

NEWT: SCALE General Geometry 2D Transport Kernel

SCALE 6.3.3

ORNL is managed by UT-Battelle, LLC for the US Department of Energy



U.S. DEPARTMENT OF
ENERGY

Learning goals

- **Understand** capabilities of the NEWT neutron transport code
- **Understand** NEWT's output files
- **Learn** how to set up a NEWT input

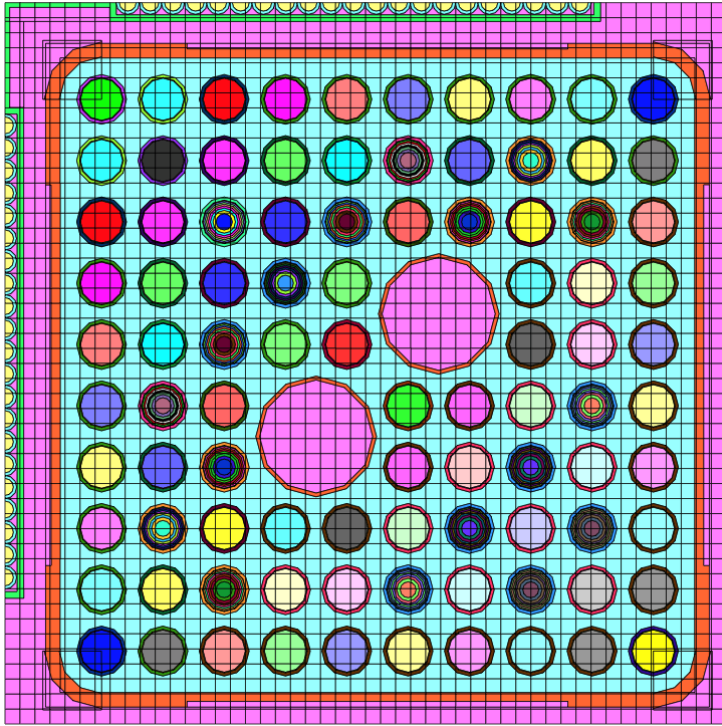
What is NEWT?



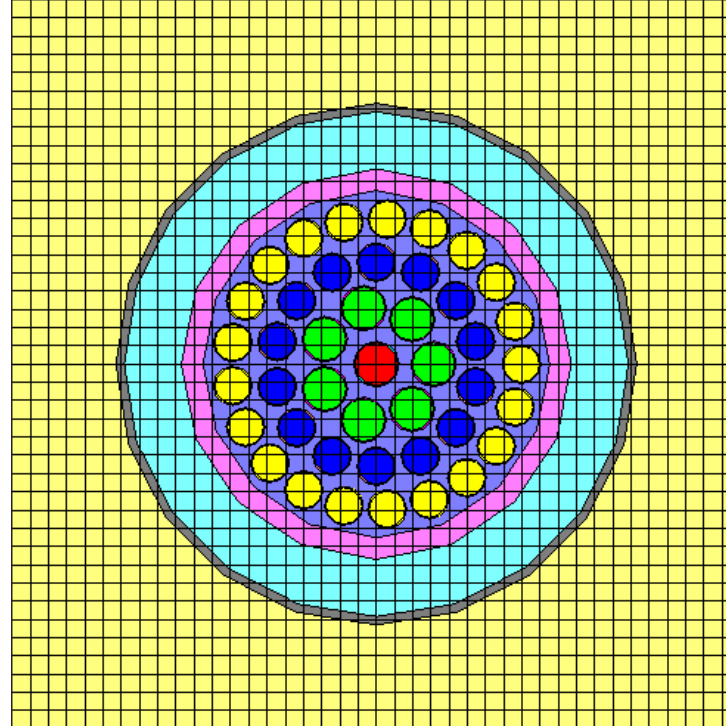
NEWT – Introduction

- NEWT provides a discrete ordinates solution for **arbitrary, non-orthogonal configurations**
- NEWT uses extended step characteristic (ESC) discretization:
 - Allows cells in the form of arbitrary polygons
 - Able to closely approximate any curved surface (with volume conserved)
 - Able to exactly represent any straight-sided body

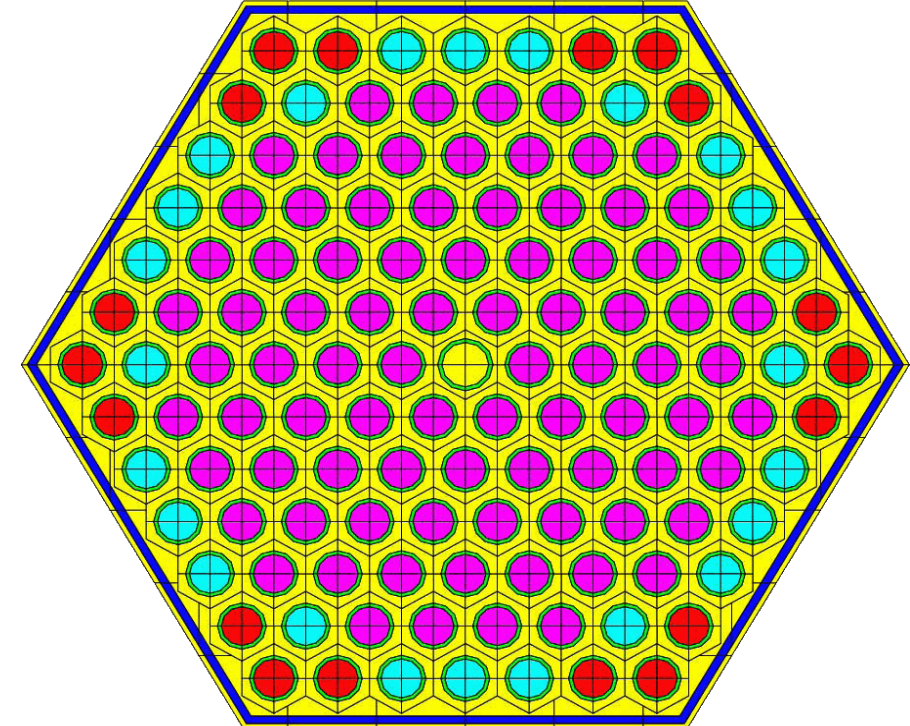
Application examples



BWR fuel assembly



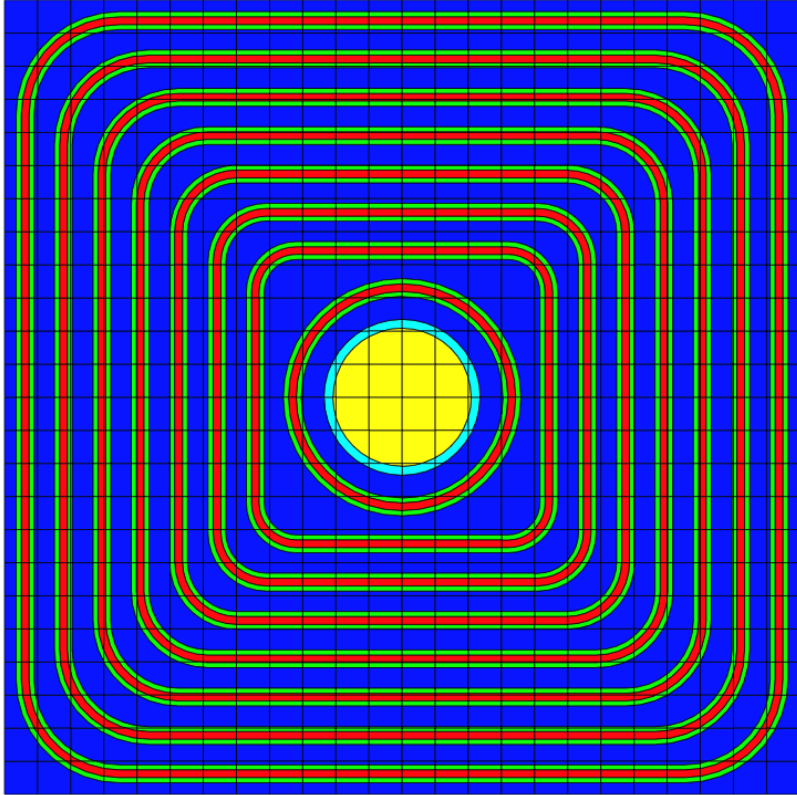
ACR-700 fuel assembly



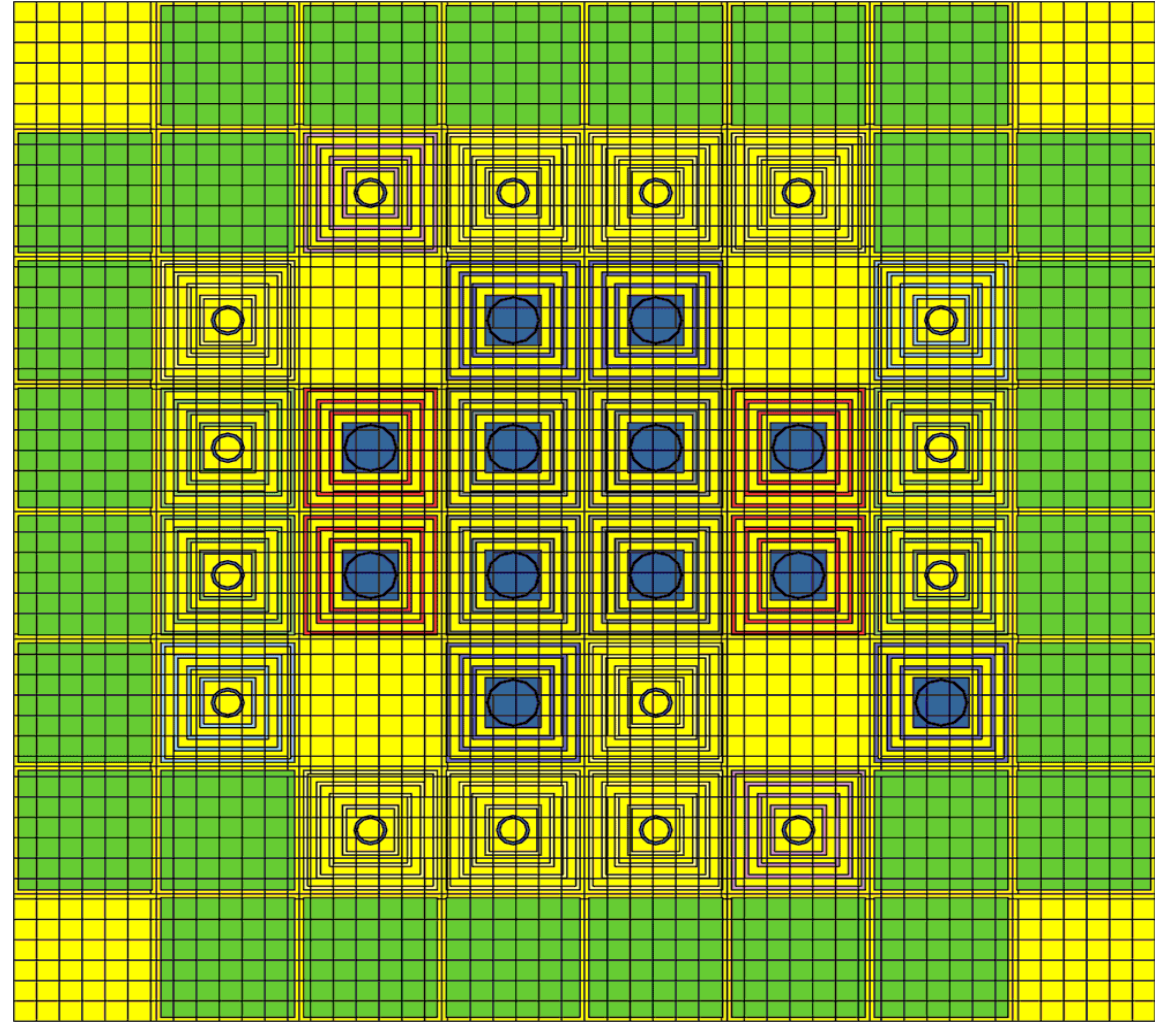
VVER-440 fuel assembly

BWR: Boiling Water Reactor
ACR: Advanced CANDU Reactor
VVER: Water-Water Energetic Reactor

Application examples



IRT fuel assembly



IRT core

Features and capabilities

- Provides automated grid generation (fast and easy)
- Runs multi-group neutron transport calculation
- Provides a rigorous and highly accurate deterministic solution for non-trivial configurations
- Has solution modes
 - Eigenvalue
 - Eigenvalue with B1 critical spectrum
 - Fixed source

Features and capabilities

- Provides postscript-based graphics of the transport model and grid
- Allows four boundary conditions
 - Reflective
 - White
 - Vacuum
 - Periodic
- Offers convergence control
- Allows microscopic and macroscopic cross section collapse
- Has comparable speed to other 2D Sn codes
- Uses geometry specification similar to that of KENO-VI (2D instead of 3D)

Features and capabilities

- Invoked by T-NEWT sequence that automatically executes nuclear data processing followed by the NEWT neutron transport calculation
- Is used by other SCALE sequences:
 - TRITON for depletion
 - TSUNAMI for sensitivity and uncertainty analysis
- Provides relevant output core simulators:
 - Flux-weighted cross sections
 - By nuclide in each mixture
 - Homogenized for multiple mixtures
 - Other data required for nodal diffusion codes
 - Betas (delayed neutron fractions), lambdas (decay constants), diffusion coefficients, etc.

NEWT Input

- PARAMETER and MATERIALS block

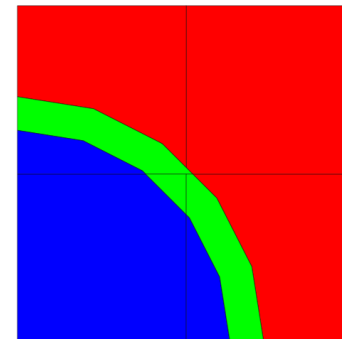


NEWT input

- Keyword-based input
- Input parameters included in data blocks
- Basic model must include these data blocks:
 - Control parameters
 - Mixture identification
 - Cross section processing (not required, but typically needed)
 - Geometry description
- Additional optional data blocks are used for special operations

NEWT input example

```
=t-newt  
Infinite lattice PWR pin cell  
V7.1-56  
read composition ... end composition  
read celldata ... end celldata  
read model  
read parm  
collapse=yes outers=100 epsilon=1.0e-5 kguess=1.34 prtxsec=yes prtmxsec=yes prtbroad=yes  
end parm  
read materials  
mix=3 pn=2 com='moderator' end  
mix=2 pn=1 com='clad' end  
mix=1 pn=1 com='fuel' end  
end materials  
read geom  
global unit 1  
cylinder 10 0.41000 chord +x=0 chord +y=0 sides=20 com='fuel'  
cylinder 20 0.4750 chord +x=0 chord +y=0 sides=20 com='clad'  
cuboid 30 0.65635 0.0 0.65635 0.0 com='moderator'  
media 1 1 10  
media 2 1 20 -10  
media 3 1 30 -20  
boundary 30 2 2  
end geom  
read bounds  
all=refl  
end bounds  
end model  
end
```



PWR 1/4 pin cell

Input blocks

READ PARAM	problem control data	END PARAM	}	Required
READ MATERIALS	material data	END MATERIALS		
READ GEOMETRY	model geometry specifications	END GEOMETRY		
READ BOUNDS	boundary conditions	END BOUNDS		
READ COLLAPSE	energy collapse data	END COLLAPSE	}	Optional
READ ARRAY	arrays	END ARRAY		
READ SOURCE	fixed source specifications	END SOURCE		
READ HOMOG	spatial homogenization data	END HOMOG		
READ ADF	discontinuity factor data	END ADF		
READ FLUXPLANE	flux and current edit	END FLUXPLANE		

PARAMETER block: Basic control options

- **solntype**=(keff/b1/src) – Eigenvalue/B1 critical spectrum/Fixed source
- **sn**=(2/4/6/8/10/12/14/16) – Level symmetric quadrature order
- **nazim**=integer – number of equally spaced azimuthal directions in product quadrature set (0, level symmetric is default)
- **npolar**=integer – number of polar angles in product quadrature set (0, level symmetric is default)
- **epseigen**=real – convergence criterion for k (0.0001)
- **epsinner**=real – spatial convergence criterion for inner iterations (0.0001)
- **epsouter**=real – spatial convergence criterion for outer iterations (0.0001)
- **epsthrm**=real – spatial convergence criterion for thermal-upscattering iterations, if enabled (0.0001)
- **epsilon**=real – simultaneously sets all (spatial and eigenvalue) convergence criteria to the same value
- **inners**=integer – max number of inner iterations per energy group (5)
- **outers**=integer – max number of outer iterations (250)
- **converge**=(cell/mix) – region within which convergence testing is applied
- **therm**=(yes/no) – enables/disables thermal-upscattering iterations (yes)
- **therms**=integer – max number of thermal-upscattering iterations, if enabled via “therm=yes” (2)

PARAMETER block: Output control options

- **timed**=(yes/**no**) – print iteration timing and CPU usage data
- **drawit**=(yes/**no**) – create PostScript geometry/material plots
- **echo**=(yes/**no**) – echo outer iteration edits to standard error
- **prtflex**=(yes/**no**) – print fine-group flux edit PostScript file; if collapse is performed, print broad-group flux edit file
- **prtmxsec**=(yes/**no**/1d) – print mixed (macroscopic) cross-sections; 1d will not print scattering cross sections
- **prtmxtab**=(yes/**no**) – print input mixing table
- **prtxsec**=(yes/**no**/1d) – print microscopic cross sections read from library; 1d will not print scattering cross sections

drawit enables generation of an image of the NEWT geometry – the geometry that is used in the neutron transport calculation
→ this image must always be inspected before running a problem

PARAMETER block: Advanced control options

- **cmfd**=(yes/**no**/rect/part) – coarse-mesh finite-difference accelerator
- **xcmfd**=*integer* – number of fine-grid cells per coarse cell in x (**1**)
- **ycmfd**=*integer* – number of fine-grid cells per coarse cell in y (**1**)
- **collapse**=(yes/**no**) – perform flux-weighted cross section collapse
- **prtbroad**=(yes/**no**/1d) – print broad-group (collapsed) cross sections; 1d will not print scattering cross sections
- **prthmmix**=(**yes**/no) – print broad-group homogenized cross sections

PARAMETER block: Examples

- Most defaults are sufficient
- Common parameters manually set:
 - Epsilon (stricter convergence than default $1e-4$)
 - Converge (mix instead of cell)
 - CMFD (see later slides)
 - Print, draw, and echo

```
read parm  
prtflux=no  
prtbroad=yes  
collapse=yes  
sn=6  
inners=4  
outers=100  
epsinner=1.0e-4  
epsouter=1.0e-4  
epseigen=1.0e-3  
kguess=1.34  
prtxsec=yes  
prtmxsec=yes  
prtmxtab=yes  
end parm
```

```
read parm  
cmfd=yes  
timed=yes  
converge=mix  
drawit=yes  
echo=yes  
prtflux=yes  
epsilon=1e-5  
end parm
```

MATERIAL block

- Keywords:
 - **mix**: mixture number (consistent with COMPOSITION block)
 - **pn**: Legendre scattering order
 - **com**: mixture description
- **pn** recommendations:
 - Moderators with high scattering: 2
 - Fuel and cladding: 1
 - Gap: 0 or 1
- If not specified, **pn** defaults to 1
- **com** is optional but useful:
 - Comments help avoid input errors
 - **Comment is used as mixture label in NEWT-generated plot**

```
read materials
  mix=I1 pn=N1 com='text' end
  mix=I2 pn=N2 com='text' end
  ...
end materials
```

MATERIAL block: Example

```
read materials
```

```
  mix=1 pn=1 com='fuel'      end
```

```
  mix=2 pn=1 com='clad'      end
```

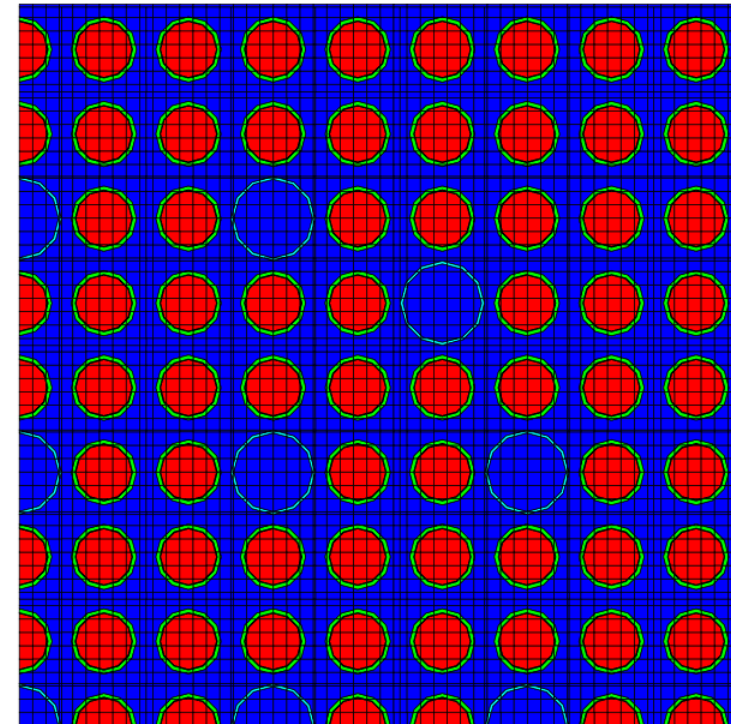
```
  mix=3 pn=2 com='coolant'   end
```

```
  mix=4 pn=0 com='gap'       end
```

```
  mix=5 pn=1 com='guide tube' end
```

```
end materials
```

- 1 fuel
- 2 clad
- 3 coolant
- 4 gap
- 5 guide tube



`${BASENAME}.newtmatl.ps` image
generated via `drawit=yes`

NEWT input

- Part 1: Basic geometry



NEWT geometry

- NEWT uses the SCALE Generalized Geometry Package (SGGP)
- For users familiar with KENO-VI or other Monte Carlo codes, the pattern is similar to that for KENO-VI, with some exceptions:
 - Limited to 2D (x-y plane)
 - Approximates specified curved surfaces (e.g., cylinders) as polygons, with user-control over number of sides (default is 12 sides/full circle)
 - Limits the global geometry to cuboid, wedge, or hexagon
 - Allows specification of underlying grid structure

NEWT geometry

Three main blocks:

- **GEOMETRY** block
 - Define problem geometry
- **ARRAY** block (optional)
 - Define problem arrays
- **BOUNDS** block
 - Define problem boundary conditions

GEOMETRY block

- Geometry is defined in terms of units
- Many units may be defined, but the global domain is defined by a single global unit
- Within each unit, one may place other structures in the form of
 - shapes
 - arrays (discussed later)
 - holes (discussed later)

GEOMETRY block – Unit

- Basic building blocks of model
- One global unit is always required
- Other units may be defined and are placed in the global unit through arrays or holes
- Keyword unit is followed by a unique unit identification number
 - Example: **unit 10**
- Each unit contains:
 - shapes
 - arrays (optional)
 - holes (optional)
 - **media** specifications
 - **boundary** definition

GEOMETRY block – Shapes

- Shapes are predefined bodies:
 - cuboid
 - cylinder
 - hexprism
 - rhexprism (rotated hexprism)
 - wedge*
 - polygon*

**Polygons and wedges cannot be displayed with Fulcrum in SCALE 6.3*

GEOMETRY block – Shapes

- Example:

```
unit 20
cylinder      10      0.6
cylinder      20      0.8
cuboid        30      1.0 -1.0 1.0 -1.0
media statements
boundary definition
```

- Order of shapes is not important
- Each shape must be assigned a unique ID within each unit

GEOMETRY block – Shapes

SGGP Geometry Keyword	Geometry Record Input
CUBOID	BodyID, +X, -X, +Y, -Y positions
CYLINDER	BodyID, radius
HEXPRISM, RHEXPRISM	BodyID, inner radius (half pitch)
WEDGE	BodyID, Xbase, Xpt, Ypt
POLYGON	BodyID x0, y0, x1, y1, ..., xN, yN, x0, y0

General freeform input (FIDO)

Count Field	Option Field	Operand Field	Comment
	f	j	Fills the remainder of the array with j, starting with the current position
i	r or *	j	Repeats entry j a total of i times
i	p	j	Stores j in the next i positions, alternating the sign each time
i	z		Stores zero in the next i positions
Refer to Table 8.1.19 for a complete list of options			

Examples:

3r2.5

4p3

5z

is identical with

is identical with

is identical with

2.5 2.5 2.5

3 -3 3 -3

0 0 0 0 0

Commonly used to define cuboid

 OAK RIDGE
National Laboratory

SCALE 6.3.3| Last Updated: October 2025

28

Cuboid

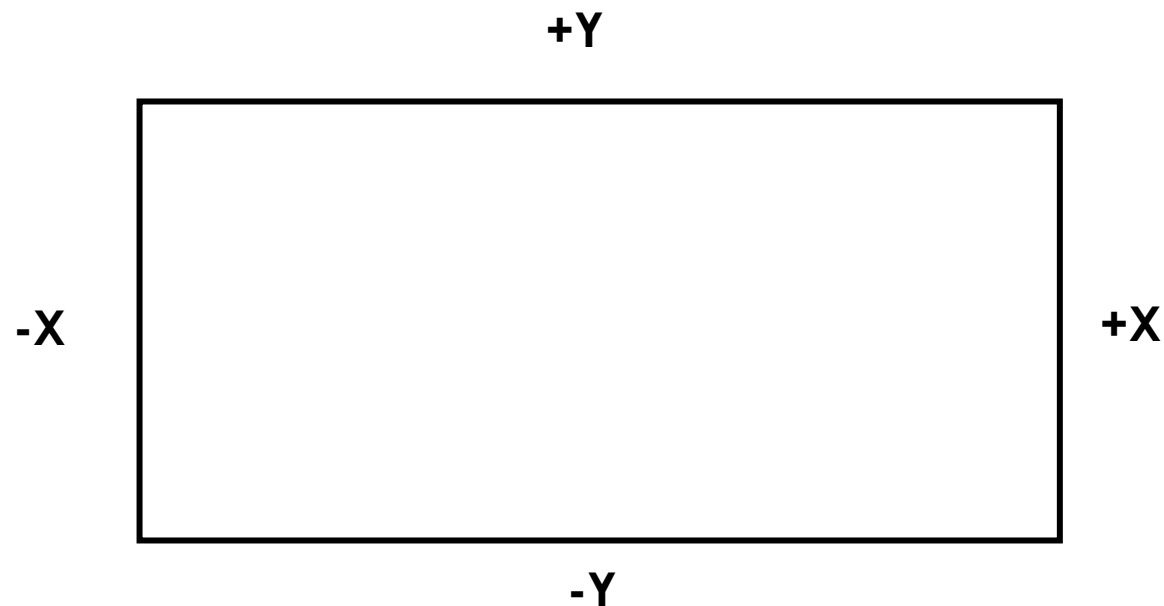
Text Input

```
global unit 1  
cuboid 30 0.715 -0.715 0.715 -0.715
```

or

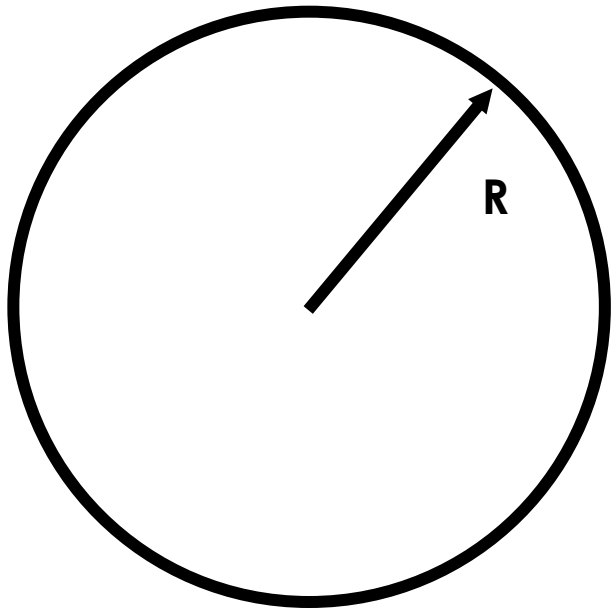
```
global unit 1  
cuboid 30 4p0.715
```

Fulcrum Visualization



CUBOID	BodyID	+X	-X	+Y	-Y
--------	--------	----	----	----	----

Cylinder

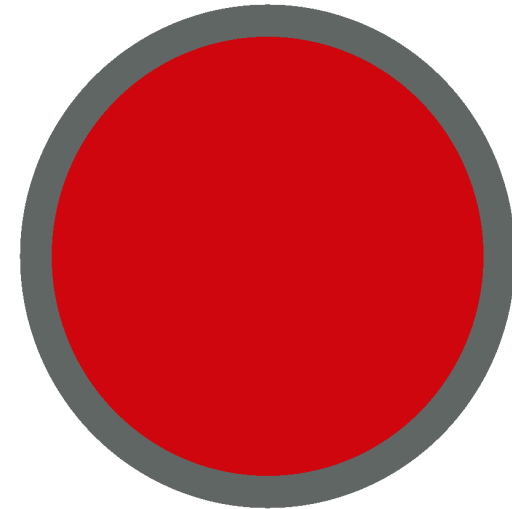


CYLINDER BodyID R

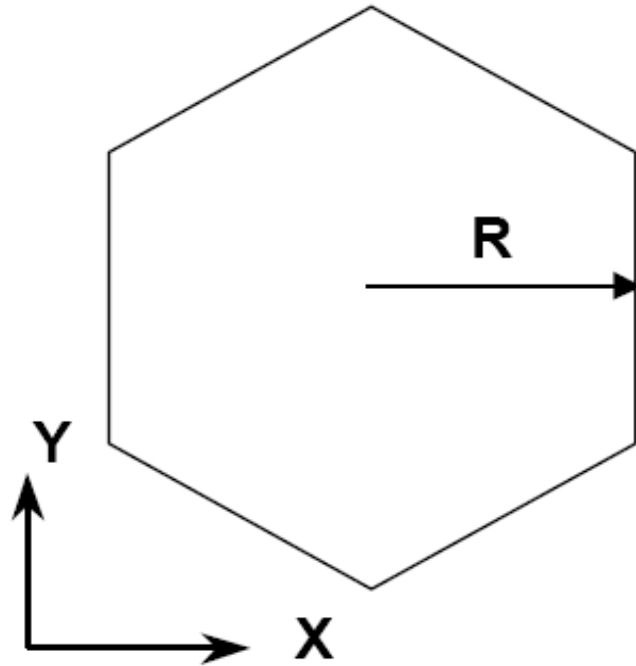
Text Input

```
global unit 1  
cylinder 10 0.4674  
cylinder 20 0.53595
```

Fulcrum Visualization



Hexagonal prism

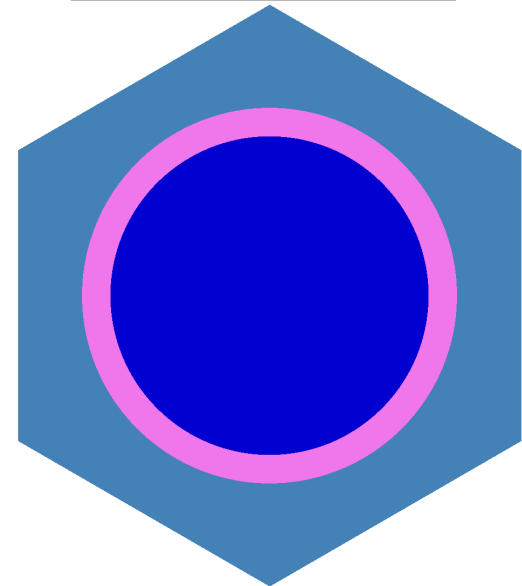


HEXPRISM BodyID R

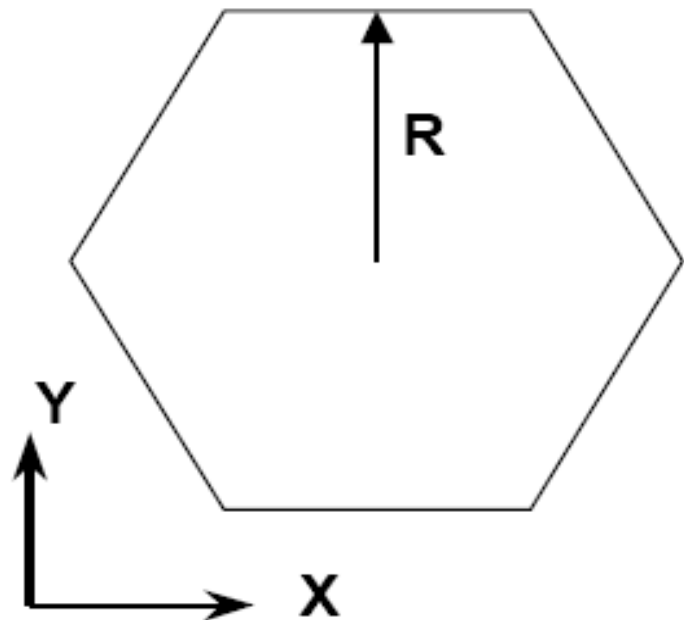
Text Input

```
global unit 1  
cylinder 1 0.386  
cylinder 2 0.455  
hexprism 3 0.61
```

Fulcrum Visualization



Rotated hexagonal prism

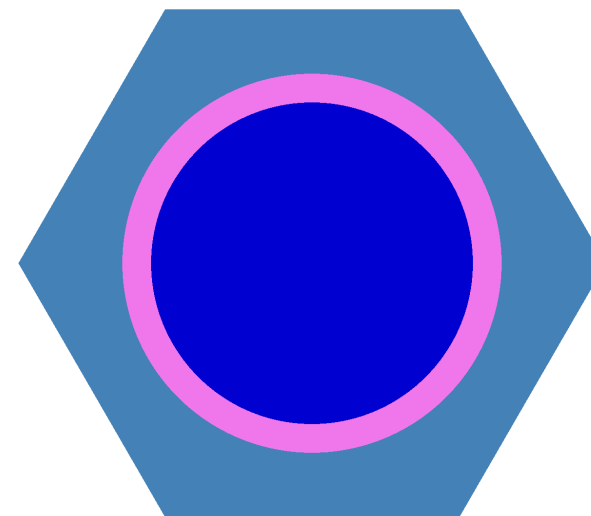


RHEXPRISM BodyID R

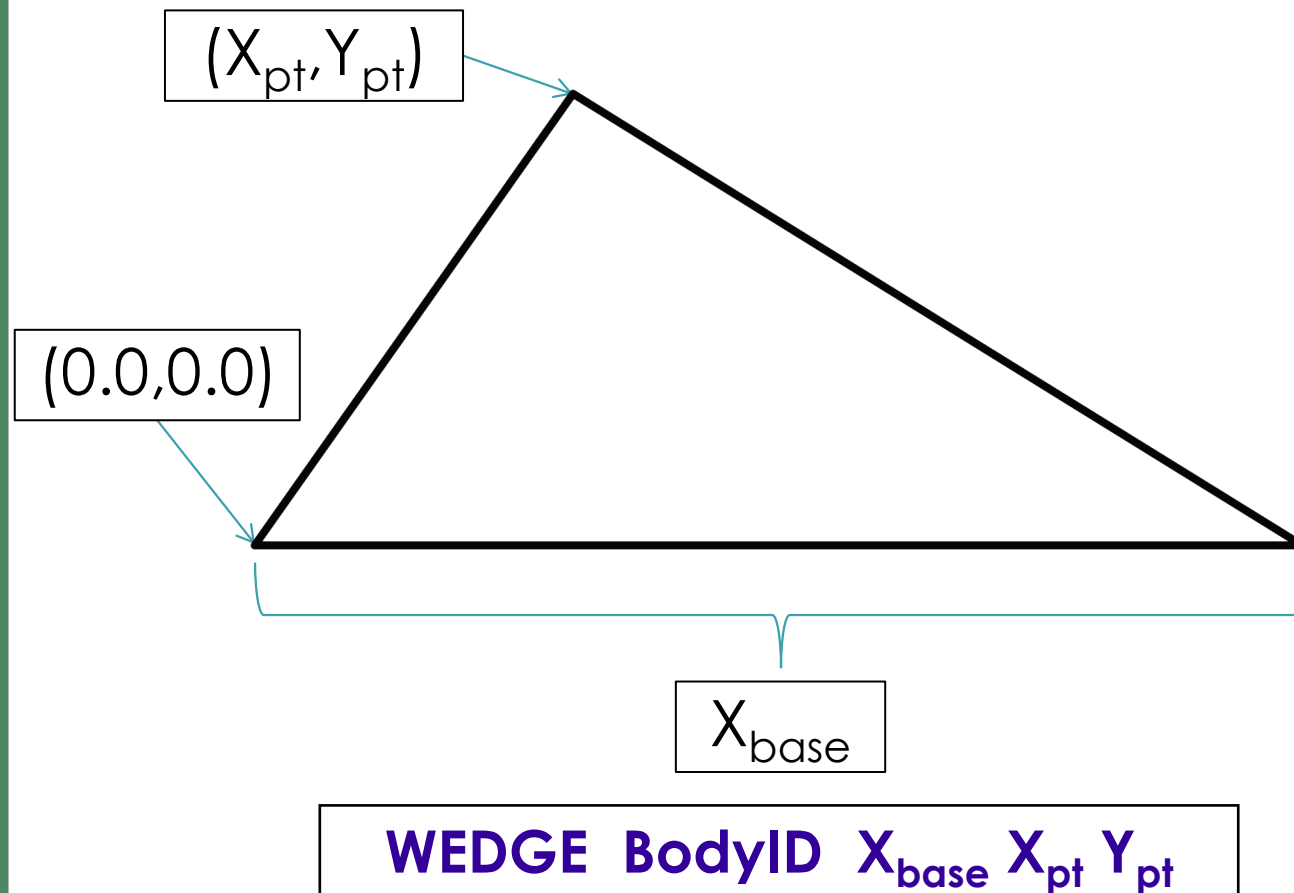
Text Input

```
global unit 1  
cylinder 1 0.386  
cylinder 2 0.455  
rhexprism 3 0.61
```

Fulcrum Visualization



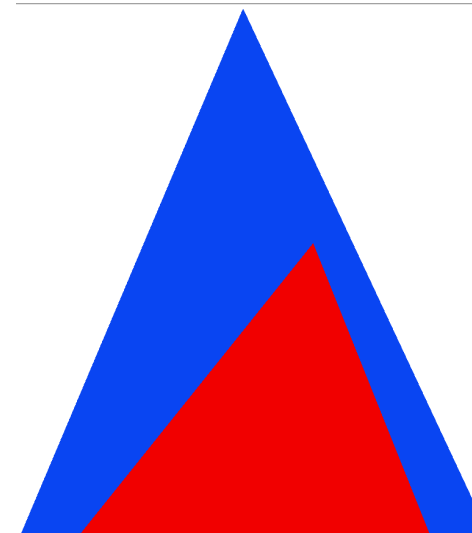
Wedge



Text Input

```
global unit 1
wedge 1 0.3 0.2 0.25 origin x=0.05
wedge 2 0.4 0.19 0.45
```

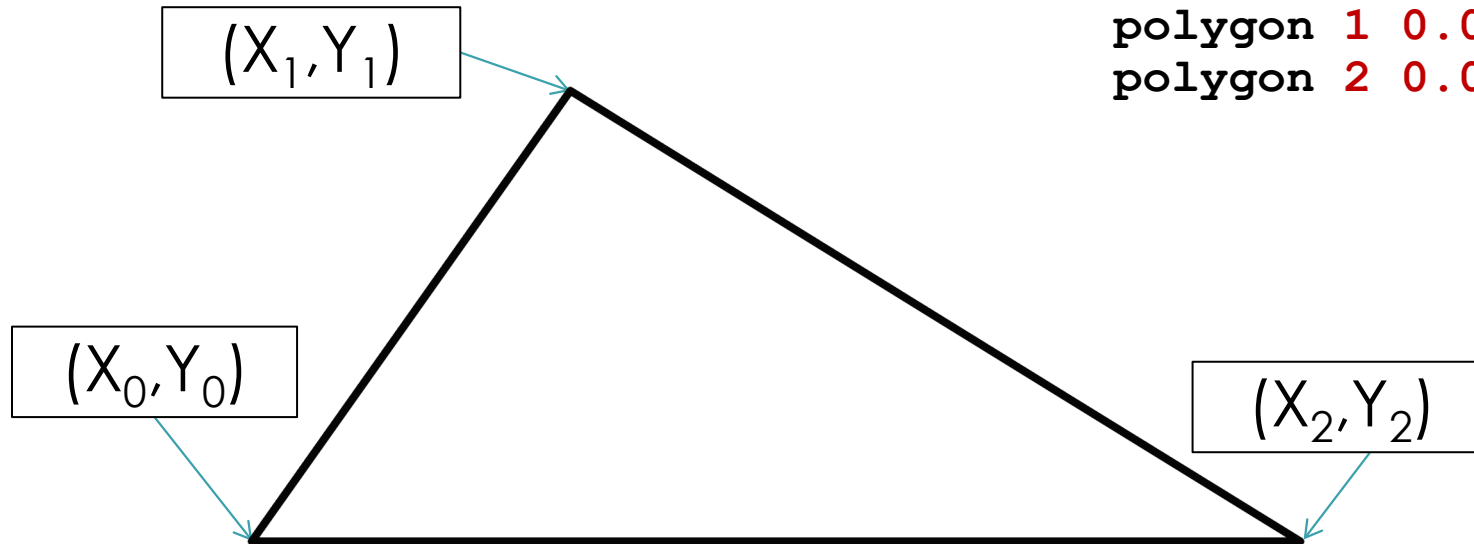
Fulcrum Visualization



Polygon (with an arbitrary number of sides)

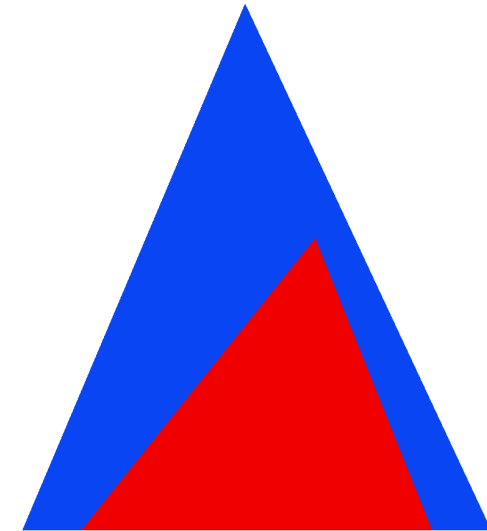
Text Input

```
global unit 1
polygon 1 0.05 0.0 0.25 0.25 0.35 0.0 0.05 0.0
polygon 2 0.0 0.0 0.19 0.45 0.4 0.0 0.0 0.0
```



POLYGON BodyID X_0 Y_0 X_1 Y_1 ... X_N Y_N X_0 Y_0

Fulcrum Visualization



GEOMETRY block – Examples

- Cuboid format: `cuboid bodyid xmax xmin ymax ymin`

- Cuboid example (1 cm by 2 cm rectangle, ID=3):

`cuboid 3 1.0 0.0 2.0 0.0`

- Cylinder format: `cylinder bodyid radius [sides=n]`

- Cylinder placed at (0,0) by default
- 12-sided polygon used by default to model curved surface

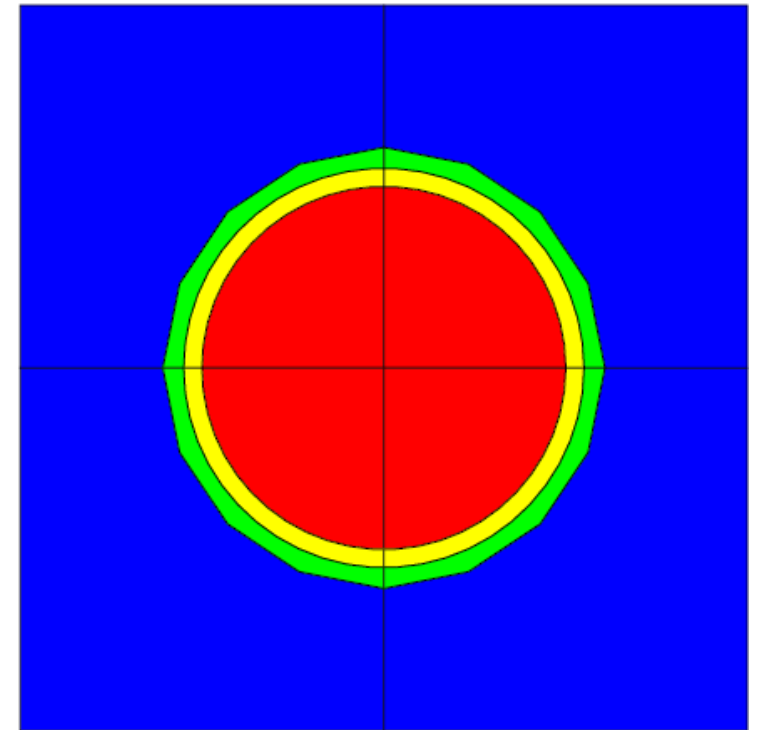
- Cylinder example (R=5cm, 16-sided polygon, ID=2):

`cylinder 2 5.0 sides=16`

GEOMETRY block – Shapes within a unit

Example:

```
global unit 1
  cylinder 10 0.5 sides=50 com='fuel'
  cylinder 20 0.55 sides=50 com='gas gap'
  cylinder 30 0.600 sides=16 com='clad'
  cuboid 40 1.0 -1.0 1.0 -1.0 com='moderator'
media 1 1 10
media 4 1 20 -10
media 2 1 30 -20
media 3 1 40 -30
boundary 40 2 2
```



Reminder: *sides* and *com* are optional; default is *sides*=12

GEOMETRY block – Media

- Shape input is used to define bodies within unit
- Media input follows shape specifications within a unit
- Media input is used to define mixtures associated with bodies
- Format:

media material_id bias_id region_def_vector

- The *bias_id* entry is not (currently) used but is kept for consistency with KENO and is **required**; use value of 1 for consistency with KENO

GEOMETRY block – Media

- Region definition vector
 - Describes the location of a given mixture (media) within a model
 - Provides list of shapes (by shape ID number) for which the media is either inside or outside
 - If a media is inside a body, the body is listed with a positive number
 - If a media is outside a body, the body is listed with a negative number

GEOMETRY block – Media

- Example:

media 1 1 10

media 4 1 20 -10

media 2 1 30 -20

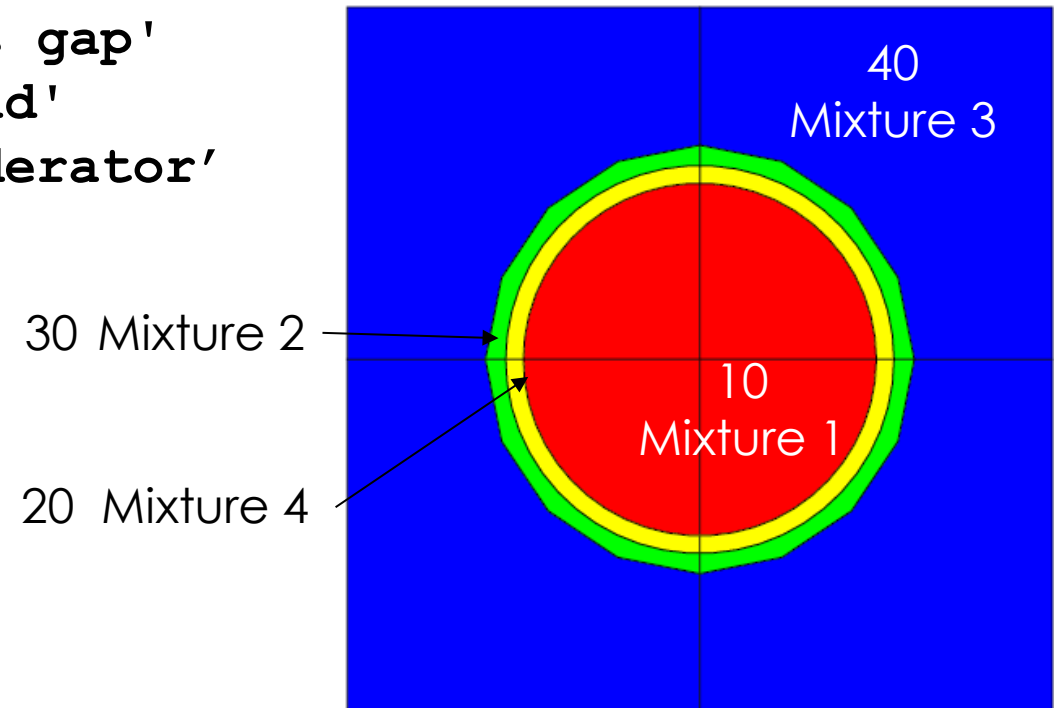
media 3 1 40 -30

- mixture 1 is inside body 10
- mixture 4 is inside body 20 “and” outside body 10
- mixture 2 is inside body 30 “and” outside body 20
- mixture 3 is inside body 40 “and” outside body 30

GEOMETRY block – Defining regions with media

Example:

```
global unit 1
cylinder 10 0.5 sides=50 com='fuel'
cylinder 20 0.55 sides=50 com='gas gap'
cylinder 30 0.600 sides=16 com='clad'
cuboid 40 1.0 -1.0 1.0 -1.0 com='moderator'
media 1 1 10
media 4 1 20 -10
media 2 1 30 -20
media 3 1 40 -30
boundary 40 2 2
```



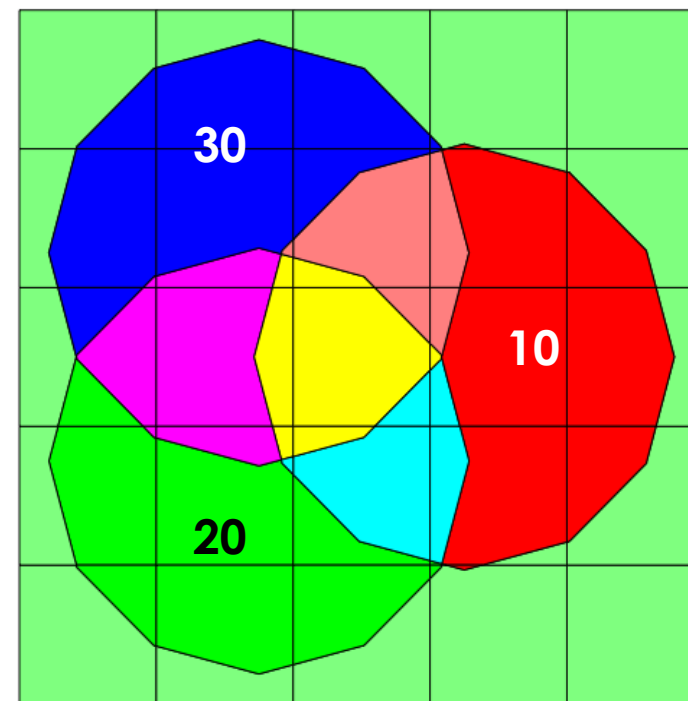
GEOMETRY block – Defining regions with media

Positive body number = Inside
Negative body number = Outside

```
global unit 1
cylinder 10 9.0 origin x= 4.5
cylinder 20 9.0 origin y=-4.5 x=-4.5
cylinder 30 9.0 origin y= 4.5 x=-4.5
cuboid 40 4p15
```

```
media 1 1 10 -20 -30
media 2 1 20 -10 -30
media 3 1 30 -10 -20
media 4 1 10 20 30
media 5 1 10 20 -30
media 6 1 20 30 -10
media 7 1 10 30 -20
media 8 1 40 -10 -20 -30
```

```
boundary 40 5 5
```



GEOMETRY block – Boundary

- Boundary definition follows all shape and media input
- Only one boundary specification may be input
- Boundary definition designates outermost body of unit (i.e., unit boundary)
- Boundary definition also provides the underlying grid for each unit
- Format:

boundary bodyid x_grid y_grid

Note: All defined regions in a unit must be within the outermost shape used as boundary. Do not "cut" a larger geometry with a boundary shape, as is often done in Monte Carlo codes.

GEOMETRY block – Boundary

- Example:

boundary 40 2 2

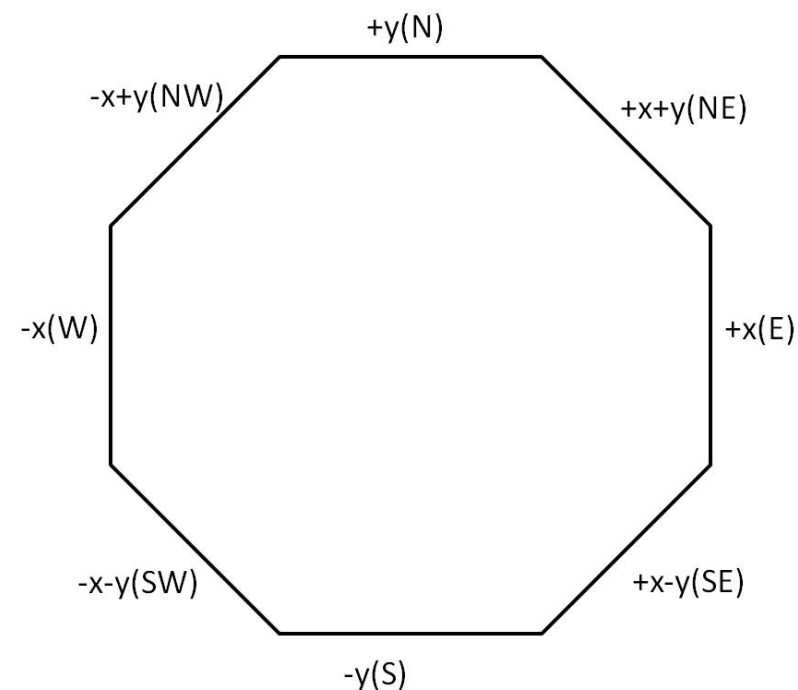
- Body 40 is the outermost body of this unit
- The unit has an underlying 2x2 rectangular grid

BOUNDS block

- BOUNDS block is used to define boundary conditions on global unit

- Eight boundaries exist:

$-x=$ (west side)	$+x+y=$ (northeast side)
$+x=$ (east side)	$+x-y=$ (southeast side)
$+y=$ (north side)	$-x-y=$ (southwest side)
$-y=$ (south side)	$-x+y=$ (northwest side)



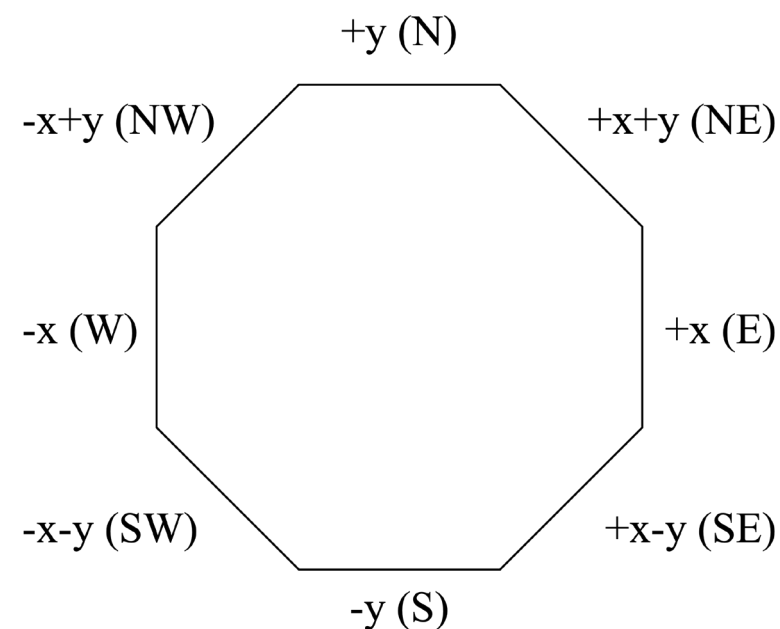
BOUNDS block

- Combined boundary conditions:

<code>+fc=, +xy=</code>	<code>(+x and +y)</code>
<code>-fc=, -xy=</code>	<code>(-x and -y)</code>
<code>xyf=</code>	<code>(four (N/S/E/W) faces)</code>
<code>xfc=</code>	<code>(+x and -x faces)</code>
<code>yfc=</code>	<code>(+y and -y faces)</code>
<code>all=</code>	<code>(all eight faces)</code>

BOUNDS block

- NEWT supports the use of cuboid, hexprism, and wedge shapes to define outer boundaries
- Available boundary conditions: vacuum (vac/vacuum), reflected (refl/reflected/mirror), white, periodic
- The default boundary conditions depend on the global boundary:
 - Cuboid: reflected in all directions
 - Hexprism: reflected in W and E, white in NE, SE, SW, NW
 - Wedge: reflected in W, E, S, N white in NE, SE, SW, NW



BOUNDS block

- Bounds block always comes after geom block
 - *SCALE is generally moving away from requiring specific orders of blocks*
 - *However, in this case, it is an actual requirement; while Fulcrum does not report a validation error, the calculation will crash if this order is not observed*

- Examples:

```
read bounds
  all=vac
end bounds
```

```
read bounds
  xfc=periodic
  yfc=refl
end bounds
```

```
read bounds
  +x=vac +y=vac -x=refl -y=refl
end bounds
```

Workshop

- Part I: Basic input

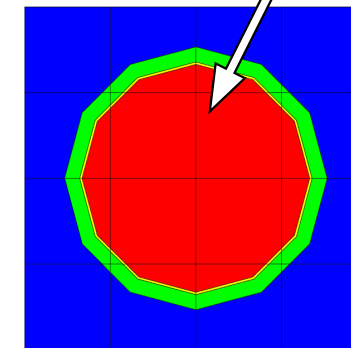
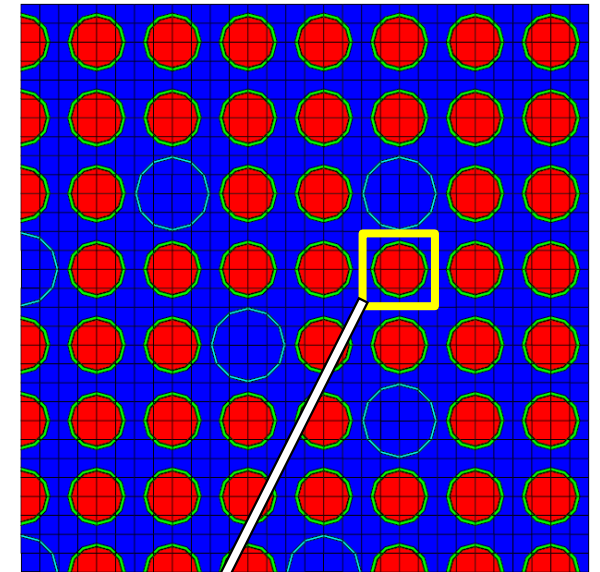


Problem 1: PWR pin cell

Create a T-NEWT input for a fuel pin in a Westinghouse 15x15 assembly

- General data:
 - t-newt, v7-56, centrm
- Composition and cell geometry:
 - Fuel: UO_2 (5.5% enr), 95%TD $R=0.4674\text{cm}$, 870K
 - Clad: Zirc-2, default density, $R=0.53595\text{ cm}$, 595K
 - Moderator: 700 ppm borated water, 0.7135 g/cc, 595 K, pitch=1.43 cm
 - squarepitch cell
- NEWT parameters:
 - echo on, drawit on (yes), sn=8
- NEWT materials:
 - Use P1 scattering for fuel and cladding
 - Use P2 scattering for the water
 - Give the materials a label using the com parameter
- NEWT geometry:
 - Use 4x4 mesh with reflective boundary conditions
- Compute k-eff

Westinghouse 15x15 assembly



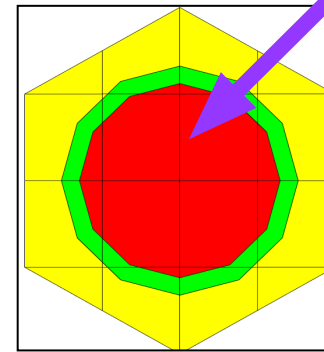
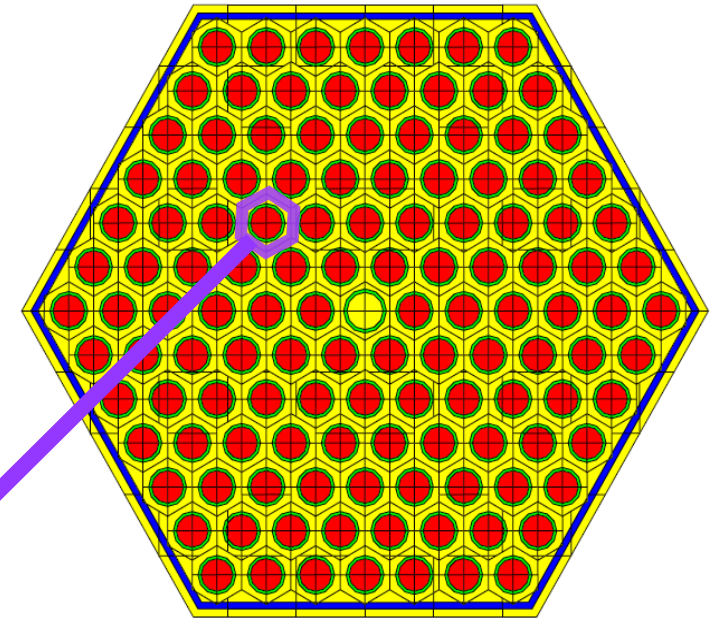
Pin cell

Problem 2: VVER pin cell

Create a T-NEWT input for a fuel pin in a VVER-440 assembly

- Open “vver_pincell_xsproc.inp” and save as “vver_pincell.inp”
- General data:
 - t-newt, centrm
- NEWT parameters:
 - echo on
 - drawit on
 - npolar=3, nazim=3 (must use product quadrature set, with nazim=3 or 6 or 9 or ...)
- NEWT materials:
 - Use P1 scattering for fuel and cladding
 - Use P2 scattering for the water
 - Give the materials a label through the com parameter
- NEWT geometry:
 - hexprism shape
 - 4x4 mesh
 - reflective boundary conditions in all directions
- Compute k-eff

VVER-440 assembly



Pin cell

Solution 1: PWR pin cell

T-NEWT input for a fuel pin in a Westinghouse 15x15 assembly

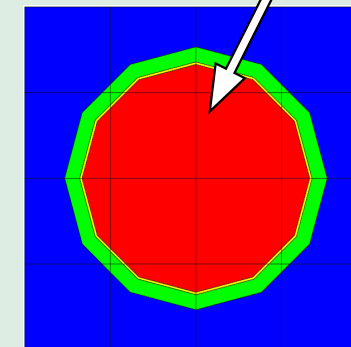
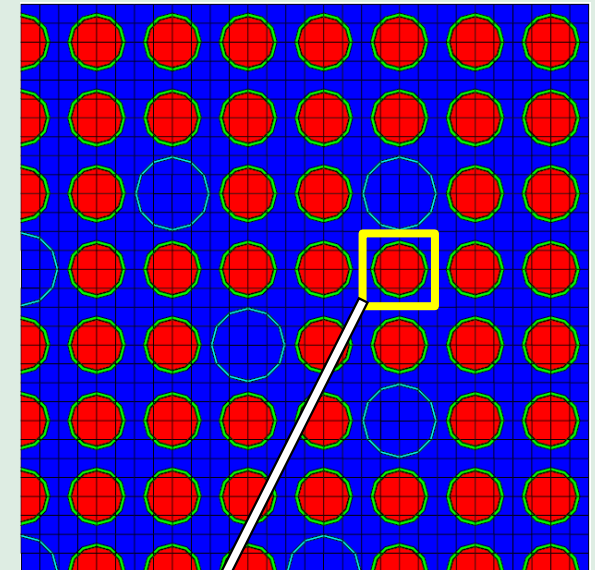
k-eff =1.36500

Runtime 38 seconds

```
=t-newt parm=(centrm)
PWR pin cell
v7-56
read composition
uo2 1 0.95 870
92235 5.5
92238 94.5 end
zirc2 20 1 595 end
h2o 30 den=0.7135 1.0000 595 end
boron 30 den=0.7135 0.0007 595 end
end composition
read celldata
latticecell squarepitch
  fuelr=0.4674 1
  cladr=0.53595 20
  hpitch=0.715 30 end
end celldata
```

```
read model
read parameter
drawit=yes
echo=yes
sn=8
end parameter
read materials
mix=1 pn=1 com='fuel, 5.5% enriched' end
mix=20 pn=1 com='clad, zirc4' end
mix=30 pn=2 com='mod, h2o + boron' end
end materials
read geometry
global unit 1
com='fuel pin, 5.5% enr'
cylinder 10 0.4674
cylinder 20 0.53595
cuboid 30 0.715 -0.715 0.715 -0.715
media 1 1 10
media 20 1 -10 20
media 30 1 -20 30
boundary 30 4 4
end geometry
end model
end
```

Westinghouse 15x15 assembly



Pin cell

Solution 2: VVER pin cell

T-NEWT input for a fuel pin in a VVER-440 assembly

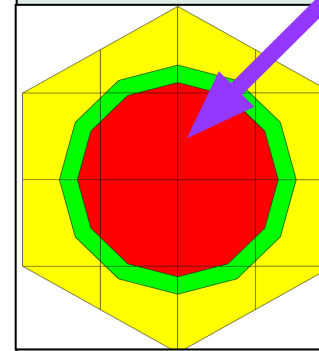
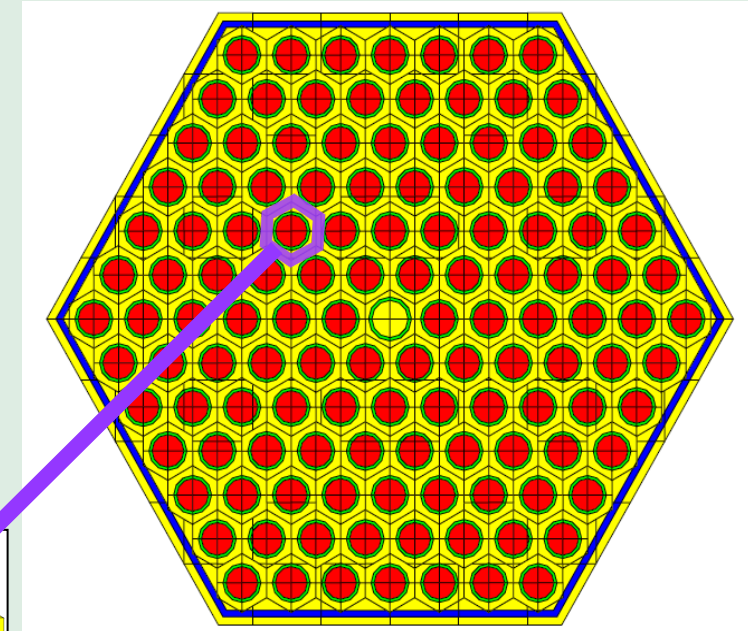
k-eff = 1.24853

Runtime 26 seconds

```
=t-newt parm=(centrm)
vver pin cell
v7-56
read composition
uo2 1 0.95 900
92235 4.0
92238 96.0 end
zirc2 20 1 600 end
h2o 30 den=0.75 1.0000 540 end
boron 30 den=0.75 0.0009 540 end
end composition
read celldata
latticecell triangpitch
  fuelr=0.386 1
  cladr=0.455 20
  hpitch=0.61 30 end
end celldata
```

```
read model
read parameter
drawit=yes
echo=yes
nazim=3
npolar=3
end parameter
read materials
mix=1 pn=1 com='fuel' end
mix=20 pn=1 com='clad' end
mix=30 pn=2 com='moderator' end
end materials
read geometry
global unit 1
cylinder 1 0.386
cylinder 2 0.455
rhexprism 3 0.61
media 1 1 1
media 20 1 2 -1
media 30 1 3 -2
boundary 3 4 4
end geometry
read bnds
all=mirror
end bnds
end model
end
```

VVER-440 assembly



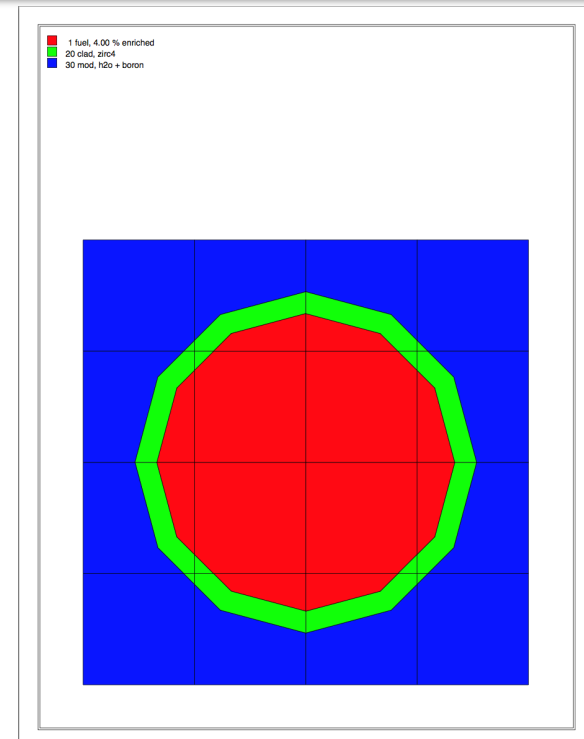
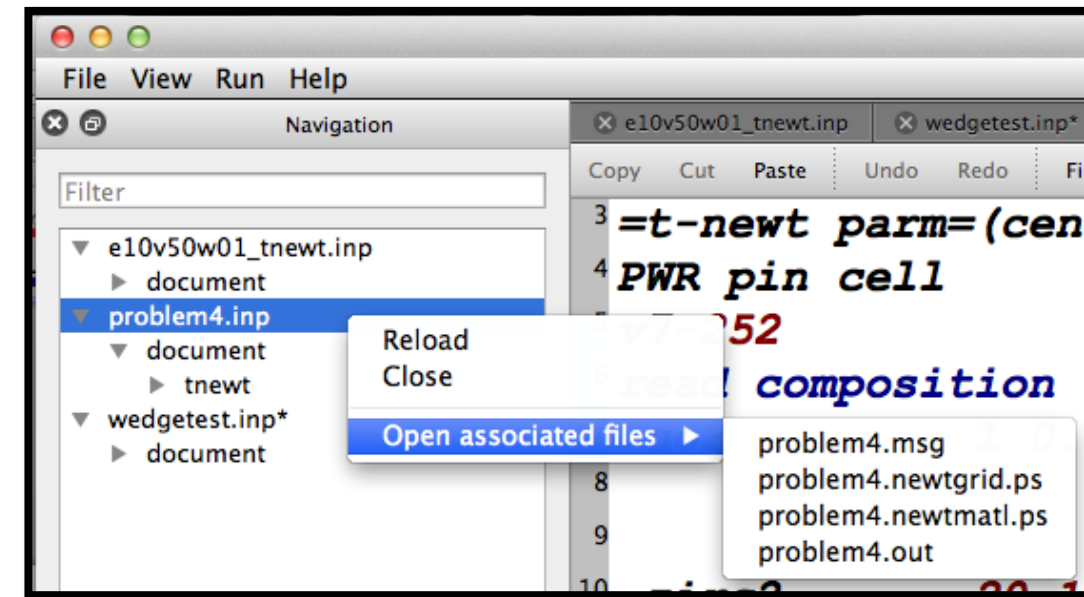
Pin cell

NEWT PostScript files



PostScript plot files

- NEWT provides simple PostScript plot files for input review and output analysis
- Macintosh and Unix machines generally support viewing or PDF conversion of .ps files natively
- Windows systems can:
 - Install 2 freeware packages
 - Ghostscript
 - Ghostview
 - Use Adobe Distiller in full version of Acrobat (not Acrobat Reader)
- Fulcrum will open the PostScript if a default postscript converter is available on the system

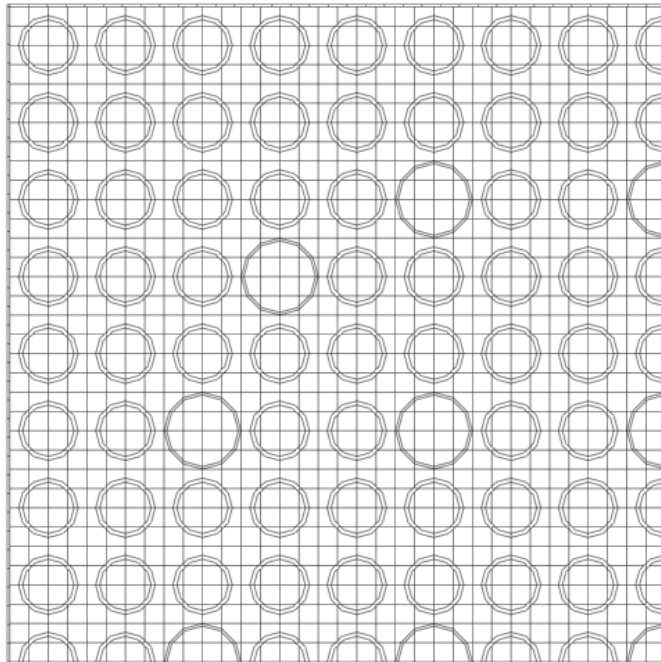


PostScript plot files

- Generation of PostScript plots is enabled in NEWT's PARAMETER block
 - **drawit=yes** will generate two plot files:
 - `${BASENAME}.newtgrid.ps` – geometry outline and grid
 - `${BASENAME}.newtmatl.ps` – geometry with materials filled into regions, and grid
 - **prtflux=yes** generates one or two plot files
 - `${BASENAME}.fluxplot_Ng.ps` – where N is the number of energy groups in the input library
 - `${BASENAME}.fluxplot_Bg.ps` – where B is the number of energy groups in the collapsed library (if collapse was used)

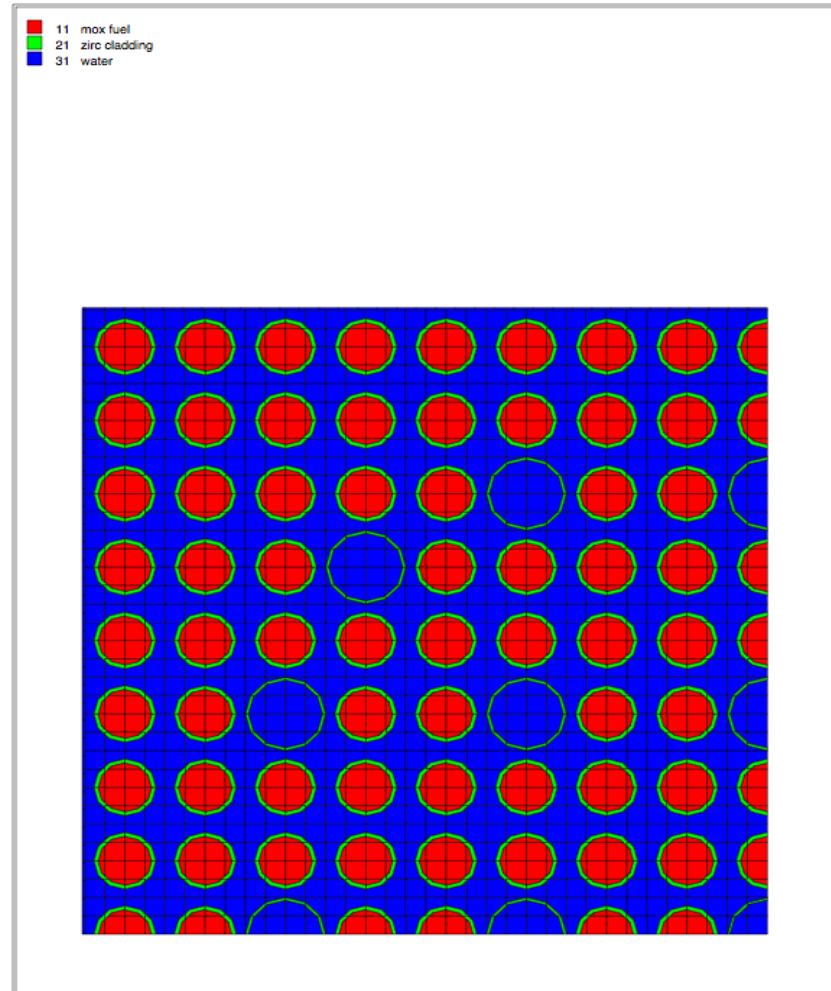
filename.newtgrid.ps

Generated first if a consistent geometry model is read.



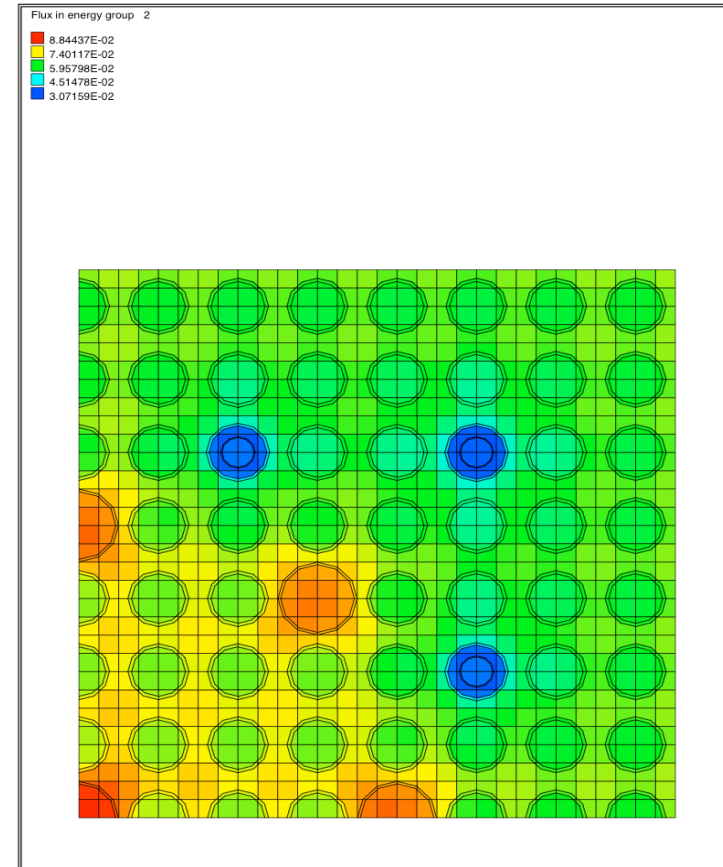
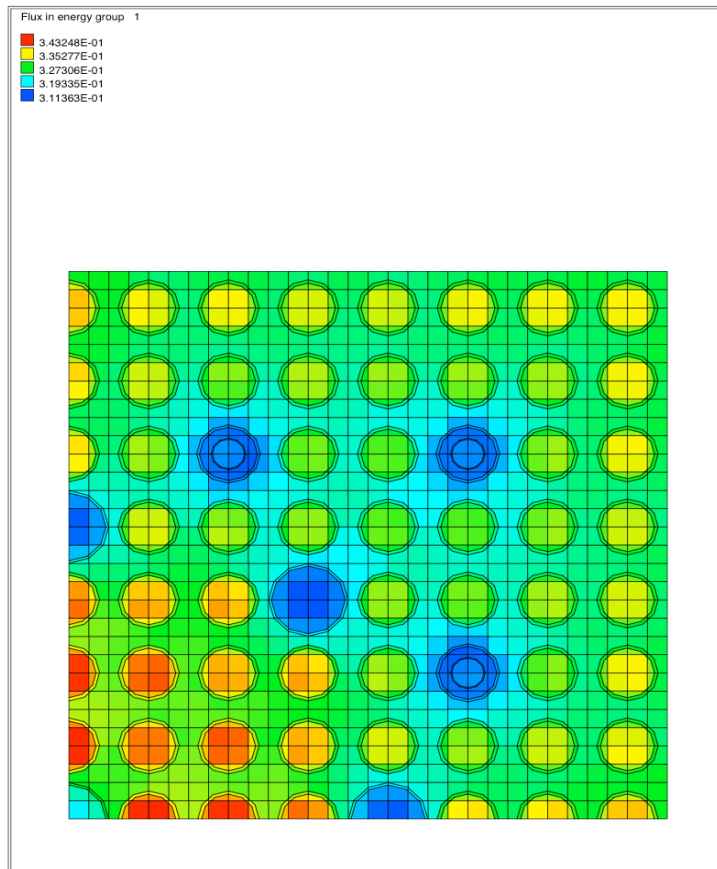
filename.newtmatl.ps

Generated second, after media placement operations are completed.

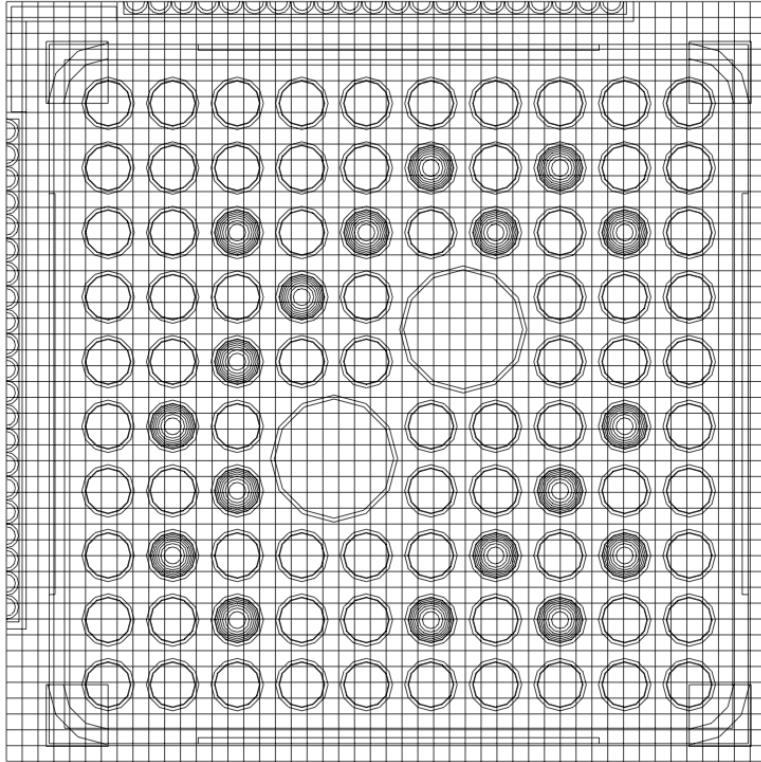


fluxplot_Ng.ps

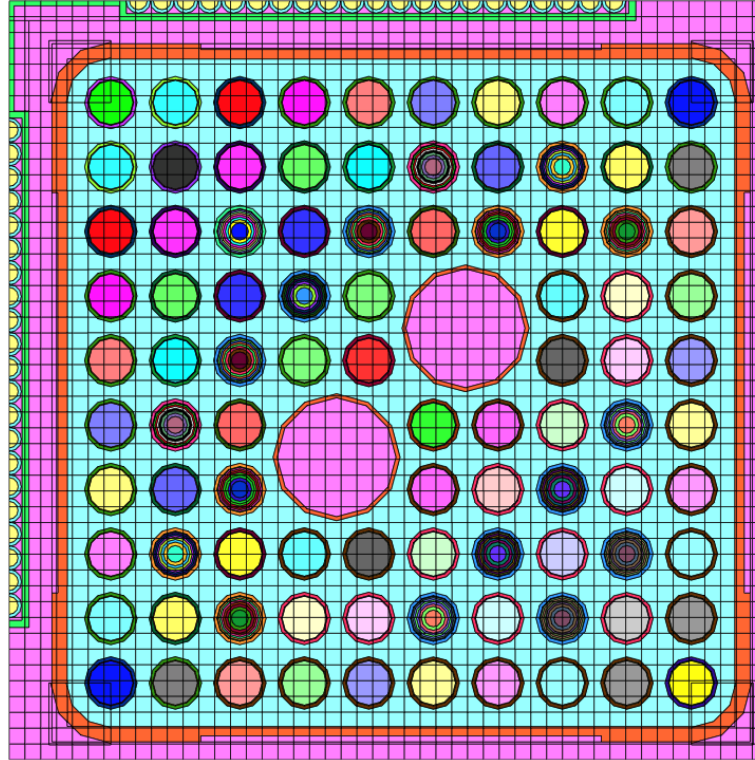
Multipage file, one plot for each energy group



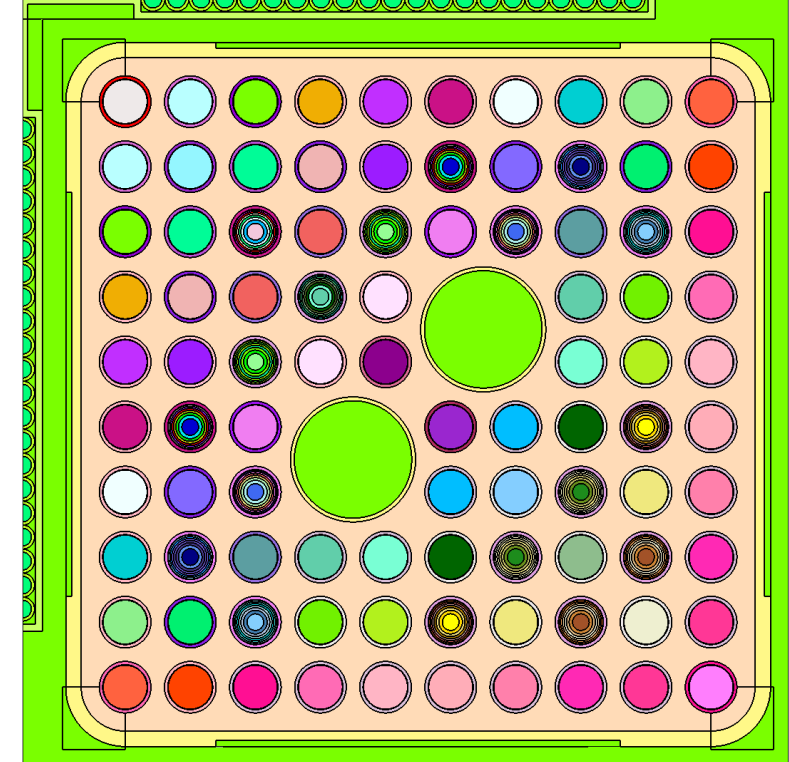
NEWT vs. Fulcrum visualization of a BWR fuel assembly



Grid plot



Material plot



Fulcrum plot

The PostScript files visualize the actual NEWT geometry
→ **This image must always be inspected before running a problem**

NEWT input

- Part II: Shape modifiers



Geometry block – Shape modifiers

- Every shape description may be followed by modifiers that affect that shape:
 - **origin** (moves origin of hexagons and cylinders)
 - **chord** (removes portion of hexagons and cylinders)
 - **rotate** (rotates body around origin for all shapes except cylinders)
 - **com** (comment – all shapes)
 - **sides** (number of sides to approximate cylinder)

Geometry block – Geometry modification

These options are always performed in the order listed below*

	Keyword	Input Values
1	rotate	A1=
2	origin	X=, Y= (either or both)
3	chord	+X=, -X=, +Y=, -Y= (only one)
4	<i>none</i>	SIDES=

Exceptions:

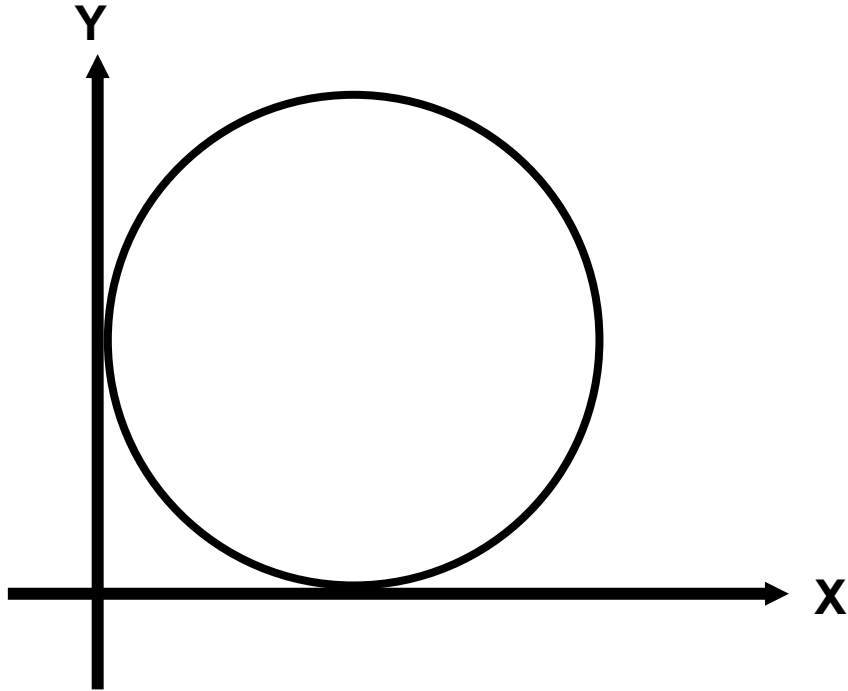
- Chord does not apply to cuboids
- Rotate does not apply to cylinders

*Note: This order is not identical to the order of geometry modifications in KENO-VI models

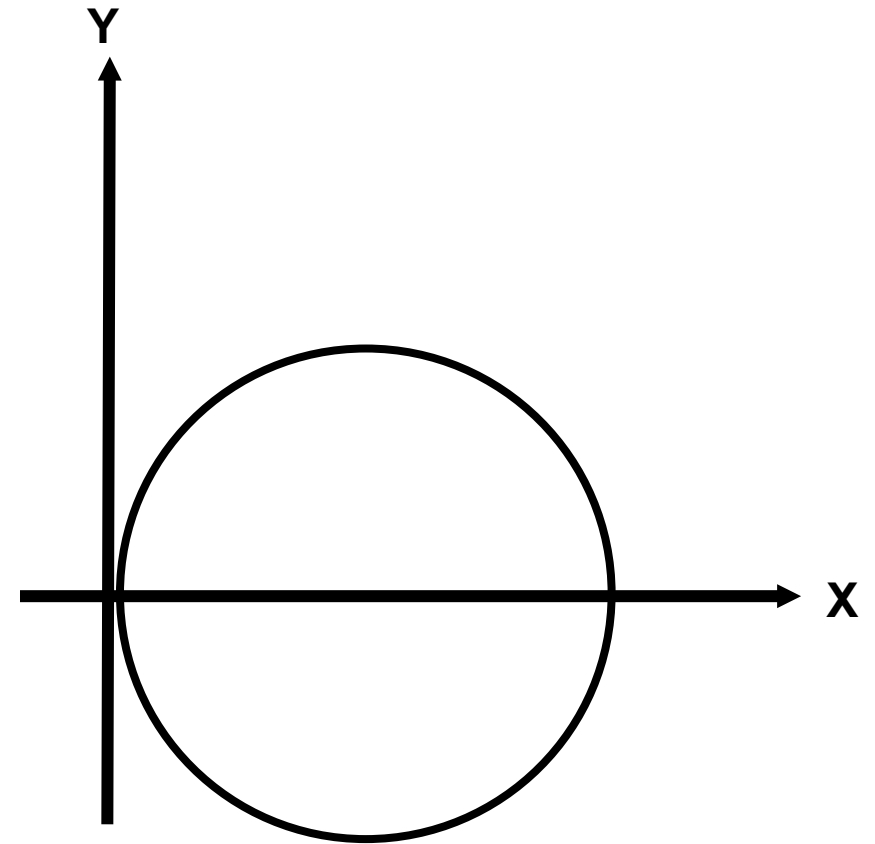
Origin

- Translates the origin of a volume to a new specified position; only non-zero data need to be entered
- Applies to the shape or hole record that immediately precedes it
- Contains the following components
 - **origin** keyword
 - Subordinate keywords (X=, Y=)

Origin example



cylinder 1 10.0 origin x=10.0 y=10.0

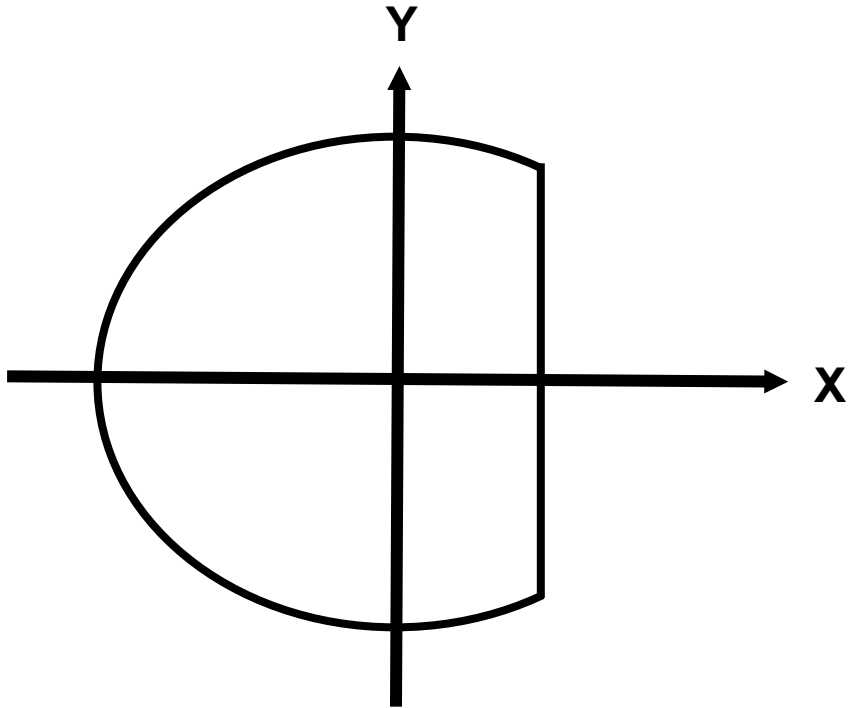


cylinder 1 10.0 origin x=10.0

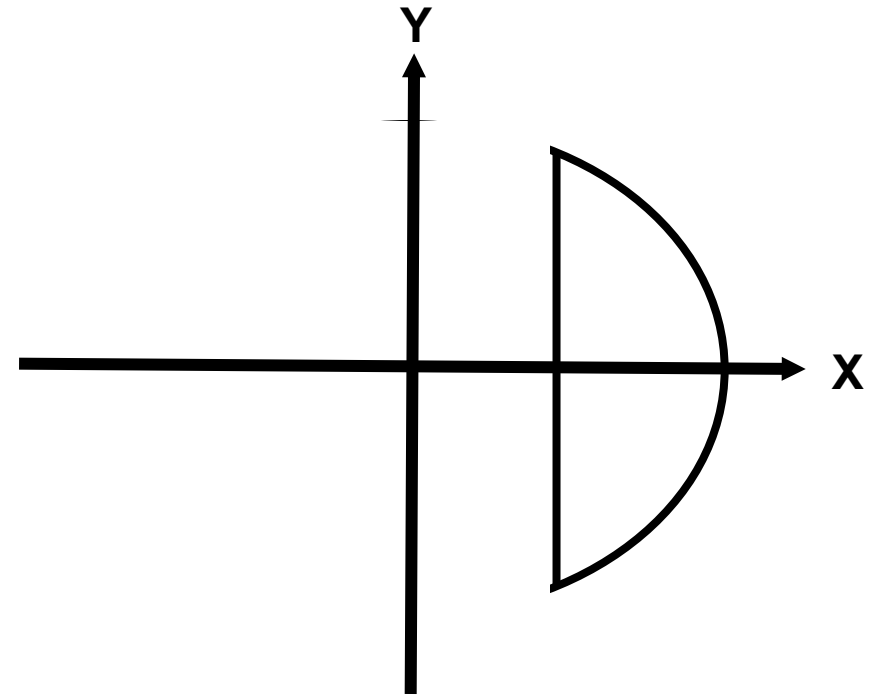
Chord

- Truncates a geometric volume with a plane perpendicular to a major axis
- Applies to the shape that immediately precedes it
- Applies to the shape after translation
- May not be used for cuboids
- Contains the following components
 - **chord** keyword
 - A single subordinate keyword
(+X=, -X=, +Y=, -Y=)

Chord examples

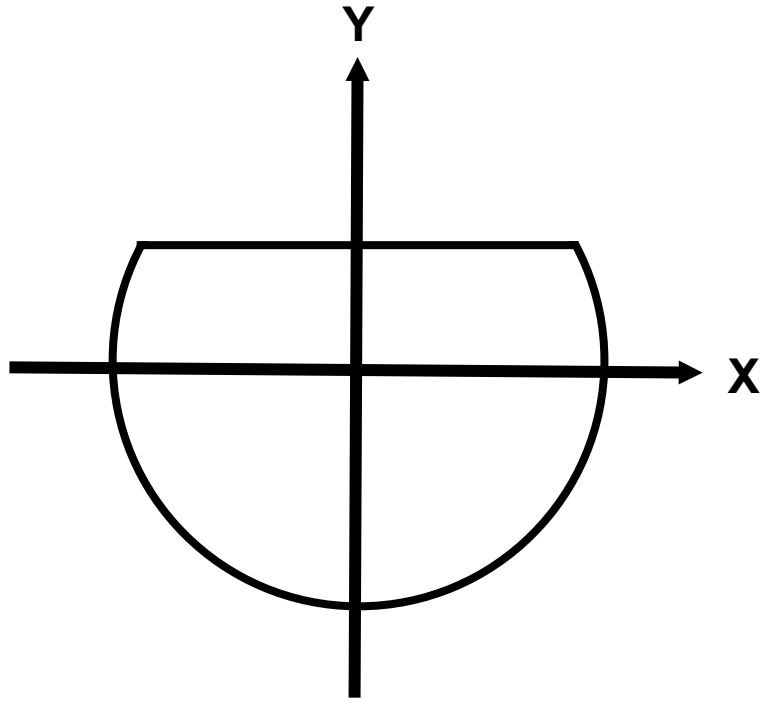


cylinder 1 10.0 chord $-x=5.0$

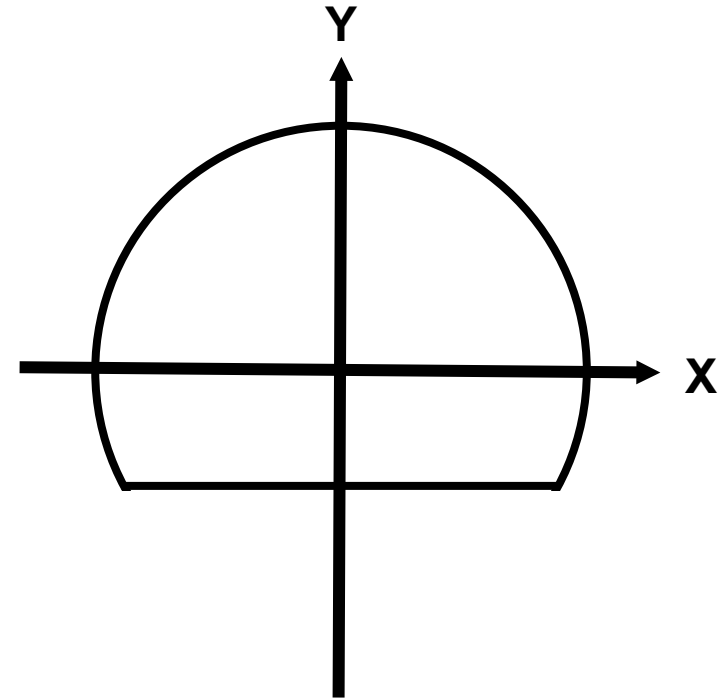


cylinder 2 10.0 chord $+y=5.0$

Chord examples

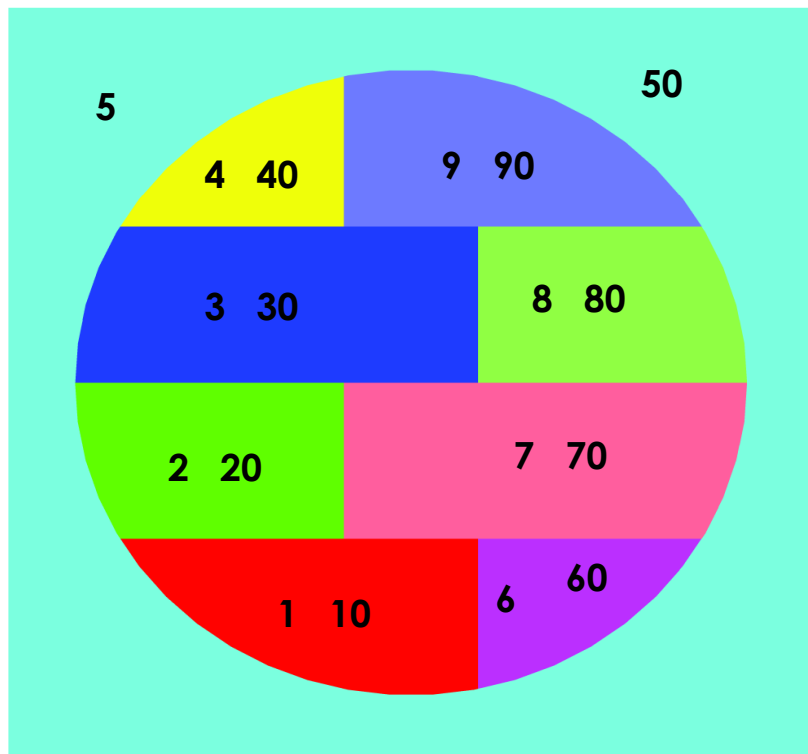


cylinder 3 10.0 chord $-y=5.0$



cylinder 4 10.0 chord $+y=-5.0$

Chord examples



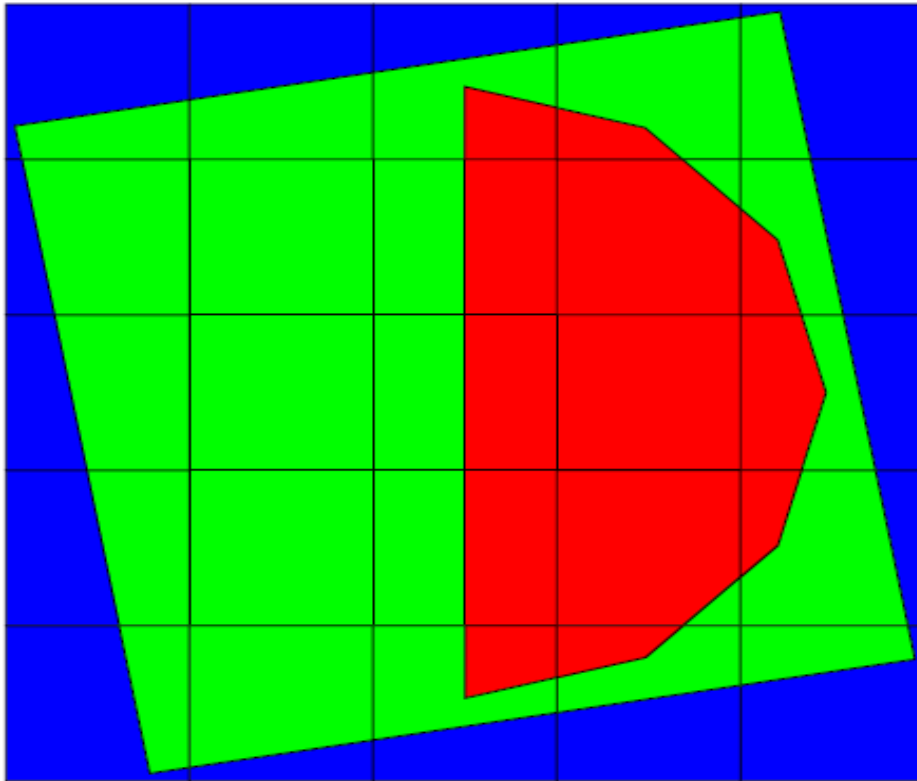
```

global unit 1
cylinder 10 10.0 chord -y=-5.0 chord -x= 2.0 sides=50
cylinder 20 10.0 chord +y=-5.0 chord -y= 0.0 chord -x=-2.0 sides=50
cylinder 30 10.0 chord +y= 0.0 chord -y= 5.0 chord -x= 2.0 sides=50
cylinder 40 10.0 chord +y= 5.0 chord -x=-2.0 sides=50
cylinder 60 10.0 chord -y=-5.0 chord +x= 2.0 sides=50
cylinder 70 10.0 chord +y=-5.0 chord -y= 0.0 chord +x=-2.0 sides=50
cylinder 80 10.0 chord +y= 0.0 chord -y= 5.0 chord +x= 2.0 sides=50
cylinder 90 10.0 chord +y= 5.0 chord +x=-2.0 sides=50
cuboid 50 4p12.0
media 5 1 50 -10 -20 -30 -40 -60 -70 -80 -90
media 1 1 10
media 2 1 20
media 3 1 30
media 4 1 40
media 6 1 60
media 7 1 70
media 8 1 80
media 9 1 90
boundary 50 12 12
    
```

Rotate

- Rotates a volume counter-clockwise about the origin
- Rotate operation occurs after origin operation
- Applies to the shape (or hole) that immediately precedes it
- Contains the following components
 - **rotate** keyword
 - Subordinate keyword (A1=)
- Note: Bodies are rotated, but NEWT's grid lines are NOT rotated

Rotate example



```
read geom
```

```
global unit 50
```

```
cylinder 10 10.0 chord +x=0.0
```

```
cuboid 20 4p11.0 rotate a1=10
```

```
cuboid 30 4p13.0
```

```
media 1 1 10
```

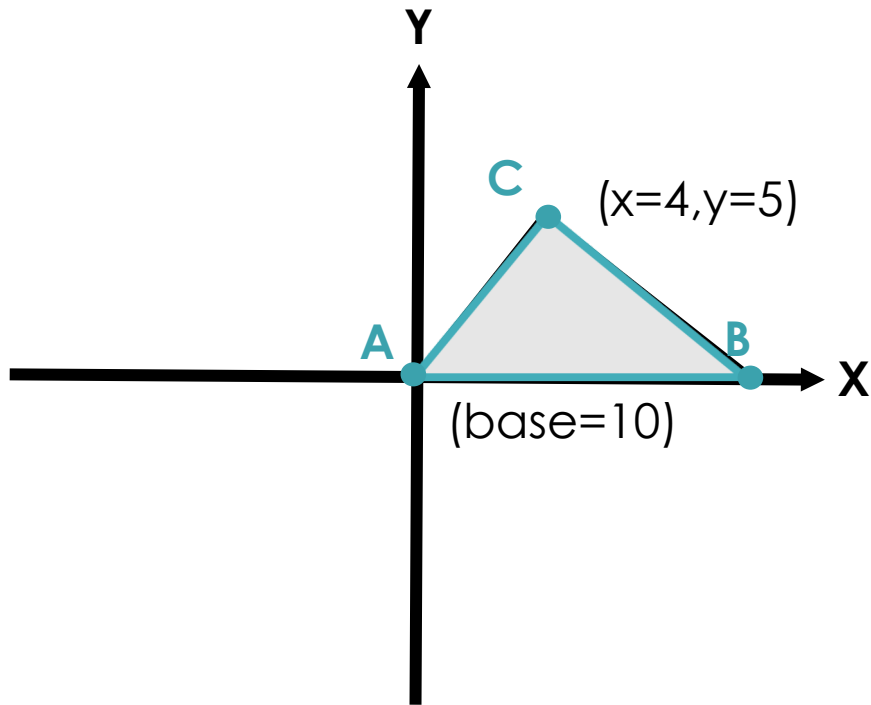
```
media 2 1 20 -10
```

```
media 3 1 30 -20
```

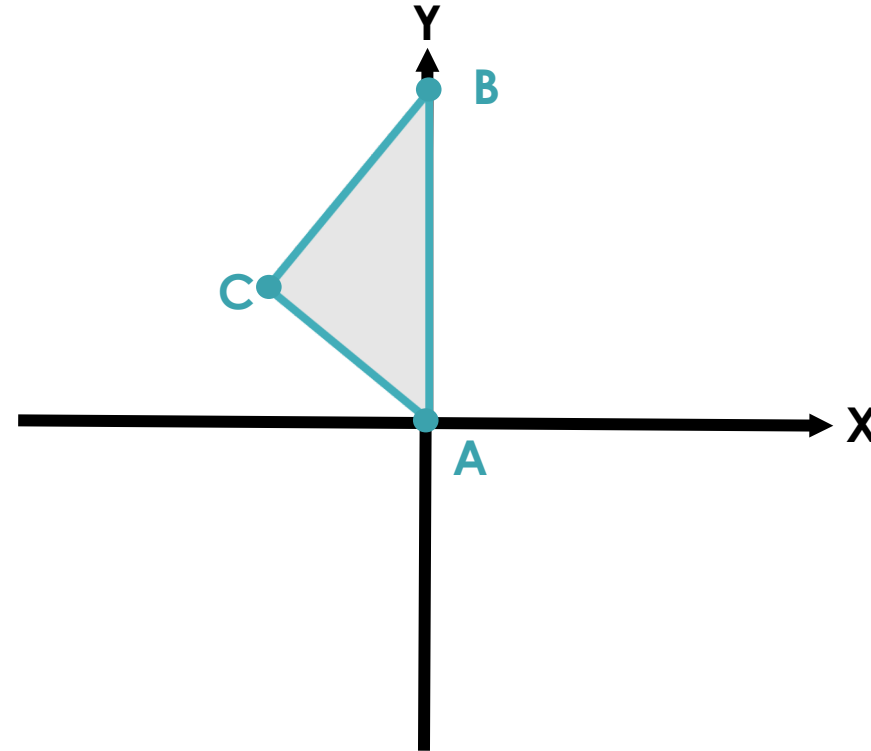
```
boundary 30 5 5
```

```
end geom
```

Rotate example



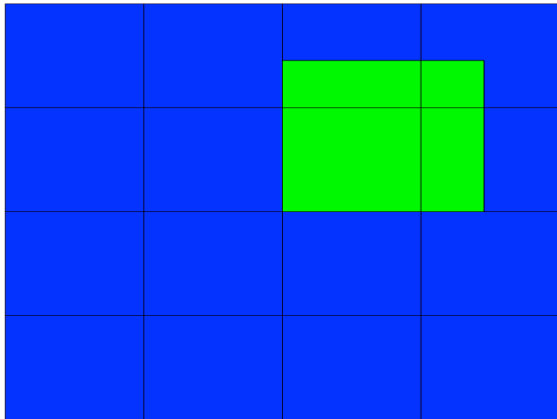
wedge 1 10 4 5



wedge 1 10 4 5 rotate a1=90

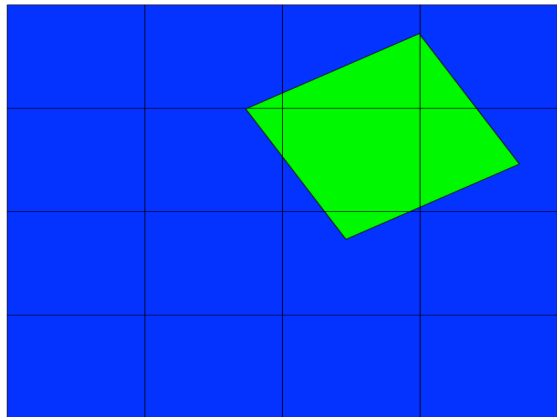
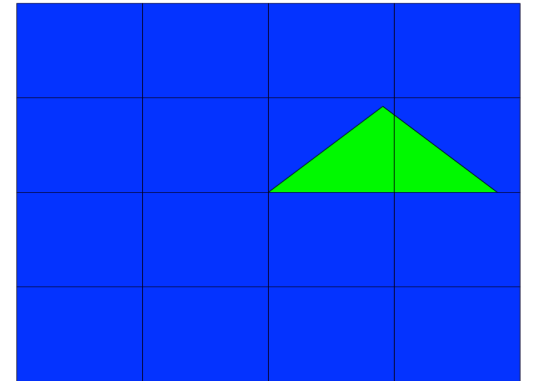
Rotate examples

- Cuboids are rotated around the body centroid
- Wedges are rotated about the origin



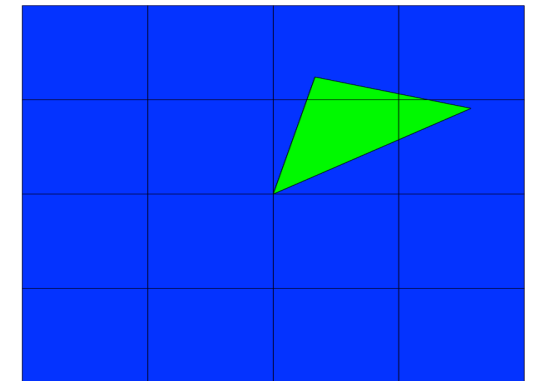
```
cuboid 20 0.8 0.0 0.8 0.0
```

```
wedge 20 1 0.5 0.5
```



```
cuboid 20 0.8 0.0 0.8 0.0  
rotate a1=30
```

```
wedge 20 1 0.5 0.5  
rotate a1=30
```



Workshop

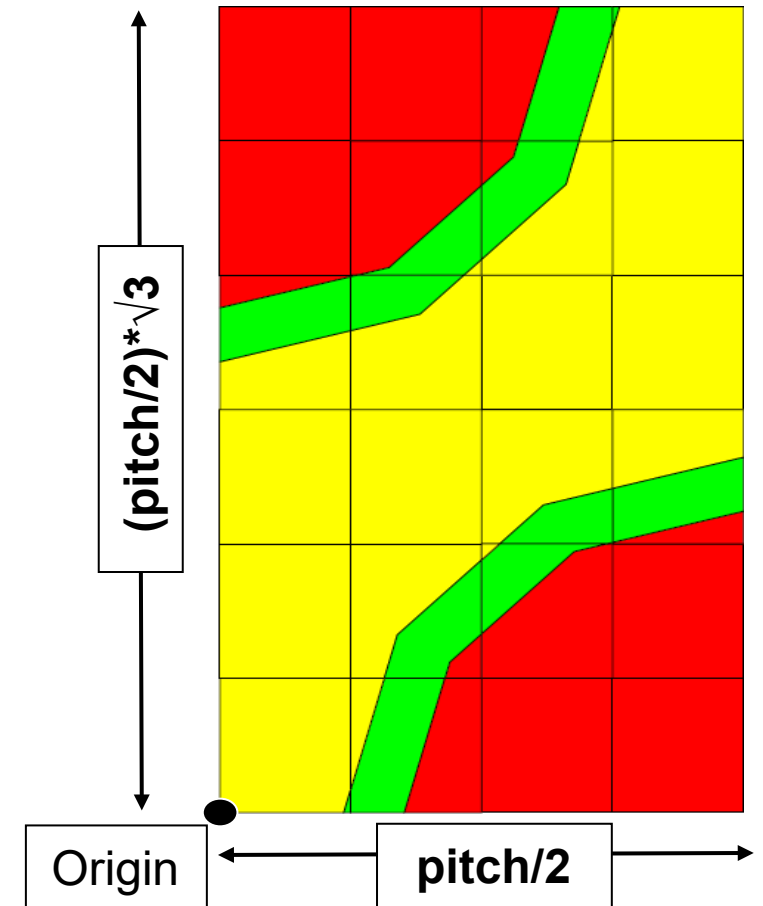
- Part II: Shape modifiers



Problem 3: VVER pin cell variation

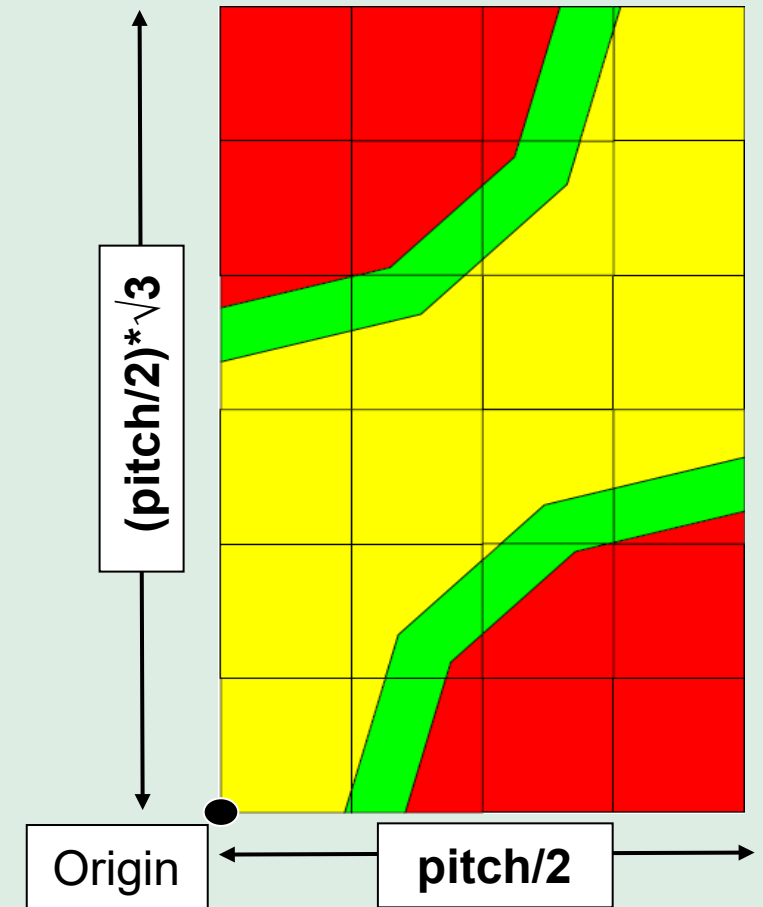
Create a T-NEWT model for a fuel pin in a VVER-440 assembly using chords and origin modifiers

- Open “vver_pincell.inp” and save as “vver_pincell_cuboid.inp”
- NEWT geometry:
 - Use **cuboid** geometry rather than hexprism
 - Use cuboid origin as shown in picture
 - Use **chord** and **origin** modifiers appropriately
 - Use 4x6 mesh with reflective boundary conditions
- Compute k-eff
- Why would k-eff be different than before?
- Why is chord needed even though the outermost boundary is chopping off the rest of the cylinders in the unit cell?



Solution 3: VVER pin cell variation

- The NEWT geometry uses
 - **origin** to move the cylinders to the corners
 - **chord** to cut the unused sections of the cylinders
- Cuboid model: k-eff = 1.24820
- Previous hex model: k-eff = 1.24853
- $\Delta k = 34$ pcm

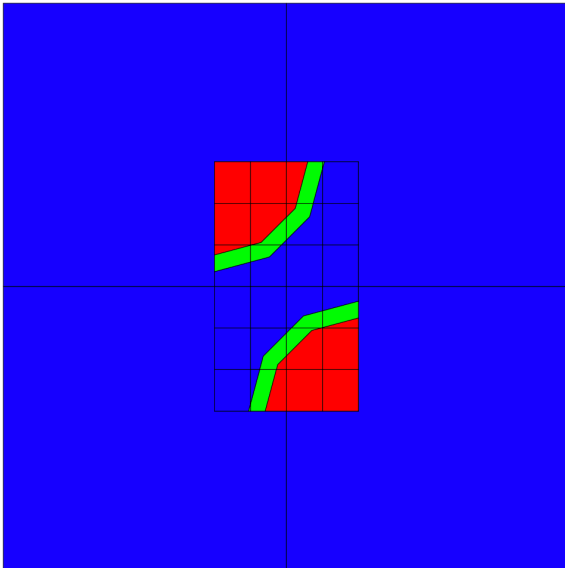


```
cylinder 1 0.455 chord -y=1.056551 chord +x=0 origin x=0 y=1.056551
cylinder 2 0.386 chord -y=1.056551 chord +x=0 origin x=0 y=1.056551
cylinder 3 0.455 chord -x=0.61 chord +y=0 origin x=0.61 y=0
cylinder 4 0.386 chord -x=0.61 chord +y=0 origin x=0.61 y=0
cuboid 5 0.61 0 1.056551 0
```

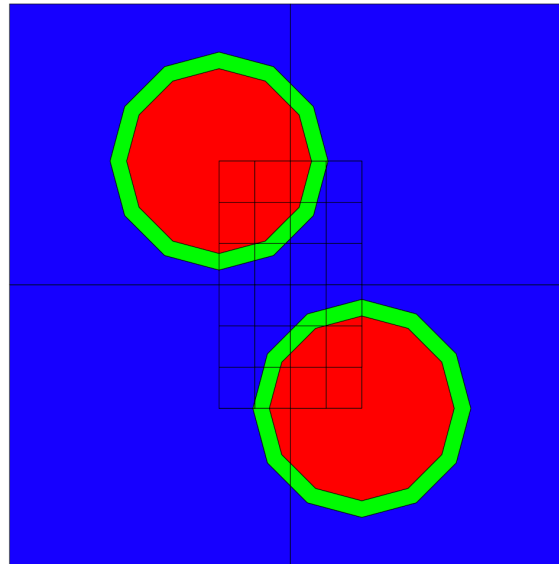
Why do I need chords in NEWT?

NEWT does not actually truncate the geometry with the boundary record. When placing the unit into another unit as a hole or within an array, the chords make a difference!

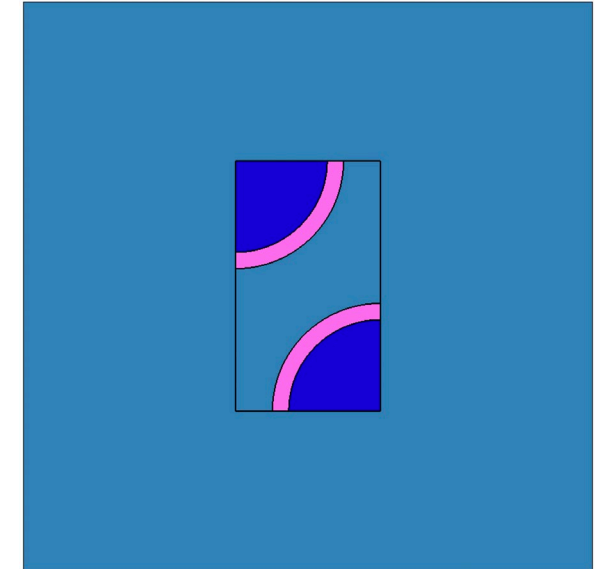
**NEWT PS file:
Input uses chords**



**NEWT PS file:
Input DOES NOT use chords**



**Careful: Fulcrum view
for both cases!**



This is NEWT-specific... Monte Carlo modeling does not require these chords.

NEWT input

- Part III: Holes and arrays



SCALE unit construction

- Thus far, geometry has been composed of a single unit, and that unit has been the global unit
- In a SCALE geometry, units are the basic building blocks of a model
- Models are typically composed of several units
 - Holes and arrays are used to place individual units within other units or in the global unit
- Only the geometry placed in the global unit is constructed in the geometry
- Model development strategy
 - Build each unit that you want in your model and then change each unit to the global unit to plot only that portion of the geometry
 - When all subordinate units are constructed, place those units into the global unit using holes or arrays

Holes

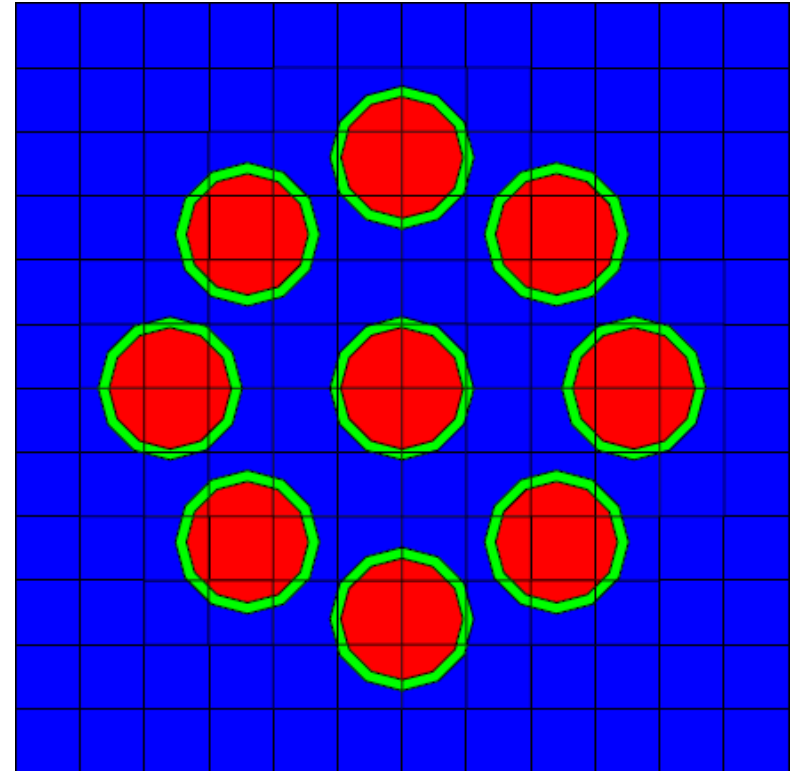
- Holes are used to place a single unit within another unit
- May only place one unit
- Multiple holes may be placed within a unit
- Useful when cells are repeated but not in a regular pattern
- Format:

hole unit id [origin x= y=] [rotate a1=]

- Origin keyword translates (0,0) of unit to new location
- Rotate keyword rotates the hole about the central point
- Holes may not intersect with other holes

Holes example

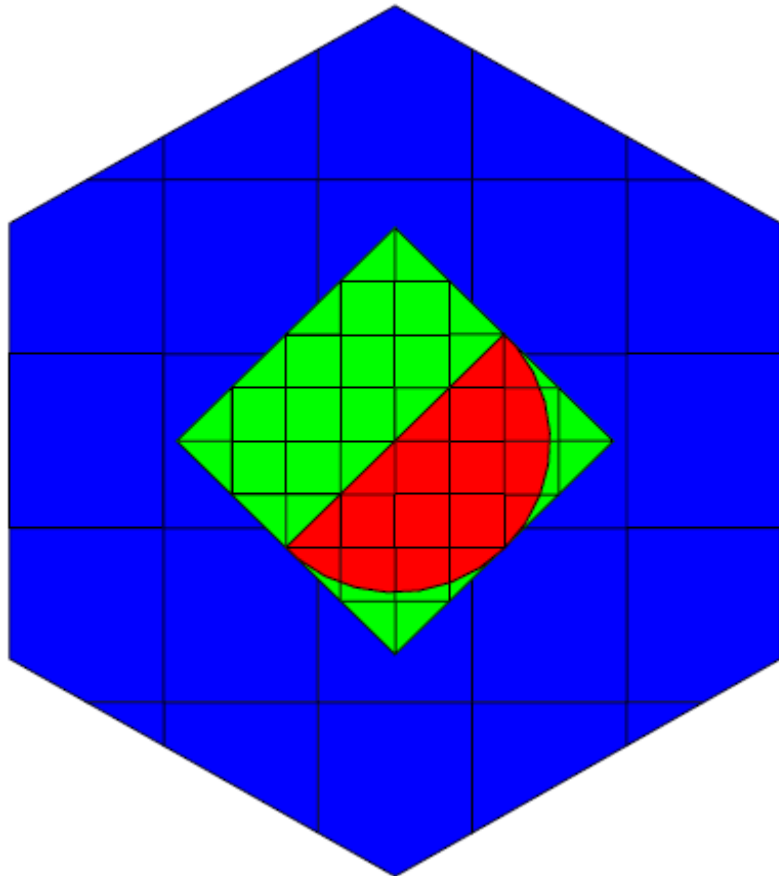
```
unit 1
  cylinder 30 0.3860
  cylinder 40 0.4550
  media 1 1 30
  media 2 1 40 -30
  boundary 40
global unit 2
  hole 1
  hole 1 origin x= 1.0 y= 1.0
  hole 1 origin x=-1.0 y= 1.0
  hole 1 origin x=-1.0 y=-1.0
  hole 1 origin x= 1.0 y=-1.0
  hole 1 origin x= 0 y= 1.5
  hole 1 origin x= 0 y=-1.5
  hole 1 origin x= 1.5 y= 0
  hole 1 origin x=-1.5 y= 0
  cuboid 10 4p2.5
  media 3 1 10
  boundary 10 12 12
```



Local mesh refinement

- Holes (and arrays) allow for local mesh refinement
 - **Hole example:**
 - The unit ***being inserted*** into the hole placement may have a 4 x 4 mesh
 - The unit ***receiving*** the hole may have a 13 x 13 mesh
 - This allows for very flexible mesh capabilities
- User guidance:
 - Global unit **MUST** have a base grid defined
 - Other units **MAY** have a local grid defined, but if omitted, the global grid will be used
 - Local grids are not rotated (see next slide)

Mesh for units with a rotated hole



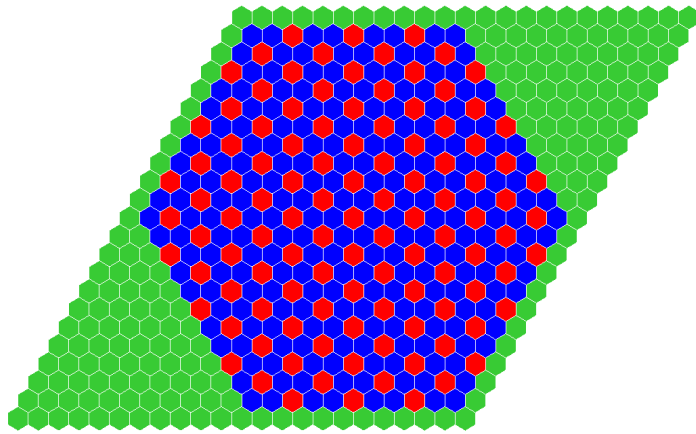
```
read geom
unit 1
  cylinder 10 10.0 chord +x=0 sides=30
  cuboid 20 4p10
  media 1 1 10
  media 2 1 20 -10
  boundary 20 8 8
global unit 2
  hexprism 30 25
  media 3 1 30
  hole 1 rotate a1=-45
  boundary 30 5 5
end geom
```

ARRAY block

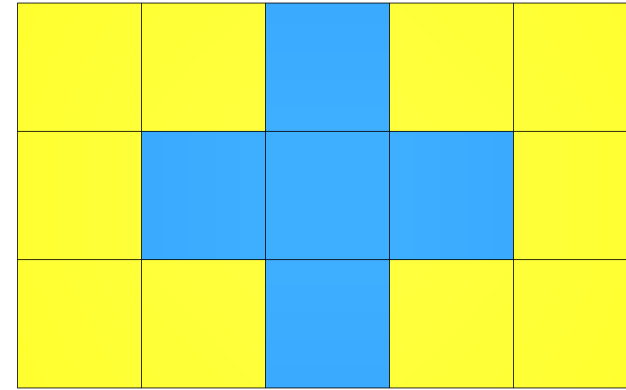
- Array specification consists of two parts:
 - **array placement** as a “body” in a unit in the geometry block
array arrayid bodyid place ix jy x_{ix} y_{jy}
 - **ARRAY block**, following the geometry block, used to specify the contents (unit numbers) that make up the array

```
read array
  ara=ID nux=I nuy=J typ=GEOM
  fill
    UNIT NUMBERS
  end fill
end array
```

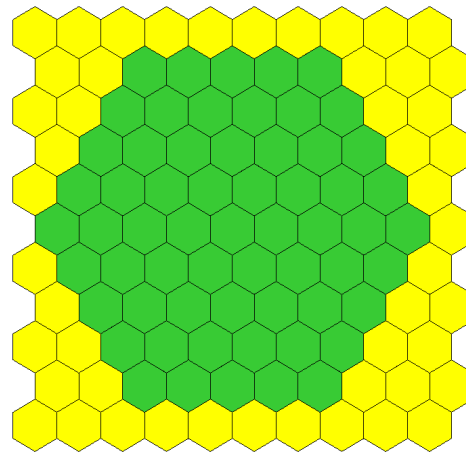
NEWT array types



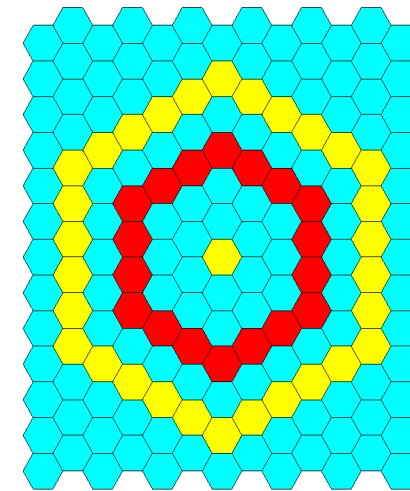
Hexagonal (or Triangular)



Rectangular



Stacked hex



Rotated hex

ARRAY block

- Arrays are used to place multiple units within another unit in a regular pattern
- Arrays are placed within a shape
- Multiple arrays may be placed within a unit
- Unit containing arrays may be placed in other arrays and in holes
- Unlike holes, arrays cannot be rotated unless placed inside a hole
- **The shape into which an array is placed must also be filled with a media**
 - It seems doubly defined, but NEWT knows that the array takes priority over the media
 - This allows arrays to be smaller than the shape into which the array is placed
 - This is one of the key differences between a NEWT model and a KENO/Shift model (when converting inputs, this must doubly defined space must be addressed)

ARRAY parameters

Keyword	Default	Type of Data
ARA=	1	Number defining the array
TYP=	Square	Square (cuboidal) Hexagonal (triangular) Shexagonal Rhexagonal Dodecahedral
COM=	none	delimiter comment delimiter 132 character maximum
NUX=	1	No. of units in the X direction
NUY=	1	No. of units in the Y direction
PINPOW	No	Print pin power (yes/no)

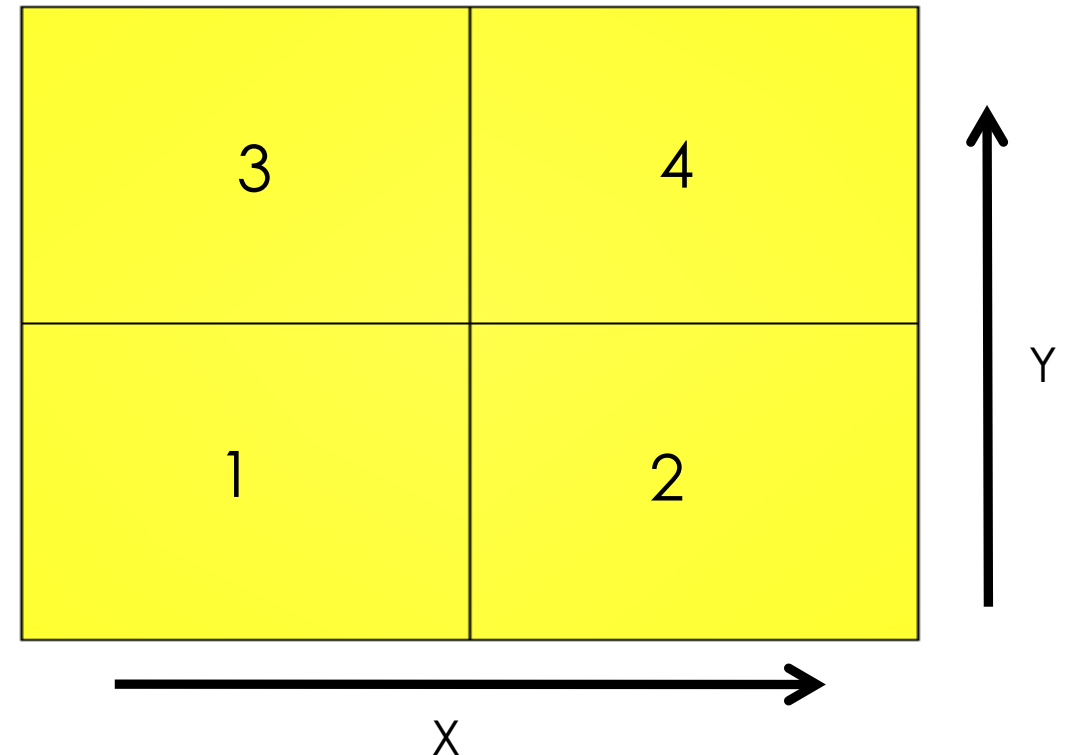
Fill parameter

- Fills the array with the unit numbers in the specified order
- Begins with keyword “fill”
- Ends with keywords “end fill”
- Fills the array by incrementing
 - X from 1 to NUX
 - then Y from 1 to NUY
- Fills the array from lower left to upper right

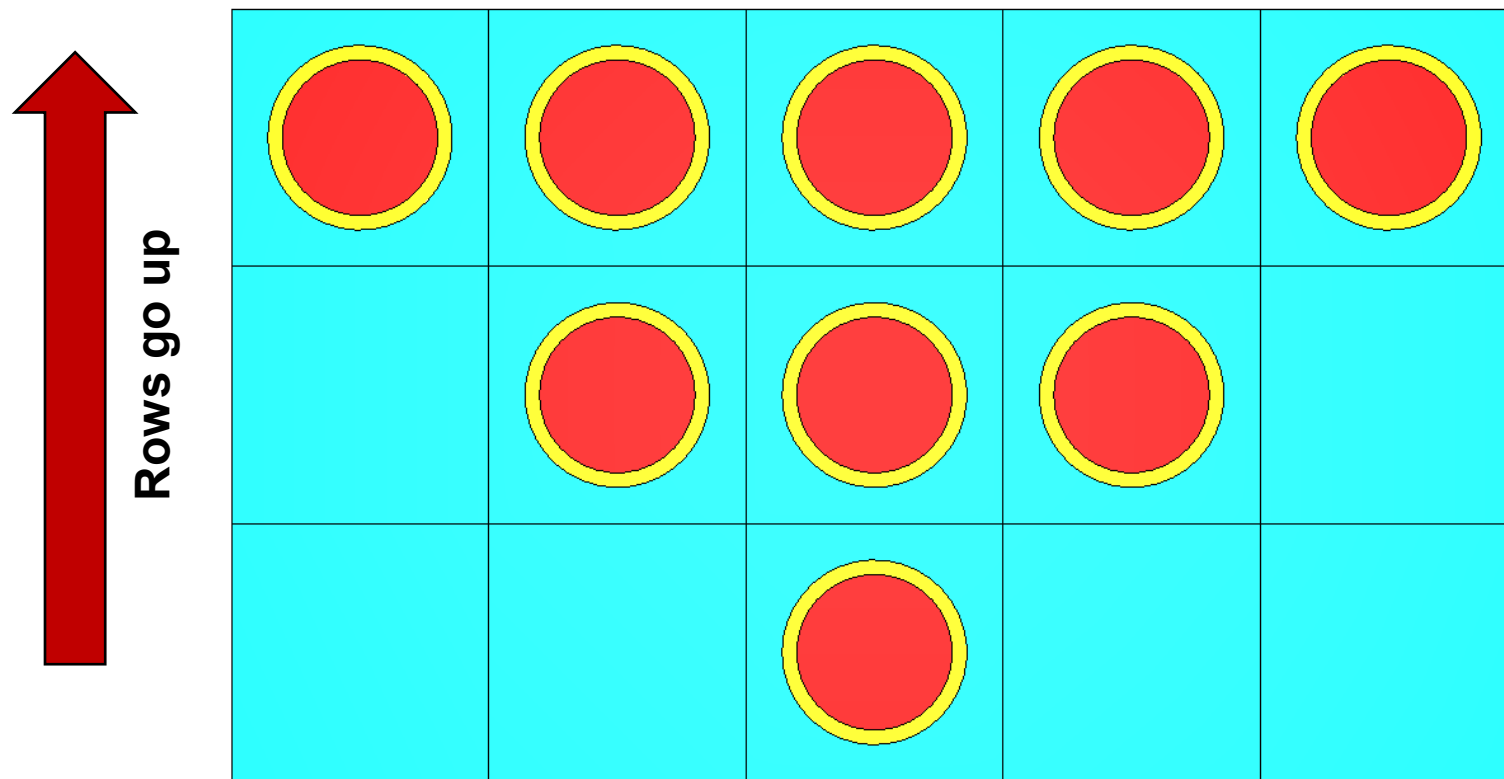
Fill data

- For a 2 x 2 array, the order of the units stored in the array using the FILL parameter is:

(1,1) (2,1) (1,2) (2,2)



Rectangular array



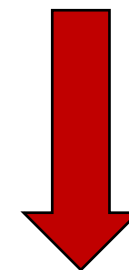
read array
ara=1 nux=5 nuy=3
typ=square

fill

2	2	1	2	2
2	1	1	1	2
1	1	1	1	1

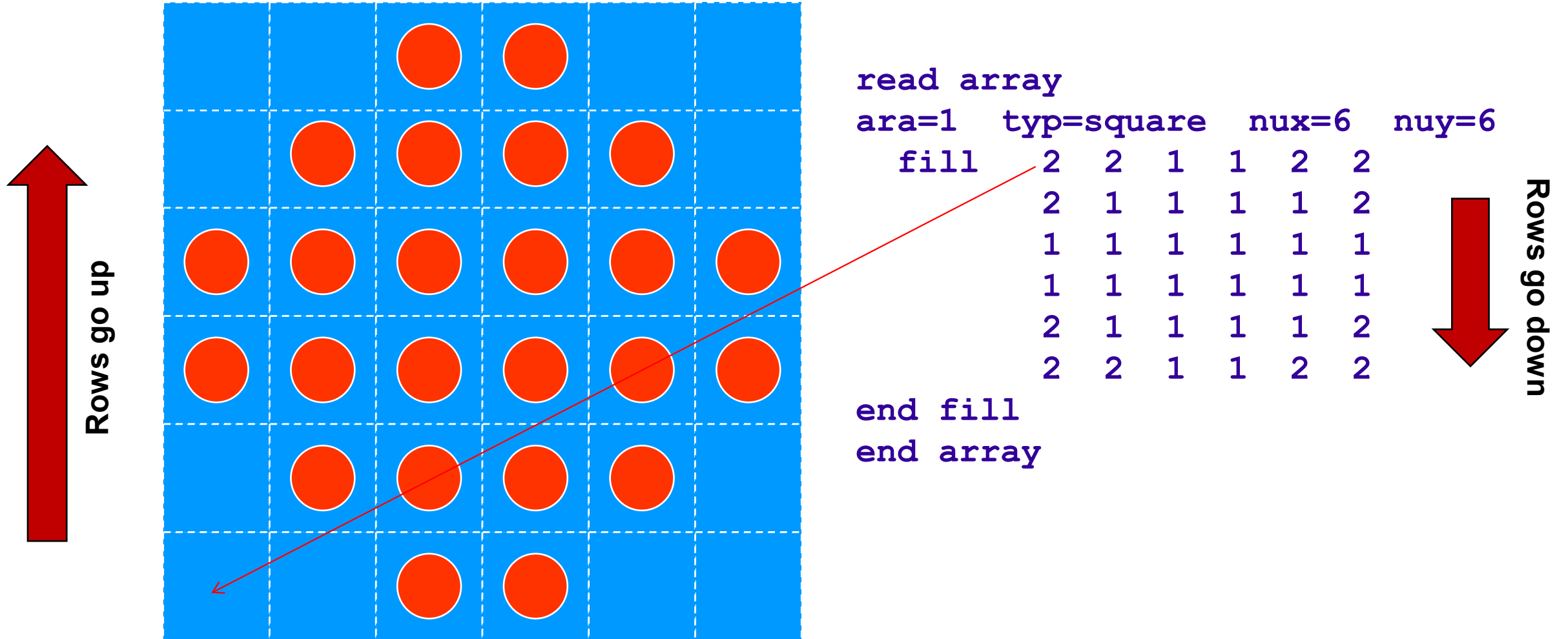
end fill

end array

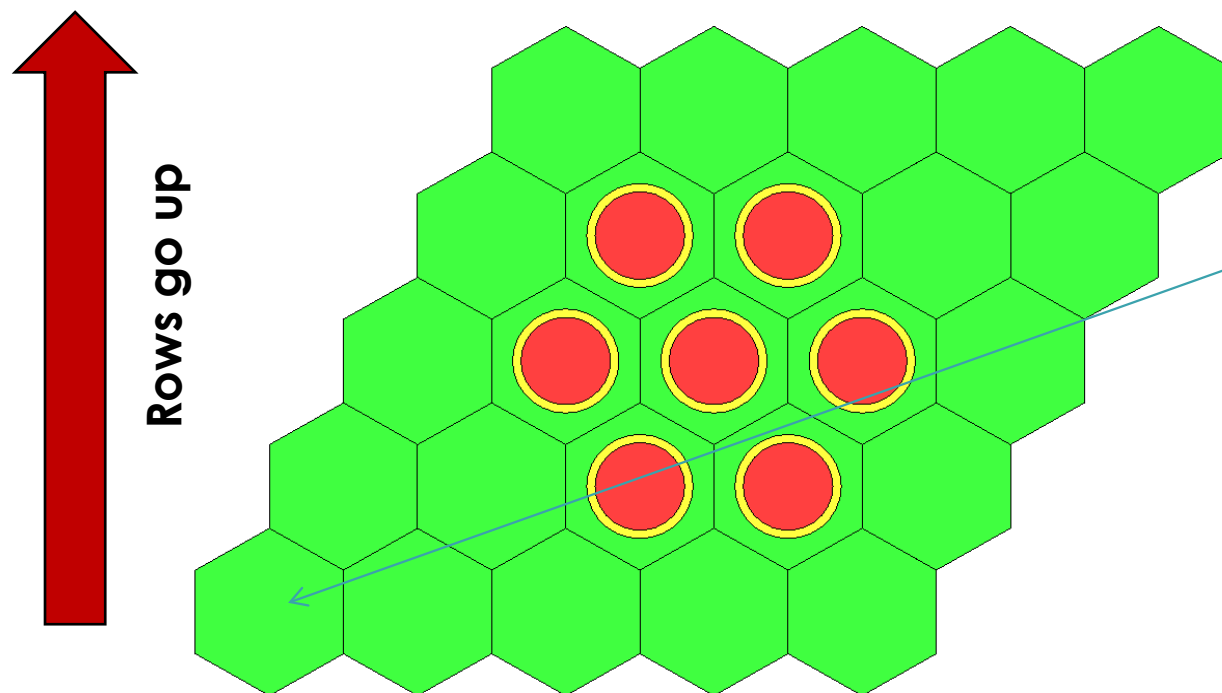


Rows go down

Square array

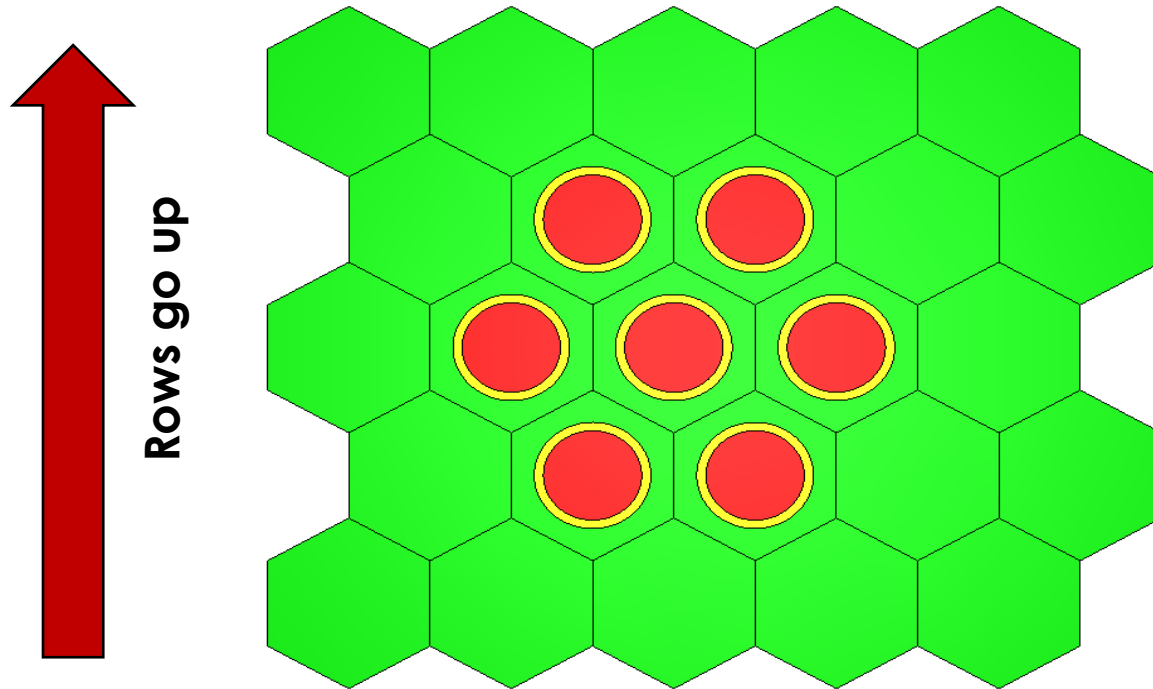


Hexagonal array



```
read array
ara=1 nux=5 nuy=5
typ=hexagonal
fill
  2 2 2 2 2
    2 2 1 1 2
      2 1 1 1 2
        2 1 1 2 2
          2 2 2 2 2
end fill
end array
```

Stacked (rotated) hexagonal array



```
ara=2 nux=5 nuy=5 typ=shexagonal
fill
  2 2 2 2 2
  2 1 1 2 2
  2 1 1 1 2
  2 1 1 2 2
  2 2 2 2 2
end fill
end array
```

Array placement

- Format:

array arrayid bodyid place ix jy x_{ix} y_{jy}

- Arrays are defined in the *ARRAY* block
- Data following the keyword **place** are used to position the array in a region:
 - the (I, J) position of the unit in the array used to establish the location of the array
 - the (X, Y) coordinates of the surrounding unit, containing the origin of the unit used to position the array

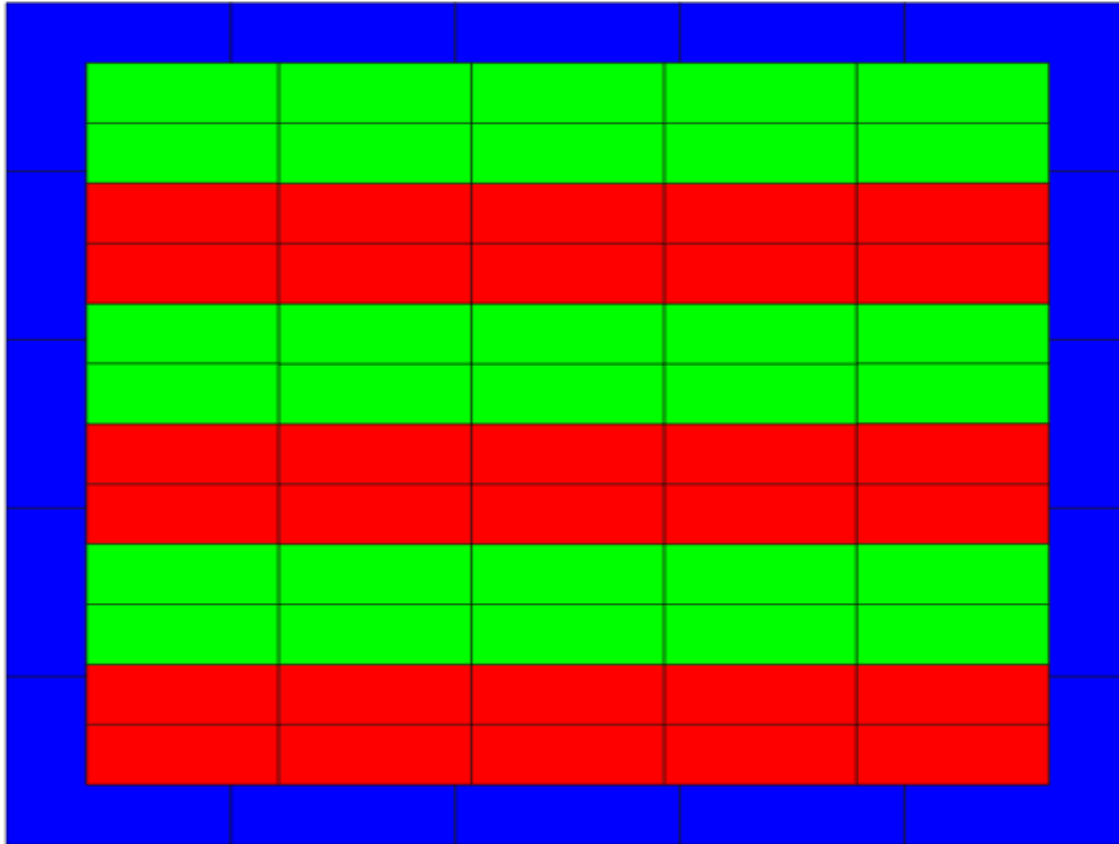
In other words:

Place the origin of the unit that is the ij^{th} element in the array at position (X, Y) in the current unit coordinate system

Notes on arrays

- Each element of an array (or a hole) may have its own local grid structure, or it may use the base structure
- Different grid refinements may be used within different elements in an array

Array example



```
read geometry
```

```
unit 1
```

```
  cuboid 10 2p6.0 2p1.0
```

```
  media 1 1 10
```

```
  boundary 10 5 2
```

```
unit 2
```

```
  cuboid 10 2p6.0 2p1.0
```

```
  media 2 1 10
```

```
  boundary 10 5 2
```

```
global unit 3
```

```
  cuboid 10 4p6.0
```

```
  cuboid 20 4p7.0
```

```
  array 1 10 place 1 1 0.0 -5.0
```

```
  media 3 1 20 -10
```

```
  boundary 20 5 5
```

```
end geometry
```

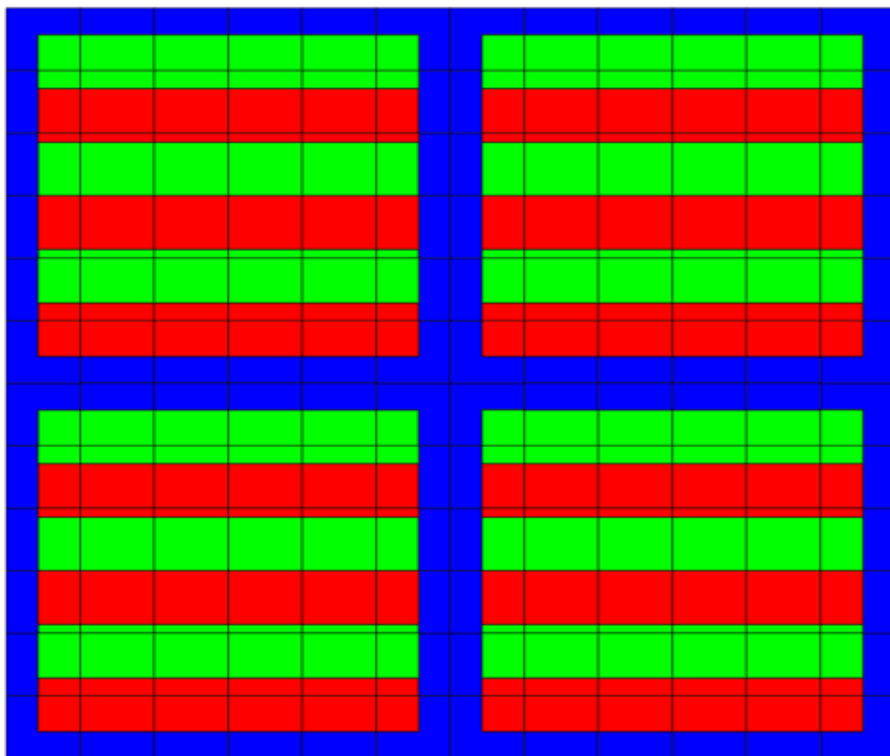
```
read array
```

```
  ara=1 nux=1 nuy=6
```

```
  fill 1 2 1 2 1 2 end fill
```

```
end array
```

Arrays within an array



```
read geometry
```

```
unit 1
```

```
  cuboid 10 2p6.0 2p1.0
```

```
  media 1 1 10
```

```
  boundary 10
```

```
unit 2
```

```
  cuboid 10 2p6.0 2p1.0
```

```
  media 2 1 10
```

```
  boundary 10
```

```
unit 3
```

```
  cuboid 10 4p6.0
```

```
  cuboid 20 4p7.0
```

```
  array 1 10 place 1 1 0.0 -5.0
```

```
  media 3 1 20 -10
```

```
  boundary 20
```

```
global unit 4
```

```
  cuboid 10 4p14.0
```

```
  array 2 10 place 1 1 -7.0 -7.0
```

```
  media 3 1 10
```

```
  boundary 10 12 12
```

```
end geometry
```

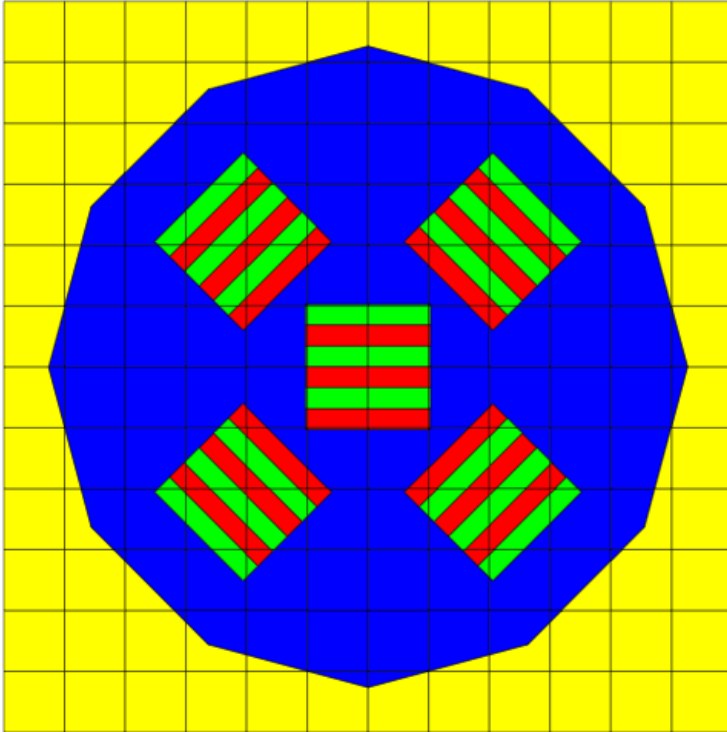
```
read array
```

```
  ara=1 nux=1 nuy=6 fill 1 2 1 2 1 2 end fill
```

```
  ara=2 nux=2 nuy=2 fill 3 3 3 3 end fill
```

```
end array
```

Rotating arrays with holes



```

read geometry
unit 1
  cuboid 10 2p6.0 2p1.0
  media 1 1 10
  boundary 10
unit 2
  cuboid 10 2p6.0 2p1.0
  media 2 1 10
  boundary 10
unit 3
  cuboid 10 4p6.0
  array 1 10 place 1 1 0.0 -5.0
  media 3 1 10
  boundary 10
global unit 4
  cylinder 10 30.0
  cuboid 20 4p35.0
  hole 3
  hole 3 origin x=12.0 y=12.0 rotate a1=-45
  hole 3 origin x=12.0 y=-12.0 rotate a1=-135
  hole 3 origin x=-12.0 y=-12.0 rotate a1=135
  hole 3 origin x=-12.0 y=12.0 rotate a1=45
  media 3 1 10
  media 4 1 20 -10
  boundary 20 12 12
end geometry
read array
  ara=1 nux=1 nuy=6
  fill 1 2 1 2 1 2 end fill
end array

```

Arrays and grids

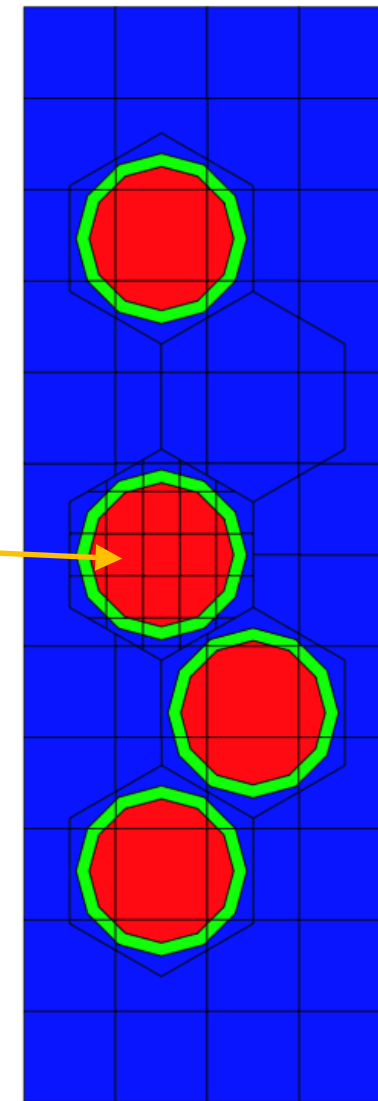
```
unit 1
  cylinder 30 0.3860
  cylinder 40 0.4550
  hexprism 50 0.5000
  media 1 1 30
  media 20 1 40 -30
  media 30 1 50 -40
  boundary 50 5 5
```

```
unit 2
  hexprism 50 0.5000
  media 30 1 50
  boundary 50
```

```
unit 3
  cylinder 30 0.3860
  cylinder 40 0.4550
  hexprism 50 0.5000
  media 1 1 30
  media 20 1 40 -30
  media 30 1 50 -40
  boundary 50
```

```
global unit 10
  cuboid 10 2.0 0.0 6.0 0.0
  array 1 10 place 1 3 0.75 3.0
  media 30 1 10
  boundary 10 4 12
end geometry

read array
  ara=1 nux=1 nuy=5
  typ=shexagonal
  fill 3 3 1 2 3 end fill
end array
```



FIDO-like shortcuts for FILL

Count Field	Option Field	Operand Field	Comment
		j	Stores <i>j</i> in current array position
	f	j	Fills the remainder of the array with <i>j</i> starting with the current position
i	r or *	j	Repeats entry <i>j</i> a total of <i>i</i> times
i	q	j	Repeats the previous <i>j</i> entries <i>i</i> times

Refer to Table 8.1.19 for a complete list of options

Examples:

<code>fill 5 5 5 5 5 end fill</code>	is identical with	<code>fill f5 end fill</code>
<code>fill 10 10 5 5 5 end fill</code>	is identical with	<code>fill 2r10 3r5 end fill</code>
<code>fill 1 2 3 1 2 3 1 2 3 end fill</code>	is identical with	<code>fill 1 2 3 2q3 end fill</code>

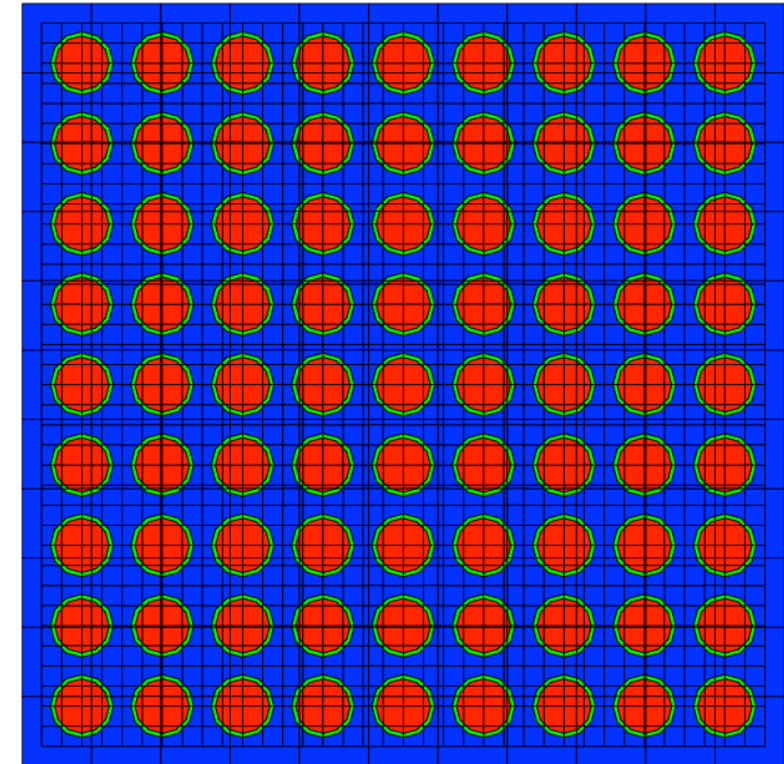
Workshop

- Part III: Holes and arrays



Problem 4a: 9x9 fuel array

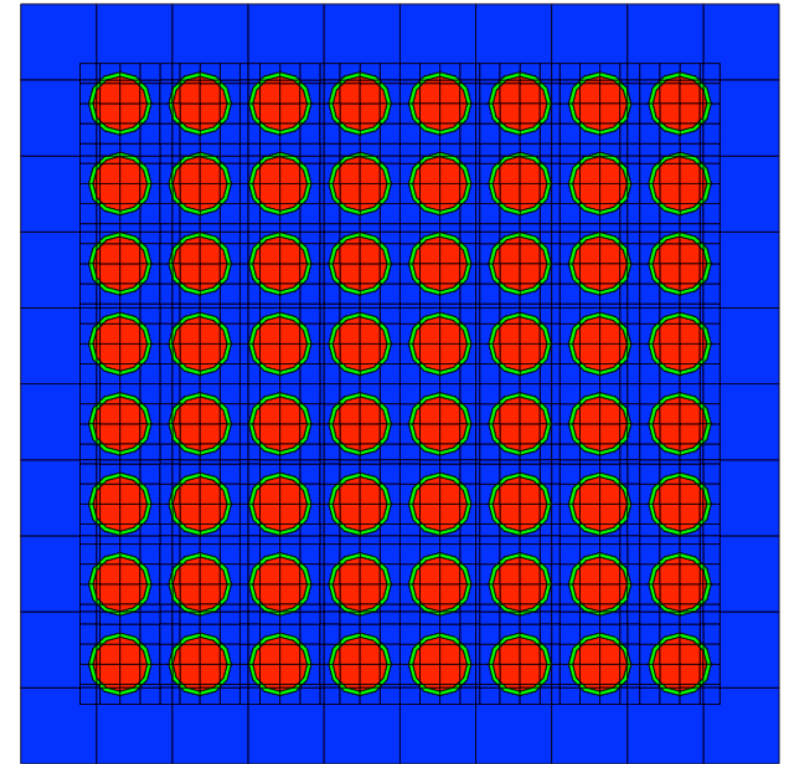
- Create a T-NEWT model for a 9x9 array of fuel pins
- Open "materials.inp" and save as "array_9x9.inp"
- General data:
 - t-newt, v7-56, parm=centrm
- NEWT materials:
 - Use P1 scattering for fuel and cladding
 - Use P2 scattering for the water
 - Give the materials a label through the com parameter
- NEWT geometry:
 - Pitch=1.875 cm, mixture 30, H₂O
 - Fuel diameter = 1.242 cm, mixture 1, UO₂
 - Cladding diameter = 1.430 cm, mixture 20, Zirc-2
 - Assembly pitch = 17.78 cm, mixture 30, H₂O
 - 4x4 grid for each unit cell
 - 10x10 global grid
- NEWT parameters:
 - cmfd = yes, converge = mix
 - Turn flux plots on
- Reflective boundary conditions
- Calculate k-eff and record NEWT runtime



9x9 pin array

Problem 4b: 8x8 fuel array

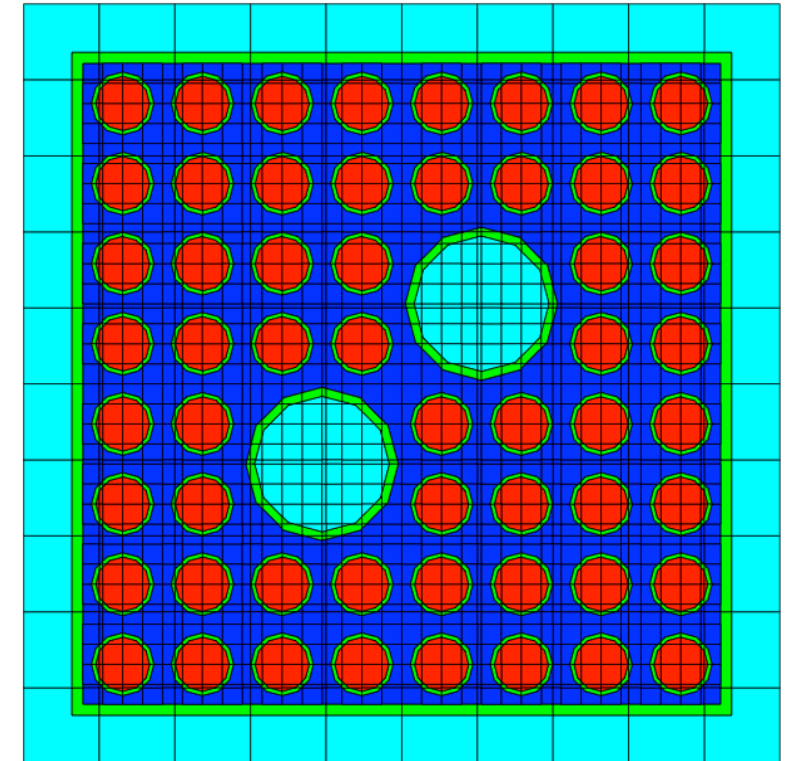
- Change the 9x9 fuel pin array to an 8x8 fuel pin array centered at (0,0)
- Open "array_9x9.inp" and save as "array_8x8.inp"
- Change fuel array to an 8x8 array
- Center of array should be placed at the center of the model
- Calculate k-eff and compare it to 9x9 array
- Compare NEWT runtime to 9x9 array



8x8 pin array

Problem 4c: 8x8 fuel array with water rods

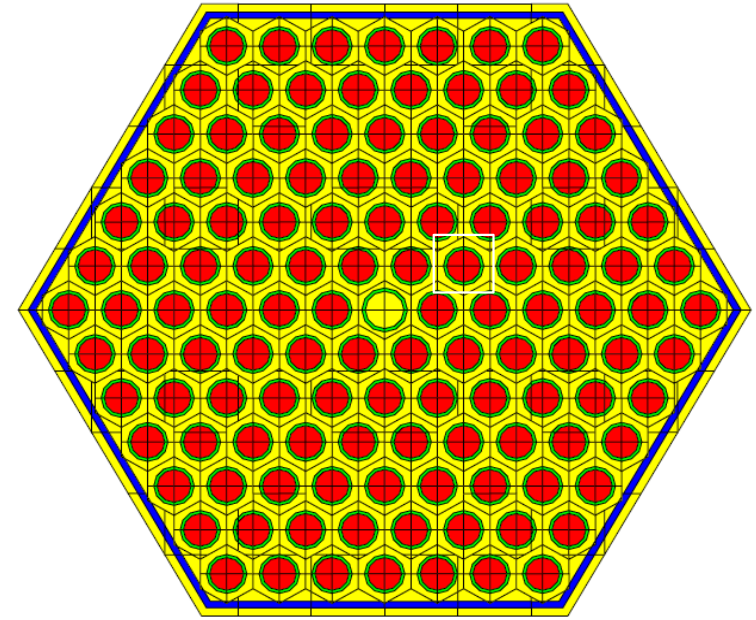
- Add water rods and a channel box to the 8x8 fuel pin array; use holes to place water rods
- Open "array_8x8.inp" and save as "array_8x8_waterrods.inp"
- Materials
 - Remove boron from water
 - Water inside channel should be 0.18 g/cc (~80% void)
 - Water in water rods and outside channel should be 0.75 g/cc
- Water holes:
 - Outer diameter = 3.5 cm
 - Thickness = 0.2 cm
 - Material = Zirc-2 filled with water
 - Centered at (-1.875, -1.875) and (1.875, 1.875) cm
- Channel Box
 - Inner span = 15 cm
 - Thickness = 0.254 cm
- Calculate k-eff and compare it to 8x8 array without water rods
- Compare flux to 8x8 array without water rods



**8x8 pin array
with water rods**

Problem 5 (Bonus): VVER assembly

- Build a VVER assembly using a hexagonal array
- Start with “vver_pincell.inp” and save it as “vver_array.inp”
- Add a unit for the central guide tube and empty units for around the outside of the fuel array
 - Central guide tube same as fuel cladding, but filled with water
- Array should be 13 cells across
- Add a channel box:
 - 14.20 cm flat-to-flat, 0.15 cm thick
 - Same material as fuel clad material
- Assembly pitch: 15 cm assembly pitch
- Hint: Use zeros in the array for empty units



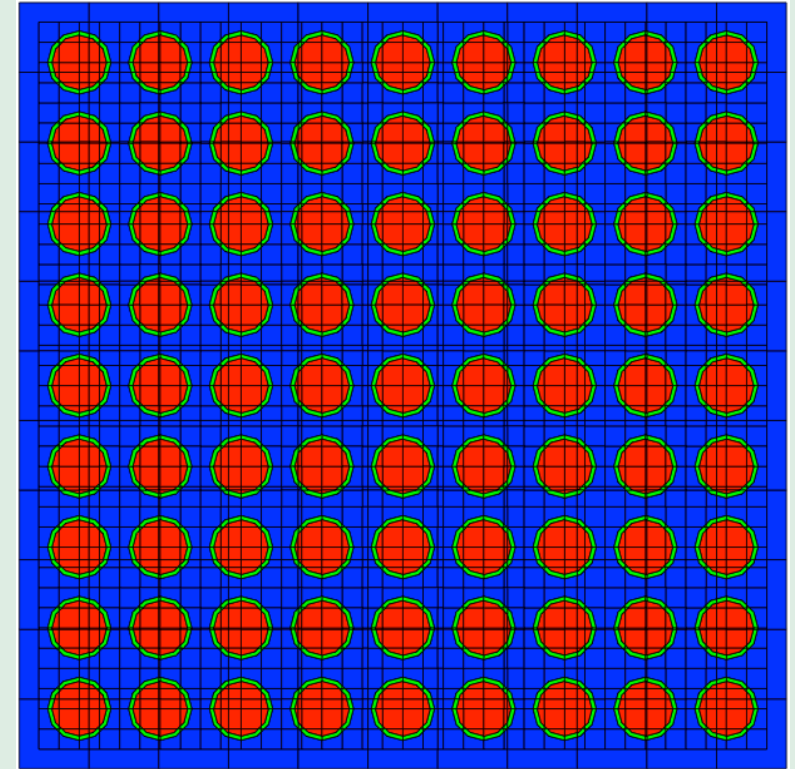
VVER assembly

Note:

- “zero” array elements are accepted in NEWT as empty units
- Fulcrum cannot plot geometries with “zero” array elements
- Check NEWT geometry to make sure it is correct

Solution 4a: 9x9 fuel array

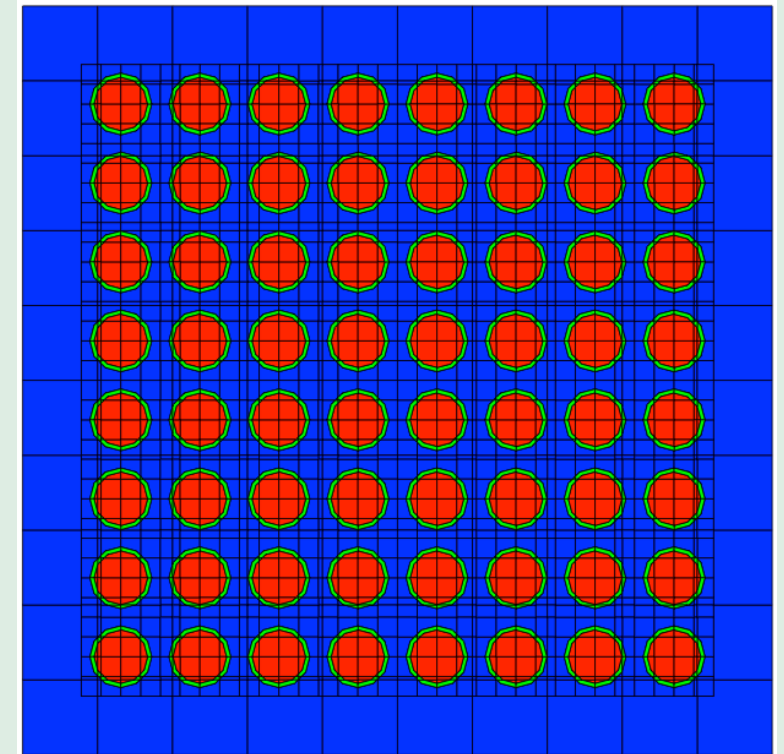
- T-NEWT model for a 9x9 array of fuel pins
- $k\text{-eff} = 1.34662$
- Runtime 58 seconds
- Array placement:
 - `array 1 10 place 5 5 0 0`



9x9 pin array

Problem 4b: 8x8 fuel array

- Change the 9x9 fuel pin array to an 8x8 fuel pin array centered at (0,0)
- $k\text{-eff} = 1.27811$
 - ~6,850 pcm lower than 9x9 array
- Runtime 55 seconds, similar to 9x9 array
- Center of array is placed at the center of the model by shifting the 5th pin of the 5th row by half a pitch
 - `array 1 10 place 5 5 0.9375 0.9375`



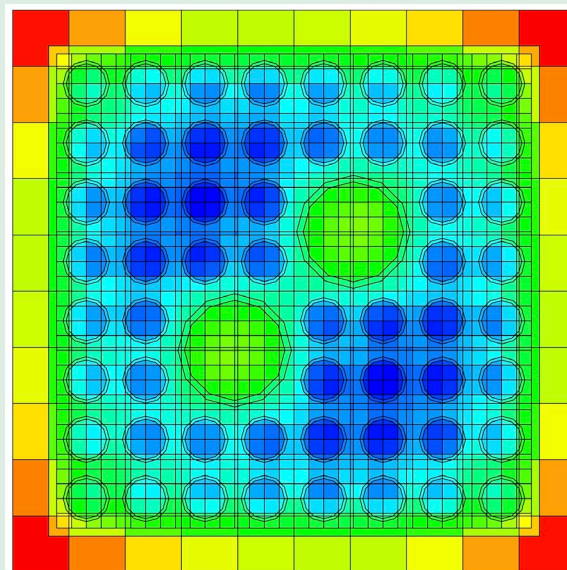
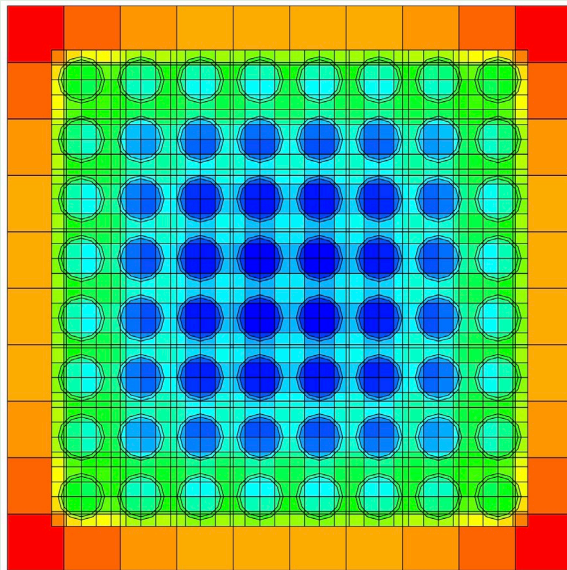
8x8 pin array

Solution 4c: 8x8 fuel array with water rods

- K-eff = 1.44787
 - ~17,000 pcm higher than without water rods
- Runtime 60 seconds, similar to 8x8 array

```
global unit 100
cuboid 10 7.5 -7.5 7.5 -7.5
cuboid 12 7.754 -7.754 7.754 -7.754
cuboid 14 8.89 -8.89 8.89 -8.89
media 30 1 10
media 20 1 -10 12
media 31 1 -12 14
array 1 10 place 5 5 0.9375 0.9375
hole 20 origin x=-1.875 y=-1.875
hole 20 origin x= 1.875 y= 1.875
boundary 14 10 10
```

Flux comparison of group 41, ~0.625 eV

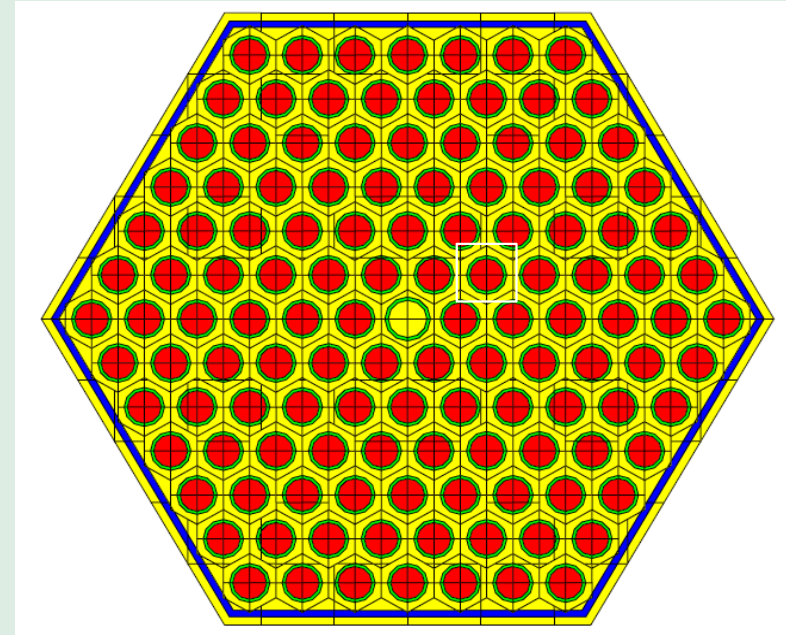


Problem 5 (Bonus): VVER assembly

- Build a VVER assembly using a hexagonal array
- $k\text{-eff} = 1.26909$

```
unit 1
...
hexprism 3      0.61
boundary 3
unit 2
hexprism 3      0.61
...
boundary 3
global unit 10
rhexprism 10 7.1
rhexprism 20 7.25
rhexprism 30 7.5
array 1 10 place 7 7 0.0 0.0
media 30 1 10
media 20 1 20 -10
media 30 1 30 -20
boundary 30 50 50
```

```
read array
ara=1 typ=shexagonal nux=13 nuy=13
fill
0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 2 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 0 0 0
end fill
end array
```



VVER assembly

- Note:
- “zero” array elements are accepted in NEWT as empty units
 - Fulcrum cannot plot geometries with “zero” array elements
 - Check NEWT geometry to make sure it is correct

NEWT Acceleration Techniques

- Part IV: CMFD



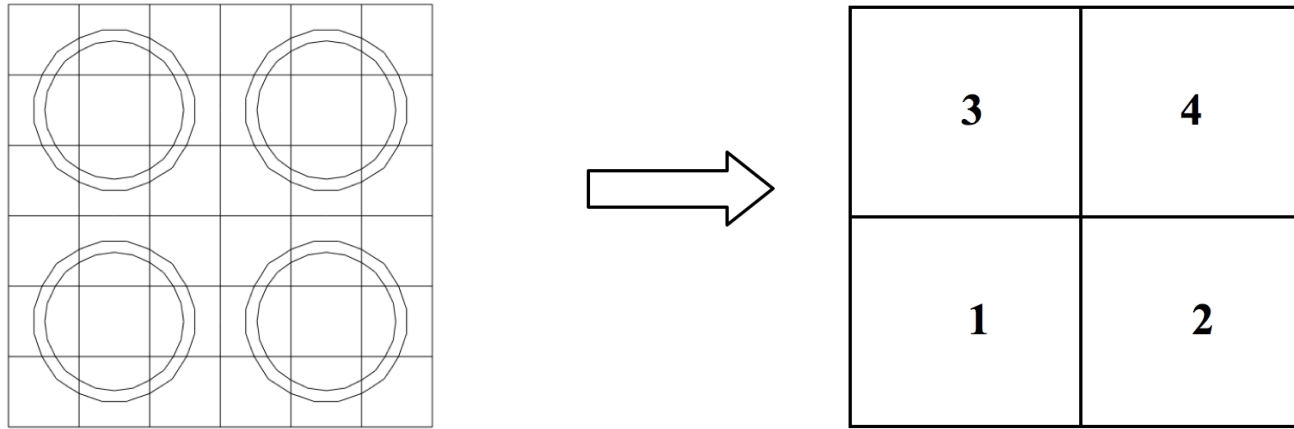
Coarse-mesh finite-difference (CMFD) acceleration

- The source iteration approach in NEWT can be slow to converge when a problem contains a large degree of scattering
 - reflector
 - graphite/heavy water moderation
- This requires a large number of iterations and time
- Full assembly calculations can take considerable time simply due to size

CMFD acceleration

- CMFD accelerator has been incorporated into NEWT to accelerate the spatial solution
- In CMFD, cells are homogenized and the reduced spatial mesh is solved using a lower-order solution
- The results are then passed back to the fine-mesh solution as improved flux guess
- This approach can accelerate convergence by a factor of 2 or 3 (or more)

CMFD homogenization



- Transport solution flux (or flux guess) is used to generate 2-group and multigroup parameters
 - macroscopic cross sections
 - diffusion coefficients
- Currents on cell edges are used to determine coupling coefficients between cells
- Boundary condition treatments are based on transport boundary conditions

CMFD solution

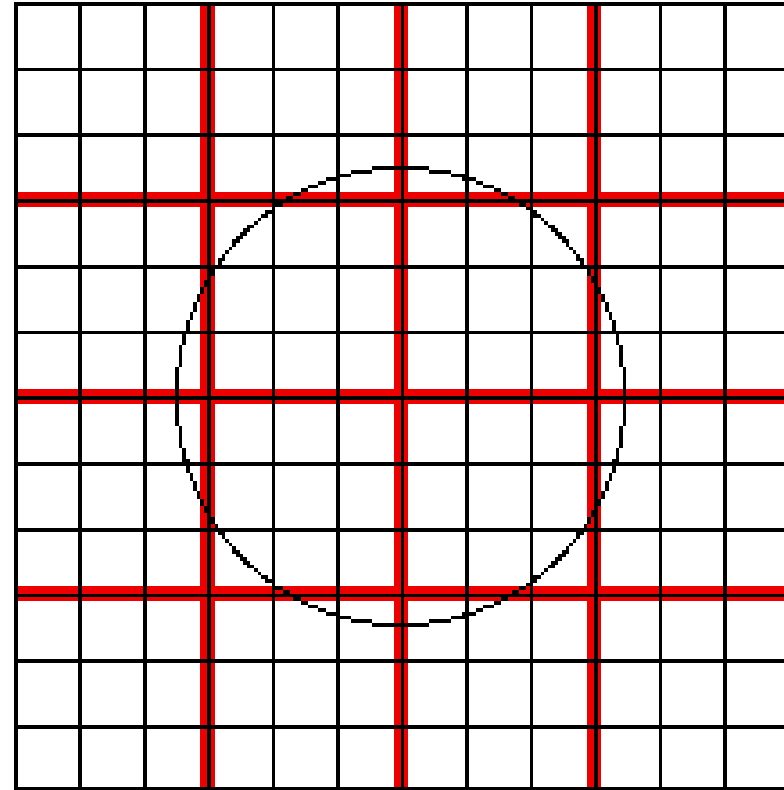
- CMFD runs at the beginning of each outer iteration
- NEWT starts with a 2-group CMFD solution
- Converged 2-group solution is used as a first guess for sources for a CMFD multigroup solution (using NEWT's energy structure)
- Once converged, the MG CMFD flux solution is folded onto the transport grid (using local/cell average ratio in each cell)
- The transport solution now proceeds with an improved guess at the fission/scattering source

CMFD implementation

- CMFD is disabled by default; to turn it on, enter "cmfd=yes" in *PARAMETER* block
- You must also specify the number of fine-mesh (rectangular) cells (in the global grid) that are to be included in each coarse mesh cell, e.g.,
 - xcmfd=2 ycmfd=2 ; or xycmfd=2

CMFD example

- 12x12 global grid with $xycmfd=3$ coarse-mesh grid
- Simple 16 cell problem
- Properties within each red (coarse) cell are homogenized
- The transport solution still solves the fine, non-orthogonal grid



CMFD and underlying grid

- Normally, when array elements are placed in a problem, the underlying global grid is removed
- Because the CMFD grid is based on the global grid, the global grid is retained when *cmfd=yes*
- An alternative approach for implementing CMFD is to use the cells in an array to define the grid
 - This is invoked with *xycmfd=0* specification
 - In this case, the array must completely fill the global boundary

CMFD example (cmfd=no)

Outer Iter. #	Eigenvalue	Eigenvalue Delta	Max Flux Delta	Max Flux Location(r,g)	Max Fuel Delta	Max Fuel Location(r,g)	Wall Clock	Elapsed CPU Time	Iteration CPU Time	CPU Usage	Inners Converged
1	1.00000	0.000E+00	6.987E+04	(65,252)	1.000E+00	(20, 0)	20:11:05	0.5 s	0.5 s	52.0%	F
2	1.09236	8.455E-02	2.193E+00	(112, 1)	1.594E-01	(147, 0)	20:11:05	1.1 s	0.6 s	56.0%	F
3	1.46067	2.522E-01	1.977E+00	(112, 1)	1.569E-01	(65, 0)	20:11:06	1.8 s	0.7 s	74.0%	F
4	1.41302	3.372E-02	1.922E+00	(59, 1)	1.781E-02	(121, 0)	20:11:07	2.5 s	0.7 s	68.0%	F
5	1.39733	1.123E-02	1.896E+00	(147, 1)	5.769E-03	(151, 0)	20:11:07	3.1 s	0.6 s	56.0%	F
6	1.39547	1.335E-03	1.867E+00	(53, 1)	7.987E-04	(139, 0)	20:11:08	3.6 s	0.5 s	55.0%	F
7	1.39483	4.577E-04	1.812E+00	(137, 1)	2.850E-04	(112, 0)	20:11:08	4.1 s	0.5 s	45.0%	F
8	1.39458	1.800E-04	1.678E+00	(67, 1)	1.283E-04	(120, 0)	20:11:09	4.4 s	0.4 s	39.0%	F
9	1.39448	7.527E-05	1.388E+00	(54, 1)	6.694E-05	(121, 0)	20:11:09	4.7 s	0.2 s	24.0%	F
10	1.39441	5.087E-05	9.267E-01	(54, 1)	5.372E-05	(147, 0)	20:11:09	4.9 s	0.2 s	18.0%	F

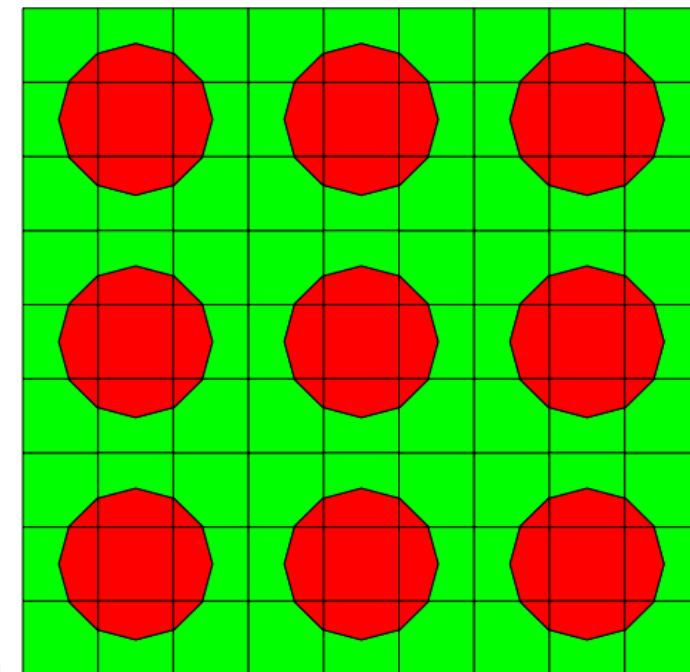
k-eff = 1.39440693

Time per Iteration (sec):

Transport Sweeping: 0.422
 Inscatter Source : 0.065
 Coarse-Mesh Accel : 0.000
 Else (Convergence): 0.024
 Total (per outer): 0.511

Total iteration time breakdown (sec):

Transport Sweeping: 4.223
 Inscatter Source : 0.646
 Coarse-Mesh Accel : 0.000
 Else (Convergence): 0.243
 Total Time : 5.112



CMFD example (cmfd=1, xycmfd=3)

Outer Iter. #	Eigenvalue	Eigenvalue Delta	Max Flux Delta	Max Flux Location(r,g)	Max Fuel Delta	Max Fuel Location(r,g)	Wall Clock	Elapsed CPU Time	Iteration CPU Time	CPU Usage	Inners Converged
1	1.39487	2.831E-01	1.750E+01	(65,252)	1.000E+00	(20, 0)	20:12:38	0.7 s	0.7 s	69.0%	F
2	1.39427	4.321E-04	3.674E-02	(65,252)	1.208E-01	(143, 0)	20:12:39	1.4 s	0.7 s	69.0%	F
3	1.39434	5.147E-05	5.201E-03	(65,252)	1.232E-03	(108, 0)	20:12:39	1.9 s	0.5 s	52.0%	F
4	1.39434	5.058E-07	9.561E-05	(72, 2)	2.541E-04	(143, 0)	20:12:39	2.2 s	0.3 s	27.0%	F
5	1.39434	3.933E-08	1.326E-05	(63, 1)	1.164E-05	(151, 0)	20:12:39	2.4 s	0.2 s	19.0%	T
6	1.39434	4.413E-08	4.787E-06	(137, 1)	2.384E-06	(81, 0)	20:12:40	2.5 s	0.2 s	17.0%	T
k-eff = 1.39433972											

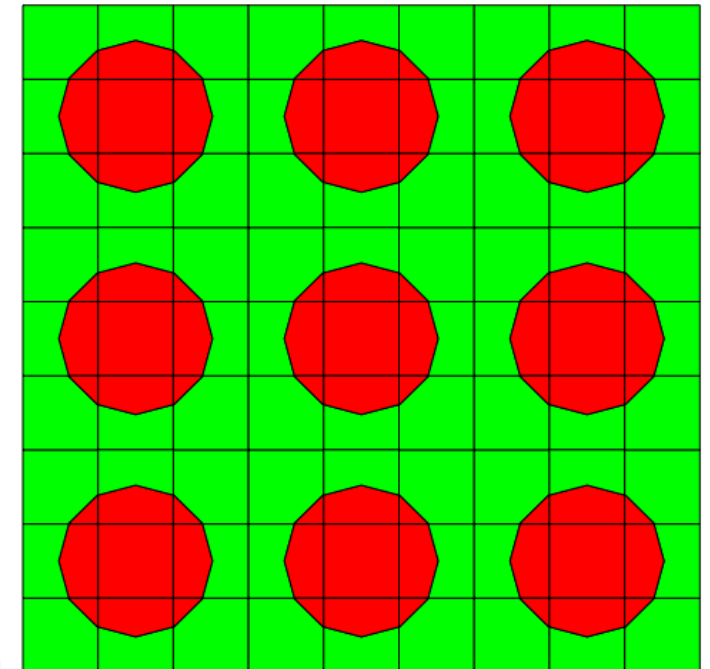
Time per Iteration (sec):

Transport Sweeping: 0.272
Inscatter Source : 0.041
Coarse-Mesh Accel : 0.179
Else (Convergence): 0.039
Total (per outer): 0.530

Total iteration time breakdown (sec):

Transport Sweeping: 1.631
Inscatter Source : 0.243
Coarse-Mesh Accel : 1.074
Else (Convergence): 0.234
Total Time : 3.182

~1.6x Speedup
with CMFD



Workshop

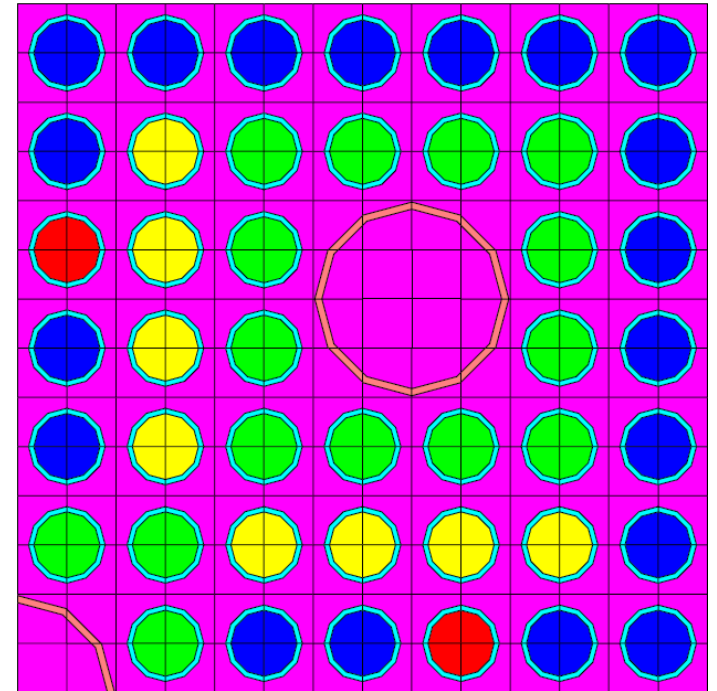
- Part IV: CMFD



Problem 6: CMFD

1. Copy "ce14_assm_tnewt.inp" to a new file "ce14_cmfd.inp"
2. Update the model:
 - parm=(bonami)
 - drawit=no
 - Update the grid: global unit 28x28, all other 4x4
3. Set up 5 cases (5 inputs) with different cmfd options:

case	cmfd option
1	no
2	yes, xycmfd=0
3	yes, xycmfd=1
4	yes, xycmfd=2
5	yes, xycmfd=4
4. Run all five cases and record:
 - k-eff
 - Number of outer iterations
 - Total CPU Time
5. What can you infer about the CMFD acceleration method from the results?

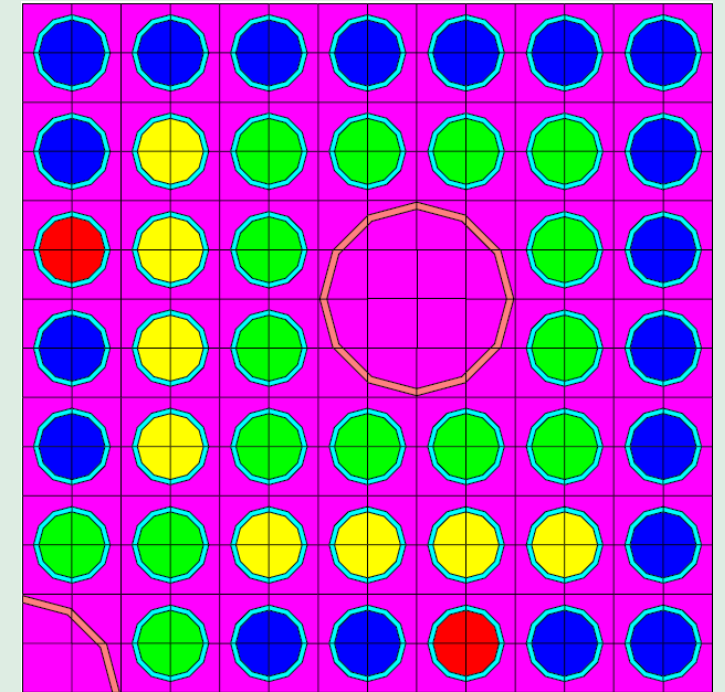


CE 14x14 assembly

Solution 6: CMFD

case	cmfd	k-eff	time [sec]	# outer iterations
1	no	1.29241	10.98	19
2	yes, xycmfd=0	1.29246	5.87	7
3	yes, xycmfd=1	1.29228	36.26	16
4	yes, xycmfd=2	1.29247	7.69	8
5	yes, xycmfd=4	1.29246	5.84	7

- k_{inf} results are consistent
- xycmfd=0 is fast because it sets the coarse mesh based on the size of the array elements
- xycmfd=4 is somewhat close to cmfd=0 and therefore similarly fast



CE 14x14 assembly

Converting a NEWT input to a KENO-VI input

Relevance:

- Verify NEWT results with KENO-VI result
- Generate MCDancoff input



Converting NEWT to KENO-VI

- **Redefine the sequence:**
 - **=t-depl** or **=t-newt** to **=mcdancoff** or **=csas6**
 - Change XS library for MCDancoff
 - **v7.1-252** => **xn01**
- **Add the Z-Axis:**
 - Add **+z** and **-z** to all bodies
 - Add **nuz=1** terms in arrays
 - Add z terms in array placements and holes
 - Add reflective axial boundary conditions
- **Remove all Aliases:**
 - Use Fulcrum "Expand Aliases" function
- **Remove **DEPLETION**, **BURNDATA**, and **BRANCH** blocks**
 - Change fuel pins to the correct material if using the assign function of the branch block
- **Remove **CELLDATA** block if running in continuous energy mode** (optional)
- **Reset the parameter block:**
 - Remove all NEWT parameters
 - Include KENO parameters
- **Remove spatial discretization:**
 - Boundary [surface] **nx ny**
- **Remove the NEWT **MATERIALS** block**
- **Arrays:**
 - Remove media in shapes occupied by array
 - Change 2d arrays to 3d arrays
- **Change block indicators:**
 - Remove "read model"
 - Replace "end model" with "end data"
- **Watch out for the global unit!**
 - NEWT is more forgiving than KENO on the global unit
 - For KENO, you need a surface exactly enclosing your global array

Questions?

NEWT training goals

- Understand capabilities of the NEWT neutron transport code
- Understand NEWT's output files
- Learn how to set up a NEWT input

