In [1]:
```
import yt
ds = yt.load("IsolatedGalaxy/galaxy0030/galaxy0030")
```

Data Containers

The Whole Dataset

In [2]:
```
ad = ds.all_data()
```

In [4]:
```
print ad["density"].in_units("Msun/ly**3")
```

```
[  2.09839736e-10   2.10363561e-10   2.10286681e-10 ...,   4.80676642e-05
   6.79464932e-05   4.67670301e-04] Msun/ly**3
```

In [5]:
```
print ad["density"].size
```

```
3644460
```

In [6]:
```
print ad["grid_level"]
```

```
[ 0.  0.  0. ...,  8.  8.  8.] dimensionless
```

In [8]:
```
print ad["dx"].in_units("kpc")
```

```
[ 31.25326528  31.25326528  31.25326528 ...,   0.12208307   0.12208307
   0.12208307] kpc
```

In [10]:
```
print ad["x"].in_units("Mpc")
```

```
[ 0.01562663  0.01562663  0.01562663 ...,  0.49901454  0.49901454
  0.49901454] Mpc
```

Fields are Numpy arrays.

In [11]:
```
print ad["density"] > ds.quan(1e-25, "g/cm**3")
```

```
[False False False ...,  True  True  True]
```

In [13]:
```
my_filter = ad["density"] > ds.quan(1e-25, "g/cm**3")
print ad["temperature"][my_filter]
print ad["temperature"][my_filter].size
```

```
[ 11206.47167969  10385.08398438  10334.16015625 ...,  11824.51855469
  11588.16699219  10173.02148438] K
30949
```

Boxes

In [16]:
```
print ds.domain_left_edge, ds.domain_right_edge
```

```
[ 0.  0.  0.] code_length [ 1.  1.  1.] code_length
```

In [17]:
```
center = [0.5, 0.5, 0.5]
left_edge = [0.4, 0.4, 0.4]
right_edge = [0.6, 0.6, 0.6]
my_region = ds.region(center, left_edge, right_edge)
```

In [18]:
```
print my_region["density"]
```

```
[   2.05686132e-27   1.98130330e-27   1.88577721e-27 ...,   1.12879234e-25
    1.59561490e-25   1.09824903e-24] g/cm**3
```

In [19]:
```
print my_region["density"].size
```

```
2974536
```

In [20]:
```
my_box = ds.box(left_edge, right_edge)
```

In [21]:
```
print my_box["density"].size
```

```
2974536
```

Spheres

In [22]:
```
sphere_center = ds.domain_center
sphere_radius = ds.quan(5, "kpc")
```

In [23]:
```
my_sphere = ds.sphere(sphere_center, sphere_radius)
```

In [24]:
```
print my_sphere["density"]
```

```
[  1.05388186e-25   9.72223032e-26   3.75294504e-26 ...,   1.12879234e-25
   1.59561490e-25   1.09824903e-24] g/cm**3
```

In [25]:
```
print my_sphere["density"].size
```

```
195343
```

In [28]:
```
print my_sphere.center.in_units("ly"), my_sphere.radius.in_units("pc")
```

```
[ 1630987.12163  1630987.12163  1630987.12163] ly 5000.0 pc
```

In [30]:
```
print my_sphere["radius"].in_units("pc")
```

```
[ 4945.11777518  4948.13079467  4954.15133627 ...,  2726.44591729
  2715.49079145  2709.99662134] pc
```

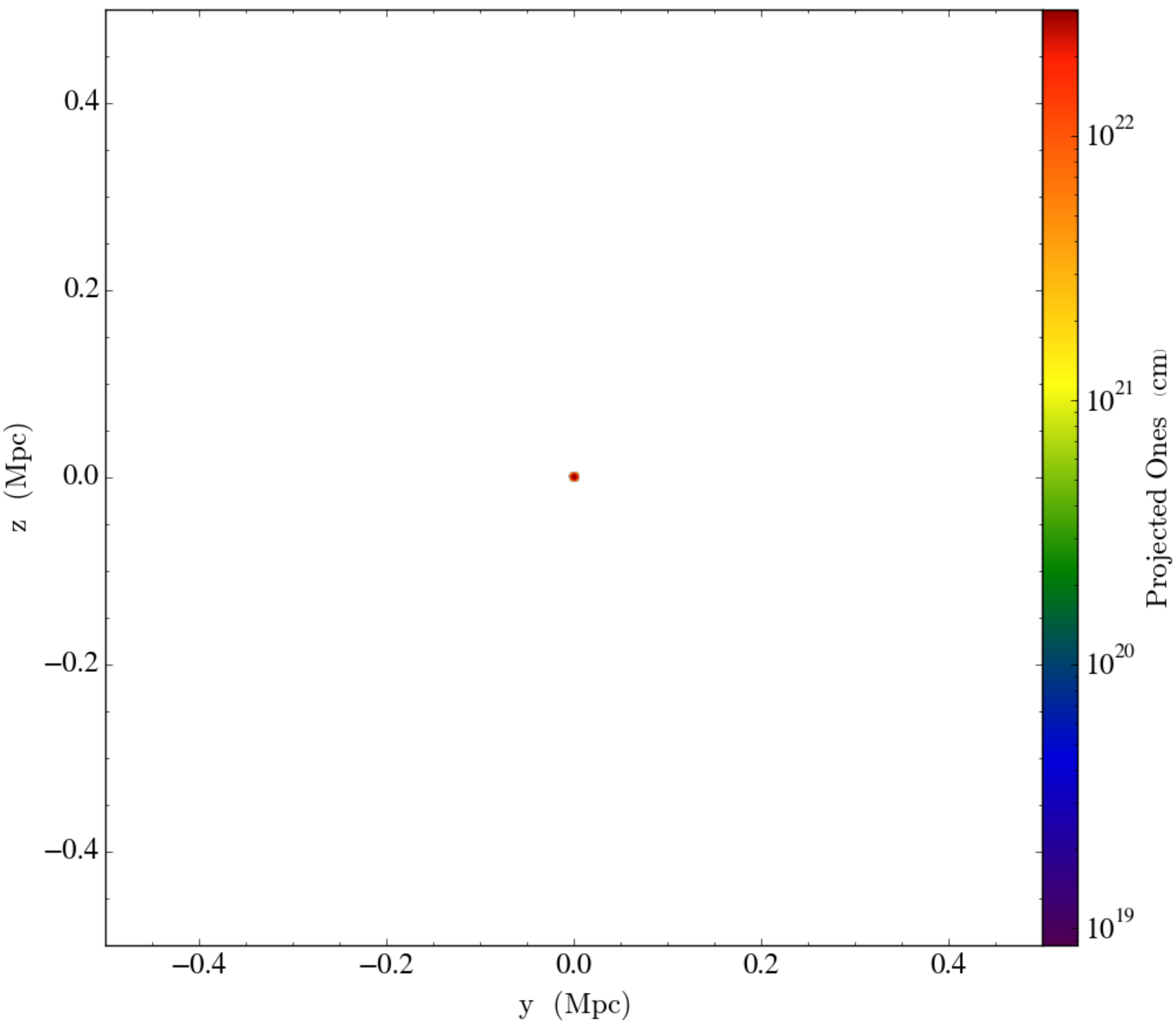
In [31]:
```
print my_sphere["radius"].in_units("kpc")
```

```
[ 4.94511778  4.94813079  4.95415134 ...,  2.72644592  2.71549079
  2.70999662] kpc
```

Prove to me this is a sphere!

In [32]:
```
p = yt.ProjectionPlot(ds, "x", "ones", data_source=my_sphere)
```

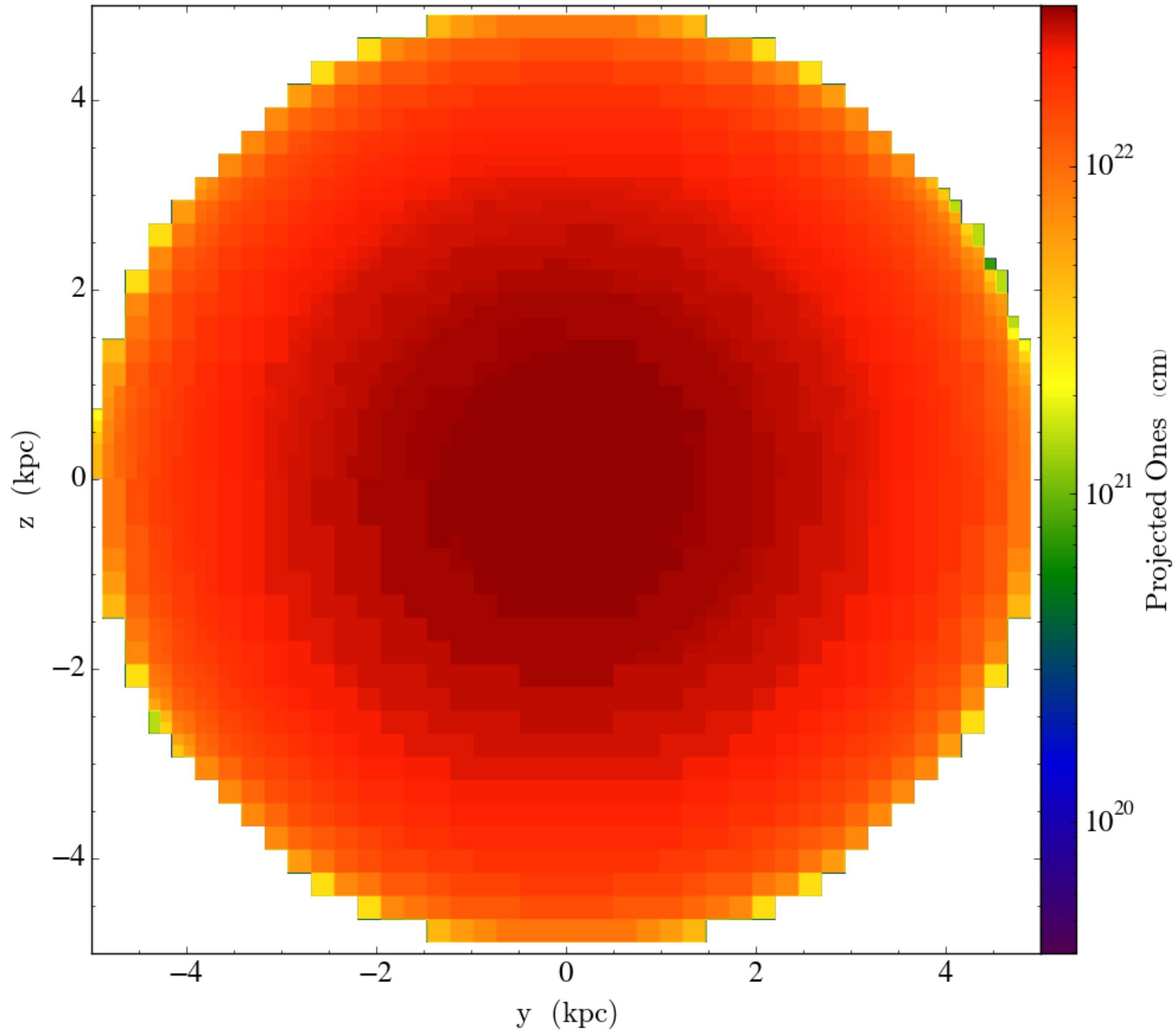
In [33]:
```
p.show()
```



0.4
0.2
0.0
−0.2
−0.4
z (Mpc)
−0.4
−0.2
0.0
0.2
0.4
y (Mpc)
10^{22}
10^{21}
10^{20}
10^{19}
Projected Ones (cm)

In [34]:
```
p.set_width(10, "kpc")
```

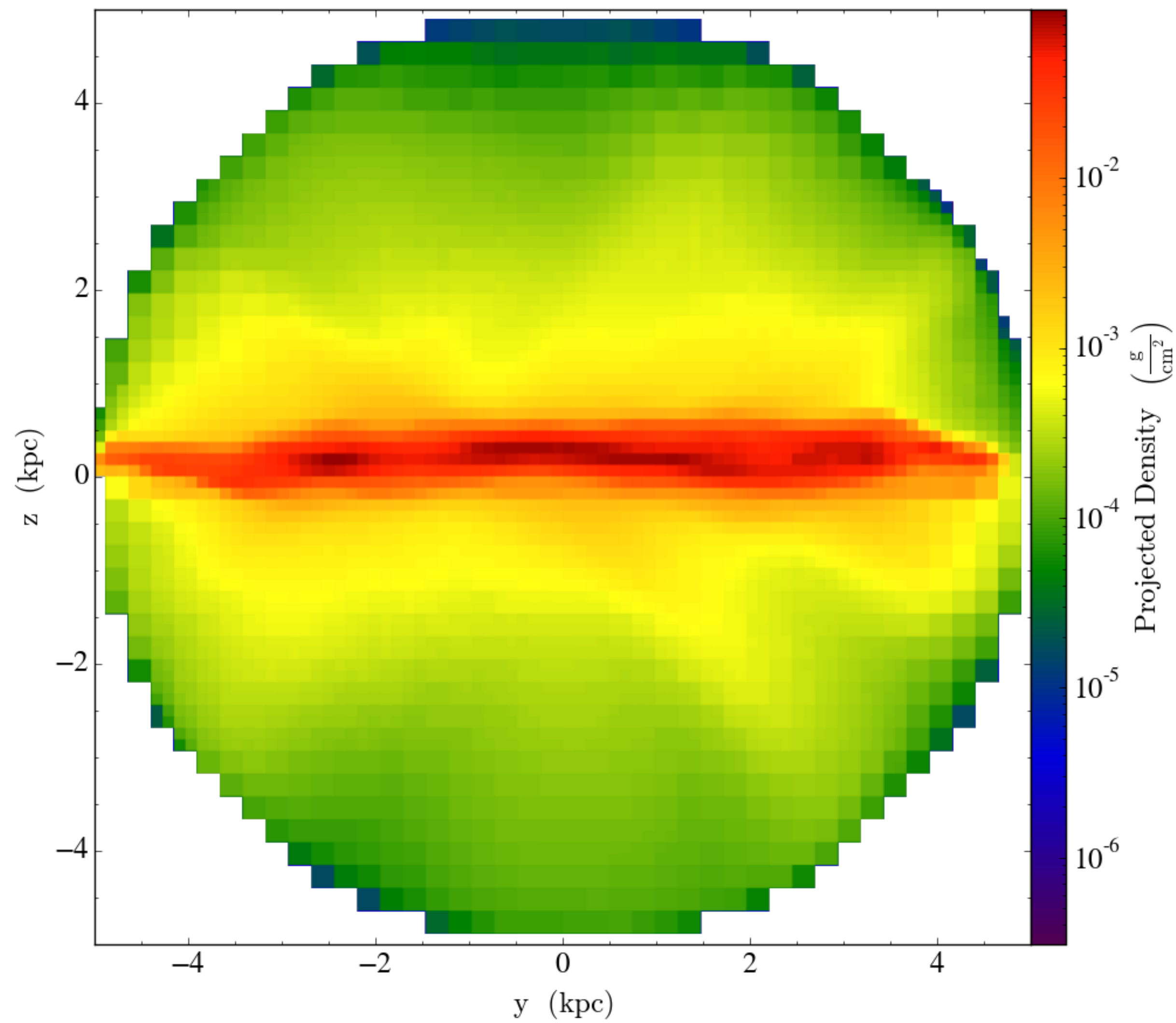
Out[34]:



In [35]:
```
p = yt.ProjectionPlot(ds, "x", "density", data_source=my_sphere)
```

In [36]:
```
p.set_width(10, "kpc")
```

Out[36]:



Disks

In [37]:
```
disk_center = sphere_center
disk_normal = [0, 0, 1]
disk_radius = sphere_radius
disk_height = ds.quan(1, "kpc")
```

In [38]:
```
my_disk = ds.disk(disk_center, disk_normal, disk_radius, disk_height)
```

In [39]:
```
print my_disk["density"]
```

```
[  1.05388186e-25   9.72223032e-26   3.75294504e-26 ...,   1.12879234e-25
   1.59561490e-25   1.09824903e-24] g/cm**3
```

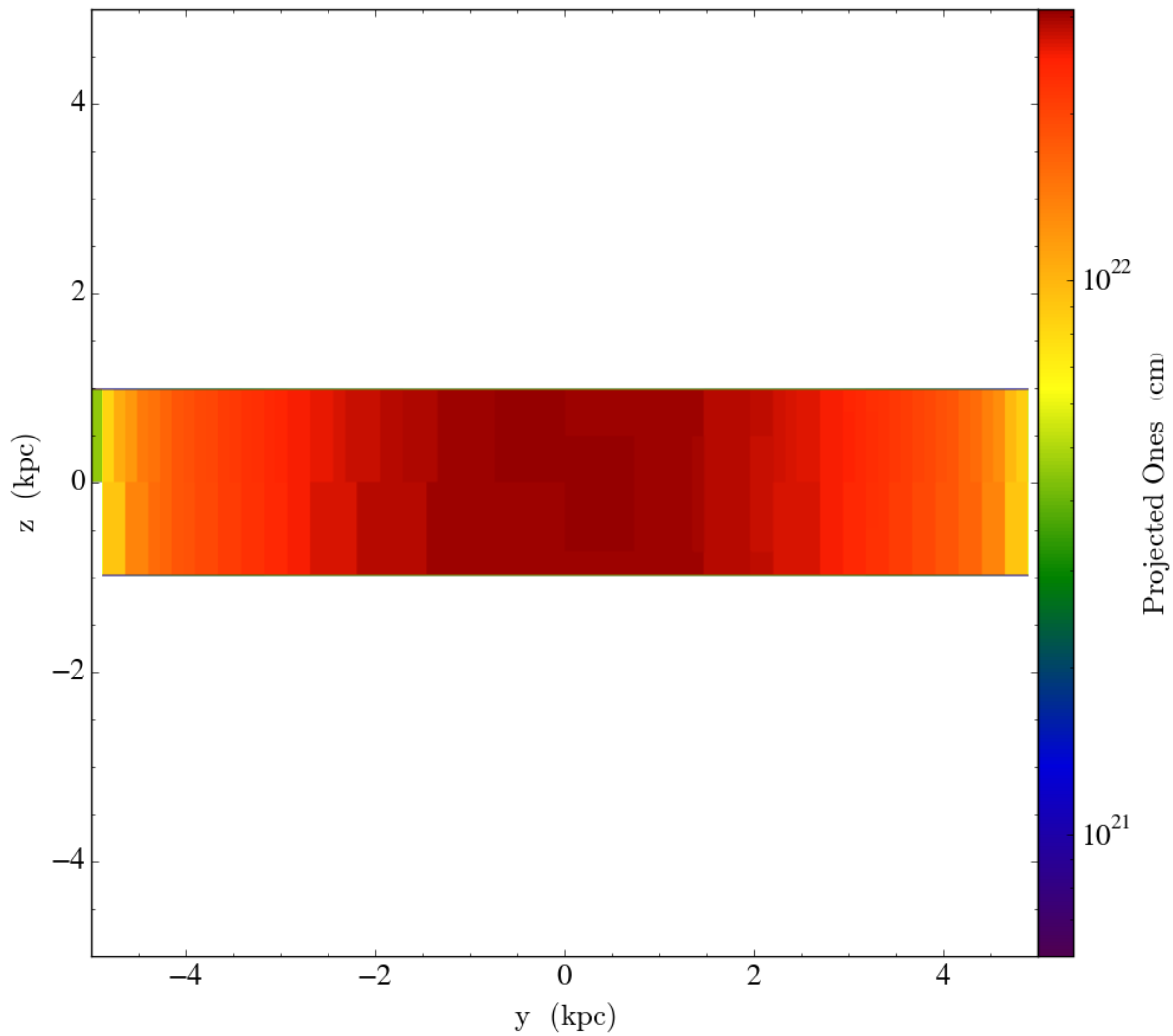
In [40]:
```
print my_disk["density"].size
```

72365

Prove it!

In [41]:
```
p = yt.ProjectionPlot(ds, "x", "ones", data_source=my_disk)
p.set_width(10, "kpc")
```

Out[41]:



"radius" is spherical radius

In [42]:
```
radius = my_disk["radius"].in_units("pc")
print radius.min(), radius.max()
```

105.72703782 pc 5078.93439548 pc

"cylindrical_r" is cylindrical radius

In [43]:
```
cyl_rad = my_disk["cylindrical_r"].in_units("pc")
print cyl_rad.min(), cyl_rad.max()
```

```
86.3257648916 pc 4995.71907778 pc
```

In [44]:
```
height = my_disk["cylindrical_z"].in_units("pc")
print height.min(), height.max()
```

```
61.041533746 pc 915.62300619 pc
```

Ray

In [45]:
```
my_ray = ds.ray(ds.domain_left_edge, ds.domain_right_edge)
```

In [47]:
```
my_ray["density"].size
```

Out[47]:
```
280
```

Derived Quantities

Derived quantity functions are associated with data containers.

In [48]:
```
my_sphere.quantities.total_mass()
```

Out[48]:
```
[2.43089826394e+42 g, 5.1819442798e+43 g]
```

In [49]:
```
my_disk.quantities.total_mass()
```

Out[49]:
```
[2.2686566769e+42 g, 3.47151444103e+43 g]
```

View all available derived quantities:

In [53]:
```
print my_sphere.quantities.keys()
```

```
['SpinParameter', 'MinLocation', 'WeightedVariance', 'TotalMass', 'Angula
rMomentumVector', 'WeightedAverageQuantity', 'TotalQuantity', 'CenterOfMa
ss', 'BulkVelocity', 'Extrema', 'MaxLocation']
```

Two access methods:

["TotalMass"] or .total_mass

In [54]:
```
my_sphere.quantities["TotalMass"]()
```

Out[54]:
```
[2.43089826394e+42 g, 5.1819442798e+43 g]
```

In [55]:
```
my_sphere.quantities.total_mass()
```

Out[55]:
```
[2.43089826394e+42 g, 5.1819442798e+43 g]
```

Available quantities:

Extrema

In [56]:
```
ad.quantities.extrema("temperature")
```

Out[56]:
```
(20.8445072174 K, 24826104.0 K)
```

Location of Maximum Value

In [58]:
```
val, i, x, y, z = ad.quantities.max_location("density")
print val, x, y, z
```

```
7.73426503924e-24 g/cm**3 0.504089355469 code_length 0.499816894531 code_
length 0.500183105469 code_length
```

Location of Minimum Value

In [59]:
```
val, i, x, y, z = ad.quantities.min_location("density")
print val, x, y, z
```

```
8.47293750754e-32 g/cm**3 0.44921875 code_length 0.51953125 code_length 0
.31640625 code_length
```

Center of Mass

In [60]:
```
ad.quantities.center_of_mass()
```

Out[60]:
```
YTArray([ 0.49995915,  0.50011153,  0.50005016]) code_length
```

Bulk Velocity

In [61]:
```
ad.quantities.bulk_velocity()
```

Out[61]:
```
YTArray([ 37021.05826389,  35794.63088302,  82204.27080637]) cm/s
```

Spin Parameter

In [62]:
```
ad.quantities.spin_parameter()
```

Out[62]:
```
0.343863349035 sqrt(erg)*s/(cm*sqrt(g))
```

Weighted Averages

In [65]:
```
ad.quantities.weighted_average_quantity("temperature", "cell_mass")
```

Out[65]:
```
10593.6137122 K
```

Weighted Variance

In [67]:
```
var, mean = ad.quantities.weighted_variance("temperature", "cell_mass")
print mean, var
```

```
10593.6137122 K 18510.9276191 K
```

Summation

In [68]:
```
ad.quantities.total_quantity("cell_mass")
```

Out[68]:
```
3.68449915786e+43 g
```

Angular Momentum Vector

In [69]:
```
ad.quantities.angular_momentum_vector()
```

Out[69]:
```
(7.50868209385e+26 cm**2/s,
 -1.06032596384e+27 cm**2/s,
 -2.19274002072e+29 cm**2/s)
```

Total Mass

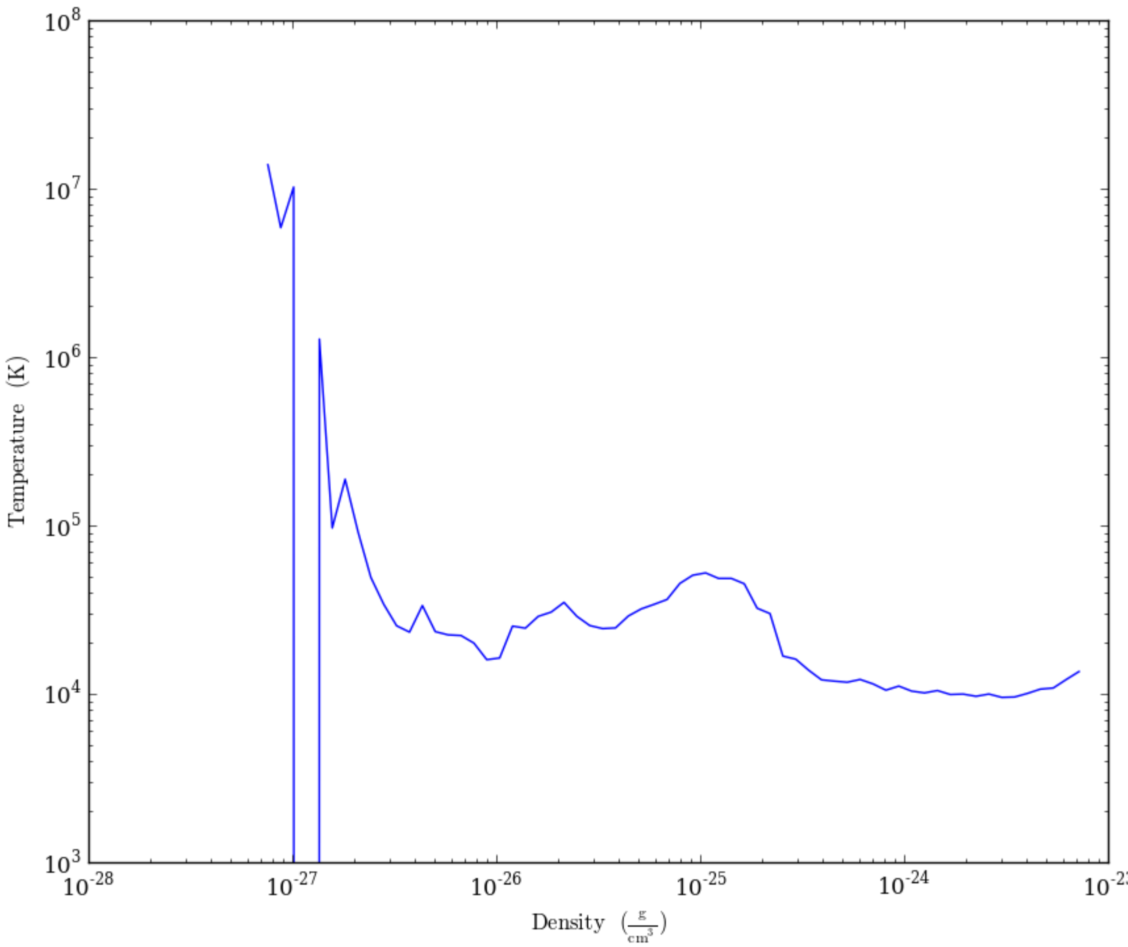
In [70]:
```
gas, dark_matter = ad.quantities.total_mass()
print gas, dark_matter
```

```
3.68449915786e+43 g 4.4634255032e+44 g
```

Time for a challenge!

1D Profiles

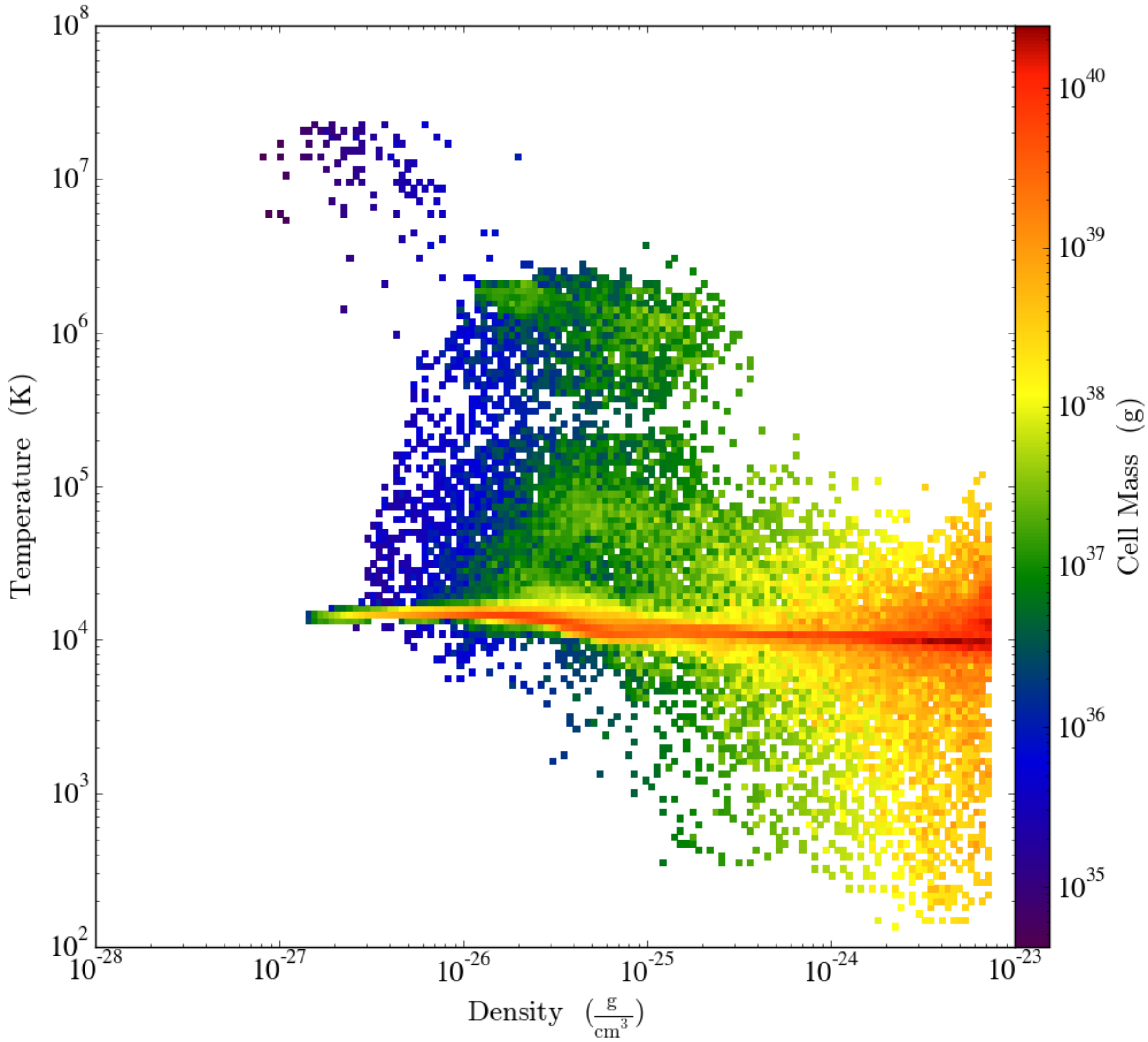
In [74]:
```
p = yt.ProfilePlot(my_sphere, "density", "temperature",
                   weight_field="cell_mass")
p.show()
```



2D Profiles (Phase Plots)

In [75]:
```
p = yt.PhasePlot(my_sphere, "density", "temperature", "cell_mass",
                 weight_field=None)

p.show()
```



Derived Fields

Define a field function.

In [76]:
```
from yt.utilities.physical_constants import kb
print kb
def my_field(field, data):
    return kb * data["temperature"] * data["number_density"]**(-2./3)
```

```
1.3806488e-16 erg/K
```

Add the field to the dataset.

In [78]:
```
ds.add_field("my_entropy", function=my_field, units="keV*cm**2")
```

Use the new field!

In [79]:
```
print ad["my_entropy"]
```

```
[  1.27681821e+01   1.27711027e+01   1.27674566e+01 ...,   5.44439902e-03
   4.42738458e-03   1.28677870e-03] cm**2*keV
```

In [80]:
```
print ad["my_entropy"].in_units("ly**2*J")
```

```
[  2.28564435e-51   2.28616717e-51   2.28551447e-51 ...,   9.74607014e-55
   7.92550298e-55   2.30347471e-55] J*ly**2
```

Some fields require additional information.

For example, the "radial_velocity" field should be with to the motion of the object.

In [81]:
```
print my_disk["radial_velocity"]
```

```
[ 14044810.45986197  13154876.26945174  18859748.09329093 ...,
  15726817.82514748  14216578.67894871   1630144.98317282] cm/s
```

Field parameters can be set to provide this information.

In [82]:
```
disk_bulk_velocity = my_disk.quantities.bulk_velocity()
print disk_bulk_velocity
my_disk.set_field_parameter("bulk_velocity", disk_bulk_velocity)
```

```
[ -316351.73861518   258546.74074861  1039941.20812727] cm/s
```

In [84]:
```
del my_disk["radial_velocity"]
print my_disk["radial_velocity"]
```

```
[ 13777371.27463491  12861941.91878596  18541542.89788134 ...,
  15493479.7711093   14029052.95664273   1489087.5529269 ] cm/s
```

This is how fields like "radius" work.

In [ ]:

```
def my_radial_velocity(field, data):
    redshift = data.ds.current_redshift
    bulk_velocity = data.get_field_parameter("bulk_velocity")
    # do some calcuation
    return data[...]
```

Challenge time!

In [ ]:

```
```