```
/*
COMPLETENESS DISCLAIMER: The source code for all file, graphic, string, and memory primitives
such as FileAddLine, Gcircle, StrDelimit, and MemClear used herein is not included but could easily be
created for the platform for which this source code will be compiled
*/

/*------------------------------------------------------------------------------------------------------------------- */
/*                                                                                                                     */
/*                                                                                                                     */
/*                          SPHERICAL 3D VARIABLE PARTICLE CORE ROUTINES                                               */
/*                                                                                                                     */
/*                                                                                                                     */
/*------------------------------------------------------------------------------------------------------------------- */
/* (c) Copyright Terence Witt 2007, All Rights Reserved */
/* NULLPHYSICS.COM */
/* PARTICLE CORE RECESSION */

/* master includes */
#include "witt.h"
#include "resource.h"
#include "zproject.h"

/* prototypes */
void Discores(long numpar, double numpos, double numneg);
double ParseCorePosition(long parnum, char *textvalue);
double SpaceEnergyCoulomb(long coulombmode, long numpar, double d);

BOOL RecessedEnergy(BOOL vary_ereso, BOOL x_symmetric, long conmode, long numpar, double rc,
        double rscan, double dR, double corrlimit, double corrpow, double reso, double d, double etotal,
        double centralsamplesize, double *ereso, double *edelta, double *converge, double *lastconverge,
        double *recessrate, double *Ecenter1, double *Ecenter2, double *Esum, double *Ecentralsample,
        double *totalnonsuperE, double *corevolume, double *deltavolume,
        double *dRfin, double timstart, double *ptvol, double *pteng, char *tottim);

long LoadParticles(char *parfile, BOOL *onepass, BOOL *vary_d, BOOL *vary_ereso, long *conmode, long *coulombmode,
        double rbasis, double *reso, double *Ereso, double *d, double *standoff, double *rscan, double *conrate,
        double *recessrate, double *rc, double *corrlimit, double *corrpow, double *numvary, double *numfixed,
        double *numneg, double *numpos, double *numneutral, double *totalparE, double *totalneutralE);

void TestPrecision(long numpar, double rmid, double rscan, double Ecenter1, double Ecenter2, double corevolume,
        double *Vexpect, double *Eexpect1, double *Eexpect2, double *Vexpectfract, double *Eexpectfract1,
        double *Eexpectfract2);

/* initialize particle field constants */
void InitGlobals(void)
{
        /* unit polarvolume defines field profile */
        HVpar = h_*c/(4.0*PI);

        /* proton radius */
        Rp = (9.0*HVpar)/(mp*c*c);

        /* distribution constants for deflection and slope */
        KKt = (3.0*HVpar)/(4.0*PI);
        KKs = (9.0*HVpar)/(4.0*PI);

        /* energy at a radius constant */
        KKp = 9.0*HVpar;
}

/*------------------------------------------------------------------------------------------------------------------- */
/*                                                                                                                     */
/*                                                                          */
/*                  MULTINUCLEON CORE RECESSION (OPTIMIZED)                 */
/*                                                                          */
/*                                                                          */
/*------------------------------------------------------------------------- */

/* integrate space energy with multiple layers of nucleons */
/* can do small systems of mixed core size or large systems of equal core size */
/* has ability to fix or vary interparticle distance, set fixed standoffs, fix or vary core radii */
/* fixed cores should precede variable cores in particle file */
/* variable core radius is rc */
BOOL CoreRecession(char *particlefile, char *resultfile, BOOL x_symmetric, BOOL force_once, double force_d,
        double force_rc, double force_rscan, double force_corrlimit, double force_centralsamplesize, char *tottim)
{
        double  dinit=0.0, rc=0.0, rscan=0.0, rbasis, rcinit, rmid, dR, dRfin, d, Rrecess, Ecenter1, Ecenter2;
        double  Esum, Ecentralsample=0.0, Etotal, Ereso, Eresoinit, reso, converge=0.0, lastconverge=0.0;
        double  Edelta=0.0, Eexpect1=0.0, Eexpect2=0.0, Eexpectfract1=0.0, Eexpectfract2=0.0, standoff=0.0;
        double  conrate, corevolume, Vexpect=0.0, Vexpectfract=0.0, integrity=0.0, targetd, minslope=0.0;
        double  Recessfract, timdel, timstart=0, totalnonsuperE=0.0, retension=0.0, Erepulse, neutralE;
        double  recessrate, rlimit, testr, totalparE, totalneutralE, numvary=0.0, newx=0.0, newy=0.0, newz=0.0;
        double  numfixed=0.0, numpos=0.0, numneg=0.0, numneutral=0.0, atomicnum=0.0, centralsamplesize=1.0;
        double  corrlimit=0.0, corrpow=0.0, Ecentralsampleratio=0.0, parity=0.0, ptvol=0, pteng=0, deltavolume=0.0;
        long    numpar, coulombmode, nn, cycles=0, conmode=0;
        char    af[300], parfile[200], calcfile[200], tim[200];
        BOOL    abortit=FALSE, nestled=FALSE, vary_d, vary_ereso, onepass;

        MemClear(&xc[0], sizeof(xc));
        MemClear(&yc[0], sizeof(yc));
        MemClear(&zc[0], sizeof(zc));
        MemClear(&chargec[0], sizeof(chargec));
        MemClear(&fixcorec[0], sizeof(fixcorec));
        MemClear(&coresizec[0], sizeof(coresizec));
        MemClear(&Ebasec[0], sizeof(Ebasec));
        MemClear(&colorc[0], sizeof(colorc));
        strcpy(parfile, particlefile);
        strcpy(calcfile, resultfile);
        if(!calcfile[0])
        {
                strcpy(calcfile, OUTFILE);
        }

        /* clear message frame */
        Gframe(MESSAGEFRAME);

        InitGlobals();

        /* two-dimensional modeling is fullly supported */
        longcalc = IsDlgButtonChecked(H(DLGMAIN), IDC_LONGCALC);
        displaycore = IsDlgButtonChecked(H(DLGMAIN), IDC_DISPLAYCORE);

        onepass = FALSE;
        vary_d = TRUE;
        vary_ereso = TRUE;
        standoff = 0.0;
        conrate = 1.5;
        recessrate = 1.5;
        conmode = 0;

        /* beginning and rate of dr expansion */
        corrlimit = 1.33;
        corrpow = 2.0;
```

```
minslope = 1.0e99;
Ereso = Eresoinit = 100000.0;
reso = 200.0;
rc = rbasis = Rp;
rscan = 100e-15;

Erepulse = Ecenter1 = Ecenter2 = Esum = corevolume = totalparE = totalneutralE = neutralE = 0.0;
oldEdelta = oldEcenter1 = oldEcenter2 = oldEsum = oldEcentralsample = oldtotalnonsuperE = 0.0;
oldcorevolume = olddRfin = 0.0;

numvary = 0.0;
numpar = coulombmode = 0;

/* load particle file */
numpar = LoadParticles(parfile, &onepass, &vary_d, &vary_ereso, &conmode, &coulombmode, rbasis, &reso,
        &Ereso, &d, &standoff, &rscan, &conrate, &recessrate, &rc, &corrlimit, &corrpow, &numvary,
        &numfixed, &numneg, &numpos, &numneutral, &totalparE, &totalneutralE);
MessageClear(0);
if(!numpar)
{
        return(abortit);
}

/* forced parameters, if they exist */
if(force_once)
{
        onepass = TRUE;
}
if(force_d>0.0)
{
        d = (force_d*Rp);
        vary_d = FALSE;
}
if(force_rscan>0.0)
{
        rscan = force_rscan*Rp;
}
if(force_rc>0.0)
{
        rc = force_rc*Rp;
        for(nn=1; nn<=numpar; nn++)
        {
                coresizec[nn] = rc;
        }
}
if(force_corrlimit>0.0)
{
        corrlimit = force_corrlimit;
}
if(force_centralsamplesize>0.0)
{
        centralsamplesize = force_centralsamplesize;
}

/* show configuration and exit */
if(vary_d)
{
        dinit = d = 2.0*rc;
}else{
        dinit = d;
}
rcinit = rc;
```

```
if(displaycore)
{
        Discores(numpar, numpos, numneg);
        return(abortit);
}

/* initial dimensional adjustment ------------------------------------------------------------------------ */
if(!onepass)
{
        /* adjust lateral distance for smaller nuclei when core sizes differ,  Everything is scaled by d */
        if(numfixed)
        {
                if((numpar>1) && (numpar<6) && fixcorec[1] && !fixcorec[numpar])
                {
                        /* set initial distance */
                        if(vary_d)
                        {
                                d = coresizec[1] + coresizec[numpar];
                        }

                        /* symmetry is necessary, adjust y or z depending on last variable core.
                        Target d is */
                        targetd = coresizec[1] + coresizec[numpar];
                        newx = newy = newz = 0.0;
                        if(zc[numpar]!=0.0)
                        {
                                /* adjust z */
                                newz = sqrt((targetd*targetd/(d*d)) - (xc[numpar]*xc[numpar])
                                        - (yc[numpar]*yc[numpar]));
                        }else{
                                if(yc[numpar]!=0.0)
                                {
                                        /* adjust y */
                                        newy = sqrt((targetd*targetd/(d*d)) - (xc[numpar]*xc[numpar]));
                                }else{
                                        if(vary_d)
                                        {
                                                newx = targetd/d;
                                        }
                                }
                        }
                        for(nn=2; nn<=numpar; nn++)
                        {
                                if(!fixcorec[nn])
                                {
                                        if((newx>0.0) && (xc[nn]!=0.0))
                                        {
                                                if(xc[nn]<0.0)
                                                {
                                                        xc[nn] = -newx;
                                                }else{
                                                        xc[nn] = newx;
                                                }
                                        }
                                        if((newy>0.0) && (yc[nn]!=0.0))
                                        {
                                                if(yc[nn]<0.0)
                                                {
                                                        yc[nn] = -newy;
                                                }else{
                                                        yc[nn] = newy;
                                                }
```

```
                    }
                    if((newz>0.0) && (zc[nn]!=0.0))
                    {
                        if(zc[nn]<0.0)
                        {
                            zc[nn] = -newz;
                        }else{
                            zc[nn] = newz;
                        }
                    }
                }
            }
        }
    }

    /* preserve core integrity in systems with uniform core size */
    if(!numfixed)
    {
        if(vary_d)
        {
            d = rc*(2.0+standoff);
        }else{
            if((2.0*rc)>d)
            {
                rc = d/2.0;
                for(nn=1; nn<=numpar; nn++)
                {
                    if(!fixcorec[nn])
                    {
                        coresizec[nn] = rc;
                    }
                }
            }
        }
    }
}

/* calculate ------------------------------------------------------------------------------------------- */
Eresoinit = Ereso;

/* STEP 1 calculate total system energy based on number of nucleons and scanning range */

/* calculate total input energy */
Etotal = (totalparE*KKp*((1.0/rbasis) - (1.0/rscan)));

/* add Coulomb element */
if(coulombmode)
{
    Erepulse = SpaceEnergyCoulomb(coulombmode, numpar, d);
}else{
    Erepulse = 0.0;
}
Etotal += Erepulse;

/* remove neutral energy from consideraiton */
neutralE = totalneutralE*mp*c*c;
Etotal -= neutralE;

timstart = AgeNow();

dR = rbasis/reso;
for(;;)
```

```
{
	/* STEP 2 calculate delta energy at current core radius and distance */
	cycles++;
	strcpy(note_text, parfile);
	FileCore(parfile, note_text);
	abortit = RecessedEnergy(vary_ereso, x_symmetric, conmode, numpar, rc, rscan, dR,
		corrlimit, corrpow, reso, d, Etotal, centralsamplesize, &Ereso, &Edelta, &converge,
		&lastconverge, &recessrate, &Ecenter1, &Ecenter2, &Esum, &Ecentralsample, &totalnonsuperE,
		&corevolume, &deltavolume, &dRfin, timstart, &ptvol, &pteng, tottim);
	if(!abortit)
	{
		/* calculate slope retension in the interstitial areas */
		if(totalnonsuperE>0.0)
		{
			retension = Esum/totalnonsuperE;
		}

		/* accuracy checks done on the fly in case of abort */
		rmid = d/2.0;
		if(rscan<rmid)
		{
			rmid = rscan;
		}
		TestPrecision(numpar, rmid, rscan, Ecenter1, Ecenter2, corevolume, &Vexpect, &Eexpect1,
			&Eexpect2, &Vexpectfract, &Eexpectfract1, &Eexpectfract2);

		/* STEP 3 adjust core radii until Esum is the same as Etotal for fixed distance */
		if(!onepass)
		{
			if(fabs(converge)<1.0)
			{
				nestled = TRUE;
			}else{
				/* calculate recession amount */
				Rrecess = -(recessrate/numvary)*((rc*rc)/KKp)*Edelta;

				/* insure stable recession -------------------------------------------- */
				testr = rc + Rrecess;
				if(testr>(rc*conrate))
				{
					/* limit rate of growth */
					rc *= conrate;
				}else{
					if(testr<(rc/conrate))
					{
						/* limit rate of shrinkage */
						rc /= conrate;
					}else{
						rc += Rrecess;
					}
				}

				/* preserve core integrity in systems with uniform core size */
				if(!numfixed)
				{
					if(vary_d)
					{
						d = rc*(2.0+standoff);
					}else{
						if((2.0*rc)>d)
						{
							rc = d/2.0;
```

```
                }
        }
}

/* adjust lateral distance for smaller nuclei when core sizes differ,
Everything is scaled by d */
if(numfixed)
{
        if((numpar>1) && (numpar<6) && fixcorec[1] && !fixcorec[numpar])
        {
                /* set initial distance */
                if(vary_d)
                {
                        d = coresizec[1] + coresizec[numpar];
                }

                /* symmetry is necessary, adjust y or z depending
                on last variable core.  Target d is */
                targetd = rc + coresizec[1];
                newx = newy = newz = 0.0;
                if(zc[numpar]!=0.0)
                {
                        /* adjust z */
                        newz = sqrt((targetd*targetd/(d*d))
                                - (xc[numpar]*xc[numpar])
                                - (yc[numpar]*yc[numpar]));
                }else{
                        if(yc[numpar]!=0.0)
                        {
                                /* adjust y */
                                newy = sqrt((targetd*targetd/(d*d))
                                        - (xc[numpar]*xc[numpar]));
                        }else{
                                if(vary_d)
                                {
                                        newx = targetd/d;
                                }
                        }
                }
                for(nn=2; nn<=numpar; nn++)
                {
                        if(!fixcorec[nn])
                        {
                                if((newx>0.0) && (xc[nn]!=0.0))
                                {
                                        if(xc[nn]<0.0)
                                        {
                                                xc[nn] = -newx;
                                        }else{
                                                xc[nn] = newx;
                                        }
                                }
                                if((newy>0.0) && (yc[nn]!=0.0))
                                {
                                        if(yc[nn]<0.0)
                                        {
                                                yc[nn] = -newy;
                                        }else{
                                                yc[nn] = newy;
                                        }
                                }
                                if((newz>0.0) && (zc[nn]!=0.0))
                                {
                                    if(zc[nn]<0.0)
                                    {
                                        zc[nn] = -newz;
                                    }else{
                                        zc[nn] = newz;
                                    }
                                }
                            }
                        }
                    }
                }

                /* set all variable cores to new rc value */
                for(nn=1; nn<=numpar; nn++)
                {
                    if(!fixcorec[nn])
                    {
                        coresizec[nn] = rc;
                    }
                }
            }
        }

        /* test integrity */
        if(numvary==numpar)
        {
            integrity = d/(2.0*coresizec[1]);
        }else{
            integrity = d/(coresizec[1]+coresizec[numpar]);
        }
    }

    /* the cores are nestled */
    if(nestled||abortit||onepass)
    {
        break;
    }
}

/* output results ------------------------------------------------------------------------- */
Gjust(TA_CENTER);
Gbold(1);

Recessfract = rc/rbasis;
if(numneg>0.0)
{
    atomicnum = numpar - 2.0*numneg;
}else{
    if(numneutral>0.0)
    {
        atomicnum = numpar - numneutral;
    }
}
Ecentralsampleratio = Ecentralsample/(mp*c*c);

/* completion duration */
timdel = AgeNow() - timstart;
AgeTxt(timdel, tim);
rlimit = rc*corrlimit*pow(numpar, 0.33333);

if(Efield[1]>0.0)
```

```
{
        parity = Efield[2]/Efield[1];
}

FileAddLine(calcfile, "");
strcpy(af, "SPHERICAL MULTIPARTICLE CORE RECESSION");
if(vary_d)
{
        strcat(af, "   (Variable d)");
}else{
        strcat(af, "   (Fixed d)");
}
if(x_symmetric)
{
        strcat(af, "   (X symmetric)");
}
if(onepass)
{
        strcat(af, "   (Single Pass)");

}
if(abortit)
{
        strcat(af, "   (ABORTED)");
}
FileAddLine(calcfile, af);
sprintf(af, "    (%s)   Cores: %ld   Cycles: %ld   Reso: %ld    Ver: %s",
        parfile, numpar, cycles, (long)reso, VERSIONNUMBER);
FileAddLine(calcfile, af);
if(!onepass && numfixed)
{
        sprintf(af, "    NewX:   %6.4f     NewY:   %6.4f     NewZ:  %6.4f", newx, newy, newz);
        FileAddLine(calcfile, af);
}
sprintf(af, "    Rinit:   %-6.4f   Rscan:   %-7.2f   Rlim: %-6.4f     Dinit:   %-6.4f",
        rcinit/Rp, rscan/Rp, rlimit/Rp, dinit/Rp);
FileAddLine(calcfile, af);
sprintf(af, "    Corrlim: %-6.2f   Corrpow: %-6.2f    Conv: %-8.2f   Conmode: %-ld", corrlimit, corrpow,
converge, conmode);
FileAddLine(calcfile, af);
sprintf(af, "    Reso:    %-6.0f   EresIni: %-6.0f    Eres: %-6.0f     Mpts/s:  %-6.2f",
        reso, Eresoinit, Ereso, (pteng/1.0e6)/timdel);
FileAddLine(calcfile, af);
sprintf(af, "    Charge:  %-6.1f   Resrate: %-4.1f      Soff: %-6.1f", atomicnum, recessrate, standoff);
FileAddLine(calcfile, af);
if(x_symmetric)
{
        sprintf(af, "    Retens:  %-7.5f", retension);
        FileAddLine(calcfile, af);
}else{
        sprintf(af, "    Retens:  %-7.5f  Estital: %-8.6f  Centersize: %-7.5f",
                retension, Ecentralsampleratio, centralsamplesize);
        FileAddLine(calcfile, af);
}
sprintf(af, "    Ecal:    %7.5e (%7.5e)         Gpts: %8.6f", Ecenter1, Eexpect1, pteng/1.0e9);
FileAddLine(calcfile, af);
if(x_symmetric)
{
        sprintf(af, "           Err1:  %10.8f %%   Err2:    %10.8f %%", Eexpectfract1, Eexpectfract2);
        FileAddLine(calcfile, af);
        sprintf(af, "           Esum: %7.5e    Enonsum: %7.5e", Esum, totalnonsuperE);
        FileAddLine(calcfile, af);
```

```
	}else{
		sprintf(af, "		Err1: %10.8f %%    Err2:    %10.8f %%", Eexpectfract1, Eexpectfract2);
		FileAddLine(calcfile, af);
		sprintf(af, "		Esum: %7.5e    Enonsum: %7.5e   E2/E1: %10.8f",
			Esum, totalnonsuperE, parity);
		FileAddLine(calcfile, af);
		sprintf(af, "    Core V:  %7.5e (%7.5e)         Mpts: %8.6f", corevolume, Vexpect, ptvol/1.0e6);
		FileAddLine(calcfile, af);
		sprintf(af, "		Err:  %10.8f %%", Vexpectfract);
		FileAddLine(calcfile, af);
	}
	sprintf(af, "    dRbeg: %7.5e   dRfin: %7.5e   Ratio:  %-10.2f", dR, dRfin, dRfin/dR);
	FileAddLine(calcfile, af);
	sprintf(af, "    Rc:    %7.5e       d: %7.5e   Recess: %-8.6f", rc, d, Recessfract);
	FileAddLine(calcfile, af);
	sprintf(af, "    d:     %8.6e    DelR:   %8.6e    (%s)", d, Rp-rc, tim);
	FileAddLine(calcfile, af);

	Gframe(MESSAGEFRAME);
	sprintf(af, "Cores: %ld    Rc: %7.5e    d: %7.5e    Recess: %-8.6f    (%s)",
		numpar, rc, d, Recessfract, tim);
	Gtextframesmall(MESSAGEFRAME, COLORBLUE, "", af, 21);
	Gbold(0);

	/* return to default on exit in case of small core calc */
	longcalc = FALSE;

	return(abortit);
}

/*------------------------------------------------------------------------------------------------------------------- */
/*                                                                                                                     */
/*	ENERGY SUMMATION FOR CORE RECESSION (OPTIMIZED)                                                                   */
/*                                                                                                                     */
/*------------------------------------------------------------------------------------------------------------------- */
/* calculate energy in a spatial volume for multiple particles */
BOOL RecessedEnergy(BOOL vary_ereso, BOOL x_symmetric, long conmode, long numpar, double rc, double rscan,
	double dR, double corrlimit, double corrpow, double reso, double d, double etotal, double centralsamplesize,
	double *ereso, double *edelta, double *converge, double *lastconverge, double *recessrate, double *Ecenter1,
	double *Ecenter2, double *Esum, double *Ecentralsample, double *totalnonsuperE, double *corevolume,
	double *deltavolume, double *dRfin, double timstart, double *ptvol, double *pteng, char *tottim)
{
	COLORREF		colormap[]={0, COLORBLUE, COLORGREEN, COLORSKY, COLORRED, COLORPURPLE,
		COLORYELLOW, COLORWHITE, COLORORANGE, COLORGRAY};
	COLORREF		parcolor, rcolor;
	char			af[300], rtxt[200], curtim[200];
	double			rlimit, raxis, maxrecess;
	double			B, slope, dE;
	double			rphi, theta, slopevector, phi, dPhi, dTheta, dEsum, dV, convergerate;
	double			corrmax, corrdR, corr, rstatus, timdel, d_centralsamplesize;
	double			xx, yy, zz, tx, ty, tz, dEfield;
	long			n=0, markit=0, atnum=0;
	BOOL			abortit=FALSE, incore, addit;

	/* manganese is too big not to use arrays, have to just take the performance hit */
	*Ecenter1 = *Ecenter2 = *Esum = *Ecentralsample = *totalnonsuperE = *corevolume = 0.0;
	*deltavolume = 0.0;

	MemClear(&Efield[0], sizeof(Efield));

	/* differential counters */
	*ptvol = *pteng = 0;
```

```
MemClear(&r_1[0], sizeof(r_1));
MemClear(&r_2[0], sizeof(r_2));
MemClear(&r_4[0], sizeof(r_4));
MemClear(&r_5[0], sizeof(r_5));
MemClear(&xcd[0], sizeof(xcd));
MemClear(&ycd[0], sizeof(ycd));
MemClear(&zcd[0], sizeof(zcd));

dTheta = (2.0*PI)/reso;
dPhi = dTheta;
d_centralsamplesize = d*centralsamplesize;
maxrecess = 10.0;

/* define graphics boundaries for square space, y is the limitation */
/* when all boundaries have been included allow dR to increase with decreasing profile */
/* this is defined by the number of particles, assuming a volumetric distribution */
rlimit = corrlimit*pow(numpar, 0.33333)*rc;
corrmax = KKs/pow(rlimit, corrpow);
corrdR = corrmax*dR;

raxis = 4.0*coresizec[1]*pow(numpar, 0.3333);
if((numpar>2) && (numpar<6))
{
        raxis = 2.5*coresizec[1]*pow(numpar, 0.3333);
}
/* specific for muon */
if((numpar==3) && (coresizec[1]>(5.0*Rp)))
{
        raxis = 10.0*coresizec[1]*pow(numpar, 0.3333);
}
/* initialize scanning radius, always centered on primary particle */
r_1[1] = coresizec[1];

ProcessGraphClear();
ProcessGraphAxes(FALSE, raxis);
ProcessGraphSeparation(FALSE, 0.0, 0.0, raxis);
/* calculate atomic number and scale distance */
for(n=1; n<=numpar; n++)
{
        atnum += (long)chargec[n];
        xcd[n] = d*xc[n];
        ycd[n] = d*yc[n];
        zcd[n] = d*zc[n];
}

/* THREE DIMENSIONAL CALC ------------------------------------------------------------------------------- */
for(;;)
{
        r_2[1] = r_1[1]*r_1[1];
        r_5[1] = r_2[1]*r_2[1]*r_1[1];
        slope = KKs/(r_2[1]*r_2[1]);
        corr = KKs/pow(r_1[1], corrpow);

        /* havent graphed this dr yet */
        markit = 0;

        /* integrate rings from 0 to PI deltheta keeps point density higher on smaller shells ------------- */
        for(phi=0.0; phi<PI; phi+=dPhi)
        {
                zz = r_1[1]*cos(phi);
```

```
/* mark closest to z=0 */
if(!markit)
{
        if(phi>(PI/2.0))
        {
                markit = 1;
        }
}

/* integrate around each ring */
rphi = r_1[1]*sin(phi);

/* differential volume element */
dV = r_1[1]*rphi*dPhi*dTheta*dR;

for(theta=0.0; theta<(2.0*PI); theta+=dTheta)
{
        /* switch to cartesian for calculations */
        xx = rphi*cos(theta);
        yy = rphi*sin(theta);


        /* half energy of ambient field, no summation.  Irrespective of whether or not
        central particle actually contributes */
        dE = slope*dV;
        if(xx<=0.0)
        {
                *Ecenter1 += dE;
        }else{
                if(r_1[1]<=(d/2.0))
                {
                        *Ecenter2 += dE;
                }
        }

        addit = TRUE;
        if(x_symmetric && (xx>d/2.0))
        {
                addit = FALSE;
        }

        /* define squares of radii */
        for(n=2; n<=numpar; n++)
        {
                r_2[n] = ((xx+xcd[n])*(xx+xcd[n])) + ((yy+ycd[n])*(yy+ycd[n]))
                        + ((zz+zcd[n])*(zz+zcd[n]));
        }

        if(addit)
        {
                /* this point is within one of the particle cores */
                incore = FALSE;
                for(n=2; n<=numpar; n++)
                {
                        if(r_2[n]<(coresizec[n]*coresizec[n]))
                        {
                                incore = TRUE;
                                break;
                        }
                }

                /* interstitial space ---------------------------------------------------- */
```

```
if(!incore)
{
        /* calculate individual radii and energy differential */
        for(n=2; n<=numpar; n++)

        {
                r_1[n] = sqrt(r_2[n]);
                r_5[n] = r_2[n]*r_2[n]*r_1[n];
        }

        /* calculate summed deflection and slope vector components */
        tx = ty = tz = 0.0;
        for(n=1; n<=numpar; n++)
        {
                B = -KKs*chargec[n]/r_5[n];
                tx += B*(xx+xcd[n]);
                ty += B*(yy+ycd[n]);
                tz += B*(zz+zcd[n]);
                /* individual particle fields */
                dEfield = r_1[n]*fabs(B);
                /* total nonsuperimposed energy used to calculate retension */
                *totalnonsuperE += (dEfield*dV);
                Efield[n] += (dEfield*dV);
        }

        /* this performs the absolute value automatically */
        slopevector = sqrt((tx*tx) + (ty*ty) + (tz*tz));
        dEsum = slopevector*dV;

        /* total energy in noncore regions */
        *Esum += dEsum;
        *pteng = (*pteng + 1);

        /* central particle's immediate interstitual energy */
        if(r_1[1]<=d_centralsamplesize)
        {
                *Ecentralsample += dEsum;
        }

        /* mark interstitial space on the z=0 plane */
        if(ggrafon)
        {
                if(markit==1)
                {
                        ProcessGraphPoint(COLORGREEN, xx, yy, FALSE);
                }
        }
}else{
        /* another cross-check of algorithm; volume of surrounding cores */
        *corevolume += dV;
        *ptvol = (*ptvol + 1);
        if(ggrafon)
        {
                /* markit not used here in order to demonstrate 3-d effect */
                for(n=2; n<=numpar; n++)
                {
                        if((colorc[n]>0) && (colorc[n]<MAX_PARCOLOR))
                        {
                                /* color the core it is in */
                                if(r_2[n]<(coresizec[n]*coresizec[n]))
                                {
                                        parcolor = colormap[colorc[n]];
```

```
                                                                ProcessGraphPoint(parcolor,
                                                                        xx, yy, FALSE);
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }

        /* cancel marker so that only z = 0 is marked */
        if(markit==1)
        {
                markit = 2;
        }
}

/* increment radius and test for finished */
r_1[1] += dR;
if(r_1[1]>=rscan)
{
        break;
}

/* optimize for flatter topology */
if(r_1[1]>rlimit)
{
        dR = corrdR/corr;
}

/* display update less so as well as last converge value ------------------------------------ */
if(KeyDown(VK_ESCAPE))
{
        abortit = TRUE;
        break;
}
timdel = AgeNow() - timstart;
AgeTxt(timdel, curtim);
if(*edelta==0.0)
{
        strcpy(rtxt, "( --- )");
        rcolor = COLORBLACK;
}else{
        if(*edelta>0.0)
        {
                strcpy(rtxt, "(R too small)");
                rcolor = COLORDECREASE;
        }else{
                strcpy(rtxt, "(R too large)");
                rcolor = COLORINCREASE;
        }
}
rstatus = 100.0*r_1[1]/rscan;
Gframe(MESSAGEFRAME);
if(tottim[0])
{
        sprintf(af, "%s(c%ld-a%ld): %7.6f %%   Rc:%7.5e   d:%7.5e   Recess:%4.2f   Erez:%6.0f   \
                Con:%3.1f >> %3.1f %s %s (%s)",
                note_text, numpar, atnum, rstatus, rc, d, *recessrate, *ereso,
                lastlastconverge, *converge, rtxt, curtim, tottim);
}else{
        sprintf(af, "%s(c%ld-a%ld): %7.6f %%   Rc:%7.5e   d:%7.5e   Recess:%4.2f   Erez:%6.0f   \
					Con:%3.1f >> %3.1f %s %s",
					note_text, numpar, atnum, rstatus, rc, d, *recessrate, *ereso,
					lastlastconverge, *converge, rtxt, curtim);
		}
		Gbold(1);
		Gjust(TA_CENTER);
		Gtextframesmall(MESSAGEFRAME, rcolor, "", af, 21);
	}

	/* double to get entire field since only x<0 was summed */
	*Ecenter1 = 2.0*(*Ecenter1);
	*Ecenter2 = 2.0*(*Ecenter2);
	if(x_symmetric)
	{
		*Esum = 2*(*Esum);
		*totalnonsuperE = 2.0*(*totalnonsuperE);
	}

	/* reset to values from last valid pass */
	if(abortit)
	{
		*edelta = oldEdelta;
		*Ecenter1 = oldEcenter1;
		*Ecenter2 = oldEcenter2;
		*Esum = oldEsum;
		*Ecentralsample = oldEcentralsample;
		*totalnonsuperE = oldtotalnonsuperE;
		*corevolume = oldcorevolume;
		*dRfin = olddRfin;
	}else{
		/* check for the difference between total input and integrated slope */
		*edelta = etotal - (*Esum);

		/* test and adjust for nonconvergence */
		*converge = *edelta/(etotal/(*ereso));

		/* autocorrecting recessionrate and energy resolution for nonconvergence,
		although seldom the problem */
		if(*lastconverge!=0.0)
		{
			/* not converging */
			if(fabs(*converge)>fabs(*lastconverge))
			{
				switch(conmode)
				{
					/* end on first divergence if conmode contains a 2 */
					case 3:
					case 2:
						*lastconverge = *converge;
						*converge = 0.0;
						break;

					/* adjust for divergence and continue if exit bit (2) not set */
					case 0:
					case 1:
						if(*recessrate>0.01)
						{
							(*recessrate) =  (*recessrate)/2.0;
						}
						if(vary_ereso)
						{
							if(*ereso>20000.0)
```

```
                            {
                                *ereso =  (*ereso)/2.0;
                            }
                        }
                        break;
                }
            }else{
                switch(conmode)
                {
                    /* converging too slowly, extrapolate recessrate if fixed bit (1) not set */
                    case 2:
                    case 0:
                        convergerate = (*lastconverge - *converge)/(*lastconverge);
                        *recessrate = *recessrate/convergerate;
                        if(*recessrate>maxrecess)
                        {
                            *recessrate = maxrecess;
                        }
                        break;
                }
            }
        }

        /* if divergence marker was not set */
        if(*converge!=0.0)
        {
            lastlastconverge = *lastconverge;
            *lastconverge = *converge;
        }

        /* keep track of radial differential */
        *dRfin = dR;

        /* save values from valid pass */
        oldEdelta = *edelta;
        oldEcenter1 =  *Ecenter1;
        oldEcenter2 =  *Ecenter2;
        oldEsum = *Esum;
        oldEcentralsample = *Ecentralsample;
        oldtotalnonsuperE = *totalnonsuperE;
        oldcorevolume = *corevolume;
        olddRfin = *dRfin;
    }

    /* just end this routine without existing main program */
    KeyRelease(VK_ESCAPE);

    return(abortit);
}

/*------------------------------------------------------------------------------------------------------------------- */
/*                                                                                                                   */
/*                                          CORE RECESSION PROFILE                                                   */
/*                                                                                                                   */
/*------------------------------------------------------------------------------------------------------------------- */
void InitProfileFile(long reso, long enreso, char converge, char fixed)
{
    char    af[200];

    /* generate particle file */
    FileOutLine(DEUTERIUM_FILE, "++ Recession Profile");
    FileAddLine(DEUTERIUM_FILE,
		"Reso	Ereso	d	off	scan	conrate recess	rc	corlim	corpow");
	sprintf(af, "|%-5.0ld c%c|%-5.0ld%c |	|	|1000	|	|1.0f	|	|1.333	|2.0",
		reso, converge, enreso, fixed);
	FileAddLine(DEUTERIUM_FILE, af);
	FileAddLine(DEUTERIUM_FILE, "");
	FileAddLine(DEUTERIUM_FILE,
		"x	y	z	charge	fixcore	energy	color");
	/* 0,0,0 has to be first particle */
	FileAddLine(DEUTERIUM_FILE,
		"| 0.0	|0.0	|0.0	|1	|0	|1.0	|5");
	FileAddLine(DEUTERIUM_FILE,
		"|-1.0	|0.0	|0.0	|1	|0	|1.0	|5");
	FileOutLine(PROFILE_FILE,
		"Trace		Scaling		Differential	x		y");
	FileDelete(PROFIL_TEMPFILE);
}

void WriteProfileLine(double d, double deltar)
{
	char	af[200];

	sprintf(af, "1						%8.7e	%8.7e", d, deltar);
	FileAddLine(PROFILE_FILE, af);
}

/* generate a function of core recession versus particle separation, spherical coordinates */
void CoreRecessionProfile(void)
{
	WFILE	chan;
	char	af[300], tim[200], converge, fixed;
	double	dstep=0.5, dlow=1.0, dhigh=2.0, d, dd, rc, deltar, rscan=1000.0, dstepgrowth=1.0;
	double	timstart, timdel;
	long	reso=200, enreso=50000;

	converge = ' ';
	fixed = 'f';

	if(GetText(hWndMain, "Spherical Recession Profile Starting d (in units of Rp)"))
	{
		dlow = atof(note_text);
		if(GetText(hWndMain, "Spherical Recession Profile Ending d (in units of Rp)"))
		{
			dhigh = atof(note_text);
			sprintf(af, "Distance step (in units of Rp) (v=variable) [%3.1f]", dstep);
			if(GetText(hWndMain, af))
			{
				if(atof(note_text)>0.0)
				{
					dstep = atof(note_text);
					if(StrString(note_text, "v", FALSE))
					{
						if(GetText(hWndMain, "Distance step growth factor"))
						{
							if(atof(note_text)>1.0)
							{
								dstepgrowth = atof(note_text);
							}
						}
					}
				}
			}
			sprintf(af, "Rscan [%6.2f]", rscan);
```

```
if(GetText(hWndMain, af))
{
        if(atof(note_text)>0)
        {
                rscan = atof(note_text);
        }
}
sprintf(af, "Resolution [%ld]", reso);
if(GetText(hWndMain, af))
{
        if(atol(note_text)>0)
        {
                reso = atol(note_text);
        }
}
sprintf(af, "Energy Resolution [%ld]", enreso);
if(GetText(hWndMain, af))
{
        if(atol(note_text)>0)
        {
                enreso = atol(note_text);
        }
}
if(GetText(hWndMain, "Fixed [Y]"))
{
        strupr(note_text);
        if(note_text[0]=='N')
        {
                fixed = ' ';
        }
}
if(GetText(hWndMain, "Converge [Y]"))
{
        strupr(note_text);
        if(note_text[0]=='N')
        {
                converge = 'o';
        }
}
if((reso<=20000)&&(dlow>0.0)&&(dlow<=1000.0)&&(dhigh>=dlow)&&(dhigh<=1000.0))
{
        if((dstep>0.0)&&(dstep<dhigh))

        {
                InitProfileFile(reso, enreso, converge, fixed);

                timstart = AgeNow();
                for(d=dlow; d<=dhigh; d+=dstep)
                {
                        /* use best initial guess for particle radius */
                        dd = d*Rp;
                        deltar = (2.0*0.084*Rp*Rp)/dd - ((0.081*Rp*Rp*Rp*Rp)/(dd*dd*dd));
                        rc = Rp - deltar;
                        rc /= Rp;

                        /* completion duration */
                        timdel = AgeNow() - timstart;
                        AgeTxt(timdel, tim);
                        if(CoreRecession(DEUTERIUM_FILE, PROFIL_TEMPFILE,
                                TRUE, FALSE, d, rc, rscan, d, 0.0, tim))
                        {
                                break;
                        }

                        dstep *= dstepgrowth;
                    }

                    /* completion duration */
                    timdel = AgeNow() - timstart;
                    AgeTxt(timdel, tim);

                    /* compile profile data */
                    chan = FileOpenRead(PROFIL_TEMPFILE);
                    if(FileOK(chan))
                    {
                        while(FileInput(chan, af, sizeof(af)))
                        {
                            if(StrString(af, "DelR:", TRUE))
                            {
                                d = atof(&af[11]);
                                deltar = atof(&af[36]);
                                WriteProfileLine(d, deltar);
                            }
                        }
                        FileClose(chan);
                    }

                    Gframe(MESSAGEFRAME);
                    sprintf(af, "Total Elapsed Time: %s", tim);
                    Gbold(1);
                    Gjust(TA_CENTER);
                    Gtextframesmall(MESSAGEFRAME, COLORBLUE, "", af, 21);
                    FileAddLine(PROFIL_TEMPFILE, "");
                    FileAddLine(PROFIL_TEMPFILE, af);
                }
            }
        }
    }
}
```