Electron & Positron Model Wave Function and Field calculation code

This is a portion of the model I wrote to model the Electron/Positron and their associated fields; such as Electric, Magnetic, Vector Potential fields. It is written in the Delphi language and is the function that calculates the fields from the mathematical wave function.

```
procedure TForm1.RecalcFields(scr:smallint);
var
  Current_Ex,Current_Ey,Current_Ez: extended;
  Current_Bx,Current_By,Current_Bz : extended;
  r,x,y,z,unit_x,unit_y,unit_z,k : extended;
  theta, delta, theta_const, expTheta, lnTheta, term0, term1, term2, term3 : extended;
  normal_x,normal_y,normal_z,dir_x,dir_y,dir_z : extended;
  scalar_amp, Vector_amp, SpinConstant, E_amp : extended;
  NewScreen : smallint;
  xpos,ypos,zpos,midx,midy,midz:smallint;
  ThisGroup,NewGroup: PointGrp;
  vect,CurlVect,DivVect: vector;
  Scalar_Group: ScalarGrp;
  VectGrp: VectorGrp;
  I: Integer;
  Etotal: Extended;
  ShellThickness: Extended;
  dist: longint;

begin

  if scr=0 then NewScreen:=1 else NewScreen:=0; {determine which data to update}

  if not Flip_YZ then begin

    midx:=Trunc(GridWidth/2);
    midy:=Trunc(GridHeight/2);
    midz:=Trunc(GridDepth/2);

    ////////////////////////////////////

    SpinConstant:=( Hhat / ElectronMass ); // Metres^2/(Radians*Second)
    delta := ( sqrt(2) * ElectronCharge * Hhat ) / ( 8 * Pi * ElectronMass * SpeedOfLight * Permittivity );

    // theta_const is in Radians/Second ( i.e. the same as solving E = hf for f, where E=mc^2, and h=2*Pi*Hhat,
    // then converting f to angular frequency w, via w = 2*Pi*f )
    // ( theta_const could be, equivalently : - c^2/SpinConstant )
    theta_const:=( -ElectronMass * sqr(SpeedOfLight) ) / Hhat;

    k:=FREQ_FACTOR/SpeedOfLight; // Seconds/Metre ( multiply by 1E-5 to make PsiWave visible (magnify) in model )
```

```
//
// Thus Total Electron Wave Equation (Ye) is:
//
// Ye = ((sqrt(2) * Qe*Hhat) / (8 * Pi * Me*c*Eo )) * Exp( ( - i * Me * c^2 / Hhat ) * ( T - r/c ) )
//
// and the Electric Potential div(psi) in spherical coordinates is
//
// V = ((sqrt(2) * Qe) / ( 4 * Pi * r * Eo )) * Exp( ( - i * Me * c^2 / Hhat ) * ( T - r/c ) )
//
// Where:
// Ye is Electron Wave Function (psi)
// Qe is Electron's Charge
// Pi is 3.14159 etc
// Eo is the Permittivity of free space
// Exp is the Exponential function
// i is the Complex number (square root of -1)
// Me is the Mass of an Electron
// c is the speed of light
// Hhat is the reduced Plancks constant ( i.e. h/(2*Pi) )
// T is Time
// r is the radial distance from the center of the Electron
//
// exp(-theta) = cos(theta) - isin(theta)
// using x,y,z coordinates:
// x = cos(theta)
// y = sin(theta)

// theta:=theta_const*(Time - k*r);
//
// term1:=delta
//
// term2:=cos(theta);
// term3:=-sin(theta);
//
// if ( ViewTop ) then begin // Assign values to x, y, z coordinates, depending on view from the top or side.
//   x:=term1 * term2;
//   y:=term1 * term3;
//   z:=0;
// end
// else begin
//   x:=term1 * term2;
//   y:=0;
//   z:=term1 * term3;
// end;

////////////////////////////////////
for xpos:=0 to GridWidth-1 do begin {scan grid's x coords}
  for ypos:=0 to GridHeight-1 do begin {scan grid's y coords}
    for zpos:=0 to GridDepth-1 do begin {scan grid's z coords}
      ThisGroup:=PointGroup(scr, xpos, ypos, zpos);

      x:= xpos - midx;
      y:= ypos - midy;
      z:= zpos - midz;

      r:=sqrt( sqr(x) + sqr(y) + sqr(z) );
      if ( r < 0.00000000001 ) then r:=0.00000000001;   // prevent divide by zero errors
```

```
      unit_x:= x/r;
      unit_y:= y/r;
      unit_z:= z/r;

      r:=r*(ActualWidth/GridWidth);   // get actual distance in metres
      if ( r < 0.00000000001 ) then r:=0.00000000001;   // prevent divide by zero errors

      ////////////////////////////////////
      /// WAVE FUNCTION TO TEST
      ///
      ///

      case StartOption of
       1: begin

       if ( electron ) then begin                 // if electron being modelled
         theta:=theta_const*(Time - k*r);
         term1:=delta;
       end
       else begin                                  // if positron being modelled
         theta:=theta_const*(Time + k*r);
         term1:=-delta;
       end;

       term2:=cos(theta);
       term3:=-sin(theta);

       // Assign values to x, y, z coordinates, depending on view from the top or side.
       with points[NewScreen,xpos,ypos,zpos].PsiVect do begin
         if ( ViewTop ) then begin
           x:=term1 * term2;
           y:=term1 * term3;
           z:=0;
         end
           else begin
           x:=term1 * term2;
           y:=0;
           z:=term1 * term3;
         end;
       end;
       points[NewScreen,xpos,ypos,zpos].Psi := term1;
       end;
      end;

      ///
      ///
      ////////////////////////////////////

    end;
  end;
end;   // end {scan grid's x coords}

for xpos:=0 to GridWidth-1 do begin {scan grid's x coords}
  for ypos:=0 to GridHeight-1 do begin {scan grid's y coords}
    for zpos:=0 to GridDepth-1 do begin {scan grid's z coords}

      ThisGroup:=PointGroup(scr, xpos, ypos, zpos);
```

```
      NewGroup:=PointGroup(NewScreen, xpos, ypos, zpos);

      with points[NewScreen,xpos,ypos,zpos] do begin
       if (smoothing) then begin
        x:= xpos - midx;
        y:= ypos - midy;
        z:= zpos - midz;

        r:=sqrt( sqr(x) + sqr(y) + sqr(z) );
        if ( r < 0.004 ) then r:=0.004;   // prevent divide by zero errors

        ElectricPotential:=ElectronCharge/(4*Pi*r*Permittivity);
       end
       else begin
        VectGrp:=VectorGroup(NewGroup, PSI_VECTOR_FIELD);
        ElectricPotential:=VectDiv(VectGrp);
       end;
      end;
    end;
  end;
end; // end {scan grid's x coords}

for xpos:=0 to GridWidth-1 do begin {scan grid's x coords}
  for ypos:=0 to GridHeight-1 do begin {scan grid's y coords}
    for zpos:=0 to GridDepth-1 do begin {scan grid's z coords}

      ThisGroup:=PointGroup(scr, xpos, ypos, zpos);
      NewGroup:=PointGroup(NewScreen, xpos, ypos, zpos);

      { ThisGroup's points are assigned as follows: P3           P5
                                                  P1 P0 P2
                                                     P4     P6

      Where P5 & P6 are in the Z plane (P5 at the back and P6 at the front) }

      x:= xpos - midx;
      y:= ypos - midy;
      z:= zpos - midz;

      r:=sqrt( sqr(x) + sqr(y) + sqr(z) );
      if ( r < 0.00000000001 ) then r:=0.00000000001;   // prevent divide by zero errors

      unit_x:= x/r;
      unit_y:= y/r;
      unit_z:= z/r;

      r:=r*(ActualWidth/GridWidth);   // get actual distance in metres

      if ( ViewTop ) then begin   // Calculate Normal vector to current unit vector (depending on view from top or side)
       normal_x:=unit_y;
       normal_y:=-unit_x;
       normal_z:=unit_z;
      end
      else begin
       normal_x:=unit_z;
       normal_y:=unit_y;
       normal_z:=-unit_x;
      end;
```

```
        // Electric Field is: -div of ElectricPotential Field - d/dt of Vector Potential field
        Scalar_Group:=ScalarGroup(NewGroup, ELECTRIC_POTENTIAL_FIELD);

        // This is the div of ElectricPotential Field (will add the rest once the Vector Potential is known)
        points[NewScreen,xpos,ypos,zpos].Electric:=ScalarGrad(Scalar_Group);

        // get amplitude of Static Electric field component
        E_amp:=VectSize(points[NewScreen,xpos,ypos,zpos].Electric);

        // From Schrodinger's wave equation:
        //
        // d(psi)/dt = i * Hhat/ElectronMass * Laplacian(psi)
        //
        // Note: div(V) = Laplacian(psi)
        // SpinConstant = Hhat/ElectronMass
        //
        // So…
        // d(psi)/dt = i*SpinConstant*div(V)
        //
        // VectorPotential = (1/c)*d(psi)/dt
        //
        // A is orthogonal to div(V) and also proportional to div(V)
        // so use the Normal vector and the Static Electric field amplitude (E_amp).

        with points[NewScreen,xpos,ypos,zpos].VectorPotential do begin
          x := normal_x*SpinConstant*E_amp/SpeedOfLight;
          y := normal_y*SpinConstant*E_amp/SpeedOfLight;
          z := normal_z*SpinConstant*E_amp/SpeedOfLight;
        end;

        // Electric Field is: -div of ElectricPotential Field - d/dt of Vector Potential field
        // In Electric, we already have div of ElectricPotential Field, now make negative & subtract
        // d/dt of Vector Potential field

        with points[NewScreen,xpos,ypos,zpos].Electric do begin
          // E = -div(V) - (1/c)*dA/dt
          // dA/dt = (A^2/r) (i.e. like acceleration of angular velocity in circular motion)
          x := -x - unit_x*((sqr(points[NewScreen,xpos,ypos,zpos].VectorPotential.x)/r)/SpeedOfLight);
          y := -y - unit_y*((sqr(points[NewScreen,xpos,ypos,zpos].VectorPotential.y)/r)/SpeedOfLight);
          z := -z - unit_z*((sqr(points[NewScreen,xpos,ypos,zpos].VectorPotential.z)/r)/SpeedOfLight);
        end;

        // Magnetic Field is Curl of Vector Potential Field
        VectGrp:=VectorGroup(NewGroup, VECTOR_POTENTIAL_FIELD);
        CurlVect:=VectCurl(VectGrp);

        with points[NewScreen,xpos,ypos,zpos].Magnetic do begin
          x:=Permeability*CurlVect.x;
          y:=Permeability*CurlVect.y;
          z:=Permeability*CurlVect.z;
        end;
       end;
     end;
   end; // end {scan grid's x coords}
 end; //if Flip_YZ
end;
```