Projeto Física Computacional

Instituto Superior Técnico
Grupo B09

Pedro Pereira 78889
João Alves 79006

Prof. Fernando Barão

Simulador dum detetor de Luz de Cherenkov

Objetivos

- Simular ruído dum detetor real.
- Simular rasto deixado no detetor por partículas.
- Reconstruir evento original a partir da informação do detetor.
- Analisar Eventos (Desenhos, Histogramas).

Filosofia

Independência entre:

- Simulação e Reconstrução
- Análise

Solução:

- Registo de dados num ficheiro e posterior leitura e interpretação.

Simulação I

Problema: Criar a matriz de pixeis (Detetor)

Solução:

- Uso de TF2 do ROOT;
- Uso dos seus métodos:
- GetBinXYZ;
- GetBin
- SetBin

Vantagens:

- Simplifica todos os processos de identificação de pixeis e criação da matriz de pixeis (detetor).

Desvantagens:

- É preciso traduzir a informação criada e interpretada pelo computador para algo inteligível por um comum mortal.

Ex: Criação de arrays H(uman)”Array” para mostrar ao utilizador, sendo que internamente só são usados os “Array”.

Simulação IIA

Sinal Aleatório

Problema:

- A partícula ao passar num ponto (x,y) , ativa 5 pontos aleatórios duma matriz de pixeis 3x3 à volta do pixel que contém (x,y).

Solução:

- Assumir que dos 9 pixeis, 5 são selecionados aleatoriamente, que corresponde a escolher uma de $\binom{9}{5}$ combinações.

Sinal Ruído

Problema:

- Percentagem de ruído: 0.5%

Solução:

- Varrimento de todos os pixeis, gerando um aleatório de 0 a 100 em cada passo.
- Aleatório <= 0.5 → Pixel Atual é Ruidoso.

Simulação IIB

Sinal Circular

Problema:

- Ativar N0 pixeis (aleatoriamente) numa circunferência de raio proporcional à velocidade da partícula.
- Restrição: Pixel é ativável se contém mais do que 1.5 mm de troço da circunferência.

Solução:

- Varrimento de 0 a 2π com passo $\frac{1.5}{10R}$.
- Em cada iteração gravava-se o número do pixel. Se o identificador se repete 10 vezes ou mais, o pixel é ativável.
- Após copiar esses identificadores para outro array, v, faz-se N0 iterações, gerando um aleatório, a, de 0 ao (size-1) do vetor, gravando noutro vetor o valor v[a]. A partir da primeira iteração faz-se um teste para verificar se o v[a] selecionado já está no array final, se estiver, escolhe-se outro.

Simulação III

- A partir dos data members, o método MakeEvent escreve toda a informação dos eventos num ficheiro, da forma que se mostra:

```
void Event::MakeEvent(float* l,float bb, double probnoise, Event A) {
  //contador do numero do evento
  static int counter=0;
  counter+=1;
  ofstream out;
 //faz as contas e imprime no ficheiro  as variaveis.
  out.open("data.txt", std::ios_base::app);
  out << counter << endl;
```

```
  for (size_t i=0 ; i<PID.size() ; ++i) {
    out << PID[i] <<" ";
  }
  out << endl;
```

Reconstrução I

Problema:
Obter N0, R e β, com um erro aceitável.

Solução:

- Obtém-se N0 com um determinado erro (observa-se o número de pixeis com sinal a uma determinada distância do aglomerado central). Soma-se 0.5 a esse valor e assume-se um erro de +/- 0.5.
- Desse valor obtém-se β e R, com um erro dado pela derivada parcial em ordem a N0 e a β, respetivamente, multiplicado pelo erro dessa variável.
- Seguidamente define-se um retângulo máximo onde possa estar o ponto (x,y) em que a partícula embateu.

Reconstrução II

Problema:

- Determinar o ponto de embate da partícula

Solução:

- Escolheu-se um píxel da circunferência e determinou-se o seu ponto central (ppx,ppy). Fez-se um varrimento do valor de x para um y fixo do retângulo referido anteriormente, registando os valores de xR e yR que minimizam a equação da circunferência. Repete para outro valor de y. (Ver imagem).
- O valor reconstruído é o último valor registado. O erro é igual nas duas coordenadas, pois depende do erro do raio e da seleção do ponto central do pixel ativo na circunferência como ponto que pertence à circunferência real.
- No fim, envia para o ficheiro valores das variáveis reconstruídas + erro.

```
int ilim=maxx-minx;
int klim=maxy-miny;

//para um y fixo varre todos os valores de x com um passo de 0.1
ilor de xR e yR que minimizam a equaçao da circunferencia
while(k<=klim*10) {

  while( i<=ilim*10) {
    tmp=fabs((xx-ppx)*(xx-ppx)+(yy-ppy)*(yy-ppy)-RR*RR);
    if(tmp<result){
      result=tmp;

      xR=xx;
      yR=yy;  }
    xx+=0.1;
    ++i;
  }
  i=0;
  xx=minx;
  yy+=0.1;
  ++k;
}
```

Reconstrução III

Limitações

- Considerando um detetor LxL, tudo isto funciona bem quando a partícula cai dentro do retângulo definido por 0.15L a 0.9L em x e y. Quando cai fora, no caso da sua velocidade ser próxima da da luz, o seu Raio pode ser superior aos limites do detetor, e assim o número de pixeis ativados na circunferencia não corresponder a N0.
- Nesses casos considera-se o ponto de embate da partícula, o ponto central do retângulo máximo.

  Maior erro, contudo não há um “gap” nos resultados, caso simulemos partículas a cair em todo o detetor.

Análise IA

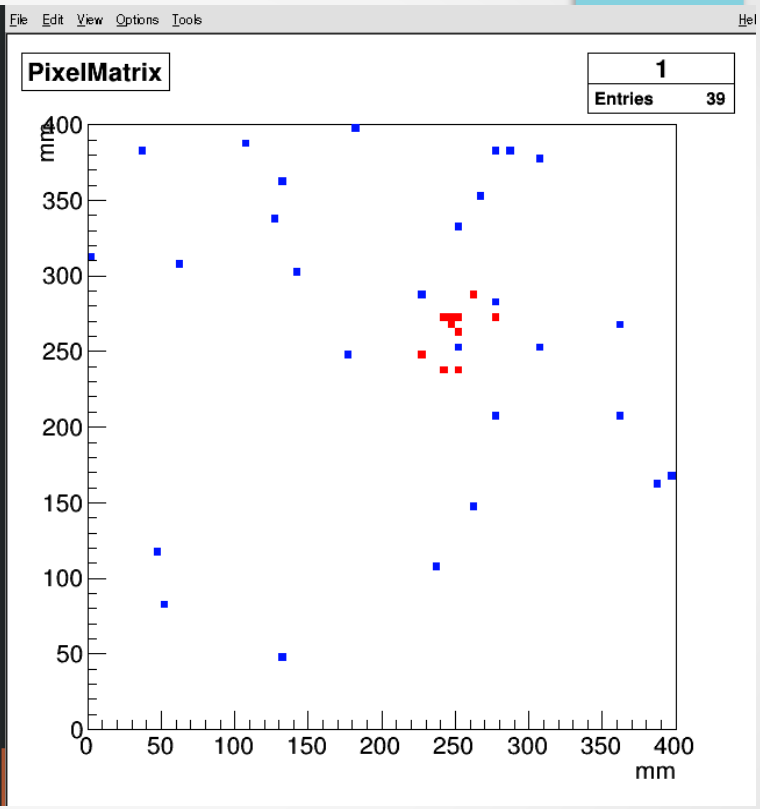
- O método InfoDraw desenha o evento n, indo ao ficheiro buscar as suas informações. Imprime no terminal os valores reconstruídos desse evento, detalhadamente.

```
void Event::InfoDraw(int n) {

   //cleanup
  Pixmat->Reset();
  Noisy.clear();
  HNoisy.clear();
  PID.clear();
  HPID.clear();
  PringID.clear();
  HPringID.clear();
  PringL.clear();
  AziAng.clear();

 //acontecimento 1 -> linha 0
  //acontecimento 2 -> linha 24
  //...
  ifstream in("data.txt");
  double startline=n+23*(n-1);
```

```
//leitor do ficheiro; le linha a linha as variaveis.
//de 24 em 24 linhas troca de acontecimento.
string line;
int k=0;
double tmp=0;
while(getline(in,line)){
  ++k;
  if (k==(startline))
    in >> beta;
  if (k==(startline+1))
    in >> R;
  if (k==(startline+2))
    in >> x;
  if (k==(startline+3))
    in >> y;
  if (k==(startline+4)) {
    char buffer [100000];
    in.get(buffer, 100000,'\n');
    int charcount=0;
    for(int m=0; buffer[m]; ++m) {
      if(buffer[m]==' ') {
        ++charcount;  }
    }
    stringstream s;
    s << buffer;
    for(int i=0;i<charcount; ++i){
      s >> tmp;
      Noisy.push_back(tmp);
    }
  }
```

Análise IB

```
Pixeis c Sinal
4368
4369
4370
4210
4289
ring 4612
ring 4375
ring 3810
ring 3965
ring 3808
ring 4612
Troços (por ordem)
1.65
5.25
4.95
4.65
5.1
1.65
Angulo Azimutal
1.06156
0.344872
4.90364
3.92291
4.54261
1.06156
***RECONSTRUCTION****
N0: 6
v = (0.936278+/-0.0173385)c
Raio: 27.7555+/-1.21655 mm
Intervalo de x possivel [240,250]
Intervalo de y possivel [260,270]
Ponto da Circunferencia escolhido: (262.5,287.5)
Posiçao (x,y): (243.9,266.9) +/- (3.36023,3.36023)
***REAL****
v = 0.937166c
Raio: 27.8364mm
Posiçao (x,y): (247.435,265.419)
Tempo a Reconstruir: 0.118 ms
```



- Qualquer evento pode ser analisado neste nível de detalhe.

Análise II

Histogramas são feitos lendo o ficheiro evento a evento.

```
D.CreatePad("Pad2");
ifstream in("data.txt");
string line;
int k=1;
TH1F * p = new TH1F("Diferenca Beta","Dif Beta",100,0,0.1);
p->GetYaxis()->SetTitle("#");
gStyle->SetOptStat("ne");
gStyle->SetPalette(1);
int n=1;
double startline=1;
double d=0;
while(getline(in,line)){
  if (k==(startline)) {
    in >> beta;
  }
  if (k==(startline+13)) {
    in >> betaR;
  }
  if(((k)%24==0)  ){
    d=fabs(beta-betaR);
    ++n;
    startline=n+23*(n-1);
    p->Fill(d);
  }
  ++k;
}

D.AddObject(p,"Pad2");
D.DumpPad("Pad2");
D.DrawPad("Pad2");
D.Print("HistBeta.pdf");
```

Vantagens do método

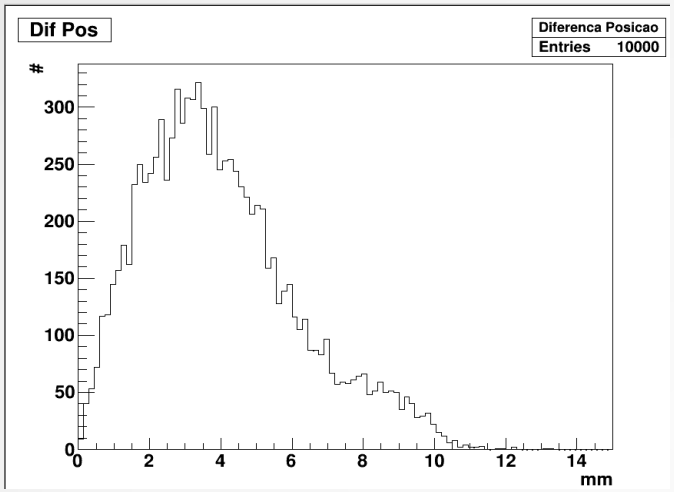
É fácil analisar em detalhe uma parte do total de dados.

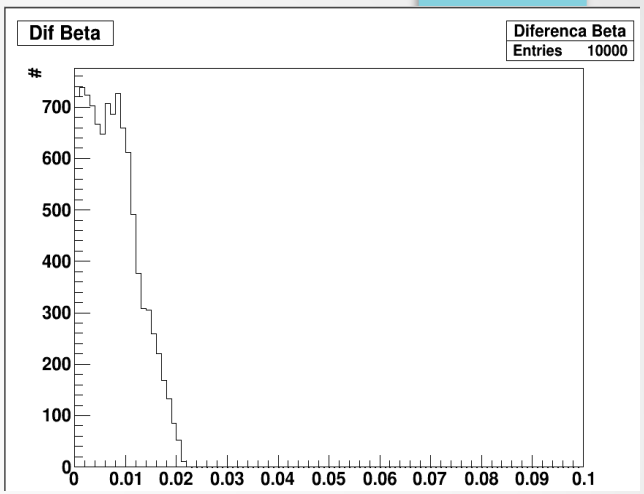
Ex: Desenhar todos os eventos com Erro na reconstrução do raio superior a 2 mm.

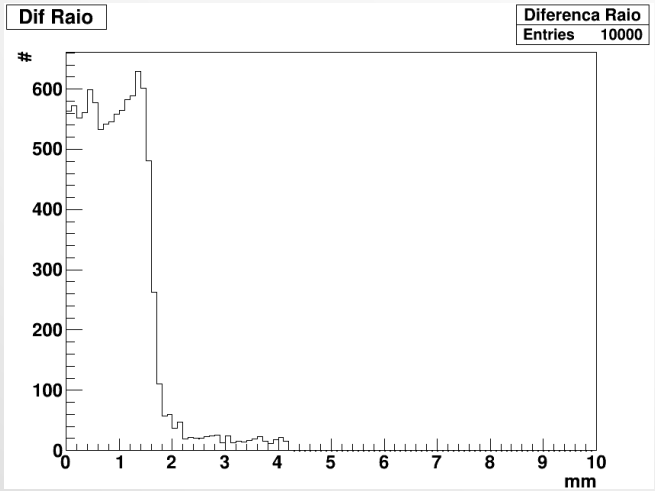
- No método HistR colocar um teste, se passar, guardar n num array de ints.
  No .C , dentro dum ciclo, InfoDraw(v[i]).
- Semelhante para os histogramas...

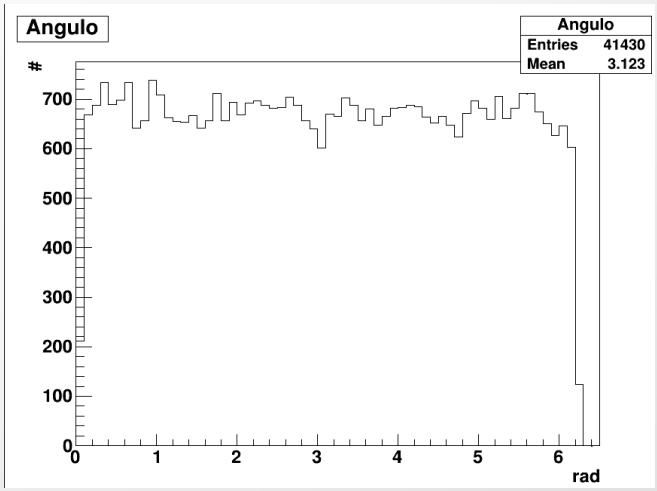
  Pode ser usado para analisar dados reais, basta que estejam num ficheiro com a mesma estrutura do criado pelo simulador.

Resultados A

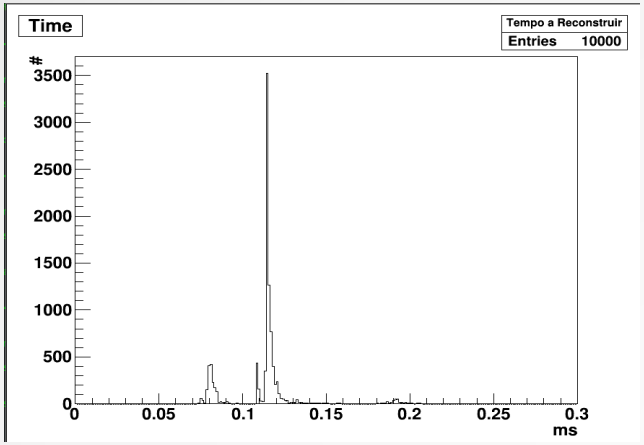


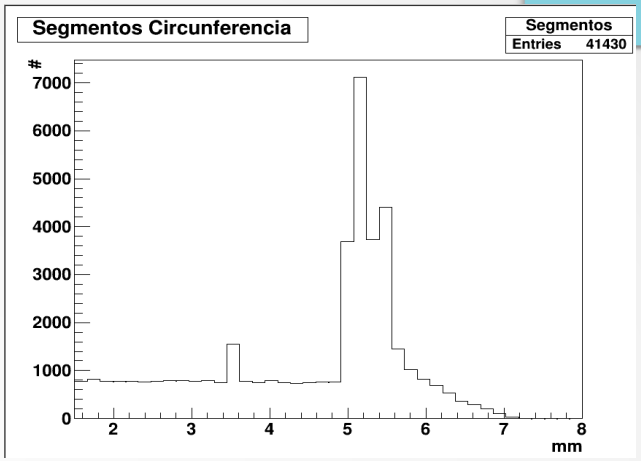


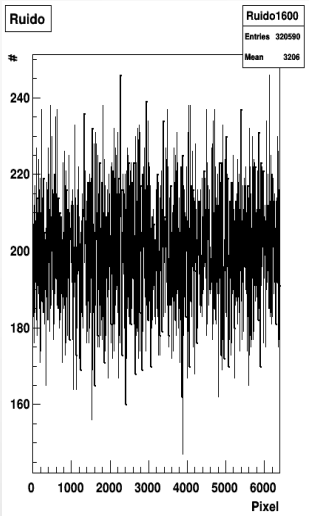


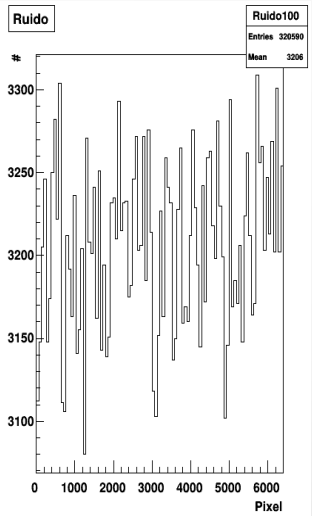


Resultados B









Updates

Má prática identificada:

- “God Object”

Solução:

```
EventAnalyzer:: EventAnalyzer(int a, int b) : EventGenerator(a,b) { }
```

```
EventAnalyzer A(6400,5);

A.CleanData();
//posiçao particula teste
float* p=new float [2];
TRandom3 rand(0);
for (int i=0; i<10000; ++i) {
  EventGenerator E(6400,5);
  p[0]=rand.Uniform(100,300);
  p[1]=rand.Uniform(100,300);
  float beta=rand.Uniform(1/1.3+0.001,1);
  E.MakeEvent(p,beta,0.005,E);
  E.RecEvent();
}
A.InfoDraw(1);
A.Hist();
```

Trajetória dum Protão nas Imediações da Terra

Objetivos:

- Estudar protão sobre a influência do campo magnético terrestre;
- Prever a trajetória do protão;
- Estabelecer classe com a trajetória;
- Descrever as linhas de campo magnético terrestre.

Descrição do Problema IA

- O campo magnético Terrestre é dado por:

$$\vec{B} = B_0 \left(\frac{Re}{r}\right)^3 \left(3 < \frac{\vec{d}}{d}, \frac{\vec{r}}{r} > \frac{\vec{r}}{r} - \frac{\vec{d}}{d}\right)$$

Em coordenadas cartesianas,

- $\vec{B} = B_0 \left(\frac{Re}{\sqrt{x^2+y^2+z^2}}\right)^3 \left(3z\frac{(x,y,z)}{r^2} - (0,0,1)\right)$

Uma partícula num campo magnético sofre a seguinte força:

- $\vec{F} = q\vec{v} \times \vec{B}$ (Força de Lorentz)

Descrição do Problema IB

Aplicando a 2ª Lei de Newton ao sistema,

$$\begin{cases} \ddot{x} = \dfrac{qB_0{R_e}^3(3xz\dot{z} - 3\dot{x}z^2 - \dot{x}(x^2+y^2+z^2)^2)}{m(x^2+y^2+z^2)^{\frac{5}{2}}\sqrt{1-\dfrac{\dot{x}^2+\dot{y}^2+\dot{z}^2}{c^2}}} \\ \ddot{y} = \dfrac{qB_0{R_e}^3(3z^2\dot{y} - 3yz\dot{z} - \dot{y}(x^2+y^2+z^2)^2)}{m(x^2+y^2+z^2)^{\frac{5}{2}}\sqrt{1-\dfrac{\dot{x}^2+\dot{y}^2+\dot{z}^2}{c^2}}} \\ \ddot{z} = \dfrac{qB_0{R_e}^3(3\dot{x}yz - 3x\dot{y}z)}{m(x^2+y^2+z^2)^{\frac{5}{2}}\sqrt{1-\dfrac{\dot{x}^2+\dot{y}^2+\dot{z}^2}{c^2}}} \end{cases}$$

Condições iniciais da(s) Partícula(s)

Obteve-se a velocidade da partícula recorrendo a:

- $\vec{p} = \gamma m \vec{v} \leftrightarrow \gamma \vec{v} = \frac{\vec{p}}{m} \leftrightarrow \vec{v}\sqrt{1-\beta^2} = \frac{\vec{p}}{m}, \qquad \beta = \frac{v}{c}$

Esta foi convertida para coordenadas cartesianas, recorrendo a:

$$\begin{cases} v_x = \sin\theta\cos\varphi\,\dot{r} + \cos\theta\cos\varphi\,\dot{\theta} - \sin\varphi\,\dot{\varphi} \\ v_y = \sin\theta\sin\varphi\,\dot{r} + \cos\theta\sin\varphi\,\dot{\theta} + \cos\varphi\,\dot{\varphi} \\ v_z = \cos\theta\,\dot{r} - \sin\theta\,\dot{\theta} \end{cases}$$

Obtendo-se:

- v_x = -0.07c m/s, v_y = 0 m/s, v_z = -0.07c m/s (Partícula 1)
- v_x = 0 m/s, v_y = 0 m/s, v_z = -0.998c m/s (Partícula 2)

Declaração da Classe

Variáveis privadas

vector<TPolyLine3D*> vp; Guarda as trajetórias das partículas

TF1 *f; dvx/dt = f(x,y,z,vx,vy,vz)

TF1* g; dvy/dt = g(x,y,z,vx,vy,vz)

TF1* h; dvz/dt = h(x,y,z,vx,vy,vz)

MagField *(double Q=q, double M=mp, double B=b, double R=r)*

Construtor da classe. Inicializa f, g e h. Por *default* coloca-as de acordo com a situação pedida.

Declaração da Classe (cont)

static double* MagForce *(double x, double y, double z)* ->

Calcula a força exercida pelo campo magnético *B* num ponto (*x*,*y*,*z*). Por ser estática pode ser usada sem instanciar um objeto da classe

void MagLines *(double x, double y, double z, double h, int n)*

Recebe um ponto inicial (*x*,*y*,*z*), do qual descreve as linhas de campo com um passo de *h*, calculando *n* pontos.

void Draw *(string s="opt")*

Desenha as trajetórias/linhas de campo contidas no objeto. Possui diversas funcionalidades, selecionadas através da variável *s*.

Descrição da Classe (cont)

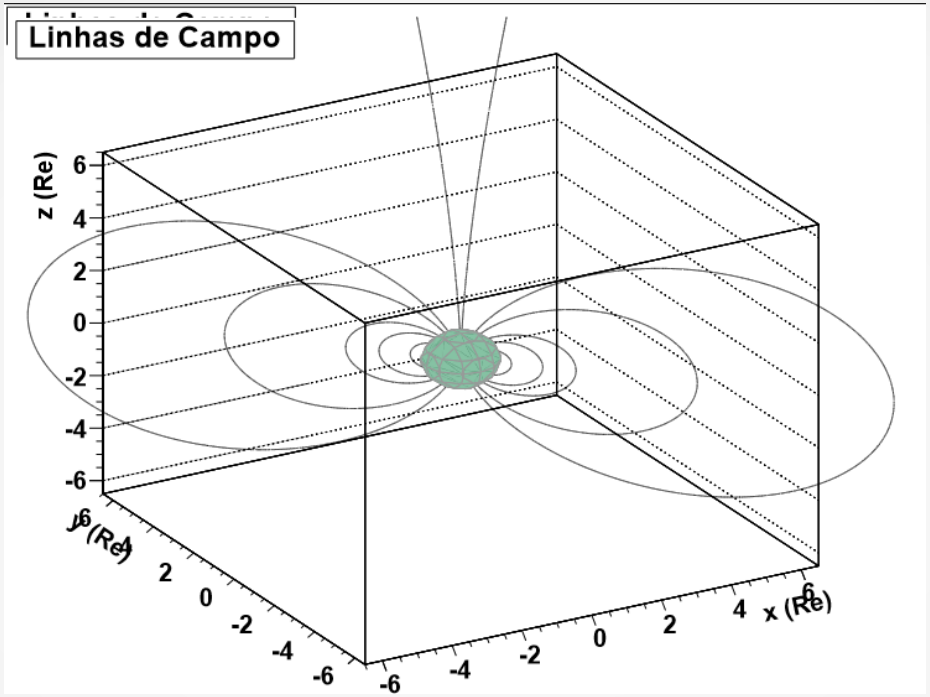
void PartTraj1 *(double x, double y, double z, double vx, double vy, double vz, double h, int n, char cc='m')*

Resolve o movimento de uma partícula de posição e velocidade inicial (x,y,z) e (vx, vy, vz), respetivamente. Recorre ao método RK2, com passo h, calculando n pontos. Se cc='r' verifica a condição r<25*Re

void PartTraj2 *(double x, double y, double z, double vx, double vy, double vz, double h, int n, char cc='m')*

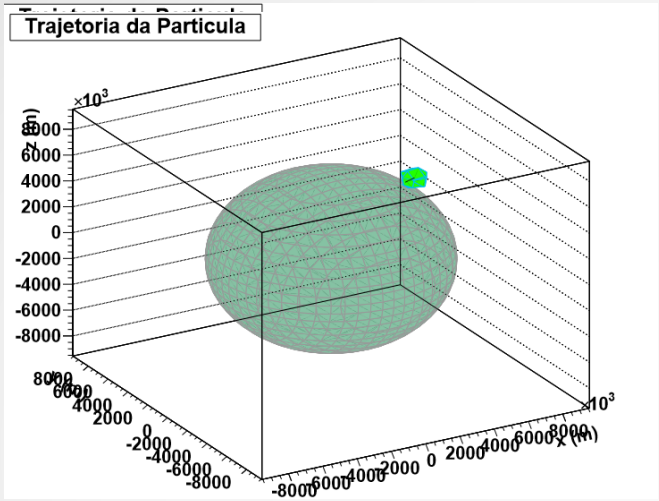
Resolve o movimento de uma partícula de posição e velocidade inicial (x,y,z) e (vx, vy, vz), respetivamente. Recorre ao método RK4, com passo h, calculando n pontos. Se cc='r' verifica a condição r<25*Re

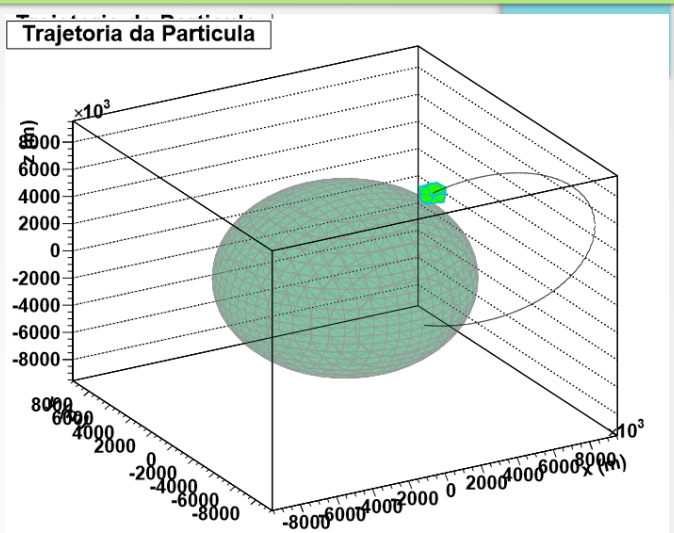
Linhas de Campo Magnético

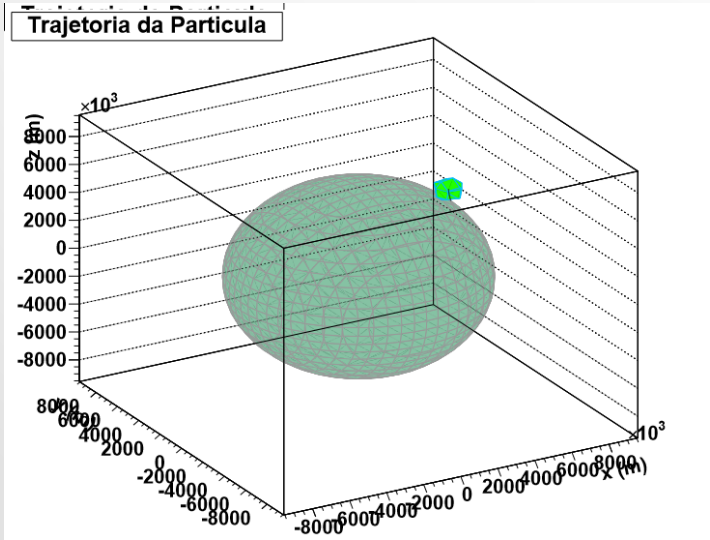


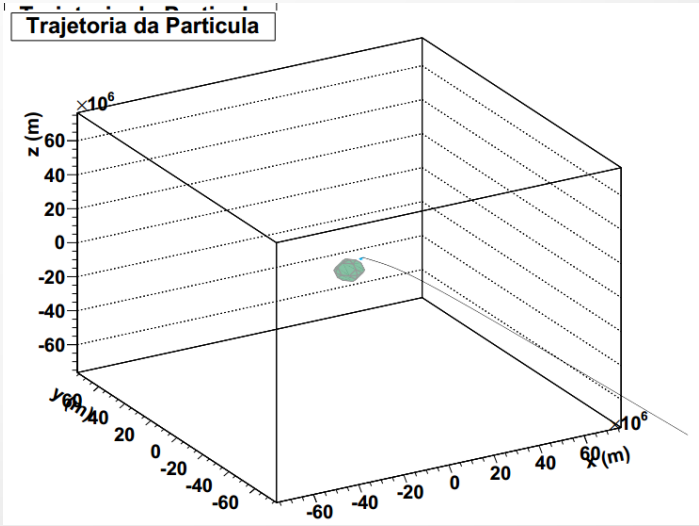

```
for(int i=0;i<n;++i){
   u=MagForce(x[i],y[i],z[i]);
   x[i+1]=x[i]+h*u[0]/sqrt(u[0]*u[0]+u[1]*u[1]+u[2]*u[2]);
   y[i+1]=y[i]+h*u[1]/sqrt(u[0]*u[0]+u[1]*u[1]+u[2]*u[2]);
   z[i+1]=z[i]+h*u[2]/sqrt(u[0]*u[0]+u[1]*u[1]+u[2]*u[2]);
}
```

Trajetórias das Partículas









The End


WRITES ALL THE DNA CODE FOR ALL SPECIES
DOESN'T WRITE A SINGLE COMMENT
quickmeme.com