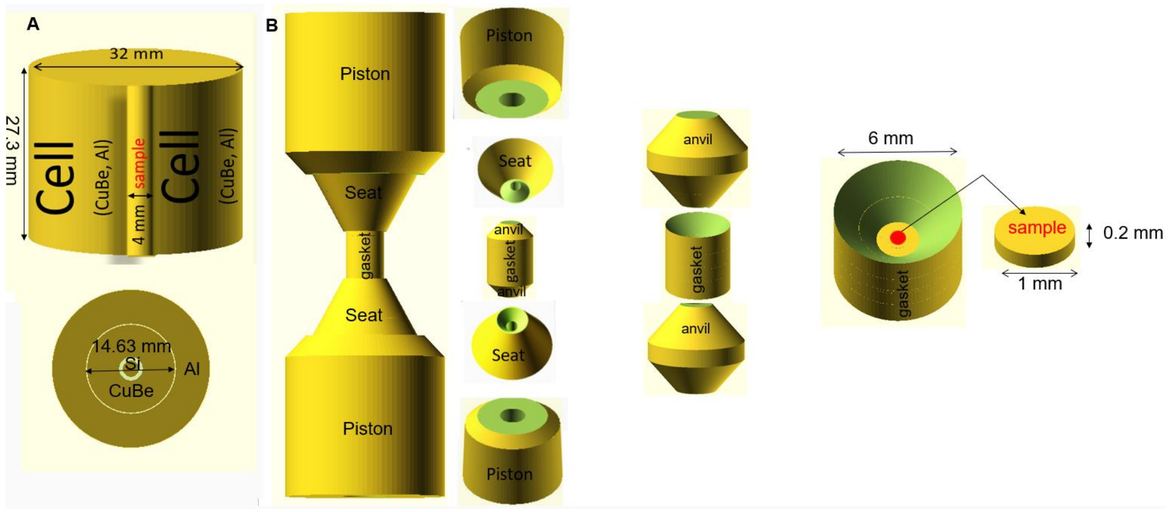
Direct Transfer of Monte Carlo Ray tracing Models to Additive Manufacturing Systems

Fahima Islam[1], Jiao Lin, Bianca Haberl[1],Ian Lumsden[1], David Anderson[2], Garrett Granroth[1],
[1]*Neutron Scattering Division, Oak Ridge National Laboratory*
[2]*Neutron Technologies Division , Oak Ridge National Laboratory*

Abstract

Advanced computer-aided design (CAD) modelling is a general prerequisite in 3D printing or Additive Manufacturing (AM) processes. Monte Carlo simulations, used to model the neutronic performance of an instrument, have 3-D representations of the instrument, but transferring this to CAD models is usually done by hand and is the most time-consuming and error prone part of the entire process chain. This contribution describes a tool to take components modeled in McVine and convert them straight to a format ready for AM. Over the last three decades, the STereoLithography (STL) file format, has become the de facto standard, used in many, if not all, AM systems to exchange information between design programs and AM systems. Our system uses scripts to generate a computer model, in which python statements describe the geometry of the required object by using Constructive Solid Geometry (CGS). CGS is a way of building complex objects from simple primitives using Boolean operations. The script encodes the set of CGS primitives representing the object onto a hierarchical tree. This script saves the tree structure onto an XML file used in McVine for the Monte Carlo Simulation. We developed a program the leverages OpenSCAD to create CAD models from the geometry defined in the xml file. OpenSCAD directly exports the geometry to an STL file, which is used for 3-D printer open-source slicing programs. This tool was used to create CAD files representing two pressure cells with different complex shapes and collimators to reduce background from these cells. The neutron scattering from the components is simulated by MCViNE. Furthermore it has been used to generate collimator designs that are printed via AM.

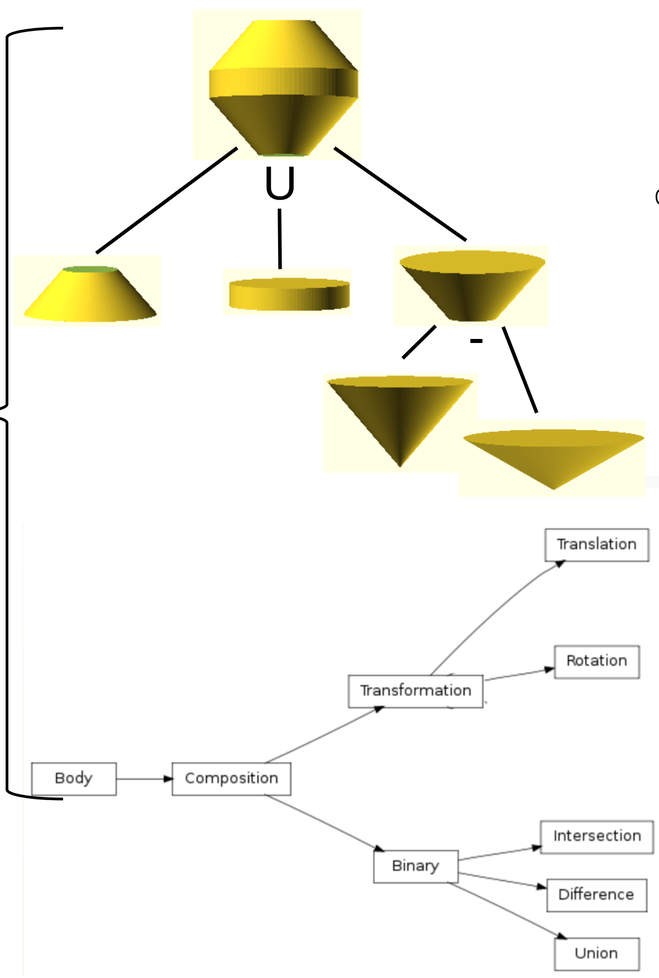
Examples of 3D Models

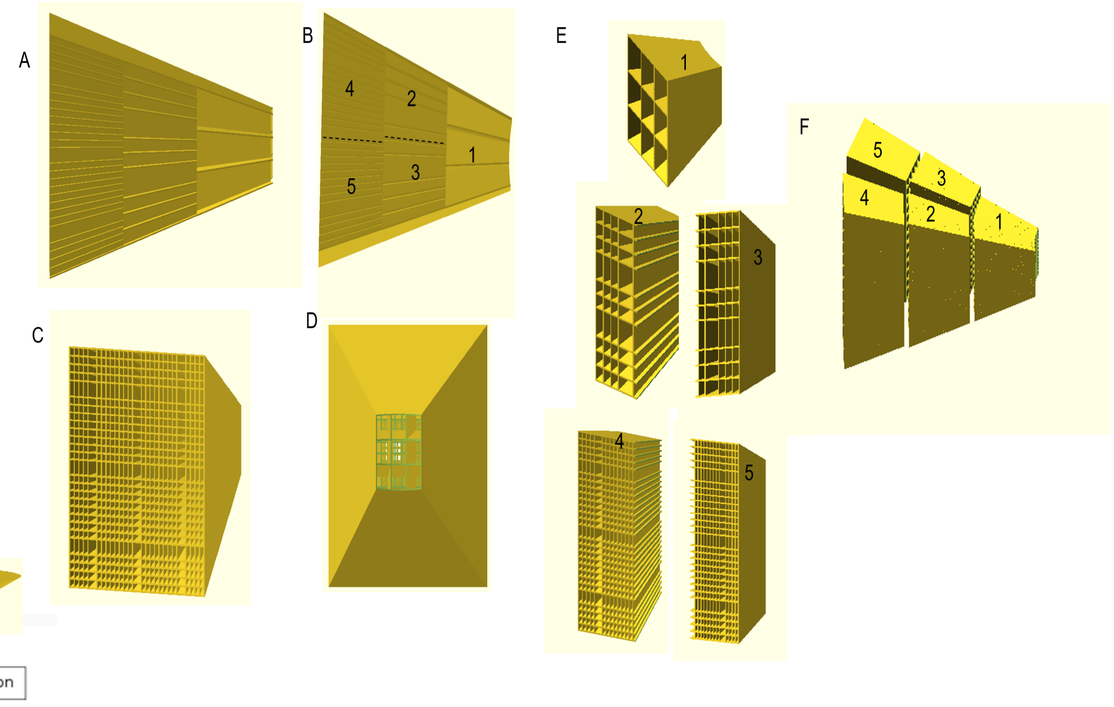


Flow chart

Object Physical modeling written in python script using CSG tree

```python
def anvil(self, girdle_length=6.):
    crown_top_triangle_height=self.crown_top_triangle_height()

    crown_total_triangle_height = self.crown_total_triangle_height(girdle_length)

    pavilion_bottom_triangle_height=self.pavilion_bottom_triangle_height()

    pavilion_total_triangle_height= self.pavilion_total_triangle_height(girdle_length)

    crown_top_cone=operations.translate(shapes.cone(radius='%s *mm' %500, height= '%s*mm' %(crown_top_triangle_height)),
                                        vertical='%s *mm' % (-crown_top_triangle_height))

    crown_total_cone=operations.translate(shapes.cone(radius='%s*mm' % (girdle_length/2,), height= '%s*mm' %crown_total_triangle_height),
                                        vertical='%s *mm' % (-crown_total_triangle_height))

    crown=operations.subtract(crown_total_cone,crown_top_cone)

    pavilion_bottom_cone = operations.translate(shapes.cone(radius='%s *mm' %500., height='%s*mm' %(pavilion_bottom_triangle_height)),
                                        vertical='%s *mm' % (-pavilion_bottom_triangle_height))

    pavilion_total_cone = operations.translate(
        shapes.cone(radius='%s*mm' %(girdle_length / 2), height='%s*mm' %pavilion_total_triangle_height),
        vertical='%s *mm' % (-pavilion_total_triangle_height))

    pavilion=operations.rotate(operations.subtract(pavilion_total_cone,pavilion_bottom_cone),
                               transversal=1, angle='%s *degree' % (180))

    girdle=shapes.cylinder(radius='%s*mm' %(girdle_length/2.), height='%s*mm' %(self.girdle_height))
    girdle_inplace=operations.translate(girdle, vertical='%s *mm' % (pavilion_total_triangle_height+self.girdle_height/2.))
    crown_inplace= operations.translate(crown, vertical='%s *mm' % (pavilion_total_triangle_height+self.girdle_height-crown_total_triangle_height))

    upper_diamond_anvil=operations.unite(operations.unite(crown_inplace, girdle_inplace), pavilion)

    upper_diamond_anvil_at_center= operations.translate(upper_diamond_anvil,vertical='%s *mm' % (-pavilion_bottom_triangle_height) )

    upper_diamond_anvil_at_positive_pt_one_from_center = operations.translate(upper_diamond_anvil_at_center,vertical='%s *mm' % (self.sample_height/2.) )

    lower_diamond_anvil_at_neg_pt_one_from_center = operations.rotate(operations.rotate(upper_diamond_anvil_at_positive_pt_one_from_center,
                                                transversal=1, angle='%s *degree' % (-180)),
                                    vertical=1, angle='%s *degree' % 180)
    return( operations.unite(upper_diamond_anvil_at_positive_pt_one_from_center,lower_diamond_anvil_at_neg_pt_one_from_center   ))
```





Save the geometry into the .xml file with or without file information depending where the geometry is feed into OpenScad or Monte Carlo simulation

Make the sample assembly file including .xml file of the geometry, .cif file for the unit cell information and scattering kernel file (name-scatterer.xml) that feeds into McVine for Monte Carlo simulation

```python
def sample_block(name, shape_name, formula, strutureFiletype):
  return """  <PowderSample name="{name}" type="sample">
    <Shape>
      &{shape_name};
    </Shape>
    <Phase type="crystal">
      <ChemicalFormula>{formula}</ChemicalFormula>
      <{strutureFiletype}file>{formula}.{strutureFiletype}</{strutureFiletype}file>
    </Phase>
  </PowderSample>
""".format(name=name, shape_name=shape_name, formula=formula, strutureFiletype=strutureFiletype)
```

```python
def makeSAXML(sampleassembly_fileName, scatterers=scatterers):

  shape_file_entries = [shape_file_entry(shape_name, shape_fileName)
                        for name, shape_name, shape_fileName, formula,strutureFiletype  in scatterers]
  shape_file_entries='\n'.join(shape_file_entries)
  sample_blocks = [sample_block(name, shape_name, formula,strutureFiletype)
                   for name, shape_name, shape_fileName, formula,strutureFiletype  in scatterers]
  sample_blocks = '\n'.join(sample_blocks)
  lines = ['<Register name="{}" position="(0,0,0)" orientation="(0,0,0)"/>' .format(name)
           for name, shape_name, shape_fileName, formula,strutureFiletype in scatterers]
  geom_regs = '\n '.join(lines)
  text = template.format(shape_file_entries=shape_file_entries, sample_blocks=sample_blocks, geom_regs=geom_regs)
  with open(os.path.join(thisdir, '../sample/sampleassembly_{}.xml'.format(sampleassembly_fileName)), "w") as sam_new:
      sam_new.write(text)
  # return(sampleassembly_fileName)
  return()
```

Run McVine simulation with sample assembly

```python
run_script.run_mpi(
    instrument,simdir, beam=scattered,
    ncount=ncount, nodes=nodes, sample=sample,
    sourceTosample=sourceTosample,
    detector_size=detector_size,
    overwrite_datafiles=True)
```

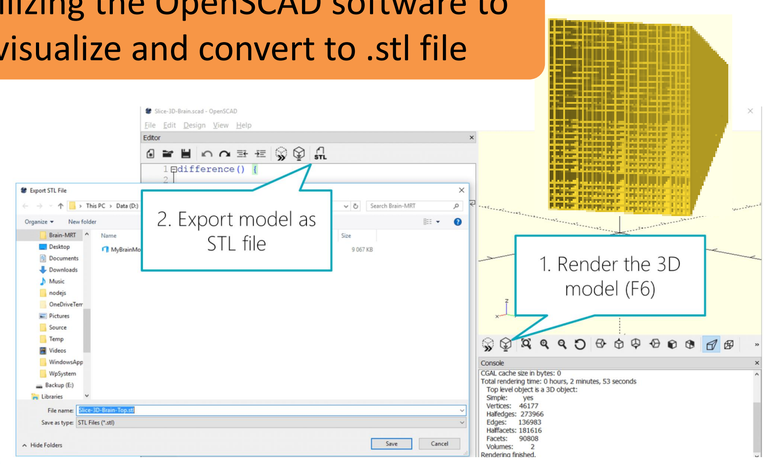
.xml file of the geometry converted to .scad file using SCADGen software of McVinde

```python
def createSCAD(self):
    """
    This function creates the OpenSCAD file from a Parser object.
    """
    fname = self.filename[:-3] + "scad"
    scadfile = open(fname, "w+")
    # If the XML file contains a torus, the torus module is
    # added to the beginning of the OpenSCAD file.
    if self.containsTorus:
        scadfile.write(self.printTorusModule())
    # This for loop causes any root elements that do not have
    # children to not be printed to the OpenSCAD file.
    for elem in self.rootelems:
        if self.containsop and elem.isComp():
            continue
        scadfile.write("{0!s}\n".format(elem))
    scadfile.close()
    return
```

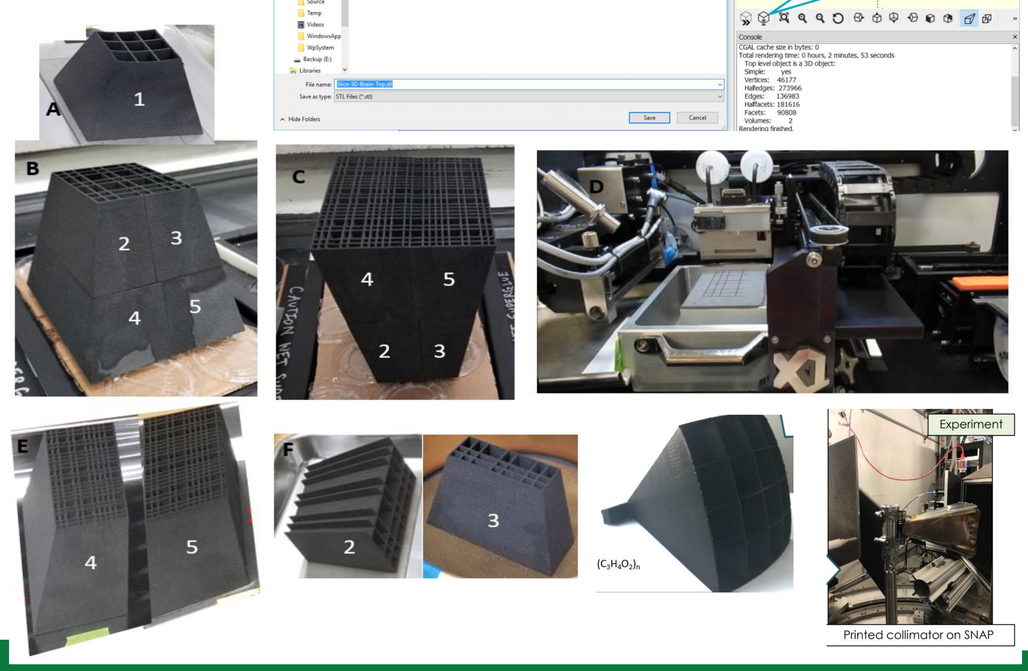
```python
p = SCADGen.Parser.Parser(outputfile)
p.createSCAD()
```

```python
def instrument(beam=None, sample='',
               angleMon1=45,
               angleMon2=135,
               detector_size=0.5,
               sourceTosample_x=0.,
               sourceTosample_y=0., sourceTosample_z=0.):
    a_source = mcvine.components.sources.\
               NeutronFromStorage('source', beam)
    samplename = sample
    samplexml = os.path.join(thisdir,
                '../sample/sampleassembly_%s.xml' % samplename)
    sample = mcvine.components.samples.\
             SampleAssemblyFromXml('sample', samplexml)
    instrument.append(sample,
                      position=(sourceTosample_x, sourceTosample_y,
                       sourceTosample_z), relativeTo=a_source)
```

Utilizing the OpenSCAD software to visualize and convert to .stl file



3D printed parts



Software Used

MCViNE

OpenScad

NumPy

This work is supported by LDRD
SNAP beam times are allocated for this work

OAK RIDGE National Laboratory | SPALLATION NEUTRON SOURCE

U.S. DEPARTMENT OF ENERGY | Office of Science