COLT

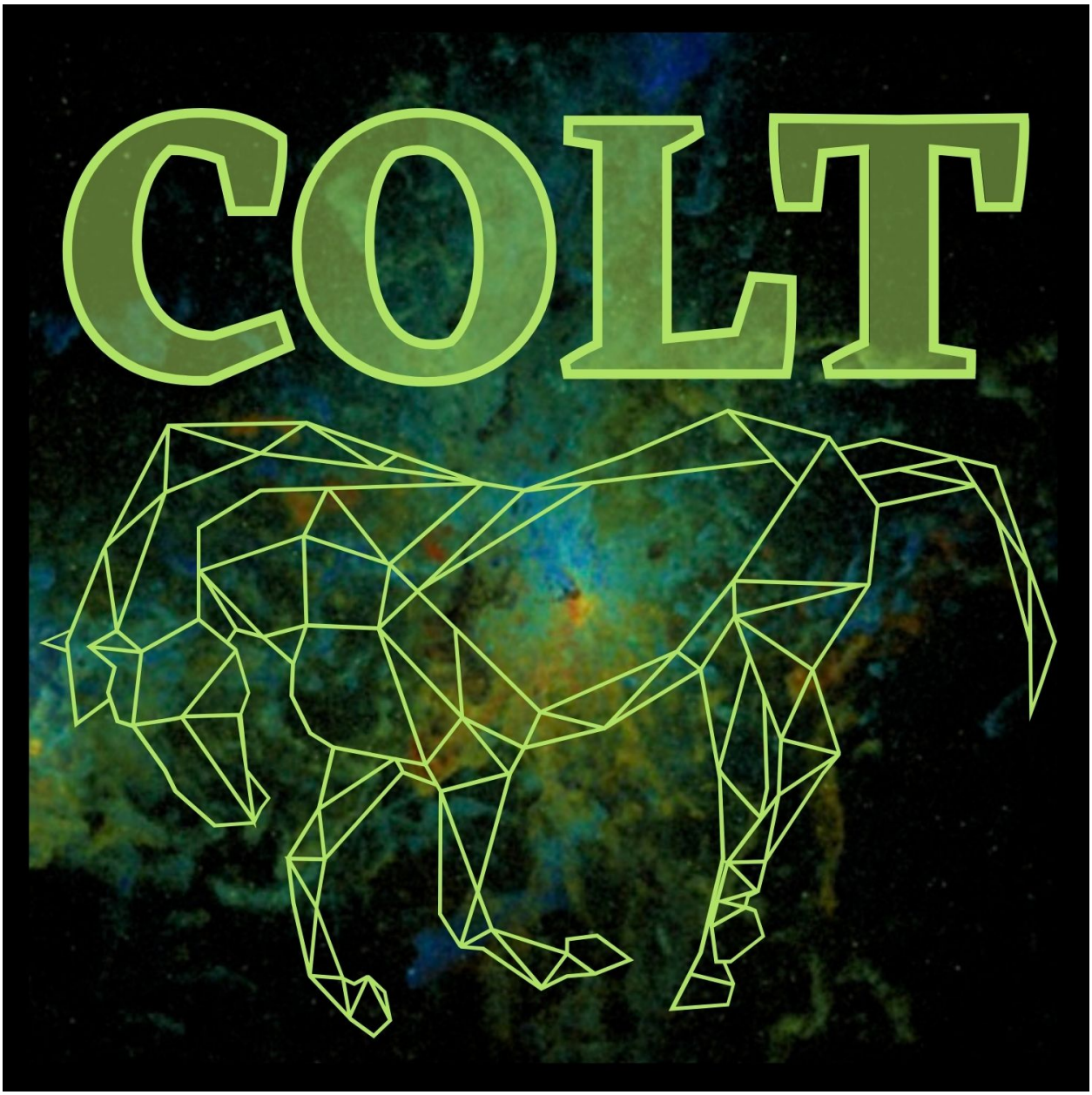
Release 1.0

Aaron Smith

May 09, 2024

GENERAL USAGE

The Cosmic Lyman-alpha Transfer (COLT) code is a Monte Carlo radiative transfer (MCRT) solver for post-processing hydrodynamical simulations on arbitrary grids. These include a plane parallel slabs, spherical geometry, 3D Cartesian grids, adaptive resolution octrees, unstructured Voronoi tessellations, and secondary outputs. COLT also includes several visualization and analysis tools that exploit the underlying ray-tracing algorithms or otherwise benefit from an efficient hybrid MPI + OpenMP parallelization strategy within a flexible C++ framework.



CHAPTER

ONE

SUPPORT

If you have any issues obtaining, compiling, or running the code, please consult the below documentation, or let us know by email (Aaron Smith, asmith@utdallas.edu). We are also open to collaboration, which is particularly encouraged when using or implementing new features.

1.1 Getting Started

This page explains the basic features of COLT from installation to running simulations.

1.1.1 Compilation

Obtain the COLT source code by cloning the repository:

```
$ git clone git@bitbucket.org:aaron_smith/colt.git
```

If you do not have a bitbucket account set up with ssh keys then you can still sync via https:

```
$ git clone https://aaron_smith@bitbucket.org/aaron_smith/colt.git
```

This creates a new folder named `colt` containing the source code (`source`), documentation (`docs`), and compilation files (e.g. `Makefile`). After installing the required libraries (see *Dependencies*) and setting up the build environment (see *Makefile*) the executable may be compiled with:

```
$ cd colt
$ make -j
```

1.1.2 Dependencies

Note that there are required dependencies:

- **MPI:** Message Passing Interface library for standard code parallelization
- **HDF5:** Hierarchical Data Format library for reading and writing files (with C++ wrappers)
- **yaml-cpp:** Library for parsing YAML config files (github repository)

There is also an additional dependency for Voronoi geometry:

- **CGAL:** Computational Geometry Algorithms Library for constructing Delaunay triangulations

1.1.3 Instructions for MacOS

On MacOS all of these dependencies can be conveniently installed with the Homebrew package manager with the following command:

```
$ brew install openmpi libomp hdf5 yaml-cpp cgal
```

Adjustments for Apple Silicon: The default Apple Silicon compiler does not support OpenMP. As a workaround replace `openmpi` with `mpich` and then rebuild with the following command:

```
$ brew reinstall --build-from-source mpich
```

This may first require resolving any package conflicts by unlinking or uninstalling as suggested by Homebrew:

```
$ brew unlink conflicting-package
$ brew uninstall conflicting-package
$ brew install mpich
```

After completing these steps, relaunch the terminal to proceed with the COLT compilation using the command: `make -j`. If this fails with an error about `libomp` then you may need to either manually link the OpenMP library, e.g. `ln -s /opt/homebrew/opt/libomp/lib/libomp.dylib /opt/homebrew/lib/libomp.dylib`, or set the `LD_LIBRARY_PATH` environment variable, e.g. `export LD_LIBRARY_PATH=/opt/homebrew/opt/libomp/lib:$LD_LIBRARY_PATH`. Please send a message to the COLT developers if you encounter any other issues.

1.1.4 Instructions for Linux Clusters

On shared Linux clusters and supercomputers some dependencies may be available system wide. Use `module avail` or `module spider` to find the correct module names. It is convenient to add a module file `~/.colt` and load these with `. ~/.colt` before compiling or running:

```
#!/bin/bash
module purge; module load openmpi hdf5 yaml-cpp cgal
```

In many cases you may still need to install a dependency locally. For example, to install yaml-cpp in your home directory:

```
$ module load cmake  # After loading the appropriate gcc/mpi modules
$ git clone https://github.com/jbeder/yaml-cpp.git
$ cd yaml-cpp; mkdir build; cd build  # Build in a separate directory
$ cmake -DYAML_BUILD_SHARED_LIBS=ON -DCMAKE_INSTALL_PREFIX=${HOME} ..
$ make -j; make install  # Installs to ${HOME}/lib[64] and ${HOME}/include
```

1.1.5 Makefile

The provided `Makefile` allows a flexible build based on the specific computing environment. Currently, COLT is designed to have minimal compile time options summarized in the provided `template-defines.yaml` file:

```
MACHINE: gcc       # Makefile build system
GEOMETRY: voronoi  # Geometry type: slab, spherical, cartesian, octree, voronoi
HAVE_CGAL: true    # Include the CGAL library (requires voronoi geometry)
```

These options can either be provided in the `Makefile` itself or in a yaml (or bash) file with a default location of `defines.yaml` (see note below). The main option is the `MACHINE` with known systems: *homebrew*, *gcc*, *pleiades*,

comet, *odyssey*, *stampede*, and *supermuc*. Other systems may require slight modification based on the existing examples in the `Makefile`. The other main option is the `GEOMETRY` with known types: *slab*, *spherical*, *cartesian*, *octree*, *voronoi*. More information is given in the initial conditions specifications.

Note: It is usually most convenient to perform a default in-place build, which generates the compiled object files and executable in the same directory. However, to allow the flexibility of out-of-place builds you may customize the `DEFS`, `BUILD`, and `EXE` paths, e.g. to run multiple geometries with the same source distribution.

$ make -j -C /source/path DEFS={path}/defines.yaml BUILD={path}/build EXE={path}/colt

---

1.1.6 Running a Simulation

After compilation, the executable `colt` can be run by specifying a YAML configuration file, usually called `config.yaml`. In case you are streamlining analysis over several snapshots with the same config file you can also specify the snapshot number. It is also importantant to specify the number of OpenMP threads before execution. A typical command to run COLT with 4 tasks and 16 threads looks like the following:

```
$ export OMP_NUM_THREADS=16
$ mpirun -n 4 ./colt config.yaml [snapshot]
```

Note: Please consult the system documentation about running hybrid MPI + OpenMP jobs as some environments may require special commands, e.g. a linux cluster might need `--bind-to none` to release the core binding. Supercomputers typically supply job scripts and specific advice for hybrid execution within their user guide documentation.

1.2 Simulation Parameters

This page explains the options for configuring COLT simulations.

1.2.1 Configuration Files

COLT uses the **yaml-cpp** library to allow for flexible and convenient configuration of run-time parameters. We introduce the `config.yaml` file with the following example:

```yaml
# COLT config file
--- !mcrt                       # Monte Carlo radiative transfer module

init_dir: ics                   # Initial conditions directory (default: "ics")
init_base: colt                 # Initial conditions base name (optional)
output_dir: output              # Output directory name (default: "output")
output_base: Lya                # Output file base name (default: "colt")

cameras:                        # Add camera directions manually
  - [1,0,0]
  - [0,1,0]
  - [0,0,1]

... Additional run-time parameters
```

The format is essentially a human readable python dictionary of `key: value` pairs that allows #-style comments and complex specifications such as the above list of camera directions.

The YAML header `--- !mcrt` specifies which module to run, a convention that sets the module apart from more typical run-time parameters. The value is interpreted in the main file prior to instantiating the simulation to allow high level class polymorphism. By default the MCRT module is assumed in cases where the header line is not included.

Note: COLT configuration is design to be minimally verbose, transparent, and catch common unintended errors. In practice this means that only expected parameters are allowed and an error is raised for unexpected or repeated parameters. If a parameter is not specified then the default value will be assigned. Note that the default may depend on other parameters but this is determined by the code logic, i.e. the order of parameters does not matter. Only a few parameters are actually required, such as the initial conditions file naming scheme.

1.2.2 General Parameters

The following parameters are shared by most modules:

- `verbose` – Verbose output for tests and debugging. [Default value: false]
- `init_file` – Initial conditions file name. [Required if `init_base` is not specified]
- `init_dir` – Initial conditions directory. [Optional]
- `init_subdir` – Initial conditions subdirectory, e.g. allows `ics/snap_000` organization. [Optional]
- `init_base` – Initial conditions file base. [Optional]
- `init_ext` – Initial conditions file extension. [Default value: hdf5]
- `output_dir` – Output directory. [Default value: output]
- `output_subdir` – Output subdirectory, e.g. allows `output/snap_000` organization. [Optional]
- `output_base` – Output file base. [Default value: colt]
- `output_ext` – Output file extension. [Default value: hdf5]
- `snap_padding` – Leading zeros for snapshot number formatting. [Default value: 3]
- `select_subhalo` – Optional toggle for the command line subhalo/group specifier. If true then this is interpreted as a subhalo index, otherwise it is interpreted as a group index.

Note: In practice, an initial conditions file may be specified by the full name via `init_file` or the file pattern `<init_dir>/<init_base>_<snapshot>.<init_ext>`, which is then mirrored in the convention for output files, e.g. `output/colt_000.hdf5`.

1.2.3 Cosmological Parameters

The following parameters are related to the cosmology:

- `cosmological` – Flag for cosmological simulations. Note: This is for cosmological vs. intrinsic observers, i.e. it changes the output units but not the physics. [Default value: false]
- `Omega0` – Matter density in units of the current critical density $\rho_{\mathrm{crit},0}$. [Default value: 0.3111]
- `OmegaB` – Baryon density in units of the current critical density $\rho_{\mathrm{crit},0}$. [Default value: 0.04897]
- `h100` – Hubble constant in units of 100 km/s/Mpc. [Default value: 0.6766]

Note: Cosmological parameters are needed for calculating observed distances and other module specific quantities. The values are from the Planck mission (2018 results). For consistency, these parameters may also be read from the initial conditions file.

1.2.4 Gas Parameters

The following parameters are related to the gas environment:

- `constant_temperature` – Enforce a constant temperature, T. Note: This overrides automatically reading from the input file. [Optional | Units: K]
- `T_floor` – Apply a temperature floor to all gas cells, $T_{\rm floor}$. [Optional | Units: K]
- `T_rtol` – Relative tolerance for temperature changes, e.g. in photoionization equilibrium calculations. [Default value: 0.1]
- `electron_fraction` – Enforce a constant electron fraction, $x_e = n_e/n_{\rm H}$. [Optional]
- `HI_fraction` or `neutral_fraction` – Enforce a constant neutral fraction, $x_{\rm HI} = n_{\rm HI}/n_{\rm H}$. Note: In general, specifying an ionization state overrides automatically reading from the input file. If neither is present, we assume neutral gas compositions. [Optional]
- `HII_fraction` or `ionized_fraction` – Enforce a constant ionized fraction, $x_{\rm HII} = n_{\rm HII}/n_{\rm H}$. [Optional]
- `H2_fraction` or `molecular_fraction` – Enforce a constant molecular hydrogen fraction, $x_{\rm H2} = n_{\rm H2}/n_{\rm H}$. [Optional]
- `HeI_fraction` – Enforce a constant HeI fraction, $x_{\rm HeI} = n_{\rm HeI}/n_{\rm He}$. [Optional]
- `HeII_fraction` – Enforce a constant HeII fraction, $x_{\rm HeII} = n_{\rm HeII}/n_{\rm He}$. [Optional]
- `HeIII_fraction` – Enforce a constant HeIII fraction, $x_{\rm HeIII} = n_{\rm HeIII}/n_{\rm He}$. [Optional]
- `CI_fraction` – Enforce a constant CI fraction, $x_{\rm CI} = n_{\rm CI}/n_{\rm C}$. [Optional] Note: This is also available for `CII`, `CIII`, `CIV`, `CV`, `CVI`, and `CVII`.
- `NI_fraction` – Enforce a constant NI fraction, $x_{\rm NI} = n_{\rm NI}/n_{\rm N}$. [Optional] Note: This is also available for `NII`, `NIII`, `NIV`, `NV`, `NVI`, `NVII`, and `NVIII`.
- `OI_fraction` – Enforce a constant OI fraction, $x_{\rm OI} = n_{\rm OI}/n_{\rm O}$. [Optional] Note: This is also available for `OII`, `OIII`, `OIV`, `OV`, `OVI`, `OVII`, `OVIII`, and `OIX`.
- `NeI_fraction` – Enforce a constant NeI fraction, $x_{\rm NeI} = n_{\rm NeI}/n_{\rm Ne}$. [Optional] Note: This is also available for `NeII`, `NeIII`, `NeIV`, `NeV`, `NeVI`, `NeVII`, `NeVIII`, and `NeIX`.
- `MgI_fraction` – Enforce a constant MgI fraction, $x_{\rm MgI} = n_{\rm MgI}/n_{\rm Mg}$. [Optional] Note: This is also available for `MgII`, `MgIII`, `MgIV`, `MgV`, `MgVI`, `MgVII`, `MgVIII`, `MgIX`, `MgX`, and `MgXI`.
- `SiI_fraction` – Enforce a constant SiI fraction, $x_{\rm SiI} = n_{\rm SiI}/n_{\rm Si}$. [Optional] Note: This is also available for `SiII`, `SiIII`, `SiIV`, `SiV`, `SiVI`, `SiVII`, `SiVIII`, `SiIX`, `SiX`, `SiXI`, `SiXII`, and `SiXIII`.
- `SI_fraction` – Enforce a constant SI fraction, $x_{\rm SI} = n_{\rm SI}/n_{\rm S}$. [Optional] Note: This is also available for `SII`, `SIII`, `SIV`, `SV`, `SVI`, `SVII`, `SVIII`, `SIX`, `SX`, `SXI`, `SXII`, `SXIII`, `SXIV`, and `SXV`.
- `FeI_fraction` – Enforce a constant FeI fraction, $x_{\rm FeI} = n_{\rm FeI}/n_{\rm Fe}$. [Optional] Note: This is also available for `FeII`, `FeIII`, `FeIV`, `FeV`, `FeVI`, `FeVII`, `FeVIII`, `FeIX`, `FeX`, `FeXI`, `FeXII`, `FeXIII`, `FeXIV`, `FeXV`, `FeXVI`, and `FeXVII`.
- `hydrogen_fraction` – Enforce a constant hydrogen mass fraction. Overrides automatically reading X from the input file. If neither is present, we assume the primordial value of $X = 0.76$. [Optional | Units: Mass fraction]

- `helium_fraction` – Enforce a constant helium mass fraction. Overrides automatically reading Y from the input file. If neither is present, we assume the primordial value of $Y = 1 - X$. [Optional | Units: Mass fraction]
- `metallicity` – Enforce a constant metallicity. Overrides automatically reading Z from the input file. If neither is present, we assume solar metallicity. [Optional | Units: Mass fraction]
- `carbon_metallicity` – Enforce a constant carbon metallicity, $Z_{\rm C}$. Note: In general, specifying an element specific metallicity overrides automatically reading from the input file. If neither is present, we assume solar scaled abundances, e.g. $Z_{\rm C} = Z_{\odot,{\rm C}}(Z/Z_\odot)$. [Optional | Units: Mass fraction]
- `nitrogen_metallicity` – Enforce a constant nitrogen metallicity, $Z_{\rm N}$. [Optional | Units: Mass fraction]
- `oxygen_metallicity` – Enforce a constant oxygen metallicity, $Z_{\rm O}$. [Optional | Units: Mass fraction]
- `neon_metallicity` – Enforce a constant neon metallicity, $Z_{\rm Ne}$. [Optional | Units: Mass fraction]
- `magnesium_metallicity` – Enforce a constant magnesium metallicity, $Z_{\rm Mg}$. [Optional | Units: Mass fraction]
- `silicon_metallicity` – Enforce a constant silicon metallicity, $Z_{\rm Si}$. [Optional | Units: Mass fraction]
- `sulfer_metallicity` – Enforce a constant sulfer metallicity, $Z_{\rm S}$. [Optional | Units: Mass fraction]
- `iron_metallicity` – Enforce a constant iron metallicity, $Z_{\rm Fe}$. [Optional | Units: Mass fraction]
- `mass_weighted_metallicities` – Adopt mass-weighted average metallicity values. [Default value: false]
- `dust_to_metal` – Enforce a constant dust-to-metal ratio, DTM $\equiv \rho_{\rm dust}/\rho_Z$. [Optional | Units: Mass fraction]
- `dust_to_gas` – Enforce a constant dust-to-gas ratio, $\mathcal{D} \equiv \rho_{\rm dust}/\rho$. [Optional | Units: Mass fraction]
- `dust_boost` – Rescale the dust values by this boost factor. [Optional]
- `v_turb` – Effective microturbulent velocity. [Default value: 0 | Units: cm/s]
- `v_turb_kms` – Effective microturbulent velocity. [Default value: 0 | Units: km/s]
- `T_turb` – Effective microturbulent temperature. [Default value: 0 | Units: K]
- `scaled_microturb` – Scaled microturbulence model where high density gas is more turbulent. Specifically, $T_{\rm turb} = {\rm K\,km^2}\rho/(\gamma k_{\rm B})$ with an adiabatic index of $\gamma = 5/3$. [Default value: false]

Note: The microturbulence parameter follows the standard treatment of adding Gaussian line broadening in quadrature: $b = \sqrt{v_{\rm th}^2 + v_{\rm turb}^2}$. This acts as an effective temperature floor for line radiative transfer with $v_{\rm turb}^2 = 2k_{\rm B}T_{\rm turb}/m_{\rm carrier}$.

1.2.5 Camera Parameters

The following parameters are related to controlling cameras:

- `camera` – Add a single camera direction manually in *[1 0 0]* format. [Optional]
- `cameras` – Add a list of camera directions manually in *[1 0 0]* format. [Optional]
- `n_exp` – Healpix exponent for camera directions, resulting in $12 \times 4^{n_{\rm exp}}$ directions in RING ordering. [Optional | Valid range: n_exp >= 0]
- `n_side` – Healpix number of base pixel subdivisions for camera directions, resulting in $12 \times n_{\rm side}^2$ directions in RING ordering. [Overrides n_exp | Optional | Valid range: n_side >= 1]
- `n_rot` – Number of rotation camera directions, which are evenly spaced in angle around a common axis, e.g. to make movies. [Optional | Valid range: n_rot > 0]
- `phi_start` – Starting azimuthal angle for rotations. [Requires n_rot | Units: degrees]

- phi_end – Ending azimuthal angle for rotations. [Requires n_rot | Units: degrees]
- rotate_with_snaps – Flag to offset rotations by the snapshot number. [Requires n_rot | Default value: false]
- rotation_axis – Axis for rotation camera directions in *[0 0 1]* format. [Requires n_rot | Default value: (0,0,1)]
- camera_center – Camera target position in *[0 0 0]* format. [Default value: (0,0,0) | Units: cm]
- camera_motion – Camera target velocity in *[0 0 0]* format. [Default value: (0,0,0) | Units: cm/s]
- focus_cameras_on_emission – Focus the camera target position on the center of luminosity. [Default value: false]
- shift_cameras_on_emission – Shift the camera target velocity on the center of luminosity. [Default value: false]
- align_cameras_on_emission – Rotate the camera target direction to align camera_north with the angular momentum vector based on the center of luminosity. [Default value: false]
- camera_north – Camera north orientation in *[0 0 1]* format. [Default value: (0,0,1)]
- image_width – Image width defining a square aperture. Note: Similar setup for slit_width and cube_width. [Units: cm]
- image_widths – Image (x, y) widths defining a rectangular aperture. Note: Similar setup for cube_widths. [Units: cm]
- image_radius – Image radius or half of the image width. Note: Similar setup for slit_radius and cube_radius. [Units: cm]
- image_radii – Image (x, y) radii or half of the image widths. Note: Similar setup for cube_radii. [Units: cm]
- image_radius_bbox – Image radius or half of the image width. Note: Similar setup for slit_radius_bbox and cube_radius_bbox. [Units: bounding box]
- image_radii_bbox – Image (x, y) radii or half of the image widths. Note: Similar setup for cube_radii_bbox. [Units: bounding box]
- n_pixels – Number of (x, y) image pixels. Note: Similar setup for n_cube_pixels. [Default value: 100]
- nx_pixels – Number of x image pixels, overriding n_pixels if both are specified. Note: Similar setup for nx_cube_pixels. [Default value: 100]
- ny_pixels – Number of y image pixels, overriding n_pixels if both are specified. Note: Similar setup for ny_cube_pixels. [Default value: 100]
- n_slit_pixels – Number of slit pixels in the spatially-resolved direction. [Default value: 100]
- pixel_width – Intrinsic width of each image pixel. Note: Similar setup for slit_pixel_width and cube_pixel_width. [Units: cm]
- pixel_widths – Intrinsic (x, y) widths of each image pixel. Note: Similar setup for cube_pixel_widths. [Units: cm]
- pixel_arcsec – Observed angular width of each image pixel. Note: Similar setup for slit_pixel_arcsec and cube_pixel_arcsec. [Units: arcseconds]
- pixel_arcsecs – Observed angular (x, y) widths of each image pixel. Note: Similar setup for cube_pixel_arcsecs. [Units: arcseconds]
- pixel_arcsec2 – Observed angular area of each image pixel. Note: Similar setup for cube_pixel_arcsec2. [Units: arcseconds^2]
- slit_aperture – Slit aperture width. [Units: cm]

- slit_aperture_arcsec – Observed angular width of the slit aperture. [Units: arcseconds]
- radial_image_radius – Radial image radius. Note: Similar setup for radial_cube_radius. [Units: cm]
- radial_image_radius_bbox – Radial image radius. Note: Similar setup for radial_cube_radius_bbox. [Units: bounding box]
- n_radial_pixels – Number of radial image pixels. Note: Similar setup for n_radial_cube_pixels. [Default value: 100]
- radial_pixel_width – Intrinsic width of each radial image pixel. Note: Similar setup for radial_cube_pixel_width. [Units: cm]
- radial_pixel_arcsec – Observed angular width of each radial image pixel. Note: Similar setup for radial_cube_pixel_arcsec. [Units: arcseconds]

Note: Some combinations of camera parameters are redundant but the configuration detects inconsistencies and gives sensible error messages to address these. If left unspecified, the cameras are set up to capture the entire domain as defined by the simulation bounding box.

1.2.6 Escape Parameters

The following parameters are related to controlling the conditions for escape:

- spherical_escape – Flag to restrict ray-tracing a spherical region. [Default value: false]
- escape_center – Center of the escape region in *[0 0 0]* format. [Default value: (0,0,0) | Units: cm]
- escape_radius – Radius for the spherical escape region. [Units: cm]
- escape_radius_bbox – Radius for the spherical escape region. [Units: bounding box]
- emission_radius – Radius for the spherical emission region. [Units: cm]
- emission_radius_bbox – Radius for the spherical emission region. [Units: bounding box]

1.2.7 Additional File Parameters

The following parameters are related to specific additional I/O files:

- save_connections – Save the Voronoi connections to a file. [Default value: true]
- save_circulators – Save the Voronoi circulators to a file. [Default value: true]
- avoid_edges – Avoid ray-tracing through edge cells. [Default value: true]
- inner_edges – Flag to also treat inner neighbors of edge cells as edges. [Default value: false]
- output_inner_edges – Flag to output inner edge cells (for tracking purposes). [Default value: true]
- cgal_file – CGAL connectivity file (optional). [Default value: init_file]
- cgal_dir – CGAL connectivity directory (optional). [Default value: init_dir]
- cgal_subdir – CGAL connectivity subdirectory, e.g. allows ics/snap_000 organization. [Default value: cgal_dir]
- cgal_base – CGAL connectivity file base (optional). [Default value: init_base]
- cgal_ext – CGAL connectivity file extension (optional). [Default value: init_ext]

Note: The CGAL connectivity is usually saved to the original initial conditions file for future use. Here we allow the option of either not saving the connections or writing them to a different file.

- `abundances_file` – Abundances initial conditions file (optional). [Default value: `init_file`]
- `abundances_dir` – Abundances initial conditions directory (optional). [Default value: `init_dir`]
- `abundances_subdir` – Abundances initial conditions subdirectory, e.g. allows `ics/snap_000` organization. [Default value: `abundances_dir`]
- `abundances_base` – Abundances initial conditions file base (optional). [Default value: `init_base`]
- `abundances_ext` – Abundances initial conditions file extension (optional). [Default value: `init_ext`]
- `abundances_output_file` – Abundances output file (optional). [Default value: `abundances_file`]
- `abundances_output_dir` – Abundances output directory (optional). [Default value: `abundances_dir`]
- `abundances_output_subdir` – Abundances output subdirectory, e.g. allows `ics/snap_000` organization. [Default value: `abundances_output_dir`]
- `abundances_output_base` – Abundances output file base (optional). [Default value: `abundances_base`]
- `abundances_output_ext` – Abundances output file extension (optional). [Default value: `abundances_ext`]

Note: The abundances are usually read/saved to the original initial conditions file for future use. Here we allow the option of reading/writing them to a different file.

1.2.8 Advanced Parameters

The following parameters are related to advanced options:

- `set_density_from_mass` – Set the density by reading the mass and dividing by volume (`m` expecting units of `g`). Note: This can be useful for applying density-like analyses methods, e.g. ray-based projections, to point-like data types such as SPH data, star particles, or dark matter. [Default value: false]
- `read_density_as_mass` – Similar to `set_density_from_mass` but reads the density as a mass-like field before dividing by volume (`rho` expecting units of `g/cm^3`). [Default value: false]
- `use_internal_energy` – Set the temperature by reading the (mass) specific internal energy (`e_int` expecting units of `cm^2/s^2`). Note: When activated this overrides reading the temperature directly. When the temperature is not present in the initial conditions file and `constant_temperature` is not set, the default for this flag will update to true. [Default value: false]

1.3 Input and Output Files

COLT uses the Hierarchical Data Format (HDF5) library for reading and writing files (with C++ wrappers). This page explains the general structure and philosophy of initial conditions and simulation output files.

1.3.1 Supported Geometries

- Plane Parallel Slab (`slab`)
- Spherical Shells (`spherical`)
- 3D Cartesian grid (`cartesian`)
- Adaptive Resolution Octree (`octree`)
- Unstructured Voronoi Mesh (`voronoi`)
- Secondary Outputs (module specific)

1.3.2 Initial Conditions Files

The following example code outlines the general procedure for writing a COLT initial conditions file. Specific use cases can vary significantly, and many datasets are optional or can be overridden by configuration options. The presentation shows functional pseudo code with if statements to clarify geometry or module specific options.

```
# Write colt file (generally uses cgs units)
filename = f'{colt_dir}/colt_{snap:03d}.hdf5'
with h5py.File(filename, 'w') as f:
    # Simulation properties
    if cosmological:
        f.attrs['redshift'] = z     # Current simulation redshift
        f.attrs['Omega0'] = Omega0  # Matter density [rho_crit_0]
        f.attrs['OmegaB'] = OmegaB  # Baryon density [rho_crit_0]
        f.attrs['h100'] = h         # Hubble constant [100 km/s/Mpc]
    else:
        f.attrs['time'] = time      # Simulation time [s]

    # Geometry properties
    # Note: Due to camera positioning, it is most convenient to center the box at [0,0,0]
    f.attrs['n_cells'] = np.int32(n_cells) # Total number of cells (optional)
    if geometry == 'cartesian':
        f.attrs['nx'] = np.int32(nx) # Number of x cells
        f.attrs['ny'] = np.int32(ny) # Number of y cells
        f.attrs['nz'] = np.int32(nz) # Number of z cells
        R = ...                      # Box radius [cm]
        f.attrs['r_box'] = R         # For irregular boxes use bbox instead
        # bbox = np.array([[-R,-R,-R],[R,R,R]], dtype=np.float64)
        # f.create_dataset('bbox', data=bbox) # Bounding box [cm]
        # f['bbox'].attrs['units'] = b'cm'
    elif geometry == 'octree':
        r = ...                      # Cell left corner positions [cm]
        w = ...                      # Cell widths [cm]
        f.create_dataset('r', data=r, dtype=np.float64)
        f['r'].attrs['units'] = b'cm'
        f.create_dataset('w', data=w, dtype=np.float64)
        f['w'].attrs['units'] = b'cm'
        parent = ...                 # Cell parent indices
        child = ...                  # Cell child indices
        child_check = ...            # True if cell has children
        f.create_dataset('parent', data=parent, dtype=np.int32)
```

(continues on next page)

(continued from previous page)

```
        f.create_dataset('child', data=child, dtype=np.int32)
        f.create_dataset('child_check', data=child_check, dtype=np.int32)
    elif geometry == 'slab':
        r_edges = ...               # Cell edge positions (1D) [cm]
        f.create_dataset('r_edges', data=r_edges, dtype=np.float64)
        f['r_edges'].attrs['units'] = b'cm'
    elif geometry == 'spherical':
        r_edges = ...               # Shell edge positions (1D) [cm]
        f.create_dataset('r_edges', data=r_edges, dtype=np.float64)
        f['r_edges'].attrs['units'] = b'cm'
    elif geometry == 'voronoi':
        R = ...                     # Box radius [cm]
        f.attrs['r_box'] = R        # For irregular boxes use bbox instead
        # bbox = np.array([[-R,-R,-R],[R,R,R]], dtype=np.float64)
        # f.create_dataset('bbox', data=bbox) # Bounding box [cm]
        # f['bbox'].attrs['units'] = b'cm'
        # If neither is specified then bbox is inferred from particle positions
        r = ...                     # Mesh generating points [cm]
        f.create_dataset('r', data=r, dtype=np.float64)
        f['r'].attrs['units'] = b'cm'

    # Gas properties
    v = ...                         # Velocities [cm/s] (optional)
    f.create_dataset('v', data=v, dtype=np.float64)
    f['v'].attrs['units'] = b'cm/s'
    if set_density_from_mass:
        m = ...                     # Masses [g]
        f.create_dataset('m', data=m, dtype=np.float64)
        f['m'].attrs['units'] = b'g'
    else:
        rho = ...                   # Densities [g/cm^3]
        f.create_dataset('rho', data=rho, dtype=np.float64)
        f['rho'].attrs['units'] = b'g/cm^3'
    if use_internal_energy:
        e_int = ...                 # Internal energies [cm^2/s^2]
        f.create_dataset('e_int', data=e_int, dtype=np.float64)
        f['e_int'].attrs['units'] = b'cm^2/s^2'
    else:
        T = ...                     # Temperatures [K]
        f.create_dataset('T', data=T, dtype=np.float64)
        f['T'].attrs['units'] = b'K'
    x_HI = ...                      # Neutral hydrogen fraction (optional)
    x_HII = ...                     # Ionized hydrogen fraction (optional)
    x_H2 = ...                      # Molecular hydrogen fraction (optional)
    x_HeI = ...                     # Neutral helium fraction (optional)
    x_HeII = ...                    # Ionized helium fraction (optional)
    x_e = ...                       # Electron fraction (optional)
    f.create_dataset('x_HI', data=x_HI, dtype=np.float64)
    f.create_dataset('x_HII', data=x_HII, dtype=np.float64)
    f.create_dataset('x_H2', data=x_H2, dtype=np.float64)
    f.create_dataset('x_HeI', data=x_HeI, dtype=np.float64)
    f.create_dataset('x_HeII', data=x_HeII, dtype=np.float64)
```

(continues on next page)

(continued from previous page)

```
    f.create_dataset('x_e', data=x_e, dtype=np.float64)
    Z = ...                         # Metallicity [mass fraction] (optional)
    D = ...                         # Dust-to-gas ratio [mass fraction] (optional)
    f.create_dataset('Z', data=Z, dtype=np.float64)
    f.create_dataset('D', data=D, dtype=np.float64)
    # B = ...                         # Magnetic field [Gauss] (optional)
    # f.create_dataset('B', data=B, dtype=np.float64)
    # f['B'].attrs['units'] = b'G'
    # e_int = ...                     # Specific internal energy [cm^2/s^2] (optional)
    # f.create_dataset('e_int', data=e_int, dtype=np.float64)
    # f['e_int'].attrs['units'] = b'cm^2/s^2'
    # SFR = ...                       # Star formation rate [Msun/yr]
    # f.create_dataset('SFR', data=SFR, dtype=np.float64)
    # f['SFR'].attrs['units'] = b'Msun/yr'

    # Star properties
    f.attrs['n_stars'] = n_stars
    r_star = ...                    # Star positions [cm]
    f.create_dataset('r_star', data=r_star, dtype=np.float64)
    f['r_star'].attrs['units'] = b'cm'
    v_star = ...                    # Star velocities [cm]
    f.create_dataset('v_star', data=v_star, dtype=np.float64)
    f['v_star'].attrs['units'] = b'cm/s'
    m_init_star = ...               # Star initial masses [Msun]
    f.create_dataset('m_init_star', data=m_init_star, dtype=np.float64)
    f['m_init_star'].attrs['units'] = b'Msun'
    Z_star = ...                    # Star metallicity [mass fraction]
    f.create_dataset('Z_star', data=Z_star, dtype=np.float64)
    age_star = ...                  # Star ages [Gyr]
    f.create_dataset('age_star', data=age_star, dtype=np.float64)
    f['age_star'].attrs['units'] = b'Gyr'
```

1.3.3 Example Configuration File

While the format for configuration files was already described, we now provide a working example configuration file for H-alpha Monte Carlo radiative transfer.

```
# COLT config file
--- !mcrt                       # Monte Carlo radiative transfer module

init_dir: ics                   # Initial conditions directory (default: "ics")
init_base: colt                 # Initial conditions base name (optional)
output_dir: Ha                  # Output directory name (default: "output")
output_base: Ha                 # Output file base name (default: "colt")

recombinations: true            # Include recombination emission
collisions: true                # Include collisional excitation

cosmological: true              # Indicates whether the simulation is cosmological
line: Balmer-alpha              # Name of the line (default "Lyman-alpha")
v_turb_kms: 10                  # Microturbulent velocity [km/s]
```

(continues on next page)

(continued from previous page)

```
dust_model: MW                  # Dust model: SMC, MW, etc.

# Information about sources
n_photons: 1000000              # Number of photon packets (increase for production runs)
j_exp: 0.75                     # Luminosity boosting exponent

# Information about escape
output_photons: false           # Output escaped photon packets
spherical_escape: true          # Photons escape from a sphere
escape_radius_bbox: 0.9         # Escape radius relative to the bbox
emission_radius_bbox: 0.75      # Emission radius relative to the bbox

# Information about cameras
freq_min: -500                  # Frequency minimum [km/s]
freq_max: 500                   # Frequency maximum [km/s]
n_bins: 500                     # Number of frequency bins
image_radius_bbox: 0.5          # Image radius relative to the bbox (1 Rvir)
n_pixels: 512                   # Number of image pixels
cameras:                        # Add camera directions manually
  - [0,0,1]
  - [1,0,0]
output_freq_stds: true          # Output frequency standard deviations (default: false)
output_freq2_images: true       # Frequency moment images (default: false)
```

1.3.4 Output Files

The HDF5 file format allows self-describing output files. More specific details are coming soon.

1.4 Field Names and Descriptions

COLT aims to have short but intuitive names for its internal fields. This page provides a list of internal fields along with brief descriptions, units, and notes that might be helpful to remember for users or developers. Many fields are optional and only allocated when necessary and most can be used wih the projections module.

1.4.1 Gas Fields

- `n_cells` – Number of cells n_{cells}. In general, gas fields are arrays with this many elements.
- `r` – Position coordinates (x, y, z). Each geometry may interpret this differently. [Units: cm]
- `V` – Cell volume V. [Units: cm^3]
- `v` – Bulk velocity (v_x, v_y, v_z). [Units: cm/s]
- `dvdr` – Bulk velocity gradient $\mathrm{d}v/\mathrm{d}r$. [Units: 1/s]
- `v0` – Velocity extrapolation $v_0 = v - r\nabla v$. [Units: cm/s]
- `rho` – Gas density ρ. [Units: g/cm^3]
- `T` – Temperature T. [Units: K]
- `e_int` – Internal energy e_{int}. [Units: cm^2/s^2]

- `n_H` – Hydrogen number density $n_{\rm H}$. [Units: cm^-3]
- `n_He` – Helium number density $n_{\rm He}$. [Units: cm^-3]
- `n_C` – Carbon number density $n_{\rm C}$. [Units: cm^-3]
- `n_N` – Nitrogen number density $n_{\rm N}$. [Units: cm^-3]
- `n_O` – Oxygen number density $n_{\rm O}$. [Units: cm^-3]
- `n_Ne` – Neon number density $n_{\rm Ne}$. [Units: cm^-3]
- `n_Mg` – Magnesium number density $n_{\rm Mg}$. [Units: cm^-3]
- `n_Si` – Silicon number density $n_{\rm Si}$. [Units: cm^-3]
- `n_S` – Sulfer number density $n_{\rm S}$. [Units: cm^-3]
- `n_Fe` – Iron number density $n_{\rm Fe}$. [Units: cm^-3]
- `n_upper` and `n_lower` – Number density of the upper and lower transitions of a two-level atom $n_{\rm upper}$ and $n_{\rm lower}$. [Units: cm^-3]
- `x_HI` and `x_HII` – Hydrogen ionization fractions for neutral and ionized gas ($x_{\rm HI} = n_{\rm HI}/n_{\rm H}$). There are also ionized fractions for metal number densities under the convention that each is normalized to unity for the given species, e.g. for helium ($x_{\rm HeI} = n_{\rm HeI}/n_{\rm He}$) [Units: mass fraction]

- `x_H2` – Molecular hydrogen mass fraction (x

1.4.2 Star Fields

- `n_stars` – Number of stars $n_{\rm stars}$. In general, star fields are arrays with this many elements.
- `r_star` – Position coordinates (x, y, z). [Units: cm]
- `age_star` – Star age. [Units: Gyr]
- `Z_star` – Star metallicity $Z_\star$. [Units: mass fraction]
- `m_init_star` – Initial stellar mass $m_{\star,0}$. [Units: Msun]
- `m_massive_star` – Current stellar mass contained in massive stars. [Units: Msun]
- `L_cont_star` – Star continuum band flux $L_{\rm cont}$. [Units: erg/s/angstrom]
- `L_line_star` – Star line luminosity $L_{\rm line}$. [Units: erg/s]

1.5 Monte Carlo Radiative Transfer

This page explains the options for Monte Carlo radiative transfer (MCRT) simulations.

1.5.1 General MCRT Parameters

The following parameters are important MCRT parameters:

- `n_photons` – Actual number of photon packets used. [Default value: 1]
- `output_photons` – Flag to output escaped and absorbed photon packets. [Default value: true]
- `photon_file` – Output a separate photon file with this base name, e.g. "photons". [Optional]
- `output_n_scat` – Flag to output the number of scattering events for photon packets. [Default value: false]
- `output_collisions` – Flag to output collisional excitation data for photons and cameras. [Default value: false]
- `output_source_position` – Flag to output the emission position for photon packets. [Default value: false]

1.5.2 Line Parameters

The following parameters are related to line specifications:

- `line` – Name of the line to be modeled. [Default value: Lyman-alpha | Other options: Balmer-alpha, Balmer-beta, Balmer-gamma, Balmer-delta, Balmer-epsilon, Paschen-alpha, Paschen-beta, Brackett-alpha, CIII-1907-1909, CIV-1548-1550, NII-6550, NII-6585, NIII-1747-1749, NIV-1486, OI-6302, OI-6366, OII-3727-3730, OIII-1661-1666, OIII-4364, OIII-4933, OIII-4960, OIII-5008, NeIII-3969, MgII-2796-2803, SiII-1190-1193, SiII-1260, SiII-1304, SiII-1527, SiIII-1206, SiIV-1394-1403, SII-6718, SII-6733]
- `dust_model` – Dust model for the line radiative transfer. [Options: SMC, MW, LAURSEN_SMC, filename]
- `kappa_dust` – Dust opacity at the wavelength of the line. [Default value: Based on dust_model and line | Units: cm^2/g of dust]
- `sigma_dust` – Dust cross section at the wavelength of the line. [Default value: Based on dust_model and line | Units: cm^2/Z/hydrogen atom]
- `albedo` – Dust scattering albedo defined as $A = k_{\rm scat}/(k_{\rm abs} + k_{\rm scat})$. [Default value: Based on dust_model and line | Valid range: 0 < albedo < 1]
- `g_dust` – Anisotropy parameter $\langle \mu \rangle$ for dust scattering. [Default value: Based on dust_model and line | Valid range: -1 < g_dust < 1]
- `f_ion` – Survival fraction of dust in H II regions. [Default value: 1 | Valid range: 0 < f_ion < 1]
- `T_sputter` – Thermal sputtering temperature cutoff, such that no dust exists within gas hotter than this value. [Default value: 1e6 | Units: K]
- `p_dest` – Line destruction probability between absorption and re-emission. [Default value: 0 | Valid range: 0 < p_dest < 1]

1.5.3 Resonance Line Parameters

The following parameters are related to resonance lines:

- `recoil` – Include recoil induced Doppler shifting with line scattering. [Default value: true]
- `x_crit` – Constant critical frequency for core-skipping. [Default value: 0 | Valid range: x_crit > 0]
- `dynamical_core_skipping` – Non-local core-skipping for resonance lines. [Default value: false]
- `n_exp_atau` – Healpix exponent for the non-local $a\tau_0$ core-skipping estimates, resulting in $12 \times 4^{n_{\rm exp,atau}}$ directions. [Default value: 0 | Valid range: n_exp_atau >= 0]

- `n_side_atau` – Healpix number of base pixel subdivisions for the non-local $a\tau_0$ core-skipping estimates, resulting in $12 \times n^2_{\rm side,atau}$ directions. [Overrides n_exp_atau | Default value: 1 | Valid range: n_side_atau >= 1]

1.5.4 Exit Parameters

The following parameters are related to exit options:

- `doppler_frequency` – Output frequency in units of Doppler widths, otherwise as a velocity offset from line center (in units of km/s). [Default value: false]
- `T_exit` – Exit temperature for standardized Doppler frequency. [Default value: 1e4 | Units: K]
- `v_exit` – Exit velocity for shifted observer frames. [Default value: (0,0,0) | Units: cm/s]
- `exit_wrt_com` – Exit with respect to the center of mass velocity frame. [Default value: false]

Note: In what follows the units of "freq" refer to km/s unless using doppler_frequency.

1.5.5 Radiation Source Parameters

The following parameters are related to line radiative transfer options:

- `recombinations` – Include recombination emission with luminosity $\mathcal{L}_{\rm rec} = h\nu_0 \int \alpha_{\rm B,eff}(T) n_e n_{\rm HII} {\rm d}V$. [Default value: false]
- `collisions` – Include collisional excitation emission with luminosity for Ly of $\mathcal{L}_{\rm col} = h\nu_0 \int q_{\rm col}(T) n_e n_{\rm HI} {\rm d}V$. Note: This flag also works for all implemented hydrogen lines and many collisionally excited metal lines. [Default value: false]
- `continuum` – Include stellar continuum emission around the line. [Default value: false]
- `continuum_range` – Continuum velocity offset range for frequency sampling. [Default value: (-2000,2000) | Units: km/s]
- `j_exp` – Luminosity boosting exponent for power-law emission biasing, i.e. probabilities are rescaled according to $\propto \mathcal{L}_{\rm cell}^{j_{\rm exp}}$ such that $j_{\rm exp} < 1$ can better sample lower emissivity regions. [Default value: 1 | Valid range: 0 < j_exp < 1]
- `emission_n_min` – Minimum number density of emitting cells. [Default value: 0 | Units: cm^-3]
- `emission_n_max` – Maximum number density of emitting cells. [Default value: infinity | Units: cm^-3]
- `emission_ne_min` – Minimum electron number density of emitting cells. [Default value: 0 | Units: cm^-3]
- `emission_ne_max` – Maximum electron number density of emitting cells. [Default value: infinity | Units: cm^-3]
- `point` – Point source position in *[0 0 0]* format. [Optional | Units: cm]
- `x_point` – Point source x position. [Optional | Units: cm]
- `y_point` – Point source y position. [Optional | Units: cm]
- `z_point` – Point source z position. [Optional | Units: cm]
- `L_point` – Point source luminosity. [Default value: 1 | Units: erg/s]
- `r_shell` – Shell source radius. [Optional | Units: cm]
- `L_shell` – Shell source luminosity. [Default value: 1 | Units: erg/s]

1.5.6 Camera Parameters

The following parameters are related to cameras options:

- `n_bins` – Number of frequency bins. [Default value: 100]
- `n_slit_bins` – Number of frequency bins for slits. [Default value: n_bins]
- `n_cube_bins` – Number of frequency bins for data cubes. [Default value: n_bins]
- `n_radial_cube_bins` – Number of frequency bins for radial data cubes. [Default value: n_bins]
- `freq_range` – Frequency extrema in *[-1000 1000]* format. Note: Similar setup for `slit_freq_range`, `cube_freq_range`, and `radial_cube_freq_range`. [Default value: (-1000,1000) | Units: freq]
- `freq_min` – Frequency range minimum. Note: Similar setup for `slit_freq_min`, `cube_freq_min`, and `radial_cube_freq_min`. [Default value: -1000 | Units: freq]
- `freq_max` – Frequency range maximum. Note: Similar setup for `slit_freq_max`, `cube_freq_max`, and `radial_cube_freq_max`. [Default value: 1000 | Units: freq]
- `output_fluxes` – Output spectral fluxes. [Default value: true | Dimensions: (n_cameras, n_bins) | Output units: erg/s/cm^2/angstrom]
- `output_images` – Output surface brightness images. [Default value: true | Dimensions: (n_cameras, nx_pixels, ny_pixels) | Output units: erg/s/cm^2/arcsec^2]
- `output_slits` – Output spectral slits. [Default value: false | Dimensions: (n_cameras, n_slit_pixels, n_slit_bins) | Output units: erg/s/cm^2/arcsec^2/angstrom]
- `output_cubes` – Output spectral data cubes. [Default value: false | Dimensions: (n_cameras, nx_cube_pixels, ny_cube_pixels, n_cube_bins) | Output units: erg/s/cm^2/arcsec^2/angstrom]
- `output_images2` – Output statistical moment images, e.g. to estimate convergence statistics as the relative error is approximately $\sqrt{\text{images2}}/\text{images}$. [Default value: false | Dimensions: (n_cameras, nx_pixels, ny_pixels) | Output units: erg^2/s^2/cm^4/arcsec^4]
- `output_radial_images` – Output radial surface brightness images. [Default value: false | Dimensions: (n_cameras, n_radial_pixels) | Output units: erg/s/cm^2/arcsec^2]
- `output_radial_cubes` – Output radial spectral data cubes. [Default value: false | Dimensions: (n_cameras, n_radial_cube_pixels, n_radial_cube_bins) | Output units: erg/s/cm^2/arcsec^2/angstrom]
- `output_radial_images2` – Output statistical moment radial images, e.g. to estimate convergence statistics as the relative error is approximately $\sqrt{\text{radial\_images2}}/\text{radial\_images}$. [Default value: false | Dimensions: (n_cameras, n_radial_pixels) | Output units: erg^2/s^2/cm^4/arcsec^4]

The following parameters produce intrinsic or attenuation only images:

- `output_mcrt_emission` – Output intrinsic emission without transport based on MCRT sampling (also works for radial images). [Default value: false | Dimensions: (n_cameras, nx_pixels, ny_pixels) | Output units: erg/s/cm^2/arcsec^2]
- `output_mcrt_attenuation` – Output attenuated emission without scattering based on MCRT sampling (also works for radial images). [Default value: false | Dimensions: (n_cameras, nx_pixels, ny_pixels) | Output units: erg/s/cm^2/arcsec^2]
- `output_proj_emission` – Output intrinsic emission without transport with ray-based imaging. [Default value: false | Dimensions: (n_cameras, nx_pixels, ny_pixels) | Output units: erg/s/cm^2/arcsec^2]
- `output_proj_attenuation` – Output attenuated emission without scattering with ray-based imaging. [Default value: false | Dimensions: (n_cameras, nx_pixels, ny_pixels) | Output units: erg/s/cm^2/arcsec^2]

Note: The intrinsic and attenuation only images do not require scattering. Therefore, the projection outputs are preferred as they employ a ray-based quadtree imaging method for a noiseless solution with adaptive spatial convergence.

The following parameters produce spatially integrated camera frequency moments [Dimensions: n_cameras]:

- `output_escape_fractions` – Output camera escape fractions. [Default value: true]
- `output_freq_avgs` – Output camera frequency averages. [Default value: false | Output units: freq]
- `output_freq_stds` – Output frequency standard deviations. [Default value: false | Output units: freq]
- `output_freq_skews` – Output frequency skewnesses. [Default value: false]
- `output_freq_kurts` – Output frequency kurtoses. [Default value: false]

The following parameters produce camera frequency moments for each image pixel [Dimensions: (n_cameras, nx_pixels, ny_pixels)]:

- `output_freq_images` – Output average frequency images. [Default value: false | Output units: freq]
- `output_freq2_images` – Output frequency^2 images. [Default value: false | Output units: freq^2]
- `output_freq3_images` – Output frequency^3 images. [Default value: false | Output units: freq^3]
- `output_freq4_images` – Output frequency^4 images. [Default value: false | Output units: freq^4]

The following parameters produce camera frequency moments for each radial image pixel [Dimensions: (n_cameras, n_radial_pixels)]:

- `output_freq_radial_images` – Output average frequency radial images. [Default value: false | Output units: freq]
- `output_freq2_radial_images` – Output frequency^2 radial images. [Default value: false | Output units: freq^2]
- `output_freq3_radial_images` – Output frequency^3 radial images. [Default value: false | Output units: freq^3]
- `output_freq4_radial_images` – Output frequency^4 radial images. [Default value: false | Output units: freq^4]

The following parameters are useful for on-the-fly false color imaging:

- `output_rgb_images` – Output flux-weighted frequency RGB images. Note: The normalized image log intensity can be used as the alpha opacity of the image to give a nice illustration of the spectral data cube. [Default value: false | Dimensions: (n_cameras, nx_pixels, ny_pixels, 3) | Output units: RGB color]
- `output_rgb_radial_images` – Output flux-weighted frequency RGB radial images. [Default value: false | Dimensions: (n_cameras, n_radial_pixels, 3) | Output units: RGB color]
- `output_rgb_map` – Output flux-weighted frequency RGB Healpix map. [Default value: false | Dimensions: (12 * n_side_map^2, 3) | Output units: RGB color]
- `rgb_freq_range` – Frequency extrema in *[-1000 1000]* format. [Default value: (-1000,1000) | Units: freq]
- `rgb_freq_min` – Frequency range minimum. [Default value: -1000 | Units: freq]
- `rgb_freq_max` – Frequency range maximum. [Default value: 1000 | Units: freq]
- `adjust_camera_frequency` – Adjust frequencies for each camera based on offsets from a file. Note: This was implemented to center the RGB and other cameras for Ly on the H line center. Thus, the file must have a dataset called "freq_avgs" of length n_cameras. [Default value: false]
- `freq_offset_file` – Frequency offset file (optional). [Default value: `init_file`]

- `freq_offset_dir` – Frequency offset directory (optional). [Default value: `init_dir`]
- `freq_offset_base` – Frequency offset file base (optional). [Default value: `init_base`]
- `freq_offset_ext` – Frequency offset file extension (optional). [Default value: hdf5]

1.5.7 Healpix Binning Parameters

The following parameters are related to line-of-sight Healpix binning options:

- `n_exp_map` – Healpix exponent for the line-of-sight map, resulting in $12 \times 4^{n_{\rm exp,map}}$ directions. Note: Similar setup for `n_exp_flux`, `n_exp_radial`, and `n_exp_cube`. [Default value: 2 | Valid range: n_exp_map >= 0]
- `n_side_map` – Healpix number of base pixel subdivisions for the line-of-sight map, resulting in $12 \times n^2_{\rm side,map}$ directions. Note: Similar setup for `n_side_flux`, `n_side_radial`, and `n_side_cube`. [Overrides n_exp_map | Default value: 4 | Valid range: n_side_map >= 1]
- `n_map_bins` – Number of frequency bins for the line-of-sight flux map. Note: Similar setup for `n_cube_map_bins`. [Default value: 100]
- `map_freq_range` – Frequency extrema in *[-1000 1000]* format for the line-of-sight flux map. Note: Similar setup for `cube_map_freq_range`. [Default value: (-1000,1000) | Units: freq]
- `map_freq_min` – Frequency range minimum for the line-of-sight flux map. Note: Similar setup for `cube_map_freq_min`. [Default value: -1000 | Units: freq]
- `map_freq_max` – Frequency range maximum for the line-of-sight flux map. Note: Similar setup for `cube_map_freq_max`. [Default value: 1000 | Units: freq]
- `map_radius` – Radius for the line-of-sight Healpix map of radial surface brightness images. Note: Similar setup for `cube_map_radius`. [Units: cm]
- `map_radius_bbox` – Radius for the line-of-sight Healpix map of radial surface brightness images. Note: Similar setup for `cube_map_radius_bbox`. [Units: bounding box]
- `n_map_pixels` – Number of radial pixels for the line-of-sight Healpix map of radial surface brightness images. Note: Similar setup for `n_cube_map_pixels`. [Default value: 100]
- `map_pixel_width` – Intrinsic width of each radial pixel for the line-of-sight Healpix map of radial surface brightness images. Note: Similar setup for `cube_map_pixel_width`. [Units: cm]
- `map_pixel_arcsec` – Observed angular width of each radial pixel for the line-of-sight Healpix map of radial surface brightness images. Note: Similar setup for `cube_map_pixel_arcsec`. [Units: arcseconds]
- `output_map` – Output a line-of-sight Healpix map of escape fractions. [Default value: false | Dimensions: (12 * n_side_map^2)]
- `output_map2` – Output statistical moment line-of-sight Healpix map, e.g. to estimate convergence statistics as the relative error is approximately $\sqrt{\rm map2}/{\rm map}$. [Default value: false | Dimensions: (12 * n_side_map^2)]
- `output_flux_map` – Output a line-of-sight Healpix map of spectral fluxes. [Default value: false | Dimensions: (12 * n_side_flux^2, n_map_bins) | Output units: erg/s/cm^2/angstrom]
- `output_radial_map` – Output a line-of-sight Healpix map of radial surface brightness images. [Default value: false | Dimensions: (12 * n_side_radial^2, n_map_pixels) | Output units: erg/s/cm^2/arcsec^2]
- `output_cube_map` – Output a line-of-sight Healpix map of radial spectral data cubes. [Default value: false | Dimensions: (12 * n_side_cube^2, n_cube_map_pixels, n_cube_map_bins) | Output units: erg/s/cm^2/arcsec^2/angstrom]

The following parameters produce frequency moments for each Healpix map pixel [Dimensions: 12 * n_side_map^2]:

- `output_freq_map` – Output average frequency map. [Default value: false | Output units: freq]

- `output_freq2_map` – Output frequency^2 map. [Default value: false | Output units: freq^2]
- `output_freq3_map` – Output frequency^3 map. [Default value: false | Output units: freq^3]
- `output_freq4_map` – Output frequency^4 map. [Default value: false | Output units: freq^4]

1.5.8 Angular Binning Parameters

The following parameters are related to line-of-sight angular cosine ($\mu = \cos\theta$) binning options:

- `n_mu` – Number of directional bins for the line-of-sight μ map. Note: Similar setup for `n_mu_flux`. [Default value: 100]
- `n_mu_bins` – Number of frequency bins for the line-of-sight flux μ map. [Default value: 100]
- `mu_freq_range` – Frequency extrema in *[-1000 1000]* format for the line-of-sight flux μ map. [Default value: (-1000,1000) | Units: freq]
- `mu_freq_min` – Frequency range minimum for the line-of-sight flux μ map. [Default value: -1000 | Units: freq]
- `mu_freq_max` – Frequency range maximum for the line-of-sight flux μ map. [Default value: 1000 | Units: freq]
- `output_mu` – Output a line-of-sight μ map of escape fractions. [Default value: false | Dimensions: n_mu)]
- `output_mu2` – Output statistical moment line-of-sight μ map, e.g. to estimate convergence statistics as the relative error is approximately $\sqrt{\text{map2}}/\text{map}$. [Default value: false | Dimensions: n_mu)]
- `output_flux_mu` – Output a line-of-sight μ map of spectral fluxes. [Default value: false | Dimensions: (n_mu, n_mu_bins) | Output units: erg/s/cm^2/angstrom]

The following parameters produce frequency moments for each μ map [Dimensions: n_mu]:

- `output_freq_mu` – Output average frequency μ map. [Default value: false | Output units: freq]
- `output_freq2_mu` – Output frequency^2 μ map. [Default value: false | Output units: freq^2]
- `output_freq3_mu` – Output frequency^3 μ map. [Default value: false | Output units: freq^3]
- `output_freq4_mu` – Output frequency^4 μ map. [Default value: false | Output units: freq^4]

1.5.9 Advanced Parameters

The following parameters are related to advanced options:

- `two_level_atom` – Line radiative transfer incorporates occupation number densities from both lower and upper transitions. If this is false then we assume that bound electrons are always in the ground state. Note: This currently also switches from a Voigt to a Gaussian line profile, and assumes complete redistribution for scattering. [Default value: false]
- `resonant_scattering` – Turn resonant scattering on/off. Note: It is not recommended to override this option unless you know exactly what you are doing. [Default value: Based on line]
- `load_balancing` – Use MPI load balancing algorithms. Note: Due to the OpenMP load balancing this is not recommended in general as the number of MPI tasks is usually much smaller than the number of photon packets. [Default value: false]

1.6 Ray-tracing Radiative Transfer

This page explains the options for ray-tracing radiative transfer (RT) simulations.

1.6.1 General RT Parameters

The following parameters are important MCRT parameters:

Coming soon.

1.7 Adaptive Projections

This page explains the options for adaptive convergence volumetric projections.

1.7.1 General Parameters

The following are important projection parameters in addition to the general *Simulation* ones:

- `proj_depth` – Specifies the depth of the projection. If not explicitly set, it defaults to the value of `image_width`. [Units: cm]
- `proj_radius` – Specifies the radius of the projection, which is half of the projection depth. This cannot be used if `proj_depth` is set. [Units: cm]
- `proj_radius_bbox` – Specifies the radius of the projection relative to the bounding box. This cannot be used if `proj_depth` or `proj_radius` is set. [Default value: 0.]
- `adaptive` – Enables the use of adaptive convergence in projections. When set to true, the projection algorithm uses a quadtree to automatically determine the optimal number of rays to use for each pixel. This is achieved by recursively subdividing the pixel until the error is below the specified tolerance. The error is estimated by comparing the results from a 4-point trapezoidal integration scheme to a second-order 9-point Simpson's rule. [Default value: false]
- `pixel_rtol` – Sets the relative tolerance per pixel when using adaptive convergence. It controls the maximum allowed error per pixel, allowing for accurate quantity-conserving projection images. [Default value: 0.01]
- `perspective` – Enables the use of a perspective camera for rendering the projection. When set to true, the image rays will no longer be plane-parallel but will instead converge to a focal point. This provides a more realistic perception of distances as closer objects will appear larger than those that are farther away. [Default value: false]
- `stepback_factor` – This determines the stepback distance for the projection, expressed as a factor relative to the image radius. It adjusts the viewpoint to conveniently capture the external perspective of an entire simulation volume, e.g. for large-scale cosmological boxes, as well as for creating animations of rotating galaxies. [Default value: 0.]
- `stepback_factor_bbox` – Identical to the previous `stepback_factor` parameter for perspective cameras but is instead relative to the bounding box. [Default value: 0.]
- `proj_sphere` – Enables the projection to be through a spherical volume. When set to true, the rays will be truncated to be within a sphere rather than a rectangular box. This is useful for creating rotation animations where features do not enter or exit the image due to being near the corners of the projected volume. [Default value: false]

- `proj_cylinder` – Enables the projection to be through a cylindrical volume. Similar to the `proj_sphere` option, but the rays are truncated to be within a cylinder oriented along the camera north vector. This may be a more natural choice for rotation animations than using a sphere. [Default value: false]
- `override_volume` – Allows for manually overridiong the projection volume normalization. Specifically, it allows for the pixel integration depths to vary according to the bounding box and projection shape volumes rather than using a constant value across the entire image based on the standard rectangular volume. [Default value: false]
- `shape_factor` – Determines the extraction shape factor for the projection, expressed relative to the image radius. For example, this sets the radius of the sphere or cylinder for the `proj_sphere` and `proj_cylinder` options, respectively. [Default value: 0.]
- `shape_factor_bbox` – Identical to the previous `shape_factor` parameter but is instead relative to the bounding box. [Default value: 0.]
- `fade_in_length` – Specifies the characteristic position where the weights gradually increase from zero, setting the location of a Sigmoid filter within the projection volume. This is useful for minimizing abrupt entrance and exit effects when flying through a simulation volume, as well as maintaining the focus on the central object rather than the transition to the background. By default there is no fading. [Units: cm]
- `fade_in_length_bbox` – Identical to the previous `fade_in_length` parameter but is instead relative to the bounding box. [Default value: 0.]
- `fade_in_width` – Specifies the characteristic distance over which the weights gradually increase from zero, effectively controlling the smoothness of the projection integration edges setting the width of a Sigmoid filter within the projection volume. See the further comments regarding the `fade_in_length` parameter. [Units: cm]
- `fade_in_width_bbox` – Identical to the previous `fade_in_width` parameter but is instead relative to the bounding box. [Default value: 0.]
- `fade_out_length` – Identical to the `fade_in_length` parameter but for the exit side of the projection volume. [Units: cm]
- `fade_out_length_bbox` – Identical to the previous `fade_out_length` parameter but is instead relative to the bounding box. [Default value: 0.]
- `fade_out_width` – Identical to the `fade_in_width` parameter but for the exit side of the projection volume. [Units: cm]
- `fade_out_width_bbox` – Identical to the previous `fade_out_width` parameter but is instead relative to the bounding box. [Default value: 0.]
- `field_weight_pairs` – Accepts a list of projection field and weight pairs, which are used in the integration calculations. The field is the primary quantity of interest, e.g. density or temperature, while the weight biases the contribution based on a secondary quantity or normalization scheme. For example, providing `[rho, avg]` produces a volume-weighted density image along the line of sight, or $\langle\rho\rangle_V \equiv \int \rho \, \mathrm{d}V / \int \mathrm{d}V$. Likewise, providing `[T, mass]` produces a mass-weighted temperature image, or $\langle T\rangle_m \equiv \int T\rho \, \mathrm{d}V / \int \rho \, \mathrm{d}V$, and providing `[SFR, sum]` produces a total star formation rate image, or $\sum \mathrm{SFR} \equiv \int \rho_{\mathrm{SFR}} \, \mathrm{d}V$, which is useful for conservative (non-density-like) fields. Finally, providing an arbitrary field and weight is equivalent to $\langle f\rangle_w \equiv \int f w \, \mathrm{d}V / \int w \, \mathrm{d}V$ and will automatically produce the denominator image as well to allow reconstructing the numerator image later. The field names are case-sensitive and must match the simulation field names (see *Gas Fields*), however known aliases are also accepted, e.g. `Density` is equivalent to `rho`. The weights also allow aliases of `Average`, `Mass`, and `Sum` for `avg`, `mass`, and `sum`, respectively.
- `fields` – Alternatively, a list of projection fields can be specified with volume-weighted integrations by default (`avg`). This is useful for simple projections where the weights are not needed. For example, `[rho, T]` is equivalent to `[[rho, avg], [T, avg]]`.
- `weights` – When the `fields` list is provided, a list of projection weights can be specified to override the `avg` default. This is useful if only the first weights differ from the default behavior. For example, `[mass]` is equivalent

to `[[rho, mass], [T, avg]]` for the previous example.

- `kappa_field` – Specifies the field name for the absorption opacity in the same sense as the radiative transfer equation. This is useful for emulating synthetic observations of emissivity- and obscuration-like fields, or for creating more intuitive images accounting for gas/dust column densities. The absorption opacity can be further modified via power-law modifications as specified below, such that the optical depth is $\tau = \int \kappa_0 \kappa^\beta \rho^\gamma \, \mathrm{d}\ell$, where κ_0 is an opacity scaling constant, β is an opacity exponent, γ is a density exponent, and ℓ is the line-of-sight distance. By default the absorption opacity is disabled. [Default value: “”]
- `kappa_constant` – Determines the absorption opacity scaling constant κ_0 (see the `kappa_field` description). [Default value: 1.]
- `kappa_exponent` – Determines the absorption opacity exponent β (see the `kappa_field` description). [Default value: 1.]
- `rho_exponent` – Determines the absorption opacity density exponent γ (see the `kappa_field` description). [Default value: 1.]

These parameters should be defined in your configuration file, e.g., `config-proj.yaml`.

1.7.2 Quickstart Example

This section provides detailed explanations of setting up, running, and visualizing the results from the COLT projections module. For simplicity, our example is based on a dodecahedron shape, constructed from a Voronoi mesh with a single interior cell and 12 edge cells. The final image should look like the following:

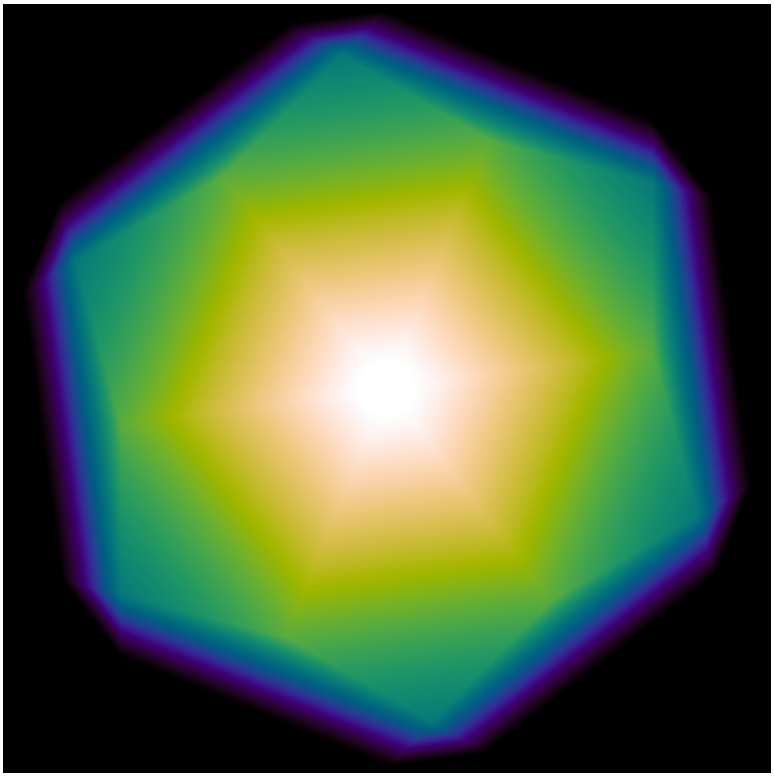


Fig. 1: **Projection Example:** This image is a volume-weighted density image of a dodecahedron.

We suggest first setting up a directory structure as follows:

```
.proj-test             # Dodecahedron test directory
├── config-proj.yaml   # Configuration file
├── defines.yaml       # Compile time definitions
├── ics                # Initial conditions directory
│   ├── dodec.hdf5     # Initial conditions file (created by dodec.py)
│   └── dodec.py       # Initial conditions script (creates dodec.hdf5)
├── output             # Output directory (created by COLT)
│   └── proj.hdf5      # Projection output file (created by COLT)
├── plots              # Visualization directory
│   ├── dodec.png      # Final image (created by plot.py)
│   └── plot.py        # Visualization script (creates dodec.png)
└── run.sh             # Compilation and execution convenience script
```

The `defines.yaml` compile time definitions file should contain the following:

```
# Compile time definitions

GEOMETRY: voronoi  # Geometry type: slab, spherical, cartesian, octree, voronoi
# HAVE_CGAL: true    # Include the CGAL library (requires voronoi geometry)
```

CGAL is not strictly necessary for this example, but provides the general Voronoi mesh construction utilities COLT relies on. The initial conditions script `ics/dodec.py` should contain the following:

```
import numpy as np
import h5py

pc = 3.085677581467192e18  # 1 pc  = 3e18 cm
kpc = 1e3 * pc             # 1 kpc = 3e21 cm

# Create dodecahedron mesh
n_cells = 13  # Number of cells
C0 = 0.  # Center of dodecahedron
C1 = np.sqrt(2. - 2./np.sqrt(5.)) * kpc  # = 1.05146 kpc
C2 = np.sqrt(2. + 2./np.sqrt(5.)) * kpc  # = 1.70130 kpc
r = np.array([[ C0,  C0,  C0],  # Mesh generating points
    [ C1,  C0,  C2], [ C2,  C1,  C0], [ C0,  C2,  C1], [ C0,  C2, -C1],
    [ C1,  C0, -C2], [-C1,  C0,  C2], [ C0, -C2,  C1], [ C2, -C1,  C0],
    [ C0, -C2, -C1], [-C2,  C1,  C0], [-C1,  C0, -C2], [-C2, -C1,  C0]], dtype=np.
↪float64)
one = np.ones(n_cells)  # Array of ones
cen = np.zeros(n_cells); cen[0] = 1.  # Center cell
with h5py.File(f'dodec.hdf5', 'w') as f:
    # Basic data (gas properties)
    f.attrs['n_cells'] = np.int32(n_cells)  # Number of cells
    f.attrs['r_box'] = np.float64(2.*kpc)  # Box radius [cm]
    f.create_dataset('r', data=r)  # Positions [cm]
    f['r'].attrs['units'] = b'cm'
    f.create_dataset('v', data=np.zeros_like(r))  # Velocities [cm/s]
    f['v'].attrs['units'] = b'cm/s'
    f.create_dataset('rho', data=cen)  # Density [g/cm^3]
    f['rho'].attrs['units'] = b'g/cm^3'
    f.create_dataset('T', data=one)  # Temperature [K]
    f['T'].attrs['units'] = b'K'
```

(continues on next page)

(continued from previous page)

```
    # Voronoi mesh data (CGAL equivalent)
    f.attrs['n_circulators_tot'] = np.int32(60)  # Number of circulators
    f.attrs['n_edges'] = np.int32(12)  # Number of edge cells
    f.attrs['n_inner_edges'] = np.int32(1)  # Number of inner edge cells
    f.attrs['n_neighbors_tot'] = np.int32(84)  # Number of neighbors
    V0 = 10. * np.sqrt(130. - 58.*np.sqrt(5.)) * kpc**3  # = 5.55029 kpc^3
    f.create_dataset('V', data=V0*cen)  # Cell volumes [cm^3]
    f['V'].attrs['units'] = b'cm^3'
    ci = 60 * np.ones(85, dtype=np.int32)
    ci[:12] = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55]
    cip = np.array([2, 8, 7, 6, 3, 3, 4, 5, 8, 1, 2, 4, 10, 6, 1, 10, 3, 2, 5,
        11, 9, 8, 2, 4, 11, 1, 3, 10, 12, 7, 6, 12, 9, 8, 1, 9, 7, 1, 2, 5, 11,
        12, 7, 8, 5, 12, 6, 3, 4, 11, 9, 12, 10, 4, 5, 6, 10, 11, 9, 7], dtype=np.int32)
    ni = np.array([0, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84], dtype=np.
↪int32)
    nip = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 0, 2, 3, 6, 7, 8, 0,
        1, 3, 4, 5, 8, 0, 1, 2, 4, 6, 10, 0, 2, 3, 5, 10, 11, 0, 2, 4, 8, 9, 11,
        0, 1, 3, 7, 10, 12, 0, 1, 6, 8, 9, 12, 0, 1, 2, 5, 7, 9, 0, 5, 7, 8, 11,
        12, 0, 3, 4, 6, 11, 12, 0, 4, 5, 9, 10, 12, 0, 6, 7, 9, 10, 11], dtype=np.int32)
    f.create_dataset('circulator_indices', data=ci)  # Circulator indices
    f.create_dataset('circulator_indptr', data=cip)  # Circulator indptr
    f.create_dataset('edges', data=np.arange(1, 13, dtype=np.int32))  # Edge indices
    f.create_dataset('inner_edges', data=np.array([0], dtype=np.int32))  # Inner edges
    f.create_dataset('neighbor_indices', data=ni)  # Neighbor indices
    f.create_dataset('neighbor_indptr', data=nip)  # Neighbor indptr
```

This will create the `ics/dodec.hdf5` file, which is specified as the initial conditions file in the configuration file `config-proj.yaml`. The HDF5 file stores the basic gas properties (density, temperature, etc.) as well as the Voronoi mesh connectivity data (edge cells, neighbors, circulators, etc.) that COLT uses for the projection.

The `config-proj.yaml` runtime configuration file should contain the following:

```
# COLT config file
--- !projections              # Ray-based projections module

init_file: ics/dodec.hdf5     # Initial conditions file
output_dir: output            # Output directory name
output_base: proj             # Output file base name

avoid_edges: false            # Avoid ray-tracing through edge cells
field_weight_pairs:
  - [rho, avg]                # Pairs of fields and weights

# Camera Information
image_radius_bbox: 0.6        # Image radius relative to the bounding box
proj_radius_bbox: 0.6         # Projection radius relative to the bounding box
n_pixels: 400                 # Number of image pixels (resolution)
camera: [1,1,1]               # Camera orientation
```

This supplies the initial conditions file, output directory, and output file base name. The `avoid_edges` option is set to false to allow the rays to pass through the edge cells without artificial termination. The `field_weight_pairs` option specifies the density field and volume-weighted integration. The `image_radius_bbox` and `proj_radius_bbox` options specify the image size and projection depth relative to the bounding box, respectively. The `n_pixels` option

specifies the number of image pixels, and the `camera` option specifies the viewing direction.

At this point COLT can be compiled and executed, which will create the `output/proj.hdf5` file that stores the raw projection data. To visualize the results, we provide a `plots/plot.py` script that creates the final image and should contain the following:

```
import numpy as np
import h5py, cmasher
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

# Plot dodecahedron image
with h5py.File(f'../output/proj.hdf5', 'r') as f:
    Z = f['proj_rho_avg'][0]; Z /= np.max(Z)
    R = f.attrs['image_radius']
    extent = [-R, R, -R, R]
fig = plt.figure(figsize=(4., 4.))
ax = plt.axes([0, 0, 1, 1])
ax.imshow(Z.T, origin='lower', extent=extent, cmap='cmr.rainforest', aspect='equal',
            interpolation='bicubic', norm=Normalize(vmin=0., vmax=1.))
ax.set_axis_off()
fig.savefig('dodec.png', bbox_inches='tight', pad_inches=0., dpi=100)
```

For convenience, we provide a full bash script for generating initial conditions, compiling, running, and plotting the results for the dodecahedron example. This should be run within the top `proj-test` directory with `bash run.sh` and contains the following:

```
#!/bin/bash

# Generate initial conditions
cd ics  # Navigate to the ics directory
python3 dodec.py  # Create the initial conditions file
cd ..  # Navigate back to the test directory

# Compile COLT
code_path=$(realpath "/path/to/colt")  # Set path to COLT source code
run_path=$(realpath ".")  # Set path to COLT test directory
opts="DEFS=$run_path/defines.yaml BUILD=$run_path/build EXE=$run_path/colt"
make clean_exe $opts -C $code_path
make -j $opts -C $code_path

# Run COLT
export OMP_NUM_THREADS=16
mpirun -n 1 ./colt config-proj.yaml

# Plot the results
cd plots  # Navigate to the plots directory
python3 plot.py  # Create the plot
cd ..  # Navigate back to the test directory
```

Note that the `OMP_NUM_THREADS` environment variable is set to enable OpenMP parallelism for the projection algorithm to process multiple image pixels at a time. When making projections of large simulations for multiple cameras, it is recommended to also use MPI parallelism to process multiple cameras at a time, which can be achieved with `mpirun -n [n_tasks] ./colt config-proj.yaml`. We recommend using one MPI task per node (at most the number of cameras) and one OpenMP thread per core for optimal performance.

1.8 MCRT Ionization Equilibrium

This page explains the options for ionization equilibrium Monte Carlo radiative transfer (MCRT) simulations.

1.8.1 General Parameters

The following parameters are important module parameters:

Coming soon.

1.9 Ray Data Extractions

This page explains the options for the extraction of ray-like data from simulations.

1.9.1 General Ray Parameters

The ray directions are set by the standard camera options. In addition, the following parameters affect how ray origins are selected:

- `origins_bbox` – List of ray origins (optional). [Units: bounding box]
- `origin_bbox` – Common origin for rays (optional). [Units: bounding box]

Otherwise, the origin is taken from the group catalogs using the `group` or `subhalo` command line argument following the `snapshot` argument:

- `fof_base` – Friends of friends base name (optional). [Default value: fof_subhalo_tab]
- `black_hole_origin` – Flag to use the most massive black hole of the subhalo (optional). [Default value: false]
- `lyman_alpha_origin` – Flag to use the Lyman-alpha subhalo catalog (optional). [Default value: false]

The following parameters control the start and length of the rays:

- `start_fvir` – Ray starting radius offset intended for halo origins. [Units: virial radius]
- `start_cMpc` – Ray starting radius offset intended for halo origins. [Units: cMpc]
- `length_kms` – Ray length proxy of Doppler shift induced by Hubble flow. [Units: km/s]
- `length_bbox` – Ray length in special box units. [Units: bounding box]
- `length_Mpc` – Ray length in physical units. [Units: Mpc]
- `length_cMpc` – Ray length in comoving units. [Units: cMpc]
- `impact_kpc` – Cylinder impact radius in physical units (optional optimization). [Units: kpc]

The following parameters control the ray output fields (note that density is always output):

- `output_dust` – Flag to output the dust metallicity. [Default value: true]
- `output_metallicity` – Flag to output the metallicity. [Default value: true]
- `output_HI` – Flag to output the HI fraction. [Default value: true]
- `output_HeI` – Flag to output the HeI fraction. [Default value: false]
- `output_HeII` – Flag to output the HeII fraction. [Default value: false]
- `output_neutral` – Flag to output the neutral hydrogen abundance. [Default value: false]

- `output_electrons` – Flag to output the electron abundance. [Default value: true]
- `output_internal_energy` – Flag to output the internal energy. [Default value: true]
- `output_SFR` – Flag to output the star formation rate. [Default value: true]
- `output_parallel_velocity` – Flag to output the parallel velocity. [Default value: true]
- `output_velocities` – Flag to output the full (x, y, z) velocities. [Default value: false]
- `output_acceleration` – Flag to output the full (x, y, z) acceleration. [Default value: false]
- `output_radiation` – Flag to output the photon density. [Default value: false]
- `output_magnetic_field` – Flag to output the magnetic field. [Default value: false]
- `output_metals` – Flag to output the metal species fractions. [Default value: false]
- `output_metals_as_densities` – Flag to output the metal species as densities rather than fractions. [Default value: false]

1.10 Reionization Box Analysis

This page explains the options for reionization box simulation analaysis.

1.10.1 General Parameters

The following parameters are important reionization analysis parameters:

Coming soon.

1.11 Publications

This page maintains a current list of publications using various aspects of COLT.

1.11.1 Original Methods Paper:

- Smith, A., Safranek-Shrader, C., Bromm, V., Milosavljević, M., 2015, MNRAS, 449, 4336: 'The Lyman signature of the first galaxies' (arXiv:1409.4480)

1.11.2 Science Papers:

- McClymont, W., Tacchella, S., Smith, A., Kannan, R., Maiolino, R., Belfiore, F., Hernquist, L., Li, H., Vogelsberger, M., 2024, MNRAS, submitted: 'The nature of diffuse ionised gas in star-forming galaxies' (arXiv:2403.03243)
- Garaldi, E., Kannan, R., Smith, A., Borrow, J., Vogelsberger, M., Pakmor, R., Springel, V., Hernquist, L., Galárraga-Espinosa, D., Yeh, J. Y.-C., Shen, X., Xu, C., Neyer, M., Spina, B., Almualla, M., Zhao, Y., 2024, MNRAS, in press: 'The THESAN project: public data release of radiation-hydrodynamic simulations matching reionization-era JWST observations' (arXiv:2309.06475)
- Xu, C., Smith, A., Borrow, J., Garaldi, E., Kannan, R., Vogelsberger, M., Pakmor, R., Springel, V., Hernquist, L., 2023, MNRAS, 521, 4356: 'The THESAN project: Lyman- emitter luminosity function calibration' (arXiv:2210.16275)

- Yeh, J. Y.-C., Smith, A., Kannan, R., Garaldi, E., Vogelsberger, M., Borrow, J., Pakmor, R., Springel, V., Hernquist, L., 2023, MNRAS, 520, 2757: 'The THESAN project: ionizing escape fractions of reionization-era galaxies' (arXiv:2205.02238)
- Jahn, E. D., Sales, L. V., Marinacci, F., Vogelsberger, M., Torrey, P., Qi, J., Smith, A., Li, H., Kannan, R., Burger, J. D., Zavala, J., 2023, MNRAS, 520, 461: 'Real and counterfeit cores: how feedback expands haloes and disrupts tracers of inner gravitational potential in dwarf galaxies' (arXiv:2110.00142)
- Smith, A., Kannan, R., Tacchella, S., Vogelsberger, M., Hernquist, L., Marinacci, F., Sales, L.V., Torrey, P., Li, H., Yeh, J. Y.-C., Qi, J., 2022, MNRAS, 517, 1: 'The physics of Lyman- escape from disc-like galaxies' (arXiv:2111.13721)
- Tacchella, S., Smith, A., Kannan, R., Marinacci, F., Hernquist, L., Vogelsberger, M., Torrey, P., Sales, L., Li, H., 2022, MNRAS, 513, 2904: 'H emission in local galaxies: star formation, time variability, and the diffuse ionized gas' (arXiv:2112.00027)
- Garaldi, E., Kannan, R., Smith, A., Springel, V., Pakmor, R., Vogelsberger, M., Hernquist, L., 2022, MNRAS, 512, 4909: 'The THESAN project: properties of the intergalactic medium and its connection to reionization-era galaxies' (arXiv:2110.01628)
- Smith, A., Kannan, R., Garaldi, E., Vogelsberger, M., Pakmor, R., Springel, V., Hernquist, L., 2022, MNRAS, 512, 3243: 'The THESAN project: Lyman- emission and transmission during the Epoch of Reionization' (arXiv:2110.02966)
- Kimock, B., Narayanan, D., Smith, A., Ma, X., Feldmann, R., Anglés-Alcázar, D., Bromm, V., Davé, R., Geach, J., Hopkins, P., Kereš, D., 2021, ApJ, 909, 119: 'The Origin and Evolution of Lyman- Blobs in Cosmological Galaxy Formation Simulations' (arXiv:2004.08397)
- Lao, B.-X., Smith, A., 2020, MNRAS, 497, 3925: 'Resonant-line radiative transfer within power-law density profiles' (arXiv:2005.09692)
- Yang, Y.-L., Evans, N. J., Smith, A., Lee, J.-E., Tobin, J. J., Terebey, S., Calcutt, H., Jørgensen, J. K., Green, J. D., Bourke, T. L., 2020, ApJ, 891, 61: 'Constraining the Infalling Envelope Models of Embedded Protostars: BHR 71 and Its Hot Corino' (arXiv:2002.01478)
- Smith, A., Ma, X., Bromm, V., Finkelstein, S. L., Hopkins, P. F., Faucher-Giguère, C.-A., Kereš, D., 2019, MNRAS, 484, 39: 'The physics of Lyman escape from high-redshift galaxies' (arXiv:1810.08185)
- Smith, A., Tsang, B. T.-H., Bromm, V., Milosavljević, M., 2018, MNRAS, 479, 2065: 'Discrete diffusion Lyman radiative transfer' (arXiv:1709.10187)
- Smith, A., Becerra, F., Bromm, V., Hernquist, L., 2017, MNRAS, 472, 205: 'Radiative effects during the assembly of direct collapse black holes' (arXiv:1706.02751)
- Smith, A., Bromm, V., Loeb, A., 2017, MNRAS, 464, 2963: 'Lyman radiation hydrodynamics of galactic winds before cosmic reionization' (arXiv:1607.07166)
- Smith, A., Bromm, V., Loeb, A., 2016, MNRAS, 460, 3143: 'Evidence for a direct collapse black hole in the Lyman source CR7' (arXiv:1602.07639)

1.12 Gallery

This page provides a gallery of figures to illustrate various capabilities and potential applications.

Coming soon.

1.13 Examples

This page provides several quickstart examples for new users of the COLT code.

Coming soon.

1.14 pycolt

constexpr double **km** = 1e5

class **Escape** : public Simulation

Public Functions

virtual void **run**() override

Protected Functions

virtual void **module_config**(YAML::Node &file) override

virtual void **setup**() override

virtual void **print_info**() override

virtual void **write_module**(const H5::H5File &f) override

Private Functions

void **ray_trace**(const int star, const int camera)

void **calculate_escape**()

void **correct_units**()

Private Members

string **source_model** = ""

vector<double> **global_sigma_HI**

vector<double> **global_sigma_HeI**

vector<double> **global_sigma_HeII**

vector<double> **global_mean_energy**

double **L_tot** = 0.

double **Ndot_tot** = 0.

vector<double> **bin_L_tot**

vector<double> **bin_Ndot_tot**

vector<int> **cell_of_star**

vector<double> **global_kappa**

bool **output_emission** = false

bool **output_bin_escape_fractions** = true

vectors **bin_f_escs**

bool **output_dust_absorptions** = true

vector<double> **f_abss**

bool **output_bin_dust_absorptions** = false

vectors **bin_f_abss**

bool **output_HI_absorptions** = true

bool **output_HeI_absorptions** = true

bool **output_HeII_absorptions** = true

vector<double> **f_HIs**

vector<double> **f_HeIs**

vector<double> **f_HeIIs**

bool **output_bin_HI_absorptions** = false

bool **output_bin_HeI_absorptions** = false

bool **output_bin_HeII_absorptions** = false

vectors **bin_f_HIs**

vectors **bin_f_HeIs**

vectors **bin_f_HeIIs**

bool **output_absorption_distances** = true

vector<double> **mean_dists**

bool **output_bin_absorption_distances** = false

vectors **bin_mean_dists**

Images **images_int**

Cubes **bin_images**

Cubes **bin_images_int**

vectors **radial_images_int**

Images **bin_radial_images**

Images **bin_radial_images_int**

bool **output_gas_columns_int** = true

vector<double> **gas_columns_int**

bool **output_gas_columns_esc** = true

vector<double> **gas_columns_esc**

bool **output_dust_columns_int** = true

vector<double> **dust_columns_int**

bool **output_dust_columns_esc** = true

vector<double> **dust_columns_esc**

bool **output_metal_columns_int** = true

vector<double> **metal_columns_int**

bool **output_metal_columns_esc** = true

vector<double> **metal_columns_esc**

bool **output_HI_columns_int** = true

vector<double> **HI_columns_int**

bool **output_HI_columns_esc** = true

vector<double> **HI_columns_esc**

1.15 Index

- *Index*

INDEX

E

K