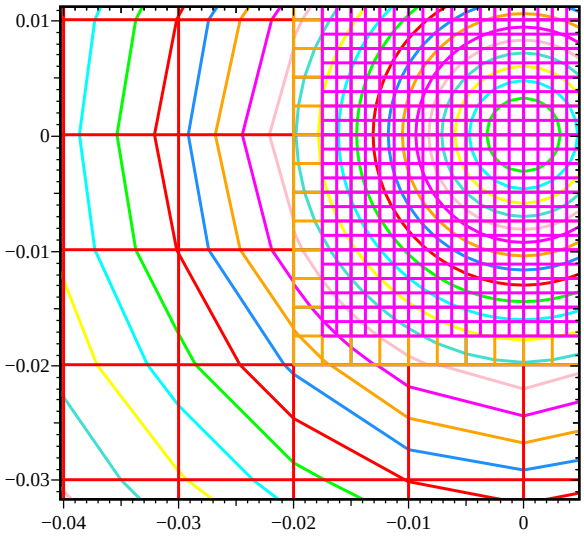
EUROTeV-Report-2007-071

Faktor2 : Rationale

Warner Bruns, CERN*



December 3, 2007

Abstract

This writeup describes some of the algorithms implemented in FAKTOR2 . The rationale why the algorithms are chosen, and some details of the implementation are given.

*This work is supported by the Commission of the European Communities under the Framework Programme "Structuring the European Research Area", contract number RIDS-011899

Contents

Faktor2 is an implementation of the *Particle in Cell* algorithm to simulate the buildup of electron clouds. Faktor2 can also compute the trajectories of ions trapped in the potential field of a beam.
The charges move in volume which is bounded by perfectly conducting material. The shape of the geometry is arbitrary. An arbitrary number of electrodes with prescribed potential may be present. The charges are accelerated by their self forces and by the forces of a beam field. A static magnetic field may be prescribed.
The self forces of the charges are computed via an electrostatic approximation. This restricts the applicability to configurations where the velocity of the charges are significantly less than the speed of light.

1 Discretisation

The underlying grid is a rectangular one. There is a hierarchy of grids. A grid may contain one or more subgrids. The gridspacing of a subgrid is half of the spacing of the parent grid. The cells are marked for subdividing, if they are near to a boundary, of if the beam passes through. The code which decides what cells are marked for subdivision is (Gridrefinement.f90):

```
ALLOCATE(marker (-1:nx,-1:ny))
DO iy= LBOUND(marker, DIM= 2), UBOUND(marker, DIM= 2), 1
   y= yMin + iy*dy + dy*0.5
   DO ix= LBOUND(marker, DIM= 1), UBOUND(marker, DIM= 1), 1
      x= xMin + ix*dx + dx*0.5
      CALL may_or_must_refine (x, y, depth, Must, May)
      IF (Must) THEN
         marker (ix,iy)= .TRUE.
      ELSE IF (May) THEN
         IF (RefineNearBeam) THEN
            CALL within_beam (x,y,dx,flag)
         ELSE
            flag= .FALSE.
         ENDIF
         marker (ix,iy)= flag
         IF ((.NOT. flag) .AND. at_the_border) THEN
            flag= potential_known (Material(ix,iy))
            marker (ix,iy)= &
                 (flag .NEQV. potential_known (Material (ix+1,iy  ))) &
            .OR. (flag .NEQV. potential_known (Material (ix  ,iy+1))) &
            .OR. (flag .NEQV. potential_known (Material (ix+1,iy+1)))
         ENDIF
      ELSE
         marker (ix,iy)= .FALSE.
```

```
        ENDIF
      ENDDO
    ENDDO
```

The SUBROUTINE may_or_must_refine decides whether a cell is within a region which is marked in the inputfile as a region where a refinement is allowed, or where a refinement is enfoced. The LOGICAL array potential_known holds the information whether the material is at a known potential, ie. is a boundary condition. The SUBROUTINE within_beam decides whether a position is near where the beam travels.
The so marked gridcells are clustered via an algorithm which was taken form the BearCLAW-Package (http://www.amath.unc.edu/Faculty/mitran/bearclaw.html).

1.1 Data structure for boundary detection

When a particle hits a metallic wall, secondary particles are created. The initial position and the vectorial velocity of a secondary particle depend on the position where the seeding charge hits the wall. That position should be detected as accurate as possible. The detection should also be very efficiently, ie. fast.
Each cell carries information, whether there is a material boundary in its vicinity. When a particle is within such a cell, at each timestep the crossing point of its trajectory with the material boundary is computed. If there is a crossing point, secondary particles are generated.
The information of the material boundaries is stored in a datastructure which on average needs less than 32 bits per cell. The datastructure of the grid has an INTEGER field `Grid% Borderindex (:,:)`. If the `Grid% Borderindex (ix,iy)` of a cell is nonzero, the cell contains a material boundary. The `Grid% Borderindex (ix,iy)` is the number of the border which describes the border. The border is decribed by a point on the border (one of the points of the straight segment), and the normal of the border, which is perpendicular to the connecting line between the endpoints of the border segment.

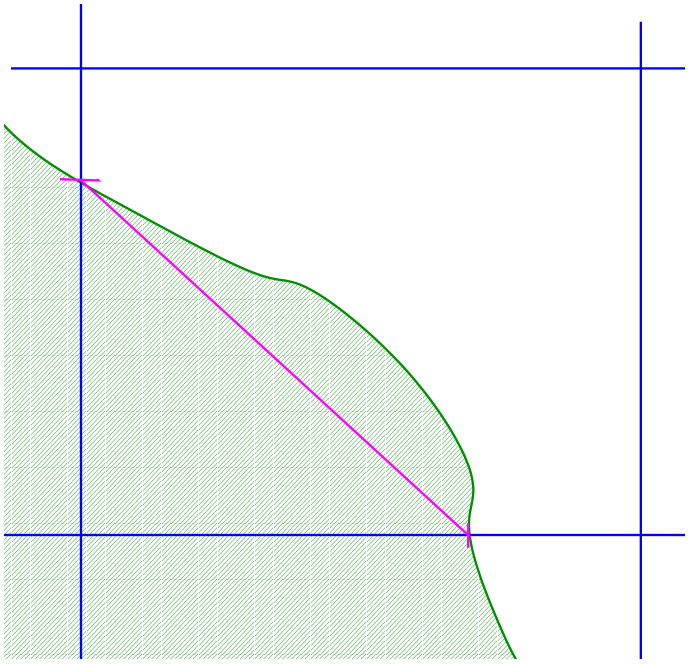


Figure 1: Encoding of material boundaries. The real material boundary is approximated by a straight line, connecting the crossing points of the real boundary with the edges of the grid.

Whether a particle is near a border is detected while the fieldstrengths at the particle's position are interpolated from the surrounding field components.
(`Gridrefinement.f90, SUBROUTINE Give_EH_and_Boundarydetection`). If it is near a boundary, a flag is set and the plane normal and the position of a point on the boundary

are copied to global accessible variables. These data for computing the distance to the nearest border is in (Current_Borderdescription.f90).

```
MODULE Current_Borderdescription
USE Kinds
PUBLIC

   LOGICAL,                          SAVE :: Near_Border= .FALSE.
   REAL(KIND= Real8), DIMENSION(3), SAVE :: Normal, Point_on_Border
   INTEGER,                          SAVE :: Number_Borderindex
END MODULE Current_Borderdescription
```

The flag Near_Border is TRUE if the particle is near a boundary. If it is, the signed distance between the particle's position before and after the timestep is computed. (incs/EnBorisAdaptiv.inc, lines 92 and 260)

```
 92    Distance_before= DOT_PRODUCT(Normal, Position - Point_on_Border)

260    Distance_now= DOT_PRODUCT(Normal, &
261                              ThisCloud% Position2 - Point_on_Border)
```

If the sign of these distances changes, the particle has crossed a material boundary and the subroutine for handling that case is called.
The encoding of the borders is done as part of the mesh generation. (Gridrefinement.f90, SUBROUTINE Compute_Borderindices.) A gridcell contains a border if at one of its points the potential is known and on a neighbour point the potential is not known. The exact point where the border crosses the connecting line between the points is computed by bi-section. The code to detect and compute a material boundary between a gridpoint and its neighbour in positive x-direction is

```
      IF (Potential_Known (Material (ix,iy)) &
    .NEQV. Potential_Known (Material (ix+1,iy))) THEN
         !# Materialboundary in x-direction
         x1= x0 + ix*dx
         x2= x0 + (ix+1)*dx
         y= y0 + iy*dy
         Status1= Potential_Known (Material (ix,iy))
         Status2= Potential_Known (Material (ix+1,iy))
         DO i= 1, 24, 1
            x= (x1+x2)/2
            CALL MaterialComputation (x,y,m)
            Status= Potential_Known (m)
            IF (Status .NEQV. Status1) THEN
               !# Boundary between x1 and x.
               !# Next interval is (x1,x2) = (x1,x).
               x2= x
```

```
            Status2= Status
         ELSE IF (Status .NEQV. Status2) THEN
            !# Boundary between x and x2.
            !# Next interval is (x1,x2) = (x,x2).
            x1= x
            Status1= Status
         ELSE
         ENDIF
      ENDDO
      Position (1,nCells)= (x - (x0 + ix*dx))/dx
   ENDIF
```

The gridcells which do not have a border in them are assigned the encoding of their parent's cells.

1.2 Finite difference coefficients

The field computation is solving POISSONs equation for the electrostatic potential and deriving the electric field by differencing that potential. The derivation of the finite difference equation for the potential values at the grid points follows the Finite Integration Theory. The integrated flux density shall be equal to the enclosed charge. For this, on the outer boundary of the integration only the normal component of the flux density needs to be known.
POISSONs equation for static fields follows from the MAXWELLs equation

$$\nabla \varepsilon \vec{E} = \varrho \tag{1}$$

when $\vec{E}$ is written as the gradient of a scalar potential.

$$\nabla \varepsilon (-\nabla \varphi) = \varrho \tag{2}$$

Only if ε is spatially constant, it can be put before the div-operator and one yields

$$\nabla \nabla \varphi = \Delta \varphi = -\frac{\varrho}{\varepsilon} \tag{3}$$

The *Finite Integration Theory* does not start from the differential forms of MAXWELLs equations, but from the integral forms. The integral form of the relevant equation reads:

$$\int\int \varepsilon \vec{E} \cdot d\vec{F} = \int\int\int \varrho \, dV \tag{4}$$

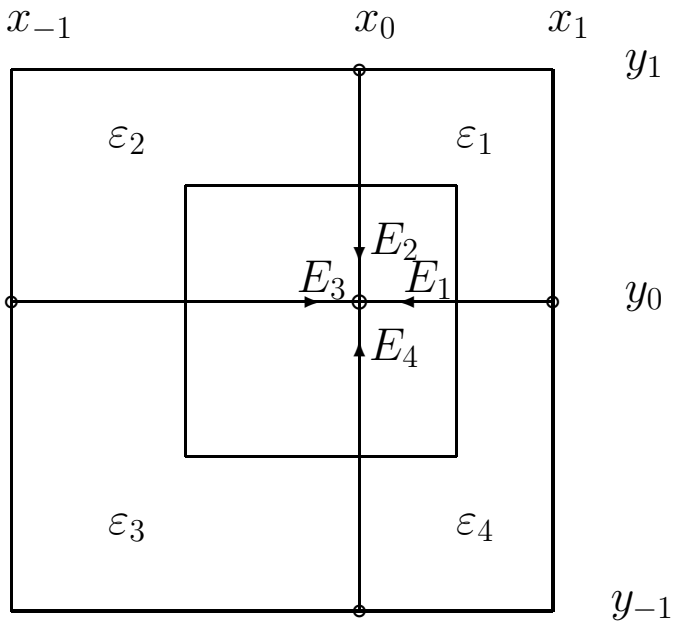


Figure 2: The flux integral of the electric displacement equals the contained charge. The surface over which the flux integral is performed encloses the set of points which is nearest to the central gridpoint. Only the normal component of $\vec{E}$ is needed to evaluate the flux integral.

In 2D, the integral $\int\int \varepsilon\vec{E}\cdot d\vec{F}$ can be evaluated as the sum of eight integrals (see scetch 2).

$$\begin{aligned}
&\int\int \varepsilon\vec{E}\cdot d\vec{F} \approx \\
&\quad -E_1\varepsilon_4\Delta z\frac{y_0-y_{-1}}{2} - E_1\varepsilon_1\Delta z\frac{y_1-y_0}{2} \\
&\quad -E_2\varepsilon_1\Delta z\frac{x_1-x_0}{2} - E_2\varepsilon_2\Delta z\frac{x_0-x_{-1}}{2} \\
&\quad -E_3\varepsilon_2\Delta z\frac{y_1-y_0}{2} - E_3\varepsilon_3\Delta z\frac{y_0-y_{-1}}{2} \\
&\quad -E_4\varepsilon_3\Delta z\frac{x_0-x_{-1}}{2} - E_4\varepsilon_4\Delta z\frac{x_1-x_0}{2} \approx \int\int\int \varrho\, dV = Q
\end{aligned}$$

The four normal components E_1, E_2, E_3, E_4 are given by difference quotients of potentials at gridpoints and grid spacings:

$$\begin{aligned}
E_1 &= \frac{U(x_1,y_0)-U(x_0,y_0)}{x_1-x_0}; \qquad E_2 = \frac{U(x_0,y_1)-U(x_0,y_0)}{y_1-y_0} \\
E_3 &= \frac{U(x_{-1},y_0)-U(x_0,y_0)}{x_0-x_{-1}}; \qquad E_4 = \frac{U(x_0,y_{-1})-U(x_0,y_0)}{y_0-y_{-1}}
\end{aligned}$$

The equation for the potential at the gridpoint (x_0, y_0) then reads:

$$
\begin{aligned}
-\frac{1}{2}\left\{ \begin{array}{l} \dfrac{\varepsilon_4(y_0-y_{-1})+\varepsilon_1(y_1-y_0)}{x_1-x_0} \\ +\dfrac{\varepsilon_1(x_1-x_0)+\varepsilon_2(x_0-x_{-1})}{y_1-y_0} \\ +\dfrac{\varepsilon_2(y_1-y_0)+\varepsilon_3(y_0-y_{-1})}{x_0-x_{-1}} \\ +\dfrac{\varepsilon_3(x_0-x_{-1})+\varepsilon_4(x_1-x_0)}{y_0-y_{-1}} \end{array} \right\} U(x_0,y_0) \\
+\frac{1}{2}\left\{ \frac{\varepsilon_4(y_0-y_{-1})+\varepsilon_1(y_1-y_0)}{x_1-x_0} \right\} U(x_1,y_0) \\
+\frac{1}{2}\left\{ \frac{\varepsilon_1(x_1-x_0)+\varepsilon_2(x_0-x_{-1})}{y_1-y_0} \right\} U(x_0,y_1) \\
+\frac{1}{2}\left\{ \frac{\varepsilon_2(y_1-y_0)+\varepsilon_3(y_0-y_{-1})}{x_0-x_{-1}} \right\} U(x_{-1},y_0) \\
+\frac{1}{2}\left\{ \frac{\varepsilon_3(x_0-x_{-1})+\varepsilon_4(x_1-x_0)}{y_0-y_{-1}} \right\} U(x_0,y_{-1}) \quad \approx \quad -\frac{1}{\Delta z}\int\int\int \varrho \, dV
\end{aligned}
$$

The integral over the charge is known because the charge is known. To get the equation for the central potential value, one divides the equation by the factor before $U(x_0, y_0)$.
`SUBROUTINE make_grid (Gridrefinement.f90)`

1.3 Solving Poissons equation

The electrostatic potential is the solution of POISSONs equation. Via the finite difference approximation, the potentials at the gridpoints which are not fixed to a known potential obey a simple equation which involves the neighbour points and the charges at the gridpoints. The potentials within a grid are obtained by applying a multigrid solver (`EnMultigrid.f90, SUBROUTINE Multigrid_2D_Type3`). The potentials at the boundary of a grid must have the same value as the corresponding potentials of neighbour grids, or if there is no neighbour grid, as the potential values of the enclosing grid. This grid coupling is achieved by solving for the potentials of all grids iteratively, see figure 3. This is implemented in `Solver.f90 SUBROUTINE Solve_PotentialProblem`.

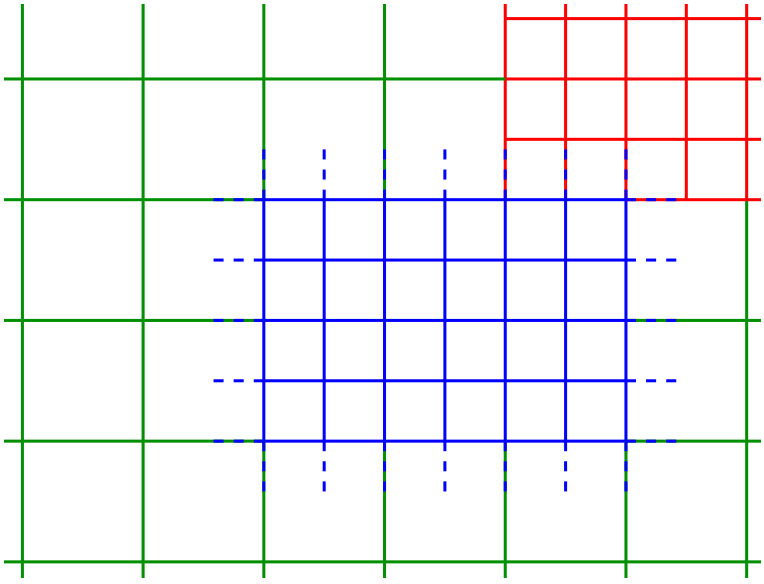


Figure 3: Grid coupling within an iteration to solve POISSONs equation: The potential values outside of the blue grid are computed by interpolating the values of the enclosing green grid and, when possible, by taking values from the red grid. After POISSONs equation for the blue grid is solved, the potential values are copied to the corresponding grid points of the green and red grid.

The copying from a neighbour grid and back needs information about what the neighbour grids are. The number of neighbour grids is not the same for each subgrid and the neighbour grids are not necessarily direct siblings. They could be subgrids of the neighbours of the parent's grid. This information is stored the grids datastructure component `grid% neighbours`. This datastructure is built at meshgeneration time in (`Gridrefinement.f90, SUBROUTINE Connect_Neighbourhood`). The algorithm to find the neighbours is: An array is defined which shall contain for each gridcell a pointer to the subgrid, if it exists. the outmost entries of this array contain pointers to the subgrids of the neighbours. For each subgrid, pointers just outside of the region which is covered by the subgrid itself point to the neighbours of the subgrid. See also the comments within that subroutine.

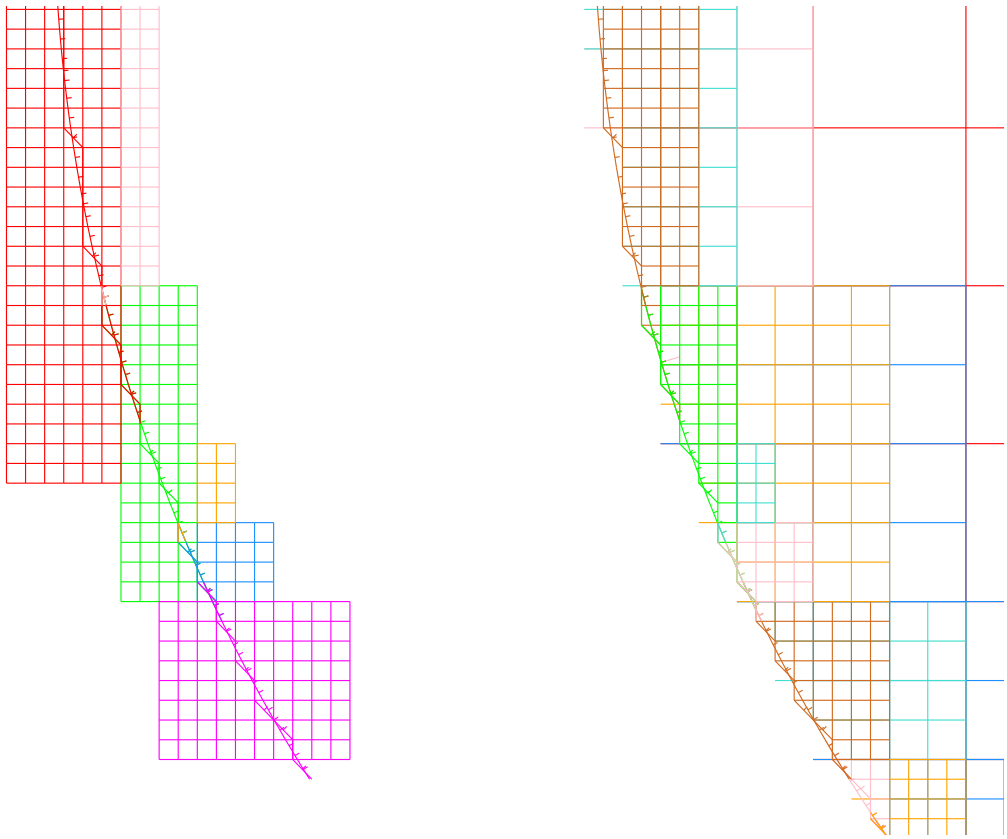


Figure 4: Grid coupling from a subgrid to a neighbours subgrid. The neighbours are not necessarily siblings of the same parent. Left: Shown are the neighbours of the green grid. Right: The green grid with its neighbours and the parent grids.

2 Particle Pushing

The particle pushing is done via an algorithm originally invented by D. Schulte The actual implementation as can be found in ECLOUD is taken and modified for the datastructures of Faktor2 . `SUBROUTINE scstep (Particles.f90)`.

2.1 Interpolation

The push depends on the electromagnetic field at the position of the center of mass of the particle. The electromagnetic field is retrieved from the grid via interpolation. For each field component the value at the position is interpolated from the values nearest to the position. In 3D, the interpolation is between twelve values. In 2D, the interpolation is between six values.
The field interpolation is implemented in
`Gridrefinement.f90 SUBROUTINE Give_EH_and_Boundarydetection`

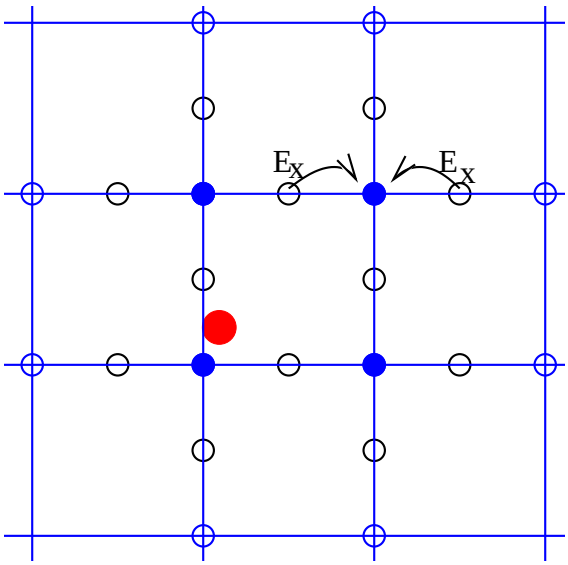


Figure 5: Interpolation of the field components. The electrostatic potentials at the blue circles are known from the solution of POISSONs equation. The x- and y-components of the electric field at the black circles are computed from the difference of the nearby potentials. The x- and y-components at the filled blue circles are computed as the average of the values at the nearby black circles. The field at the position of a particle, red circle, is computed by linear interpolation between the values at the four surrounding filled blue circles.

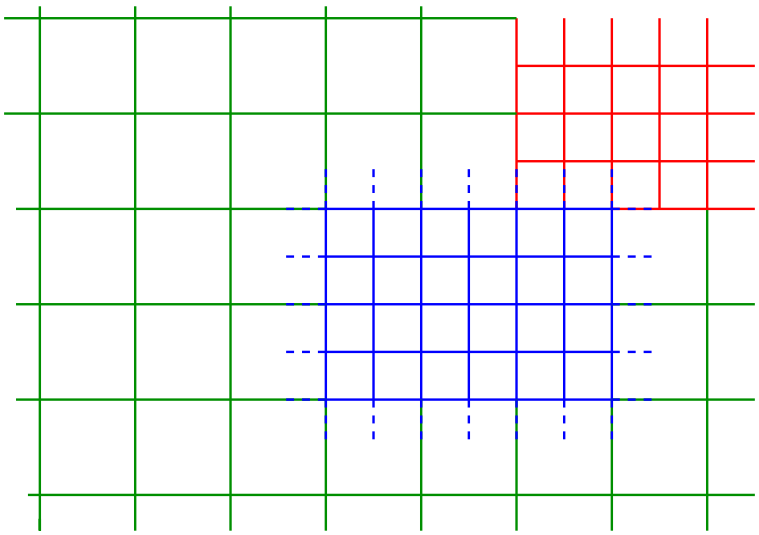


Figure 6: The green grid is a parent grid. The blue and red grid are subgrids. To interpolate field values within the area of the blue grid, some components outside the area are needed, if the position to interpolate for is near the boundary of the area. The components outside of the are are indicated with dotted lines. These components must be constructed either by copying them from a neighbour grid, or by interpolation from the parent grid.

When the position is near to the border of a grid, some of the values needed for the interpolation are outside of the grid. These values must be constructed from information of some other grid. If there is a neighbour grid available, use that values. Otherwise, construct the values via interpolation from the parent grid. The values outside of the grid which must be constructed are components which are normal to the border of the grid. These field components are in the *ghost cells* of the grid.

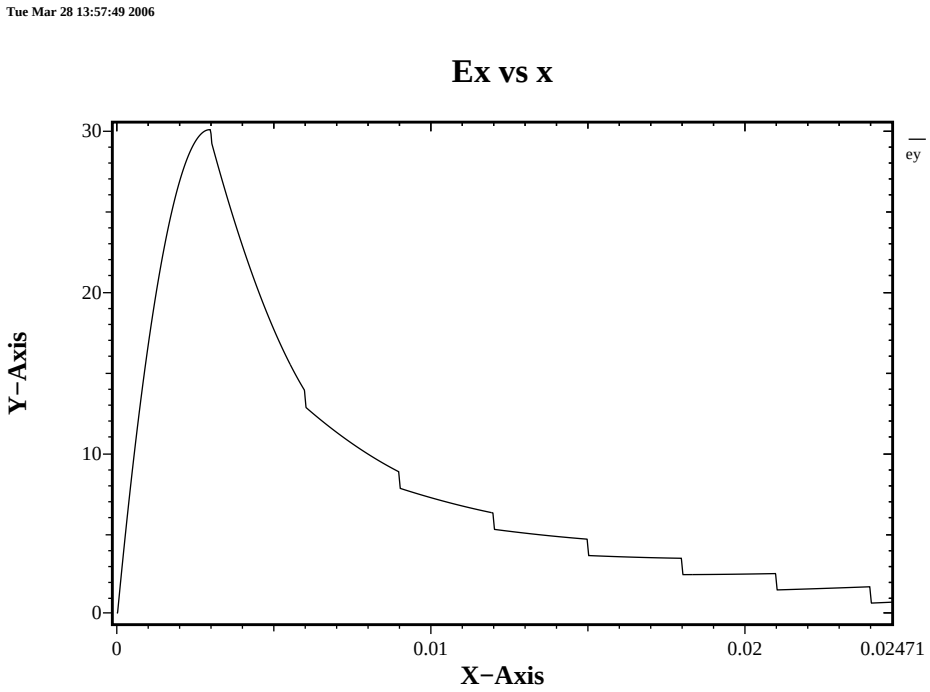


Figure 7: Computed attraction force on a particle as a function of the distance to the boundary at x=0.

2.1.1 Self force

While a particle moves trough free space, it should not be accelerated by its own field. Free space cannot easily be discretised. In a finite volume, a particle will be accelerated toward the nearest boundary. It will be attracted towards the nearest image charge.

2.2 Choice of timestep

PIC-codes normally use a global timestep, with which all particles are tracked. The advantage of a global timestep which is the same for all particles, is that the force due the particles can be computed from the position of the charges at their proper positions at the global times. The disadvantage of a global timestep is

- the tracking of slower particles is performed more often than necessary to resolve their motion,
- the computation of the charge due to the slower particles is performed more often than necessary.

Faktor2 does not use a global timestep for all particles. The trajectories are integrated with an individual timestep for each particle. This is strictly valid only if there would be exclusively external forces, no self forces. The timestep of a particle is chosen so small that within a timestep it can travel not more than 1/10 of the width of the gridcell where it is located. After each particle is tracked not longer than that timestep, the self-force is recomputed, and the particles which have not yet been tracked until the next synchronisation time are tracked again with their individual timestep.
The procedure can be described by:

- At a time t, all particles are at their positions at the same time t. All their local times are set to $t_i = t$.
- Tracking with individual timesteps, until all individual timesteps are zero:
  - Tracking of all particles over individual timesteps dt, where $dt = \min(t+\Delta t - t_i, dt_i)$. After that tracking, their times are set to $t_i := t_i + dt$. dt_i is chosen such that the particle cannot move more than 1/10 of the gridspacing where the particle is located. After tracking all particles, they are at different times, but they have reached positions which are not wrong by more than 1/10 of the local gridsize.

    Slow particles will be tracked with a large individual timestep, reaching the time $t + \Delta t$ in a single step. Fast particles will be tracked by several smaller timesteps.
  - Computation of the self forces via evaluating the charge distribution due to the newly reached positions.
- After all individual timesteps are zero, all particles have reached the same time $t + \Delta t$.

The effect of the individual timesteps is such, that the slower particles seem to jump to their new positions, while the faster particles move as they would move when all particles are tracked by the same small timestep.
The width of the local timesteps should be so small that within a timestep the particles

- do not travel more than their extension,
- do not travel more than the local gridspacing,
- do not change their direction significantly.

3 Charge Deposition

The charge in a gridcell is the fraction of the volume of the macroparticle which is inside the cell, multiplied by the charge density of the macroparticle. The shape of the macroparticles is assumed to be rectangular, in 2D the shape is a square, in 3D the shape is a cube. To compute the charge in the gridcells, all gridcells must processed which contain at least a fraction of the macroparticles volume. Therefore, the larger the size of a macroparticle is, the more CPU time is consumed to compute the charge distribution in the grid. It is therefore decided to compute with charges which are the same size as the gridcells. Then, in 2D the volume of a macroparticle is in no more than four gridcells. In 3D, the macroparticle is in no more than eight gridcells. But since Faktor2 uses a hierarchy of grids, there is no single size of gridcells. One does not want to use a size of macroparticles which is much smaller than the local gridsize, because then the charge within a gridcell would jump quite large when a macroparticle enters the gridcell. It jumps quite large, because a small macroparticle needs only a short time to fully transverse a grid face.

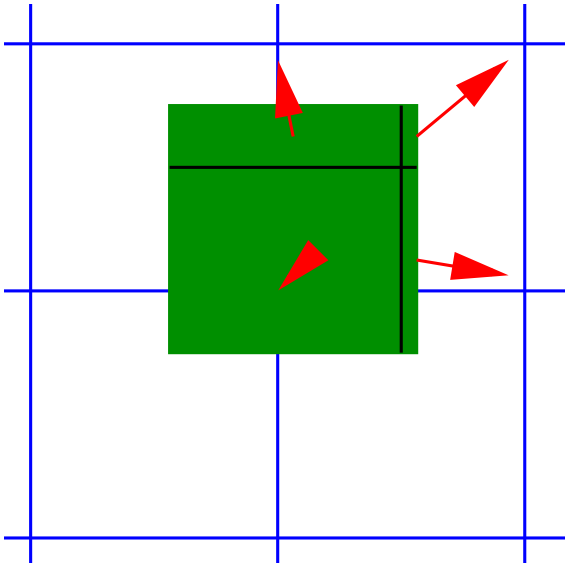


Figure 8: The charge is deposited on the grid according to the area of the charges which is within the rectangular area which is nearest to a given grid point.

The size of the macroparticles is assumed to be the size of the gridcell where the macroparticle is in. Therefore the size can change.

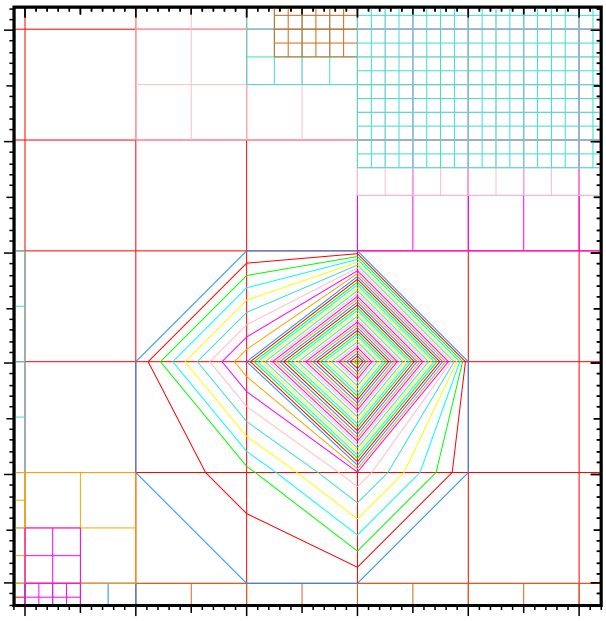


Figure 9: A macroparticle in a coarse grid region. The irregular lines are lines of equal charge density. The highest charge density is at (-0.2,-0.5) mm.

The deposition scheme shall dispose the charge such that the charge of a macroparticle sitting on a coarse fine transition shall be half within the fine grid, the other half in the coarse grid.

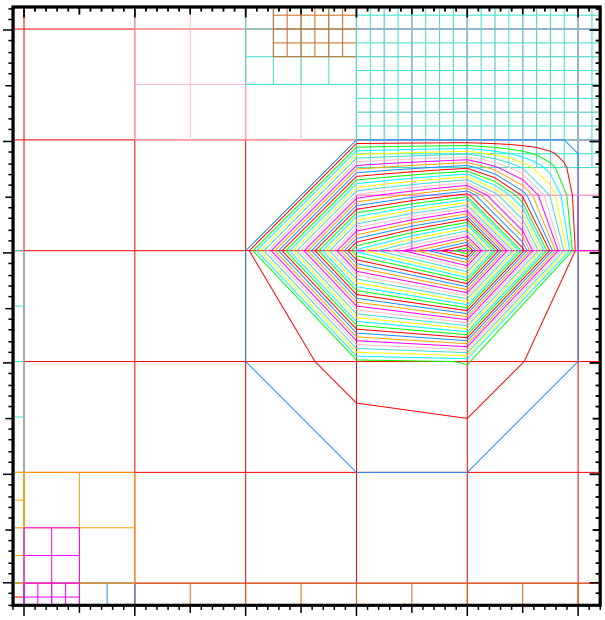


Figure 10: A macroparticle entering a region of medium sized cells. The irregular lines are lines of equal charge density. The highest charge density is at (-0.1,-0.4) mm.

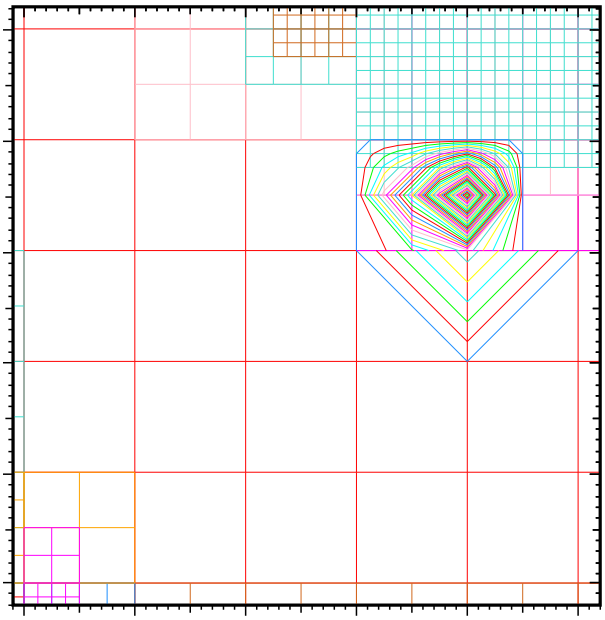


Figure 11: A macroparticle within a region of medium sized cells. The irregular lines are lines of equal charge density. The highest charge density is at (-0.1,-0.35) mm.

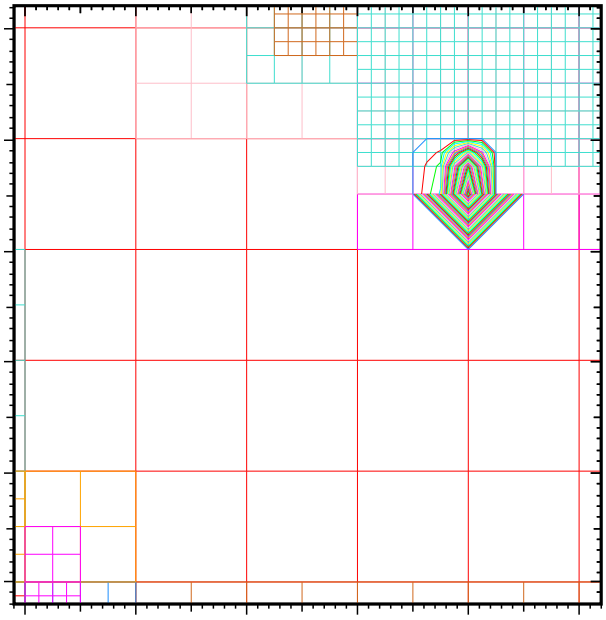


Figure 12: A macroparticle within between medium sized and small sized cells. The irregular lines are lines of equal charge density. The highest charge density is at (-0.1,-0.35) mm.

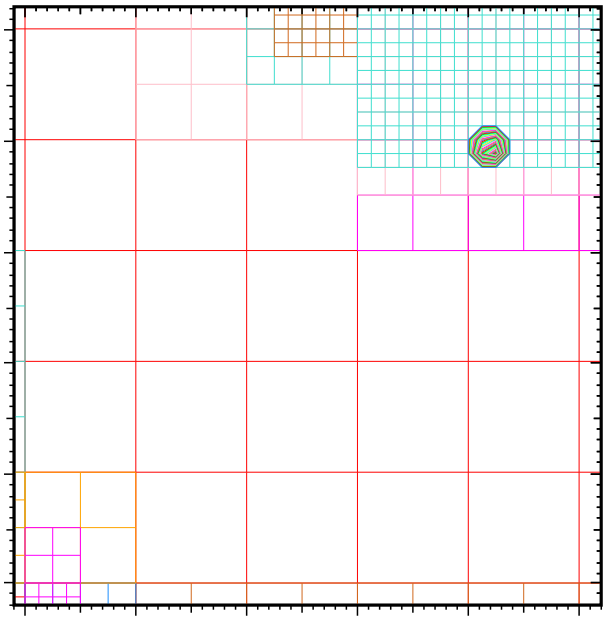


Figure 13: A macroparticle within small sized cells. The irregular lines are lines of equal charge density. The highest charge density is at (-0.08,-0.31) mm.

4 Combining Charges

Why macroparticles need to be merged When a charge hits a wall, secondary charges are created, depending on the energy and angle of impact. Faktor2 simulates this effect. In the simulation, if a macroparticle hits a wall, up[1] to three macroparticles are created for simulating the secondary charges. Therefore the number of macroparticles to track grows with time. Each macroparticle costs CPU-time and memory. To limit the number of macroparticles, every 10 timesteps nearby macroparticles are merged.
The problem is the identification, as a naive implementation via a comparison of the position of a particle with the positions of all other N-1 particles is a process with N^2 complexity. That would dominate the computation time.

Identification with complexity N log(N) In Faktor2 , the identification of nearby particles is done with a recursive algorithm which has a complexity of N log(N) with N the number of particles.

```
The initial region is the whole computational volume.
   A list is built, containing the particles in the region.
   ANALYSE THE PARTICLES IN REGION:
      IF the extension of the region is small enough,
         or the number of particles in the region is small enough,
         each particle's position and momentum is compared to the other
         particles in the region. If they are close in position and
         momentum, they are merged.
      ELSE
         Subdivide the region in two regions.
         ANALYSE THE PARTICLES IN REGION 1
         ANALYSE THE PARTICLES IN REGION 2
```

The algorithm is implemented in `Combine_Clouds.f90`. The building of the list has a complexity of N. The number of partitionings is proportional to the logarithm of N.

[1] When the charge of the hitting macroparticle is less than 5 × the average charge of the macroparticles, only a a single macroparticle is created for the secondary charges.