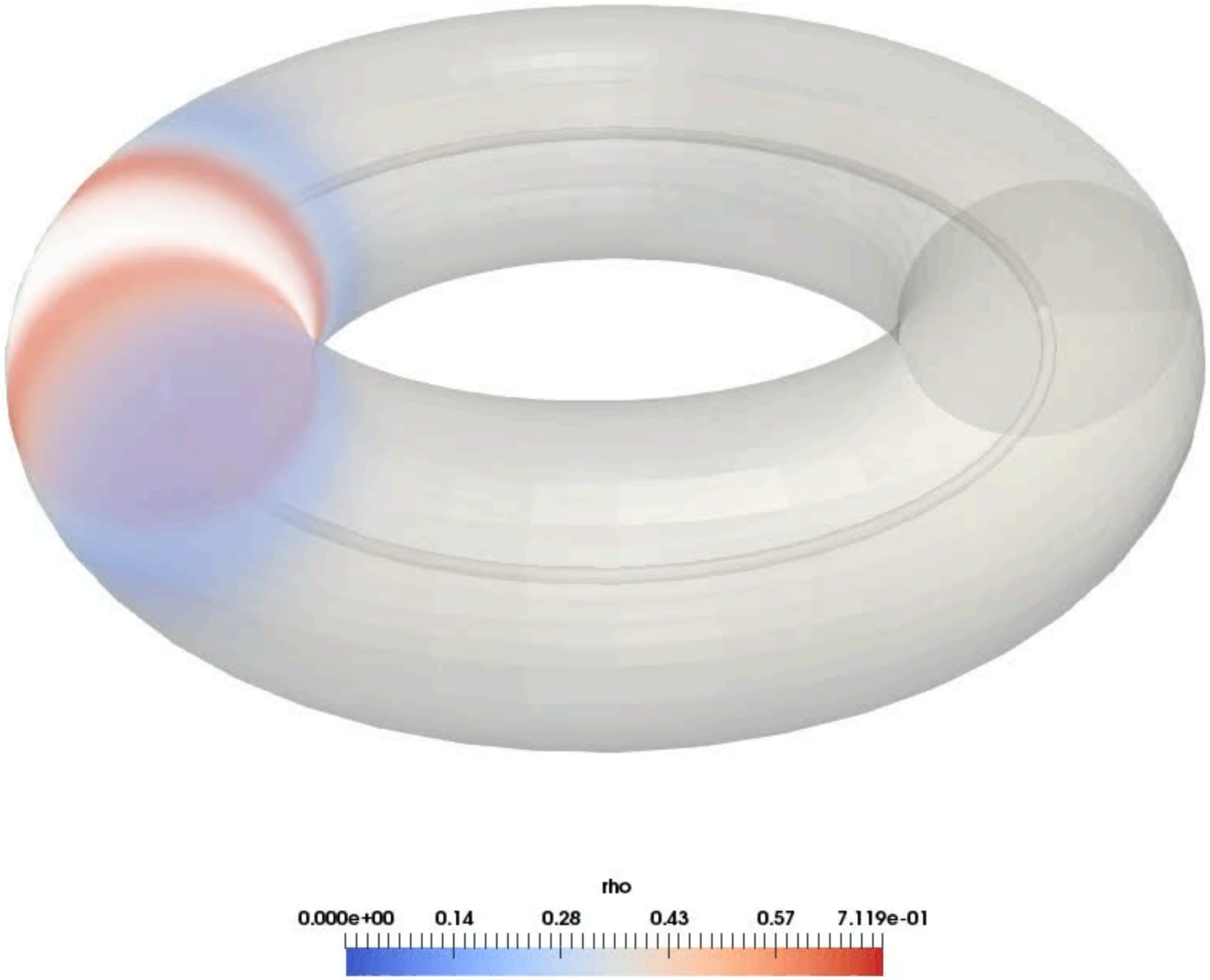




Quantity	Comp. units	Exp. value
Δx, Δr, Δz	1 cell	0.36cm
Δt	1 timestep	2.592×10^{-7}s
D	0.0048 cells^2/timestep	2400cm^2/s
V	0.004896 cells/timestep	6800cm/s
a	11 cells	3.96cm $\approx$ 4cm
Z	430 cells	154.8cm
t_{max}	57000 timestep	0.147744s

Torus

Pure diffusion.



Curved channel

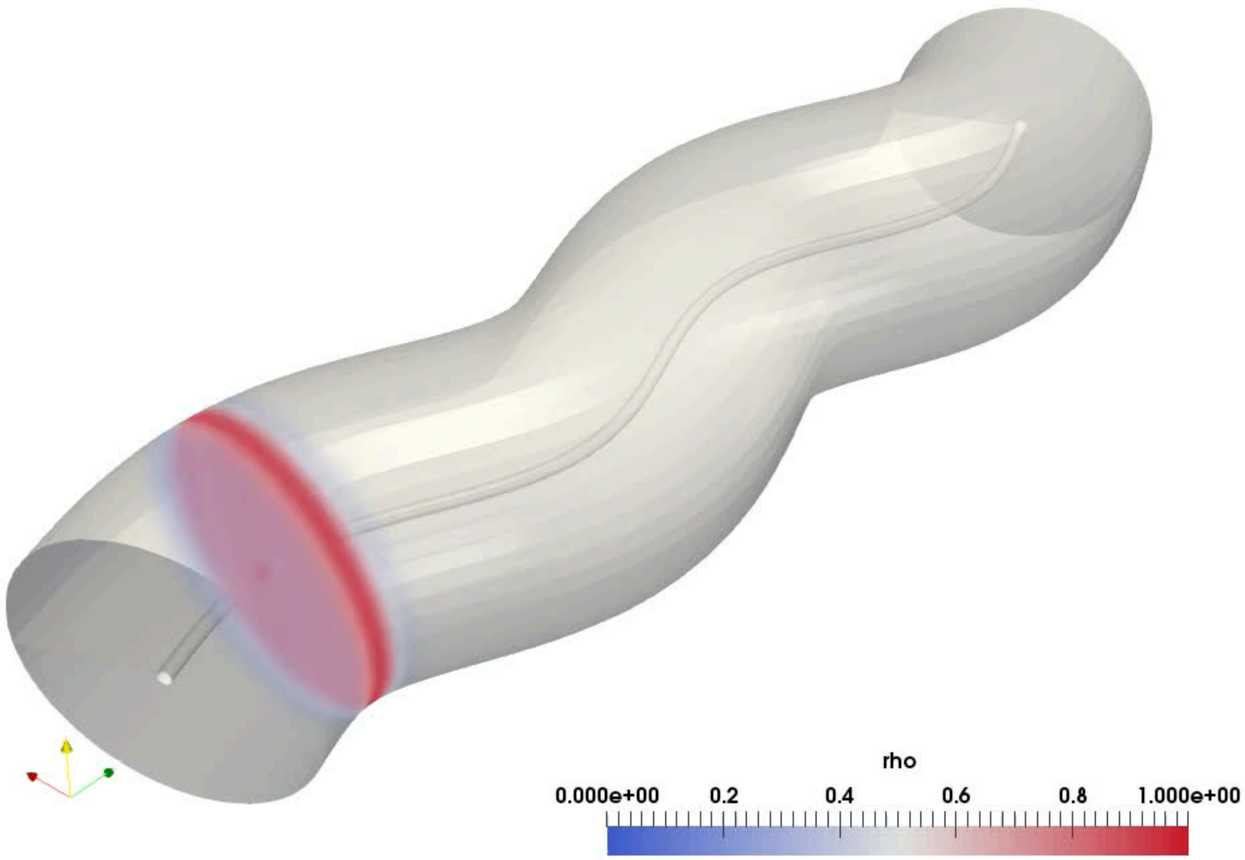
Advection and diffusion.

Initial distribution:

$$\rho(r,\theta,\zeta) = \exp\left(-\frac{(z(r,\theta,\zeta) - z_0(r,\theta,\zeta))^2}{2\sigma_0^2}\right)$$

Velocity profile:

$$u_\zeta(r) = 2V\left(1 - \left(\frac{r}{a}\right)^2\right)$$



Let's go for waves!

The wave equation on generalized coordinates

$$\frac{\partial^2 P}{\partial t^2} - \frac{c^2}{\sqrt{g}} \frac{\partial}{\partial x^\alpha} \left(\sqrt{g} g^{\alpha\beta} \frac{\partial P}{\partial x^\beta} \right) = 0,$$

Ali Mauricio Velasco

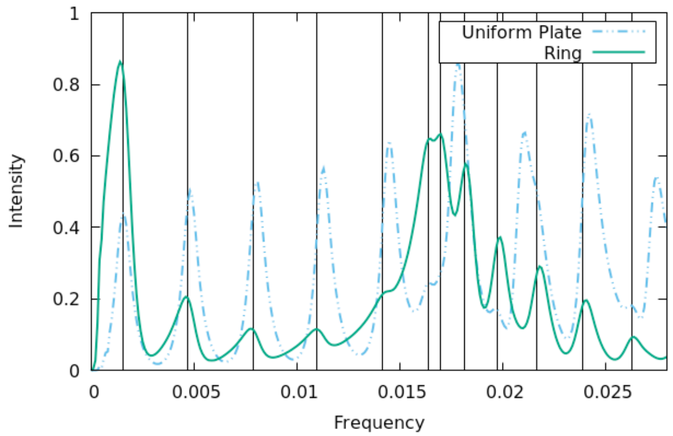
Can be reproduced with this model

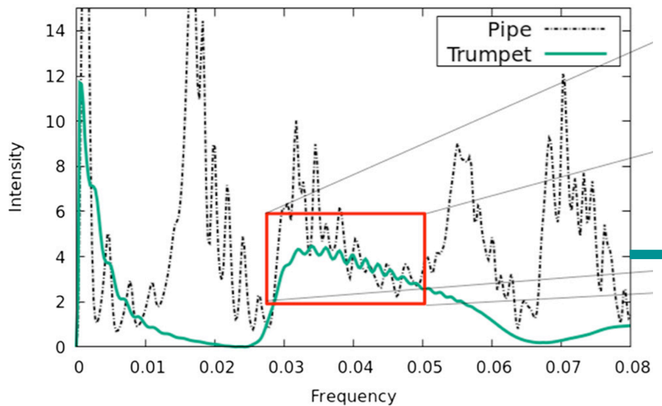
$$f^{eq} = \begin{cases} w_0 \sqrt{g} P & \text{for } i = 0 \\ w_i \sqrt{g} \left[P + \frac{\xi_i^k J'^k}{c_s^2} \right] & \text{otherwise} \end{cases}$$

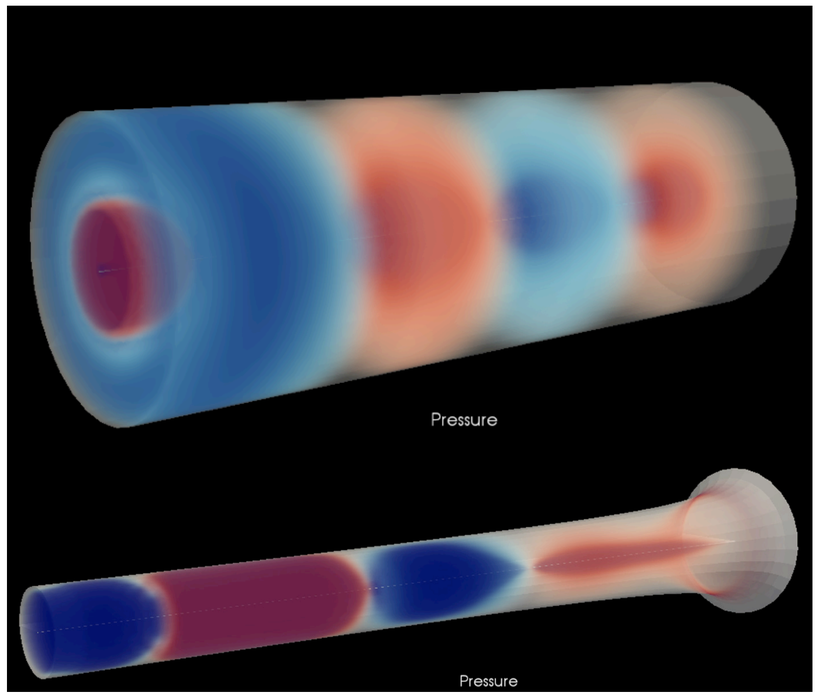
Forcement

$$\sqrt{g} P = \sum_l f_l^{eq}$$

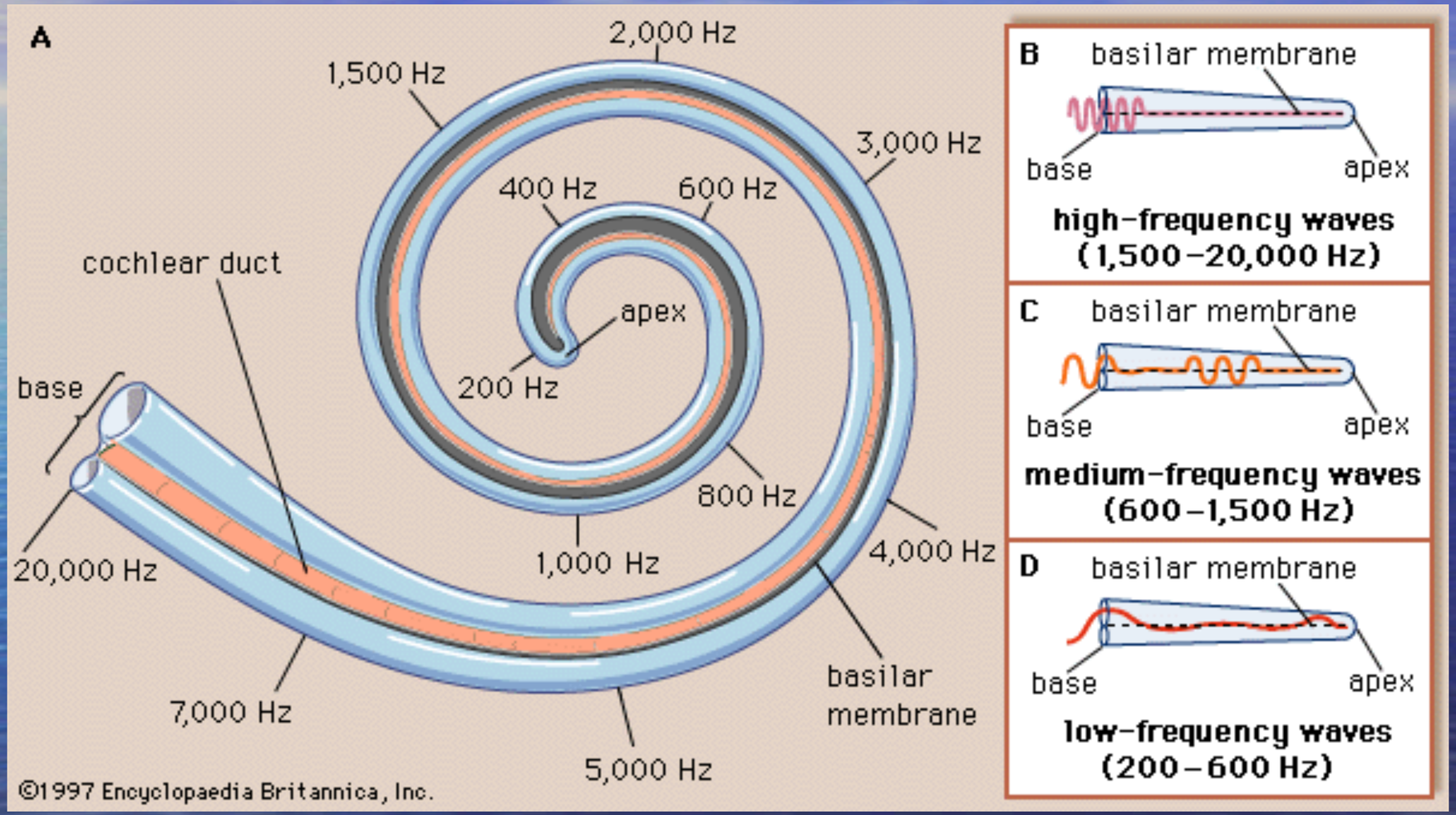
$$\sqrt{g} J'^i = \sum_l f_l^{eq} \xi_l^i - \frac{1}{2} c^2 P \Gamma^i_{jk} g^{jk} + \frac{1}{2} \partial_j \left[\sqrt{g} c^2 P \left(g^{ij} - \delta^{ij} \right) \right]$$



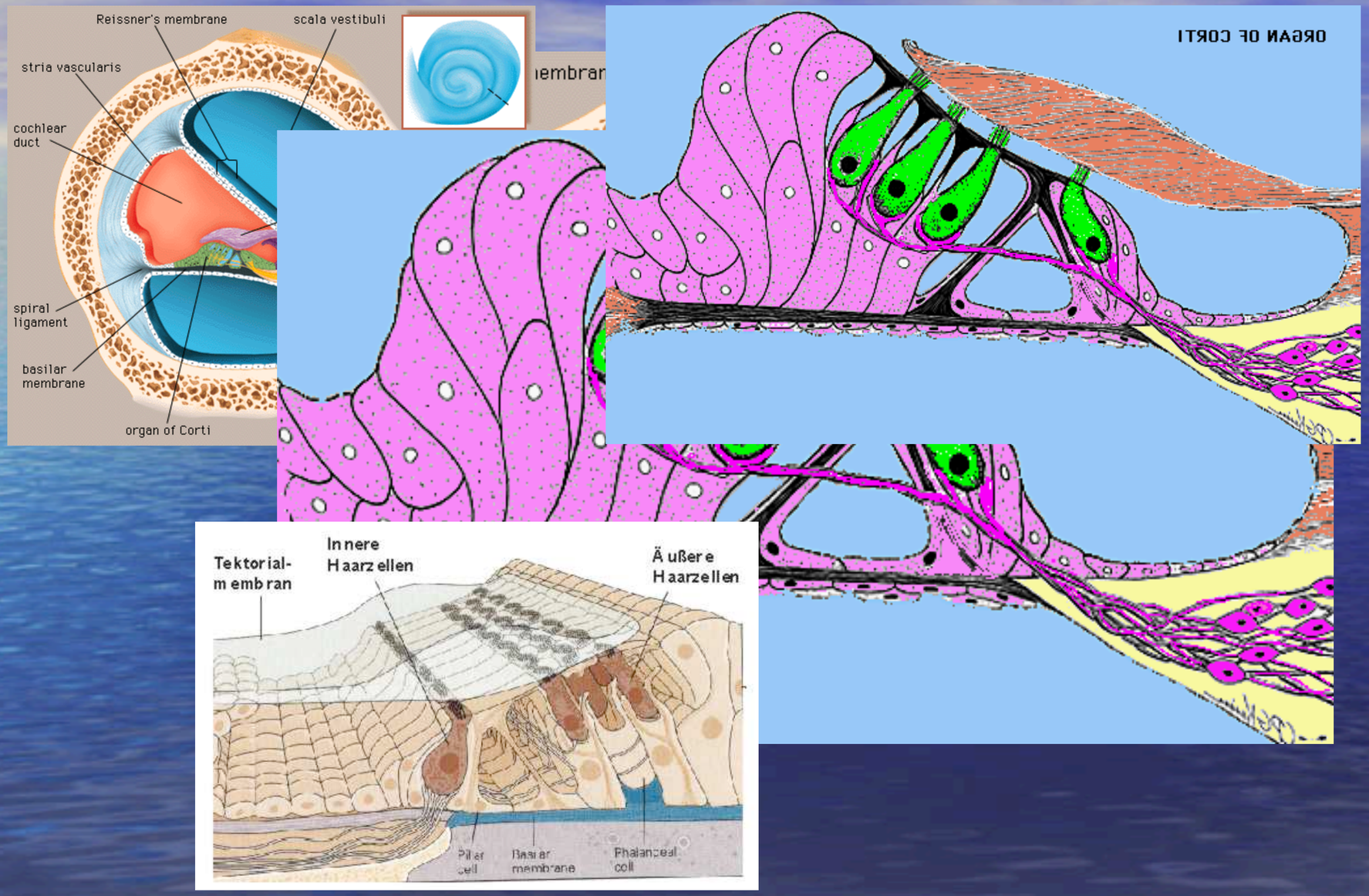




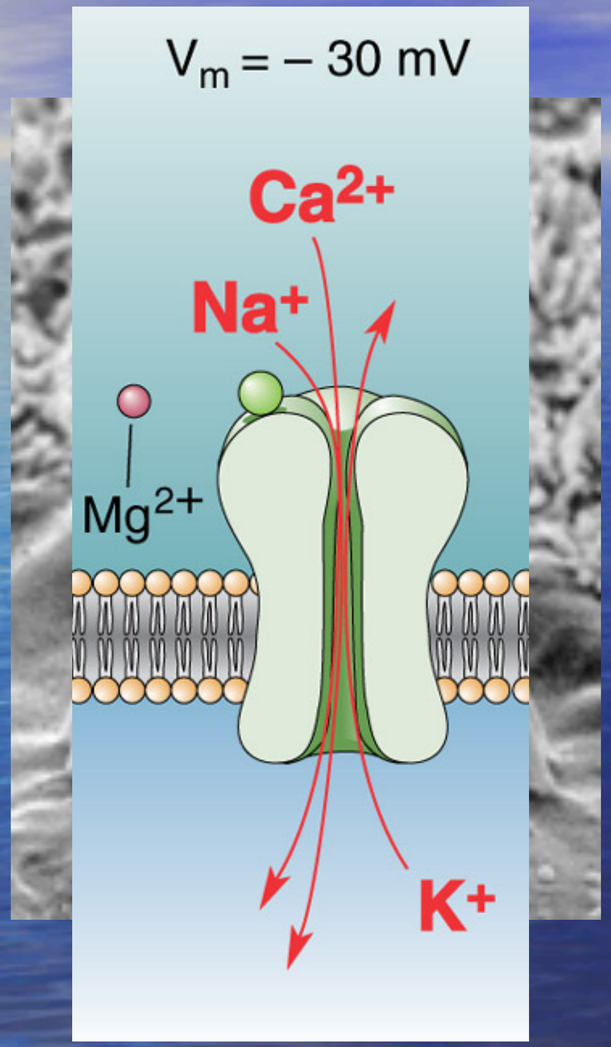
The Cochlea

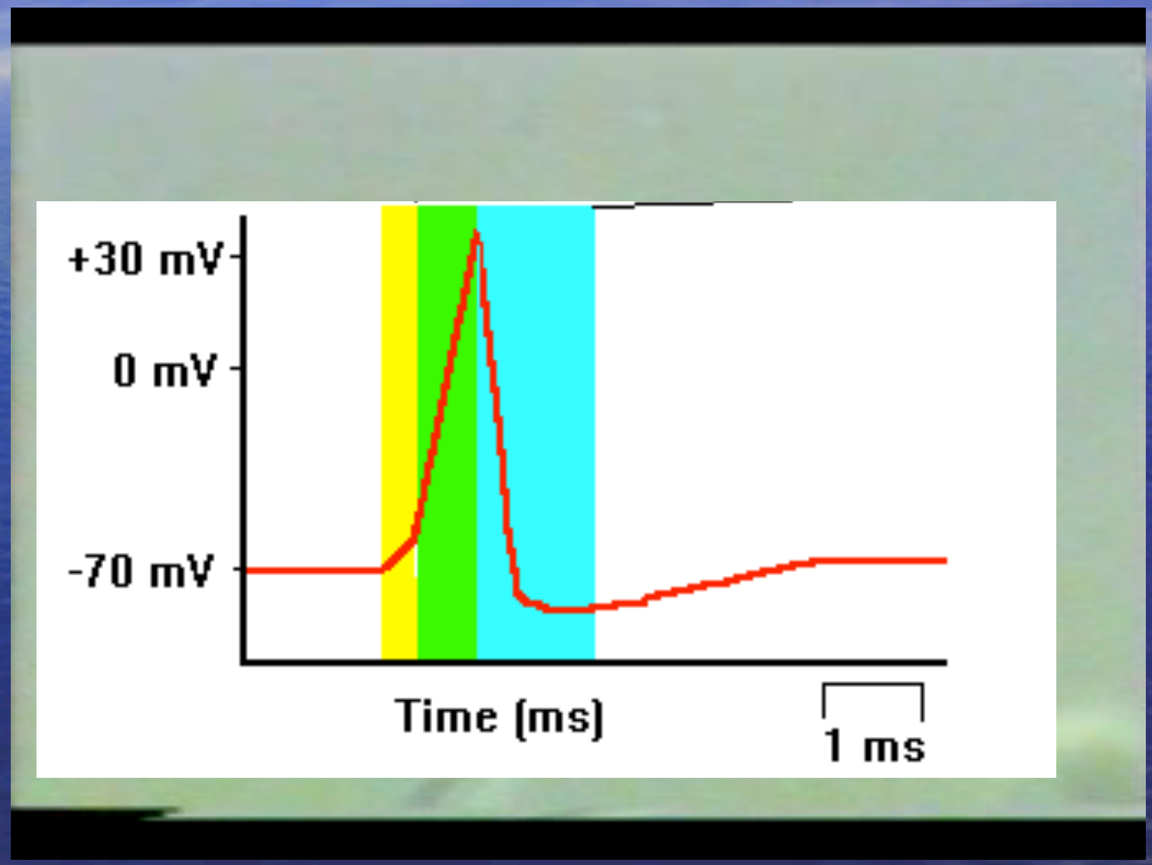

A
2,000 Hz
1,500 Hz
3,000 Hz
400 Hz
600 Hz
cochlear duct
apex
200 Hz
base
800 Hz
1,000 Hz
4,000 Hz
20,000 Hz
7,000 Hz
basilar membrane
5,000 Hz
©1997 Encyclopaedia Britannica, Inc.
B
basilar membrane
base
apex
high-frequency waves (1,500–20,000 Hz)
C
basilar membrane
base
apex
medium-frequency waves (600–1,500 Hz)
D
basilar membrane
base
apex
low-frequency waves (200–600 Hz)

The Corti organ

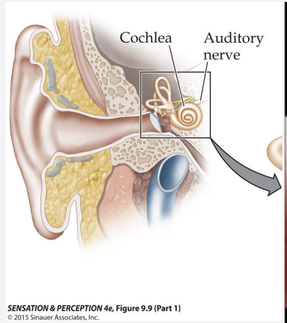


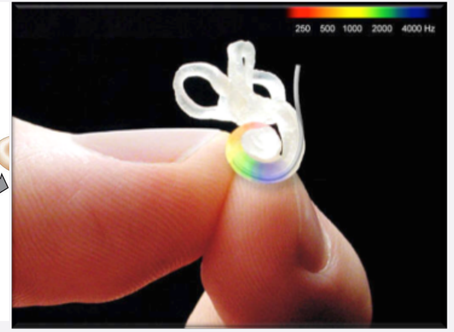
The acoustic sensing cells

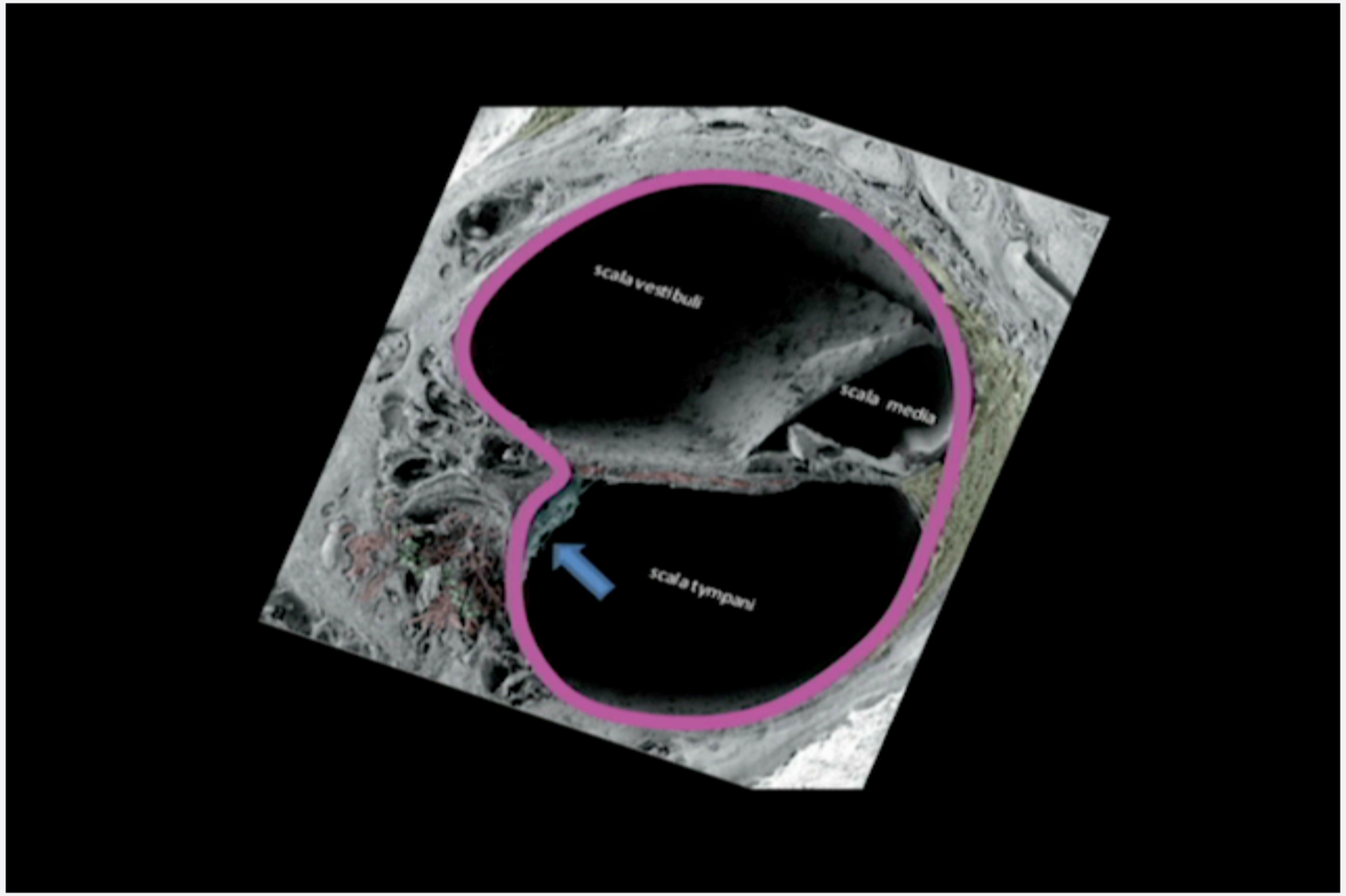




The human cochlea





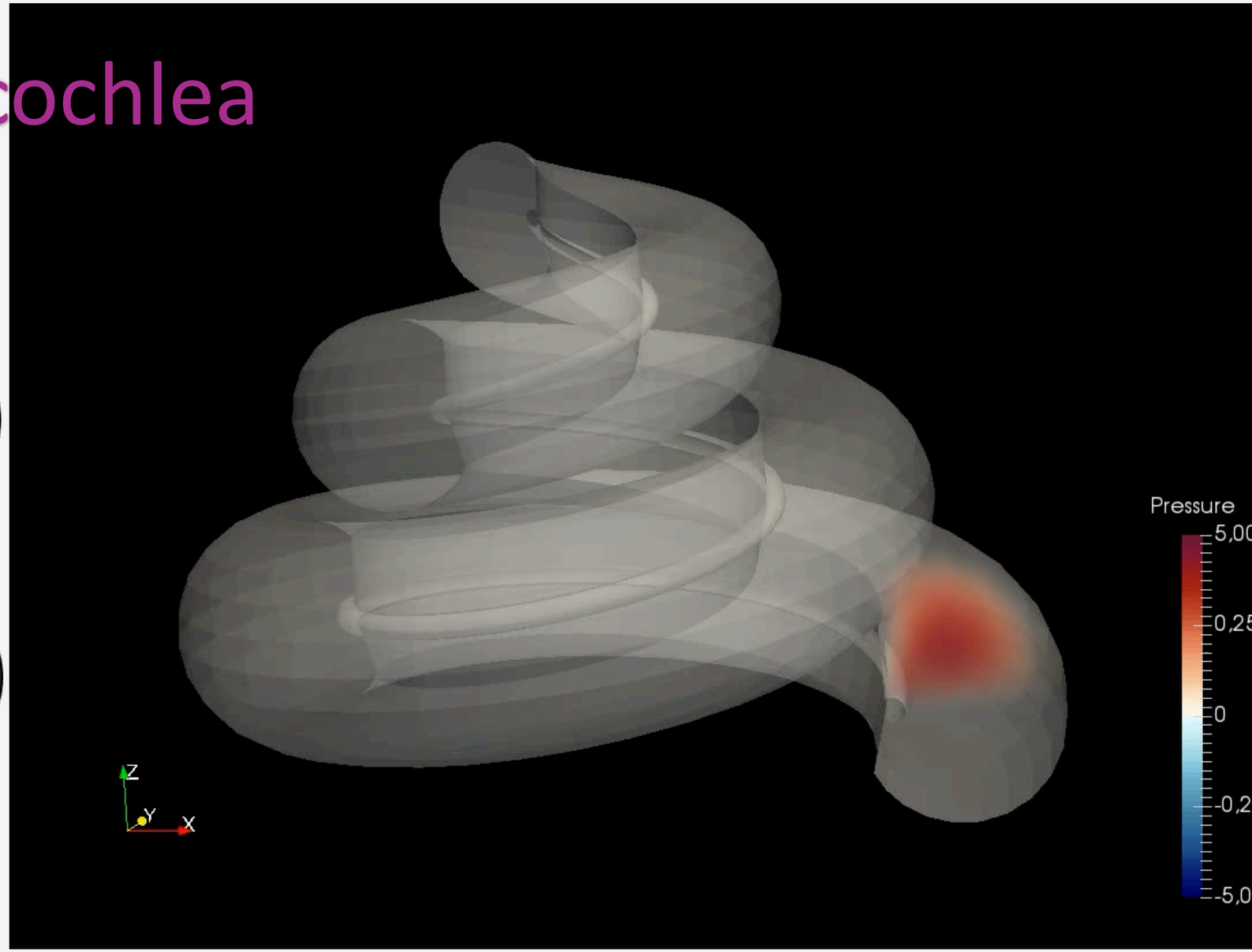


The human cochlea

$$x := \cos(\theta)\left(R - a\theta + r\cos(\phi)(1-\cos(\phi+\pi))\cos\left(\frac{\theta}{T}\right)\right)$$

$$y := \sin(\theta)\left(R - a\theta + r\cos(\phi)(1-\cos(\phi+\pi))\cos\left(\frac{\theta}{T}\right)\right)$$

$$z := r\sin(\phi)\left(1-\cos(\phi+\pi)\right)\cos\left(\frac{\theta}{T}\right) + A\theta$$



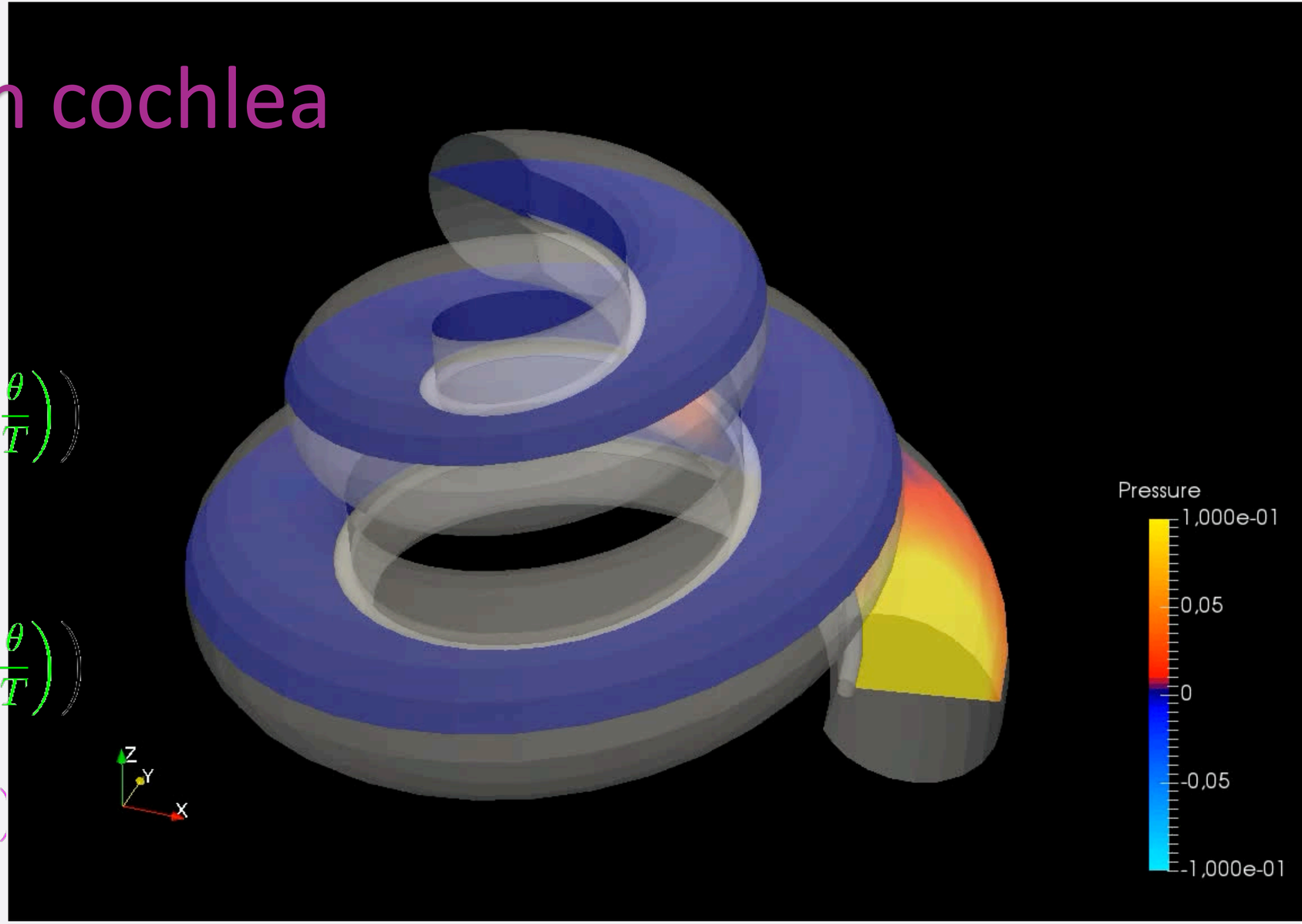
$\lambda = 523.3\text{mm}$ $\lambda = 130.8\text{mm}$

The human cochlea

$$x := \cos(\theta)\left(R - a\theta + r\cos(\phi)(1-\cos(\phi+\pi))\cos\left(\frac{\theta}{T}\right)\right)$$

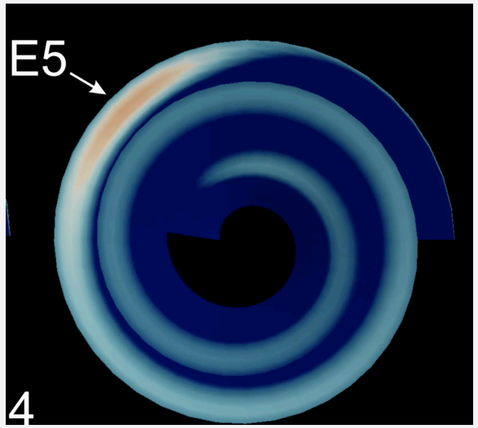
$$y := \sin(\theta)\left(R - a\theta + r\cos(\phi)(1-\cos(\phi+\pi))\cos\left(\frac{\theta}{T}\right)\right)$$

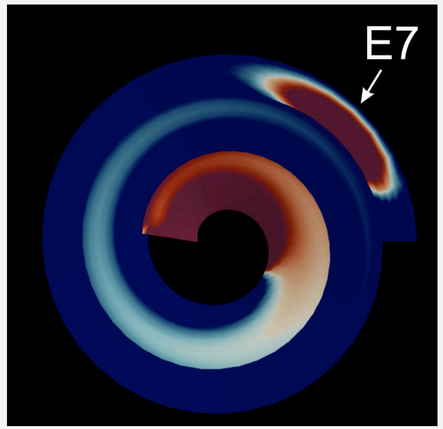
$$z := r\sin(\phi)\left(1-\cos(\phi+\pi)\right)\cos\left(\frac{\theta}{T}\right) + A\theta$$

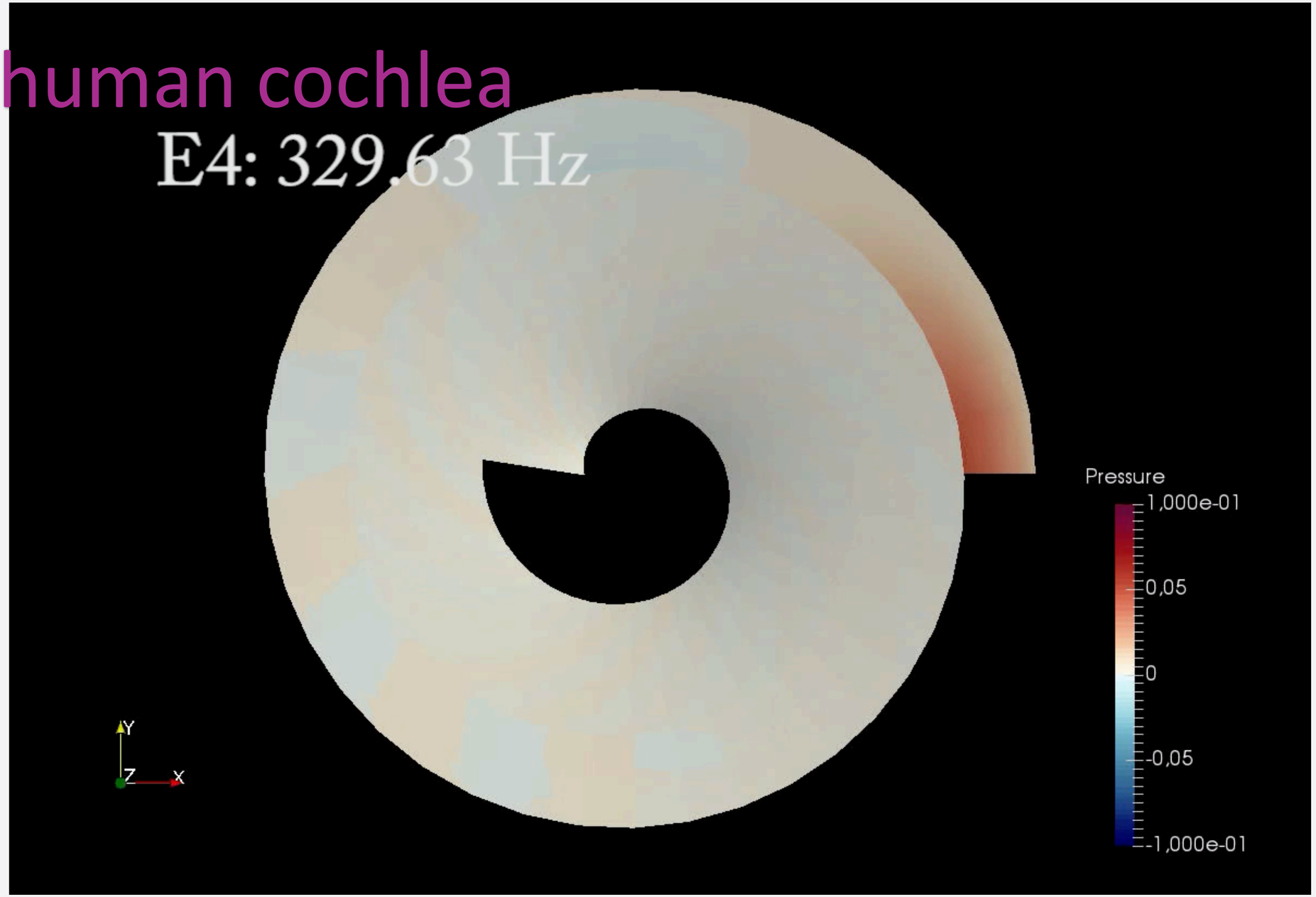


$\lambda = 523.3\text{mm}$ $\lambda = 130.8\text{mm}$

The human cochlea







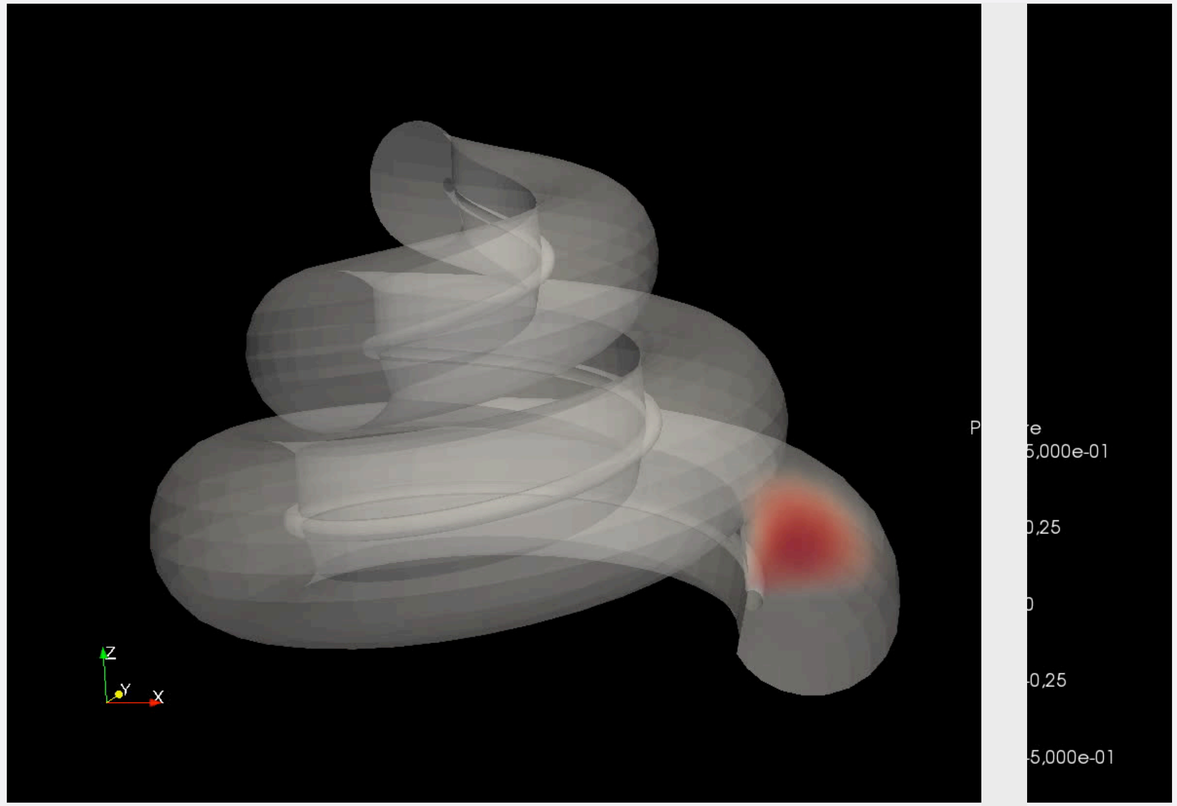
$\lambda = 523.3\text{mm}$ $\lambda = 130.8\text{mm}$

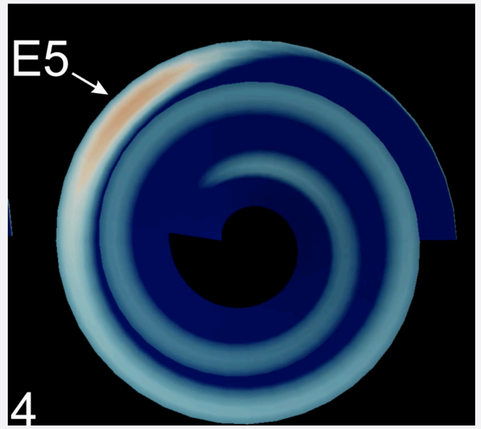
The Cochlea

$$x := \cos(\theta)\left(R - a\theta + r\cos(\phi)(1-\cos(\phi+\pi))\cos\left(\frac{\theta}{T}\right)\right)$$

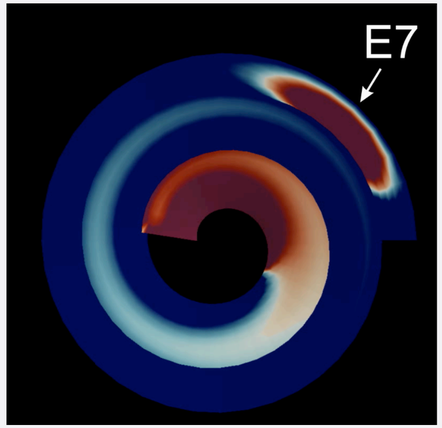
$$y := \sin(\theta)\left(R - a\theta + r\cos(\phi)(1-\cos(\phi+\pi))\cos\left(\frac{\theta}{T}\right)\right)$$

$$z := r\sin(\phi)\left(1-\cos(\phi+\pi)\right)\cos\left(\frac{\theta}{T}\right) + A\theta$$





$\lambda = 523.3\text{mm}$



$\lambda = 130.8\text{mm}$

Let's go for waves!

The wave equation on generalized coordinates

$$\frac{\partial^2 P}{\partial t^2} - \frac{c^2}{\sqrt{g}} \frac{\partial}{\partial x^\alpha} \left(\sqrt{g} g^{\alpha\beta} \frac{\partial P}{\partial x^\beta} \right) = 0,$$

Ali Mauricio Velasco

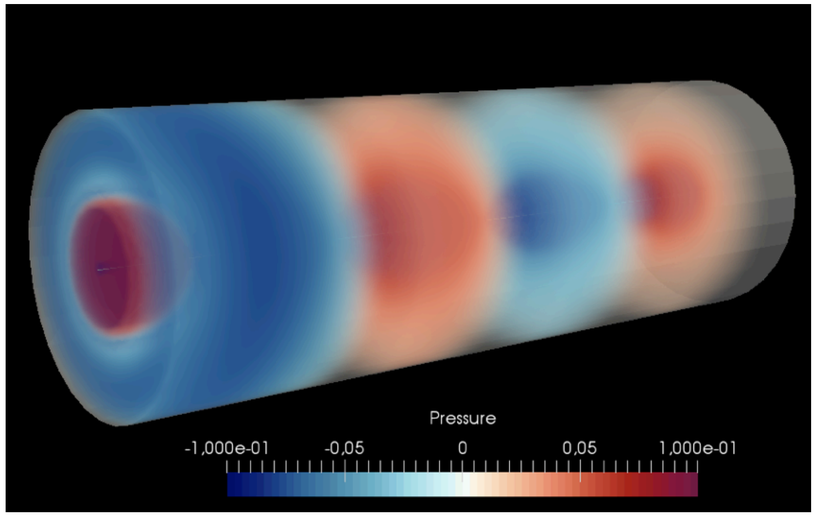
Can be reproduced with this model

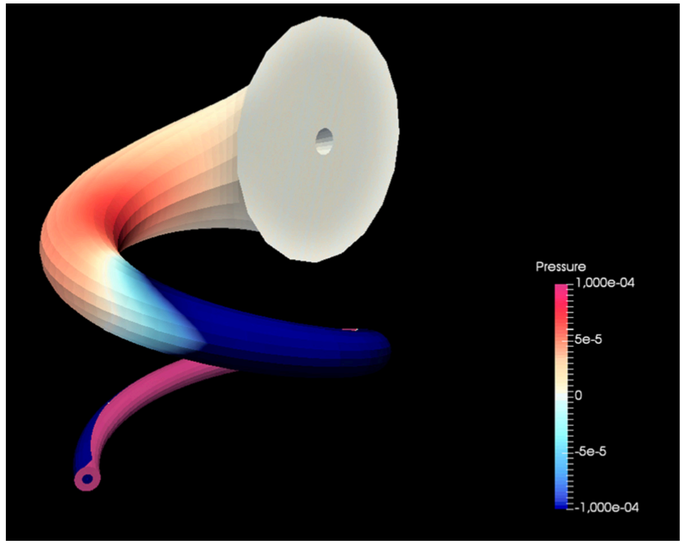
$$f^{eq} = \begin{cases} w_0 \sqrt{g} P & \text{for } i = 0 \\ w_i \sqrt{g} \left[P + \frac{\xi_i^k J'^k}{c_s^2} \right] & \text{otherwise} \end{cases}$$

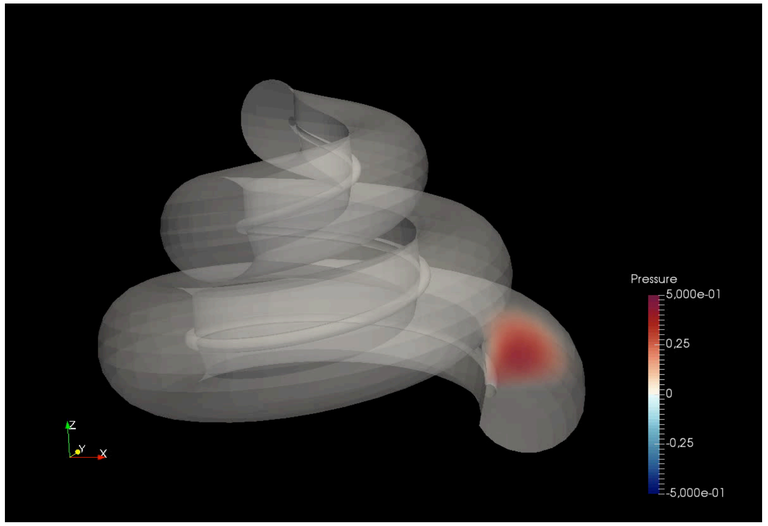
Forcement

$$\sqrt{g} P = \sum_l f_l^{eq}$$

$$\sqrt{g} J'^i = \sum_l f_l^{eq} \xi_l^i - \frac{1}{2} c^2 P \Gamma^i_{jk} g^{jk} + \frac{1}{2} \partial_j \left[\sqrt{g} c^2 P \left(g^{ij} - \delta^{ij} \right) \right]$$







Let's go for waves!

The wave equation on generalized coordinates

$$\frac{\partial^2 P}{\partial t^2} - \frac{c^2}{\sqrt{g}} \frac{\partial}{\partial x^\alpha} \left(\sqrt{g} g^{\alpha\beta} \frac{\partial P}{\partial x^\beta} \right) = 0,$$

Ali Mauricio Velasco

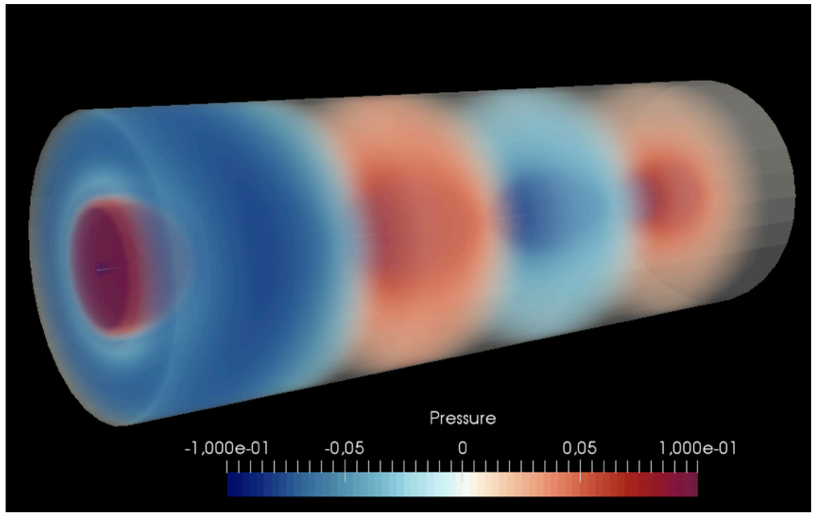
Can be reproduced with this model

$$f^{eq} = \begin{cases} w_0 \sqrt{g} P & \text{for } i = 0 \\ w_i \sqrt{g} \left[P + \frac{\xi_i^k J'^k}{c_s^2} \right] & \text{otherwise} \end{cases}$$

Forcement

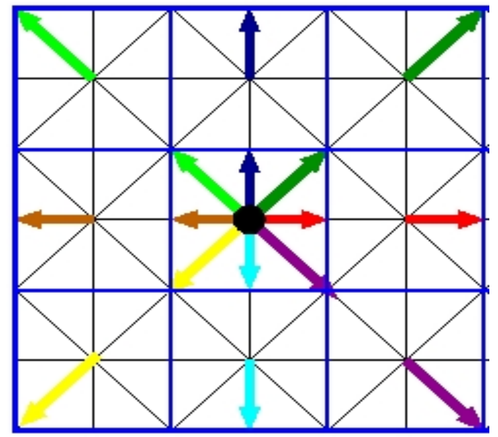
$$\sqrt{g} P = \sum_l f_l^{eq}$$

$$\sqrt{g} J'^i = \sum_l f_l^{eq} \xi_l^i - \frac{1}{2} c^2 P \Gamma^i_{jk} g^{jk} + \frac{1}{2} \partial_j \left[\sqrt{g} c^2 P \left(g^{ij} - \delta^{ij} \right) \right]$$



Spatial partial derivatives

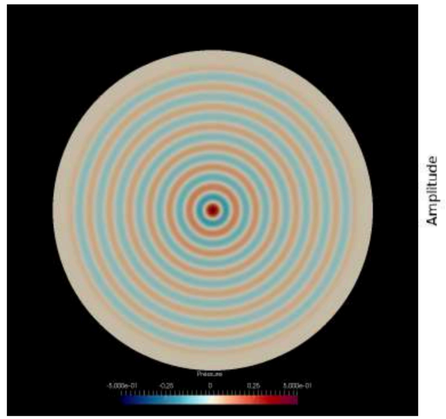
$$\partial_k \phi(\vec{x}) = \frac{1}{c_s^2 \Delta t} \sum_\lambda^m \omega_\lambda c_\lambda^k \phi(\vec{x} + c_\lambda \Delta t)$$

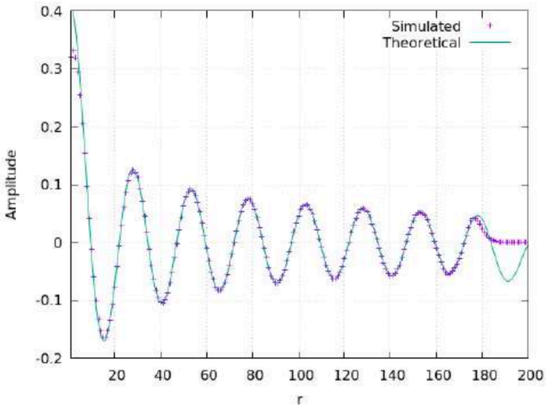


S. P. Thampi et al.. *Isotropic discrete laplacian operators from lattice hydrodynamics*. Journal of Computational Physics, 234(1):1–7, 2013.

Example:
2D Wave on polar coordinates

LB Waves D2Q5 on Curvilinear Coordinates.cpp





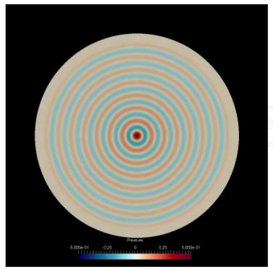
0 Actually 1D !! (Axial simmetry)

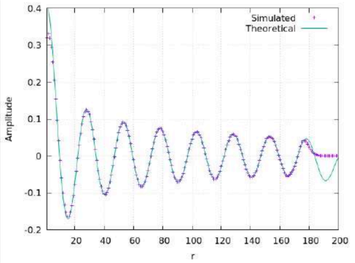
```
const int Lr=200;
const int Ltheta=1;
```

1 A new class for the geometry

```
//--------------------- class Geometry ------------
class Geometry{
private:
  double r_min,dr,theta_min,dtheta;
public:
  //i,j=0 is r; i,j=1 is theta
  Geometry(void);
  double r(int ir){return r_min+dr*ir;};
  double theta(int itheta){return theta_min+dtheta*itheta;};
  double gRoot(int ir,int itheta);
  double gTrace(int ir,int itheta);
  double MetricTensor_g(int i,int j,int ir,int itheta);
  double CristoffelSymbol_Gamma(int i,int j,int k,int ir,int itheta);
  double x(int ir,int itheta);
  double y(int ir,int itheta);
};
```

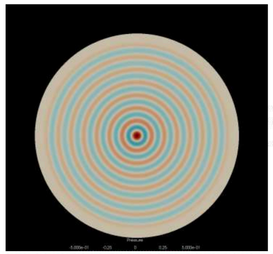
3 A new forcement term

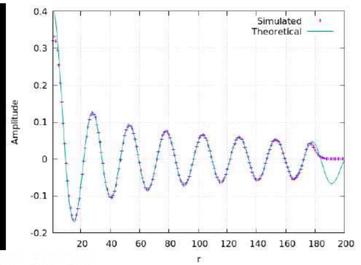





```
double LatticeBoltzmann::F(int i,int ir,int itheta,bool UseNew,Geometry {
  double sum1,sum2,rho0,deltaij,aux; int j,k,lambda,irnext,ithetanext;
  for(sum1=0,j=0;j<2;j++)
    for(k=0;k<2;k++)
      sum1+=Polar.CristoffelSymbol_Gamma(i,j,k,ir,itheta)*Polar.MetricTensor_g(j,k,ir,itheta);

  for(sum2=0,j=0;j<2;j++){
    if(j==i) deltaij=1; else deltaij=0;
    for(lambda=0;lambda<Q;lambda++){
      irnext=(ir+Vx[lambda]+Lr)%Lr; ithetanext=(itheta+Vy[lambda]+Ltheta)%Ltheta;
      aux=3*w[lambda]*rhog(irnext,ithetanext,UseNew)
          *(Polar.MetricTensor_g(i,j,irnext,ithetanext)-deltaij);
      if(j==0)
        sum2+=aux*Vx[lambda];
      else if(j==1)
        sum2+=aux*Vy[lambda];
    }
  }
  rho0=rho(ir,itheta,UseNew,Polar);
  return 0.5*C2*(rho0*sum1-sum2);
}
```

4 A different definition for the macroscopic fields





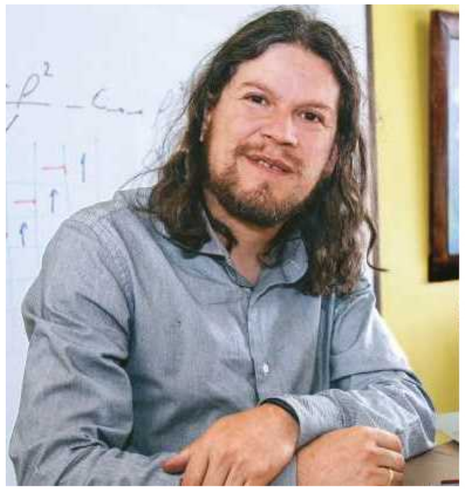

```
double LatticeBoltzmann::Jxg(int ir,int itheta,bool UseNew,Geometry & Polar){
  double sum,rho0; int i,n0;
  for(sum=0,i=0;i<Q;i++){
    n0=n(ir,itheta,i);
    if(UseNew) sum+=Vx[i]*fnew[n0]; else sum+=Vx[i]*f[n0];
  }
  rho0=rho(ir,itheta,UseNew,Polar);
  return sum-0.5*F(0,ir,itheta,UseNew,Polar);
}
```

5 And for the equilibrium function

```
double  LatticeBoltzmann::feq(double rhog0,double Jxg0,double Jyg0,int i,double gTrace0){
  if(i==0)
    return rhog0*w[0];
  else
    return w[i]*(rhog0+3*(Vx[i]*Jxg0+Vy[i]*Jyg0));
}
```

jdmunozc@gmail.com

¡Vielen Dank!



Prof. José-Daniel Muñoz

jdmunozc@unal.edu.co

Website

HI ERN
Helmholtz Institute
Erlangen-Nürnberg

UNIVERSIDAD
NACIONAL
DE COLOMBIA