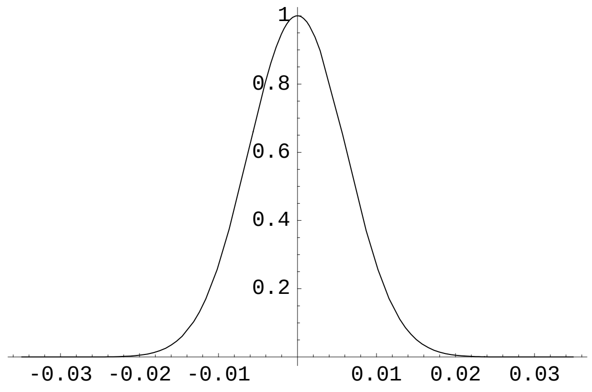
Making a Sun Source Using the GridOfRays and CustomDiffuser Functions

Example scattering function that has a user-defined Gaussian distribution

▪ create data points

```
Plot[Exp[-x^2/N[.5*Degree]^2],{x,-2.*Degree,2.*Degree}];
```



```
scatterdatapoints = Table[{x,Exp[-x^2/N[.5*Degree]^2]},{x,-2.*Degree,2.*Degree,10.Degree/100.
```

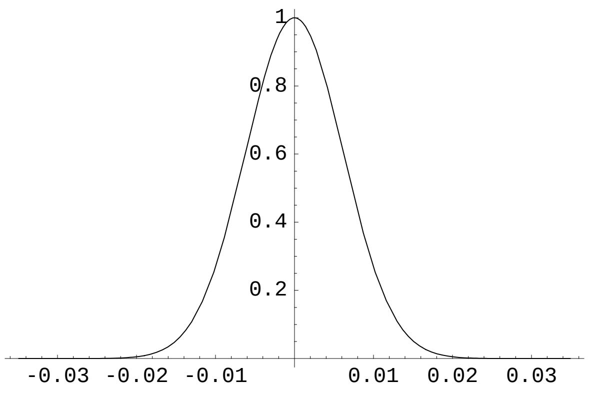
```
{{-0.0349066, 1.12535×10^-7}, {-0.0331613, 5.35535×10^-7}, {-0.0314159, 2.35258×10^-6},
 {-0.0296706, 9.54016×10^-6}, {-0.0279253, 0.0000357128}, {-0.0261799, 0.00012341},
 {-0.0244346, 0.000393669}, {-0.0226893, 0.00115923}, {-0.020944, 0.00315111},
 {-0.0191986, 0.00790705}, {-0.0174533, 0.0183156}, {-0.015708, 0.0391639},
 {-0.0139626, 0.0773047}, {-0.0122173, 0.140858}, {-0.010472, 0.236928},
 {-0.00872665, 0.367879}, {-0.00698132, 0.527292}, {-0.00523599, 0.697676},
 {-0.00349066, 0.852144}, {-0.00174533, 0.960789}, {8.67362×10^-19, 1.}, {0.00174533, 0.960789},
 {0.00349066, 0.852144}, {0.00523599, 0.697676}, {0.00698132, 0.527292},
 {0.00872665, 0.367879}, {0.010472, 0.236928}, {0.0122173, 0.140858}, {0.0139626, 0.0773047},
 {0.015708, 0.0391639}, {0.0174533, 0.0183156}, {0.0191986, 0.00790705},
 {0.020944, 0.00315111}, {0.0226893, 0.00115923}, {0.0244346, 0.000393669},
 {0.0261799, 0.00012341}, {0.0279253, 0.0000357128}, {0.0296706, 9.54016×10^-6},
 {0.0314159, 2.35258×10^-6}, {0.0331613, 5.35535×10^-7}, {0.0349066, 1.12535×10^-7}}
```

▪ make interpolation function

```
scatterfunction = Interpolation[scatterdatapoints]
```

```
InterpolatingFunction[{{-0.0349066, 0.0349066}}, <>]
```

```
Plot[scatterfunction[x],{x,-2.Degree,2.Degree}];
```



■ create diffuser plate with custom scatter function data

```
?Diffuser
```

Diffuser[scatteringparameter, aperture, label, options] and Diffuser[scatteringparameter, aperture, thickness, label, options] denote a planar component that splits incoming rays into diffused rays using the ray scattering distribution pattern given in scatteringparameter.

The scatteringparameter can either be light source function or a listing of one or more angles that describe the scattering behavior. If a light source function is specified, then the ScatterRays function is called internally. An example of a light-source scatteringparameter is WedgeOfRays[90, NumberOfRays->1, MonteCarlo -> True]. Otherwise, if a single number or a list of numeric scatter angles are passed in the scatteringparameter, then AddRandomRayTilt is called instead. Three examples of this are: 90, {90,90}, and {{-45,45},{-45,45}}.

The thickness refers to the refractive substrate of the diffuser. If no thickness value is given, the component is a single-sided diffuser. Diffuser weights the scattered ray Intensity values according to seed ray Intensity values. For Lambertian->True, the scattered rays are oriented about the surface normal. For Lambertian->False, the scattered rays are oriented with the incoming ray direction. The user-named label parameter is optional and can be omitted. When it is present, its text content is used to identify the object in both the rendered graphics and the output cell expression. When it is omitted, Rayica uses the default setting of the Labels option with the rendered graphics.

See also: ScatterRays, AddRandomRayTilt, DiffuserMirror, SphericalDiffuserMirror, CustomDiffuser, and CustomDiffuserMirror.

```
(*Note that Diffuser uses AddRandomRayTilt internally*)
```

```
?AddRandomRayTilt
```

AddRandomRayTilt[component, {{x1,x2},{y1,y2}}, options] is a generic building block that introduces a random factor to the tilt direction of each propagated ray.

Here, the {{x1,x2},{y1,y2}} parameters specify the angular ranges of the ray tilt fluctuations along the horizontal and vertical directions. When the random behavior is symmetric about the surface normal, then the single-valued full-angles x and y can be specified in place of {x1,x2} and {y1,y2}. When the random factor is non-isotropic and only varies in x, then the y parameters can be omitted. Three examples of this are: 90, {90,90}, and {{-45,45},{-45,45}}.

The Lambertian option is also used by ScatterRays. For Lambertian->True, the scattered rays are oriented about the surface normal. For Lambertian->False, the scattered rays are oriented with the incoming ray direction.

See also: Lambertian, LimitScatter, and LimitByReflection.

```
Options[Diffuser]
```

```
{Labels → D, LabelPositions → Automatic, ComponentDescription → Automatic,
 Temperature → Temperature, Tension → Tension, ComponentMedium → BK7,
 Transmittance → Transmittance, Reflectance → Reflectance, Lambertian → False,
 CosineCompensation → False, RayScatterFunction → Automatic,
 LimitScatter → True, LimitByReflection → Automatic, GraphicDesign → Automatic,
 FrontSurface → True, SwitchDirectionOnReflection → False,
 Automatic → {SurfaceRendering → Trace, EdgeRendering → Mesh, CrossRendering → {{Fill, Trace}}},
 Sketch → {SurfaceRendering → {Mesh, Trace}, EdgeRendering → Trace, CrossRendering → Empty},
 Wire → {SurfaceRendering → Mesh, EdgeRendering → Mesh, CrossRendering → {{Fill, Trace}}},
 Solid → {SurfaceRendering → {{Fill, Trace}}, EdgeRendering → Fill, CrossRendering → Empty}}
```

```
?RayScatterFunction
```

```
RayScatterFunction -> scatteringfunction is an option of AddRandomRayTilt that
  indicates an angular dependent scattering function for diffuse optical surfaces.

Typical RayScatterFunction settings include: Function[Cos[#]] for cosine-dependant (lambertian)
  scattering, Function[Cos[#]^2] for non-lambertian scattering, and Function[1] for uniform
  scattering. RayScatterFunction -> {scatteringfunction1,scatteringfunction2} can also be used
  to specify a different scattering function in the horizontal and vertical angular directions.
  Finally, RayScatterFunction -> Automatic works together with the CosineCompensation option.
  In particular, with CosineCompensation -> True and RayScatterFunction -> Automatic,
  RayScatterFunction is internally set to Function[Cos[#]]. Similarly, with CosineCompensation
  -> False and RayScatterFunction -> Automatic, Function[1] is used internally instead.
```

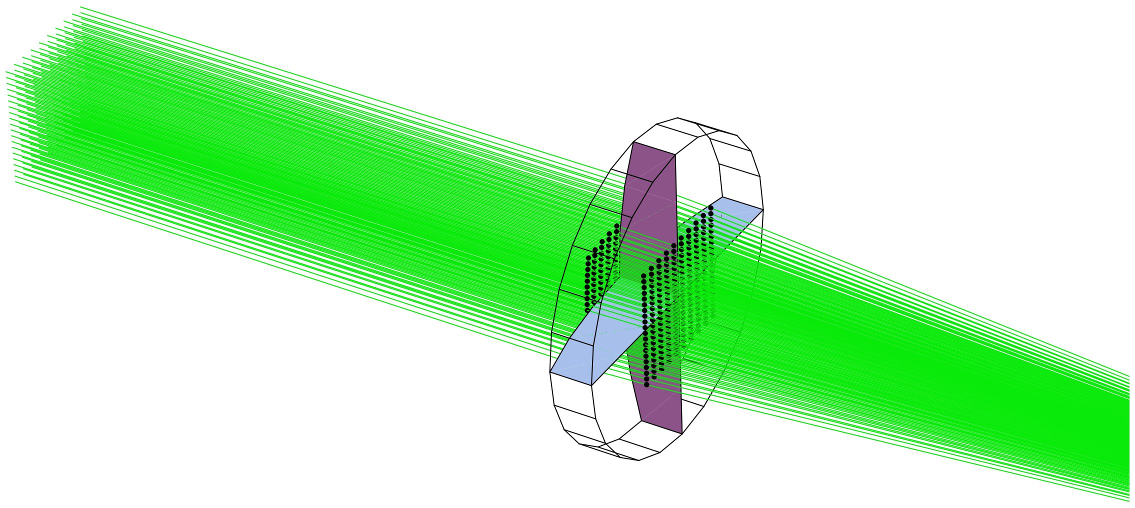
```
customdiffuser = Diffuser[{4,4},{50,50},RayScatterFunction->scatterfunction,RunningCommentary
```

```
RunningCommentary -> True for ComponentFoundation.

Constructing single surface.

Constructing SurfaceFunction format.

Constructing symbolic values.

Constructing SymbolicSurfaceFunction.

Constructing analytic SurfaceNormalFunction.

Constructing SymbolicSurfaceNormalFunction.

Constructing analytic JacobianMatrix.

Constructing symbolic analytic JacobianMatrix.

ExtrapolateSurfaceFunction -> False.

Plugging together.

Calculating  SurfaceRayIntersectionFunction.

Calculating  SymbolicSurfaceRayIntersectionFunction.

Exiting ComponentFoundation.

Begin ComponentRendering.

Adding Magnification.

Removing any existing rendering information

Modifying Labels.

Calling SingleSurfaceGraphics.

Exiting ComponentRendering.

inputscatter = {4, 4}

Lambertian = False

LimitScatter = True

CosineCompensation = False

LimitByReflection = False

{scatterx, scattery} = {{-2, 2}, {-2, 2}}

Construct code for two-dimensional scattering

Using isotropic scatter about the original
  ray axis and user-specified RayScatterFunction away from ray axis.

      Diffuser[{4, 4}, {50, 50},
       {RayScatterFunction → gaussianscatterfunction, RunningCommentary → True}]
```

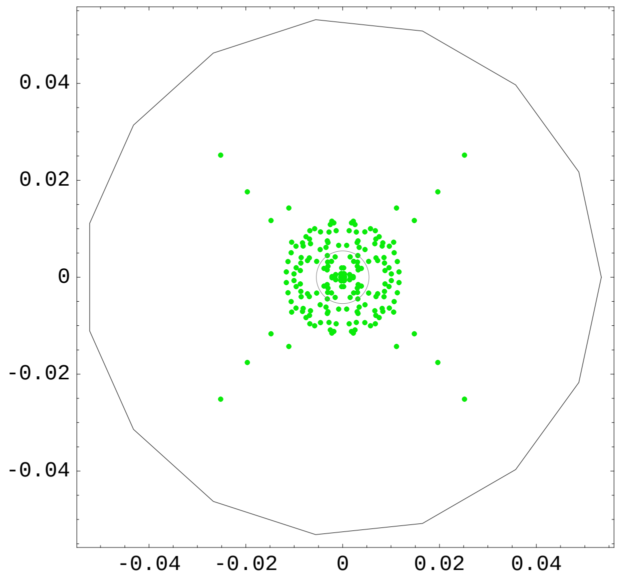
Build system to test performance

standard performance without diffuser (*By inserting a focusing element, angles get converted to position on the surface and we can use FindIntensity to measure a histogram of angular distribution*)

```
standard = TurboPlot[{GridOfRays[{20,20},NumberOfRays->{10,20}],
    Move[PlanoConvexLens[200,50,10],100],
    Move[Screen[{50,50}],302.699]}];
```



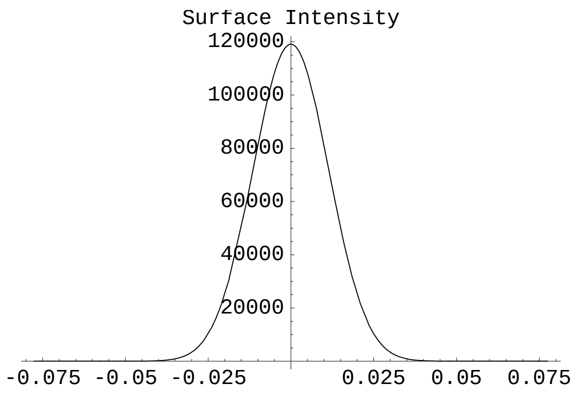
```
FindFocus[standard]
```



```
{Screen → Move[Screen[0.106825], 302.699],
 TurboSystem → -traced system-, FocalPoint → {302.699, 0, 0}, FocusType → RMSFocus,
 WeightedSpotSize → 0.0108705, SpotSize → 0.0108705, BackFocalLength → 192.699,
 FocalPlaneTilt → {1., 0, 0}, TurboRays → -ray intercepts of 1 surfaces-}
```

```
FindIntensity[standard,Plot2D->True];
```

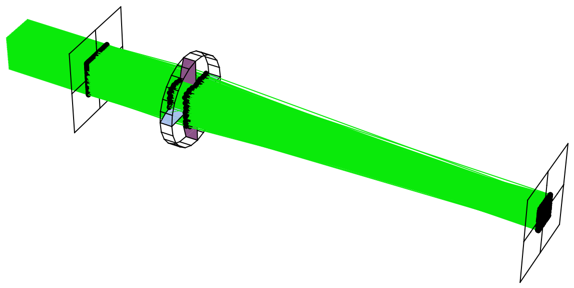
```
Surface Information :
 {ComponentNumber → 2., SurfaceNumber → 1., NumberOfRays → 200, SmoothKernelSize → 0.0130854}
```



▪ normal diffuser

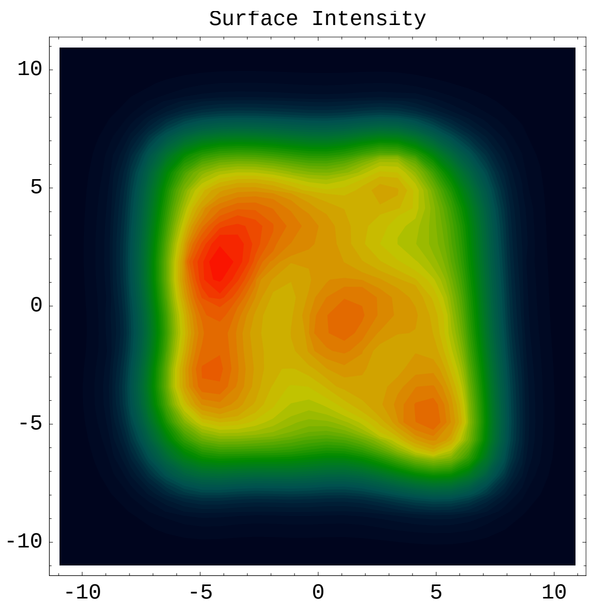
```
results = TurboPlot[{GridOfRays[{20,20},NumberOfRays->{30,30}],
    Move[Diffuser[{4,4},{50,50},RunningCommentary->True],50],
    Move[PlanoConvexLens[200,50,10],100],
    Move[Screen[{50,50}],302.699]}];
```

```
RunningCommentary -> True for ComponentFoundation.

Constructing single surface.

Constructing SurfaceFunction format.

Constructing symbolic values.
Constructing SymbolicSurfaceFunction.

Constructing analytic SurfaceNormalFunction.

Constructing SymbolicSurfaceNormalFunction.

Constructing analytic JacobianMatrix.

Constructing symbolic analytic JacobianMatrix.

ExtrapolateSurfaceFunction -> False.

Plugging together.

Calculating  SurfaceRayIntersectionFunction.

Calculating  SymbolicSurfaceRayIntersectionFunction.

Exiting ComponentFoundation.

Begin ComponentRendering.

Adding Magnification.

Removing any existing rendering information

Modifying Labels.

Calling SingleSurfaceGraphics.

Exiting ComponentRendering.

inputscatter = {4, 4}

Lambertian = False

LimitScatter = True

CosineCompensation = False

LimitByReflection = False

{scatterx, scattery} = {{-2, 2}, {-2, 2}}

Construct code for two-dimensional scattering

Using uniform scattering for x scattering direction.

Using uniform scattering for y scattering direction.
```



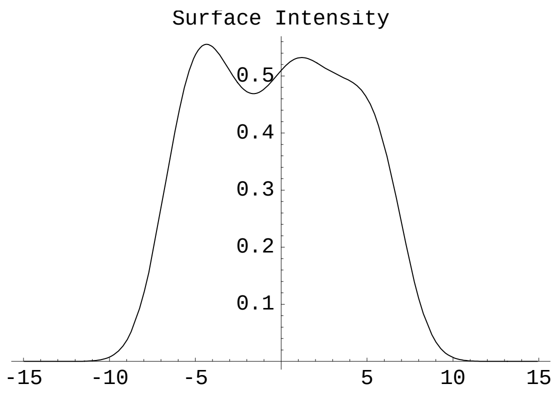
```
FindIntensity[results,2];
```

```
Surface Information :
 {ComponentNumber → 3., SurfaceNumber → 1., NumberOfRays → 900, SmoothKernelSize → 2}
```



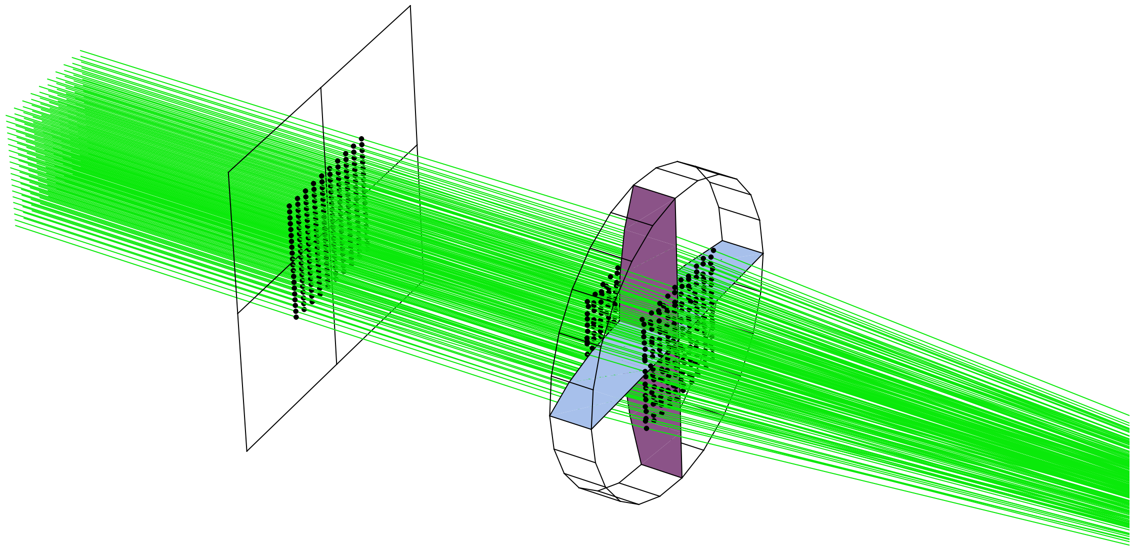
FindIntensity[results,2,Plot2D->True];

```
Surface Information :
 {ComponentNumber → 3., SurfaceNumber → 1., NumberOfRays → 900, SmoothKernelSize → 2}
```



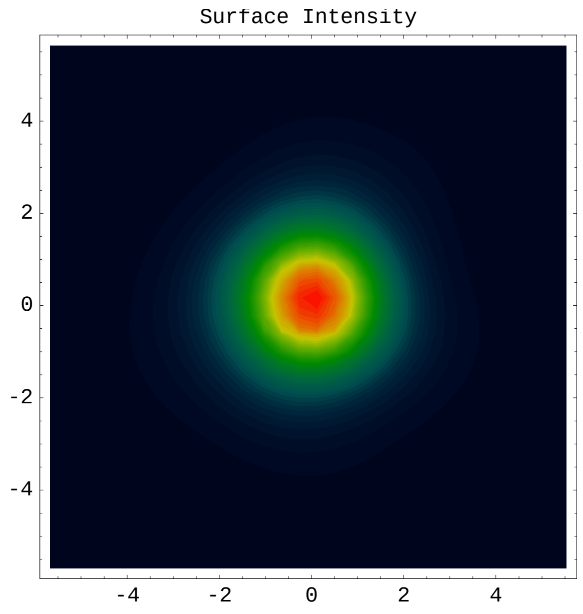
■ add custom diffuser

```
results = TurboPlot[{GridOfRays[{20,20},NumberOfRays->{10,20}],
    Move[customdiffuser,50],
    Move[PlanoConvexLens[200,50,10],100],
    Move[Screen[{50,50}],302.699]}];
```



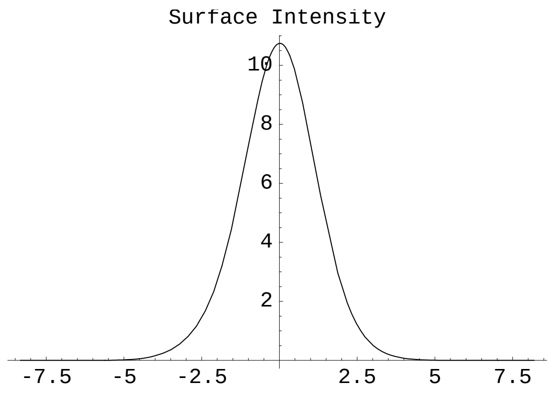
```
FindIntensity[results];
```

```
Surface Information :
 {ComponentNumber → 3., SurfaceNumber → 1., NumberOfRays → 200, SmoothKernelSize → 1.32815}
```



```
FindIntensity[results,Plot2D->True];
```

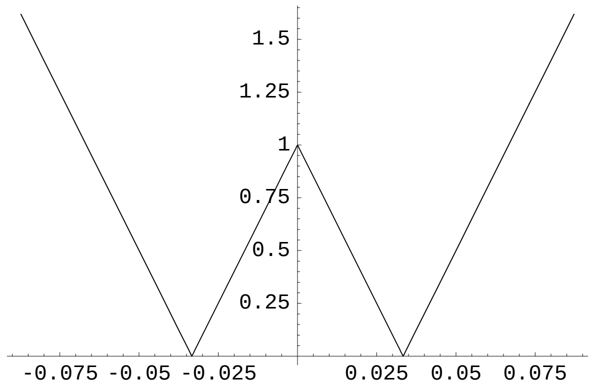
```
Surface Information :
 {ComponentNumber → 3., SurfaceNumber → 1., NumberOfRays → 200, SmoothKernelSize → 1.32815}
```



Example scattering function that has a custom-shaped distribution

■ create data points

```
Plot[Abs[1-Abs[30*x]],{x,-5.Degree,5.*Degree}];
```



```
scatterdatapoints = Table[{x,Abs[1-Abs[30*x]]},{x,-5.*Degree,5.*Degree,10.Degree/100.}]
```

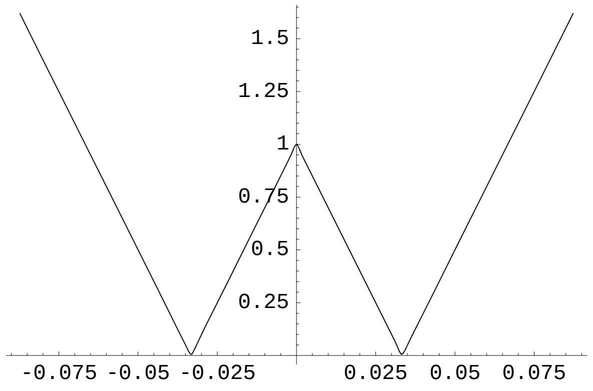
```
{{-0.0872665, 1.61799}, {-0.0855211, 1.56563}, {-0.0837758, 1.51327}, {-0.0820305, 1.46091},
 {-0.0802851, 1.40855}, {-0.0785398, 1.35619}, {-0.0767945, 1.30383}, {-0.0750492, 1.25147},
 {-0.0733038, 1.19911}, {-0.0715585, 1.14675}, {-0.0698132, 1.0944}, {-0.0680678, 1.04204},
 {-0.0663225, 0.989675}, {-0.0645772, 0.937315}, {-0.0628319, 0.884956},
 {-0.0610865, 0.832596}, {-0.0593412, 0.780236}, {-0.0575959, 0.727876},
 {-0.0558505, 0.675516}, {-0.0541052, 0.623156}, {-0.0523599, 0.570796},
 {-0.0506145, 0.518436}, {-0.0488692, 0.466077}, {-0.0471239, 0.413717},
 {-0.0453786, 0.361357}, {-0.0436332, 0.308997}, {-0.0418879, 0.256637},
 {-0.0401426, 0.204277}, {-0.0383972, 0.151917}, {-0.0366519, 0.0995574},
 {-0.0349066, 0.0471976}, {-0.0331613, 0.00516233}, {-0.0314159, 0.0575222},
 {-0.0296706, 0.109882}, {-0.0279253, 0.162242}, {-0.0261799, 0.214602},
 {-0.0244346, 0.266962}, {-0.0226893, 0.319322}, {-0.020944, 0.371681}, {-0.0191986, 0.424041},
 {-0.0174533, 0.476401}, {-0.015708, 0.528761}, {-0.0139626, 0.581121}, {-0.0122173, 0.633481},
 {-0.010472, 0.685841}, {-0.00872665, 0.738201}, {-0.00698132, 0.79056},
 {-0.00523599, 0.84292}, {-0.00349066, 0.89528}, {-0.00174533, 0.94764}, {5.63785×10^-18, 1.},
 {0.00174533, 0.94764}, {0.00349066, 0.89528}, {0.00523599, 0.84292}, {0.00698132, 0.79056},
 {0.00872665, 0.738201}, {0.010472, 0.685841}, {0.0122173, 0.633481}, {0.0139626, 0.581121},
 {0.015708, 0.528761}, {0.0174533, 0.476401}, {0.0191986, 0.424041}, {0.020944, 0.371681},
 {0.0226893, 0.319322}, {0.0244346, 0.266962}, {0.0261799, 0.214602}, {0.0279253, 0.162242},
 {0.0296706, 0.109882}, {0.0314159, 0.0575222}, {0.0331613, 0.00516233},
 {0.0349066, 0.0471976}, {0.0366519, 0.0995574}, {0.0383972, 0.151917},
 {0.0401426, 0.204277}, {0.0418879, 0.256637}, {0.0436332, 0.308997}, {0.0453786, 0.361357},
 {0.0471239, 0.413717}, {0.0488692, 0.466077}, {0.0506145, 0.518436}, {0.0523599, 0.570796},
 {0.0541052, 0.623156}, {0.0558505, 0.675516}, {0.0575959, 0.727876}, {0.0593412, 0.780236},
 {0.0610865, 0.832596}, {0.0628319, 0.884956}, {0.0645772, 0.937315}, {0.0663225, 0.989675},
 {0.0680678, 1.04204}, {0.0698132, 1.0944}, {0.0715585, 1.14675}, {0.0733038, 1.19911},
 {0.0750492, 1.25147}, {0.0767945, 1.30383}, {0.0785398, 1.35619}, {0.0802851, 1.40855},
 {0.0820305, 1.46091}, {0.0837758, 1.51327}, {0.0855211, 1.56563}, {0.0872665, 1.61799}}
```

make interpolation function

```
scatterfunction = Interpolation[scatterdatapoints]
```

```
InterpolatingFunction[{{-0.0872665, 0.0872665}}, <>]
```

```
Plot[scatterfunction[x],{x,-5.Degree,5.Degree}];
```



create one-dimensional diffuser plate with custom scatter function data

```
customdiffuser = Diffuser[10,{50,50},RayScatterFunction->scatterfunction,RunningCommentary->T
```

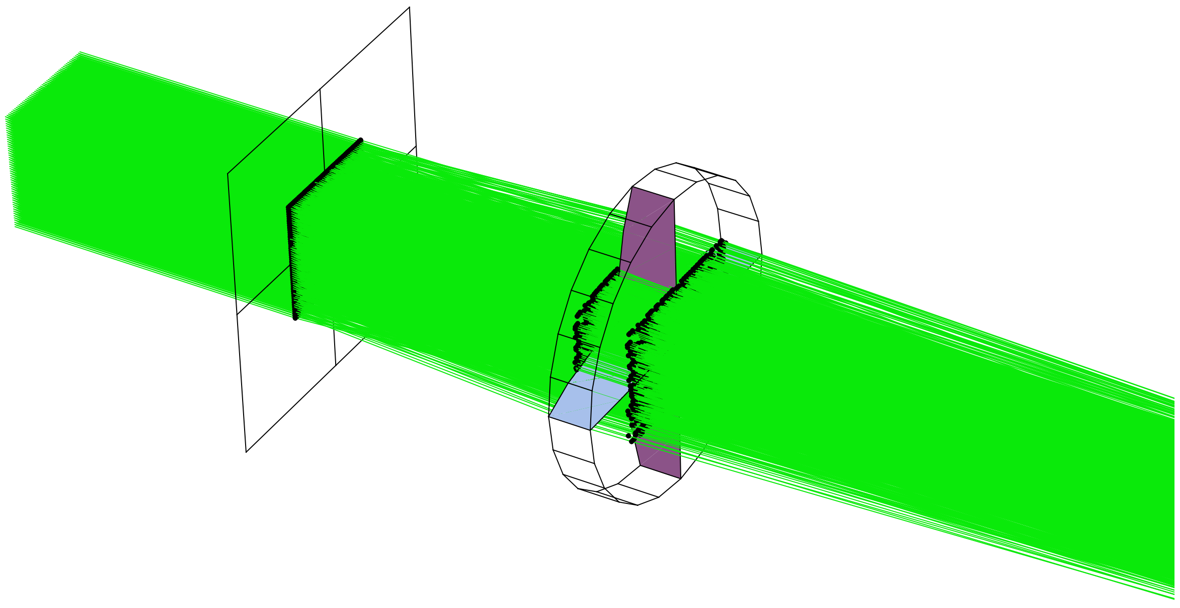
```
RunningCommentary -> True for ComponentFoundation.

Constructing single surface.

Constructing SurfaceFunction format.

Constructing symbolic values.

Constructing SymbolicSurfaceFunction.

Constructing analytic SurfaceNormalFunction.

Constructing SymbolicSurfaceNormalFunction.

Constructing analytic JacobianMatrix.

Constructing symbolic analytic JacobianMatrix.

ExtrapolateSurfaceFunction -> False.

Plugging together.

Calculating  SurfaceRayIntersectionFunction.

Calculating  SymbolicSurfaceRayIntersectionFunction.

Exiting ComponentFoundation.

Begin ComponentRendering.

Adding Magnification.

Removing any existing rendering information

Modifying Labels.

Calling SingleSurfaceGraphics.

Exiting ComponentRendering.

inputscatter = {10}

Lambertian = False

LimitScatter = True

CosineCompensation = False

LimitByReflection = False

{scatterx, scattery} = {{-5, 5}, {0, 0}}

Construct code for one-dimensional scattering
```

```
Diffuser[10, {50, 50},
 {RayScatterFunction → InterpolatingFunction[{{-0.0872665, 0.0872665}}, {2, 0, True, Real, {3},
    {0}}, {{-0.0872665, -0.0855211, -0.0837758, -0.0820305, -0.0802851, -0.0785398,
     -0.0767945, -0.0750492, -0.0733038, -0.0715585, -0.0698132, -0.0680678, -0.0663225,
     -0.0645772, -0.0628319, -0.0610865, -0.0593412, -0.0575959, -0.0558505, -0.0541052,
     -0.0523599, -0.0506145, -0.0488692, -0.0471239, -0.0453786, -0.0436332, -0.0418879,
     -0.0401426, -0.0383972, -0.0366519, -0.0349066, -0.0331613, -0.0314159, -0.0296706,
     -0.0279253, -0.0261799, -0.0244346, -0.0226893, -0.020944, -0.0191986, -0.0174533,
     -0.015708, -0.0139626, -0.0122173, -0.010472, -0.00872665, -0.00698132, -0.00523599,
     -0.00349066, -0.00174533, 5.63785×10^-18, 0.00174533, 0.00349066, 0.00523599,
     0.00698132, 0.00872665, 0.010472, 0.0122173, 0.0139626, 0.015708, 0.0174533, 0.0191986,
     0.020944, 0.0226893, 0.0244346, 0.0261799, 0.0279253, 0.0296706, 0.0314159, 0.0331613,
     0.0349066, 0.0366519, 0.0383972, 0.0401426, 0.0418879, 0.0436332, 0.0453786, 0.0471239,
     0.0488692, 0.0506145, 0.0523599, 0.0541052, 0.0558505, 0.0575959, 0.0593412, 0.0610865,
     0.0628319, 0.0645772, 0.0663225, 0.0680678, 0.0698132, 0.0715585, 0.0733038, 0.0750492,
     0.0767945, 0.0785398, 0.0802851, 0.0820305, 0.0837758, 0.0855211, 0.0872665}},
    {{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
     23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
     43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
     63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
     83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101},
    {1.61799, 1.56563, 1.51327, 1.46091, 1.40855, 1.35619, 1.30383, 1.25147, 1.19911,
     1.14675, 1.0944, 1.04204, 0.989675, 0.937315, 0.884956, 0.832596, 0.780236, 0.727876,
     0.675516, 0.623156, 0.570796, 0.518436, 0.466077, 0.413717, 0.361357, 0.308997, 0.256637,
     0.204277, 0.151917, 0.0995574, 0.0471976, 0.00516233, 0.0575222, 0.109882, 0.162242,
     0.214602, 0.266962, 0.319322, 0.371681, 0.424041, 0.476401, 0.528761, 0.581121,
     0.633481, 0.685841, 0.738201, 0.79056, 0.84292, 0.89528, 0.94764, 1., 0.94764, 0.89528,
     0.84292, 0.79056, 0.738201, 0.685841, 0.633481, 0.581121, 0.528761, 0.476401, 0.424041,
     0.371681, 0.319322, 0.266962, 0.214602, 0.162242, 0.109882, 0.0575222, 0.00516233,
     0.0471976, 0.0995574, 0.151917, 0.204277, 0.256637, 0.308997, 0.361357, 0.413717,
     0.466077, 0.518436, 0.570796, 0.623156, 0.675516, 0.727876, 0.780236, 0.832596, 0.884956,
     0.937315, 0.989675, 1.04204, 1.0944, 1.14675, 1.19911, 1.25147, 1.30383, 1.35619,
     1.40855, 1.46091, 1.51327, 1.56563, 1.61799}}, {Automatic}], RunningCommentary → True}]
```

Build system to test performance

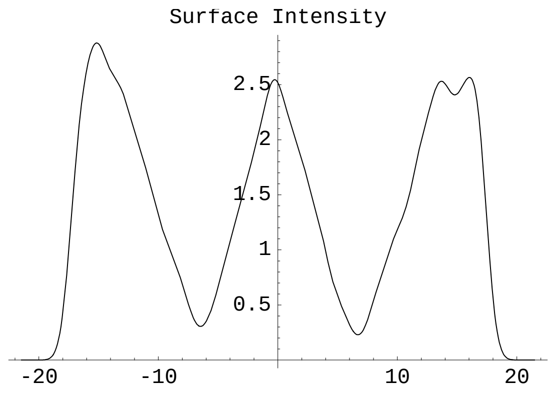
create custom diffuser

```
results = TurboPlot[{GridOfRays[{20,20},NumberOfRays->{50,50}],
    Move[customdiffuser,50],
    Move[PlanoConvexLens[200,50,10],100],
    Move[Screen[{50,50}],302.699]}];
```



```
FindIntensity[results,1.,Plot2D->True];
```

```
Surface Information :
 {ComponentNumber → 3., SurfaceNumber → 1., NumberOfRays → 2500, SmoothKernelSize → 1.}
```


Surface Intensity