# `Illuminator` Distributed Visualization Library

Adam Powell

Version 0.11.0 released March 24, 2008
Built December 14, 2011 on a `unknown` running `linux-gnu`

**Abstract**

`Illuminator` is a library whose current primary purpose is to graph `PETSc` 3-D distributed arrays' isoquant contour surfaces using `Geomview`. It is not at all "complete", rather a first cut. It currently creates the contour surface triangulation in parallel, then sends all of the triangles to the first node for display. Future plans include making the display process more efficient on that first node, and later doing some kind of distributed rendering for faster visualization without the use of `Geomview`. As of version 0.4, `Illuminator` also does distributed storage and retrieval of distributed arrays of any dimensionality, allowing local hard drives on Beowulf cluster nodes to act as a single giant "RAID-0" array with extremely high bandwidth; this can be used for fast retrieval and visualization of time series data.

# Contents


**1 Introduction** **8**

**2 Developer's Manual** **9**
2.1 Visualization . . . 9
2.1.1 Significant bug . . . 9
2.2 Distributed storage . . . 9

**3 Example Programs Using Illuminator** **11**
3.1 3-D function visualizer `3dgf` . . . 11
3.2 Transient Cahn-Hilliard `chts` . . . 11
3.2.1 Cahn-Hilliard user interface `chui` . . . 13

**4 Frequently Asked Questions** **14**
4.1 General install problems . . . 14

**5 Appendix** **15**
A Version History . . . 15
A.1 `PETScGraphics` 0.1 . . . 15
A.2 `PETScGraphics` 0.2 . . . 15
A.3 `PETScGraphics` 0.2.1 . . . 15
A.4 `PETScGraphics` 0.2.2 . . . 15
A.5 `Illuminator` 0.2.3 . . . 15
A.6 `Illuminator` 0.2.4 . . . 15
A.7 `Illuminator` 0.2.5 . . . 15
A.8 `Illuminator` 0.3.0 . . . 16
A.9 `Illuminator` 0.3.1 . . . 16
A.10 `Illuminator` 0.3.2 . . . 16
A.11 `Illuminator` 0.4.0 . . . 16
A.12 `Illuminator` 0.4.1 . . . 16
A.13 `Illuminator` 0.4.2 . . . 16
A.14 `Illuminator` 0.4.3 . . . 16
A.15 `Illuminator` 0.4.4 . . . 16
A.16 `Illuminator` 0.5.0 . . . 16
A.17 `Illuminator` 0.6.0 . . . 17
A.18 `Illuminator` 0.6.1 . . . 17
A.19 `Illuminator` 0.6.2 . . . 17
A.20 `Illuminator` 0.6.9 . . . 17
A.21 `Illuminator` 0.7.0 . . . 17
A.22 `Illuminator` 0.8.0 . . . 17
A.23 `Illuminator` 0.8.9 . . . 17
A.24 `Illuminator` 0.9.0 . . . 17
A.25 `Illuminator` 0.9.1 . . . 17
A.26 `Illuminator` 0.10.0 . . . 17

B Copying . . . 18
C File illuminator.h . . . 18
C.1 Type definitions . . . 19
C.1.1 Typedef ISurface . . . 19
C.1.2 Typedef IDisplay . . . 19
C.1.3 Typedef field_plot_type . . . 19
C.1.4 Typedef IllLayerStyle . . . 19
C.1.5 Typedef IllImageFormat . . . 20
C.2 Functions . . . 20
C.2.1 Local Function DATriangulate() . . . 20
C.2.2 Local Function DATriangulateLocal() . . . 20
D File libluminate/structures.h . . . 20
D.1 Type definitions . . . 21
D.1.1 Type struct isurface . . . 21
D.1.2 Typedef zbuffer . . . 21
D.1.3 Type struct idisplay . . . 21
E File libluminate/illuminator.c . . . 21
E.1 Functions . . . 22
E.1.1 Global Function Draw3DBlock() . . . 22
E.1.2 Global Function DrawHex() . . . 23
E.1.3 Global Function DrawTet() . . . 23
E.1.4 Global Function ISurfDestroy() . . . 23
E.1.5 Global Function SurfClear() . . . 24
E.1.6 Global Function SurfCreate() . . . 24
E.1.7 Local Function DrawTetWithPlane() . . . 24
E.1.8 Local Function storetri() . . . 25
F File libluminate/render.c . . . 25
F.1 Functions . . . 26
F.1.1 Global Function render_composition_path() . . . 26
F.1.2 Global Function render_rgb_local_2d() . . . 26
F.1.3 Global Function render_rgb_local_3d() . . . 27
F.1.4 Global Function render_scale_2d() . . . 28
F.1.5 Local Function pseudocolor() . . . 28
F.1.6 Local Function pseudohueintcolor() . . . 28
F.1.7 Local Function pseudoshearcolor() . . . 28
F.1.8 Local Function pseudoternarycolor() . . . 28
F.1.9 Local Function pseudoternarysquarecolor() . . . 29
G File libluminate/utility.c . . . 29
G.1 Functions . . . 29
G.1.1 Global Function auto_scale() . . . 29
G.1.2 Global Function field_indices() . . . 30
H File libluminate/petsc.c . . . 30
H.1 Functions . . . 31
H.1.1 Global Function DATriangulateLocalRange() . . . 31
H.1.2 Global Function DATriangulateRange() . . . 31
H.1.3 Global Function IllErrorHandler() . . . 32
I File libluminate/illumulti.c . . . 32
I.1 Functions . . . 34
I.1.1 Global Function IlluMultiLoad() . . . 34
I.1.2 Global Function IlluMultiRead() . . . 34
I.1.3 Global Function IlluMultiSave() . . . 35
I.1.4 Local Function IlluMultiParseData() . . . 35
I.1.5 Local Function IlluMultiParseXML() . . . 36
I.1.6 Local Function IlluMultiStoreData() . . . 37

I.1.7 Local Function IlluMultiStoreXML() . . . 38
I.1.8 Local Function checkagree() . . . 39
J File libluminate/importexport.c . . . 39
J.1 Functions . . . 40
J.1.1 Global Function IllImageSave() . . . 40
J.1.2 Local Function IllImageWrite() . . . 40
K File libluminate/geomview.c . . . 41
K.1 Functions . . . 41
K.1.1 Global Function GeomviewBegin() . . . 41
K.1.2 Global Function GeomviewDisplayTriangulation() . . . 42
K.1.3 Global Function GeomviewEnd() . . . 42
L File libluminate/imlib2.c . . . 42
L.1 Functions . . . 43
L.1.1 Global Function imlib2_render_triangles() . . . 43
M File tsview.c . . . 43
M.1 Variables . . . 44
M.1.1 Variable basefilename . . . 44
M.1.2 Local Variables . . . 44
M.2 Functions . . . 44
M.2.1 Global Function PetscSynchronizedFReadline() . . . 44
M.2.2 Global Function main() . . . 44
M.2.3 Global Function myfilter() . . . 45
M.2.4 Global Function rl_gets() . . . 45
N File tsview-ng.c . . . 45
N.1 Type definitions . . . 46
N.1.1 Typedef terntype . . . 46
N.2 Variables . . . 46
N.2.1 Variable xml . . . 46
N.2.2 Variable Surf . . . 46
N.2.3 Variable Disp . . . 46
N.2.4 Variable entrynum . . . 46
N.2.5 Variable total_entries . . . 46
N.2.6 Variable current_timestep . . . 46
N.2.7 Variable the_basename . . . 46
N.2.8 Variable basedirname . . . 47
N.2.9 Variable stepnames . . . 47
N.2.10 Variable current_time . . . 47
N.2.11 Variable log_text . . . 47
N.2.12 Variable ternary_dp_color . . . 47
N.2.13 Variable dp_supp_colors . . . 47
N.2.14 Variable dp_supp_color_points . . . 47
N.2.15 Variable dp_supp_red . . . 47
N.2.16 Variable dp_supp_green . . . 47
N.2.17 Variable dp_supp_blue . . . 47
N.2.18 Variable dp_supp_alpha . . . 47
N.2.19 Variable dp_supp_AB . . . 47
N.2.20 Variable width . . . 47
N.2.21 Variable height . . . 47
N.2.22 Variable bpp . . . 47
N.2.23 Variable scale_size . . . 48
N.2.24 Variable nx . . . 48
N.2.25 Variable ny . . . 48
N.2.26 Variable transform . . . 48
N.2.27 Variable dataview_count . . . 48

N.2.28 Variable dataviews . . . 48
N.2.29 Variable scalar_disp . . . 48
N.2.30 Variable ternary_square_disp . . . 48
N.2.31 Variable ternary_triangle_disp . . . 48
N.2.32 Variable ternary_disp . . . 48
N.2.33 Variable vector_disp . . . 48
N.2.34 Variable shear_disp . . . 48
N.2.35 Variable scalar_auto_set . . . 48
N.2.36 Variable ternary_auto_set . . . 48
N.2.37 Variable vector_auto_set . . . 48
N.2.38 Variable shear_auto_set . . . 49
N.2.39 Variable sizemag . . . 49
N.2.40 Variable transp . . . 49
N.2.41 Variable lastern . . . 49
N.2.42 Variable thistern . . . 49
N.2.43 Variable theda . . . 49
N.2.44 Variable global . . . 49
N.2.45 Variable minmax . . . 49
N.2.46 Variable scales . . . 49
N.2.47 Variable fieldtypes . . . 49
N.2.48 Variable dimensions . . . 49
N.2.49 Variable num_fields . . . 49
N.2.50 Variable current_field . . . 49
N.2.51 Variable field_index . . . 49
N.2.52 Variable num_variables . . . 49
N.2.53 Variable variable_indices . . . 50
N.2.54 Local Variables . . . 50
N.3 Functions . . . 50
N.3.1 Global Function change_ternary() . . . 50
N.3.2 Global Function change_variable() . . . 50
N.3.3 Global Function display_timestep() . . . 50
N.3.4 Global Function main() . . . 50
N.3.5 Global Function my_notime_filter() . . . 50
N.3.6 Global Function my_time_filter() . . . 50
N.3.7 Global Function on_about_activate() . . . 51
N.3.8 Global Function on_log_reload_button_clicked() . . . 51
N.3.9 Global Function on_mag_spin_value_changed() . . . 51
N.3.10 Global Function on_path_filename_entry_activate() . . . 51
N.3.11 Global Function on_plot_area_expose_event() . . . 51
N.3.12 Global Function on_refresh_activate() . . . 51
N.3.13 Global Function on_run_log_activate() . . . 51
N.3.14 Global Function on_save_activate() . . . 51
N.3.15 Global Function on_save_all_activate() . . . 51
N.3.16 Global Function on_scalar_auto_checkbutton_toggled() . . . 51
N.3.17 Global Function on_scalar_max_entry_changed() . . . 51
N.3.18 Global Function on_scalar_min_entry_changed() . . . 51
N.3.19 Global Function on_scalar_scale_area_expose_event() . . . 52
N.3.20 Global Function on_scale_size_entry_activate() . . . 52
N.3.21 Global Function on_shear_auto_checkbutton_toggled() . . . 52
N.3.22 Global Function on_shear_max_entry_changed() . . . 52
N.3.23 Global Function on_shear_scale_area_expose_event() . . . 52
N.3.24 Global Function on_ternary_1A_entry_changed() . . . 52
N.3.25 Global Function on_ternary_1B_entry_changed() . . . 52
N.3.26 Global Function on_ternary_2A_entry_changed() . . . 52

N.3.27 Global Function on_ternary_2B_entry_changed() . . . 52
N.3.28 Global Function on_ternary_3A_entry_changed() . . . 52
N.3.29 Global Function on_ternary_3B_entry_changed() . . . 52
N.3.30 Global Function on_ternary_auto_checkbutton_toggled() . . . 52
N.3.31 Global Function on_ternary_dp_alpha_entry_changed() . . . 52
N.3.32 Global Function on_ternary_dp_blue_entry_changed() . . . 52
N.3.33 Global Function on_ternary_dp_green_entry_changed() . . . 53
N.3.34 Global Function on_ternary_dp_red_entry_changed() . . . 53
N.3.35 Global Function on_ternary_scale_area_expose_event() . . . 53
N.3.36 Global Function on_timestep_spin_value_changed() . . . 53
N.3.37 Global Function on_transform_activate() . . . 53
N.3.38 Global Function on_vector_auto_checkbutton_toggled() . . . 53
N.3.39 Global Function on_vector_max_entry_changed() . . . 53
N.3.40 Global Function on_vector_scale_area_expose_event() . . . 53
N.3.41 Global Function on_vector_symm_spinbutton_changed() . . . 53
N.3.42 Global Function refresh_stepnames() . . . 53
N.3.43 Global Function render_dataviews() . . . 53
N.3.44 Global Function square_to_triangle() . . . 53
N.3.45 Global Function triangle_to_square() . . . 53
O File 3dgf.c . . . 54
O.1 Variables . . . 54
O.1.1 Local Variables . . . 54
O.2 Functions . . . 54
O.2.1 Global Function main() . . . 54
O.2.2 Local Function function_2d() . . . 55
O.2.3 Local Function function_3d() . . . 55
P File chts.c . . . 55
P.1 Variables . . . 56
P.1.1 Variable Surf . . . 56
P.1.2 Variable Disp . . . 56
P.1.3 Local Variables . . . 56
P.2 Functions . . . 56
P.2.1 Global Function FormInitialCondition() . . . 56
P.2.2 Global Function InitializeProblem() . . . 56
P.2.3 Global Function ch_ts_jacobian_2d() . . . 56
P.2.4 Global Function ch_ts_jacobian_3d() . . . 57
P.2.5 Global Function ch_ts_monitor() . . . 57
P.2.6 Global Function ch_ts_residual_vector_2d() . . . 57
P.2.7 Global Function ch_ts_residual_vector_3d() . . . 57
P.2.8 Global Function main() . . . 58
Q File cahnhill.c . . . 58
Q.1 Functions . . . 58
Q.1.1 Global Function ch_jacobian_2d() . . . 58
Q.1.2 Global Function ch_jacobian_3d() . . . 59
Q.1.3 Global Function ch_jacobian_alpha_2d() . . . 59
Q.1.4 Global Function ch_jacobian_alpha_3d() . . . 59
Q.1.5 Global Function ch_residual_vector_2d() . . . 59
Q.1.6 Global Function ch_residual_vector_3d() . . . 59
Q.1.7 Local Function ch_psidoubleprime() . . . 60
Q.1.8 Local Function ch_psiprime() . . . 60
Q.1.9 Local Function ch_residual_2d() . . . 60
Q.1.10 Local Function ch_residual_3d() . . . 61
R File cahnhill.h . . . 62
R.1 Type definitions . . . 62

R.1.1 Typedef AppCtx . . . 62
S File chui.c . . . 63
S.1 Variables . . . 64
S.1.1 Variable xml . . . 64
S.1.2 Variable simulation_input_file . . . 64
S.1.3 Variable simulation_output_file . . . 64
S.1.4 Variable Lx . . . 64
S.1.5 Variable nx . . . 64
S.1.6 Variable dt . . . 64
S.1.7 Variable dt_factor . . . 64
S.1.8 Variable dt_max . . . 64
S.1.9 Variable last_tstep . . . 64
S.1.10 Variable display_x . . . 64
S.1.11 Variable display_text . . . 64
S.1.12 Variable remote_host . . . 64
S.1.13 Variable remote_hostname . . . 64
S.1.14 Variable thetransport . . . 64
S.1.15 Variable mpirun_command . . . 65
S.1.16 Variable number_cpus . . . 65
S.1.17 Variable twodee . . . 65
S.1.18 Variable options_filename . . . 65
S.1.19 Variable pipe_input_tag . . . 65
S.1.20 Variable from_simulation_pipe . . . 65
S.1.21 Variable to_simulation_pipe . . . 65
S.2 Functions . . . 65
S.2.1 Global Function main() . . . 65
S.2.2 Global Function on_2d_activate() . . . 65
S.2.3 Global Function on_3d_activate() . . . 65
S.2.4 Global Function on_about_activate() . . . 65
S.2.5 Global Function on_last_timestep_changed() . . . 65
S.2.6 Global Function on_load_ok_clicked() . . . 65
S.2.7 Global Function on_max_timestep_changed() . . . 65
S.2.8 Global Function on_mpirun_changed() . . . 66
S.2.9 Global Function on_num_cpus_changed() . . . 66
S.2.10 Global Function on_pause_clicked() . . . 66
S.2.11 Global Function on_remote_check_toggled() . . . 66
S.2.12 Global Function on_remote_host_changed() . . . 66
S.2.13 Global Function on_resolution_changed() . . . 66
S.2.14 Global Function on_rsh_item_activate() . . . 66
S.2.15 Global Function on_run_activate() . . . 66
S.2.16 Global Function on_save_ok_clicked() . . . 66
S.2.17 Global Function on_show_options_activate() . . . 66
S.2.18 Global Function on_show_options_toggled() . . . 66
S.2.19 Global Function on_show_output_activate() . . . 66
S.2.20 Global Function on_show_output_toggled() . . . 66
S.2.21 Global Function on_ssh_item_activate() . . . 66
S.2.22 Global Function on_stop_clicked() . . . 66
S.2.23 Global Function on_textdisplay_toggled() . . . 67
S.2.24 Global Function on_time_factor_changed() . . . 67
S.2.25 Global Function on_timestep_changed() . . . 67
S.2.26 Global Function on_width_changed() . . . 67
S.2.27 Global Function on_xdisplay_toggled() . . . 67
S.2.28 Global Function open_params() . . . 67
S.2.29 Global Function read_simulation_data() . . . 67

S.2.30 Global Function save_params() . . . . . 67
S.2.31 Global Function save_params_as() . . . . . 67
T File config.h . . . . . 67

# Chapter 1

# Introduction

PETSc is a great distributed object manager and Newton-Krylov nonlinear system solver. Geomview is a wonderful 3-D surface visualization tool. It seemed logical marry the two, and produce isoquant contour surfaces from PETSc distributed objects in parallel, then display them using Geomview.

This is a first cut attempt at such a project. Each node generates its own isoquant contour surface(s), then all of those generated triangles are gathered to the first node for viewing in Geomview. Right now, it does not do a very efficient job of this: it sends three vertices for each triangle to the first node, which sends those three vertices to Geomview, in plain text format. In the future, it may do a number of things to reduce the number of triangles involved, reduce the number of vertices generated per triangle, and more efficiently transmit the vertices to Geomview.

Long-term, there are other nice 3-D viewers out there such as Data Explorer from IBM, some kind of unified front end to them would be nice. Furthermore, with parallel array storage, much more powerful parallel visualization concepts are possible. Like if we can assign to each point a certain luminosity and transparency, it's easy to integrate those values along a line through the locally-stored part of the array to produce a total transparency and luminosity for that line in the local array, then just combine that with other line integrals through the other local sections (maybe in e-vas?), and we have generated a total image in parallel. However, to do this easily each section must be convex, or there will likely be regions with points both in front and behind a given other region; so this will work with PETSc distributed arrays, but not with general partitioned finite element meshes. But as you can see, this opens many interesting avenues for exploration...

For now, this humble beginning. Share and enjoy.

# Chapter 2

# Developer's Manual

`Illuminator` is intended for developers who are familiar with `PETSc` and want to visualize distributed arrays, and as such, this manual assumes knowledge of `PETSc` programming.
For more details, refer to the complete source code reference which begins at appendix C.

## 2.1 Visualization

With a bit of `PETSc` programming knowledge, using `Illuminator` is pretty simple. Start by invoking `GeomviewBegin()` (appendix **??**) to open the `Geomview` display and the pipes to and from `Geomview`.
At an appropriate checkpoint in the program, invoke `DATriangulate()`, or for a local array `DATriangulateLocal()`, or else their `DATriangulate(Local)Range()` variants (appendices **??** and **??**), to create the distributed triangulation on each processor. When that is complete, the function `GeomviewDisplayTriangulation()` (appendix **??**) gathers the triangulations to node 0, sends them to `Geomview` and resets the triangle counts to zero. This cycle of `DATriangulate()` and `GeomviewDisplayTriangulation()` may be repeated as necessary, subsequent cycles simply replace the existing triangle set in `Geomview` with a new set of triangles. In the example program `chts`, this is done in the function `ch_ts_monitor()` (appendix P.2.5) whose function pointer is passed to `PETSc`'s `TS_Set_Monitor()` so it displays the contour surfaces at the end of every timestep calculation.
Finally, when you are done, call `GeomviewEnd()` (appendix **??**) to exit the `Geomview` process and close its pipes.

### 2.1.1 Significant bug

If an isoquant surface happens to exactly intersect one or more of the vertices, the triangles on the adjacent tetrahedra may be generated with coordinates "`nan nan nan`". This is a problem, and I'll try to figure out a solution at some point. In the meantime, the workaround is to choose a slightly different isoquant surface value to avoid the intersection (e.g. if 1000 intersects a vertex, then try 999.999).

## 2.2 Distributed storage

Because 3-D timestep data can be quite voluminous, `Illuminator` comes with functions for distributed loading and storage of data. That is, if you have a Beowulf cluster with a hard disk on each node, why send all of the data to the head node to store it? That would be a waste of bandwidth, and would neglect the available local hard drive storage!
Instead, you can use Illuminator's function `IlluMultiSave()` (appendix **??**) to save the local data to a local hard drive. This will create two files for each CPU, `{basename}.cpu####.meta` which contains XML-formatted metadata describing that CPU's data, and `{basename}.cpu####.data`, which contains the data stored in the distributed array (`####` is the CPU number). The data can be compressed using `gzip` and/or by converting floating point values to 32-, 16- or 8-bit integers scaled to the minimum and maximum values of each field. The function `IlluMultiLoad()` (appendix **??**) creates a new distributed array based on the

stored data, and `IlluMultiRead()` (appendix **??**) reads the stored data into an existing distributed array and vector, after verifying a match between existing and stored distributed array parameters.
It is also possible to save a distributed array stored on multiple CPUs, and load/read it back into a single CPU, `IlluMultiLoad()` and `IlluMultiRead()` automatically rearrange the data at load time accordingly. At some point, it may be possible to load to arbitrary numbers of CPUs, but for now, only $n \to n$ and $n \to 1$ storage/loading are supported.
`IlluMultiSave()` saves in the native byte order of the CPU on which it is compiled, and `IlluMultiLoad()` and `IlluMultiRead()` will automatically byte-swap if stored data doesn't match the CPU's endianness.

# Chapter 3

# Example Programs Using Illuminator

Two simple example programs using the `Illuminator` library are provided: function visualizer `3dgf`, and transient Cahn-Hilliard `chts` (and its GNOME/Glade front-end `chui`).

## 3.1 3-D function visualizer `3dgf`

The 3-D function visualizer in `3dgf.c` (appendix O) currently displays Green's function data in three dimensions using the `Geomview` frontend. (Without `Geomview` installed, `3dgf` will not build.) It includes two modes of operation: default contour surfaces for a 3-D function $f(x, y, z) = C$ provided in `function_3d()` (appendix O.2.3), if `-twodee` is specified on the command line then height vs. 2-D space for a 2-D function $z = f(x, y)$ provided in `function_2d()` (appendix O.2.2). Note there is no default box size, one must always use the `-da_grid_x`, `_y` and `_z` options to define this.

## 3.2 Transient Cahn-Hilliard `chts`

This example program provided with `Illuminator` solves the Cahn-Hilliard equation in 3-D. The implementation is split into files `cahnhill.c` (appendix Q) containing all of the free energy functions, derivatives, etc.; and `chts.c` (appendix P) with `main()`, initialization, callbacks—essentially, the overhead of the program; and a small header with the data structure in `cahnhill.h` (appendix R). The calls to `Illuminator` library functions are contained in `chts.c`.
The idea behind Cahn-Hilliard is that we have a concentration field $C$, with associated thermodynamic free energy density $f$ given by

$$f = \frac{\alpha}{2} |\nabla C|^2 + \beta \Psi(C), \tag{3.1}$$

where the two terms are the gradient penalty and homogeneous free energy $\Psi$, and $\alpha$ and $\beta$ are model constants. This functional form was first used to explain the finite domain size during the initial phase separation in spinodal decomposition. A simple polynomial homogeneous free energy function (the one used in `cahnhill.c`) looks like

$$\Psi = C^2(1 - C)^2. \tag{3.2}$$

This functional form gives two stable equilibria (minima) at $C = 0$ and $C = 1$, and an unstable equilibrium (maximum) at $C = \frac{1}{2}$.
The total free energy of the system occupying the body $\Omega$ is

$$\mathcal{F} = \int_\Omega f dV. \tag{3.3}$$

The "chemical" potential $\mu$ is given by

$$\mu = \frac{\delta \mathcal{F}}{\delta C} = -\alpha \nabla^2 C + \beta \Psi'(C), \tag{3.4}$$

which leads to the transport equation

$$\frac{\partial C}{\partial t} = \nabla \cdot (\kappa \nabla \mu) \tag{3.5}$$

where $\kappa$ is the mobility.
The interface thickness $\epsilon$ is on the order of $\sqrt{\alpha/\beta}$, and the surface energy $\gamma$ on the order of $\sqrt{\alpha\beta}$, so we can set $\alpha = \gamma\epsilon$ and $\beta = \gamma/\epsilon$, with constants which we'll worry about later.
Returning to equation 3.5, it expands to

$$\frac{\partial C}{\partial t} = \nabla\kappa \cdot \nabla\mu + \kappa\nabla^2\mu \tag{3.6}$$

and further in terms of $C$ to

$$\frac{\partial C}{\partial t} = \nabla\kappa \cdot \nabla(-\alpha\nabla^2 C + \beta\Psi') - \kappa\nabla^2(\alpha\nabla^2 C + \beta\nabla^2\Psi'(C)) \tag{3.7}$$

Now we turn to the `PETSc` implementation, using distributed arrays and `TS` timestepping solvers. The nonlinear timestepping code uses runtime-selectable timestepping solvers (explicit, implicit, or default Crank-Nicholson) to solve the equation

$$\frac{\partial u_i}{\partial t} = F_i(u_j), \tag{3.8}$$

where $u_i$ is the vector of unknowns (here concentrations). This is converted by `PETSc` into a system of nonlinear equations according to the solver type and then solved using its `SNES` solvers, but we must provide subroutines to calculate $F_i$ and the derivatives $\frac{\partial F_i}{\partial u_j}$ which are used to calculate the Jacobian of the nonlinear system (unless running matrix-free using the `-snes_mf` option). Starting for now with constant $\kappa$, $\alpha$ and $\beta$, the $F_i$ are given by

$$F_i = \kappa\left(-\alpha\nabla^2\nabla^2 C + \beta\nabla^2\Psi'(C)\right) \tag{3.9}$$

Let's suppose $C$ is on a 2-D finite difference mesh with uniform but possibly different spacings $h_x$ and $h_y$ in the $x$- and $y$-directions, so we'll let $C_{x,y}$ be the value of $C$ at coordinates $(xh_x, yh_y)$ for integer $x$ and $y$.
Starting with the $\beta$ term, the Laplacian $\nabla^2\Psi'(C)$ at $(x, y)$ can be approximated using the standard 5-node finite difference stencil:

$$\nabla^2\Psi'(C) \simeq \frac{\Psi'(C_{x-1,y}) - 2\Psi'(C_{x,y}) + \Psi'(C_{x+1,y})}{h_x^2} + \frac{\Psi'(C_{x,y-1}) - 2\Psi'(C_{x,y}) + \Psi'(C_{x,y+1})}{h_y^2} \tag{3.10}$$

or expressed slightly differently, as the sum of the terms:

$$\begin{array}{ccc} & \Psi'(C_{x,y+1})h_y^{-2} & \\ \Psi'(C_{x-1,y})h_x^{-2} & \Psi'(C_{x,y})(-2h_x^{-2} - 2h_y^{-2}) & \Psi'(C_{x+1,y})h_x^{-2} \\ & \Psi'(C_{x,y-1})h_y^{-2} & \end{array} \tag{3.11}$$

So the product of $\kappa\beta$ and this is one part of the function $F_i$, and that part of the derivative $\frac{\partial F_i}{\partial C_j}$ for the Jacobian is simply the appropriate coefficient from equation 3.11 times the second derivative $\Psi''(C)$ evaluated at the appropriate point.
For the $\alpha$-term, the Laplacian of the Laplacian (also known as the *biharmonic operator*) is a bit more messy. Using the notation in equation 3.11 but only in the first quadrant (the coefficients are symmetric), the term will be $-\kappa\alpha$ times:

$$\begin{array}{ccccc} & & C_{x,y+2}h_y^{-4} & & \\ & \ldots & C_{x,y+1}(-4h_y^{-4} - 4h_x^{-2}h_y^{-2}) & 2C_{x+1,y+1}h_x^{-2}h_y^{-2} & \\ \ldots & \ldots & C_{x,y}(6h_x^{-4} + 6h_y^{-4} + 8h_x^{-2}h_y^{-2}) & C_{x+1,y}(-4h_x^{-4} - 4h_x^{-2}h_y^{-2}) & C_{x+2,y}h_x^{-4} \\ & \ldots & \ldots & \ldots & \\ & & \ldots & & \end{array} \tag{3.12}$$

(ellipsis indicates symmetry). This is quite a bit more complicated, but at least it's linear so the derivatives $\frac{\partial F_i}{\partial C_j}$ are constant.

These $F_i$ functions are calculated by the function `ch_residual_2d()` (appendix Q.1.9) and assembled into a `PETSc` vector in `ch_residual_vector_2d()` (appendix Q.1.5). The derivatives are calculated in two parts: the $\alpha$ term's derivative matrix is built at the start of the run in `ch_jacobian_alpha_2d()` (appendix Q.1.3), and with each nonlinear iteration, the $\beta$ term's derivative matrix is added to that in `ch_jacobian_2d()` (appendix Q.1.1).
Note that this is all in 2-D; the 3-D version is left as an exercise to the reader, though it's already coded in the corresponding `_3d` functions in `cahnhill.c`.

### 3.2.1 Cahn-Hilliard user interface `chui`

The Cahn-Hilliard User Interface, or CHUI, is a simple prototype Glade front-end to `chts` in a single C program file `chui.c` (appendix S) and Glade interface description file `chui.glade`. It is included here as a demonstration/example rich front end to `PETSc` programs with simultaneous `Illuminator` visualization. As such, it includes a number of standard options such as size of the simulation and properties, controls the simultaneous visualization options, and also permits remote running, optionally in multiple processes on a multi-processor machine or cluster. Finally, it includes multiple simultaneous progress bars to track progress of the linear and nonlinear solvers and timesteps.

# Chapter 4

# Frequently Asked Questions

## 4.1 General install problems

**Q: "`make dist`" doesn't work. What's up with that?**

A: That's because automake 1.4 doesn't like foreign variables from included makefiles, like `PETSC_DM_LIB` used here. For this reason, illuminator through 0.3.0 had targets mydistdir, mydist and mydistcheck in Makefile.am, use those instead of distdir, dist and distcheck. For 0.3.1 and beyond, you must use automake 1.5 or higher.

# Chapter 5

# Appendix

## A Version History

A brief summary of information in the `ChangeLog` file.

### A.1 `PETScGraphics` 0.1

The first release, this included basic functionality of making and displaying multiple transparent contour surfaces in 3-D, but only in uniprocessor operation.

### A.2 `PETScGraphics` 0.2

This release included the first truly parallel visualization, including gathering of the triangles to node 1 for viewing. It also included lots of code cleanup, changes from the `PETSc` coding style to something more like GNU format. And it added the `-random` flag to the chts test program for random initialization.

### A.3 `PETScGraphics` 0.2.1

This was largely a documentation update to the new 0.2 interface, reflecting changes to the new version.

### A.4 `PETScGraphics` 0.2.2

More documentation updates, esp. for the example program; minor changes to support gcc on Alpha processors; chts -twodee now works.

### A.5 `Illuminator` 0.2.3

Name change, as PETSc may be one of several backends in the future. Shuffled some functions around to petsc.c and geomview.c.

### A.6 `Illuminator` 0.2.4

Updated to work with PETSc 2.1.1.

### A.7 `Illuminator` 0.2.5

Minor fixes: loops in DATriangulate now work for non-periodic distributed arrays; changed VecMin() and VecMax() to VecStrideMin() and VecStrideMax() to properly obtain the minimum and maximum value of the target field.

### A.8 `Illuminator` 0.3.0

A major new addition is the `chui` (Cahn-Hilliard User Interface) program, this is not quite complete but illustrates what can be done with a bit of `libglade`. The goal is eventually to use this kind of thing and `Illuminator` to provide an interactive distributed simulation.

### A.9 `Illuminator` 0.3.1

A bugfix release, among other things this gets chui.glade into the distribution (which sort of helps), and requires automake 1.5 or above.

### A.10 `Illuminator` 0.3.2

Another bugfix release, changed Makefile.am to remove static libs/objects from shared libluminate to fix building on PIC-sensitive platforms.

### A.11 `Illuminator` 0.4.0

Major addition: the `IlluMulti` system for extreme performance distributed storage of PETSc distributed arrays on local disks of Beowulf cluster nodes. This is intended to enable rapid (real-time?) distributed creation of visualization movies from distributed timestep data stored on the local drives, essentially turning those hard drives into a giant RAID-0 array. For details, see the source-embedded documentation in appendix ??.

### A.12 `Illuminator` 0.4.1

Primarily a bugfix release (since 0.4.0 didn't compile out of the box), this version also adds the `3dgf` program, designed to make it very easy to visualize the isoquant surfaces of a function in 3-D. As the name implies, it has a couple of simple Green's functions in it now, and the distributed nature of the function computation lends it to efficient parallel computation and visualization of more complex Green's functions. At some point this should have a `chui`-like `libglade`-based interface to make interactive modification of isoquants, colors, limits, resolution, function parameters and such a lot easier.

### A.13 `Illuminator` 0.4.2

One bugfix: `illuminator.m4` now installs. Added `debian` directory to make future upgrades package more easily.

### A.14 `Illuminator` 0.4.3

Subtle bugfixes for `IlluMultiLoad()` and `IlluMultiRead()`, needed for large $n \to 1$ redistributions, and one bugfix for `GeomviewDisplayTriangulation()`.

### A.15 `Illuminator` 0.4.4

Subtle bugfix for `IlluMultiParseXML()`; added `tsview` program for generic viewing of timestep data (just 2-D for now; moved from `RheoPlast`). Also added a `-with-geomview=` configuration option.

### A.16 `Illuminator` 0.5.0

New `DATriangulateLocal()` function should save time when the local array is available; transparency option in `GeomviewDisplayTriangulation()`; tsview supports 3-D, with ability to cycle through displayed field and turn transparency on and off at will.

### A.17 `Illuminator` 0.6.0

Two new attributes added to IlluMulti file format, now version 0.2: field visualization types to tag special scalar, vector, and even tensor fields; and floating-point size of the "physical" array in each direction. Support added to `tsview` for physical size in 2-D (was already there in 3-D).

### A.18 `Illuminator` 0.6.1

New field visualization types, small change to 0.2 IlluMulti file format, little bug fixes. (This is a "brown paper bag" release.)

### A.19 `Illuminator` 0.6.2

Bug fixes for `tsview` viewer: non-square/cube geometry, "v" command to change fields in 3-D, and field name for 3-D all now work properly.

### A.20 `Illuminator` 0.6.9

New `tsview` commands: "s" (size `PetscViewer` windows in 2-D) and "p" (set plot contours in 3-D). Also new HeVeA LaTeX to HTML translator option, with updates to much of the documentation build system. And a preview of the new tsview called tsview-ng, which is sufficiently premature that it is not included in the Debian package, but will be a substantial new addition to 0.7.0 (hence 0.6.9 for this version).

### A.21 `Illuminator` 0.7.0

A private release, with new `DATriangulateLocal` function, new cut options for `DATriangulate(Local)`, three new tsview commands, readline support in tsview, and a new `-vector_max` option to `tsview-ng` to scale the maximum vector size.

### A.22 `Illuminator` 0.8.0

Upgrade `chui` and `tsview-ng` to gtk+/GNOME version 2 libraries, including new `.desktop` entries. The recommended 2-D viewer is now `tsview-ng`.

### A.23 `Illuminator` 0.8.9

Fix for `tsview-ng` so it can really be recommended. Moved 2-D rendering from `tsview-ng` into the library, and added a prototype for 3-D (planned to work in 0.9). New variations on `DATriangulate(Local)` create triangles for just a range of grid points.

### A.24 `Illuminator` 0.9.0

New imlib2-based 3-D rendering engine, 2-D shear rendering, and and many enhancements to `tsview-ng` including manual scaling with scale figures and rudimentary 3-D.

### A.25 `Illuminator` 0.9.1

2-D ternary includes a "square" option for composition mapped onto a rectangle rather than a standard triangle.

### A.26 `Illuminator` 0.10.0

New data classes encapsulate triangle data (`ISurface`) and display methods (`IDisplay`) for better abstraction from the application. Both `Geomview` pipe and bitmap images can be written to `.ppm` files, and this is exposed via the user interfaces of `tsview` and `tsview-ng`. 2-D rendering has new transforms, and both square and triangle ternary plots can generate diffusion path images as well in any color; these are displayed in the `tsview-ng` scale area. `tsview-ng` can now view the Run log and support non-timeseries data.

# B Copying

`Illuminator` Distributed Visualization Library

You may contact the corresponding author by email at `hazelsct@mit.edu`.

# C File illuminator.h

### RCS Header

This is the interface for the Illuminator library.

## Included Files

```
#include </usr/lib/petsc/include/petscda.h>
```

```
#include </usr/include/glib-2.0/glib.h>
```

```
#include </usr/include/gtk-2.0/gtk/gtk.h>
```

```
#include <X11/Xlib.h>
```

```
#include <Imlib2.h>
```

## Preprocessor definitions

```
#define ORBIT2 1
```

```
#define _REENTRANT 1
```

```
#define IMLIB2_EXISTS 1
```

```
#define ILLUMINATOR_H
```

```
#define FLIP_HORIZONTAL 0x01
```

```
#define FLIP_VERTICAL 0x02
```

```
#define ROTATE_LEFT 0x04
```

```
#define COMPRESS_INT_MASK 0x30
```

```
#define COMPRESS_INT_NONE 0x00
```

```
#define COMPRESS_INT_LONG 0x10
```

```
#define COMPRESS_INT_SHORT 0x20
```

```
#define COMPRESS_INT_CHAR 0x30
```

```
#define COMPRESS_GZIP_MASK 0x0F

#define COMPRESS_GZIP_NONE 0x00

#define COMPRESS_GZIP_FAST 0x01

#define COMPRESS_GZIP_BEST 0x0A
```

## C.1 Type definitions

### C.1.1 Typedef ISurface

The `ISurface` type is the container (or object class) for triangle data which illuminator will render.

```
typedef void* ISurface
```

### C.1.2 Typedef IDisplay

The `IDisplay` type is the container for display data, such as the geomview output pipe, RGB buffer, multi-layer z-buffer, etc.

```
typedef void* IDisplay
```

### C.1.3 Typedef field_plot_type

A value of `field_plot_type` is attached to each field in a simulation in order to visualize them properly. Types are as follows:

```
typedef enum {...} field_plot_type
```

```
enum
 {
```

| | |
|---|---|
| `FIELD_SCALAR;` | Scalar field. |
| `FIELD_TERNARY;` | Ternary composition field with two components (third component is inferred from first two). |
| `FIELD_TERNARY_SQUARE;` | Ternary composition with pseudo-components mapping onto a rectangle instead of a triangle. |
| `FIELD_VECTOR;` | Vector field. |
| `FIELD_TENSOR_FULL;` | Full ds*ds tensor field, e.g. transformation. |
| `FIELD_TENSOR_SYMMETRIC;` | Symmetric tensor field (using lines in principal stress directions). |
| `FIELD_TENSOR_SHEAR;` | Shear tensor field, both symmetric and inferring last diagonal from the opposite of the sum of the others. |

```
 }
```

### C.1.4 Typedef IllLayerStyle

```
typedef enum {...} IllLayerStyle
```

```
enum
 {
   DEFAULT;
   X_UP;
   X_DOWN;
   Y_UP;
   Y_DOWN;
   Z_UP;
   Z_DOWN;
 }
```

### C.1.5 Typedef IllImageFormat

```
typedef enum {...} IllImageFormat
enum
 {
   PPM;
   TIFF;
   PNG;
   JPEG;
   GIF;
 }
```

## C.2 Functions

### C.2.1 Local Function DATriangulate()

```
static inline int DATriangulate ( ISurface Surf, DA theda, Vec globalX, int this, PetscScalar*
minmax, int n_quants, PetscScalar* isoquants, PetscScalar* colors, PetscTruth xcut, PetscTruth
ycut, PetscTruth zcut )
```

### C.2.2 Local Function DATriangulateLocal()

```
static inline int DATriangulateLocal ( ISurface Surf, DA theda, Vec localX, int this, PetscScalar*
minmax, int n_quants, PetscScalar* isoquants, PetscScalar* colors, PetscTruth xcut, PetscTruth
ycut, PetscTruth zcut )
```

# D File libluminate/structures.h

### RCS Header

This file defines the isurface and idisplay structures which map to the ISurface and IDisplay data types in illuminator.h. It also contains the zbuffer data type for multiz.c.

## Included Files

```
#include <illuminator.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define SURFACE_H
```

## D.1 Type definitions

### D.1.1 Type struct isurface

```
struct isurface
 {
   int num_triangles;
   int vertisize;
   PetscScalar* vertices;
 }
```

Number of triangles in this node. Its value is initialized to zero, and incremented as each triangle is added, then reset to zero when the triangulation is displayed.

Number of triangles allocated in the vertices array.

Array of vertex corners of triangles. The number of triangles is given by num_triangles, and size of the array by vertisize. For each triangle, this array has the coordinates of the three nodes, and its R, G, B and A color values, hence 13 PetscScalars for each triangle.

### D.1.2 Typedef zbuffer

The zbuffer type, used to store an entry for a pixel.

```
typedef struct {...} zbuffer
struct
 {
   guchar r;
   guchar g;
   guchar b;
   guchar a;
   float z;
 }
```

### D.1.3 Type struct idisplay

```
struct idisplay
 {
   FILE* to_geomview;
   guchar* rgb;
   int rgb_width;
   int rgb_height;
   int rgb_rowskip;
   int rgb_bpp;
   zbuffer* zbuf;
   int zbuf_width;
   int zbuf_height;
   int zbuf_rowskip;
   int zbuf_depth;
 }
```

Geomview output pipe

# E File libluminate/illuminator.c

### RCS Header

This is the illuminator.c main file. It has all of the routines which compute the triangulation in a distributed way.

## Included Files

```
#include "config.h"                                                    (Section T)

#include "illuminator.h"                                               (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>

#include "libluminate/structures.h"                                    (Section D)
   #include <illuminator.h>

#include <stdlib.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt, args...  )

#define MAX_TRIANGLES 10000000

#define __FUNCT__ "ISurfCreate"

#define __FUNCT__ "ISurfDestroy"

#define __FUNCT__ "ISurfClear"

#define __FUNCT__ "storetri"

#define __FUNCT__ "DrawTetWithPlane"

#define COORD( c1, c2, index )

#define __FUNCT__ "DrawTet"

#define __FUNCT__ "DrawHex"

#define __FUNCT__ "Draw3DBlock"
```

## E.1 Functions

### E.1.1 Global Function Draw3DBlock()

Calculate vertices of isoquant triangle(s) in a 3-D regular array of right regular hexahedra. This loops through a 3-D array and calls DrawHex to calculate the triangulation of each hexahedral cell.

`int Draw3DBlock ( ISurface Surf, int xd, int yd, int zd, int xs, int ys, int zs, int xm, int ym, int zm, PetscScalar* minmax, PetscScalar* vals, int skip, int n_quants, PetscScalar* isoquants, PetscScalar* colors )`

- `int Draw3DBlock` Returns 0 or an error code.
- `ISurface Surf` ISurface object into which to draw this block's isoquants.
- `int xd` Overall x-width of function value array.
- `int yd` Overall y-width of function value array.
- `int zd` Overall z-width of function value array.
- `int xs` X-index of the start of the array section we'd like to draw.

- `int ys` — Y-index of the start of the array section we'd like to draw.
- `int zs` — Z-index of the start of the array section we'd like to draw.
- `int xm` — X-width of the array section we'd like to draw.
- `int ym` — Y-width of the array section we'd like to draw.
- `int zm` — Z-width of the array section we'd like to draw.
- `PetscScalar* minmax` — Position of block corners: xmin, xmax, ymin, ymax, zmin, zmax.
- `PetscScalar* vals` — The array of function values at vertices.
- `int skip` — Number of interlaced fields in this array.
- `int n_quants` — Number of isoquant surfaces to draw (isoquant values).
- `PetscScalar* isoquants` — Array of function values at which to draw triangles.
- `PetscScalar* colors` — Array of color R,G,B,A quads for each isoquant.

### E.1.2 Global Function DrawHex()

This divides a right regular hexahedron into tetrahedra, and loops over them to generate triangles on each one. It calculates edge and whichplane parameters so it can use DrawTetWithPlane directly.

```
int DrawHex ( ISurface Surf, PetscScalar* coords, PetscScalar* vals, PetscScalar isoquant,
PetscScalar* color )
```

- `int DrawHex` — Returns 0 or an error code.
- `ISurface Surf` — ISurface object into which to draw this hexahedron's isoquant.
- `PetscScalar* coords` — Coordinates of hexahedron corner points: xmin, xmax, ymin, etc.
- `PetscScalar* vals` — Function values at hexahedron corners: f0, f1, f2, etc.
- `PetscScalar isoquant` — Function value at which to draw triangles.
- `PetscScalar* color` — R,G,B,A quad for this hexahedron.

### E.1.3 Global Function DrawTet()

This sets the edge and whichplane parameters and then passes everything to DrawTetWithPlane to actually draw the triangle. It is intended for use by developers with distributed arrays based on tetrahedra, e.g. a finite element mesh.

```
int DrawTet ( ISurface Surf, PetscScalar* coords, PetscScalar* vals, PetscScalar isoquant,
PetscScalar* color )
```

- `int DrawTet` — Returns 0 or an error code.
- `ISurface Surf` — ISurface object into which to draw this tetrahedron's isoquant.
- `PetscScalar* coords` — Coordinates of tetrahedron corner points: x0, y0, z0, x1, etc.
- `PetscScalar* vals` — Function values at tetrahedron corners: f0, f1, f2, f3.
- `PetscScalar isoquant` — Function value at which to draw triangle.
- `PetscScalar* color` — R,G,B,A quad for this tetrahedron.

### E.1.4 Global Function ISurfDestroy()

Destructor for ISurface data object.

```
int ISurfDestroy ( ISurface Surf )
```

- `int ISurfDestroy` — Returns zero or an error code.
- `ISurface Surf` — ISurface object to destroy.

### E.1.5 Global Function SurfClear()

Clear out an isurface data object without freeing its memory.

`int SurfClear ( ISurface Surf )`

- `int SurfClear` Returns zero or an error code.
- `ISurface Surf` ISurface object to clear out.

### E.1.6 Global Function SurfCreate()

Constructor for ISurface data object.

`int SurfCreate ( ISurface* newsurf )`

- `int SurfCreate` Returns zero or an error code.

Surf *newsurf Address in which to put the new ISurface object.

### E.1.7 Local Function DrawTetWithPlane()

This function calculates triangle vertices for an isoquant surface in a linear tetrahedron, using the whichplane information supplied by the routine calling this one, and "draws" them using storetri(). This is really an internal function, not intended to be called by user programs. It is used by DrawTet() and DrawHex().

`static inline int DrawTetWithPlane ( PetscScalar x0, PetscScalar y0, PetscScalar z0, PetscScalar f0, PetscScalar x1, PetscScalar y1, PetscScalar z1, PetscScalar f1, PetscScalar x2, PetscScalar y2, PetscScalar z2, PetscScalar f2, PetscScalar x3, PetscScalar y3, PetscScalar z3, PetscScalar f3, PetscScalar isoquant, PetscScalar edge0, PetscScalar edge1, PetscScalar edge3, int whichplane, PetscScalar* color, struct isurface* thesurf )`

- `int DrawTetWithPlane` Returns 0 or an error code.
- `PetscScalar x0` X-coordinate of vertex 0.
- `PetscScalar y0` Y-coordinate of vertex 0.
- `PetscScalar z0` Z-coordinate of vertex 0.
- `PetscScalar f0` Function value at vertex 0.
- `PetscScalar x1` X-coordinate of vertex 1.
- `PetscScalar y1` Y-coordinate of vertex 1.
- `PetscScalar z1` Z-coordinate of vertex 1.
- `PetscScalar f1` Function value at vertex 1.
- `PetscScalar x2` X-coordinate of vertex 2.
- `PetscScalar y2` Y-coordinate of vertex 2.
- `PetscScalar z2` Z-coordinate of vertex 2.
- `PetscScalar f2` Function value at vertex 2.
- `PetscScalar x3` X-coordinate of vertex 3.
- `PetscScalar y3` Y-coordinate of vertex 3.
- `PetscScalar z3` Z-coordinate of vertex 3.
- `PetscScalar f3` Function value at vertex 3.
- `PetscScalar isoquant` Function value at which to draw triangle.
- `PetscScalar edge0` Normalized intercept at edge 0, 0. at node 0, 1. at node 1.
- `PetscScalar edge1` Normalized intercept at edge 1, 0. at node 1, 1. at node 2.
- `PetscScalar edge3` Normalized intercept at edge 3, 0. at node 0, 1. at node 3.
- `int whichplane` Index of which edge intercept(s) is between zero and 1.

- `PetscScalar* color` R,G,B,A quad for this tetrahedron.
- `struct isurface* thesurf` ISurface object to put the triangles in.

### E.1.8 Local Function storetri()

This little inline routine just implements triangle storage.

```
static inline int storetri ( PetscScalar x0, PetscScalar y0, PetscScalar z0, PetscScalar x1,
PetscScalar y1, PetscScalar z1, PetscScalar x2, PetscScalar y2, PetscScalar z2, PetscScalar* color,
struct isurface* thesurf )
```

- `int storetri` Returns 0 or an error code.
- `PetscScalar x0` X-coordinate of corner 0.
- `PetscScalar y0` Y-coordinate of corner 0.
- `PetscScalar z0` Z-coordinate of corner 0.
- `PetscScalar x1` X-coordinate of corner 1.
- `PetscScalar y1` Y-coordinate of corner 1.
- `PetscScalar z1` Z-coordinate of corner 1.
- `PetscScalar x2` X-coordinate of corner 2.
- `PetscScalar y2` Y-coordinate of corner 2.
- `PetscScalar z2` Z-coordinate of corner 2.
- `PetscScalar* color` R,G,B,A quad for this triangle.
- `struct isurface* thesurf` ISurface object to put the triangles in.

# F File libluminate/render.c

### RCS Header

This file contains the rendering code for Illuminator, which renders 2-D or 3-D data into an RGB(A) unsigned char array (using perspective in 3-D).

## Included Files

```
#include "illuminator.h"                                                    (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>

#include "libluminate/structures.h"                                         (Section D)
   #include <illuminator.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt, args...  )

#define __FUNCT__ "pseudocolor"

#define __FUNCT__ "pseudoternarycolor"
```

```
#define __FUNCT__ "pseudoternarysquarecolor"
#define __FUNCT__ "pseudohueintcolor"
#define __FUNCT__ "pseudoshearcolor"
#define __FUNCT__ "render_scale_2d"
#define __FUNCT__ "render_composition_path"
#define __FUNCT__ "render_rgb_local_2d"
#define __FUNCT__ "render_rgb_local_3d"
```

## F.1 Functions

### F.1.1 Global Function render_composition_path()

Render a composition path, such as a diffusion path or phase diagram, onto an IDisplay image.

`int render_composition_path ( IDisplay Disp, PetscScalar* comp_array, int gridpoints, int num_fields, field_plot_type fieldtype, PetscScalar* scale, PetscScalar red, PetscScalar green, PetscScalar blue, PetscScalar alpha )`

- `int render_composition_path` Returns zero or an error code.
- `IDisplay Disp` IDisplay object into which to draw the path.
- `PetscScalar* comp_array`
- `int gridpoints` Number of gridpoints in the composition array.
- `int num_fields` Total number of fields in the composition array.
- `field_plot_type fieldtype` The type of this field.
- `PetscScalar* scale` Array of minimum and maximum values to pass to the various pseudocolor functions; if NULL, call auto_scale to determine those values.
- `PetscScalar red` Red color for composition path.
- `PetscScalar green` Green color for composition path.
- `PetscScalar blue` Blue color for composition path.
- `PetscScalar alpha` Alpha channel for composition path.

PetscScalar comp_array Array of compositions for drawing.

### F.1.2 Global Function render_rgb_local_2d()

Render data from global_array into local part of an RGB buffer. When running in parallel, these local buffers should be collected and layered to produce the full image.

`int render_rgb_local_2d ( IDisplay Disp, PetscScalar* global_array, int num_fields, int display_field, field_plot_type fieldtype, PetscScalar* scale, int nx, int ny, int xs, int ys, int xm, int ym, int transform, IDisplay SDisp, PetscScalar dpred, PetscScalar dpgreen, PetscScalar dpblue, PetscScalar dpalpha )`

- `int render_rgb_local_2d` Returns zero or an error code.
- `IDisplay Disp` Display object holding the RGB buffer and its characteristics.
- `PetscScalar* global_array` Local array of global vector data to render.
- `int num_fields` Number of field variables in the array.
- `int display_field` The (first) field we are rendering now.
- `field_plot_type fieldtype` The type of this field.
- `PetscScalar* scale` Array of minimum and maximum values to pass to the various pseudocolor functions; if NULL, call auto_scale to determine those values.

- `int nx` — Width of the array.
- `int ny` — Height of the array.
- `int xs` — Starting $x$-coordinate of the local part of the global vector.
- `int ys` — Starting $y$-coordinate of the local part of the global vector.
- `int xm` — Width of the local part of the global vector.
- `int ym` — Height of the local part of the global vector.
- `int transform` — Transformation flags.
- `IDisplay SDisp` — Display object in which to draw the diffusion path (optional, ignored if NULL).
- `PetscScalar dpred` — Red color for diffusion path points (0-1).
- `PetscScalar dpgreen` — Green color for diffusion path points (0-1).
- `PetscScalar dpblue` — Blue color for diffusion path points (0-1).
- `PetscScalar dpalpha` — Alpha channel for diffusion path points (0-1).

### F.1.3 Global Function render_rgb_local_3d()

Render triangle data into an RGB buffer. When called in parallel, the resulting images should be layered to give the complete picture. Zooming is done by adjusting the ratio of the dir vector to the right vector.
The coordinate transformation is pretty simple. A point $\vec{p}$ in space is transformed to $x, y$ on the screen by representing it as:

$$\vec{p} = \vec{\imath} + a\vec{d} + ax\vec{r} + ay\vec{u}, \tag{5.1}$$

where $\vec{\imath}$ is the observer's point (passed in as `eye`), $\vec{d}$ is the direction of observation (passed in as `dir`), $\vec{r}$ is the rightward direction to the observer (passed in as `right`), and $\vec{u}$ is the upward direction to the observer which is given the direction of $\vec{r} \times \vec{d}$ and the magnitude of $\vec{r}$.
This system in equation 5.1 can easily be solved for $x$ and $y$ by first making it into a matrix equation:

$$\begin{pmatrix} d_x & r_x & u_x \\ d_y & r_y & u_y \\ d_z & r_z & u_z \end{pmatrix} \begin{pmatrix} a \\ ax \\ ay \end{pmatrix} = \begin{pmatrix} p_x - i_x \\ p_y - i_y \\ p_z - i_z \end{pmatrix}. \tag{5.2}$$

Calling the matrix $M$ the inverse of the matrix on the left side of equation 5.2 gives the result:

$$a = M_{00}(p_x - i_x) + M_{01}(p_y - i_y) + M_{02}(p_z - i_z), \tag{5.3}$$

$$x = \frac{1}{a}[M_{10}(p_x - i_x) + M_{11}(p_y - i_y) + M_{12}(p_z - i_z)], \tag{5.4}$$

$$y = \frac{1}{a}[M_{20}(p_x - i_x) + M_{21}(p_y - i_y) + M_{22}(p_z - i_z)]. \tag{5.5}$$

The triple product of the vector from the light source to the triangle centroid and the two vectors making up the triangle edges determines the cosine of the angle made by the triangle normal and the incident light, whose absolute value is used here to shade the triangle. At this point, the light source is taken as the observer location at "`eye`", but that can easily be modified to use one or more independent light sources.

```
int render_rgb_local_3d ( IDisplay Disp, ISurface Surf, PetscScalar* eye, PetscScalar* dir,
PetscScalar* right )
```

- `int render_rgb_local_3d` — Returns zero or an error code.
- `IDisplay Disp` — Display object holding the RGB buffer and its characteristics.
- `ISurface Surf` — ISurface object containing triangles to render using imlib2 backend.
- `PetscScalar* eye` — Point from where we're looking (x,y,z).
- `PetscScalar* dir` — Direction we're looking (x,y,z).
- `PetscScalar* right` — Rightward direction in physical space (x,y,z).

### F.1.4 Global Function render_scale_2d()

This draws a little rwidth x rheight image depicting the scale of a field variable.

`int render_scale_2d ( IDisplay Disp, field_plot_type fieldtype, int symmetry )`

- `int render_scale_2d` — It returns zero or an error code.
- `IDisplay Disp` — Display object into which to draw the scale.
- `field_plot_type fieldtype` — Type of plot.
- `int symmetry` — Symmetry order for vector scale image.

### F.1.5 Local Function pseudocolor()

This little function converts a scalar value into an rgb color from red to blue.

`static inline void pseudocolor ( PetscScalar val, PetscScalar* scale, guchar* pixel )`

- `PetscScalar val` — Value to convert.
- `PetscScalar* scale` — Array with minimum and maximum values in which to scale val.
- `guchar* pixel` — Address in rgb buffer where this pixel should be painted.

### F.1.6 Local Function pseudohueintcolor()

This little function converts a vector into an rgb color with hue indicating direction (green, yellow, red, blue at 0, 90, 180, 270 degrees) and intensity indicating magnitude relative to reference magnitude in scale[1].

`static inline void pseudohueintcolor ( PetscScalar vx, PetscScalar vy, PetscScalar* scale, guchar* pixel )`

- `PetscScalar vx` — Vector's $x$-component.
- `PetscScalar vy` — Vector's $y$-component.
- `PetscScalar* scale` — Array whose second entry has the reference magnitude.
- `guchar* pixel` — Address in rgb buffer where this pixel should be painted.

### F.1.7 Local Function pseudoshearcolor()

This little function converts a shear tensor (symmetric, diagonals sum to zero) into an rgb color with hue indicating direction of the tensile stress (red, yellow, green, cyan, blue, magenta at 0, 30, 60, 90, 120, 150 degrees respectively; 180 is equivalent to zero for stress) and intensity indicating magnitude relative to reference magnitude in scale[0].

`static inline void pseudoshearcolor ( PetscScalar gxx, PetscScalar gxy, PetscScalar* scale, guchar* pixel )`

- `PetscScalar gxx` — Tensor's $xx$-component.
- `PetscScalar gxy` — Tensor's $xy$-component.
- `PetscScalar* scale` — Array whose first entry has the reference magnitude.
- `guchar* pixel` — Address in rgb buffer where this pixel should be painted.

### F.1.8 Local Function pseudoternarycolor()

This little function converts two ternary fractions into an rgb color, with yellow, cyan and magenta indicating the corners.

`static inline void pseudoternarycolor ( PetscScalar A, PetscScalar B, PetscScalar* scale, guchar* pixel )`

- `PetscScalar A` — First ternary fraction.

- `PetscScalar B` — Second ternary fraction.
- `PetscScalar* scale` — Array first and second ternary fractions of each of the three corner values for scaling.
- `guchar* pixel` — Address in rgb buffer where this pixel should be painted.

#### F.1.9 Local Function pseudoternarysquarecolor()

This little function converts two composition values into an rgb color, with blue, red, green and yellow indicating the corners.

```
static inline void pseudoternarysquarecolor ( PetscScalar A, PetscScalar B, PetscScalar* scale,
guchar* pixel )
```

- `PetscScalar A` — First ternary composition.
- `PetscScalar B` — Second ternary composition.
- `PetscScalar* scale` — Array: minimum and maximum first and second compositions for scaling.
- `guchar* pixel` — Address in rgb buffer where this pixel should be painted.

# G File libluminate/utility.c

**RCS Header**

This file contains small utility functions for various aspects of visualization and storage.

## Included Files

```
#include "config.h"
```
*(Section T)*

```
#include "illuminator.h"
```
*(Section C)*

```
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt, args...  )

#define __FUNCT__ "auto_scale"

#define __FUNCT__ "field_indices"
```

## G.1 Functions

### G.1.1 Global Function auto_scale()

Determine a sensible scale for plotting, returned in *scale. If a scalar field, returns the minimum and maximum; if a vector field, returns the minimum and maximum magnitudes (in 1-D, just plain minimum and maximum); if a ternary, returns the corners of the smallest equilateral triangle in ternary space in which all of the data fit.

```
int auto_scale ( PetscScalar* global_array, int points, int num_fields, int display_field,
field_plot_type fieldtype, int dimensions, PetscScalar* scale )
```

- `int auto_scale` — Returns zero or an error code.
- `PetscScalar* global_array` — Array with values to scan for scale.
- `int points` — Number of points in array to scan.
- `int num_fields` — Number of fields in array.
- `int display_field` — This display field (at least the start).
- `field_plot_type fieldtype` — Type of field.
- `int dimensions` — Number of dimensions.
- `PetscScalar* scale` — Array in which to return the minimum/maximum values.

### G.1.2 Global Function field_indices()

Given an array of `field_plot_type` enums, fill (part of) the `indices` array with integers pointing to the true variable starts. For example, in 2-D with a vector field (two fields), a scalar field (one field), a symmetric tensor field (three fields) and a ternary composition field (two fields) for a total of 8 fields, this will fill the indices array with the values 0, 2, 3, 6 and pad the rest of indices with -1, indicating when those true field variables start in the overall set of field variables.

```
void field_indices ( int nfields, int ds, field_plot_type* plottypes, int* indices )
```

- `int nfields` — Total number of fields.
- `int ds` — Dimensionality of the space (used to determine the number of fields used for a vector or tensor field).
- `field_plot_type* plottypes` — Array of `field_plot_type` enums with length `nfields`.
- `int* indices` — Array to hold the return values.

# H File libluminate/petsc.c

### RCS Header

This is the petsc.c main file. It has all of the PETSc-dependent functions.

## Included Files

```
#include <stdlib.h>

#include </usr/lib/petsc/include/petscda.h>

#include "config.h"                                                    (Section T)

#include "illuminator.h"                                               (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>

#include "libluminate/structures.h"                                    (Section D)
   #include <illuminator.h>
```

## Preprocessor definitions

```
#define ORBIT2 1
```

```
#define _REENTRANT 1
```

```
#define IMLIB2_EXISTS 1
```

```
#define __FUNCT__ "DATriangulateRange"
```

```
#define __FUNCT__ "DATriangulateLocalRange"
```

```
#define __FUNCT__ "IllErrorHandler"
```

## H.1 Functions

### H.1.1 Global Function DATriangulateLocalRange()

Calculate vertices of isoquant triangles in a 3-D distributed array. This takes a PETSc DA object, does some sanity checks, calculates array sizes, and then gets array and passes it to Draw3DBlock for triangulation.

```
int DATriangulateLocalRange ( ISurface Surf, DA theda, Vec localX, int this, PetscScalar* minmax,
int n_quants, PetscScalar* isoquants, PetscScalar* colors, int xmin, int xmax, int ymin, int ymax,
int zmin, int zmax )
```

- `int DATriangulateLocalRange` — Returns 0 or an error code.
- `ISurface Surf` — ISurface object into which to draw this DA's isoquants.
- `DA theda` — The PETSc distributed array object.
- `Vec localX` — PETSc local vector object associated with the DA with data we'd like to graph.
- `int this` — Index of the field we'd like to draw.
- `PetscScalar* minmax` — Position of block corners: xmin, xmax, ymin, ymax, zmin, zmax.
- `int n_quants` — Number of isoquant surfaces to draw (isoquant values). Note `PETSC_DECIDE` is not a valid option here, because it's impossible to know the global maximum and minimum and have consistent contours without user-supplied information.
- `PetscScalar* isoquants` — Array of function values at which to draw isoquants.
- `PetscScalar* colors` — Array of color R,G,B,A quads for each isoquant.
- `int xmin` — Smallest grid x-coordinate to render.
- `int xmax` — Largest grid x-coordinate to render, -1 goes to full x maximum, -2 in periodic systems goes to one short of x maximum.
- `int ymin` — Smallest grid y-coordinate to render.
- `int ymax` — Largest grid y-coordinate to render, -1 goes to full y maximum, -2 in periodic systems goes to one short of y maximum.
- `int zmin` — Smallest grid z-coordinate to render.
- `int zmax` — Largest grid z-coordinate to render, -1 goes to full z maximum, -2 in periodic systems goes to one short of z maximum.

### H.1.2 Global Function DATriangulateRange()

Calculate vertices of isoquant triangles in a 3-D distributed array. This takes a PETSc DA object, does some sanity checks, calculates array sizes, gets the local vector and array, and then calls `DATriangulateLocal()` to do the rest. Note that global array access (i.e. this function) is necessary for using default isoquant values, since we need to be able to calculate the maximum and minimum on the global array.

`int DATriangulateRange ( ISurface Surf, DA theda, Vec globalX, int this, PetscScalar* minmax, int n_quants, PetscScalar* isoquants, PetscScalar* colors, int xmin, int xmax, int ymin, int ymax, int zmin, int zmax )`

| | |
|---|---|
| · `int DATriangulateRange` | Returns 0 or an error code. |
| · `ISurface Surf` | ISurface object into which to draw this DA's isoquants. |
| · `DA theda` | The PETSc distributed array object. |
| · `Vec globalX` | PETSc global vector object associated with the DA with data we'd like to graph. |
| · `int this` | Index of the field we'd like to draw. |
| · `PetscScalar* minmax` | Position of block corners: xmin, xmax, ymin, ymax, zmin, zmax. |
| · `int n_quants` | Number of isoquant surfaces to draw (isoquant values), or `PETSC_DECIDE` to use red, yellow, green, blue at 0.2, 0.4, 0.6 and 0.8 between the vector's global minimum and maximum values. |
| · `PetscScalar* isoquants` | Array of function values at which to draw isoquants, or `PETSC_NULL` if `n_quants=PETSC_DECIDE`. |
| · `PetscScalar* colors` | Array of color R,G,B,A quads for each isoquant, or `PETSC_NULL` if `n_quants=PETSC_DECIDE`. |
| · `int xmin` | Smallest grid x-coordinate to render. |
| · `int xmax` | Largest grid x-coordinate to render, -1 goes to full x maximum, -2 in periodic systems goes to one short of x maximum. |
| · `int ymin` | Smallest grid y-coordinate to render. |
| · `int ymax` | Largest grid y-coordinate to render, -1 goes to full y maximum, -2 in periodic systems goes to one short of y maximum. |
| · `int zmin` | Smallest grid z-coordinate to render. |
| · `int zmax` | Largest grid z-coordinate to render, -1 goes to full z maximum, -2 in periodic systems goes to one short of z maximum. |

### H.1.3 Global Function IllErrorHandler()

Handle errors, in this case the PETSc way.

`int IllErrorHandler ( int id, char* message )`

| | |
|---|---|
| · `int IllErrorHandler` | Returns the error code supplied. |
| · `int id` | Index of the error, defined in petscerror.h. |
| · `char* message` | Text of the error message. |

# I File libluminate/illumulti.c

**RCS Header**

This file contains the functions `IlluMultiSave()`, `IlluMultiLoad()` and `IlluMultiRead()` designed to handle distributed storage and retrieval of data on local drives of machines in a Beowulf cluster. This should allow rapid loading of timestep data for "playback" from what is essentially a giant RAID-0 array of distributed disks, at enormously higher speeds than via NFS from a hard drive or RAID array on the head node. The danger of course is that if one node's disk goes down, you don't have a valid data set any more, but that's the nature of RAID-0, right?
The filenames saved are:

- <`basename`>`.cpu####.meta`, where `####` is replaced by the CPU number (more than four digits if there are more than 9999 CPUs :-), which has the metadata for the whole thing in XML format (written by `GNOME libxml`), as described in the notes on the `IlluMultiStoreXML()` function.

- <basename>.cpu####.data which is simply a stream of the raw data, optionally compressed by gzip.

If one were saving timesteps, one might include a timestep number in the basename, and also timestep and simulation time in the metadata. The metadata can also hold simulation parameters, etc.
This supports 1-D, 2-D and 3-D distributed arrays. As an extra feature, you can load a multi-CPU distributed array scattered over lots of files into a single CPU, to facilitate certain modes of data visualization.

## Included Files

```
#include "illuminator.h"                                    (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>

#include </usr/include/glib-2.0/glib.h>

#include </usr/lib/petsc/include/petscblaslapack.h>

#include </usr/include/libxml2/libxml/tree.h>

#include </usr/include/libxml2/libxml/parser.h>

#include <stdio.h>

#include <errno.h>

#include <string.h>

#include <stdlib.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt, args...  )

#define __FUNCT__ "IlluMultiParseXML"

#define __FUNCT__ "IlluMultiParseData"

#define __FUNCT__ "IlluMultiStoreXML"

#define __FUNCT__ "IlluMultiStoreData"

#define __FUNCT__ "checkagree"

#define __FUNCT__ "IlluMultiRead"

#define __FUNCT__ "IlluMultiLoad"

#define __FUNCT__ "IlluMultiSave"
```

## I.1 Functions

### I.1.1 Global Function IlluMultiLoad()

This creates a new distributed array of the appropriate size and loads the data into the vector contained in it (as retrieved by `DAGetVector()`). It also reads the user metadata parameters into arrays stored at the supplied pointers.

```
int IlluMultiLoad ( MPI_Comm comm, char* basename, DA* theda, PetscScalar* wx, PetscScalar* wy,
PetscScalar* wz, field_plot_type** fieldtypes, int* usermetacount, char*** usermetanames, char***
usermetadata )
```

- `int IlluMultiLoad` — It returns zero or an error code.
- `MPI_Comm comm` — MPI communicator, if NULL it uses PETSC_COMM_WORLD.
- `char* basename` — Base file name.
- `DA* theda` — Pointer to a DA object (to be created by this function).
- `PetscScalar* wx` — Physical overall width in the $x$-direction.
- `PetscScalar* wy` — Physical overall width in the $y$-direction.
- `PetscScalar* wz` — Physical overall width in the $z$-direction.
- `field_plot_type** fieldtypes` — Data (plot) types for field variables.
- `int* usermetacount` — Pointer to an int where we put the number of user metadata parameters loaded.
- `char*** usermetanames` — Pointer to a char ** where the loaded parameter names are stored. This is `malloc`ed by this function, so a call to `free()` is needed to free up its data.
- `char*** usermetadata` — Pointer to a char ** where the loaded parameter strings are stored. This is `malloc`ed by this function, so a call to `free()` is needed to free up its data.

First it gets the parameters from the XML file.
Next it creates a distributed array based on those parameters, and sets the names of its fields.
Then it streams the data into the distributed array's vector in one big slurp.

### I.1.2 Global Function IlluMultiRead()

This reads the data into an existing distributed array and vector, checking that the sizes are right etc.

```
int IlluMultiRead ( MPI_Comm comm, DA theda, Vec X, char* basename, int* usermetacount, char***
usermetanames, char*** usermetadata )
```

- `int IlluMultiRead` — It returns zero or an error code.
- `MPI_Comm comm` — MPI communicator, if NULL it uses PETSC_COMM_WORLD.
- `DA theda` — Distributed array object controlling the data to read.
- `Vec X` — Vector into which to read the data.
- `char* basename` — Base file name.
- `int* usermetacount` — Pointer to an int where we put the number of user metadata parameters loaded.
- `char*** usermetanames` — Pointer to a char ** where the loaded parameter names are stored. This is `malloc`ed by this function, so a call to `free()` is needed to free up its data.
- `char*** usermetadata` — Pointer to a char ** where the loaded parameter strings are stored. This is `malloc`ed by this function, so a call to `free()` is needed to free up its data.

First it gets the properties of the distributed array for comparison with the metadata.
Next it parses the XML metadata file into the document tree, and reads its content into the appropriate structures, comparing parameters with those of the existing distributed array structure.
Then it streams in the data in one big slurp.

### I.1.3 Global Function IlluMultiSave()

This saves the vector X in multiple files, two per process.

```
int IlluMultiSave ( MPI_Comm comm, DA theda, Vec X, char* basename, PetscScalar wx, PetscScalar wy, PetscScalar wz, field_plot_type* fieldtypes, int usermetacount, char** usermetanames, char** usermetadata, int compressed )
```

- `int IlluMultiSave` — it returns zero or an error code.
- `MPI_Comm comm` — MPI communicator, if NULL it uses PETSC_COMM_WORLD.
- `DA theda` — Distributed array object controlling data saved.
- `Vec X` — Vector whose data are actually being saved.
- `char* basename` — Base file name.
- `PetscScalar wx`
- `PetscScalar wy`
- `PetscScalar wz`
- `field_plot_type* fieldtypes`
- `int usermetacount` — Number of user metadata parameters.
- `char** usermetanames` — User metadata parameter names.
- `char** usermetadata` — User metadata parameter strings.
- `int compressed` — Data compression: if zero then no compression (fastest), 1-9 then gzip compression level, 10-15 then gzip –best. If 16-31 then save guint32s representing relative values between min and max for each field, compressed according to this value minus 16. Likewise for 32-47 and guint16s, and 48-63 for guint8s. Yes, these alternative formats lose information and can't be used for accurate checkpointing, but they should retain enough data for visualization (except perhaps for the guint8s, which are possibly acceptable for vectors but certainly not contours).

First a check to verify a supported value of `compressed`, but no fancy guint* compression for complex!
Then get the distributed array parameters and processor number, and store all this data in the XML .meta file.
Finally, the data just stream out to the data file or gzip pipe in one big lump.

### I.1.4 Local Function IlluMultiParseData()

This function reads in the data stored by IlluMultiStoreData(), complete with int/gzip compression.

```
static int IlluMultiParseData ( MPI_Comm comm, PetscScalar* globalarray, char* basename, int rank, int compressed, int gridpoints, int dof, int wrongendian, PetscScalar* fieldmin, PetscScalar* fieldmax )
```

- `int IlluMultiParseData` — It returns zero or an error code.
- `MPI_Comm comm` — MPI communicator, if NULL it uses PETSC_COMM_WORLD.
- `PetscScalar* globalarray` — Array into which to load the (local) data.
- `char* basename` — Base file name.
- `int rank` — CPU number to read data for.

- `int compressed` — Data compression: if zero then no compression (fastest), 1-9 then gzip compression level, 10-15 then gzip –best. If 16-31 then save guint32s representing relative values between min and max for each field, compressed according to this value minus 16. Likewise for 32-47 and guint16s, and 48-63 for guint8s. Yes, these alternative formats lose information and can't be used for accurate checkpointing, but they should retain enough data for visualization (except perhaps for the guint8s, which are possibly acceptable for vectors but likely not contours).
- `int gridpoints` — Number of gridpoints to read data for.
- `int dof` — Degrees of freedom at each node, a.k.a. number of field variables.
- `int wrongendian` — Tells whether the data are stored in the opposite endian format from this platform, and thus must be switched when the data are streamed in.
- `PetscScalar* fieldmin` — Minimum value of each field variable.
- `PetscScalar* fieldmax` — Maximum value of each field variable.

### I.1.5 Local Function IlluMultiParseXML()

This function reads in the XML metadata document and returns the various parameter values in the addresses pointed to by the arguments. It is called by IlluMultiLoad() and IlluMultiRead().

```
static int IlluMultiParseXML ( MPI_Comm comm, char* basename, int rank, int* compressed, int*
wrongendian, int* dim, int* px, int* py, int* pz, int* nx, int* ny, int* nz, PetscScalar* wx,
PetscScalar* wy, PetscScalar* wz, int* xm, int* ym, int* zm, int* dof, int* sw, DAStencilType* st,
DAPeriodicType* wrap, char*** fieldnames, field_plot_type** fieldtypes, PetscScalar** fieldmin,
PetscScalar** fieldmax, int* usermetacount, char*** usermetanames, char*** usermetadata )
```

- `int IlluMultiParseXML` — It returns zero or an error code.
- `MPI_Comm comm` — MPI communicator, if NULL it uses PETSC_COMM_WORLD.
- `char* basename` — Base file name.
- `int rank` — CPU number to read data for.
- `int* compressed` — Data compression: if zero then no compression (fastest), 1-9 then gzip compression level, 10-15 then gzip –best. If 16-31 then save guint32s representing relative values between min and max for each field, compressed according to this value minus 16. Likewise for 32-47 and guint16s, and 48-63 for guint8s. Yes, these alternative formats lose information and can't be used for accurate checkpointing, but they should retain enough data for visualization (except perhaps for the guint8s, which are possibly acceptable for vectors but likely not contours).
- `int* wrongendian` — Tells whether the data are stored in the opposite endian format from this platform, and thus must be switched when the data are streamed in.
- `int* dim` — Dimensionality of the space.
- `int* px` — Number of processors in the $x$-direction.
- `int* py` — Number of processors in the $y$-direction.
- `int* pz` — Number of processors in the $z$-direction.
- `int* nx` — Number of grid points over the entire array in the $x$-direction.
- `int* ny` — Number of grid points over the entire array in the $y$-direction.
- `int* nz` — Number of grid points over the entire array in the $z$-direction.

- `PetscScalar* wx` — Physical overall width in the $x$-direction, `PETSC_NULL` if not needed.
- `PetscScalar* wy` — Physical overall width in the $y$-direction, `PETSC_NULL` if not needed.
- `PetscScalar* wz` — Physical overall width in the $z$-direction, `PETSC_NULL` if not needed.
- `int* xm` — Number of grid points over the local part of the array in the $x$-direction.
- `int* ym` — Number of grid points over the local part of the array in the $y$-direction.
- `int* zm` — Number of grid points over the local part of the array in the $z$-direction.
- `int* dof` — Degrees of freedom at each node, a.k.a. number of field variables.
- `int* sw` — Stencil width.
- `DAStencilType* st` — Stencil type, given by the `PETSc enum` values.
- `DAPeriodicType* wrap` — Periodic type, given by the `PETSc enum` values.
- `char*** fieldnames` — Names of the field variables.
- `field_plot_type** fieldtypes` — Data (plot) types for field variables, `PETSC_NULL` if not needed.
- `PetscScalar** fieldmin` — Minimum value of each field variable.
- `PetscScalar** fieldmax` — Maximum value of each field variable.
- `int* usermetacount` — Number of user metadata parameters.
- `char*** usermetanames` — User metadata parameter names.
- `char*** usermetadata` — User metadata parameter strings.

For GlobalSize, since there's no *size attribute (for an 0.1 version document), assume 1.
If the type attribute is missing from the Field node (as it is in version 0.1 documents), assume `FIELD_SCALAR`.

### I.1.6 Local Function IlluMultiStoreData()

This function stores the data file.

```
static int IlluMultiStoreData ( MPI_Comm comm, PetscScalar* globalarray, char* basename, int rank,
int compressed, int gridpoints, int dof, PetscScalar* fieldmin, PetscScalar* fieldmax )
```

- `int IlluMultiStoreData` — It returns zero or an error code.
- `MPI_Comm comm` — MPI communicator, if NULL it uses PETSC_COMM_WORLD.
- `PetscScalar* globalarray` — Array from which to save the (local) data.
- `char* basename` — Base file name.
- `int rank` — CPU number to read data for.
- `int compressed` — Data compression: if zero then no compression (fastest), 1-9 then gzip compression level, 10-15 then gzip –best. If 16-31 then save guint32s representing relative values between min and max for each field, compressed according to this value minus 16. Likewise for 32-47 and guint16s, and 48-63 for guint8s. Yes, these alternative formats lose information and can't be used for accurate checkpointing, but they should retain enough data for visualization (except perhaps for the guint8s, which are possibly acceptable for vectors but likely not contours).
- `int gridpoints` — Number of gridpoints to store data for.
- `int dof` — Degrees of freedom at each node, a.k.a. number of field variables.
- `PetscScalar* fieldmin` — Minimum value of each field variable.
- `PetscScalar* fieldmax` — Maximum value of each field variable.

### I.1.7 Local Function IlluMultiStoreXML()

This function opens, stores and closes the XML metadata file for IlluMulti format storage. It is called by IlluMultiSave().

```
static int IlluMultiStoreXML ( MPI_Comm comm, char* basename, int rank, int compressed, int dim,
int px, int py, int pz, int nx, int ny, int nz, PetscScalar wx, PetscScalar wy, PetscScalar wz, int
xm, int ym, int zm, int dof, int sw, int st, int wrap, char** fieldnames, field_plot_type*
fieldtypes, PetscReal* fieldmin, PetscReal* fieldmax, int usermetacount, char** usermetanames,
char** usermetadata )
```

- `int IlluMultiStoreXML` — It returns zero or an error code.
- `MPI_Comm comm` — MPI communicator, if NULL it uses PETSC_COMM_WORLD.
- `char* basename` — Base file name.
- `int rank` — CPU number to store data for.
- `int compressed` — Data compression: if zero then no compression (fastest), 1-9 then gzip compression level, 10-15 then gzip –best. If 16-31 then save guint32s representing relative values between min and max for each field, compressed according to this value minus 16. Likewise for 32-47 and guint16s, and 48-63 for guint8s. Yes, these alternative formats lose information and can't be used for accurate checkpointing, but they should retain enough data for visualization (except perhaps for the guint8s, which are possibly acceptable for vectors but certainly not contours).
- `int dim` — Dimensionality of the space.
- `int px` — Number of processors in the $x$-direction.
- `int py` — Number of processors in the $y$-direction.
- `int pz` — Number of processors in the $z$-direction.
- `int nx` — Number of grid points over the entire array in the $x$-direction.
- `int ny` — Number of grid points over the entire array in the $y$-direction.
- `int nz` — Number of grid points over the entire array in the $z$-direction.
- `PetscScalar wx` — Physical overall width in the $x$-direction.
- `PetscScalar wy` — Physical overall width in the $y$-direction.
- `PetscScalar wz` — Physical overall width in the $z$-direction.
- `int xm` — Number of grid points over the local part of the array in the $x$-direction.
- `int ym` — Number of grid points over the local part of the array in the $y$-direction.
- `int zm` — Number of grid points over the local part of the array in the $z$-direction.
- `int dof` — Degrees of freedom at each node, a.k.a. number of field variables.
- `int sw` — Stencil width.
- `int st` — Stencil type, given by the `PETSc enum` values.
- `int wrap` — Periodic type, given by the `PETSc enum` values.
- `char** fieldnames` — Names of the field variables.
- `field_plot_type* fieldtypes` — Data (plot) types for field variables.
- `PetscReal* fieldmin` — Minimum value of each field variable.
- `PetscReal* fieldmax` — Maximum value of each field variable.
- `int usermetacount` — Number of user metadata parameters.
- `char** usermetanames` — User metadata parameter names.
- `char** usermetadata` — User metadata parameter strings.

The XML tags in the .meta file consist of:

| | |
|---|---|
| `IlluMulti` | Primary tag, with attributes `version`, `endian` (`big` or `little`) and `compression` (none, 1-9, best, float*, long*, short* or char*[1]). |
| `GlobalCPUs` | Number of CPUs in each direction, with attributes `dimensions`, `xwidth`, `ywidth` and `zwidth` |
| `GlobalSize` | Size of the entire distributed array, with attributes `xwidth`, `ywidth`, `zwidth`, `xsize`**, `ysize`**, `zsize`** and `fields` |
| `LocalSize` | Size of the entire local part of the array, with attributes `xwidth`, `ywidth` and `zwidth` |
| `Stencil` | Stencil and periodic data, with attributes `width`, `type` and `periodic` (using `PETSc enum` values) |
| `Field` | Data on each field, attributes `name`, `type`**, `min` and `max` |
| `User` | User parameters, attributes `name` and `value` |

*Lossy compression to smaller data types.
**Represents new attribute for IlluMulti 0.2 file format.

### I.1.8 Local Function checkagree()

Ancillary routine for `IlluMultiRead()`: checks agreement of parameters and reports disagreement if necessary.

```
static inline int checkagree ( MPI_Comm comm, int da, int file, char* parameter )
```

- `int checkagree` — Returns 0 if they agree, 1 otherwise.
- `MPI_Comm comm` — MPI communicator, if NULL it uses PETSC_COMM_WORLD.
- `int da` — Integer parameter from the existing DA.
- `int file` — Integer parameter read from the file.
- `char* parameter` — Parameter name for reporting.

# J File libluminate/importexport.c

### RCS Header

This file contains functions for importing and exporting data to/from foreign file formats. Thus far only .tiff images are supported.

## Included Files

```
#include "illuminator.h"                                    (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt, args...  )
```

```
#define __FUNCT__ "IllImageWrite"
```

```
#define __FUNCT__ "IllImageSave"
```

## J.1 Functions

### J.1.1 Global Function IllImageSave()

This function stores one field as an image. In 2-D, it stores the whole thing; in 3-D, it stores one image per layer.

```
int IllImageSave ( MPI_Comm comm, DA theda, Vec X, char* basename, int redfield, int greenfield,
int bluefield, PetscScalar* rgbmin, PetscScalar* rgbmax, int* coordrange, IllLayerStyle layer,
IllImageFormat format )
```

- `int IllImageSave` — It returns zero or an error code.
- `MPI_Comm comm`
- `DA theda` — Distributed array object controlling data saved.
- `Vec X` — Vector whose data are actually being saved.
- `char* basename` — Base file name.
- `int redfield`
- `int greenfield`
- `int bluefield`
- `PetscScalar* rgbmin` — Field values to use for image R,G,B zero.
- `PetscScalar* rgbmax` — Field values to use for image R,G,B max.
- `int* coordrange` — Minimum and maximum x, y, z to store (PETSC_NULL for the whole thing).
- `IllLayerStyle layer` — Layer style to use for 3-D DA.
- `IllImageFormat format` — Image format.

MPI_COMM comm MPI communicator, if NULL it uses PETSC_COMM_WORLD.
int field Field index.

### J.1.2 Local Function IllImageWrite()

This writes a single monochrome .tif image from a double array for IllImageSave().

```
static inline int IllImageWrite ( double* array, int xm, int ym, int stride, int gxm, int goff, int
boff, char* basename, PetscScalar minval, PetscScalar maxval )
```

- `double* array` — Array of values to use as intensity values in the image, pointing to first red value.
- `int xm` — Width of the array rows.
- `int ym` — Number of rows in the array.
- `int stride` — Number of doubles per point.
- `int gxm` — Number of points between row starts.
- `int goff` — Offest for green field.
- `int boff` — Offest for blue field.
- `char* basename` — Base file name; if the last four chars aren't ".tif" this appends that.
- `PetscScalar minval` — Array value mapping to black.
- `PetscScalar maxval` — Array value mapping to white.

inline int IllImageWrite It returns zero or an error code.

# K File libluminate/geomview.c

### RCS Header

This file has the Geomview interface, including the PETSc vector gather operations to bring everything to CPU 0.

## Included Files

```
#include <stdio.h>

#include </usr/lib/petsc/include/petscvec.h>

#include "config.h"                                                    (Section T)

#include "illuminator.h"                                               (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>

#include "libluminate/structures.h"                                    (Section D)
   #include <illuminator.h>

#include <stdlib.h>

#include <unistd.h>

#include </usr/include/glib-2.0/glib.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt, args...  )

#define __FUNCT__ "GeomviewBegin"

#define __FUNCT__ "GeomviewEnd"

#define GEOMVIEW_SET_INT( j, x )

#define GEOMVIEW_SET_FLOAT( j, x )

#define __FUNCT__ "GeomviewDisplayTriangulation"
```

## K.1 Functions

### K.1.1 Global Function GeomviewBegin()

Spawn a new geomview process. Most of this was shamelessly ripped from Ken Brakke's Surface Evolver.

```
int GeomviewBegin ( MPI_Comm comm, IDisplay* newdisp )
```

- `int GeomviewBegin` — Returns 0 or an error code.
- `MPI_Comm comm` — MPI communicator for rank information, if NULL it uses PETSC_COMM_WORLD.
- `IDisplay* newdisp` — Pointer in which to put a new IDisplay object.

### K.1.2 Global Function GeomviewDisplayTriangulation()

Pipe the current triangulation to Geomview for display. Much of this is based on Ken Brakke's Surface Evolver.

```
int GeomviewDisplayTriangulation ( MPI_Comm comm, ISurface Surf, IDisplay Disp, PetscScalar*
minmax, char* name, PetscTruth transparent )
```

- `int GeomviewDisplayTriangulation` Returns 0 or an error code.
- `MPI_Comm comm` MPI communicator for rank information, if NULL it uses PETSC_COMM_WORLD.
- `ISurface Surf` ISurface object containing triangles to render using Geomview.
- `IDisplay Disp` IDisplay object whose geomview FILE we're closing.
- `PetscScalar* minmax` Position of block corners: xmin, xmax, ymin, ymax, zmin, zmax.
- `char* name` Name to give the Geomview OOGL object which we create.
- `PetscTruth transparent` Geomview transparency flag.

First, this creates global and local vectors for all of the triangle vertices.
It then gathers ("scatters") all vertex data to processor zero,
and puts them in an array.
Finally, it sends everything to Geomview,
and cleans up the mess.

### K.1.3 Global Function GeomviewEnd()

Exit the current running Geomview process and close its pipe. Based in part on Ken Brakke's Surface Evolver.

```
int GeomviewEnd ( MPI_Comm comm, IDisplay Disp )
```

- `int GeomviewEnd` Returns 0 or an error code.
- `MPI_Comm comm` MPI communicator for rank information, if NULL it uses PETSC_COMM_WORLD.
- `IDisplay Disp` IDisplay object whose geomview FILE we're closing.

# L File libluminate/imlib2.c

### RCS Header

This file uses Imlib2 to turn guchar arrays into Imlib images, render sets of triangles into those images, and send back the guchar arrays.

## Included Files

```
#include "illuminator.h"                                                    (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1
```

```
#define IMLIB2_EXISTS 1

#define __FUNCT__ "imlib2_render_triangles"
```

## L.1 Functions

### L.1.1 Global Function imlib2__render__triangles()

This simply takes a bunch of triangle vertex and color data and renders it into the "data" buffer in RGBA format using Imlib2.

`int imlib2_render_triangles ( unsigned int* data, int width, int height, int num_triangles, int* triangle_coords, PetscScalar* triangle_colors, int color_skip, PetscScalar* triangle_shades, int shade_skip )`

- `int imlib2_render_triangles` It returns zero or an error code.
- `unsigned int* data`
- `int width` Width of the data buffer in pixels.
- `int height` Height of the data buffer in pixels.
- `int num_triangles` Number of triangles to render.
- `int* triangle_coords` Integer coordinates of the triangles.
- `PetscScalar* triangle_colors` R,G,B,A colors of the triangles between 0 and 1.
- `int color_skip` Number of PetscScalars to skip between color sets.
- `PetscScalar* triangle_shades` Shading of each triangle, zero for black to one for normal.
- `int shade_skip` Number of PetscScalars to skip between shades.

DATA32 *data The data buffer into which to render with 4×width×height bytes.

# M File tsview.c

### RCS Header

This program views the output of a time series saved using `IlluMultiSave()`. It basically just switches between timesteps; future versions may be more interesting. The neat part of it is that it loads multiprocessor data and displays it on a single CPU.

## Included Files

```
#include "illuminator.h"                                        (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>

#include <sys/dir.h>

#include <libgen.h>

#include <string.h>

#include <stdlib.h>

#include <term.h>

#include <readline/readline.h>

#include <readline/history.h>
```

### Preprocessor definitions

```
#define ORBIT2 1
```

```
#define _REENTRANT 1
```

```
#define IMLIB2_EXISTS 1
```

```
#define HELP_STRING "tsview commands:\n <enter> Display next timestep\n b Display previous timestep\n i increment Set the next timestep increment\n ### Jump to timestep ###\n t Toggle Geomview transparency (3-D only)\n v Change field displayed (3-D only)\n d Dump geomview to picture (3-D only), creates basename-f%d.ppm\n a Dump all timesteps to pictures (3-D only)\n p [v1 v2 ...] Set contour values for plotting or \"auto\" (3-D only)\n r Reloads entries in a directory\n s size Set maximum dimension of PETSc viewer windows (2-D only)\n cx, cy, cz Toggle xcut, ycut, zcut (cut last row/plane of periodic DA)\n gx30-90, y,z Set plot x range to 30-90, same for y and z\n h/? Print this information\n q/x Quit tsview\n"
```

```
#define DPRINTF( fmt, args...  )
```

```
#define __FUNCT__ "myfilter"
```

```
#define __FUNCT__ "main"
```

## M.1 Variables

### M.1.1 Variable basefilename

```
char* basefilename
```

### M.1.2 Local Variables

**help**
```
static char help[]
```

**line_read**
Functions for reading the command line and avoiding reading empty lines
Probably this function is not Petsc safe, but we'll see.

```
static char* line_read
```

## M.2 Functions

### M.2.1 Global Function PetscSynchronizedFReadline()

```
int PetscSynchronizedFReadline ( MPI_Comm comm, char* message, char* string )
```

### M.2.2 Global Function main()

This is `main()`.

```
int main ( int argc, char* argv[] )
```

- `int main` — It returns an int to the OS.
- `int argc` — Argument count.
- `char* argv[]` — Arguments.

After `PETSc` initialization, it gets the list of files matching the basename.
In the main loop, the various timesteps are displayed, with options:

- A number jumps to that entry in the files table.
- `<return>` loads the next file.

- `b` goes back one file.
- `q` quits the program.

The Illuminator-based 3-D viewer can only display one field at a time. At the beginning, that is field 0, and is cycled using the `v` command.

### M.2.3 Global Function myfilter()

This function returns non-zero for "qualifying" file names which start with the stored files' basename and end with `.cpu0000.meta`. It is used as the `select()` function for `scandir()` in `main()`.

```
int myfilter ( const struct dirent* direntry )
```

| | |
|---|---|
| · `int myfilter` | Returns non-zero for qualifying filenames. |
| · `const struct dirent* direntry` | Directory entry with filename to test. |

### M.2.4 Global Function rl_gets()

```
char* rl_gets ( char* message )
```

# N File tsview-ng.c

**RCS Header**

This program views the output of a time series saved using `IlluMultiSave()`. It basically just switches between timesteps; future versions may be more interesting. The neat part of it is that it loads multiprocessor data and displays it on a single CPU.

## Included Files

```
#include "illuminator.h"                                                    (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>

#include </usr/include/libglade-2.0/glade/glade.h>

#include </usr/include/libgnomeui-2.0/gnome.h>

#include </usr/include/libgnomeui-2.0/libgnomeui/libgnomeui.h>

#include <sys/dir.h>

#include <libgen.h>

#include <string.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt, args...  )

#define DEFAULT_RENDER_SIZE 300

#define SCALE_BPP 3
```

```
#define __FUNCT__ "render_dataviews"

#define RENDER_WIDTH

#define RENDER_HEIGHT

#define __FUNCT__ "my_time_filter"

#define __FUNCT__ "my_time_filter"

#define __FUNCT__ "main"
```

## N.1 Type definitions

### N.1.1 Typedef terntype

```
typedef enum {...} terntype
```

```
enum
 {
   GEN_TRI;
   UNIT_TRI;
   EQ_TRI;
   GEN_RECT;
   UNIT_SQ;
   SQUARE;
 }
```

## N.2 Variables

### N.2.1 Variable xml

Declared in illuminator.c, these give the current number of triangles on this node and corner coordinates and color information for each triangle.

```
GladeXML* xml
```

### N.2.2 Variable Surf

```
ISurface Surf
```

### N.2.3 Variable Disp

```
IDisplay Disp[1]
```

### N.2.4 Variable entrynum

```
int entrynum
```

### N.2.5 Variable total_entries

```
int total_entries
```

### N.2.6 Variable current_timestep

```
int current_timestep
```

### N.2.7 Variable the_basename

```
char* the_basename
```

### N.2.8 Variable basedirname

```
char* basedirname
```

### N.2.9 Variable stepnames

```
char** stepnames
```

### N.2.10 Variable current_time

```
double current_time
```

### N.2.11 Variable log_text

```
char* log_text
```

### N.2.12 Variable ternary_dp_color

```
PetscScalar ternary_dp_color[4]
```

### N.2.13 Variable dp_supp_colors

```
int dp_supp_colors
```

### N.2.14 Variable dp_supp_color_points

```
int* dp_supp_color_points
```

### N.2.15 Variable dp_supp_red

```
PetscScalar* dp_supp_red
```

### N.2.16 Variable dp_supp_green

```
PetscScalar* dp_supp_green
```

### N.2.17 Variable dp_supp_blue

```
PetscScalar* dp_supp_blue
```

### N.2.18 Variable dp_supp_alpha

```
PetscScalar* dp_supp_alpha
```

### N.2.19 Variable dp_supp_AB

```
PetscScalar* dp_supp_AB
```

### N.2.20 Variable width

```
int width
```

### N.2.21 Variable height

```
int height
```

### N.2.22 Variable bpp

```
int bpp
```

#### N.2.23 Variable scale_size

```
int scale_size
```

#### N.2.24 Variable nx

```
int nx
```

#### N.2.25 Variable ny

```
int ny
```

#### N.2.26 Variable transform

```
int transform
```

#### N.2.27 Variable dataview_count

```
int dataview_count
```

#### N.2.28 Variable dataviews

```
GtkWidget* dataviews[1]
```

#### N.2.29 Variable scalar_disp

```
IDisplay scalar_disp
```

#### N.2.30 Variable ternary_square_disp

```
IDisplay ternary_square_disp
```

#### N.2.31 Variable ternary_triangle_disp

```
IDisplay ternary_triangle_disp
```

#### N.2.32 Variable ternary_disp

```
IDisplay ternary_disp
```

#### N.2.33 Variable vector_disp

```
IDisplay vector_disp
```

#### N.2.34 Variable shear_disp

```
IDisplay shear_disp
```

#### N.2.35 Variable scalar_auto_set

```
gboolean scalar_auto_set
```

#### N.2.36 Variable ternary_auto_set

```
gboolean ternary_auto_set
```

#### N.2.37 Variable vector_auto_set

```
gboolean vector_auto_set
```

#### N.2.38 Variable shear_auto_set

```
gboolean shear_auto_set
```

#### N.2.39 Variable sizemag

```
gdouble sizemag
```

#### N.2.40 Variable transp

```
PetscTruth transp
```

#### N.2.41 Variable lastern

```
terntype lastern
```

#### N.2.42 Variable thistern

```
terntype thistern
```

#### N.2.43 Variable theda

```
DA theda
```

#### N.2.44 Variable global

```
Vec global
```

#### N.2.45 Variable minmax

```
PetscScalar minmax[6]
```

#### N.2.46 Variable scales

```
PetscScalar scales[15]
```

#### N.2.47 Variable fieldtypes

```
field_plot_type* fieldtypes
```

#### N.2.48 Variable dimensions

```
int dimensions
```

#### N.2.49 Variable num_fields

```
int num_fields
```

#### N.2.50 Variable current_field

```
int current_field
```

#### N.2.51 Variable field_index

```
int* field_index
```

#### N.2.52 Variable num_variables

```
int num_variables[1]
```

#### N.2.53 Variable variable_indices

```
int** variable_indices
```

#### N.2.54 Local Variables

**help**
```
static char help[]
```

**basefilename**
Little variable for my_*_filter() and refresh_stepnames().
```
static char* basefilename
```

### N.3 Functions

#### N.3.1 Global Function change_ternary()

```
void change_ternary ( GtkWidget* widget, gpointer user_data )
```

#### N.3.2 Global Function change_variable()

```
void change_variable ( GtkWidget* widget, gpointer user_data )
```

#### N.3.3 Global Function display_timestep()

```
void display_timestep ( int usermetacount, char** usermetanames, char** usermetadata )
```

#### N.3.4 Global Function main()

This is `main()`.

```
int main ( int argc, char* argv[] )
```

- `int main` — It returns an int to the OS.
- `int argc` — Argument count.
- `char* argv[]` — Arguments.

After `PETSc` and glade/GNOME initialization, it gets the list of files matching the basename.

#### N.3.5 Global Function my_notime_filter()

This function returns non-zero for "qualifying" file names which start with the stored files' basename and end with `.cpu0000.meta`. It is used as the `select()` function for `scandir()` in `main()`.

```
int my_notime_filter ( const struct dirent* direntry )
```

- `int my_notime_filter` — Returns non-zero for qualifying filenames.
- `const struct dirent* direntry` — Directory entry with filename to test.

#### N.3.6 Global Function my_time_filter()

This function returns non-zero for "qualifying" file names which start with the stored files' basename.time and end with `.cpu0000.meta`. It is used as the `select()` function for `scandir()` in `main()`.

```
int my_time_filter ( const struct dirent* direntry )
```

- `int my_time_filter` — Returns non-zero for qualifying filenames.
- `const struct dirent* direntry` — Directory entry with filename to test.

### N.3.7 Global Function on_about_activate()

```
void on_about_activate ( GtkWidget* none, gpointer user_data )
```

### N.3.8 Global Function on_log_reload_button_clicked()

This reloads the .log file.

```
void on_log_reload_button_clicked ( GtkWidget* none, gpointer user_data )
```

- `GtkWidget* none` — Empty GtkWidget (unusable because it's a menu item).
- `gpointer user_data` — Empty pointer.

### N.3.9 Global Function on_mag_spin_value_changed()

```
void on_mag_spin_value_changed ( GtkWidget* mag_spin, gpointer user_data )
```

### N.3.10 Global Function on_path_filename_entry_activate()

```
void on_path_filename_entry_activate ( GtkWidget* theentry, gpointer user )
```

### N.3.11 Global Function on_plot_area_expose_event()

```
void on_plot_area_expose_event ( GtkWidget* widget, GdkEventExpose* event, gpointer user_data )
```

### N.3.12 Global Function on_refresh_activate()

```
void on_refresh_activate ( GtkWidget* none, gpointer user_data )
```

### N.3.13 Global Function on_run_log_activate()

```
void on_run_log_activate ( GtkWidget* none, gpointer user_data )
```

### N.3.14 Global Function on_save_activate()

This function saves both the current field image and also the scale image.

```
void on_save_activate ( GtkWidget* widget, gpointer user_data )
```

- `GtkWidget* widget` — Standard GTK+ callback argument, ignored here.
- `gpointer user_data` — Standard GTK+ callback argument, ignored here.

### N.3.15 Global Function on_save_all_activate()

```
void on_save_all_activate ( GtkWidget* none, gpointer user_data )
```

### N.3.16 Global Function on_scalar_auto_checkbutton_toggled()

```
void on_scalar_auto_checkbutton_toggled ( GtkWidget* thebutton, gpointer user_data )
```

### N.3.17 Global Function on_scalar_max_entry_changed()

```
void on_scalar_max_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.18 Global Function on_scalar_min_entry_changed()

```
void on_scalar_min_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.19 Global Function on_scalar_scale_area_expose_event()

```
void on_scalar_scale_area_expose_event ( GtkWidget* widget, GdkEventExpose* event, gpointer
user_data )
```

### N.3.20 Global Function on_scale_size_entry_activate()

```
void on_scale_size_entry_activate ( GtkWidget* theentry, gpointer user )
```

### N.3.21 Global Function on_shear_auto_checkbutton_toggled()

```
void on_shear_auto_checkbutton_toggled ( GtkWidget* thebutton, gpointer user_data )
```

### N.3.22 Global Function on_shear_max_entry_changed()

```
void on_shear_max_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.23 Global Function on_shear_scale_area_expose_event()

```
void on_shear_scale_area_expose_event ( GtkWidget* widget, GdkEventExpose* event, gpointer
user_data )
```

### N.3.24 Global Function on_ternary_1A_entry_changed()

```
void on_ternary_1A_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.25 Global Function on_ternary_1B_entry_changed()

```
void on_ternary_1B_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.26 Global Function on_ternary_2A_entry_changed()

```
void on_ternary_2A_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.27 Global Function on_ternary_2B_entry_changed()

```
void on_ternary_2B_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.28 Global Function on_ternary_3A_entry_changed()

```
void on_ternary_3A_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.29 Global Function on_ternary_3B_entry_changed()

```
void on_ternary_3B_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.30 Global Function on_ternary_auto_checkbutton_toggled()

```
void on_ternary_auto_checkbutton_toggled ( GtkWidget* thebutton, gpointer user_data )
```

### N.3.31 Global Function on_ternary_dp_alpha_entry_changed()

```
void on_ternary_dp_alpha_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.32 Global Function on_ternary_dp_blue_entry_changed()

```
void on_ternary_dp_blue_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.33 Global Function on_ternary_dp_green_entry_changed()

```
void on_ternary_dp_green_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.34 Global Function on_ternary_dp_red_entry_changed()

```
void on_ternary_dp_red_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.35 Global Function on_ternary_scale_area_expose_event()

```
void on_ternary_scale_area_expose_event ( GtkWidget* widget, GdkEventExpose* event, gpointer
user_data )
```

### N.3.36 Global Function on_timestep_spin_value_changed()

```
void on_timestep_spin_value_changed ( GtkWidget* timestep_spin, gpointer user_data )
```

### N.3.37 Global Function on_transform_activate()

```
void on_transform_activate ( GtkWidget* widget, gpointer user_data )
```

### N.3.38 Global Function on_vector_auto_checkbutton_toggled()

```
void on_vector_auto_checkbutton_toggled ( GtkWidget* thebutton, gpointer user_data )
```

### N.3.39 Global Function on_vector_max_entry_changed()

```
void on_vector_max_entry_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.40 Global Function on_vector_scale_area_expose_event()

```
void on_vector_scale_area_expose_event ( GtkWidget* widget, GdkEventExpose* event, gpointer
user_data )
```

### N.3.41 Global Function on_vector_symm_spinbutton_changed()

```
void on_vector_symm_spinbutton_changed ( GtkWidget* theentry, gpointer user_data )
```

### N.3.42 Global Function refresh_stepnames()

This loads the names of the files into a long list.

```
int refresh_stepnames ( void )
```

### N.3.43 Global Function render_dataviews()

```
int render_dataviews ( void )
```

### N.3.44 Global Function square_to_triangle()

```
void square_to_triangle ( void )
```

### N.3.45 Global Function triangle_to_square()

```
void triangle_to_square ( void )
```

# O File 3dgf.c

### RCS Header

This is a neat 3-D graphing application of Illuminator. Just put whatever function you like down in line 110 or so (or graph the examples provided), or run with -twodee and use PETSc's native 2-D graphics (though that would be BORING!). You might want to run it as:

```
./3dgf -da_grid_x 50 -da_grid_y 50 -da_grid_z 50
```

and hit return to end.

## Included Files

```
#include "illuminator.h"                                    (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt, args...  )

#define __FUNCT__ "function_3d"

#define __FUNCT__ "function_2d"

#define __FUNCT__ "main"
```

## O.1 Variables

### O.1.1 Local Variables

**help**

```
static char help[]
```

## O.2 Functions

### O.2.1 Global Function main()

The usual main function.

```
int main ( int argc, char** argv )
```

- `int main` Returns 0 or error.
- `int argc` Number of args.
- `char** argv` The args.

The program first calls `PetscInitialize()` and creates the distributed arrays. Note that even though this program doesn't do any communication between the CPUs, illuminator must do so in order to make the isoquants at CPU boundaries, so the stencil width must be at least one.
Next it gets the distributed array's local corner and global size information. It gets the global vector, and loops over the part stored on this CPU to set all of the function values, using `function_3d()` or `function_2d()` depending on whether the `-twodee` command line switch was used at runtime.

It then uses `GeomviewBegin()` or `PetscViewerDrawOpen()` to start the viewer, and either `DATriangulate()` and `GeomviewDisplayTriangulation()` or `VecView()` to display the solution.
Finally, it prompts the user to hit <return> before wrapping up.

### O.2.2 Local Function function_2d()

This is where you put the 2-D function you'd like to graph using PETSc's native 2-D "contour" color graphics.

```
static inline PetscScalar function_2d ( PetscScalar x, PetscScalar y )
```

- `PetscScalar function_2d` It returns the function value.
- `PetscScalar x` The $x$-coordinate at which to calculate the function value.
- `PetscScalar y` The $y$-coordinate at which to calculate the function value.

### O.2.3 Local Function function_3d()

This is where you put the 3-D function you'd like to graph, or the 2-D function you'd like to graph in 3-D using the zero contour of $f(x, y) - z$.

```
static inline PetscScalar function_3d ( PetscScalar x, PetscScalar y, PetscScalar z )
```

- `PetscScalar function_3d` It returns the function value.
- `PetscScalar x` The $x$-coordinate at which to calculate the function value.
- `PetscScalar y` The $y$-coordinate at which to calculate the function value.
- `PetscScalar z` The $z$-coordinate at which to calculate the function value.

# P File chts.c

### RCS Header

This is the Cahn Hilliard timestepping code. It is provided here as an example usage of the Illuminator Distributed Visualization Library.

## Included Files

```
#include <stdlib.h>

#include "cahnhill.h"                                          (Section R)
   #include </usr/lib/petsc/include/petscts.h>
   #include </usr/lib/petsc/include/petscda.h>

#include "illuminator.h"                                       (Section C)
   #include </usr/lib/petsc/include/petscda.h>
   #include </usr/include/glib-2.0/glib.h>
   #include </usr/include/gtk-2.0/gtk/gtk.h>
   #include <X11/Xlib.h>
   #include <Imlib2.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt, args...  )

#define __FUNCT__ "ch_ts_monitor"
```

```
#define __FUNCT__ "main"
```

```
#define __FUNCT__ "FormInitialCondition"
```

```
#define __FUNCT__ "InitializeProblem"
```

## P.1 Variables

### P.1.1 Variable Surf

```
ISurface Surf
```

### P.1.2 Variable Disp

```
IDisplay Disp
```

### P.1.3 Local Variables

**help**
```
static char help[]
```

## P.2 Functions

### P.2.1 Global Function FormInitialCondition()

Like it says, put together the initial condition.

```
int FormInitialCondition ( AppCtx* user, Vec X )
```

- `int FormInitialCondition` Returns zero or error.
- `AppCtx* user` The user context structure.
- `Vec X` Vector in which to place the initial condition.

### P.2.2 Global Function InitializeProblem()

This takes the gory details of initialization out of the way (importing parameters into the user context, etc.).

```
int InitializeProblem ( AppCtx* user, Vec* xvec )
```

- `int InitializeProblem` Returns zero or error.
- `AppCtx* user` The user context to fill.
- `Vec* xvec` Vector into which to put the initial condition.

### P.2.3 Global Function ch__ts__jacobian__2d()

Wrapper for `ch_jacobian_2d()` in `cahnhill.c`.

```
int ch_ts_jacobian_2d ( TS thets, PetscScalar time, Vec X, Mat* A, Mat* B, MatStructure* flag,
void* ptr )
```

- `int ch_ts_jacobian_2d` Usual return: zero or error.
- `TS thets` Timestepping context, ignored here.
- `PetscScalar time` Current time, also ignored.
- `Vec X` Current solution vector.
- `Mat* A` Place to put the new Jacobian.
- `Mat* B` Place to put the new conditioning matrix.
- `MatStructure* flag` Flag describing the volatility of the structure.
- `void* ptr` User data pointer.

### P.2.4 Global Function ch_ts_jacobian_3d()

Wrapper for `ch_jacobian_3d()` in `cahnhill.c`.

```
int ch_ts_jacobian_3d ( TS thets, PetscScalar time, Vec X, Mat* A, Mat* B, MatStructure* flag,
void* ptr )
```

- `int ch_ts_jacobian_3d` Usual return: zero or error.
- `TS thets` Timestepping context, ignored here.
- `PetscScalar time` Current time, also ignored.
- `Vec X` Current solution vector.
- `Mat* A` Place to put the new Jacobian.
- `Mat* B` Place to put the new conditioning matrix.
- `MatStructure* flag` Flag describing the volatility of the structure.
- `void* ptr` User data pointer.

### P.2.5 Global Function ch_ts_monitor()

Monitor routine which displays the current state, using `Illuminator`'s `geomview` front-end (unless -no_contours is used); and also saves it using `IlluMultiSave()` if -save_data is specified.

```
int ch_ts_monitor ( TS thets, int stepno, PetscScalar time, Vec X, void* ptr )
```

- `int ch_ts_monitor` Usual return: zero or error.
- `TS thets` Timestepping context, ignored here.
- `int stepno` Current time step number.
- `PetscScalar time` Current time.
- `Vec X` Vector of current solved field values.
- `void* ptr` User data pointer.

### P.2.6 Global Function ch_ts_residual_vector_2d()

Wrapper for `ch_residual_vector_2d()` in `cahnhill.c`.

```
int ch_ts_residual_vector_2d ( TS thets, PetscScalar time, Vec X, Vec F, void* ptr )
```

- `int ch_ts_residual_vector_2d` Usual return: zero or error.
- `TS thets` Timestepping context, ignored here.
- `PetscScalar time` Current time, also ignored.
- `Vec X` Current solution vector.
- `Vec F` Function vector to return.
- `void* ptr` User data pointer.

### P.2.7 Global Function ch_ts_residual_vector_3d()

Wrapper for `ch_residual_vector_3d()` in `cahnhill.c`.

```
int ch_ts_residual_vector_3d ( TS thets, PetscScalar time, Vec X, Vec F, void* ptr )
```

- `int ch_ts_residual_vector_3d` Usual return: zero or error.
- `TS thets` Timestepping context, ignored here.
- `PetscScalar time` Current time, also ignored.
- `Vec X` Current solution vector.
- `Vec F` Function vector to return.
- `void* ptr` User data pointer.

### P.2.8 Global Function main()

The usual main function.

```
int main ( int argc, char** argv )
```

- `int main` Returns 0 or error.
- `int argc` Number of args.
- `char** argv` The args.

# Q File cahnhill.c

**RCS Header**

This file contains the heart of the Cahn-Hilliard formulation, in particular the functions which build the finite difference residuals and Jacobian.

## Included Files

```
#include "cahnhill.h"                                                    (Section R)
   #include </usr/lib/petsc/include/petscts.h>
   #include </usr/lib/petsc/include/petscda.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define PSIPRIME_FLOPS 5

#define PSIDOUBLEPRIME_FLOPS 8

#define RESIDUAL_FLOPS_2D

#define RESIDUAL_FLOPS_3D
```

## Q.1 Functions

### Q.1.1 Global Function ch_jacobian_2d()

This computes the Jacobian matrix at each iteration, starting with the alpha term which is pre-computed at the beginning by `ch_jacobian_alpha_2d()`.

```
int ch_jacobian_2d ( Vec X, Mat* A, Mat* B, MatStructure* flag, void* ptr )
```

- `int ch_jacobian_2d` It returns 0 or an error code.
- `Vec X` The vector of unknowns.
- `Mat* A` The Jacobian matrix returned to PETSc.
- `Mat* B` The matrix preconditioner, in this case the same matrix.
- `MatStructure* flag` Flag saying the nonzeroes are the same each time.
- `void* ptr` Application context structure.

### Q.1.2 Global Function ch_jacobian_3d()

This computes the Jacobian matrix at each iteration, starting with the alpha term which is pre-computed at the beginning by `ch_jacobian_alpha_3d()`.

```
int ch_jacobian_3d ( Vec X, Mat* A, Mat* B, MatStructure* flag, void* ptr )
```

- `int ch_jacobian_3d` — It returns 0 or an error code.
- `Vec X` — The vector of unknowns.
- `Mat* A` — The Jacobian matrix returned to PETSc.
- `Mat* B` — The matrix preconditioner, in this case the same matrix.
- `MatStructure* flag` — Flag saying the nonzeroes are the same each time.
- `void* ptr` — Application context structure.

### Q.1.3 Global Function ch_jacobian_alpha_2d()

This creates the initial alpha and J matrices, where alpha is the alpha term component of the Jacobian. Since the alpha term is linear, this part of the Jacobian need only be calculated once.

```
int ch_jacobian_alpha_2d ( AppCtx* user )
```

- `int ch_jacobian_alpha_2d` — It returns zero or an error message.
- `AppCtx* user` — The application context structure pointer.

### Q.1.4 Global Function ch_jacobian_alpha_3d()

This creates the initial alpha and J matrices, where alpha is the alpha term component of the Jacobian. Since the alpha term is linear, this part of the Jacobian need only be calculated once.

```
int ch_jacobian_alpha_3d ( AppCtx* user )
```

- `int ch_jacobian_alpha_3d` — It returns zero or an error message.
- `AppCtx* user` — The application context structure pointer.

### Q.1.5 Global Function ch_residual_vector_2d()

This evaluates the nonlinear finite difference approximation to the residuals $R_i$. Note that it loops on the interior points and the boundary separately, to avoid conditional statements within the double loop over the local grid indices.

```
int ch_residual_vector_2d ( Vec X, Vec F, void* ptr )
```

- `int ch_residual_vector_2d` — It returns zero or an error value.
- `Vec X` — The pre-allocated local vector of unknowns.
- `Vec F` — The pre-allocated local vector of residuals, filled by this function.
- `void* ptr` — Data passed in the application context.

### Q.1.6 Global Function ch_residual_vector_3d()

This evaluates the nonlinear finite difference approximation to the residuals $R_i$. Note that it loops on the interior points and the boundary separately, to avoid conditional statements within the double loop over the local grid indices.
Thus far, only periodic boundary conditions work.

```
int ch_residual_vector_3d ( Vec X, Vec F, void* ptr )
```

- `int ch_residual_vector_3d` — It returns zero or an error value.
- `Vec X` — The pre-allocated local vector of unknowns.

- `Vec F` The pre-allocated local vector of residuals, filled by this function.
- `void* ptr` Data passed in the application context.

### Q.1.7 Local Function ch_psidoubleprime()

This calculates the second derivative of homogeneous free energy with respect to concentration, for insertion into the Jacobian matrix. See the `ch_psiprime()` function for the definition of $\Psi$.

```
static inline PetscScalar ch_psidoubleprime ( PetscScalar C, PetscScalar mparam )
```

- `PetscScalar ch_psidoubleprime` It returns the second derivative $\Psi''(C)$.
- `PetscScalar C` The concentration.
- `PetscScalar mparam` The model parameter $m$.

### Q.1.8 Local Function ch_psiprime()

This calculates the derivative of homogeneous free energy with respect to concentration, which is a component of the chemical potential. It currently uses

$$\Psi' = C(1-C)\left(\frac{1}{2} + m - C\right)$$

which gives (meta)stable equilibria at $C = 0$ and 1 and an unstable equilibrium at $C = \frac{1}{2} + m$; if $m > 0$ then the 0 phase is stable and vice versa.

```
static inline PetscScalar ch_psiprime ( PetscScalar C, PetscScalar mparam )
```

- `PetscScalar ch_psiprime` It returns the derivative itself.
- `PetscScalar C` The concentration.
- `PetscScalar mparam` The model parameter $m$.

### Q.1.9 Local Function ch_residual_2d()

This function computes the residual from indices to points in the concentration array. "Up" refers to the positive $y$-direction, "down" to negative $y$, "left" to negative $x$ and "right" to positive $x$.

```
static inline PetscScalar ch_residual_2d ( PetscScalar* conc, PetscScalar alpha, PetscScalar beta,
PetscScalar mparam, PetscScalar hx, PetscScalar hy, int upup, int upleft, int up, int upright, int
leftleft, int left, int current, int right, int rightright, int downleft, int down, int downright,
int downdown )
```

- `PetscScalar ch_residual_2d` Returns the residual itself
- `PetscScalar* conc` Array of concentrations
- `PetscScalar alpha` Model parameter $\kappa\alpha$
- `PetscScalar beta` Model parameter $\kappa\beta$
- `PetscScalar mparam` Model parameter $m$
- `PetscScalar hx` Inverse square $x$-spacing $h_x^{-2}$
- `PetscScalar hy` Inverse square $y$-spacing $h_y^{-2}$
- `int upup` Index to array position two cells up from current
- `int upleft` Index to array position one cell up and one left from current
- `int up` Index to array position one cell up from current
- `int upright` Index to array position one cell up and one right from current
- `int leftleft` Index to array position two cells left of current
- `int left` Index to array position one cell left of current

- `int current` Index to current cell array position
- `int right` Index to array position one cell right of current
- `int rightright` Index to array position two cells right of current
- `int downleft` Index to array position one cell down and one left from current
- `int down` Index to array position one cell down from current
- `int downright` Index to array position one cell down and one right from current
- `int downdown` Index to array position two cells down from current

This calculates the $\beta$-term, $\kappa\beta$ times the Laplacian of $\Psi'(C)$,
then subtracts the $\alpha$-term, $\kappa\alpha\nabla^2\nabla^2 C$.

### Q.1.10 Local Function ch_residual_3d()

This function computes the residual from indices to points in the concentration array. "Front refers to the positive $z$-direction, "back" to negative $z$, "up" to positive $y$, "down" to negative $y$, "left" to negative $x$ and "right" to positive $x$.

```
static inline PetscScalar ch_residual_3d ( PetscScalar* conc, PetscScalar alpha, PetscScalar beta,
PetscScalar mparam, PetscScalar hx, PetscScalar hy, PetscScalar hz, int frontfront, int frontup,
int frontleft, int front, int frontright, int frontdown, int upup, int upleft, int up, int upright,
int leftleft, int left, int current, int right, int rightright, int downleft, int down, int
downright, int downdown, int backup, int backleft, int back, int backright, int backdown, int
backback )
```

- `PetscScalar ch_residual_3d` Returns the residual itself
- `PetscScalar* conc` Array of concentrations
- `PetscScalar alpha` Model parameter $\kappa\alpha$
- `PetscScalar beta` Model parameter $\kappa\beta$
- `PetscScalar mparam` Model parameter $m$
- `PetscScalar hx` Inverse square $x$-spacing $h_x^{-2}$
- `PetscScalar hy` Inverse square $y$-spacing $h_y^{-2}$
- `PetscScalar hz` Inverse square $z$-spacing $h_z^{-2}$
- `int frontfront` Index to array position two cells in front of current
- `int frontup` Index to array position one cell front and one up from current
- `int frontleft` Index to array position one cell front and one left from current
- `int front` Index to array position one cell in front of current
- `int frontright` Index to array position one cell front and one right from current
- `int frontdown` Index to array position one cell front and one down from current
- `int upup` Index to array position two cells up from current
- `int upleft` Index to array position one cell up and one left from current
- `int up` Index to array position one cell up from current
- `int upright` Index to array position one cell up and one right from current
- `int leftleft` Index to array position two cells left of current
- `int left` Index to array position one cell left of current
- `int current` Index to current cell array position
- `int right` Index to array position one cell right of current
- `int rightright` Index to array position two cells right of current
- `int downleft` Index to array position one cell down and one left from current

| | |
|---|---|
| · `int down` | Index to array position one cell down from current |
| · `int downright` | Index to array position one cell down and one right from current |
| · `int downdown` | Index to array position two cells down from current |
| · `int backup` | Index to array position one cell back and one up from current |
| · `int backleft` | Index to array position one cell back and one left from current |
| · `int back` | Index to array position one cell in back of current |
| · `int backright` | Index to array position one cell back and one right from current |
| · `int backdown` | Index to array position one cell back and one down from current |
| · `int backback` | Index to array position two cells in back of current |

This calculates the $\beta$-term, $\kappa\beta$ times the Laplacian of $\Psi'(C)$,
then subtracts the $\alpha$-term, $\kappa\alpha\nabla^2\nabla^2 C$.

# R File cahnhill.h

### RCS Header

Common files for cahnhill.c and programs which use it (e.g. chts.c), based on PETSc SNES tutorial common8and9.h.

## Included Files

```
#include </usr/lib/petsc/include/petscts.h>
#include </usr/lib/petsc/include/petscda.h>
```

## Preprocessor definitions

```
#define ORBIT2 1
#define _REENTRANT 1
#define IMLIB2_EXISTS 1
#define CAHNHILL_H
#define C( i )
```

## R.1 Type definitions

### R.1.1 Typedef AppCtx

User-defined application context for chts.c - contains data needed by the application-provided callbacks: ch_residual_vector_xd() (x is 2 or 3).

```
typedef struct {...} AppCtx
struct
 {
   PetscTruth threedee;
   PetscScalar kappa;
   PetscScalar epsilon;
   PetscScalar gamma;
   PetscScalar mparam;
   int mx;
   int my;
   int mz;
   int mc;
   int chvar;
   Vec localX;
```

```
  Vec localF;
  DA da;
  int rank;
  int size;
  MPI_Comm comm;
  int ilevel;
  int nlevels;
  Vec x_old;
  Mat J;
  Mat alpha;
  DAPeriodicType period;
  ISLocalToGlobalMapping isltog;
  PetscViewer theviewer;
  char** label;
  PetscTruth print_grid;
  PetscTruth print_vecs;
  PetscTruth no_contours;
  PetscTruth random;
  PetscTruth save_data;
  int load_data;
}
```

# S File chui.c

### RCS Header

This is a Glade front end to the little phase field/fluid structure interactions program based on PETSc, but can serve as a front end to a variety of PETSc programs with minor adjustments.
Callback functions are grouped here according to where they appear in the main window (except on_run_clicked is below main window items), then the run control dialog, and save dialog.
I haven't put function comments in to keep the line count down, the functions are generally pretty simple GTK+ callbacks.

## Included Files

```
#include </usr/include/libglade-2.0/glade/glade.h>

#include </usr/include/libgnomeui-2.0/gnome.h>

#include </usr/include/libgnomeui-2.0/libgnomeui/libgnomeui.h>

#include <stdio.h>

#include <math.h>
```

## Preprocessor definitions

```
#define ORBIT2 1

#define _REENTRANT 1

#define IMLIB2_EXISTS 1

#define DPRINTF( fmt...  )

#define MAX_COMMAND_LINE_OPTIONS 22
```

## S.1 Variables

### S.1.1 Variable xml

```
GladeXML* xml
```

### S.1.2 Variable simulation_input_file

```
FILE* simulation_input_file
```

### S.1.3 Variable simulation_output_file

```
FILE* simulation_output_file
```

### S.1.4 Variable Lx

```
double Lx
```

### S.1.5 Variable nx

```
int nx
```

### S.1.6 Variable dt

```
double dt
```

### S.1.7 Variable dt_factor

```
double dt_factor
```

### S.1.8 Variable dt_max

```
double dt_max
```

### S.1.9 Variable last_tstep

```
int last_tstep
```

### S.1.10 Variable display_x

```
gboolean display_x
```

### S.1.11 Variable display_text

```
gboolean display_text
```

### S.1.12 Variable remote_host

```
gboolean remote_host
```

### S.1.13 Variable remote_hostname

```
const gchar* remote_hostname
```

### S.1.14 Variable thetransport

```
gchar thetransport[50]
```

#### S.1.15 Variable mpirun_command

```
const gchar* mpirun_command
```

#### S.1.16 Variable number_cpus

```
int number_cpus
```

#### S.1.17 Variable twodee

```
gboolean twodee
```

#### S.1.18 Variable options_filename

```
gchar* options_filename
```

#### S.1.19 Variable pipe_input_tag

```
gint pipe_input_tag
```

#### S.1.20 Variable from_simulation_pipe

```
int from_simulation_pipe[2]
```

#### S.1.21 Variable to_simulation_pipe

```
int to_simulation_pipe[2]
```

### S.2 Functions

#### S.2.1 Global Function main()

```
int main ( int argc, char* argv[] )
```

#### S.2.2 Global Function on_2d_activate()

```
void on_2d_activate ( GtkWidget* unused, gpointer user_data )
```

#### S.2.3 Global Function on_3d_activate()

```
void on_3d_activate ( GtkWidget* unused, gpointer user_data )
```

#### S.2.4 Global Function on_about_activate()

```
void on_about_activate ( GtkWidget* null_widget, gpointer user_data )
```

#### S.2.5 Global Function on_last_timestep_changed()

```
void on_last_timestep_changed ( GtkWidget* last_timestep, gpointer user_data )
```

#### S.2.6 Global Function on_load_ok_clicked()

```
void on_load_ok_clicked ( GtkWidget* load_file, gpointer user_data )
```

#### S.2.7 Global Function on_max_timestep_changed()

```
void on_max_timestep_changed ( GtkWidget* max_timestep, gpointer user_data )
```

### S.2.8 Global Function on_mpirun_changed()

```
void on_mpirun_changed ( GtkWidget* mpirun, gpointer user_data )
```

### S.2.9 Global Function on_num_cpus_changed()

```
void on_num_cpus_changed ( GtkWidget* num_cpus, gpointer user_data )
```

### S.2.10 Global Function on_pause_clicked()

```
void on_pause_clicked ( GtkWidget* forgot, gpointer user_data )
```

### S.2.11 Global Function on_remote_check_toggled()

```
void on_remote_check_toggled ( GtkWidget* remote_check, gpointer user_data )
```

### S.2.12 Global Function on_remote_host_changed()

```
void on_remote_host_changed ( GtkWidget* remote_host, gpointer user_data )
```

### S.2.13 Global Function on_resolution_changed()

```
void on_resolution_changed ( GtkWidget* resolution, gpointer user_data )
```

### S.2.14 Global Function on_rsh_item_activate()

```
void on_rsh_item_activate ( GtkWidget* widget, gpointer user_data )
```

### S.2.15 Global Function on_run_activate()

```
void on_run_activate ( GtkWidget* null_widget, gpointer user_data )
```

### S.2.16 Global Function on_save_ok_clicked()

```
void on_save_ok_clicked ( GtkWidget* save_file, gpointer user_data )
```

### S.2.17 Global Function on_show_options_activate()

```
void on_show_options_activate ( GtkWidget* null_widget, gpointer user_data )
```

### S.2.18 Global Function on_show_options_toggled()

```
void on_show_options_toggled ( GtkWidget* show_options, gpointer user_data )
```

### S.2.19 Global Function on_show_output_activate()

```
void on_show_output_activate ( GtkWidget* null_widget, gpointer user_data )
```

### S.2.20 Global Function on_show_output_toggled()

```
void on_show_output_toggled ( GtkWidget* show_output, gpointer user_data )
```

### S.2.21 Global Function on_ssh_item_activate()

```
void on_ssh_item_activate ( GtkWidget* widget, gpointer user_data )
```

### S.2.22 Global Function on_stop_clicked()

```
void on_stop_clicked ( GtkWidget* output_window, gpointer user_data )
```

### S.2.23 Global Function on_textdisplay_toggled()

```
void on_textdisplay_toggled ( GtkWidget* textdisplay, gpointer user_data )
```

### S.2.24 Global Function on_time_factor_changed()

```
void on_time_factor_changed ( GtkWidget* time_factor, gpointer user_data )
```

### S.2.25 Global Function on_timestep_changed()

```
void on_timestep_changed ( GtkWidget* timestep, gpointer user_data )
```

### S.2.26 Global Function on_width_changed()

```
void on_width_changed ( GtkWidget* width, gpointer user_data )
```

### S.2.27 Global Function on_xdisplay_toggled()

```
void on_xdisplay_toggled ( GtkWidget* xdisplay, gpointer user_data )
```

### S.2.28 Global Function open_params()

```
void open_params ( GtkWidget* null_widget, gpointer user_data )
```

### S.2.29 Global Function read_simulation_data()

```
void read_simulation_data ( gpointer user_data, gint source, GdkInputCondition condition )
```

### S.2.30 Global Function save_params()

```
void save_params ( GtkWidget* null_widget, gpointer user_data )
```

### S.2.31 Global Function save_params_as()

```
void save_params_as ( GtkWidget* null_widget, gpointer user_data )
```

# T File config.h

## Preprocessor definitions

```
#define ORBIT2 1
#define _REENTRANT 1
#define IMLIB2_EXISTS 1
#define HAVE_DLFCN_H 1
#define HAVE_INTTYPES_H 1
#define HAVE_LIBPETSC 1
#define HAVE_MEMORY_H 1
#define HAVE_READLINE_HISTORY_H 1
#define HAVE_READLINE_READLINE_H 1
#define HAVE_STDINT_H 1
#define HAVE_STDLIB_H 1
#define HAVE_STRINGS_H 1
```

```
#define HAVE_STRING_H 1
#define HAVE_SYS_STAT_H 1
#define HAVE_SYS_TYPES_H 1
#define HAVE_UNISTD_H 1
#define PACKAGE "illuminator"
#define PACKAGE_BUGREPORT ""
#define PACKAGE_NAME ""
#define PACKAGE_STRING ""
#define PACKAGE_TARNAME ""
#define PACKAGE_VERSION ""
#define STDC_HEADERS 1
#define VERSION "0.11.0"
```