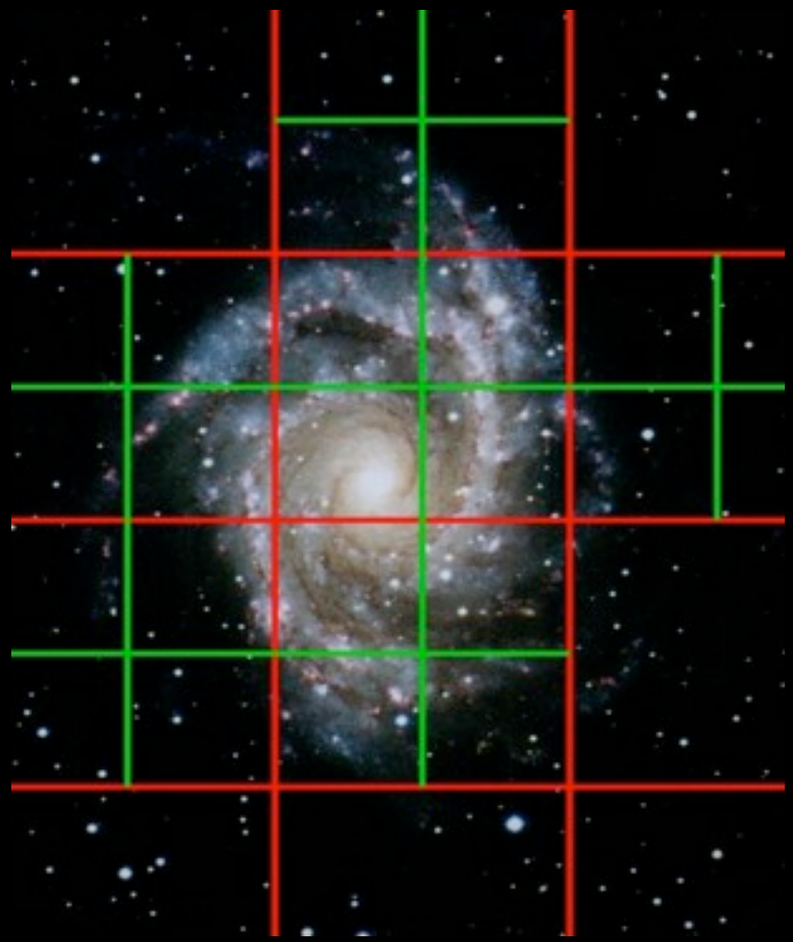
Setting up a new simulation

Elizabeth Tasker

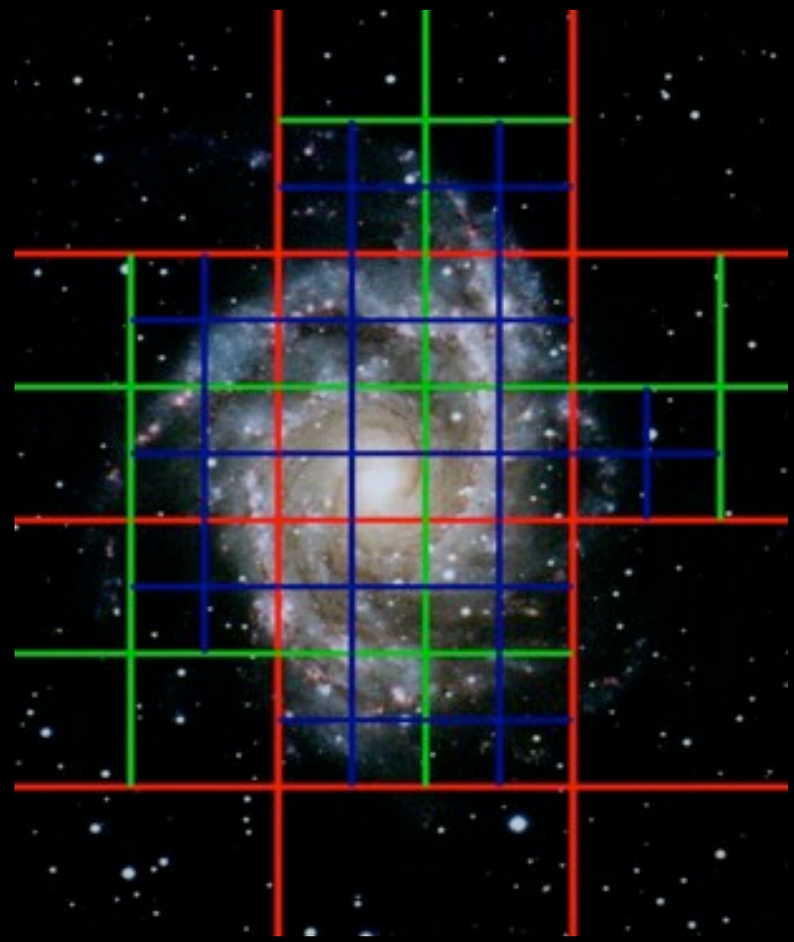
Hokkaido University

Terminology

level 0



level 1



level 2

"Top Grid"

"Root Grid"

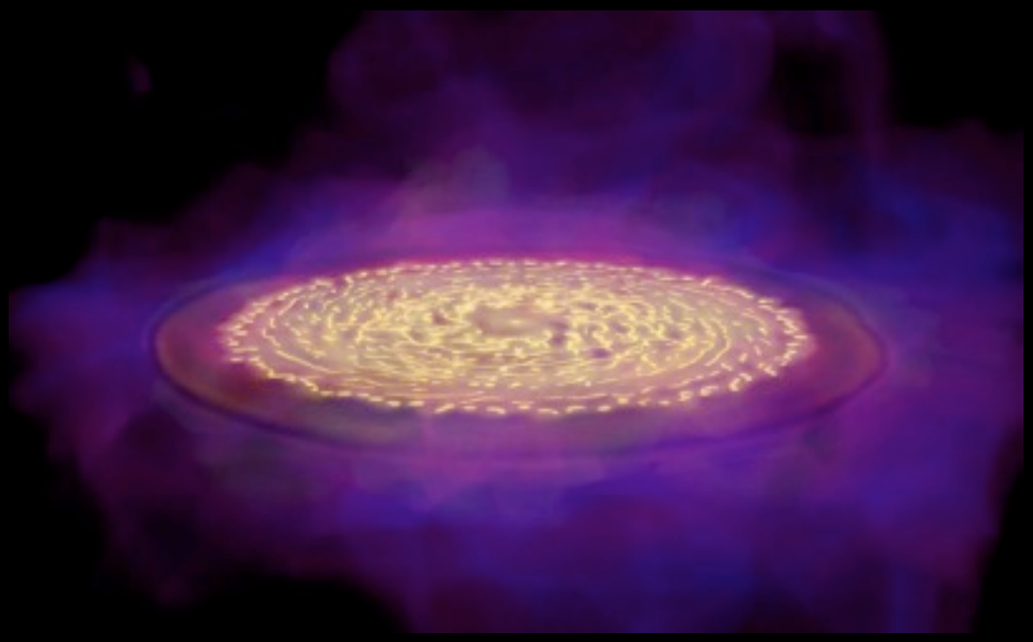
parent

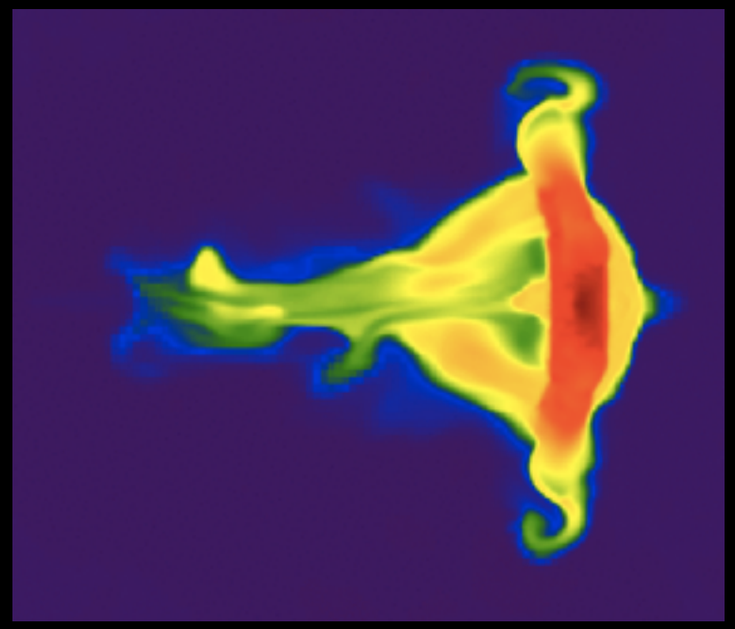
"Child Grid"

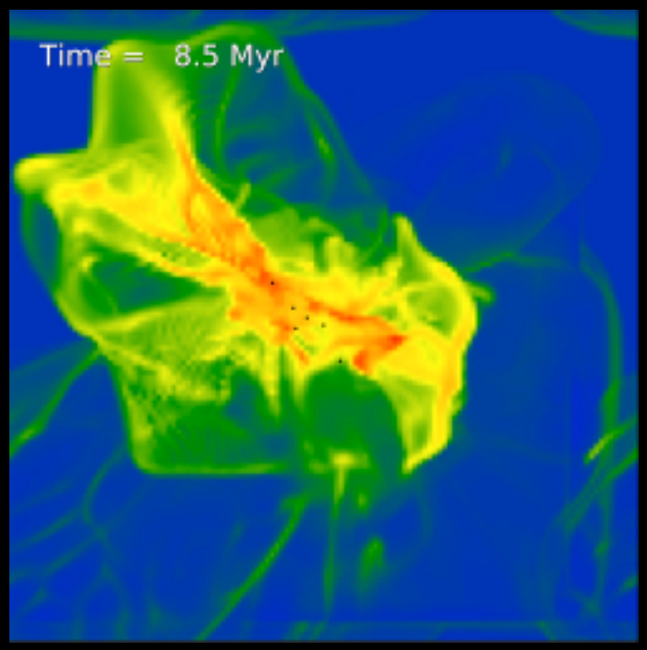
parent

"Child Grid"

A cosmology simulation is only 1 choice...

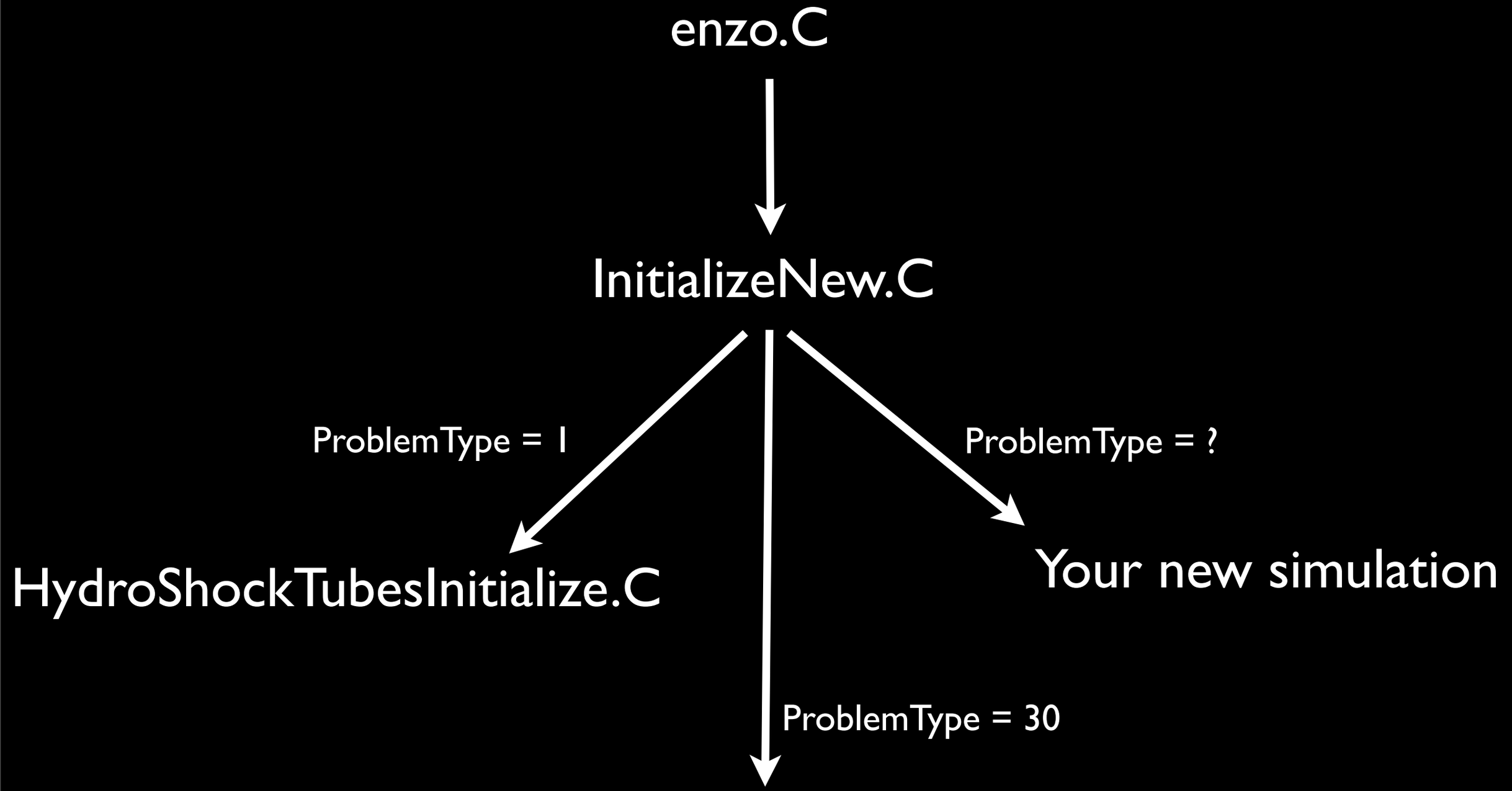






Since Enzo models both gas and particles,
it can be used for most astrophysical problems

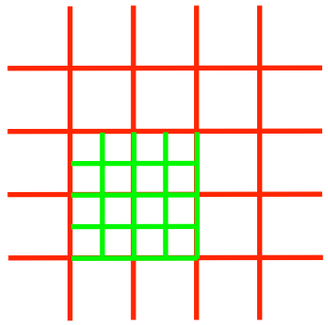
When you run Enzo:



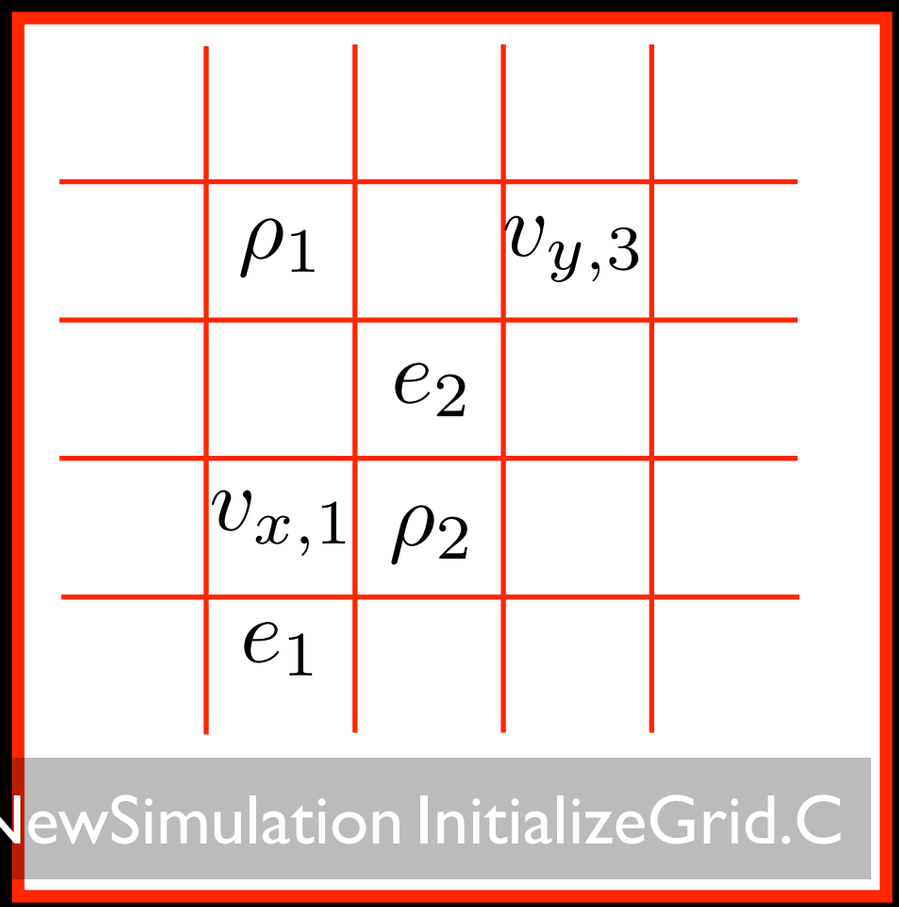
NewSimulationInitialize.C

Read parameters

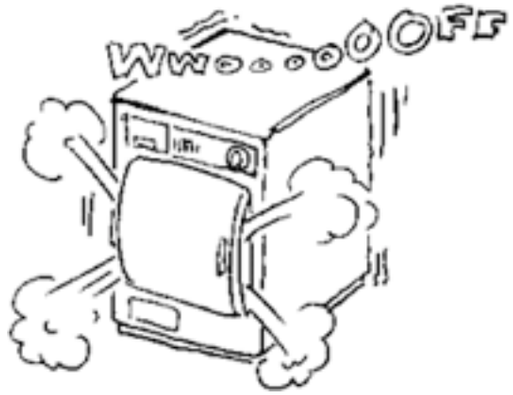
Set up
grid levels



ProblemType = 12
StarParticleCreation = 1
StarParticleFeedback = 1
StaticHierarchy = 0
.
.
.



Grid_NewSimulation InitializeGrid.C



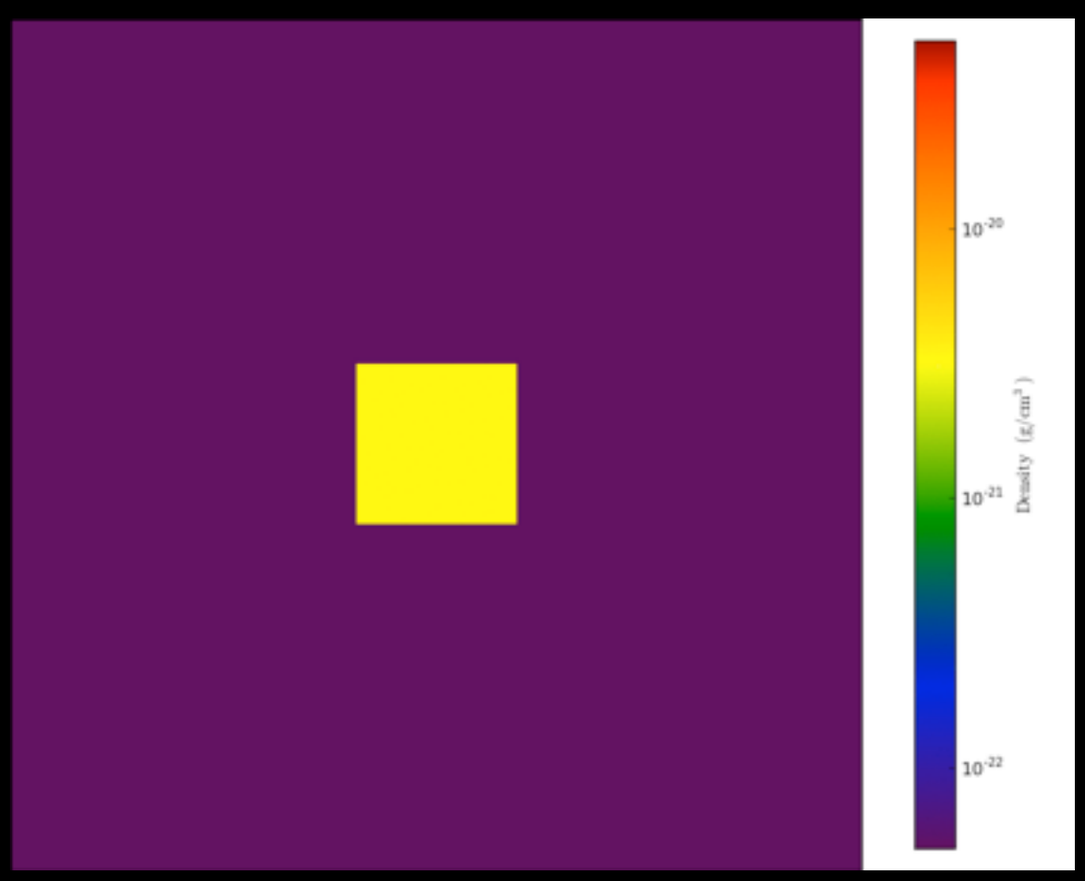
Enzo

An example: Rotating Cylinder

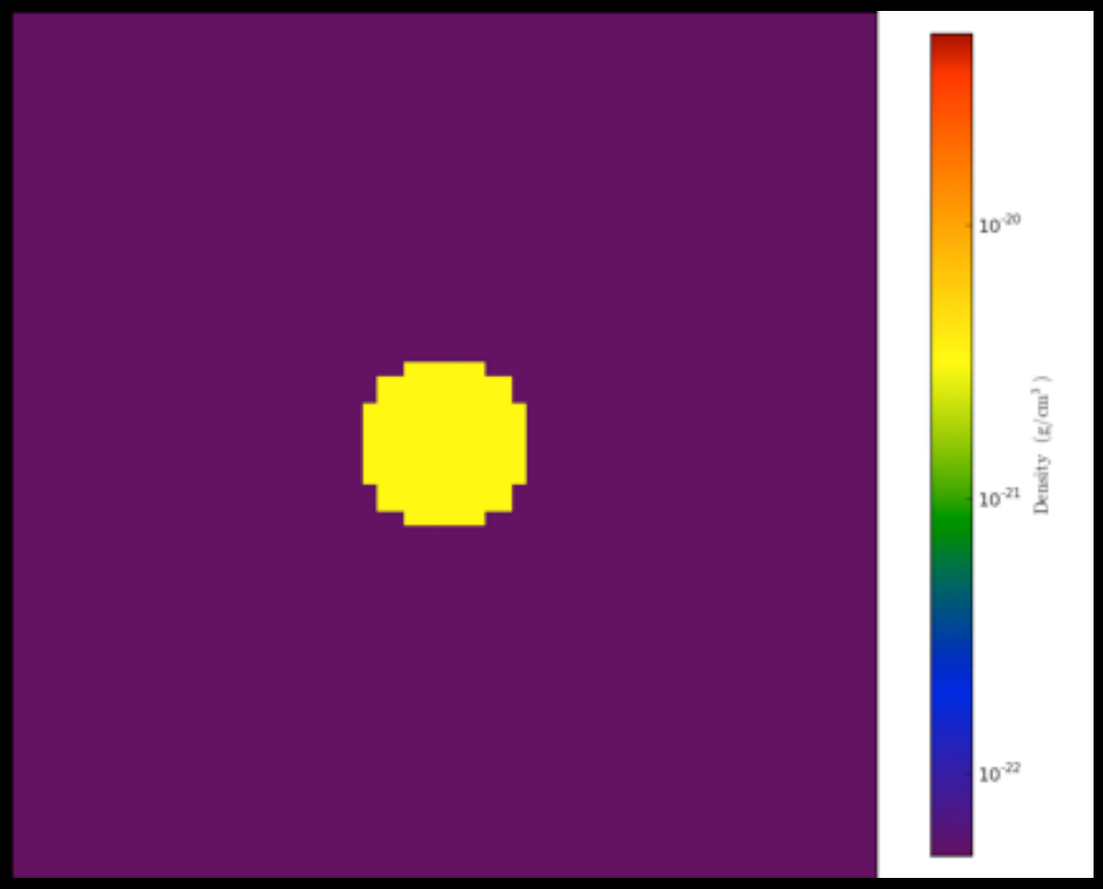
cp workshop/RotatingCylinderInitialize.C enzo/src/enzo/.

cp workshop/Grid_RotatingCylinderInitializeGrid.C enzo/src/enzo/.

x-direction

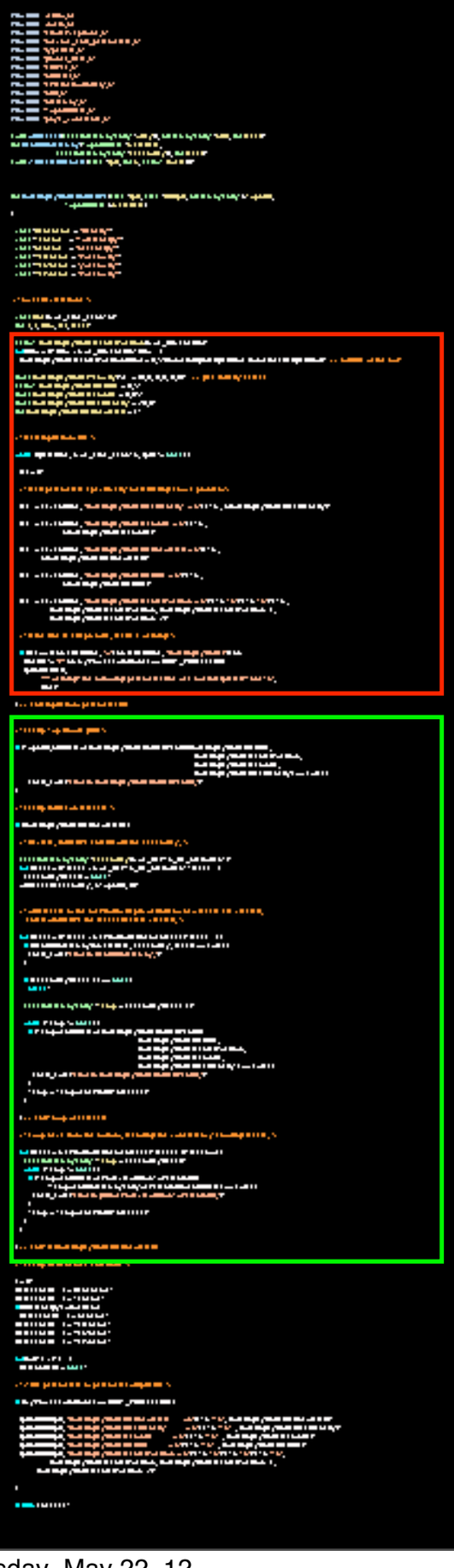


z-direction



RotatingCylinderInitialize.C

(called by InitializeNew.C)



Read parameters

Set up grid levels

2 choices for setting up initial grids

Adaptive meshes

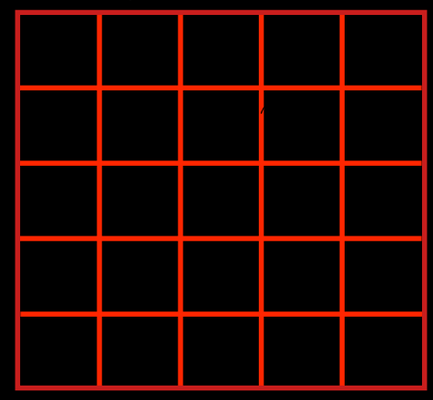
Child grid location based on parent grid properties

ρ_1 ρ_1 ρ_1 ρ_1 ρ_1
ρ_1 ρ_1 ρ_1 ρ_1 ρ_1
ρ_1 ρ_1 ρ_1 ρ_1 ρ_1
ρ_1 ρ_1 ρ_1 ρ_1 ρ_1
ρ_1 ρ_1 ρ_1 ρ_1 ρ_1

Subgrids

Child grid location based on fixed position

(x_l, y_l, z_l)



(x_r, y_r, z_r)

RotatingCylinderInitialize.C

```
FLOAT RotatingCylinderCenterPosition[MAX_DIMENSION];
for(dim = 0; dim < MAX_DIMENSION; dim++)
  RotatingCylinderCenterPosition[dim] = 0.5*(DomainRightEdge[dim]+DomainLeftEdge[dim]);  // middle of the box

float RotatingCylinderVelocity[3]   = {0.0, 0.0, 0.0};   // gas initally at rest
FLOAT RotatingCylinderRadius = 0.3;
float RotatingCylinderLambda = 0.05;
float RotatingCylinderOverdensity = 20.0;
int RotatingCylinderRefineAtStart = 1;
```

Set defaults

```
/* read input from file */

while (fgets(line, MAX_LINE_LENGTH, fptr) != NULL) {

  ret = 0;

  /* read parameters specifically for radiating shock problem*/

  ret += sscanf(line, "RotatingCylinderOverdensity  = %"FSYM, &RotatingCylinderOverdensity);

  ret += sscanf(line, "RotatingCylinderLambda = %"FSYM,
                &RotatingCylinderLambda);

  ret += sscanf(line, "RotatingCylinderRefineAtStart = %"ISYM,
                &RotatingCylinderRefineAtStart);

  ret += sscanf(line, "RotatingCylinderRadius = %"PSYM,
                &RotatingCylinderRadius);

  ret += sscanf(line, "RotatingCylinderCenterPosition = %"PSYM" %"PSYM" %"PSYM,
                RotatingCylinderCenterPosition, RotatingCylinderCenterPosition+1,
                RotatingCylinderCenterPosition+2);
```

Read problem-specific parameters

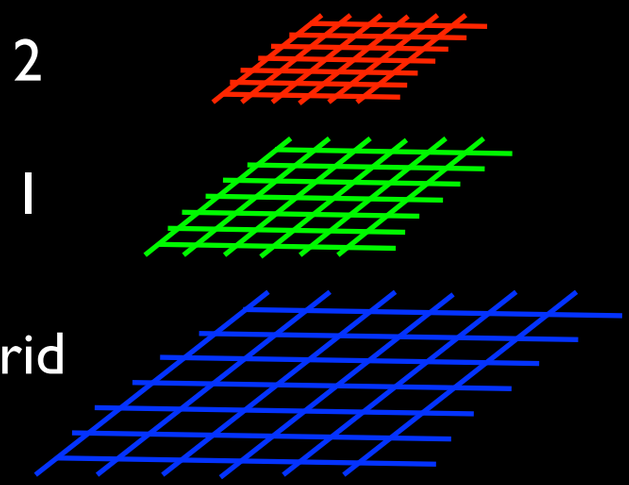
```
  /* if the line is suspicious, issue a warning */

  if (ret == 0 && strstr(line, "=") && (strstr(line, "RotatingCylinder")) &&
      line[0] != '#' && MyProcessorNumber == ROOT_PROCESSOR)
    fprintf(stderr,
            "*** warning: the following parameter line was not interpreted:\n%s\n",
            line);

} // end input from parameter file
```

Check we've not missed any

```
| /* Set up Top (Root)  grid */

  if (TopGrid.GridData->RotatingCylinderInitializeGrid(RotatingCylinderRadius,
                                                       RotatingCylinderCenterPosition,
                                                       RotatingCylinderLambda,
                                                       RotatingCylinderOverdensity) == F
      ENZO_FAIL("Error in RotatingCylinderInitializeGrid.");
  }

  /* Set up initial AMR levels */

  if (RotatingCylinderRefineAtStart) {

    /* Declare, initialize and fill out the LevelArray. */

    LevelHierarchyEntry *LevelArray[MAX_DEPTH_OF_HIERARCHY];
    for (level = 0; level < MAX_DEPTH_OF_HIERARCHY; level++)
      LevelArray[level] = NULL;
    AddLevel(LevelArray, &TopGrid, 0);


    /* Add levels to the maximum depth or until no new levels are created,
       and re-initialize the level after it is created. */

    for (level = 0; level < MaximumRefinementLevel; level++) {
      if (RebuildHierarchy(&MetaData, LevelArray, level) == FAIL) {
        ENZO_FAIL("Error in RebuildHierarchy.");
      }

      if (LevelArray[level+1] == NULL)
        break;

      LevelHierarchyEntry *Temp = LevelArray[level+1];

      while (Temp != NULL) {
        if (Temp->GridData->RotatingCylinderInitializeGrid(
                                        RotatingCylinderRadius,
                                        RotatingCylinderCenterPosition,
                                        RotatingCylinderLambda,
                                        RotatingCylinderOverdensity ) == FAIL) {
          ENZO_FAIL("Error in RotatingCylinderInitializeGrid.");
        }
        Temp = Temp->NextGridThisLevel;
      }

    } // end: loop over levels
```

Set $\rho, e, \bar{v}$ cells in top grid

Create AMR hierarchy

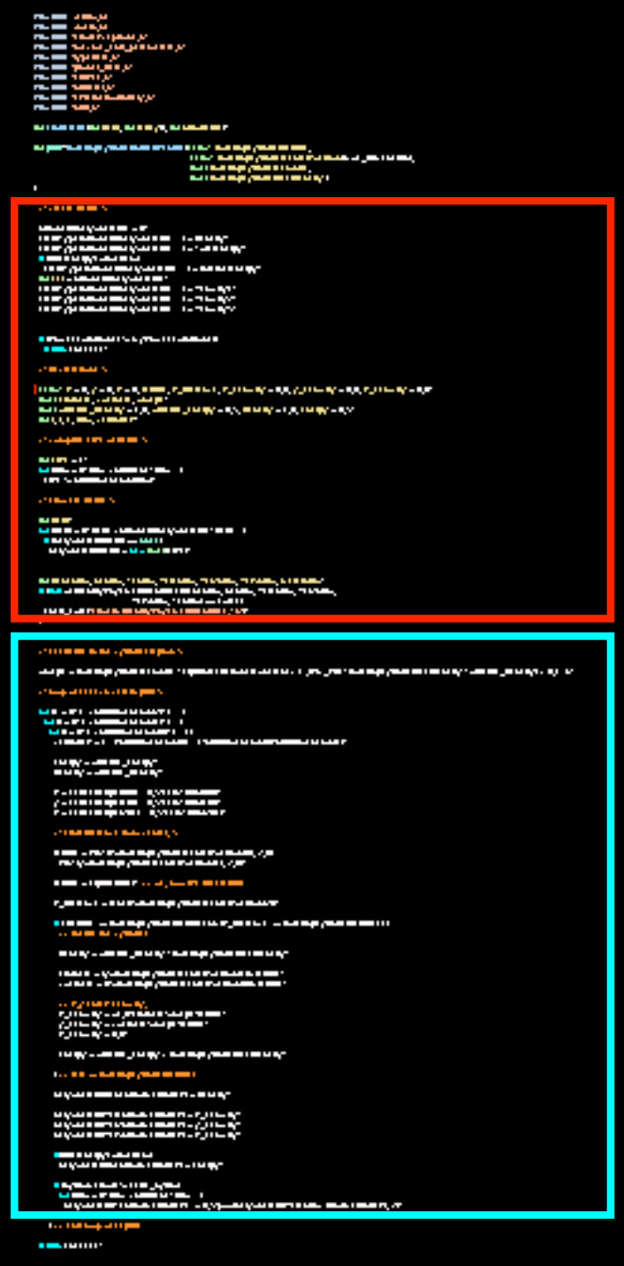


Set $\rho, e, \bar{v}$ cells in child grid

Largely identical for all problem types

Grid_RotatingCylinderInitializeGrid.C

(called by RotatingCylinderInitialize.C)



Assign memory for $\rho, e, \bar{v}$ fields

Set $\rho, e, \bar{v}$ for each cell

Grid_RotatingCylinderInitializeGrid.C

```
/* create fields */

NumberOfBaryonFields = 0;
FieldType[NumberOfBaryonFields++] = Density;
FieldType[NumberOfBaryonFields++] = TotalEnergy;
if (DualEnergyFormalism)
  FieldType[NumberOfBaryonFields++] = InternalEnergy;
int vel = NumberOfBaryonFields;
FieldType[NumberOfBaryonFields++] = Velocity1;
FieldType[NumberOfBaryonFields++] = Velocity2;
FieldType[NumberOfBaryonFields++] = Velocity3;
```

Create $\rho, e, \bar{v}$ fields

```
if (ProcessorNumber != MyProcessorNumber)
  return SUCCESS;
```

Only do this on 1 processor

```
/* declarations */

FLOAT x = 0, y = 0, z = 0, radius, z_distance, x_velocity = 0.0, y_velocity = 0.0, z_velocity = 0.0;
float sintheta, costheta, omega;
float outside_density = 1.0, outside_energy = 0.5, density = 1.0, energy = 0.5;
int i, j, k, dim, cellindex;

/* compute size of fields */

int size = 1;
for (dim = 0; dim < GridRank; dim++)
  size *= GridDimension[dim];

/* allocate fields */

int field;
for (field = 0; field < NumberOfBaryonFields; field++)
  if (BaryonField[field] == NULL)
    BaryonField[field] = new float[size];
```

Assign memory for fields

```
int DensNum, GENum, TENum, Vel1Num, Vel2Num, Vel3Num, MetalNum;
if (this->IdentifyPhysicalQuantities(DensNum, GENum, Vel1Num, Vel2Num,
                                     Vel3Num, TENum) == FAIL) {
  ENZO_FAIL("Error in IdentifyPhysicalQuantities.\n");
}
```

Useful function for finding fields

```
for (k = 0; k < GridDimension[2]; k++)
 for (j = 0; j < GridDimension[1]; j++)
  for (i = 0; i < GridDimension[0]; i++) {
   cellindex = i + j*GridDimension[0] + k*GridDimension[0]*GridDimension[1];
```

Loop over cells

```
   energy = outside_energy;
   density = outside_density;
```

set background values

```
   x = CellLeftEdge[0][i] + 0.5*CellWidth[0][i];
   y = CellLeftEdge[1][j] + 0.5*CellWidth[1][j];
   z = CellLeftEdge[2][k] + 0.5*CellWidth[2][k];

   /* Find distance from center. */

   radius = POW(x-RotatingCylinderCenterPosition[0], 2.0) +
    POW(y-RotatingCylinderCenterPosition[1], 2.0);

   radius = sqrt(radius);  // ok, now it's just radius

   z_distance = fabs(z-RotatingCylinderCenterPosition[2]);
```

Cell position

```
   if ( (radius <= RotatingCylinderRadius) && (z_distance <= RotatingCylinderRadius) ) {
    // inside the cylinder

    density = outside_density * RotatingCylinderOverdensity;

    sintheta = (y-RotatingCylinderCenterPosition[1])/radius;
    costheta = (x-RotatingCylinderCenterPosition[0])/radius;

    // x,y and z velocity.
    x_velocity = -1.0*sintheta*omega*radius;
    y_velocity = costheta*omega*radius;
    z_velocity = 0.0;

    energy = outside_energy / RotatingCylinderOverdensity;

   } // if (r <= RotatingCylinderRadius)
```

$\rho, e, \bar{v}$ inside cylinder

```
   BaryonField[DensNum][cellindex] = density;

   BaryonField[Vel1Num][cellindex] = x_velocity;
   BaryonField[Vel2Num][cellindex] = y_velocity;
   BaryonField[Vel3Num][cellindex] = z_velocity;

   BaryonField[TENum][cellindex] = energy;
```

Set final field value

Adding a new test problem

Make the 2 new files (or copy from another test problem):

MyProblemInitialize.C

Grid_MyProblemInitializeGrid.C

Add the 2 new files to Make.config.objects (so they get compiled)

/src/enzo/Make.config.objects

Adding a new test problem

Make the 2 new files (or copy from another test problem):

MyProblemInitialize.C

Grid_MyProblemInitializeGrid.C

Add the 2 new files to Make.config.objects (so they get compiled)

/src/enzo/Make.config.objects

```
rotate2d.o \
rotate3d.o \
RotatingCylinderInitialize.o \
s66_st1.o \
s90_st1.o \
  .
  .
  .
Grid_RHIonizationClumpInitializeGrid.o \
Grid_RHIonizationSteepInitializeGrid.o \
Grid_RHIonizationTestInitializeGrid.o \
Grid_RotatingCylinderInitialize.o \
Grid_SedovBlastInitializeGrid.o \
Grid_SedovBlastInitializeGrid3D.o \
```

Adding a new test problem

Make the 2 new files (or copy from another test problem):

MyProblemInitialize.C

Grid_MyProblemInitializeGrid.C

Add the 2 new files to Make.config.objects (so they get compiled)

/src/enzo/Make.config.objects

Add definition of Grid initialization routine to Grid.h

/src/enzo/Grid.h

Adding a new test problem

Make the 2 new files (or copy from another test problem):

MyProblemInitialize.C

Grid_MyProblemInitializeGrid.C

Add the 2 new files to Make.config.objects (so they get compiled)

/src/enzo/Make.config.objects

Add definition of Grid initialization routine to Grid.h

/src/enzo/Grid.h

```
                                int RadiatingShockRandomSeedInitialize,
                                FLOAT RadiatingShockCenterPosition[MAX_DIMENSION]);

/* Initialize a grid for a rotating cylinder collapse */
int RotatingCylinderInitializeGrid(FLOAT RotatingCylinderRadius,
                                   FLOAT RotatingCylinderCenterPosition[MAX_DIMENSION],
                                   float RotatingCylinderLambda,
                                   float RotatingCylinderOverdensity);


int RotatingSphereInitializeGrid(FLOAT RotatingSphereRadius,
                                   FLOAT RotatingSphereCenterPosition[MAX_DIMENSION],
                                   float RotatingSphereLambda,
```

Adding a new test problem

Make the 2 new files (or copy from another test problem):

MyProblemInitialize.C

Grid_MyProblemInitializeGrid.C

Add the 2 new files to Make.config.objects (so they get compiled)

/src/enzo/Make.config.objects

Add definition of Grid initialization routine to Grid.h

/src/enzo/Grid.h

Add prototype and call for non-Grid initialization routine in InitializeNew (make sure it has a unique ProblemType #).

/src/enzo/InitializeNew.C

Adding a new test problem

Make the 2 new files (or copy from another test problem):

MyProblemInitialize.C

Grid_MyProblemInitializeGrid.C

Add the 2 new files to Make.config.objects (so they get compiled)

/src

```
int ShockInABoxInitialize(FILE *fptr, FILE *Outfptr, HierarchyEntry &TopGrid,
                          TopGridData &MetaData, ExternalBoundary &Exterior);
int ImplosionInitialize(FILE *fptr, FILE *Outfptr, HierarchyEntry &TopGrid,
                        TopGridData &MetaData);
int RotatingCylinderInitialize(FILE *fptr, FILE *Outfptr, HierarchyEntry &TopGrid,
                               TopGridData &MetaData);
int ConductionTestInitialize(FILE *fptr, FILE *Outfptr, HierarchyEntry &TopGrid,
```

Add defir Grid.h

/src

```
// 10) RotatingCylinder

if (ProblemType == 10)
  ret = RotatingCylinderInitialize(fptr, Outfptr, TopGrid, MetaData);
```

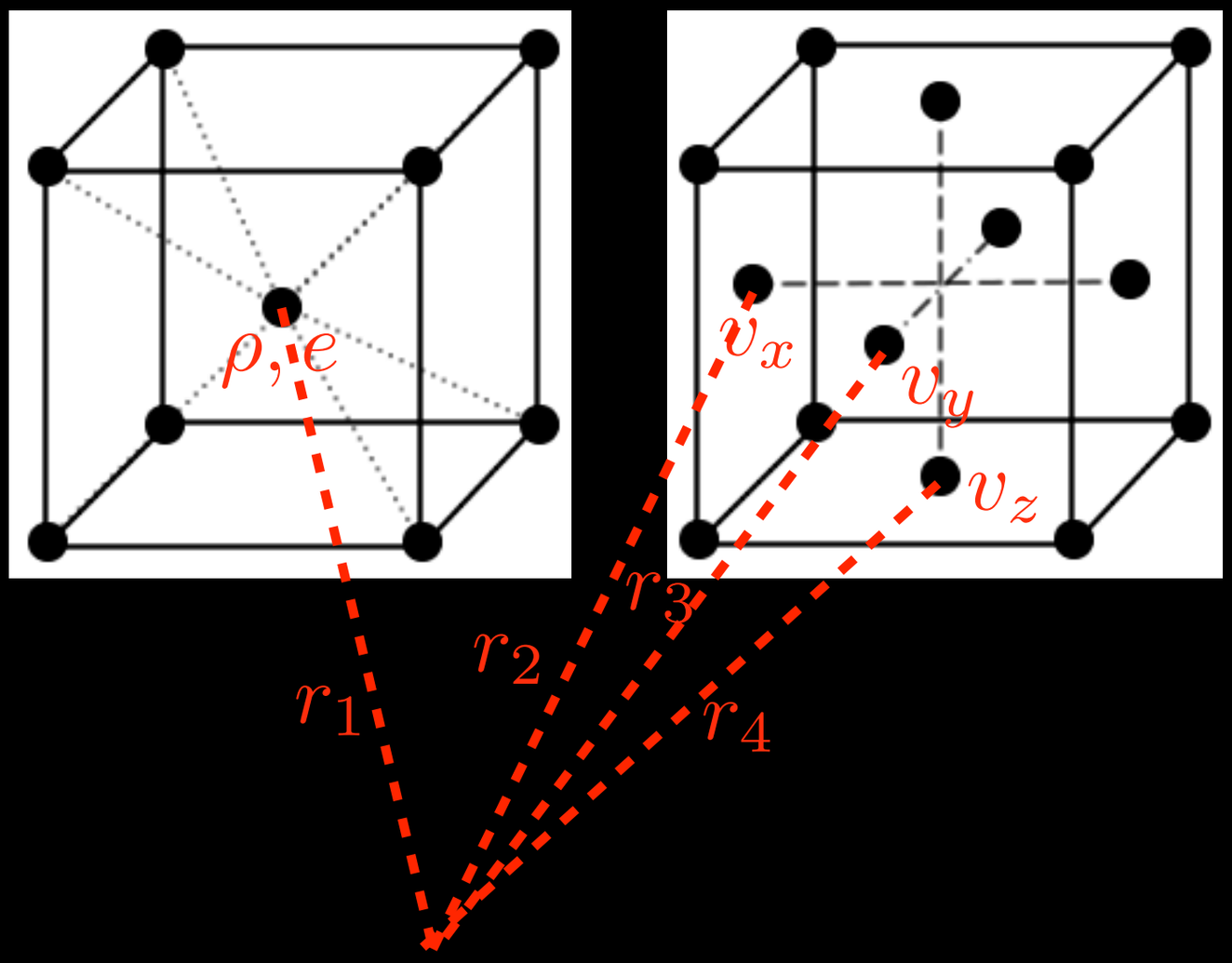
Add prototype and call for non-Grid initialization routine in InitializeNew (make sure it has a unique ProblemType #).

/src/enzo/InitializeNew.C

Points to watch

Using Zeus (HydroMethod = 2)

Zeus uses a face-centered velocity




```
/* Loop over dims if using Zeus (since vel's face-centered). */

for (dim = 0; dim < 1+(HydroMethod == Zeus_Hydro ? GridRank : 0);
     dim++) {

  /* Compute position. */

  xpos = x-DiskPosition[0] -
    (dim == 1 ? 0.5*CellWidth[0][0] : 0.0);
  ypos = y-DiskPosition[1] -
    (dim == 2 ? 0.5*CellWidth[1][0] : 0.0);
  zpos = z-DiskPosition[2] -
    (dim == 3 ? 0.5*CellWidth[2][0] : 0.0);



/* Compute velocty: L x r_perp. */

if (dim == 0 || dim == 1)
  Velocity[0] = DiskVelocityMag*(AngularMomentum[1]*xhat[2] -
                                 AngularMomentum[2]*xhat[1]);
if (dim == 0 || dim == 2)
  Velocity[1] = DiskVelocityMag*(AngularMomentum[2]*xhat[0] -
                                 AngularMomentum[0]*xhat[2]);
if (dim == 0 || dim == 3)
  Velocity[2] = DiskVelocityMag*(AngularMomentum[0]*xhat[1] -
                                 AngularMomentum[1]*xhat[0]);
```

It also uses internal energy, not total energy

Points to watch

Energy

BaryonField[TENum] is energy/mass

Particle Mass

ParticleMass[i] is particle mass / cell volume

GravitationalConstant

GravitationalConstant = 4 pi G

Must be in code units if SelfGravity = 1