A User's Guide to Radiation Units in Athena++

Avery Bailey

There are two hooks for defining a radiation unit system in Athena++. The first is through the use of `T_unit`, `density_unit`, `length_unit`, and `molecular_weight` in the radiation block of the input file. The second unit system is set by `prat` and `crat`. Either of these hooks may be selected by the input file variable `unit = 1` or `0` respectively with the latter being the default. Ultimately, either hook is valid and we will go over the best practices for usage, but the former is more user friendly and will be our starting point. In this document, quantities which are unitless and unit-full will be explicitly marked with a 'code' or 'cgs' subscript respectively. The fixed unit definitions which define a unit system and can be used to convert between dimensionless and dimensionful variables (e.g. $T_{\rm code} = T_{\rm cgs}/T_0$) will be given a subscript 0.

1 The Foolproof Method: `T_unit`, `density_unit`, `length_unit`, and `molecular_weight`

In this case one defines a temperature scale T_0, density scale ρ_0, length scale l_0, and mean molecular weight μ_0 which forms a system of units for the hydrodynamics. These should all be chosen to be close to characteristic scales of your problem to limit the influence of roundoff error – e.g. if you are simulating a disk a reasonable choice of length scale could be a disk scale height or a disk radius. These should be specified in cgs units in the `radiation` block of the input file with `unit = 1`, with the exception of μ_0 which is in amu. For example,

```
<radiation>
unit             = 1
T_unit           = 1.0e3    # Kelvin
density_unit     = 1.0e-8   # g/cm^3
length_unit      = 1.496e13 # cm
molecular_weight = 2.0      # atomic mass units
```

By themselves, the above variables do not close a system of hydrodynamic units (mass, length, time) as there is no formal time unit in the above code block. With the above, the radiation module adopts a convention that the velocity unit will be defined by the isothermal sound speed at the user specified temperature and molecular weight $v_0 \equiv \sqrt{\mathcal{R}T_0/\mu_0}$, so that the time unit is $t_0 \equiv l_0/v_0 = l_0(\mathcal{R}T_0/\mu_0)^{-1/2}$ (where $\mathcal{R} = 8.31 \times 10^7$ erg/mole/K is the ideal gas constant). This choice of velocity unit means that if one wants to run an

equivalent isothermal simulation, they should set `iso_sound_speed = 1.0` in the input file. This choice also means that in code units, the ideal gas law is simply $p_{\rm code} = \rho_{\rm code} T_{\rm code}$. To see this we write the dimensional ideal gas law $p_{\rm cgs} = \rho_{\rm cgs} \mathcal{R} T_{\rm cgs}/\mu$ and divide both sides by $p_0 \equiv \rho_0 l_0^2/t_0^2$ to get

$$\frac{p_{\rm cgs}}{p_0} = \left(\frac{\rho_{\rm cgs}}{\rho_0}\right) \frac{\mathcal{R} T_{\rm cgs}}{\mu} \left(\frac{t_0^2}{l_0^2}\right) \tag{1}$$

with constant $\mu = \mu_0$ and using $t_0 \equiv l_0 (\mathcal{R} T_0/\mu_0)^{-1/2}$, one gets

$$\frac{p_{\rm cgs}}{p_0} = \left(\frac{\rho_{\rm cgs}}{\rho_0}\right) \left(\frac{T_{\rm cgs}}{T_0}\right) \tag{2}$$

which is just $p_{\rm code} = \rho_{\rm code} T_{\rm code}$.

Finally, we remark on units of the radiation subsystem of the code. Fundamentally the code cares about one variable the specific intensity I or radiation energy per unit area per unit time per unit direction (per unit frequency, if using frequency dependent transfer). The code units of intensity are specifically chosen to be

$$I_{\rm code} = \frac{I_{\rm cgs}}{acT_0^4/(4\pi)}. \tag{3}$$

This has the convenient property that if one wants to define an intensity which is in radiative equilbrium with gas described by an LTE source function $B = acT^4/(4\pi)$, this is prescribed simply by setting $I_{\rm code} = T_{\rm code}^4$. In addition to intensity, there are three derived quantities or moments of intensity which the code will compute. These are mean radiation energy density E_r, radiative flux $\mathbf{F}_r$, and the radiation pressure tensor P_r. Normally these are defined as angular moments of intensity as follows

$$E_r = \frac{1}{c} \int I(\hat{\mathbf{n}}) d\Omega \tag{4}$$

$$\mathbf{F}_r = \int I(\hat{\mathbf{n}}) \hat{\mathbf{n}} d\Omega \tag{5}$$

$$\mathsf{P}_r = \frac{1}{c} \int I(\hat{\mathbf{n}}) \hat{\mathbf{n}}\hat{\mathbf{n}} d\Omega \tag{6}$$

The code however, normalizes angle integrals to 1 instead of 4π ($d\Omega_{\rm code} = d\Omega_{\rm cgs}/4\pi$, so $1 = \int d\Omega_{\rm code}$) and drops all factors of $1/c$ outside the integral. This makes the code definitions

$$E_{r,{\rm code}} = \int I(\hat{\mathbf{n}})_{\rm code} d\Omega_{\rm code} \tag{7}$$

$$\mathbf{F}_{r,{\rm code}} = \int I(\hat{\mathbf{n}})_{\rm code} \hat{\mathbf{n}} d\Omega_{\rm code} \tag{8}$$

$$\mathsf{P}_{r,{\rm code}} = \int I(\hat{\mathbf{n}})_{\rm code} \hat{\mathbf{n}}\hat{\mathbf{n}} d\Omega_{\rm code} \tag{9}$$

which can be combined with the definition $d\Omega_{\rm code}$ and equation 3, to yield the cgs conversion factors

$$E_{r,\rm code} = \int \frac{I(\hat{\mathbf{n}})_{\rm cgs}}{acT_0^4} d\Omega_{\rm cgs} = \frac{E_{r,\rm cgs}}{aT_0^4} \tag{10}$$

$$\mathbf{F}_{r,\rm code} = \int \frac{I(\hat{\mathbf{n}})_{\rm code}}{acT_0^4} \hat{\mathbf{n}} d\Omega_{\rm cgs} = \frac{\mathbf{F}_{r,\rm cgs}}{acT_0^4} \tag{11}$$

$$\mathsf{P}_{r,\rm code} = \int \frac{I(\hat{\mathbf{n}})_{\rm code}}{acT_0^4} \hat{\mathbf{n}}\hat{\mathbf{n}} d\Omega_{\rm cgs} = \frac{\mathsf{P}_{r,\rm cgs}}{aT_0^4} \tag{12}$$

The last radiation quantity is the opacity κ. The opacity the code uses is not the usual opacity with units of g/cm^2, rather it is the product of opacity and density $\sigma \equiv \kappa\rho$, this is a cross-section per volume or an inverse mean-free path and has units of (length)$^{-1}$. Confusingly both σ and κ are often referred to as 'opacities'. The opacity with units of inverse length is defined in code units just in terms of the user-supplied l_0, $\rightarrow \sigma_{\rm code} = \sigma_{\rm cgs} l_0$. Note that if you have opacity $\kappa_{\rm cgs}$, the appropriate way to enroll it is $\sigma_{\rm code} = \rho_{\rm code}(\kappa_{\rm cgs}\rho_0 l_0) \neq \rho_0(\kappa_{\rm cgs}\rho_0 l_0)$, as $\rho_{\rm code}$ is often a function of position and ρ_0 is by definition a constant.

Below is a summary of all conversion factors one needs to know for this setup. To convert from code to cgs and vice-versa simply divide by the appropriate factor ($p_{\rm code} = p_{\rm cgs}/p_0$).

Table 1: Unit conversions in terms of fundamental parameters $(\rho_0, T_0, l_0, \mu_0)$

unit	symbol	conversion factor
density	ρ_0	`density_unit`
temperature	T_0	`T_unit`
length	l_0	`length_unit`
molecular weight	μ_0	`molecular_weight`
velocity	v_0	$\sqrt{\mathcal{R}T_0/\mu_0}$
time	t_0	$l_0/\sqrt{\mathcal{R}T_0/\mu_0}$
pressure	p_0	$\rho_0\mathcal{R}T_0/\mu_0$
intensity	I_0	$acT_0^4/(4\pi)$
radiation energy density	$E_{r,0}$	aT_0^4
radiative flux	$F_{r,0}$	acT_0^4
radiation pressure	$p_{r,0}$	aT_0^4
opacity	σ_0	$(l_0)^{-1}$

1.1 Best Practices

1. **Quantities given in the input file should be supplied in cgs units.**

   You can always give preferred units or dimensionless equivalents as comments next to the cgs values e.g.

```
<problem>
softening = 7.1492e9 # cm = 1 Jupiter radius
opacity   = 0.01     # cm^2/g = 100 dimensionless opacity
```

The exception is mesh quantities (like `mesh/x1min`) which, since they go directly to the mesh constructor, need to be in code units.

2. **Convert dimensional quantities to dimensionless quantities in the `InitUserMeshData` and `InitUserMeshBlockData` functions.**

   Having cgs variables floating around that need to be converted to dimensionless values every time they are used is cumbersome. It is best to read in the cgs values and immediately convert them to dimensionless. This also compartmentalizes conversions to a block of code so they are immediately readable and retrievable. This can be done for fundamental constants as well, as is done for the gravitational constant in the following.

```
static Real T0;
static Real H0;
static Real rho0;
static Real mol_weight;
static Real t0;
static Real heatrate; // ergs/g/s
static Real opacity;  // cm^2/g
static Real constG = 6.6743e-8;

void Mesh::InitUserMeshData(ParameterInput *pin)
{
  T0 = pin->GetReal("radiation", "T_unit");
  H0 = pin->GetReal("radiation", "length_unit");
  rho0 = pin->GetReal("radiation", "density_unit");
  mol_weight = pin->GetReal("radiation", "molecular_weight");
  Real r_ideal = 8.314462618e7/mol_weight;
  t0 = H0/std::sqrt(r_ideal*T0);
  heatrate = pin->GetOrAddReal("problem", "heatrate", 0.0);
  opacity = pin->GetOrAddReal("problem", "opacity", 0.0);
  // make quantities dimensionless
  heatrate *= t0*t0*t0/H0/H0;
  opacity *= rho0*H0;
  constG *= rho0*t0*t0;
  return;
}
```