```
Clear["Global`*"];
(* Do a simple Method of Characteristics MOC demonstation *)
(* Many thanks to barnhart@wri.com for LensLab's
    inspirations ------------------------------------------------------------ *)
(* basic
  definitions ------------------------------------------------------------------
                    ---------------------------------- *)
Options[Rays] = {NumberOfRays → 2};
Options[DrawSystem] = {PlotType → 3 D};
Options[PolygonalNode] = {NodeType → ich, NodeID → Fuel};
FilterOptions[command_Symbol, opts___] :=
  Module[{keywords = First /@ Options[command]},
   Sequence @@ Select[{opts}, MemberQ[keywords, First[#]] &]];
(* the surfaces are parametrized as (f[y,z],y,z)-vectors ;
the surfaces can be rotated around the z-axis *)
(* fx, fy, fz are the respective coordinate
 projections of the surface onto the x,y and z axis *)
fx[f_, x_, theta_] := Function[{tx, tz},
   Evaluate[Chop[N[Cos[theta Degree] f[tx, tz] - Sin[theta Degree] tx + x]]]];
fy[f_, y_, theta_] := Function[{tx, tz},
   Evaluate[Chop[N[Sin[theta Degree] f[tx, tz] + Cos[theta Degree] tx + y]]]];
fz[f_, z_, theta_] := Function[{tx, tz},
   Evaluate[Chop[N[Sin[theta Degree] f[tx, tz] + Cos[theta Degree] tz + z]]]];

InverseTan[y_, x_] := Module[{tempAngle},
   tempAngle = If[x ≠ 0, N[Chop[ArcTan[Abs[y / x]] / Degree]], If[y > 0, 90, -90]];
   If[x > 0 && y > 0, tempAngle, If[x < 0 && y < 0, -180 + tempAngle,
     If[x < 0 && y ≥ 0, -180 - tempAngle, If[x ≠ 0, -tempAngle, tempAngle]]]]];

(* define set of horizontal
 rays ------------------------------------------------------------------------
        ---------------- *)
HorizontalParallelRays[{xpos_, ypos_, zpos_}, {rayanglex_, rayanglez_},
   width_, options___] := Module[{rays, h, tabs},
   rays = N[NumberOfRays /. {options} /. Options[Rays]]; If[Cos[rayanglex Degree] ≠ 0,
    tabs = Table[Chop[N[Ray[{xpos - h Sin[rayanglex Degree] ,
          ypos + h Cos[rayanglex Degree] , zpos}, {rayanglex, rayanglez}, vacuum]]],
      {h, -width / (2), width / (2), width / ((rays - 1))}],
    tabs = Table[Chop[N[Ray[{xpos - h Sin[rayanglex Degree] ,
          ypos + h Cos[rayanglex Degree] , zpos}, {rayanglex, rayanglez}, vacuum]]],
      {h, -width / (2), width / (2), width / (rays - 1)}]]; tabs
  ];

CenteredParallelRays[{xposcenter_, yposcenter_, zposcenter_},
   {rayanglex_, rayanglez_}, width_, leng_, options___] := HorizontalParallelRays[
   {xposcenter - leng * Cos[rayanglez Degree] * Cos[rayanglex Degree],
    yposcenter - leng * Cos[rayanglez Degree] * Sin[rayanglex Degree], zposcenter -
     leng * Sin[rayanglez Degree]}, {rayanglex, rayanglez}, width, options];
```

```
(* define outer boundary of system --------
 ------------------------------------------------------------------------------
  *)
Boundary[{x1_, y1_, z1_}, {x2_, y2_, z2_}] :=
  PlacedBoundary[{fx[0 &, x2, 0], fy[0 &, (y2 + y1) / 2, 0], fz[0 &, 0, 0]},
   {fx[0 &, (x2 + x1) / 2, 90], fy[0 &, y2, 90], fz[0 &, 0, 0]},
   {fx[0 &, x1, 0], fy[0 &, (y2 + y1) / 2, 0], fz[0 &, 0, 0]},
   {fx[0 &, (x2 + x1) / 2, 90], fy[0 &, y1, 90], fz[0 &, 0, 0]}, 0 &, 0 &,
   0 &, 0 &, 0, 90, 0, 90, y2 - y1, x2 - x1, y2 - y1, x2 - x1, z2 - z1];

(* define a polygon ; fuel rods are simulated as many-
 faced polygons ------------------------------------------------------ *)

Place[PolygonalNode[NumberOfSides_, Diameter_, Height_],
   {x_, y_, thetax_}, options___] :=
  PlacedPolygonalNode[NumberOfSides, Diameter, Height, {x, y, thetax}, options];
(* propagate a ray through a single sourface --------
 ------------------------------------------------------------------------ *)
(* for every ray an interaction with every surface is checked;
if there is a hit with a surface a new ray is created,
   otherwise the original ray is returned *)
Off[MainSolve::elist];
propagateRay[{}, _] := {};
propagateRay[{Ray[], _}, _] := {};
propagateRay[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, rayid_], index_},
    Surface[{fx_, fy_, fz_}, fraw_, {lensxtheta_, lensztheta_},
     {width_, height_}, indexL_, indexR_, componentType_, surID_]] :=
  Module[{tx, tz, txx, tzz, tl, t1, tl1, rx2, ry2, rz2, thetax2, thetaz2,
    df, thetaxsurface, thetazsurface, ignored, intersections, typus},
   (*
   If[(componentType===Absorptive&&Abs[thetax1-lensxtheta]==90),
     Return[{Ray[{rx1,ry1,rz1},{thetax1,thetaz1},stopped],index}]];
   *)
   intersections =
    Chop[N[{tl, tx, tz} /. Solve[Chop[N[{rx1 + tl Cos[thetax1 Degree] == fx[tx, tz],
            ry1 + tl Sin[thetax1 Degree] == fy[tx, tz],
            rz1 + tl Sin[thetaz1 Degree] == fz[tx, tz]}]], {tl, tx, tz}]]];

   If[width == Abs[width], intersections =
     Select[intersections, Apply[Function[{tl, tx, tz}, (Abs[tx] ≤ width / 2) &&
           (Abs[tz] ≤ height / 2) && (Im[tx] == 0) && (Im[tl] == 0) && (Im[tz] == 0) &&
           (tl > 0) && (Abs[N[rx1 + tl Cos[thetax1 Degree] - fx[tx, tz]]] < 10^-5) &&
           (Abs[N[ry1 + tl Sin[thetax1 Degree] - fy[tx, tz]]] < 10^-5) &&
           (Abs[N[rz1 + tl Sin[thetaz1 Degree] - fz[tx, tz]]] < 10^-5)], #] &]
    (*else*), intersections = Select[intersections, Apply[Function[{tl, tx, tz},
          (tx^2 + tz^2 ≤ (width / 2)^2) && (Im[tx] == 0) && (Im[tl] == 0) && (Im[tz] == 0) &&
           (tl > 0) && (Abs[N[rx1 + tl Cos[thetax1 Degree] - fx[tx, tz]]] < 10^-5) &&
           (Abs[N[ry1 + tl Sin[thetax1 Degree] - fy[tx, tz]]] < 10^-5) &&
           (Abs[N[rz1 + tl Sin[thetaz1 Degree] - fz[tx, tz]]] < 10^-5)], #] &]];
```

```
    If[Length[intersections] == 0,
     Return[{Ray[{rx1, ry1, rz1}, {thetax1, thetaz1}, rayid], index}]];

    {tl1, tx, tz} = First[Sort[intersections]];

    rx2 = N[rx1 + tl1 Cos[thetax1 Degree]];
    ry2 = N[ry1 + tl1 Sin[thetax1 Degree]];
    rz2 = N[rz1 + tl1 Sin[thetaz1 Degree]];


    (* If[componentType===Absorptive,
      Return[{Ray[{rx2,ry2,rz2},{I,I}],index}]]; *)

    (* {Ray[{rx1+1,ry1+1,rz1},{thetax1,thetaz1}],indexR}
       {Ray[{rx2,ry2,rz2},{thetax2,thetaz2}],indexR}
     ]; *)
    {Ray[{rx2, ry2, rz2}, {thetax1, thetaz1}, surID], index}
  ];

(* turn components into surfaces ----------
 ------------------------------------------------------------------------ *)
componentToSurfaces[Null] := {};

componentToSurfaces[
    PlacedBoundary[f1_, f2_, f3_, f4_, fraw1_, fraw2_, fraw3_, fraw4_, theta1_,
     theta2_, theta3_, theta4_, width1_, width2_, width3_, width4_, height_]] :=
  {Surface[f1, fraw1, {theta1, 0}, {width1, height}, 0 &, 1 &, Absorptive, vacuum],
    Surface[f2, fraw2, {theta2, 0}, {width2, height}, 0 &, 1 &, Absorptive, vacuum],
    Surface[f3, fraw3, {theta3, 0}, {width3, height}, 0 &, 1 &, Absorptive, vacuum],
    Surface[f4, fraw4, {theta4, 0}, {width4, height}, 0 &, 1 &, Absorptive, vacuum]};

componentToSurfaces[PlacedPolygonalNode[NumberOfSides_,
     Diameter_, Height_, {x_, y_, thetax_}, options___]] :=
  Module[{subwidth, subtheta, offsettheta, sourcex, sourcey, sourcez,
     newthetax, unitangle, x1, x2, y1, y2, thetaT, thetaT1,
     thetaT2, xpos, xposr, ypos, PrimeAngle, typus, mix},
    subwidth = Abs[Chop[N[Diameter Sin[Pi / NumberOfSides]]]];
    {sourcex, sourcey, sourcez} = Source;
    unitangle = N[360 / NumberOfSides];
    typus = (NodeID /. {options} /. Options[PolygonalNode])[[1]];
    PrimeAngle[theta_] :=
     If[theta > 180, theta - 360, If[theta < -180, theta + 360, theta]];
    offsettheta = Chop[N[Round[InverseTan[(sourcey - y), (sourcex - x)] / unitangle]
        unitangle]];
    newthetax = Chop[N[thetax - Round[thetax / unitangle] unitangle]];

    x1 := Diameter Cos[(thetaT - unitangle / 2) Degree] / 2;
    y1 := Diameter Sin[(thetaT - unitangle / 2) Degree] / 2;
```

```
    x2 := Diameter Cos[(thetaT + unitangle / 2) Degree] / 2;
    y2 := Diameter Sin[(thetaT + unitangle / 2) Degree] / 2;
    xpos := Chop[N[(x2 + x1) / 2]] + x; xposr := -Chop[N[(x2 + x1) / 2]] + x;
    ypos := Chop[N[(y2 + y1) / 2]] + y;

    mix = Flatten[{thetaT = PrimeAngle[newthetax + offsettheta];
        If[EvenQ[NumberOfSides],
         {Surface[{fx[0 &, xpos, thetaT], fy[0 &, ypos, thetaT], fz[0 &, 0, 0]},
           0 &, {thetaT, 0}, {subwidth, Height}, 1 &, 1 &, Refractive, typus],
          Surface[{fx[0 &, xposr, thetaT], fy[0 &, ypos, thetaT], fz[0 &, 0, 0]},
           0 &, {thetaT, 0}, {subwidth, Height}, 1 &, 1 &, Refractive, typus]},
         Surface[{fx[0 &, xpos, thetaT], fy[0 &, ypos, thetaT], fz[0 &, 0, 0]},
          0 &, {thetaT, 0}, {subwidth, Height}, 1 &, 1 &, Refractive, typus]],
        Table[{thetaT1 = newthetax - subtheta + offsettheta;
          thetaT2 = newthetax + subtheta + offsettheta;
          If[Abs[thetaT1 - offsettheta] < Abs[thetaT2 - offsettheta],
           thetaT = PrimeAngle[thetaT1];
           thetaT2 = PrimeAngle[thetaT2], thetaT = PrimeAngle[thetaT2];
           thetaT2 = PrimeAngle[thetaT1]];
          Surface[{fx[0 &, xpos, thetaT], fy[0 &, ypos, thetaT], fz[0 &, 0, 0]},
           0 &, {thetaT, 0}, {subwidth, Height}, 1 &,
           1 &, Refractive, typus], thetaT = thetaT2;
          Surface[{fx[0 &, xpos, thetaT], fy[0 &, ypos, thetaT], fz[0 &, 0, 0]}, 0 &,
           {thetaT, 0}, {subwidth, Height}, 1 &, 1 &, Refractive, typus]}, {subtheta,
          unitangle, Round[(NumberOfSides - 4) / 2 + 1.001] unitangle, unitangle}]}];
    mix];


(* render components  ------------------------------------------------------------
 ---------------------------------- *)
renderComponent[PlacedBoundary[{f1x_, f1y_, f1z_}, {f2x_, f2y_, f2z_}, f3_, _, _,
     _, _, _, _, _, _, _, _, _, dy_, dx_, dz_]] := Module[{x2, y2}, x2 = f1x[0, 0];
    y2 = f2y[0, 0]; Return[{Hue[0.8], Point[{x2 - dx, y2 - dy, -dz / 2}],
       Point[{x2 - dx, y2 - dy, +dz / 2}], Line[{{x2 - dx, y2 - dy, 0}, {x2, y2 - dy, 0}}],
       Line[{{x2, y2 - dy, 0}, {x2, y2, 0}}], Line[{{x2, y2, 0}, {x2 - dx, y2, 0}}],
       Line[{{x2 - dx, y2 - dy, 0}, {x2 - dx, y2, 0}}]}];];

renderComponent[Null] := Return[{}];
renderComponent[PlacedPolygonalNode[NumberOfSides_, Diameter_, Height_,
     {x_, y_, thetax_}, options___]] := Module[{subtheta, topside, bottomside, box},
    box[theta_] := Module[{outline, x1, y1, x2, y2, ret},
       x1 = N[Diameter Cos[(theta - 180 / NumberOfSides) Degree] / 2 + x];
       y1 = N[Diameter Sin[(theta - 180 / NumberOfSides) Degree] / 2 + y];
       x2 = N[Diameter Cos[(theta + 180 / NumberOfSides) Degree] / 2 + x];
       y2 = N[Diameter Sin[(theta + 180 / NumberOfSides) Degree] / 2 + y];
       outline = Join[{{x1, y1, -Height / 2}, {x2, y2, -Height / 2},
            {x2, y2, Height / 2}, {x1, y1, Height / 2}, {x1, y1, -Height / 2}}] // N;
       typus = (NodeType /. {options} /. Options[PolygonalNode])[[1]];
       If[typus == Moderator,
```

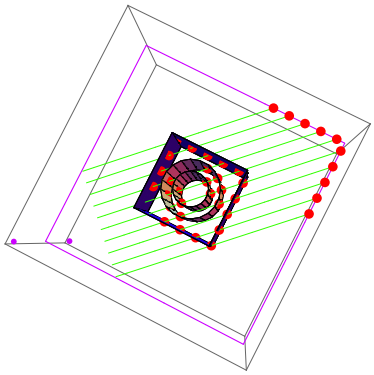
```
      Return[{Hue[0.7], Polygon[outline], GrayLevel[0], Line[outline]}]];
     If[typus == Fuel, Return[{GrayLevel[0.9], Polygon[outline],
        GrayLevel[0], Line[outline]}]];
    ];

   Join[Flatten[
     {Table[box[subtheta + thetax], {subtheta, 0, 360, 360 / NumberOfSides}]}]]];

renderRay[rays_] := Map[twoRaysToLine, Partition[rays, 2, 1]];

twoRaysToLine[{{}, _}] := {};
twoRaysToLine[{_, {}}] := {};
twoRaysToLine[{{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, _], _},
    {Ray[{rx2_, ry2_, rz2_}, {thetax2_, thetaz2_}, _], _}}] :=
  {Hue[0.3], Line[{{rx1, ry1, rz1}, {rx2, ry2, rz2}}], Hue[1],
   AbsolutePointSize[5], Point[{rx2, ry2, rz2}]};

(* do the ray
 tracing ------------------------------------------------------------------------
           -------------------- *)
(* this is the main ray propagation
  machine: the function "propagateRay" is acting on a single ray and is testing
    if the original ray or its translated ray
    has an interaction with any surface
    FoldList[f,x,{a,b,c,d}]= {x,f[x,a],f[f[x,a],b],
  f[f[f[x,a],b],c],f[f[f[f[x,a],b],c],d]} *)
(* we need to order the surfaces so that the nearest
 surfaces are treated first *)

Abstand[position_, Surface[{fx_, fy_, fz_}, fraw_, {lensxtheta_, lensztheta_},
    {width_, height_}, indexL_, indexR_, componentType_, surID_]] :=
  Module[{ref = {fx[0, 0], fy[0, 0], fz[0, 0]}}, Norm[position - ref]
  ];

Abstand[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, _], _},
   {Ray[{rx2_, ry2_, rz2_}, {thetax2_, thetaz2_}, _], _}] :=
  Module[{}, Norm[{rx1, ry1, rz1} - {rx2, ry2, rz2}]
  ];

FindNearestSurfacePos[position_, sure_] :=
  Module[{mix = Table[{}, {0}], pos = position, l = Length[sure], k, d},
   For[k = 0, k < l, k++, d = Abstand[position, sure[[k + 1]]];
    mix = AppendTo[mix, d];]; Ordering[mix, 1][[1]]
  ];

OrderSurfaces[position_, sure_] :=
  Module[{mix = Table[{}, {0}], surf = sure, l = Length[sure], j, k},
   For[k = 0, k < l, k++, j = FindNearestSurfacePos[position, surf];
    mix = AppendTo[mix, Take[surf, {j}]]; surf = Drop[surf, {j}];];
```

```
    mix];

doRay[Ray[{positionx_, positiony_, positionz_}, {anglex_, anglez_}, id_],
    surfaces_] := Module[{mix, sure = surfaces},
    sure = Flatten[OrderSurfaces[{positionx, positiony, positionz}, sure]];
    mix = FoldList[propagateRay[#1, #2] &,
      {Ray[{positionx, positiony, positionz}, {anglex, anglez}, id], 1 &}, sure];
    mix];

(* accumulate all objects, propagate rays and request 3D
 plot of everything ------------------------------------ *)
DrawSystem[rays_, components_, options___] :=
  Module[{placedComponents, placedSurfaces, placedRays, plottype},
    Source = {0, 0, 0};
    ScreenPoints = {};
    placedComponents = Flatten[components];
    placedSurfaces = Flatten[Map[componentToSurfaces, placedComponents]];
    placedRays = TallyAll[Map[doRay[#, placedSurfaces] &, Flatten[rays]]];
    (* print tracked rays Print[TallyAll[placedRays]]; *)
     graphicaloutput = {Thickness[0.003], Map[renderComponent, placedComponents],
      Thickness[0.0005], Map[renderRay, placedRays]};
    (*  graphicaloutput={Thickness[0.003],
       Map[renderComponent,placedComponents]}; *)

    ShowSystem[options]];

(* accumulate all objects, propagate rays and return tracks  -----------------
 ----------------------------------- *)
TrackSystem[rays_, components_] :=
  Module[{placedComponents, placedSurfaces, placedRays},
    Source = {0, 0, 0};
    ScreenPoints = {};
    placedComponents = Flatten[components];
    placedSurfaces = Flatten[Map[componentToSurfaces, placedComponents]];
    placedRays = Map[doRay[#, placedSurfaces] &, Flatten[rays]];
    TallyAll[placedRays]];

(* do a 3D plot of the system ----------
 ------------------------------------------------------------------------- *)
ShowSystem[options___] := Module[{plottype, c1, c2, opts},
    plottype = PlotType /. {options} /. Options[DrawSystem];
    opts = FilterOptions[Graphics3D, options];
    Show[Graphics3D[graphicaloutput],
     opts, AspectRatio → Automatic, PlotRange → All]];

(* tally MOCs -----------
 ------------------------------------------------------------------------------
                                          ------------ *)
GetRayCoord[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, rayid_], index_}] :=
```

```
  {rx1, ry1, rz1};
GetRayID[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, rayid_], _}] :=
  Module[{}, rayid];

TallyMe[liste_] := Module[{k = Length [liste], r1, r2, lis = liste, l, j},
   For[l = 0, l < k, l++, r1 = lis[[l + 1]];
    For[j = l + 1, j < k, j++, r2 = lis[[j + 1]];
     If[GetRayCoord[r1] == GetRayCoord[r2], lis = Drop[lis, {j}]; k = k - 1; j = j - 1];
    ];
   ];
   lis];

TallyAll[liste_] := Module[{k = Length [liste], lis = liste, r1, l},
   For[l = 0, l < k, l++,
    r1 = TallyMe[lis[[l + 1]]]; lis = ReplacePart[lis, (l + 1) → r1];
   ];
   lis];

(* calculate angular flux in regions MOCs -----------
 ------------------------------------------------------------------ *)
(* as an example we take a 1-energy-group structure; *)
(* store average flux in the following table for moderator IDm1,
fuel region 1 IDf1 and fuel region 2 IDf2 *)
GlobalAverageFluxStore = {4 π, 4 π, 4 π}; (* 0th order guess *)
(* store angular flux in the following table for moderator IDm1,
fuel region 1 IDf1 and fuel region 2 IDf2;
   we will consider 8 distinct directions
 therefore every region has 8 inital entries;
   each direction has unit weight *)
GlobalAngularFluxStore = {{1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8},
   {1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8},
   {1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8, 1 / 8}}; (* 0th order guess *)

GetTotalXS[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, rayid_], _}] :=
  Module[{XS = 0},
   If[rayid === vacuum, XS = 0.0]; If[rayid === IDm1, XS = 0.049];
   If[rayid === IDf1, XS = 0.026]; If[rayid === IDf2, XS = 0.026];
   XS];

GetScatterXS[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, rayid_], _}] :=
  Module[{XS = 0},
   If[rayid === vacuum, XS = 0.0]; If[rayid === IDm1, XS = 0.045];
   If[rayid === IDf1, XS = 0.005]; If[rayid === IDf2, XS = 0.005];
   XS];

GetnuBarXS[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, rayid_], _}] :=
  Module[{XS = 0},
   If[rayid === vacuum, XS = 0.0]; If[rayid === IDm1, XS = 0.0];
   If[rayid === IDf1, XS = 0.07]; If[rayid === IDf2, XS = 0.07];
```

```
    XS];

GetFlux[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, rayid_], _}] :=
  Module[{XS = 0},
   If[rayid === vacuum, XS = 0.0];
   If[rayid === IDm1, XS = GlobalAverageFluxStore[[1]]];
   If[rayid === IDf1, XS = GlobalAverageFluxStore[[2]]];
   If[rayid === IDf2, XS = GlobalAverageFluxStore[[3]]];
   XS];

GetAngularFlux[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, rayid_], _},
   direct_] := Module[{XS = 0},
   If[rayid === vacuum, XS = 0.0]; If[rayid === IDm1,
    XS = GlobalAngularFluxStore[[1]][[direct]]];
   If[rayid === IDf1, XS = GlobalAngularFluxStore[[2]][[direct]]];
   If[rayid === IDf2, XS = GlobalAngularFluxStore[[3]][[direct]]];
   XS];

CheckInside[{Ray[{rx1_, ry1_, rz1_}, {thetax1_, thetaz1_}, rayid1_], _},
   {Ray[{rx2_, ry2_, rz2_}, {thetax2_, thetaz2_}, rayid2_], _}] := Module[{wo = 0},
   If[rayid1 == rayid2, wo = 2]; If[rayid1 === vacuum, wo = 1];
   If[rayid2 === vacuum, wo = 0];
   (* given the combination of material IDs from ray1 and ray2 we
    have to tell the code within which region the track is moving,
   i.e. entering or leaving the material region of ray 1 *)
   If[rayid1 === IDm1 && rayid2 == IDf1, wo = 1];
   If[rayid2 === IDm1 && rayid1 == IDf1, wo = 0];
   If[rayid1 === IDf1 && rayid2 == IDf2, wo = 1];
   If[rayid2 === IDf1 && rayid1 == IDf2, wo = 0];
   wo];

IntegrateAngularFlux[track_, direct_] :=
  Module[{mix = Table[{}, {0}], segments = Partition[track, 2, 1], l,
    sourceQ, sourceF, d, Xst, phi, phi0, phi1, phiAve, ray1, ray2, wo},
   (* Print[segments]; *) l = Length[segments];
   For[k = 0, k < l, k++,
    ray1 = segments[[k + 1]][[1]]; ray2 = segments[[k + 1]][[2]];
    wo = CheckInside[ray1, ray2];
    If[wo > 0, Xst = GetTotalXS[ray1]; phi = GetFlux[ray1]; sourceQ =
      phi * GetScatterXS[ray1] / (4 π); sourceF = phi * GetnuBarXS[ray1] / (4 π),
     Xst = GetTotalXS[ray2]; phi = GetFlux[ray2]; sourceQ =
      phi * GetScatterXS[ray2] / (4 π); sourceF = phi * GetnuBarXS[ray2] / (4 π) ];
    (* when we start in vacuum or do the first crossover
     from vacuum to outer boundary we take inital
     condition from boundary conditions otherwise from
     result of last track integral *)
    If[k == 0 || k == 1, phi0 = GetAngularFlux[ray1, direct], phi0 = mix[[k]][[2]]];
    d = Abstand[ray1, ray2];
    phi1 = If[Xst > 0, phi0 * Exp[-Xst * d] +
```

```
        (1 - Exp[-Xst * d]) * (sourceQ + sourceF) / (Xst), phi0 * Exp[-Xst * d]];
    phiAve = If[Xst > 0, NIntegrate[phi0 * Exp[-Xst * x] +
         (1 - Exp[-Xst * x]) * (sourceQ + sourceF) / (Xst), {x, 0, d}],
       NIntegrate[phi0 * Exp[-Xst * x], {x, 0, d}]];
    mix = AppendTo[mix, {phi0, phi1, phiAve, d}];
   ];
   mix
  ];

(* visualize MOCs ------------------
 ------------------------------------------------------------------------------
                                          ---------- *)
DrawSystem[
    {
        CenteredParallelRays[{100, 0, 0}, {45, 0}, 100, 90, NumberOfRays → 10]
    },
    {
       Place[PolygonalNode[4, 100, 100],
   {100, 0, 0}, {NodeType → Moderator, NodeID → IDm1}],
  Place[PolygonalNode[20, 50, 100], {100, 0, 0}, {NodeType → Fuel, NodeID → IDf1}] ,
  Place[PolygonalNode[20, 30, 100], {100, 0, 0}, {NodeType → Fuel, NodeID -> IDf2}] ,
        Boundary[{0, -100, -100}, {200, 100, 100}]
    }, PlotType → 3 D
]
```



```
(* system
 definition ----------------------------------------------------------------
                     ------------------------------------ *)
MySystem[direct_] := Module[{}, {{CenteredParallelRays[{100, 0, 0},
       {direct, 0}, 100, 90, NumberOfRays → 10] },
    { Place[PolygonalNode[4, 100, 100], {100, 0, 0},
       {NodeType → Moderator, NodeID → IDm1}],
         Place[PolygonalNode[20, 50, 100], {100, 0, 0},
       {NodeType → Fuel, NodeID → IDf1}] ,
         Place[PolygonalNode[20, 30, 100], {100, 0, 0},
       {NodeType → Fuel, NodeID -> IDf2}] ,
    Boundary[{0, -100, -100}, {200, 100, 100}] }}];
```

```
(* calculate average angular flux within a region  --------
 ------------------------------------------------------------------ *)
CalcAverageAngularFlux[trackstore_, phistore_, mat_, dir_] :=
  Module[{ndirect = Length[trackstore], nrays = Length[trackstore[[1]]],
    k, l, j, wi, track, ids, ray1, ray2, wo, sumL, sumP, fluss},
   wo = 0;
   k = dir; (* initialize direction *)
   sumP = 0.0; sumL = 0.0;
   For[l = 0, l < nrays, l++, (* sum over number of tracks *)
    track = trackstore[[k]][[l + 1]];
    For[j = 0, j < Length[track] - 1, j++,
     (* sum over number of interaction points per track *)
     ids = GetRayID[track[[j + 1]]]; ray1 = track[[j + 1]]; ray2 = track[[j + 2]];
     wo = CheckInside[ray1, ray2];
     (* Print[ToString[wo]<>" "<>ids<>" "<>GetRayID[track[[j+2]]]];*)
     If[ids === mat,
      If[wo > 0, sumP = sumP + phistore[[k]][[l + 1]][[j + 1]][[3]] / wo;
       sumL = sumL + phistore[[k]][[l + 1]][[j + 1]][[4]] / wo];
      If[wo == 0, sumP = sumP + phistore[[k]][[l + 1]][[j + 0]][[3]];
       sumL = sumL + phistore[[k]][[l + 1]][[j + 0]][[4]]];
     ];
    ];
   ];
   If[sumL > 0, fluss = sumP / sumL, fluss = 0];
   fluss];

(* calculate average flux within a region  ------------
 ------------------------------------------------------------------- *)
CalcAverageFlux[trackstore_, phistore_, mat_] :=
  Module[{ndirect = Length[trackstore], nrays = Length[trackstore[[1]]],
    k, l, j, wi, track, ids, ray1, ray2, wo, fluss, sumL, sumP},
   wi = 1.0 / ndirect; wo = 0; fluss = 0;
   For[k = 0, k < ndirect, k++, (* sum over directions *)
    sumP = 0.0; sumL = 0.0;
    For[l = 0, l < nrays, l++, (* sum over number of tracks *)
     track = trackstore[[k + 1]][[l + 1]];
     For[j = 0, j < Length[track] - 1, j++,
      (* sum over number of interaction points per track *)
      ids = GetRayID[track[[j + 1]]]; ray1 = track[[j + 1]]; ray2 = track[[j + 2]];
      wo = CheckInside[ray1, ray2];
      If[ids === mat,
       If[wo > 0, sumP = sumP + phistore[[k + 1]][[l + 1]][[j + 1]][[3]] / wo;
        sumL = sumL + phistore[[k + 1]][[l + 1]][[j + 1]][[4]] / wo];
       If[wo == 0, sumP = sumP + phistore[[k + 1]][[l + 1]][[j + 0]][[3]];
        sumL = sumL + phistore[[k + 1]][[l + 1]][[j + 0]][[4]]];
      ];
     ];
    ];
    fluss = fluss + wi * sumP / sumL;
```

```
   ];
   fluss];

(* do one global iterations  -------------
 ------------------------------------------------------------------------------
                                          ---- *)
For[iter = 0, iter < 1, iter++,
  (* ------------------------------------------------------------------------
                                          ----------------------------------
                                          -------- *)
  TrackStore = Table[{}, {0}]; PhiStore = Table[{}, {0}];
  (* calculate flux on all segments  ---------------------
   ------------------------------------------------------------------- *)
  For[w = 0, w < 8, w++, (* do 8 directions in steps of 45 degrees *)
   TempStore = Table[{}, {0}];
   mocs = TrackSystem[MySystem[0 + 45 * w][[1]], MySystem[0 + 45 * w][[2]]];
   TrackStore = AppendTo[TrackStore, mocs];
   For[v = 0, v < Length[mocs], v++,
    (* integrate angular flux along each segment of a track *)
    ints = IntegrateAngularFlux[mocs[[v + 1]], w + 1];
    TempStore = AppendTo[TempStore, ints];
   ];
   PhiStore = AppendTo[PhiStore, TempStore];
  ];

  (* update global angular flux store ---------------------
   ------------------------------------------------------------------- *)
  For[w = 0, w < 8, w++,
   GlobalAngularFluxStore[[1]][[w + 1]] =
    CalcAverageAngularFlux[TrackStore, PhiStore, IDm1, w + 1];
   GlobalAngularFluxStore[[2]][[w + 1]] =
    CalcAverageAngularFlux[TrackStore, PhiStore, IDf1, w + 1];
   GlobalAngularFluxStore[[3]][[w + 1]] =
    CalcAverageAngularFlux[TrackStore, PhiStore, IDf2, w + 1];
  ];
  (* Print[GlobalAngularFluxStore]; *)
  (* update global average flux store ---------------------
   ------------------------------------------------------------------- *)
  GlobalAverageFluxStore[[1]] = CalcAverageFlux[TrackStore, PhiStore, IDm1];
  GlobalAverageFluxStore[[2]] = CalcAverageFlux[TrackStore, PhiStore, IDf1];
  GlobalAverageFluxStore[[3]] = CalcAverageFlux[TrackStore, PhiStore, IDf2];
  (* Print[GlobalAverageFluxStore]; *)
 ];
Print[GlobalAverageFluxStore]
{0.250618, 0.417788, 0.64684}
```