The GridFire Fire Behavior Model

Gary W. Johnson, Ph.D., David Saah, Ph.D., Max Moritz, Ph.D., Kenneth Cheung

1 Preface

This document is a Literate Program[1], containing both the source code of the software it describes as well as the rationale used in each step of its design and implementation. The purpose of this approach is to enable anyone sufficiently experienced in programming to easily retrace the author's footsteps as they read through the text and code. By the time they have reached the end of this document, the reader should have just as strong a grasp of the system as the original programmer.

To execute the code illustrated within this document, you will need to install several pieces of software, all of which are open source and/or freely available for all major operating systems. These programs are listed in Table 1 along with their minimum required versions and URLs from which they may be downloaded.

Table 1: Software necessary to evaluate the code in this document

Name	Version	URL
Java Development Kit	11+	https://jdk.java.net
Clojure CLI Tools	1.10+	https://clojure.org/guides/getting_started
Postgresql	10+	https://www.postgresql.org/download
PostGIS	3+	https://postgis.net/install
GDAL	3+	https://gdal.org

GridFire is written in the Clojure programming language[2], which is a modern dialect of Lisp hosted on the Java Virtual Machine.(Hickey, 2008) As a result, a Java Development Kit is required to compile and run the code shown throughout this document.

The Clojure CLI tools are used to download required libraries and provide a code evaluation prompt (a.k.a. REPL) into which we will enter the code making up this fire model.

Postgresql (along with the PostGIS spatial extensions) will be used to load and serve raster-formatted GIS layers to the GridFire program. Although it is beyond the scope of this document, PostGIS[3] provides a rich API for manipulating both raster and vector layers through SQL.

[1] https://en.wikipedia.org/wiki/Literate_programming
[2] https://clojure.org
[3] https://postgis.net
[4] https://www.eclipse.org/legal/epl-2.0/

2 Setting Up the Clojure Environment

Because Clojure is implemented on the Java Virtual Machine (JVM), we must explicitly list all of the libraries used by our program on the Java classpath. Fortunately, the Clojure CLI tools can handle downloading and storing these libraries as well as making them available to the Clojure process at runtime. However, in order for Clojure to know which libraries are needed, we must first create its build configuration file, called "deps.edn", and place it in the directory from which we will call our Clojure program. The complete deps.edn for the current GridFire version is shown below.

```
{:paths ["src" "resources"]

 :deps {cnuernber/dtype-next                {:mvn/version "9.000"}
        commons-codec/commons-codec         {:mvn/version "1.15"}
        com.nextjournal/beholder            {:mvn/version "1.0.0"}
        com.taoensso/tufte                  {:mvn/version "2.2.0"}
        hikari-cp/hikari-cp                 {:mvn/version "2.13.0"}
        manifold/manifold                   {:mvn/version "0.2.4"}
        org.apache.commons/commons-compress {:mvn/version "1.21"}
        org.clojars.lambdatronic/matrix-viz {:mvn/version "2022.02.03"}
        org.clojure/clojure                 {:mvn/version "1.10.3"}
        org.clojure/core.async              {:mvn/version "1.3.622"}
        org.clojure/data.csv                {:mvn/version "1.0.0"}
        org.clojure/data.json               {:mvn/version "2.4.0"}
        org.clojure/java.jdbc               {:mvn/version "0.7.12"}
        org.clojure/tools.cli               {:mvn/version "1.0.206"}
        org.postgresql/postgresql           {:mvn/version "42.2.23"}
        sig-gis/magellan                    {:git/url "https://github.com/sig-gis/magellan" :sha "6730285f033704dd9cd863535aaf94b11
        sig-gis/triangulum                  {:git/url "https://github.com/sig-gis/triangulum"
                                             :sha     "5b179a97ebd8fbcbff51776db06d9770cb649b9d"}
        vvvvalvalval/supdate                {:mvn/version "0.2.3"}}

 :mvn/repos {"osgeo" {:url "https://repo.osgeo.org/repository/release/"}}

 :aliases {:build             {:deps       {io.github.clojure/tools.build
                                            {:git/tag "v0.8.3" :git/sha "0d20256"}}
                               :ns-default build}
           :with-tools-build  {:extra-deps {io.github.clojure/tools.build
                                            {:git/tag "v0.8.3" :git/sha "0d20256"}}}
           :build-test-db     {:extra-paths ["test"]
                               :main-opts   ["-m" "gridfire.build-test-db"]}
           :run               {:main-opts ["-m" "gridfire.cli"]
                               :jvm-opts  ["-XX:MaxRAMPercentage=90"]}
           :repl              {:main-opts ["-e" "(require,'gridfire.core)"
                                           "-e" "(in-ns,'gridfire.core)"
                                           "-r"]}
           :vsampling         {:jvm-opts ["-Dgridfire.utils.vsampling.enabled=true"]}
           :gen-raster        {:main-opts ["-m" "gridfire.gen-raster"]}
           :test              {:extra-paths ["test"]
                               :extra-deps  {com.cognitect/test-runner
                                             {:git/url "https://github.com/cognitect-labs/test-runner.git"
                                              :sha     "dd6da11611eeb87f08780a30ac8ea6012d4c05ce"}}
                               :main-opts   ["-e" "(do,(set!,*warn-on-reflection*,true),nil)"
                                             "-m" "cognitect.test-runner"]}
           :test-unit         {:extra-paths ["test"]
                               :extra-deps  {com.cognitect/test-runner
                                             {:git/url "https://github.com/cognitect-labs/test-runner.git"
                                              :sha     "dd6da11611eeb87f08780a30ac8ea6012d4c05ce"}}
                               :main-opts   ["-e" "(do,(set!,*warn-on-reflection*,true),nil)"
                                             "-m" "cognitect.test-runner"
                                             "--include" ":unit"]}
           ;; A moderately fast high-coverage test suite to be run before submitting code, using command:
           ;; clojure -M:test-ci
           :test-ci           {:extra-paths ["test"]
                               :extra-deps  {com.cognitect/test-runner
                                             {:git/url "https://github.com/cognitect-labs/test-runner.git"
```

```
                                    :sha     "dd6da11611eeb87f08780a30ac8ea6012d4c05ce"}}
                     :main-opts   ["-e" "(do,(set!,*warn-on-reflection*,true),nil)"
                                   "-m" "cognitect.test-runner"
                                   "--include" ":unit"
                                   "--include" ":simulation"
                                   "--exclude" ":database"
                                   "--exclude" ":canonical"]}
      :perf-testing      {:extra-paths ["test"]
                          :extra-deps  {com.clojure-goes-fast/clj-async-profiler  {:mvn/version "0.5.1"}
                                        com.clojure-goes-fast/clj-java-decompiler {:mvn/version "0.3.1"}
                                        criterium/criterium                       {:mvn/version "0.4.5"}}}
      :check-reflections {:extra-paths ["test"]
                          :main-opts   ["-e" "(do,(set!,*warn-on-reflection*,true),nil)"
                                        "-e" "(require,'gridfire.cli)"
                                        "-e" "(require,'gridfire.server2.socket)"
                                        "-e" "(require,'gridfire.build-test-db)"]}
      :check-deps        {:extra-deps {olical/depot {:mvn/version "2.3.0"}}
                          :main-opts  ["-m" "depot.outdated.main"]}}}
```

Once this file is created, we need to instruct Clojure to download these library dependencies and then run the built-in test suite to verify that GridFire compiles and runs as expected on our local computer.

Before we run the tests, we'll need to set up a test database and import some rasters into it. We will be prompted for the postgres and gridfire_test users' passwords. The postgres user's password will be whatever it is when we set up Postgresql. For the gridfire_test user's password, refer to "src/sql/create_test_db.sql". The default value is simply "gridfire_test".

The following command builds the test database:

```
clojure -M:build-test-db
```

Once that has completed, you can run the following command to launch the test suite:

```
clojure -M:test
```

3 Setting Up the PostGIS Database

GridFire may make use of any raster-formatted GIS layers that are loaded into a PostGIS database. Therefore, we must begin by creating a spatially-enabled database on our local Postgresql server.

When installing Postgresql, we should have been prompted to create an initial superuser called **postgres**, who has full permissions to create new databases and roles. We can log into the Postgresql server as this user with the following **psql** command.

```
psql -U postgres
```

Once logged in, we issue the following commands to first create a new database role and to then create a new database (owned by this role) in which to store our raster data. Finally, we import the PostGIS spatial extensions into the new database.

```
CREATE ROLE gridfire WITH LOGIN CREATEDB;
CREATE DATABASE gridfire WITH OWNER gridfire;
\c gridfire
CREATE EXTENSION postgis;
```

4 Importing Rasters into the Database

Whenever we want to add a new raster-formatted GIS layer to our database, we can simply issue the **raster2pgsql** command as follows, replacing the raster name and table name to match our own datasets.

```
SRID=4326
RASTER=dem.tif
TABLE=dem
DATABASE=gridfire
raster2pgsql -s $SRID $RASTER $TABLE | psql $DATABASE
```

Note: The raster2pgsql command has several useful command line options, including automatic tiling of the raster layer in the database, creating fast spatial indeces after import, or setting raster constraints on the newly created table. Run **raster2pgsql -?** from the command line for more details.

Here's an example shell script that will tile multiple large rasters (asp.tif, cbd.tif, cbh.tif, etc) into 100x100 tiles and import them into our database.

Note: Here we specified a schema (e.g, landfire) along with the table name so as to match the sample config file in "resources/sample_config.edn".

First create the schema in our database.

```
CREATE SCHEMA landfire;
```

Then we can use the following script to import LANDFIRE layers into our database given the username and schema as inputs.

Note: This script needs to be run in the same folder as where these rasters reside. The filenames of these rasters should match the elements in the for loop (i.e. asp.tif, cbd.tif etc)

```
#!/usr/bin/env bash

USERNAME=$1
SCHEMA=$2
SRID=$3

for LAYER in asp cbd cbh cc ch dem fbfm13 fbfm40 slp
do
    raster2pgsql -t auto -I -C -s $SRID $LAYER.tif $SCHEMA.$LAYER | psql -h localhost -U $USERNAME
done
```

To run the script, give it our username, schema, and srid we wish the layers to have.

```
sh import_landfire_rasters.sh gridfire landfire 90914
```

Whenever we want to add a new spatial reference system to our database, we can insert a record into our spatial_ref_sys table.

```
INSERT INTO public.spatial_ref_sys (srid, auth_name, auth_srid, srtext, proj4text)
VALUES (900914, 'user-generated', 900914,
        'PROJCS["USA_Contiguous_Albers_Equal_Area_Conic_USGS_version",' ||
        'GEOGCS["NAD83",' ||
        'DATUM["North_American_Datum_1983",' ||
        'SPHEROID["GRS 1980",6378137,298.2572221010002,' ||
        'AUTHORITY["EPSG","7019"]],' ||
        'AUTHORITY["EPSG","6269"]],' ||
        'PRIMEM["Greenwich",0],' ||
        'UNIT["degree",0.0174532925199433],' ||
        'AUTHORITY["EPSG","4269"]],' ||
```

```
        'PROJECTION["Albers_Conic_Equal_Area"],' ||
        'PARAMETER["standard_parallel_1",29.5],' ||
        'PARAMETER["standard_parallel_2",45.5],' ||
        'PARAMETER["latitude_of_center",23],' ||
        'PARAMETER["longitude_of_center",-96],' ||
        'PARAMETER["false_easting",0],' ||
        'PARAMETER["false_northing",0],' ||
        'UNIT["metre",1,' ||
        'AUTHORITY["EPSG","9001"]]]',
        '+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=23 +lon_0=-96 +x_0=0 +y_0=0' ||
        ' +datum=NAD83 +units=m +no_defs');
```

We may also want to import initial ignition rasters into our database. We can do so with a similar script as importing LANDFIRE rasters.

First create a new schema.

```
CREATE SCHEMA ignition;
```

Then we can use the following script to import an ignition raster into our database given the schema and username as inputs.

Note: This script needs to be run in the same folder as where this raster resides. The filename of this raster should match the value assigned to the LAYER variable (i.e., ign) plus a .tif extension.

```
#!/usr/bin/env bash

USERNAME=$1
SCHEMA=$2
SRID=$3

LAYER="ign"
raster2pgsql -I -C -t auto -s $SRID $LAYER.tif $SCHEMA.$LAYER | psql -h localhost -U $USERNAME
```

To run the script, give it the username, schema name, and srid we wish the layers to have.

```
sh import_ignition_rasters.sh gridfire ignition 90014
```

We may also want to import weather rasters into our database. We can do so with a similar script as importing LANDFIRE rasters.

First create a new schema.

```
CREATE SCHEMA weather;
```

Then we can use the following script to import weather rasters into our database given the schema and username as inputs.

Note: This script needs to be run in the same folder as where this rasters resides. The filename of these rasters should match the elements in the for loop (i.e. tmpf_to_sample.tif)

```
#!/usr/bin/env bash

USERNAME=$1
SCHEMA=$2
SRID=$3
TILING=$4

for LAYER in tmpf wd ws rh
```

```
do
    if [ -z "$TILING" ]
    then
        raster2pgsql -I -C -t auto -s $SRID ${LAYER}_to_sample.tif $SCHEMA.$LAYER | psql -h localhost -U $USERNAME
    else
        raster2pgsql -I -C -t $TILING -s $SRID ${LAYER}_to_sample.tif $SCHEMA.$LAYER | psql -h localhost -U $USERNAME

    fi
done
```

To run the script, give it the username, schema name, and srid we wish the layers to have.

```
sh import_weather_rasters.sh gridfire weather 90014
```

You may optionally include a fourth argument to set the tiling (defaults to auto).

```
sh import_weather_rasters.sh gridfire weather 90014 800x800
```

Note: This script needs to be run in the same folder as where these rasters reside.

5 Fire Spread Model

GridFire implements the following fire behavior formulas from the fire science literature:

- Surface Fire Spread: Rothermel 1972 with FIREMODS adjustments from Albini 1976
- Crown Fire Initiation: Van Wagner 1977
- Passive/Active Crown Fire Spread: Cruz 2005
- Flame Length and Fire Line Intensity: Byram 1959
- Midflame Wind Adjustment Factor: Albini & Baughman 1979 parameterized as in BehavePlus, FARSITE, FlamMap, FSPro, and FPA according to Andrews 2012
- Fire Spread on a Raster Grid: Morais 2001 (method of adaptive timesteps and fractional distances)
- Spot Fire: Perryman 2013

The following fuel models are supported:

- Anderson 13: no dynamic loading
- Scott & Burgan 40: dynamic loading implemented according to Scott & Burgan 2005

The method used to translate linear fire spread rates to a 2-dimensional raster grid were originally developed by Marco Morais at UCSB as part of his HFire system.(Peterson et al., 2011, 2009, Morais, 2001) Detailed information about this software, including its source code and research article references can be found here:

http://firecenter.berkeley.edu/hfire/about.html

Outputs from GridFire include fire size (ac), fire line intensity (Btu/ft/s), flame length (ft), fire volume (ac*ft), fire shape (ac/ft) and conditional burn probability (times burned/fires initiated). Fire line intensity and flame length may both be exported as either average values per fire or as maps of the individual values per burned cell.

In the following sections, we describe the operation of this system in detail.

5.1 Fuel Model Definitions

All fires ignite and travel through some form of burnable fuel. Although the effects of wind and slope on the rate of fire spread can be quite pronounced, its fundamental thermodynamic characteristics are largely determined by the fuel type in which it is sustained. For wildfires, these fuels are predominantly herbaceous and woody vegetation (both alive and dead) as well as decomposing elements of dead vegetation, such as duff or leaf litter. To estimate the heat output and rate of spread of a fire burning through any of these fuels, we must determine those physical properties that affect heat absorption and release.

Of course, measuring these fuel properties for every kind of vegetation that may be burned in a wildfire is an intractable task. To cope with this, fuels are classified into categories called "fuel models" which share similar burning characteristics. Each fuel model is then assigned a set of representative values for each of the thermally relevant physical properties shown in Table 2.

Table 2: Physical properties assigned to each fuel model

Property	Description	Units
δ	fuel depth	ft
w_o	ovendry fuel loading	lb/ft^2
σ	fuel particle surface-area-to-volume ratio	ft^2/ft^3
M_x	moisture content of extinction	lb moisture/lb ovendry wood
h	fuel particle low heat content	Btu/lb
ρ_p	ovendry particle density	lb/ft^3
S_T	fuel particle total mineral content	lb minerals/lb ovendry wood
S_e	fuel particle effective mineral content	lb silica-free minerals/lb ovendry wood
M_f	fuel particle moisture content	lb moisture/lb ovendry wood

Note: While M_f is not, in fact, directly assigned to any of these fuel models, their definitions remain incomplete for the purposes of fire spread modelling (particularly those reliant on the curing formulas of dynamic fuel loading) until it is provided as a characteristic of local weather conditions.

The fuel models supported by GridFire include the standard 13 fuel models of Rothermel, Albini, and Anderson(Anderson, 1982) and the additional 40 fuel models defined by Scott and Burgan(Scott and Burgan, 2005). In addition, these fuel model numbers are duplicated to encode their belonging to the Wildland Urban Interface (WUI), which enables making WUI-specific adjustments to fuel behavior. These are all concisely encoded in an internal data structure, which may be updated to include additional custom fuel models desired by the user.

```
(ns gridfire.fuel-models)

(set! *unchecked-math* :warn-on-boxed)

(def fuel-models
  "Lookup table including one entry for each of the Anderson 13 and
   Scott & Burgan 40 fuel models. The fields have the following
   meanings:
   {fuel-model-number
    [name delta M_x-dead h
     [w_o-dead-1hr w_o-dead-10hr w_o-dead-100hr w_o-live-herbaceous w_o-live-woody]
     [sigma-dead-1hr sigma-dead-10hr sigma-dead-100hr sigma-live-herbaceous sigma-live-woody]]
   }"
  {
   ;; Anderson 13:
   ;; Grass and Grass-dominated (short-grass,timber-grass-and-understory,tall-grass)
   1   [:R01 1.0 12 8 [0.0340 0.0000 0.0000 0.0000 0.0000] [3500.0   0.0  0.0    0.0    0.0]]
   2   [:R02 1.0 15 8 [0.0920 0.0460 0.0230 0.0230 0.0000] [3000.0 109.0 30.0 1500.0    0.0]]
   3   [:R03 2.5 25 8 [0.1380 0.0000 0.0000 0.0000 0.0000] [1500.0   0.0  0.0    0.0    0.0]]
   ;; Chaparral and Shrubfields (chaparral,brush,dormant-brush-hardwood-slash,southern-rough)
```

```
   4   [:R04 6.0 20 8 [0.2300 0.1840 0.0920 0.2300 0.0000] [2000.0 109.0 30.0 1500.0    0.0]]
   5   [:R05 2.0 20 8 [0.0460 0.0230 0.0000 0.0920 0.0000] [2000.0 109.0  0.0 1500.0    0.0]]
   6   [:R06 2.5 25 8 [0.0690 0.1150 0.0920 0.0000 0.0000] [1750.0 109.0 30.0    0.0    0.0]]
   7   [:R07 2.5 40 8 [0.0520 0.0860 0.0690 0.0170 0.0000] [1750.0 109.0 30.0 1550.0    0.0]]
   ;; Timber Litter (closed-timber-litter,hardwood-litter,timber-litter-and-understory)
   8   [:R08 0.2 30 8 [0.0690 0.0460 0.1150 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   9   [:R09 0.2 25 8 [0.1340 0.0190 0.0070 0.0000 0.0000] [2500.0 109.0 30.0    0.0    0.0]]
   10  [:R10 1.0 25 8 [0.1380 0.0920 0.2300 0.0920 0.0000] [2000.0 109.0 30.0 1500.0    0.0]]
   ;; Logging Slash (light-logging-slash,medium-logging-slash,heavy-logging-slash)
   11  [:R11 1.0 15 8 [0.0690 0.2070 0.2530 0.0000 0.0000] [1500.0 109.0 30.0    0.0    0.0]]
   12  [:R12 2.3 20 8 [0.1840 0.6440 0.7590 0.0000 0.0000] [1500.0 109.0 30.0    0.0    0.0]]
   13  [:R13 3.0 25 8 [0.3220 1.0580 1.2880 0.0000 0.0000] [1500.0 109.0 30.0    0.0    0.0]]
   ;; Nonburnable (NB)
   91  [:NB1 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
   92  [:NB2 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
   93  [:NB3 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
   98  [:NB4 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
   99  [:NB5 0.0  0 0 [0.0000 0.0000 0.0000 0.0000 0.0000] [   0.0   0.0  0.0    0.0    0.0]]
   ;; Scott & Burgan 40:
   ;; Grass (GR)
   101 [:GR1 0.4 15 8 [0.0046 0.0000 0.0000 0.0138 0.0000] [2200.0 109.0 30.0 2000.0    0.0]]
   102 [:GR2 1.0 15 8 [0.0046 0.0000 0.0000 0.0459 0.0000] [2000.0 109.0 30.0 1800.0    0.0]]
   103 [:GR3 2.0 30 8 [0.0046 0.0184 0.0000 0.0689 0.0000] [1500.0 109.0 30.0 1300.0    0.0]]
   104 [:GR4 2.0 15 8 [0.0115 0.0000 0.0000 0.0872 0.0000] [2000.0 109.0 30.0 1800.0    0.0]]
   105 [:GR5 1.5 40 8 [0.0184 0.0000 0.0000 0.1148 0.0000] [1800.0 109.0 30.0 1600.0    0.0]]
   106 [:GR6 1.5 40 9 [0.0046 0.0000 0.0000 0.1561 0.0000] [2200.0 109.0 30.0 2000.0    0.0]]
   107 [:GR7 3.0 15 8 [0.0459 0.0000 0.0000 0.2479 0.0000] [2000.0 109.0 30.0 1800.0    0.0]]
   108 [:GR8 4.0 30 8 [0.0230 0.0459 0.0000 0.3352 0.0000] [1500.0 109.0 30.0 1300.0    0.0]]
   109 [:GR9 5.0 40 8 [0.0459 0.0459 0.0000 0.4132 0.0000] [1800.0 109.0 30.0 1600.0    0.0]]
   ;; Grass-Shrub (GS)
   121 [:GS1 0.9 15 8 [0.0092 0.0000 0.0000 0.0230 0.0298] [2000.0 109.0 30.0 1800.0 1800.0]]
   122 [:GS2 1.5 15 8 [0.0230 0.0230 0.0000 0.0275 0.0459] [2000.0 109.0 30.0 1800.0 1800.0]]
   123 [:GS3 1.8 40 8 [0.0138 0.0115 0.0000 0.0666 0.0574] [1800.0 109.0 30.0 1600.0 1600.0]]
   124 [:GS4 2.1 40 8 [0.0872 0.0138 0.0046 0.1561 0.3260] [1800.0 109.0 30.0 1600.0 1600.0]]
   ;; Shrub (SH)
   141 [:SH1 1.0 15 8 [0.0115 0.0115 0.0000 0.0069 0.0597] [2000.0 109.0 30.0 1800.0 1600.0]]
   142 [:SH2 1.0 15 8 [0.0620 0.1102 0.0344 0.0000 0.1768] [2000.0 109.0 30.0    0.0 1600.0]]
   143 [:SH3 2.4 40 8 [0.0207 0.1377 0.0000 0.0000 0.2847] [1600.0 109.0 30.0    0.0 1400.0]]
   144 [:SH4 3.0 30 8 [0.0390 0.0528 0.0092 0.0000 0.1171] [2000.0 109.0 30.0 1800.0 1600.0]]
   145 [:SH5 6.0 15 8 [0.1653 0.0964 0.0000 0.0000 0.1331] [ 750.0 109.0 30.0    0.0 1600.0]]
   146 [:SH6 2.0 30 8 [0.1331 0.0666 0.0000 0.0000 0.0643] [ 750.0 109.0 30.0    0.0 1600.0]]
   147 [:SH7 6.0 15 8 [0.1607 0.2433 0.1010 0.0000 0.1561] [ 750.0 109.0 30.0    0.0 1600.0]]
   148 [:SH8 3.0 40 8 [0.0941 0.1561 0.0390 0.0000 0.1997] [ 750.0 109.0 30.0    0.0 1600.0]]
   149 [:SH9 4.4 40 8 [0.2066 0.1125 0.0000 0.0712 0.3214] [ 750.0 109.0 30.0 1800.0 1500.0]]
   ;; Timber-Understory (TU)
   161 [:TU1 0.6 20 8 [0.0092 0.0413 0.0689 0.0092 0.0413] [2000.0 109.0 30.0 1800.0 1600.0]]
   162 [:TU2 1.0 30 8 [0.0436 0.0826 0.0574 0.0000 0.0092] [2000.0 109.0 30.0    0.0 1600.0]]
   163 [:TU3 1.3 30 8 [0.0505 0.0069 0.0115 0.0298 0.0505] [1800.0 109.0 30.0 1600.0 1400.0]]
   164 [:TU4 0.5 12 8 [0.2066 0.0000 0.0000 0.0000 0.0918] [2300.0 109.0 30.0    0.0 2000.0]]
   165 [:TU5 1.0 25 8 [0.1837 0.1837 0.1377 0.0000 0.1377] [1500.0 109.0 30.0    0.0  750.0]]
   ;; Timber Litter (TL)
   181 [:TL1 0.2 30 8 [0.0459 0.1010 0.1653 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   182 [:TL2 0.2 25 8 [0.0643 0.1056 0.1010 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   183 [:TL3 0.3 20 8 [0.0230 0.1010 0.1286 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   184 [:TL4 0.4 25 8 [0.0230 0.0689 0.1928 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   185 [:TL5 0.6 25 8 [0.0528 0.1148 0.2020 0.0000 0.0000] [2000.0 109.0 30.0    0.0 1600.0]]
   186 [:TL6 0.3 25 8 [0.1102 0.0551 0.0551 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   187 [:TL7 0.4 25 8 [0.0138 0.0643 0.3719 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   188 [:TL8 0.3 35 8 [0.2663 0.0643 0.0505 0.0000 0.0000] [1800.0 109.0 30.0    0.0    0.0]]
   189 [:TL9 0.6 35 8 [0.3053 0.1515 0.1905 0.0000 0.0000] [1800.0 109.0 30.0    0.0 1600.0]]
   ;; Slash-Blowdown (SB)
   201 [:SB1 1.0 25 8 [0.0689 0.1377 0.5051 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   202 [:SB2 1.0 25 8 [0.2066 0.1951 0.1837 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   203 [:SB3 1.2 25 8 [0.2525 0.1263 0.1377 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   204 [:SB4 2.7 25 8 [0.2410 0.1607 0.2410 0.0000 0.0000] [2000.0 109.0 30.0    0.0    0.0]]
   })

(def WUI-model-number->original
```

```
  "Some variants of the above fuel models,
  with a different fuel model number to distinguish them as
  belonging to the Wildland Urban Interface (WUI).

  Each of these WUI fuel models has the same physical characteristics
  as one of the fuel models in the above table,
  but having a different model number enables a different behavior
  for other aspects, such as spread-rate adjustment or spotting.

  This table provides the mapping from variant model number -> original model number.
  Notice how the difference of model numbers is usually 100 or 110."
  {;; Grass (GR)
   211 101
   212 102
   213 103
   214 104
   215 105
   216 106
   217 107
   218 108
   ;; Grass-Shrub (GS)
   221 121
   222 122
   223 123
   224 124
   ;; Shrub (SH)
   241 141
   242 142
   243 143
   244 144
   245 145
   246 146
   247 147
   248 148
   249 149
   ;; Timber-Understory (TU)
   261 161
   262 162
   263 163
   265 165
   ;; Timber Litter (TL)
   281 181
   282 182
   283 183
   284 184
   285 185
   286 186
   287 187
   288 188
   289 189
   ;; Slash-Blowdown (SB)
   301 201
   302 202
   303 203})

(defrecord FuelModel
    [name
     ^long number
     ^double delta
     M_x
     w_o
     sigma
     h
     rho_p
     S_T
     S_e
     M_f
     f_ij
```

```
     f_i
     g_ij])

(defn is-dynamic-fuel-model-number?
  [^long fuel-model-number]
  (> fuel-model-number 100))

(defn compute-fuel-model
  [^long fuel-model-number]
  (let [[name delta M_x-dead h
         [w_o-dead-1hr w_o-dead-10hr w_o-dead-100hr
          w_o-live-herbaceous w_o-live-woody]
         [sigma-dead-1hr sigma-dead-10hr sigma-dead-100hr
          sigma-live-herbaceous sigma-live-woody]]
        (fuel-models fuel-model-number)
        M_x-dead   (* ^long M_x-dead 0.01)
        h          (* ^long h 1000.0)
        fuel-model {:name   name
                    :number fuel-model-number
                    :delta  delta
                    :M_x    [M_x-dead M_x-dead M_x-dead 0.0 0.0 0.0]
                    :w_o    [w_o-dead-1hr w_o-dead-10hr w_o-dead-100hr 0.0 w_o-live-herbaceous w_o-live-woody]
                    :sigma  [sigma-dead-1hr sigma-dead-10hr sigma-dead-100hr 0.0 sigma-live-herbaceous sigma-live-woody]
                    :h      [h h h h h h]
                    :rho_p  [32.0 32.0 32.0 32.0 32.0 32.0]
                    :S_T    [0.0555 0.0555 0.0555 0.0555 0.0555 0.0555]
                    :S_e    [0.01 0.01 0.01 0.01 0.01 0.01]}]
    (if (and (is-dynamic-fuel-model-number? fuel-model-number)
             (pos? ^double w_o-live-herbaceous))
      ;; Set dead-herbaceous values
      (-> fuel-model
          (assoc-in [:M_x   3] M_x-dead)
          (assoc-in [:sigma 3] sigma-live-herbaceous))
      ;; No dead-herbaceous values
      fuel-model)))

(defn- add-synonym-models
  [variant->original-model-number number->model-data]
  (reduce-kv (fn [ret v-num o-num]
               (assoc ret v-num (or (get number->model-data o-num)
                                    (throw (ex-info (format "Cannot find synonym fuel model number: %s" (pr-str o-num))
                                                    {::variant-fuel-model-number  v-num
                                                     ::original-fuel-model-number o-num})))))
             number->model-data
             variant->original-model-number))

(def fuel-models-precomputed (->> fuel-models
                                  (keys)
                                  (into {} (map #(vector % (map->FuelModel (compute-fuel-model %)))))
                                  (add-synonym-models WUI-model-number->original)))
(defn is-burnable-fuel-model-number?
  [^double fm-number]
  (and (pos? fm-number)
       (or (< fm-number 91.0)
           (> fm-number 99.0))))

(defn is-dynamic-grass-fuel-model-number?
  [^long fm-number]
  (or (and (< 100 fm-number) (< fm-number 110))
      (and (< 210 fm-number) (< fm-number 220))))
```

Once fuel moisture is added to the base fuel model definitions, they will each contain values for the following six fuel size classes:

1. Dead 1 hour (< 1/4" diameter)
2. Dead 10 hour (1/4"–1" diameter)

3. Dead 100 hour (1"–3" diameter)
4. Dead herbaceous (dynamic fuel models only)
5. Live herbaceous
6. Live woody

In order to more easily encode mathematical operations over these size classes, we define a collection of utility functions that will later be used in both the fuel moisture and fire spread algorithms.

```
(defn map-category [f]
  [(f 0) (f 1)])

(defn map-size-class [f]
  [(f 0) (f 1) (f 2) (f 3) (f 4) (f 5)])

(defn category-sum ^double [f]
  (+ ^double (f 0) ^double (f 1)))

(defn size-class-sum [f]
  [(+ (+ ^double (f 0) ^double (f 1))
      (+ ^double (f 2) ^double (f 3)))
   (+ ^double (f 4) ^double (f 5))])
```

Using these new size class processing functions, we can translate the encoded fuel model definitions into human-readable representations of the fuel model properties.

```
(defn build-fuel-model
  [fuel-model-number]
  (let [[name delta ^double M_x-dead ^double h
         [w_o-dead-1hr w_o-dead-10hr w_o-dead-100hr
          w_o-live-herbaceous w_o-live-woody]
         [sigma-dead-1hr sigma-dead-10hr sigma-dead-100hr
          sigma-live-herbaceous sigma-live-woody]]
        (fuel-models fuel-model-number)
        M_x-dead (* M_x-dead 0.01)
        h        (* h 1000.0)]
    {:name   name
     :number fuel-model-number
     :delta  delta
     :M_x    {:dead {:1hr        M_x-dead
                     :10hr       M_x-dead
                     :100hr      M_x-dead
                     :herbaceous 0.0}
              :live {:herbaceous 0.0
                     :woody      0.0}}
     :w_o    {:dead {:1hr        w_o-dead-1hr
                     :10hr       w_o-dead-10hr
                     :100hr      w_o-dead-100hr
                     :herbaceous 0.0}
              :live {:herbaceous w_o-live-herbaceous
                     :woody      w_o-live-woody}}
     :sigma  {:dead {:1hr        sigma-dead-1hr
                     :10hr       sigma-dead-10hr
                     :100hr      sigma-dead-100hr
                     :herbaceous 0.0}
              :live {:herbaceous sigma-live-herbaceous
                     :woody      sigma-live-woody}}
     :h      {:dead {:1hr        h
                     :10hr       h
                     :100hr      h
                     :herbaceous h}
              :live {:herbaceous h
                     :woody      h}}
     :rho_p  {:dead {:1hr        32.0
                     :10hr       32.0
```

```
                    :100hr      32.0
                    :herbaceous 32.0}
             :live {:herbaceous 32.0
                    :woody      32.0}}
    :S_T    {:dead {:1hr        0.0555
                    :10hr       0.0555
                    :100hr      0.0555
                    :herbaceous 0.0555}
             :live {:herbaceous 0.0555
                    :woody      0.0555}}
    :S_e    {:dead {:1hr        0.01
                    :10hr       0.01
                    :100hr      0.01
                    :herbaceous 0.01}
             :live {:herbaceous 0.01
                    :woody      0.01}}}))
```

Although most fuel model properties are static with respect to environmental conditions, the fuel moisture content can have two significant impacts on a fuel model's burning potential:

1. Dynamic fuel loading
2. Live moisture of extinction

These two topics are discussed in the remainder of this section.

5.1.1 Dynamic Fuel Loading

All of the Scott & Burgan 40 fuel models with a live herbaceous component are considered dynamic. In these models, a fraction of the live herbaceous load is transferred to a new dead herbaceous category as a function of live herbaceous moisture content (see equation below).(Burgan, 1979) The dead herbaceous category uses the dead 1 hour moisture content, dead moisture of extinction, and live herbaceous surface-area-to-volume-ratio. In the following formula, M_f^{lh} is the live herbaceous moisture content.

$$\text{FractionGreen} = \begin{cases} 0 & M_f^{lh} \leq 0.3 \\ 1 & M_f^{lh} \geq 1.2 \\ \frac{M_f^{lh}}{0.9} - \frac{1}{3} & \text{else} \end{cases}$$

$$\text{FractionCured} = 1 - \text{FractionGreen}$$

```
(defn add-dynamic-fuel-loading
  [fuel-model M_f]
  (let [^long number                (:number fuel-model)
        w_o                         (:w_o fuel-model)
        ^double live-herbaceous-load (w_o 4)] ; 4 = live-herbaceous
    (if (and (is-dynamic-fuel-model-number? number) (pos? live-herbaceous-load))
      ;; dynamic fuel model
      (let [name           (:name fuel-model)
            delta          (:delta fuel-model)
            M_x            (:M_x fuel-model)
            sigma          (:sigma fuel-model)
            h              (:h fuel-model)
            rho_p          (:rho_p fuel-model)
            S_T            (:S_T fuel-model)
            S_e            (:S_e fuel-model)
            fraction-green (max 0.0 (min 1.0 (- (/ ^double (M_f 4) 0.9) 0.3333333333333333))) ; 4 = live-herbaceous
            fraction-cured (- 1.0 fraction-green)
            dynamic-M_f    (assoc M_f 3 (M_f 0)) ; 0 = dead-1hr, 3 = dead-herbaceous
            dynamic-w_o    [(w_o 0)
```

```
                        (w_o 1)
                        (w_o 2)
                        (* live-herbaceous-load fraction-cured)
                        (* live-herbaceous-load fraction-green)
                        (w_o 5)]]
      (->FuelModel name number delta M_x dynamic-w_o sigma h rho_p S_T S_e dynamic-M_f nil nil nil))
    ;; static fuel model
    (assoc fuel-model :M_f M_f))))
```

Once the dynamic fuel loading is applied, we can compute the size class weighting factors expressed in equations 53-57 in Rothermel 1972(Rothermel, 1972). For brevity, these formulas are elided from this text.

```
(defn add-weighting-factors
  [fuel-model]
  (let [name                 (:name fuel-model)
        number               (:number fuel-model)
        delta                (:delta fuel-model)
        M_x                  (:M_x fuel-model)
        w_o                  (:w_o fuel-model)
        sigma                (:sigma fuel-model)
        h                    (:h fuel-model)
        rho_p                (:rho_p fuel-model)
        S_T                  (:S_T fuel-model)
        S_e                  (:S_e fuel-model)
        M_f                  (:M_f fuel-model)
        A_ij                 (map-size-class (fn ^double [i]
                                               (/ (* ^double (sigma i) ^double (w_o i))
                                                  ^double (rho_p i))))

        A_i                  (size-class-sum (fn ^double [i] ^double (A_ij i)))

        A_T                  (category-sum   (fn ^double [i] ^double (A_i i)))

        f_ij                 (map-size-class (fn ^double [^long i]
                                               (let [^double A (A_i (quot i 4))]
                                                 (if (pos? A)
                                                   (/ ^double (A_ij i) A)
                                                   0.0))))

        f_i                  (map-category   (fn ^double [i]
                                               (if (pos? A_T)
                                                 (/ ^double (A_i i) A_T)
                                                 0.0)))

        firemod-size-classes (map-size-class (fn ^long [i]
                                               (let [^double s (sigma i)]
                                                 (if (>= s 1200.0)
                                                   1
                                                   (if (>= s 192.0)
                                                     2
                                                     (if (>= s 96.0)
                                                       3
                                                       (if (>= s 48.0)
                                                         4
                                                         (if (>= s 16.0)
                                                           5
                                                           6)))))))))

        g_ij                 (map-size-class (fn ^double [^long i]
                                               (let [c (firemod-size-classes i)]
                                                 (if (< i 4)
                                                   (+ (+ (if (= c (firemod-size-classes 0)) ^double (f_ij 0) 0.0)
                                                         (if (= c (firemod-size-classes 1)) ^double (f_ij 1) 0.0))
                                                      (+ (if (= c (firemod-size-classes 2)) ^double (f_ij 2) 0.0)
                                                         (if (= c (firemod-size-classes 3)) ^double (f_ij 3) 0.0)))
                                                   (+ (if (= c (firemod-size-classes 4)) ^double (f_ij 4) 0.0)
```

```
                                                    (if (= c (firemod-size-classes 5)) ^double (f_ij 5) 0.0))))))]
  (->FuelModel name number delta M_x w_o sigma h rho_p S_T S_e M_f f_ij f_i g_ij)))
```

5.1.2 Live Moisture of Extinction

The live moisture of extinction for each fuel model is determined from the dead fuel moisture content, the dead moisture of extinction, and the ratio of dead fuel loading to live fuel loading using Equation 88 from Rothermel 1972, adjusted according to Albini 1976 Appendix III to match the behavior of Albini's original FIREMODS library.(Rothermel, 1972, Albini, 1976) Whenever the fuel moisture content becomes greater than or equal to the moisture of extinction, a fire will no longer spread through that fuel. Here are the formulas referenced above:

$$M_x^l = \max(M_x^d, 2.9\,W'\,(1 - \frac{M_f^d}{M_x^d}) - 0.226)$$

$$W' = \frac{\sum_{c \in D} w_o^c\, e^{-138/\sigma^c}}{\sum_{c \in L} w_o^c\, e^{-500/\sigma^c}}$$

$$M_f^d = \frac{\sum_{c \in D} w_o^c\, M_f^c\, e^{-138/\sigma^c}}{\sum_{c \in D} w_o^c\, e^{-138/\sigma^c}}$$

where M_x^l is the live moisture of extinction, M_x^d is the dead moisture of extinction, D is the set of dead fuel size classes (1hr, 10hr, 100hr, herbaceous), L is the set of live fuel size classes (herbaceous, woody), w_o^c is the dry weight loading of size class c, σ^c is the surface area to volume ratio of size class c, and M_f^c is the moisture content of size class c.

```
(defn add-live-moisture-of-extinction
  "Equation 88 from Rothermel 1972 adjusted by Albini 1976 Appendix III."
  [fuel-model]
  (let [w_o                          (:w_o fuel-model)
        sigma                        (:sigma fuel-model)
        M_f                          (:M_f fuel-model)
        M_x                          (:M_x fuel-model)
        loading-factors              (map-size-class (fn ^double [^long i]
                                                       (let [^double sigma_ij (sigma i)
                                                             A (if (< i 4) -138.0 -500.0)]
                                                         (if (pos? sigma_ij)
                                                           (* ^double (w_o i)
                                                              (Math/exp (/ A sigma_ij)))
                                                           0.0))))
        [^double dead-loading-factor
         ^double live-loading-factor] (size-class-sum (fn ^double [i] ^double (loading-factors i)))
        [^double dead-moisture-factor] (size-class-sum (fn ^double [i]
                                                         (* ^double (M_f i)
                                                            ^double (loading-factors i))))
        ^double dead-to-live-ratio   (when (pos? live-loading-factor)
                                       (/ dead-loading-factor live-loading-factor))
        dead-fuel-moisture           (if (pos? dead-loading-factor)
                                       (/ dead-moisture-factor dead-loading-factor)
                                       0.0)
        ^double M_x-dead             (M_x 0)
        M_x-live                     (if (pos? live-loading-factor)
                                       (max M_x-dead
                                            (- (-> 2.9
                                                   (* dead-to-live-ratio)
                                                   (* (- 1.0 (/ dead-fuel-moisture M_x-dead))))
                                               0.226))
                                       M_x-dead)]
```

```
    (assoc fuel-model :M_x [(M_x 0)
                            (M_x 1)
                            (M_x 2)
                            (M_x 3)
                            M_x-live
                            M_x-live])))

(defn moisturize
  [fuel-model fuel-moisture]
  (-> fuel-model
      (add-dynamic-fuel-loading fuel-moisture)
      (add-weighting-factors)
      (add-live-moisture-of-extinction)))
```

This concludes our coverage of fuel models and and fuel moisture.

5.2 Surface Fire Formulas

To simulate fire behavior in as similar a way as possible to the US government-sponsored fire models (e.g., FARSITE, FlamMap, FPA, BehavePlus), we adopt the surface fire spread and reaction intensity formulas from Rothermel's 1972 publication "A Mathematical Model for Predicting Fire Spread in Wildland Fuels".(Rothermel, 1972)

Very briefly, the surface rate of spread of a fire's leading edge R is described by the following formula:

$$R = \frac{I_R \, \xi \, (1 + \phi_W + \phi_S)}{\rho_b \, \epsilon \, Q_{ig}}$$

where these terms have the meanings shown in Table 3.

Table 3: Inputs to Rothermel's surface fire rate of spread equation

Term	Meaning
R	surface fire spread rate
I_R	reaction intensity
ξ	propagating flux ratio
ϕ_W	wind coefficient
ϕ_S	slope factor
ρ_b	oven-dry fuel bed bulk density
ϵ	effective heating number
Q_{ig}	heat of preignition

FIXME add units column.

For a full description of each of the subcomponents of Rothermel's surface fire spread rate equation, see the Rothermel 1972 reference above. In addition to applying the base Rothermel equations, GridFire can reduce the spread rates for all of the Scott & Burgan 40 fuel models of the grass subgroup (101-109) by 50% by enabling the `:grass-suppression?` configuration. This addition was originally suggested by Chris Lautenberger of REAX Engineering.

For efficiency, the surface fire spread equation given above is computed first without introducing the effects of wind and slope ($\phi_W = \phi_S = 0$).

```
(ns gridfire.surface-fire
  (:require [gridfire.conversion  :refer [deg->rad rad->deg]]
            [gridfire.fuel-models :refer [is-dynamic-grass-fuel-model-number? map-category map-size-class category-sum size-class-s

(set! *unchecked-math* :warn-on-boxed)
```

```
(defn calc-mineral-damping-coefficients [f_ij S_e]
  (let [S_e_i (size-class-sum (fn ^double [i] (* ^double (f_ij i) ^double (S_e i))))]
    (map-category (fn ^double [i]
                    (let [^double S_e_i (S_e_i i)]
                      (if (pos? S_e_i)
                        (/ 0.174 (Math/pow S_e_i 0.19))
                        1.0))))))

(defn calc-moisture-damping-coefficients [f_ij M_f M_x]
  (let [M_f_i (size-class-sum (fn ^double [i] (* ^double (f_ij i) ^double (M_f i))))
        M_x_i (size-class-sum (fn ^double [i] (* ^double (f_ij i) ^double (M_x i))))]
    (map-category (fn ^double [i]
                    (let [^double M_f (M_f_i i)
                          ^double M_x (M_x_i i)
                          r_M (if (pos? M_x)
                                (min 1.0 (/ M_f M_x))
                                1.0)]
                      (-> 1.0
                          (+ (* -2.59 r_M))
                          (+ (* 5.11 (Math/pow r_M 2.0)))
                          (+ (* -3.52 (Math/pow r_M 3.0)))))))))

(defn calc-low-heat-content [f_ij h]
  (size-class-sum (fn ^double [i] (* ^double (f_ij i) ^double (h i)))))

(defn calc-net-fuel-loading [g_ij w_o S_T]
  (size-class-sum (fn ^double [i]
                    (let [^double g_ij (g_ij i)
                          ^double w_o  (w_o i)
                          ^double S_T  (S_T i)]
                      (-> g_ij
                          (* w_o)
                          (* (- 1.0 S_T)))))))

(defn calc-packing-ratio ^double [w_o rho_p ^double delta]
  (let [beta_i (size-class-sum (fn ^double [i] (/ ^double (w_o i) ^double (rho_p i))))]
    (if (pos? delta)
      (/ (category-sum (fn ^double [i] ^double (beta_i i))) delta)
      0.0)))

(defn calc-surface-area-to-volume-ratio ^double [f_i f_ij sigma]
  (let [sigma'_i (size-class-sum (fn ^double [i] (* ^double (f_ij i) ^double (sigma i))))]
    (category-sum (fn ^double [i] (* ^double (f_i i) ^double (sigma'_i i))))))

(defn calc-optimum-packing-ratio ^double [^double sigma']
  (if (pos? sigma')
    (/ 3.348 (Math/pow sigma' 0.8189))
    1.0))

(defn calc-optimum-reaction-velocity ^double [^double beta ^double sigma' ^double beta_op]
  (let [;; Albini 1976 replaces (/ 1 (- (* 4.774 (Math/pow sigma' 0.1)) 7.27))
        A          (if (pos? sigma')
                     (/ 133.0 (Math/pow sigma' 0.7913))
                     0.0)
        B          (Math/pow sigma' 1.5)
        C          (/ beta beta_op)
        ;; Maximum reaction velocity (1/min)
        Gamma'_max (/ B (+ 495.0 (* 0.0594 B)))]
    ;; Optimum reaction velocity (1/min)
    (-> Gamma'_max
        (* (Math/pow C A))
        (* (Math/exp (* A (- 1.0 C)))))))

(defn calc-heat-per-unit-area ^double [eta_S_i eta_M_i h_i W_n_i]
  (category-sum
   (fn ^double [i]
     (let [^double W_n   (W_n_i i)
           ^double h     (h_i i)
```

```
           ^double eta_M (eta_M_i i)
           ^double eta_S (eta_S_i i)]
       (-> W_n (* h) (* eta_M) (* eta_S))))))

(defn calc-reaction-intensity ^double [^double Gamma' ^double Btus]
  (* Gamma' Btus))

(defn calc-propagating-flux-ratio ^double [^double beta ^double sigma']
  (/ (Math/exp (* (+ 0.792 (* 0.681 (Math/pow sigma' 0.5)))
                  (+ beta 0.1)))
     (+ 192.0 (* 0.2595 sigma'))))

(defn calc-heat-of-preignition [M_f]
  (map-size-class (fn ^double [i] (+ 250.0 (* 1116.0 ^double (M_f i))))))

(defn calc-heat-distribution [sigma Q_ig f_ij]
  (size-class-sum (fn ^double [i]
                    (let [^double sigma (sigma i)
                          ^double Q_ig  (Q_ig i)]
                      (if (pos? sigma)
                        (-> ^double (f_ij i)
                            (* (Math/exp (/ -138.0 sigma)))
                            (* Q_ig))
                        0.0)))))

(defn calc-ovendry-bulk-density ^double [w_o ^double delta]
  (let [rho_b_i (size-class-sum (fn ^double [i] ^double (w_o i)))]
    (if (pos? delta)
      (/ (category-sum (fn ^double [i] ^double (rho_b_i i))) delta)
      0.0)))

(defn calc-heat-total ^double [f_i epsilon_i]
  (category-sum (fn ^double [i] (* ^double (f_i i) ^double (epsilon_i i)))))

(defn calc-surface-fire-spread-rate ^double [^double I_R ^double xi ^double rho_b ^double epsilon]
  (let [rho_b-epsilon-Q_ig (* rho_b epsilon)]
    (if (pos? rho_b-epsilon-Q_ig)
      (/ (* I_R xi) rho_b-epsilon-Q_ig)
      0.0)))

;; Addition proposed by Chris Lautenberger (REAX 2015)
(defn calc-suppressed-spread-rate ^double [^double R ^long number grass-suppression?]
  (let [spread-rate-multiplier (if (and grass-suppression?
                                        (is-dynamic-grass-fuel-model-number? number))
                                 0.5
                                 1.0)]
    (* R spread-rate-multiplier)))

(defn calc-residence-time ^double [^double sigma']
  (/ 384.0 sigma'))

(defn get-phi_S-fn
  [^double beta]
  (let [G (* 5.275 (Math/pow beta -0.3))]
    (if (pos? beta)
      (fn ^double [^double slope]
        (if (pos? slope)
          (-> slope
              (Math/pow 2.0)
              (* G))
          0.0))
      (fn ^double [^double _]
        0.0))))

(defn get-phi_W-fn
  [^double beta ^double B ^double C ^double F]
  (let [C-over-F (/ C F)]
    (if (pos? beta)
```

```
      (fn ^double [^double midflame-wind-speed]
        (if (pos? midflame-wind-speed)
          (-> midflame-wind-speed
              (Math/pow B)
              (* C-over-F))
          0.0))
      (fn ^double [^double _]
        0.0))))

(defn get-wind-speed-fn
  [^double B ^double C ^double F]
  (let [F-over-C  (/ F C)
        B-inverse (/ 1.0 B)]
    (fn ^double [^double phi_W]
      (-> phi_W
          (* F-over-C)
          (Math/pow B-inverse)))))

(defrecord SurfaceFireMin
    [^double unadj-spread-rate
     ^double reaction-intensity
     ^double residence-time
     ^double fuel-bed-depth
     ^double heat-of-combustion
     get-phi_S
     get-phi_W
     get-wind-speed])

;; TODO: Pass fuel-model in as a typed record instead of a map
(defn rothermel-surface-fire-spread-no-wind-no-slope
  "Returns the rate of surface fire spread in ft/min and the reaction
   intensity (i.e., amount of heat output) of a fire in Btu/ft^2*min
   given a map containing these keys:
   - number [fuel model number]
   - delta [fuel depth (ft)]
   - w_o [ovendry fuel loading (lb/ft^2)]
   - sigma [fuel particle surface-area-to-volume ratio (ft^2/ft^3)]
   - h [fuel particle low heat content (Btu/lb)]
   - rho_p [ovendry particle density (lb/ft^3)]
   - S_T [fuel particle total mineral content (lb minerals/lb ovendry wood)]
   - S_e [fuel particle effective mineral content (lb silica-free minerals/lb ovendry wood)]
   - M_x [moisture content of extinction (lb moisture/lb ovendry wood)]
   - M_f [fuel particle moisture content (lb moisture/lb ovendry wood)]
   - f_ij [percent of load per size class (%)]
   - f_i [percent of load per category (%)]
   - g_ij [percent of load per size class from Albini_1976_FIREMOD, page 20]"
  [fuel-model grass-suppression?]
  (let [number         (:number fuel-model)
        delta          (:delta fuel-model)
        w_o            (:w_o fuel-model)
        sigma          (:sigma fuel-model)
        h              (:h fuel-model)
        rho_p          (:rho_p fuel-model)
        S_T            (:S_T fuel-model)
        S_e            (:S_e fuel-model)
        M_x            (:M_x fuel-model)
        M_f            (:M_f fuel-model)
        f_ij           (:f_ij fuel-model)
        f_i            (:f_i fuel-model)
        g_ij           (:g_ij fuel-model)
        eta_S_i        (calc-mineral-damping-coefficients f_ij S_e)
        eta_M_i        (calc-moisture-damping-coefficients f_ij M_f M_x)
        h_i            (calc-low-heat-content f_ij h)
        W_n_i          (calc-net-fuel-loading g_ij w_o S_T)                ; (lb/ft^2)
        beta           (calc-packing-ratio w_o rho_p delta)
        sigma'         (calc-surface-area-to-volume-ratio f_i f_ij sigma)
        beta_op        (calc-optimum-packing-ratio sigma')
        Gamma'         (calc-optimum-reaction-velocity beta sigma' beta_op) ; (1/min)
```

```
      Btus           (calc-heat-per-unit-area eta_S_i eta_M_i h_i W_n_i)  ; (Btu/ft^2)
      I_R            (calc-reaction-intensity Gamma' Btus)                ; (Btu/ft^2*min)
      xi             (calc-propagating-flux-ratio beta sigma')
      Q_ig           (calc-heat-of-preignition M_f)                       ; (Btu/lb)
      epsilon_i      (calc-heat-distribution sigma Q_ig f_ij)             ; (Btu/lb)
      rho_b          (calc-ovendry-bulk-density w_o delta)                ; (lb/ft^3)
      epsilon        (calc-heat-total f_i epsilon_i)                      ; (Btu/lb)
      R              (calc-surface-fire-spread-rate I_R xi rho_b epsilon) ; (ft/min)
      R'             (calc-suppressed-spread-rate R number grass-suppression?)
      t_res          (calc-residence-time sigma')
      B              (* 0.02526 (Math/pow sigma' 0.54))
      C              (* 7.47 (Math/exp (* -0.133 (Math/pow sigma' 0.55))))
      E              (* 0.715 (Math/exp (* -3.59 (/ sigma' 10000.0))))
      F              (Math/pow (/ beta beta_op) E)
      get-phi_S      (get-phi_S-fn beta)
      get-phi_W      (get-phi_W-fn beta B C F)
      get-wind-speed (get-wind-speed-fn B C F)]
  (->SurfaceFireMin R'
                    I_R
                    t_res
                    delta
                    (h 0)
                    get-phi_S
                    get-phi_W
                    get-wind-speed)))
```

Later, this no-wind-no-slope value is used to compute the maximum spread rate and direction for the leading edge of the surface fire under analysis. Since Rothermel's original equations assume that the wind direction and slope are aligned, the effects of cross-slope winds must be taken into effect. Like Morais' HFire system, GridFire implements the vector addition procedure defined in Rothermel 1983 that combines the wind-only and slope-only spread rates independently to calculate the effective fire spread direction and magnitude.(Peterson et al., 2011, 2009, Morais, 2001, Rothermel, 1983)

A minor wrinkle is introduced when putting these calculations into practice because Rothermel's formulas all expect a measure of midflame wind speed. However, wind speed data is often collected at a height 20 feet above either unsheltered ground or a tree canopy layer if present. To convert this 20-ft wind speed to the required midflame wind speed value, GridFire uses the **wind adjustment factor** formula from Albini & Baughman 1979, parameterized as in BehavePlus, FARSITE, FlamMap, FSPro, and FPA according to Andrews 2012(Albini and Baughman, 1979, Andrews et al., 2012). This formula is shown below:

$$WAF = \begin{cases} \frac{1.83}{\ln(\frac{20.0+0.36FBD}{0.13FBD})} & CC = 0 \\ \frac{0.555}{\sqrt{(CH(CC/300.0))}\ln(\frac{20+0.36CH}{0.13CH})} & CC > 0 \end{cases}$$

where WAF is the unitless wind adjustment factor, FBD is the fuel bed depth in feet, CH is the canopy height in ft, and CC is the canopy cover percentage (0-100).

```
(defn wind-adjustment-factor
  "ft ft 0-100"
  ^double
  [^double fuel-bed-depth ^double canopy-height ^double canopy-cover]
  (cond
    ;; sheltered: equation 2 based on CC and CH, CR=1 (Andrews 2012)
    (and (pos? canopy-cover)
         (pos? canopy-height))
    (/ 0.555 (* (Math/sqrt (* (/ canopy-cover 300.0) canopy-height))
                (Math/log (/ (+ 20.0 (* 0.36 canopy-height)) (* 0.13 canopy-height)))))

    ;; unsheltered: equation 6 H_F = H (Andrews 2012)
```

```
    (pos? fuel-bed-depth)
    (/ 1.83 (Math/log (/ (+ 20.0 (* 0.36 fuel-bed-depth)) (* 0.13 fuel-bed-depth))))

    ;; non-burnable fuel model
    :otherwise
    0.0))

(defn wind-adjustment-factor-elmfire
  "ft m 0-1"
  ^double
  [^double fuel-bed-depth ^double canopy-height ^double canopy-cover]
  (cond
    ;; sheltered WAF
    (and (pos? canopy-cover)
         (pos? canopy-height))
    (* (/ 1.0 (Math/log (/ (+ 20.0 (* 0.36 (/ canopy-height 0.3048)))
                           (* 0.13 (/ canopy-height 0.3048)))))
       (/ 0.555 (Math/sqrt (* (/ canopy-cover 3.0) (/ canopy-height 0.3048)))))

    ;; unsheltered WAF
    (pos? fuel-bed-depth)
    (* (/ (+ 1.0 (/ 0.36 1.0))
          (Math/log (/ (+ 20.0 (* 0.36 fuel-bed-depth))
                       (* 0.13 fuel-bed-depth))))
       (- (Math/log (/ (+ 1.0 0.36) 0.13)) 1.0))

    ;; non-burnable fuel model
    :otherwise
    0.0))
```

The midflame wind speed that would be required to produce the combined spread rate in a no-slope scenario is termed the effective windspeed U_{eff}. Following the recommendations given in Appendix III of Albini 1976, these midflame wind speeds are all limited to $0.9I_R$.(Albini, 1976)

Next, the effective wind speed is used to compute the length to width ratio $\frac{L}{W}$ of an ellipse that approximates the fire front using equation 9 from Rothermel 1991.(Rothermel, 1991) This length to width ratio is then converted into an eccentricity measure of the ellipse using equation 8 from Albini and Chase 1980.(Albini and Chase, 1980) Finally, this eccentricity E is used to project the maximum spread rate to any point along the fire front. Here are the formulas used:

$$\frac{L}{W} = 1 + 0.002840909\, U_{\text{eff}}\, \text{EAF}$$

$$E = \frac{\sqrt{(\frac{L}{W})^2 - 1}}{\frac{L}{W}}$$

$$R_\theta = R_{\max}\left(\frac{1-E}{1-E\cos\theta}\right)$$

where θ is the angular offset from the direction of maximum fire spread, $\text{R}_{\max}$ is the maximum spread rate, R_θ is the spread rate in direction θ, and EAF is the ellipse adjustment factor, a term introduced by Marco Morais and Seth Peterson in their HFire work that can be increased or decreased to make the fire shape more elliptical or circular respectively.(Peterson et al., 2009)

Note: The coefficient 0.002840909 in the $\frac{L}{W}$ formula is in units of min/ft. The original equation from Rothermel 1991 used 0.25 in units of hr/mi, so this was converted to match GridFire's use of ft/min for U_{eff}.

Another point of note in the code below is that it enables applying an adjustment factor to the no-wind-no-slope spread rate, typically configured by fuel number (corresponding to the **:fuel-number-¿spread-rate-adjustment** configuration key).

```
(defn almost-zero? [^double x]
  (< (Math/abs x) 0.000001))

(defrecord SurfaceFireMax
    [^double max-spread-rate
     ^double max-spread-direction
     ^double effective-wind-speed
     ^double eccentricity])

(defn scale-spread-to-max-wind-speed
  [spread-properties ^double spread-rate ^double max-wind-speed ^double phi-max]
  (let [effective-wind-speed (:effective-wind-speed spread-properties)
        max-spread-direction (:max-spread-direction spread-properties)]
   (if (> ^double effective-wind-speed max-wind-speed)
     (->SurfaceFireMax (* spread-rate (+ 1.0 phi-max)) max-spread-direction max-wind-speed 0.0)
     spread-properties)))

(defn add-eccentricity
  [spread-properties ^double ellipse-adjustment-factor]
  (let [effective-wind-speed (:effective-wind-speed spread-properties)
        length-width-ratio   (+ 1.0 (-> 0.002840909
                                        (* ^double effective-wind-speed)
                                        (* ellipse-adjustment-factor)))
        eccentricity         (/ (Math/sqrt (- (Math/pow length-width-ratio 2.0) 1.0))
                                length-width-ratio)]
    (assoc spread-properties :eccentricity eccentricity)))

(defn smallest-angle-between
  "Computes the absolute difference between two angles as an angle between 0° and 180°.

  The return angle has the same cosine as (- theta1 theta2), but may have an opposite sine."
  ^double [^double theta1 ^double theta2]
  (let [angle (Math/abs (- theta1 theta2))]
    (if (> angle 180.0)
      (- 360.0 angle)
      angle)))

(defn determine-spread-drivers
  [^double midflame-wind-speed ^double wind-to-direction ^double slope ^double slope-direction]
  (if (almost-zero? slope)
    (if (almost-zero? midflame-wind-speed)
      :no-wind-no-slope
      :wind-only)
    (if (almost-zero? midflame-wind-speed)
      :slope-only
      (if (< (smallest-angle-between wind-to-direction slope-direction) 15.0)
        :wind-blows-upslope
        :wind-blows-across-slope))))

(defn spread-info-max-no-wind-no-slope
  [^double spread-rate]
  (->SurfaceFireMax spread-rate 0.0 0.0 0.0))

(defn spread-info-max-wind-only
  [^double spread-rate ^double phi_W ^double midflame-wind-speed ^double wind-to-direction]
  (->SurfaceFireMax (* spread-rate (+ 1.0 phi_W)) wind-to-direction midflame-wind-speed 0.0))

(defn spread-info-max-slope-only
  [^double spread-rate ^double phi_S ^double slope-direction get-wind-speed]
  (->SurfaceFireMax (* spread-rate (+ 1.0 phi_S)) slope-direction (get-wind-speed phi_S) 0.0))

(defn spread-info-max-wind-blows-upslope
  [^double spread-rate ^double phi-combined ^double slope-direction get-wind-speed]
  (->SurfaceFireMax (* spread-rate (+ 1.0 phi-combined)) slope-direction (get-wind-speed phi-combined) 0.0))

(defn spread-info-max-wind-blows-across-slope
  [spread-rate phi_W phi_S wind-to-direction slope-direction get-wind-speed]
```

```
  (let [spread-rate        (double spread-rate)
        wind-to-direction  (double wind-to-direction)
        slope-direction    (double slope-direction)
        wind-magnitude     (* spread-rate ^double phi_W)
        slope-magnitude    (* spread-rate ^double phi_S)
        difference-angle   (deg->rad
                            (mod (- wind-to-direction slope-direction) 360.0))
        x                  (+ slope-magnitude
                              (* wind-magnitude (Math/cos difference-angle)))
        y                  (* wind-magnitude (Math/sin difference-angle))
        combined-magnitude (Math/sqrt (+ (* x x) (* y y)))]
    (if (almost-zero? combined-magnitude)
      (->SurfaceFireMax spread-rate 0.0 0.0 0.0)
      (let [max-spread-rate      (+ spread-rate combined-magnitude)
            phi-combined         (- (/ max-spread-rate spread-rate) 1.0)
            offset               (rad->deg
                                  (Math/asin (/ (Math/abs y) combined-magnitude)))
            offset'              (if (>= x 0.0)
                                   (if (>= y 0.0)
                                     offset
                                     (- 360.0 offset))
                                   (if (>= y 0.0)
                                     (- 180.0 offset)
                                     (+ 180.0 offset)))
            max-spread-direction (mod (+ slope-direction offset') 360.0)
            effective-wind-speed (get-wind-speed phi-combined)]
        (->SurfaceFireMax max-spread-rate max-spread-direction effective-wind-speed 0.0)))))

(defn resolve-spread-rate ^double
  [^SurfaceFireMin surface-fire-min ^double adjustment-factor]
  (-> surface-fire-min
      (:unadj-spread-rate)
      (double)
      (* adjustment-factor)))

(defn rothermel-surface-fire-spread-max
  "Note: fire ellipse adjustment factor, < 1.0 = more circular, > 1.0 = more elliptical"
  [surface-fire-min midflame-wind-speed wind-from-direction slope aspect ellipse-adjustment-factor spread-rate-adjustment]
  (let [spread-rate               (resolve-spread-rate surface-fire-min (or spread-rate-adjustment 1.0))
        reaction-intensity        (double (:reaction-intensity surface-fire-min))
        get-phi_S                 (:get-phi_S surface-fire-min)
        get-phi_W                 (:get-phi_W surface-fire-min)
        get-wind-speed            (:get-wind-speed surface-fire-min)
        midflame-wind-speed       (double midflame-wind-speed)
        wind-from-direction       (double wind-from-direction)
        slope                     (double slope)
        aspect                    (double aspect)
        ellipse-adjustment-factor (double ellipse-adjustment-factor)
        slope-direction           (mod (+ aspect 180.0) 360.0)
        wind-to-direction         (mod (+ wind-from-direction 180.0) 360.0)
        max-wind-speed            (* 0.9 reaction-intensity)
        ^double phi_S             (get-phi_S slope)
        ^double phi_W             (get-phi_W midflame-wind-speed)
        ^double phi-max           (get-phi_W max-wind-speed)]
    (->
     (case (determine-spread-drivers midflame-wind-speed wind-to-direction slope slope-direction)
       :no-wind-no-slope        (spread-info-max-no-wind-no-slope spread-rate)
       :wind-only               (spread-info-max-wind-only spread-rate phi_W midflame-wind-speed wind-to-direction)
       :slope-only              (spread-info-max-slope-only spread-rate phi_S slope-direction get-wind-speed)
       :wind-blows-upslope      (spread-info-max-wind-blows-upslope spread-rate (+ phi_W phi_S)
                                                                    slope-direction get-wind-speed)
       :wind-blows-across-slope (spread-info-max-wind-blows-across-slope spread-rate phi_W phi_S wind-to-direction
                                                                         slope-direction get-wind-speed))
     (scale-spread-to-max-wind-speed spread-rate max-wind-speed phi-max)
     (add-eccentricity ellipse-adjustment-factor))))

(defn compute-spread-rate ^double
  [^double max-spread-rate ^double max-spread-direction ^double eccentricity ^double spread-direction]
```

```
  (let [theta (smallest-angle-between max-spread-direction spread-direction)]
    (if (or (almost-zero? eccentricity) (almost-zero? theta))
      max-spread-rate
      (* max-spread-rate (/ (- 1.0 eccentricity)
                            (- 1.0 (* eccentricity
                                      (Math/cos (deg->rad theta))))))))
```

Using these surface fire spread rate and reaction intensity values, we next calculate fire intensity values by applying Anderson's flame depth formula and Byram's fire line intensity and flame length equations as described below.(Anderson, 1969, Byram, 1959)

$$t = \frac{384}{\sigma}$$
$$D = Rt$$
$$I = \frac{I_R D}{60}$$
$$L = 0.45(I)^{0.46}$$

where σ is the weighted sum by size class of the fuel model's surface area to volume ratio in ft^2/ft^3, t is the residence time in minutes, R is the surface fire spread rate in ft/min, D is the flame depth in ft, I_R is the reaction intensity in $Btu/ft^2/min$, I is the fire line intensity in Btu/ft/s, and L is the flame length in ft.

```
(defn anderson-flame-depth
  "Returns the depth, or front-to-back distance, of the actively flaming zone
   of a free-spreading fire in ft given:
   - spread-rate (ft/min) orthogonal to the fire line.
   - residence-time (min)"
  ^double
  [^double spread-rate ^double residence-time]
  (* spread-rate residence-time))

(defn byram-fire-line-intensity
  "Returns the rate of heat release per unit of fire edge in Btu/ft*s given:
   - reaction-intensity (Btu/ft^2*min)
   - flame-depth (ft)"
  ^double
  [^double reaction-intensity ^double flame-depth]
  (/ (* reaction-intensity flame-depth) 60.0))

(defn byram-flame-length
  "Returns the average flame length in ft given:
   - fire-line-intensity (Btu/ft*s)"
  ^double
  [^double fire-line-intensity]
  (* 0.45 (Math/pow fire-line-intensity 0.46)))
```

This concludes our coverage of the surface fire behavior equations implemented in GridFire. In Section 5.4, these formulas will be translated from one-dimension to two-dimensional spread on a raster grid. Before we move on to that, however, the following section explains how crown fire behavior metrics are incorporated into our model.

5.3 Crown Fire Formulas

In order to incorporate the effects of crown fire behavior, GridFire includes the crown fire initiation routine from Van Wagner 1977.(Wagner, 1977) According to this approach, there are two threshold values (*critical*

intensity and *critical spread rate*) that must be calculated in order to determine whether a fire will become an active or passive crown fire or simply remain a surface fire. The formulas for these thresholds are as follows:

$$H = 460 + 2600M^f$$
$$I^* = (0.01\, Z_b\, H)^{1.5}$$
$$R^* = \frac{3.0}{B_m}$$

where H is the heat of ignition for the herbaceous material in the canopy in kJ/kg, M^f is the foliar moisture content in lb moisture/lb ovendry weight, Z_b is the canopy base height in meters, I^* is the critical intensity in kW/m, B_m is the crown bulk density in kg/m^3, and R^* is the critical spread rate in m/min.

If the canopy cover is greater than 40% and the surface fire line intensity is greater than the critical intensity ($I > I^*$), then crown fire initiation occurs.

```
(ns gridfire.crown-fire
  (:require [gridfire.conversion :as convert]))

(set! *unchecked-math* :warn-on-boxed)

(defn van-wagner-critical-fire-line-intensity
  "Ouputs the critical fire line intensity (kW/m) using:
   - canopy-base-height (m)
   - foliar-moisture (0-100 %)"
  ^double
  [^double canopy-base-height ^double foliar-moisture]
  (-> foliar-moisture
      (* 26.0)
      (+ 460.0) ;; heat-of-ignition = kJ/kg
      (* 0.01)  ;; empirical estimate for C in eq. 4
      (* canopy-base-height)
      (Math/pow 1.5))) ;; critical-intensity = kW/m

(defn van-wagner-crown-fire-initiation-metric?
  "- canopy-cover (0-100 %)
   - canopy-base-height (m)
   - foliar-moisture (0-100 %)
   - fire-line-intensity (kW/m)"
  [^double canopy-cover ^double canopy-base-height ^double foliar-moisture ^double fire-line-intensity]
  (and (> canopy-cover 40.0)
       (> fire-line-intensity 0.0)
       (> canopy-base-height 0.0)
       (>= fire-line-intensity (van-wagner-critical-fire-line-intensity canopy-base-height foliar-moisture))))

(defn van-wagner-crown-fire-initiation?
  "- canopy-cover (0-100 %)
   - canopy-base-height (ft)
   - foliar-moisture (0-1)
   - fire-line-intensity (Btu/ft*s)"
  [^double canopy-cover ^double canopy-base-height ^double foliar-moisture ^double fire-line-intensity]
  (van-wagner-crown-fire-initiation-metric? canopy-cover
                                            (convert/ft->m canopy-base-height)
                                            (convert/dec->percent foliar-moisture)
                                            (convert/Btu-ft-s->kW-m fire-line-intensity)))
```

If crowning occurs, then the active and passive crown fire spread rates are calculated from the formulas given in Cruz 2005.(Cruz et al., 2005)

$$\mathrm{CROS}_A = 11.02\, U_{10m}^{0.90}\, B_m^{0.19}\, e^{-0.17\,\mathrm{EFFM}}$$
$$\mathrm{CROS}_P = \mathrm{CROS}_A\, e^{\frac{-\mathrm{CROS}_A}{R^*}}$$

where CROS_A is the active crown fire spread rate in m/min, U_{10m} is the 10 meter windspeed in km/hr, B_m is the crown bulk density in kg/m^3, EFFM is the estimated fine fuel moisture as a percent (0-100), and CROS_P is the passive crown fire spread rate in m/min.

If the active crown fire spread rate is greater than the critical spread rate ($\mathrm{CROS}_A > R^*$), then the crown fire will be active, otherwise passive.

```
(defn cruz-active-crown-fire-spread
  "Returns active spread-rate in m/min given:
   - wind-speed-10m (km/hr)
   - crown-bulk-density (kg/m^3)
   - estimated-fine-fuel-moisture (0-100 %)"
  ^double
  [^double wind-speed-10m ^double crown-bulk-density ^double estimated-fine-fuel-moisture]
  (* 11.02
     (Math/pow wind-speed-10m 0.90)
     (Math/pow crown-bulk-density 0.19)
     (Math/exp (* -0.17 estimated-fine-fuel-moisture))))

(defn cruz-passive-crown-fire-spread
  "Returns passive spread-rate in m/min given:
   - active-spread-rate (m/min)
   - critical-spread-rate (m/min)"
  ^double
  [^double active-spread-rate ^double critical-spread-rate]
  (* active-spread-rate
     (Math/exp (- (/ active-spread-rate critical-spread-rate)))))

(defn cruz-crown-fire-spread-metric
  "Returns spread-rate in m/min given:
   - wind-speed-10m (km/hr)
   - crown-bulk-density (kg/m^3)
   - estimated-fine-fuel-moisture (-> M_f :dead :1hr) (0-100 %)
   NOTE: A positive spread-rate indicates active crowning.
         A negative spread-rate indicates passive crowning."
  ^double
  [^double wind-speed-10m ^double crown-bulk-density ^double estimated-fine-fuel-moisture]
  (let [active-spread-rate   (cruz-active-crown-fire-spread wind-speed-10m
                                                            crown-bulk-density
                                                            estimated-fine-fuel-moisture)
        critical-spread-rate (/ 3.0 crown-bulk-density)] ;; m/min
    (if (> active-spread-rate critical-spread-rate)
      active-spread-rate
      (- (cruz-passive-crown-fire-spread active-spread-rate critical-spread-rate)))))  ; NOTE: Use minus as passive flag

(defn cruz-crown-fire-spread
  "Returns spread-rate in ft/min given:
   - wind-speed-20ft (mph)
   - crown-bulk-density (lb/ft^3)
   - estimated-fine-fuel-moisture (-> M_f :dead :1hr) (0-1)
   NOTE: A positive spread-rate indicates active crowning.
         A negative spread-rate indicates passive crowning."
  ^double
  [^double wind-speed-20ft ^double crown-bulk-density ^double estimated-fine-fuel-moisture]
  (convert/m->ft
   (cruz-crown-fire-spread-metric
    (-> wind-speed-20ft (convert/mph->km-hr) (convert/wind-speed-20ft->wind-speed-10m))
    (convert/lb-ft3->kg-m3 crown-bulk-density)
    (convert/dec->percent estimated-fine-fuel-moisture))))
```

Once the crown fire spread rate is determined, the crown fire line intensity and flame lengths may be derived using the following formulas:

$$I_c = \frac{R_c B (Z - Z_b) h}{60}$$
$$L_c = 0.45(I + I_c)^{0.46}$$

where I_c is the crown fire line intensity in Btu/ft/s, R_c is the crown fire spread rate (either CROS_A or CROS_P) in ft/min, B is the crown bulk density in lb/ft^3, Z is the canopy height in ft, Z_b is the canopy base height in ft, h is the fuel model heat of combustion (generally 8000 Btu/lb), L_c is the crown flame length in ft, and I is the surface fire line intensity in Btu/ft/s.

```
;; heat of combustion is h from the fuel models (generally 8000 Btu/lb)
(defn crown-fire-line-intensity
  "Returns the crown fire line intensity in Btu/ft*s OR kW/m, given:
   - crown spread rate (ft/min OR m/min)
   - crown bulk density (lb/ft^3 OR kg/m^3)
   - canopy height difference (canopy height - canopy base height) (ft OR m)
   - heat of combustion (Btu/lb OR kJ/kg)

   (ft/min * lb/ft^3 * ft * Btu/lb)/60 = (Btu/ft*min)/60 = Btu/ft*s
   OR
   (m/min * kg/m^3 * m * kJ/kg)/60 = (kJ/m*min)/60 = kJ/m*s = kW/m"
  ^double
  [^double crown-spread-rate ^double crown-bulk-density ^double canopy-height-difference ^double heat-of-combustion]
  (-> crown-spread-rate
      (* crown-bulk-density)
      (* canopy-height-difference)
      (* heat-of-combustion)
      (/ 60.0)))

;; FIXME: unused
(defn crown-fire-line-intensity-elmfire
  "Returns the crown fire line intensity in kW/m, given:
   - surface-fire-line-intensity (kW/m)
   - crown-spread-rate (ft/min)
   - crown-bulk-density (kg/m^3)
   - canopy height difference (canopy height - canopy base height) (m)
   - heat of combustion (kJ/kg) <-- Set to a constant of 18,000 kJ/kg.

   kW/m + (m/min * kg/m^3 * m * kJ/kg)/60 = kW/m + (kJ/m*min)/60 = kW/m + kJ/m*s = kW/m + kW/m = kW/m"
  ^double
  [^double surface-fire-line-intensity ^double crown-spread-rate ^double crown-bulk-density ^double canopy-height-difference]
  (+ surface-fire-line-intensity
     (crown-fire-line-intensity
       (convert/ft->m crown-spread-rate) ;; m/min
       crown-bulk-density
       canopy-height-difference
       18000.0))) ;; kJ/kg
```

As with surface fire spread, the wind speed (this time the 20-ft wind speed in mph U_{20}) is used to compute the length to width ratio $\frac{L}{W}$ of an ellipse that approximates the crown fire front using equation 9 from Rothermel 1991.(Rothermel, 1991) This length to width ratio is then converted into an eccentricity measure of the ellipse using equation 8 from Albini and Chase 1980.(Albini and Chase, 1980) Finally, this eccentricity E is used to project the maximum spread rate to any point along the fire front. Here are the formulas used:

$$\frac{L}{W} = 1 + 0.125\, U_{20}\, \mathrm{EAF}$$

$$E = \frac{\sqrt{(\frac{L}{W})^2 - 1}}{\frac{L}{W}}$$

$$R_\theta = R_{\max}\left(\frac{1-E}{1-E\cos\theta}\right)$$

where θ is the angular offset from the direction of maximum fire spread, R_{max} is the maximum spread rate, R_θ is the spread rate in direction θ, and EAF is the ellipse adjustment factor, a term introduced by Marco Morais and Seth Peterson in their HFire work that can be increased or decreased to make the fire shape more elliptical or circular respectively.(Peterson et al., 2009)

```
(defn crown-length-to-width-ratio
  "Calculate the length-to-width ratio of the crown fire front using eq. 9 from
   Rothermel 1991 given:
   - wind-speed-20ft (mph)
   - ellipse-adjustment-factor (dimensionless, < 1.0 circular, > 1.0 elliptical)

   L/W = 1 + 0.125 * U20_mph * EAF"
  ^double
  [^double wind-speed-20ft ^double ellipse-adjustment-factor]
  (-> 0.125
      (* wind-speed-20ft)
      (* ellipse-adjustment-factor)
      (+ 1.0)))

(defn crown-fire-eccentricity
  "Calculate the eccentricity (E) of the crown fire front using eq. 9 from
   Rothermel 1991, and eq. 8 from Albini and Chase 1980 given:
   - wind-speed-20ft (mph)
   - ellipse-adjustment-factor (dimensionless, < 1.0 circular, > 1.0 elliptical)

   L/W = 1 + 0.125 * U20_mph * EAF
   E = sqrt( L/W^2 - 1 ) / L/W"
  ^double
  [^double wind-speed-20ft ^double ellipse-adjustment-factor]
  (let [length-width-ratio (crown-length-to-width-ratio wind-speed-20ft ellipse-adjustment-factor)]
    (-> length-width-ratio
        (Math/pow 2.0)
        (- 1.0)
        (Math/sqrt)
        (/ length-width-ratio))))

;; FIXME: unused
(defn elmfire-length-to-width-ratio
  "true/false mph int>0 ft/min
   Crown L/W = min(1.0 + 0.125*U20_mph, L/W_max)
   Surface L/W = 0.936*e^(0.2566*Ueff_mph) + 0.461*e^(-0.1548*Ueff_mph) - 0.397"
  ^double
  [crown-fire? ^double wind-speed-20ft ^double max-length-to-width-ratio ^double effective-wind-speed]
  (if crown-fire?
    (min (+ 1.0 (* 0.125 wind-speed-20ft)) max-length-to-width-ratio)
    (min (+ (* 0.936 (Math/exp (/ (* 0.2566 effective-wind-speed 60.0) 5280.0)))
            (* 0.461 (Math/exp (/ (* -0.1548 effective-wind-speed 60.0) 5280.0)))
            -0.397)
         8.0)))
```

This concludes our discussion of the crown fire behavior formulas used in GridFire.
FIXME document estimation of directional spread.

5.4 Fire Spread on a Raster Grid

Although Rothermel's spread rate formula provides some useful insight into how quickly a fire's leading edge may travel, it offers no specific mechanism for simulating fire movement in two or more dimensions. Therefore, when attempting to use the Rothermel equations in any spatial analysis, one must begin by choosing a model of space and then decide how best to employ the spread rate equations along each possible burn trajectory.

In GridFire, SIG adopted a raster grid view of space so as to reduce the potentially exponential complexity of modeling a fractal shape (i.e., fire front) at high resolutions using vector approximation. This also provided the practical benefit of being able to work directly with widely used raster datasets, such as LANDFIRE, without a geometric lookup step or *a priori* translation to vector space.

In simulation tests versus FARSITE on several historical California fires, Marco Morais wrote that he saw similarly accurate results from both his HFire model and from FARSITE but experienced several orders of magnitude improvement in runtime efficiency.(Peterson et al., 2011, 2009, Morais, 2001) His explanation for this phenomenon was in the same vein as that described above, namely, that it was FARSITE's choice of vector space that slowed it down versus the faster raster-based HFire system.

Taking a cue from HFire's success in this regard, GridFire has adopted HFire's two-dimensional spread algorithm, called the *method of adaptive timesteps and fractional distances.* (Peterson et al., 2011, 2009, Morais, 2001) The following pseudo-code lays out the steps taken in this procedure:

5.4.1 FIXME old explanation

1. Inputs

   (a) Read in the values shown in Table 4.

Table 4: Inputs to SIG's raster-based fire behavior model

Value	Units	Type
max-runtime	minutes	double
cell-size	feet	double
elevation-matrix	feet	core.matrix 2D double array
slope-matrix	vertical feet/horizontal feet	core.matrix 2D double array
aspect-matrix	degrees clockwise from north	core.matrix 2D double array
fuel-model-matrix	fuel model numbers 1-256	core.matrix 2D double array
canopy-height-matrix	feet	core.matrix 2D double array
canopy-base-height-matrix	feet	core.matrix 2D double array
crown-bulk-density-matrix	lb/ft^3	core.matrix 2D double array
canopy-cover-matrix	0-100	core.matrix 2D double array
wind-speed-20ft	miles/hour	double
wind-from-direction	degrees clockwise from North	double
fuel-moisture	%	map of doubles per fuel size class
foliar-moisture	%	double
ellipse-adjustment-factor	< 1.0 = circle, > 1.0 = ellipse	double
initial-ignition-site	point represented as [row col]	vector

2. Initialization

   (a) Verify that **initial-ignition-site** and at least one of its neighboring cells has a burnable fuel model (not 91-99). Otherwise, terminate the simulation, indicating that no fire spread is possible.

(b) Create three new matrices, called **fire-spread-matrix**, **flame-length-matrix**, and **fire-line-intensity-matrix**. All three are initialized to zero except for a value of 1 at the **initial-ignition-site**.

(c) Set **global-clock** to 0. This will track the amount of time that has passed since the initial ignition in minutes.

(d) Create a new hash-map, called **ignited-cells**, which maps the **initial-ignition-site** to a set of trajectories into each of its burnable neighbors. See "Computing Burn Trajectories" below for the steps used in this procedure.

3. Computing Burn Trajectories

(a) Look up the fuel model, slope, aspect, canopy height, canopy base height, crown bulk density, and canopy cover associated with the ignited cell in the input matrices.

(b) Calculate the dead herbaceous size class parameters, live moisture of extinction, and size class weighting factors for this fuel model.

(c) Use the Rothermel equations to calculate the minimum surface rate of spread (i.e., wind = slope = 0) leaving this cell.

(d) Compute Albini and Baughman's wind adjustment factor for this cell using the fuel bed depth, canopy height, and canopy cover. Multiply this value by the 20-ft wind speed to derive the local midflame wind speed.

(e) Calculate the maximum surface rate of spread (and bearing) originating from this cell using the Rothermel equations and taking into account the effects of downhill and cross-slope winds as described in Rothermel 1983.

(f) Use the Cruz formulas to calculate the maximum crown fire spread rate from the 20-ft wind speed, crown bulk density, and dead 1-hr fuel moisture.

(g) Determine the surface and crown elliptical eccentricities by calculating their length-to-width ratios using the equations from Rothermel 1991.

(h) For each burnable neighboring cell:

i. Use the eccentricity values to determine the possible surface and crown rates of spread into it from the ignited cell.

ii. Compute Byram's surface fire line intensity and Rothermel's crown intensity from these spread rates.

iii. Apply Van Wagner's crown initiation model to determine if the fire will be a passive or active crown fire or remain a surface fire.

iv. In the surface fire case, the spread rate into this neighbor will simply be the surface spread rate calculated above. The fire line intensity is the surface fire line intensity, and the flame length is calculated from this intensity value using Byram's relation.

v. In the case of a crown fire, the spread rate into this neighbor will be the maximum of the surface and crown spread rates. The fire line intensity is the sum of the surface and crown intensities, and the flame length is once again computed from Byram's relation.

vi. Store this neighboring cell, the bearing to it from the ignited cell, and the spread rate, fire line intensity, and flame length values computed above in a burn trajectory record. Also include the terrain (e.g., 3d) distance between this cell and the ignited cell. Finally, set its **fractional-distance** value to be 0, or in the event that this bearing matches an overflow bearing from a previous iteration, set it to the **overflow-heat** value.

(i) Return a collection of burn trajectory records, one per burnable neighboring cell.

4. Main Loop

(a) If **global-clock** has not yet reached **max-runtime** and **ignited-cells** is not empty, proceed to 4.(b). Otherwise, jump to 5.(a).

(b) The timestep for this iteration of the model is calculated by dividing **cell-size** by the maximum spread rate into any cell from those cells in the **ignited-cells** map. As spread rates increase, the timesteps grow shorter and the model takes more iterations to complete. Similarly, the model has longer timesteps and takes less iterations as spread rates decrease. This is called the *method of adaptive timesteps*.

(c) If the timestep calculated in 4.(b) would cause the **global-clock** to exceed the max-runtime, then the timestep is set to the difference between **max-runtime** and **global-clock**.

(d) For each burn trajectory in **ignited-cells**:

i. Multiply the spread rate (ft/min) by the timestep (min) to get the distance traveled by the fire (ft) along this path during this iteration.

ii. Divide this distance traveled by the terrain distance between these two cells to get the new spread fraction $\in [0, 1]$ and increment the **fractional-distance** associated with the trajectory by this value.

iii. If the new **fractional-distance** is greater than or equal to 1, append this updated burn trajectory record to a list called **ignition-events**.

(e) If more than one trajectory in **ignition-events** shares the same target cell, retain only the trajectory with the largest **fractional-distance** value.

(f) For each trajectory in **ignition-events**:

i. Set the target cell's value to 1 in **fire-spread-matrix**, **flame-length** in **flame-length-matrix**, and **fire-line-intensity** in **fire-line-intensity-matrix**.

ii. If the target cell has any burnable neighbors, append an entry to **ignited-cells**, mapping this cell to each of the burn trajectories emanating from it, which are calculated by following the steps in section "Computing Burn Trajectories" above. If its **fractional-distance** value is greater than 1, add the overflow amount above 1 to the outgoing trajectory with the same bearing along which this cell was ignited. That is, if this cell was ignited by a neighbor to the southeast, then pass any overflow heat onto the trajectory leading to the northwest.

(g) Remove any trajectories from **ignited-cells** that have as their targets any of the cells in **ignition-events**.

(h) Remove any cells from **ignited-cells** that no longer have any burnable neighbors.

(i) Increment the **global-clock** by this iteration's **timestep**.

(j) Repeat from 4.(a).

5. Outputs

(a) Return an associative map with the fields shown in Table 5.

Table 5: Outputs from SIG's raster-based fire behavior model

Value	Units	Type
global-clock	minutes	double
initial-ignition-site	point represented as [row col]	vector
ignited-cells	list of points represented as [row col]	list of vectors
fire-spread-matrix	[0,1]	core.matrix 2D double array
flame-length-matrix	feet	core.matrix 2D double array
fire-line-intensity-matrix	Btu/ft/s	core.matrix 2D double array

```
(ns gridfire.fire-spread-old
  (:require [clojure.core.reducers         :as r]
            [gridfire.common               :refer [burnable-fuel-model?
                                                   burnable?
                                                   calc-fuel-moisture
                                                   in-bounds?
                                                   burnable-neighbors?
                                                   get-neighbors
                                                   distance-3d
                                                   non-zero-indices]]
            [gridfire.conversion           :refer [mph->fpm]]
            [gridfire.crown-fire           :refer [crown-fire-eccentricity
                                                   crown-fire-line-intensity
                                                   cruz-crown-fire-spread
                                                   van-wagner-crown-fire-initiation?]]
            [gridfire.fire-spread          :refer [rothermel-surface-fire-wrapped]]
            [gridfire.fuel-models-old      :refer [build-fuel-model moisturize]]
            [gridfire.grid-lookup          :as grid-lookup]
            [gridfire.spotting-old         :as spot]
            [gridfire.surface-fire-old     :refer [anderson-flame-depth
                                                   byram-fire-line-intensity
                                                   byram-flame-length
                                                   rothermel-surface-fire-spread-any
                                                   rothermel-surface-fire-spread-max
                                                   rothermel-surface-fire-spread-no-wind-no-slope
                                                   wind-adjustment-factor]]
            [tech.v3.datatype              :as d]
            [tech.v3.datatype.functional   :as dfn]
            [tech.v3.tensor                :as t]
            [taoensso.tufte :as tufte]))

;; for surface fire, tau = 10 mins, t0 = 0, and t = global-clock
;; for crown fire, tau = 20 mins, t0 = time of first torch, t = global-clock
;; (defn lautenberger-spread-acceleration
;;   [equilibrium-spread-rate t0 t tau]
;;   (* equilibrium-spread-rate (- 1.0 (Math/exp (/ (- t0 t 0.2) tau)))))
;;
;; Note: Because of our use of adaptive timesteps, if the spread rate on
;;       the first timestep is not at least 83 ft/min, then the timestep will
;;       be calculated as greater than 60 minutes, which will terminate the
;;       one hour fire simulation instantly.

(defn random-cell
  "Returns a random [i j] pair with i < num-rows and j < num-cols."
  [num-rows num-cols]
  [(rand-int num-rows)
   (rand-int num-cols)])

(def offset-to-degrees
  "Returns clockwise degrees from north."
  {[-1  0] 0.0   ; N
   [-1  1] 45.0  ; NE
   [ 0  1] 90.0  ; E
   [ 1  1] 135.0 ; SE
```

```
   [ 1  0] 180.0 ; S
   [ 1 -1] 225.0 ; SW
   [ 0 -1] 270.0 ; W
   [-1 -1] 315.0}) ; NW

(defn rothermel-fast-wrapper
  [fuel-model-number fuel-moisture grass-suppression?]
  (let [fuel-model      (-> (build-fuel-model (int fuel-model-number))
                            (moisturize fuel-moisture))
        spread-info-min (rothermel-surface-fire-spread-no-wind-no-slope fuel-model grass-suppression?)]
    [fuel-model spread-info-min]))

(defrecord BurnTrajectory
    [cell
     source
     trajectory ;;=> use integer, bit, or angle
     ^double terrain-distance
     ^double spread-rate
     ^double fire-line-intensity
     ^double flame-length
     fractional-distance
     fire-type
     crown-fire?])

(defn compute-burn-trajectory
  [neighbor here surface-fire-min surface-fire-max crown-bulk-density
   canopy-cover canopy-height canopy-base-height foliar-moisture crown-spread-max
   crown-eccentricity elevation-matrix cell-size overflow-trajectory overflow-heat
   crown-type]
  (let [trajectory                (mapv - neighbor here)
        spread-direction          (offset-to-degrees trajectory)
        surface-spread-rate       (rothermel-surface-fire-spread-any surface-fire-max
                                                                     spread-direction)
        residence-time            (:residence-time surface-fire-min)
        reaction-intensity        (:reaction-intensity surface-fire-min)
        surface-intensity         (->> (anderson-flame-depth surface-spread-rate residence-time)
                                       (byram-fire-line-intensity reaction-intensity))
        crown-fire?               (van-wagner-crown-fire-initiation? canopy-cover
                                                                     canopy-base-height
                                                                     foliar-moisture
                                                                     surface-intensity)
        ^double crown-spread-rate (when crown-fire?
                                    (rothermel-surface-fire-spread-any
                                     (assoc surface-fire-max
                                            :max-spread-rate crown-spread-max
                                            :eccentricity crown-eccentricity)
                                     spread-direction))
        ^double crown-intensity   (when crown-fire?
                                    (crown-fire-line-intensity crown-spread-rate
                                                               crown-bulk-density
                                                               (- canopy-height canopy-base-height)
                                                               (:heat-of-combustion surface-fire-min))) ; 0 = dead-1hr
        spread-rate               (if crown-fire?
                                    (max surface-spread-rate crown-spread-rate)
                                    surface-spread-rate)
        fire-line-intensity       (if crown-fire?
                                    (+ surface-intensity crown-intensity)
                                    surface-intensity)
        flame-length              (byram-flame-length fire-line-intensity)]
    (->BurnTrajectory neighbor
                      here
                      trajectory
                      (distance-3d elevation-matrix cell-size here neighbor)
                      spread-rate
                      fire-line-intensity
                      flame-length
                      (volatile! (if (= trajectory overflow-trajectory)
                                   overflow-heat
```

```
                                0.0))
                      (if crown-fire? crown-type :surface)
                      crown-fire?)))

;;TODO Optimize me!
(defn compute-neighborhood-fire-spread-rates!
  "Returns a vector of entries of the form:
  {:cell [i j],
   :trajectory [di dj],
   :terrain-distance ft,
   :spread-rate ft/min,
   :fire-line-intensity Btu/ft/s,
   :flame-length ft,
   :fractional-distance [0-1]}, one for each cell adjacent to here."
  [{:keys
    [get-aspect get-canopy-base-height get-canopy-cover get-canopy-height get-crown-bulk-density
     get-fuel-model get-slope elevation-matrix fuel-model-matrix get-wind-speed-20ft
     get-wind-from-direction get-temperature get-relative-humidity get-foliar-moisture
     ellipse-adjustment-factor cell-size num-rows num-cols get-fuel-moisture-dead-1hr
     get-fuel-moisture-dead-10hr get-fuel-moisture-dead-100hr get-fuel-moisture-live-herbaceous
     get-fuel-moisture-live-woody grass-suppression?]}
   fire-spread-matrix
   [i j :as here]
   overflow-trajectory
   overflow-heat
   global-clock]
  (let [band                          (int (/ global-clock 60.0))
        aspect                        (grid-lookup/double-at get-aspect i j)
        canopy-base-height            (grid-lookup/double-at get-canopy-base-height i j)
        canopy-height                 (grid-lookup/double-at get-canopy-height i j)
        canopy-cover                  (grid-lookup/double-at get-canopy-cover i j)
        crown-bulk-density            (grid-lookup/double-at get-crown-bulk-density i j)
        fuel-model                    (grid-lookup/double-at get-fuel-model i j)
        slope                         (grid-lookup/double-at get-slope i j)
        relative-humidity             (grid-lookup/double-at get-relative-humidity band i j)
        temperature                   (grid-lookup/double-at get-temperature band i j)
        wind-speed-20ft               (grid-lookup/double-at get-wind-speed-20ft band i j)
        wind-from-direction           (grid-lookup/double-at get-wind-from-direction band i j)
        fuel-moisture-dead-1hr        (if get-fuel-moisture-dead-1hr
                                        (grid-lookup/double-at get-fuel-moisture-dead-1hr band i j)
                                        (calc-fuel-moisture relative-humidity temperature :dead :1hr))
        fuel-moisture-dead-10hr       (if get-fuel-moisture-dead-10hr
                                        (grid-lookup/double-at get-fuel-moisture-dead-10hr band i j)
                                        (calc-fuel-moisture relative-humidity temperature :dead :10hr))
        fuel-moisture-dead-100hr      (if get-fuel-moisture-dead-100hr
                                        (grid-lookup/double-at get-fuel-moisture-dead-100hr band i j)
                                        (calc-fuel-moisture relative-humidity temperature :dead :100hr))
        fuel-moisture-live-herbaceous (if get-fuel-moisture-live-herbaceous
                                        (grid-lookup/double-at get-fuel-moisture-live-herbaceous i j)
                                        (calc-fuel-moisture relative-humidity temperature :live :herbaceous))
        fuel-moisture-live-woody      (if get-fuel-moisture-live-woody
                                        (grid-lookup/double-at get-fuel-moisture-live-woody i j)
                                        (calc-fuel-moisture relative-humidity temperature :live :woody))
        foliar-moisture               (grid-lookup/double-at get-foliar-moisture band i j)
        surface-fire-min              (rothermel-surface-fire-wrapped
                                        fuel-model
                                        [fuel-moisture-dead-1hr
                                         fuel-moisture-dead-10hr
                                         fuel-moisture-dead-100hr
                                         0.0                ; fuel-moisture-dead-herbaceous
                                         fuel-moisture-live-herbaceous
                                         fuel-moisture-live-woody]
                                        grass-suppression?)
        midflame-wind-speed           (mph->fpm
                                        (* wind-speed-20ft
                                          (wind-adjustment-factor (:fuel-bed-depth surface-fire-min)
                                            canopy-height
                                            canopy-cover)))
```

```
        surface-fire-max              (rothermel-surface-fire-spread-max surface-fire-min
                                        midflame-wind-speed
                                        wind-from-direction
                                        slope
                                        aspect
                                        ellipse-adjustment-factor)
        crown-spread-max              (cruz-crown-fire-spread wind-speed-20ft crown-bulk-density fuel-moisture-dead-1hr)
        crown-type                    (if (neg? crown-spread-max) :passive-crown :active-crown)
        crown-spread-max              (Math/abs crown-spread-max)
        crown-eccentricity            (crown-fire-eccentricity wind-speed-20ft
                                        ellipse-adjustment-factor)]
    (into []
          (comp
           (filter #(and (in-bounds? num-rows num-cols %)
                         (burnable? fire-spread-matrix fuel-model-matrix here %)))
           (map #(compute-burn-trajectory % here surface-fire-min surface-fire-max
                                          crown-bulk-density canopy-cover canopy-height
                                          canopy-base-height foliar-moisture crown-spread-max
                                          crown-eccentricity elevation-matrix cell-size
                                          overflow-trajectory overflow-heat crown-type)))
           (get-neighbors here))))

(defn- get-old-fractional-distance
  [{:keys [trajectory-combination]} {:keys [fractional-distance]} fractional-distance-matrix [i j]]
  (if (= trajectory-combination :sum)
    (t/mget fractional-distance-matrix i j)
    @fractional-distance))

(defn- update-fractional-distance-matrix!
  "Update the fractional distance matrix with the largest fractional distance calculated."
  [fractional-distance-matrix max-fractionals]
  (doseq [[cell fractional-distance] @max-fractionals]
    (let [[i j] cell]
      (t/mset! fractional-distance-matrix i j fractional-distance))))

(defn- update-fractional-distance!
  "Update fractional distance for given trajectory into the current cell. Return a tuple of [old-value new-value]"
  [{:keys [trajectory-combination] :as inputs} max-fractionals trajectory fractional-distance-matrix timestep cell]
  (let [terrain-distance    (double (:terrain-distance trajectory))
        spread-rate         (double (:spread-rate trajectory)) ;TODO recompute spread rates when crossing hourly boundary
        new-spread-fraction (/ (* spread-rate timestep) terrain-distance)
        old-total           (get-old-fractional-distance inputs trajectory fractional-distance-matrix cell)
        new-total           (+ old-total new-spread-fraction)]
    (if (= trajectory-combination :sum)
      (let [max-fractional-distance (max (get @max-fractionals cell 0.0) new-total)]
        (swap! max-fractionals assoc cell max-fractional-distance))
      (vreset! (:fractional-distance trajectory) new-total))
    [old-total new-total]))

(defn- update-overflow-heat
  [{:keys [num-rows num-cols]} fractional-distance-matrix {:keys [cell trajectory]} fractional-distance]
  (let [[i j :as target] (mapv + cell trajectory)]
    (when (in-bounds? num-rows num-cols target)
      (t/mset! fractional-distance-matrix i j (- fractional-distance 1.0)))))

(defn ignition-event-reducer
  [inputs max-fractionals fractional-distance-matrix timestep trajectory-combination fire-spread-matrix
   acc trajectory]
  (let [{:keys [source cell]}                trajectory ;TODO cell -> target
        [i j]                                source
        [^double old-total ^double new-total] (update-fractional-distance! inputs
                                                                           max-fractionals
                                                                           trajectory
                                                                           fractional-distance-matrix
                                                                           timestep
                                                                           cell)]
    (if (and (>= new-total 1.0)
             (> new-total ^double (get-in acc [cell :fractional-distance] 0.0)))
```

```
      (do (when (and (= trajectory-combination :sum) (> new-total 1.0))
            (update-overflow-heat inputs fractional-distance-matrix trajectory new-total))
          (assoc! acc cell (merge trajectory {:fractional-distance  new-total
                                              :dt-adjusted          (* (/ (- 1.0 old-total) (- new-total old-total))
                                                                       timestep)
                                              :ignition-probability (t/mget fire-spread-matrix i j)})))
      acc)))

(defn find-new-ignitions ;51%
  [{:keys [trajectory-combination] :as inputs}
   {:keys [fire-spread-matrix fractional-distance-matrix]}
   burn-trajectories
   ^double timestep]
  (let [max-fractionals (atom {})
        reducer-fn      (fn [acc trajectory]
                          (ignition-event-reducer inputs max-fractionals fractional-distance-matrix
                                                  timestep trajectory-combination fire-spread-matrix
                                                  acc trajectory))
        ignition-events (->> burn-trajectories
                             (reduce reducer-fn (transient {}))
                             persistent!
                             vals)]
    (when (= trajectory-combination :sum)
      (update-fractional-distance-matrix! fractional-distance-matrix max-fractionals))
    ignition-events))

;; Tufte 31%
(defn update-burn-trajectories
  [{:keys [fuel-model-matrix num-rows num-cols parallel-strategy] :as constants}
   burn-trajectories
   ignition-events
   fire-spread-matrix
   global-clock]
  (let [parallel-bin-size        (max 1 (quot (count ignition-events) (.availableProcessors (Runtime/getRuntime))))
        newly-burn-trajectories  (into #{} (map :cell) ignition-events)
        pruned-burn-trajectories (into [] (remove #(contains? newly-burn-trajectories (:cell %))) burn-trajectories)
        reducer-fn               (if (= parallel-strategy :within-fires)
                                   #(->> (r/fold parallel-bin-size r/cat r/append! %)
                                         (reduce (fn [acc v] (into acc v)) pruned-burn-trajectories))
                                   #(reduce (fn [acc v] (into acc v)) pruned-burn-trajectories %))]
    (->> ignition-events
         (r/map (fn [{:keys [cell trajectory fractional-distance]}]
                  (let [fractional-distance (double fractional-distance)]
                    (when (burnable-neighbors? fire-spread-matrix
                                               fuel-model-matrix
                                               num-rows num-cols
                                               cell)
                      (compute-neighborhood-fire-spread-rates!
                       constants
                       fire-spread-matrix
                       cell
                       trajectory
                       (- fractional-distance 1.0)
                       global-clock)))))
         (r/remove nil?)
         (reducer-fn))))

(defn generate-burn-trajectories
  [inputs fire-spread-matrix cells]
  (reduce (fn [burn-trajectories cell]
            (into burn-trajectories
                  (compute-neighborhood-fire-spread-rates! inputs
                                                           fire-spread-matrix
                                                           cell
                                                           nil
                                                           0.0
                                                           0.0)))
          []
```

```
          cells))

(defn identify-spot-ignition-events
  [global-clock spot-ignitions]
  (let [to-ignite-now (group-by (fn [[_ [time _]]]
                                  (let [time (double time)]
                                    (>= ^double global-clock time)))
                                spot-ignitions)
        ignite-later  (into {} (get to-ignite-now false))
        ignite-now    (into {} (get to-ignite-now true))]
    [ignite-later ignite-now]))

(defn spot-burn-trajectories
  "Updates matrices for spot ignited cells
  Returns a map of ignited cells"
  [constants
   global-clock
   {:keys [fire-spread-matrix burn-time-matrix spread-rate-matrix fire-type-matrix
           flame-length-matrix fire-line-intensity-matrix spot-matrix]}
   spot-ignite-now]
  (let [ignited?          (fn [[k v]]
                            (let [[i j] k
                                  [_ p] v]
                              (> ^double (t/mget fire-spread-matrix i j) ^double p)))
        spot-ignite-now   (remove ignited? spot-ignite-now)
        burn-trajectories (generate-burn-trajectories constants
                                                      fire-spread-matrix
                                                      (keys spot-ignite-now))]
    (doseq [cell spot-ignite-now
            :let [[i j]                    (key cell)
                  [_ ignition-probability] (val cell)]]
      (t/mset! fire-spread-matrix i j ignition-probability)
      (t/mset! burn-time-matrix i j global-clock)
      (t/mset! flame-length-matrix i j 1.0)
      (t/mset! fire-line-intensity-matrix i j 1.0)
      (t/mset! spread-rate-matrix i j -1.0)
      (t/mset! fire-type-matrix i j -1.0)
      (t/mset! spot-matrix i j 1.0))
    burn-trajectories))

(defn new-spot-ignitions
  "Returns a map of [x y] locations to [t p] where:
  t: time of ignition
  p: ignition-probability"
  [{:keys [spotting] :as inputs} matrices ignition-events global-clock]
  (when spotting
    (reduce (fn [acc ignition-event]
              (merge-with (partial min-key first)
                          acc
                          (->> (spot/spread-firebrands
                                inputs
                                matrices
                                ignition-event
                                global-clock)
                               (into {}))))
            {}
            ignition-events)))

(def fire-type-to-value
  {:surface       1.0
   :passive-crown 2.0
   :active-crown  3.0})

(defn- find-max-spread-rate ^double
  [^double max-spread-rate ^BurnTrajectory burn-trajectory]
  (Math/max max-spread-rate ^double (:spread-rate burn-trajectory)))

(defn- compute-dt ^double
```

```
  [^double cell-size burn-trajectories]
  (if (seq burn-trajectories)
    (let [max-spread-rate (double (reduce find-max-spread-rate 0.0 burn-trajectories))]
      (/ cell-size max-spread-rate))
    10.0))

(defn- compute-spot-trajectories
  [inputs matrices global-clock ignition-events spot-ignitions]
  (let [new-spot-ignitions     (new-spot-ignitions inputs ;TODO optimize
                                                   matrices
                                                   ignition-events
                                                   global-clock)
        [spot-ignite-later
         spot-ignite-now]      (identify-spot-ignition-events global-clock ;TODO optimize
                                                              (merge-with (partial min-key first)
                                                                          spot-ignitions
                                                                          new-spot-ignitions))
        spot-burn-trajectories (spot-burn-trajectories inputs ;TODO optimize
                                                       global-clock
                                                       matrices
                                                       spot-ignite-now)]
    [spot-ignite-later spot-burn-trajectories]))

(defn- store-ignition-events!
  [{:keys [fire-spread-matrix flame-length-matrix fire-line-intensity-matrix burn-time-matrix
           spread-rate-matrix fire-type-matrix]}
   global-clock
   ignition-events]
  (doseq [{:keys
           [cell flame-length fire-line-intensity
            ignition-probability spread-rate fire-type
            dt-adjusted]} ignition-events] ;TODO investigate using records for ignition-events
    (let [[i j] cell]
      (t/mset! fire-spread-matrix         i j ignition-probability)
      (t/mset! flame-length-matrix        i j flame-length)
      (t/mset! fire-line-intensity-matrix i j fire-line-intensity)
      (t/mset! burn-time-matrix           i j (+ global-clock ^double dt-adjusted))
      (t/mset! spread-rate-matrix         i j spread-rate)
      (t/mset! fire-type-matrix           i j (fire-type fire-type-to-value)))))) ;TODO Use number

(defn run-loop
  [{:keys [max-runtime cell-size ignition-start-time] :as inputs}
   {:keys
    [fire-spread-matrix flame-length-matrix fire-line-intensity-matrix burn-time-matrix
     spread-rate-matrix fire-type-matrix fractional-distance-matrix spot-matrix] :as matrices}
   ignited-cells]
  (let [max-runtime         (double max-runtime)
        cell-size           (double cell-size)
        ignition-start-time (double ignition-start-time)
        ignition-stop-time  (+ ignition-start-time max-runtime)]
    (loop [global-clock      ignition-start-time
           burn-trajectories (generate-burn-trajectories inputs fire-spread-matrix ignited-cells)
           spot-ignitions    {}
           spot-count        0
           crown-count       0]
      (if (and (< global-clock ignition-stop-time)
               (or (seq burn-trajectories) (seq spot-ignitions)))
        (let [timestep        (Math/min (compute-dt cell-size burn-trajectories)
                                        (- ignition-stop-time global-clock))
              ignition-events (find-new-ignitions inputs matrices burn-trajectories timestep)]
          (store-ignition-events! matrices global-clock ignition-events)
          (let [[spot-ignite-later
                 spot-burn-trajectories] (compute-spot-trajectories inputs matrices global-clock
                                                                    ignition-events spot-ignitions)]
            (recur (+ global-clock timestep)
                   (update-burn-trajectories inputs
                                             (into spot-burn-trajectories burn-trajectories)
                                             ignition-events
```

```
                                        fire-spread-matrix
                                        global-clock)
                    spot-ignite-later
                    (+ spot-count (count spot-burn-trajectories))
                    (+ crown-count (count (filterv :crown-fire? ignition-events))))))
         {:global-clock               global-clock
          :exit-condition             (if (>= global-clock ignition-stop-time) :max-runtime-reached :no-burnable-fuels)
          :fire-spread-matrix         fire-spread-matrix
          :flame-length-matrix        flame-length-matrix
          :fire-line-intensity-matrix fire-line-intensity-matrix
          :burn-time-matrix           burn-time-matrix
          :spot-matrix                spot-matrix
          :spread-rate-matrix         spread-rate-matrix
          :fire-type-matrix           fire-type-matrix
          :crown-fire-count           crown-count
          :spot-count                 spot-count})))

(defmulti run-fire-spread
  "Runs the raster-based fire spread model with a map of these arguments:
  - max-runtime: double (minutes)
  - cell-size: double (feet)
  - elevation-matrix: core.matrix 2D double array (feet)
  - slope-matrix: core.matrix 2D double array (vertical feet/horizontal feet)
  - aspect-matrix: core.matrix 2D double array (degrees clockwise from north)
  - fuel-model-matrix: core.matrix 2D double array (fuel model numbers 1-256)
  - canopy-height-matrix: core.matrix 2D double array (feet)
  - canopy-base-height-matrix: core.matrix 2D double array (feet)
  - crown-bulk-density-matrix: core.matrix 2D double array (lb/ft^3)
  - canopy-cover-matrix: core.matrix 2D double array (0-100)
  - wind-speed-20ft: double (miles/hour)
  - wind-from-direction: double (degrees clockwise from north)
  - fuel-moisture: doubles (0-1) {:dead {:1hr :10hr :100hr} :live {:herbaceous :woody}}
  - foliar-moisture: double (0-1)
  - ellipse-adjustment-factor: (< 1.0 = more circular, > 1.0 = more elliptical)
  - initial-ignition-site: One of the following:
     - point represented as [row col]
     - a core.matrix 2D double array (0-2)
  - num-rows: integer
  - num-cols: integer"
  (fn [{:keys [initial-ignition-site]}]
    (if (vector? initial-ignition-site)
      :ignition-point
      :ignition-perimeter)))

;;-----------------------------------------------------------------------------
;; Igniiton Point
;;-----------------------------------------------------------------------------

(defn- initialize-point-ignition-matrices
  [{:keys [num-rows num-cols initial-ignition-site ignition-start-time spotting trajectory-combination]}]
  (let [[i j]                       initial-ignition-site
        shape                       [num-rows num-cols]
        burn-time-matrix            (t/new-tensor shape)
        fire-line-intensity-matrix  (t/new-tensor shape)
        fire-spread-matrix          (t/new-tensor shape)
        fire-type-matrix            (t/new-tensor shape)
        firebrand-count-matrix      (when spotting (t/new-tensor shape))
        flame-length-matrix         (t/new-tensor shape)
        fractional-distance-matrix  (when (= trajectory-combination :sum) (t/new-tensor shape))
        spot-matrix                 (t/new-tensor shape) ;;TODO check if spot-matrix requires spotting
        spread-rate-matrix          (t/new-tensor shape)]
    (t/mset! burn-time-matrix i j ignition-start-time)
    (t/mset! fire-line-intensity-matrix i j 1.0)       ;TODO should this be zero?
    (t/mset! fire-spread-matrix i j 1.0)
    (t/mset! fire-type-matrix i j -1.0)                ;TODO should this be zero?
    (t/mset! flame-length-matrix i j 1.0)              ;TODO should this be zero?
    (t/mset! spread-rate-matrix i j -1.0)              ;TODO should this be zero?
    {:burn-time-matrix           burn-time-matrix
```

```
     :fire-line-intensity-matrix fire-line-intensity-matrix
     :fire-spread-matrix         fire-spread-matrix
     :fire-type-matrix           fire-type-matrix
     :firebrand-count-matrix     firebrand-count-matrix
     :flame-length-matrix        flame-length-matrix
     :fractional-distance-matrix fractional-distance-matrix
     :spot-matrix                spot-matrix
     :spread-rate-matrix         spread-rate-matrix}))

(defmethod run-fire-spread :ignition-point
  [{:keys [initial-ignition-site] :as inputs}]
  (run-loop inputs (initialize-point-ignition-matrices inputs) [initial-ignition-site]))

;;------------------------------------------------------------------------------
;; Ignition Perimeter
;;------------------------------------------------------------------------------

(defn- initialize-perimeter-ignition-matrices
  [{:keys [num-rows num-cols spotting trajectory-combination initial-ignition-site]}]
  (let [shape              [num-rows num-cols]
        positive-burn-scar initial-ignition-site
        negative-burn-scar (d/clone (dfn/* -1.0 positive-burn-scar))]
    {:burn-time-matrix           negative-burn-scar
     :fire-line-intensity-matrix (d/clone negative-burn-scar)
     :fire-spread-matrix         (d/clone positive-burn-scar)
     :fire-type-matrix           (d/clone negative-burn-scar)
     :firebrand-count-matrix     (when spotting (t/new-tensor shape))
     :flame-length-matrix        (d/clone negative-burn-scar)
     :fractional-distance-matrix (when (= trajectory-combination :sum) (d/clone positive-burn-scar))
     :spot-matrix                (t/new-tensor shape) ;TODO check if spot-matrix requires spotting
     :spread-rate-matrix         (d/clone negative-burn-scar)}))

(defn- get-non-zero-indices [m]
  (let [{:keys [row-idxs col-idxs]} (non-zero-indices m)]
    (map vector row-idxs col-idxs)))

(defmethod run-fire-spread :ignition-perimeter
  [{:keys [num-rows num-cols initial-ignition-site fuel-model-matrix] :as inputs}]
  (when-let [ignited-cells (->> (get-non-zero-indices initial-ignition-site)
                                (filter #(burnable-neighbors? initial-ignition-site
                                                              fuel-model-matrix
                                                              num-rows
                                                              num-cols
                                                              %))
                                seq)]
    (run-loop inputs (initialize-perimeter-ignition-matrices inputs) ignited-cells)))
```

This concludes our description of GridFire's raster-based fire spread algorithm.

5.4.2 FIXME new explanation

5.4.3 Overview

The physical model behind elliptical wavelets is as follows: the fire perimeter is bounded by a curve, and each point along this curve is the focus point of an infinitesimal ellipse (the elliptical wavelet). After an infinitesimal time-step elapses, the perimeter has grown by set-unioning it with all the infinitesimal wavelets. The speed at which each wavelet grows around its focus point is determined by the above-described Rothermel's equations.

GridFire works by making several **discrete approximations to this model:**

1. space is discretized as a **2D cartesian grid** (hence the "Grid" in GridFire);

2. each elliptical wavelet is approximated by (up to) 8 **burn vectors**. Each burn vector travels from the center of one cell to the center of a neighboring cell, in a direction which is either grid-aligned or diagonal.

3. the state of the simulation is updated in **discrete time steps** (not of equal length).

The GridFire simulation loop maintains a representation of the fire front consisting of a list of ignited cells, and a list of burn vectors. Basically, when the center of a cell is reached for a first time by a burn vector, it becomes ignited, and new burn vectors a created that grow away from the center of that cell. Note, however, that this description of the algorithm is somewhat simplistic, because:

1. In addition to contiguous spread, GridFire updates the ignited cells and burn vectors though **spotting** (embers cast to distant cells) and **suppression** (fire fighters) events.

2. GridFire optionally maintains a "probability" associated with each burn vector, representing a spectrum of more or less optimistic evolutions of the fire.

The speed of a burn vector is set at creation time, and whenever its environment changes: either because it entered a new cells, or because the simulation enters into a new time band for weather conditions. These calculations are derived from the formulas presented in the previous sections.

5.4.4 Core Implementation of Spread

Concretely, this 2D-spread behavior is implemented in **gridfire.fire-spread**

```
(ns gridfire.fire-spread
  (:require [clojure.string                :as s]
            [gridfire.common               :refer [burnable-cell?
                                                   burnable-fuel-model?
                                                   calc-fuel-moisture
                                                   in-bounds-optimal?
                                                   non-zero-indices
                                                   overtakes-lower-probability-fire?
                                                   terrain-distance-fn
                                                   terrain-distance-from-cell-getter
                                                   terrain-distance-invoke]]
            [gridfire.conversion           :refer [deg->rad mph->fpm hour->min min->hour]]
            [gridfire.crown-fire           :refer [crown-fire-eccentricity
                                                   crown-fire-line-intensity
                                                   cruz-crown-fire-spread
                                                   van-wagner-crown-fire-initiation?]]
            [gridfire.elliptical           :as ellip]
            [gridfire.fuel-models          :refer [fuel-models-precomputed
                                                   moisturize]]
            [gridfire.grid-lookup          :as grid-lookup]
            [gridfire.spotting             :as spotting]
            [gridfire.structs.burn-vector  :as burn-vec]
            [gridfire.structs.rfwo         :as rfwo-struct]
            [gridfire.suppression          :as suppression]
            [gridfire.surface-fire         :refer [rothermel-surface-fire-spread-no-wind-no-slope
                                                   rothermel-surface-fire-spread-max
                                                   compute-spread-rate
                                                   anderson-flame-depth
                                                   byram-fire-line-intensity
                                                   byram-flame-length
                                                   wind-adjustment-factor]]
            [gridfire.utils.flow           :refer [case-double]]
            [gridfire.utils.gradient       :as gradient]
            [tech.v3.datatype              :as d]
            [tech.v3.datatype.functional   :as dfn]
            [tech.v3.tensor                :as t])
```

```
  (:import (clojure.lang IFn$OLLDO)
           java.time.ZoneId
           (java.util ArrayList Date)))

(set! *unchecked-math* :warn-on-boxed)
```

Each simulated fire maintains a set of 2D matrices, which are used for several purposes: recording simulation evolution, holding simulation state, and memoizing some results to avoid redundant computation.

```
;;------------------------------------------------------------------------------
;; SimulationMatrices Record
;;------------------------------------------------------------------------------
(defrecord SimulationMatrices
    [burn-time-matrix
     eccentricity-matrix
     fire-line-intensity-matrix
     fire-spread-matrix
     fire-type-matrix
     firebrand-count-matrix
     flame-length-matrix
     directional-flame-length-matrix
     max-spread-direction-matrix
     max-spread-rate-matrix
     modified-time-matrix
     residence-time-matrix
     reaction-intensity-matrix
     spot-matrix
     spread-rate-matrix
     spread-rate-sum-matrix
     travel-lines-matrix                    ; INTRO the set of burn vector directions currently blocked in each cell, encoded as 8 bi
     x-magnitude-sum-matrix
     y-magnitude-sum-matrix])

(def matrix-k->out-of-bounds-value
  {:burn-time-matrix -1.0})

(defn make-simulation-matrices
  [m]
  (map->SimulationMatrices (->> m
                                (map (fn [[k m]]
                                       [k (when (t/tensor? m)
                                            (grid-lookup/add-double-getter m
                                                                           (get matrix-k->out-of-bounds-value k)))]))
                                (into {}))))

(comment

  :fire-spread-matrix ; INTRO keeps track of which cells have burned, as a probability between 0 and 1.
  :burn-probability   ; INTRO holds the same probability values in Burn Vectors.
  ;; The idea is that the simulation simulates not a single fire line,
  ;; but a superposition of fire lines corresponding to a spectrum of 'parallel worlds' with different fire spread.
  ;; A burn probability of 0.84 means that this cell burns in 84% of these parallel worlds.
  ;; INVARIANT: a burn vector with a :burn-probability of P can only ignite cells that have not been reached
  ;; by BVs of :burn-probability higher than P, and will then update :fire-spread-matrix to P.
  ;; If the generated :burn-probability values are only allowed to be 0 or 1,
  ;; we retrieve the more classical and less sophisticated model of a single fire spread.
  ;; Why probabilistic values? The motivation comes from the random nature of spotting.
  ;; WARNING: it is tricky to reason about the probability model underlying this logic.
  ;; In particular, it is NOT equivalent to drawing i.i.d spot ignitions, since burn vectors
  ;; of independent origins can block one another.
  ;; It is an error to think of this probabilistic-updating logic as one would think of, say,
  ;; the diffusion equation associated with a random walk.
  ;; One way to think about this is that there is a 1D latent random variable,
  ;; the 'luck level' of the fire evolution, ranging from 0 to 1;
  ;; A burn vector with :burn-probability 0.23 is present in 23% of the parallel universes,
  ;; those with luck level no greater than 0.23.
```

```
  ;; TODO turn this into a proper analysis in the Org article.

  ;; Here's a summary of this approach by Gary Johnson (https://mattermost.sig-gis.com/cec-project/pl/8go9hbcdciyp5bz6rmdu7sw59o):
  ;; The logic is fairly simple:
  ;; 1. The initial ignition site (or perimeter) is assigned a burn-probability of 1.0 at simulation start time since the fact that t
  ;; 2. All burn vectors originating by means of a surface or crown fire from these cells are considered part of the "main fire" and
  ;; 3. If a burned cell's fire line intensity is high enough to cause either surface or crown spotting, then we cast a number of emb
  ;; 4. Any unburned cells which receive an ember are assigned an ignition probability based on formulas from the academic literature
  ;; 5. Each timestep, we use a random sampling process to determine which ember-containing cells may ignite and become burned. If an
  ;; 6. Whenever burn vectors from different fires (either any combination of the main fire with spot fires or just spot fires with o
  ;; 7. At the end of a simulation, the burn-probability value in each cell (stored in the unfortunately named fire-spread-matrix) sh
  ;; 8. Since spot fires can also cast embers with a high enough fire line intensity, we calculate the burn-probability of the child
  ;; The bulk of the code related to burn-probability is, of course, in the collision detection and resolution logic. If we want to c

  *e)
```

A simulated fire is based on various inputs, describing both the environmental conditions (fuels, weather, topography…) and the initial conditions (point or perimeter ignited at the beginning of the simulation), as well as algorithm parameters:

```
;;-------------------------------------------------------------------------
;; Main Simulation Entry Point - Dispatches to Point/Perimeter Ignition
;;-------------------------------------------------------------------------

;; TODO: Move this multimethod check into run-simulations to avoid running it in every thread
(defmulti ^:private run-fire-spread*
  (fn [inputs]
    (if (vector? (:initial-ignition-site inputs))
      :ignition-point
      :ignition-perimeter)))

(defn run-fire-spread
  "Runs the raster-based fire spread model with a SimulationInputs record containing these fields:
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | Key                                             | Value Type         | Value Units
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | :num-rows                                       | long               | column count of fuel-model-matrix
  | :num-cols                                       | long               | row count of fuel-model-matrix
  | :cell-size                                      | double             | feet
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | :ignition-start-time                            | double             | minutes
  | :max-runtime                                    | double             | minutes
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | :initial-ignition-site                          | [i,j] or 2D tensor | [y,x] coordinate or categories 0-2 in tensor
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | :crowning-disabled?                             | boolean            | true or false
  | :ellipse-adjustment-factor                      | double             | < 1.0 = more circular, > 1.0 = more elliptical
  | :grass-suppression?                             | boolean            | true or false
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | :rand-gen                                       | java.util.Random   | uniform sample [0-1]
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | :get-elevation                                  | (i,j) -> v         | feet
  | :get-slope                                      | (i,j) -> v         | vertical feet/horizontal feet
  | :get-aspect                                     | (i,j) -> v         | degrees clockwise from north [0-360)
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | :get-canopy-cover                               | (i,j) -> v         | percent [0-100]
  | :get-canopy-height                              | (i,j) -> v         | feet
  | :get-canopy-base-height                         | (i,j) -> v         | feet
  | :get-crown-bulk-density                         | (i,j) -> v         | lb/ft^3
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | :get-fuel-model                                 | (i,j) -> v         | fuel model numbers [1-256]
  |-------------------------------------------------+--------------------+-----------------------------------------------------
  | :get-temperature                                | (b,i,j) -> v       | degrees Fahrenheit
  | :get-relative-humidity                          | (b,i,j) -> v       | percent [0-100]
  | :get-wind-speed-20ft                            | (b,i,j) -> v       | miles/hour
  | :get-wind-from-direction                        | (b,i,j) -> v       | degrees clockwise from north
```

```
  |-----------------------------------------------+--------------------+-----------------------------------------------------
  | :get-fuel-moisture-dead-1hr                   | (b,i,j) -> v       | ratio [0-1]
  | :get-fuel-moisture-dead-10hr                  | (b,i,j) -> v       | ratio [0-1]
  | :get-fuel-moisture-dead-100hr                 | (b,i,j) -> v       | ratio [0-1]
  | :get-fuel-moisture-live-herbaceous            | (b,i,j) -> v       | ratio [0-1]
  | :get-fuel-moisture-live-woody                 | (b,i,j) -> v       | ratio [0-1]
  | :get-foliar-moisture                         | (b,i,j) -> v       | ratio [0-1]
  |-----------------------------------------------+--------------------+-----------------------------------------------------
  | :spotting                                     | map                | :decay-constant -> double
  |                                               |                    | :num-firebrands -> long
  |                                               |                    | :surface-fire-spotting -> map
  |                                               |                    | :crown-fire-spotting-percent -> double or [double doub
  |-----------------------------------------------+--------------------+-----------------------------------------------------
  | :fuel-number->spread-rate-adjustment-array-lookup | array of doubles   | unitless
  |-----------------------------------------------+--------------------+-----------------------------------------------------
  | :suppression-dt                               | double             | minutes
  | :suppression-coefficient                      | double             | none
  | :sdi-containment-overwhelming-area-growth-rate | double             | Acres/day
  | :sdi-reference-suppression-speed              | double             | percent/day
  | :sdi-sensitivity-to-difficulty                | double             | none
  |-----------------------------------------------+--------------------+-----------------------------------------------------
  [inputs]
  (let [sfmin-memoization (get-in inputs [:memoization :surface-fire-min])]
    (binding [rothermel-fast-wrapper-optimal (if (= sfmin-memoization :within-sims) ; NOTE :within-sims to avoid the memory leaks co
                                               (memoize-rfwo rothermel-fast-wrapper-optimal)
                                               rothermel-fast-wrapper-optimal)]
      (run-fire-spread* inputs))))

;;-----------------------------------------------------------------------------
;; Point Ignition
;;-----------------------------------------------------------------------------

(defn- initialize-point-ignition-matrices
  [inputs]
  (let [num-rows                    (long (:num-rows inputs))
        num-cols                    (long (:num-cols inputs))
        [i j]                       (:initial-ignition-site inputs)
        ignition-start-time         (:ignition-start-time inputs)
        spotting                    (:spotting inputs)
        compute-directional-values? (:compute-directional-values? inputs)
        shape                       [num-rows num-cols]
        burn-time-matrix            (-> (* num-rows num-cols)
                                        (float-array -1.0)
                                        (t/ensure-tensor)
                                        (t/reshape shape)
                                        (t/mset! i j ignition-start-time))]
    (make-simulation-matrices
     {:burn-time-matrix                burn-time-matrix
      :eccentricity-matrix             (t/new-tensor shape :datatype :float32)
      :fire-line-intensity-matrix      (t/new-tensor shape :datatype :float32)
      :fire-spread-matrix              (-> (t/new-tensor shape :datatype :float32) (t/mset! i j 1.0))
      :fire-type-matrix                (t/new-tensor shape :datatype :float32)
      :firebrand-count-matrix          (when spotting (t/new-tensor shape :datatype :int32))
      :flame-length-matrix             (t/new-tensor shape :datatype :float32)
      :directional-flame-length-matrix (when compute-directional-values? (t/new-tensor shape :datatype :float32))
      :max-spread-direction-matrix     (t/new-tensor shape :datatype :float32)
      :max-spread-rate-matrix          (t/new-tensor shape :datatype :float32)
      :modified-time-matrix            (t/new-tensor shape :datatype :int32)
      :residence-time-matrix           (when compute-directional-values? (t/new-tensor shape :datatype :float32))
      :reaction-intensity-matrix       (when compute-directional-values? (t/new-tensor shape :datatype :float32))
      :spot-matrix                     (when spotting (t/new-tensor shape :datatype :float32))
      :spread-rate-matrix              (t/new-tensor shape :datatype :float32)
      :spread-rate-sum-matrix          (when compute-directional-values? (t/new-tensor shape :datatype :float32))
      :travel-lines-matrix             (t/new-tensor shape :datatype :int16)
      :x-magnitude-sum-matrix          (when compute-directional-values? (t/new-tensor shape :datatype :float32))
      :y-magnitude-sum-matrix          (when compute-directional-values? (t/new-tensor shape :datatype :float32))})))

(defmethod run-fire-spread* :ignition-point
```

```
  [inputs]
  (run-loop inputs
            (initialize-point-ignition-matrices inputs)
            [(:initial-ignition-site inputs)]))

;;-----------------------------------------------------------------------------
;; Perimeter Ignition
;;-----------------------------------------------------------------------------

(defn- add-ignited-cells!
  [matrix ignited-cells value]
  (doseq [[i j] ignited-cells]
    (t/mset! matrix i j value))
  matrix)

(defn- initialize-perimeter-ignition-matrices
  [inputs ignited-cells]
  (let [num-rows                    (long (:num-rows inputs))
        num-cols                    (long (:num-cols inputs))
        positive-burn-scar          (t/clone (:initial-ignition-site inputs) :datatype :float32)
        ignition-start-time         (:ignition-start-time inputs)
        spotting                    (:spotting inputs)
        compute-directional-values? (:compute-directional-values? inputs)
        shape                       [num-rows num-cols]
        negative-burn-scar          (d/clone (dfn/* -1.0 positive-burn-scar))
        burn-time-matrix            (-> (* num-rows num-cols)
                                        (float-array -1.0)
                                        (t/ensure-tensor)
                                        (t/reshape shape)
                                        (add-ignited-cells! ignited-cells ignition-start-time))]
    (make-simulation-matrices
     {:burn-time-matrix                burn-time-matrix
      :eccentricity-matrix             (d/clone negative-burn-scar)
      :fire-line-intensity-matrix      negative-burn-scar
      :fire-spread-matrix              (d/clone positive-burn-scar)
      :fire-type-matrix                (d/clone negative-burn-scar)
      :firebrand-count-matrix          (when spotting (t/new-tensor shape :datatype :int32))
      :flame-length-matrix             (d/clone negative-burn-scar)
      :directional-flame-length-matrix (when compute-directional-values? (d/clone negative-burn-scar))
      :max-spread-direction-matrix     (d/clone negative-burn-scar)
      :max-spread-rate-matrix          (d/clone negative-burn-scar)
      :modified-time-matrix            (t/new-tensor shape :datatype :int32)
      :residence-time-matrix           (when compute-directional-values? (d/clone negative-burn-scar))
      :reaction-intensity-matrix       (when compute-directional-values? (d/clone negative-burn-scar))
      :spot-matrix                     (when spotting (t/new-tensor shape :datatype :float32))
      :spread-rate-matrix              (d/clone negative-burn-scar)
      :spread-rate-sum-matrix          (when compute-directional-values? (t/new-tensor shape :datatype :float32))
      :travel-lines-matrix             (t/new-tensor shape :datatype :int16)
      :x-magnitude-sum-matrix          (when compute-directional-values? (t/new-tensor shape :datatype :float32))
      :y-magnitude-sum-matrix          (when compute-directional-values? (t/new-tensor shape :datatype :float32))})))

;; TODO: Move this step into run-simulations to avoid running it in every thread
(defn- get-perimeter-cells
  [inputs]
  (let [num-rows                    (:num-rows inputs)
        num-cols                    (:num-cols inputs)
        initial-ignition-site       (:initial-ignition-site inputs)
        get-fuel-model              (:get-fuel-model inputs)
        {:keys [row-idxs col-idxs]} (non-zero-indices initial-ignition-site)
        num-idxs                    (d/ecount row-idxs)]
    (loop [idx 0
           acc (transient [])]
      (if (< idx num-idxs)
        (let [i (row-idxs idx)
              j (col-idxs idx)]
          (if (burnable-neighbors? get-fuel-model
                                   (grid-lookup/add-double-getter initial-ignition-site)
                                   1.0
```

```
                                num-rows
                                num-cols
                                i
                                j)
            (recur (inc idx)
                   (conj! acc [i j]))
            (recur (inc idx)
                   acc)))
        (persistent! acc)))))

(defmethod run-fire-spread* :ignition-perimeter
  [inputs]
  (let [ignited-cells (get-perimeter-cells inputs)]
    (when (seq ignited-cells)
      (run-loop inputs
                (initialize-perimeter-ignition-matrices inputs ignited-cells)
                ignited-cells))))
```

The next snippet shows the details of GridFire's fire spread algorithm. In brief, simulation Mmtrices get mutated and the sets of burn vectors and ignited cells get updated through various phases:

```
;;-------------------------------------------------------------------------------
;; Fire spread
;;-------------------------------------------------------------------------------

(defn- find-max-spread-rate ^double
  [^double max-spread-rate burn-vector]
  (max max-spread-rate (burn-vec/get-spread-rate burn-vector)))

(defn- compute-dt ^double
  [^double cell-size burn-vectors]
  (if (pos? (count burn-vectors))
    (/ cell-size (double (reduce find-max-spread-rate 0.0 burn-vectors)))
    10.0)) ; Wait 10 minutes for spot ignitions to smolder and catch fire

#_(defn- compute-terrain-distance-slow
    [inputs ^long i ^long j ^double direction]
    (let [cell-size          (double (:cell-size inputs))
          cell-size-diagonal (double (:cell-size-diagonal inputs))
          get-aspect         (:get-aspect inputs)
          get-slope          (:get-slope inputs)
          ^double aspect     (grid-lookup/double-at get-aspect i j)
          ^double slope      (grid-lookup/double-at get-slope i j)
          theta              (Math/abs (- aspect direction))
          slope-factor       (/
                              (if (<= theta 90.0)
                                (- 90.0 theta)
                                (if (<= theta 180.0)
                                  (- theta 90.0)
                                  (if (<= theta 270.0)
                                    (- 270.0 theta)
                                    (- theta 270.0))))
                              90.0)
          run                (case direction
                               0.0   cell-size
                               45.0  cell-size-diagonal
                               90.0  cell-size
                               135.0 cell-size-diagonal
                               180.0 cell-size
                               225.0 cell-size-diagonal
                               270.0 cell-size
                               315.0 cell-size-diagonal)
          rise               (* run slope slope-factor)]
      (Math/sqrt (+ (* run run) (* rise rise)))))

(defn rothermel-surface-fire-wrapped
```

```
  [^double fuel-model-number fuel-moisture grass-suppression?]
  (-> (long fuel-model-number)
      (fuel-models-precomputed)
      (moisturize fuel-moisture)
      (rothermel-surface-fire-spread-no-wind-no-slope grass-suppression?)))

(defn ^:dynamic ^:redef rothermel-fast-wrapper-optimal
  [rfwo-args]
  (rothermel-surface-fire-wrapped (rfwo-struct/get-fuel-model-number rfwo-args)
                                  (rfwo-struct/get-fuel-moisture-vec rfwo-args)
                                  (rfwo-struct/get-grass-suppression? rfwo-args)))

(defn- memoize-1arg
  "Like clojure.core/memoize, but optimized for 1-argument functions,
  avoiding varargs overhead."
  [f]
  (let [mem       (atom {})
        not-found (Object.)]
    (fn mem-wrapper [arg]
      (let [v (get @mem arg not-found)]
        (if (identical? v not-found)
          (let [v (f arg)]
            (swap! mem assoc arg v)
            v)
          v)))))

(defn memoize-rfwo
  "Memoization function specialized for rothermel-fast-wrapper-optimal."
  [f]
  ;; NOTE currently implemented using clojure.core/memoize, but that will change.
  (memoize-1arg f))

(defn- store-if-max!
  [matrix ^long i ^long j ^double new-value]
  (let [old-value (grid-lookup/mget-double-at matrix i j)]
    (when (> new-value old-value)
      (t/mset! matrix i j new-value))))

(defn- lookup-spread-rate-adjustment
  ^double [fuel-number->spread-rate-adjustment-array-lookup fuel-model]
  (aget (doubles fuel-number->spread-rate-adjustment-array-lookup) fuel-model))

(defn- compute-max-in-situ-values!
  "Computes and saves fire-spread behavior quantities for a cell-band.
  This function must be called the first time a burn-vector appears in the given cell
  during the given hourly band, be because it travelled (transitioned) into it,
  or because the cell got ignited in some other way (e.g. spotting or initial ignition);
  in the first case, `incoming-burnvec` must be non-nil, so that it can be used
  to estimate the fireline-normal direction."
  [inputs matrices band i j incoming-burnvec]
  (let [compute-directional-values?                    (:compute-directional-values? inputs)
        get-slope                                      (:get-slope inputs)
        get-aspect                                     (:get-aspect inputs)
        get-canopy-cover                               (:get-canopy-cover inputs)
        get-canopy-height                              (:get-canopy-height inputs)
        get-canopy-base-height                         (:get-canopy-base-height inputs)
        get-crown-bulk-density                         (:get-crown-bulk-density inputs)
        get-fuel-model                                 (:get-fuel-model inputs)
        get-temperature                                (:get-temperature inputs)
        get-relative-humidity                          (:get-relative-humidity inputs)
        get-wind-speed-20ft                            (:get-wind-speed-20ft inputs)
        get-wind-from-direction                        (:get-wind-from-direction inputs)
        get-fuel-moisture-dead-1hr                     (:get-fuel-moisture-dead-1hr inputs)
        get-fuel-moisture-dead-10hr                    (:get-fuel-moisture-dead-10hr inputs)
        get-fuel-moisture-dead-100hr                   (:get-fuel-moisture-dead-100hr inputs)
        get-fuel-moisture-live-herbaceous              (:get-fuel-moisture-live-herbaceous inputs)
        get-fuel-moisture-live-woody                   (:get-fuel-moisture-live-woody inputs)
        get-foliar-moisture                            (:get-foliar-moisture inputs)
```

```
fuel-number->spread-rate-adjustment-array-lookup (:fuel-number->spread-rate-adjustment-array-lookup inputs)
crowning-disabled?                               (:crowning-disabled? inputs)
ellipse-adjustment-factor                        (:ellipse-adjustment-factor inputs)
grass-suppression?                               (:grass-suppression? inputs)
max-spread-rate-matrix                           (:max-spread-rate-matrix matrices)
max-spread-direction-matrix                      (:max-spread-direction-matrix matrices)
spread-rate-matrix                               (:spread-rate-matrix matrices)
flame-length-matrix                              (:flame-length-matrix matrices)
fire-line-intensity-matrix                       (:fire-line-intensity-matrix matrices)
fire-type-matrix                                 (:fire-type-matrix matrices)
modified-time-matrix                             (:modified-time-matrix matrices)
eccentricity-matrix                              (:eccentricity-matrix matrices)
residence-time-matrix                            (:residence-time-matrix matrices)
reaction-intensity-matrix                        (:reaction-intensity-matrix matrices)
i                                                (long i)
j                                                (long j)
band                                             (long band)
slope                                            (grid-lookup/double-at get-slope i j)
aspect                                           (grid-lookup/double-at get-aspect i j)
canopy-cover                                     (grid-lookup/double-at get-canopy-cover i j)
canopy-height                                    (grid-lookup/double-at get-canopy-height i j)
canopy-base-height                               (grid-lookup/double-at get-canopy-base-height i j)
crown-bulk-density                               (grid-lookup/double-at get-crown-bulk-density i j)
fuel-model                                       (grid-lookup/double-at get-fuel-model i j)
temperature                                      (grid-lookup/double-at get-temperature band i j)
relative-humidity                                (grid-lookup/double-at get-relative-humidity band i j)
wind-speed-20ft                                  (grid-lookup/double-at get-wind-speed-20ft band i j)
wind-from-direction                              (grid-lookup/double-at get-wind-from-direction band i j)
fuel-moisture-dead-1hr                           (if get-fuel-moisture-dead-1hr
                                                   (grid-lookup/double-at get-fuel-moisture-dead-1hr band i j)
                                                   (calc-fuel-moisture relative-humidity temperature :dead :1hr))
fuel-moisture-dead-10hr                          (if get-fuel-moisture-dead-10hr
                                                   (grid-lookup/double-at get-fuel-moisture-dead-10hr band i j)
                                                   (calc-fuel-moisture relative-humidity temperature :dead :10hr))
fuel-moisture-dead-100hr                         (if get-fuel-moisture-dead-100hr
                                                   (grid-lookup/double-at get-fuel-moisture-dead-100hr band i j)
                                                   (calc-fuel-moisture relative-humidity temperature :dead :100hr))
fuel-moisture-live-herbaceous                    (if get-fuel-moisture-live-herbaceous
                                                   (grid-lookup/double-at get-fuel-moisture-live-herbaceous band i j)
                                                   (calc-fuel-moisture relative-humidity temperature :live :herbaceous))
fuel-moisture-live-woody                         (if get-fuel-moisture-live-woody
                                                   (grid-lookup/double-at get-fuel-moisture-live-woody band i j)
                                                   (calc-fuel-moisture relative-humidity temperature :live :woody))
foliar-moisture                                  (grid-lookup/double-at get-foliar-moisture band i j)
surface-fire-min                                 (rothermel-fast-wrapper-optimal (rfwo-struct/make-RothFWOArgs fuel-model
                                                                                                               fuel-moistur
                                                                                                               fuel-moistur
                                                                                                               fuel-moistur
                                                                                                               0.0 ; fuel-m
                                                                                                               fuel-moistur
                                                                                                               fuel-moistur
                                                                                                               (boolean gra
midflame-wind-speed                              (mph->fpm
                                                  (* wind-speed-20ft
                                                     (wind-adjustment-factor ^double (:fuel-bed-depth surface-fire-min)
                                                                             canopy-height
                                                                             canopy-cover)))
spread-rate-adjustment                           (some-> fuel-number->spread-rate-adjustment-array-lookup
                                                         (lookup-spread-rate-adjustment fuel-model))
surface-fire-max                                 (rothermel-surface-fire-spread-max surface-fire-min
                                                                                    midflame-wind-speed
                                                                                    wind-from-direction
                                                                                    slope
                                                                                    aspect
                                                                                    ellipse-adjustment-factor
                                                                                    spread-rate-adjustment)
max-spread-rate                                  (:max-spread-rate surface-fire-max)
max-spread-direction                             (:max-spread-direction surface-fire-max) ; IMPROVEMENT primitive lookup, f
```

```
        eccentricity                                    (:eccentricity surface-fire-max)
        residence-time                                  (:residence-time surface-fire-min)
        reaction-intensity                              (:reaction-intensity surface-fire-min)
        burn-time-getter                                (grid-lookup/mgetter-double (:burn-time-matrix matrices))
        fln-incidence-cos                               (estimate-fireline-incidence-cosine burn-time-getter
                                                                                            incoming-burnvec
                                                                                            (double max-spread-direction)
                                                                                            default-incidence-cosine)
        fln-spread-rate-scalar                          (ellip/fireline-normal-spread-rate-scalar (double eccentricity) fln-incide
        fireline-normal-spread-rate                     (* (double max-spread-rate) fln-spread-rate-scalar)
        surface-intensity                               (->> (anderson-flame-depth fireline-normal-spread-rate ^double residence-t
                                                             (byram-fire-line-intensity ^double reaction-intensity))]
    (if (and (not crowning-disabled?)
             (van-wagner-crown-fire-initiation? canopy-cover
                                                canopy-base-height
                                                foliar-moisture
                                                surface-intensity))
      (let [crown-spread-max        (cruz-crown-fire-spread wind-speed-20ft
                                                            crown-bulk-density
                                                            fuel-moisture-dead-1hr)
            crown-type              (if (neg? crown-spread-max) 2.0 3.0) ; 2=passive, 3=active
            crown-spread-max        (Math/abs crown-spread-max)
            crown-eccentricity      (if (> (double max-spread-rate) crown-spread-max)
                                      (double eccentricity)
                                      (crown-fire-eccentricity wind-speed-20ft ellipse-adjustment-factor))
            crown-fln-spread-rate   (* crown-spread-max
                                       (ellip/fireline-normal-spread-rate-scalar crown-eccentricity fln-incidence-cos))
            crown-intensity         (crown-fire-line-intensity crown-fln-spread-rate
                                                               crown-bulk-density
                                                               (- canopy-height canopy-base-height)
                                                               (:heat-of-combustion surface-fire-min))
            tot-fire-line-intensity (+ surface-intensity crown-intensity)
            max-spread-rate         (max ^double max-spread-rate crown-spread-max)]
        (t/mset! max-spread-rate-matrix i j max-spread-rate)
        (t/mset! max-spread-direction-matrix i j max-spread-direction)
        (t/mset! eccentricity-matrix i j crown-eccentricity)
        (t/mset! modified-time-matrix i j (inc band))
        (when compute-directional-values?
          (t/mset! residence-time-matrix i j residence-time)
          (t/mset! reaction-intensity-matrix i j reaction-intensity))
        (store-if-max! spread-rate-matrix i j crown-fln-spread-rate)
        (store-if-max! flame-length-matrix i j (byram-flame-length tot-fire-line-intensity))
        (store-if-max! fire-line-intensity-matrix i j tot-fire-line-intensity)
        (store-if-max! fire-type-matrix i j crown-type))
      (do
        (t/mset! max-spread-rate-matrix i j max-spread-rate)
        (t/mset! max-spread-direction-matrix i j max-spread-direction)
        (t/mset! eccentricity-matrix i j eccentricity)
        (t/mset! modified-time-matrix i j (inc band))
        (when compute-directional-values?
          (t/mset! residence-time-matrix i j residence-time)
          (t/mset! reaction-intensity-matrix i j reaction-intensity))
        (store-if-max! spread-rate-matrix i j fireline-normal-spread-rate)
        (store-if-max! flame-length-matrix i j (byram-flame-length surface-intensity))
        (store-if-max! fire-line-intensity-matrix i j surface-intensity)
        (store-if-max! fire-type-matrix i j 1.0)))))

(defn- burnable-neighbors?
  [get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i j]
  (let [i  (long i)
        j  (long j)
        i- (- i 1)
        i+ (+ i 1)
        j- (- j 1)
        j+ (+ j 1)]
    (or (burnable-cell? get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i- j-)
        (burnable-cell? get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i- j)
        (burnable-cell? get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i- j+)
```

```
        (burnable-cell? get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i  j-)
        (burnable-cell? get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i  j+)
        (burnable-cell? get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i+ j-)
        (burnable-cell? get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i+ j)
        (burnable-cell? get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i+ j+))))

(defmacro dir-bits-reduce
  "Macro for efficiently reducing over the 8 direction bits, similarly to clojure.core/areduce.

  `dir-bit-sym` will be bound to a primitive long,
  and `ret-sym` to the reduction accumulator,
  which will be initialized by evaluating `init`
  and updated by evaluating `expr`."
  [[dir-bit-sym ret-sym] init expr]
  {:pre [(symbol? dir-bit-sym)
         (symbol? ret-sym)]}
  `(loop [~dir-bit-sym (long 0)
          ~ret-sym     ~init]
     (if (= ~dir-bit-sym 8)
       ~ret-sym
       (recur (unchecked-inc ~dir-bit-sym)
              ~expr))))

;; NAMING CONVENTION the '-pfn' suffix stands for 'Prepared FuNction' or 'Partial'd FuNction'.
(defn create-new-burn-vectors!-pfn
  "Prepares a partialed primitive-signature function to be called via `create-new-burn-vectors!-invoke`."
  [num-rows num-cols cell-size get-elevation
   travel-lines-matrix fire-spread-matrix
   max-spread-rate-matrix max-spread-direction-matrix eccentricity-matrix]
  (let [num-rows                   (long num-rows)
        num-cols                   (long num-cols)
        max-spread-rate-getter     (grid-lookup/mgetter-double max-spread-rate-matrix)
        max-spread-direction-getter (grid-lookup/mgetter-double max-spread-direction-matrix)
        eccentricity-getter        (grid-lookup/mgetter-double eccentricity-matrix)
        fire-spread-getter         (grid-lookup/mgetter-double fire-spread-matrix)]
    (fn launch-from! [bvs-acc ^long i ^long j ^double burn-probability]
      (let [max-spread-rate      (grid-lookup/double-at max-spread-rate-getter i j)
            max-spread-direction (grid-lookup/double-at max-spread-direction-getter i j)
            eccentricity         (grid-lookup/double-at eccentricity-getter i j)
            travel-lines         (long (t/mget travel-lines-matrix i j)) ;; IMPROVEMENT mget-long-at, or something like it.
            get-ij-terrain-dist  (terrain-distance-from-cell-getter get-elevation num-rows num-cols cell-size i j)]
        (dir-bits-reduce [dir-bit bvs-acc]
          bvs-acc
          (if (bit-test travel-lines dir-bit)
            ;; INVARIANT: at most 1 Burn Vector at a time per travel line.
            bvs-acc
            (let [direction       (direction-bit->angle dir-bit)
                  new-i           (+ i (direction-angle->i-incr direction))
                  new-j           (+ j (direction-angle->j-incr direction))
                  new-burn-vector (when (and (in-bounds-optimal? num-rows num-cols new-i new-j)
                                             (overtakes-lower-probability-fire? burn-probability
                                                                                (grid-lookup/double-at fire-spread-getter new-i new
                                    (let [spread-rate      (compute-spread-rate max-spread-rate
                                                                                max-spread-direction
                                                                                eccentricity
                                                                                direction)
                                          terrain-distance (grid-lookup/double-at get-ij-terrain-dist new-i new-j)]
                                      ;; NOTE we start at fractional-distance = 0.5, which represents the center of the cell. (Val,
                                      (burn-vec/make-burn-vector i j direction 0.5 spread-rate terrain-distance burn-probability)))
              (if new-burn-vector
                (do
                  ;; TODO move into function
                  (as-> (t/mget travel-lines-matrix i j) $
                        (bit-set $ dir-bit)
                        (t/mset! travel-lines-matrix i j $))
                  (conj! bvs-acc new-burn-vector))
                bvs-acc))))))))
```

```
(defmacro create-new-burn-vectors!-invoke
  "Adds the relevant new burn vectors to `bvs-acc` starting from cell `[i j]` with the given `burn-probability`.

  `create-new-burn-vectors!` must be the function prepared by `#'create-new-burn-vectors!-pfn`;
  the point of this macro is to wrap the Java interop which invokes it efficiently,
  hiding it from callers."
  [create-new-burn-vectors! bvs-acc i j burn-probability]
  (let [cnbv-sym (-> (gensym 'cvbv)
                     (vary-meta assoc :tag `IFn$OLLDO))]
    `(let [~cnbv-sym ~create-new-burn-vectors!]
       (.invokePrim ~cnbv-sym ~bvs-acc ~i ~j ~burn-probability))))

(defn- identify-spot-ignition-events
  [new-clock spot-ignitions]
  (loop [to-process   spot-ignitions
         ignite-later (transient {})
         ignite-now   (transient {})]
    (if (seq to-process)
      (let [[cell spot-info] (first to-process)
            [t _]            spot-info]
        (if (>= ^double new-clock ^double t)
          (recur (rest to-process) ignite-later (assoc! ignite-now cell spot-info))
          (recur (rest to-process) (assoc! ignite-later cell spot-info) ignite-now)))
      [(persistent! ignite-later) (persistent! ignite-now)])))

(defn- merge-spot-ignitions [a b]
  (reduce (fn [acc [cell spot-info]]
            (if-let [existing-entry (acc cell)]
              (let [[cur-t cur-p] existing-entry
                    [new-t new-p] spot-info]
                (cond (> ^double cur-p ^double new-p)  acc
                      (< ^double cur-p ^double new-p)  (assoc! acc cell spot-info)
                      (<= ^double cur-t ^double new-t) acc
                      :else                            (assoc! acc cell spot-info)))
              (assoc! acc cell spot-info)))
          a
          b))

(defn- compute-new-spot-ignitions
  "Returns a map of [x y] locations to [t p] where:
  t: time of ignition
  p: ignition-probability"
  [inputs matrices ignited-cells]
  (when (:spotting inputs)
    (persistent!
     (reduce (fn [acc [i j]]
               (merge-spot-ignitions acc (spotting/spread-firebrands inputs matrices i j)))
             (transient {})
             ignited-cells))))

(defn- compute-spot-burn-vectors!
  [inputs matrices spot-ignitions ignited-cells band new-clock]
  (let [num-rows                   (:num-rows inputs)
        num-cols                   (:num-cols inputs)
        cell-size                  (:cell-size inputs)
        get-elevation              (:get-elevation inputs)
        fire-spread-matrix         (:fire-spread-matrix matrices)
        fire-spread-getter         (grid-lookup/mgetter-double fire-spread-matrix)
        burn-time-matrix           (:burn-time-matrix matrices)
        travel-lines-matrix        (:travel-lines-matrix matrices)
        max-spread-rate-matrix     (:max-spread-rate-matrix matrices)
        max-spread-direction-matrix (:max-spread-direction-matrix matrices)
        eccentricity-matrix        (:eccentricity-matrix matrices)
        modified-time-getter       (grid-lookup/mgetter-double (:modified-time-matrix matrices))
        band                       (long band)
        new-clock                  (double new-clock)
        new-spot-ignitions         (compute-new-spot-ignitions inputs matrices ignited-cells)
        [spot-ignite-later
```

```
         spot-ignite-now]           (identify-spot-ignition-events new-clock (persistent!
                                                                              (merge-spot-ignitions
                                                                               (transient spot-ignitions)
                                                                               new-spot-ignitions)))
        ignited?                    (fn [[k v]]
                                      (let [[i j] k
                                            [_ p] v]
                                        (>= (grid-lookup/double-at fire-spread-getter (long i) (long j)) (double p))))
        pruned-spot-ignite-later    (into {} (remove ignited? spot-ignite-later)) ; TODO move to identify-spot-ignition
        pruned-spot-ignite-now      (filterv #(not (ignited? %)) spot-ignite-now) ; TODO move to identify-spot-ignition
        create-new-bvs!             (create-new-burn-vectors!-pfn num-rows num-cols cell-size get-elevation
                                                                  travel-lines-matrix fire-spread-matrix
                                                                  max-spread-rate-matrix max-spread-direction-matrix eccentricity-m
        spot-burn-vectors           (persistent!
                                     (reduce
                                      (fn [bvs-acc [cell spot-info]]
                                        (let [[i j]         cell
                                              [burn-time _] spot-info]
                                          (when (> band (dec (long
                                                              ;; FIXME check if double-typed
                                                              (grid-lookup/double-at modified-time-getter i j))))
                                            (compute-max-in-situ-values! inputs matrices band i j nil))
                                          (t/mset! fire-spread-matrix i j 1.0) ;TODO parameterize burn-probability instead of 1.0
                                          ;; (t/mset! fire-spread-matrix i j burn-probability)
                                          (t/mset! burn-time-matrix i j burn-time)
                                          (create-new-burn-vectors!-invoke create-new-bvs! bvs-acc i j 1.0))) ; TODO parameterize b
                                      (transient [])
                                      pruned-spot-ignite-now))]
    [spot-burn-vectors (count pruned-spot-ignite-now) pruned-spot-ignite-later (keys pruned-spot-ignite-now)]))

(defn- ignited-in-this-timestep?
  [^double global-clock-before-timestep ^double burn-time-matrix-ij]
  (< global-clock-before-timestep burn-time-matrix-ij))

(defn- grow-burn-vectors!
  "Updates the progress of the Burn Vectors along their travel lines, detecting which cells get ignited.

  Returns a [new-burn-vectors ignited-cells] tuples,
  and mutates :burn-time-matrix and :fire-spread-matrix."
  [matrices global-clock timestep burn-vectors]
  (let [fire-spread-matrix (:fire-spread-matrix matrices)
        fire-spread-getter (grid-lookup/mgetter-double fire-spread-matrix)
        burn-time-matrix   (:burn-time-matrix matrices)
        burn-time-getter   (grid-lookup/mgetter-double burn-time-matrix)
        global-clock       (double global-clock)
        timestep           (double timestep)
        ignited-cells-list (ArrayList.)]
    [(persistent!
      (reduce (fn [bvs-acc burn-vector]
                (let [i                        (burn-vec/get-i burn-vector)
                      j                        (burn-vec/get-j burn-vector)
                      direction                (burn-vec/get-direction burn-vector)
                      fractional-distance      (burn-vec/get-fractional-distance burn-vector)
                      spread-rate              (burn-vec/get-spread-rate burn-vector)
                      terrain-distance         (burn-vec/get-terrain-distance burn-vector)
                      burn-probability         (burn-vec/get-burn-probability burn-vector)
                      fractional-distance-delta (/ (* spread-rate timestep) terrain-distance)
                      new-fractional-distance  (+ fractional-distance fractional-distance-delta)
                      crossed-center?          (and (< fractional-distance 0.5)
                                                    (>= new-fractional-distance 0.5))
                      local-burn-time          (grid-lookup/double-at burn-time-getter i j)]
                  (when crossed-center?
                    ;; INVARIANT a cell is ignited when a burn vector traverses its center,
                    ;; which corresponds to a :fractional-distance of 0.5.
                    (let [local-burn-probability (grid-lookup/double-at fire-spread-getter i j)]
                      (when (>= burn-probability local-burn-probability)
                        (let [burn-time (-> 0.5
                                            (- fractional-distance)
```

```
                                        (/ fractional-distance-delta)
                                        (* timestep)
                                        (+ global-clock))]
                          (if (> burn-probability local-burn-probability)
                            ;; INVARIANT :burn-time-matrix and :fire-spread-matrix record the fire of highest :burn-probability.
                            (do
                              (t/mset! fire-spread-matrix i j burn-probability)
                              (t/mset! burn-time-matrix i j burn-time))
                            (when (< burn-time local-burn-time)
                              (comment (assert (= burn-probability local-burn-probability)))
                              ;; Invariant: burn-time-matrix records the time of the earliest event causing the cell to burn.
                              (t/mset! burn-time-matrix i j burn-time)))))))
                    (when (and crossed-center? (not (ignited-in-this-timestep? global-clock local-burn-time))) ; first to cross cente
                      (.add ignited-cells-list [i j]))
                    (conj! bvs-acc (burn-vec/make-burn-vector i j direction new-fractional-distance
                                                             spread-rate terrain-distance burn-probability))))
                (transient [])
                ;;      (if crossed-center? ; first to cross center of the cell this timestep
                ;;        (conj! ignited-cells [i j])
                ;;        ignited-cells)]))
                ;; [(transient []) (transient #{})]
                burn-vectors))
        (vec ignited-cells-list)]))

(defn- ignited-cells->burn-vectors
  "Adds new Burn Vectors from the list of newly ignited cells.

  Returns a bigger burn-vectors vector,
  and mutates :travel-lines-matrix.

  IMPORTANT: this function must be followed by #'promote-burn-vectors."
  [inputs matrices ignited-cells burn-vectors]
  (let [num-rows                    (:num-rows inputs)
        num-cols                    (:num-cols inputs)
        cell-size                   (:cell-size inputs)
        get-elevation               (:get-elevation inputs)
        fire-spread-matrix          (:fire-spread-matrix matrices)
        fire-spread-getter          (grid-lookup/mgetter-double fire-spread-matrix)
        travel-lines-matrix         (:travel-lines-matrix matrices)
        max-spread-rate-matrix      (:max-spread-rate-matrix matrices)
        max-spread-direction-matrix (:max-spread-direction-matrix matrices)
        eccentricity-matrix         (:eccentricity-matrix matrices)
        create-new-bvs!             (create-new-burn-vectors!-pfn num-rows num-cols cell-size get-elevation
                                                                  travel-lines-matrix fire-spread-matrix
                                                                  max-spread-rate-matrix max-spread-direction-matrix eccentricity-m
    (persistent!
     (reduce
      (fn [bvs-acc [i j]]
        (let [i                (long i)
              j                (long j)
              burn-probability (grid-lookup/double-at fire-spread-getter i j)]
          (create-new-burn-vectors!-invoke create-new-bvs! bvs-acc i j burn-probability)))
      (transient burn-vectors)
      ignited-cells))))

(defn- update-directional-magnitude-values!
  [matrices direction spread-rate i j]
  (let [x-magnitude-sum-matrix (:x-magnitude-sum-matrix matrices)
        y-magnitude-sum-matrix (:y-magnitude-sum-matrix matrices)
        spread-rate-sum-matrix (:spread-rate-sum-matrix matrices)
        direction              (double direction)
        spread-rate            (double spread-rate)
        cur-x                  (grid-lookup/mget-double-at x-magnitude-sum-matrix i j)
        cur-y                  (grid-lookup/mget-double-at y-magnitude-sum-matrix i j)
        cur-spread-rate        (grid-lookup/mget-double-at spread-rate-sum-matrix i j)]
    (t/mset! x-magnitude-sum-matrix i j (+ cur-x
                                           (-> direction
                                               (Math/toRadians)
```

```
                                          (Math/cos)
                                          (* spread-rate))))
    (t/mset! y-magnitude-sum-matrix i j  (+ cur-y
                                        (-> direction
                                            (Math/toRadians)
                                            (Math/sin)
                                            (* spread-rate))))
    (t/mset! spread-rate-sum-matrix i j (+ cur-spread-rate spread-rate))))

(defn- bv-can-spread-fire-to-target-cell?
  "Whether this burn vector still has potential to ignite its target cell."
  [get-fuel-model fire-spread-matrix num-rows num-cols
   i j direction burn-probability]
  ;; HACK relying on an implementation detail of Clojure for performance; we could also make a custom Java interface.
  (let [i         (long i)
        j         (long j)
        direction (double direction)
        new-i     (+ i (direction-angle->i-incr direction))
        new-j     (+ j (direction-angle->j-incr direction))]
    ;; IMPROVEMENT currify burnable-cell? for performance. (Val, 16 Nov 2022)
    (and (burnable-cell? get-fuel-model fire-spread-matrix burn-probability
                         num-rows num-cols new-i new-j)
         (not (and (diagonal? direction)
                   ;; NOTE we use a named local for clarity; we avoid using a function for performance. (Val, 22 Nov 2022)
                   (let [crossing-diagonal-barrier? (and (not (burnable-fuel-model? (grid-lookup/double-at get-fuel-model new-i j))
                                                         (not (burnable-fuel-model? (grid-lookup/double-at get-fuel-model i new-j))
                     crossing-diagonal-barrier?))))))

(defn- promote-burn-vectors
  "Replaces or removes the burn vector which have become obsolete because of an ignition.

  In particular, if the cell just got ignited at :burn-probability p,
  the burn vectors with :burn-probability < p get replaced by a Burn Vector
  starting from the center of the cell."
  [inputs matrices global-clock new-clock max-fractional-distance burn-vectors]
  (let [num-rows                   (:num-rows inputs)
        num-cols                   (:num-cols inputs)
        get-fuel-model             (:get-fuel-model inputs)
        compute-directional-values? (:compute-directional-values? inputs)
        fire-spread-matrix         (:fire-spread-matrix matrices)
        fire-spread-getter         (grid-lookup/mgetter-double fire-spread-matrix)
        burn-time-matrix           (:burn-time-matrix matrices)
        burn-time-getter           (grid-lookup/mgetter-double burn-time-matrix)
        travel-lines-matrix        (:travel-lines-matrix matrices)
        global-clock               (double global-clock)
        new-clock                  (double new-clock)
        max-fractional-distance    (double max-fractional-distance)]
    (persistent!
     (reduce
      (fn [bvs-acc burn-vector]
        (let [i                     (burn-vec/get-i burn-vector)
              j                     (burn-vec/get-j burn-vector)
              burn-probability      (burn-vec/get-burn-probability burn-vector)
              local-burn-probability (grid-lookup/double-at fire-spread-getter i j)
              local-burn-time       (grid-lookup/double-at burn-time-getter i j)]
          (if-not (and (ignited-in-this-timestep? global-clock local-burn-time)
                       ;; A burn vector of higher :burn-probability than the ignition remains unaffected by that ignition.
                       ;; WARNING: when (> 0.5 (:fractional-distance burn-vector)),
                       ;; this causes the low-probability spread in that direction to be delayed until the higher-probability
                       ;; burn vector reaches the center. This is something of a modeling inconsistency.
                       (not (overtakes-lower-probability-fire? burn-probability local-burn-probability)))
            (conj! bvs-acc burn-vector) ; the burn vector remains as is.
            (let [direction   (burn-vec/get-direction burn-vector)
                  spread-rate (burn-vec/get-spread-rate burn-vector)]
              (when compute-directional-values?
                (update-directional-magnitude-values! matrices direction spread-rate i j))
              (if (bv-can-spread-fire-to-target-cell? get-fuel-model fire-spread-matrix num-rows num-cols
                                                      i j direction burn-probability)
```

```
                (let [spread-rate             (burn-vec/get-spread-rate burn-vector)
                      terrain-distance        (burn-vec/get-terrain-distance burn-vector)
                      dt-after-ignition       (- new-clock local-burn-time)
                      new-fractional-distance (min max-fractional-distance
                                                   (+ 0.5 (/ (* spread-rate dt-after-ignition) terrain-distance)))
                      new-spread-rate         (if (< new-fractional-distance max-fractional-distance)
                                                spread-rate
                                                ;; Re-computing an empirical spread rate,
                                                ;; because we have artificially reduced the spread rate of the burn vector
                                                ;; using max-fractional-distance.
                                                ;; NOTE that we'll have (<= new-spread-rate spread-rate).
                                                (/ (* (- new-fractional-distance 0.5) terrain-distance) dt-after-ignition))
                      ;; The new BV's :burn-probability is inherited from the ignition.
                      new-burn-probability    local-burn-probability]
                  (conj! bvs-acc
                         ;; This burn vector got its progress updated.
                         (burn-vec/make-burn-vector i j direction new-fractional-distance new-spread-rate terrain-distance new-burn
                (do
                  ;; This travel line gets cleared of its Burn Vector because there is no longer a target cell to burn.
                  (t/mset! travel-lines-matrix i j ; TODO make into function
                           (bit-clear (t/mget travel-lines-matrix i j)
                                      (direction-angle->bit direction)))
                  bvs-acc))))))
      (transient [])
      burn-vectors))))

#_(defn- fd->dt
    [fractional-distance terrain-distance spread-rate]
    (-> fractional-distance
        (* terrain-distance)
        (/ spread-rate)))

(defn- transition-burn-vectors
  "Detects when burn vector cross from one cell to another and updates accordingly.

  In addition to updating matrices (side-effect!),
  returns a [new-burn-vectors newly-ignited-cells] tuple,
  in which newly-ignited-cells is an [[i j] ...] sequence."
  [inputs matrices band global-clock new-clock max-fractional-distance burn-vectors]
  ;; IMPROVEMENT for performance (fewer object creations and flatter memory layout), we might want to return
  ;; the newly-ignited-cells not as [i j] tuples, but as BurnVectors or something like it.
  (let [cell-size                   (:cell-size inputs)
        get-elevation               (:get-elevation inputs)
        num-rows                    (:num-rows inputs)
        num-cols                    (:num-cols inputs)
        get-fuel-model              (:get-fuel-model inputs)
        travel-lines-matrix         (:travel-lines-matrix matrices)
        fire-spread-matrix          (:fire-spread-matrix matrices)
        fire-spread-getter          (grid-lookup/mgetter-double fire-spread-matrix)
        modified-time-matrix        (:modified-time-matrix matrices)
        modified-time-getter        (grid-lookup/mgetter-double modified-time-matrix)
        max-spread-rate-matrix      (:max-spread-rate-matrix matrices)
        max-spread-rate-getter      (grid-lookup/mgetter-double max-spread-rate-matrix)
        max-spread-direction-matrix (:max-spread-direction-matrix matrices)
        max-spread-direction-getter (grid-lookup/mgetter-double max-spread-direction-matrix)
        eccentricity-matrix         (:eccentricity-matrix matrices)
        eccentricity-getter         (grid-lookup/mgetter-double eccentricity-matrix)
        burn-time-matrix            (:burn-time-matrix matrices)
        burn-time-getter            (grid-lookup/mgetter-double burn-time-matrix)
        band                        (long band)
        global-clock                (double global-clock)
        new-clock                   (double new-clock)
        max-fractional-distance     (double max-fractional-distance)
        ignited-cells-list          (ArrayList.)
        terrain-dist-fn             (terrain-distance-fn get-elevation num-rows num-cols cell-size)]
    [(persistent!
      (reduce
       (fn [new-bvs-acc burn-vector]
```

```
(let [fractional-distance        (burn-vec/get-fractional-distance burn-vector)
      hasnt-entered-target-cell? (< fractional-distance 1.0)]
  (if hasnt-entered-target-cell?
    (conj! new-bvs-acc burn-vector)
    ;; The burn vector has entered the target cell.
    (let [i                (burn-vec/get-i burn-vector)
          j                (burn-vec/get-j burn-vector)
          direction        (burn-vec/get-direction burn-vector)
          burn-probability (burn-vec/get-burn-probability burn-vector)
          direction-bit    (direction-angle->bit direction)]
      ;; TODO make into function
      ;; Clearing the corresponding travel line in the origin cell. (Val, 16 Nov 2022)
      (as-> (t/mget travel-lines-matrix i j) $
            (bit-clear $ direction-bit)
            (t/mset! travel-lines-matrix i j $))
      (let [new-i (+ i (direction-angle->i-incr direction))
            new-j (+ j (direction-angle->j-incr direction))]
        (if-not (bv-can-spread-fire-to-target-cell? get-fuel-model fire-spread-matrix num-rows num-cols
                                                    i j direction burn-probability)
          ;; Dropping this burn vector - more exactly, not creating a new BV in the target cell.
          ;; Note that we cleared the original BV's travel line above.
          new-bvs-acc
          (do
            (when (> band (dec (long (grid-lookup/double-at modified-time-getter new-i new-j))))
              ;; vector is first in this timestep to compute
              ;; NOTE using the old burn-vector here is helpful:
              ;; it increases the chance that we'll use the cell of origin
              ;; for ToA gradient estimation.
              (compute-max-in-situ-values! inputs matrices band new-i new-j burn-vector))
            ;; TODO move to function
            (as-> (t/mget travel-lines-matrix new-i new-j) $
                  (bit-set $ direction-bit)
                  (t/mset! travel-lines-matrix new-i new-j $))
            (let [spread-rate             (burn-vec/get-spread-rate burn-vector)
                  terrain-distance        (burn-vec/get-terrain-distance burn-vector)
                  max-spread-rate         (grid-lookup/double-at max-spread-rate-getter new-i new-j)
                  max-spread-direction    (grid-lookup/double-at max-spread-direction-getter new-i new-j)
                  eccentricity            (grid-lookup/double-at eccentricity-getter new-i new-j)
                  new-spread-rate         (compute-spread-rate max-spread-rate
                                                               max-spread-direction
                                                               eccentricity
                                                               direction)
                  new-terrain-distance    (terrain-distance-invoke terrain-dist-fn
                                                                   new-i
                                                                   new-j
                                                                   (+ new-i (direction-angle->i-incr direction))
                                                                   (+ new-j (direction-angle->j-incr direction)))
                  ;; time since entering new cell
                  dt-in-neighbor          (-> fractional-distance (- 1.0)
                                              (* terrain-distance) ; the length spent in the new cell
                                              (/ spread-rate))
                  new-fractional-distance (min max-fractional-distance
                                               (/ (* new-spread-rate dt-in-neighbor) new-terrain-distance))
                  new-spread-rate         (if (< new-fractional-distance max-fractional-distance)
                                            new-spread-rate
                                            (/ (* new-fractional-distance new-terrain-distance) dt-in-neighbor))
                  new-bvs-acc+1           (conj! new-bvs-acc
                                                 ;; This new Burn Vector is the result of moving ('transitioning') the old
                                                 (burn-vec/make-burn-vector new-i new-j direction new-fractional-distance
                                                                            new-terrain-distance burn-probability))
                  bv-has-reached-center?  (>= new-fractional-distance 0.5)]
              (when bv-has-reached-center?
                (let [local-burn-probability     (grid-lookup/double-at fire-spread-getter new-i new-j)
                      failed-to-ignite-new-cell? (< burn-probability local-burn-probability)]
                  (when-not failed-to-ignite-new-cell?
                    (let [relative-burn-time (-> (/ dt-in-neighbor new-fractional-distance)
                                                 (* 0.5))
                          burn-time          (-> new-clock
```

```
                                                    (- dt-in-neighbor)
                                                    (+ relative-burn-time))
                                  recorded-burn-time (grid-lookup/double-at burn-time-getter new-i new-j)]
                              (if (overtakes-lower-probability-fire? burn-probability local-burn-probability)
                                (do
                                  (t/mset! fire-spread-matrix new-i new-j burn-probability) ;; logically a max update
                                  (t/mset! burn-time-matrix new-i new-j burn-time)
                                  ;; For performance, avoiding redundant additions to ignited-cells.
                                  (when-not (ignited-in-this-timestep? global-clock recorded-burn-time)
                                    (.add ignited-cells-list [new-i new-j])))
                                ;; This conditional branch corresponds to: (= burn-probability local-burn-probability)
                                ;; this is why we don't add to ignited cells (it's already ignited).
                                (when (< burn-time recorded-burn-time)
                                  ;; We have just realized that the cell actually ignited earlier than recorded until now,
                                  ;; so we correct the recorded-burn-time.
                                  ;; Invariant: burn-time-matrix holds the earliest time a burn vector ignited it.
                                  (t/mset! burn-time-matrix new-i new-j burn-time)))))))
                      new-bvs-acc+1))))))))
      (transient [])
      burn-vectors))
    (vec ignited-cells-list)]))

(defn- parse-burn-period
  "Return the number of minutes into the day given HH:MM"
  ^double
  [burn-period]
  (let [[hour minute] (mapv #(Integer/parseInt %) (s/split burn-period #":"))]
    (+ (hour->min hour) ^double minute)))

(defn- recompute-burn-vectors
  [inputs matrices ^long band burn-vectors]
  (let [modified-time-matrix        (:modified-time-matrix matrices)
        modified-time-getter        (grid-lookup/mgetter-double modified-time-matrix)
        max-spread-rate-matrix      (:max-spread-rate-matrix matrices)
        max-spread-rate-getter      (grid-lookup/mgetter-double max-spread-rate-matrix)
        max-spread-direction-matrix (:max-spread-direction-matrix matrices)
        max-spread-direction-getter (grid-lookup/mgetter-double max-spread-direction-matrix)
        eccentricity-matrix         (:eccentricity-matrix matrices)
        eccentricity-getter         (grid-lookup/mgetter-double eccentricity-matrix)]
    (mapv (fn [burn-vector]
            (let [i (burn-vec/get-i burn-vector)
                  j (burn-vec/get-j burn-vector)]
              (when (> band (dec (long (grid-lookup/double-at modified-time-getter i j))))
                (compute-max-in-situ-values! inputs matrices band i j burn-vector))
              (let [direction            (burn-vec/get-direction burn-vector)
                    max-spread-rate      (grid-lookup/double-at max-spread-rate-getter i j)
                    max-spread-direction (grid-lookup/double-at max-spread-direction-getter i j)
                    eccentricity         (grid-lookup/double-at eccentricity-getter i j)
                    new-spread-rate      (compute-spread-rate max-spread-rate
                                                              max-spread-direction
                                                              eccentricity
                                                              direction)]
                (assoc burn-vector :spread-rate new-spread-rate))))
          burn-vectors)))

(defn- compute-fire-front-direction!
  [matrices ^long i ^long j]
  (let [x-magnitude-sum-matrix (:x-magnitude-sum-matrix matrices)
        y-magnitude-sum-matrix (:y-magnitude-sum-matrix matrices)
        spread-rate-sum-matrix (:spread-rate-sum-matrix matrices)
        x-magnitude-sum        (grid-lookup/mget-double-at x-magnitude-sum-matrix i j)
        y-magnitude-sum        (grid-lookup/mget-double-at y-magnitude-sum-matrix i j)
        spread-rate-sum        (grid-lookup/mget-double-at spread-rate-sum-matrix i j)]
    (t/mset! x-magnitude-sum-matrix i j 0.0)
    (t/mset! y-magnitude-sum-matrix i j 0.0)
    (t/mset! spread-rate-sum-matrix i j 0.0)
    (-> (Math/atan2 (/ y-magnitude-sum spread-rate-sum)
                    (/ x-magnitude-sum spread-rate-sum))
```

```
        (Math/toDegrees)
        (mod 360))))

(defn- compute-directional-in-situ-values!
  [matrices ignited-cells]
  (when (seq ignited-cells)
    (let [max-spread-rate-matrix          (:max-spread-rate-matrix matrices)
          max-spread-rate-getter          (grid-lookup/mgetter-double max-spread-rate-matrix)
          max-spread-direction-matrix     (:max-spread-direction-matrix matrices)
          max-spread-direction-getter     (grid-lookup/mgetter-double max-spread-direction-matrix)
          eccentricity-matrix             (:eccentricity-matrix matrices)
          eccentricity-getter             (grid-lookup/mgetter-double eccentricity-matrix)
          residence-time-matrix           (:residence-time-matrix matrices)
          residence-time-getter           (grid-lookup/mgetter-double residence-time-matrix)
          reaction-intensity-matrix       (:reaction-intensity-matrix matrices)
          reaction-intensity-getter       (grid-lookup/mgetter-double reaction-intensity-matrix)
          directional-flame-length-matrix (:directional-flame-length-matrix matrices)]
      (doseq [[i j] ignited-cells]
        (let [i                    (long i)
              j                    (long j)
              direction            (compute-fire-front-direction! matrices i j)
              max-spread-rate      (grid-lookup/double-at max-spread-rate-getter i j)
              max-spread-direction (grid-lookup/double-at max-spread-direction-getter i j)
              eccentricity         (grid-lookup/double-at eccentricity-getter i j)
              spread-rate          (compute-spread-rate max-spread-rate
                                                        max-spread-direction
                                                        eccentricity
                                                        direction)
              residence-time       (grid-lookup/double-at residence-time-getter i j)
              reaction-intensity   (grid-lookup/double-at reaction-intensity-getter i j)
              fire-line-intensity  (->> (anderson-flame-depth spread-rate residence-time)
                                        (byram-fire-line-intensity reaction-intensity))
              flame-length         (byram-flame-length fire-line-intensity)]
          (t/mset! directional-flame-length-matrix i j flame-length))))))

(defn- initialize-fire-in-situ-values!
  [inputs matrices band ignited-cells]
  (let [compute-directional-values?     (:compute-directional-values? inputs)
        flame-length-matrix             (:flame-length-matrix matrices)
        directional-flame-length-matrix (:directional-flame-length-matrix matrices)]
    (doseq [[i j] ignited-cells]
      (compute-max-in-situ-values! inputs matrices band i j nil)
      (when compute-directional-values?
        (t/mset! directional-flame-length-matrix i j (grid-lookup/mget-double-at flame-length-matrix i j))))))

(def ^:const ^:private minutes-per-24h 1440.0)

;; FIXME Update target spread rate on burn-vectors if new band > band
(defn- run-loop
  [inputs matrices ignited-cells]
  (let [compute-directional-values?      (:compute-directional-values? inputs)
        cell-size                        (double (:cell-size inputs))
        max-runtime                      (double (:max-runtime inputs))
        ignition-start-time              (double (:ignition-start-time inputs))
        ignition-stop-time               (+ ignition-start-time max-runtime)
        ignition-start-timestamp         (-> ^Date (:ignition-start-timestamp inputs)
                                             (.toInstant)
                                             (.atZone (ZoneId/of "UTC")))
        ignition-start-time-min-into-day (+ (hour->min (.getHour ignition-start-timestamp))
                                            (double (.getMinute ignition-start-timestamp)))
        burn-period-start                (parse-burn-period (:burn-period-start inputs))
        burn-period-end                  (let [bp-end (parse-burn-period (:burn-period-end inputs))]
                                           (if (< bp-end burn-period-start)
                                             (+ bp-end minutes-per-24h)
                                             bp-end))
        burn-period-dt                   (- burn-period-end burn-period-start)
        non-burn-period-dt               (- minutes-per-24h burn-period-dt)
        burn-period-clock                (+ ignition-start-time
```

```
                                        (double
                                         ;; This correction ensures that the ignition occurs at the start of the Burn Period.
                                         (cond
                                           ;; If too early in the day:
                                           ;; skipping ahead to the beginning of today's Burn Period
                                           (< ignition-start-time-min-into-day burn-period-start)
                                           (- burn-period-start ignition-start-time-min-into-day)

                                           ;; If too late in the day:
                                           ;; skipping ahead to the beginning of tomorrow's Burn Period
                                           (> ignition-start-time-min-into-day burn-period-end)
                                           (+ (- minutes-per-24h ignition-start-time-min-into-day) burn-period-start)

                                           :else
                                           (- burn-period-start ignition-start-time-min-into-day))))
    non-burn-period-clock            (+ burn-period-clock burn-period-dt)
    ignition-start-time              (max ignition-start-time burn-period-clock)
    band                             (min->hour ignition-start-time)
    suppression-dt                   (double (or (:suppression-dt inputs) Double/NaN))
    suppression-dt?                  (not (Double/isNaN suppression-dt))]
(initialize-fire-in-situ-values! inputs matrices band ignited-cells)
(loop [global-clock                          ignition-start-time
       band                                  band
       non-burn-period-clock                 non-burn-period-clock
       suppression-clock                     (double (if suppression-dt? (+ ignition-start-time suppression-dt) max-runtime))
       burn-vectors                          (ignited-cells->burn-vectors inputs matrices ignited-cells [])
       spot-ignitions                        {}
       spot-count                            0
       total-cells-suppressed                0
       previous-num-perimeter-cells          0
       ignited-cells-since-last-suppression  []
       fraction-contained                    0.0]
  (if (and (< global-clock ignition-stop-time)
           (or (seq burn-vectors) (seq spot-ignitions)))
    (let [dt-until-max-runtime               (- ignition-stop-time global-clock)]
      (cond
        (and suppression-dt? (= global-clock suppression-clock))
        (let [max-runtime-fraction (/ (- global-clock ignition-start-time) max-runtime)
              [bvs-to-process-next
               total-cells-suppressed
               previous-num-perimeter-cells
               fraction-contained] (suppression/suppress-burn-vectors inputs
                                                                      max-runtime-fraction
                                                                      previous-num-perimeter-cells
                                                                      total-cells-suppressed
                                                                      burn-vectors
                                                                      ignited-cells-since-last-suppression
                                                                      fraction-contained)]
          (recur global-clock
                 band
                 non-burn-period-clock
                 (+ global-clock suppression-dt)
                 bvs-to-process-next
                 spot-ignitions
                 spot-count
                 (long total-cells-suppressed)
                 (long previous-num-perimeter-cells)
                 []
                 (double fraction-contained)))

        (= global-clock non-burn-period-clock)
        (let [timestep  (double (min non-burn-period-dt dt-until-max-runtime))
              new-clock (+ global-clock timestep)
              new-band  (min->hour new-clock)]
          (if (and suppression-dt? (<= suppression-clock new-clock))
            (let [suppression-clocks     (iterate #(+ (double %) suppression-dt) suppression-clock)
                  last-suppression-clock (double (last (take-while #(<= (double %) new-clock) suppression-clocks)))
                  [bvs-to-process-next
```

```
                    total-cells-suppressed
                    previous-num-perimeter-cells
                    fraction-contained]   (suppression/suppress-burn-vectors inputs
                                                                              (/ (- last-suppression-clock ignition-start-time) m
                                                                              previous-num-perimeter-cells
                                                                              total-cells-suppressed
                                                                              burn-vectors
                                                                              ignited-cells-since-last-suppression
                                                                              fraction-contained)]
                (recur new-clock
                       new-band
                       (+ new-clock burn-period-dt)
                       (+ last-suppression-clock suppression-dt)
                       bvs-to-process-next
                       (if (zero? non-burn-period-dt)
                         spot-ignitions
                         {})
                       spot-count
                       (long total-cells-suppressed)
                       (long previous-num-perimeter-cells)
                       []
                       (double fraction-contained)))
              (recur new-clock
                     new-band
                     (+ new-clock burn-period-dt)
                     suppression-clock
                     burn-vectors
                     (if (zero? non-burn-period-dt)
                       spot-ignitions
                       {})
                     spot-count
                     total-cells-suppressed
                     previous-num-perimeter-cells
                     ignited-cells-since-last-suppression
                     fraction-contained)))

          :else
          (let [dt-until-new-hour              (- 60.0 (rem global-clock 60.0))
                dt-until-non-burn-period-clock (- non-burn-period-clock global-clock)
                bvs                            (if (and (> global-clock ignition-start-time)
                                                        (or (= dt-until-new-hour 60.0)
                                                            (= dt-until-non-burn-period-clock burn-period-dt)))
                                                 (recompute-burn-vectors inputs matrices band burn-vectors)
                                                 burn-vectors)
                timestep                       (double
                                                (cond-> (compute-dt cell-size bvs)
                                                  dt-until-new-hour              (min dt-until-new-hour)
                                                  dt-until-max-runtime           (min dt-until-max-runtime)
                                                  dt-until-non-burn-period-clock (min dt-until-non-burn-period-clock)
                                                  suppression-dt?                (min (let [dt-until-suppression-clock (- suppres
                                                                                         dt-until-suppression-clock))))
                new-clock                      (+ global-clock timestep)
                [grown-bvs
                 ignited-cells]                (grow-burn-vectors! matrices global-clock timestep bvs)
                [transitioned-bvs
                 transition-ignited-cells]     (->> grown-bvs
                                                    (ignited-cells->burn-vectors inputs matrices ignited-cells)
                                                    ;; Why this 1.99 max-fractional-distance values? To enforce the
                                                    ;; INVARIANT that any Burn Vector in the current timestep cannot travel more t
                                                    ;; (does not account elevation just horizontal). See compute-dt. There is one
                                                    ;; EDGE CASE in which bvs can burn pass one cell size given the spread rate an
                                                    ;; the computed timestep. It is when a bv gets created from ignited-cells->bur
                                                    ;; The spread rate of this bv was not included in the compute-dt step and so i
                                                    ;; higher spread rate than any other bv used in compute-dt then it will burn m
                                                    ;; single cell size.
                                                    ;; An alternative to capping the fractional-distance would be to adjust the ti
                                                    ;; to be an upper bound of the time required for the burn vectors to cross.
                                                    ;; We tried that based on max-spread-rate calculations, but found it to be ext
```

```
                                        ;; inefficient because it made the time steps unreasonably small.
                                        ;; Capping the fractional distance caused only small changes in prediction acc
                                        ;; so we went with that.
                                        ;; More background on this approach and how it relates to the Morais 2001 pape
                                        ;; https://github.com/pyregence/gridfire/pull/125#issuecomment-1322861034
                                        (promote-burn-vectors inputs matrices global-clock new-clock 1.99)
                                        ;; (= 0.99 max-fractional-distance) to make sure that
                                        ;; transitioning Burn Vectors will not double-transition.
                                        (transition-burn-vectors inputs matrices band global-clock new-clock 0.99))
            promoted-transitioned-bvs   (->> transitioned-bvs
                                        (ignited-cells->burn-vectors inputs matrices transition-ignited-cells)
                                        (promote-burn-vectors inputs matrices global-clock new-clock 0.99)) ;TODO opt
            ;; NOTE why are we promoting exactly twice in the above? Because there are 2 ignition-causing phases: grow-burn-ve
            [spot-bvs
             spot-ignite-now-count
             spot-ignite-later
             spot-ignited-cells]        (compute-spot-burn-vectors! inputs
                                                                    matrices
                                                                    spot-ignitions
                                                                    (into ignited-cells transition-ignited-cells)
                                                                    band
                                                                    new-clock)
            promoted-spot-bvs           (->> (into promoted-transitioned-bvs spot-bvs)
                                        (promote-burn-vectors inputs matrices global-clock new-clock 1.49)) ;TODO opt
            [transition-promoted-spot-bvs
             _]                         (transition-burn-vectors inputs matrices band global-clock new-clock 0.49 promoted
            all-ignited-cells           (-> ignited-cells
                                        (into transition-ignited-cells)
                                        (into spot-ignited-cells))]
        ;; TODO if spot ignitions is updated to have varying burn probability make sure there are no duplicates
        ;; of ignited cells in this list of ignited cells passed to compute-directional-in-situ-values!
        (when compute-directional-values?
          (compute-directional-in-situ-values! matrices all-ignited-cells))
        (recur new-clock
               (min->hour new-clock)
               non-burn-period-clock
               suppression-clock
               transition-promoted-spot-bvs
               spot-ignite-later
               (+ spot-count ^long spot-ignite-now-count)
               total-cells-suppressed
               previous-num-perimeter-cells
               (into ignited-cells-since-last-suppression all-ignited-cells)
               fraction-contained))))
    (let [fire-type-matrix (:fire-type-matrix matrices)]
      {:exit-condition                  (if (>= global-clock ignition-stop-time) :max-runtime-reached :no-burnable-fuels)
       :global-clock                    global-clock
       :burn-time-matrix                (:burn-time-matrix matrices)
       :fire-line-intensity-matrix      (:fire-line-intensity-matrix matrices)
       :fire-spread-matrix              (:fire-spread-matrix matrices)
       :fire-type-matrix                fire-type-matrix
       :flame-length-matrix             (:flame-length-matrix matrices)
       :directional-flame-length-matrix (:directional-flame-length-matrix matrices)
       :spot-matrix                     (:spot-matrix matrices)
       :spread-rate-matrix              (:spread-rate-matrix matrices)
       :surface-fire-count              (->> fire-type-matrix
                                        (d/emap #(if (= ^double % 1.0) 1 0) :int64)
                                        (dfn/sum))
       :crown-fire-count                (->> fire-type-matrix
                                        (d/emap #(if (>= ^double % 2.0) 1 0) :int64)
                                        (dfn/sum))
       :spot-count                      spot-count}))))
```

GridFire features some computational helpers for fast handling of spread directions:

```
(defn- direction-bit->angle
  ^double [^long dir-bit]
```

```
  (case dir-bit
    0 0.0
    1 45.0
    2 90.0
    3 135.0
    4 180.0
    5 225.0
    6 270.0
    7 315.0))

(defn direction-angle->bit
  ^long [^double dir-angle]
  (case-double dir-angle
    0.0   0
    45.0  1
    90.0  2
    135.0 3
    180.0 4
    225.0 5
    270.0 6
    315.0 7))

(defn direction-angle->i-incr
  ^long [^double dir-angle]
  (case-double dir-angle
    (0.0 45.0 315.0)   -1
    (135.0 180.0 225.0) 1
    (90.0 270.0)        0))

(defn direction-angle->j-incr
  ^long [^double dir-angle]
  (case-double dir-angle
    (45.0 90.0 135.0)   1
    (0.0 180.0)         0
    (225.0 270.0 315.0) -1))

(defn diagonal? [^double direction-angle]
  (case-double direction-angle
    (0.0  90.0  180.0 270.0) false
    (45.0 135.0 225.0 315.0) true))
```

5.5 Estimation of Fireline-normal Spread Direction and Rate

A subtle point is of GridFire's fire-spread computation is that, despite the fact that it directly represents wavelets, it still needs to **estimate the fireline-normal direction of spread.** This is needed for fireline intensity computations, which require a rate of spread orthogonal to the fireline. Burn vectors cannot directly provide that information because they represent the wavelets, not the fireline. GridFire estimates the fireline-normal direction by computing the **gradient of the Time of Arrival** (recorded in **:burn-time-matrix**):

```
(def ^:const default-incidence-cosine
  "The default guess for the cosine of the fireline-incidence of maximum spread.

  When lacking information to estimate the fireline-normal direction,
  we conservatively assume that it is orthogonal to the direction of maximum spread (flanking fire),
  and therefore the cosine is 0."
  0.0)

(defn estimate-fireline-incidence-cosine
  "Estimates the fireline-normal direction based on Time-of-Arrival data; more precisely, returns the cosine of
  the 'incidence' angle between the direction of advancing fire front and
  some direction supplied as `dir-angle` (in degrees from North).
```

```
Equivalently, returns (a guess of) the dot-product <duv|fln>, in which
|duv> is the unit vector in the supplied direction and
|fln> is the fireline-normal unit vector.

The arguments are:
- `burn-time-getter`: a getter for the Time of Arrival of the fire,
  as returned by `(grid-lookup/mgetter-double burn-time-matrix)`
- `incoming-burnvec` (optional): the Burn Vector where we want to estimate the fireline-normal spread rate,
   or nil if not available;
- `dir-angle`: angle to the North direction in degrees, representing the direction of interest
   (typically :max-spread-direction);
- `fallback-guess`: a default cosine to return when estimation fails.

Returns `fallback-guess` when there is not enough information to make a gradient estimate,
which happens in particular when `incoming-burnvec` is nil."
;; NOTE why return the cosine to some other direction, (Val, 30 Nov 2022)
;; rather than an angle or vector representing the fireline-normal direction? Several reasons:
;; 1. When the fireline-normal direction cannot be estimated,
;;    it can be difficult to find a sensible "default value" for a direction.
;;    Less so for a cosine.
;; 2. The cosine is what we need downstream, and is easy and fast to compute.
;;    Turning that into an angle would be a useless and costly computation.
^double [burn-time-getter incoming-burnvec ^double dir-angle ^double fallback-guess]
(if (nil? incoming-burnvec)
  fallback-guess
  ;; NOTE what with gradient geometry and geospatial encoding conventions,
  ;; this function combines so many confusing part of the code
  ;; that I felt the need for abundant commenting and naming. (Val, 30 Nov 2022)
  ;; NOTATION this code will be using the <bra|ket> notation for vectors and dot-products:
  ;; <a|b> represents the dot-product between vectors |a> and |b>.
  ;; |i> and |j> represent the unit vectors which respectively increment i and j by 1,
  ;; such that the i-coordinate of |a> is <i|a> (or equivalently <a|i>),
  ;; and <a|b> = <a|i><i|b> + <a|j><j|b>.
  ;; Our dot-product of choice <.|.> is thus defined by #{|i> |j>} forming an orthonormal basis.
  ;; By definition of the gradient, <grad-F|i> = F/i and <grad-F|j> = F/j.
  (let [ignited-this-cell? (>= (burn-vec/get-fractional-distance incoming-burnvec)
                               0.5)
        bv-dir-angle       (burn-vec/get-direction incoming-burnvec)
        ;; The fireline-normal direction should be given by the gradient of the Time of Arrival (ToA).
        ;; The problem is that the gradient is not easy to estimate at the fire perimeter,
        ;; because we might not yet have a gradient for all neighboring cells.
        ;; For estimating the gradient, we choose the cell of origin of the Burn Vector,
        ;; because we want this cell to have enough neighbors which can contribute a ToA
        ;; to the gradient computation.
        grad-cell-i        (- (burn-vec/get-i incoming-burnvec)
                              (if ignited-this-cell?
                                0
                                (direction-angle->i-incr bv-dir-angle)))
        grad-cell-j        (- (burn-vec/get-j incoming-burnvec)
                              (if ignited-this-cell?
                                0
                                (direction-angle->j-incr bv-dir-angle)))
        ;; Having chosen a nearby cell, we now estimate the ToA-gradient at that cell:
        toa-ij             (grid-lookup/double-at burn-time-getter grad-cell-i grad-cell-j)
        ;; IMPROVEMENT it might be interesting to factor out the gradient computation to a function.
        ;; The problem is avoiding performance regressions: packing 2 primitive values into an object might be significantly less
        ;; One (hacky) strategy might be to encode 2 32-bit floats into 1 primitive long.
        <grad-ToA|i>       (let [toa-i   toa-ij
                                 toa-i+1 (grid-lookup/double-at burn-time-getter (unchecked-inc grad-cell-i) grad-cell-j)
                                 toa-i-1 (grid-lookup/double-at burn-time-getter (unchecked-dec grad-cell-i) grad-cell-j)]
                             (gradient/estimate-dF -1.0 toa-i-1 toa-i toa-i+1))
        <grad-ToA|j>       (let [toa-j   toa-ij
                                 toa-j+1 (grid-lookup/double-at burn-time-getter grad-cell-i (unchecked-inc grad-cell-j))
                                 toa-j-1 (grid-lookup/double-at burn-time-getter grad-cell-i (unchecked-dec grad-cell-j))]
                             (gradient/estimate-dF -1.0 toa-j-1 toa-j toa-j+1))
        grad-ToA-norm      (Math/sqrt (+ (* <grad-ToA|i> <grad-ToA|i>)
                                         (* <grad-ToA|j> <grad-ToA|j>)))]
    ;; A zero gradient happens in particular when gradient estimation has failed on both axes, for lack of ToA data.
```

```
      ;; IMPROVEMENT: when that happens, we might want to retry with diagonal axes.
      ;; IMPROVEMENT we might also want to discard a near-zero gradient - probably too chaotic.
      ;; The problem is that we'd probably need to know the cell size to determine a sensible threshold for 'near-zero'.
      (if (zero? grad-ToA-norm)
        fallback-guess
        (let [;; The fireline-normal unit vector |fln> is estimated by scaling the ToA gradient to unit length:
              <fln|i>   (/ <grad-ToA|i> grad-ToA-norm)
              <fln|j>   (/ <grad-ToA|j> grad-ToA-norm)
              ;; We will now resolve the unit vector |duv> corresponding to the supplied `dir-angle`:
              dir-rad   (deg->rad dir-angle)
              ;; These 2 lines reflect our convention for what how we define "angle from North" and the x,y axes:
              <duv|y>   (Math/cos dir-rad)
              <duv|x>   (Math/sin dir-rad)
              ;; These 2 lines reflect our convention relating matrix coordinates to spatial coordinates: |i> = -|y> and |j> = |x>,
              ;; where |x> is the unit vector in the direction of increasing y, and likewise of |y>.
              <duv|i>   (- <duv|y>)
              <duv|j>   <duv|x>
              ;; Finally computing our dot-product, by applying the coordinates-based formula: <a|b> = <a|i><i|b> + <a|j><j|b>.
              ;; Note: if you want to reconcile the angular-directions and unit-vectors perspectives,
              ;; it can be insightful to view the following formula as an application of
              ;; the trigonometric identity: cos(a-b) = cos(a)*cos(b) + sin(a)*sin(b)
              <duv|fln> (+ (* <duv|i> <fln|i>)
                           (* <duv|j> <fln|j>))]
          <duv|fln>)))))
```

The above estimation of direction feeds into elliptical-wavelet computations which must now be based on the exact continuous model rather than a discrete representation:

```
(ns gridfire.elliptical
  "Computations based on the Elliptical Wavelets model.")

(defn fireline-normal-spread-rate-scalar
  "Computes the ratio of fireline-normal spread rate to maximal (heading) spread rate,
  in our idealized elliptical-wavelet model
  (which assumes wavelets shaped as ellipses growing around their focus point).

  Given:
  - eccentricity (dimensionless, in [0.0, 1.0)): the eccentricity of the elliptical fire-spread wavelet;
  - cos-ang (dimensionless, in [-1.0 1.0]): the cosine of the angle
  from the unit vector orthogonal to the fireline (pointing towards the unburned area)
  and the unit vector in the direction of maximum spread (in flat terrain, the wind direction)
  -equivalently, cos-ang is the dot-product between those unit vectors-
  e.g. cos-ang = 1.0 for heading fire, cos-ang = -1.0 for backing fire,
  and cos-ang = 0 for flanking fire;

  returns a number by which to downscale max-spread-rate to get
  the fireline-normal spread rate,
  i.e. the speed at which this segment of the fire line advances
  orthogonally to itself."
  ^double
  [^double eccentricity
   ^double cos-ang]
  (let [E         eccentricity
        E*cos-ang (* E cos-ang)]
    ;; NOTE for a mathematical derivation of this formula, see files:
    ;; org/elliptical/elliptical-wavelet-spread-rate-math.jpg
    ;; org/elliptical/elliptical-wavelet-spread-rate-math2.jpg
    (/ (+ E*cos-ang
          (Math/sqrt (+ (- 1.0 (* E E))
                        (* E*cos-ang E*cos-ang))))
       (+ 1.0 E))))
```

```
(ns gridfire.elliptical-test
  (:require [clojure.test                 :refer [deftest is testing]]
            [gridfire.elliptical          :refer [fireline-normal-spread-rate-scalar]]
```

```
            [gridfire.surface-fire         :refer [compute-spread-rate]]
            [gridfire.conversion           :as convert]))

(defn almost-=?
  [^double d1 ^double d2]
  (< (Math/abs (- d1 d2))
     1e-5))

(deftest ^:unit fireline-normal-spread-rate-scalar--examples
  (testing "With non-zero eccentricity,"
    (let [eccentricity 0.9]
      (testing "for the heading fire (max-spread and fireline-normal directions are aligned),"
        (let [cos-ang 1.0]
          (is (almost-=? 1.0
                         (fireline-normal-spread-rate-scalar eccentricity cos-ang))
              "the fireline-normal spread rate achieves the max spread rate.")))
      (testing "for the backing fire (max-spread and fireline-normal directions are opposed),"
        (let [cos-ang         -1.0
              min-spread-rate (-> 1.0 (* (- 1.0 eccentricity)) (/ (+ 1.0 eccentricity)))]
          (is (almost-=? min-spread-rate
                         (fireline-normal-spread-rate-scalar eccentricity cos-ang))
              "the fireline-normal spread rate achieves the minimum spread rate.")))
      (testing "for the flanking fire (max-spread and fireline-normal directions are orthogonal),"
        (let [cos-ang              0.0
              LoW                  (Math/pow (- 1.0 (Math/pow eccentricity 2))
                                             -0.5)
              ;; ellipse half-length (A) and half-width (B)
              A                    (/ 1.0 (+ 1.0 eccentricity))
              B                    (/ A LoW)
              flanking-spread-rate B]
          (is (almost-=? flanking-spread-rate
                         (fireline-normal-spread-rate-scalar eccentricity cos-ang))
              "the fireline-normal spread rate is the flanking spread rate (ellipse half-width)."))))
      (let [;; arbitrary values
            max-spread-rate  12.0
            spread-direction 0.0]
        (testing "the fireline-normal and burn-vector spread rates,"
          (testing "when the max-spread and fireline-normal directions are not aligned,"
            (doseq [cos-ang [-0.8 -0.2 0.0 0.3 0.5 0.7]]
              (let [burn-vec-spread-rate (compute-spread-rate max-spread-rate
                                                              (-> cos-ang (double) (Math/acos) (convert/rad->deg))
                                                              eccentricity
                                                              spread-direction)]
                (is (< burn-vec-spread-rate
                       (* max-spread-rate (fireline-normal-spread-rate-scalar eccentricity cos-ang)))
                    "are not equal (and yes, that is expected, because some non-normal directions achieve faster normal-projected s
          (testing "when the max-spread and fireline-normal directions are aligned,"
            (doseq [cos-ang [-1.0 1.0]]
              (let [burn-vec-spread-rate (compute-spread-rate max-spread-rate
                                                              (-> cos-ang (double) (Math/acos) (convert/rad->deg))
                                                              eccentricity
                                                              spread-direction)]
                (is (almost-=? burn-vec-spread-rate
                               (* max-spread-rate (fireline-normal-spread-rate-scalar eccentricity cos-ang)))
                    "are equal (and only then.)"))))))
      (testing "the fire-normal spread rate increases as the angle gets reduced"
        (is (apply <
                   (concat [0.0]
                           (->> (range -1.0 1.0 0.1)
                                (map (fn [^double cos-ang]
                                       (fireline-normal-spread-rate-scalar eccentricity cos-ang))))
                           [1.0])))))))
```

5.6 Spotting Model Formulas

Gridfire can optionally include spot fires using a cellular automata model described in (Perryman et al., 2012). The model is broken up into four submodels: Surface Spread, Tree Torching, Firebrand Dispersal, and Spot Ignition. For Surface Spread and Tree Torching, the Perryman model uses Rothermel (1972) and Van Wagner 1977 respectively. Gridfire will use the same models described in the previous sections.

5.6.1 Firebrand Dispersal Model

The Firebrand Dispersal model describes the distributions of firebrands relative to the wind direction. The location where the firebrand lands is represented by the random vector "delta" from the location of origin:

$$\vec{\Delta} := \Delta_X.\vec{w} + \Delta_Y.\vec{w}_\perp$$

in which $\vec{w}, \vec{w}_\perp$ are unit vectors respectively parallel and perpendicular to the wind direction, and Δ_X, Δ_Y are the random variables for coordinates.

Following Perryman et al. (2012), Sardoy et al. (2008) and Himoto and Tanaka (2005), we model Δ_X and Δ_Y to be independent, with Δ_X following as a log-normal distribution, and Δ_Y following a zero-mean normal distribution, conditional on the fire behavior Φ at the cell of origin:

$$\begin{aligned}\ln(\Delta_X/1\text{m})|\Phi &\sim \text{Normal}(\mu = \mu_X(\Phi), \sigma = \sigma_X(\Phi)) \\ \Delta_Y|\Phi &\sim \text{Normal}(\mu = 0, \sigma = \sigma_Y(\Phi))\end{aligned}$$

where $\text{Normal}(\mu, \sigma)$ denotes a one-dimensional Gaussian distribution with mean μ and standard deviation σ. Note that μ_X and σ_X are in log-space, therefore dimensionless. For the sake of light notation, the conditioning on Φ will be implicit from now on, e.g. we will write Δ_X instead of $\Delta_X|\Phi$.

Since the results are distance deltas relative to the wind direction we must convert this to deltas in our coordinate plane. We can convert these deltas by using trigonometric functions.

```
(defn hypotenuse ^double
  [x y]
  (Math/sqrt (+ (Math/pow x 2) (Math/pow y 2))))

(defn deltas-wind->coord
  "Converts deltas from the torched tree in the wind direction to deltas
  in the coordinate plane"
  [deltas ^double wind-direction]
  ;; IMPROVEMENT re-implement using sin, cos and dot-products, cleaner and faster. (Val, 16 Mar 2023)
  (mapv (fn [[d-paral d-perp]]
          (let [d-paral (double d-paral)
                d-perp  (double d-perp)
                H       (hypotenuse d-paral d-perp)
                t1      wind-direction
                t2      (convert/rad->deg (Math/atan (/ d-perp d-paral)))
                t3      (+ t1 t2)]
            [(* H (Math/sin (convert/deg->rad t3)))
             (* -1 H (Math/cos (convert/deg->rad t3)))]))
        deltas))

(defn firebrands
  "Returns a sequence of cells [i,j] that firebrands land in.
   Note: matrix index [i,j] refers to [row, column]. Therefore, we need to flip
   [row,column] to get to [x,y] coordinates."
  [deltas wind-towards-direction cell ^double cell-size]
  (let [step         (/ cell-size 2)
```

```
      [y x]          (mapv #(+ step (* ^double % cell-size)) cell)
      x              (double x)
      y              (double y)
      coord-deltas (deltas-wind->coord deltas wind-towards-direction)]
  (mapv (fn [[dx dy]]
          (let [dx (double dx)
                dy (double dy)]
            [(long (Math/floor (/ (+ dy y) cell-size)))
             (long (Math/floor (/ (+ dx x) cell-size)))]))
        coord-deltas)))
```

5.6.2 ELMFIRE-like resolution of log-normal parameters

We now need to define the functions $\mu_X(\Phi)$ and $\sigma_X(\Phi)$. Here we depart from Sardoy et al. (2008), and reproduce the model of ELMFIRE, which models that dependency using the following relationships between the moments of the distribution and the fire line intensity I and wind speed U:

$$\begin{aligned}\mathbb{E}[\Delta_X] =& \Delta_1 \left(\frac{I}{1\ \text{kW/m}}\right)^{e_I} \left(\frac{U}{1\ \text{m/s}}\right)^{e_U} \\ \text{Var}[\Delta_X] =& r_{\frac{V}{E}} \mathbb{E}[\Delta_X]\end{aligned}$$

In which $\Delta_1, e_I, e_U, r_{\frac{V}{E}}$ are configured by the keys shown in Table 6.

Table 6: Spotting parameters for the downwind distribution per the ELMFIRE model

Parameter	Unit	Description	Configuration key
Δ_1	m	Mean landing distance in unit conditions	:mean-distance
e_I	-	Intensity exponent	:flin-exp
e_U	-	Wind exponent	:ws-exp
$r_{\frac{V}{E}}$	m	Variance-over-Mean ratio	:normalized-distance-variance

Note: one potential way in which this model can misbehave is that the variance is proportional to the expected value, and therefore the coefficient of variation is driven to zero as the expected value goes to infinity, making the distribution less and less dispersed around its mean. This means that, in high-wind/high-intensity conditions, all the firebrands will tend to land at approximately the same (large) distance, following a narrow near-normal distribution.

From the above moments, μ_X and σ_X can be obtained using the properties of the log-normal distribution:

$$\begin{aligned}\sigma_X =& \ln\left(1 + \frac{\text{Var}[\Delta_X]}{\mathbb{E}[\Delta_X]^2}\right) \\ \mu_X =& \ln\frac{\mathbb{E}[\Delta_X]^2}{\sqrt{\text{Var}[\Delta_X] + \mathbb{E}[\Delta_X]^2}}\end{aligned}$$

```
(ns gridfire.spotting.elmfire
  "Resolution of spotting parameters as done in ELMFIRE.")

(defn resolve-exp-delta-x
  "Computes the expected value E[ΔX] of the downwind spotting distance ΔX (in meters), as a function of:
  - `spotting-config`: a map of spotting parameters,
  - `fire-line-intensity` (kW/m)
```

```
  - `wind-speed-20ft` (m/s)"
  ^double [spotting-config ^double fire-line-intensity ^double wind-speed-20ft]
  (let [a        (-> spotting-config :mean-distance (double))
        flin-exp (-> spotting-config :flin-exp (double))
        ws-exp   (-> spotting-config :ws-exp (double))]
    (* a
       (* (Math/pow fire-line-intensity flin-exp)
          (Math/pow wind-speed-20ft ws-exp)))))

(defn resolve-var-delta-x
  "Computes the variance Var[ΔX] (in m^2) of the downwind spotting distance ΔX, as a function of:
  - `spotting-config`: a map of spotting parameters,
  - `exp-delta-x`: E[ΔX] (in m)"
  ^double [spotting-config ^double exp-delta-x]
  (* (-> spotting-config :normalized-distance-variance (double))
     exp-delta-x))

(defn lognormal-mu-from-moments
  ^double [^double mean ^double variance]
  (let [m2 (Math/pow mean 2)]
    (Math/log (/ m2
                 (Math/sqrt (+ m2 variance))))))

(defn lognormal-sigma-from-moments
  ^double [^double mean ^double variance]
  (Math/log (+ 1.0
               (/ variance mean))))

(defn resolve-lognormal-params
  [spotting-config ^double fire-line-intensity ^double wind-speed-20ft]
  (let [exp-delta-x (resolve-exp-delta-x spotting-config fire-line-intensity wind-speed-20ft)
        var-delta-x (resolve-var-delta-x spotting-config exp-delta-x)]
    {:prob.lognormal/mu    (lognormal-mu-from-moments exp-delta-x var-delta-x)
     :prob.lognormal/sigma (lognormal-sigma-from-moments exp-delta-x var-delta-x)}))
```

5.6.3 Sardoy-like resolution of log-normal parameters FIXME move

Following Sardoy et al. (2008), we model $\mu_X(\Phi)$ and $\sigma_X(\Phi)$ to be functions of the fireline intensity I and the wind speed U:

$$\mu_X(I,U) = a_\mu \left(\frac{I}{1\ \mathrm{MW/m}}\right)^{p_\mu} \left(\frac{U}{1\ \mathrm{m/s}}\right)^{q_\mu} + b_\mu$$
$$\sigma_X(I,U) = a_\sigma \left(\frac{I}{1\ \mathrm{MW/m}}\right)^{p_\sigma} \left(\frac{U}{1\ \mathrm{m/s}}\right)^{q_\sigma} + b_\sigma$$

in which the $a., b., p., q.$ are parameters supplied from the GridFire configuration, with names starting with **:delta-x-ln-** (examples below).

```
(ns gridfire.spotting.sardoy)

;; IMPROVEMENT allow for this model to be configured instead of the Elmfire one.
(defn sardoy-resolve-mu-x
  ^double [spotting-config ^double fire-line-intensity ^double wind-speed-20ft]
  (+ (* (double (:delta-x-ln-mu-a spotting-config))
        (double
         (* (Math/pow (-> fire-line-intensity
                          ;; kW/m -> MW/m
                          (* 1e-3))
                      (double (:delta-x-ln-mu-p spotting-config)))
            (Math/pow wind-speed-20ft                          ; m/s
                      (double (:delta-x-ln-mu-q spotting-config)))))))
```

```
     (double (:delta-x-ln-mu-b spotting-config))))

(defn sardoy-resolve-sigma-x
  ^double [spotting-config ^double fire-line-intensity ^double wind-speed-20ft]
  (+ (* (double (:delta-x-ln-sigma-a spotting-config))
        (* (Math/pow (-> fire-line-intensity
                         ;; kW/m -> MW/m
                         (* 1e-3))
                     (double (:delta-x-ln-sigma-p spotting-config)))
           (Math/pow wind-speed-20ft                    ; m/s
                     (double (:delta-x-ln-sigma-q spotting-config)))))
     (double (:delta-x-ln-sigma-b spotting-config))))
```

Typical ranges are shown in the following table:

Table 7: Typical ranges for Δ_X distribution characteristics from (Sardoy et al., 2008)

Model	$I^{p_\mu}U^{q_\mu}$	μ_X	$I^{p_\sigma}U^{q_\sigma}$	σ_X	$\mathbb{E}[\Delta_X]$ (m)	$\text{CV}[\Delta_X]$
Wind-driven	2.25 - 3.80	2.95 - 5.00	0.88 - 0.93	0.876 - 1.124	92.00 - 911.67	107% - 159%
Buoyancy-driven	1.80 - 2.60	3.79 - 4.96	1.05 - 1.30	1.093 - 1.308	262.82 - 1102.77	152% - 213%

The above values were derived from (Sardoy et al., 2008), see code block below. $\text{CV}[\Delta_X]$ is the coefficient of variation $\text{CV}[\Delta_X] := \text{Var}[\Delta_X]^{\frac{1}{2}}/\mathbb{E}[\Delta_X] = \sqrt{e^{\sigma_X^2} - 1}$, a measure of how dispersed the distribution is around its mean.

The behavior of these formulas on the values found in (Sardoy et al., 2008) is demonstrated here:

```
(def sardoy-wind-driven-lognormal-params
  {:delta-x-ln-mu-a    1.32
   :delta-x-ln-mu-p    0.26
   :delta-x-ln-mu-q    0.11
   :delta-x-ln-mu-b    -0.02
   :delta-x-ln-sigma-a 4.95
   :delta-x-ln-sigma-p -0.01
   :delta-x-ln-sigma-q -0.02
   :delta-x-ln-sigma-b -3.48})

(def sardoy-buoyancy-driven-lognormal-params
  {:delta-x-ln-mu-a    1.47
   :delta-x-ln-mu-p    0.54
   :delta-x-ln-mu-q    -0.55
   :delta-x-ln-mu-b    1.14
   :delta-x-ln-sigma-a 0.86
   :delta-x-ln-sigma-p -0.21
   :delta-x-ln-sigma-q 0.44
   :delta-x-ln-sigma-b 0.19})

(defn sardoy-dist-values
  [spotting-config mu-iu sigma-iu]
  (let [mu-x        (+ (* (double (:delta-x-ln-mu-a spotting-config))
                          mu-iu)
                       (double (:delta-x-ln-mu-b spotting-config)))
        sigma-x     (+ (* (double (:delta-x-ln-sigma-a spotting-config))
                          sigma-iu)
                       (double (:delta-x-ln-sigma-b spotting-config)))
        exp-delta-x (spotting/deltax-expected-value mu-x sigma-x)]
    {"$I^{p_\\mu}U^{q_\\mu}$"               (format "%.2f" mu-iu)
     "$\\mu_X$"                             (format "%.2f" mu-x)
     "$I^{p_\\sigma}U^{q_\\sigma}$"         (format "%.2f" sigma-iu)
     "$\\sigma_X$"                          (format "%.3f" sigma-x)
     "$\\mathbb{E}[\\Delta_X]\\text{ (m)}$" (format "%.2f" exp-delta-x)
     "$\\text{CV}[\\Delta_X]$"              (str (format "%.0f" (* 100 (spotting/deltax-coefficient-of-variation sigma-x)))
                                                 "%")}))
```

```
(defn sardoy-typical-ranges-row
  [model-name spotting-config [mu-iu-min mu-iu-max] [sigma-iu-min sigma-iu-max]]
  (into {"Model" model-name}
        (merge-with (fn [v1 v2] (str v1 " - " v2))
                    (sardoy-dist-values spotting-config mu-iu-min sigma-iu-min)
                    (sardoy-dist-values spotting-config mu-iu-max sigma-iu-max))))

(comment

  (pp/print-table
   (->> [["Wind-driven" sardoy-wind-driven-lognormal-params
          [2.25 3.8]
          ;; NOTE it appears that the 1st plot of Fig 10 has incorrect x labels, (Val, 16 Mar 2023)
          ;; so I worked out the above bounds by inverting the y = ax + b relationship.
          #_[1.5 2.0]
          [0.88 0.93]]
         ["Buoyancy-driven" sardoy-buoyancy-driven-lognormal-params [1.8 2.6] [1.05 1.3]]]
        (mapv #(apply sardoy-typical-ranges-row %))))
  ;|           Model | £I^{p_\mu}U^{q_\mu}£ |     £\mu_X£ | £I^{p_\sigma}U^{q_\sigma}£ |    £\sigma_X£ | £\mathbb{E}[\Delta_X]\text{
  ;|-----------------+----------------------+-------------+---------------------------+---------------+----------------------------
  ;|     Wind-driven |          2.25 - 3.80 | 2.95 - 5.00 |               0.88 - 0.93 | 0.876 - 1.124 |                  92.00 - 9
  ;| Buoyancy-driven |          1.80 - 2.60 | 3.79 - 4.96 |               1.05 - 1.30 | 1.093 - 1.308 |                262.82 - 11

  *e)

(deftest lognormal-params-sanity-checks-test
  (let [fireline-intensity 30e3
        wind-speed         11.17]
    (testing "When using the Sardoy2008 wind-driven parameters"
      (let [spotting-params sardoy-wind-driven-lognormal-params
            expected-mux    4.147842572687786
            expected-sigmax 1.0790349643407962
            expected-sigmay 457.4026146906983
            avg-deltax      (spotting/deltax-expected-value expected-mux expected-sigmax)]
        (testing "the value of mu_X"
          (is (within? (spotting-sardoy/sardoy-resolve-mu-x spotting-params fireline-intensity wind-speed)
                       expected-mux
                       1e-6))
          (testing "is close to the value in Fig9 of Sardoy2008"
            (is (within? expected-mux 4.20 0.1))))
        (testing "the value of sigma_X"
          (is (within? (spotting-sardoy/sardoy-resolve-sigma-x spotting-params fireline-intensity wind-speed)
                       expected-sigmax
                       1e-6))
          (testing "is close to the value in Fig9 of Sardoy2008"
            (is (within? expected-sigmax 0.96 0.2))))
        (testing "E[ΔX] (the average ΔX)"
          (is (within? avg-deltax 371.72303737837905
                       1e-3)
              (format "is about %.0f ft." avg-deltax)))
        (testing "the default value of sigma_Y"
          (is (within? (spotting/himoto-resolve-default-sigma-y-from-lognormal-params expected-mux expected-sigmax)
                       expected-sigmay
                       1e-3)
              (format "is about %.0f ft." expected-sigmay))
          (is (> expected-sigmay avg-deltax)
              "is larger than E[ΔX], which means it is probably a bad idea to use it."))))
    (testing "When using the Sardoy2008 buoyancy-driven parameters"
      (let [spotting-params sardoy-buoyancy-driven-lognormal-params
            expected-mux    3.586434356261935
            expected-sigmax 1.407436684216556
            expected-sigmay 1112.8976983507514
            avg-deltax      (spotting/deltax-expected-value expected-mux expected-sigmax)]
        (is (within? (spotting-sardoy/sardoy-resolve-mu-x spotting-params fireline-intensity wind-speed)
                     expected-mux
                     1e-6)
            (format "mu_X is about %.2f" expected-mux))
```

```
      (is (within? (spotting-sardoy/sardoy-resolve-sigma-x spotting-params fireline-intensity wind-speed)
                   expected-sigmax
                   1e-6)
          (format "sigma_X is about %.2f" expected-sigmax))
      (testing "E[ΔX] (the average ΔX)"
        (is (within? avg-deltax 318.94593966227154
                     1e-3)
            (format "is about %.0f ft." avg-deltax)))
      (testing "the default value of sigma_Y"
        (is (within? (spotting/himoto-resolve-default-sigma-y-from-lognormal-params expected-mux expected-sigmax)
                     expected-sigmay
                     1e-3)
            (format "is about %.0f ft" expected-sigmay))
        (is (> expected-sigmay avg-deltax)
            "is larger than E[ΔX], which means it is probably a bad idea to use it.")))))
```

5.6.4 Wind-perpendicular Dispersal

For σ_Y, we either let the user specify an explicit value in the GridFire configuration (at key **:delta-y-sigma**, in meters), which makes it equivalent to the model of (Perryman et al., 2012), or we default to deriving them from equation (28) of Himoto and Tanaka (2005), which is equivalent to:

$$\sigma_Y = 0.92D = 0.92\frac{0.47}{0.88^2}\frac{\mathrm{Var}[\Delta_X]}{\mathbb{E}[\Delta_X]}$$

$\mathrm{Var}[\Delta_X]$ and $\mathbb{E}[\Delta_X]$ can be calculated from the properties of the log-normal distribution:

$$\mathbb{E}[\Delta_X] = 1\mathrm{m} \times \exp\left(\mu_X + \frac{1}{2}\sigma_X^2\right)$$
$$\mathrm{Var}[\Delta_X] = \mathbb{E}[\Delta_X]^2\left(e^{\sigma_X^2} - 1\right)$$

Combining the above equations and applying some algebra yields a formula for σ_Y:

$$\sigma_Y = 0.92D = 1\mathrm{m} \times 0.5584 \times e^{\mu_X} \times e^{\frac{1}{2}\sigma_X^2}\left(e^{\frac{1}{2}\sigma_X^2} - 1\right)\left(e^{\frac{1}{2}\sigma_X^2} + 1\right)$$

Typical values are shown in table 8.

Table 8: Typical values for $\vec{\Delta}$ distribution characteristics derived from (Himoto and Tanaka, 2005)

B^*	$\mathrm{CV}[\Delta_X]$	σ_X	$\sigma_Y/\mathbb{E}[\Delta_X]$
20	69%	0.72	0.27
50	51%	0.64	0.14
100	40%	0.58	0.09
150	35%	0.55	0.07
200	32%	0.53	0.06

These values were obtained using the following code:

```
(defn himoto-eq-28-values
  [B*]
  (let [D                  0.08
        ;; NOTE the above value for D seems absurdly low, and so do the predicted E[ΔX].
```

```
        ;; I suspect a typo in the Himoto2005 paper.
        std-delta-x-over-D (* 0.88 (Math/pow B* (/ 1.0 3)))
        exp-delta-x-over-D (* 0.47 (Math/pow B* (/ 2.0 3)))

        cv-delta-x         (/ std-delta-x-over-D exp-delta-x-over-D)
        sigma-x            (Math/sqrt (Math/log (+ 1.0 cv-delta-x)))
        exp-delta-x        (* exp-delta-x-over-D D)
        mu-x               (- (Math/log exp-delta-x)
                              (/ (Math/pow sigma-x 2)
                                 2))
        sigma-y            (* D 0.92)]
    {"$B^*$"                                B*
     ;; NOTE the following ended up being nonsensical:
     ;"E[ΔX] (m)" exp-delta-x
     "$\\text{CV}[\\Delta_X]$"              (str (format "%.0f" (* 100 cv-delta-x)) "%")
     ;"_X"       mu-x
     "$\\sigma_X$"                          (format "%.2f" sigma-x)
     "$\\sigma_Y/\\mathbb{E}[\\Delta_X]$" (format "%.2f" (/ sigma-y exp-delta-x))}))

(comment
  (pp/print-table
   (->> [20 50 100 150 200]
        (mapv himoto-eq-28-values)))
  ;| £B^*£ | £\text{CV}[\Delta_X]£ | £\sigma_X£ | £\sigma_Y/\mathbb{E}[\Delta_X]£ |
  ;|-------+-----------------------+------------+---------------------------------|
  ;|    20 |                   69% |       0.72 |                            0.27 |
  ;|    50 |                   51% |       0.64 |                            0.14 |
  ;|   100 |                   40% |       0.58 |                            0.09 |
  ;|   150 |                   35% |       0.55 |                            0.07 |
  ;|   200 |                   32% |       0.53 |                            0.06 |

  *e)
```

CAUTION (FIXME REVIEW): we have found the above formula to be problematic when applied to the parameters values found in (Sardoy et al., 2008), because it tends to yield nonsensical $\sigma_Y > \mathbb{E}[\Delta_X]$. In fact, it can be seen that $\sigma_Y > s\mathbb{E}[\Delta_X]$ if and only if $\sigma_X > \sqrt{\ln\left(1 + \frac{.88^2}{.92\times.47}s\right)}$, in particular $\sigma_X > 1.013$ for $s = 1$, which is unfortunately the case with the range of σ_X values in (Sardoy et al., 2008). This reflects a divergence between (Himoto and Tanaka, 2005) and (Sardoy et al., 2008), the latter allowing for more dispersed Δ_X distributions (higher coefficient of variation $\mathrm{Var}[\Delta_X]^{\frac{1}{2}}/\mathbb{E}[\Delta_X] = \sqrt{e^{\sigma_X^2} - 1}$), whereas the former typically predicts a low coefficient of variation, as is perceptible in Fig 6 of (Himoto and Tanaka, 2005). Fig 10 of (Sardoy et al., 2008) suggests coefficients of variation ranging from 95% to 210%, whereas Fig 6 of (Himoto and Tanaka, 2005) suggests coefficients of variation ranging from 30% to 70%. For this reason, we **strongly recommend to supply σ_Y directly** through the configuration key **:delta-y-sigma**.

Having computed the μ_X, σ_X and σ_Y parameters, it remains to draw values of Δ_X and Δ_Y by sampling from log-normal and normal distributions. This can be done by sampling from a standard normal distribution, then transforming by affine and exponential functions:

```
(ns gridfire.spotting
  (:require [gridfire.common              :refer [burnable-cell?
                                                  burnable-fuel-model?
                                                  calc-fuel-moisture
                                                  in-bounds-optimal?
                                                  terrain-distance-fn
                                                  terrain-distance-invoke]]
            [gridfire.conversion          :as convert]
            [gridfire.grid-lookup         :as grid-lookup]
            [gridfire.spotting.elmfire    :as spotting-elm]
            [gridfire.utils.random        :refer [my-rand-range]]
            [tech.v3.tensor               :as t]
            [vvvvalvalval.supdate.api     :as supd])
```

```
  (:import java.util.Random))
(set! *unchecked-math* :warn-on-boxed)

;;-------------------------------------------------------------------------
;; Formulas
;;-------------------------------------------------------------------------

(defn- sample-normal
  "Returns sample from normal/gaussian distribution given mu and sd."
  ^double
  [^Random rand-gen ^double mu ^double sd]
  (.nextGaussian rand-gen)
  (+ mu (* sd (.nextGaussian rand-gen))))

(defn- sample-lognormal
  "Returns sample from log-normal distribution given mu and sd."
  ^double
  [^Random rand-gen ^double mu ^double sd]
  (Math/exp (sample-normal rand-gen mu sd)))

(defn deltax-expected-value
  ^double [^double mu-x ^double sigma-x]
  (convert/m->ft (Math/exp (+ mu-x
                              (/ (Math/pow sigma-x 2)
                                 2.0)))))

(defn deltax-coefficient-of-variation
  ^double [^double sigma-x]
  (Math/sqrt (- (Math/exp (Math/pow sigma-x 2))
                1)))

;; NOTE might be turned into a multimethod.
(defn resolve-lognormal-params
  [spotting-config ^double fire-line-intensity ^double  wind-speed-20ft]
  (spotting-elm/resolve-lognormal-params spotting-config fire-line-intensity wind-speed-20ft))

(defn delta-x-sampler
  "Returns a function for randomly sampling ΔX, the spotting jump along the wind direction (in ft)."
  [spotting-config ^double fire-line-intensity ^double wind-speed-20ft]
  (let [ln-params (resolve-lognormal-params spotting-config fire-line-intensity wind-speed-20ft)
        mux       (:prob.lognormal/mu ln-params)
        sigmax    (:prob.lognormal/sigma ln-params)]
    (fn ^double draw-deltax-ft [rand-gen]
      (-> (sample-lognormal rand-gen mux sigmax)
          (convert/m->ft)))))

(def ^:private sigma-y-scalar-ft
  (* (convert/m->ft 1.0)
     0.92
     (/ 0.47
        (Math/pow 0.88 2))))

(defn himoto-resolve-default-sigma-y-from-lognormal-params
  ^double [^double mu-x ^double sigma-x]
  (let [es2h (Math/exp (/ (Math/pow sigma-x 2)
                          2))
        avg-deltax-m (* (Math/exp mu-x) es2h)]
    (* (double sigma-y-scalar-ft)
       avg-deltax-m
       (+ es2h 1.0)
       (- es2h 1.0))))

(comment
  ;; When will we have the default sigma_Y > E[ΔX]?
  ;; It can be seen that this nonsensical situation
  ;; happens iff sigma_X exceeds the following number:
  (Math/sqrt
   (Math/log (+ 1.0
```

```
               (/ (Math/pow 0.88 2)
                  (* 0.92 0.47)))))
  ;; =>
  1.0131023746492023

  *e)

(defn himoto-resolve-default-sigma-y
  ^double [spotting-config ^double fire-line-intensity ^double wind-speed-20ft]
  (let [ln-params (resolve-lognormal-params spotting-config fire-line-intensity wind-speed-20ft)
        mux       (:prob.lognormal/mu ln-params)
        sigmax    (:prob.lognormal/sigma ln-params)]
    (himoto-resolve-default-sigma-y-from-lognormal-params mux sigmax)))

(defn resolve-delta-y-sigma
  ^double [spotting-config ^double fire-line-intensity ^double wind-speed-20ft]
  (or (some-> (:delta-y-sigma spotting-config) (double))
      (himoto-resolve-default-sigma-y spotting-config fire-line-intensity wind-speed-20ft)))

(defn delta-y-sampler
  "Returns a function for randomly sampling ΔY, the spotting jump perpendicular to the wind direction (in ft)."
  [spotting-config ^double fire-line-intensity ^double wind-speed-20ft]
  (let [sigma-y (resolve-delta-y-sigma spotting-config fire-line-intensity wind-speed-20ft)]
    (fn draw-deltay-ft ^double [rand-gen]
      (convert/m->ft (sample-normal rand-gen 0 sigma-y)))))

(defn- sample-wind-dir-deltas
  "Draws a random sequence of [ΔX ΔY] pairs of signed distances (in ft) from the supplied cell,
  representing the coordinates of the spotting jump
  in the directions parallel and perpendicular to the wind.
  ΔX will typically be positive (downwind),
  and positive ΔY means to the right of the downwind direction."
  [inputs fire-line-intensity wind-speed-20ft]
  (let [spotting-config   (:spotting inputs)
        rand-gen          (:rand-gen inputs)
        fl-intensity      (convert/Btu-ft-s->kW-m fire-line-intensity)
        ws20ft            (convert/mph->mps wind-speed-20ft)
        sample-delta-x-fn (delta-x-sampler spotting-config fl-intensity ws20ft)
        sample-delta-y-fn (delta-y-sampler spotting-config fl-intensity ws20ft)
        ;; FIXME consider drawing num-firebrands from Poisson distribution, for consistency.
        num-firebrands    (long (:num-firebrands spotting-config))]
    (vec (repeatedly num-firebrands
                     (fn sample-delta-tuple []
                       [(sample-delta-x-fn rand-gen)
                        (sample-delta-y-fn rand-gen)])))))
```

5.6.5 Spot Ignition Model

The Spot Ignition model describes the probability of a spot ignition as well as when the spot ignition should occur. Perryman uses the method described in Schroeder (1969) but adjusts the result to take into account the distance a firebrand lands from the source tree (using Albini 1979) and the number of firebrands that land in a cell (using Stauffer 2008).

$$P(I)_l = P(I)\exp(-\lambda_s l)$$
$$P(I)_l^{FB} = 1 - (1 - P(I)_l)^{n_b}$$

where λ_s is a positive number representing the decay constant, l is the firebrand's landing distance away from the source cell. $P(I)_l$ is the probability of spot ignition taking into consideration of l. $P(I)_l^{FB}$ is the probability of spot fire ignition taking into consideration n_b, the number of firebrands landing in a cell. These formulas simply say that the firebrands are i.i.d causes of ignition (this is consistent with

the goal of making the model insensitive to grid resolution), with a per-firebrand probability that decays exponentially with distance (i.e. adding $1/\lambda_s$ to the distance divides the probability by e, or equivalently $l \mapsto P(I)_l$ has derivative $\lambda_s P(I)_l$).

```
(defn heat-of-preignition
  "Returns heat of preignition given:
   - Temperature: (Celsius)
   - Fine fuel moisture (0-1 ratio)

   Q_ig = 144.512 - 0.266*T_o - 0.00058 * (T_o)^2 - T_o * M + 18.54 * (1 - exp ( -15.1 * M ) ) + 640 * M  (eq. 10)"
  ^double
  [^double temperature ^double fine-fuel-moisture]
  (let [T_o temperature
        M   fine-fuel-moisture

        ;; heat required to reach ignition temperature
        Q_a (+ 144.512 (* -0.266 T_o) (* -0.00058 (Math/pow T_o 2.0)))

        ;; heat required to raise moisture to reach boiling point
        Q_b (* -1.0 T_o M)

        ;; Heat of desorption
        Q_c (* 18.54 (- 1.0 (Math/exp (* -15.1 M))))

        ;; Heat required to vaporize moisture
        Q_d (* 640.0 M)]
    (+ Q_a Q_b Q_c Q_d)))

(defn schroeder-ign-prob
  "Returns the probability of ignition as described in Shroeder (1969) given:
   - Temperature: (Celsius)
   - Fine fuel moisture (0-1 ratio)

   X = (400 - Q_ig) / 10
   P(I) = (0.000048 * X^4.3) / 50    (pg. 15)"
  ^double
  [^double temperature ^double fine-fuel-moisture]
  (let [Q_ig (heat-of-preignition temperature fine-fuel-moisture)
        X    (/ (- 400.0 Q_ig) 10.0)]
    (-> X
        (Math/pow 4.3)
        (* 0.000048)
        (/ 50.0)
        (Math/min 1.0)
        (Math/max 0.0))))

(defn- one-minus ^double [^double x] (- 1.0 x))

(defn spot-ignition-probability
  "Returns the probability of spot fire ignition (Perryman 2012) given:
   - Schroeder's probability of ignition [P(I)] (0-1)
   - Decay constant [lambda] (0.005)
   - Distance from the torched cell [d] (meters)
   - Number of firebrands accumulated in the cell [b]

   P(Spot Ignition) = 1 - (1 - (P(I) * exp(-lambda * d)))^b"
  ^double
  [^double ignition-probability ^double decay-constant ^double spotting-distance ^double firebrand-count]
  (-> decay-constant
      (* -1.0)
      (* spotting-distance)
      (Math/exp)
      (* ignition-probability)
      (one-minus)
      (Math/pow firebrand-count)
      (one-minus)))
```

A firebrand will cause an unburned cell to transition to a burned state if the cell receives at least one firebrand and the cell's probability of ignition as calculated by the above equations is greater than a randomly generated uniform number. Once a cell has been determined to ignite then the time until ignition is calculated. The time until ignition is a sum of three time intervals: the amount of time required for the firebrand to reach its maximum vertical height t_v, the amount of time required for the firebrand to descend from the maximum vertical height to the forest floor t_g, and the amount of time required for a spot fire to ignite and build up to the steady-state t_I. Perryman assumes t_v and t_g to be equal and used the formula from Albini (1979) to calculate it. t_I is also assumed to be 20 min as used in McAlpine and Wakimoto (1991).

```
(defn spot-ignition?
  [rand-gen ^double spot-ignition-probability]
  (let [random-number (my-rand-range rand-gen 0 1)]
    (>= spot-ignition-probability random-number)))

(defn albini-t-max
  "Returns the time of spot ignition using (Albini 1979) in minutes given:
   - Flame length: (m) [z_F]

   a = 5.963                                                    (D33)
   b = a - 1.4                                                  (D34)
   D = 0.003
   t_c = 1
   w_F = 2.3 * (z_F)^0.5                                        (A58)
   t_o = t_c / (2 * z_F / w_F)
   z =  0.39 * D * 10^5
   t_T = t_o + 1.2 + (a / 3) * ( ( (b + (z/z_F) )/a )^3/2 - 1 )  (D43)"
  ^double
  [^double flame-length]
  (let [a     5.963                          ; constant from (D33)
        b     4.563                          ; constant from (D34)
        z-max 117.0                          ; max height given particle diameter of 0.003m
        w_F   (* 2.3 (Math/sqrt flame-length)) ; upward axial velocity at flame tip
        t_0   (/ w_F (* 2.0 flame-length))]    ; period of steady burning of tree crowns (t_c, min) normalized by 2*z_F / w_F
    (-> z-max
        (/ flame-length)
        (+ b)
        (/ a)
        (Math/pow 1.5)
        (- 1.0)
        (* (/ a 3.0))
        (+ 1.2)
        (+ t_0))))

(defn spot-ignition-time
  "Returns the time of spot ignition using (Albini 1979) and (Perryman 2012) in minutes given:
   - Global clock: (min)
   - Flame length: (m)

   t_spot = clock + (2 * t_max) + t_ss"
  ^double
  [^double burn-time ^double flame-length]
  (let [t-steady-state 20.0] ; period of building up to steady state from ignition (min)
    (-> (albini-t-max flame-length)
        (* 2.0)
        (+ burn-time)
        (+ t-steady-state))))
```

Once the locations, ignition probabilities, and time of ignition has been calculated for each of the firebrands a sequence of key value pairs are returned, to be processed in 'gridfire.cli' (FIXME wat?). The key is [x y] location of the firebrand and the value [t p] where t is the time of ignition and p is the

ignition probability (FIXME that is, the highly confusing notion of probability in the sense of 'simulation pessimism', not the probability of ignition computed above).

```
(defn- update-firebrand-counts!
  [inputs firebrand-count-matrix fire-spread-matrix source firebrands]
  (let [num-rows                (:num-rows inputs)
        num-cols                (:num-cols inputs)
        get-fuel-model          (:get-fuel-model inputs)
        [i j]                   source
        source-burn-probability (grid-lookup/mget-double-at fire-spread-matrix (long i) (long j))]
    (doseq [[y x] firebrands]
      (when (burnable-cell? get-fuel-model
                            fire-spread-matrix
                            source-burn-probability
                            num-rows
                            num-cols
                            y
                            x)
        (->> (grid-lookup/mget-double-at firebrand-count-matrix y x)
             (long)
             (inc)
             (t/mset! firebrand-count-matrix y x))))))

(defn- in-range?
  [[min max] fuel-model-number]
  (<= min fuel-model-number max))

(defn- intranges-mapping-lookup
  "Looks up a value in a mapping from fuel number to anything,
  encoded as either a single value v (constant mapping),
  or as a vector of [[min-fuel-number max-fuel-number] v] pairs,
  such as:
  [[[1 140]   0.0]
   [[141 149] 1.0]
   [[150 256] 1.0]]"
  [intranges-mapping fuel-model-number]
  (if (vector? intranges-mapping)
    ;; IMPROVEMENT for performance, we could do a non-sequential lookup, (Val, 02 Nov 2022)
    ;; e.g. a dichotomic search,
    ;; or even better just (aget) an array into which we have indexed the decompressed mapping.
    (loop [irm-entries intranges-mapping]
      (when-let [[fuel-range v] (first irm-entries)]
        (if (in-range? fuel-range fuel-model-number)
          v
          (recur (rest irm-entries)))))
    intranges-mapping))

(defn surface-fire-spot-fire?
  "Expects surface-fire-spotting config to be a sequence of tuples of
  ranges [lo hi] and spotting probability. The range represents the range (inclusive)
  of fuel model numbers that the spotting probability is set to.
  [[[1 140] 0.0]
  [[141 149] 1.0]
  [[150 256] 1.0]]"
  [inputs [i j] ^double fire-line-intensity]
  (let [rand-gen                    (:rand-gen inputs)
        get-fuel-model              (:get-fuel-model inputs)
        fuel-model-number           (long (grid-lookup/double-at get-fuel-model i j))
        surface-fire-spotting       (:surface-fire-spotting (:spotting inputs))
        critical-fire-line-intensity (-> (:critical-fire-line-intensity surface-fire-spotting)
                                         (intranges-mapping-lookup fuel-model-number)
                                         (or 0.0)
                                         (double))]
    (when (and surface-fire-spotting
               (> fire-line-intensity critical-fire-line-intensity))
      (let [spot-percent (-> (:spotting-percent surface-fire-spotting)
                             (intranges-mapping-lookup fuel-model-number)
```

```
                            (or 0.0)
                            (double))]
        (>= spot-percent (my-rand-range rand-gen 0.0 1.0))))))

(defn crown-spot-fire?
  "Determine whether crowning causes spot fires. Config key `:spotting` should
   take either a vector of probabilities (0-1) or a single spotting probability."
  [inputs]
  (when-some [spot-percent (:crown-fire-spotting-percent (:spotting inputs))] ; WARNING 'percent' is misleading. (Val, 17 Mar 2023)
    (let [rand-gen (:rand-gen inputs)
          p        (double spot-percent)]
      (>= p (my-rand-range rand-gen 0.0 1.0)))))

(defn- spot-fire? [inputs crown-fire? here fire-line-intensity]
  (if crown-fire?
    (crown-spot-fire? inputs)
    (surface-fire-spot-fire? inputs here fire-line-intensity)))

;; FIXME: Drop cell = [i j]
(defn spread-firebrands
  "Returns a sequence of key value pairs where
  key: [x y] locations of the cell
  val: [t p] where:
  t: time of ignition
  p: ignition-probability"
  [inputs matrices i j]
  (let [num-rows                  (:num-rows inputs)
        num-cols                  (:num-cols inputs)
        cell-size                 (:cell-size inputs)
        rand-gen                  (:rand-gen inputs)
        spotting                  (:spotting inputs)
        get-elevation             (:get-elevation inputs)
        get-fuel-model            (:get-fuel-model inputs)
        get-temperature           (:get-temperature inputs)
        get-relative-humidity     (:get-relative-humidity inputs)
        get-wind-speed-20ft       (:get-wind-speed-20ft inputs)
        get-wind-from-direction   (:get-wind-from-direction inputs)
        get-fuel-moisture-dead-1hr (:get-fuel-moisture-dead-1hr inputs)
        firebrand-count-matrix    (:firebrand-count-matrix matrices)
        fire-spread-matrix        (:fire-spread-matrix matrices)
        fire-line-intensity-matrix (:fire-line-intensity-matrix matrices)
        flame-length-matrix       (:flame-length-matrix matrices)
        fire-type-matrix          (:fire-type-matrix matrices)
        burn-time-matrix          (:burn-time-matrix matrices)
        burn-time                 (grid-lookup/mget-double-at burn-time-matrix i j)
        i                         (long i)
        j                         (long j)
        cell                      [i j]
        fire-line-intensity       (grid-lookup/mget-double-at fire-line-intensity-matrix i j)
        crown-fire?               (-> fire-type-matrix (grid-lookup/mget-double-at i j) (double) (> 1.0))]
    (when (spot-fire? inputs crown-fire? cell fire-line-intensity)
      (let [band                   (long (/ burn-time 60.0))
            ws                     (grid-lookup/double-at get-wind-speed-20ft band i j)
            wd                     (grid-lookup/double-at get-wind-from-direction band i j)
            deltas                 (sample-wind-dir-deltas inputs fire-line-intensity ws)
            wind-to-direction      (mod (+ 180 wd) 360)
            firebrands             (firebrands deltas wind-to-direction cell cell-size)
            source-burn-probability (grid-lookup/mget-double-at fire-spread-matrix i j)
            terrain-dist-fn        (terrain-distance-fn get-elevation num-rows num-cols cell-size)]
        (update-firebrand-counts! inputs firebrand-count-matrix fire-spread-matrix cell firebrands)
        (->> (for [[x y] firebrands]
               (let [x (long x)
                     y (long y)]
                 (when (and
                        (in-bounds-optimal? num-rows num-cols x y)
                        (burnable-fuel-model? (grid-lookup/double-at get-fuel-model x y)))
                   (let [temperature          (grid-lookup/double-at get-temperature band x y)
                         fine-fuel-moisture   (if get-fuel-moisture-dead-1hr
```

```
                                        (grid-lookup/double-at get-fuel-moisture-dead-1hr band x y)
                                        (calc-fuel-moisture
                                         (grid-lookup/double-at get-relative-humidity band i j)
                                         temperature :dead :1hr))
                    ignition-probability (schroeder-ign-prob (convert/F->C (double temperature)) fine-fuel-moisture)
                    decay-constant       (double (:decay-constant spotting))
                    spotting-distance    (convert/ft->m (terrain-distance-invoke terrain-dist-fn i j x y))
                    firebrand-count      (t/mget firebrand-count-matrix x y)
                    spot-ignition-p      (spot-ignition-probability ignition-probability
                                                                    decay-constant
                                                                    spotting-distance
                                                                    firebrand-count)
                    burn-probability     (* spot-ignition-p source-burn-probability)]
                (when (and (>= burn-probability 0.1)   ; TODO parametrize 0.1 in gridfire.edn
                           (> (double burn-probability) (grid-lookup/mget-double-at fire-spread-matrix x y))
                           (spot-ignition? rand-gen spot-ignition-p))
                  (let [t (spot-ignition-time burn-time
                                              (convert/ft->m (grid-lookup/mget-double-at flame-length-matrix i j)))]
                    [[x y] [t burn-probability]])))))))
          (remove nil?))))))
```

5.6.6 Sampling of parameters

To represent uncertainty about the physical models, the spotting parameters can be configured to be drawn randomly from a uniform distribution, at the beginning of each simulation:

```
(defn- sample-spotting-param
  ^double
  [param rand-gen]
  (if (and (map? param) (contains? param :lo) (contains? param :hi))
    (let [{:keys [lo hi]} param
          ;; FIXME REVIEW Why on Earth are we sampling from a range with random bounds? (Val, 17 Mar 2023)
          l               (if (vector? lo) (my-rand-range rand-gen (lo 0) (lo 1)) lo)
          h               (if (vector? hi) (my-rand-range rand-gen (hi 0) (hi 1)) hi)]
      (my-rand-range rand-gen l h))
    param))

(defn- sample-from-uniform
  "Draws a random number from a Uniform Distribution,
  encoded as either a single number (no randomness, a.k.a. Constant Distribution)
  or a [min max] range."
  ^double [values-range rand-gen]
  (cond
    (number? values-range) (double values-range)
    (vector? values-range) (let [[min max] values-range]
                             (my-rand-range rand-gen min max))))

(defn- sample-intranges-mapping-values
  [pairs rand-gen]
  (if (sequential? pairs)
    (supd/supdate pairs [{1 #(sample-from-uniform % rand-gen)}])
    pairs))

(defn sample-spotting-params
  "Resolves values for the spotting parameters which are configured as a range to draw from.

  Given a GridFire spotting configuration map,
  replaces those parameters which are configured as a range
  with randomly drawn values from that range,
  returning a new map."
  [spotting-config rand-gen]
  (-> spotting-config
      ;; Yes, it' a mess, that's what we get for having made the configuration so irregular. (Val, 17 Mar 2023)
      (as-> sp-params
            (reduce (fn [sp k]
```

```
                  (supd/supdate sp {k sample-spotting-param}))
                sp-params
                [:num-firebrands
                 :decay-constant
                 :mean-distance
                 :normalized-distance-variance
                 :flin-exp
                 :ws-exp
                 :delta-y-sigma])
          (supd/supdate sp-params
                        {:crown-fire-spotting-percent #(some-> % (sample-from-uniform rand-gen))
                         :surface-fire-spotting       {:critical-fire-line-intensity sample-intranges-mapping-values
                                                       :spotting-percent             sample-intranges-mapping-values}})))))
```

5.7 Suppression Model Formulas

GridFire supports two ways to compute the fraction of the fire perimeter that should be contained during a suppression event.

5.7.1 Suppression Curve

Fraction Contained (FC) is computed by

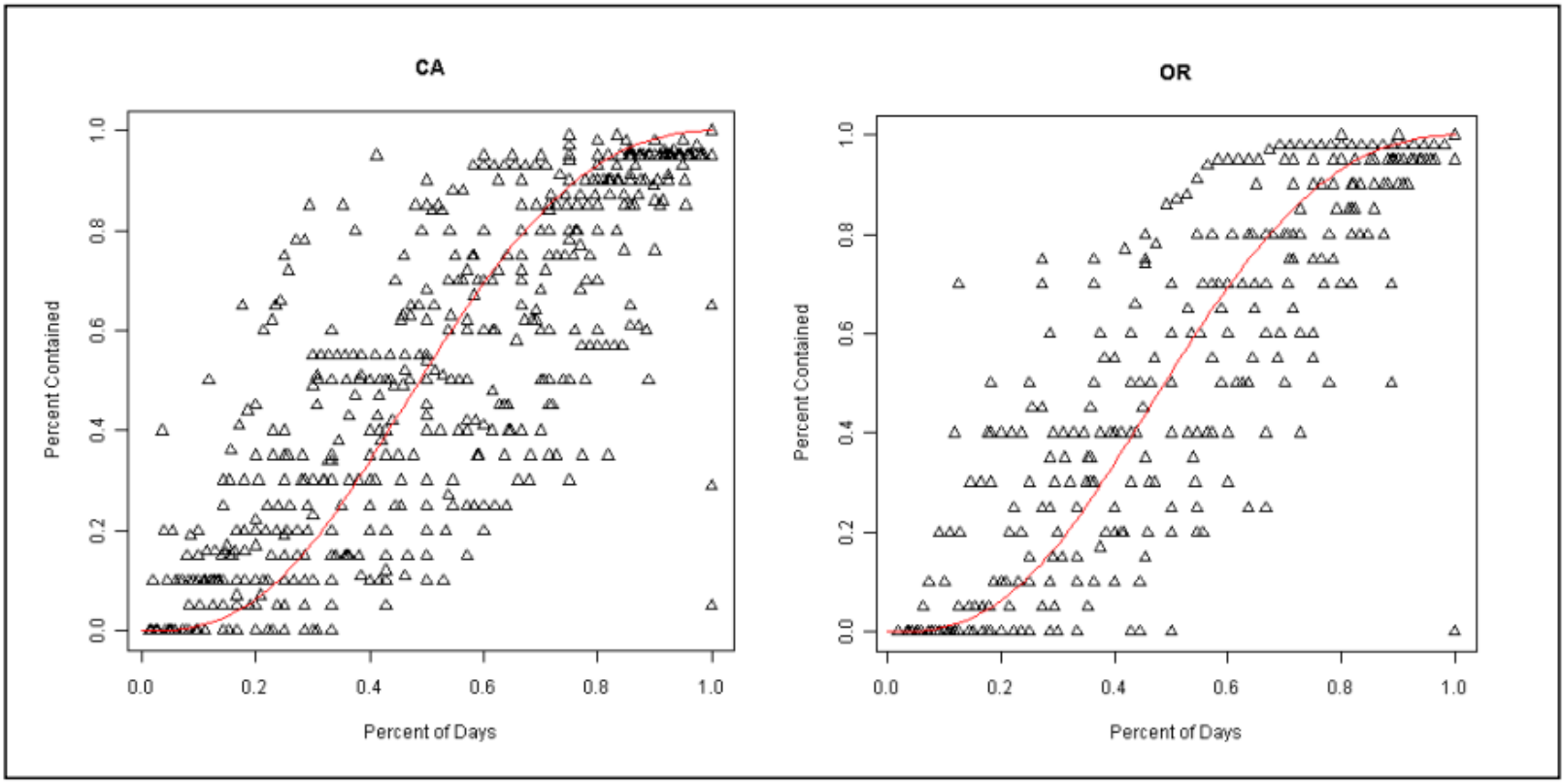
$$FC = (\frac{2x}{1+x^2})^a$$

Where symbols are defined as follows:

a alpha (a.k.a **:suppression-coefficient** in the config file) a positive number, As alpha approaches infinity, most of the suppression happens sharply near the end; as it approaches 0, most of the suppression happens sharply near the beginning. Values above 0 make the curve sigmoid-shaped.

x The runtime-fraction (ratio) x at which a fraction y of the perimeter got suppressed is an increasing function of yˆ(1/alpha); in particular, it's an increasing function of both y and alpha.

Here are two example plots of the above equation. For most of the regressions, the value of alpha has been between 2.0 and 3.0 as for California and Oregon respectively.



5.7.2 Suppression Difficulty Index (SDI)

The SDI approach computes change in fraction contained ΔC during time interval Δt (in days) between suppression events.

$$\frac{\Delta C}{\Delta t} = \chi \times f\left(1 - \frac{\ln(A_d/A_\chi)}{\ln(A_{d0}/A_\chi)}, B \times \mathrm{SDI}_{\Delta t}\right)$$

$$f(U,V) := \begin{cases} U\exp(-V), & U \geq 0 \\ U\exp(V), & U < 0 \end{cases}$$

Where symbols are defined as follows:

A_d (ac/day) Areal growth on a daily basis.

A_χ (ac/day, a.k.a **sdi-reference-areal-growth-rate** in source code) **reference areal-growth-rate** parameter, currently hardcoded to 1 ac/day. The areal-growth-rate A_d at which $\frac{\Delta C}{\Delta t} = C \times \chi$ in 0-SDI settings.

χ (%/day) "Chi" (a.k.a, **:sdi-reference-suppression-speed** in the config file) a positive number. Higher means faster evolution of the fraction contained. χ^{-1} is a characteristic time of containment-variation. This calibration parameter is similar to the suppression curve calibration coefficient in that it varies regionally based on available resources. In a reference situation where $A_d = A_\chi$ and SDI = 0, a value of 100 means the fire can be 100% contained in 24 hours, and a value of 800 means the fire can be 100% contained in 3 hours.

A_{d0} (ac/day, a.k.a **:sdi-containment-overwhelming-area-growth-rate** in config file) **containment-overwhelming areal growth rate** parameter. For $A_d > A_{d0}$, the variation in containment $\Delta C/\Delta t$ becomes negative.

B (SDI^{-1}, a.k.a **:sdi-sensitivity-to-difficulty** in config file) **sensitivity to difficulty** parameter. Higher means that suppression behavior is more sensitive to changes in SDI. Changing the terrain's SDI by B^{-1} causes an e-folding of the containment-variation $\Delta C/\Delta t$, all else being equal.

$\text{SDI}_{\Delta t}$ is calculated as the mean suppression difficulty index during interval Δt.

$$\overline{SDI_{\Delta t}} = \frac{1}{n} \times \sum_{i=1}^{n} SDI_i$$

Where symbols are defined as follows:

n Number of burned pixels during time interval Δt.

i Pixel index.

SDI_i Suppression difficulty index of pixel i. The value of this will come from a SDI raster layer.

The $\left(1 - \frac{\ln(A_d/A_\chi)}{\ln(A_{d0}/A_\chi)}\right)$ factor accounts for the effect of fire-growth-rate on containment: fast-growing fires are harder to contain, and very fast-growing fires ($A_d > A_{d0}$) make containment a losing battle.

The $\exp(\pm B \times \text{SDI}_{\Delta t})$ factor captures the effect of the terrain's suppression-difficulty on containment: it is a factor by which the containement-variation-speed $\Delta C/\Delta t$ is penalized or amplified due to suppression-difficulty.

The suppression algorithm was built using the method described in "resources/suppression_curve_algorithm.pdf".

```
(ns gridfire.suppression
  "An algorithm emulating human interventions reacting to fire spread
  by suppressing ('putting out') chosen contiguous segments of the
  fire front, typically backing and flanking fires."
  (:require [gridfire.conversion :refer [cells->acres
                                         min->day
                                         percent->dec
                                         rad->deg]]))

(set! *unchecked-math* :warn-on-boxed)

(defn- combine-average ^double
  [^double avg-old ^long count-old ^double avg-new ^long count-new]
  (if (zero? (+ count-old count-new))
    0.0
    (/ (+ (* avg-old count-old)
          (* avg-new count-new))
       (+ count-old count-new))))

(defn- remove-average ^double
  [^double avg-old ^long count-old ^double avg-to-remove ^long count-of-avg-to-remove]
  (if (zero? (- count-old count-of-avg-to-remove))
    0.0
    (/ (- (* avg-old count-old) (* avg-to-remove count-of-avg-to-remove))
       (- count-old count-of-avg-to-remove))))

(defn- compute-contiguous-slices
  "Given number of cells to suppress and a map of average directional spread
  rate data with the form: angular-slice -> [average-dsr cell-count] return a
```

```
  sorted map where each map entry:

  [[`list-of-slices` `avg-dsr`] `cell-count`]

  represents a contiguous segment of the fire front, which we locate
  by `list-of-slices`, the list of successive degree slices covering
  it; `cell-count` represents the number of active perimeter cells in
  that segment, with `cell-count` no smaller than
  `num-cells-to-suppress`, but possibly bigger.

  NOTE: This constraint may be violated if a segment is adjacent to an
  already suppressed slice, in which case the segment will be included
  in the returned map even if its `cell-count` is smaller than
  `num-cells-to-suppress`.

  Note that the returned segments will tend to overlap - think of a
  sliding window of (up to `num-cells-to-suppress`) contiguous active
  cells, rotating around the centroid: the segments returned by this
  function are regular snapshots of this window."
  [^long num-cells-to-suppress angular-slice->avg-dsr+num-cells]
  (loop [sorted-contiguous-slices (sorted-map-by (fn [[_ x1] [_ x2]]
                                                   (compare x1 x2)))
         slice-data               (into [] (seq angular-slice->avg-dsr+num-cells))
         cur-contiguous-slices    []
         cur-dsr                  0.0
         cur-count                0
         left-idx                 -1
         right-idx                0]
    (cond

      (= left-idx 0)
      sorted-contiguous-slices

      ;; Do not include already suppressed regions in the longest
      ;; contiguous slice calculation.
      (let [[_ [_ cell-count]] (nth slice-data right-idx)
            cell-count         (long cell-count)]
        (and (< cur-count num-cells-to-suppress) (zero? cell-count)))
      (let [next-right-idx (if (= right-idx (dec (count slice-data)))
                             0
                             (+ right-idx 1))]
        (recur (if (seq cur-contiguous-slices)
                 (assoc sorted-contiguous-slices [cur-contiguous-slices cur-dsr] cur-count)
                 sorted-contiguous-slices)
               slice-data
               []
               0.0
               0
               (if (< right-idx left-idx) 0 next-right-idx)
               next-right-idx))

      (< cur-count num-cells-to-suppress)
      ;; expand right
      (let [[slice [avg-dsr cell-count]] (nth slice-data right-idx)
            cell-count                   (long cell-count)]
        (recur sorted-contiguous-slices
               slice-data
               (conj cur-contiguous-slices slice)
               (combine-average cur-dsr cur-count avg-dsr cell-count)
               (+ cur-count cell-count)
               left-idx
               (if (= right-idx (dec (count slice-data)))
                 0
                 (+ right-idx 1))))

      :else
      ;; shrink left
      (let [[_ [avg-dsr cell-count]] (nth slice-data (if (= -1 left-idx) 0 left-idx))
```

```
            cell-count               (long cell-count)]
        (recur (assoc sorted-contiguous-slices [cur-contiguous-slices cur-dsr] cur-count)
               slice-data
               (subvec cur-contiguous-slices 1)
               (remove-average cur-dsr cur-count avg-dsr cell-count)
               (- cur-count cell-count)
               (long
                (cond
                  (= left-idx (dec (count slice-data))) 0
                  (= -1 left-idx)                       1
                  :else                                 (+ left-idx 1)))
               right-idx)))))

(defn- compute-sub-segment
  [angular-slice->avg-dsr+num-cells slices cells-needed]
  (let [slices                           (set slices)
        angular-slice->avg-dsr+num-cells (reduce (fn [acc [slice avg-dsr+num-cells]]
                                                   (if (contains? slices slice)
                                                     (assoc acc slice avg-dsr+num-cells)
                                                     (assoc acc slice [0.0 0.0]))) ;; Needed because the segment should not be treat
                                                 (sorted-map)
                                                 angular-slice->avg-dsr+num-cells) ;; FIXME: This seems inefficient
        contiguous-slices                (compute-contiguous-slices cells-needed angular-slice->avg-dsr+num-cells)
        [[slices _] cell-count]          (first contiguous-slices)]
    [slices cell-count]))

(defn- compute-slices-to-suppress
  "Chooses slices to be suppressed, and reports the number of suppressed cells.

  Given:
  - num-cells-to-suppress: a number of cells, the suppression objective,
  - angular-slice->avg-dsr+num-cells: a map of statistics over all angular slices,

  returns a tuple [slices-to-suppress suppressed-count], in which:
  - slices-to-suppress is the set of angular slices to suppress, chosen as a tradeoff
  between contiguity, closeness to the num-cells-to-suppress objective,
  and low average spread rate.
  - suppressed-count is the number of cells in slices-to-suppress,
  which is returned to save callers the work of re-computing it.
  Warning: it may well be that suppressed-count > num-cells-to-suppress.

  This algorithm will convert `angular-slice->avg-dsr+num-cells` to a sorted map of
  `angular-slices+avg-dsr->num-cells`, representing candidate segments for suppression.
  Using this map the algorithm will collect the sequence of angular-slices
  until we have a cell-count of at least `num-cells-to-suppress`, if possible."
  [^long num-cells-to-suppress angular-slice->avg-dsr+num-cells]
  ;; NOTE this algorithm is most likely under-optimized;
  ;; having said that, it's probably not a performance bottleneck
  ;; (suppression events are typically few and far between)
  ;; and experience has shown that we'd better make this right
  ;; before making it fast.
  ;; TODO enhance performance or rethink the overall suppression algorithm.
  (letfn [(n-cells-in-slices ^long [slices]
            (transduce (map (fn n-cells-in-slice [slice]
                              (let [[_ num-cells] (get angular-slice->avg-dsr+num-cells slice)]
                                num-cells)))
                       (completing +)
                       0
                       slices))]
    (let [angular-slices+avg-dsr->num-cells (compute-contiguous-slices num-cells-to-suppress angular-slice->avg-dsr+num-cells)]
      (loop [remaining-segments angular-slices+avg-dsr->num-cells
             n-cells-needed     num-cells-to-suppress
             slices-to-suppress #{}]
        (if-some [segment (when (pos? n-cells-needed)
                            (first remaining-segments))]
          (let [[[slices _]]            segment
                yet-unsuppressed-slices (remove slices-to-suppress slices)
                n-would-be-suppressed   (long (n-cells-in-slices yet-unsuppressed-slices))]
```

```
            (if (<= n-would-be-suppressed n-cells-needed)
              (let [new-n-cells-needed     (- n-cells-needed n-would-be-suppressed)
                    new-slices-to-suppress (into slices-to-suppress yet-unsuppressed-slices)]
                (recur (rest remaining-segments)
                       new-n-cells-needed
                       new-slices-to-suppress))
              ;; this segment has more than we need, compute subsegment:
              (let [[sub-segment-slices _] (compute-sub-segment angular-slice->avg-dsr+num-cells slices n-cells-needed)
                    n-more-suppressed      (long (n-cells-in-slices (remove slices-to-suppress sub-segment-slices)))
                    new-n-cells-needed     (- n-cells-needed n-more-suppressed)
                    new-slices-to-suppress (into slices-to-suppress sub-segment-slices)]
                (recur (rest remaining-segments)
                       new-n-cells-needed
                       new-slices-to-suppress))))
          ;; no more segments needed or available, so we return:
          (let [n-suppressed (- num-cells-to-suppress n-cells-needed)]
            [slices-to-suppress n-suppressed])))))

(defn- average
  [coll]
  (/ (double (reduce + coll)) (long (count coll))))

(defn- compute-avg-dsr
  [burn-vectors]
  (-> (reduce (fn ^double [^double acc burn-vector]
                (+ acc (double (:spread-rate burn-vector))))
              0.0
              burn-vectors)
      double
      (/ (count burn-vectors))))

(defn- compute-cell-count
  [burn-vectors]
  (count
   (into #{}
         (map (juxt :i :j))
         burn-vectors)))

(defn- compute-avg-dsr-data
  "Returns a sorted map where each map entry:

  [`angular-slice` [`directional-spread-rate` `cell-count`]]

  represents a collection of stats computed for an `angular-slice`. The
  `directional-spread-rate` is the average value among the active
  perimeter cells that fall within that slice. The `cell-count` is the
  count of those perimeter cells."
  [^double angular-slice-size slice->BurnVectors]
  (reduce (fn [acc slice]
            (let [burn-vectors (get slice->BurnVectors slice)]
              (if (seq burn-vectors)
                (assoc acc slice [(compute-avg-dsr burn-vectors) (compute-cell-count burn-vectors)])
                (assoc acc slice [0.0 0.0]))))
          (sorted-map)
          (range 0.0 (/ 360.0 angular-slice-size))))

(defn- angle-cw-from-east ^double
  [^long i1 ^long j1 ^long i0 ^long j0]
  (let [di    (- i1 i0)
        dj    (- j1 j0)
        theta (rad->deg (Math/atan2 di dj))]
    (if (neg? di)
      (+ theta 360.0)
      theta)))

(defn- nearest-angular-slice ^double
  [^double theta ^double angular-slice-size]
  (Math/floor (/ theta angular-slice-size)))
```

```
(defn- group-burn-vectors
  "Returns a map where each entry:

  [`angular-slice` [BurnVector BurnVector ...]]

  represents a collection of BurnVectors that fall within an `angular-slice`.
  The `angular-slice` is defined as the degree clockwise from EAST of the
  `centroid` cell. angular-slice 0 = East = 0.0 degrees."
  [centroid ^double angular-slice-size burn-vectors]
  (let [[i0 j0] centroid]
    (group-by (fn [burn-vector] (-> (angle-cw-from-east (:i burn-vector) (:j burn-vector) i0 j0)
                                    (nearest-angular-slice angular-slice-size)))
              burn-vectors)))

(defn- compute-centroid-cell
  "Returns [i j] that is the centroid of a given list of [i j] cells"
  [cells]
  (let [row (average (mapv #(nth % 0) cells))
        col (average (mapv #(nth % 1) cells))]
    [(long row) (long col)]))

(defn- compute-suppression-difficulty-factor ^double
  [^double sdi-sensitivity-to-difficulty ^double change-in-fraction-contained-sign-multiplier ^double mean-sdi]
  (double
   (if (>= change-in-fraction-contained-sign-multiplier 0.0)
     (Math/exp (* -1.0 sdi-sensitivity-to-difficulty mean-sdi))
     (Math/exp (* sdi-sensitivity-to-difficulty mean-sdi)))))

(defn- compute-mean-sdi ^double
  [get-suppression-difficulty-index ignited-cells]
  (/ (double
      (reduce (fn ^double [^double acc [i j]]
                (+ acc (double (get-suppression-difficulty-index i j))))
              0.0
              ignited-cells))
     (count ignited-cells)))

(defn- compute-area-growth-rate ^double
  [^double cell-size ^double suppression-dt ignited-cells-since-last-suppression]
  (/ (cells->acres cell-size (count ignited-cells-since-last-suppression))
     (min->day suppression-dt)))

(def ^:const sdi-reference-areal-growth-rate
  "[ac/day] a shape parameter for the suppression curve, the area-growth-rate A_d at which ΔC/Δt=χ in 0-SDI settings,
  in which χ is the :sdi-reference-suppression-speed parameter."
  1.0)

(defn- compute-change-in-fraction-contained-sign-multiplier ^double
  [^double sdi-containment-overwhelming-area-growth-rate ^double area-growth-rate]
  (- 1.0
     ;; Note that the following ratio is insensitive to the choice of logarithm base.
     (/ (Math/log (/ area-growth-rate
                     sdi-reference-areal-growth-rate))
        (Math/log (/ sdi-containment-overwhelming-area-growth-rate
                     sdi-reference-areal-growth-rate)))))

(defn- compute-fraction-contained-sdi
  "Compute the updated fraction contained using suppression difficulty index algorithm"
  [inputs ignited-cells-since-last-suppression ^double previous-fraction-contained]
  (let [cell-size                                  (double (:cell-size inputs))
        get-suppression-difficulty-index           (:get-suppression-difficulty-index inputs)
        suppression-dt                             (double (:suppression-dt inputs))
        sdi-containment-overwhelming-area-growth-rate (double (:sdi-containment-overwhelming-area-growth-rate inputs))
        sdi-sensitivity-to-difficulty              (double (:sdi-sensitivity-to-difficulty inputs))
        sdi-reference-suppression-speed            (double (:sdi-reference-suppression-speed inputs))
        area-growth-rate                           (compute-area-growth-rate cell-size suppression-dt ignited-cells-since-last-s
        change-in-fraction-contained-sign-multiplier  (compute-change-in-fraction-contained-sign-multiplier sdi-containment-overwhe
```

```
                                                                                                area-growth-rate)
        mean-sdi                                    (compute-mean-sdi get-suppression-difficulty-index ignited-cells-since-last-s
        suppression-difficulty-factor               (compute-suppression-difficulty-factor sdi-sensitivity-to-difficulty
                                                                                           change-in-fraction-contained-sign-mult
                                                                                           mean-sdi)
        change-in-fraction-contained                (-> (* sdi-reference-suppression-speed
                                                           change-in-fraction-contained-sign-multiplier
                                                           suppression-difficulty-factor)
                                                        (percent->dec)
                                                        (* (min->day suppression-dt)))]
    (max 0.0 (+ previous-fraction-contained change-in-fraction-contained))))

(defn- compute-fraction-contained-sc
  "Compute fraction contained using suppression curve algorithm"
  ^double
  [^double max-runtime-fraction ^double suppression-coefficient]
  (Math/pow (/ (* 2.0 max-runtime-fraction)
               (+ 1.0 (Math/pow max-runtime-fraction 2.0)))
            suppression-coefficient))

(defn suppress-burn-vectors
  [inputs
   max-runtime-fraction
   previous-num-perimeter-cells
   previous-suppressed-count
   burn-vectors
   ignited-cells-since-last-suppression
   previous-fraction-contained]
  (let [max-runtime-fraction         (double max-runtime-fraction)
        suppression-coefficient      (:suppression-coefficient inputs)
        previous-num-perimeter-cells (long previous-num-perimeter-cells)
        previous-suppressed-count    (long previous-suppressed-count)
        active-perimeter-cells       (into #{}
                                           (map (juxt :i :j))
                                           burn-vectors)
        fraction-contained           (if suppression-coefficient
                                       (compute-fraction-contained-sc max-runtime-fraction
                                                                      (double suppression-coefficient))
                                       (compute-fraction-contained-sdi inputs
                                                                       ignited-cells-since-last-suppression
                                                                       previous-fraction-contained))
        num-tracked-perimeter-cells  (+ (long (count active-perimeter-cells)) previous-suppressed-count)
        num-fizzled-perimeter-cells  (max 0 (- previous-num-perimeter-cells num-tracked-perimeter-cells))
        num-perimeter-cells          (max previous-num-perimeter-cells num-tracked-perimeter-cells)
        current-suppressed-count     (+ previous-suppressed-count num-fizzled-perimeter-cells)
        next-suppressed-count        (long (* ^double fraction-contained num-perimeter-cells))
        num-cells-to-suppress        (- next-suppressed-count current-suppressed-count)]
    (if (> num-cells-to-suppress 0)
      (let [centroid-cell          (compute-centroid-cell active-perimeter-cells)
            angular-slice-size     5.0
            slice->BurnVectors     (group-burn-vectors centroid-cell angular-slice-size burn-vectors)
            [slices-to-suppress
             suppressed-count]     (->> (compute-avg-dsr-data angular-slice-size slice->BurnVectors)
                                        (compute-slices-to-suppress num-cells-to-suppress))
            slices-to-suppress-set (set slices-to-suppress)
            slices-to-keep         (remove #(contains? slices-to-suppress-set %) (keys slice->BurnVectors))
            burn-vectors-to-keep   (into []
                                         (mapcat #(get slice->BurnVectors %))
                                         slices-to-keep)]
        [burn-vectors-to-keep (+ current-suppressed-count ^long suppressed-count) num-perimeter-cells fraction-contained])
      [burn-vectors current-suppressed-count num-perimeter-cells fraction-contained])))
```

6 User Interface

The GridFire model described in the previous section may be called directly from the REPL through the **run-fire-spread** function. However, this would require that the user had already prepared all of their map layers as 2D Clojure core.matrix values. In order to enable GridFire to easily access a wide range of raster formatted GIS layers directly, we have the following options:

1. A simple Clojure interface to a Postgresql database, containing the PostGIS spatial extensions. This interface is described in Section 6.1.

2. Magellan, a Clojure library for interacting with geospatial datasets. This interface is described in Section 6.2.

Section 6.3 describes GridFire's command line interface along with its input configuration file format, which allows users to select between the PostGIS and Magellan data import options easily.

Using one of these options along with a simple client interface in clojure Section 6.3 which describes GridFire's command line interface along with its input configuration file format.

6.1 PostGIS Bridge

Extracting raster layers from a PostGIS database is performed by a single function, called **postgis-raster-to-matrix**, which constructs a SQL query for the layer, sends it to the database in a transaction, and returns the result as a core.matrix 2D double array with nodata values represented as -1.0. The georeferencing information associated with this tile is also included in the returned results. This function may be called directly from the REPL or indirectly through GridFire's command line interface.

```
(ns gridfire.postgis-bridge
  (:require [clojure.java.jdbc :as jdbc]
            [hikari-cp.core    :as h]
            [tech.v3.datatype  :as d]
            [tech.v3.tensor    :as t])
  (:import java.util.UUID
           org.postgresql.jdbc.PgArray))

(set! *unchecked-math* :warn-on-boxed)

(defn extract-matrix [result]
  (->> result
       :matrix
       (#(.getArray ^PgArray %)) ; Note: I can pass a HashMap of {String,Class}
       t/->tensor                ; to automatically convert SQL types to Java types.
       (d/emap #(or % -1.0) nil) ; FIXME: is this step necessary?
       d/clone))                 ; What happens to null values when read?

(defn build-rescale-query [rescaled-table-name resolution table-name]
  (format (str "CREATE TEMPORARY TABLE %s "
               "ON COMMIT DROP AS "
               "SELECT ST_Rescale(rast,%s,-%s,'NearestNeighbor') AS rast "
               "FROM %s")
          rescaled-table-name
          resolution
          resolution
          table-name))

(defn build-threshold-query [threshold]
  (format (str "ST_MapAlgebra(rast,band,NULL,"
               "'CASE WHEN [rast.val] < %s"
               " THEN 0.0 ELSE [rast.val] END')")
          threshold))
```

```
(defn build-data-query [threshold threshold-query metadata table-name]
  (format (str "SELECT ST_DumpValues(%s,%s) AS matrix "
               "FROM generate_series(1,%s) AS band "
               "CROSS JOIN %s")
          (if threshold threshold-query "rast")
          (if threshold 1 "band")
          (:numbands metadata)
          table-name))

(defn build-meta-query [table-name]
  (format "SELECT (ST_Metadata(rast)).* FROM %s" table-name))

(defn- parse-subname [subname]
  {:database (re-find #"(?<=\/)[\w]*$" subname)
   :port     (re-find #"(?<=:)[0-9]*[0-9](?=\/)" subname)
   :server   (re-find #"(?<=\/)[\w]*(?=:)" subname)})

(defn build-datasource-options [{:keys [user password subname]}]
  (let [{:keys [database port server]} (parse-subname subname)]
      {:auto-commit        true
       :read-only          false
       :connection-timeout 30000
       :validation-timeout 5000
       :idle-timeout       600000
       :max-lifetime       1800000
       :minimum-idle       10
       :maximum-pool-size  10
       :pool-name          "db-pool"
       :adapter            "postgresql"
       :username           user
       :password           password
       :database-name      database
       :server-name        server
       :port-number        port
       :register-mbeans    false}))

(defonce db-pool-cache (atom nil))

(defn make-db-pool [db-spec]
  (or @db-pool-cache
      (reset! db-pool-cache (h/make-datasource (build-datasource-options db-spec)))))

(defn close-db-pool []
  (h/close-datasource @db-pool-cache)
  (reset! db-pool-cache nil))

(defmacro with-db-connection-pool [db-spec & body]
  `(if-let [db-spec# ~db-spec]
     (let [_#      (make-db-pool db-spec#)
           result# (do ~@body)]
       (close-db-pool)
       result#)
     (do ~@body)))

(defn postgis-raster-to-matrix
  "Send a SQL query to the PostGIS database given by db-spec for a
  raster tile from table table-name. Optionally resample the raster to
  match resolution and set any values below threshold to 0. Return the
  post-processed raster values as a Clojure matrix using the
  core.matrix API along with all of the georeferencing information
  associated with this tile in a hash-map with the following form:
  {:srid 900916,
   :upperleftx -321043.875,
   :upperlefty -1917341.5,
   :width 486,
   :height 534,
   :scalex 2000.0,
```

```
   :scaley -2000.0,
   :skewx 0.0,
   :skewy 0.0,
   :numbands 10,
   :matrix #vectorz/matrix Large matrix with shape: [10,534,486]}"
  [db-spec table-name & [resolution threshold]]
  (jdbc/with-db-transaction [conn {:datasource (make-db-pool db-spec)}]
    (let [table-name      (if-not resolution
                            table-name
                            (let [rescaled-table-name (str "gridfire_" (subs (str (UUID/randomUUID)) 0 8))
                                  rescale-query       (build-rescale-query rescaled-table-name resolution table-name)]
                              ;; Create a temporary table to hold the rescaled raster.
                              ;; It will be dropped when the transaction completes.
                              (jdbc/db-do-commands conn [rescale-query])
                              rescaled-table-name))
          meta-query      (build-meta-query table-name)
          metadata        (first (jdbc/query conn [meta-query]))
          threshold-query (build-threshold-query threshold)
          data-query      (build-data-query threshold threshold-query metadata table-name)
          matrix          (when-let [results (seq (jdbc/query conn [data-query]))]
                            (if (= (count results) 1)
                              (extract-matrix (first results))
                              (t/->tensor (mapv extract-matrix results))))]
      (assoc metadata :matrix matrix))))
```

6.2 Magellan

Reading raster layers from disk is performed by a single function, called **geotiff-raster-to-tensor**. Given the location of a GeoTIFF file, this function will read the raster into memory and return the same map of information as the **postgis-raster-to-matrix** function, described in the previous section.

```
(ns gridfire.magellan-bridge
  (:require [clojure.java.io         :as io]
            [magellan.core           :refer [read-raster register-new-crs-definitions-from-properties-file!]]
            [magellan.raster.inspect :as inspect]
            [tech.v3.tensor          :as t])
  (:import org.geotools.coverage.grid.GridGeometry2D
           org.geotools.referencing.operation.transform.AffineTransform2D))

(set! *unchecked-math* :warn-on-boxed)

(defn register-custom-projections! []
  (register-new-crs-definitions-from-properties-file! "CUSTOM" (io/resource "custom_projections.properties")))

(defn geotiff-raster-to-tensor
  "Reads a raster from a file using the magellan.core library. Returns the
   post-processed raster values as a Clojure tensor using the dtype-next/tech.v3.tensor API
   along with all of the georeferencing information associated with this tile in a
   hash-map with the following form:
  {:upperleftx -321043.875,
   :upperlefty -1917341.5,
   :width 486,
   :height 534,
   :scalex 2000.0,
   :scaley -2000.0,
   :skewx 0.0,
   :skewy 0.0,
   :numbands 10,
   :matrix #tech.v3.tensor<datatype>[10 534 486]}"
  [file-path & [datatype convert-fn]]
  (let [raster   (read-raster file-path)
        grid     ^GridGeometry2D (:grid raster)
        image    (inspect/describe-image (:image raster))
        envelope (inspect/describe-envelope (:envelope raster))
        crs2d    ^AffineTransform2D (.getGridToCRS2D grid)]
```

```
   {:upperleftx (get-in envelope [:x :min])
    :upperlefty (get-in envelope [:y :max])
    :width      (:width image)
    :height     (:height image)
    :scalex     (.getScaleX crs2d)
    :scaley     (.getScaleY crs2d)
    :skewx      0.0 ; FIXME not used?
    :skewy      0.0 ; FIXME not used?
    :numbands   (:bands image)
    :matrix     (inspect/extract-tensor raster :datatype datatype :convert-fn convert-fn)}))
```

6.3 Command Line Interface

The entire GridFire system is available for use directly from the Clojure REPL. This enables straightforward analysis and introspection of the fire behavior functions and their results over a range of inputs. However, if you just want to simulate an individual ignition event, GridFire comes with a simple command line interface that can be parameterized by a single configuration file, specifying the ignition location, burn duration, weather values, and the location of the PostGIS raster layers to use for topography and fuels.

GridFire's command line interface can be built as an uberjar using the following command:

```
clojure -X:make-uberjar
```

The advantage of the uberjar format is that the single uberjar file can be shared easily between computers and can be run by anyone with a recent version of Java installed, without needing to install Clojure, Git, or any of the dependency libraries that GridFire uses.

The command above will output the uberjar into this repository's top level "target" directory. It can be run from the command line as follows:

```
java -jar gridfire.jar myconfig.edn
```

When run, the executable connects to the PostGIS database specified in the passed-in config file, downloads the necessary raster layers, simulates the ignition event for the requested duration, and returns 2D maps showing the spatial distributions of fire spread, flame length, and fire line intensity respectively. Finally, it prints out the final clock time from when the simulation was terminated as well as the total number of ignited cells on the raster grid at that point.

Which maps are created (and in what formats) may be configured by setting the following options in GridFire's input config file to true or false:

1. :output-landfire-inputs?
2. :output-geotiffs?
3. :output-pngs?

```
(ns gridfire.core
  (:require [clojure.spec.alpha          :as spec]
            [gridfire.fire-spread        :refer [memoize-rfwo rothermel-fast-wrapper-optimal]]
            [gridfire.inputs             :as inputs]
            [gridfire.outputs            :as outputs]
            [gridfire.simulations        :as simulations]
            [gridfire.spec.config        :as config-spec]
            [gridfire.utils.async        :as gf-async]
            [gridfire.utils.files        :as files]
            [manifold.deferred           :as mfd]
            [taoensso.tufte              :as tufte]
            [triangulum.logging          :refer [log log-str]]))
```

```
(set! *unchecked-math* :warn-on-boxed)

(defn write-outputs!
  [outputs]
  (->
   (mfd/zip
    (outputs/write-landfire-layers! outputs)
    (outputs/write-aggregate-layers! outputs)
    (outputs/write-csv-outputs! outputs))
   (deref))
  :success)

(defmacro with-multithread-profiling
  [& body]
  `(do (tufte/remove-handler! :accumulating)
       (let [stats-accumulator# (tufte/add-accumulating-handler! {:handler-id :accumulating})
             result#            (do ~@body)]
         (when simulations/*log-performance-metrics*
           (Thread/sleep 1000))
         (as-> {:format-pstats-opts {:columns [:n-calls :min :max :mean :mad :clock :total]}} $#
           (tufte/format-grouped-pstats @stats-accumulator# $#)
           (when simulations/*log-performance-metrics*
             (log $# :truncate? false)))
         result#)))

(defn run-simulations!
  [{:keys [^long simulations parallel-strategy] :as inputs}]
  (with-multithread-profiling                              ; TODO: Disable this to see how much performance is gained.
    (log-str "Running simulations")
    (let [sfmin-memoization (get-in inputs [:memoization :surface-fire-min] :across-sims)
          pmap-fn           (if (= parallel-strategy :between-fires)
                              (fn [f coll]
                                (->> coll
                                     (gf-async/pmap-in-n-threads (or (:max-parallel-simulations inputs) ; INTRO :max-parallel-simul
                                                                     (.availableProcessors (Runtime/getRuntime)))
                                                                 f)
                                     (deref)))
                              ;; NOTE the optimization behind {:parallel-strategy :within-fires} is no longer implemented, so we def
                              map)
          summary-stats     (with-redefs [rothermel-fast-wrapper-optimal (if (= sfmin-memoization :across-sims)
                                                                           (memoize-rfwo rothermel-fast-wrapper-optimal)
                                                                           rothermel-fast-wrapper-optimal)]
                              ;; NOTE :across-sims is useful to share the memo across simulations, with a risk of running out of mem
                              ;; WARNING: omitting :memoization {} is not equivalent to :memoization {:surface-fire-min nil}, but to
                              (->> (range simulations)
                                   (pmap-fn (fn run-ith-simulation [i]
                                              (simulations/run-simulation! i inputs)))
                                   (remove nil?)
                                   (vec)))]
      (assoc inputs :summary-stats summary-stats))))

(defn load-inputs!
  [config]
  (-> config
      (inputs/add-input-layers)
      (inputs/add-misc-params)
      (inputs/add-ignition-csv)
      (inputs/add-sampled-params)
      (inputs/add-perturbation-params)
      (inputs/add-weather-params)
      (inputs/add-fuel-moisture-params)
      (inputs/add-random-ignition-sites)
      (inputs/add-aggregate-matrices)
      (inputs/add-ignition-start-times)
      (inputs/add-ignition-start-timestamps)
      (inputs/add-burn-period-samples)
      (inputs/add-suppression)
      (inputs/add-fuel-number->spread-rate-adjustment-array-lookup-samples)))
```

```
(defn load-config-or-throw!
  [config-file-path]
  (let [config (files/read-situated-edn-file config-file-path)]
    (if (spec/valid? ::config-spec/config config)
      (assoc config :config-file-path config-file-path)
      (throw (ex-info (format "Invalid config file [%s]:\n%s"
                              config-file-path
                              (spec/explain-str ::config-spec/config config))
                      {::config           config
                       ::spec-explanation (spec/explain-data ::config-spec/config config)})))))

(defn load-config!
  [config-file-path]
  (try
    (load-config-or-throw! config-file-path)
    (catch Exception err
      (log-str (ex-message err)))))

(defn process-config-file!
  [config-file-path]
  (try
    (some-> config-file-path
            (load-config!)
            (load-inputs!)
            (run-simulations!)
            (write-outputs!))
    (catch Exception e
      (log-str (ex-message e)))))
```

```
(ns gridfire.cli
  (:gen-class)
  (:require [clojure.core.async             :refer [<!!]]
            [clojure.edn                    :as edn]
            [clojure.java.io                :as io]
            [clojure.tools.cli              :refer [parse-opts]]
            [gridfire.core                  :as gridfire]
            [gridfire.magellan-bridge       :refer [register-custom-projections!]]
            [gridfire.server.pyrecast-async :as server]
            [gridfire.server.sync           :refer [start-with-cli-args!]]
            [gridfire.utils.server          :refer [hostname? nil-on-error]]))

(set! *unchecked-math* :warn-on-boxed)

(def pyrecast-server-cli-options
  [["-c" "--server-config CONFIG" "Server config file"
    :validate [#(.exists  (io/file %)) "The provided --server-config does not exist."
               #(.canRead (io/file %)) "The provided --server-config is not readable."]]

   ["-h" "--host HOST" "Host domain name"
    :validate [hostname? "The provided --host is invalid."]]

   ["-p" "--port PORT" "Port number"
    :parse-fn #(if (int? %) % (Integer/parseInt %))
    :validate [#(< 0 % 0x10000) "The provided --port is not a number between 0 and 65536."]]])

(def program-banner
  (str "gridfire: Launch fire spread simulations via config files or in server mode.\n"
       "Copyright © 2014-2022 Spatial Informatics Group, LLC.\n"))

(defn run-config-files!
  [args]
  (register-custom-projections!)
  (doseq [config-file args]
    (gridfire/process-config-file! config-file)))

(defn run-default!
```

```
  "CLI execution when no verb is provided. Much of this is here for backwards-compatibility."
  [args]
  (let [{:keys [options arguments summary errors]} (parse-opts args pyrecast-server-cli-options)]
    ;; {:options   The options map, keyed by :id, mapped to the parsed value
    ;;  :arguments A vector of unprocessed arguments
    ;;  :summary   A string containing a minimal options summary
    ;;  :errors    A vector of error message strings thrown during parsing; nil when no errors exist
    (cond
      ;; Errors encountered during input parsing
      (seq errors)
      (do
        (run! println errors)
        (println (str "\nUsage:\n" summary))
        (System/exit 1))

      ;; Server mode invoked
      (every? options [:server-config :host :port])
      (if-let [config-file-params (nil-on-error (edn/read-string (slurp (:server-config options))))]
        (do
          (register-custom-projections!)
          (<!! (server/start-server! (merge config-file-params (dissoc options :server-config)))))
        (do
          (println (:server-config options) "does not contain well-formed EDN.")
          (println (str "\nUsage:\n" summary))
          (System/exit 1)))

      ;; CLI mode invoked
      (seq arguments)
      (run-config-files! args)

      ;; Incorrect CLI invocation
      :else
      (do
        (->> ["Usage:"
              "1) For running one or more GridFire config files, use sub-command:"
              "run-config-files /path/to/first/config.edn [/path/to/second/config.edn ...]"
              "2) For starting a synchronous socket server, use sub-command:"
              "start-sync-socket-server --port MY-PORT-NUMBER"
              (format "%s will be printed to standard out once the server is ready."
                      (pr-str :gridfire.server.sync/ready-to-accept-connections))]
             (run! println))
        (System/exit 1)))))

(defn -main [& args]
  (println program-banner)
  (case (first args)
    "start-sync-socket-server" (start-with-cli-args! (rest args))
    "run-config-files"         (run-config-files! (rest args))
    (run-default! args))
  ;; Exit cleanly
  (System/exit 0))
```

```
(ns gridfire.utils.random
  (:import (java.util ArrayList Collection Collections Random)))

(defn my-rand
  (^double [^Random rand-generator] (.nextDouble rand-generator))
  (^double [^Random rand-generator n] (* n (.nextDouble rand-generator))))

(defn my-rand-int
  ^long
  [rand-generator n]
  (long (my-rand rand-generator n)))

(defn my-rand-nth
  [rand-generator coll]
  (nth coll (my-rand-int rand-generator (count coll))))
```

```
(defn my-rand-range
  ^double
  [rand-generator min-val max-val]
  (let [range (- max-val min-val)]
    (+ min-val (my-rand rand-generator range))))

(defn- sample-from-list
  [rand-generator n xs]
  (repeatedly n #(my-rand-nth rand-generator xs)))

(defn- sample-from-range
  [rand-generator n min-val max-val]
  (repeatedly n #(my-rand-range rand-generator min-val max-val)))

(defn draw-samples
  [rand-generator n x]
  (into []
        (cond (list? x)   (sample-from-list rand-generator n x)
              (vector? x) (sample-from-range rand-generator n (x 0) (x 1))
              :else       (repeat n x))))

(defn my-shuffle
  [^Random rand-gen ^Collection coll]
  (if (< (count coll) 2)
    (if (vector? coll)
      coll
      (vec coll))
    (let [al (ArrayList. coll)]
      (Collections/shuffle al rand-gen)
      (vec (.toArray al)))))
```

6.4 Server Interface

GridFire runs on a the Java Virtual Machine, a code interpreter designed to learn how to run the current program faster by monitoring its own execution, in particular through the use of Just-In-Time compilation (JIT). These efficiency gains can only be reaped by long-running programs, which is why GridFire features running in server mode.

Concretely, GridFire offers a synchronous server which can be called through TCP. That server can be started from the GridFire CLI. For convenience and illustration, GridFire provides a client script in Clojure for calling a running server:

```
#!/usr/bin/env bb
(ns gridfire.server.sync.call
  "A zero-deps script for invoking a running `gridfire.server.sync`.

  More precisely, for executing run-config commands

  Can be run via clojure -M / java -classpath / babashka / babashka --classpath / ...
  see the (comment ...) at the end of this file."
  (:require [clojure.edn     :as edn]
            [clojure.java.io :as io]
            [clojure.pprint  :refer [pprint]]
            [clojure.string  :as str])
  (:import (java.io PrintWriter PushbackReader)
           (java.net Socket))
  (:gen-class))

(defn call-sync-server-with-run-config!
  [^String host ^String port ^String config-file]
  (with-open [client-socket (Socket. host (int (Integer/parseInt port 10)))
              in            (PushbackReader. (io/reader (.getInputStream client-socket)))
              out           (PrintWriter. (io/writer (.getOutputStream client-socket)) true)]
```

```
    (.println out (str/join " " ["run-config" config-file]))
    (edn/read {:default tagged-literal} ;; This makes the EDN parsing tolerant of any tagged literal, which lets this program convey
              in)))

;; NOTE why not make this a subpath of gridfire.cli? Because:
;; 1) that would add a big startup time (loading all of GridFire's codebase);
;; 2) that would make it impossible to run through Babashka.
(defn -main [host port config-file]
  (let [result (call-sync-server-with-run-config! host port config-file)]
    (if (:gridfire.run-config/succeeded result)
      (do
        (pprint result)
        (System/exit 0))
      (binding [*out* *err*]
        (pprint result)
        (System/exit 1)))))

;; Inspiration: https://book.babashka.org/#main_file
(when (= *file* (System/getProperty "babashka.file"))
  (apply -main *command-line-args*))

(comment
  ;; HOW TO RUN THIS SCRIPT (assuming a running gridfire.server.sync on localhost:8085):
  ;; clojure -M -m gridfire.server.sync.call                                  localhost 8085 path/to/my/gridfire.edn
  ;; bb src/gridfire/server/sync/call.clj                                     localhost 8085 path/to/my/gridfire.edn
  ;; java -classpath my-gridfire-uberjar.jar gridfire.server.sync.call        localhost 8085 path/to/my/gridfire.edn
  ;; bb --classpath my-gridfire-uberjar.jar --main gridfire.server.sync.call localhost 8085 path/to/my/gridfire.edn

  *e)
```

6.4.1 Implementation Code

```
(ns gridfire.server.protocols)

(defprotocol JobHandler
  (schedule-command [this command =notifications-channel=] "Schedules the given command to be processed.
  Returns a Manifold Deferred of the processing result.
  =notifications-channel= must be a core.async channel, which will receive progress notifications.")
  (n-queued [this] "Returns the number of commands currently waiting to be processed.")
  (halt [this] "Terminates the logical process handling commands."))
```

```
(ns gridfire.server.sync
  "A very basic, mono-threaded socket-based API into a GridFire server."
  (:require [clojure.core.async         :as async]
            [clojure.java.io            :as io]
            [clojure.string             :as str]
            [clojure.tools.cli          :as clj-cli]
            [gridfire.server.protocols  :as server-protocols]
            [gridfire.server.run-config :refer [start-run-config-handler!]]
            [triangulum.logging         :refer [log-str]])
  (:import (java.io BufferedReader PrintWriter)
           (java.net ServerSocket Socket)))

(def help-message-lines
  ["This socket server lets you call an already-running GridFire program, thus avoiding the slowness of cold starts."
   "Usage: send a line formatted as one of"
   "(1) run-config PATH_TO_MY_GRIDFIRE_CONFIG_FILE"
   "Runs simulations from a GridFire config file."
   "Upon success, you will see below a result like #:gridfire.run-config{:succeeded true}."
   "(2) help"
   "Shows this help message."
   ""
   "(Do not write the (1)/(2)/... at the beginning of the line.)"
   "The responses to your commands are EDN-encoded. This line starts with ;; because it is an EDN comment."
   "Calling programs can thus read the results as a stream of EDN data."])
```

```
;; NOTE we're using core.async as a wrapper for line-based IO.

(defn- =print-lines-channel=
  "Returns a core.async channel `=chan=` wrapping the given PrintWriter `dest`,
  so that the Strings put into `=chan=` will result in printing to `dest`."
  [^PrintWriter dest]
  (let [=ret= (async/chan)]
    (async/pipeline-blocking 1
                             (async/dropping-buffer 0)      ; this channel will receive nothing, it's only here to make async/pipeli
                             (comp (mapcat (fn [^String s]
                                             (str/split-lines s)))
                                   (mapcat (fn [^String l]
                                             (.println dest l)
                                             ;; HACK so that core.async does not complain of receiving nil.
                                             [])))
                             =ret=)
    =ret=))

(defn- print-help-message!
  [=notifications-channel=]
  (->> help-message-lines
       (str/join "\n")
       (async/>!! =notifications-channel=)))

(defn- run-config-file!
  [run-config-handler args =notifications-channel=]
  (let [[gridfire-config-path] args
        n-queued               (server-protocols/n-queued run-config-handler)
        _                      (async/>!! =notifications-channel= (format "Scheduled for processing (behind %s queued items): %s"
                                                                          (pr-str n-queued)
                                                                          (pr-str gridfire-config-path)))
        completion-dfr         (server-protocols/schedule-command run-config-handler gridfire-config-path =notifications-channel=)]
    (try
      (let [success-data @completion-dfr]
        (into {:gridfire.run-config/succeeded true} success-data))
      (catch Exception err
        (let [err-map (or (try
                            (Throwable->map err)
                            (catch Exception _parsing-err nil))
                          {:message (str "(FAILED Throwable->map) " (ex-message err))})]
          {:gridfire.run-config/succeeded  false
           :gridfire.run-config/error-data err-map})))))

(defn- handle-input-line!
  [run-config-handler ^String input-line =results-channel= =notifications-channel=]
  (let [[cmd & args] (str/split input-line #"\s+")]
    (case cmd
      "run-config" (async/>!! =results-channel=
                              (run-config-file! run-config-handler args =notifications-channel=))
      (print-help-message! =notifications-channel=))))

(def xform-results->edn
  "A transducer transforming values to EDN."
  (map (fn edn-encode-safe [res]
         (try
           (pr-str res)
           (catch Exception err
             (pr-str (tagged-literal 'gridfire.server.sync/failed-printing-result
                                     {::ex-class-name (.getName (class err))
                                      ::ex-message    (ex-message err)})))))))

(def xform-notifs->edn-comments
  "A transducer for a sequence of notification Strings,
  which converts them to lines of EDN comments."
  (comp (mapcat str/split-lines)
        (map (fn to-edn-comment [^String notif-line]
               (str ";; " notif-line)))))
```

```
(defn- listen-to-client!
  [run-config-handler ^Socket client-socket]
  (with-open [client-socket client-socket
              out           (PrintWriter. (io/writer client-socket) true)
              in            (io/reader client-socket)]
    (let [=out=                    (=print-lines-channel= out)
          =results-channel=        (async/chan 8)
          =notifications-channel= (async/chan 8)]
      ;; Piping results and notifications to the text output:
      (async/pipeline 1
                      =out=
                      xform-results->edn
                      =results-channel=
                      false)
      (async/pipeline 1
                      =out=
                      xform-notifs->edn-comments
                      =notifications-channel=
                      false)
      (loop []
        (when-some [l (.readLine ^BufferedReader in)]
          (handle-input-line! run-config-handler l =results-channel= =notifications-channel=)
          (recur))))))

(defn- listen! [^long port]
  (let [run-config-handler (start-run-config-handler!)]
    (with-open [server-socket (ServerSocket. (int port))]
      (prn ::ready-to-accept-connections)                    ;; So that calling programs know when to connect.
      (log-str (str `gridfire.server.sync ": ready to accept connection on port " port "."))
      (loop []
        (let [client-socket (.accept server-socket)]
          (async/thread (listen-to-client! run-config-handler client-socket))
          (recur))))))

(comment

  (future (listen! 8085))

  *e)

(def cli-options
  [["-p" "--port PORT" "Port number"
    :parse-fn #(if (int? %) % (Long/parseLong % 10))
    :validate [#(< 0 % 0x10000) "The provided --port is not a number between 0 and 65536."]]])

(defn start-with-cli-args! [args]
  (let [parsed-opts (clj-cli/parse-opts args cli-options :strict true)
        errors      (concat (:errors parsed-opts)
                            (->> {:port "--port"}
                                 (keep (fn error-missing-required-option [[k opt-name]]
                                         (when-not (get (:options parsed-opts) k)
                                           (format "Missing required option: %s" opt-name)))))))]
    (if (seq errors)
      (do
        (run! println errors)
        (println (str "\nUsage: " "start-sync-socket-server" "\n" (:summary parsed-opts)))
        (System/exit 1))
      (listen! (:port (:options parsed-opts))))))

(comment
  ;; Example client code in JVM Clojure:

  (require '[clojure.java.shell :as sh])
  (require '[clojure.edn])
  (sh/sh "clojure" "-M:run" "start-sync-socket-server" "--port" "8085")
  ;;:gridfire.server.sync/ready-to-accept-connections
  ;;01/10 16:47:06 gridfire.server.sync: ready to accept connection on port 8085.
```

```
  (def in
    (let [client-socket (java.net.Socket. "localhost" (int 8085))
          in            (io/reader (.getInputStream client-socket))
          out           (PrintWriter. (io/writer (.getOutputStream client-socket)) true)]
      (.println out "help")
      (.println out "run-config test/gridfire/lab/benchmarks/rhino-input-deck/gridfire.edn")
      in))

  (.readLine in)
  ;;=>
  ";; This socket server lets you call an already-running GridFire program, thus avoiding the slowness of cold starts."

  (clojure.edn/read (java.io.PushbackReader. in))
  ;;=>
  #:gridfire.run-config{:succeeded true}

  *e)
```

```
(ns gridfire.server.run-config
  "Sequential processing of run-config requests through an in-memory job queue."
  (:require [clojure.core.async        :as async]
            [gridfire.core             :as gridfire]
            [gridfire.magellan-bridge  :refer [register-custom-projections!]]
            [gridfire.server.protocols :as server-protocols]
            [manifold.deferred         :as mfd])
  (:import (java.util.concurrent BlockingQueue LinkedBlockingDeque)))

;; NOTE why make this async job handler, when currently the only calling code is a synchronous server? Several reasons:
;; 1. even with several clients connected to the sync server, this ensures that only 1 config is run at a time;
;; 2. we might evolve to provide new access points into a running GridFire server, for example an async HTTP server.

(defn- schedule-command
  [^BlockingQueue queue config-path =notifications-channel=]
  (let [completion-dfr (mfd/deferred)]
    (.put queue [config-path =notifications-channel= completion-dfr])
    completion-dfr))

(defn- n-queued
  [^BlockingQueue queue]
  (.size queue))

(defrecord RunConfigHandler
  [queue halt-callback]
  server-protocols/JobHandler
  (schedule-command [_this command =notifications-channel=]
   (schedule-command queue command =notifications-channel=))
  (n-queued [_this]
   (n-queued queue))
  (halt [_this] (halt-callback)))

(defn- run-gridfire-config!
  [config-path =notifications-channel= completion-dfr]
  (try
    (let [_       (async/>!! =notifications-channel= (format "Starting to run-config: %s" (pr-str config-path)))
          config  (gridfire/load-config-or-throw! config-path)
          _       (async/>!! =notifications-channel= "Parsed and validated the config. Loading inputs...")
          inputs  (gridfire/load-inputs! config)
          _       (async/>!! =notifications-channel= "Running simulations...")
          outputs (gridfire/run-simulations! inputs)]
      (async/>!! =notifications-channel= "Writing outputs...")
      (gridfire/write-outputs! outputs)
      (async/>!! =notifications-channel= "... done.")
      (mfd/success! completion-dfr
                    ;; This map might get enriched in the future. (Val, 11 Jan 2023)
                    {}))
    (catch Exception err
```

```
      (mfd/error! completion-dfr err))))

(defn start-run-config-handler!
  "Starts a logical process which will process run-config commands sequentially.

  Returns a `gridfire.server.protocols/RunConfigHandler`, for which:
  - a `command` is a GridFire config path (a String);
  - =notifications-channel= will receive human-readable String messages."
  []
  (register-custom-projections!)
  (let [queue (LinkedBlockingDeque.)                        ; Using LinkedBlockingDeque as a BlockingQueue implementation because it
        ;; NOTE why use a queue instead of a core.async channel? Because we want to be able
        ;; to query the number of elements awaiting processing.
        *keep-handling (atom true)]
    (async/thread
     (while @*keep-handling
       (let [[config-path =notifications-channel= completion-dfr] (.take queue)]
         (run-gridfire-config! config-path =notifications-channel= completion-dfr))))
    (->RunConfigHandler queue (fn halt [] (reset! *keep-handling false)))))
```

7 Configuration File

The configuration file for GridFire's command line interface is a text file in Extensible Data Notation (EDN) format.[5] A sample configuration file is provided below and in "resources/sample_config.edn". The format should be self-evident at a glance, but it is worth noting that EDN is case-sensitive but whitespace-insensitive. Comments are anything following two semi-colons (;;). Strings are contained in double-quotes (""). Keywords are prefixed with a colon (:). Vectors are delimited with square brackets ([]). Associative lookup tables (a.k.a. maps) are delimited with curly braces ({}) and are used to express key-value relationships.

The configuration file can be broken up into 5 sections as described below:

7.1 Section 1: Landscape data to be shared by all simulations

GridFire allows us to choose how we want to ingest landscape data through the configuration file. We can choose to get LANDFIRE layers from our PostGIS database, or we can read raster files from disk. This behavior is controlled as follows:

Include the following mapping at the top level of the configuraiton file:

- **landfire-layers**: a map of fetch specifications

For the fetch specifications include the following mappings:

- **type**: the method for fetching the layer
- **source**: the string input for the fetch method

To fetch layers from a Postgresql database you must also include the follwing mapping:

- **db-spec**: a map of database connection information for our Postgresql database

Here's an example of fetching LANDFIRE layers from a Postgresql database.

[5] https://github.com/edn-format/edn

```
{:db-spec            {:classname   "org.postgresql.Driver"
                      :subprotocol "postgresql"
                      :subname     "//localhost:5432/gridfire"
                      :user        "gridfire"}
 :landfire-layers    {:aspect              {:type   :postgis
                                            :source "landfire.asp WHERE rid=100"}
                      :canopy-base-height {:type   :postgis
                                            :source "landfire.cbh WHERE rid=100"}
                      :canopy-cover       {:type   :postgis
                                            :source "landfire.cc WHERE rid=100"}
                      :canopy-height      {:type   :postgis
                                            :source "landfire.ch WHERE rid=100"}
                      :crown-bulk-density {:type   :postgis
                                            :source "landfire.cbd WHERE rid=100"}
                      :elevation          {:type   :postgis
                                            :source "landfire.fbfm40 WHERE rid=100"}
                      :fuel-model         {:type   :postgis
                                            :source "landfire.slp WHERE rid=100"}
                      :slope              {:type   :postgis
                                            :source "landfire.dem WHERE rid=100"}}}
```

Here's an example of fetching LANDFIRE layers from files on disk using relative paths to where the gridfire.edn resides.

```
{:landfire-layers {:aspect              {:type   :geotiff
                                         :source #gridfire.utils.files/from-this-file "./asp.tif"}
                   :canopy-base-height {:type   :geotiff
                                         :source #gridfire.utils.files/fromj-this-file "./cbh.tif"}
                   :canopy-cover       {:type   :geotiff
                                         :source #gridfire.utils.files/fromj-this-file "./cc.tif"}
                   :canopy-height      {:type   :geotiff
                                         :source #gridfire.utils.files/fromj-this-file "./ch.tif"}
                   :crown-bulk-density {:type   :geotiff
                                         :source #gridfire.utils.files/fromj-this-file "./cbd.tif"}
                   :elevation          {:type   :geotiff
                                         :source #gridfire.utils.files/fromj-this-file "./dem.tif"}
                   :fuel-model         {:type   :geotiff
                                         :source #gridfire.utils.files/fromj-this-file "./fbfm40.tif"}
                   :slope              {:type   :geotiff
                                         :source #gridfire.utils.files/fromj-this-file "./slp.tif"}}}
```

Here's an example of fetching LANDFIRE layers from files on disk using full file paths.

```
{:landfire-layers {:aspect              {:type   :geotiff
                                         :source "test/gridfire/resources/asp.tif"}
                   :canopy-base-height {:type   :geotiff
                                         :source "test/gridfire/resources/cbh.tif"}
                   :canopy-cover       {:type   :geotiff
                                         :source "test/gridfire/resources/cc.tif"}
                   :canopy-height      {:type   :geotiff
                                         :source "test/gridfire/resources/ch.tif"}
                   :crown-bulk-density {:type   :geotiff
                                         :source "test/gridfire/resources/cbd.tif"}
                   :elevation          {:type   :geotiff
                                         :source "test/gridfire/resources/dem.tif"}
                   :fuel-model         {:type   :geotiff
                                         :source "test/gridfire/resources/fbfm40.tif"}
                   :slope              {:type   :geotiff
                                         :source "test/gridfire/resources/slp.tif"}}}
```

Gridfire uses imperial units for its calculations. Gridfire optionally allows us to use LANDFIRE LAYERS in different units and scale.

To specify the need for conversion from metric to imperial, include the following mapping in the fetch specifications:

- **units**: keyword :metric

```
{:canopy-height {:type   :geotiff
                 :source "test/gridfire/resources/weather-test/ch.tif"
                 :units  :metric}}
```

To specify a scaling factor, include the following mapping in the fetch specifications:

- **multiplier**: int or float

```
{:canopy-height {:type       :geotiff
                 :source     "test/gridfire/resources/weather-test/ch.tif"
                 :multiplier 0.1}}
```

In the example above the input raster's units are meters * 10.[6] Thus a value of 5 on the canopy height grid layer is actually 0.5 meters. The mutipler factor needed to convert to meters is 0.1.

Include the following required mapping on all configurations:

```
{:srid      "CUSTOM:900914"
 :cell-size 98.425} ; (feet)
```

7.2 Section 2: Ignition data from which to build simulation inputs

GridFire allows us to choose how we want to initialize the ignition area. We can choose one of 2 options: to initialize a single point or an existing burn perimeter (raster).

To initialize a single point, include the following mappings:

- **ignition-row**: (single, list, or range of values)
- **ignition-col**: (single, list, or range of values)

For this method of ignition, values may be entered in one of three ways:

1. If a single value is provided, it will be kept the same for all simulations.
2. For a list of values, a value from the list will be randomly selected in each simulation.
3. For a range of values, a value from the range [inclusive exclusive] will be randomly selected in each simulation.

```
{:ignition-row [10 90]
 :ignition-col [20 80]}
```

Another way we can ignite a single point is to omit the keys **ignition-row** and **ignition-col**. With this method, we can optionally constrain the ignition location by an ignition-mask raster and/or an edge-buffer. For specifiying these constraints include these optional mappings:

- **ignition-mask**: a map of fetch specifications (see section 1)
- **edge-buffer**: the thickness (feet) along the edge of the computational domain where ignitions cannot occur

[6] https://landfire.gov/faqprint.php

Note: Nonzero values in the ignition mask are considered ignitable
Here's an example of specifiying ignition points using an ignition mask from a geotiff file.

```
{:ignition-mask {:raster {:type :geotiff
                          :source "test/gridfire/resources/weather-test/ignition-mask.tif"}
                 :edge-buffer 98.4}}
```

Note: ignition-row and ignition-col must be omitted for this feature.
To initialize an existing burn perimeter from a raster, we have two options. We can read rasters from a Postgresql database or a raster file on disk. This behavior is controlled as follows:
Include the following mapping at the top level of the configuraiton file:

- **ignition-layer**: a map of fetch specifications

For the fetch specifications include the following mappings:

- **type**: the method for fetching the layer
- **source**: the string input for the fetch method

Here's an example of fetching an intial burn perimeter from a Postgresql database.
,*Note*: be sure to include the map of database connection (**:db-spec**) as described in section 1. ,#+begin_src clojure {:ignition-layer {:type :postgis :source "ignition.ign WHERE rid=1"}} #+end_src
Here's an example of fetching an intial burn perimeter from a file on disk

```
{:fetch-ignition-method :geotiff
 :ignition-layer        "test/gridfire/resources/ign.tif"}
```

GridFire makes use of clojure's multimethods to dispatch control to different handlers for fetching ignition layers. The dispatch depends on what is in the config file. Here's the namespace that implements this functionality.

```
(ns gridfire.fetch
  (:require [clojure.string                 :as s]
            [gridfire.conversion            :as convert]
            [gridfire.fetch.base            :refer [convert-tensor-as-requested get-wrapped-tensor get-wrapped-tensor-multi]]
            [gridfire.fetch.grid-of-rasters :refer [map-grid2d stitch-grid-of-layers]]
            [gridfire.inputs.envi-bsq       :as gf-bsq]
            [gridfire.magellan-bridge       :refer [geotiff-raster-to-tensor]]
            [gridfire.postgis-bridge        :refer [postgis-raster-to-matrix]]
            [magellan.core                  :refer [make-envelope]]
            [manifold.deferred              :as mfd]
            [tech.v3.datatype               :as d]
            [tech.v3.tensor                 :as t]))

(set! *unchecked-math* :warn-on-boxed)

;;-------------------------------------------------------------------------
;; Utilities
;;-------------------------------------------------------------------------

(defn layer->envelope
  [{:keys [^double upperleftx
           ^double upperlefty
           ^double width
           ^double height
           ^double scalex
           ^double scaley]}
   srid]
```

```
  (make-envelope srid
                 upperleftx
                 (+ upperlefty (* height scaley))
                 (* width scalex)
                 (* -1.0 height scaley)))

(defn- comp-convert-fns
  "Like clojure.core/comp but for convert-fns, which can be nil and return ^double."
  ([cf0] cf0)
  ([cf1 cf0]
   (fn ^double [^double v]
     (-> v
         (cond-> (some? cf0) (cf0))
         (double)
         (cf1)))))

(defn- add-angle360
  ^double [^double v ^double angle360]
  (-> v (+ angle360) (mod 360.0)))

(defn get-convert-fn
  ([layer-name layer-spec fallback-unit]
   (get-convert-fn layer-name layer-spec fallback-unit 1.0))
  ([layer-name layer-spec fallback-unit fallback-multiplier]
   (-> (convert/get-units-converter layer-name
                                    (or (:units layer-spec) fallback-unit)
                                    (or (:multiplier layer-spec) fallback-multiplier))
       (as-> convert-fn
             (if-some [angle360 (:gridfire.input/add-correction-angle360 layer-spec)]
               (comp-convert-fns (fn add-angular-correction [^double v] (add-angle360 v angle360))
                                 convert-fn)
               convert-fn)))))

;;--------------------------------------------------------------------------------
;; Data sources
;;--------------------------------------------------------------------------------

(defmethod get-wrapped-tensor-multi :postgis
  [{:keys [db-spec]} {:keys [source]} convert-fn target-dtype]
  (-> (postgis-raster-to-matrix db-spec source)
      (convert-tensor-as-requested convert-fn target-dtype)))

(defmethod get-wrapped-tensor-multi :geotiff
  [_env {:keys [source]} convert-fn target-dtype]
  (geotiff-raster-to-tensor source target-dtype convert-fn))

(defn adapt-bsq-tensor
  [tensor3d]
  (let [[n-bands _w _h] (d/shape tensor3d)]
    (-> tensor3d
        (cond->
         ;; Mimicks the behavior of Magellan. (Val, 21 Oct 2022)
         (= 1 n-bands) (t/mget 0)))))

(defn read-bsq-file
  [source]
  (-> (gf-bsq/read-bsq-file source)
      (mfd/chain
       (fn [layer-map]
         (-> layer-map (update :matrix adapt-bsq-tensor))))
      (deref)))

(defn request-dtype-like-magellan
  [tensor-dtype]
  (case tensor-dtype
    :int16 :int32
    nil))
```

```
(defmethod get-wrapped-tensor-multi :gridfire-envi-bsq
  [_env {:keys [source]} convert-fn target-dtype]
  (-> (read-bsq-file source)
      (as-> layer
            (let [t (:matrix layer)]
              (convert-tensor-as-requested layer
                                           convert-fn
                                           (or target-dtype
                                               (let [tensor-dtype (d/elemwise-datatype t)]
                                                 (request-dtype-like-magellan tensor-dtype))))))))

(defn get-grid-of-rasters
  [env layer-spec convert-fn target-dtype]
  (let [fetched-rasters-grid (->> (:rasters-grid layer-spec)
                                  (map-grid2d (fn [layer-spec-ij]
                                                (future (get-wrapped-tensor env layer-spec-ij convert-fn target-dtype))))
                                  (map-grid2d deref))]
    (stitch-grid-of-layers fetched-rasters-grid)))

(defmethod get-wrapped-tensor-multi :grid-of-rasters
  [env layer-spec convert-fn target-dtype]
  ;; It's good practice to not do advanced work inside (defmethod ...): (Val, 25 Oct 2022)
  ;; code located in (defn ...) is more developer-friendly.
  (get-grid-of-rasters env layer-spec convert-fn target-dtype))

;;--------------------------------------------------------------------------
;; LANDFIRE
;;--------------------------------------------------------------------------

(defn landfire-layer
  [{:keys [landfire-layers] :as config} layer-name]
  (let [layer-spec (get landfire-layers layer-name)
        layer-spec (if (map? layer-spec)
                     layer-spec
                     {:type   :postgis
                      :source layer-spec
                      :units  :metric})
        convert-fn (get-convert-fn layer-name layer-spec nil 1.0)
        datatype   (if (= layer-name :fuel-model)
                     ;; TODO investigate why postgis fuel-model is not converted to int32
                     ;; NOTE: might have been fixed by refactoring to use get-wrapped-tensor. (Val, 25 Oct 2022)
                     :int32
                     :float32)]
    (get-wrapped-tensor config layer-spec convert-fn datatype)))

;;--------------------------------------------------------------------------
;; Initial Ignition
;;--------------------------------------------------------------------------

(defn ignition-layer
  [{:keys [ignition-layer] :as config}]
  (when ignition-layer
    (get-wrapped-tensor config
                        ignition-layer
                        (if-let [burn-values (:burn-values ignition-layer)]
                          (let [{:keys [burned unburned]} burn-values]
                            (fn [x] (cond
                                      (= x burned)   1.0
                                      (= x unburned) 0.0
                                      :else          -1.0)))
                          nil)
                        :float32)))

;;--------------------------------------------------------------------------
;; Ignition Mask
;;--------------------------------------------------------------------------

(defn ignition-mask-layer
```

```
  [{:keys [random-ignition] :as config}]
  (when (map? random-ignition)
    (let [spec (:ignition-mask random-ignition)]
      (get-wrapped-tensor config spec nil nil))))

;;------------------------------------------------------------------------------
;; Weather
;;------------------------------------------------------------------------------

(defn weather-layer
  "Returns a layer map for the given weather name. Units of available weather:
  - temperature:         fahrenheit
  - relative-humidity:   percent (0-100)
  - wind-speed-20ft:     mph
  - wind-from-direction: degreees clockwise from north"
  [config weather-name]
  (let [weather-spec (get config weather-name)]
    (when (map? weather-spec)
      (get-wrapped-tensor config
                          weather-spec
                          (get-convert-fn weather-name weather-spec nil)
                          :float32))))

;;------------------------------------------------------------------------------
;; Moisture Layers
;;------------------------------------------------------------------------------

(defn fuel-moisture-layer
  [{:keys [fuel-moisture] :as config} category size]
  (let [spec (get-in fuel-moisture [category size])]
    (when (map? spec)
      (get-wrapped-tensor config
                          spec
                          (let [fuel-moisture-name (keyword
                                                    ;; WARNING we rely on keyword structure:
                                                    ;; renaming those keywords would break the program.
                                                    (s/join "-" ["fuel-moisture" (name category) (name size)]))]
                            (get-convert-fn fuel-moisture-name spec :percent 1.0))
                          :float32))))


;;------------------------------------------------------------------------------
;; Suppression Difficulty Index Layer
;;------------------------------------------------------------------------------

(defn sdi-layer
  [{:keys [suppression] :as config}]
  (when-let [layer-spec (:sdi-layer suppression)]
    (get-wrapped-tensor config
                        layer-spec
                        (get-convert-fn :suppression layer-spec nil)
                        :float32)))
```

7.3 Section 3: Weather data from which to build simulation inputs

For all the options in this section, you may enter values in one of three ways (as described in section 2): single, list, or range of values.

```
{:temperature              (50 65 80)      ; (degrees Fahrenheit)
 :relative-humidity        (1 10 20)       ; (%)
 :wind-speed-20ft          (10 15 20)      ; (miles/hour)
 :wind-from-direction      (0 90 180 270) ; (degrees clockwise from north)
 :foliar-moisture          90}             ; (%)
```

Temperature, relative humidity, wind speed, and wind direction accepts an additional type of input. GridFire allows us to use weather data from rasters. To use weather data from raster we have two options. This behavior is controlled as follows:

Include the following mapping at the top level of the configuraiton file:

- [**weather-type**]: a map of fetch specifications

For the fetch specifications include the following mappings:

- **type**: the method for fetching the layer
- **source**: the string input for the fetch method

Here's an example of fetching weather rasters from a Postgresql database. **Note**: be sure to include the map of database connection (**:db-spec**) as described in section 1.

```
{:temperature         {:type   :postgis
                       :source "weather.tmpf WHERE rid=100"}
 :relative-humidity   {:type   :postgis
                       :source "weather.rh WHERE rid=100"}
 :wind-speed-20ft     {:type   :postgis
                       :source "weather.ws WHERE rid=100"}
 :wind-from-direction {:type   :postgis
                       :source "weather.wd WHERE rid=100"}}
```

Here's an example of fetching weather rasters from files on disk.

```
{:temperature         {:type   :geotiff
                       :source "test/gridfire/resources/weather-test/tmpf_to_sample.tif"}
 :relative-humidity   {:type   :geotiff
                       :source "test/gridfire/resources/weather-test/rh_to_sample.tif"}
 :wind-speed-20ft     {:type   :geotiff
                       :source "test/gridfire/resources/weather-test/ws_to_sample.tif"}
 :wind-from-direction {:type   :geotiff
                       :source "test/gridfire/resources/weather-test/d_to_sample.tif"}}
```

NOTE: Gridfire expects weather raster's resolution and the landfire's resolution as designated by the ':cell-size' must be exact multiples of one another. This means you may choose to use raster's of different cell sizes to improve preformance.

Gridfire uses imperial units for its calculations. Gridfire optionally allows us to use weather in different units and scale.

To specify the need for conversion from metric to imperial, include the following mapping in the fetch specifications:

- **units**: keyword :metric

To specify the need for conversion from absolute to imperial, include

- **units**: keyword :absolute

```
{:temperture {:type   :geotiff
              :source "test/gridfire/resources/weather-test/tmpf_to_sample.tif"
              :units  :metric}}
```

7.4 Section 4: Number of simulations and (optional) random seed perimeter

```
{:max-runtime               60            ; (minutes)
 :simulations               10
 :ellipse-adjustment-factor 1.0           ; (< 1.0 = more circular, > 1.0 = more elliptical)
 :random-seed 1234567890}                 ; long value (optional)
```

7.5 Section 5: Outputs

Currently supported Geotiff layers for output

- fire-spread
- flame-length
- fire-line-intensity
- burn-history

To control the layers to output include the following mappings:

- **output-layers**: map of layers-name to timestep (in minutes) or the keyword ':final'
- **output-geotiff**: boolean

```
{:output-layers  {:fire-spread  10
                  :burn-history :final}
 :output-geotiff true}
```

The configuration above specify that we'd like to output one firespread geotiff every 10 minutes in the simulation. For the burn history we'd like to output the geotiff file at the final timestep of the simulation.

Note: if entry for ':output-layers' is omitted but ':output-geotiff' is set to true then Gridfire will output all layers above at the final timestep.

Gridfire also supports a number of layers that aggregate data across simulations.

To control the output of the burn probability layer, which is calcualted as the number of times a cell burned divided by the number of simulations, include the following mapping:

- **output-burn-probability**: timestep (in minutes) or keyword ':final'

```
{:output-burn-probability 10}
```

To specify the output of the flame length sum layer, which is the sum of flame lengths across simulations, include the following mapping:

- **output-flame-length-sum**: keyword ':max' or ':directional'

```
{:output-flame-length-sum :max}
```

To specify the output of the flame length max layer, which is the max of flame lengths across simulations, include the following mapping:

- **output-flame-length-max**: keyword ':max' or ':directional'

```
{:output-flame-length-max :directional}
```

To specify the output of the burn count layer, which is the number of times a cell has burned across simulations, include the following mapping:

- **output-burn-count**: boolean

```
{:output-burn-count true}
```

To specify the output of the spot count layer, which is the number of times a spot ignition occured in a cell, include the following mapping:

- **output-spot-count**: boolean

```
{:output-spot-count true}
```

Other output mappings:

```
{:outfile-suffix          "_tile_100"
 :output-landfire-inputs? true
 :output-pngs?            true
 :output-csvs?            true}
```

7.6 Section 6: Perturbations

Gridfire supports applying perturbations to input rasters during simulations in order to account for inherent uncertainty in the input data. A uniform random sampling of values within a given range is used to address these uncertanties.

To specify this in the config file include the following mappings:

- **perturbations:** a map of layer names to a map of perturbation configurations

```
{:perturbations {:canopy-height {:spatial-type :global
                                 :range        [-1.0 1.0]}}}
```

The above config specifies that a randomly selected value between -1.0 and 1.0 should be added to the canopy height value. This perturbation will be applied globally to all cells. We could also, instead, specify that each individual cell should be perturbed independently by setting **:spatial-type** to **:pixel**. Finally, we can specify a non-constant but smoother perturbation by setting **:spatial-type** to **:smoothed-supergrid** along with some more configuration:

```
{:perturbations {:canopy-height {:spatial-type :smoothed-supergrid
                                 ;; Supergrid size param: number of subdivisions along each of the (b, i, j) axes
                                 ;; For a value of [sb si sj], the number of supergrid points will be (sb + 2)(si + 2)(sj + 2).
                                 ;; Recommendation: stick to small integers to avoid making the perturbation too wiggly.
                                 :gridfire.perturbation.smoothed-supergrid/supergrid-size [2 3 3]
                                 :range [-1. 1.]}}}
```

The above will behave similarly to a ':pixel' perturbation, except that the i.i.d 'white noise' is sampled not on each individual pixel of the space-time grid, but on a small-cardinality coarser-grained 'supergrid'; values for each individual pixel will then be resolved by linear smoothing (i.e. averaging) between the 8 corners of the supergrid cell containing it. **Limitations:** smoothed-supergrid perturbations are not isotropic in space, nor are they spatially homogeneous (generated perturbations will be smoother near the center of supergrid cells than near their edges) - a potential direction to address these shortcomings would be to generate the supergrid by sampling an RBF Gaussian Process rather than i.i.d uniform distributions.

Gridfire expects perturbations to be in imperial units. If these perturbations are meant to be in metric, you must include an entry for the units:

```
{:perturbations {:canopy-height {:spatial-type :global
                                 :range        [-1.0 1.0]
                                 :units        :metric}}}
```

For efficient execution, a custom pseudo-random generation algorithm has been designed for {`:spatial-type :pixel`} perturbations, which we call Hash-Determined Pixel Perturbations. The tradeoff of this algorithm is efficient execution (very low cost in both time and memory usage) at the expense of a potentially slightly degraded pseudo-randomness.

7.6.1 Hash-Determined Pixel Perturbations

Logically, {`:spatial-type :pixel`} perturbations consist of filling a discrete grid with "white noise", by sampling one independent random variable per grid cell ('pixel'). Computationally, however, exhaustively realizing the whole grid of random perturbations would be prohibitively expensive, and also wasteful, since oftentimes only a small fraction of the perturbations are ever read by the simulation program - this is the case when the perturbations must happen on Hour-band × X × Y grid (`[b i j]` in code), which can contain billions of cells, of which only a few millions will actually use their perturbation during simulation.

One strategy we previously used is laziness: sampling the per-pixel random variables as they are requested by the simulation, while maintaining a cache of the sampled values, such that subsequent reads all see the same results.

We have found a more efficient strategy, however, which we dubbed *Hash-Determined Pixel Perturbations*: HDPP approximates the p: cell $\mapsto$ perturbation$random function by a composition of$:

1. a deterministic hashing function h: cell $\mapsto$ hash-bucket, $where$hash-bucket$is an integer between 0 and 2d(say, 1024) arand$
hash-bucket $\mapsto$ perturbation

   Concretely, the computation of perturbations goes as follows:

   1. Eagerly generate and store a 2^d-length array 'h-¿perturb' of randomly-sampled pertubation values.
   2. When the perturbation for cell `[b i j]` is requested, compute it as:

```
(fn resolve-perturbation [h->perturb b i j]
  (let [coords-hash (hash-pixel-coordinates b i j)]
    (nth h->perturb coords-hash)))
```

The above pseudo-code is very close to Gridfire's actual implementation code for HDPP, which differs only to leverage more efficient JVM-primitive operations:

```
(ns gridfire.perturbations.pixel.hash-determined
  "Implements 'Hash-Determined Pixel Perturbations' (a name chosen by the GridFire team),
  an algorithm for efficiently emulating a 'white noise' random process on a large discrete grid,
```

```
  without having to generate a random perturbation for each cell of the entire grid,
  nor even for each requested cell."
  (:import (org.apache.commons.codec.digest MurmurHash3)))

(defn- is-power-of-2?
  [^long n]
  (if-not (pos-int? n)
    false
    (cond
      (= n 1)  true
      (odd? n) false
      :else    (recur (quot n 2)))))

(comment

  (def n 2r1000)
  (is-power-of-2? 2r1000)
  (is-power-of-2? 2r1010)

  *e)

(defn gen-hash->perturbation
  "Randomly populates an array mapping hash buckets to perturbations.

  Given:
  - n-buckets: a power of 2, the number of hash buckets,
  - gen-perturbation: a 1-arg function, accepting a hash bucket h (an integer < n-buckets)
    and returning a double, presumably sampled from the desired distribution of perturbations
    (Typically, the argument h will not be used, but it might serve to implement a form of reproducible
    pseudo-randomness.),

  returns an array of length n-buckets."
  ^doubles
  [n-buckets gen-perturbation]
  {:pre [(is-power-of-2? n-buckets)]}
  (into-array
   Double/TYPE
   (map gen-perturbation
        (range n-buckets))))

(defn resolve-perturbation-for-coords
  "Resolves the random perturbation for the grid cell ('pixel') of the supplied coordinates,
  using the h->perturb returned by #'gen-hash->perturbation.

  This function is deterministic: all the randomness happened when h->perturb was created."
  (^double [^doubles h->perturb ^long i ^long j]
   (let [n-buckets   (alength h->perturb)
         coords-hash (as-> (int 36791) h
                       (MurmurHash3/hash32 i j h)
                       (bit-and h (dec n-buckets)))]
     (aget h->perturb coords-hash)))
  (^double [^doubles h->perturb ^long b ^long i ^long j]
   (let [n-buckets   (alength h->perturb)
         coords-hash (as-> (int 6053696) h
                       ;; Compared to clojure.core's (hash) or (hash-combine) functions, MurmurHash3 buys us 2 things:
                       ;; 1. a guarantee of stable behaviour - the hashing functions will still return
                       ;; the same results after upgrading. That's not guaranteed with (hash).
                       ;; 2. Better performance, since those methods have primitive signatures.
                       ;; (this performance gain was empirically observed).
                       (MurmurHash3/hash32 i j h)
                       (MurmurHash3/hash32 b h)
                       (bit-and h
                                ;; fast modulo op - that's why we require n-buckets to be a power of 2.
                                ;; NOTE there is no significant efficiency gain to replacing this expr with a hardcoded int like 2r1
                                (dec n-buckets)))]
     (aget h->perturb coords-hash))))
```

This approach is very efficient because we only generate and store 1024 random values instead of one per requested pixel, and both Murmur3 hashing and small-constant-array lookup are very fast operations.

Fundamentally, we are leveraging the fact that hashing functions like Murmur3 have been designed to efficiently emulate a form of randomness.

With respect to simulation behaviour, such a perturbation process is effectively equivalent to "truly random" perturbations. The use of a hashing function might slightly degrade the quality of the randomness, creating more correlations than true IID randomness, but we have already accepted a similar tradeoff by adopting pseudo-random generators like `java.util.Random`. If our sampling algorithm for individual values was truly random, we would indeed be reducing the entropy of our perturbations process from billions of bits to a few thousands; but we're basing all this on deterministic pseudo-random generators anyway, which means that the entropy is, strictly speaking, 0. Visual inspection shows that an HDPP-generated perturbation map exhibits no more perceptible patterns than an iid-generated one.

7.6.2 What to Expect from Perturbations

The typical intent of perturbations is to **propagate uncertainty** about the inputs, which can mean:

1. verifying that simulation outcomes are robust to small changes in the inputs.
2. in a MCMC mindset, sampling the effects of inputs uncertainty on simulation outcomes.

WARNING: in order to achieve this last goal, *the probability distribution of perturbations must be representative of your beliefs about input uncertainty.* It's quite likely that independently-sampled uniform distributions will fall short of this requirement, as realistic uncertainty about weather and landscape will involve non-uniform distributions and correlations between inputs; that is a limitation of GridFire's **{:spatial-type :global}** perturbations, in which case we recommend that you model your uncertainty using a less-trivial sort of probability distribution (such as a multivariate Gaussian, or mixture thereof) and sample from it upstream of GridFire.

Pixel perturbations are also not good at modeling input uncertainty: while they do provide a form of spatial variability, that variability is not realistic for modeling input uncertainty, as it ignores potential spatial correlations between grid cells. For example, inaccuracies in weather forecasting will never look like random i.i.d noise (a.k.a "static"): the actual temperatures will be consistently above or below the forecast on large contiguous regions. In other words, pixel perturbations are not good at exploring the subspace of credible inputs; they will mostly move the inputs in directions orthogonal to that subspace.

However, **pixel perturbations can be useful for re-creating small-scale spatial heterogeneity** (for example, the fact that canopy height might not be constant even on a small scale). This concern is orthogonal to propagating uncertainty: it's likely more relevant to view it as a parameter for tuning the fire-spread algorithm. If Percolation Theory and statistical physics are any indication, the fire-spreading behavior might respond with sharp threshold effects to varying the amplitude of pixel perturbations: if you are relying on this, tune carefully.

7.7 Section 7: Spotting

Gridfire supports spot fires. To turn on spot ignitions include the key **spotting** at the top level of the config file. The value is a map containg these required entries:

- **num-firebrands**: number of firebrands each torched tree will produce
- **decay-constant**: positive number

- **crown-fire-spotting-percent**: probability a crown fire ignition will spot fires

You may also choose to include surface fire spotting. This behavior is controlled by including the following mappings under the **:spotting** configuration:

- **surface-fire-spotting**: a map containing these required entries:
  - **spotting-percent**: a vector of fuel range and percent pairs where the fuel range is a tuple of integers representing the fuel model numbers and the percent is the percentage of surface fire igntion events that will spot fires
  - **critical-fire-line-intensity**: the fireline intensity below which surface fire spotting does not occur

```
{:spotting {:num-firebrands              [10 50]
            :decay-constant              0.005
            :crown-fire-spotting-percent 0.
            :surface-fires-spotting      {:spotting-percent             [[1 100] 1.0]
                                          :critical-fire-line-intensity 2000}}} ;(kW/m)
```

7.8 Section 8: Fuel moisture data from which to build simulation inputs

GridFire allows us to optinally use fuel moisture from rasters instead of calculating it (by default) from temperature and relative humidity. To specifiy this feature include the following mappings at the top level of the config file:

- **fuel-moisture**: a map of fuel moisture specicfiations

Fuel moisture specifications can be either a specification map as described in section 1 or a ratio value.

```
{:fuel-moisture {:dead {:1hr   {:type   :geotiff
                                :source "test/gridfire/resources/weather-test/m1_to_sample.tif"}
                        :10hr  {:type   :geotiff
                                :source "test/gridfire/resources/weather-test/m10_to_sample.tif"}
                        :100hr {:type   :geotiff
                                :source "test/gridfire/resources/weather-test/m100_to_sample.tif"}}
                 :live {:woody      0.80
                        :herbaceous 0.30}}}
```

Note: Dead fuel moistures are expected to be multiband rasters and live fuel moistures are singleband.

7.9 Section 9: Optimization

Gridifre allows us use multithread processing to improve performance of the simulation run. To specify the type of parallel strategy we'd like to use include the following mapping at the top level of the config file:

- **parallel-strategy**: a keyword of the stratgey

```
{:parallel-strategy :within-fires}
```

Gridfire supports two types of paralleliztion. To prallelize:

- between ensemble of simulations use the keyword **:between-fires**
- within each ensemble member use the keyword **:within-fires**

7.10 Section 10: Spread Algorithm

Gridfire uses the method of adaptive timestep and fractional distances cording to Morais2001. Gridfire provides implementation for two interpretations on how fractional distances are handled. When calculating the fractional distance of a cell, there are 8 possible trajectories from which fire can spread into the cell. When fractional distances are preserved between temesteps, they can be individually tracked so that each trajectory does not have an effect on another, or combined using the largest value among the trajectories. By default Gridfire will track fractional distances individually.

To specify trajectories to have a cumulative effect include the following key value pair at the top level of the config file:

- **:fractional-distance-combination** : the keyword ‘:sum‘

7.11 Section 11: Burn period

Gridfire allows you to specify a burn period in the day that fires are allowed to burn. The burn period can be specified in two ways:

1. Explicitly setting the start and end clock.
2. Inferring the start and end clock by computing the approximate sunrise and sunset times using the geospatial data in the input landfire layer.

By default the burn-period is set from ”00:00” to ”24:00”.
To explicitly set a burn period, you must include a few keys at the top level of the config file.

1. **:burn-period** :: map with the following keys:
2. **:start** a string with the format HH:MM
3. **:end** a string with the format HH:MM
4. **:weather-start-timestamp** :: inst (timezone aware)
5. **:ignition-start-timestamp** :: inst (timezone aware)

```
{:burn-period              {:start "08:00", :end "9:00"}
 :weather-start-timestamp  #inst "2022-07-12T09:00:00.000+00:00"
 :ignition-start-timestamp #inst "2022-07-12T09:05:00.000-00:00"}
```

To infer the burn period using the sunrise and sunset of the input data, you must include a few keys at the top level of the config file.

1. **:burn-period-frac** :: A number from 0.0 to 1.0, positioning the center of the burn period in the interval between sunrise and sunset
2. **:burn-period-length** :: Length of the burn period, in hours
3. **:weather-start-timestamp** :: inst (timezone aware)
4. **:ignition-start-timestamp** :: inst (timezone aware)

```
{:burn-period-frac         0.6
 :burn-period-duration     10.0
 :weather-start-timestamp  #inst "2022-07-12T09:00:00.000+00:00"
 :ignition-start-timestamp #inst "2022-07-12T09:05:00.000-00:00"}
```

7.12 Section 12: Suppression

GridFire supports suppression of fires. Suppression emulates human interventions reacting to fire spread by suppressing ('putting out') chosen contiguous segments of the fire front, typically backing and flanking fires. There are two suppression algorithms supported. Each algorithm is aimed at computing the percentage of the perimeter of the fire that should be contained. To enable suppression you must include the key **:suppression** at the top level of the config file.

The **:suppression** key value is a map containing a different set of required entries depending on which suppression algorithm you want to use.

One approach uses a suppression curve algorithm. To enable this include the following:

suppression-dt the time (in minutes) betweeen suppression events

suppression-coefficient a constant calibration unit (see section 5.6.1)

```
{:suppression {:suppression-dt          60.0
               :suppression-coefficient 2.0}}
```

Another approach uses the suppression difficulty index raster layer. To enable this include the following:

suppression-dt the time (in minutes) betweeen suppression events

sdi-layer a map of fetch specifications

sdi-sensitivity-to-difficulty a calibration constant

sdi-reference-suppression-speed a calibration constant

sdi-containment-overwhelming-area-growth-rate a calibration constant

For more information about these parameters see section 5.6.2

```
:suppression {:suppression-dt                                60.0
              :sdi-layer                                     {:type       :geotiff
                                                              :source     "path/to/geotiff"
                                                              :multiplier 0.01}
              :sdi-sensitivity-to-difficulty                 2.0
              :sdi-containment-overwhelming-area-growth-rate 5000.0
              :sdi-reference-suppression-speed               100.0}
```

8 Example Configuration Files

Here is a complete sample configuration for using landfire layers from our postigs enabled database and initializing burn points from a range of values.

Here is a complete sample configuration for using LANDFIRE layers from our PostGIS-enabled database with ignition points randomly sampled from a range.

```
{;; Section 1: Landscape data to be shared by all simulations
 :db-spec                  {:classname   "org.postgresql.Driver"
                            :subprotocol "postgresql"
                            :subname     "//localhost:5432/gridfire"
                            :user        "gridfire"
                            :password    "gridfire"}
 :landfire-layers           {:aspect             {:type   :postgis
```

```
                                          :source "landfire.asp WHERE rid=100"}
                          :canopy-base-height {:type   :postgis
                                               :source "landfire.cbh WHERE rid=100"}
                          :canopy-cover       {:type   :postgis
                                               :source "landfire.cc WHERE rid=100"}
                          :canopy-height      {:type   :postgis
                                               :source "landfire.ch WHERE rid=100"}
                          :crown-bulk-density {:type   :postgis
                                               :source "landfire.cbd WHERE rid=100"}
                          :elevation          {:type   :postgis
                                               :source "landfire.dem WHERE rid=100"}
                          :fuel-model         {:type   :postgis
                                               :source "landfire.fbfm40 WHERE rid=100"}
                          :slope              {:type   :postgis
                                               :source "landfire.slp WHERE rid=100"}}
 :srid                    "CUSTOM:900914"
 :cell-size               98.425          ; (feet)

 ;; Section 2: Ignition data from which to build simulation inputs
 :ignition-row            [10 90]
 :ignition-col            [20 80]

 ;; Section 3: Weather data from which to build simulation inputs
 ;; For all options in this section, you may enter values in one of five ways:
 ;;   1. Single Value: 25
 ;;   2. List of Values: (2 17 9)
 ;;   3. Range of Values: [10 20]
 ;;   4. Raster from file on disk: {:type :geotiff :source "path/to/file/weather.tif"}
 ;;   5. Raster from Postgresql database: {:type :postgis :source "weather.ws WHERE rid=1"}
 ;;
 ;; If a single value is provided, it will be kept the same for all simulations.
 ;; For a list of values, the list will be randomly sampled from in each simulation.
 ;; For a range of values, the range [inclusive exclusive] will be randomly sampled from in each simulation.
 :temperature             (50 65 80)      ; (degrees Fahrenheit)
 :relative-humidity       (1 10 20)       ; (%)
 :wind-speed-20ft         (10 15 20)      ; (miles/hour)
 :wind-from-direction     (0 90 180 270) ; (degrees clockwise from north)
 :foliar-moisture         90              ; (%)

 ;; Section 4: Number of simulations and (optional) random seed parameter
 :max-runtime             60              ; (minutes)
 :simulations             10
 :random-seed             1234567890      ; long value (optional)

 ;; Section 5: Fire spread model behavior
 :ellipse-adjustment-factor 1.0           ; (< 1.0 = more circular, > 1.0 = more elliptical)
 :crowning-disabled?      false

 ;; Section 6: Types and names of outputs
 :outfile-suffix          "_tile_100"
 :output-landfire-inputs? true
 :output-geotiffs?        true
 :output-pngs?            true
 :output-csvs?            true}
```

Here is a complete sample configuration for reading both the LANDFIRE layers, initial burn perimeter, and weather layers from GeoTIFF files on disk.

```
{;; Section 1: Landscape data to be shared by all simulations
 :landfire-layers         {:aspect            {:type   :geotiff
                                               :source "test/gridfire/resources/asp.tif"}
                           :canopy-base-height {:type   :geotiff
                                               :source "test/gridfire/resources/cbh.tif"}
                           :canopy-cover       {:type   :geotiff
                                               :source "test/gridfire/resources/cc.tif"}
                           :canopy-height      {:type   :geotiff
```

```
                                          :source "test/gridfire/resources/ch.tif"}
                        :crown-bulk-density {:type   :geotiff
                                             :source "test/gridfire/resources/cbd.tif"}
                        :elevation          {:type   :geotiff
                                             :source "test/gridfire/resources/dem.tif"}
                        :fuel-model         {:type   :geotiff
                                             :source "test/gridfire/resources/fbfm40.tif"}
                        :slope              {:type   :geotiff
                                             :source "test/gridfire/resources/slp.tif"}}
:srid                   "CUSTOM:900914"
:cell-size              98.425          ; (feet)

;; Section 2: Ignition data from which to build simulation inputs
:ignition-layer         {:type   :geotiff
                         :source "test/gridfire/resources/ign.tif"}

;; Section 3: Weather data from which to build simulation inputs
;; For all options in this section, you may enter values in one of five ways:
;;   1. Single Value: 25
;;   2. List of Values: (2 17 9)
;;   3. Range of Values: [10 20]
;;   4. Raster from file on disk: {:type :geotiff :source "path/to/file/weather.tif"}
;;   5. Raster from Postgresql database: {:type :postgis :source "weather.ws WHERE rid=1"}
;;
;; If a single value is provided, it will be kept the same for all simulations.
;; For a list of values, the list will be randomly sampled from in each simulation.
;; For a range of values, the range [inclusive exclusive] will be randomly sampled from in each simulation.

:temperature         {:type   :geotiff
                      :source "test/gridfire/resources/weather-test/tmpf_to_sample.tif"} ; (degrees Fahrenheit)
:relative-humidity   {:type   :geotiff
                      :source "test/gridfire/resources/weather-test/rh_to_sample.tif"}   ; (%)
:wind-speed-20ft     {:type   :geotiff
                      :source "test/gridfire/resources/weather-test/ws_to_sample.tif"}   ; (miles/hour)
:wind-from-direction {:type   :geotiff
                      :source "test/gridfire/resources/weather-test/wd_to_sample.tif"}   ; (degrees cw from north)
:foliar-moisture     90                                                                  ; (%)

;; Section 4: Number of simulations and (optional) random seed parameter
:max-runtime            60              ; (minutes)
:simulations            10
:random-seed            1234567890      ; long value (optional)

;; Section 5: Fire spread model behavior
:crowning-disabled?     false           ; when true, disables crown fire initiation. Can be useful to avoid over-prediction.
:ellipse-adjustment-factor 1.0          ; (< 1.0 = more circular, > 1.0 = more elliptical)

;; Section 6: Types and names of outputs
:outfile-suffix         "_from_raster_ignition"
:output-landfire-inputs? true
:output-geotiffs?       true
:output-pngs?           true
:output-csvs?           true}
```

This concludes our discussion of GridFire's command line interface.

References

Frank A Albini. Estimating wildfire behavior and effects. *General technical report/Intermountain forest and range experiment station. USDA (no. INT-30)*, 1976.

Frank A Albini and Robert G Baughman. Estimating windspeeds for predicting wildland fire behavior. *USDA Forest Service Research Paper INT (USA)*, 1979.

Frank A Albini and Carolyn H Chase. *Fire containment equations for pocket calculators*, volume 268. Intermountain Forest and Range Experiment Station, Forest Service, US Dept. of Agriculture, 1980.

Hal E Anderson. Heat transfer and fire spread. *USDA forest service research paper. Intermountain and range experiment station*, (69), 1969.

Hal E Anderson. Aids to determining fuel models for estimating fire behavior. *The Bark Beetles, Fuels, and Fire Bibliography*, page 143, 1982.

Patricia L Andrews et al. Modeling wind adjustment factor and midflame wind speed for rothermel's surface fire spread model. 2012.

Robert E Burgan. Estimating live fuel moisture for the 1978 national fire danger rating system. *USDA Forest Service Research Paper INT (USA). no. 226.*, 1979.

George M Byram. Combustion of forest fuels. *Forest fire: Control and use*, 1:61–89, 1959.

Miguel G Cruz, Martin E Alexander, and Ronald H Wakimoto. Development and testing of models for predicting crown fire rate of spread in conifer forest stands. *Canadian Journal of Forest Research*, 35 (7):1626–1639, 2005.

Rich Hickey. The clojure programming language. In *Proceedings of the 2008 Symposium on Dynamic Languages*, DLS '08, New York, NY, USA, 2008. ACM.

KEISUKE Himoto and Takeyoshi Tanaka. Transport of disk-shaped firebrands in a turbulent boundary layer. *Fire Safety Science*, 8:433–444, 2005.

Marco Emanuel Morais. Comparing spatially explicit models of fire spread through chaparral fuels: A new model based upon the rothermel fire spread equation. Master's thesis, University of California, Santa Barbara, 2001.

Holly A Perryman, Christopher J Dugaw, J Morgan Varner, and Diane L Johnson. A cellular automata model to link surface fires to firebrand lift-off and dispersal. *International journal of wildland fire*, 22 (4):428–439, 2012.

Seth H Peterson, Marco E Morais, Jean M Carlson, Philip E Dennison, Dar A Roberts, Max A Moritz, David R Weise, et al. Using hfire for spatial modeling of fire in shrublands. 2009.

Seth H Peterson, Max A Moritz, Marco E Morais, Philip E Dennison, and Jean M Carlson. Modelling long-term fire regimes of southern california shrublands. *International Journal of Wildland Fire*, 20(1): 1–16, 2011.

Richard C Rothermel. A mathematical model for predicting fire spread in wildland fuels. *Research paper/Intermountain forest and range experiment station. USDA (INT-115)*, 1972.

Richard C Rothermel. How to predict the spread and intensity of forest and range fires. *General technical report/Intermountain Forest and Range Experiment Station. USDA (no. INT-143)*, 1983.

Richard C Rothermel. Predicting behavior and size of crown fires in the northern rocky mountains. *Research paper/United States Department of Agriculture, Intermountain Research Station (INT-438)*, 1991.

N Sardoy, JL Consalvi, Ahmed Kaiss, AC Fernandez-Pello, and B Porterie. Numerical study of ground-level distribution of firebrands generated by line fires. *Combustion and Flame*, 154(3):478–488, 2008.

Joe H Scott and Robert E Burgan. Standard fire behavior fuel models: a comprehensive set for use with rothermel's surface fire spread model. *The Bark Beetles, Fuels, and Fire Bibliography*, page 66, 2005.

Charles E Van Wagner. Conditions for the start and spread of crown fire. *Canadian Journal of Forest Research*, 7(1):23–34, 1977.

9 Appendix: Implementation Code

9.1 GridFire Configuration Format

9.1.1 gridfire.config-validation-test

```
(ns gridfire.config-validation-test
  (:require [clojure.spec.alpha   :as s]
            [clojure.test         :refer [deftest is]]
            [gridfire.spec.config :as spec]))

;;--------------------------------------------------------------------------------
;; Config
;;--------------------------------------------------------------------------------

(def resources-path "test/gridfire/resources/")

;;--------------------------------------------------------------------------------
;; Utils
;;--------------------------------------------------------------------------------

(defn in-file-path [filename]
  (str resources-path filename))

;;--------------------------------------------------------------------------------
;; Validator tests
;;--------------------------------------------------------------------------------

(deftest ^:unit valid-landfire-geotiff-test
  (let [config {:aspect             {:type   :geotiff
                                     :source (in-file-path "asp.tif")}
                :canopy-base-height {:type   :geotiff
                                     :source (in-file-path "cbh.tif")}
                :canopy-cover       {:type   :geotiff
                                     :source (in-file-path "cc.tif")}
                :canopy-height      {:type   :geotiff
                                     :source (in-file-path "ch.tif")}
                :crown-bulk-density {:type   :geotiff
                                     :source (in-file-path "cbd.tif")}
                :elevation          {:type   :geotiff
                                     :source (in-file-path "dem.tif")}
                :fuel-model         {:type   :geotiff
                                     :source (in-file-path "fbfm40.tif")}
                :slope              {:type   :geotiff
                                     :source (in-file-path "slp.tif")}}]
    (is (s/valid? ::spec/landfire-layers config))))

(deftest ^:unit valid-weather-geotiff-test
  (let [config {:type      :geotiff
                :source    (in-file-path "weather-test/tmpf_to_sample.tif")
                :cell-size 10.0}]
    (is (s/valid? ::spec/weather config))))

(deftest ^:unit valid-weather-postgis-test
  (let [config {:type      :postgis
                :source    "weather.ws WHERE rid=1"
```

```
                :cell-size 10.0}]
    (is (s/valid? ::spec/weather config))))
```

9.1.2 gridfire.spec.memoization

```
(ns gridfire.spec.memoization
  (:require [clojure.spec.alpha :as s]))

(s/def ::surface-fire-min (s/or :no-memoization nil? :memoization-strategy #{:within-sims :across-sims}))

(s/def ::memoization
  (s/keys :opt-un [::surface-fire-min]))
```

9.1.3 gridfire.spec.common

```
(ns gridfire.spec.common
  (:require [clojure.java.io                      :as io]
            [clojure.spec.alpha                   :as s]
            [gridfire.spec.inputs.file-raster     :as spec-file]
            [gridfire.spec.inputs.grid-of-rasters :as spec-grid]
            [gridfire.spec.inputs.sql             :as spec-sql]))

;;============================================================================
;; Numeric Samples
;;============================================================================

(s/def ::ordered-pair (fn [[a b]] (< a b)))

(s/def ::ratio (s/and float? #(<= 0.0 % 10.0)))

(s/def ::ratio-range
  (s/and (s/coll-of ::ratio :kind vector? :count 2)
         ::ordered-pair))

(s/def ::integer-range
  (s/and (s/coll-of integer? :kind vector? :count 2)
         ::ordered-pair))

(s/def ::float-range
  (s/and (s/coll-of float? :kind vector? :count 2)
         ::ordered-pair))

(s/def ::number-range
  (s/and (s/coll-of number? :kind vector? :count 2)
         ::ordered-pair))

(s/def ::ratio-or-range
  (s/or :ratio ::ratio
        :range ::ratio-range))

(s/def ::integer-or-range
  (s/or :integer integer?
        :range  ::integer-range))

(s/def ::float-or-range
  (s/or :float float?
        :range  ::float-range))

(s/def ::number-or-range
  (s/or :number number?
        :range  ::number-range))

(s/def ::ratio-sample
  (s/or :ratio ::ratio
```

```
        :range ::ratio-range
        :list  (s/coll-of ::ratio :kind list?)))

(s/def ::integer-sample
  (s/or :integer integer?
        :range   ::integer-range
        :list    (s/coll-of integer? :kind list?)))

(s/def ::float-sample
  (s/or :float float?
        :range ::float-range
        :list  (s/coll-of float? :kind list?)))

(s/def ::number-sample
  (s/or :number number?
        :range  ::number-range
        :list   (s/coll-of number? :kind list?)))

(s/def ::lo ::number-or-range)
(s/def ::hi ::number-or-range)

(s/def ::number-or-range-map
  (s/or :number    number?
        :range-map (s/and (s/keys :req-un [::lo ::hi])
                          (fn [{:keys [lo hi]}]
                            (apply <= (flatten [(val lo) (val hi)]))))))

;;=============================================================================
;; File Access
;;=============================================================================

(def file-path-regex      #"^(((\.){1,2}/)*|(/){1})?(([\w\-]*)/)*([\w\-\.]+)$")
(def directory-path-regex #"^(((\.){1,2}/)*|(/){1})?(([\w\-]*)/)*([\w\-]+)/?$")

(s/def ::file-path      (s/and string? #(re-matches file-path-regex %)))
(s/def ::directory-path (s/and string? #(re-matches directory-path-regex %)))

(def ^:dynamic *check-files-exist?*
  "Whether to check for existing files when validating the GridFire config."
  true)

(defn file-exists? [f]
  (or (not *check-files-exist?*)
      (.exists (io/file f))))

(defn file-readable? [f]
  (or (not *check-files-exist?*)
      (.canRead (io/file f))))

(defn file-writable? [f]
  (or (not *check-files-exist?*)
      (.canWrite (io/file f))))

(s/def ::readable-file (s/and ::file-path file-exists? file-readable?))
(s/def ::writable-file (s/and ::file-path file-exists? file-writable?))

(s/def ::readable-directory (s/and ::directory-path file-exists? file-readable?))
(s/def ::writable-directory (s/and ::directory-path file-exists? file-writable?))

;;=============================================================================
;; Layer Coords
;;=============================================================================

;; NOTE that this spec is refined in each subtype of inputs.
(s/def ::type keyword?)

(s/def ::raw-layer-coords-map
  (s/and (s/keys :req-un [::type])
```

```
        (s/or :postgis         ::spec-sql/postgis-coords-map
              :grid-of-rasters ::spec-grid/raw-layer-coords-map
              :file-raster     ::spec-file/raw-layer-coords-map)))

(s/def ::cell-size number?)

(s/def ::unit #{:imperial :metric})

(s/def ::multiplier number?)

(s/def :gridfire.input/add-correction-angle360 number?)

(s/def ::layer-coords-map
  (s/and (s/nonconforming ::raw-layer-coords-map)
         (s/keys :opt-un [::cell-size ::unit ::multiplier]
                 :opt    [:gridfire.input/add-correction-angle360])))

(s/def ::layer-coords (s/or :sql ::spec-sql/sql
                            :map ::layer-coords-map))

(s/def ::ratio-or-layer-coords
  (s/or :ratio  ::ratio
        :raster ::layer-coords))

;;================================================================================
;; Macros
;;================================================================================

(defmacro one-or-more-keys [ks]
  (let [keyseq (map (comp keyword name) ks)]
    `(s/and (s/keys :opt-un ~ks)
            #(some % '~keyseq))))
```

9.1.4 gridfire.spec.perturbations

```
(ns gridfire.spec.perturbations
  (:require [clojure.spec.alpha   :as s]
            [gridfire.spec.common :as common]))

(s/def ::spatial-type #{:global :pixel :smoothed-supergrid})

(s/def ::range ::common/number-range)

;; FIXME: Define a ::units field since it is added by elm_to_grid.clj
(s/def ::perturbation
  (s/keys :req-un [::spatial-type ::range]
          :opt-un [:gridfire.perturbation.smoothed-supergrid/supergrid-size]))

(s/def ::canopy-base-height            ::perturbation)
(s/def ::canopy-cover                  ::perturbation)
(s/def ::canopy-height                 ::perturbation)
(s/def ::crown-bulk-density            ::perturbation)
(s/def ::temperature                   ::perturbation)
(s/def ::relative-humidity             ::perturbation)
(s/def ::wind-speed-20ft               ::perturbation)
(s/def ::wind-direction                ::perturbation)
(s/def ::fuel-moisture-dead-1hr        ::perturbation)
(s/def ::fuel-moisture-dead-10hr       ::perturbation)
(s/def ::fuel-moisture-dead-100hr      ::perturbation)
(s/def ::fuel-moisture-live-herbaceous ::perturbation)
(s/def ::fuel-moisture-live-woody      ::perturbation)
(s/def ::foliar-moisture               ::perturbation)

(s/def :gridfire.perturbation.smoothed-supergrid/supergrid-size
  (s/tuple [:gridfire.perturbation.smoothed-supergrid/supergrid-size-b
            :gridfire.perturbation.smoothed-supergrid/supergrid-size-i
```

```
            :gridfire.perturbation.smoothed-supergrid/supergrid-size-j]))

(s/def :gridfire.perturbation.smoothed-supergrid/supergrid-size-b pos-int?)
(s/def :gridfire.perturbation.smoothed-supergrid/supergrid-size-i pos-int?)
(s/def :gridfire.perturbation.smoothed-supergrid/supergrid-size-j pos-int?)
```

9.1.5 gridfire.spec.burn-period

```
(ns gridfire.spec.burn-period
  (:require [clojure.spec.alpha   :as s]
            [gridfire.spec.common :as common]))

(def burn-period-regex #"^([01]\d|2[0-3]):?([0-5]\d)$")

(s/def ::HH:MM (s/and string? #(re-matches burn-period-regex %)))

(s/def ::start ::HH:MM)

(s/def ::end ::HH:MM)

(s/def ::burn-period
  (s/keys :req-un [::start ::end]))

(s/def ::burn-period-frac ::common/ratio)

(s/def ::burn-period-length float?)

;; Test
#_(s/explain ::burn-period {:start "08:00"
                            :end   "20:00"})
```

9.1.6 gridfire.spec.suppression

```
(ns gridfire.spec.suppression
  (:require [clojure.spec.alpha   :as s]
            [gridfire.spec.common :as common]))

(s/def ::suppression-dt                                number?)
(s/def ::suppression-coefficient                       number?)
(s/def ::sdi-layer                                     ::common/layer-coords)
(s/def ::sdi-sensitivity-to-difficulty                 ::common/number-sample)
(s/def ::sdi-containment-overwhelming-area-growth-rate ::common/number-sample)
(s/def ::sdi-reference-suppression-speed               ::common/number-sample)

;; explicity-samples
(s/def ::explicit-samples (s/coll-of number? :kind vector?))

(s/def ::suppression-dt-samples                                ::explicit-samples)
(s/def ::suppression-coefficient-samples                       ::explicit-samples)
(s/def ::sdi-sensitivity-to-difficult-samples                  ::explicit-samples)
(s/def ::sdi-containment-overwhelming-area-growth-rate-samples ::explicit-samples)
(s/def ::sdi-reference-suppression-speed-samples               ::explicit-samples)

(s/def ::mutually-exclusive-keys
  (fn [{:keys [suppression-coefficient sdi-layer]}]
    (or (and suppression-coefficient (nil? sdi-layer))
        (and (nil? suppression-coefficient) sdi-layer))))
```

9.1.7 gridfire.spec.inputs.grid-of-rasters

```
(ns gridfire.spec.inputs.grid-of-rasters
  (:require [clojure.spec.alpha :as s]))
```

```
(s/def ::type #{:grid-of-rasters})

(s/def ::rasters-grid
  ;; NOTE the blocks in the rasters grid must be arranged in an order
  ;; which is consistent with the layout of the sub-tensors.
  ;; Typically, if inside each block the pixels are arranged
  ;; in a [y decreasing, x increasing] layout,
  ;; then the blocks must be arranged in the same way.
  ;; An equivalent way of saying this is that
  ;; the blocks are arranged in a way which is GIS-agnostic,
  ;; and natural for describing a matrix by blocks:
  ;; for example, traversing the stitched matrix in the [i j] -> [i+1 j] direction
  ;; will traverse the blocks in the [grid-i grid-j] -> [grid-i+1 grid-j] direction.
  ;; This organizing principle was chosen because it is the most conceptually simple,
  ;; being agnostic of whatever GIS interpretations we might superimpose on it.
  ;; WARNING: however, this organizing principle might feel at odds with your GIS intuition
  ;; of organizing blocks in a [x increasing, y increasing] layout:
  ;; if so, don't blame :rasters-grid - blame the sub-tensors.
  (s/coll-of (s/coll-of :gridfire.spec.common/raw-layer-coords-map)))

(s/def ::raw-layer-coords-map (s/keys :req-un [::type ::rasters-grid]))
```

9.1.8 gridfire.spec.inputs.sql

```
(ns gridfire.spec.inputs.sql
  (:require [clojure.spec.alpha :as s]))

(def postgis-sql-regex #"[a-z0-9]+(\.[a-z0-9]+)? WHERE rid=[0-9]+")

(s/def ::sql (s/and string? #(re-matches postgis-sql-regex %)))

(s/def ::type #{:postgis})

(s/def ::source ::sql)

(s/def ::postgis-coords-map (s/keys :req-un [::type ::source]))
```

9.1.9 gridfire.spec.inputs.file-raster

```
(ns gridfire.spec.inputs.file-raster
  (:require [clojure.spec.alpha :as s]))

(def raster-file-regex #".*\.(tif|bsq)$")

(s/def ::raster-file-path (s/and :gridfire.spec.common/readable-file #(re-matches raster-file-regex %)))

(s/def ::source ::raster-file-path)

(s/def ::type #{:geotiff :gridfire-envi-bsq})

(s/def ::raw-layer-coords-map (s/keys :req-un [::type ::source]))
```

9.1.10 gridfire.spec.config

```
(ns gridfire.spec.config
  (:require [clojure.spec.alpha          :as s]
            [gridfire.spec.burn-period   :as burn-period]
            [gridfire.spec.common        :as common]
            [gridfire.spec.memoization   :as memoization]
            [gridfire.spec.perturbations :as perturbations]
            [gridfire.spec.suppression   :as suppression]))

;;=============================================================================
```

```
;; Required Keys
;;=============================================================================

(s/def ::max-runtime     ::common/number-sample)
(s/def ::simulations     integer?)
(s/def ::srid            string?)
(s/def ::cell-size       number?)
(s/def ::foliar-moisture ::common/number-sample)

;; Weather

(s/def ::weather-start-timestamp inst?)

(s/def ::weather
  (s/or :matrix ::common/layer-coords
        :global ::common/number-sample))

(s/def ::temperature         ::weather)
(s/def ::relative-humidity   ::weather)
(s/def ::wind-speed-20ft     ::weather)
(s/def ::wind-from-direction ::weather)

;; LANDFIRE

(s/def ::aspect             ::common/layer-coords)
(s/def ::canopy-base-height ::common/layer-coords)
(s/def ::canopy-cover       ::common/layer-coords)
(s/def ::canopy-height      ::common/layer-coords)
(s/def ::crown-bulk-density ::common/layer-coords)
(s/def ::elevation          ::common/layer-coords)
(s/def ::fuel-model         ::common/layer-coords)
(s/def ::slope              ::common/layer-coords)

(s/def ::landfire-layers
  (s/keys
   :req-un [::aspect
            ::canopy-base-height
            ::canopy-cover
            ::canopy-height
            ::crown-bulk-density
            ::elevation
            ::fuel-model
            ::slope]))

;;=============================================================================
;; Optional Keys
;;=============================================================================

(s/def ::random-seed                     (s/or :nil nil? :long integer?))
(s/def ::crowning-disabled?              boolean?)
(s/def ::ellipse-adjustment-factor       ::common/number-sample)
(s/def ::fractional-distance-combination #{:sum}) ; FIXME This is currently unused.
(s/def ::parallel-strategy               #{:within-fires :between-fires})
(s/def ::max-parallel-simulations        (s/or :omitted nil? :provided pos-int?))

;; DB Connection

(s/def ::classname   string?)
(s/def ::subprotocol string?)
(s/def ::subname     string?)
(s/def ::user        string?)
(s/def ::password    string?)

(s/def ::db-spec
  (s/keys
   :req-un [::classname
            ::subprotocol
            ::subname
```

```
            ::user
            ::password]))

;; Ignitions

(s/def ::ignition-start-timestamp inst?)
(s/def ::ignition-row ::common/integer-sample)
(s/def ::ignition-col ::common/integer-sample)

(s/def ::burned   float?)
(s/def ::unburned float?)

(s/def ::burn-values
  (s/keys :req-un [::burned ::unburned]))

(s/def ::ignition-layer
  (s/and
   (s/nonconforming ::common/raw-layer-coords-map)
   (s/keys :opt-un [::burn-values])))

(s/def ::ignition-csv ::common/readable-file)

(s/def ::ignition-mask ::common/raw-layer-coords-map)

(s/def ::edge-buffer number?)

(s/def ::random-ignition
  (s/or
   :boolean boolean?
   :map     (common/one-or-more-keys [::ignition-mask ::edge-buffer])))

(s/def ::simulations-or-ignition-csv
  (fn [{:keys [simulations ignition-csv]}]
    (or simulations ignition-csv)))

(s/def ::max-runtime-or-ignition-csv
  (fn [{:keys [max-runtime ignition-csv]}]
    (or max-runtime ignition-csv)))

(s/def ::ignition-layer-or-ignition-csv
  (fn [{:keys [ignition-layer ignition-csv]}]
    (not (and ignition-layer ignition-csv))))

;; Fuel Moisture

(s/def ::1hr        ::common/ratio-or-layer-coords)
(s/def ::10hr       ::common/ratio-or-layer-coords)
(s/def ::100hr      ::common/ratio-or-layer-coords)
(s/def ::woody      ::common/ratio-or-layer-coords)
(s/def ::herbaceous ::common/ratio-or-layer-coords)

(s/def ::dead
  (s/keys :req-un [::1hr ::10hr ::100hr]))

(s/def ::live
  (s/keys :req-un [::woody ::herbaceous]))

(s/def ::fuel-moisture
  (s/keys :req-un [::dead ::live]))

(s/def ::rh-or-fuel-moisture
  (fn [{:keys [relative-humidity fuel-moisture]}]
    (or relative-humidity fuel-moisture)))

;; Spotting

(s/def ::decay-constant               ::common/number-or-range-map)
(s/def ::num-firebrands               ::common/number-or-range-map)
```

```
(s/def ::mean-distance                ::common/number-or-range-map)
(s/def ::flin-exp                     ::common/number-or-range-map)
(s/def ::ws-exp                       ::common/number-or-range-map)
(s/def ::normalized-distance-variance ::common/number-or-range-map)
(s/def ::delta-y-sigma                ::common/number-or-range-map)
(s/def ::crown-fire-spotting-percent  ::common/ratio-or-range)

(s/def ::valid-fuel-range             (fn [[lo hi]] (< 0 lo hi (inc 303))))
(s/def ::fuel-number-range            (s/and ::common/integer-range ::valid-fuel-range))
(s/def ::fuel-range+number            (s/tuple ::fuel-number-range ::common/float-or-range))
(s/def ::spotting-percent             (s/coll-of ::fuel-range+number :kind vector?))
(s/def ::critical-fire-line-intensity (s/or :number            number?
                                            :fuel-percent-pair (s/coll-of ::fuel-range+number :kind vector?)))

(s/def ::surface-fire-spotting
  (s/keys :req-un [::spotting-percent
                   ::critical-fire-line-intensity]))

(s/def ::spotting
  (s/keys :req-un [::decay-constant
                   ::num-firebrands
                   ::mean-distance
                   ::flin-exp
                   ::ws-exp
                   ::normalized-distance-variance
                   ::crown-fire-spotting-percent]
          :opt-un [::delta-y-sigma
                   ::surface-fire-spotting]))

;; Suppression

(s/def ::suppression
  (s/and
   (s/keys :req-un [::suppression/suppression-dt]
           :opt-un [::suppression/suppression-coefficient
                    ::suppression/sdi-layer
                    ::suppression/sdi-sensitivity-to-difficulty
                    ::suppression/sdi-containment-overwhelming-area-growth-rate
                    ::suppression/sdi-reference-suppression-speed])
   ::suppression/mutually-exclusive-keys))

;; Perturbations

(s/def ::perturbations
  (common/one-or-more-keys
   [::perturbations/canopy-base-height
    ::perturbations/canopy-cover
    ::perturbations/canopy-height
    ::perturbations/crown-bulk-density
    ::perturbations/temperature
    ::perturbations/relative-humidity
    ::perturbations/wind-speed-20ft
    ::perturbations/wind-direction
    ::perturbations/fuel-moisture-dead-1hr
    ::perturbations/fuel-moisture-dead-10hr
    ::perturbations/fuel-moisture-dead-100hr
    ::perturbations/fuel-moisture-live-herbaceous
    ::perturbations/fuel-moisture-live-woody
    ::perturbations/foliar-moisture]))

;; Outputs

(s/def ::output-directory        ::common/writable-directory)
(s/def ::outfile-suffix          string?)
(s/def ::output-landfire-inputs? boolean?)
(s/def ::output-geotiffs?        boolean?)
(s/def ::output-pngs?            boolean?)
(s/def ::output-csvs?            boolean?)
```

```
(s/def ::output-binary?          boolean?)

(s/def ::output-frequency
  (s/or :number number?
        :key    #{:final}))

(s/def ::fire-spread         ::output-frequency)
(s/def ::flame-length        ::output-frequency)
(s/def ::fire-line-intensity ::output-frequency)
(s/def ::spread-rate         ::output-frequency)
(s/def ::burn-history        ::output-frequency)
(s/def ::fire-type           ::output-frequency)

(s/def ::output-layers
  (common/one-or-more-keys
   [::fire-spread
    ::flame-length
    ::fire-line-intensity
    ::spread-rate
    ::burn-history
    ::fire-type]))

(s/def ::output-burn-probability  ::output-frequency) ; FIXME: Why isn't this also just boolean?
(s/def ::output-burn-count?       boolean?)
(s/def ::output-spot-count?       boolean?)
(s/def ::output-flame-length-max  #{:max :directional})
(s/def ::output-flame-length-sum  #{:max :directional})

;;================================================================================
;; Spread Rate Adjustment
;;================================================================================

(defn- long? [x]
  (instance? Long x))

(s/def ::fuel-number->spread-rate-adjustment
  (s/map-of long? double?))

(s/def ::fuel-number->spread-rate-adjustment-samples
  (s/coll-of ::fuel-number->spread-rate-adjustment :kind vector?))

;;================================================================================
;; Burn Period
;;================================================================================

(s/def ::burn-period-required-keys
  (fn [{:keys [burn-period burn-period-frac burn-period-length weather-start-timestamp]}]
    (or (every? nil? [burn-period burn-period-frac burn-period-length])
        (and burn-period weather-start-timestamp)
        (and burn-period-frac burn-period-length weather-start-timestamp))))

;;================================================================================
;; Timestamps
;;================================================================================

(defn- not-after? [t1 t2]
  (<= (inst-ms t1) (inst-ms t2)))

(s/def ::valid-timestamps
  (fn [{:keys [ignition-start-timestamp weather-start-timestamp]}]
    (or (nil? ignition-start-timestamp)
        (and weather-start-timestamp
             ignition-start-timestamp
             (not-after? weather-start-timestamp ignition-start-timestamp)))))

;;================================================================================
;; Mutually exclusive-keys
;;================================================================================
```

```
(s/def ::mutually-exclusive-keys
  (fn [{:keys [ignition-start-timestamp ignition-csv]}]
    (or (and ignition-start-timestamp (nil? ignition-csv))
        (and (nil? ignition-start-timestamp) ignition-csv)
        (every? nil? [ignition-start-timestamp ignition-csv]))))

;;================================================================================
;; Config Map
;;================================================================================

(s/def ::config
  (s/and
   (s/keys
    :req-un [::srid
             ::cell-size
             ::foliar-moisture
             ::temperature
             ::wind-speed-20ft
             ::wind-from-direction
             ::landfire-layers]
    :opt-un [::max-runtime
             ::burn-period/burn-period
             ::burn-period/burn-period-frac
             ::burn-period/burn-period-length
             ::simulations
             ::random-seed
             ::crowning-disabled?
             ::ellipse-adjustment-factor
             ::fractional-distance-combination
             ::fuel-number->spread-rate-adjustment-samples
             ::parallel-strategy
             ::max-parallel-simulations
             ::db-spec
             ::ignition-row
             ::ignition-col
             ::ignition-layer
             ::ignition-csv
             ::ignition-start-timestamp
             ::weather-start-timestamp
             ::random-ignition
             ::relative-humidity
             ::fuel-moisture
             ::memoization/memoization
             ::spotting
             ::suppression
             ::perturbations
             ::output-directory
             ::outfile-suffix
             ::output-landfire-inputs?
             ::output-geotiffs?
             ::output-pngs?
             ::output-csvs?
             ::output-binary?
             ::output-layers
             ::output-burn-probability
             ::output-burn-count?
             ::output-spot-count?
             ::output-flame-length-max
             ::output-flame-length-sum
             ::suppression/suppression-dt-samples
             ::suppression/suppression-coefficient-samples
             ::suppression/sdi-sensitivity-to-difficult-samples
             ::suppression/sdi-containment-overwhelming-area-growth-rate-samples
             ::suppression/sdi-reference-suppression-speed-samples])
   ::ignition-layer-or-ignition-csv
   ::max-runtime-or-ignition-csv
   ::simulations-or-ignition-csv
```

```
   ::rh-or-fuel-moisture
   ::burn-period-required-keys
   ::valid-timestamps
   ::mutually-exclusive-keys))
```

9.1.11 gridfire.spec.common-test

```
(ns gridfire.spec.common-test
  (:require [clojure.spec.alpha   :as s]
            [clojure.test         :refer [deftest is testing]]
            [gridfire.spec.common :as common]))

(deftest ^:unit number-or-range-map-test
  (testing "scalar"
    (is (s/valid? ::common/number-or-range-map 5.0)
        "scalar can be float")

    (is (s/valid? ::common/number-or-range-map 5)
        "scalar can be int"))

  (testing "range"
    (is (s/valid? ::common/number-or-range-map {:lo 5.0 :hi 15.0})
        "lo and hi can both be scalar floats")

    (is (s/valid? ::common/number-or-range-map {:lo 5 :hi 15})
        "lo and hi can both be scalar ints")

    (is (s/valid? ::common/number-or-range-map {:lo [1.0 5.0] :hi [15.0 16.0]})
        "lo and hi can both be float tuples")

    (is (s/valid? ::common/number-or-range-map {:lo [1 5] :hi [15 16]})
        "lo and hi can both be int tuples")

    (is (s/valid? ::common/number-or-range-map {:lo 5.0 :hi [15.0 16.0]})
        "lo and hi can either be scalar or range"))

  (testing "edge cases"
    (is (s/valid? ::common/number-or-range-map {:lo 5.0 :hi [5.0 6.0]})
        "edge of ranges can overlap")

    (is (s/valid? ::common/number-or-range-map {:lo [1.0 2.0] :hi [2.0 3.0]})
        "edge of ranges can overlap"))

  (testing (pr-str :gridfire.input/add-correction-angle360)
    (is (s/valid? :gridfire.input/add-correction-angle360 1008.0)
        "may be > 360°")
    (is (s/valid? :gridfire.input/add-correction-angle360 -56.0)
        "may be negative"))

  (testing "invalid range"
    (let [config {:lo 10.0
                  :hi 1.0 }]
      (is (not (s/valid? ::common/number-or-range-map config))
          "lo should not be higher than hi"))

    (let [config {:lo [1.0 3.0]
                  :hi [2.0 3.0]}]
      (is (not (s/valid? ::common/number-or-range-map config))
          "should not have overlapping ranges"))))

(deftest ^:unit file-path-test
  (is (s/valid? ::common/file-path "/absolute/file/path"))
  (is (s/valid? ::common/file-path "./relative/file/path"))
  (is (s/valid? ::common/file-path "../relative/file/path"))
  (is (s/valid? ::common/file-path "../../relative/file/path"))
  (is (not (s/valid? ::common/file-path ".../relative/file/path")))
```

```
  (is (not (s/valid? ::common/file-path "./relative/file/path/"))
      "should not end in /"))

(deftest ^:unit directory-path-test
  (is (s/valid? ::common/directory-path "/absolute/directory/path")
      "may end without /")
  (is (s/valid? ::common/directory-path "/absolute/directory/path/")
      "may end with /")
  (is (s/valid? ::common/directory-path "../relative/directory/path/"))
  (is (s/valid? ::common/directory-path "../../relative/directory/path/")))
```

9.1.12 gridfire.spec.config-test

```
(ns gridfire.spec.config-test
  (:require [clojure.spec.alpha   :as s]
            [clojure.test         :refer [deftest is testing]]
            [gridfire.spec.config :as config]))

(deftest ^:unit valid-timestamps-test
  (testing "valid"
    (let [config {:weather-start-timestamp  #inst "1970-01-01T00:00:00.000-00:00"
                  :ignition-start-timestamp #inst "1970-01-01T08:00:00.000-00:00"}]
      (is (s/valid? ::config/valid-timestamps config)
          "weather-start-timestamp should be before ignition-start-timestamp"))

    (let [config {:weather-start-timestamp  #inst "1970-01-01T00:00:00.000-00:00"
                  :ignition-start-timestamp #inst "1970-01-01T00:00:00.000-00:00"}]
      (is (s/valid? ::config/valid-timestamps config)
          "weather-start-timestamp can be equal to ignition-start-timestamp")))

  (testing "invalid"
    (let [config {:weather-start-timestamp  #inst "1970-01-01T08:00:00.000-00:00"
                  :ignition-start-timestamp #inst "1970-01-01T00:00:00.000-00:00"}]
      (is (not (s/valid? ::config/valid-timestamps config))
          "weather-start-timestamp should be not be before ignition-start-timestamp"))))
```

9.1.13 gridfire.spec.ignition-test

```
(ns gridfire.spec.ignition-test
  (:require [clojure.spec.alpha   :as s]
            [clojure.test         :refer [deftest is]]
            [gridfire.spec.config :as config]))

(deftest ^:unit path-test
  (is (s/valid? ::config/ignition-layer {:type   :geotiff
                                         :source "test/gridfire/resources/asp.tif"})))

(deftest ^:unit sql-test
  (is (s/valid? ::config/ignition-layer {:type   :postgis
                                         :source "landfire.ign WHERE rid=1"})))

(deftest ^:unit burn-value-test
  (is (s/valid? ::config/ignition-layer {:type        :geotiff
                                         :source      "test/gridfire/resources/asp.tif"
                                         :burn-values {:burned   1.0
                                                       :unburned -1.0}})))
```

9.1.14 gridfire.spec.spotting-test

```
(ns gridfire.spec.spotting-test
  (:require [clojure.spec.alpha     :as s]
            [clojure.test           :refer [deftest is testing]]
            [gridfire.spec.config   :as config]))
```

```
(deftest ^:unit spotting-test
  (let [required {:decay-constant               0.005
                  :mean-distance                {:lo 5.0 :hi 15.0}
                  :normalized-distance-variance {:lo 250.0 :hi 600.0}
                  :flin-exp                     {:lo 0.2 :hi 0.4}
                  :ws-exp                       {:lo 0.4 :hi 0.7}
                  :crown-fire-spotting-percent  [0.1 0.8]
                  :num-firebrands               10}]
    (testing "required"
      (is (s/valid? ::config/spotting required)))

    (testing "optional surface-fire-spotting"
      (let [optional {:delta-y-sigma         20.0
                      :surface-fire-spotting {:spotting-percent             [[[1   149] 1.0]
                                                                             [[150 169] 2.0]
                                                                             [[169 204] 3.0]]
                                              :critical-fire-line-intensity 2000.0}}]
        (is (s/valid? ::config/spotting (merge required optional)))))))

(deftest ^:unit crown-fire-spotting-percent-test
  (testing "scalar"
    (is (s/valid? ::config/crown-fire-spotting-percent 0.1)))

  (testing "range"
    (is (s/valid? ::config/crown-fire-spotting-percent [0.1 0.8])))

  (testing "invalid range"
    (is (not (s/valid? ::config/crown-fire-spotting-percent [0.8 0.1]))
        "first value should not be larger than the second")))

(deftest ^:unit num-firebrands-test
  (testing "scalar"
    (is (s/valid? ::config/num-firebrands 10)))

  (testing "range"
    (is (s/valid? ::config/num-firebrands {:lo 1 :hi 10})
        "lo and hi can both be scalar")

    (is (s/valid? ::config/num-firebrands {:lo [1 2] :hi [9 10]})
        "lo and hi can both be ranges")

    (is (s/valid? ::config/num-firebrands {:lo 1 :hi [9 10]})
        "lo and hi can either be scalar or range"))

  (testing "edge cases"
    (is (s/valid? ::config/num-firebrands {:lo 1 :hi [1 2]})
        "edge of ranges can overlap")

    (is (s/valid? ::config/num-firebrands {:lo [1 2] :hi [2 3]})
        "edge of ranges can overlap"))

  (testing "invalid range"
    (let [config {:lo 10
                  :hi 1 }]
      (is (not (s/valid? ::config/num-firebrands config))
          "lo syhould not be higher than hi"))

    (let [config {:lo [1 3]
                  :hi [2 3]}]
      (is (not (s/valid? ::config/num-firebrands config))
          "should not have overlapping ranges"))))

(deftest ^:unit surface-fire-spotting-test
  (testing "scalar percents"
    (let [config [[[1   149] 1.0]
                  [[150 169] 2.0]
                  [[169 204] 3.0]]]
```

```
      (is (s/valid? ::config/spotting-percent config))))

  (testing "range percents"
    (let [config [[[1   149] [1.0 2.0]]
                  [[150 169] [3.0 4.0]]
                  [[169 204] [0.2 0.4]]]]

      (is (s/valid? ::config/spotting-percent config)))))
```

9.1.15 gridfire.spec.output-test

```
(ns gridfire.spec.output-test
  (:require [clojure.spec.alpha   :as s]
            [clojure.test         :refer [deftest is]]
            [gridfire.spec.config :as config]))

(deftest ^:unit scalar-test
  (let [config {:fire-spread 10}]
    (is (s/valid? ::config/output-layers config))))

(deftest ^:unit final-keyword-test
  (let [config {:fire-spread :final}]
    (is (s/valid? ::config/output-layers config))))

(deftest ^:unit flame-length-test
  (let [config {:flame-length 10}]
    (is (s/valid? ::config/output-layers config))))

(deftest ^:unit fire-line-inentisyt-test
  (let [config {:fire-line-intensity 10}]
    (is (s/valid? ::config/output-layers config))))

(deftest ^:unit burn-history-test
  (let [config {:burn-history 10}]
    (is (s/valid? ::config/output-layers config))))

(deftest ^:unit multiple-output-layer-test
  (let [config {:fire-spread         10
                :flame-lenth         10
                :fire-line-intensity 10
                :burn-history        10}]
    (is (s/valid? ::config/output-layers config))))
```

9.1.16 gridfire.spec.fuel-moisture-test

```
(ns gridfire.spec.fuel-moisture-test
  (:require [clojure.spec.alpha   :as s]
            [clojure.test         :refer [deftest is]]
            [gridfire.spec.config :as config]
            [gridfire.utils.test  :as utils]))

(def resources-path "test/gridfire/resources/weather-test")

(deftest ^:unit basic-test
  (let [config {:dead {:1hr   {:type   :geotiff
                               :source (utils/in-file-path resources-path "m1_to_sample.tif")}
                       :10hr  {:type   :geotiff
                               :source (utils/in-file-path resources-path "m10_to_sample.tif")}
                       :100hr {:type   :geotiff
                               :source (utils/in-file-path resources-path "m100_to_sample.tif")}}
                :live {:woody      {:type   :geotiff
                                    :source (utils/in-file-path resources-path "mlw_to_sample.tif")}
                       :herbaceous {:type   :geotiff
```

```
                                  :source (utils/in-file-path resources-path "mlh_to_sample.tif")}}}]
    (is (s/valid? ::config/fuel-moisture config))))

(deftest ^:unit scalar-test
  (let [config {:dead {:1hr   0.10
                       :10hr  0.10
                       :100hr 0.10}
                :live {:woody      0.10
                       :herbaceous 0.10}}]
    (is (s/valid? ::config/fuel-moisture config))))

(deftest ^:unit mixed-test
  (let [config {:dead {:1hr   {:type   :geotiff
                               :source (utils/in-file-path resources-path "m1_to_sample.tif")}
                       :10hr  {:type   :geotiff
                               :source (utils/in-file-path resources-path "m10_to_sample.tif")}
                       :100hr {:type   :geotiff
                               :source (utils/in-file-path resources-path "m100_to_sample.tif")}}
                :live {:woody      0.10
                       :herbaceous 0.10}}]
    (is (s/valid? ::config/fuel-moisture config))))
```

9.2 Core Algorithm

9.2.1 gridfire.fuel-models-test

```
(ns gridfire.fuel-models-test
  (:require [clojure.test         :refer [deftest is testing]]
            [gridfire.fuel-models :as f-opt]))

(defn- classical-model-numbers
  []
  (keys f-opt/fuel-models))

(defn- WUI-model-numbers
  []
  (keys f-opt/WUI-model-number->original))

(defn- all-model-numbers
  []
  (keys f-opt/fuel-models-precomputed))

(deftest ^:unit fuel-number-predicates-test
  (testing "The WUI fuel models"
    (testing "are all burnable."
      (is (every? f-opt/is-burnable-fuel-model-number? (WUI-model-numbers))))

    (testing "all support dynamic fuel loading."
      (is (every? f-opt/is-dynamic-fuel-model-number? (WUI-model-numbers)))))

  (testing "Magic numbers:"
    (is (= 204 (apply max (classical-model-numbers))))
    (is (= 303 (apply max (all-model-numbers))))))
```

9.2.2 gridfire.fuel-models-old-test

```
(ns gridfire.fuel-models-old-test
  (:require [clojure.test                :refer [deftest testing is run-tests]]
            [gridfire.behaveplus-results :refer [behaveplus5-surface-fire-values-dry-no-wind-no-slope
                                                 behaveplus5-surface-fire-values-mid-no-wind-no-slope
                                                 sb40-fuel-models
                                                 test-fuel-moisture
                                                 within]]
            ;; FIXME do we want to keep these tests? If so, how to remove that dependency?
```

```
            [gridfire.fuel-models-old    :refer [build-fuel-model
                                                 moisturize]]))

;; Checks live fuel moisture of extinction and dynamic fuel loading for the S&B40 fuel models under fully cured conditions
(deftest ^:unit moisturize-test-dry
  (doseq [fm-number sb40-fuel-models]
    (let [gridfire-fuel-model-dry   (build-fuel-model fm-number)
          gridfire-fuel-model-wet   (moisturize gridfire-fuel-model-dry (test-fuel-moisture :dry))
          gridfire-M_x-live         (* 100 (-> gridfire-fuel-model-wet :M_x :live :herbaceous))
          gridfire-fraction-cured   (if (pos? (-> gridfire-fuel-model-dry :w_o :live :herbaceous))
                                      (* 100 (/ (-> gridfire-fuel-model-wet :w_o :dead :herbaceous)
                                                (-> gridfire-fuel-model-dry :w_o :live :herbaceous)))
                                      0)
          behaveplus-outputs        (-> (:name gridfire-fuel-model-dry)
                                        (behaveplus5-surface-fire-values-dry-no-wind-no-slope))
          behaveplus-M_x-live       (behaveplus-outputs 4)
          behaveplus-fraction-cured (behaveplus-outputs 6)]
      (is (within gridfire-M_x-live behaveplus-M_x-live 6)) ;; all are within 1% except TU2 at -6%
      (is (within gridfire-fraction-cured behaveplus-fraction-cured 0.1)))))

;; Checks live fuel moisture of extinction and dynamic fuel loading for the S&B40 fuel models under 50% cured conditions
(deftest ^:unit moisturize-test-mid
  (doseq [num sb40-fuel-models]
    (let [gridfire-fuel-model-dry   (build-fuel-model num)
          gridfire-fuel-model-wet   (moisturize gridfire-fuel-model-dry (test-fuel-moisture :mid))
          gridfire-M_x-live         (* 100 (-> gridfire-fuel-model-wet :M_x :live :herbaceous))
          gridfire-fraction-cured   (if (pos? (-> gridfire-fuel-model-dry :w_o :live :herbaceous))
                                      (* 100 (/ (-> gridfire-fuel-model-wet :w_o :dead :herbaceous)
                                                (-> gridfire-fuel-model-dry :w_o :live :herbaceous)))
                                      0)
          behaveplus-outputs        (-> (:name gridfire-fuel-model-dry)
                                        (behaveplus5-surface-fire-values-mid-no-wind-no-slope))
          behaveplus-M_x-live       (behaveplus-outputs 4)
          behaveplus-fraction-cured (behaveplus-outputs 6)]
      (is (within gridfire-M_x-live behaveplus-M_x-live 6)) ;; all are within 1% except TU2 at -6%
      (is (within gridfire-fraction-cured behaveplus-fraction-cured 0.1)))))

;; TODO: Add moisturize-test-wet

(comment
  (run-tests)
  )
```

9.2.3 gridfire.common

```
(ns gridfire.common
  (:require [gridfire.fuel-models         :as f-opt]
            [gridfire.grid-lookup         :as grid-lookup]
            [tech.v3.datatype             :as d]
            [tech.v3.datatype.argops      :as da]
            [tech.v3.datatype.functional  :as dfn]
            [tech.v3.tensor               :as t])
  (:import (clojure.lang IFn$LLLLD)))

(set! *unchecked-math* :warn-on-boxed)

(defn calc-emc
  "Computes the Equilibrium Moisture Content (EMC) from rh (relative
   humidity in %) and temp (temperature in F)."
  ^double
  [^double rh ^double temp]
  (/ (double
      (cond (< rh 10.0) (+ 0.03229 (* 0.281073 rh) (* -0.000578 rh temp))
            (< rh 50.0) (+ 2.22749 (* 0.160107 rh) (* -0.01478 temp))
            :else       (+ 21.0606 (* 0.005565 rh rh) (* -0.00035 rh temp) (* -0.483199 rh))))
     30.0))
```

```
(defn calc-fuel-moisture
  ^double
  [^double relative-humidity ^double temperature category size]
  (let [equilibrium-moisture (calc-emc relative-humidity temperature)]
    (case [category size]
      [:dead :1hr]        (+ equilibrium-moisture 0.002)
      [:dead :10hr]       (+ equilibrium-moisture 0.015)
      [:dead :100hr]      (+ equilibrium-moisture 0.025)
      [:live :herbaceous] (* equilibrium-moisture 2.0)
      [:live :woody]      (* equilibrium-moisture 0.5))))

(defn in-bounds?
  "Returns true if the point lies within the bounds [0,rows) by [0,cols)."
  [^long rows ^long cols [^long i ^long j]]
  (and (>= i 0)
       (>= j 0)
       (< i rows)
       (< j cols)))

(defn in-bounds-optimal?
  "Returns true if the point lies within the bounds [0,rows) by [0,cols)."
  [^long rows ^long cols ^long i ^long j]
  (and (>= i 0)
       (>= j 0)
       (< i rows)
       (< j cols)))

(defn burnable-fuel-model?
  ;; NOTE: the motivation for accepting a double-typed argument
  ;; is that (grid-lookup/double-at) return doubles.
  [^double fm-number]
  (f-opt/is-burnable-fuel-model-number? fm-number))

(defn overtakes-lower-probability-fire?
  "True when the cell [i j] has never burned at a :burn-probability level >= bv-burn-probability."
  [^double bv-burn-probability ^double fire-spread-matrix_ij]
  (> bv-burn-probability fire-spread-matrix_ij))

;; FIXME: This logic doesn't look right.
(defn burnable?
  "Returns true if cell [x y] has not yet been ignited (but could be)."
  [fire-spread-matrix fuel-model-matrix [i j] [x y]]
  (and (burnable-fuel-model? (grid-lookup/mget-double-at fuel-model-matrix x y))
       (let [ignition-probability-here   (grid-lookup/mget-double-at fire-spread-matrix x y)
             source-ignition-probability (grid-lookup/mget-double-at fire-spread-matrix i j)]
         (< ignition-probability-here (if (= source-ignition-probability 0.0)
                                        1.0
                                        source-ignition-probability)))))

(defn get-neighbors
  "Returns the eight points adjacent to the passed-in point."
  [[i j]]
  (let [i  (long i)
        j  (long j)
        i- (- i 1)
        i+ (+ i 1)
        j- (- j 1)
        j+ (+ j 1)]
    (list [i- j-] [i- j] [i- j+]
          [i  j-]        [i  j+]
          [i+ j-] [i+ j] [i+ j+])))

(defn burnable-neighbors?
  [fire-spread-matrix fuel-model-matrix num-rows num-cols cell]
  (some #(and (in-bounds? num-rows num-cols %)
              (burnable? fire-spread-matrix fuel-model-matrix cell %))
        (get-neighbors cell)))
```

```
(defn distance-3d
  "Returns the terrain distance between two points in feet."
  ^double
  [elevation-matrix ^double cell-size [i1 j1] [i2 j2]]
  (let [i1 (long i1)
        j1 (long j1)
        i2 (long i2)
        j2 (long j2)
        di (* cell-size (- i1 i2))
        dj (* cell-size (- j1 j2))
        dz (- (grid-lookup/mget-double-at elevation-matrix i1 j1)
              (grid-lookup/mget-double-at elevation-matrix i2 j2))]
    (Math/sqrt (+ (* di di) (* dj dj) (* dz dz)))))

(defn non-zero-indices [tensor]
  (let [[_ cols] (:shape (t/tensor->dimensions tensor))
        indices  (da/argfilter pos? tensor)
        row-idxs (dfn/quot indices cols)
        col-idxs (dfn/rem indices cols)]
    {:row-idxs (d/emap #(d/unchecked-cast % :int64) :int64 row-idxs)
     :col-idxs (d/emap #(d/unchecked-cast % :int64) :int64 col-idxs)}))

(defn non-zero-count [tensor]
  (-> tensor dfn/pos? dfn/sum (d/unchecked-cast :int64)))

;; NOTE I don't see a workable way to primitive-partialize this function given how it's currently used. (Val, 24 Nov 2022)
(defn burnable-cell?
  [get-fuel-model fire-spread-matrix burn-probability num-rows num-cols i j]
  (let [i (long i)
        j (long j)]
    (and (in-bounds-optimal? num-rows num-cols i j)
         (burnable-fuel-model? (grid-lookup/double-at get-fuel-model i j))
         (overtakes-lower-probability-fire? (double burn-probability) (grid-lookup/mget-double-at fire-spread-matrix i j)))))

(defmacro dist-expr
  "A macro expanding to an expression for computing 3D terrain distance.

  i0, j0 must evaluate to positive integers < num-rows, num-cols.
  z0 and z1 must evaluate to elevations (in ft); they may remain unevaluated."
  [num-rows num-cols cell-size
   i0 j0 z0
   i1 j1 z1]
  `(let [i1#      ~i1
         j1#      ~j1
         cs#      ~cell-size
         di#      (* cs# (- ~i0 i1#))
         dj#      (* cs# (- ~j0 j1#))
         di2+dj2# (+ (* di# di#)
                     (* dj# dj#))]
     (Math/sqrt (if (in-bounds-optimal? ~num-rows ~num-cols i1# j1#)
                  (let [dz# (- ~z0 ~z1)]
                    (+ di2+dj2# (* dz# dz#)))
                  di2+dj2#))))

(defn terrain-distance-fn
  "Prepares a primitive-signature function for computing terrain distance,
  which can be invoked very efficiently using macro terrain-distance-invoke."
  [get-elevation ^long num-rows ^long num-cols ^double cell-size]
  (fn between-coords
    ^double [^long i0 ^long j0 ^long i1 ^long j1]
    (dist-expr num-rows num-cols cell-size
               i0 j0 (grid-lookup/double-at get-elevation i0 j0)
               i1 j1 (grid-lookup/double-at get-elevation i1 j1))))

(defmacro terrain-distance-invoke
  "Macro hiding the JVM interop for efficiently invoking the return value of `terrain-distance-fn`."
  [terrain-dist-fn i0 j0 i1 j1]
```

```
  (let [tdf-sym (vary-meta (gensym 'terrain-dist-fn) assoc :tag `IFn$LLLLD)]
    `(let [~tdf-sym ~terrain-dist-fn]
       (.invokePrim ~tdf-sym ~i0 ~j0 ~i1 ~j1))))

(defn terrain-distance-from-cell-getter
  "Prepares a function for computing the distance from an already-identified cell,
  to be invoked using `gridfire.grid-lookup/double-at`."
  [get-elevation num-rows num-cols cell-size i0 j0]
  (let [num-rows  (long num-rows)
        num-cols  (long num-cols)
        cell-size (double cell-size)
        i0        (long i0)
        j0        (long j0)
        z0        (grid-lookup/double-at get-elevation i0 j0)]
    (fn distance-from-cell
      ^double [^long i1 ^long j1]
      (dist-expr num-rows num-cols cell-size
                 i0 j0 z0
                 i1 j1 (grid-lookup/double-at get-elevation i1 j1)))))
```

9.2.4 gridfire.simulations

```
(ns gridfire.simulations
  (:require [clojure.java.io                              :as io]
            [gridfire.binary-output                       :as binary]
            [gridfire.common                              :refer [get-neighbors in-bounds?]]
            [gridfire.conversion                          :refer [min->hour kebab->snake snake->kebab]]
            [gridfire.fire-spread                         :refer [run-fire-spread]]
            [gridfire.grid-lookup                         :as grid-lookup]
            [gridfire.outputs                             :as outputs]
            [gridfire.perturbations.pixel.hash-determined :as pixel-hdp]
            [gridfire.spotting                            :as spotting]
            [gridfire.utils.async                         :as gf-async]
            [gridfire.utils.random                        :refer [my-rand-range]]
            [manifold.deferred                            :as mfd]
            [taoensso.tufte                               :as tufte]
            [tech.v3.datatype                             :as d]
            [tech.v3.datatype.functional                  :as dfn]
            [tech.v3.tensor                               :as t])
  (:import java.util.Random))

(set! *unchecked-math* :warn-on-boxed)

(def ^:dynamic *log-performance-metrics*
  "Whether to log performance monitoring metrics,
  which adds latency to simulations,
  and verbosity to the logs."
  (= "true"
     ;; TIP: to re-def this Var as true in your REPL,
     ;; comment out the following expr and reload the code, using #_
     (System/getProperty "gridfire.simulations.log-performance-metrics")))

(defn layer-snapshot [burn-time-matrix layer-matrix ^double t]
  (d/clone
   (d/emap (fn [^double layer-value ^double burn-time]
             (if (and (not (neg? burn-time)) (<= burn-time t))
               layer-value
               0.0))
           :float64
           layer-matrix
           burn-time-matrix)))

(defn previous-active-perimeter?
  [[i j :as here] matrix]
  (let [[num-rows num-cols] (:shape (t/tensor->dimensions matrix))]
    (and
```

```
     (= (t/mget matrix i j) -1.0)
     (->> (get-neighbors here)
          (filter #(in-bounds? num-rows num-cols %))
          (map #(apply t/mget matrix %))
          (some pos?)))))

(defn to-color-map-values [burn-time-matrix ^double current-clock]
  (t/compute-tensor
   (:shape (t/tensor->dimensions burn-time-matrix))
   (fn [i j]
     (let [^double burn-time (t/mget burn-time-matrix i j)
           delta-hours       (->> (- current-clock burn-time)
                                  min->hour)]
       (cond
         (previous-active-perimeter? [i j] burn-time-matrix) 201
         (= burn-time -1.0)                                  200
         (< 0 delta-hours 5)                                 delta-hours
         (>= delta-hours 5)                                  5
         :else                                               0)))
   :int64))

(defn process-output-layers-timestepped
  [{:keys [simulation-id] :as config}
   {:keys [global-clock burn-time-matrix] :as fire-spread-results}
   name layer timestep envelope]
  (let [global-clock (double global-clock)
        output-times (range 0 (inc global-clock) timestep)]
    (->> output-times
         (mapv
           (fn [output-time]
             (->
               (outputs/exec-in-outputs-writing-pool
                 (fn []
                   (let [matrix (if (= name "burn_history")
                                  (to-color-map-values layer output-time)
                                  (fire-spread-results layer))]
                     (layer-snapshot burn-time-matrix matrix output-time))))
               (mfd/chain
                 (fn [filtered-matrix]
                   (mfd/zip
                     (outputs/output-geotiff config filtered-matrix name envelope simulation-id output-time)
                     (outputs/output-png config filtered-matrix name envelope simulation-id output-time))))
               (gf-async/nil-when-completed))))
         (gf-async/nil-when-all-completed))))

(def layer-name+matrix-key+is-optional
  [["fire_spread"              :fire-spread-matrix]
   ["flame_length"             :flame-length-matrix]
   ["directional_flame_length" :directional-flame-length-matrix true]
   ["fire_line_intensity"      :fire-line-intensity-matrix]
   ["burn_history"             :burn-time-matrix]
   ["spread_rate"              :spread-rate-matrix]
   ["fire_type"                :fire-type-matrix]])

(defn filter-output-layers [output-layers]
  (let [layers-to-filter (set (map (comp kebab->snake name) (keys output-layers)))]
    (filter (fn [[name _]] (contains? layers-to-filter name)) layer-name+matrix-key+is-optional)))

(defn process-output-layers!
  [{:keys [output-layers output-geotiffs? output-pngs?] :as config}
   {:keys [global-clock] :as fire-spread-results}
   envelope
   simulation-id]
  (let [layers (if output-layers
                 (filter-output-layers output-layers)
                 layer-name+matrix-key+is-optional)]
    (->> layers
         (mapv
```

```
          (fn [[name layer is-optional?]]
            (let [kw       (keyword (snake->kebab name))
                  timestep (get output-layers kw)]
              (if (int? timestep)
                (process-output-layers-timestepped config
                                                   fire-spread-results
                                                   name
                                                   layer
                                                   timestep
                                                   envelope)
                (when-some [matrix0 (or (get fire-spread-results layer)
                                        (if is-optional?
                                          nil
                                          (throw (ex-info (format "missing layer %s in fire-spread-results" (pr-str layer))
                                                          {::layer-key layer}))))]
                  (-> (outputs/exec-in-outputs-writing-pool
                       (fn []
                         (-> matrix0
                             ;; TODO that check will never be true, (Val, 03 Nov 2022)
                             ;; since we are comparing a Keyword to a String,
                             ;; but I suspect we've been relying on that bug until now.
                             (cond-> (= layer "burn_history") (to-color-map-values global-clock)))))
                      (mfd/chain
                       (fn [matrix]
                         (mfd/zip
                          (when output-geotiffs?
                            (outputs/output-geotiff config matrix name envelope simulation-id))
                          (when output-pngs?
                            (outputs/output-png config matrix name envelope simulation-id)))))
                      (gf-async/nil-when-completed)))))))
        (gf-async/nil-when-all-completed))))

(defn process-burn-count!
  [{:keys [fire-spread-matrix burn-time-matrix global-clock]}
   burn-count-matrix
   timestep]
  (if (int? timestep)
    (let [global-clock (double global-clock)
          timestep     (long timestep)]
      (doseq [^double clock (range 0 (inc global-clock) timestep)]
        (let [filtered-fire-spread (d/clone
                                    (d/emap (fn [^double layer-value ^double burn-time]
                                              (if (<= burn-time clock)
                                                layer-value
                                                0.0))
                                            :float64
                                            fire-spread-matrix
                                            burn-time-matrix))
              band                (int (quot clock timestep))
              burn-count-matrix-i  (nth burn-count-matrix band)]
          (d/copy! (dfn/+ burn-count-matrix-i filtered-fire-spread) burn-count-matrix-i))))
    (d/copy! (dfn/+ burn-count-matrix fire-spread-matrix) burn-count-matrix)))

(defn process-aggregate-output-layers!
  [{:keys [output-flame-length-sum output-flame-length-max burn-count-matrix flame-length-max-matrix flame-length-sum-matrix
           output-burn-probability spot-count-matrix]} fire-spread-results]
  ;; WARNING this code is probably not thread-safe, as it operates by mutating shared collections. (Val, 24 Jan 2023)
  ;; Race conditions could become likely if simulations are very fast.
  (when-let [timestep output-burn-probability]
    (process-burn-count! fire-spread-results burn-count-matrix timestep))
  (when flame-length-sum-matrix
    (if (= output-flame-length-sum :directional)
      (d/copy! (dfn/+ flame-length-sum-matrix (:directional-flame-length-matrix fire-spread-results))
               flame-length-sum-matrix)
      (d/copy! (dfn/+ flame-length-sum-matrix (:flame-length-matrix fire-spread-results))
               flame-length-sum-matrix)))
  (when flame-length-max-matrix
    (if (= output-flame-length-max :directional)
```

```
      (d/copy! (dfn/max flame-length-max-matrix (:directional-flame-length-matrix fire-spread-results))
               flame-length-max-matrix)
      (d/copy! (dfn/max flame-length-max-matrix (:flame-length-matrix fire-spread-results))
               flame-length-max-matrix)))
  (when spot-count-matrix
    (d/copy! (dfn/+ spot-count-matrix (:spot-matrix fire-spread-results))
             spot-count-matrix)))

(defn process-binary-output!
  [{:keys [output-binary? output-directory]}
   {:keys [burn-time-matrix flame-length-matrix spread-rate-matrix fire-type-matrix]}
   ^long simulation]
  (when output-binary?
    (outputs/exec-in-outputs-writing-pool
      (fn []
        (let [output-name (format "toa_0001_%05d.bin" (inc simulation))
              output-path (if output-directory
                            (.getPath (io/file output-directory output-name))
                            output-name)]
          ;; NOTE that's not parallelizable, as the matrices get written to the same file.
          (binary/write-matrices-as-binary output-path
                                           [:float :float :float :int]
                                           [(->> burn-time-matrix
                                                 (d/emap (fn ^double [^double x] (if (pos? x) (* 60.0 x) x)) :float64)
                                                 (d/clone))
                                            flame-length-matrix
                                            spread-rate-matrix
                                            fire-type-matrix])))))))
(defn cells-to-acres ^double
  [^double cell-size ^long num-cells]
  (let [acres-per-cell (/ (* cell-size cell-size) 43560.0)]
    (* acres-per-cell num-cells)))

(defn summarize-fire-spread-results
  [fire-spread-results cell-size]
  (let [flame-lengths              (filterv pos? (t/tensor->buffer (:flame-length-matrix fire-spread-results)))
        fire-line-intensities      (filterv pos? (t/tensor->buffer (:fire-line-intensity-matrix fire-spread-results)))
        burned-cells               (count flame-lengths)
        surface-fire-size          (cells-to-acres cell-size (:surface-fire-count fire-spread-results))
        crown-fire-size            (cells-to-acres cell-size (:crown-fire-count fire-spread-results))
        flame-length-mean          (/ (dfn/sum flame-lengths) burned-cells)
        fire-line-intensity-mean   (/ (dfn/sum fire-line-intensities) burned-cells)
        flame-length-stddev        (->> flame-lengths
                                        (d/emap #(Math/pow (- flame-length-mean ^double %) 2.0) nil)
                                        (dfn/sum)
                                        (#(/ ^double % burned-cells))
                                        (Math/sqrt))
        fire-line-intensity-stddev (->> fire-line-intensities
                                        (d/emap #(Math/pow (- fire-line-intensity-mean ^double %) 2.0) nil)
                                        (dfn/sum)
                                        (#(/ ^double % burned-cells))
                                        (Math/sqrt))]
    {:fire-size                  (+ surface-fire-size crown-fire-size)
     :surface-fire-size          surface-fire-size
     :crown-fire-size            crown-fire-size
     :flame-length-mean          flame-length-mean
     :flame-length-stddev        flame-length-stddev
     :fire-line-intensity-mean   fire-line-intensity-mean
     :fire-line-intensity-stddev fire-line-intensity-stddev
     :spot-count                 (:spot-count fire-spread-results)}))

(defn- matrix-or-i
  [inputs layer-name i]
  (or (get inputs (-> (name layer-name)
                      (str "-matrix")
                      keyword))
      (get-in inputs [(-> (name layer-name)
                          (str "-samples")
```

```
                        keyword)
                    i])))

(defn- get-index-multiplier
  [inputs layer-name]
  (get inputs (-> (name layer-name)
                  (str "-index-multiplier")
                  keyword)))

(def n-buckets 1024)

(defmulti perturbation-getter (fn [_inputs perturb-config _rand-gen] (:spatial-type perturb-config)))

(defn- sample-h->perturb
  [perturb-config rand-gen]
  (let [[range-min range-max] (:range perturb-config)
        gen-perturbation      (fn gen-in-range [_h]
                                (my-rand-range rand-gen range-min range-max))
        h->perturb            (pixel-hdp/gen-hash->perturbation n-buckets gen-perturbation)]
    h->perturb))

(defmethod perturbation-getter :global
  [_inputs perturb-config rand-gen]
  (let [h->perturb (sample-h->perturb perturb-config rand-gen)]
    (fn get-global-perturbation
      (^double [^long _i ^long _j]
       ;; NOTE we used to resolve {:spatial-type :global} perturbations using an atom-held cache,
       ;; which is not a problem performance-wise,
       ;; but even in this case a Hash-Determined strategy is better,
       ;; because it makes the perturbation more robustly reproducible,
       ;; as it won't be affected by other uses of the random generator
       ;; during the simulation loop.
       (pixel-hdp/resolve-perturbation-for-coords h->perturb -1 -1 -1))
      (^double [^long b ^long _i ^long _j]
       (pixel-hdp/resolve-perturbation-for-coords h->perturb b -1 -1)))))

(defmethod perturbation-getter :pixel
  [_inputs perturb-config rand-gen]
  (let [h->perturb (sample-h->perturb perturb-config rand-gen)]
    (fn get-pixel-perturbation
      (^double [^long i ^long j]
       (pixel-hdp/resolve-perturbation-for-coords h->perturb i j))
      (^double [^long b ^long i ^long j]
       (pixel-hdp/resolve-perturbation-for-coords h->perturb b i j)))))

(defn- ppsc-tuple
  "Computes a tuple of pixels per supergrid cell."
  [{:keys [num-rows num-cols max-runtime] :as _inputs} pert]
  (let [max-b                  (/ (double max-runtime) 60.)
        [sb si sj]             (:gridfire.perturbation.smoothed-supergrid/supergrid-size pert)
        [ppsc-b ppsc-i ppsc-j] (mapv (fn pixels-per-supergrid-cell [s m] (/ (double m) (long s)))
                                     [sb si sj]
                                     [max-b num-rows num-cols])]
    [ppsc-b ppsc-i ppsc-j]))

(defmethod perturbation-getter :smoothed-supergrid
  [inputs perturb-config rand-gen]
  (let [{:keys [range]} perturb-config
        [rng-min rng-max] range
        gen-perturbation   (fn gen-in-range [_h]
                             (my-rand-range rand-gen rng-min rng-max))
        [sb si sj]         (:gridfire.perturbation.smoothed-supergrid/supergrid-size perturb-config)
        [pb pi pj]         (ppsc-tuple inputs perturb-config)
        ppsc-b             (double pb)
        ppsc-i             (double pi)
        ppsc-j             (double pj)
        gen-sampled-grid   (reduce (fn [g s]
                                     (fn gen-tensor []
```

```
                                    (vec (repeatedly s g))))
                                #(gen-perturbation nil)
                                (->> [sb si sj]
                                     (map (fn [s] (+ (long s) 2)))
                                     (reverse)))
        sampled-grid       (t/->tensor (gen-sampled-grid))
        ;; Why offset the supergrid? Pixels in the interior of supergrid cells
        ;; tend to have different distributions (less variance, smoother local variation)
        ;; than those near the boundaries.
        ;; Randomly offsetting ensures that all cells have an equal change
        ;; of being near the supergrid cell boundaries.
        offsets            (repeatedly 3 #(my-rand-range rand-gen 0. 1.))
        [o-b o-i o-j]      offsets
        o-b                (double o-b)
        o-i                (double o-i)
        o-j                (double o-j)
        lin-terpolate      (fn lin-terpolate ^double [^double frac ^double v0 ^double v1]
                             (+
                              (* (- 1. frac) v0)
                              (* frac v1)))
        get-pert-at-coords (fn average-cube-corners ^double [^long b ^long i ^long j]
                             ;; Implements a linear spline smoothing algorithm.
                             (let [b-pos  (-> b (/ ppsc-b) (+ o-b))
                                   i-pos  (-> i (/ ppsc-i) (+ o-i))
                                   j-pos  (-> j (/ ppsc-j) (+ o-j))

                                   b-pos- (-> b-pos (Math/floor) (long))
                                   b-pos+ (inc b-pos-)
                                   b-posf (- b-pos b-pos-)
                                   i-pos- (-> i-pos (Math/floor) (long))
                                   i-pos+ (inc i-pos-)
                                   i-posf (- i-pos i-pos-)
                                   j-pos- (-> j-pos (Math/floor) (long))
                                   j-pos+ (inc j-pos-)
                                   j-posf (- j-pos j-pos-)

                                   ;; FIXME grid-lookup/mget-double-at or similar
                                   p000   (t/mget sampled-grid b-pos- i-pos- j-pos-)
                                   p001   (t/mget sampled-grid b-pos- i-pos- j-pos+)
                                   p010   (t/mget sampled-grid b-pos- i-pos+ j-pos-)
                                   p011   (t/mget sampled-grid b-pos- i-pos+ j-pos+)
                                   p100   (t/mget sampled-grid b-pos+ i-pos- j-pos-)
                                   p101   (t/mget sampled-grid b-pos+ i-pos- j-pos+)
                                   p110   (t/mget sampled-grid b-pos+ i-pos+ j-pos-)
                                   p111   (t/mget sampled-grid b-pos+ i-pos+ j-pos+)]
                               (lin-terpolate b-posf
                                              (lin-terpolate i-posf
                                                             (lin-terpolate j-posf p000 p001)
                                                             (lin-terpolate j-posf p010 p011))
                                              (lin-terpolate i-posf
                                                             (lin-terpolate j-posf p100 p101)
                                                             (lin-terpolate j-posf p110 p111)))))]
    (fn get-pixel-perturbation
      (^double [^long i ^long j] (get-pert-at-coords 0 i j))
      (^double [^long b ^long i ^long j] (get-pert-at-coords b i j)))))

(defn- grid-getter
  "Pre-computes a 'getter' for resolving perturbed input values in the GridFire space-time grid.

  Returns a 'getter' object, to be called with (gridfire.grid-lookup/double-at getter b i j)
  in order to resolve the input value at coordinates [b i j].
  At the time of writing, this getter object will be a function, but that's an implementation detail,
  do not rely on it.

  The perturbed value might be lazily computed, and its computation may involve pseudo-randomness,
  but calling (grid-lookup/double-at ...) several times at the same coordinates
  will always return the same result."
  [{:keys [perturbations] :as inputs} rand-gen layer-name i]
```

```
  {:post [(or (nil? %) (grid-lookup/suitable-for-primitive-lookup? %))]}
  (when-let [matrix-or-num (matrix-or-i inputs layer-name i)]
    (let [index-multiplier (get-index-multiplier inputs layer-name)
          tensor-lookup    (grid-lookup/tensor-cell-getter matrix-or-num nil)
          get-unperturbed  (if (number? matrix-or-num)
                             tensor-lookup
                             (if (nil? index-multiplier)
                               tensor-lookup
                               (let [index-multiplier (double index-multiplier)]
                                 (fn multiplied-tensor-lookup
                                   (^double [^long i ^long j]
                                    (let [row (long (* i index-multiplier))
                                          col (long (* j index-multiplier))]
                                      (grid-lookup/double-at tensor-lookup row col)))
                                   (^double [^long b ^long i ^long j]
                                    (let [row (long (* i index-multiplier))
                                          col (long (* j index-multiplier))]
                                      (grid-lookup/double-at tensor-lookup b row col))))))))
          get-perturbation (if-some [perturb-config (get perturbations layer-name)]
                             (perturbation-getter inputs perturb-config rand-gen)
                             nil)]
      (fn grid-getter
        (^double [^long i ^long j]
         (cond-> (grid-lookup/double-at get-unperturbed i j)
           (some? get-perturbation)
           (-> (+ (grid-lookup/double-at get-perturbation i j))
               (max 0.))))
        (^double [^long b ^long i ^long j]
         (cond-> (grid-lookup/double-at get-unperturbed b i j)
           (some? get-perturbation)
           (-> (+ (grid-lookup/double-at get-perturbation b i j))
               (max 0.)))))))

(defrecord SimulationInputs
    [^long num-rows
     ^long num-cols
     ^double cell-size
     ^Random rand-gen
     initial-ignition-site
     ^double ignition-start-time
     ignition-start-timestamp
     burn-period-start
     burn-period-end
     ^double max-runtime
     compute-directional-values?
     fuel-number->spread-rate-adjustment-array-lookup
     get-aspect
     get-canopy-base-height
     get-canopy-cover
     get-canopy-height
     get-crown-bulk-density
     get-elevation
     get-foliar-moisture
     get-fuel-model
     get-fuel-moisture-dead-100hr
     get-fuel-moisture-dead-10hr
     get-fuel-moisture-dead-1hr
     get-fuel-moisture-live-herbaceous
     get-fuel-moisture-live-woody
     get-relative-humidity
     get-slope
     get-suppression-difficulty-index
     get-temperature
     get-wind-from-direction
     get-wind-speed-20ft
     ^double ellipse-adjustment-factor
     ^boolean crowning-disabled?
     ^boolean grass-suppression?
```

```
    sdi-containment-overwhelming-area-growth-rate
    sdi-reference-suppression-speed
    sdi-sensitivity-to-difficulty
    memoization
    spotting
    suppression-coefficient
    suppression-dt])

(defn prepare-simulation-inputs
  [^long i
   {:keys
    [num-rows num-cols crowning-disabled? grass-suppression? ignition-matrix cell-size max-runtime-samples
     ignition-rows ignition-cols ellipse-adjustment-factor-samples random-seed ignition-start-times spotting
     burn-period-samples ignition-start-timestamps
     output-flame-length-sum output-flame-length-max output-layers]
    :as inputs}]
  (let [rand-gen           (if random-seed (Random. (+ ^long random-seed i)) (Random.))
        burn-period        (burn-period-samples i)
        simulation-inputs  {:num-rows                                      num-rows
                            :num-cols                                      num-cols
                            :cell-size                                     cell-size
                            :rand-gen                                      rand-gen
                            :initial-ignition-site                         (or ignition-matrix
                                                                               [(ignition-rows i) (ignition-cols i)])
                            :ignition-start-time                           (get ignition-start-times i 0.0)
                            :ignition-start-timestamp                      (get ignition-start-timestamps i)
                            :burn-period-start                             (:burn-period-start burn-period)
                            :burn-period-end                               (:burn-period-end burn-period)
                            :max-runtime                                   (max-runtime-samples i)
                            :compute-directional-values?                   (or (= output-flame-length-max :directional)
                                                                               (= output-flame-length-sum :directional)
                                                                               (:directional-flame-length output-layers))
                            :get-aspect                                    (grid-getter inputs rand-gen :aspect i)
                            :get-canopy-base-height                        (grid-getter inputs rand-gen :canopy-base-height i)
                            :get-canopy-cover                              (grid-getter inputs rand-gen :canopy-cover i)
                            :get-canopy-height                             (grid-getter inputs rand-gen :canopy-height i)
                            :get-crown-bulk-density                        (grid-getter inputs rand-gen :crown-bulk-density i)
                            :get-elevation                                 (grid-getter inputs rand-gen :elevation i)
                            :get-foliar-moisture                           (grid-getter inputs rand-gen :foliar-moisture i)
                            :get-fuel-model                                (grid-getter inputs rand-gen :fuel-model i)
                            :get-fuel-moisture-dead-100hr                  (grid-getter inputs rand-gen :fuel-moisture-dead-100h
                            :get-fuel-moisture-dead-10hr                   (grid-getter inputs rand-gen :fuel-moisture-dead-10hr
                            :get-fuel-moisture-dead-1hr                    (grid-getter inputs rand-gen :fuel-moisture-dead-1hr
                            :get-fuel-moisture-live-herbaceous             (grid-getter inputs rand-gen :fuel-moisture-live-herb
                            :get-fuel-moisture-live-woody                  (grid-getter inputs rand-gen :fuel-moisture-live-wood
                            :get-relative-humidity                         (grid-getter inputs rand-gen :relative-humidity i)
                            :get-slope                                     (grid-getter inputs rand-gen :slope i)
                            :get-suppression-difficulty-index              (grid-getter inputs rand-gen :suppression-difficulty-
                            :get-temperature                               (grid-getter inputs rand-gen :temperature i)
                            :get-wind-from-direction                       (grid-getter inputs rand-gen :wind-from-direction i)
                            :get-wind-speed-20ft                           (grid-getter inputs rand-gen :wind-speed-20ft i)
                            :crowning-disabled?                            (true? crowning-disabled?) ; NOTE not using samples y
                            :ellipse-adjustment-factor                     (ellipse-adjustment-factor-samples i)
                            :grass-suppression?                            (true? grass-suppression?)
                            :sdi-containment-overwhelming-area-growth-rate (some-> (:sdi-containment-overwhelming-area-growth-ra
                            :sdi-reference-suppression-speed               (some-> (:sdi-reference-suppression-speed-samples inp
                            :sdi-sensitivity-to-difficulty                 (some-> (:sdi-sensitivity-to-difficulty-samples input
                            :memoization                                   (:memoization inputs)
                            :spotting                                      (spotting/sample-spotting-params spotting rand-gen)
                            :fuel-number->spread-rate-adjustment-array-lookup (some-> (:fuel-number->spread-rate-adjustment-array-l
                            :suppression-coefficient                       (some-> (:suppression-coefficient-samples inputs) (ge
                            :suppression-dt                                (some-> (:suppression-dt-samples inputs) (get i))}]
    (map->SimulationInputs simulation-inputs)))

(defn process-simulation-results!
  [^long i
   {:keys [output-csvs? envelope cell-size ignition-rows ignition-cols] :as inputs}
   simulation-inputs
```

```
   simulation-results]
  (when simulation-results
    (->
      (mfd/zip
       (process-output-layers! inputs simulation-results envelope i)
       (process-aggregate-output-layers! inputs simulation-results)
       (process-binary-output! inputs simulation-results i))
      (gf-async/nil-when-completed)
      (deref)))
  (when output-csvs?
    (-> simulation-inputs
        (dissoc :get-aspect
                :get-canopy-height
                :get-canopy-base-height
                :get-canopy-cover
                :get-crown-bulk-density
                :get-fuel-model
                :get-slope
                :get-elevation
                :get-temperature
                :get-relative-humidity
                :get-wind-speed-20ft
                :get-wind-from-direction
                :get-fuel-moisture-dead-1hr
                :get-fuel-moisture-dead-10hr
                :get-fuel-moisture-dead-100hr
                :get-fuel-moisture-live-herbaceous
                :get-fuel-moisture-live-woody
                :get-foliar-moisture)
        (merge {:simulation         (inc i)
                :ignition-row       (get ignition-rows i)
                :ignition-col       (get ignition-cols i)
                :global-clock       (:global-clock simulation-results)
                :exit-condition     (:exit-condition simulation-results :no-fire-spread)
                :surface-fire-count (:surface-fire-count simulation-results)
                :crown-fire-count   (:crown-fire-count simulation-results)
                :spot-count         (:spot-count simulation-results)})
        (merge
         (if simulation-results
           (tufte/p
            :summarize-fire-spread-results
            (summarize-fire-spread-results simulation-results cell-size))
           {:fire-size                  0.0
            :flame-length-mean          0.0
            :flame-length-stddev        0.0
            :fire-line-intensity-mean   0.0
            :fire-line-intensity-stddev 0.0})))))

(defn run-simulation!
  [^long i inputs]
  (tufte/profile
   {:id :run-simulation}
   (let [simulation-inputs  (prepare-simulation-inputs i inputs)
         simulation-results (tufte/p :run-fire-spread
                                     (run-fire-spread simulation-inputs))]
     (process-simulation-results! i inputs simulation-inputs simulation-results))))
```

9.2.5 gridfire.simulations-test

```
(ns gridfire.simulations-test
  (:require [gridfire.grid-lookup :as grid-lookup]
            [gridfire.simulations :as simulations]
            [clojure.test :refer [deftest is testing]]
            [tech.v3.tensor :as t])
  (:import java.util.Random))
```

```
(defn- identical-matrix [band width height value]
  (t/->tensor
   (into-array
    (repeat band
            (into-array
             (repeat height
                     (into-array
                      (repeat width value)))))))))

(deftest ^:unit get-layer-fn-test
  (testing "no perturbations"
    (testing "scalar"
      (let [inputs       {:perturbations       nil
                          :temperature-samples [10.]}
            get-layer-fn (#'simulations/grid-getter inputs (Random. 1234) :temperature 0)]

        (is (= 10.0 (grid-lookup/double-at get-layer-fn 0 0 0)))))

    (testing "matrix"
      (let [inputs       {:perturbations      nil
                          :temperature-matrix (identical-matrix 72 10. 10. 1.0)}
            get-layer-fn (#'simulations/grid-getter inputs (Random. 1234) :temperature 0)]

        (is (= 1.0 (grid-lookup/double-at get-layer-fn 0 0 0))))))

  (testing "with perturbations"
    (testing "scalar"
      (testing "spatial type pixel"
        (let [inputs       {:perturbations       {:temperature {:spatial-type :pixel
                                                                 :range        [-1.0 1.0]}}
                            :temperature-samples [10]}
              get-layer-fn (#'simulations/grid-getter inputs (Random. 1234) :temperature 0)]

          (is (<= 9.0 (grid-lookup/double-at get-layer-fn 0 0 0) 11.0))))

      (testing "spatial type global"
        (let [inputs {:perturbations       {:temperature {:spatial-type :global
                                                          :range        [-1.0 1.0]}}
                      :temperature-samples [10]}

              get-layer-fn (#'simulations/grid-getter inputs (Random. 1234) :temperature 0)]

          (is (<= 9.0 (grid-lookup/double-at get-layer-fn 0 0 0) 11.0)))))

    (testing "matrix"
      (testing "spatial type pixel"
        (let [inputs       {:perturbations      {:temperature {:spatial-type :pixel
                                                               :range        [-1.0 1.0]}}
                            :temperature-matrix (identical-matrix 72 100 100 1.0)}
              get-layer-fn (#'simulations/grid-getter inputs (Random. 1234) :temperature 0)]

          (is (<= 0.0 (grid-lookup/double-at get-layer-fn 0 0 0) 2.0))))

      (testing "spatial type global"
        (let [inputs       {:perturbations      {:temperature {:spatial-type :global
                                                               :range        [-1.0 1.0]}}
                            :temperature-matrix (identical-matrix 72 100 100 1.0)}
              get-layer-fn (#'simulations/grid-getter inputs (Random. 1234) :temperature 0)]

          (is (<= 0.0 (grid-lookup/double-at get-layer-fn 0 0 0) 2.0)))))))
```

9.2.6 gridfire.surface-fire-test

```
(ns gridfire.surface-fire-test
  (:require [clojure.test               :refer [deftest testing is run-tests]]
            [gridfire.fuel-models-old       :refer [fuel-models build-fuel-model moisturize]]
```

```
            [gridfire.surface-fire-old       :refer [anderson-flame-depth
                                                     grass-fuel-model?
                                                     byram-fire-line-intensity
                                                     byram-flame-length
                                                     rothermel-surface-fire-spread-no-wind-no-slope
                                                     rothermel-surface-fire-spread-max
                                                     wind-adjustment-factor
                                                     wind-adjustment-factor-elmfire]]
            [gridfire.conversion         :refer [ft->m mph->fpm]]
            [gridfire.behaveplus-results :refer [behaveplus5-surface-fire-values-dry-no-wind-no-slope
                                                 behaveplus5-surface-fire-values-mid-no-wind-no-slope
                                                 behaveplus5-surface-fire-values-mid-with-wind-and-slope
                                                 behaveplus5-surface-fire-values-mid-with-cross-wind-and-slope-90-180
                                                 behaveplus5-surface-fire-values-mid-with-cross-wind-and-slope-135-180
                                                 sb40-fuel-models
                                                 test-fuel-moisture
                                                 within]]))

;; Tests fuel model weighting factors, rothermel equations, and byram's flame length and fire line intensity under fully cured condi
(deftest ^:unit rothermel-surface-fire-spread-no-wind-no-slope-test-dry
  (testing "Fully cured conditions."
    (doseq [number sb40-fuel-models]
      (let [gridfire-fuel-model                                    (moisturize (build-fuel-model number) (test-fuel-moisture :dry)
            {:keys [spread-rate reaction-intensity residence-time]} (rothermel-surface-fire-spread-no-wind-no-slope gridfire-fuel-m
            corrected-spread-rate-ch-per-hr                        (/ spread-rate 1.1)
            fire-line-intensity                                    (->> (anderson-flame-depth spread-rate residence-time)
                                                                        (byram-fire-line-intensity reaction-intensity))
            flame-length                                           (byram-flame-length fire-line-intensity)
            [ros_max fli fl ri _ rt _]                             (behaveplus5-surface-fire-values-dry-no-wind-no-slope (:name gr
        (is (within ros_max corrected-spread-rate-ch-per-hr 0.1))
        (is (within ri reaction-intensity 12.0))
        (is (within rt residence-time 0.01))
        (is (within fli fire-line-intensity 0.5))
        (is (within fl flame-length 0.05)))))

  (testing "Fully cured conditions with grass fuel suppression enabled."
    (doseq [number sb40-fuel-models]
      (let [gridfire-fuel-model                                    (moisturize (build-fuel-model number) (test-fuel-moisture :dry)
            {:keys [spread-rate reaction-intensity residence-time]} (rothermel-surface-fire-spread-no-wind-no-slope gridfire-fuel-m
            corrected-spread-rate                                  (if (grass-fuel-model? number) (* 2.0 spread-rate) spread-rate)
            corrected-spread-rate-ch-per-hr                        (/ corrected-spread-rate 1.1)
            fire-line-intensity                                    (->> (anderson-flame-depth corrected-spread-rate residence-time
                                                                        (byram-fire-line-intensity reaction-intensity))
            flame-length                                           (byram-flame-length fire-line-intensity)
            [ros_max fli fl ri _ rt _]                             (behaveplus5-surface-fire-values-dry-no-wind-no-slope (:name gr
        (is (within ros_max corrected-spread-rate-ch-per-hr 0.1))
        (is (within ri reaction-intensity 12.0))
        (is (within rt residence-time 0.01))
        (is (within fli fire-line-intensity 0.5))
        (is (within fl flame-length 0.05))))))

;; Tests fuel model weighting factors, rothermel equations, and byram's flame length and fire line intensity under 50% cured conditio
(deftest ^:unit rothermel-surface-fire-spread-no-wind-no-slope-test-mid
  (doseq [number sb40-fuel-models]
    (let [gridfire-fuel-model                                    (moisturize (build-fuel-model number) (test-fuel-moisture :mid))
          {:keys [spread-rate reaction-intensity residence-time]} (rothermel-surface-fire-spread-no-wind-no-slope gridfire-fuel-mod
          corrected-spread-rate-ch-per-hr                        (/ spread-rate 1.1)
          fire-line-intensity                                    (->> (anderson-flame-depth spread-rate residence-time)
                                                                      (byram-fire-line-intensity reaction-intensity))
          flame-length                                           (byram-flame-length fire-line-intensity)
          [ros_max fli fl ri _ rt _]                             (behaveplus5-surface-fire-values-mid-no-wind-no-slope (:name grid
      (is (within ros_max corrected-spread-rate-ch-per-hr 0.1))
      (is (within ri reaction-intensity 12.0))
      (is (within rt residence-time 0.01))
      (is (within fli fire-line-intensity 0.85))
      (is (within fl flame-length 0.05)))))

;; Tests fuel model weighting factors, rothermel equations, and
```

```
;; byram's flame length and fire line intensity under 50% cured
;; conditions with 10mph wind speed and 20% slope. Also tests phi_W
;; and phi_S from the rothermel equations.
(deftest ^:unit rothermel-surface-fire-spread-with-wind-and-slope-test-mid
  (doseq [number sb40-fuel-models]
    (let [gridfire-fuel-model                                    (moisturize (build-fuel-model number) (test-fuel-moisture :mid))
          {:keys [_spread-rate reaction-intensity residence-time
                  get-phi_W get-phi_S] :as spread-info-min}       (rothermel-surface-fire-spread-no-wind-no-slope gridfire-fuel-mod
          midflame-wind-speed                                    (mph->fpm 10.0)
          slope                                                  0.2 ;; vertical feet/horizontal feet
          {:keys [max-spread-rate _max-spread-direction
                  _effective-wind-speed _eccentricity]}            (rothermel-surface-fire-spread-max spread-info-min
                                                                                                      midflame-wind-speed 0.0 slop
          max-spread-rate-ch-per-hr                              (/ max-spread-rate 1.1)
          phi_W                                                  (get-phi_W midflame-wind-speed)
          phi_S                                                  (get-phi_S slope)
          fire-line-intensity                                    (->> (anderson-flame-depth max-spread-rate residence-time)
                                                                      (byram-fire-line-intensity reaction-intensity))
          flame-length                                           (byram-flame-length fire-line-intensity)
          [ros_max fli fl ri _ rt _ pW pS]                       (behaveplus5-surface-fire-values-mid-with-wind-and-slope (:name g
      (is (within ros_max max-spread-rate-ch-per-hr 0.3))
      (is (within ri reaction-intensity 12.0))
      (is (within rt residence-time 0.01))
      (is (within fli fire-line-intensity 25.1))
      (is (within fl flame-length 0.05))
      (is (within pW phi_W 0.1))
      (is (within pS phi_S 0.05)))))

;; Tests fuel model weighting factors, rothermel equations, and
;; byram's flame length and fire line intensity under 50% cured
;; conditions with 10mph wind speed and 20% slope. Also tests max
;; spread direction and effective wind speed for cross-slope winds.
(deftest ^:unit rothermel-surface-fire-spread-with-cross-wind-and-slope-test-mid-90-180
  (doseq [number sb40-fuel-models]
    (let [gridfire-fuel-model                                    (moisturize (build-fuel-model number) (test-fuel-moisture :mid))
          {:keys [reaction-intensity residence-time
                  _get-phi_W _get-phi_S] :as spread-info-min}       (rothermel-surface-fire-spread-no-wind-no-slope gridfire-fuel-m
          midflame-wind-speed                                    (mph->fpm 10.0)
          slope                                                  0.2 ;; vertical feet/horizontal feet
          {:keys [max-spread-rate max-spread-direction
                  effective-wind-speed _eccentricity]}             (rothermel-surface-fire-spread-max spread-info-min
                                                                                                      midflame-wind-speed 90.0 slope
          max-spread-rate-ch-per-hr                              (/ max-spread-rate 1.1)
          effective-wind-speed-mph                               (/ effective-wind-speed 88.0)
          fire-line-intensity                                    (->> (anderson-flame-depth max-spread-rate residence-time)
                                                                      (byram-fire-line-intensity reaction-intensity))
          flame-length                                           (byram-flame-length fire-line-intensity)
          [ros_max fli fl max_theta eff_wsp]                     (behaveplus5-surface-fire-values-mid-with-cross-wind-and-slope-90
      (is (within ros_max max-spread-rate-ch-per-hr 0.3))
      (is (within fli fire-line-intensity 25.1))
      (is (within fl flame-length 0.06))
      (is (within max_theta max-spread-direction 0.5))
      (is (within eff_wsp effective-wind-speed-mph 0.1)))))

;; Tests fuel model weighting factors, rothermel equations, and
;; byram's flame length and fire line intensity under 50% cured
;; conditions with 10mph wind speed and 20% slope. Also tests max
;; spread direction and effective wind speed for cross-slope winds.
(deftest ^:unit rothermel-surface-fire-spread-with-cross-wind-and-slope-test-mid-135-180
  (doseq [number sb40-fuel-models]
    (let [gridfire-fuel-model                                    (moisturize (build-fuel-model number) (test-fuel-moisture :mid))
          {:keys [reaction-intensity residence-time
                  _get-phi_W _get-phi_S] :as spread-info-min}       (rothermel-surface-fire-spread-no-wind-no-slope gridfire-fuel-m
          midflame-wind-speed                                    (mph->fpm 10.0)
          slope                                                  0.2 ;; vertical feet/horizontal feet
          {:keys [max-spread-rate max-spread-direction
                  effective-wind-speed eccentricity]}            (rothermel-surface-fire-spread-max spread-info-min
                                                                                                      midflame-wind-speed 135.0 slop
```

```
          max-spread-rate-ch-per-hr                                    (/ max-spread-rate 1.1)
          effective-wind-speed-mph                                     (/ effective-wind-speed 88.0)
          fire-line-intensity                                          (->> (anderson-flame-depth max-spread-rate residence-time)
                                                                            (byram-fire-line-intensity reaction-intensity))
          flame-length                                                 (byram-flame-length fire-line-intensity)
          [ros_max fli fl max_theta eff_wsp lw]                        (behaveplus5-surface-fire-values-mid-with-cross-wind-and-slope-13
          behaveplus-eccentricity                                      (/ (Math/sqrt (- (Math/pow lw 2.0) 1.0)) lw)]
      (is (within ros_max max-spread-rate-ch-per-hr 1.7))
      (is (within fli fire-line-intensity 25.1))
      (is (within fl flame-length 0.06))
      (is (within max_theta max-spread-direction 0.5))
      (is (within eff_wsp effective-wind-speed-mph 0.1))
      (is (within behaveplus-eccentricity eccentricity 0.01)))))

(deftest ^:unit wind-adjustment-factor-test
  (doseq [fuel-bed-depth (map second (vals (select-keys fuel-models sb40-fuel-models)))] ;; ft
    (doseq [canopy-height (range 0 121 10)] ;; ft
      (doseq [canopy-cover (range 0 101 10)] ;; %
        (is (within (wind-adjustment-factor fuel-bed-depth canopy-height canopy-cover)
                    (wind-adjustment-factor-elmfire fuel-bed-depth (ft->m canopy-height) (* 0.01 canopy-cover))
                    0.001))))))

;; TODO: Add R_theta test

(comment
  (run-tests)
  )
```

9.2.7 gridfire.suppression-test

```
(ns gridfire.suppression-test
  (:require
   [clojure.test :refer [are deftest testing is]]
   [gridfire.suppression :as su]))


(deftest ^:unit angle-cw-from-east-test
  (are [i j expected] (= (#'gridfire.suppression/angle-cw-from-east i j 10 10) expected)
    10 11 0.0     ;E
    11 11 45.0    ;SE
    11 10 90.0    ;S
    11 9  135.0   ;SW
    10 9  180.0   ;W
    9  9  225.0   ;NW
    9  10 270.0   ;N
    9  11 315.0)) ;NE

(deftest ^:unit nearest-floor-test
  (let [slice-size 5]
    (are [degree expected-bin] (= (#'gridfire.suppression/nearest-angular-slice degree slice-size) expected-bin)
      0.0  0.0
      2.5  0.0
      5.0  1.0
      7.5  1.0
      10.0 2.0
      12.5 2.0
      15.0 3.0)))

(deftest ^:unit combine-remove-average-test
  (let [[avg-old count-old] [100.0 10]
        [avg-new count-new] [75.0  5]]
    (is (= avg-old
           (-> (#'gridfire.suppression/combine-average avg-old count-old avg-new count-new)
               (#'gridfire.suppression/remove-average (+ count-old count-new) avg-new count-new)))
        "should get the original average if adding and then removing the same average.")))
```

```
(deftest ^:unit compute-contiguous-slices-test
  (testing "simple case"
    (let [num-cells-to-suppress 100
          avg-dsr-data          {0.0 [2.0 25]
                                 1.0 [2.0 25]
                                 2.0 [2.0 25]
                                 3.0 [2.0 25]}]

      (is (= {['(2.0 1.0 0.0 3.0) 2.0] 100}
             (#'gridfire.suppression/compute-contiguous-slices num-cells-to-suppress avg-dsr-data)))))

  (testing "segments sorted by average spread-rate"
    (let [num-cells-to-suppress 6
          avg-dsr-data          {0.0 [6.0 2]
                                 1.0 [5.0 2]
                                 2.0 [4.0 2]
                                 3.0 [3.0 2]
                                 4.0 [2.0 2]
                                 5.0 [1.0 2]}]
      (is (= {['(5.0 4.0 3.0) 2.0] 6
              ['(0.0 5.0 4.0) 3.0] 6
              ['(1.0 0.0 5.0) 4.0] 6
              ['(2.0 1.0 0.0) 5.0] 6}
             (#'gridfire.suppression/compute-contiguous-slices num-cells-to-suppress avg-dsr-data)))))

  (testing "lowest averge spread rate segments span over bin 0.0"
    (let [num-cells-to-suppress 6
          avg-dsr-data          {0.0 [2.0 2]
                                 1.0 [1.0 2]
                                 2.0 [6.0 2]
                                 3.0 [5.0 2]
                                 4.0 [4.0 2]
                                 5.0 [3.0 2]}]
      (is (= {['(1.0 0.0 5.0) 2.0] 6
              ['(0.0 5.0 4.0) 3.0] 6
              ['(5.0 4.0 3.0) 4.0] 6
              ['(4.0 3.0 2.0) 5.0] 6}
             (#'gridfire.suppression/compute-contiguous-slices num-cells-to-suppress avg-dsr-data))))))
```

9.2.8 gridfire.core-test

```
(ns gridfire.core-test
  (:require [clojure.string              :as str]
            [clojure.test                :refer [deftest is testing use-fixtures are compose-fixtures]]
            [gridfire.binary-output      :as binary]
            [gridfire.conversion         :refer [m->ft]]
            [gridfire.core               :as core]
            [gridfire.fetch              :as fetch]
            [gridfire.utils.test         :as utils]
            [tech.v3.datatype.functional :as dfn]))

;;-----------------------------------------------------------------------------
;; Config
;;-----------------------------------------------------------------------------

(def resources-path "test/gridfire/resources")

(def db-spec {:classname   "org.postgresql.Driver"
              :subprotocol "postgresql"
              :subname     "//localhost:5432/gridfire_test"
              :user        "gridfire_test"
              :password    "gridfire_test"})

(def test-config-base
  {:landfire-layers           {:aspect             {:type   :geotiff
                                                    :source "test/gridfire/resources/asp.tif"}
```

```
                              :canopy-base-height {:type   :geotiff
                                                   :source "test/gridfire/resources/cbh.tif"}
                              :canopy-cover       {:type   :geotiff
                                                   :source "test/gridfire/resources/cc.tif"}
                              :canopy-height      {:type   :geotiff
                                                   :source "test/gridfire/resources/ch.tif"}
                              :crown-bulk-density {:type   :geotiff
                                                   :source "test/gridfire/resources/cbd.tif"}
                              :elevation          {:type   :geotiff
                                                   :source "test/gridfire/resources/dem.tif"}
                              :fuel-model         {:type   :geotiff
                                                   :source "test/gridfire/resources/fbfm40.tif"}
                              :slope              {:type   :geotiff
                                                   :source "test/gridfire/resources/slp.tif"}}
   :srid                      "CUSTOM:900914"
   :cell-size                 98.425     ;; (feet)
   :ignition-row              [10 10]
   :ignition-col              [20 20]
   :max-runtime               60         ;; (minutes)
   :temperature               '(50)      ;; (degrees Fahrenheit)
   :relative-humidity         '(1)       ;; (%)
   :wind-speed-20ft           '(10)      ;; (miles/hour)
   :wind-from-direction       '(0)       ;; (degrees clockwise from north)
   :foliar-moisture           90         ;; (%)
   :crowning-disabled?        false
   :ellipse-adjustment-factor 1.0        ;; (< 1.0 = more circular, > 1.0 = more elliptical)
   :simulations               1
   :random-seed               1234567890 ;; long value (optional)
   :output-csvs?              true})

;;------------------------------------------------------------------------------
;; Utils
;;------------------------------------------------------------------------------

(defn in-file-path [filename]
  (str/join "/" [resources-path filename]))

(defn run-test-simulation! [config]
  (let [inputs (core/load-inputs! config)]
    (map #(dissoc % :rand-gen)
         (:summary-stats (core/run-simulations! inputs)))))

(defn valid-exits? [results]
  (when (seq results)
    (every? #(#{:max-runtime-reached :no-burnable-fuels} (:exit-condition %)) results)))

(defn results-signature
  "Computes a value representing the logical behavior of the simulation."
  [results]
  (mapv (fn [r]
          (select-keys r [:crown-fire-count
                          :crown-fire-size
                          :exit-condition
                          :fire-line-intensity-mean
                          :fire-line-intensity-stddev
                          :fire-size
                          :flame-length-mean
                          :flame-length-stddev
                          :global-clock
                          :spot-count
                          :surface-fire-count
                          :surface-fire-size]))
        results))

;;------------------------------------------------------------------------------
;; Fixtures
;;------------------------------------------------------------------------------
```

```
(use-fixtures :once (-> utils/with-temp-output-dir
                        (compose-fixtures utils/with-reset-db-pool)
                        (compose-fixtures utils/with-register-custom-projections)))

;;-----------------------------------------------------------------------------
;; Tests
;;-----------------------------------------------------------------------------

(deftest ^{:database true :simulation true} fetch-landfire-layers-test
  (testing "Fetching layers from postgis and geotiff files"
    (let [postgis-config {:db-spec         db-spec
                          :srid            "CUSTOM:900914"
                          :landfire-layers {:aspect             {:type   :postgis
                                                                 :source "landfire.asp WHERE rid=1"}
                                            :canopy-base-height {:type   :postgis
                                                                 :source "landfire.cbh WHERE rid=1"}
                                            :canopy-cover       {:type   :postgis
                                                                 :source "landfire.cc WHERE rid=1"}
                                            :canopy-height      {:type   :postgis
                                                                 :source "landfire.ch WHERE rid=1"}
                                            :crown-bulk-density {:type   :postgis
                                                                 :source "landfire.cbd WHERE rid=1"}
                                            :elevation          {:type   :postgis
                                                                 :source "landfire.dem WHERE rid=1"}
                                            :fuel-model         {:type   :postgis
                                                                 :source "landfire.fbfm40 WHERE rid=1"}
                                            :slope              {:type   :postgis
                                                                 :source "landfire.slp WHERE rid=1"}}}
          geotiff-config {:landfire-layers {:aspect             {:type   :geotiff
                                                                 :source (in-file-path "asp.tif")}
                                            :canopy-base-height {:type   :geotiff
                                                                 :source (in-file-path "cbh.tif")}
                                            :canopy-cover       {:type   :geotiff
                                                                 :source (in-file-path "cc.tif")}
                                            :canopy-height      {:type   :geotiff
                                                                 :source (in-file-path "ch.tif")}
                                            :crown-bulk-density {:type   :geotiff
                                                                 :source (in-file-path "cbd.tif")}
                                            :elevation          {:type   :geotiff
                                                                 :source (in-file-path "dem.tif")}
                                            :fuel-model         {:type   :geotiff
                                                                 :source (in-file-path "fbfm40.tif")}
                                            :slope              {:type   :geotiff
                                                                 :source (in-file-path "slp.tif")}}}]

      (is (dfn/equals (:matrix (fetch/landfire-layer postgis-config :aspect))
                      (:matrix (fetch/landfire-layer geotiff-config :aspect))))

      (is (dfn/equals (:matrix (fetch/landfire-layer postgis-config :canopy-cover))
                      (:matrix (fetch/landfire-layer geotiff-config :canopy-cover))))

      (is (dfn/equals (:matrix (fetch/landfire-layer postgis-config :canopy-height))
                      (:matrix (fetch/landfire-layer geotiff-config :canopy-height))))

      (is (dfn/equals (:matrix (fetch/landfire-layer postgis-config :crown-bulk-density))
                      (:matrix (fetch/landfire-layer geotiff-config :crown-bulk-density))))

      (is (dfn/equals (:matrix (fetch/landfire-layer postgis-config :elevation))
                      (:matrix (fetch/landfire-layer geotiff-config :elevation))))

      (is (dfn/equals (:matrix (fetch/landfire-layer postgis-config :fuel-model))
                      (:matrix (fetch/landfire-layer geotiff-config :fuel-model))))

      (is (dfn/equals (:matrix (fetch/landfire-layer postgis-config :slope))
                      (:matrix (fetch/landfire-layer geotiff-config :slope)))))))
;; TODO Add test for envelope

;;-----------------------------------------------------------------------------
```

```
;; Landfire Layer Tests
;;-------------------------------------------------------------------------

(deftest ^{:database true :simulation true} run-test-simulation!-test
  (testing "Running simulation with different ways to fetch LANDFIRE layers"
    (let [postgis-config  (merge test-config-base
                                 {:db-spec         db-spec
                                  :landfire-layers {:aspect             {:type   :postgis
                                                                         :source "landfire.asp WHERE rid=1"}
                                                    :canopy-base-height {:type   :postgis
                                                                         :source "landfire.cbh WHERE rid=1"}
                                                    :canopy-cover       {:type   :postgis
                                                                         :source "landfire.cc WHERE rid=1"}
                                                    :canopy-height      {:type   :postgis
                                                                         :source "landfire.ch WHERE rid=1"}
                                                    :crown-bulk-density {:type   :postgis
                                                                         :source "landfire.cbd WHERE rid=1"}
                                                    :elevation          {:type   :postgis
                                                                         :source "landfire.dem WHERE rid=1"}
                                                    :fuel-model         {:type   :postgis
                                                                         :source "landfire.fbfm40 WHERE rid=1"}
                                                    :slope              {:type   :postgis
                                                                         :source "landfire.slp WHERE rid=1"}}})
          geotiff-config  (merge test-config-base
                                 {:landfire-layers {:aspect             {:type   :geotiff
                                                                         :source (in-file-path "asp.tif")}
                                                    :canopy-base-height {:type   :geotiff
                                                                         :source (in-file-path "cbh.tif")}
                                                    :canopy-cover       {:type   :geotiff
                                                                         :source (in-file-path "cc.tif")}
                                                    :canopy-height      {:type   :geotiff
                                                                         :source (in-file-path "ch.tif")}
                                                    :crown-bulk-density {:type   :geotiff
                                                                         :source (in-file-path "cbd.tif")}
                                                    :elevation          {:type   :geotiff
                                                                         :source (in-file-path "dem.tif")}
                                                    :fuel-model         {:type   :geotiff
                                                                         :source (in-file-path "fbfm40.tif")}
                                                    :slope              {:type   :geotiff
                                                                         :source (in-file-path "slp.tif")}}})
          postgis-results (run-test-simulation! postgis-config)

          geotiff-results (run-test-simulation! geotiff-config)]

      (is (valid-exits? postgis-results))

      (is (valid-exits? geotiff-results))

      (is (= (mapv :fire-size postgis-results) (mapv :fire-size geotiff-results))))))
;;-------------------------------------------------------------------------
;; Ignition Layer Tests
;;-------------------------------------------------------------------------

(deftest ^:simulation geotiff-ignition-test
  (testing "Running simulation with ignition layers read from geotiff files"
    (let [config (merge test-config-base
                        {:ignition-layer           {:type   :geotiff
                                                    :source (in-file-path "ign.tif")}
                         ;; While we're here, using this test case for testing :max-parallel-simulations without wasting the time of
                         :max-parallel-simulations 2})]

      (is (valid-exits? (run-test-simulation! config))))))

(deftest ^{:database true :simulation true} postgis-ignition-test
  (testing "Running simulation with ignition layers read from Postgres database"
    (let [config (merge test-config-base
                        {:db-spec         db-spec
                         :ignition-layer {:type   :postgis
```

```
                                        :source "ignition.ign WHERE rid=1"}})]
      (is (valid-exits? (run-test-simulation! config))))))

(deftest ^:simulation burn-value-test
  (testing "Running simulation with burned and unburned values different from Gridfire's definition"
    (let [config (merge test-config-base
                        {:ignition-layer {:type        :geotiff
                                          :source      (in-file-path "ign-inverted.tif")
                                          :burn-values {:burned   -1.0
                                                        :unburned 1.0}}})]
      (is (valid-exits? (run-test-simulation! config))))))

;;------------------------------------------------------------------------------
;; Weather Layer Tests
;;------------------------------------------------------------------------------

(def landfire-layers-weather-test
  {:aspect             {:type   :geotiff
                        :source (in-file-path "weather-test/asp.tif")}
   :canopy-base-height {:type       :geotiff
                        :source     (in-file-path "weather-test/cbh.tif")
                        :unit       :metric
                        :multiplier 0.1}
   :canopy-cover       {:type   :geotiff
                        :source (in-file-path "weather-test/cc.tif")}
   :canopy-height      {:type       :geotiff
                        :source     (in-file-path "weather-test/ch.tif")
                        :unit       :metric
                        :multiplier 0.1}
   :crown-bulk-density {:type       :geotiff
                        :source     (in-file-path "weather-test/cbd.tif")
                        :unit       :metric
                        :multiplier 0.01}
   :elevation          {:type   :geotiff
                        :source (in-file-path "weather-test/dem.tif")}
   :fuel-model         {:type   :geotiff
                        :source (in-file-path "weather-test/fbfm40.tif")}
   :slope              {:type   :geotiff
                        :source (in-file-path "weather-test/slp.tif")}})

(def weather-layers
  {:temperature         {:type   :geotiff
                         :source (in-file-path "weather-test/tmpf_to_sample.tif")}
   :relative-humidity   {:type   :geotiff
                         :source (in-file-path "weather-test/rh_to_sample.tif")}
   :wind-speed-20ft     {:type   :geotiff
                         :source (in-file-path "weather-test/ws_to_sample.tif")}
   :wind-from-direction {:type   :geotiff
                         :source (in-file-path "weather-test/wd_to_sample.tif")}})

(deftest ^:simulation run-test-simulation!-weather-test
  (doseq [weather weather-layers]
    (let [config (merge test-config-base
                        weather
                        {:landfire-layers landfire-layers-weather-test
                         :max-runtime     120})]

      (is (valid-exits? (run-test-simulation! config))))))


(deftest ^:simulation geotiff-landfire-weather-ignition
  (testing "Running simulation using landfire, weather, and ignition data from geotiff files"
    (let [config (merge test-config-base
                        weather-layers
                        {:landfire-layers landfire-layers-weather-test
                         :ignition-layer  {:type   :geotiff
                                           :source (in-file-path "weather-test/phi.tif")}
                         :max-runtime     120})]
```

```
      (is (valid-exits? (run-test-simulation! config))))))

(deftest ^:simulation run-test-simulation!-using-lower-resolution-weather-test
  (testing "Running simulation using temperature data from geotiff file"
    (let [config (merge test-config-base
                        {:cell-size       (m->ft 30)
                         :landfire-layers landfire-layers-weather-test
                         :temperature     {:type   :geotiff
                                           :source (in-file-path "weather-test/tmpf_to_sample_lower_res.tif")}})]

      (is (valid-exits? (run-test-simulation! config))))))

;;-----------------------------------------------------------------------------
;; Perturbation Tests
;;-----------------------------------------------------------------------------

(deftest ^:simulation run-test-simulation!-with-landfire-perturbations
  (testing "with global perturbation value"
    (let [config (merge test-config-base
                        {:perturbations {:canopy-height {:spatial-type :global
                                                         :range        [-1.0 1.0]}}})]
      (is (valid-exits? (run-test-simulation! config)))))

  (testing "with pixel by pixel perturbation"
    (let [config (merge test-config-base
                        {:perturbations {:canopy-height {:spatial-type :pixel
                                                         :range        [-1.0 1.0]}}})]
      (is (valid-exits? (run-test-simulation! config))))))

(deftest ^:simulation run-test-simulation!-with-weather-perturbations
  (testing "temperature"
    (are [config] (valid-exits? (run-test-simulation! config))
      (merge test-config-base
             {:perturbations {:temperature {:spatial-type :global
                                            :range        [-1.0 1.0]}}})
      (merge test-config-base
             {:perturbations {:temperature {:spatial-type :smoothed-supergrid
                                            :gridfire.perturbation.smoothed-supergrid/supergrid-size [3 2 2]
                                            :range        [-1.0 1.0]}}})
      (merge test-config-base
             {:perturbations {:temperature {:spatial-type :pixel
                                            :range        [-1.0 1.0]}}})
      (merge test-config-base
             {:perturbations   {:temperature {:spatial-type :global
                                              :range        [-1.0 1.0]}}
              :landfire-layers landfire-layers-weather-test
              :temperature     (:temperature weather-layers)})
      (merge test-config-base
             {:perturbations {:temperature {:spatial-type :smoothed-supergrid
                                            :gridfire.perturbation.smoothed-supergrid/supergrid-size [1 1 1]
                                            :range        [-1.0 1.0]}}})
      (merge test-config-base
             {:perturbations   {:temperature {:spatial-type :pixel
                                              :range        [-1.0 1.0]}}
              :landfire-layers landfire-layers-weather-test
              :temperature     (:temperature weather-layers)})))

  (testing "wind-speed-20ft"
    (are [config] (valid-exits? (run-test-simulation! config))
      (merge test-config-base
             {:perturbations {:wind-speed-20ft {:spatial-type :global
                                                :range        [-1.0 1.0]}}})
      (merge test-config-base
             {:perturbations {:wind-speed-20ft {:spatial-type :pixel
                                                :range        [-1.0 1.0]}}})
      (merge test-config-base
             {:perturbations   {:wind-speed-20ft {:spatial-type :global
```

```
                                          :range        [-1.0 1.0]}}
             :landfire-layers landfire-layers-weather-test
             :wind-speed-20ft (:wind-speed-20ft weather-layers)})

      (merge test-config-base
             {:perturbations   {:wind-speed-20ft {:spatial-type :pixel
                                                  :range        [-1.0 1.0]}}
             :landfire-layers landfire-layers-weather-test
             :wind-speed-20ft (:wind-speed-20ft weather-layers)})))

  (testing "fuel-moisture"
    (are [config] (valid-exits? (run-test-simulation! config))
      (merge test-config-base
             {:perturbations {:fuel-moisture-dead-1hr        {:spatial-type :global
                                                              :range        [-1.0 1.0]}
                              :fuel-moisture-dead-10hr       {:spatial-type :global
                                                              :range        [-1.0 1.0]}
                              :fuel-moisture-dead-100hr      {:spatial-type :global
                                                              :range        [-1.0 1.0]}
                              :fuel-moisture-live-herbaceous {:spatial-type :global
                                                              :range        [-1.0 1.0]}
                              :fuel-moisture-live-woody      {:spatial-type :global
                                                              :range        [-1.0 1.0]}}
              :fuel-moisture {:dead {:1hr   0.2
                                     :10hr  0.2
                                     :100hr 0.2}
                              :live {:herbaceous 0.3
                                     :woody      0.6}}})
      (merge test-config-base
             {:perturbations   {:fuel-moisture-dead-1hr        {:spatial-type :global
                                                                :range        [-1.0 1.0]}
                                :fuel-moisture-dead-10hr       {:spatial-type :global
                                                                :range        [-1.0 1.0]}
                                :fuel-moisture-dead-100hr      {:spatial-type :global
                                                                :range        [-1.0 1.0]}
                                :fuel-moisture-live-herbaceous {:spatial-type :global
                                                                :range        [-1.0 1.0]}
                                :fuel-moisture-live-woody      {:spatial-type :global
                                                                :range        [-1.0 1.0]}}
              :landfire-layers landfire-layers-weather-test
              :fuel-moisture   {:dead {:1hr   {:type   :geotiff
                                               :source (in-file-path "weather-test/m1_to_sample.tif")}
                                       :10hr  {:type   :geotiff
                                               :source (in-file-path "weather-test/m10_to_sample.tif")}
                                       :100hr {:type   :geotiff
                                               :source (in-file-path "weather-test/m100_to_sample.tif")}}
                                :live {:herbaceous 0.3
                                       :woody      0.6}}})))))

;;-----------------------------------------------------------------------------
;; Outputs
;;-----------------------------------------------------------------------------

(deftest ^:simulation binary-output-files-test
  (let [config         (merge test-config-base
                              {:output-binary?   true
                               :output-directory "test/output"})
        _              (run-test-simulation! config)
        binary-results (binary/read-matrices-as-binary (utils/out-file-path "toa_0001_00001.bin")
                                                       [:float :float :float :int])]
    (is (some? binary-results))))

;;-----------------------------------------------------------------------------
;; Ignition Mask
;;-----------------------------------------------------------------------------

(deftest ^:simulation igniton-mask-test
  (let [config (merge test-config-base
```

```
                    {:landfire-layers landfire-layers-weather-test
                     :ignition-row    nil
                     :ignition-col    nil
                     :random-ignition {:ignition-mask {:type   :geotiff
                                                       :source (in-file-path "weather-test/ignition_mask.tif")}
                                       :edge-buffer   9843.0}})]
    (is (valid-exits? (run-test-simulation! config)))))

;;------------------------------------------------------------------------------
;; Crowning disabled
;;------------------------------------------------------------------------------

(deftest ^:simulation crowning-disabled-test
  (let [config (merge test-config-base
                      {:crowning-disabled? true})]
    (is (valid-exits? (run-test-simulation! config)))))

;;------------------------------------------------------------------------------
;; Moisture Rasters
;;------------------------------------------------------------------------------

(deftest ^:simulation moisture-rasters-test
  (let [config (merge test-config-base
                      {:landfire-layers landfire-layers-weather-test
                       :ignition-row    nil
                       :ignition-col    nil
                       :random-ignition {:ignition-mask {:type   :geotiff
                                                         :source (in-file-path "weather-test/ignition_mask.tif")}
                                         :edge-buffer   9843.0}
                       :fuel-moisture   {:dead {:1hr   {:type   :geotiff
                                                        :source (in-file-path "weather-test/m1_to_sample.tif")}
                                                :10hr  {:type   :geotiff
                                                        :source (in-file-path "weather-test/m10_to_sample.tif")}
                                                :100hr {:type   :geotiff
                                                        :source (in-file-path "weather-test/m100_to_sample.tif")}}
                                         :live {:woody      {:type   :geotiff
                                                             :source (in-file-path "weather-test/mlw_to_sample.tif")}
                                                :herbaceous {:type   :geotiff
                                                             :source (in-file-path "weather-test/mlh_to_sample.tif")}}}}})]
    (is (valid-exits? (run-test-simulation! config)))))

(deftest ^:simulation moisture-scalars-only-test
  (let [config (merge test-config-base
                      {:landfire-layers landfire-layers-weather-test
                       :ignition-row    nil
                       :ignition-col    nil
                       :random-ignition {:ignition-mask {:type   :geotiff
                                                         :source (in-file-path "weather-test/ignition_mask.tif")}
                                         :edge-buffer   9843.0}
                       :fuel-moisture   {:dead {:1hr   0.10
                                                :10hr  0.10
                                                :100hr 0.10}
                                         :live {:woody      0.80
                                                :herbaceous 0.80}}})]
    (is (valid-exits? (run-test-simulation! config)))))

(deftest ^:simulation moisture-mix-raster-scalars-test
  (let [config (merge test-config-base
                      {:landfire-layers landfire-layers-weather-test
                       :ignition-row    nil
                       :ignition-col    nil
                       :random-ignition {:ignition-mask {:type   :geotiff
                                                         :source (in-file-path "weather-test/ignition_mask.tif")}
                                         :edge-buffer   9843.0}
                       :fuel-moisture   {:dead {:1hr   {:type   :geotiff
                                                        :source (in-file-path "weather-test/m1_to_sample.tif")}
                                                :10hr  {:type   :geotiff
                                                        :source (in-file-path "weather-test/m10_to_sample.tif")}
```

```
                                          :100hr {:type   :geotiff
                                                  :source (in-file-path "weather-test/m100_to_sample.tif")}}
                                   :live {:woody      80.0
                                          :herbaceous 30.0}}})]
    (is (valid-exits? (run-test-simulation! config)))))

;;------------------------------------------------------------------------------
;; Ignition CSV
;;------------------------------------------------------------------------------

(deftest ^:simulation ignition-csv-test
  (let [results (run-test-simulation! (assoc test-config-base :ignition-csv (in-file-path "sample_ignitions.csv")))]

    (is (valid-exits? results))

    (is (= 3 (count results))
        "Should have the same number of simulations as ignition rows in sample_ignitions.csv")

    (is (= 10.0 (:global-clock (first results)))
        "Global clock should end at start_time + max_runtime in sample_ignitions.csv")

    (is (= 20.0 (:global-clock (second results)))
        "Global clock should end at start_time + max_runtime in sample_ignitions.csv")))


;;------------------------------------------------------------------------------
;; Pyrome spread rate adjustment
;;------------------------------------------------------------------------------

(deftest ^:simulation spread-rate-adjustment-test
  (testing "successful run using explicitly defined spread-rate-adjustments for each simulation in the ensemble run "
    (let [config (merge test-config-base
                        {:fuel-number->spread-rate-adjustment-samples [{144 0.5
                                                                        148 0.5
                                                                        164 0.5
                                                                        184 0.5
                                                                        188 0.5
                                                                        102 0.5
                                                                        204 0.5
                                                                        104 0.5
                                                                        106 0.5
                                                                        108 0.5
                                                                        122 0.5
                                                                        124 0.5
                                                                        141 0.5
                                                                        145 0.5
                                                                        149 0.5
                                                                        161 0.5
                                                                        165 0.5
                                                                        181 0.5
                                                                        185 0.5
                                                                        189 0.5
                                                                        201 0.5
                                                                        142 0.5
                                                                        146 0.5
                                                                        162 0.5
                                                                        182 0.5
                                                                        186 0.5
                                                                        101 0.5
                                                                        202 0.5
                                                                        103 0.5
                                                                        105 0.5
                                                                        107 0.5
                                                                        109 0.5
                                                                        121 0.5
                                                                        123 0.5
                                                                        143 0.5
                                                                        147 0.5
```

```
                                                                  163 0.5
                                                                  183 0.5
                                                                  187 0.5
                                                                  203 0.5}]})]

      (is (valid-exits? (run-test-simulation! config))))))

(deftest ^:simulation spread-rate-adjustment-for-fuel-model-test
  (testing "successful run using explicitly defined sdi suppresion constants for each simulation in the ensemble run"
    (let [config (merge test-config-base
                        {:suppression                                          {:sdi-layer      {:type   :geotiff
                                                                                                 :source "test/gridfire/resources/
                                                                                 :suppression-dt 10}
                         :sdi-sensitivity-to-difficulty-samples                [1.0]
                         :sdi-containment-overwhelming-area-growth-rate-samples [50000.0]
                         :sdi-reference-suppression-speed-samples              [600.0]})]

      (is (valid-exits? (run-test-simulation! config))))))

(deftest ^:simulation surface-fire-min-memoization-test
  (let [default-res (run-test-simulation! test-config-base)]
    (testing (str (pr-str '{:memoization {:surface-fire-min XXX}}) " changes how surface-fire-min cacheing happens, with possible v
      (testing (str (pr-str :across-sims) "(default): one cache per simulation.")
        (let [config (assoc test-config-base :memoization {:surface-fire-min :across-sims})
              res    (run-test-simulation! config)]
          (is (valid-exits? res))
          (is (= (results-signature default-res)
                 (results-signature res))
              "the logical behavior of the simulation is unchanged.")))
      (testing (str (pr-str :within-sims) ": one cache per simulation.")
        (let [config (assoc test-config-base :memoization {:surface-fire-min :within-sims})
              res    (run-test-simulation! config)]
          (is (valid-exits? res))
          (is (= (results-signature default-res)
                 (results-signature res))
              "the logical behavior of the simulation is unchanged.")))
      (testing (str (pr-str nil) ": no memoization.")
        (let [config (assoc test-config-base :memoization {:surface-fire-min nil})
              res    (run-test-simulation! config)]
          (is (valid-exits? res))
          (is (= (results-signature default-res)
                 (results-signature res))
              "the logical behavior of the simulation is unchanged."))))))
```

9.2.9 gridfire.fire-spread-test

```
(ns gridfire.fire-spread-test
  (:require [clojure.test                :refer [deftest are is testing]]
            [gridfire.fire-spread        :refer [create-new-burn-vectors!-pfn
                                                 create-new-burn-vectors!-invoke
                                                 diagonal?
                                                 direction-angle->bit
                                                 direction-angle->i-incr
                                                 direction-angle->j-incr]]
            [gridfire.grid-lookup        :as grid-lookup]
            [tech.v3.datatype            :as d]
            [tech.v3.tensor              :as t]))

(defn- is-this-fn-equivalent-to-that-double-map?
  [f d->v]
  (->> d->v
       (every? (fn [[d v]]
                 (is (= v (f d)))))))

(deftest ^:unit case-double-examples-test
  (testing (pr-str `diagonal?)
```

```
    (is-this-fn-equivalent-to-that-double-map? diagonal?
                                               {0.0   false
                                                90.0  false
                                                180.0 false
                                                270.0 false
                                                45.0  true
                                                135.0 true
                                                225.0 true
                                                315.0 true}))
  (testing (pr-str `direction-angle->bit)
    (is-this-fn-equivalent-to-that-double-map? direction-angle->bit
                                               {0.0   0
                                                45.0  1
                                                90.0  2
                                                135.0 3
                                                180.0 4
                                                225.0 5
                                                270.0 6
                                                315.0 7}))
  (testing (pr-str `direction-angle->i-incr)
    (is-this-fn-equivalent-to-that-double-map? direction-angle->i-incr
                                               {0.0   -1
                                                45.0  -1
                                                315.0 -1
                                                135.0 1
                                                180.0 1
                                                225.0 1
                                                90.0  0
                                                270.0 0}))
  (testing (pr-str `direction-angle->j-incr)
    (is-this-fn-equivalent-to-that-double-map? direction-angle->j-incr
                                               {45.0  1
                                                90.0  1
                                                135.0 1
                                                0.0   0
                                                180.0 0
                                                225.0 -1
                                                270.0 -1
                                                315.0 -1})))

(defn- new-travel-lines-matrix
  [shape]
  (-> (t/new-tensor shape :datatype :int16)
      (grid-lookup/add-double-getter)))

(deftest ^:unit create-new-burn-vectors_test
  (let [num-rows                    10
        num-cols                    10
        shape                       [num-rows num-cols]
        cell-size                   98.425
        get-elevation               (grid-lookup/tensor-cell-getter 1.0 nil)
        burn-probability            1.0
        zero-tensor                 (grid-lookup/add-double-getter (t/new-tensor shape))
        fire-spread-matrix          (d/clone zero-tensor)
        max-spread-rate-matrix      (d/clone zero-tensor)
        max-spread-direction-matrix (d/clone zero-tensor)
        eccentricity-matrix         (d/clone zero-tensor)
        i                           5
        j                           5]
    (are [result travel-lines-matrix] (let [create-new-burn-vectors! (create-new-burn-vectors!-pfn num-rows num-cols cell-size get-
                                                                                                   travel-lines-matrix fire-spread-
                                                                                                   max-spread-rate-matrix max-sprea
                                            burn-vectors             (persistent! (create-new-burn-vectors!-invoke create-new-burn-
                                       (= result (count burn-vectors)))
      8 (new-travel-lines-matrix shape)                          ; No burn vectors in cell
      7 (t/mset! (new-travel-lines-matrix shape) i j 2r00000001) ; N burn vector exists
      7 (t/mset! (new-travel-lines-matrix shape) i j 2r00010000) ; S burn vector exists
      6 (t/mset! (new-travel-lines-matrix shape) i j 2r00010001) ; N & S burn vector exists
```

```
      7 (t/mset! (new-travel-lines-matrix shape) i j 2r00000010) ; NE burn vector exists
      7 (t/mset! (new-travel-lines-matrix shape) i j 2r00100000) ; SW burn vector exists
      6 (t/mset! (new-travel-lines-matrix shape) i j 2r00100010) ; NE & SW burn vector exists
      7 (t/mset! (new-travel-lines-matrix shape) i j 2r00000100) ; E burn vector exists
      7 (t/mset! (new-travel-lines-matrix shape) i j 2r01000000) ; W burn vector exists
      6 (t/mset! (new-travel-lines-matrix shape) i j 2r01000100) ; E & W burn vector exists
      7 (t/mset! (new-travel-lines-matrix shape) i j 2r00001000) ; SE burn vector exists
      7 (t/mset! (new-travel-lines-matrix shape) i j 2r10000000) ; NW burn vector exists
      6 (t/mset! (new-travel-lines-matrix shape) i j 2r10001000) ; SE & NW burn vector exists
      6 (t/mset! (new-travel-lines-matrix shape) i j 2r01000001) ; N & W burn vectors exists
      5 (t/mset! (new-travel-lines-matrix shape) i j 2r11000001) ; N & W & NW burn vectors exists
      4 (t/mset! (new-travel-lines-matrix shape) i j 2r11000011)))) ; N & W & NW & NE burn vectors exists
```

9.2.10 gridfire.crown-fire-test

FIXME how about we lift this up to the Crowning section?

```
(ns gridfire.crown-fire-test
  (:require [clojure.test        :refer [deftest testing are run-tests]]
            [gridfire.conversion :as c]
            [gridfire.crown-fire :refer [crown-fire-line-intensity
                                         crown-fire-eccentricity
                                         crown-length-to-width-ratio
                                         cruz-crown-fire-spread
                                         cruz-crown-fire-spread-metric
                                         cruz-active-crown-fire-spread
                                         cruz-passive-crown-fire-spread
                                         van-wagner-critical-fire-line-intensity
                                         van-wagner-crown-fire-initiation-metric?
                                         van-wagner-crown-fire-initiation?]]))

(defn- within-%? [^double a ^double b ^double percent]
  (<= (Math/abs ^double (- a b)) (Math/abs (* percent a))))

(defn- within-5%? [^double a ^double b]
  (within-%? a b 0.05))

(defn- crown-fire-within-5%? [fire-type spread-rate-a spread-rate-b]
  (if (= fire-type :passive)
    (and
     (neg? spread-rate-b)
     (within-5%? spread-rate-a (* -1 spread-rate-b)))

    (and
     (pos? spread-rate-b)
     (within-5%? spread-rate-a spread-rate-b))))

(deftest ^:unit test-van-wagner-crown-fire-line-intensity
  (testing "Fire intensity using Van Wagner (1976), equation 4."
    (are [expected args] (within-5%? expected (apply van-wagner-critical-fire-line-intensity args))
         ; Crit. surf. intensity (kW/m) [canopy-base-height (m) foliar-moisture epsilon (%)]
         0.0                          [0 0]      ; No Canopy Base Height/Intensity
         2890.0                       [7 95]     ; Red pine (C6)
         4560.0                       [7 135]    ; Red pine (C4)
         210.0                        [1 120])))  ; Balsam fir under pine (F3)

(deftest ^:unit test-van-wagner-crown-fire-initation-metric?
  (testing "Fire intensity reaches Van Wagner crown fire threshold using SI units."
    (are [expected args] (= expected (apply van-wagner-crown-fire-initiation-metric? args))
         ; true/false [canopy-cover (%) canopy-base-height (m) foliar-moisture (%) final-intensity (kW/m)]
         false        [0   0.0  0.0  0]      ; No canopy cover
         false        [50  0.0  0.0  0]      ; No Canopy Base Height/No final-intensity
         false        [50  0.0  0.0  10]     ; No Canopy Base Height
         true         [50  1.0  0.0  10]     ; CC, CBH, No Moisture
         true         [50  7.0  95   10500]  ; Red pine (C6)
         true         [50  7.0  135  9500]   ; Red pine (C4)
```

```
        false       [50  1.0  120  85])))  ; Balsam fir under pine (F3)

(deftest ^:unit test-van-wagner-crown-fire-initation?
  (testing "Fire intensity reaches Van Wagner crown fire threshold using imperial units."
    (are [expected args] (= expected (apply van-wagner-crown-fire-initiation? args))
         ; true/false [canopy-cover (%) canopy-base-height (m) foliar-moisture (ratio) final-intensity (kW/m)]
         false       [0   0.0          0.0  0]      ; No canopy cover
         false       [50  0.0          0.0  0]      ; No Canopy Base Height/No final-intensity
         false       [50  0.0          0.0  (c/kW-m->Btu-ft-s 10.0)]    ; No Canopy Base Height
         true        [50  (c/m->ft 1.0) 0.0  (c/kW-m->Btu-ft-s 10.0)]    ; CC, CBH, No Moisture
         true        [50  (c/m->ft 7.0) 0.95 (c/kW-m->Btu-ft-s 10500.0)] ; Red pine (C6)
         true        [50  (c/m->ft 7.0) 1.35 (c/kW-m->Btu-ft-s 9500.0)]  ; Red pine (C4)
         false       [50  (c/m->ft 1.0) 1.20 (c/kW-m->Btu-ft-s 85.0)])))  ; Balsam fir under pine (F3)

(deftest ^:unit test-cruz-active-fire-spread
  (testing "Crown Fire spread rate (Cruz 2005) using SI units."
    (are [expected args] (within-5%? expected (apply cruz-active-crown-fire-spread args))
         ; m/min [wind-speed-10m (km/hr) canopy-bulk-density (kg/m^3) est. fine fuel moisture (%)]
         22.6    [15.8 0.27 8.8]    ; Mean values
         137     [50   0.2  4.0]    ; 1983 Mount Muirhead Fire
         72.2    [74   0.1  9.0])))  ; 1973 Burnt Fire

(deftest ^:unit test-cruz-passive-fire-spread
  (testing "Passive fire spread rate (Cruz 2005) using SI units."
    (are [expected args] (let [active-spread    (apply cruz-active-crown-fire-spread args)
                               crit-spread-rate (/ 3.0 (second args))]
                           (within-5%? expected (cruz-passive-crown-fire-spread active-spread crit-spread-rate)))
         ; m/min [wind-speed-10m (km/hr) canopy-bulk-density (kg/m^3) est. fine fuel moisture (%)]
         7.1     [16.3 0.16 8.6]    ; Mean values
         7.6     [7    0.08 8.0])))  ; Bor Island Fire Experiment

(deftest ^:unit test-cruz-fire-spread
  (testing "Active crown fires using SI units."
    (are [expected args] (crown-fire-within-5%? :active expected (apply cruz-crown-fire-spread-metric args))
         ; m/min [wind-speed-10m (km/hr) canopy-bulk-density (kg/m^3) est. fine fuel moisture (%)]
         22.6    [15.8 0.27 8.8] ; Mean values
         32.0    [35 0.1 10.0]))

  (testing "Passive crown fires using SI units."
    (are [expected args] (crown-fire-within-5%? :passive expected (apply cruz-crown-fire-spread-metric args))
         ; m/min [wind-speed-10m (km/hr) canopy-bulk-density (kg/m^3) est. fine fuel moisture (%)]
         7.6     [7  0.08 8.0] ; Bor Island Fire Experiment
         11.0    [30 0.1 10.0])))

(deftest ^:unit test-cruz-fire-spread-imperial
  (testing "Testing using imperial units fires."
    (are [expected args] (crown-fire-within-5%? :active expected (apply cruz-crown-fire-spread args))
         ; ft/min       [wind-speed-20ft (mph) canopy-bulk-density (lb/ft^3) est. fine fuel moisture (0-1)]
         (c/m->ft 22.6) [(-> 15.8 (c/km-hr->mph) (c/wind-speed-10m->wind-speed-20ft)) (c/kg-m3->lb-ft3 0.27) 0.088]
         (c/m->ft 32.0) [(-> 35.0 (c/km-hr->mph) (c/wind-speed-10m->wind-speed-20ft)) (c/kg-m3->lb-ft3 0.1)  0.1]))

  (testing "Passive crown fires"
    (are [expected args] (crown-fire-within-5%? :passive expected (apply cruz-crown-fire-spread args))
         ; ft/min       [wind-speed-20ft (mph) canopy-bulk-density (lb/ft^3) est. fine fuel moisture (0-1)]
         (c/m->ft 7.6)  [(-> 7.0  (c/km-hr->mph) (c/wind-speed-10m->wind-speed-20ft)) (c/kg-m3->lb-ft3 0.08) 0.08]
         (c/m->ft 11.0) [(-> 30.0 (c/km-hr->mph) (c/wind-speed-10m->wind-speed-20ft)) (c/kg-m3->lb-ft3 0.1)  0.1])))

(deftest ^:unit test-crown-fire-line-intensity
  (testing "Crown Fire line intensity using SI units."
    (are [expected args] (within-5%? expected (apply crown-fire-line-intensity args))
         ; kW/m  [crown-spread-rate (m/min) crown-bulk-density (kg/m^3) canopy-height-difference (m) heat-of-combustion (kJ/kg)]
         78 [1.0 0.25 1.0 (c/Btu-lb->kJ-kg 8000.0)])) ; Metric

  (testing "Crown Fire line intensity using imperial units."
    (are [expected args] (within-5%? expected (apply crown-fire-line-intensity args))
         ; Btu/ft*s  [crown-spread-rate (f/min) crown-bulk-density (lb/ft^3) canopy-height-difference (ft) heat-of-combustion (Btu/l
         22 [(c/m->ft 1.0) (c/kg-m3->lb-ft3 0.25) (c/m->ft 1.0) 8000.0])))
```

```
(deftest ^:unit test-crown-length-to-width-ratio
  (testing "Crown fire length/width ratio"
    (are [expected args] (within-5%? expected (apply crown-length-to-width-ratio args))
         ; L/W [wind-speed-20ft ellipse-adjustment-factor]
         1.0   [0 0]
         1.0   [1 0]
         1.125 [1 1]
         1.25  [1 2])))

(deftest ^:unit test-crown-fire-eccentricity
  (testing "Crown fire eccentricity"
    (are [expected args] (within-5%? expected (apply crown-fire-eccentricity args))
         ; E  [wind-speed-20ft ellipse-adjustment-factor]
         0.0  [1 0]
         0.45 [1 1]
         0.75 [1 4.0]
         0.9  [1 10.0])))

(comment
  (run-tests 'gridfire.crown-fire-test)
  )
```

9.2.11 gridfire.spotting-old

```
(ns gridfire.spotting-old
  (:require [gridfire.common       :refer [distance-3d
                                           calc-fuel-moisture
                                           in-bounds?
                                           burnable?]]
            [gridfire.utils.random :refer [my-rand-range]]
            [gridfire.conversion   :as convert]
            [gridfire.grid-lookup  :as grid-lookup]
            [tech.v3.tensor        :as t])
  (:import java.util.Random))

;;-------------------------------------------------------------------------------
;; Formulas
;;-------------------------------------------------------------------------------

(defn- sample-spotting-params
  ^double
  [param rand-gen]
  (if (map? param)
    (let [{:keys [lo hi]} param
          l               (if (vector? lo) (my-rand-range rand-gen (lo 0) (lo 1)) lo)
          h               (if (vector? hi) (my-rand-range rand-gen (hi 0) (hi 1)) hi)]
      (my-rand-range rand-gen l h))
    param))

(defn- mean-variance
  "Returns mean spotting distance and it's variance given:
  fire-line-intensity: (kWm^-1)
  wind-speed-20ft: (ms^-1)"
  [{:keys [^double mean-distance ^double flin-exp ^double ws-exp ^double normalized-distance-variance]}
   rand-gen ^double fire-line-intensity ^double wind-speed-20ft]
  (let [a (sample-spotting-params mean-distance rand-gen)
        b (sample-spotting-params flin-exp rand-gen)
        c (sample-spotting-params ws-exp rand-gen)
        m (* a (Math/pow fire-line-intensity b) (Math/pow wind-speed-20ft c))]
    {:mean m :variance (* m (sample-spotting-params normalized-distance-variance rand-gen))}))

(defn- standard-deviation
  "Returns standard deviation for the lognormal distribution given:
  mean spotting distance and it's variance"
  ^double
  [^double m ^double v]
```

```
  (Math/sqrt (Math/log (+ 1 (/ v (Math/pow m 2))))))

(defn- normalized-mean
  "Returns normalized mean for the lognormal distribution given:
  mean spotting distance and it's variance"
  ^double
  [^double m ^double v]
  (Math/log (/ (Math/pow m 2)
               (Math/sqrt (+ v (Math/pow m 2))))))

(defn- sample-normal
  "Returns sample from normal/gaussian distribution given mu and sd."
  ^double
  [^Random rand-gen ^double mu ^double sd]
  (+ mu (* sd (.nextGaussian rand-gen))))

(defn- sample-lognormal
  "Returns sample from log-normal distribution given mu and sd."
  ^double
  [^Random rand-gen ^double mu ^double sd]
  (Math/exp (sample-normal rand-gen mu sd)))

(defn- sample-wind-dir-deltas
  "Returns a sequence of [x y] distances (meters) that firebrands land away
  from a torched cell at i j where:
  x: parallel to the wind
  y: perpendicular to the wind (positive values are to the right of wind direction)"
  [{:keys [spotting rand-gen]}
   fire-line-intensity-matrix
   wind-speed-20ft [i j]]
  (let [num-firebrands          (long (sample-spotting-params (:num-firebrands spotting) rand-gen))
        intensity               (convert/Btu-ft-s->kW-m (t/mget fire-line-intensity-matrix i j))
        {:keys [mean variance]} (mean-variance spotting rand-gen intensity wind-speed-20ft)
        mu                      (normalized-mean mean variance)
        sd                      (standard-deviation mean variance)
        parallel-values         (repeatedly num-firebrands #(sample-lognormal rand-gen mu sd))
        perpendicular-values    (repeatedly num-firebrands #(sample-normal rand-gen 0.0 0.92))]
    (mapv (fn [x y] [(convert/m->ft x) (convert/m->ft y)])
          parallel-values
          perpendicular-values)))

(defn hypotenuse ^double
  [x y]
  (Math/sqrt (+ (Math/pow x 2) (Math/pow y 2))))

(defn deltas-wind->coord
  "Converts deltas from the torched tree in the wind direction to deltas
  in the coordinate plane"
  [deltas ^double wind-direction]
  (mapv (fn [[d-paral d-perp]]
          (let [d-paral (double d-paral)
                d-perp  (double d-perp)
                H  (hypotenuse d-paral d-perp)
                t1 wind-direction
                t2 (convert/rad->deg (Math/atan (/ d-perp d-paral)))
                t3 (+ t1 t2)]
            [(* H (Math/sin (convert/deg->rad t3)))
             (* -1 H (Math/cos (convert/deg->rad t3)))]))
        deltas))

(defn firebrands
  "Returns a sequence of cells [i,j] that firebrands land in.
   Note: matrix index [i,j] refers to [row, column]. Therefore, we need to flip
   [row,column] to get to [x,y] coordinates."
  [deltas wind-towards-direction cell ^double cell-size]
  (let [step         (/ cell-size 2)
        [y x]        (mapv #(+ step (* ^double % cell-size)) cell)
        x            (double x)
```

```
        y            (double y)
        coord-deltas (deltas-wind->coord deltas wind-towards-direction)]
    (mapv (fn [[dx dy]]
            (let [dx (double dx)
                  dy (double dy)]
              [(long (Math/floor (/ (+ dy y) cell-size)))
               (long (Math/floor (/ (+ dx x) cell-size)))]))
          coord-deltas)))

(defn heat-of-preignition
  "Returns heat of preignition given:
   - Temperature: (Celsius)
   - Fine fuel moisture (0-1 ratio)

   Q_ig = 144.512 - 0.266*T_o - 0.00058 * (T_o)^2 - T_o * M + 18.54 * (1 - exp ( -15.1 * M ) ) + 640 * M  (eq. 10)"
  ^double
  [^double temperature ^double fine-fuel-moisture]
  (let [T_o temperature
        M   fine-fuel-moisture

        ;; heat required to reach ignition temperature
        Q_a (+ 144.512 (* -0.266 T_o) (* -0.00058 (Math/pow T_o 2.0)))

        ;; heat required to raise moisture to reach boiling point
        Q_b (* -1.0 T_o M)

        ;; Heat of desorption
        Q_c (* 18.54 (- 1.0 (Math/exp (* -15.1 M))))

        ;; Heat required to vaporize moisture
        Q_d (* 640.0 M)]
    (+ Q_a Q_b Q_c Q_d)))

(defn schroeder-ign-prob
  "Returns the probability of ignition as described in Shroeder (1969) given:
   - Temperature: (Celsius)
   - Fine fuel moisture (0-1 ratio)

   X = (400 - Q_ig) / 10
   P(I) = (0.000048 * X^4.3) / 50    (pg. 15)"
  ^double
  [^double temperature ^double fine-fuel-moisture]
  (let [Q_ig (heat-of-preignition temperature fine-fuel-moisture)
        X    (/ (- 400.0 Q_ig) 10.0)]
    (-> X
        (Math/pow 4.3)
        (* 0.000048)
        (/ 50.0)
        (Math/min 1.0)
        (Math/max 0.0))))

(defn- one-minus ^double [^double x] (- 1.0 x))

(defn spot-ignition-probability
  "Returns the probability of spot fire ignition (Perryman 2012) given:
   - Schroeder's probability of ignition [P(I)] (0-1)
   - Decay constant [lambda] (0.005)
   - Distance from the torched cell [d] (meters)
   - Number of firebrands accumulated in the cell [b]

   P(Spot Ignition) = 1 - (1 - (P(I) * exp(-lambda * d)))^b"
  ^double
  [^double ignition-probability ^double decay-constant ^double spotting-distance ^double firebrand-count]
  (-> decay-constant
      (* -1.0)
      (* spotting-distance)
      (Math/exp)
      (* ignition-probability)
```

```
      (one-minus)
      (Math/pow firebrand-count)
      (one-minus)))

(defn spot-ignition?
  [rand-gen ^double spot-ignition-probability]
  (let [random-number (my-rand-range rand-gen 0 1)]
    (> spot-ignition-probability random-number)))

(defn albini-t-max
  "Returns the time of spot ignition using (Albini 1979) in minutes given:
   - Flame length: (m) [z_F]

   a = 5.963                                                    (D33)
   b = a - 1.4                                                  (D34)
   D = 0.003
   t_c = 1
   w_F = 2.3 * (z_F)^0.5                                        (A58)
   t_o = t_c / (2 * z_F / w_F)
   z =  0.39 * D * 10^5
   t_T = t_o + 1.2 + (a / 3) * ( ( (b + (z/z_F) )/a )^3/2 - 1 )  (D43)"
  ^double
  [^double flame-length]
  (let [a     5.963                          ; constant from (D33)
        b     4.563                          ; constant from (D34)
        z-max 117.0                          ; max height given particle diameter of 0.003m
        w_F   (* 2.3 (Math/sqrt flame-length)) ; upward axial velocity at flame tip
        t_0   (/ w_F (* 2.0 flame-length))]    ; period of steady burning of tree crowns (t_c, min) normalized by 2*z_F / w_F
    (-> z-max
        (/ flame-length)
        (+ b)
        (/ a)
        (Math/pow 1.5)
        (- 1.0)
        (* (/ a 3.0))
        (+ 1.2)
        (+ t_0))))

(defn spot-ignition-time
  "Returns the time of spot ignition using (Albini 1979) and (Perryman 2012) in minutes given:
   - Global clock: (min)
   - Flame length: (m)

   t_spot = clock + (2 * t_max) + t_ss"
  ^double
  [^double global-clock ^double flame-length]
  (let [t-steady-state 20.0] ; period of building up to steady state from ignition (min)
    (-> (albini-t-max flame-length)
        (* 2.0)
        (+ global-clock)
        (+ t-steady-state))))

(defn- update-firebrand-counts!
  [{:keys [num-rows num-cols fuel-model-matrix]}
   firebrand-count-matrix
   fire-spread-matrix
   source
   firebrands]
  (doseq [[x y :as here] firebrands
          :when          (and (in-bounds? num-rows num-cols [x y])
                              (burnable? fire-spread-matrix
                                         fuel-model-matrix
                                         source
                                         here))
          :let           [new-count (inc ^double (t/mget firebrand-count-matrix x y))]]
    (t/mset! firebrand-count-matrix x y new-count)))

(defn- in-range?
```

```
  [[min max] fuel-model-number]
  (<= min fuel-model-number max))

(defn surface-spot-percent
  "Returns the surface spotting probability, given:
   - A vector of vectors where the first entry is a vector range of fuel models,
     and the second entry is either a single probability or vector range of probabilities
     of those fuels spotting (e.g. `[[[10 20] 0.2]]` or `[[[10 20] [0.2 0.4]]]`)
   - The fuel model number for the particular cell
   - A random number generator, which is used to generate the probability when
     a range of probabilities is given"
  ^double
  [fuel-range-percents fuel-model-number rand-gen]
  (reduce (fn [acc [fuel-range percent]]
            (if (in-range? fuel-range fuel-model-number)
              (if (vector? percent)
                (my-rand-range rand-gen (percent 0) (percent 1))
                percent)
              acc))
          0.0
          fuel-range-percents))

(defn surface-fire-spot-fire?
  "Expects surface-fire-spotting config to be a sequence of tuples of
  ranges [lo hi] and spotting probability. The range represents the range (inclusive)
  of fuel model numbers that the spotting probability is set to.
  [[[1 140] 0.0]
  [[141 149] 1.0]
  [[150 256] 1.0]]"
  [{:keys [spotting rand-gen fuel-model-matrix]} [i j] ^double fire-line-intensity]
  (let [{:keys [surface-fire-spotting]} spotting]
    (when (and
           surface-fire-spotting
           (> fire-line-intensity ^double (:critical-fire-line-intensity surface-fire-spotting)))
      (let [fuel-range-percents (:spotting-percent surface-fire-spotting)
            fuel-model-number   (long (t/mget fuel-model-matrix i j))
            spot-percent        (surface-spot-percent fuel-range-percents fuel-model-number rand-gen)]
        (>= spot-percent (my-rand-range rand-gen 0.0 1.0))))))

(defn crown-spot-fire?
  "Determine whether crowning causes spot fires. Config key `:spotting` should
   take either a vector of probabilities (0-1) or a single spotting probability."
  [{:keys [spotting rand-gen]}]
  (when-let [spot-percent (:crown-fire-spotting-percent spotting)]
    (let [^double p (if (vector? spot-percent)
                      (let [[lo hi] spot-percent]
                        (my-rand-range rand-gen lo hi))
                      spot-percent)]
      (>= p (my-rand-range rand-gen 0.0 1.0)))))

(defn- spot-fire? [inputs crown-fire? here fire-line-intensity]
  (if crown-fire?
    (crown-spot-fire? inputs)
    (surface-fire-spot-fire? inputs here fire-line-intensity)))

(defn spread-firebrands
  "Returns a sequence of key value pairs where
  key: [x y] locations of the cell
  val: [t p] where:
  t: time of ignition
  p: ignition-probability"
  [{:keys
    [num-rows num-cols cell-size fuel-model-matrix elevation-matrix spotting rand-gen
     get-temperature get-relative-humidity get-wind-speed-20ft get-wind-from-direction
     get-fuel-moisture-dead-1hr] :as inputs}
   {:keys [firebrand-count-matrix fire-spread-matrix fire-line-intensity-matrix flame-length-matrix]}
   {:keys [cell fire-line-intensity crown-fire?]}
   global-clock]
```

```
  (when (spot-fire? inputs crown-fire? cell fire-line-intensity)
    (let [band              (long (/ global-clock 60.0))
          [i j]             cell
          tmp               (grid-lookup/double-at get-temperature band i j)
          rh                (grid-lookup/double-at get-relative-humidity band i j)
          ws                (grid-lookup/double-at get-wind-speed-20ft band i j)
          wd                (grid-lookup/double-at get-wind-from-direction band i j)
          m1                (if get-fuel-moisture-dead-1hr
                              (grid-lookup/double-at get-fuel-moisture-dead-1hr band i j)
                              (calc-fuel-moisture rh tmp :dead :1hr))
          deltas            (sample-wind-dir-deltas inputs
                                                    fire-line-intensity-matrix
                                                    (convert/mph->mps ws)
                                                    cell)
          wind-to-direction (mod (+ 180 wd) 360)
          firebrands        (firebrands deltas wind-to-direction cell cell-size)]
      (update-firebrand-counts! inputs firebrand-count-matrix fire-spread-matrix cell firebrands)
      (->> (for [[x y] firebrands
                 :when (and (in-bounds? num-rows num-cols [x y])
                            (burnable? fire-spread-matrix fuel-model-matrix cell [x y]))
                 :let  [fine-fuel-moisture   (double m1)
                        ignition-probability (schroeder-ign-prob (convert/F->C (double tmp)) fine-fuel-moisture)
                        decay-constant       (double (:decay-constant spotting))
                        spotting-distance    (convert/ft->m (distance-3d elevation-matrix
                                                                         (double cell-size)
                                                                         [x y]
                                                                         cell))
                        firebrand-count      (t/mget firebrand-count-matrix x y)
                        spot-ignition-p      (spot-ignition-probability ignition-probability
                                                                        decay-constant
                                                                        spotting-distance
                                                                        firebrand-count)]]
            (when (spot-ignition? rand-gen spot-ignition-p)
              (let [[i j] cell
                    t     (spot-ignition-time global-clock
                                              (convert/ft->m (t/mget flame-length-matrix i j)))]
                [[x y] [t spot-ignition-p]])))
           (remove nil?)))))
```

9.2.12 gridfire.spotting-test

```
(ns gridfire.spotting-test
  (:require [clojure.pprint              :as pp]
            [clojure.test                :refer [are deftest is testing use-fixtures run-tests]]
            [gridfire.conversion         :as c]
            [gridfire.grid-lookup        :as grid-lookup]
            [gridfire.spotting           :as spotting]
            [gridfire.spotting.sardoy    :as spotting-sardoy]
            [tech.v3.datatype            :as d]
            [tech.v3.datatype.functional :as dfn]
            [tech.v3.tensor              :as t])
  (:import java.util.Random))

(def ^:private seed 123456789)
(def ^{:dynamic true :private true} *rand-gen* nil)

(defn- create-random-generator [f]
  (binding [*rand-gen* (Random. seed)]
    (f)))

(use-fixtures :each create-random-generator)

(defn- within? [^double a ^double b ^double epsilon]
  (> epsilon (Math/abs (- a b))))

(deftest ^:unit deltas-axis-test
```

```
  (testing "Simple 1-unit parallel to wind."
    (are [result args] (let [[dx-res dy-res] result
                             [dx dy]         (first (apply spotting/deltas-wind->coord args))]
                         (and (within? dx-res dx 0.01) (within? dy-res dy 0.01)))
         [0.0     -1.0]   [[[1 0]] 0]     ;; North
         [0.707   -0.707] [[[1 0]] 45]    ;; NE
         [1.0     0.0]    [[[1 0]] 90]    ;; East
         [0.707   0.707]  [[[1 0]] 135]   ;; SE
         [0.0     1.0]    [[[1 0]] 180]   ;; South
         [-0.707  0.707]  [[[1 0]] 225]   ;; SW
         [-1.0    0.0]    [[[1 0]] 270]   ;; West
         [-0.707  -0.707] [[[1 0]] 315]   ;; NW
         [0.0     -1.0]   [[[1 0]] 360])) ;; North

  (testing "1 unit parallel, 1 unit perpindicular to wind direction."
    (are [result args] (let [[dx-res dy-res] result
                             [dx dy]         (first (apply spotting/deltas-wind->coord args))]
                         (and (within? dx-res dx 0.01) (within? dy-res dy 0.01)))
         [1.0    -1.0]   [[[1 1]] 0]      ;; North
         [1.414  0.0]    [[[1 1]] 45]     ;; NE
         [1.0    1.0]    [[[1 1]] 90]     ;; East
         [0.0    1.414]  [[[1 1]] 135]    ;; SE
         [-1.0   1.0]    [[[1 1]] 180]    ;; South
         [-1.414 0.0]    [[[1 1]] 225]    ;; SW
         [-1.0   -1.0]   [[[1 1]] 270]    ;; West
         [0.0    -1.414] [[[1 1]] 315]    ;; NW
         [1.0    -1.0]   [[[1 1]] 360]))) ;; North

(deftest ^:unit firebrand-test
  (testing "Convert deltas to (i,j) cell in matrix."
    ; NOTE: matrix index [i,j] refers to [row, column]. Therefore, we need to flip
    ; [row,column] to get to [x,y] coordinates."
    (let [cell-size 10.0
          cell      [8 12]]
      (are [result args] (let [[i-res j-res] result
                               [delta wd]    args
                               [i j]         (first (spotting/firebrands [delta] wd cell cell-size))]
                           (and (= i-res i) (= j-res j)))
           [8   12]  [[1.0  0.0]  0]    ;; Same origin
           [7   12]  [[10.0 0.0]  0]    ;; North
           [8   13]  [[10.0 0.0]  90]   ;; East
           [9   12]  [[10.0 0.0]  180]  ;; South
           [8   11]  [[10.0 0.0]  270]  ;; West
           [7   13]  [[10.0 10.0] 0]    ;; North w/ perpindicular component
           [9   13]  [[10.0 10.0] 90]   ;; East w/ perpindicular component
           [9   11]  [[10.0 10.0] 180]  ;; South w/ perpindicular component
           [7   11]  [[10.0 10.0] 270]  ;; West w/ perpindicular component
           [-12 12]  [[200  0]    0]    ;; North, out of bounds
           [8  -8]   [[200  0]    270]  ;; West, out of bounds
          ))))

(comment                                                ;; FIXME restore or delete
  (deftest ^:unit surface-spot-percents
    (testing "Fuel model is within expected ranges."
      (let [spot-percents [[[1 140] 1.0]
                           [[141 149] 2.0]
                           [[150 256] 3.0]]]

        (is (= 1.0 (spotting/surface-spot-percent spot-percents 5 *rand-gen*)))

        (is (= 2.0 (spotting/surface-spot-percent spot-percents 143 *rand-gen*)))

        (is (= 3.0 (spotting/surface-spot-percent spot-percents 155 *rand-gen*)))))

    (testing "Overlapping ranges"
      (let [spot-percents [[[1 140] 1.0]
                           [[130 149] 2.0]]]
```

```
        (is (= 2.0 (spotting/surface-spot-percent spot-percents 133 *rand-gen*))
            "should overwite percents of previous range in the sequence")))

    (testing "TODO: Given a pair of percentages, random percent is generated")))

(deftest ^:unit test-heat-of-preignition
  (testing "Heat of Preignition using eq. 9 of Schroeder (1969)."
    (are [result args] (within? result (apply spotting/heat-of-preignition args) 0.001)
         ; Q_ig (cal/g) [temperature (C) fuel-moisture (%)]
         0.0            [320.0          0.0]
         144.512        [0.0            0.0]

         ; Schroeder Fig. 2
         153.816        [(c/F->C 30.0)  0.01]
         193.996        [(c/F->C 160.0) 0.1]
         248.086        [(c/F->C 70.0)  0.15]
         292.814        [(c/F->C 130.0) 0.25])))

(deftest ^:unit test-schroeder-ign-prob
  (testing "Using Schroeder (1969)."
    (are [result args] (within? result (apply spotting/schroeder-ign-prob args) 0.01)
         ; Probability (0-1) [temperature (C) fuel-moisture (%)]
         0.0                [0.0   0.3703]
         1.0                [150.0 0.01]

         ; Schroeder Table 1
         0.87               [(c/F->C 35.0) 0.015]
         0.97               [(c/F->C 75.0) 0.015]
         0.74               [(c/F->C 35.0) 0.025]
         0.84               [(c/F->C 75.0) 0.025]
         0.51               [(c/F->C 35.0) 0.05]
         0.59               [(c/F->C 75.0) 0.05])))

(deftest ^:unit test-spot-ignition-probability
  (testing "Using Perryman (2012)."
    (let [lambda 0.005]
      (are [result args] (within? result (apply spotting/spot-ignition-probability args) 0.001)
           ; Probability (0-1) [ignition-probability (0-1) decay-constant spotting-distance (m) firebrand-count]

           ; Increasing Schroeder Ignition Probability
           0.121              [0.2 lambda 100 1.0]
           0.242              [0.4 lambda 100 1.0]
           0.363              [0.6 lambda 100 1.0]
           0.485              [0.8 lambda 100 1.0]
           0.606              [1.0 lambda 100 1.0]

           ; Increasing Firebrands Count
           0.0                [0.5 lambda 100 0.0]
           0.303              [0.5 lambda 100 1.0]
           0.514              [0.5 lambda 100 2.0]
           0.973              [0.5 lambda 100 10.0]

           ; Increasing Distance
           0.303              [0.5 lambda 100  1.0]
           0.183              [0.5 lambda 200  1.0]
           0.0                [0.5 lambda 2000 1.0]))))

(deftest ^:unit test-spot-ignition-time
  (testing "Calculating t-max from Albini (1976)."
    (are [result args] (within? result (apply spotting/albini-t-max args) 0.1)
         ; Time to max height (min) [Flame length (m)]
         183.3                    [1.0]
         19.9                     [5.0]
         8.5                      [10.0]
         2.7                      [30.0]
         1.28                     [84.0]))

  (testing "Calculating time to spot ignition using Perryman."
```

```
    (are [result args] (within? result (apply spotting/spot-ignition-time args) 0.1)
         ; Ignition time (m) [Global clock time (m) Flame length (m)]
         386.6               [0 1.0]
         59.8                [0 5.0]
         37.0                [0 10.0]
         25.5                [0 30.0]
         22.56               [0 84.0]))

(deftest ^:unit test-spot-ignition?
  (testing "Spot ignition probability exceeds a random number generator."
    (are [result args] (= result (apply spotting/spot-ignition? args))
         ; Result [RNG spot-probability (0-1)]
         true     [*rand-gen* 1.0]
         false    [*rand-gen* 0.0]
         true     [*rand-gen* 0.5])))

(deftest ^:unit test-crown-spot-fire?
  (testing "Whether spotting from crown fire occurs based on a single or range of probabilities."
    (let [->argmap (fn [spotting] {:rand-gen *rand-gen*
                                   :spotting {:crown-fire-spotting-percent spotting}})]
      (are [result args] (= result (apply spotting/crown-spot-fire? args))
           ; Result [crown fire spotting probability (0-1)]
           true     [(->argmap 1.0)]
           false    [(->argmap 0.0)]
           true     [(->argmap 0.5)]))))

(deftest ^:unit test-surface-fire-spot-fire?
  (testing "Whether spotting from surface fire occurs based on a single or range of probabilities."
    (let [fuel-model (d/clone (dfn/+ (t/new-tensor [10 10]) 10))
          ->argmap   (fn [crit-fire-line-intensity spotting]
                       {:rand-gen       *rand-gen*
                        :get-fuel-model (-> fuel-model
                                            (grid-lookup/add-double-getter)
                                            (grid-lookup/mgetter-double))
                        :spotting       {:surface-fire-spotting
                                         {:critical-fire-line-intensity crit-fire-line-intensity
                                          :spotting-percent             spotting}}})]
      (are [result args] (= result (apply spotting/surface-fire-spot-fire? args))
           ; Result [crit. fire line intensity (kW/m) surface fire spotting probability (0-1) here (coords) fire line intensity (kW/
           nil      [(->argmap 1500 nil)                [0 0] 1000]
           true     [(->argmap 500  [[[1 20] 1.0]])     [0 0] 1000]
           false    [(->argmap 500  [[[1 20] 0.001]])   [0 0] 1000]
           true     [(->argmap 500  [[[1 20] 0.999]])   [0 0] 1000]))))

(comment
  (run-tests 'gridfire.spotting-test)
  )
```

9.2.13 gridfire.behaveplus-results

```
(ns gridfire.behaveplus-results
  (:require [gridfire.fuel-models :refer [fuel-models]]))

(defn within [a b epsilon]
  (<= (Math/abs ^double (- a b)) epsilon))

(defn is-sb40-fuel-model-number?
  [^long fm-number]
  (and (> fm-number 100)
       ;; NOTE This check is redundant... at the time of writing. (Val, 27 Oct 2022)
       ;; I don't trust that it will always be so.
       (<= fm-number 204)))

(def sb40-fuel-models (filterv is-sb40-fuel-model-number? (sort (keys fuel-models))))

(def test-fuel-moisture
```

```
  {:dry {:dead {:1hr 0.04 :10hr 0.04 :100hr 0.06}
         :live {:herbaceous 0.30 :woody 0.70}}
   :mid {:dead {:1hr 0.04 :10hr 0.04 :100hr 0.06}
         :live {:herbaceous 0.75 :woody 0.70}}
   :wet {:dead {:1hr 0.04 :10hr 0.04 :100hr 0.06}
         :live {:herbaceous 1.20 :woody 0.70}}})

;; INPUTS TO BEHAVEPLUS5:
;; - fuel-moisture = (test-fuel-moisture :dry)
;; - midflame wind speed (mph) = 0
;; - slope steepness (%) = 0
;;
;; OUTPUTS LISTED IN TABLE BELOW:
;; - ROS_max (ch/h)
;; - Fireline Intensity (Btu/ft/s)
;; - Flame Length (ft)
;; - Reaction Intensity (Btu/ft2/min)
;; - Live Fuel Moisture of Extinction (%)
;; - Residence Time (min)
;; - Fuel Load Transferred (%)
(def behaveplus5-surface-fire-values-dry-no-wind-no-slope
  {:GR1 [1.1  2   0.6  430   15   0.19 100]
   :GR2 [2.3  10  1.3  1135  15   0.21 100]
   :GR3 [2.8  22  1.9  1487  30   0.30 100]
   :GR4 [4.7  40  2.4  2209  15   0.21 100]
   :GR5 [4.8  88  3.5  4260  40   0.24 100]
   :GR6 [7.2  164 4.7  6520  40   0.19 100]
   :GR7 [8.7  258 5.8  7770  15   0.21 100]
   :GR8 [8.6  449 7.5  9700  30   0.29 100]
   :GR9 [14.9 977 10.7 14978 40   0.24 100]
   :GS1 [1.1  6   1.0  1512  258  0.21 100]
   :GS2 [1.6  15  1.6  2467  304  0.21 100]
   :GS3 [2.4  44  2.6  4110  456  0.24 100]
   :GS4 [3.2  245 5.7  17939 227  0.24 100]
   :SH1 [0.4  2   0.6  966   76   0.23 100]
   :SH2 [0.8  20  1.8  6083  124  0.23 0]
   :SH3 [0.3  3   0.8  1981  53   0.28 0]
   :SH4 [1.8  28  2.1  3610  128  0.23 0]
   :SH5 [2.9  93  3.6  5720  337  0.31 0]
   :SH6 [1.6  51  2.8  5375  670  0.34 0]
   :SH7 [2.4  104 3.8  7668  356  0.31 0]
   :SH8 [1.7  73  3.2  8462  197  0.28 0]
   :SH9 [3.1  215 5.3  13425 283  0.28 100]
   :TU1 [0.3  2   0.7  1821  202  0.24 100]
   :TU2 [1.0  8   1.2  2059  2392 0.22 0]
   :TU3 [2.0  36  2.3  4106  519  0.24 100]
   :TU4 [1.4  31  2.2  6882  503  0.17 0]
   :TU5 [1.2  62  3.0  9075  737  0.31 0]
   :TL1 [0.1  0   0.3  527   30   0.22 0]
   :TL2 [0.2  1   0.4  769   25   0.21 0]
   :TL3 [0.2  1   0.4  881   20   0.25 0]
   :TL4 [0.3  2   0.5  1101  25   0.24 0]
   :TL5 [0.5  4   0.8  1751  25   0.22 0]
   :TL6 [0.7  7   1.1  2500  25   0.20 0]
   :TL7 [0.4  4   0.9  1779  25   0.31 0]
   :TL8 [0.9  13  1.5  3617  35   0.22 0]
   :TL9 [1.3  27  2.1  5243  35   0.22 0]
   :SB1 [0.8  9   1.3  2753  25   0.23 0]
   :SB2 [1.7  36  2.3  5547  25   0.20 0]
   :SB3 [2.8  79  3.4  7633  25   0.20 0]
   :SB4 [4.5  134 4.3  8049  25   0.20 0]})

;; INPUTS TO BEHAVEPLUS5:
;; - fuel-moisture = (test-fuel-moisture :mid)
;; - midflame wind speed (mph) = 0
;; - slope steepness (%) = 0
;;
;; OUTPUTS LISTED IN TABLE BELOW:
```

```
;; - ROS_max (ch/h)
;; - Fireline Intensity (Btu/ft/s)
;; - Flame Length (ft)
;; - Reaction Intensity (Btu/ft2/min)
;; - Live Fuel Moisture of Extinction (%)
;; - Residence Time (min)
;; - Fuel Load Transferred (%)
(def behaveplus5-surface-fire-values-mid-no-wind-no-slope
  {:GR1 [0.6 1   0.4 448   404  0.19 50]
   :GR2 [1.1 5   0.9 1140  290  0.21 50]
   :GR3 [1.2 9   1.3 1435  411  0.30 50]
   :GR4 [2.2 19  1.7 2233  307  0.21 50]
   :GR5 [2.1 36  2.4 3998  411  0.24 50]
   :GR6 [2.8 57  2.9 5852  309  0.19 50]
   :GR7 [4.2 128 4.2 7937  335  0.21 50]
   :GR8 [3.6 183 4.9 9254  385  0.29 50]
   :GR9 [6.4 389 7.0 13981 401  0.24 50]
   :GS1 [0.8 4   0.9 1345  108  0.21 50]
   :GS2 [1.2 11  1.3 2291  169  0.21 50]
   :GS3 [1.5 23  1.9 3429  161  0.24 50]
   :GS4 [2.5 169 4.8 16001 116  0.24 50]
   :SH1 [0.1 0   0.2 369   56   0.23 50]
   :SH2 [0.8 20  1.8 6083  124  0.23 0 ]
   :SH3 [0.3 3   0.8 1981  53   0.28 0 ]
   :SH4 [1.8 28  2.1 3610  128  0.23 0 ]
   :SH5 [2.9 93  3.6 5720  337  0.31 0 ]
   :SH6 [1.6 51  2.8 5375  670  0.34 0 ]
   :SH7 [2.4 104 3.8 7668  356  0.31 0 ]
   :SH8 [1.7 73  3.2 8462  197  0.28 0 ]
   :SH9 [3.0 217 5.3 14085 217  0.28 50]
   :TU1 [0.3 2   0.6 1816  150  0.24 50]
   :TU2 [1.0 8   1.2 2059  2392 0.22 0 ]
   :TU3 [1.6 27  2.0 3828  317  0.24 50]
   :TU4 [1.4 31  2.2 6882  503  0.17 0 ]
   :TU5 [1.2 62  3.0 9075  737  0.31 0 ]
   :TL1 [0.1 0   0.3 527   30   0.22 0 ]
   :TL2 [0.2 1   0.4 769   25   0.21 0 ]
   :TL3 [0.2 1   0.4 881   20   0.25 0 ]
   :TL4 [0.3 2   0.5 1101  25   0.24 0 ]
   :TL5 [0.5 4   0.8 1751  25   0.22 0 ]
   :TL6 [0.7 7   1.1 2500  25   0.20 0 ]
   :TL7 [0.4 4   0.9 1779  25   0.31 0 ]
   :TL8 [0.9 13  1.5 3617  35   0.22 0 ]
   :TL9 [1.3 27  2.1 5243  35   0.22 0 ]
   :SB1 [0.8 9   1.3 2753  25   0.23 0 ]
   :SB2 [1.7 36  2.3 5547  25   0.20 0 ]
   :SB3 [2.8 79  3.4 7633  25   0.20 0 ]
   :SB4 [4.5 134 4.3 8049  25   0.20 0 ]})

;; INPUTS TO BEHAVEPLUS5:
;; - fuel-moisture = (test-fuel-moisture :mid)
;; - midflame wind speed (mph) = 10
;; - slope steepness (%) = 20
;;
;; OUTPUTS LISTED IN TABLE BELOW:
;; - ROS_max (ch/h)
;; - Fireline Intensity (Btu/ft/s)
;; - Flame Length (ft)
;; - Reaction Intensity (Btu/ft2/min)
;; - Live Fuel Moisture of Extinction (%)
;; - Residence Time (min)
;; - Fuel Load Transferred (%)
;; - Wind Factor
;; - Slope Factor
(def behaveplus5-surface-fire-values-mid-with-wind-and-slope
  {:GR1 [11.9  18     1.7  448  404  0.19 50 68.8 1.5]
   :GR2 [72.0  318    6.4  1140 290  0.21 50 65.4 1.5]
   :GR3 [84.1  659    8.9  1435 411  0.30 50 68.0 1.5]
```

```
   :GR4 [148.3 1277  12.1  2233 307  0.21 50 66.0 1.5]
   :GR5 [113.8 1965  14.7  3998 411  0.24 50 51.8 1.2]
   :GR6 [148.8 3055  18.0  5852 309  0.19 50 51.0 1.2]
   :GR7 [223.7 6814  26.1  7937 335  0.21 50 51.2 1.2]
   :GR8 [182.0 9107  29.8  9254 385  0.29 50 47.7 1.2]
   :GR9 [326.3 19924 42.7 13981 401  0.24 50 49.1 1.2]
   :GS1 [47.1  243    5.6  1345 108  0.21 50 58.3 1.3]
   :GS2 [70.8  625    8.7  2291 169  0.21 50 55.3 1.3]
   :GS3 [85.2  1275  12.1  3429 161  0.24 50 53.1 1.3]
   :GS4 [85.1  5874  24.4 16001 116  0.24 50 32.8 0.9]
   :SH1 [2.1   3      0.8   369  56  0.23 50 51.9 1.2]
   :SH2 [24.6  629    8.7  6083 124  0.23  0 29.3 0.8]
   :SH3 [12.6  128    4.2  1981  53  0.28  0 36.8 1.0]
   :SH4 [108.3 1636  13.5  3610 128  0.23  0 56.3 1.3]
   :SH5 [174.2 5603  23.9  5720 337  0.31  0 57.9 1.4]
   :SH6 [68.2  2254  15.7  5375 670  0.34  0 41.8 1.1]
   :SH7 [113.6 4972  22.6  7668 356  0.31  0 45.7 1.2]
   :SH8 [69.6  2991  17.9  8462 197  0.28  0 39.0 1.0]
   :SH9 [123.9 8914  29.5 14085 217  0.28 50 39.0 1.0]
   :TU1 [9.6   76     3.3  1816 150  0.24 50 32.4 0.9]
   :TU2 [40.5  333    6.5  2059 2392 0.22  0 39.2 1.0]
   :TU3 [77.5  1297  12.2  3828 317  0.24 50 46.6 1.1]
   :TU4 [46.4  1013  10.9  6882 503  0.17  0 30.5 0.7]
   :TU5 [26.0  1360  12.4  9075 737  0.31  0 20.3 0.7]
   :TL1 [1.2   3      0.7   527  30  0.22  0 17.3 0.5]
   :TL2 [3.1   9      1.2   769  25  0.21  0 19.1 0.5]
   :TL3 [4.3   18     1.7   881  20  0.25  0 20.3 0.6]
   :TL4 [7.4   37     2.4  1101  25  0.24  0 22.0 0.7]
   :TL5 [14.0  101    3.8  1751  25  0.22  0 24.7 0.7]
   :TL6 [19.7  179    4.9  2500  25  0.20  0 25.4 0.7]
   :TL7 [7.6   77     3.3  1779  25  0.31  0 15.7 0.6]
   :TL8 [18.7  270    5.9  3617  35  0.22  0 19.2 0.6]
   :TL9 [27.8  591    8.5  5243  35  0.22  0 20.1 0.6]
   :SB1 [19.3  226    5.5  2753  25  0.23  0 22.8 0.7]
   :SB2 [50.1  1038  11.0  5547  25  0.20  0 26.9 0.7]
   :SB3 [92.1  2558  16.6  7633  25  0.20  0 30.7 0.8]
   :SB4 [177.7 5281  23.2  8049  25  0.20  0 37.6 0.9]})

   ;;  Fuel  ROS   Fireline  Flame Direction Effective
   ;; Model (max) Intensity Length  Max ROS     Wind
   ;;        ch/h  Btu/ft/s   ft      deg       mi/h
(def behaveplus5-surface-fire-values-mid-with-cross-wind-and-slope-90-180
  {:GR1 [11.9    18      1.7     271       4.6]
   :GR2 [70.5    311     6.3     271       10.0]
   :GR3 [82.4    645     8.8     271       10.0]
   :GR4 [145.2   1250    12.0    271       10.0]
   :GR5 [111.3   1921    14.6    271       10.0]
   :GR6 [145.5   2989    17.9    271       10.0]
   :GR7 [218.7   6663    25.8    271       10.0]
   :GR8 [177.7   8893    29.5    271       10.0]
   :GR9 [318.9   19468   42.3    271       10.0]
   :GS1 [46.1    238     5.6     271       10.0]
   :GS2 [69.3    611     8.6     271       10.0]
   :GS3 [83.3    1246    11.9    271       10.0]
   :GS4 [82.9    5728    24.1    272       10.0]
   :SH1 [2.1     3       0.8     271       3.8]
   :SH2 [23.9    613     8.6     272       10.0]
   :SH3 [12.3    125     4.2     272       10.0]
   :SH4 [105.9   1600    13.4    271       10.0]
   :SH5 [170.3   5479    23.6    271       10.0]
   :SH6 [66.5    2199    15.5    272       10.0]
   :SH7 [110.9   4853    22.3    271       10.0]
   :SH8 [67.9    2917    17.7    272       10.0]
   :SH9 [120.8   8694    29.2    272       10.0]
   :TU1 [9.3     74      3.3     272       10.0]
   :TU2 [39.6    325     6.4     271       10.0]
   :TU3 [75.7    1267    12.0    271       10.0]
   :TU4 [45.4    992     10.8    271       10.0]
```

```
   :TU5 [25.2    1318    12.3    272       10.0]
   :TL1 [1.2     3       0.7     272       5.4]
   :TL2 [3.1     9       1.2     272       7.9]
   :TL3 [4.3     18      1.7     272       9.0]
   :TL4 [7.2     36      2.3     272       10.0]
   :TL5 [13.7    98      3.7     272       10.0]
   :TL6 [19.2    175     4.8     271       10.0]
   :TL7 [7.4     75      3.3     272       10.0]
   :TL8 [18.3    263     5.8     272       10.0]
   :TL9 [27.0    576     8.4     272       10.0]
   :SB1 [18.8    220     5.4     272       10.0]
   :SB2 [48.9    1013    10.9    271       10.0]
   :SB3 [89.9    2498    16.4    271       10.0]
   :SB4 [173.6   5160    23.0    271       10.0]})

   ;;  Fuel  ROS   Fireline  Flame Direction Effective
   ;; Model (max) Intensity Length  Max ROS     Wind
   ;;        ch/h  Btu/ft/s   ft      deg       mi/h
(def behaveplus5-surface-fire-values-mid-with-cross-wind-and-slope-135-180
  {:GR1 [11.9    18      1.7     316       4.6   2.1 ]
   :GR2 [71.6    316     6.4     316       10.1  3.5 ]
   :GR3 [83.6    655     8.9     316       10.1  3.5 ]
   :GR4 [147.4   1269    12.0    316       10.1  3.5 ]
   :GR5 [113.1   1952    14.7    316       10.1  3.5 ]
   :GR6 [147.8   3036    18.0    316       10.1  3.5 ]
   :GR7 [222.2   6771    26.0    316       10.1  3.5 ]
   :GR8 [180.8   9045    29.7    316       10.1  3.5 ]
   :GR9 [324.2   19791   42.6    316       10.1  3.5 ]
   :GS1 [46.8    242     5.6     316       10.1  3.5 ]
   :GS2 [70.4    621     8.7     316       10.1  3.5 ]
   :GS3 [84.7    1267    12.0    316       10.1  3.5 ]
   :GS4 [84.4    5831    24.3    316       10.1  3.5 ]
   :SH1 [2.1     3       0.8     316       3.8   1.9 ]
   :SH2 [24.4    624     8.7     316       10.1  3.5 ]
   :SH3 [12.5    128     4.2     316       10.2  3.5 ]
   :SH4 [107.6   1625    13.5    316       10.1  3.5 ]
   :SH5 [173.1   5567    23.8    316       10.1  3.5 ]
   :SH6 [67.7    2238    15.6    316       10.2  3.5 ]
   :SH7 [112.8   4938    22.5    316       10.2  3.5 ]
   :SH8 [69.1    2970    17.8    316       10.1  3.5 ]
   :SH9 [123.0   8850    29.4    316       10.2  3.5 ]
   :TU1 [9.5     76      3.3     316       10.1  3.5 ]
   :TU2 [40.3    331     6.5     316       10.1  3.5 ]
   :TU3 [77.0    1288    12.1    316       10.1  3.5 ]
   :TU4 [46.1    1007    10.8    316       10.1  3.5 ]
   :TU5 [25.8    1348    12.4    316       10.2  3.6 ]
   :TL1 [1.2     3       0.7     316       5.4   2.3 ]
   :TL2 [3.1     9       1.2     316       7.9   3.0 ]
   :TL3 [4.3     18      1.7     316       9.0   3.3 ]
   :TL4 [7.3     36      2.3     316       10.2  3.5 ]
   :TL5 [13.9    100     3.7     316       10.1  3.5 ]
   :TL6 [19.6    178     4.9     316       10.1  3.5 ]
   :TL7 [7.5     77      3.3     316       10.2  3.6 ]
   :TL8 [18.6    268     5.9     316       10.1  3.5 ]
   :TL9 [27.6    587     8.4     316       10.1  3.5 ]
   :SB1 [19.2    225     5.4     316       10.1  3.5 ]
   :SB2 [49.7    1031    10.9    316       10.1  3.5 ]
   :SB3 [91.5    2541    16.6    316       10.1  3.5 ]
   :SB4 [176.5   5246    23.1    316       10.1  3.5 ]})
```

9.2.14 Burn Period

As an approximation of reduced fire activity during the night, GridFire optionally supports limiting the simulated spread to a range of hours within each simulated day, by specifying a Burn Period.

There are several methods for specifying a Burn Period. The first method consists of explicitly providing

start and end times:

```
(ns gridfire.spec.burn-period-test
  (:require [clojure.spec.alpha        :as s]
            [clojure.test              :refer [deftest is testing]]
            [gridfire.spec.burn-period :as burn-period]))

(deftest ^:unit burn-period-test
  (testing "valid"
    (is (s/valid? ::burn-period/burn-period {:start "08:00"
                                             :end   "20:00"}))))
```

The 2nd method consists of resolving the Burn Period through astronomy computations (that is, figuring the hours of sunrise and sunset from the simulation time and location):

```
(ns gridfire.burn-period-test
  (:require [clojure.test        :refer [deftest is testing use-fixtures]]
            [gridfire.core       :refer [load-config! load-inputs!]]
            [gridfire.utils.test :refer [with-temp-directories]]))

(use-fixtures :once (with-temp-directories ["outputs/simple/"]))

(deftest ^:unit burn-period-examples-test
  (testing (str "The Burn period gets resolved into " :burn-period-samples)
    (testing (str "from sunrise/sunset when " :burn-period-frac " is configured:")
      (let [base-config  (merge (load-config! "test/gridfire/resources/canonical_test/base-config.edn")
                                {:burn-period-frac        0.6
                                 :weather-start-timestamp #inst "2022-01-01"
                                 :ignition-start-times    [(* 60 24 365 0.0)
                                                           (* 60 24 365 0.15)
                                                           (* 60 24 365 0.25)
                                                           (* 60 24 365 0.5)
                                                           (* 60 24 365 0.75)
                                                           (* 60 24 365 0.99)]})
            ign-start-ts [#inst "2022-01-01T00:00:00.000-00:00"
                          #inst "2022-02-24T18:00:00.000-00:00"
                          #inst "2022-04-02T06:00:00.000-00:00"
                          #inst "2022-07-02T12:00:00.000-00:00"
                          #inst "2022-10-01T18:00:00.000-00:00"
                          #inst "2022-12-28T08:24:00.000-00:00"]]
        (testing (str "using the sunrise-to-sunset duration when " :burn-period-length " is not configured")
          (let [config base-config
                inputs (load-inputs! config)]
            (is (= {:ignition-start-timestamps ign-start-ts
                    :burn-period-samples       [{:burn-period-start "15:23" :burn-period-end "01:07"}
                                                {:burn-period-start "14:49" :burn-period-end "02:02"}
                                                {:burn-period-start "13:58" :burn-period-end "02:34"}
                                                {:burn-period-start "12:58" :burn-period-end "03:33"}
                                                {:burn-period-start "14:06" :burn-period-end "01:56"}
                                                {:burn-period-start "15:22" :burn-period-end "01:05"}]}
                   (-> inputs
                       (select-keys [:ignition-start-timestamps
                                     :burn-period-samples])))))))
        (testing (str "using " :burn-period-length " when configured")
          (let [config (merge base-config
                              {:burn-period-length 10.0})
                inputs (load-inputs! config)]
            (is (= {:ignition-start-timestamps ign-start-ts
                    :burn-period-samples       [{:burn-period-start "16:13" :burn-period-end "02:13"}
                                                {:burn-period-start "16:33" :burn-period-end "02:33"}
                                                {:burn-period-start "16:32" :burn-period-end "02:32"}
                                                {:burn-period-start "16:43" :burn-period-end "02:43"}
                                                {:burn-period-start "16:12" :burn-period-end "02:12"}
                                                {:burn-period-start "16:12" :burn-period-end "02:12"}]}
                   (-> inputs
```

```
                    (select-keys [:ignition-start-timestamps
                                  :burn-period-samples])))))))))))
```

1. Implementation Code

```
(ns gridfire.burn-period
  (:require [gridfire.burn-period.sunrise-sunset :as burnp-sun]
            [gridfire.utils.geo                  :refer [resolve-simulation-lat+lon]]))

(defn from-sunrise-sunset
  [inputs ignition-start-timestamp]
  (burnp-sun/infer-burn-period (merge (select-keys inputs [:burn-period-frac
                                                           :burn-period-length])
                                      (let [[lat lon] (resolve-simulation-lat+lon inputs)]
                                        {::burnp-sun/lat-deg lat
                                         ::burnp-sun/lng-deg lon})
                                      {:ignition-start-timestamp ignition-start-timestamp})))
```

```
(ns gridfire.burn-period.sunrise-sunset
  (:require [clojure.test :as test])
  (:import  (java.time Instant ZonedDateTime ZoneOffset)))

;; WARNING running this script in polar circles is at your own risk!

;;==============================================================================
;; Sunrise/sunset computations.
;;==============================================================================

;; Original Fortran code from ElmFire:
;!This gives the sunrise and sunset time in local time: (Chris Lautenberger)
;
;!**
;SUBROUTINE SUNRISE_SUNSET_CALCS(LON_DEG,LAT_DEG,UTC_OFFSET_HOURS,YEAR,HOUR_OF_YEAR)
;!**
;
;REAL, INTENT(IN) :: LON_DEG, LAT_DEG
;INTEGER, INTENT(IN) :: UTC_OFFSET_HOURS, YEAR, HOUR_OF_YEAR
;LOGICAL :: LEAPYEAR
;INTEGER :: DAY_OF_YEAR, HOUR_OF_DAY
;REAL :: DAYS_PER_YEAR, GAMMA, EQTIME, COSPIMTHETA, DECL, HA, HA_SUNRISE, LAT_RAD, LON_RAD, PHI, THETA, TIME_OFFSET, TST, &
;        SUNRISE_MIN_UTC, SUNRISE_H_LOCAL, SUNRISE_H_UTC, SUNSET_MIN_UTC, HA_SUNSET, SUNSET_H_UTC, SUNSET_H_LOCAL
;
;LON_RAD = LON_DEG * PI / 180.
;LAT_RAD = LAT_DEG * PI / 180.
;
;LEAPYEAR = .FALSE.
;IF (MOD(YEAR,4) .EQ. 0) LEAPYEAR = .TRUE.
;DAYS_PER_YEAR = 365.
;IF (LEAPYEAR) DAYS_PER_YEAR = 366.
;
;DAY_OF_YEAR = 1 + FLOOR(REAL(HOUR_OF_YEAR) / 24.)
;HOUR_OF_DAY = HOUR_OF_YEAR - (DAY_OF_YEAR - 1) * 24
;GAMMA = 2.0 * (PI/DAYS_PER_YEAR) * (DAY_OF_YEAR - 1)
;
;EQTIME = 229.18 * ( 0.000075 + 0.001868COS(GAMMA) - 0.032077SIN(GAMMA) - 0.014615COS(2.GAMMA) - 0.040849SIN(2.GAMMA) )
;DECL   = 0.006918 - 0.399912COS(GAMMA) + 0.070257SIN(GAMMA) - 0.006758COS(2.GAMMA) + 0.000907SIN(2.GAMMA) - 0.002697COS(3.GAM
;
;! Begin part not needed for sunrise / sunset calcs:
;TIME_OFFSET = EQTIME + 4. * LON_DEG - 60. * UTC_OFFSET_HOURS
;TST = REAL(HOUR_OF_DAY) * 60. + TIME_OFFSET
;HA = 0.25 * TST - 180.
;PHI = ACOS(SIN(LAT_RAD)SIN(DECL)+COS(LAT_RAD)COS(DECL)COS(HAPI/180))
;COSPIMTHETA = (SIN(LAT_RAD)COS(PHI) - SIN(DECL)) / (COS(LAT_RAD)SIN(PHI))
;THETA = PI - ACOS(COSPIMTHETA)
;! End part not needed for sunrise / sunset calcs
```

```
;
;HA_SUNRISE = ACOS( COS(90.833PI/180) / (COS(LAT_RAD)COS(DECL)) -TAN(LAT_RAD)TAN(DECL) )
;SUNRISE_MIN_UTC = 720. - 4.(LON_DEG + HA_SUNRISE*180./PI) - EQTIME
;SUNRISE_H_UTC = SUNRISE_MIN_UTC / 60.
;SUNRISE_H_LOCAL = SUNRISE_H_UTC + UTC_OFFSET_HOURS
;
;HA_SUNSET = -HA_SUNRISE
;SUNSET_MIN_UTC = 720. - 4.(LON_DEG + HA_SUNSET180./PI) - EQTIME
;SUNSET_H_UTC = SUNSET_MIN_UTC / 60.
;SUNSET_H_LOCAL = SUNSET_H_UTC + UTC_OFFSET_HOURS
;
;SUNRISE_HOUR = SUNRISE_H_LOCAL ! Scope is global
;SUNSET_HOUR  = SUNSET_H_LOCAL ! Scope is global
;
;! *
;END SUBROUTINE SUNRISE_SUNSET_CALCS
;!**
;
;!Once sunrise / sunset have been determined, the burn period is controlled by two parameters - BURN_PERIOD_CENTER_FRACTION an
;
;IF (USE_DIURNAL_ADJUSTMENT_FACTOR) THEN
;  BURN_PERIOD_CENTER_HOUR = SUNRISE_HOUR + BURN_PERIOD_CENTER_FRAC * (SUNSET_HOUR - SUNRISE_HOUR)
;  BURN_PERIOD_START_HOUR  = BURN_PERIOD_CENTER_HOUR - 0.5 * BURN_PERIOD_LENGTH
;  BURN_PERIOD_STOP_HOUR   = BURN_PERIOD_CENTER_HOUR + 0.5 * BURN_PERIOD_LENGTH
;
;  HOUR_OF_DAY = FORECAST_START_HOUR + T / 3600.0
;  HOUR_OF_DAY = MODULO(HOUR_OF_DAY, 24.)
;  IF (HOUR_OF_DAY .GT. BURN_PERIOD_START_HOUR .AND. HOUR_OF_DAY .LT. BURN_PERIOD_STOP_HOUR) THEN
;     DIURNAL_ADJUSTMENT_FACTOR = 1.0
;  ELSE
;     DIURNAL_ADJUSTMENT_FACTOR = OVERNIGHT_ADJUSTMENT_FACTOR
;  ENDIF
;ENDIF

(defn fourier-series-approximation
  "Computes an approximation of a real-valued angular-arg function by a (truncated) Fourier Series.
  Given:
  - cos+sin-coeffs: A sequence of [cos-coeff-k sin-coeff-k] pairs of doubles,
  corresponding to the terms of frequency k/(2*PI) rotations/rad,
  (sin-coeff-0 need not be specified)
  - gamma: an angle in [0, 2*PI]

  Returns the Fourier series evaluated at gamma,
  in the same physical unit as the coefficients."
  ^double
  [cos+sin-coeffs ^double gamma]
  (->> cos+sin-coeffs
       (map-indexed
        (fn [^long k [cos-coeff sin-coeff]]
          (let [sin-coeff (or sin-coeff 0.0)]
            (+
             (* (double cos-coeff) (Math/cos (* k gamma)))
             (* (double sin-coeff) (Math/sin (* k gamma)))))))
       (reduce + 0.0)))

(defn eqtime-at
  "[min] the correcting offset from the sunrise-sunset midpoint to UTC-clock noon,
  based on the time-of-year angle gamma."
  ^double [^double gamma]
  (*
   229.18
   (fourier-series-approximation
    [[0.000075]
     [0.001868 -0.032077]
     [-0.014615 -0.040849]]
    gamma)))

(defn declivity-at
```

```
  "[rad] the angle between the Earth's equatorial plane and its orbital plane,
  based on the time-of-year angle gamma."
  ^double [^double gamma]
  (fourier-series-approximation
   [[0.006918]
    [-0.399912 0.070257]
    [-0.006758 0.000907]
    [-0.002697 0.00148]]
   gamma))

(defn daylight-half-angle
  "[rad] (Approximately) half of the angle exposed to sunlight in a given iso-latitude circle.

  Arguments:
  - lat: latitude [rad],
  - decl: declivity [rad]"
  ^double [^double lat ^double decl]
  (Math/acos
   (+
    ;; Approximate cosine of daylight-half-angle, see e.g:
    ;; https://vvvvalvalval.github.io/posts/2019-12-03-inferring-earth-tilt-from-day-lengths.html#Physical-Model
    (* -1.0 (Math/tan lat) (Math/tan decl))
    ;; Small correction
    (/
     (Math/cos (* Math/PI (/ 90.833 180.0)))
     (* (Math/cos lat) (Math/cos decl))))))

(defn longitudinal-angle->duration
  "[min] converts an angular offset (in rad) to a time offset."
  ^double [^double lng-angle]
  (*
   ;; APPROXIMATE - the Earth actually rotates
   ;; a little more than 360° between 2 solar noons,
   ;; due to orbital movement
   ;; (about (360/365)  0.98° more, it depends on the time of the year,
   ;; but the error is something like 4 minutes per rotation).
   4.0 ;; [min/deg] (/ (* 24 60) 360)
   (/ 180.0 Math/PI) ;; [deg/rad]
   lng-angle))

(def ^:const min-per-hour 60.0)

(defn min->hr
  ^double [^double t-min]
  (/ t-min min-per-hour))

(defn deg->rad
  ^double [^double theta-deg]
  (-> theta-deg (* Math/PI) (/ 180.0)))

(defn sunrise-hour
  "[hr] UTC hour of the day at which the sun rises."
  ^double [^double lat ^double lng ^double gamma]
  (let [eqtime (eqtime-at gamma)
        decl   (declivity-at gamma)
        ha     (daylight-half-angle lat decl)]
    (+
     (- 12.0 (min->hr eqtime))
     (-> (- lng) (- ha)
         (longitudinal-angle->duration)
         (min->hr)))))

(defn sunset-hour
  "[hr] UTC hour of the day at which the sun sets."
  ^double [^double lat ^double lng ^double gamma]
  (let [eqtime (eqtime-at gamma)
        decl   (declivity-at gamma)
        ha     (daylight-half-angle lat decl)]
```

```
    (+
     (- 12.0 (min->hr eqtime))
     (-> (- lng) (+ ha)
         (longitudinal-angle->duration)
         (min->hr)))))

(test/deftest sunrise-sunset-reality-checks
  (test/is
   (<
    (sunrise-hour 0.3 0.0 Math/PI)
    (sunrise-hour 0.1 0.0 Math/PI))
   "In Northern summer, the sun rises earlier at higher latitudes.")
  (test/is
   (>
    (sunset-hour 0.3 0.0 Math/PI)
    (sunset-hour 0.1 0.0 Math/PI))
   "... and sets later.")
  (test/is
   (>
    (sunrise-hour -0.3 0.0 Math/PI)
    (sunrise-hour -0.1 0.0 Math/PI))
   "The situation is reversed in Southern Winter.")
  (test/is
   (>
    (sunrise-hour 0.3 0.0 0.0)
    (sunrise-hour 0.1 0.0 0.0))
   "... and Northern Winter.")
  (test/is
   (every? true?
           (for [lat   [-0.4 -0.2 0.0 0.1 0.3]
                 gamma [0.0 1.0 2.0 3.0 5.0 6.0]]
             (<
              (sunrise-hour lat 0.5 gamma)
              (sunrise-hour lat -0.5 gamma))))
   "The sun rises in the East before in the West.")
  (test/is
   (every? true?
           (for [lat   [-0.4 -0.2 0.0 0.1 0.3]
                 lng   [-3.0 -2.0 -1.0 0.0 1.0 2.0 3.0]
                 gamma [0.0 1.0 2.0 3.0 5.0 6.0]]
             (<
              (sunrise-hour lat lng gamma)
              (sunset-hour lat lng gamma))))
   "Sunrise is always before sunset.")
  (test/is
   (=
    (->> (range 12)
         (map (fn [mnth]
                (-> mnth (/ 12.0) (* 2.0 Math/PI))))
         (mapv (fn [gamma]
                 (-
                  (sunset-hour 0.0 0.0 gamma)
                  (sunrise-hour 0.0 0.0 gamma)))))
    [12.120711690930627
     12.116455880448317
     12.111944801470845
     12.111445309127586
     12.115166383631657
     12.119873606728143
     12.120728143894391
     12.116778690773202
     12.11228595353197
     12.11123549802302
     12.11458941491778
     12.119607952649702])
   "At the Equator, pretty much 12 hours of sunlight per day, all year long.")
  (test/is
   (=
```

```
    (->> (range 12)
         (map (fn [mnth]
                (-> mnth (/ 12.0) (* 2.0 Math/PI))))
         (mapv (fn [gamma]
                 (-
                  (sunset-hour 0.8 0.5 gamma)
                  (sunrise-hour 0.8 0.5 gamma)))))
    [8.725544159704336
     9.650457702575103
     11.16798773916028
     12.811517288872603
     14.359379610955962
     15.485521075449928
     15.66341255886112
     14.783619423056567
     13.3366494848639
     11.72573857583448
     10.149741212187232
     8.947654067059547])
   "The situation is clearly more varied at high latitudes."))

;;=============================================================================
;; Gridfire config transformation
;;=============================================================================

(defn instant-ms
  ^long [t]
  (comment "At the time of writing," clojure.core/inst-ms "is not implemented in Babashka :/")
  (cond
    (instance? Instant t)        (* 1000 (.getEpochSecond ^Instant t))
    (instance? java.util.Date t) (.getTime ^java.util.Date t)))

;; NOTE this function is approximate,
;; and also slightly inconsistent with the original Fortran code,
;; (which is also quite approximate,)
;; and that's fine.
;; The sensitivity on gamma is quite low,
;; so we can totally afford this sort of inaccuracy;
;; in fact, we can afford several days (degrees) of inaccuracy on gamma,
;; given how this script is used.
(defn gamma-at-instant
  "[rad] Converts an instant to a time-of-year angle."
  ^double [t]
  {:pre [(inst? t)]}
  (let [dt            (-> (instant-ms t)
                          (Instant/ofEpochMilli)
                          (.atZone ZoneOffset/UTC))
        y-floor       (->
                       (ZonedDateTime/of
                        (.getYear dt)
                        (int 1) (int 1)                    ; January 1st
                        (int 0) (int 0) (int 0)
                        (int 0)
                        ZoneOffset/UTC)
                       (.toInstant))
        y-ceil       (->
                      (ZonedDateTime/of
                       (-> (.getYear dt) (inc) (int))
                       (int 1) (int 1)                     ; January 1st
                       (int 0) (int 0) (int 0)
                       (int 0)
                       ZoneOffset/UTC)
                      (.toInstant))
        year-duration (-
                       (instant-ms y-ceil)
                       (instant-ms y-floor))]
    (->
      (instant-ms t)
```

```
      (double)
      (- (instant-ms y-floor))
      (/ year-duration)
      (* 2.0 Math/PI))))

(test/deftest gamma-at-instant-examples
  (test/is
   (= 0.0
      (/ (gamma-at-instant #inst "2022-01-01") (* 2.0 Math/PI))))
  (test/is
   (= 0.21643835616438353
      (/ (gamma-at-instant #inst "2022-03-21") (* 2.0 Math/PI))))
  (test/is
   (= 0.49589041095890407
      (/ (gamma-at-instant #inst "2022-07-01") (* 2.0 Math/PI))))
  (test/is
   (= 0.7753424657534247
      (/ (gamma-at-instant #inst "2022-10-11") (* 2.0 Math/PI)))))

(defn format-fractional-hour
  [^double h]
  (format "%02d:%02d"
          (-> h (Math/floor) (long) (mod 24))
          (-> h (rem 1.0) (* 60.0) (double) (Math/round) (long))))

(test/deftest format-fractional-hour-examples
  (test/is (= "09:12" (format-fractional-hour 9.2)))
  (test/is (= "19:45" (format-fractional-hour 19.75)))
  (test/is (= "00:00" (format-fractional-hour 0.0)))
  (test/is (= "23:59" (format-fractional-hour 23.99)))
  (test/is (= "02:30" (format-fractional-hour 26.5))))

(defn infer-burn-period
  "Resolves the :burn-period map,
  given a `bp-info` map which may contain the same keys
  as a regular GridFire :burn-period,
  and some more information specific to this script."
  [{ign-t     :ignition-start-timestamp
    bp-length :burn-period-length
    bp-frac   :burn-period-frac
    lat-deg   ::lat-deg
    lng-deg   ::lng-deg
    :or       {bp-frac 0.5}
    :as       _bp-info}]
  (let [gamma           (gamma-at-instant ign-t)
        lat             (deg->rad lat-deg)
        lng             (deg->rad lng-deg)
        h-sunrise       (sunrise-hour lat lng gamma)
        h-sunset        (sunset-hour lat lng gamma)
        [h-start h-end] (if (nil? bp-length)
                          [h-sunrise h-sunset]
                          (let [bp-half-length (* bp-length 0.5)
                                h-center       (+
                                                (* (- 1.0 bp-frac) h-sunrise)
                                                (* bp-frac h-sunset))]
                            [(-> h-center (- bp-half-length))
                             (-> h-center (+ bp-half-length))]))]
    {:burn-period-start (format-fractional-hour h-start)
     :burn-period-end   (format-fractional-hour h-end)}))

(test/deftest infer-burn-period-example
  (test/is
   (=
    ;; Almost correct, Météo France says 6:15 / 21:14.
    {:burn-period-start "04:15",
     :burn-period-end   "19:13"}
    (infer-burn-period
     {:ignition-start-timestamp #inst"2022-07-19T16:54:09.073-00:00"
```

```
      ::lat-deg                43.17
      ::lng-deg                5.60}))
   "correct sunrise/sunset hours for La Ciotat, France on 2022-07-19.")

  (test/is
   (=
    {:burn-period-start "08:14",
     :burn-period-end   "18:14"}
    (infer-burn-period
     {:ignition-start-timestamp #inst"2022-07-19T16:54:09.073-00:00"
      ::lat-deg                43.17
      ::lng-deg                5.60
      :burn-period-length      10.0
      ;; Centering on 15:14 :
      :burn-period-frac        (/ (- 15.0 6.0) (- 21.0 6.0))}))
   (str "if provided, correctly uses " (pr-str :burn-period-length) " and " (pr-str :burn-period-frac) ".")))

#_(test/run-tests)
```

9.2.15 Ad hoc Data Structures

1. gridfire.structs.rfwo

```
(ns gridfire.structs.rfwo
  "Custom data structure for memoizing rothermel-fast-wrapper-optimal,
  isolated in this namespace to wrap JVM interop."
  (:import (clojure.lang IHashEq)
           (org.apache.commons.codec.digest MurmurHash3)))

;; NOTE it's good practice in Clojure to isolate custom JVM datatypes in their own small namespace;
;; doing so tends to prevent subtle issues when reloading the namespace during interactive development.
;; What matters is for the namespace's file to not be edited often.
(deftype RothFWOArgs
  ;; Custom datatype representing the argument to rothermel-fast-wrapper-optimal,
  ;; yielding much more efficient memoization.
  [^double fuel-model-number
   ^double fuel-moisture-dead-1hr
   ^double fuel-moisture-dead-10hr
   ^double fuel-moisture-dead-100hr
   ^double fuel-moisture-dead-herbaceous
   ^double fuel-moisture-live-herbaceous
   ^double fuel-moisture-live-woody
   ^boolean grass-suppression?]
  Object
  (equals [_this rothFWOArgs2]
    (let [^RothFWOArgs rothFWOArgs2 rothFWOArgs2]
      (and (= fuel-model-number             (.-fuel-model-number rothFWOArgs2))
           (= fuel-moisture-dead-1hr        (.-fuel-moisture-dead-1hr rothFWOArgs2))
           (= fuel-moisture-dead-10hr       (.-fuel-moisture-dead-10hr rothFWOArgs2))
           (= fuel-moisture-dead-100hr      (.-fuel-moisture-dead-100hr rothFWOArgs2))
           (= fuel-moisture-dead-herbaceous (.-fuel-moisture-dead-herbaceous rothFWOArgs2))
           (= fuel-moisture-live-herbaceous (.-fuel-moisture-live-herbaceous rothFWOArgs2))
           (= fuel-moisture-live-woody      (.-fuel-moisture-live-woody rothFWOArgs2))
           (= grass-suppression?            (.-grass-suppression? rothFWOArgs2)))))
  (hashCode [this] (.hasheq ^IHashEq this))
  IHashEq
  (hasheq [_this]
    (-> (MurmurHash3/hash32 (Double/doubleToRawLongBits fuel-model-number))
        (long) (MurmurHash3/hash32 (Double/doubleToRawLongBits fuel-moisture-dead-1hr))
        (long) (MurmurHash3/hash32 (Double/doubleToRawLongBits fuel-moisture-dead-10hr))
        (long) (MurmurHash3/hash32 (Double/doubleToRawLongBits fuel-moisture-dead-100hr))
        (long) (MurmurHash3/hash32 (Double/doubleToRawLongBits fuel-moisture-dead-herbaceous))
        (long) (MurmurHash3/hash32 (Double/doubleToRawLongBits fuel-moisture-live-herbaceous))
        (long) (MurmurHash3/hash32 (Double/doubleToRawLongBits fuel-moisture-live-woody))
        (long) (MurmurHash3/hash32 (long (if grass-suppression? 1 0))))))
```

```
;;-----------------------------------------------------------------------------
;; JVM interop wrappers.
;;-----------------------------------------------------------------------------

(defmacro make-RothFWOArgs
  "Fast constructor for RothFWOArgs, wrapping JVM interop."
  [& args]
  `(RothFWOArgs. ~@args))

;; Fast getters:

(defn get-fuel-model-number
  ^double [^RothFWOArgs rothFWOArgs]
  (.-fuel-model-number rothFWOArgs))

(defn get-fuel-moisture-vec
  [^RothFWOArgs rothFWOArgs]
  [(.-fuel-moisture-dead-1hr rothFWOArgs)
   (.-fuel-moisture-dead-10hr rothFWOArgs)
   (.-fuel-moisture-dead-100hr rothFWOArgs)
   (.-fuel-moisture-dead-herbaceous rothFWOArgs)
   (.-fuel-moisture-live-herbaceous rothFWOArgs)
   (.-fuel-moisture-live-woody rothFWOArgs)])

(defn get-grass-suppression?
  [^RothFWOArgs rothFWOArgs]
  (.-grass-suppression? rothFWOArgs))
```

2. gridfire.structs.tensor-meta

```
(ns gridfire.structs.tensor-meta)

(defprotocol IDoubleGetter
  (double-getter [this]))

;; A special-purpose record type meant to be used as a fast-reads metadata map on Tensor objects.
(defrecord GettersMeta
  [grid-lookup-double-getter]
  IDoubleGetter
  (double-getter [_this] grid-lookup-double-getter))
```

9.3 Inputs Resolution

9.3.1 gridfire.fetch.base

```
(ns gridfire.fetch.base
  (:require [gridfire.grid-lookup :as grid-lookup]
            [tech.v3.datatype     :as d]))

(defmulti get-wrapped-tensor-multi (fn [_context-map layer-spec _convert-fn _target-dtype] (:type layer-spec)))

(defn get-wrapped-tensor
  "Fetches a layer into a tensor wrapped in a map with geospatial metadata."
  [context-map layer-spec convert-fn target-dtype]
  {:pre [(map? context-map)
         (:type layer-spec)
         (or (nil? target-dtype)
             (keyword? target-dtype))]}
  (-> (get-wrapped-tensor-multi context-map layer-spec convert-fn target-dtype)
      ;; NOTE at the time of writing, the tensors returned by Magellan are not backed by a flat JVM array,
      ;; being assembled via d/concat:
      ;; https://github.com/sig-gis/magellan/blob/8d72d0484b15c0d26a073684d07d405dae5c715d/src/magellan/raster/inspect.clj#L145
      (update :matrix grid-lookup/ensure-flat-jvm-tensor)))
```

```
(defn convert-tensor-as-requested
  "General utility function for applying conversions and type coercions to tensors,
  useful when the data provider cannot do it directly."
  [layer-map convert-fn target-dtype]
  (let [old-tensor        (:matrix layer-map)
        old-dtype         (d/elemwise-datatype old-tensor)
        needs-converting? (or (some? convert-fn)
                              (and (some? target-dtype)
                                   (not= target-dtype old-dtype)))]
    (if needs-converting?
      (let [converted  (d/emap (or convert-fn identity)
                               target-dtype
                               old-tensor)
            new-tensor (if (= old-dtype target-dtype)
                         (do
                           ;; converts in-place when possible, as it's more efficient.
                           (d/copy! converted old-tensor)
                           ;; Redundant since the above (d/copy! ...) already returns old-tensor,
                           ;; but I'd rather be very explicit about mutation. (Val, 04 Nov 2022)
                           old-tensor)
                         (d/clone converted))]
        (assoc layer-map :matrix new-tensor))
      layer-map)))
```

9.3.2 gridfire.fetch.grid-of-rasters

```
(ns gridfire.fetch.grid-of-rasters
  (:require [clojure.spec.alpha :as s]
            [tech.v3.datatype   :as d]
            [tech.v3.tensor     :as t]))

(defn- all-grid-elements
  [grid]
  (if (vector? grid)
    (mapcat all-grid-elements grid)
    [grid]))

(defn- ij-shape
  [t]
  (->> t (d/shape) (take-last 2) (vec)))

(defn- non-ij-shape
  [t]
  (->> t (d/shape) (drop-last 2)))

(defn- has-consistent-xy-dims?
  [layer]
  (= [(:height layer) (:width layer)]
     (ij-shape (:matrix layer))))

(s/def :gridfire.spec.raster-map/has-consistent-xy-dims? has-consistent-xy-dims?)

(defn map-grid2d
  "Like clojure.core/mapv, but for matrix-shaped data: [[e ...] ...] -> [[(f e) ...] ...]."
  [f grid2d]
  (vec (for [m-i grid2d]
         (vec (for [m-ij m-i]
                (f m-ij))))))

(defn- almost-equal?
  "Like =, but leaves some room for floating-point inaccuracy."
  [^double v1 ^double v2 ^double eps]
  (< (Math/abs (- v1 v2))
     eps))

(defn- is-2D-grid?
```

```
  [fetched-rasters-grid]
  (and
   (vector? fetched-rasters-grid)
   (not-empty fetched-rasters-grid)
   (every? vector? fetched-rasters-grid)
   (every? not-empty fetched-rasters-grid)
   (apply = (map count fetched-rasters-grid))))

(s/def ::is-2D-grid? is-2D-grid?)

(defn- is-non-empty?
  [fetched-rasters-grid]
  (and
   (not-empty fetched-rasters-grid)
   (every? not-empty fetched-rasters-grid)))

(s/def ::is-non-empty? is-non-empty?)

(defn- n-y-rows
  [fetched-rasters-grid]
  (count fetched-rasters-grid))

(defn- n-x-cols
  [fetched-rasters-grid]
  (count (first fetched-rasters-grid)))

(defn- ith-grid-row
  [fetched-rasters-grid i]
  (nth fetched-rasters-grid i))

(defn- jth-grid-col
  [fetched-rasters-grid j]
  (->> fetched-rasters-grid
       (mapv (fn [row] (nth row j)))))

(defn- height-is-constant-in-each-row?
  [fetched-rasters-grid]
  (->> (range (n-y-rows fetched-rasters-grid))
       (map #(ith-grid-row fetched-rasters-grid %))
       (every? (fn compatible-heights? [ith-row]
                 (->> ith-row
                      (map :height)
                      (apply =))))))

(s/def ::height-is-constant-in-each-row? height-is-constant-in-each-row?)

(defn- width-is-constant-in-each-col?
  [fetched-rasters-grid]
  (->> (range (n-x-cols fetched-rasters-grid))
       (map #(jth-grid-col fetched-rasters-grid %))
       (every? (fn compatible-heights? [jth-col]
                 (->> jth-col
                      (map :width)
                      (apply =))))))

(s/def ::width-is-constant-in-each-col? width-is-constant-in-each-col?)

(defn- layers-have-the-same?
  [f fetched-rasters-grid]
  (->> (all-grid-elements fetched-rasters-grid) (map f) (apply =)))

(defn- all-blocks-have-the-same-scale?
  [fetched-rasters-grid]
  (and (layers-have-the-same? :scalex fetched-rasters-grid)
       (layers-have-the-same? :scaley fetched-rasters-grid)))

(s/def ::all-blocks-have-the-same-scale? all-blocks-have-the-same-scale?)
```

```
(defn- meet-at-corners?
  [fetched-rasters-grid]
  (and
   (->> (ith-grid-row fetched-rasters-grid 0)
        (map (juxt :upperleftx :width :scalex))
        (partition 2 1)
        (every? (fn x-touch? [[[ulx-j w-j sx-j] [ulx-j+1 _w-j+1 _sx-j+1]]]
                  (almost-equal? (+ (double ulx-j)
                                    (* (double w-j) (double sx-j)))
                                 (double ulx-j+1)
                                 (Math/abs (* 1e-4 (double sx-j)))))))
   (->> (jth-grid-col fetched-rasters-grid 0)
        (map (juxt :upperlefty :height :scaley))
        (partition 2 1)
        (every? (fn y-touch? [[[uly-i h-i sy-i] [uly-i+1 _h-i+1 _sy-i+1]]]
                  (almost-equal? (+ (double uly-i)
                                    (* (double h-i) (double sy-i)))
                                 (double uly-i+1)
                                 (Math/abs (* 1e-4 (double sy-i)))))))))

(s/def ::meet-at-corners? meet-at-corners?)

(s/def ::same-bands (fn same-bands? [fetched-rasters-grid]
                      (->> fetched-rasters-grid (layers-have-the-same? #(-> % :matrix (non-ij-shape))))))

(s/def ::stitchable-grid-of-layers
  (s/and ::is-2D-grid?
         ::is-non-empty?
         (s/coll-of (s/coll-of :gridfire.spec.raster-map/has-consistent-xy-dims?))
         ::height-is-constant-in-each-row?
         ::width-is-constant-in-each-col?
         ::all-blocks-have-the-same-scale?
         ::meet-at-corners?
         ::same-bands))

(defn stitch-tensors-in-grid2d
  "Computes a tensor by stitching a 2D grid of blocks,
  which must be tensors of compatible shapes.
  The block in the grid may have more than 2 dimensions;
  the stitching happens on the last 2 dimensions,
  and on the preceding dimensions all blocks must have identical shape
  (in the case of GIS rasters, that means 'same number of bands')."
  [tensors-grid]
  ;; NOTE this algorithm is deliberately not expressing GIS-specific concepts
  ;; such as width/height or x/y. It is thus made simpler, easier to reason about.
  (let [i-lengths    (->> tensors-grid
                          (map first)
                          (mapv (fn [t-i0] (-> t-i0 (ij-shape) (nth 0)))))
        i-length     (long (reduce + i-lengths))
        j-lengths    (->> tensors-grid
                          (first)
                          (mapv (fn [t-0j] (-> t-0j (ij-shape) (nth 1)))))
        j-length     (long (reduce + j-lengths))
        t00          (get-in tensors-grid [0 0])
        dst-shape    (vec (concat (drop-last 2 (d/shape t00))
                                  [i-length j-length]))
        dst-tensor   (t/new-tensor dst-shape :datatype (d/elemwise-datatype t00))
        dst-i-ranges (->> i-lengths
                          (reductions + 0)
                          (partition 2 1)
                          (vec))
        dst-j-ranges (->> j-lengths
                          (reductions + 0)
                          (partition 2 1)
                          (vec))]
    (->> (for [grid-i (range (count i-lengths))
               grid-j (range (count j-lengths))]
           (let [src-tensor    (get-in tensors-grid [grid-i grid-j])
```

```
                [imin imax+1] (nth dst-i-ranges grid-i)
                [jmin jmax+1] (nth dst-j-ranges grid-j)
                dst-selection (concat (repeat (- (count dst-shape) 2)
                                              :all)
                                      [(range imin imax+1)
                                       (range jmin jmax+1)])
                dst-subtensor (apply t/select dst-tensor dst-selection)]
            [src-tensor dst-subtensor]))
        (run! (fn copy-to-subrect [[src-tensor dst-subtensor]]
                (t/tensor-copy! src-tensor dst-subtensor))))
    dst-tensor))

(defn stitch-grid-of-layers
  [fetched-rasters-grid]
  (when-some [expl-data (s/explain-data ::stitchable-grid-of-layers fetched-rasters-grid)]
    (throw (ex-info (format "Invalid grid of rasters: %s"
                            (s/explain-str ::stitchable-grid-of-layers fetched-rasters-grid))
                    {::fetched-rasters-grid fetched-rasters-grid
                     ::explanation          expl-data})))
  (let [layer00      (get-in fetched-rasters-grid [0 0])
        tensors-grid (->> fetched-rasters-grid (map-grid2d :matrix))
        dst-tensor   (stitch-tensors-in-grid2d tensors-grid)
        [h w]        (->> dst-tensor (d/shape) (take-last 2) (vec))]
    (merge {:srid       (->> (all-grid-elements fetched-rasters-grid)
                             (keep :srid)
                             (first))
            ;; NOTE I'm not entirely sure that aggregating with the min and max is always correct, (Val, 24 Oct 2022)
            ;; but it is at least consistent with the current implementation of
            ;; gridfire.magellan-bridge/geotiff-raster-to-tensor, which uses the x-min and y-max of the envelope.
            ;; Kenneth Cheung confirmed that it is correct. (Val, 07 Nov 2022)
            :upperleftx (->> fetched-rasters-grid
                             (map first)
                             (map :upperleftx)
                             (reduce min))
            :upperlefty (->> fetched-rasters-grid
                             (first)
                             (map :upperlefty)
                             (reduce max))
            :width      w
            :height     h
            :matrix     dst-tensor}
           (select-keys layer00 [:scalex
                                 :scaley
                                 :skewx
                                 :skewy
                                 :numbands]))))
```

9.3.3 gridfire.fetch.grid-of-rasters-test

```
(ns gridfire.fetch.grid-of-rasters-test
  (:require [clojure.test                  :refer [deftest is testing]]
            [clojure.java.io               :as io]
            [gridfire.fetch]               ; :require'd to provide multimethod implementation. :as omitted to pass linting.
            [gridfire.fetch.base           :refer [get-wrapped-tensor]]
            [gridfire.inputs.envi-bsq      :refer [get-gdal-info]]
            [tech.v3.datatype              :as d]
            [tech.v3.datatype.functional   :as dfn]))

(def examples-dir-path "test/gridfire/resources/grid-of-rasters-examples")

(defn- compute-grid2d
  [[i-length j-length] ij-fn]
  (vec (for [grid-i (range i-length)]
         (vec (for [grid-j (range j-length)]
                (ij-fn grid-i grid-j))))))
```

```
(defn example-subraster-file-path
  [fname xj yi fext]
  (.getPath (io/file examples-dir-path
                     (str fname "_x" xj "_y" yi "." fext))))

(defn- create-subrasters-shell-cmds
  [fname split-ys-px split-xs-px]
  (let [x-windows (->> split-xs-px
                       (partition 2 1)
                       (mapv vec))
        y-windows (->> split-ys-px
                       (partition 2 1)
                       ;; Reversing is consistent with Elmfire's naming logic for gridded inputs:
                       ;; the y coordinates must be increasing with the file's y number.
                       (reverse)
                       (mapv vec))]
    (for [xj             (range (count x-windows))
          :let           [[xmin xmax+1] (nth x-windows xj)]
          yi             (range (count y-windows))
          :let           [[ymin ymax+1] (nth y-windows yi)]
          [fext gt-opts] [["bsq" ["-of" "ENVI" "-co" "INTERLEAVE=BSQ"]]
                          ["tif" nil]]]
      (->> (concat ["gdal_translate"]
                   ["-srcwin" xmin ymin (- xmax+1 xmin) (- ymax+1 ymin)]
                   gt-opts
                   [(.getPath (io/file examples-dir-path (str fname ".tif")))
                    (example-subraster-file-path fname xj yi fext)])
           (map str)))))

(defn example-grid-layer-spec
  [fname]
  {:type         :grid-of-rasters
   :rasters-grid (compute-grid2d [3 2]
                                 (fn [^long grid-i ^long grid-j]
                                   (let [xj   grid-j
                                         yi   (- (dec 3) grid-i)
                                         fext (nth ["tif" "bsq"] (-> (+ yi xj) (mod 2)))]
                                     {:type   (case fext
                                                "tif" :geotiff
                                                "bsq" :gridfire-envi-bsq)
                                      :source (example-subraster-file-path fname xj yi fext)})))})

(deftest ^:unit example-file-names-test
  (testing "Our test files are gridded as follows:"
    (is (= {:type         :grid-of-rasters,
            :rasters-grid [[{:type   :geotiff
                             :source "test/gridfire/resources/grid-of-rasters-examples/fbfm40_x0_y2.tif"}
                            {:type   :gridfire-envi-bsq
                             :source "test/gridfire/resources/grid-of-rasters-examples/fbfm40_x1_y2.bsq"}]
                           [{:type   :gridfire-envi-bsq
                             :source "test/gridfire/resources/grid-of-rasters-examples/fbfm40_x0_y1.bsq"}
                            {:type   :geotiff
                             :source "test/gridfire/resources/grid-of-rasters-examples/fbfm40_x1_y1.tif"}]
                           [{:type   :geotiff
                             :source "test/gridfire/resources/grid-of-rasters-examples/fbfm40_x0_y0.tif"}
                            {:type   :gridfire-envi-bsq
                             :source "test/gridfire/resources/grid-of-rasters-examples/fbfm40_x1_y0.bsq"}]]}
           (example-grid-layer-spec "fbfm40"))))
  (testing "And these files all exist:"
    (->> (example-grid-layer-spec "fbfm40")
         :rasters-grid
         (sequence cat)
         (every? (fn [layer-spec]
                   (is (.exists (io/file (:source layer-spec)))))))))

(comment

  ;; How the test files were created: (Val, 25 Oct 2022)
```

```
  (require '[clojure.java.shell :as sh])
  (.mkdirs (io/file examples-dir-path))
  ;; Getting 2 original rasters
  (sh/sh "gdal_translate"
         (str "test/gridfire/resources/envi-bsq-examples/" "fbfm40.tif")
         (str examples-dir-path "/" "fbfm40.tif"))
  (sh/sh "gdal_translate"
         "-b" "1" "-b" "2" "-b" "3"                          ; only 3 bands
         (str "test/gridfire/resources/envi-bsq-examples/" "ws_2018.tif")
         (str examples-dir-path "/" "ws_2018.tif"))
  (->> [(create-subrasters-shell-cmds "fbfm40" [0 42 103 120] [0 27 80])
        (create-subrasters-shell-cmds "ws_2018" [0 1 2 3] [0 1 2])]
       (sequence cat)
       (mapv (fn [sh-args] (apply sh/sh sh-args))))

  *e)

(defn- assert-equivalent-layers
  [l1 l2]
  (let [relevant-gis-ks [:upperleftx
                         :upperlefty
                         :width
                         :height
                         :scalex
                         :scaley]]
    (and (is (= (select-keys l1 relevant-gis-ks)
                (select-keys l2 relevant-gis-ks))
             "Same geospatial metadata.")
         (is (= (d/shape (:matrix l1))
                (d/shape (:matrix l2)))
             "Same tensor dimensions.")
         (is (dfn/equals (:matrix l1) (:matrix l2))
             "Same tensor contents."))))

(deftest ^:simulation fetch-grid-of-rasters-test
  (testing ":grid-of-rasters: Stitching the :rasters-grid yields the same layer as the original."
    (->> ["fbfm40"
          "ws_2018"]
         (every? (fn [fname]
                   (let [layer1 (get-wrapped-tensor {}
                                                    (example-grid-layer-spec fname)
                                                    nil
                                                    nil)
                         layer0 (get-wrapped-tensor {}
                                                    {:type   :geotiff
                                                     :source (str examples-dir-path "/" fname ".tif")}
                                                    nil
                                                    nil)]
                     (assert-equivalent-layers layer0 layer1)))))))

(defn are-grid-corner-coords-as-we-expect
  [rasters-grid]
  (letfn [(gdal-corner-coords [input]
            (-> input :source
                (get-gdal-info)
                (get "cornerCoordinates")))
          (corner-x [c] (nth c 0))
          (corner-y [c] (nth c 1))
          (is-lower-i=upper-i+1 [[cci cci+1]]
            ;; Putting the (is ...) assertion inside this function
            ;; yields finer error reporting than around the (every? ...).
            (is (= (-> cci (get "lowerRight") (corner-y))
                   (-> cci+1 (get "upperLeft") (corner-y)))))
          (is-right-j=left-j+1 [[ccj ccj+1]]
            (is (= (-> ccj (get "lowerRight") corner-x)
                   (-> ccj+1 (get "upperLeft") corner-x))))]
    (testing "Reminders about GDAL's cornerCoordinates: in coordinates [x y], "
      (let [cc {"upperLeft"  [-2028825.0 1953435.0],
```

```
                "lowerLeft"  [-2028825.0 1952175.0],
                "lowerRight" [-2028015.0 1952175.0],
                "upperRight" [-2028015.0 1953435.0],
                "center"     [-2028420.0 1952805.0]}]
        (testing "x corresponds to left/right."
          (is (= (-> cc (get "lowerLeft") (corner-x))
                 (-> cc (get "upperLeft") (corner-x))))
          (is (= (-> cc (get "lowerRight") (corner-x))
                 (-> cc (get "upperRight") (corner-x)))))
        (testing "y corresponds to up/down."
          (is (= (-> cc (get "lowerLeft") (corner-y))
                 (-> cc (get "lowerRight") (corner-y))))
          (is (= (-> cc (get "upperLeft") (corner-y))
                 (-> cc (get "upperRight") (corner-y)))))))
    (testing "Increasing i advances from upper to lower."
      (let [j=0-col (mapv first rasters-grid)]
        (->> j=0-col
             (pmap gdal-corner-coords)
             (partition 2 1)
             (every? is-lower-i=upper-i+1))))
    (testing "Increasing j advances from left to right."
      (let [i=0-row (first rasters-grid)]
        (->> i=0-row
             (pmap gdal-corner-coords)
             (partition 2 1)
             (every? is-right-j=left-j+1))))))

(deftest ^:simulation gdal-corner-coordinates
  (testing "How our test grids touch."
    (are-grid-corner-coords-as-we-expect (:rasters-grid (example-grid-layer-spec "fbfm40")))))
```

9.3.4 gridfire.postgis-bridge-test

```
(ns gridfire.postgis-bridge-test
  (:require [clojure.test            :refer [deftest is]]
            [gridfire.postgis-bridge :refer [postgis-raster-to-matrix]]))

(def db-spec {:classname   "org.postgresql.Driver"
              :subprotocol "postgresql"
              :subname     "//localhost:5432/gridfire_test"
              :user        "gridfire_test"
              :password    "gridfire_test"})

(deftest ^:database single-band-test
  (let [raster (postgis-raster-to-matrix db-spec "landfire.fbfm40 WHERE rid=1")]

    (is (some? raster))

    (is (= 1 (:numbands raster)))))

(deftest ^:database multi-band-test
  (let [raster (postgis-raster-to-matrix db-spec "weather.ws WHERE rid=1")]

    (is (some? raster))

    (is (= 73 (:numbands raster)))))

(deftest ^:database multi-band-rescale-test
  (let [raster (postgis-raster-to-matrix db-spec "weather.ws WHERE rid=1" 600)]

    (is (some? raster))

    (is (= 73 (:numbands raster)))))

(deftest ^:database multi-band-rescale-threshold-test
  (let [raster (postgis-raster-to-matrix db-spec "weather.ws WHERE rid=1" 600 10)]
```

```
    (is (some? raster))

    (is (= 73 (:numbands raster)))))
```

9.3.5 gridfire.inputs

```
(ns gridfire.inputs
  (:require [clojure.data.csv        :as csv]
            [clojure.java.io         :as io]
            [clojure.string          :as str]
            [gridfire.burn-period    :as burnp]
            [gridfire.common         :refer [burnable-fuel-model?]]
            [gridfire.conversion     :refer [conversion-table min->ms ms->min percent->dec]]
            [gridfire.fetch          :as fetch]
            [gridfire.postgis-bridge :refer [with-db-connection-pool]]
            [gridfire.utils.random   :refer [draw-samples my-shuffle my-rand-nth]]
            [gridfire.utils.server   :refer [throw-message]]
            [tech.v3.tensor          :as t])
  (:import java.util.Date
           java.util.Random))

(set! *unchecked-math* :warn-on-boxed)

(defn add-input-layers
  [config]
  (with-db-connection-pool (:db-spec config)
    (let [aspect-layer                          (future (fetch/landfire-layer config :aspect))
          canopy-base-height-layer              (future (fetch/landfire-layer config :canopy-base-height))
          canopy-cover-layer                    (future (fetch/landfire-layer config :canopy-cover))
          canopy-height-layer                   (future (fetch/landfire-layer config :canopy-height))
          crown-bulk-density-layer              (future (fetch/landfire-layer config :crown-bulk-density))
          elevation-layer                       (future (fetch/landfire-layer config :elevation))
          fuel-model-layer                      (future (fetch/landfire-layer config :fuel-model)) ; Use its envelope
          slope-layer                           (future (fetch/landfire-layer config :slope))
          ignition-layer                        (future (fetch/ignition-layer config))
          ignition-mask-layer                   (future (fetch/ignition-mask-layer config))
          temperature-layer                     (future (fetch/weather-layer config :temperature))
          relative-humidity-layer               (future (fetch/weather-layer config :relative-humidity))
          wind-speed-20ft-layer                 (future (fetch/weather-layer config :wind-speed-20ft))
          wind-from-direction-layer             (future (fetch/weather-layer config :wind-from-direction))
          fuel-moisture-dead-1hr-layer          (future (fetch/fuel-moisture-layer config :dead :1hr))
          fuel-moisture-dead-10hr-layer         (future (fetch/fuel-moisture-layer config :dead :10hr))
          fuel-moisture-dead-100hr-layer        (future (fetch/fuel-moisture-layer config :dead :100hr))
          fuel-moisture-live-herbaceous-layer (future (fetch/fuel-moisture-layer config :live :herbaceous))
          fuel-moisture-live-woody-layer        (future (fetch/fuel-moisture-layer config :live :woody))
          sdi-layer                             (future (fetch/sdi-layer config))]
      (assoc config
             :envelope                              (fetch/layer->envelope @fuel-model-layer (:srid config))
             :aspect-matrix                         (:matrix @aspect-layer)
             :canopy-base-height-matrix             (:matrix @canopy-base-height-layer)
             :canopy-cover-matrix                   (:matrix @canopy-cover-layer)
             :canopy-height-matrix                  (:matrix @canopy-height-layer)
             :crown-bulk-density-matrix             (:matrix @crown-bulk-density-layer)
             :elevation-matrix                      (:matrix @elevation-layer)
             :fuel-model-matrix                     (:matrix @fuel-model-layer)
             :slope-matrix                          (:matrix @slope-layer)
             :ignition-matrix                       (:matrix @ignition-layer)
             :ignition-mask-matrix                  (:matrix @ignition-mask-layer)
             :temperature-matrix                    (:matrix @temperature-layer)
             :relative-humidity-matrix              (:matrix @relative-humidity-layer)
             :wind-speed-20ft-matrix                (:matrix @wind-speed-20ft-layer)
             :wind-from-direction-matrix            (:matrix @wind-from-direction-layer)
             :fuel-moisture-dead-1hr-matrix         (:matrix @fuel-moisture-dead-1hr-layer)
             :fuel-moisture-dead-10hr-matrix        (:matrix @fuel-moisture-dead-10hr-layer)
             :fuel-moisture-dead-100hr-matrix       (:matrix @fuel-moisture-dead-100hr-layer)
```

```
            :fuel-moisture-live-herbaceous-matrix (:matrix @fuel-moisture-live-herbaceous-layer)
            :fuel-moisture-live-woody-matrix      (:matrix @fuel-moisture-live-woody-layer)
            :suppression-difficulty-index-matrix  (:matrix @sdi-layer)))))

(defn- multi-band? [matrix]
  (> ^long (:n-dims (t/tensor->dimensions matrix)) 2))

;; TODO Document: using higher resolution layers than fuel model will choose upper left corner cell of the layer from the higher reso
(defn calc-multiplier
  [inputs ^long fuel-model-matrix-height matrix-kw]
  (when-let [matrix (get inputs matrix-kw)]
    (let [height-dimension    (if (multi-band? matrix) 1 0)
          ^long matrix-height (-> (t/tensor->dimensions matrix) :shape (get height-dimension))]
      (when (not= matrix-height fuel-model-matrix-height)
        (double (/ matrix-height fuel-model-matrix-height))))))

;; TODO Document fuel-model as the resolution of the computational space. Cell size must also match fuel model.
(defn add-misc-params
  [{:keys [max-runtime random-seed fuel-model-matrix] :as inputs}]
  (let [[num-rows num-cols] (:shape (t/tensor->dimensions fuel-model-matrix))]
    (assoc inputs
           :num-rows                                      (long num-rows)
           :num-cols                                      (long num-cols)
           :max-runtime                                   (double max-runtime)
           :rand-gen                                      (if random-seed (Random. random-seed) (Random.))
           :aspect-index-multiplier                       (calc-multiplier inputs num-rows :aspect-matrix)
           :canopy-base-height-index-multiplier           (calc-multiplier inputs num-rows :canopy-base-height-matrix)
           :canopy-cover-index-multiplier                 (calc-multiplier inputs num-rows :canopy-cover-matrix)
           :canopy-height-index-multiplier                (calc-multiplier inputs num-rows :canopy-height-matrix)
           :crown-bulk-density-index-multiplier           (calc-multiplier inputs num-rows :crown-bulk-density-matrix)
           :elevation-index-multiplier                    (calc-multiplier inputs num-rows :elevation-matrix)
           :slope-index-multiplier                        (calc-multiplier inputs num-rows :slope-matrix)
           :temperature-index-multiplier                  (calc-multiplier inputs num-rows :temperature-matrix)
           :relative-humidity-index-multiplier            (calc-multiplier inputs num-rows :relative-humidity-matrix)
           :wind-speed-20ft-index-multiplier              (calc-multiplier inputs num-rows :wind-speed-20ft-matrix)
           :wind-from-direction-index-multiplier          (calc-multiplier inputs num-rows :wind-from-direction-matrix)
           :fuel-moisture-dead-1hr-index-multiplier       (calc-multiplier inputs num-rows :fuel-moisture-dead-1hr-matrix)
           :fuel-moisture-dead-10hr-index-multiplier      (calc-multiplier inputs num-rows :fuel-moisture-dead-10hr-matrix)
           :fuel-moisture-dead-100hr-index-multiplier     (calc-multiplier inputs num-rows :fuel-moisture-dead-100hr-matrix)
           :fuel-moisture-live-herbaceous-index-multiplier (calc-multiplier inputs num-rows :fuel-moisture-live-herbaceous-matrix)
           :fuel-moisture-live-woody-index-multiplier     (calc-multiplier inputs num-rows :fuel-moisture-live-woody-matrix))))

(defn add-ignition-csv
  [{:keys [ignition-csv] :as inputs}]
  (if ignition-csv
    (let [ignitions (with-open [reader (io/reader ignition-csv)]
                      (doall (rest (csv/read-csv reader))))]
      (assoc inputs
             :ignition-rows        (mapv #(Long/parseLong (get % 0)) ignitions)
             :ignition-cols        (mapv #(Long/parseLong (get % 1)) ignitions)
             :ignition-start-times (mapv #(Double/parseDouble (get % 2)) ignitions)
             :max-runtime-samples  (mapv #(Double/parseDouble (get % 3)) ignitions)
             :simulations          (count ignitions)))
    inputs))

(defn add-sampled-params
  [{:keys
    [rand-gen simulations max-runtime max-runtime-samples foliar-moisture ellipse-adjustment-factor]
    :as inputs}]
  (assoc inputs
         :max-runtime-samples               (or max-runtime-samples (draw-samples rand-gen simulations max-runtime))
         :foliar-moisture-samples           (mapv percent->dec (draw-samples rand-gen simulations foliar-moisture))
         :ellipse-adjustment-factor-samples (draw-samples rand-gen simulations (or ellipse-adjustment-factor 1.0))))

(defn- convert-ranges
  [config]
  (into config
        (map (fn [[layer {:keys [units range] :as spec}]]
```

```
              (if-let [converter (get-in conversion-table [layer units])]
                [layer (assoc spec :range (mapv converter range))]
                [layer spec])))
        config))

(defn add-perturbation-params
  [{:keys [perturbations] :as inputs}]
  (if perturbations
    (assoc inputs :perturbations (convert-ranges perturbations))
    inputs))

(defn get-weather
  [{:keys [rand-gen simulations] :as inputs} weather-type]
  (let [matrix-kw (-> weather-type
                      name
                      (str "-matrix")
                      keyword)]
    (when (and (inputs weather-type)
               (not (inputs matrix-kw)))
      (draw-samples rand-gen simulations (inputs weather-type)))))

(defn add-weather-params
  [inputs]
  (assoc inputs
         :temperature-samples         (get-weather inputs :temperature)
         :relative-humidity-samples   (get-weather inputs :relative-humidity)
         :wind-speed-20ft-samples     (get-weather inputs :wind-speed-20ft)
         :wind-from-direction-samples (get-weather inputs :wind-from-direction)))

(defn get-fuel-moisture
  [{:keys [fuel-moisture simulations rand-gen] :as inputs} category size]
  (let [matrix-kw (keyword (str/join "-" ["fuel-moisture"
                                          (name category)
                                          (name size)
                                          "matrix"]))]
    (when (and (get-in fuel-moisture [category size])
               (not (inputs matrix-kw)))
      (draw-samples rand-gen simulations (get-in fuel-moisture [category size])))))

(defn add-fuel-moisture-params
  [inputs]
  (assoc inputs
         :fuel-moisture-dead-1hr-samples        (get-fuel-moisture inputs :dead :1hr)
         :fuel-moisture-dead-10hr-samples       (get-fuel-moisture inputs :dead :10hr)
         :fuel-moisture-dead-100hr-samples      (get-fuel-moisture inputs :dead :100hr)
         :fuel-moisture-live-herbaceous-samples (get-fuel-moisture inputs :live :herbaceous)
         :fuel-moisture-live-woody-samples      (get-fuel-moisture inputs :live :woody)))

(defn- filter-ignitions
  [ignition-param buffer-size limit num-items]
  (filterv
   #(<= buffer-size % limit)
   (cond
     (vector? ignition-param) (range (first ignition-param) (inc ^long (second ignition-param)))
     (list? ignition-param)   ignition-param
     (number? ignition-param) (list ignition-param)
     :else                    (range 0 num-items))))

(defn- fill-ignition-sites
  [rand-gen ignition-sites ^long simulations]
  (let [num-sites-available (count ignition-sites)]
    (loop [num-sites-needed     (- simulations num-sites-available)
           final-ignition-sites ignition-sites]
      (if (pos? num-sites-needed)
        (let [num-additional-sites (min num-sites-needed num-sites-available)
              additional-sites     (-> (my-shuffle rand-gen ignition-sites)
                                       (subvec 0 num-additional-sites))]
          (recur (- num-sites-needed num-additional-sites)
```

```
                (into final-ignition-sites additional-sites)))
        final-ignition-sites))))

(defn- seq-no-shorter-than?
  "Computes whether a (potentially infinite) sequence contains at least n elements."
  [^long n coll]
  (if (pos? n)
    (->> coll (drop (dec n)) seq some?)
    true))

(defn- all-ignitable-cells
  [ignitable-cell? ignition-rows ignition-cols]
  (for [row   ignition-rows
        col   ignition-cols
        :when (ignitable-cell? row col)]
    [row col]))

(defn- sample-ignition-sites-shuffle
  "Selects a random subset of all ignitable cells of size (up to) S := simulations
   by shuffling them and taking the first S of them.
   Requires an exhaustive scan of all cells.
   Linear complexity: performance in (N), where N := (n-rows * n-cols).
   Returns a vector of cell coordinates."
  [{:keys [rand-gen ^long simulations]} ignitable-cell? ignition-rows ignition-cols]
  (let [ignitable-sites (my-shuffle rand-gen
                                    (all-ignitable-cells ignitable-cell? ignition-rows ignition-cols))]
    (subvec ignitable-sites 0 (min simulations (count ignitable-sites)))))

(defn sample-ignition-sites-darts
  "Selects a random subset of all ignitable cells of size (up to) S := simulations
   by repeated sampling without replacement ('throwing darts').
   Efficient in the case of small S and a high enough density p of ignitable cells,
   in which case the time complexity is O(S/p);
   very slow (superlinear) if p  S/N or lower.
   Returns a vector of cell coordinates."
  [{:keys [rand-gen simulations]} ignitable-cell? ignition-rows ignition-cols]
  (let [total-cells (* (count ignition-rows) (count ignition-cols))]
    (loop [ignitable-cells   #{}
           unignitable-cells #{}]
      (if (or (= (count ignitable-cells) simulations)
              (=
                 ;; WARNING: it can be shown that it typically takes about N*ln(N) loop iterations to reach that point,
                 ;; where N := total-cells.
                 ;; Proof hint: the expected number of never-hit cells after t iterations is N*(1 - 1/N)^t.
                 ;; You'd better hope that N is not too big! (* (Math/log 1e6) 1e6) => 1.3815510557964273e7
                 (+ (count ignitable-cells) (count unignitable-cells))
                 total-cells))
        (vec ignitable-cells)
        (let [[i j :as cell] (vector (my-rand-nth rand-gen ignition-rows) (my-rand-nth rand-gen ignition-cols))]
          (if (ignitable-cell? i j)
            (recur (conj ignitable-cells cell)
                   unignitable-cells)
            (recur ignitable-cells
                   (conj unignitable-cells cell))))))))

(defn select-ignition-algorithm
  [{:keys [^long num-rows ^long num-cols]} ignitable-cell? ignition-rows ignition-cols]
  ;; NOTE Here, the selection criterion is based on proving that the density of ignitable cells (Val, 06 Jul 2022)
  ;; exceeds a certain threshold, via an (early-stopping) linear scan of ignitable cells.
  ;; Some potential enhancements:
  ;; 1) Estimate the ignitable density p not by scanning, but by fixed-size random sampling;
  ;; e.g sampling 100 cells with replacement and counting those which are ignitable.
  ;; 2) Compare to a threshold not the ignitable density p, but N*p/S,
  ;; where N = (n-rows * n-cols) and S = (:simulations inputs);
  ;; see (doc sample-ignition-sites-darts) to see why N*p/S is a relevant quantity.
  (let [ratio-threshold (max 1 (int (* 0.0025 num-rows num-cols)))] ; the inflection point from our benchmarks
    (if (->> (all-ignitable-cells ignitable-cell? ignition-rows ignition-cols)
             (seq-no-shorter-than? ratio-threshold))
```

```
      :use-darts
      :use-shuffle)))

(defn add-random-ignition-sites
  [{:keys
    [^long num-rows ^long num-cols ignition-row ignition-col simulations ^double cell-size random-ignition
     rand-gen ignition-matrix ignition-csv config-file-path ignition-mask-matrix
     fuel-model-matrix ignition-rows] :as inputs}]
  (if (or ignition-matrix ignition-csv ignition-rows)
    inputs
    (let [ignitable-cell?   (if ignition-mask-matrix
                              (fn [row col]
                                (and (pos? ^double (t/mget ignition-mask-matrix row col))
                                     (burnable-fuel-model? (t/mget fuel-model-matrix row col))))
                              (fn [row col]
                                (burnable-fuel-model? (t/mget fuel-model-matrix row col))))
          ^long buffer-size (if-let [^double edge-buffer (:edge-buffer random-ignition)]
                              (int (Math/ceil (/ edge-buffer cell-size)))
                              0)
          ignition-rows     (filter-ignitions ignition-row buffer-size (- num-rows buffer-size 1) num-rows)
          ignition-cols     (filter-ignitions ignition-col buffer-size (- num-cols buffer-size 1) num-cols)
          ignition-sites    (if (= :use-darts (select-ignition-algorithm inputs ignitable-cell? ignition-rows ignition-cols))
                              (sample-ignition-sites-darts inputs ignitable-cell? ignition-rows ignition-cols)
                              (sample-ignition-sites-shuffle inputs ignitable-cell? ignition-rows ignition-cols))]
      (if (seq ignition-sites)
        (let [ignition-sites* (fill-ignition-sites rand-gen ignition-sites simulations)]
          (assoc inputs
                 :ignition-rows (mapv first ignition-sites*)
                 :ignition-cols (mapv second ignition-sites*)))
        (throw-message (format "Invalid config file [%s]: No valid ignition sites." config-file-path))))))

(defn initialize-burn-count-matrix
  [output-burn-probability max-runtime-samples ^long num-rows ^long num-cols]
  (if (number? output-burn-probability)
    (let [num-bands (long (Math/ceil (/ ^double (reduce max max-runtime-samples) ^double output-burn-probability)))]
      (t/new-tensor [num-bands num-rows num-cols]))
    (t/new-tensor [num-rows num-cols])))

(defn add-aggregate-matrices
  [{:keys
    [max-runtime-samples num-rows num-cols output-burn-count? output-burn-probability
     output-flame-length-sum output-flame-length-max output-spot-count?] :as inputs}]
  (assoc inputs
         :burn-count-matrix       (when (or output-burn-count? output-burn-probability)
                                    (initialize-burn-count-matrix output-burn-probability max-runtime-samples num-rows num-cols))
         :flame-length-sum-matrix (when output-flame-length-sum (t/new-tensor [num-rows num-cols]))
         :flame-length-max-matrix (when output-flame-length-max (t/new-tensor [num-rows num-cols]))
         :spot-count-matrix       (when output-spot-count? (t/new-tensor [num-rows num-cols]))))

(defn add-ignition-start-times
  [{:keys [ignition-start-times ignition-start-timestamp weather-start-timestamp simulations] :as inputs}]
  (if (and (nil? ignition-start-times) ignition-start-timestamp weather-start-timestamp)
    (let [ignition-start-time (ms->min (- (double (inst-ms ignition-start-timestamp))
                                          (double (inst-ms weather-start-timestamp))))]
      (assoc inputs :ignition-start-times (vec (repeat simulations ignition-start-time))))
    inputs))

(defn add-ignition-start-timestamps
  [{:keys [ignition-start-times simulations ignition-start-timestamp weather-start-timestamp] :as inputs}]
  (let [weather-start-ms                (long (inst-ms (or weather-start-timestamp #inst "1970-01-01T00-00:00")))
        compute-ignition-start-timestamp (fn [ignition-start-time]
                                           (Date. (+ weather-start-ms (min->ms ignition-start-time))))
        ignition-start-timestamps        (cond
                                           ignition-start-timestamp (vec (repeat simulations ignition-start-timestamp))
                                           ignition-start-times     (mapv compute-ignition-start-timestamp ignition-start-times)
                                           :else                    (vec (repeat simulations #inst "1970-01-01T00-00:00")))] ; addi
    (-> inputs
        (assoc :ignition-start-timestamps ignition-start-timestamps)
```

```
        (dissoc :ignition-start-timestamp))))

(defn add-suppression
  [{:keys
    [rand-gen simulations suppression suppression-dt-samples suppression-coefficient-samples
     sdi-sensitivity-to-difficulty-samples sdi-containment-overwhelming-area-growth-rate-samples
     sdi-reference-suppression-speed-samples] :as inputs}]
  (if suppression
    (let [{:keys
           [suppression-dt
            suppression-coefficient
            sdi-sensitivity-to-difficulty
            sdi-containment-overwhelming-area-growth-rate
            sdi-reference-suppression-speed]} suppression]
      (cond-> inputs

        (and suppression-dt
             (nil? suppression-dt-samples))
        (assoc :suppression-dt-samples
               (draw-samples rand-gen simulations suppression-dt))

        (and suppression-coefficient
             (nil? suppression-coefficient-samples))
        (assoc :suppression-coefficient-samples
               (draw-samples rand-gen simulations suppression-coefficient))

        (and sdi-sensitivity-to-difficulty
             (nil? sdi-sensitivity-to-difficulty-samples))
        (assoc :sdi-sensitivity-to-difficulty-samples
               (draw-samples rand-gen simulations sdi-sensitivity-to-difficulty))

        (and sdi-containment-overwhelming-area-growth-rate
             (nil? sdi-containment-overwhelming-area-growth-rate-samples))
        (assoc :sdi-containment-overwhelming-area-growth-rate-samples
               (draw-samples rand-gen simulations sdi-containment-overwhelming-area-growth-rate))

        (and sdi-reference-suppression-speed
             (nil? sdi-reference-suppression-speed-samples))
        (assoc :sdi-reference-suppression-speed-samples
               (draw-samples rand-gen simulations sdi-reference-suppression-speed))))
    inputs))

(defn- convert-map->array-lookup
  [lookup-map]
  (let [indices      (keys lookup-map)
        size         (inc (long (apply max indices)))
        array-lookup (double-array size)]
    (doseq [index indices]
      (aset array-lookup (int index) (double (get lookup-map index))))
    array-lookup))

(defn add-fuel-number->spread-rate-adjustment-array-lookup-samples
  [{:keys [fuel-number->spread-rate-adjustment-samples] :as inputs}]
  (if fuel-number->spread-rate-adjustment-samples
    (assoc inputs
           :fuel-number->spread-rate-adjustment-array-lookup-samples
           (mapv convert-map->array-lookup fuel-number->spread-rate-adjustment-samples))
    inputs))

(defn add-burn-period-samples
  [{:keys [ignition-start-timestamps] :as inputs}]
  (let [from-sunrise-sunset? (some? (:burn-period-frac inputs))]
    (-> inputs
        (assoc :burn-period-samples
               (->> ignition-start-timestamps
                    (mapv (if from-sunrise-sunset?
                            (fn [ignition-start-timestamp]
                              (burnp/from-sunrise-sunset inputs ignition-start-timestamp))
```

```
                          (let [{:keys [start end]} inputs]
                            (constantly {:burn-period-start (or start "00:00")
                                         :burn-period-end   (or end "24:00")}))))))))))
```

9.3.6 gridfire.inputs-test

```
(ns gridfire.inputs-test
  (:require [clojure.test    :refer [are deftest is testing use-fixtures]]
            [gridfire.inputs :as inputs]
            [tech.v3.tensor  :as t])
  (:import java.util.Random))

;;-----------------------------------------------------------------------------
;; Fixtures
;;-----------------------------------------------------------------------------

(def ^:dynamic *base-config* {})

(defn- identical-matrix [width height value]
  (t/->tensor
   (into-array (repeat height (into-array (repeat width value))))))

(defn with-base-config [test-fn]
  (let [width  256
        height 256]
    (binding [*base-config* {:simulations       5
                             :cell-size         98.425
                             :num-rows          width
                             :num-cols          height
                             :rand-gen          (Random. 1234)
                             :fuel-model-matrix (identical-matrix width height 101)}]
      (test-fn))))

(use-fixtures :once with-base-config)

;;-----------------------------------------------------------------------------
;; Tests
;;-----------------------------------------------------------------------------

(deftest  ^:unit add-random-ignition-sites-test
  (let [valid-ignition-count? (fn [inputs]
                                (= (-> inputs :ignition-rows count)
                                   (-> inputs :ignition-cols count)
                                   (:simulations inputs)))]
    (testing "ignition-row and ignition-col"
      (let [add-ignition-params (fn [config ignition-row ignition-col]
                                  (assoc config :ignition-row ignition-row :ignition-col ignition-col))
            test-with           (fn [ignition-row ignition-col]
                                  (-> *base-config*
                                      (add-ignition-params ignition-row ignition-col)
                                      inputs/add-random-ignition-sites))]
        (are [ignition-row ignition-col] (valid-ignition-count? (test-with ignition-row ignition-col))
          50           50           ;scalar
          [0 50]       [0 50]       ;range
          '(1 2 3 4 5) '(1 2 3 4 5) ;list
          '(1)         '(2))))      ;list with less items than number of simulations

    (testing "just ignition-mask"
      (let [ignition-mask-matrix (identical-matrix 256 256 0.0)
            _                    (doseq [i (range 0 10)
                                         j (range 0 10)]
                                   (t/mset! ignition-mask-matrix i j 1.0))
            inputs-before        (-> *base-config*
                                     (assoc :ignition-mask-matrix  ignition-mask-matrix))
            inputs-after         (inputs/add-random-ignition-sites inputs-before)]
```

```
        (is (true? (valid-ignition-count? inputs-after)))

        (is (every? pos? (map #(t/mget (:ignition-mask-matrix inputs-after) %1 %2)
                              (:ignition-rows inputs-after) (:ignition-cols inputs-after)))
            "All ignition sites should have positive values in the ignition mask")))))

(deftest ^:unit add-suppression-test
  (let [inputs       {:suppression {:sdi-layer                                    {:type   :geotiff
                                                                                   :source "path/to/suppression/layer"}
                                    :suppression-dt                               42.0
                                    :sdi-sensitivity-to-difficulty                42.0
                                    :sdi-containment-overwhelming-area-growth-rate 42.0
                                    :sdi-reference-suppression-speed              42.0}
                     :simulations 1
                     :rand-gen    (Random. 1234)}
        inputs-after (inputs/add-suppression inputs)]

    (is (= [42.0] (inputs-after :suppression-dt-samples)))
    (is (= [42.0] (inputs-after :sdi-sensitivity-to-difficulty-samples)))
    (is (= [42.0] (inputs-after :sdi-containment-overwhelming-area-growth-rate-samples)))
    (is (= [42.0] (inputs-after :sdi-reference-suppression-speed-samples)))))

(defn- one-dimensional-double-array? [x]
  (= (Class/forName "[D")
     (class x)))

(deftest ^:unit add-spread-rate-adjustment-factors-test
  (let [inputs       (-> *base-config*
                         (merge {:fuel-number->spread-rate-adjustment-samples [{101 0.1}]}))
        inputs-after (inputs/add-fuel-number->spread-rate-adjustment-array-lookup-samples inputs)]

    (is (contains? inputs-after :fuel-number->spread-rate-adjustment-array-lookup-samples))

    (is (one-dimensional-double-array? (first (:fuel-number->spread-rate-adjustment-array-lookup-samples inputs-after))))

    (is (= 0.1 (aget (doubles (first (:fuel-number->spread-rate-adjustment-array-lookup-samples inputs-after))) 101)))))
```

9.3.7 gridfire.inputs.envi-bsq

```
(ns gridfire.inputs.envi-bsq
  "A custom parser for BSQ-interleaved ENVI raster files.
  This is a special case of GDAL's ENVI .hdr Labelled Raster:
  https://gdal.org/drivers/raster/envi.html#raster-envi."
  (:require [clojure.data.json  :as json]
            [clojure.java.io    :as io]
            [clojure.java.shell :as sh]
            [clojure.string     :as str]
            [manifold.deferred  :as mfd]
            [tech.v3.datatype   :as d]
            [tech.v3.tensor     :as t])
  (:import (java.io ByteArrayOutputStream)
           (java.nio ByteBuffer ByteOrder Buffer)
           (org.geotools.geometry Envelope2D)
           (org.geotools.referencing CRS)
           (org.opengis.referencing.crs CoordinateReferenceSystem)))

;; NOTE Why implement a custom parser? 2 reasons: (Val, 21 Oct 2022)
;; 1) Preserving GridFire's portability and ease of setup.
;; Setting up the native JVM-gdal interop is a tedious and unpredictable endeavor.
;; 2) Performance: there's every reason to believe that going through
;; GDAL then GeoTools would add a lot of overhead.
;; And at the time of writing, we use BSQ precisely because we want high
;; inputs-loading performance. And indeed, this is much faster than
;; loading equivalent GeoTIFFs.

(defn- slurp-array
```

```
  ^bytes [in]
  (with-open [baos (ByteArrayOutputStream.)]
    (io/copy in baos)
    (.toByteArray baos)))

(defn- sh-successful-out
  [sh-result]
  (if (zero? (:exit sh-result))
    (:out sh-result)
    (throw (ex-info (format "Shell command return error code %s: %s."
                            (str (:exit sh-result))
                            (:err sh-result))
                    sh-result))))

(defn get-gdal-info [f]
  (-> (sh/sh "gdalinfo" "-json" "-proj4" (-> f (io/file) (.getPath)))
      (sh-successful-out)
      ;; NOTE not parsing keys to keywords because some of them are pathological, like "".
      (json/read-str)))

(defn- check-expectation!
  [err-msg find-error-data]
  (when-some [err-data (find-error-data)]
    (throw (ex-info (format "BSQ file violates the GridFire parser's expectation: %s" err-msg)
                    err-data))))

(defmulti byte-buffer->array (fn [^String gdal-data-type ^ByteBuffer _bytbuf ^long _n-elements]
                               gdal-data-type))

(defmethod byte-buffer->array :default
  [gdal-data-type _bytbuf _n-elements]
  (throw (ex-info (format "Unsupported GDAL data type: %s" (pr-str gdal-data-type))
                  {::gdal-data-type gdal-data-type})))

(defn- parse-integer
  ^long [^String vstr]
  (Long/parseLong vstr 10))

(defn- parse-hdr-txt
  [^String hdr-txt]
  (let [hdr-txt1 (-> hdr-txt
                     ;; Should help remove entries spanning several lines
                     (str/replace "{\n" "{")
                     (str/replace ",\n" ","))]
    (->> (for [l             (str/split-lines hdr-txt1)
               prop-to-parse [[::hdr-header-offset "header offset" parse-integer]
                              [::hdr-byte-order-num "byte order" parse-integer]
                              [::hdr-cs-string "coordinate system string"
                               (fn unwrap-cs-string [cstr]
                                 (-> (re-matches #"\{(.*)\}" cstr)
                                     (nth 1)))]]
               :let          [[k propname coerce-fn] prop-to-parse]
               :when         (str/starts-with? l propname)]
           (let [lrest    (subs l (count propname))
                 [_ vstr] (or (re-matches #"\s*=\s*([^\s](.*[^\s])?)\s*" lrest)
                              (throw (ex-info (format "Failed to parse .hdr property %s" (pr-str propname))
                                              {::hdr-prop-name propname
                                               ::hdr-line      l
                                               ::hdr-txt       hdr-txt})))
                 v        ((or coerce-fn identity) vstr)]
             [k v]))
         (into {}))))

(defn find-hdr-file
  [bsq-file]
  (let [bsq-file    (io/file bsq-file)
        fname       (.getName bsq-file)
        [_ fprefix] (re-matches #"(.+)\.[^\.]+" fname)]
```

```
    (io/file (.getParentFile bsq-file) (str fprefix ".hdr"))))

(defn- fetch-hdr-data
  [bsq-file]
  (let [hdr-file (find-hdr-file bsq-file)
        hdr-txt  (slurp hdr-file)]
    (parse-hdr-txt hdr-txt)))

(defn- parse-WKT-into-CRS
  ^CoordinateReferenceSystem [^String wkt]
  (CRS/parseWKT wkt))

(defn read-bsq-file
  "Parses a BSQ-interleaved ENVI file.
  Will shell out to gdalinfo and also use the companion .hdr file,
  which must have the same name and location except for the extension.

  Returns a Manifold Deferred,
  holding 3D (band, x, y)-ordered tensor wrapped in a map of geospatial metadata."
  [bsq-file]
  ;; Returning a Manifold Deferred favors parallel fetching,
  ;; enabling callers to fetch several files in parallel
  ;; without blocking a waiting thread.
  (let [dfr-bsq-bytes (mfd/future (slurp-array (io/file bsq-file)))
        dfr-gdalinfo  (mfd/future (get-gdal-info bsq-file))
        dfr-hdr-data  (mfd/future (fetch-hdr-data bsq-file))]
    ;; Reading files in parallel, to reduce latency.
    (-> (mfd/zip dfr-bsq-bytes dfr-gdalinfo dfr-hdr-data)
        (mfd/chain
         (fn [[^bytes bsq-bytes gdalinfo hdr-data]]
           (check-expectation! "must be ENVI with BSQ layout."
                               (fn []
                                 (when-not (and (= "ENVI .hdr Labelled" (get gdalinfo "driverLongName"))
                                                (= "BAND" (get-in gdalinfo ["metadata" "IMAGE_STRUCTURE" "INTERLEAVE"])))
                                   {::gdalinfo gdalinfo})))
           (check-expectation! (format "must have %s." (pr-str {"dataAxisToSRSAxisMapping" [1 2]}))
                               (fn []
                                 (when-not (= [1 2] (get-in gdalinfo ["coordinateSystem" "dataAxisToSRSAxisMapping"]))
                                   {::gdalinfo gdalinfo})))
           (check-expectation! "must have equivalent bands."
                               (fn []
                                 (when-not (->> (get gdalinfo "bands")
                                                (map #(select-keys % ["noDataValue" "type"]))
                                                (apply =))
                                   {::gdalinfo gdalinfo})))
           (let [[x-width y-length] (get gdalinfo "size")
                 x-width-px         (long x-width)
                 y-height-px        (long y-length)
                 [x-ul y-ul]        (get-in gdalinfo ["cornerCoordinates" "upperLeft"])
                 scalex             (double (get-in gdalinfo ["geoTransform" 1]))
                 scaley             (double (get-in gdalinfo ["geoTransform" 5]))
                 band-info          (get-in gdalinfo ["bands" 0])
                 n-bands            (long (count (get gdalinfo "bands")))
                 byte-order-num     (long (or (::hdr-byte-order-num hdr-data) 0))
                 header-offset      (long (or (::hdr-header-offset hdr-data) 0))
                 data-type          (get band-info "type")
                 n-pixels           (* x-width-px y-height-px n-bands)
                 bytbuf             (-> (ByteBuffer/wrap bsq-bytes header-offset (- (alength bsq-bytes) header-offset))
                                        (.order (case byte-order-num
                                                  0 ByteOrder/LITTLE_ENDIAN
                                                  1 ByteOrder/BIG_ENDIAN)))
                 data-arr           (byte-buffer->array data-type bytbuf n-pixels)
                 tensor             (-> (d/->buffer data-arr)
                                        (t/->tensor)
                                        ;; NOTE if we didn't have {"dataAxisToSRSAxisMapping" [1 2]},
                                        ;; I guess that this should be followed by a (t/transpose [0 2 1]).
                                        (t/reshape [n-bands y-height-px x-width-px]))]
             {;; AFAICT the :srid resolved from layers is never used; (Val, 20 Oct 2022)
```

```
            ;; what we use the SRID of the GridFire config.
            ;; The only downstream use of said SRID is to create an Envelope2D for writing outputs,
            ;; and even that does not really require an SRID:
            ;; we can create an Envelope2D from the CRS,
            ;; and that's actually how Magellan does it behind the scenes.
            ;; See the tests for an example.
            ;; NOTE: why not use gdalsrsinfo to resolve the SRID? (Val, 21 Oct 2022)
            ;; For test data on my machine, that did not return any matches;
            ;; so if we can't rely on it, I think it's not worth implementing it,
            ;; all the more so because the :srid is not really needed, as noted above.
            :srid       nil
            ;; NOTE For lack of an SRID leaving around an Envelope2D object directly,
            ;; so that the raster can still be saved.
            :envelope   (let [wkt         (::hdr-cs-string hdr-data)
                              crs         (parse-WKT-into-CRS wkt)
                              [x-lr y-lr] (get-in gdalinfo ["cornerCoordinates" "lowerRight"])
                              xmin        (min x-ul (double x-lr))
                              ymin        (min y-ul (double y-lr))]
                          (Envelope2D. crs
                                       xmin
                                       ymin
                                       (Math/abs (* scalex x-width-px))
                                       (Math/abs (* scaley y-height-px))))
            :upperleftx x-ul
            :upperlefty y-ul
            :width      x-width-px
            :height     y-height-px
            :scalex     (get-in gdalinfo ["geoTransform" 1])
            :scaley     (get-in gdalinfo ["geoTransform" 5])
            :skewx      0.0                              ; currently unused
            :skewy      0.0                              ; currently unused
            :matrix     tensor})))
        (mfd/catch
         (fn [err]
           (let [file-path (.getPath (io/file bsq-file))]
             (throw (ex-info (format "Error parsing BSQ file (%s) with GridFire's custom parser: %s %s"
                                     (-> bsq-file (io/file) (.getPath))
                                     (-> err (type) (str))
                                     (ex-message err))
                             (merge
                              {::bsq-file-path file-path}
                              (when (mfd/realized? dfr-gdalinfo)
                                (-> dfr-gdalinfo
                                    (mfd/chain (fn [gdalinfo] {::gdalinfo gdalinfo}))
                                    (mfd/catch (constantly nil))
                                    (deref))))
                             err))))))))

(defn- check-expected-buffer-size!
  [^Buffer data-buf n-elements]
  (check-expectation! "must have the expected length."
                      (fn []
                        (when-not (= n-elements (.remaining data-buf))
                          (throw (ex-info (format "expected %d elements in Buffer, got %d"
                                                  n-elements
                                                  data-buf)
                                          {::expected-buffer-size n-elements
                                           ::actual-buffer-size   (.remaining data-buf)
                                           ::data-buffer          data-buf}))))))

;; NOTE unsigned types (UInt16, etc.) are not currently supported. (Val, 20 Oct 2022)
(defmethod byte-buffer->array "Int16"
  [_ ^ByteBuffer bytbuf ^long n-elements]
  (let [data-arr (short-array n-elements)
        data-buf (.asShortBuffer bytbuf)]
    (check-expected-buffer-size! data-buf n-elements)
    (.get data-buf (shorts data-arr))
    data-arr))
```

```
(defmethod byte-buffer->array "Int32"
  [_ ^ByteBuffer bytbuf ^long n-elements]
  (let [data-arr (int-array n-elements)
        data-buf (.asIntBuffer bytbuf)]
    (check-expected-buffer-size! data-buf n-elements)
    (.get data-buf (ints data-arr))
    data-arr))

(defmethod byte-buffer->array "Float32"
  [_ ^ByteBuffer bytbuf ^long n-elements]
  (let [data-arr (float-array n-elements)
        data-buf (.asFloatBuffer bytbuf)]
    (check-expected-buffer-size! data-buf n-elements)
    (.get data-buf (floats data-arr))
    data-arr))

(defmethod byte-buffer->array "Float64"
  [_ ^ByteBuffer bytbuf ^long n-elements]
  (let [data-arr (double-array n-elements)
        data-buf (.asDoubleBuffer bytbuf)]
    (check-expected-buffer-size! data-buf n-elements)
    (.get data-buf (doubles data-arr))
    data-arr))
```

9.3.8 gridfire.inputs.envi-bsq-test

```
(ns gridfire.inputs.envi-bsq-test
  (:require [clojure.java.io             :as io]
            [clojure.string              :as str]
            [clojure.test                :refer [deftest is testing]]
            [gridfire.inputs.envi-bsq    :as gf-bsq]
            [gridfire.magellan-bridge    :refer [geotiff-raster-to-tensor]]
            [gridfire.utils.test         :as utils]
            [magellan.core               :refer [matrix-to-raster]]
            [tech.v3.datatype            :as d]
            [tech.v3.datatype.functional :as dfn]
            [tech.v3.tensor              :as t])
  (:import (java.io File)
           (org.geotools.geometry Envelope2D)))

(defn- file-name
  [^File f]
  (.getName f))

(defn- test-files-ending-with
  [ext]
  (->> (file-seq (io/file "./test/gridfire/resources/envi-bsq-examples"))
       (filter (fn [f] (-> (file-name f) (str/ends-with? ext))))
       (vec)))

(defn get-test-bsq-files
  []
  (test-files-ending-with ".bsq"))

(deftest ^:unit available-test-files-test
  (testing "We test the BSQ-parsing on clipped inputs files from a CONUS run:"
   (is (= (into (sorted-set)
                (map file-name)
                (get-test-bsq-files))
          #{"2020_90_SDIx100.bsq"
            "adj.bsq"
            "asp.bsq"
            "cbd.bsq"
            "cbh.bsq"
            "cc.bsq"
```

```
            "ch.bsq"
            "dem.bsq"
            "ercperc_2018.bsq"
            "fbfm40.bsq"
            "ignition_density.bsq"
            "lh_2018.bsq"
            "lw_2018.bsq"
            "m100_2018.bsq"
            "m10_2018.bsq"
            "m1_2018.bsq"
            "phi.bsq"
            "pyromes.bsq"
            "slp.bsq"
            "wd_2018.bsq"
            "ws_2018.bsq"})))))

(comment
  ;; How the clipped BSQ files were created: (Val, 21 Oct 2022)
  ;; 1) Let's figure out a reasonable bounding box for clipping.
  (-> (gf-bsq/get-gdal-info "../007_037/ws_2018.bsq")       ; large-pixels weather input.
      (select-keys ["size" "geoTransform" "cornerCoordinates"]))
  ;;=>
  {"size" [40 40],
   "geoTransform" [-2074425.0 1200.0 -0.0 1953435.0 -0.0 -1200.0],
   "cornerCoordinates" {"upperLeft" [-2074425.0 1953435.0],
                        "lowerLeft" [-2074425.0 1905435.0],
                        "lowerRight" [-2026425.0 1905435.0],
                        "upperRight" [-2026425.0 1953435.0],
                        "center" [-2050425.0 1929435.0]}}
  (assert (= (* 40 1200.0) (- -2026425.0 -2074425.0)))
  ;; 2) Creating smaller clipped files.
  (require '[clojure.java.shell :as sh])
  (->> (file-seq (io/file "../007_037"))
       (filter (fn [f] (-> f (str) (str/ends-with? ".bsq"))))
       (pmap
        (fn [^java.io.File src-bsq-file]
          (let [clipped-bsq-file (io/file "./test/gridfire/resources/envi-bsq-examples"
                                          (file-name src-bsq-file))
                xmax -2026425.0
                ymax 1953435.0
                ;; Chose a sub-rectangle of 2x3 "weather pixels". (Val, 21 Oct 2022)
                ;; I checked in QGis that this subrectangle contains representative, varied and non-trivial data.
                xmin (- xmax (* 1200.0 2))
                ymin (- ymax (* 1200.0 3))]
            (io/make-parents clipped-bsq-file)
            ;; NOTE I checked that this transformation did not alter
            ;; the encoding and metadata of the BSQ files.
            (sh/sh "gdalwarp"
                   "-overwrite"
                   "-te" (str xmin) (str ymin) (str xmax) (str ymax)
                   "-of" "ENVI"
                   (.getPath src-bsq-file)
                   (.getPath clipped-bsq-file)))))
       (doall))

  *e)

(defn tif-of-bsq-file
  ^File [bsq-file]
  (let [bsq-file (io/file bsq-file)]
    (-> bsq-file
        (.getParentFile)
        (io/file (str/replace (file-name bsq-file) ".bsq" ".tif")))))

(comment

  ;; How the test Geotiffs were created from the BSQ files: (Val, 21 Oct 2022)
  (->> (get-test-bsq-files)
```

```
       (pmap
        (fn [^java.io.File bsq-file]
          (let [geotiff-file (tif-of-bsq-file bsq-file)]
            (sh/sh "gdal_translate" "-if" "ENVI" (.getPath bsq-file) (.getPath geotiff-file)))))
       (doall))

  *e)

(def small-example-filenames #{"fbfm40.bsq" "cbh.bsq" "phi.bsq"})

(deftest ^:bsq-test-suite bsq-metadata-examples-test
  (testing "Examples of .hdr files."
    ;; NOTE this test case does little to guarantee correctness:
    ;; instead, it's more of a documentation test case,
    ;; with the added benefit of enforcing that our expectations
    ;; about what the HDR files contain are preserved
    ;; as the codebase evolves.
    (is (= (->> (get-test-bsq-files)
                (filter #(contains? small-example-filenames (file-name %)))
                (pmap (fn [bsq-file]
                        [(file-name bsq-file)
                         (-> (gf-bsq/find-hdr-file bsq-file)
                             (slurp)
                             (str/split-lines)
                             (vec))]))
                (into (sorted-map)))
           {"cbh.bsq"    ["ENVI"
                          "description = {"
                          "./test/gridfire/resources/envi-bsq-examples/cbh.bsq}"
                          "samples = 80"
                          "lines   = 120"
                          "bands   = 1"
                          "header offset = 0"
                          "file type = ENVI Standard"
                          "data type = 2"
                          "interleave = bsq"
                          "byte order = 0"
                          "map info = {Albers Conical Equal Area, 1, 1, -2028825, 1953435, 30, 30,North America 1983}"
                          "projection info = {9, 6378137, 6356752.314140356, 23, -96, 0, 0, 29.5, 45.5,North America 1983, Albers C
                          "coordinate system string = {PROJCS[\"unknown\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_Americ
                          "band names = {"
                          "Band 1}"
                          "data ignore value = -9999"],
            "fbfm40.bsq" ["ENVI"
                          "description = {"
                          "./test/gridfire/resources/envi-bsq-examples/fbfm40.bsq}"
                          "samples = 80"
                          "lines   = 120"
                          "bands   = 1"
                          "header offset = 0"
                          "file type = ENVI Standard"
                          "data type = 2"
                          "interleave = bsq"
                          "byte order = 0"
                          "map info = {Albers Conical Equal Area, 1, 1, -2028825, 1953435, 30, 30,North America 1983}"
                          "projection info = {9, 6378137, 6356752.314140356, 23, -96, 0, 0, 29.5, 45.5,North America 1983, Albers C
                          "coordinate system string = {PROJCS[\"unknown\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_Americ
                          "band names = {"
                          "Band 1}"
                          "data ignore value = -9999"],
            "phi.bsq"    ["ENVI"
                          "description = {"
                          "./test/gridfire/resources/envi-bsq-examples/phi.bsq}"
                          "samples = 80"
                          "lines   = 120"
                          "bands   = 1"
                          "header offset = 0"
                          "file type = ENVI Standard"
```

```
                        "data type = 4"
                        "interleave = bsq"
                        "byte order = 0"
                        "map info = {Albers Conical Equal Area, 1, 1, -2028825, 1953435, 30, 30,North America 1983}"
                        "projection info = {9, 6378137, 6356752.314140356, 23, -96, 0, 0, 29.5, 45.5,North America 1983, Albers C
                        "coordinate system string = {PROJCS[\"unknown\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_Americ
                        "band names = {"
                        "Band 1}"
                        "data ignore value = -9999"]})))
(testing "Examples of gdalinfo output."
  (is (= (->> (get-test-bsq-files)
              (filter #(contains? small-example-filenames (file-name %)))
              (pmap (fn [bsq-file]
                      [(file-name bsq-file)
                       (-> (gf-bsq/get-gdal-info bsq-file)
                           (update-in ["coordinateSystem" "wkt"]
                                      (fn elide-wkt [wkt]
                                        (str (subs wkt 0 8) " ...(ELIDED BY TESTING CODE)..."))))]))
              (utils/round-floating-point 1e2)
              (into (sorted-map)))
         {"cbh.bsq"    {"bands"             [{"band"               1
                                              "block"              [80 1]
                                              "type"               "Int16"
                                              "colorInterpretation" "Undefined"
                                              "noDataValue"        -9999.0
                                              "metadata"           {}}]
                        "driverLongName"    "ENVI .hdr Labelled"
                        "coordinateSystem"  {"wkt"                      "PROJCRS[ ...(ELIDED BY TESTING CODE)..."
                                             "dataAxisToSRSAxisMapping" [1 2]
                                             "proj4"                    "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +
                        "files"             ["./test/gridfire/resources/envi-bsq-examples/cbh.bsq"
                                             "./test/gridfire/resources/envi-bsq-examples/cbh.bsq.aux.xml"
                                             "./test/gridfire/resources/envi-bsq-examples/cbh.hdr"]
                        "wgs84Extent"       {"type"        "Polygon"
                                             "coordinates" [[[-119.67 38.33]
                                                             [-119.66 38.3]
                                                             [-119.64 38.3]
                                                             [-119.65 38.33]
                                                             [-119.67 38.33]]]}
                        "metadata"          {"" {"AREA_OR_POINT" "Area" "Band_1" "Band 1"} "IMAGE_STRUCTURE" {"INTERLEAVE" "BAND"
                        "geoTransform"      [-2028825.0 30.0 0.0 1953435.0 0.0 -30.0]
                        "size"              [80 120]
                        "cornerCoordinates" {"upperLeft"  [-2028825.0 1953435.0]
                                             "lowerLeft"  [-2028825.0 1949835.0]
                                             "lowerRight" [-2026425.0 1949835.0]
                                             "upperRight" [-2026425.0 1953435.0]
                                             "center"     [-2027625.0 1951635.0]}
                        "driverShortName"   "ENVI"
                        "description"       "./test/gridfire/resources/envi-bsq-examples/cbh.bsq"}
          "fbfm40.bsq" {"bands"             [{"band"               1
                                              "block"              [80 1]
                                              "type"               "Int16"
                                              "colorInterpretation" "Undefined"
                                              "noDataValue"        -9999.0
                                              "metadata"           {}}]
                        "driverLongName"    "ENVI .hdr Labelled"
                        "coordinateSystem"  {"wkt"                      "PROJCRS[ ...(ELIDED BY TESTING CODE)..."
                                             "dataAxisToSRSAxisMapping" [1 2]
                                             "proj4"                    "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +
                        "files"             ["./test/gridfire/resources/envi-bsq-examples/fbfm40.bsq"
                                             "./test/gridfire/resources/envi-bsq-examples/fbfm40.bsq.aux.xml"
                                             "./test/gridfire/resources/envi-bsq-examples/fbfm40.hdr"]
                        "wgs84Extent"       {"type"        "Polygon"
                                             "coordinates" [[[-119.67 38.33]
                                                             [-119.66 38.3]
                                                             [-119.64 38.3]
                                                             [-119.65 38.33]
                                                             [-119.67 38.33]]]}
```

```
                        "metadata"          {"" {"AREA_OR_POINT" "Area" "Band_1" "Band 1"} "IMAGE_STRUCTURE" {"INTERLEAVE" "BAND"
                        "geoTransform"      [-2028825.0 30.0 0.0 1953435.0 0.0 -30.0]
                        "size"              [80 120]
                        "cornerCoordinates" {"upperLeft"  [-2028825.0 1953435.0]
                                             "lowerLeft"  [-2028825.0 1949835.0]
                                             "lowerRight" [-2026425.0 1949835.0]
                                             "upperRight" [-2026425.0 1953435.0]
                                             "center"     [-2027625.0 1951635.0]}
                        "driverShortName"   "ENVI"
                        "description"       "./test/gridfire/resources/envi-bsq-examples/fbfm40.bsq"}
           "phi.bsq"    {"bands"             [{"band"                1
                                                "block"               [80 1]
                                                "type"                "Float32"
                                                "colorInterpretation" "Undefined"
                                                "noDataValue"         -9999.0
                                                "metadata"            {}}]
                        "driverLongName"    "ENVI .hdr Labelled"
                        "coordinateSystem"  {"wkt"                      "PROJCRS[ ...(ELIDED BY TESTING CODE)..."
                                             "dataAxisToSRSAxisMapping" [1 2]
                                             "proj4"                    "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +
                        "files"             ["./test/gridfire/resources/envi-bsq-examples/phi.bsq"
                                             "./test/gridfire/resources/envi-bsq-examples/phi.bsq.aux.xml"
                                             "./test/gridfire/resources/envi-bsq-examples/phi.hdr"]
                        "wgs84Extent"       {"type"        "Polygon"
                                             "coordinates" [[[-119.67 38.33]
                                                            [-119.66 38.3]
                                                            [-119.64 38.3]
                                                            [-119.65 38.33]
                                                            [-119.67 38.33]]]}
                        "metadata"          {"" {"AREA_OR_POINT" "Area" "Band_1" "Band 1"} "IMAGE_STRUCTURE" {"INTERLEAVE" "BAND"
                        "geoTransform"      [-2028825.0 30.0 0.0 1953435.0 0.0 -30.0]
                        "size"              [80 120]
                        "cornerCoordinates" {"upperLeft"  [-2028825.0 1953435.0]
                                             "lowerLeft"  [-2028825.0 1949835.0]
                                             "lowerRight" [-2026425.0 1949835.0]
                                             "upperRight" [-2026425.0 1953435.0]
                                             "center"     [-2027625.0 1951635.0]}
                        "driverShortName"   "ENVI"
                        "description"       "./test/gridfire/resources/envi-bsq-examples/phi.bsq"}})))))

(deftest ^:bsq-test-suite represented-data-types-test
  (testing "We have tested on the following data types."
    (is (= (->> (get-test-bsq-files)
                (pmap (fn [bsq-file]
                        {(-> (gf-bsq/get-gdal-info bsq-file)
                             (get-in ["bands" 0 "type"]))
                         (sorted-set (file-name bsq-file))}))
                (apply merge-with into))
           {"Int16"   #{"asp.bsq"
                        "cbd.bsq"
                        "cbh.bsq"
                        "cc.bsq"
                        "ch.bsq"
                        "dem.bsq"
                        "fbfm40.bsq"
                        "pyromes.bsq"
                        "slp.bsq"},
            "Float32" #{"2020_90_SDIx100.bsq"
                        "adj.bsq"
                        "ercperc_2018.bsq"
                        "ignition_density.bsq"
                        "lh_2018.bsq"
                        "lw_2018.bsq"
                        "m100_2018.bsq"
                        "m10_2018.bsq"
                        "m1_2018.bsq"
                        "phi.bsq"
                        "wd_2018.bsq"
```

```
                      "ws_2018.bsq"}})))))

(deftest ^:bsq-test-suite example-bsq-load-like-the-geotiffs-test
  (testing "We parse our BSQ rasters equivalently to the corresponding GeoTIFFs."
    (->> (get-test-bsq-files)
         (pmap
          (fn [bsq-file]
            (let [tif-file (tif-of-bsq-file bsq-file)
                  bsq-map  @(gf-bsq/read-bsq-file bsq-file)
                  tif-map  (geotiff-raster-to-tensor tif-file)]
              (testing "The geospatial metadata are the same."
                (let [relevant-gis-ks [:upperleftx
                                       :upperlefty
                                       :width
                                       :height
                                       :scalex
                                       :scaley]]
                  (is (= (select-keys bsq-map relevant-gis-ks)
                         (select-keys tif-map relevant-gis-ks))
                      "Same GIS metadata.")))
              (testing "We return {:srid nil}, but do provide an Envelope2D at :envelope."
                (is (nil? (:srid bsq-map)))
                (is (instance? Envelope2D (:envelope bsq-map)))
                (testing "This Envelope2D can then be supplied to Magellan to save the data as GeoTIFF."
                  (let [tensor   (:matrix bsq-map)
                        matrix2D (t/mget tensor 0)]
                    (is (some? (matrix-to-raster (-> bsq-file (io/file) (file-name))
                                                 matrix2D
                                                 (:envelope bsq-map)))))))
              (testing "The tensors contain the same values."
                (let [bsq-tensor (-> (:matrix bsq-map)
                                     (as-> t
                                           (if (-> t (d/shape) (first) (= 1))
                                             (t/mget t 0)
                                             t)))
                      tif-tensor (:matrix tif-map)]
                  (is (= (d/shape bsq-tensor) (d/shape tif-tensor))
                      "Same matrix dimensions.")
                  (is (dfn/equals bsq-tensor tif-tensor)
                      "Same numeric values."))))))
         (doall))))
```

9.3.9 gridfire.fetch-ignition-test

```
(ns gridfire.fetch-ignition-test
  (:require [clojure.test                :refer [deftest is testing use-fixtures]]
            [gridfire.fetch              :as fetch]
            [gridfire.utils.test         :as utils]
            [tech.v3.datatype.functional :as dfn]))

;;------------------------------------------------------------------------------
;; Config
;;------------------------------------------------------------------------------

(def resources-path "test/gridfire/resources/")

(def db-spec {:classname   "org.postgresql.Driver"
              :subprotocol "postgresql"
              :subname     "//localhost:5432/gridfire_test"
              :user        "gridfire_test"
              :password    "gridfire_test"})

(def test-config-base
  {:db-spec                  db-spec
   :landfire-layers          {:aspect            {:type   :geotiff
                                                  :source "test/gridfire/resources/asp.tif"}
```

```
                              :canopy-base-height {:type   :geotiff
                                                   :source "test/gridfire/resources/cbh.tif"}
                              :canopy-cover       {:type   :geotiff
                                                   :source "test/gridfire/resources/cc.tif"}
                              :canopy-height      {:type   :geotiff
                                                   :source "test/gridfire/resources/ch.tif"}
                              :crown-bulk-density {:type   :geotiff
                                                   :source "test/gridfire/resources/cbd.tif"}
                              :elevation          {:type   :geotiff
                                                   :source "test/gridfire/resources/dem.tif"}
                              :fuel-model         {:type   :geotiff
                                                   :source "test/gridfire/resources/fbfm40.tif"}
                              :slope              {:type   :geotiff
                                                   :source "test/gridfire/resources/slp.tif"}}
   :srid                      "CUSTOM:900914"
   :cell-size                 98.425     ; (feet)
   :max-runtime               60         ; (minutes)
   :temperature               '(50)      ; (degrees Fahrenheit)
   :relative-humidity         '(1)       ; (%)
   :wind-speed-20ft           '(10)      ; (miles/hour)
   :wind-from-direction       '(0)       ; (degrees clockwise from north)
   :foliar-moisture           90         ; (%)
   :ellipse-adjustment-factor 1.0        ; (< 1.0 = more circular, > 1.0 = more elliptical)
   :simulations               1
   :random-seed               1234567890 ; long value (optional)
   :output-csvs?              true})

;;-------------------------------------------------------------------------------
;; Utils
;;-------------------------------------------------------------------------------

(defn in-file-path [filename]
  (str resources-path filename))

;;-------------------------------------------------------------------------------
;; Fixtures
;;-------------------------------------------------------------------------------

(use-fixtures :once utils/with-reset-db-pool)

;;-------------------------------------------------------------------------------
;; Tests
;;-------------------------------------------------------------------------------

(deftest ^:database fetch-ignition-layer-test
  (testing "Fetching ignition layer from postgis and geotiff file"
    (let [geotiff-config         (merge test-config-base
                                        {:ignition-layer {:type   :geotiff
                                                          :source (in-file-path "ign.tif")}})
          postgis-config         (merge test-config-base
                                        {:ignition-layer {:type   :postgis
                                                          :source "ignition.ign WHERE rid=1"}})
          geotiff-ignition-layer (fetch/ignition-layer postgis-config)
          postgis-ignition-layer (fetch/ignition-layer geotiff-config)]

      (is (dfn/equals (:matrix geotiff-ignition-layer)
                      (:matrix postgis-ignition-layer))))))
```

9.3.10 gridfire.fetch-fuel-moisture-test

```
(ns gridfire.fetch-fuel-moisture-test
  (:require
   [clojure.test               :refer [deftest are]]
   [gridfire.fetch             :as fetch]
   [gridfire.magellan-bridge   :refer [geotiff-raster-to-tensor]]
   [gridfire.utils.test        :as utils]
```

```
   [gridfire.conversion         :as convert]
   [tech.v3.datatype            :as d]
   [tech.v3.datatype.functional :as dfn]))

(def resources-path "test/gridfire/resources/weather-test")

(defn- equal-matrix?
  [inputs category size]
  (let [matrix (-> (get-in inputs [:fuel-moisture category size :source])
                   geotiff-raster-to-tensor
                   :matrix)]
    (dfn/equals (:matrix (fetch/fuel-moisture-layer inputs category size))
                (d/clone (d/emap convert/percent->dec nil matrix)))))

(deftest ^:unit fuel-moisture-test
  (let [inputs {:fuel-moisture
                {:dead {:1hr   {:type   :geotiff
                                :source (utils/in-file-path resources-path "m1_to_sample.tif")}
                        :10hr  {:type   :geotiff
                                :source (utils/in-file-path resources-path "m10_to_sample.tif")}
                        :100hr {:type   :geotiff
                                :source (utils/in-file-path resources-path "m100_to_sample.tif")}}
                 :live {:woody      {:type   :geotiff
                                     :source (utils/in-file-path resources-path "mlw_to_sample.tif")}
                        :herbaceous {:type   :geotiff
                                     :source (utils/in-file-path resources-path "mlh_to_sample.tif")}}}}]

    (are [category size] (equal-matrix? inputs category size)
      :dead :1hr
      :dead :10hr
      :dead :100hr
      :live :herbaceous
      :live :woody)))
(comment
  (clojure.test/test-vars [#'fuel-moisture-test])

  *e)
```

9.3.11 gridfire.fetch-weather-test

```
(ns gridfire.fetch-weather-test
  (:require [clojure.java.jdbc           :as jdbc]
            [clojure.test                :refer [deftest is testing use-fixtures]]
            [gridfire.core               :refer [load-config!]]
            [gridfire.fetch              :as fetch]
            [gridfire.inputs             :as inputs]
            [gridfire.spec.common        :refer [*check-files-exist?*]]
            [gridfire.utils.test         :as utils]
            [magellan.core               :refer [read-raster]]
            [tech.v3.datatype            :as d]
            [tech.v3.tensor              :as t]
            [tech.v3.datatype.functional :as dfn])
  (:import java.util.Random))

;;------------------------------------------------------------------------------
;; Config
;;------------------------------------------------------------------------------

(def resources-path "test/gridfire/resources/weather-test/")

(def db-spec {:classname   "org.postgresql.Driver"
              :subprotocol "postgresql"
              :subname     "//localhost:5432/gridfire_test"
              :user        "gridfire_test"
              :password    "gridfire_test"})
```

```
(def test-config-base
  {:db-spec     db-spec
   :simulations 1
   :random-seed 1234567890
   :rand-gen    (Random. 1234567890)})

;;-------------------------------------------------------------------------------
;; Utils
;;-------------------------------------------------------------------------------

(defn in-file-path [filename]
  (str resources-path filename))


;;-------------------------------------------------------------------------------
;; Fixtures
;;-------------------------------------------------------------------------------

(use-fixtures :once utils/with-reset-db-pool)

;;-------------------------------------------------------------------------------
;; Tests
;;-------------------------------------------------------------------------------

(deftest ^:database fetch-temperature-test
  (let [geotiff-file    "tmpf_to_sample.tif"
        geotiff-config  (merge test-config-base
                               {:temperature {:type   :geotiff
                                              :source (in-file-path geotiff-file)}})
        postgis-table   "weather.tmpf WHERE rid=1"
        postgis-config  (merge test-config-base
                               {:temperature {:type   :postgis
                                              :source postgis-table}})
        geotiff-results (:matrix (fetch/weather-layer geotiff-config :temperature))
        postgis-results (:matrix (fetch/weather-layer postgis-config :temperature))]

    (is (every? t/tensor? geotiff-results))

    (is (every? t/tensor? postgis-results))

    (is (dfn/equals geotiff-results postgis-results))

    (let [numbands (count (:bands (read-raster (in-file-path geotiff-file))))]
      (is (= numbands (-> (t/tensor->dimensions geotiff-results) :shape first))))

    (let [results  (jdbc/with-db-connection [conn (:db-spec test-config-base)]
                     (jdbc/query conn [(str "SELECT (ST_Metadata(rast)).numbands FROM " postgis-table)]))
          numbands (:numbands (first results))]
      (is (= numbands (-> (t/tensor->dimensions geotiff-results) :shape first))))))

(deftest ^:database fetch-relative-humidity-test
  (let [geotiff-file    "rh_to_sample.tif"
        geotiff-config  (merge test-config-base
                               {:relative-humidity {:type   :geotiff
                                                    :source (in-file-path geotiff-file)}})
        postgis-table   "weather.rh WHERE rid=1"
        postgis-config  (merge test-config-base
                               {:relative-humidity {:type   :postgis
                                                    :source postgis-table}})
        geotiff-results (:matrix (fetch/weather-layer geotiff-config :relative-humidity))
        postgis-results (:matrix (fetch/weather-layer postgis-config :relative-humidity))]

    (is (every? t/tensor? geotiff-results))

    (is (every? t/tensor? postgis-results))

    (is (dfn/equals geotiff-results postgis-results))
```

```
    (let [numbands (count (:bands (read-raster (in-file-path geotiff-file))))]
      (is (= numbands (-> (t/tensor->dimensions geotiff-results) :shape first))))

    (let [results  (jdbc/with-db-connection [conn (:db-spec test-config-base)]
                     (jdbc/query conn [(str "SELECT (ST_Metadata(rast)).numbands FROM " postgis-table)]))
          numbands (:numbands (first results))]
      (is (= numbands (-> (t/tensor->dimensions postgis-results) :shape first))))))

(deftest ^:database fetch-wind-speed-20ft-test
  (let [geotiff-file    "ws_to_sample.tif"
        geotiff-config  (merge test-config-base
                               {:wind-speed-20ft {:type   :geotiff
                                                  :source (in-file-path geotiff-file)}})
        postgis-table   "weather.ws WHERE rid=1"
        postgis-config  (merge test-config-base
                               {:wind-speed-20ft {:type   :postgis
                                                  :source postgis-table}})
        geotiff-results (:matrix (fetch/weather-layer geotiff-config :wind-speed-20ft))
        postgis-results (:matrix (fetch/weather-layer postgis-config :wind-speed-20ft))]

    (is (every? t/tensor? geotiff-results))

    (is (every? t/tensor? postgis-results))

    (is (dfn/equals geotiff-results postgis-results))

    (let [numbands (count (:bands (read-raster (in-file-path geotiff-file))))]
      (is (= numbands (-> (t/tensor->dimensions geotiff-results) :shape first))))

    (let [results  (jdbc/with-db-connection [conn (:db-spec test-config-base)]
                     (jdbc/query conn [(str "SELECT (ST_Metadata(rast)).numbands FROM " postgis-table)]))
          numbands (:numbands (first results))]
      (is (= numbands (-> (t/tensor->dimensions postgis-results) :shape first))))))

(deftest ^:database fetch-wind-from-direction-test
  (let [geotiff-file    "wd_to_sample.tif"
        geotiff-config  (merge test-config-base
                               {:wind-from-direction {:type   :geotiff
                                                      :source (in-file-path geotiff-file)}})
        postgis-table   "weather.wd WHERE rid=1"
        postgis-config  (merge test-config-base
                               {:wind-from-direction {:type   :postgis
                                                      :source postgis-table}})
        geotiff-results (:matrix (fetch/weather-layer geotiff-config :wind-from-direction))
        postgis-results (:matrix (fetch/weather-layer postgis-config :wind-from-direction))]

    (is (every? t/tensor? geotiff-results))

    (is (every? t/tensor? postgis-results))

    (is (dfn/equals geotiff-results postgis-results))

    (let [numbands (count (:bands (read-raster (in-file-path geotiff-file))))]
      (is (= numbands (-> (t/tensor->dimensions geotiff-results) :shape first))))

    (let [results  (jdbc/with-db-connection [conn (:db-spec test-config-base)]
                     (jdbc/query conn [(str "SELECT (ST_Metadata(rast)).numbands FROM " postgis-table)]))
          numbands (:numbands (first results))]
      (is (= numbands (-> (t/tensor->dimensions postgis-results) :shape first))))))

(deftest ^:database get-weather-from-range-test
  (let [config  (assoc test-config-base
                       :temperature [0 100]
                       :simulations 10)
        results (inputs/get-weather config :temperature)]

    (is (vector results))
```

```
    (is (every? double? results))))

(deftest ^:database get-weather-from-list-test
  (let [tmp-list (list 0 10 20 30)
        config   (assoc test-config-base
                        :temperature tmp-list
                        :simulations 10)
        results  (inputs/get-weather config :temperature)]

    (is (vector results))

    (is (every? int? results))

    (is (= (set results) (set tmp-list)))))

(deftest ^:database get-weather-scalar-from-test
  (let [config  (assoc test-config-base
                       :temperature 42
                       :simulations 10)
        results (inputs/get-weather config :temperature)]

    (is (vector results))

    (is (every? int? results))

    (is (apply = results))))

(deftest ^:unit add-correction-angle360-test
  (binding [*check-files-exist?* false]
    (testing (pr-str :gridfire.input/add-correction-angle360)
      (testing (str "can be used to correct " :wind-from-direction " or " :aspect " for Grid Declination (divergence between grid N
        (let [config    (-> (load-config! "test/gridfire/resources/canonical_test/base-config.edn")
                            (assoc :wind-from-direction {:type                                    :geotiff
                                                         :source                                  "test/gridfire/resources/canonical
                                                         ;; NOTE why not correct for Grid Declination at the top-level rather than n
                                                         ;; 1) Sometimes not all inputs need the correction.
                                                         ;; 2) That would hardly be compatible with tiled inputs (:grid-of-rasters).
                                                         :gridfire.input/add-correction-angle360 12.0}))
              wd-layer  (fetch/weather-layer config :wind-from-direction)
              wd-matrix (:matrix wd-layer)]
          (is (dfn/equals wd-matrix
                          (t/const-tensor 12.0 (d/shape wd-matrix)))
              "the correction has been applied."))))))
```

9.4 Emitting Outputs

9.4.1 gridfire.outputs

```
(ns gridfire.outputs
  (:require [clojure.data.csv     :as csv]
            [clojure.java.io      :as io]
            [clojure.string       :as str]
            [gridfire.utils.async :as gf-async]
            [magellan.core        :refer [matrix-to-raster write-raster]]
            [manifold.deferred    :as mfd]
            [matrix-viz.core      :refer [save-matrix-as-png]]
            [tech.v3.datatype     :as d]))

(set! *unchecked-math* :warn-on-boxed)

(defn output-filename [name outfile-suffix simulation-id output-time ext]
  (as-> [name outfile-suffix simulation-id (when output-time (str "t" output-time))] $
    (remove str/blank? $)
    (str/join "_" $)
    (str $ ext)))
```

```clojure
(defonce ^:private outputs-writing-exectr*
  (delay
    ;; This is expensive to create, hence the (defonce ...) and (delay ...) combination.
    (let [;; AFAICT our outputs-writing tends to be CPU-bound,
          ;; hence this choice of parallelism.
          ;; Benchmarks on 4 cores have shown an only-2x performance enhancement,
          ;; so there might be some contention at play here,
          ;; and the optimal parallelism might be even lower.
          writing-parallelism (.availableProcessors (Runtime/getRuntime))]
      (gf-async/executor-with-n-named-threads
        "gridfire-outputs-writing"
        writing-parallelism
        {:onto? true}))))

(defn outputs-writing-executor
  "(Advanced) returns the pool of worker Java Threads dedicated to writing GridFire outputs.

   This enables us to parallelize the outputs-writing work (improving throughput)
   while controlling the degree of parallelism,
   and making it easy to monitor,
   as well as straightforward to orchestrate without blocking threads,
   thanks to the Manifold library.

   To achieve this, the heavy-lifting of output-writing
   must be done in tasks scheduled into this pool by calling
   (exec-in-outputs-writing-pool ...)"
  []
  @outputs-writing-exectr*)

;; Most of the following functions return either nil or Manifold Deferred,
;; resolved to nil upon completion of the function's side-effects,
;; so that they can easily be run in parallel,
;; and coordinated with minimal thread-blocking.
;; Manifold Deferreds are semantically equivalent to Promises in other languages;
;; using them makes async flow control straightforward to implement,
;; while reaping the benefits of parallelizing
;; into well-instrumented Thread pools without blocking threads.
;; Manifold was chosen because it is battle-tested, expressive,
;; well-suited to this sort of 'hierarchical' async flow control,
;; and offers good facilities for controlling thread pools.
;; It is harder to achieve these benefits cleanly by using core.async channels,
;; which force you into a side-effectful programming model,
;; and offer little opportunities for fine-tuning;
;; that said, Manifold is easy to compose with core.async,
;; which is interesting if you want to benefit from its (go ...) primitive.

(defn exec-in-outputs-writing-pool
  "Schedules the given 0-arg function to be executed in the thread pool
  dedicated to GridFire outputs-writing.

  Returns a Manifold Deferred, which will be completed with the return
  value of f.

  The function f may itself return a Manifold Deferred."
  [f]
  (letfn [(run-output-task []
            ;; Wrapping in a named fn makes it easy to spot in stacktraces,
            ;; thread dumps, flame graphs etc.
            (f))]
    (->
      (mfd/future-with (outputs-writing-executor)
        (run-output-task))
      ;; To advanced users: this helps ensure that (mfd/chain ...) callbacks
      ;; attached to this Deferred will execute in the same pool.
      ;; Even with that, it's not guaranteed to happen that way,
      ;; in situations where (f) executes so fast that this Deferred
      ;; gets completed even before the mfd/chain function is invoked:
      ;; in such cases, the callback will execute in the Thread invoking mfd/chain.
```

```
      ;; When it doubt: call (exec-in-outputs-writing-pool ...) again inside your callback.
      (mfd/onto (outputs-writing-executor)))))

(defn output-geotiff
  ([config matrix name envelope]
   (output-geotiff config matrix name envelope nil nil))

  ([config matrix name envelope simulation-id]
   (output-geotiff config matrix name envelope simulation-id nil))

  ([{:keys [output-directory outfile-suffix] :as config}
    matrix name envelope simulation-id output-time]
   (exec-in-outputs-writing-pool
     (fn []
       (let [file-name (output-filename name
                                        outfile-suffix
                                        (str simulation-id)
                                        output-time
                                        ".tif")]
         (-> (matrix-to-raster name matrix envelope)
             (write-raster (if output-directory
                             (str/join "/" [output-directory file-name])
                             file-name))))))))

(defn output-png
  ([config matrix name envelope]
   (output-png config matrix name envelope nil nil))

  ([config matrix name envelope simulation-id]
   (output-png config matrix name envelope simulation-id nil))

  ([{:keys [output-directory outfile-suffix]}
    matrix name envelope simulation-id output-time]
   (exec-in-outputs-writing-pool
     (fn []
       (let [file-name (output-filename name
                                        outfile-suffix
                                        (str simulation-id)
                                        output-time
                                        ".png")]
         (save-matrix-as-png :color 4 -1.0
                             matrix
                             (if output-directory
                               (str/join "/" [output-directory file-name])
                               (file-name))))))))

(defn write-landfire-layers!
  [{:keys [output-landfire-inputs? outfile-suffix landfire-rasters envelope]}]
  (when output-landfire-inputs?
    (->> landfire-rasters
         (mapv
           (fn [[layer matrix]]
             (exec-in-outputs-writing-pool
               (fn []
                 (-> (matrix-to-raster (name layer) matrix envelope)
                     (write-raster (str (name layer) outfile-suffix ".tif")))))))
         (gf-async/nil-when-all-completed))))

(defn write-burn-probability-layer!
  [{:keys [output-burn-probability simulations envelope output-pngs? burn-count-matrix] :as outputs}]
  (when-let [timestep output-burn-probability]
    (let [output-name        "burn_probability"
          simulations        (long simulations)
          div-by-simulations (fn ^double [^long burn-count]
                               (double (/ burn-count simulations)))]
      (if (int? timestep)
        (let [timestep (long timestep)]
          (->> (map-indexed vector burn-count-matrix)
```

```
              (mapv
               (fn [[band matrix]]
                 (->
                  (exec-in-outputs-writing-pool
                   (fn []
                     (let [output-time        (* (long band) timestep)
                           probability-matrix (d/clone (d/emap div-by-simulations nil matrix))]
                       [output-time probability-matrix])))
                  (mfd/chain
                   (fn [[output-time probability-matrix]]
                     (mfd/zip
                      (output-geotiff outputs probability-matrix output-name envelope nil output-time)
                      (when output-pngs?
                        (output-png outputs probability-matrix output-name envelope nil output-time))))))))
              (gf-async/nil-when-all-completed)))
       (->
        (exec-in-outputs-writing-pool
         (fn []
           (d/clone (d/emap div-by-simulations nil burn-count-matrix))))
        (mfd/chain
         (fn [probability-matrix]
           (mfd/zip
            (output-geotiff outputs probability-matrix output-name envelope)
            (when output-pngs?
              (output-png outputs probability-matrix output-name envelope)))))
        (gf-async/nil-when-completed))))))

(defn write-flame-length-sum-layer!
  [{:keys [envelope output-flame-length-sum flame-length-sum-matrix] :as outputs}]
  (when output-flame-length-sum
    (output-geotiff outputs flame-length-sum-matrix "flame_length_sum" envelope)))

(defn write-flame-length-max-layer!
  [{:keys [envelope output-flame-length-max flame-length-max-matrix] :as outputs}]
  (when output-flame-length-max
    (output-geotiff outputs flame-length-max-matrix "flame_length_max" envelope)))

(defn write-burn-count-layer!
  [{:keys [envelope output-burn-count? burn-count-matrix] :as outputs}]
  (when output-burn-count?
    (output-geotiff outputs burn-count-matrix "burn_count" envelope)))

(defn write-spot-count-layer!
  [{:keys [envelope output-spot-count? spot-count-matrix] :as outputs}]
  (when output-spot-count?
    (output-geotiff outputs spot-count-matrix "spot_count" envelope)))

(defn write-aggregate-layers!
  [outputs]
  (->>
    [(write-burn-probability-layer! outputs)
     (write-flame-length-sum-layer! outputs)
     (write-flame-length-max-layer! outputs)
     (write-burn-count-layer! outputs)
     (write-spot-count-layer! outputs)]
    (gf-async/nil-when-all-completed)))

(defn write-csv-outputs!
  [{:keys [output-csvs? output-directory outfile-suffix summary-stats]}]
  (when output-csvs?
    (exec-in-outputs-writing-pool
      (fn []
        (let [output-filename (str "summary_stats" outfile-suffix ".csv")]
          (with-open [out-file (io/writer (if output-directory
                                            (str/join "/" [output-directory output-filename])
                                            output-filename))]
            (->> summary-stats
                 (sort-by #(vector (:ignition-row %) (:ignition-col %)))
```

```
                (mapv (fn [{:keys [ignition-row ignition-col max-runtime temperature relative-humidity
                                   wind-speed-20ft wind-from-direction foliar-moisture ellipse-adjustment-factor
                                   fire-size flame-length-mean flame-length-stddev fire-line-intensity-mean
                                   fire-line-intensity-stddev simulation surface-fire-size crown-fire-size spot-count]}]
                        [simulation
                         ignition-row
                         ignition-col
                         max-runtime
                         temperature
                         relative-humidity
                         wind-speed-20ft
                         wind-from-direction
                         foliar-moisture
                         ellipse-adjustment-factor
                         fire-size
                         flame-length-mean
                         flame-length-stddev
                         fire-line-intensity-mean
                         fire-line-intensity-stddev
                         crown-fire-size
                         spot-count
                         surface-fire-size]))
                (cons ["simulation" "ignition-row" "ignition-col" "max-runtime" "temperature" "relative-humidity"
                       "wind-speed-20ft" "wind-from-direction" "foliar-moisture" "ellipse-adjustment-factor"
                       "fire-size" "flame-length-mean" "flame-length-stddev" "fire-line-intensity-mean"
                       "fire-line-intensity-stddev" "crown-fire-size" "spot-count" "surface-fire-size"])
                (csv/write-csv out-file))))))))
```

9.4.2 gridfire.output-test

```
(ns gridfire.output-test
  (:require [clojure.java.io     :as io]
            [clojure.test        :refer [deftest is use-fixtures]]
            [gridfire.utils.test :as utils]))

;;------------------------------------------------------------------------------
;; Config
;;------------------------------------------------------------------------------

(def resources-path "test/gridfire/resources")

(def db-spec {:classname   "org.postgresql.Driver"
              :subprotocol "postgresql"
              :subname     "//localhost:5432/gridfire_test"
              :user        "gridfire_test"
              :password    "gridfire_test"})

(def test-config-base
  {:db-spec                   db-spec
   :landfire-layers           {:aspect             {:type   :postgis
                                                    :source "landfire.asp WHERE rid=1"}
                               :canopy-base-height {:type   :postgis
                                                    :source "landfire.cbh WHERE rid=1"}
                               :canopy-cover       {:type   :postgis
                                                    :source "landfire.cc WHERE rid=1"}
                               :canopy-height      {:type   :postgis
                                                    :source "landfire.ch WHERE rid=1"}
                               :crown-bulk-density {:type   :postgis
                                                    :source "landfire.cbd WHERE rid=1"}
                               :elevation          {:type   :postgis
                                                    :source "landfire.dem WHERE rid=1"}
                               :fuel-model         {:type   :postgis
                                                    :source "landfire.fbfm40 WHERE rid=1"}
                               :slope              {:type   :postgis
                                                    :source "landfire.slp WHERE rid=1"}}
   :srid                      "CUSTOM:900914"
```

```
   :cell-size                 98.425     ;; (feet)
   :ignition-row              [10 10]
   :ignition-col              [20 20]
   :max-runtime               60         ;; (minutes)
   :temperature               '(50)      ;; (degrees Fahrenheit)
   :relative-humidity         '(1)       ;; (%)
   :wind-speed-20ft           '(10)      ;; (miles/hour)
   :wind-from-direction       '(0)       ;; (degrees clockwise from north)
   :foliar-moisture           90         ;; (%)
   :ellipse-adjustment-factor 1.0        ;; (< 1.0 = more circular, > 1.0 = more elliptical)
   :simulations               1
   :random-seed               1234567890 ;; long value (optional)
   :output-directory          "test/output"})

;;------------------------------------------------------------------------------
;; Fixtures
;;------------------------------------------------------------------------------

(use-fixtures :once utils/with-temp-output-dir)

;;------------------------------------------------------------------------------
;; Tests
;;------------------------------------------------------------------------------

(deftest ^:database output_burn_probability_test
  (let [config (merge test-config-base
                      {:output-burn-probability :final})
        _      (utils/run-gridfire! config)]

    (is (.exists (io/file "test/output/burn_probability.tif")))))

(deftest ^:database output_flame_length_sum_test
  (let [config (merge test-config-base
                      {:output-flame-length-sum? true})
        _      (utils/run-gridfire! config)]

    (is (.exists (io/file "test/output/flame_length_sum.tif")))))

(deftest ^:database output_flame_length_max_test
  (let [config (merge test-config-base
                      {:output-flame-length-max? true})
        _      (utils/run-gridfire! config)]

    (is (.exists (io/file "test/output/flame_length_max.tif")))))

(deftest ^:database output_burn_count_test
  (let [config (merge test-config-base
                      {:output-burn-count? true})
        _      (utils/run-gridfire! config)]

    (is (.exists (io/file "test/output/burn_count.tif")))))

(deftest ^:database output_spot_count_test
  (let [config (merge test-config-base
                      {:spotting           {:mean-distance                1.0
                                            :ws-exp                       1.0
                                            :flin-exp                     1.0
                                            :normalized-distance-variance 1.0
                                            :crown-fire-spotting-percent  1.0
                                            :num-firebrands               1.0
                                            :decay-constant               1.0
                                            :surface-fire-spotting
                                            {:spotting-percent             [[[1 204] [0.001 0.003]]]
                                             :critical-fire-line-intensity 1.0}}
                       :output-spot-count? true})
        _      (utils/run-gridfire! config)]
```

```
    (is (.exists (io/file "test/output/spot_count.tif")))))
```

9.4.3 gridfire.binary-output

```
(ns gridfire.binary-output
  (:require [clojure.java.io :as io]
            [gridfire.common :refer [non-zero-indices non-zero-count]]
            [taoensso.tufte  :as tufte]
            [tech.v3.tensor  :as t])
  (:import (java.io DataInputStream DataOutputStream)
           (java.nio ByteBuffer)))

(set! *unchecked-math* :warn-on-boxed)

;; We assume that matrix[0,0] is the upper left corner.
(defn non-zero-data [matrix]
  (let [[^long rows _] (:shape (t/tensor->dimensions matrix))
        {:keys
         [row-idxs
          col-idxs]} (non-zero-indices matrix)]
    {:x (mapv inc col-idxs)
     :y (mapv #(- rows ^long %) row-idxs)
     :v (mapv (fn [row col] (t/mget matrix row col)) row-idxs col-idxs)}))

;;FIXME max-row max-col is not correct. Need dimensions in binary file.
(defn indices-to-matrix [indices ttype]
  (let [^ints rows   (indices :y)
        ^ints cols   (indices :x)
        ^int max-row (reduce max 0 rows)
        ^int max-col (reduce max 0 cols)]
    (-> (if (= ttype :int)
          (let [^ints values (indices :v)
                matrix       (make-array Integer/TYPE max-row max-col)]
            (dotimes [i (count values)]
              (let [true-row (- max-row (aget rows i))
                    true-col (dec (aget cols i))]
                (aset matrix true-row true-col (aget values i))))
            (t/->tensor matrix))
          (let [^floats values (indices :v)
                matrix         (make-array Float/TYPE max-row max-col)]
            (dotimes [i (count values)]
              (let [true-row (- max-row (aget rows i))
                    true-col (dec (aget cols i))]
                (aset matrix true-row true-col (aget values i))))
            (t/->tensor matrix)))
        ;; Useful in case there is no data.
        (t/reshape [max-row max-col]))))

;;FIXME optimize
(defn write-matrix-as-binary [matrix file-name]
  (let [num-burned-cells (non-zero-count matrix)
        burned-data      (non-zero-data matrix)]
    (with-open [out (DataOutputStream. (io/output-stream file-name))]
      (.writeInt out (int num-burned-cells))               ; Int32
      (doseq [x (burned-data :x)] (.writeInt out (int x))) ; Int32
      (doseq [y (burned-data :y)] (.writeInt out (int y))) ; Int32
      (doseq [v (burned-data :v)] (.writeFloat out (float v)))))) ; Float32

(defn ints-to-bytes [int-coll]
  (let [num-ints (count int-coll)
        buf      (ByteBuffer/allocate (* 4 num-ints))]
    (doseq [i int-coll] (.putInt buf (int i)))
    (.array buf)))

(defn floats-to-bytes [float-coll]
  (let [num-floats (count float-coll)
```

```
        buf        (ByteBuffer/allocate (* 4 num-floats))]
    (doseq [i float-coll] (.putFloat buf (float i)))
    (.array buf)))

(defn write-matrices-as-binary
  [file-name types matrices]
  (tufte/p
   :write-matrices-as-binary
   (let [num-burned-cells (non-zero-count (first matrices))
         data             (mapv non-zero-data matrices)]
     (with-open [out (DataOutputStream. (io/output-stream file-name))]
       (.writeInt out (int num-burned-cells)) ; Int32
       (let [xs (:x (first data))
             ys (:y (first data))]
         (.write out (ints-to-bytes xs) 0 (* 4 (count xs)))  ; Int32
         (.write out (ints-to-bytes ys) 0 (* 4 (count ys)))) ; Int32
       (doseq [[t d] (map vector types data)]
         (when-let [nums-to-bytes (case t
                                    :int   ints-to-bytes
                                    :float floats-to-bytes
                                    nil)]
           (let [vs (:v d)]
             (.write out (nums-to-bytes vs) 0 (* 4 (count vs)))))))))))

;;FIXME optimize
(defn read-matrix-as-binary [file-name]
  (with-open [in (DataInputStream. (io/input-stream file-name))]
    (let [num-burned-cells (.readInt in)] ; Int32
      (indices-to-matrix
       {:x (into-array Integer/TYPE (repeatedly num-burned-cells #(.readInt in))) ; Int32
        :y (into-array Integer/TYPE (repeatedly num-burned-cells #(.readInt in))) ; Int32
        :v (into-array Float/TYPE (repeatedly num-burned-cells #(.readFloat in)))} ; Float32
       :float))))

(defn read-matrices-as-binary [file-name ttypes]
  (with-open [in (DataInputStream. (io/input-stream file-name))]
    (let [num-burned-cells (.readInt in)
          xs-bytes         (byte-array (* 4 num-burned-cells))
          ys-bytes         (byte-array (* 4 num-burned-cells))
          _                (.read in xs-bytes) ; Int32
          _                (.read in ys-bytes)
          xs               (int-array num-burned-cells)
          ys               (int-array num-burned-cells)
          _                (.get (.asIntBuffer (ByteBuffer/wrap xs-bytes)) xs)
          _                (.get (.asIntBuffer (ByteBuffer/wrap ys-bytes)) ys)] ; Int32
      (mapv (fn [ttype]
              (indices-to-matrix
               {:x xs
                :y ys
                :v (let [vs-bytes (byte-array (* 4 num-burned-cells))
                         _        (.read in vs-bytes)]
                     (case ttype
                       :int   (let [vs (int-array num-burned-cells)]
                                (.get (.asIntBuffer (ByteBuffer/wrap vs-bytes)) vs)
                                vs)
                       :float (let [vs (float-array num-burned-cells)]
                                (.get (.asFloatBuffer (ByteBuffer/wrap vs-bytes)) vs)
                                vs)
                       nil))} ; Float32/Int32
               ttype))
            ttypes))))

;; toa_test.bin was created with:
;; (write-matrix-as-binary [[0 1 2] [3 0 4] [5 6 0]] "toa_test.bin")
```

9.4.4 gridfire.binary-output-test

```
(ns gridfire.binary-output-test
  (:require [clojure.test                :refer [deftest is use-fixtures]]
            [gridfire.utils.test         :as utils]
            [gridfire.binary-output      :as binary]
            [tech.v3.datatype.functional :as dfn]
            [tech.v3.tensor              :as t]))

;;-------------------------------------------------------------------------------
;; Fixtures
;;-------------------------------------------------------------------------------

(use-fixtures :once utils/with-temp-output-dir)

;;-------------------------------------------------------------------------------
;; Tests
;;-------------------------------------------------------------------------------

(def binary-file (utils/out-file-path "toa_test.bin"))

(deftest ^:unit write-matrix-as-binary-test
  (let [matrix (t/->tensor [[0.0 1.0 2.0] [3.0 0.0 4.0] [5.0 6.0 0.0]])]
    (binary/write-matrix-as-binary matrix binary-file)
    (is (dfn/equals matrix (binary/read-matrix-as-binary binary-file)))))

(deftest ^:unit write-matrices-as-binary-test
  (let [matrices [(t/->tensor [[0.0 1.0 2.0]
                               [3.0 0.0 4.0]
                               [5.0 6.0 0.0]])

                  (t/->tensor [[0.0 1.0 1.0]
                               [1.0 0.0 1.0]
                               [1.0 1.0 0.0]])

                  (t/->tensor [[0.0 2.0 2.0]
                               [2.0 0.0 2.0]
                               [2.0 2.0 0.0]])

                  (t/->tensor [[0 3 3]
                               [3 0 3]
                               [3 3 0]])]
        _      (binary/write-matrices-as-binary binary-file
                                                [:float :float :float :int]
                                                matrices)
        result (binary/read-matrices-as-binary binary-file [:float :float :float :int])]
    (is (dfn/equals (first matrices) (first result)))

    (is (dfn/equals (second matrices) (second result)))

    (is (dfn/equals (nth matrices 2) (nth result 2)))

    (is (dfn/equals (nth matrices 3) (nth result 3)))))
```

9.5 Helpers for General Programming

9.5.1 gridfire.conversion

```
(ns gridfire.conversion
  (:require [clojure.string   :as str]
            [tech.v3.datatype :as d])
  (:import java.text.SimpleDateFormat
           java.util.TimeZone))

(set! *unchecked-math* :warn-on-boxed)
```

```
(def ^:const square-feet-per-acre 43560.0)

(def ^:const days-per-minute 0.000694444)

(defn F->K
  "Convert fahrenheit to kelvin."
  ^double
  [^double degrees]
  (-> degrees
      (+ 459.67)
      (* 0.5555555555555556)))

(defn K->F
  "Convert kelvin to fahrenheit."
  ^double
  [^double degrees]
  (-> degrees
      (* 1.8)
      (- 459.67)))

(defn F->C
  "Convert fahrenheit to celsius."
  ^double
  [^double degrees]
  (-> degrees
      (- 32.0)
      (* 0.5555555555555556)))

(defn C->F
  "Convert celsius to fahrenheit."
  ^double
  [^double degrees]
  (-> degrees
      (* 1.8)
      (+ 32.0)))

(defn deg->rad
  "Convert degrees to radians."
  ^double
  [^double degrees]
  (Math/toRadians degrees))

(defn rad->deg
  "Convert radians to degrees."
  ^double
  [^double radians]
  (Math/toDegrees radians))

(defn deg->ratio
  "Convert degrees to ratio."
  ^double
  [^double degrees]
  (-> degrees Math/toRadians Math/tan))

(defn ratio->deg
  "Convert ratio to degrees."
  ^double
  [^double ratio]
  (-> ratio Math/atan Math/toDegrees))

(defn m->ft
  "Convert meters to feet."
  ^double
  [^double m]
  (* m 3.281))

(defn ft->m
  "Convert feet to meters."
```

```
  ^double
  [^double ft]
  (* ft 0.30478512648582745))

(defn mph->mps
  "Convert miles per hour to meters per second."
  ^double
  [^double mph]
  (* mph 0.44701818551254696))

(defn mps->mph
  "Convert meters per second to miles per hour."
  ^double
  [^double mps]
  (* mps 2.237045454545455))

(defn mph->km-hr
  "Convert miles per hour to kilometers per hour."
  ^double
  [^double mph]
  (* mph 1.609344))

(defn mph->fpm
  "Convert miles per hour to feet per minute."
  ^double
  [^double mph]
  (* mph 88.0))

(defn km-hr->mph
  "Convert kilometers per hour to miles per hour."
  ^double
  [^double km-hr]
  (* km-hr 0.621371192237334))

(defn wind-speed-20ft->wind-speed-10m
  "Convert wind speed at 20ft to wind speed at 10m."
  ^double
  [^double wind-speed-20ft]
  (/ wind-speed-20ft 0.87))

(defn wind-speed-10m->wind-speed-20ft
  "Convert wind speed at 10m to wind speed at 20ft."
  ^double
  [^double wind-speed-10m]
  (* 0.87 wind-speed-10m))

(defn Btu-ft-s->kW-m
  "Convert BTU per feet per second to kilowatt per meter."
  ^double
  [^double Btu-ft-s]
  (* Btu-ft-s 3.46165186))

(defn kW-m->Btu-ft-s
  "Convert kilowatt per meter to BTU per feet per second."
  ^double
  [^double kW-m]
  (* kW-m 0.28887942532730604))

(defn Btu-lb->kJ-kg
  "Convert BTU per lb to kilojoule per kilogram."
  ^double
  [^double Btu-lb]
  (* Btu-lb 2.3259999996185))

(defn kJ-kg->Btu-lb
  "Convert kilojoule per kilogram to BTU per lb."
  ^double
  [^double kJ-kg]
```

```
  (/ kJ-kg 2.3259999996185))

(defn kg-m3->lb-ft3
  "Convert kilogram per cubic meter to pound per cubic foot."
  ^double
  [^double kg-m3]
  (* kg-m3 0.0624))

(defn lb-ft3->kg-m3
  "Convert pound per cubic foot to kilogram per cubic meter."
  ^double
  [^double lb-ft3]
  (* lb-ft3 16.025641025641026))

(defn percent->dec
  "Convert percent to decimal."
  ^double
  [^double percent]
  (* percent 0.01))

(defn dec->percent
  "Convert decimal to percent."
  ^double
  [^double decimal]
  (* decimal 100.0))

(defn sec->min
  "Convert seconds to minutes."
  ^double
  [^double seconds]
  (* seconds 0.016666666666666666))

(defn min->sec
  "Convert minutes to seconds."
  ^double
  [^double minutes]
  (* minutes 60.0))

(defn ms->min
  "Convert milliseconds to minutes."
  ^double
  [^double milliseconds]
  (* milliseconds 0.000016667))

(defn min->ms
  "Convert minutes to milliseconds."
  ^long
  [^double minutes]
  (long (* minutes 60000.0)))

(defn hour->min
  "Converts hours to minutes."
  ^double
  [^long hours]
  (* hours 60.0))

(defn min->hour
  "Converts minutes to hours. (rounds down)"
  ^long
  [^double minutes]
  (long (/ minutes 60.0)))

(defn min->day
  "Convert minutes to days."
  ^double
  [^double minutes]
  (* minutes days-per-minute))
```

```
(defn convert-date-string
  "Convert a date string between two formats."
  [date-str from-format to-format]
  (let [in-format  (doto (SimpleDateFormat. from-format)
                     (.setTimeZone (TimeZone/getTimeZone "UTC")))
        out-format (doto (SimpleDateFormat. to-format)
                     (.setTimeZone (TimeZone/getTimeZone "UTC")))]
    (->> date-str
         (.parse in-format)
         (.format out-format))))

(defn cells->acres
  "Converts number of cells to acres."
  ^double
  [^double cell-size ^long num-cells]
  (let [acres-per-cell (/ (* cell-size cell-size) square-feet-per-acre)]
    (* acres-per-cell num-cells)))

;; TODO remove when code is in triangulum
(defn camel->kebab
  "Converts camelString to kebab-string."
  [camel-string]
  (as-> camel-string s
    (str/split s #"(?<=[a-z])(?=[A-Z])")
    (map str/lower-case s)
    (str/join "-" s)))

;; TODO remove when code is in triangulum
(defn kebab->camel
  "Converts kebab-string to camelString."
  [kebab-string]
  (let [words (-> kebab-string
                  (str/lower-case)
                  (str/replace #"^[^a-z_$]|[^\w-]" "")
                  (str/split #"-"))]
    (->> (map str/capitalize (rest words))
         (cons (first words))
         (str/join ""))))

(defn snake->kebab
  "Converts snake_string to kebab-string."
  [snake-string]
  (str/replace snake-string #"_" "-"))

(defn kebab->snake
  "Converts kebab-string to snake_string."
  [kebab-string]
  (str/replace kebab-string #"-" "_"))

(def conversion-table
  {:elevation                     {:metric   m->ft}
   :slope                         {nil       deg->ratio}
   :canopy-base-height            {:metric   m->ft}
   :canopy-height                 {:metric   m->ft}
   :crown-bulk-density            {:metric   kg-m3->lb-ft3}
   :temperature                   {:metric   C->F
                                   :absolute K->F}
   :relative-humidity             {:ratio    dec->percent}
   :wind-speed-20ft               {:metric   mps->mph}
   :fuel-moisture-dead-1hr        {:percent  percent->dec}
   :fuel-moisture-dead-10hr       {:percent  percent->dec}
   :fuel-moisture-dead-100hr      {:percent  percent->dec}
   :fuel-moisture-live-herbaceous {:percent  percent->dec}
   :fuel-moisture-live-woody      {:percent  percent->dec}})

(defn valid-multiplier?
  [x]
  (and (number? x) (not= x 1) (not= x 1.0)))
```

```
(defn get-units-converter
  [layer-name units ^double multiplier]
  (if-let [converter (get-in conversion-table [layer-name units])]
    (if (valid-multiplier? multiplier)
      (fn ^double [^double x] (converter (* x multiplier)))
      converter)
    (if (valid-multiplier? multiplier)
      (fn ^double [^double x] (* x multiplier))
      nil)))
```

9.5.2 gridfire.simple-sockets

```
(ns gridfire.simple-sockets
  (:require [clojure.java.io    :as io]
            [clojure.string     :as s]
            [triangulum.logging :refer [log log-str]])
  (:import (java.io BufferedReader)
           (java.net ConnectException Socket ServerSocket)))

(set! *unchecked-math* :warn-on-boxed)

;;=================================
;; Client Socket
;;=================================

(defn send-to-server! [host port message]
  (try
    (with-open [socket (Socket. ^String host ^Integer port)]
      (doto (io/writer socket)
        (.write (-> message
                    (s/trim-newline)
                    (str "\n")))
        (.flush))
      (.shutdownOutput socket)
      true)
    (catch ConnectException _
      (log-str "Connection to " host ":" port " failed!"))))

;;=================================
;; Server Socket
;;=================================

(defonce ^:private global-server-thread (atom nil))
(defonce ^:private global-server-socket (atom nil))

(defn- read-socket! [^Socket socket]
  (.readLine ^BufferedReader (io/reader socket)))

(defn- accept-connections! [^ServerSocket server-socket handler]
  (while @global-server-thread
    (try
      (let [socket (.accept server-socket)]
        (try
          (->> (read-socket! socket)
               (handler))
          (catch Exception e
            (log-str "Server error: " e))
          (finally (.close socket))))
      (catch Exception _))))

(defn stop-server! []
  (if @global-server-thread
    (do
      (reset! global-server-thread nil)
      (when @global-server-socket
```

```
        (.close ^ServerSocket @global-server-socket)
        (reset! global-server-socket nil))
      (log "Server stopped."))
    (log "Server is not running.")))

(defn start-server! [port handler]
  (if @global-server-thread
    (log "Server is already running.")
    (reset! global-server-thread
            (future
              (try
                (with-open [server-socket (ServerSocket. port)]
                  (reset! global-server-socket server-socket)
                  (accept-connections! server-socket handler))
                (catch Exception e
                  (log-str "Error creating server socket: " e)
                  (stop-server!)))))))

#_(start-server! 31337 (fn [msg] :do-something))
```

9.5.3 gridfire.utils.files.tar

```
(ns gridfire.utils.files.tar
  (:require [clojure.java.io :as io])
  (:import (org.apache.commons.compress.compressors.gzip GzipCompressorInputStream)
           (org.apache.commons.compress.archivers.tar TarArchiveInputStream TarArchiveEntry)
           (java.io ByteArrayOutputStream EOFException)))

(defn targz-input-stream
  "Convenience function for obtaining a TarArchiveInputStream from a tar.gz input."
  ^TarArchiveInputStream
  [archive-input]
  (-> archive-input
      (io/input-stream)
      (GzipCompressorInputStream.)
      (TarArchiveInputStream.)))

(defn tar-archive-entries
  "Returns a lazy sequence of [entry-name entry-bytes-array] pairs for each entry
  in the given TAR archive."
  [^TarArchiveInputStream tais]
  (lazy-seq
   (when-some [^TarArchiveEntry my-entry (.getNextTarEntry tais)]
     (try
       (let [entry-name  (.getName my-entry)
             entry-bytes (with-open [os (ByteArrayOutputStream.)]
                           (io/copy tais os)
                           (.toByteArray os))]
         (cons
          [entry-name entry-bytes]
          (tar-archive-entries tais)))
       ;; WARNING sometimes thrown for no apparent reason... (Val, 26 Sep 2022)
       (catch EOFException _eofex
         ())))))
```

9.5.4 gridfire.utils.files

```
(ns gridfire.utils.files
  "File-system utilities for GridFire."
  (:require [clojure.edn     :as edn]
            [clojure.java.io :as io]
            [clojure.walk    :as walk])
  (:import java.io.File))
```

```
(defn- build-absolute-path
  [^String current-dir-path, ^String rel-path]
  (let [^File file (apply io/file current-dir-path (if (string? rel-path)
                                                     [rel-path]
                                                     rel-path))]
    (.getCanonicalPath file)))

(defn location-aware-edn-readers
  "Returns a map of EDN :readers such that
  #gridfire.utils.files/from-this-file ... will work
  and be resolved as paths relative to the directory
  containing the same file."
  [current-file-path]
  (let [current-file     (io/file current-file-path)
        current-dir-path (.getParent current-file)]
    {'gridfire.utils.files/from-this-file                    ; INTRO
     (fn [rel-path]
       (build-absolute-path current-dir-path rel-path))}))

(comment

  (edn/read-string
   {:readers (location-aware-edn-readers (System/getProperty "java.home"))}
   (format " #gridfire.utils.files/from-this-file %s "
           (pr-str "bin/java")))
  ;;"/Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/bin/java"

  *e)

(defn- expand-abbreviations
  [[abbr->v data]]
  (walk/postwalk #(get abbr->v % %)
                 data))

(def convenience-encoding-readers
  {'gridfire.config/abbreviating (fn [abbrv-tuple] (expand-abbreviations abbrv-tuple))})

(comment
  ;; EDN tag #gridfire.config/abbreviating is useful for concisely encoding redundant config: INTRO
  (edn/read-string {:readers convenience-encoding-readers}
                   "#gridfire.config/abbreviating [{m0 {:this :map :is :arguably :quite :verbose} m1 {:so :is :this :one :I :daresa
  ;;=>
  {:my-maps [{:this :map :is :arguably :quite :verbose}
             {:so :is :this :one :I :daresay}
             {:so :is :this :one :I :daresay}
             {:so :is :this :one :I :daresay}
             {:this :map :is :arguably :quite :verbose}
             {:this :map :is :arguably :quite :verbose}
             {:so :is :this :one :I :daresay}]}

  *e)

(defn read-situated-edn-file
  "Slurps and parses an EDN file such that the EDN tag
  #gridfire.utils.files/from-this-file ... will work
  and be resolved as a path relative to the directory
  containing that same file."
  [edn-file]
  (edn/read-string
   {:readers (merge (location-aware-edn-readers edn-file)
                    convenience-encoding-readers)}
   (slurp edn-file)))
```

9.5.5 gridfire.utils.files-test

```
(ns gridfire.utils.files-test
  (:require [clojure.edn          :as edn]
            [clojure.test         :refer [deftest is are testing]]
            [gridfire.utils.files :refer [convenience-encoding-readers]]))

(deftest ^:unit abbreviating-examples-test
  (testing "#gridfire.config/abbreviating eliminates redundancy in EDN-encoded text"
    (testing "transparently to the reading program, for example:"
      (is (= {:my-maps [{:this :map :is :arguably :quite :verbose}
                        {:so :is :this :one :I :daresay}
                        {:so :is :this :one :I :daresay}
                        {:so :is :this :one :I :daresay}
                        {:this :map :is :arguably :quite :verbose}
                        {:this :map :is :arguably :quite :verbose}
                        {:so :is :this :one :I :daresay}]}
             (->> (tagged-literal 'gridfire.config/abbreviating
                                  '[{m0 {:this :map :is :arguably :quite :verbose}
                                     m1 {:so :is :this :one :I :daresay}}
                                    {:my-maps [m0 m1 m1 m1 m0 m0 m1]}])
                  (pr-str)
                  (edn/read-string {:readers convenience-encoding-readers}))))))
    (testing "by using 'abbreviations' as map keys, which can be values of any type, although clojure Symbols are arguably a natura
      (let [repeated-value {:this :map :is :arguably :quite :verbose}]
        (are [abbr]
          (= {:my-maps [repeated-value
                        repeated-value
                        repeated-value]}
             (->> (tagged-literal 'gridfire.config/abbreviating
                                  [{abbr repeated-value}
                                   {:my-maps [abbr
                                              abbr
                                              abbr]}])
                  (pr-str)
                  (edn/read-string {:readers convenience-encoding-readers})))
          'my-abbr
          "my-abbr"
          :my-abbr
          42
          nil
          ["my" "abbr"]
          {:my 'abbr})))
    (testing "by reading a repeated abbreviation as identical values:"
      (let [[v0 v1] (->> (tagged-literal 'gridfire.config/abbreviating
                                         '[{m0 {:this :map :is :arguably :quite :verbose}}
                                           [m0 m0]])
                         (pr-str)
                         (edn/read-string {:readers convenience-encoding-readers}))]
        (is (identical? v0 v1))))))
```

9.5.6 gridfire.utils.async

```
(ns gridfire.utils.async
  (:require [clojure.core.async :as async]
            [manifold.deferred  :as mfd]
            [manifold.executor  :as mexec])
  (:import java.util.concurrent.ExecutorService))

;; ------------------------------------------------------------------------
;; Helper fns

(defn nil-when-completed
  "Returns a Manifold Deferred that will complete when d completes,
  but be resolved with nil instead of the result of d."
  [d]
```

```
  (mfd/chain d (constantly nil)))

(defn nil-when-all-completed
  "Waits for all the given Deferreds to complete, yielding nil.
  Returns immediately a Deferred, resolved with nil
  when all the Deferreds in ds have completed successfully.

  Useful for awaiting effects scheduled in parallel."
  [ds]
  (->> ds
       (apply mfd/zip)
       (nil-when-completed)))

;; ---------------------------------------------------------------------------
;; Executors

(defn executor-with-n-named-threads
  "Creates a fixed-sized thread pool for executing task.
  The Java threads will be named after the supplied prefix string."
  ^ExecutorService
  [thread-name-prefix n-threads exctr-options]
  (let [p-exctr (promise)
        exctr   (mexec/fixed-thread-executor
                  n-threads
                  (merge
                    {:thread-factory (mexec/thread-factory
                                       (let [*next-thread-id (atom -1)]
                                         (fn gen-thread-name []
                                           ;; Naming the threads makes them easy to observe in monitoring
                                           ;; tools such as VisualVM and jstat.
                                           (str thread-name-prefix "-" (swap! *next-thread-id inc))))
                                       p-exctr)}
                    exctr-options))]
    (deliver p-exctr exctr)
    exctr))

;; ---------------------------------------------------------------------------
;; Parallel execution

(defn pmap-in-n-threads
  "Parallel computations of `f` over a finite collection `coll`, with parallelism `n`.

  Returns a Manifold Deferred which will be realized with a value equal to (mapv f coll)."
  [n f coll]
  ;; NOTE it might seem excessive to use both core.async and Manifold here,
  ;; but Manifold has a massive advantage over core.async, which is
  ;; that it naturally lets the caller program in a functional style without side effects,
  ;; which is much more in harmony with the semantics of this function.
  ;; core.async really is nothing more than an implementation detail here,
  ;; and a more polished implementation would probably want to replace it
  ;; to have more control over the thread pool.
  (if (empty? coll)
    (mfd/success-deferred [])
    (let [=sink=  (async/chan (async/dropping-buffer 0))
          =tasks= (async/chan n)
          x+ds    (->> coll (mapv (fn [x] [x (mfd/deferred)])))]
      (async/pipeline-blocking (-> n
                                   (min (count x+ds)))
                               =sink=
                               (map (fn [[x d]]
                                      (try
                                        (mfd/success! d
                                                      (f x))
                                        (catch Exception err
                                          (mfd/error! d err)))
                                      d))
                               =tasks=)
      (async/onto-chan! =tasks= x+ds)
```

```
    (->> x+ds
         (map (fn [[_x d]] d))
         (apply mfd/zip)))))
```

9.6 Helpers for GridFire development

1. gridfire.utils.test

```
(ns gridfire.utils.test
  (:require [clojure.java.io          :as io]
            [clojure.string           :as str]
            [clojure.walk             :as walk]
            [gridfire.core            :as core]
            [gridfire.magellan-bridge :refer [register-custom-projections!]]
            [gridfire.postgis-bridge  :refer [db-pool-cache close-db-pool]])
  (:import (java.io File)))

;;------------------------------------------------------------------------------
;; Config
;;------------------------------------------------------------------------------

(def input-dirname  "test/data/")
(def output-dirname "test/output/")

;;------------------------------------------------------------------------------
;; Utils
;;------------------------------------------------------------------------------

(defn in-file-path [dir filename]
  (str/join "/" [dir filename]))

(defn out-file-path [filename]
  (str output-dirname filename))

(defn make-directory [dirname]
  (.mkdirs (io/file dirname)))

(defn delete-directory [dirname]
  (doseq [^File file (reverse (file-seq (io/file dirname)))]
    (when (.exists file)
      (io/delete-file file))))

(defn run-gridfire! [config]
  (some-> config
          (core/load-inputs!)
          (core/run-simulations!)
          (core/write-outputs!)))

;;------------------------------------------------------------------------------
;; Fixtures
;;------------------------------------------------------------------------------

(defn with-temp-output-dir [test-fn]
  (make-directory output-dirname)
  (test-fn)
  (delete-directory output-dirname))

(defn with-temp-directories [directories]
  (fn [test-fn]
    (doseq [directory directories]
      (make-directory directory))
    (test-fn)
    (doseq [directory directories]
      (delete-directory directory))))

(defn with-reset-db-pool [test-fn]
```

```
  (reset! db-pool-cache nil)
  (test-fn)
  (when-let [db-pool @db-pool-cache]
    (close-db-pool)))

(defn with-register-custom-projections [test-fn]
  (register-custom-projections!)
  (test-fn))

;;-------------------------------------------------------------------------------
;; Expectations
;;-------------------------------------------------------------------------------

(defn round-floating-point
  "Updates the floating-point numbers in a nested data structure by rounding to a certain precision.

  More precisely, each number x is rounded so that (* x precision-factor) is an integer.
  Therefore, to round to 0.1% precision, choose 1000 for precision-factor.

  Useful for making test point robust to slight floating-point variability."
  [precision-factor data]
  (let [pf (double precision-factor)]
    (walk/postwalk (fn round-if-number [x]
                     (if (float? x)
                       (-> x
                           (double)
                           (* pf)
                           (double)
                           (Math/round)
                           (/ pf))
                       x))
                   data)))
```

9.6.1 Debugging and monitoring

1. gridfire.utils.vsampling

```
(ns gridfire.utils.vsampling
  "Values-Sampling: a utility similar to logging or tracing, useful for recording
  (subsets of) the values which occur in our formulas.")

;; You can activate this e.g. with clj -J-Dgridfire.utils.vsampling.enabled="true",
;; or by using the :vsampling CLI alias.
(defn is-enabled?
  []
  (= "true" (System/getProperty "gridfire.utils.vsampling.enabled")))

(def ^:dynamic *current-point-id* nil)

(def ^:dynamic db* (atom {}))

(defn recorded-values
  "Returns the sampled values as a map {point-id (m1 m2 ...)},
  in which each m is a map of the values recorded at one (record ...) call
  (the keys are Strings), most recent first.

  Each map m also has a :gridfire.utils.vsampling/source map
  in its metadata, providing a code location."
  []
  @db*)

(defn clear-recorded!
  []
  (reset! db* {}))

(defn record*
```

```
  [point-id recorded-map]
  (swap! db*
         ;; Those awkwardly nested (fn [...] ...) are for performance. (Val, 12 Oct 2022)
         (fn [db] (update db point-id (fn [rec-maps] (-> rec-maps (or ()) (conj recorded-map)))))))

(defmacro at-point
  "When point-id is not nil, values sampled during the execution of body
   will be recorded as belonging to point-id."
  [point-id & body]
  (if (is-enabled?)
    `(binding [*current-point-id* ~point-id]
       ~@body)
    `(do ~@body)))

(defn keep-hash-bucket
  "Convenience function for selecting m out of N point-ids based on its hash."
  [point-id ^long m ^long N]
  (when (-> (hash point-id) (mod N) (< m))
    point-id))

(defmacro record
  "Samples a consistent set of named values.
  vs must be either a vector of symbols,
  which will be evaluated as a map from symbol name to symbol value,
  or an expression which will evaluate to a map from string to recorded value.

  Does nothing unless *current-point-id* is bound to a non-nil value."
  [vs]
  (when (is-enabled?)
    `(when-some [point-id# *current-point-id*]
       (record* point-id#
                (-> ~(if (vector? vs)
                       (into {}
                             (map (fn [vsym]
                                    [(name vsym) vsym]))
                             vs)
                       vs)
                    (vary-meta merge {::source ~(let [expr-meta (meta &form)
                                                      file-path *file*]
                                                  (merge
                                                   (select-keys expr-meta [:line :column])
                                                   {:file file-path}))}))))))

(comment

  (def is-enabled? (constantly true))                       ; overriding

  (def rec-vals
    (binding [db* (atom {})]                                ; using a throwaway db
      (at-point "my-point-id"
        (let [a 1
              b 2
              c (+ a b)]
          (record [a b c])
          (record {"a - b" (- a b)})
          (recorded-values)))))

  rec-vals
  ;; =>
  {"my-point-id" ({"a - b" -1} {"a" 1, "b" 2, "c" 3})}

  (-> rec-vals
      (get "my-point-id")
      (->> (mapv meta)))
  [#:gridfire.utils.vsampling{:source {:line   8,
                                       :column 11,
                                       :file   "/Users/val/projects/SIG/gridfire/src/gridfire/utils/vsampling.clj"}}
   #:gridfire.utils.vsampling{:source {:line   7,
```

```
                                    :column 11,
                                    :file   "/Users/val/projects/SIG/gridfire/src/gridfire/utils/vsampling.clj"}}]

  *e)
```

9.6.2 Performance optimization

1. gridfire.lab.benchmarks.compare-commits

```
(ns gridfire.lab.benchmarks.compare-commits
  "A babashka script for benchmarking several commits."
  (:require [babashka.process]
            [clojure.java.shell :as sh]
            [clojure.string     :as str]))

(defn- shell-safe
  [& args]
  (let [shell-ret (apply sh/sh args)]
    (if (= 0 (:exit shell-ret))
      (:out shell-ret)
      (throw (ex-info (format "Shell command %s returned error: %s"
                              (pr-str (first args))
                              (:err shell-ret))
                      {::shell-args args
                       ::shell-ret shell-ret})))))

(defn git-checkout!
  [commit-sha]
  (shell-safe "git" "checkout" commit-sha))

(defn benchmark-commits!
  [config-path commit-shas]
  (let [current-commit (-> (shell-safe "git" "rev-parse" "--short" "HEAD") (subs 0 7))]
    (try
      (->> commit-shas
           (run! (fn [commit-sha]
                   (println "------------------------------------------------")
                   (println "Benchmarking at code version:" (shell-safe "git" "show" "--no-patch" "--oneline" commit-sha))
                   (git-checkout! commit-sha)
                   (let [shell-cmd ["clojure" "-J-Dclojure.compiler.direct-linking=true" "-M:perf-testing" "-m" "gridfire.lab
                     ;; Applying Babashka recipe to print the child process' output: https://book.babashka.org/#_babashka_proc
                     @(babashka.process/process (-> shell-cmd (doto prn))
                                                {:inherit true :shutdown babashka.process/destroy-tree}))
                   (println "------------------------------------------------"))))
      (finally
        (git-checkout! current-commit)))))

(defn commits-range
  "Commits after shart-sha (exclusive) until end-sha (inclusive)."
  [start-sha end-sha]
  (->> (shell-safe "git" "log" "--pretty=format:%h" (str start-sha "..." end-sha))
       (str/split-lines)
       (reverse)
       (into [start-sha])))

(comment
  (commits-range "dafdcaa" "5e1eded")
  ["dafdcaa" "c88e8c9" "26c6676" "f2eae21" "d5565da" "34e9720" "d79e18d" "91722c4" "5e1eded"]

  (benchmark-commits! "test/gridfire/lab/benchmarks/rhino-input-deck/gridfire.edn"
                      ["dafdcaa" "c88e8c9" "26c6676" "f2eae21" "d5565da" "34e9720" "d79e18d" "91722c4" "5e1eded"])

  ;; Example:
  ;; bb test/gridfire/lab/benchmarks/compare_commits.clj "test/gridfire/lab/benchmarks/rhino-input-deck/gridfire.edn" "dafdcaa

  *e)
```

```
(defn -main [config-path & commit-shas]
  (benchmark-commits! config-path commit-shas))

;; Applying Babashka recipe: https://book.babashka.org/#main_file
(when (= *file* (System/getProperty "babashka.file"))
  (apply -main *command-line-args*))
```

2. gridfire.lab.benchmarks

```
(set! clojure.core/*assert* false)

(ns gridfire.lab.benchmarks
  (:require [clj-async-profiler.core      :as prof]
            [clojure.java.io              :as io]
            [clojure.pprint               :as pprint]
            [criterium.core               :as bench]
            [gridfire.core                :as gridfire]
            [gridfire.fire-spread         :refer [memoize-rfwo rothermel-fast-wrapper-optimal]]
            [gridfire.simulations         :as simulations]))

(defn simulation-result-summary
  [simulation-results]
  (select-keys simulation-results
               [:exit-condition
                :global-clock
                :surface-fire-count
                :crown-fire-count
                :spot-count]))

(defn- run-simulations!
  [{n-sims :simulations :as inputs}]
  (with-redefs [rothermel-fast-wrapper-optimal (memoize-rfwo rothermel-fast-wrapper-optimal)]
    (->> (range n-sims)
         (mapv (fn [sim-i]
                 (let [simulation-inputs  (simulations/prepare-simulation-inputs sim-i inputs)
                       simulation-results (gridfire.fire-spread/run-fire-spread simulation-inputs)]
                   (simulation-result-summary simulation-results)))))))

(defn- results-digest
  [sims-summaries]
  (->> sims-summaries
       (hash)))

(defn print-context!
  [inputs]
  ;; The digest helps make sure the compared versions have equivalent behavior.
  (println "Results digest:" (results-digest (run-simulations! inputs)))
  (pprint/pprint (bench/os-details))
  (pprint/pprint (into (sorted-map) (bench/runtime-details))))

(defn- benchmark-config!
  [config-file-path]
  (let [inputs (-> config-file-path
                   (gridfire/load-config!)
                   (gridfire/load-inputs!))]
    (print-context! inputs)
    (binding [bench/*report-progress* false
              bench/*report-warn* true
              bench/*report-debug* false]
      (println "Warming up...")
      (dotimes [_ 500] (run-simulations! inputs))
      (println "Measuring...")
      (bench/bench (run-simulations! inputs)
                   ;; With 25 samples,
                   ;; You can get an approximate 95% confidence interval centered around the empirical mean
```

```
                    ;; with radius computed by:
                    ;; 1. dividing the Criterium-reported standard-dev by 5 (5 = 25^0.5)
                    ;; 2. multiplying that by 2 - more precisely by 2.060, see:
                    ;; https://en.wikipedia.org/wiki/Student%27s_t-distribution#Confidence_intervals
                    :samples 25
                    :target-execution-time (* 10 bench/s-to-ns)))))

(defn- profile-config!
  [config-file-path]
  (let [inputs (-> config-file-path
                   (gridfire/load-config!)
                   (gridfire/load-inputs!))]
    (print-context! inputs)
    ;; warmup
    (dotimes [_ 30]
      (run-simulations! inputs))
    (let [flamegraph-file (prof/profile
                           {:return-file true}
                           (dotimes [_ 50]
                             (run-simulations! inputs)))]
      (println "flame graph:")
      (println (str "file://" (.getAbsolutePath (io/file flamegraph-file)))))))

(defn -main [command config-file-path]
  (case command
    "benchmark" (benchmark-config! config-file-path)
    "profile" (profile-config! config-file-path)))

;; Example:
;; £ clojure -J-Dclojure.compiler.direct-linking=true -M:perf-testing -m gridfire.lab.benchmarks benchmark test/gridfire/lab/b
;; £ clojure -J-Dclojure.compiler.direct-linking=true -J-Djdk.attach.allowAttachSelf=true -M:perf-testing -m gridfire.lab.benc
```

3. Benchmarking notes

```
(comment

  (require '[criterium.core :refer [quick-bench]])

  (def test-fuel-model
    {:f_i    {:dead 0.9186991869918698, :live 0.08130081300813012},
     :rho_p
     {:dead {:1hr 32.0, :10hr 32.0, :100hr 32.0, :herbaceous 32.0},
      :live {:herbaceous 32.0, :woody 32.0}},
     :number 101,
     :name   :GR1,
     :sigma
     {:dead {:1hr 2200.0, :10hr 109.0, :100hr 30.0, :herbaceous 2000.0},
      :live {:herbaceous 2000.0, :woody 0.0}},
     :M_x
     {:dead {:1hr 0.15, :10hr 0.15, :100hr 0.15, :herbaceous 0.15},
      :live {:herbaceous 12.539016851465732, :woody 12.539016851465732}},
     :w_o
     {:dead
      {:1hr        0.0046,
       :10hr       0.0,
       :100hr      0.0,
       :herbaceous 0.012266666666666665},
      :live {:herbaceous 0.001533333333333334, :woody 0.0}},
     :S_T
     {:dead
      {:1hr 0.0555, :10hr 0.0555, :100hr 0.0555, :herbaceous 0.0555},
      :live {:herbaceous 0.0555, :woody 0.0555}},
     :h
     {:dead
      {:1hr 8000.0, :10hr 8000.0, :100hr 8000.0, :herbaceous 8000.0},
      :live {:herbaceous 8000.0, :woody 8000.0}},
```

```
   :M_f
   {:dead {:1hr 0.1, :10hr 0.2, :100hr 0.3, :herbaceous 0.1},
    :live {:herbaceous 0.4, :woody 0.5}},
   :delta  0.4,
   :f_ij
   {:dead
    {:1hr        0.2920353982300885,
     :10hr       0.0,
     :100hr      0.0,
     :herbaceous 0.7079646017699115},
    :live {:herbaceous 1.0, :woody 0.0}},
   :g_ij
   {:dead {:1hr 1.0, :10hr 0.0, :100hr 0.0, :herbaceous 1.0},
    :live {:herbaceous 1.0, :woody 0.0}},
   :S_e
   {:dead {:1hr 0.01, :10hr 0.01, :100hr 0.01, :herbaceous 0.01},
    :live {:herbaceous 0.01, :woody 0.01}}})

(defn vectorize-fuel-model [fuel-model]
  (into {}
        (map (fn [[k {:keys [dead live] :as v}]]
               [k (cond (map? dead)    [(dead :1hr)
                                        (dead :10hr)
                                        (dead :100hr)
                                        (dead :herbaceous)
                                        (live :herbaceous)
                                        (live :woody)]
                        (number? dead) [dead live]
                        :else          v)])
             fuel-model)))

(def test-fuel-model-vector
  (vectorize-fuel-model test-fuel-model))

;; 47-63ns
(let [f_ij (:f_ij test-fuel-model-vector)
      S_e  (:S_e test-fuel-model-vector)]
  (quick-bench
   (size-class-sum (fn ^double [i] (* ^double (f_ij i) ^double (S_e i))))))

;; 18-43ns
(let [f_i       [0.1 0.2]
      epsilon_i [0.3 0.4]]
  (quick-bench (category-sum (fn ^double [i] (* ^double (f_i i) ^double (epsilon_i i))))))

;; 47-49ns
(let [M_f [0.1 0.2 0.3 0.1 0.4 0.5]]
  (quick-bench (map-size-class (fn ^double [i] (+ 250.0 (* 1116.0 ^double (M_f i)))))))

;; 92-97ns
(let [S_e_i [0.1 0.1]]
  (quick-bench
   (map-category (fn ^double [i]
                   (let [^double S_e_i (S_e_i i)]
                     (if (pos? S_e_i)
                       (/ 0.174 (Math/pow S_e_i 0.19))
                       1.0))))))

;; 2.21-2.25us
(quick-bench (rothermel-surface-fire-spread-no-wind-no-slope test-fuel-model-vector false))

;; Notes:
;; 1. Stick with longs and doubles. Don't waste time casting to ints and floats.
;; 2. Use Java collections for performance:
;;    - java.util.ArrayList
;;    - java.util.BitSet
;;    - boolean[]
;;    - java.util.PriorityQueue
```

```
  ;;    - primitive arrays (e.g., longs, doubles)

  )
```

#+end_src

9.6.3 gridfire.gen-raster

```
(ns gridfire.gen-raster
  (:require [clojure.string           :as str]
            [clojure.tools.cli        :refer [parse-opts]]
            [gridfire.conversion      :refer [deg->ratio]]
            [gridfire.magellan-bridge :refer [register-custom-projections!]]
            [magellan.core            :refer [write-raster
                                              make-envelope
                                              matrix-to-raster]]))

(set! *unchecked-math* :warn-on-boxed)

(defn- matrix->raster [raster-name matrix {:keys [^long size srid xmin ymax] :or {srid "EPSG:32610"}}]
  (let [width    (count matrix)
        height   (count (first matrix))
        envelope (make-envelope srid
                                xmin
                                ymax
                                (* size width)
                                (* size height))]
    (matrix-to-raster raster-name matrix envelope)))

(defn- identical-matrix [width height value]
  (into-array (repeat height (into-array (repeat width value)))))

(defn- dem-matrix [^long size degrees height width]
  (let [slope (deg->ratio degrees)]
    (into-array (for [^long row (range height)]
                  (into-array (repeat width (* row size slope)))))))

;; FIXME: unused
(defn- generate-dems []
  (for [degree (range 10 60 10)]
    (let [raster (matrix->raster (format "dem-%s-slp" degree)
                                 (dem-matrix 30 degree 256 256)
                                 {:size 30
                                  :xmin 500000.0
                                  :ymax 4000000.0})]
      (write-raster raster (format "test/gridfire/resources/canonical_test/dem-%s-slp.tif" degree)))))

(defn- basename [path]
  (-> path
      (str/split #"/")
      (last)
      (str/split #"\.")
      (first)))

(defn- inputs->output-raster [{:keys [output value width height] :as options}]
  (println "Creating raster with: " options)
  (let [raster (matrix->raster (basename output)
                               (identical-matrix width height value)
                               options)]
    (write-raster raster output)))

;;------------------------------------------------------------------------------
;; Command Line Interface
;;------------------------------------------------------------------------------

(def ^:private cli-options
```

```
  [["-o" "--output OUTPUT" "Raster output filename."
    :validate [#(str/ends-with? % ".tif") "--output must be filename which ends in '.tif'."]]

   ["-v" "--value VALUE" "Value to use to populate the raster. (default: 0.0)."
    :default  0.0
    :parse-fn #(Float/parseFloat %)
    :validate [float? "--value must be a float."]]

   ["-w" "--width WIDTH" "Raster width in pixels (default: 256)."
    :default  256
    :parse-fn #(Integer/parseInt %)
    :validate [integer? "--width must be an integer."]]

   ["-h" "--height HEIGHT" "Raster height in pixels (default: 256)."
    :default  256
    :parse-fn #(Integer/parseInt %)
    :validate [integer? "--height must be an integer."]]

   ["-s" "--size SIZE" "Size of each cell in the units of the provided CRS (default: 30 -- 30 meters in UTM)."
    :default  30
    :parse-fn #(Integer/parseInt %)
    :validate [integer? "--size must be a number."]]

   ["-x" "--xmin XMIN" "Upper left origin XMIN in the units of the provided CRS (default: 500000.0)."
    :default  500000.0
    :parse-fn #(Float/parseFloat %)
    :validate [float? "--xmin must be a number."]]

   ["-y" "--ymax YMAX" "Upper left origin YMAX in the units of the provided CRS (default: 4000000.0)."
    :default  4000000.0
    :parse-fn #(Float/parseFloat %)
    :validate [float? "--ymax must be a number."]]

   ["-r" "--srid SRID" "Spatial Reference ID (default: \"EPSG:32610\")"
    :default  "EPSG:32610"
    :validate [string? "--srid must be a string."]]

   [nil "--help" "Show help."
    :default  false]])

(defn -main
  "Command line tool to produce rasters of constant values. Use `clj -M:gen-raster --help` to view help guide."
  [& args]
  (let [{:keys [options _ summary errors]} (parse-opts args cli-options :strict true)]
    (cond
      (or (empty? args) (:help options))
      (println (str "\nUsage:\n" summary))

      errors
      (do (run! println errors)
          (println (str "\nUsage:\n" summary))
          (System/exit 1))

      (nil? (:output options))
      (do (println "Error: -o/--output is required.")
          (println (str "\nUsage:\n" summary))
          (System/exit 1))

      :else
      (do (println "Creating raster" (:output options) "...")
          (register-custom-projections!)
          (inputs->output-raster options)))
    (System/exit 0)))
```

9.7 Scientific Analyses

9.7.1 gridfire.canonical-test

```
(ns gridfire.canonical-test
  (:require [clojure.edn          :as edn]
            [clojure.java.io      :as io]
            [clojure.data.csv     :as csv]
            [clojure.spec.alpha   :as s]
            [clojure.string       :as str]
            [clojure.test         :refer [deftest is run-tests]]
            [gridfire.core        :as gf]
            [gridfire.spec.config :as spec])
  (:import [java.time LocalDateTime]
           [java.time.format DateTimeFormatter]))

;;; Constants

(def ^:private canonical-dir      "test/gridfire/resources/canonical_test/")
(def ^:private base-config        (edn/read-string (slurp (str canonical-dir "base-config.edn"))))
(def ^:private summary-stats-csv  "summary_stats.csv")
(def ^:private surface-scenarios  {:fuel-model         [:grass-fbfm40 :timber-litter-fbfm40 :grass-extreme-fbfm40 :shrub-fbfm40 :bl
                                   :canopy-cover       [:zero-raster]
                                   :slope              [:zero-raster :slp-10 :slp-20 :slp-30]
                                   :wind-speed-20ft    [0.0 10.0 20.0 40.0]
                                   :fuel-moisture      (range 0.0 0.25 0.05)
                                   :foliar-moisture    [0.0 0.5 1.0]
                                   :canopy-base-height [:zero-raster]
                                   :crown-bulk-density [:zero-raster]})

(def ^:private crowning-scenarios {:fuel-model         [:grass-extreme-fbfm40 :shrub-fbfm40 :blowdown-fbfm40]
                                   :canopy-cover       [:raster-100]
                                   :slope              [:zero-raster]
                                   :wind-speed-20ft    [:ws_20ft-zero_24_zero]
                                   :fuel-moisture      [0.1]
                                   :foliar-moisture    [0.1]
                                   :canopy-base-height [:raster-2 :raster-10 :raster-20 :raster-40]
                                   :crown-bulk-density [:cbd-0005 :cbd-001 :cbd-005 :cbd-01 :cbd-02 :cbd-035 :cbd-05]})

(def ^:private surface-spotting-scenarios {:fuel-model         [:firebreak]
                                           :canopy-cover       [:zero-raster]
                                           :slope              [:zero-raster]
                                           :wind-speed-20ft    [15.0] ; [5.0 10.0 15.0 20.0]
                                           :fuel-moisture      [0.0]
                                           :foliar-moisture    [0.0]
                                           :canopy-base-height [:zero-raster]
                                           :crown-bulk-density [:zero-raster]})

(def ^:private crown-spotting-scenarios {:fuel-model         [:firebreak]
                                         :canopy-cover       [:raster-100]
                                         :slope              [:zero-raster]
                                         :wind-speed-20ft    [15.0] ; [5.0 10.0 15.0 20.0]
                                         :fuel-moisture      [0.0]
                                         :foliar-moisture    [0.0]
                                         :canopy-base-height [:raster-2]
                                         :crown-bulk-density [:raster-100]})

(def ^:private suppression-curve-scenarios {:fuel-model         [:blowdown-fbfm40]
                                            :canopy-cover       [:zero-raster]
                                            :slope              [:slp-10]
                                            :wind-speed-20ft    [0.0]
                                            :fuel-moisture      [0.0]
                                            :foliar-moisture    [0.0]
                                            :canopy-base-height [:zero-raster]
                                            :crown-bulk-density [:zero-raster]})

(def ^:private suppression-sdi-scenarios {:fuel-model                  [:blowdown-fbfm40]
                                          :canopy-cover                [:zero-raster]
```

```
                                        :slope                          [:slp-10]
                                        :wind-speed-20ft                [0.0]
                                        :fuel-moisture                  [0.0]
                                        :foliar-moisture                [0.0]
                                        :canopy-base-height             [:zero-raster]
                                        :crown-bulk-density             [:zero-raster]
                                        :suppression-difficulty-index [:ones-raster]})

;;; Helpers

(defn- now []
  (.format (DateTimeFormatter/ofPattern "yyyy-MM-dd_HH-mm-ss") (LocalDateTime/now)))

(defn- mkdirs [dir]
  (.mkdirs (io/file dir)))

(defn- deep-flatten [x]
  (if (seq? x)
    (mapcat deep-flatten x)
    [x]))

(defn- deep-merge [a & maps]
  (if (map? a)
    (apply merge-with deep-merge a maps)
    (apply merge-with deep-merge maps)))

(defn- ->tif [filekey]
  {:source (str canonical-dir (name filekey) ".tif")
   :type   :geotiff})

(defn- ->dem-tif [slope]
  (case slope
    :zero-raster (assoc (->tif :zero-raster) :units :metric)
    :slp-10      (assoc (->tif :dem-10-slp)  :units :metric)
    :slp-20      (assoc (->tif :dem-20-slp)  :units :metric)
    :slp-30      (assoc (->tif :dem-30-slp)  :units :metric)))

(defn- ->ch [cbh]
  (case cbh
    :zero-raster (->tif :zero-raster)
    :raster-2    (->tif :raster-5)
    :raster-10   (->tif :raster-20)
    :raster-20   (->tif :raster-40)
    :raster-40   (->tif :raster-80)))

(defn- ->fuel-moisture-map
  "Returns a fuel-moisture map given a constant fuel-moisture:
  {:dead {:1hr        (0-1)
          :10hr       (0-1)
          :100hr      (0-1)}
   :live {:herbaceous (0-1)
          :woody      (0-1)}}"
  [fuel-moisture]
  {:dead (zipmap [:1hr :10hr :100hr]  (repeat fuel-moisture))
   :live (zipmap [:woody :herbaceous] (repeat fuel-moisture))})

(defn- dir-name [in]
  (cond
    (float? in)   (-> in (str) (str/replace #"\." "_"))
    (keyword? in) (name in)
    :else         (str in)))

(defn- output-directory [{:keys [base-dir
                                 datetime
                                 scenario-type
                                 fuel-model
                                 canopy-cover
                                 slope
```

```
                                fuel-moisture
                                foliar-moisture
                                wind-speed-20ft
                                canopy-base-height
                                crown-bulk-density]}]
  (str (when base-dir base-dir)
       "outputs"
       "/" datetime
       "/" (dir-name scenario-type)
       "/fuel-model_" (dir-name fuel-model)
       "/slope_" (dir-name slope)
       "/wind-speed-20ft_" (dir-name wind-speed-20ft)
       "/fuel-moisture_" (int (* 100 fuel-moisture))
       "/canopy-cover_" (name canopy-cover)
       "/foliar-moisture_" (int (* 100 foliar-moisture))
       "/canopy-base-height_" (dir-name canopy-base-height)
       "/crown-bulk-density_" (dir-name crown-bulk-density)
       "/"))

(defn- result->row [{:keys [params summary-stats]}]
  (let [stats (first summary-stats)]
    [(-> params :fuel-model dir-name)
     (-> params :slope dir-name)
     (-> params :wind-speed-20ft dir-name)
     (:fuel-moisture params)
     (-> params :canopy-cover dir-name)
     (:foliar-moisture params)
     (-> params :canopy-base-height dir-name)
     (-> params :crown-bulk-density dir-name)
     (:simulation stats)
     (:ignition-row stats)
     (:ignition-col stats)
     (:max-runtime stats)
     (:temperature stats)
     (:relative-humidity stats)
     (:wind-from-direction stats)
     (:ellipse-adjustment-factor stats)
     (:fire-size stats)
     (:flame-length-mean stats)
     (:flame-length-stddev stats)
     (:fire-line-intensity-mean stats)
     (:fire-line-intensity-stddev stats)
     (:crown-fire-size stats)
     (:spot-count stats)]))

(defn- run-sim! [config]
  (try
    (mkdirs (:output-directory config))

    (if (s/valid? ::spec/config config)
      (->> config
           (gf/load-inputs!)
           (gf/run-simulations!)
           (result->row))
      (s/explain ::spec/config config))
    (catch Throwable e
      (println (apply str (.getStackTrace e)))
      (println (.getMessage e)))))

(def csv-header ["fuel-model" "slope" "wind-speed-20ft" "fuel-moisture" "foliar-moisture" "canopy-cover"
                 "canopy-base-height" "canopy-bulk-density" "simulation" "ignition-row" "ignition-col"
                 "max-runtime" "temperature" "relative-humidity" "wind-from-direction"  "ellipse-adjustment-factor"
                 "fire-size" "flame-length-mean" "flame-length-stddev" "fire-line-intensity-mean"
                 "fire-line-intensity-stddev" "crown-fire-size" "spot-count"])

(defn- results->csv [filename results]
  (with-open [out (io/writer filename)]
    (->> results
```

```
        (cons csv-header)
        (csv/write-csv out))))

(defn- gen-scenario
  [{:keys
    [datetime
     fuel-model
     canopy-cover
     slope
     fuel-moisture
     foliar-moisture
     wind-speed-20ft
     canopy-base-height
     crown-bulk-density] :as params}]
  (deep-merge base-config
              {:params           params
               :landfire-layers  {:fuel-model   (->tif fuel-model)
                                  :canopy-cover (->tif canopy-cover)
                                  :slope        (->tif slope)
                                  :elevation    (->dem-tif slope)}
               :fuel-moisture    (->fuel-moisture-map fuel-moisture)
               :foliar-moisture  foliar-moisture
               :wind-speed-20ft  (if (keyword? wind-speed-20ft) (->tif wind-speed-20ft) wind-speed-20ft)
               :output-directory (output-directory (assoc params :datetime datetime))}
              (when-not (some #(= :zero-raster %) [canopy-cover canopy-base-height crown-bulk-density])
                {:landfire-layers {:canopy-base-height (->tif canopy-base-height)
                                   :canopy-height      (->ch canopy-base-height)
                                   :crown-bulk-density (->tif crown-bulk-density)}})))

(defn- gen-scenarios [scenario-type scenarios]
  (deep-flatten
   (let [datetime (now)]
     (for [fuel-model                  (:fuel-model scenarios)
           canopy-cover                (:canopy-cover scenarios)
           slope                       (:slope scenarios)
           fuel-moisture               (:fuel-moisture scenarios)
           foliar-moisture             (:foliar-moisture scenarios)
           wind-speed-20ft             (:wind-speed-20ft scenarios)
           canopy-base-height          (:canopy-base-height scenarios)
           crown-bulk-density          (:crown-bulk-density scenarios)]
       (gen-scenario {:scenario-type               scenario-type
                      :fuel-model                  fuel-model
                      :canopy-base-height          canopy-base-height
                      :canopy-cover                canopy-cover
                      :crown-bulk-density          crown-bulk-density
                      :datetime                    datetime
                      :fuel-moisture               fuel-moisture
                      :foliar-moisture             foliar-moisture
                      :slope                       slope
                      :wind-speed-20ft             wind-speed-20ft})))))

;;; Tests

(defn run-test-scenario! [{:keys [params] :as scenario}]
  (let [control-dir   (output-directory (assoc params :base-dir canonical-dir))
        control-stats (str control-dir summary-stats-csv)
        new-stats     (str (:output-directory scenario) summary-stats-csv)]
    (run-sim! scenario)
    (is (= (slurp control-stats)
           (slurp new-stats)))))

(deftest ^{:surface true :canonical true} test-surface-scenarios
  (let [test-file (str "test-surface-"(now)".csv")]
    (->> (gen-scenarios :surface surface-scenarios)
         ;(take 1) ; Uncomment me to run only 1 simulation.
         (pmap run-sim!)
         (results->csv test-file))))
```

```
(deftest ^{:crowning true :canonical true} test-crowning-scenarios
  (let [test-file (str "test-crowning-"(now)".csv")]
    (results->csv test-file (pmap run-sim! (gen-scenarios :crowning crowning-scenarios)))))

(deftest ^{:surface-spotting true :canonical true} test-surface-spotting-scenarios
  (let [test-file (str "test-surface-spotting-"(now)".csv")]
    (results->csv test-file (map run-sim! (gen-scenarios :surface-spotting surface-spotting-scenarios)))))

(deftest ^{:crown-spotting true :canonical true} test-crown-spotting-scenarios
  (let [test-file (str "test-crown-spotting-"(now)".csv")]
    (results->csv test-file (map run-sim! (gen-scenarios :crown-spotting crown-spotting-scenarios)))))
(comment (clojure.test/test-vars [#'test-surface-spotting-scenarios #'test-crown-spotting-scenarios]))
#_(deftest ^{:crown-spotting true :canonical true} test-crown-spotting-scenario
    (let [test-file (str "test-crown-spotting-"(now)".csv")]
      (run-sim! (first (gen-scenarios :spotting crown-spotting-scenarios)))))

(deftest ^{:suppression-curve true :simulation true :canonical true} test-suppression-scenario
  (run-sim! (-> (first (gen-scenarios :suppression-curve suppression-curve-scenarios))
                (assoc :output-layers {:directional-flame-length 72
                                       :flame-length             :final})
                (assoc :suppression {:suppression-dt          300.0
                                     :suppression-coefficient 2.0})
                (assoc :weather-start-timestamp #inst "1970-01-01T00-00:00"))))

(deftest ^{:suppression-sdi true :simulation true :canonical true} test-suppression-sdi-scenario
  (run-sim! (-> (first (gen-scenarios :suppression-sdi suppression-sdi-scenarios))
                (assoc :output-layers {:directional-flame-length 72
                                       :flame-length             :final})
                (update-in [:suppression]
                           #(merge % {:suppression-dt                                  300.0
                                      :sdi-layer                                       (->tif :zero-raster)
                                      :sdi-sensitivity-to-difficulty                   2.0
                                      :sdi-containment-overwhelming-area-growth-rate 50000
                                      :sdi-reference-suppression-speed                 800}))
                (assoc :weather-start-timestamp #inst "1970-01-01T00-00:00"))))

(comment
  ;; TIP: this is how you run only a subset of test cases. (Val, 03 Nov 2022)
  (clojure.test/test-vars [#'test-surface-spotting-scenarios
                           #'test-crown-spotting-scenarios
                           #'test-suppression-scenario
                           #'test-suppression-sdi-scenario])

  (run-tests 'gridfire.canonical-test))
```

9.7.2 gridfire.lab.replay

```
;; FIXME LP coverage ???
(ns gridfire.lab.replay
  (:require [clojure.java.io              :as io]
            [clojure.java.shell           :as sh]
            [clojure.pprint               :as pprint]
            [clojure.string               :as str]
            [gridfire.core                :as gridfire]
            [gridfire.fire-spread         :refer [memoize-rfwo rothermel-fast-wrapper-optimal run-fire-spread]]
            [gridfire.magellan-bridge     :refer [geotiff-raster-to-tensor]]
            [gridfire.simulations         :as simulations]
            [gridfire.utils.files.tar     :as utar]
            [gridfire.utils.vsampling     :as vsmpl]
            [hiccup.page                  :as html]
            [matrix-viz.core              :refer [save-matrix-as-png]]
            [tech.v3.datatype             :as d]
            [tech.v3.datatype.functional  :as dfn]
            [tech.v3.datatype.statistics  :as dstats]
            [tech.v3.tensor               :as t])
  (:import (java.io File)
```

```
           (java.nio.file Files)
           (java.nio.file.attribute FileAttribute)
           (java.time ZonedDateTime)
           (java.time.format DateTimeFormatter)
           (java.util Date)))

;;;; How this was set up:

;; How I installed Java and Clojure:
;;sudo apt install openjdk-17-jdk
;;sudo apt install rlwrap
;;curl -O https://download.clojure.org/install/linux-install-1.11.1.1165.sh
;;chmod +x linux-install-1.11.1.1165.sh
;;sudo ./linux-install-1.11.1.1165.sh

;; How I sync the code from my laptop to the remote VM, using Git:
;;vwaeselynck@iiwi:~£ mkdir gridfire.git
;;vwaeselynck@iiwi:~£ cd gridfire.git/
;;vwaeselynck@iiwi:~/gridfire.git£ git init --bare
;;£ git push sigum --all
;;vwaeselynck@iiwi:~£ git clone gridfire.git/
;;Cloning into 'gridfire'...
;;vwaeselynck@iiwi:~/gridfire£ nohup clj -J-XX:MaxRAMPercentage=90 -M:nREPL:mydev:gridfire-my-dev:vsampling:dataviz -m nrepl.cmdline
;;vwaeselynck@iiwi:~£ cd gridfire
;;vwaeselynck@iiwi:~/gridfire£ git checkout vwaeselynck-330-gridfire-historical-fires

;; How I start and connect to the remote nREPL server
;;vwaeselynck@iiwi:~/gridfire-scripts/lab£ nohup clj -J-XX:MaxRAMPercentage=90 -J-Djdk.attach.allowAttachSelf=true -M:nREPL:mydev:gr
;;£ ssh -i ~/.ssh/id_rsa -4N -L 18888:localhost:8888 vwaeselynck@10.1.30.147

;; How I fetch the FTP-saved PyreCast snapshots:
;; vwaeselynck@iiwi:~/pyrecast-snapshots-2022-09-30£ wget --timestamping ftps://vwaeselynck:5Km_32erTf@ftp.sig-gis.com/kcheung/*.gz

;; ------------------------------------------------------------------------
;; File and CLI utilities

;; IMPROVEMENT move those to generic ns?
(defn sh-safe
  [& args]
  (let [ret (apply sh/sh args)]
    (if (not= 0 (:exit ret))
      (let [err-string (or (not-empty (:err ret)) (:out ret))]
        (throw (ex-info
                (format "Error running sh command: %s" err-string)
                (merge
                 {:sh-args (vec args)}
                 ret))))
      ret)))

(defn file-name
  [^File file]
  (.getName file))

(defn file-abs-path
  ^String [f]
  (.getCanonicalPath (io/file f)))

(defn ensure-file
  [f]
  (let [ret (io/file f)]
    (when-not (.exists ret)
      (throw (ex-info (format "File does not exist: %s" (.getPath ret))
                      {:file  ret
                       :f-arg f})))
    ret))

(defn delete-dir-recursively!
  [file]
```

```
  (->> file (file-seq) (reverse) (run! io/delete-file)))

(defn with-tmp-dirs*
  ([prefixes callback-fn] (with-tmp-dirs* nil prefixes callback-fn))
  ([containing-dir prefixes callback-fn]
   (let [tmp-dirs
         (->> prefixes
              (mapv (fn create-tmp-dir [^String prefix]
                      (.toFile
                       (let [empty-fileattrs-array (into-array FileAttribute [])]
                         (if (nil? containing-dir)
                           (Files/createTempDirectory prefix empty-fileattrs-array)
                           (Files/createTempDirectory (-> (io/file containing-dir)
                                                          (doto (.mkdirs))
                                                          (.toPath))
                                                      prefix
                                                      empty-fileattrs-array)))))))]
     (try
       (apply callback-fn tmp-dirs)
       (finally
         (run! delete-dir-recursively! tmp-dirs))))))

(defn archive-entries->files!
  "Saves archive entries as files in a directory, optionally returning a subset of the created files in a map.

  Given:
  - target-dir: the directory into which to write the files,
  - return-key-fn: a function,
  - archive-entries: a sequence of [^String entry-name ^bytes entry-bytes] pairs,

  will create one file or directory per entry under target-dir,
  at relative path given by entry-name, and content entry-bytes.
  Returns a map; whenever (return-key-fn entry-name) returns a non-nil value entry-k,
  the returned map will contain the created java.io.File at key entry-k."
  [target-dir return-key-fn archive-entries]
  (reduce
   (fn write-entry-to-file! [returned-files [entry-name entry-bytes]]
     (let [entry-file (io/file target-dir entry-name)]
       (io/make-parents entry-file)
       (let [is-dir? (str/ends-with? entry-name "/")]
         (if is-dir?
           (.mkdir entry-file)
           (io/copy entry-bytes entry-file)))
       (merge returned-files
              (when-some [ret-k (return-key-fn entry-name)]
                {ret-k entry-file}))))
   {}
   archive-entries))



;; ------------------------------------------------------------------------
;; Extracting PyreCast snapshots

(defn parse-pyrecast-snapshot-archive-name
  [^File input-deck-targz]
  (let [file-name  (.getName input-deck-targz)
        name-cpnts (re-matches #"(.+)_(\d{8})_(\d{6})_(\d{3})\.tar\.gz" file-name)]
    (if (nil? name-cpnts)
      (throw (ex-info (format "Failed to parse PyreCast snapshot filename: %s" (pr-str file-name))
                      {:pyrcst_snapshot_archive_name file-name}))
      (let [[_ fire-name fire-date-str fire-time-str subnum] name-cpnts
            snap {; INTRO denotes a map holding information about a PyreCast snapshot.
                  :pyrcst_snapshot_archive_name file-name
                  :pyrcst_fire_name             fire-name
                  :pyrcst_fire_subnumber        (Long/parseLong subnum 10)
                  :pyrcst_snapshot_inst         (-> (str fire-date-str " " fire-time-str " Z")
                                                    (ZonedDateTime/parse (DateTimeFormatter/ofPattern "yyyyMMdd HHmmss z"))
```

```
                                                (.toInstant)
                                                (Date/from))
                  :pyrcst_snapshot_archive      input-deck-targz}]
        snap))))

(defn list-fire-snapshots-from-dir
  [pyrecast-snapshots-dir]
  (let [decks-dir (ensure-file pyrecast-snapshots-dir)]
    (->> (file-seq decks-dir)
         (filter #(-> (file-name %) (str/ends-with? ".tar.gz")))
         (map parse-pyrecast-snapshot-archive-name)
         (sort-by (juxt :pyrcst_fire_name :pyrcst_fire_subnumber :pyrcst_snapshot_inst))
         (vec))))

(defn rewrite-gridfire-config-with-relative-paths
  [^String gridfire-edn-contents ^String absolute-path-prefix]
  (str/replace gridfire-edn-contents (str "\"" absolute-path-prefix "/") "#gridfire.utils.files/from-this-file \""))

(defn sanitize-pyrecast-snapshot-entries
  "Filters, renames and rewrites the TAR entries for a PyreCast snapshot
  into a locally-runnable input deck.

  Accepts and returns a sequence of [entry-name entry-bytes] pairs."
  [snap-entries]
  (let [tar-entry-prefix   "home/gridfire_data/"
        config-path-prefix "/home/gridfire/data/"]
    (->> snap-entries
         (map (fn drop-prefix-path [[entry-name entry-bytes]]
                [(subs entry-name (count tar-entry-prefix))
                 entry-bytes]))
         (remove (fn is-unneeded-file? [[entry-name _entry-bytes]]
                   (let [[_ _snapshot-name rel-path] (re-matches #"([^\/]+)\/(.*)" entry-name)]
                     (re-matches #"outputs/.+" rel-path))))
         (map (fn repath-gridfire-edn [entry-pair]
                (let [[entry-name entry-bytes] entry-pair
                      [_ snapshot-name _rel-path] (re-matches #"([^\/]+)\/(.*)" entry-name)]
                  (if (str/ends-with? entry-name "/gridfire.edn")
                    [entry-name
                     (-> (String. (bytes entry-bytes) "UTF-8")
                         (rewrite-gridfire-config-with-relative-paths (str config-path-prefix snapshot-name))
                         (.getBytes "UTF-8"))]
                    entry-pair)))))))

(defn interesting-file-key
  [entry-name]
  (cond
    (str/ends-with? entry-name "/gridfire.edn")
    ::gridfire-config-file

    (str/ends-with? entry-name "/active_fire_polygon_utm_posnegbuff.shp")
    ::active_fire_polygon_utm_posnegbuff-shp

    (str/ends-with? entry-name "/fuels_and_topography/already_burned.tif")
    ::already_burned-tif))


(defn extract-pyrecast-snapshots!
  [pyrecast-snapshots-dir]
  (let [tmp-dir (.toFile (Files/createTempDirectory (-> (io/file "../tmp-gridfire-replay") (doto (.mkdirs)) (.toPath) (.toAbsoluteP
                                                    "extracted-snapshots"
                                                    (into-array FileAttribute []))))]
    (->> (list-fire-snapshots-from-dir pyrecast-snapshots-dir)
         (pmap
          (fn extract-snapshot! [snap]
            (merge
             snap
             {::useful-files
              (with-open [tais (utar/targz-input-stream (:pyrcst_snapshot_archive snap))]
```

```
                (->> (utar/tar-archive-entries tais)
                     (sanitize-pyrecast-snapshot-entries)
                     (archive-entries->files! tmp-dir interesting-file-key)))}))))
         (vec))))


;; ------------------------------------------------------------------------
;; Comparing observed and simulated burn areas

(defn config-max-runtime-until-stop-inst
  [inputs stop-inst]
  (assoc inputs :max-runtime (/
                              (-
                               (inst-ms stop-inst)
                               (->> [:ignition-start-timestamp
                                     :weather-start-timestamp]
                                    (keep #(get inputs %))
                                    (map inst-ms)
                                    (apply min)))
                              (* 1e3 60))))

(defn run-fire-spread-until
  [config stop-inst variation-name->info]
  (some-> config
          (config-max-runtime-until-stop-inst stop-inst)
          (-> (gridfire/load-inputs!))
          (as-> inputs
                (->> variation-name->info
                     (partition-all 16)
                     (mapcat
                      (fn compute-variations-batch [variation-name->info1]
                        (->> variation-name->info1
                             (pmap (fn compute-variation-results-entry [[variation-name variation-info]]
                                     [variation-name
                                      (binding [vsmpl/db* (atom {})]
                                        (let [transform-inputs-fn (-> variation-info ::transform-inputs-fn (or identity))
                                              inputs              (transform-inputs-fn inputs)]
                                          (merge
                                           {::simulations/inputs inputs
                                            ::simulations/result
                                            (-> (simulations/prepare-simulation-inputs 0 inputs)
                                                (run-fire-spread))}
                                           {::sampled-values (vsmpl/recorded-values)})))]))
                             (vec))))
                     (into {}))))))

;; IMPROVEMENT re-implement as t1-active_fire_polygon_utm_posnegbuff-shp - t0-active_fire_polygon_utm_posnegbuff-shp
(defn resolve-observed-burned-area
  "Computes the observed burned area between 2 PyreCast snapshots (from t0 to t1),
  as the set difference between active-fire-at-t1 and already-burned-at-t0.

  Accepts files and returns a GIS-enveloped matrix."
  [t0-already_burned-tif t1-active_fire_polygon_utm_posnegbuff-shp]
  (with-tmp-dirs*
   ["tmp_resolve-observed-burned-area"]
   (fn [tmp-dir]
     (let [t0-already_burned-tif                    (ensure-file t0-already_burned-tif)
           ;; NOTE 'posnegbuff' hints at the algorithm used for approximating the perimeter from the original hot-spots polygons,
           ;; which tend to paint a dotted landscape; a succession of buffering and negative buffering is used to approximate
           ;; the perimeter from these polygons. See Elmfire script 01-setup_run.sh in input decks. (Val, 29 Sep 2022)
           t1-active_fire_polygon_utm_posnegbuff-shp (ensure-file t1-active_fire_polygon_utm_posnegbuff-shp)
           t1-active_fire_posnegbuff-tif             (io/file tmp-dir "t1-active_fire_polygon_utm_posnegbuff.tif")
           expected-burned-area-tif                  (io/file tmp-dir "expected-burned-area.tif")]
       ;; NOTE I chose GDAL over PostGis for these computations, (Val, 29 Sep 2022)
       ;; because I judged that the GDAL executables are easier to take for granted
       ;; than a running Postgres server.
       ;; Writing a GeoTiff with zeroes
       (sh-safe "gdal_calc.py" "-A" (file-abs-path t0-already_burned-tif)
```

```
                "--NoDataValue=255"
                "--type=Byte"
                "--calc=A*0"
                "--overwrite"
                (str "--outfile=" (file-abs-path t1-active_fire_posnegbuff-tif)))
        ;; Burning the polygon into it
        (sh-safe "gdal_rasterize" "-burn" "1" "-at"
                 (file-abs-path t1-active_fire_polygon_utm_posnegbuff-shp)
                 (file-abs-path t1-active_fire_posnegbuff-tif))
        ;; Computing the expected burned area
        (sh-safe "gdal_calc.py"
                 "-A" (file-abs-path t0-already_burned-tif)
                 "-B" (file-abs-path t1-active_fire_posnegbuff-tif)
                 "--NoDataValue=255"
                 "--type=Byte"
                 "--calc=(A==0)*(B==1)"                      ; set difference
                 (str "--outfile=" (file-abs-path expected-burned-area-tif)))
        (geotiff-raster-to-tensor (file-abs-path expected-burned-area-tif) :float32 identity)))))

(defn simulated-burned-area-matrix
  [stop-inst sim-inputs sim-result]
  (let [minutes-until-stop-inst (/ (-
                                    (inst-ms stop-inst)
                                    (inst-ms (first (:ignition-start-timestamps sim-inputs))))
                                   (* 1e3 60))]
    (dfn/max                                                 ; set union
     (:ignition-matrix sim-inputs)
     (d/elemwise-cast
      (dfn/<=
       0.
       (:burn-time-matrix sim-result)
       minutes-until-stop-inst)
      :double))))

(defn fetch-snapshot-config
  [snap]
  (-> (gridfire/load-config! (-> snap
                                 ::useful-files
                                 ::gridfire-config-file
                                 (ensure-file)
                                 (.getCanonicalPath)))
      (into (sorted-map))))

(defn replay-snapshot
  [t0-snap t1-snap variation-name->info]
  (let [stop-inst          (:pyrcst_snapshot_inst t1-snap)
        observed-burn-area (resolve-observed-burned-area
                            (-> t0-snap ::useful-files ::already_burned-tif)
                            (-> t1-snap ::useful-files ::active_fire_polygon_utm_posnegbuff-shp))

        config             (-> (fetch-snapshot-config t0-snap)
                               (dissoc :perturbations)       ; to remove variance.
                               (merge {:simulations 1}))
        sim-outputs        (run-fire-spread-until config stop-inst variation-name->info)]
    {::observed-burn-area observed-burn-area
     ::sim-outputs        sim-outputs}))

(defn burn-trace-stats
  [replay-res sim-output]
  (let [stop-inst            (-> replay-res ::t1-fire :pyrcst_snapshot_inst)
        observed-burn-area   (::observed-burn-area replay-res)
        observed-burn-matrix (:matrix observed-burn-area)
        sim-result           (::simulations/result sim-output)]
    (if (nil? sim-result)
      {:observed-burn-n-cells    (dfn/sum observed-burn-matrix)
       :sim-inter-obs-n-cells    0
       :sim-minus-obs-n-cells    0
       :obs-minus-sim-n-cells    (dfn/sum observed-burn-matrix)
```

```
       :sim-inter-obs-n-crowning 0
       :sim-minus-obs-n-crowning 0
       ::comments                ["run-fire-spread returned nil, which happens when it fails to get perimeter cells with burnable n
      (let [sim-burn-matrix      (simulated-burned-area-matrix stop-inst (::simulations/inputs sim-output) sim-result)
            sim-inter-obs-matrix (dfn/* sim-burn-matrix observed-burn-matrix)
            sim-minus-obs-matrix (dfn/* sim-burn-matrix (dfn/- 1.0 observed-burn-matrix))
            did-crown-matrix     (dfn/> (:fire-type-matrix sim-result) 1.0)]
        {:observed-burn-n-cells    (dfn/sum observed-burn-matrix)
         :sim-inter-obs-n-cells    (dfn/sum sim-inter-obs-matrix)
         :sim-minus-obs-n-cells    (dfn/sum sim-minus-obs-matrix)
         :obs-minus-sim-n-cells    (dfn/sum (dfn/* observed-burn-matrix (dfn/- 1.0 sim-burn-matrix)))

         :sim-inter-obs-n-crowning (dfn/sum (dfn/* sim-inter-obs-matrix did-crown-matrix))
         :sim-minus-obs-n-crowning (dfn/sum (dfn/* sim-minus-obs-matrix did-crown-matrix))}))))

(defn can-replay-t0-to-t1?
  [t0-snap t1-snap]
  (and
   (and
    ;; Of course, both snapshots should be of the same fire...
    (= (:pyrcst_fire_name t0-snap) (:pyrcst_fire_name t1-snap))
    (= (:pyrcst_fire_subnumber t0-snap) (:pyrcst_fire_subnumber t1-snap)))
   (and
    ;; ... we also want them to have the required files...
    (-> t0-snap ::useful-files ::already_burned-tif (some?))
    (-> t0-snap ::useful-files ::gridfire-config-file (some?))
    (-> t1-snap ::useful-files ::active_fire_polygon_utm_posnegbuff-shp (some?)))
   ;; ... we want the t1 snapshot some time after t0, neither too long nor too short:
   (let [time-lapse-ms (- (-> t1-snap :pyrcst_snapshot_inst (inst-ms))
                          (-> t0-snap :pyrcst_snapshot_inst (inst-ms)))
         ms-3hr        (* 1000 60 60)
         ms-3days      (* 1000 60 60 24 3)]
     (<= ms-3hr
         ;; A too-short time lapse makes us more sensitive to inaccuracies
         ;; in estimating the real burned area.
         time-lapse-ms
         ;; short time lapses seem to be the most important to get right:
         ;; short-term accuracy will probably tend to imply long-term accuracy,
         ;; but the converse is not true.
         ;; What's more, if we let too many days elapse, the weather data
         ;; in t0-snap will be either missing or inaccurate.
         ms-3days))))

;; IMPROVEMENT rather than successive pairs
;; let's use pairs that achieve a delta-t closest to 24hr,
;; or some other optimal time lapse.
(defn replayable-successive-t0t1-pairs
  [snaps]
  (->> snaps
       (sort-by (juxt :pyrcst_fire_name :pyrcst_fire_subnumber :pyrcst_snapshot_inst))
       (partition-all 2 1)
       (filter
        (fn [[t0-snap t1-snap]]
          (can-replay-t0-to-t1? t0-snap t1-snap)))))

(defn replay-snapshot-pairs
  [variation-name->info t0+t1-snaps]
  (with-redefs [rothermel-fast-wrapper-optimal (memoize-rfwo rothermel-fast-wrapper-optimal)]
    (->> t0+t1-snaps
         (partition-all 4)                                ; parallelism
         (mapcat
          (fn [t0+t1-snaps]
            (->> t0+t1-snaps
                 (pmap (fn compare-snapshots [[t0-snap t1-snap]]
                         (println (print-str "Replaying" (:pyrcst_fire_name t0-snap) "from" (-> t0-snap :pyrcst_snapshot_inst) "to"
                         (merge
                          (select-keys t0-snap [:pyrcst_fire_name :pyrcst_fire_subnumber])
                          {::t0-fire (select-keys t0-snap [:pyrcst_snapshot_inst ::useful-files])
```

```
                          ::t1-fire (select-keys t1-snap [:pyrcst_snapshot_inst ::useful-files])}
                         (replay-snapshot t0-snap t1-snap variation-name->info))))
                (vec)))))))

(defn snapshot-pair-hash
  [[t0-snap t1-snap]]
  (hash
   [(select-keys t0-snap [:pyrcst_fire_name :pyrcst_fire_subnumber :pyrcst_snapshot_inst])
    (select-keys t1-snap [:pyrcst_fire_name :pyrcst_fire_subnumber :pyrcst_snapshot_inst])]))

(defn replay-snapshots
  ([variation-name->info snaps] (replay-snapshots variation-name->info nil snaps))
  ([variation-name->info subsample-size snaps]
   (replay-snapshot-pairs variation-name->info
                          (-> (replayable-successive-t0t1-pairs snaps)
                              (cond-> (some? subsample-size) (->> (sort-by snapshot-pair-hash) (take subsample-size)))))))

(defn toa-stats
  [replay-res sim-output]
  (let [stop-inst                  (-> replay-res ::t1-fire :pyrcst_snapshot_inst)
        observed-burn-area         (::observed-burn-area replay-res)
        observed-burn-matrix       (:matrix observed-burn-area)
        sim-result                 (::simulations/result sim-output)]
    (when-not (nil? sim-result)
      (let [sim-burn-matrix       (simulated-burned-area-matrix stop-inst (::simulations/inputs sim-output) sim-result)
            sim-inter-real-matrix (dfn/* sim-burn-matrix observed-burn-matrix)
            time-lapse-min        (-> (- (-> replay-res ::t1-fire :pyrcst_snapshot_inst (inst-ms))
                                         (-> replay-res ::t0-fire :pyrcst_snapshot_inst (inst-ms)))
                                      (double)
                                      (/ (* 1000 60)))
            toa-ratio-matrix      (d/emap
                                   (fn [burn-time sim-inter-real]
                                     (if (and (= 1.0 sim-inter-real) (> burn-time 0.))
                                       (/ burn-time
                                          time-lapse-min)
                                       Double/NaN))
                                   :double
                                   (:burn-time-matrix sim-result)
                                   sim-inter-real-matrix)
            stats-opts            {:nan-strategy :remove}
            [p50 p75 p90 p95]     (dstats/percentiles [50 75 90 95] stats-opts toa-ratio-matrix)]
        {::toa-ratio-mean (dstats/mean toa-ratio-matrix stats-opts)
         ::toa-ratio-p50  p50
         ::toa-ratio-p75  p75
         ::toa-ratio-p90  p90
         ::toa-ratio-p95  p95})))))

(defn all-stats
  [replay-res sim-output]
  (merge (burn-trace-stats replay-res sim-output)
         (toa-stats replay-res sim-output)))

(defn pprint-table-from-kv-lists
  ([pairs-vecs]
   (let [pairs0 (first pairs-vecs)
         cols   (mapv first pairs0)]
     (pprint-table-from-kv-lists cols pairs-vecs)))
  ([cols pairs-vecs]
   (pprint/print-table
    cols
    (map #(into {} %) pairs-vecs))))

(defn format-hours-between
  [t0-inst t1-inst]
  (format "+%2dhr"
          (-> (-
               (-> t1-inst (inst-ms))
               (-> t0-inst (inst-ms)))
```

```
              (/ (* 1e3 60 60))
              (double)
              (Math/round) (long))))

(defn replayed-results-stats
  [replay-results]
  (->> replay-results
       (pmap (fn [replay-res]
               (merge (select-keys replay-res [:pyrcst_fire_name])
                      {::t0-fire               (select-keys (::t0-fire replay-res) [:pyrcst_snapshot_inst])
                       ::t1-fire               (select-keys (::t1-fire replay-res) [:pyrcst_snapshot_inst])
                       ::variation-name->stats (->> replay-res ::sim-outputs
                                                    (sort-by key)
                                                    (map
                                                     (fn [[variation-name sim-output]]
                                                       [variation-name (all-stats replay-res sim-output)]))
                                                    (into (sorted-map)))})))))

(defn replayed-results-table-entries
  [rr-stats]
  (->> rr-stats
       (mapcat (fn [replay-res-stats]
                 (->> replay-res-stats ::variation-name->stats
                      (sort-by key)
                      (mapv
                       (fn [[variation-name stats]]
                         (let [n-cells-really-burned (long (:observed-burn-n-cells stats))]
                           (letfn [(format-w-pct
                                     ([^long n-cells] (format-w-pct n-cells n-cells-really-burned))
                                     ([^long n-cells ^long n-cells-denom]
                                      (if (zero? n-cells-denom)
                                        (format "%d (-)" n-cells)
                                        (format "%d (%2.1f%%)"
                                                n-cells
                                                (* 100.0 (/ n-cells n-cells-denom))))))]
                             [["Fire name" (:pyrcst_fire_name replay-res-stats)]
                              ["Variation" variation-name]
                              ["t0" (-> replay-res-stats ::t0-fire :pyrcst_snapshot_inst)]
                              ["t1" (format "%s (%s)"
                                            (-> replay-res-stats ::t1-fire :pyrcst_snapshot_inst (str))
                                            (format-hours-between (-> replay-res-stats ::t0-fire :pyrcst_snapshot_inst)
                                                                  (-> replay-res-stats ::t1-fire :pyrcst_snapshot_inst)))]
                              ["n cells really burned" (:observed-burn-n-cells stats)]
                              ["sim  real" (format-w-pct (:sim-inter-obs-n-cells stats))]
                              ["ToA-ratio: mean" (some->> (::toa-ratio-mean stats) (format "%1.2f"))]
                              ["p50" (some->> (::toa-ratio-p50 stats) (format "%1.2f"))]
                              ["p75" (some->> (::toa-ratio-p75 stats) (format "%1.2f"))]
                              ["p90" (some->> (::toa-ratio-p90 stats) (format "%1.2f"))]
                              ["p95" (some->> (::toa-ratio-p95 stats) (format "%1.2f"))]
                              ["sim - real" (format-w-pct (:sim-minus-obs-n-cells stats))]
                              ["real - sim" (format-w-pct (:obs-minus-sim-n-cells stats))]
                              ["sr crowning" (format-w-pct (:sim-inter-obs-n-crowning stats) (:sim-inter-obs-n-cells stats))]
                              ["s-r crowning" (format-w-pct (:sim-minus-obs-n-crowning stats) (:sim-minus-obs-n-cells stats))]]))))))))

(defn print-replayed-results-table
  [rr-stats]
  (pprint-table-from-kv-lists
   (replayed-results-table-entries rr-stats)))


(defn tensor-nan-if
  [pred tensr]
  (d/emap
   (fn nan-if [^double x]
     (if (pred x)
       Double/NaN
       x))
   nil
```

```
   tensr))

(defn print-inputs-quantiles-tables
  [matrix-k+expected-units+nan-fn fire-name+sim-inputs]
  (->> fire-name+sim-inputs
       (pmap
        (fn [[fire-name sim-inputs]]
          (->> matrix-k+expected-units+nan-fn
               (mapv (fn [[matrix-k expected-unit nan-fn]]
                       (-> sim-inputs
                           (get matrix-k)
                           (cond-> (some? nan-fn) (->> (tensor-nan-if nan-fn)))
                           (as-> mx
                                 (let [ks [:min :quartile-1 :median :quartile-3]]
                                   (merge
                                    {:pyrcst_fire_name fire-name
                                     :input-name (format "%s (%s)" (pr-str matrix-k) expected-unit)}
                                    (->
                                      (dfn/descriptive-statistics
                                       ks
                                       {:nan-strategy :remove}
                                       mx)
                                      ;; reordering keys
                                      (select-keys ks)))))))))))
       (sequence cat)
       (pprint/print-table)))

(def variation-name->info
  {"00) original" {}
   "01) crowning disabled" {::transform-inputs-fn (fn [inputs] (assoc inputs :canopy-base-height-matrix (:canopy-height-matrix inpu

(def rr-stats-85f1467
  (delay
   (-> (io/resource "gridfire/lab/replay/rr-stats-85f1467.edn")
       slurp
       read-string)))

(def rr-stats-9689124
  (delay
   (comment
     ;; How this was computed, at commit 9689124:
     (->> (replay-snapshots variation-name->info nil snaps)
          (replayed-results-stats)
          (vec)
          (time)))
   (-> (io/resource "gridfire/lab/replay/rr-stats-9689124.edn")
       slurp
       read-string)))

(comment

  (def snaps (extract-pyrecast-snapshots! "../pyrecast-snapshots-2022-09-30"))

  (count snaps)
  ;;=> 256

  (take 3 snaps)

  (->> snaps (map ::useful-files) (mapcat keys) frequencies)
  ;; => ;; Darn, some of them are missing
  #:gridfire.lab.replay{:gridfire-config-file                  253,
                        :already_burned-tif                    247,
                        :active_fire_polygon_utm_posnegbuff-shp 250}

  (-> (replayable-successive-t0t1-pairs snaps) (count))
  ;;=> 187

  (->> snaps
```

```
     (filter #(-> % ::useful-files ::gridfire-config-file (some?)))
     (map fetch-snapshot-config)
     (keep :suppression))
;;=> ()

(def replay-results (time (replay-snapshots variation-name->info 64 snaps)))
;;"Elapsed time: 59593.890425 msecs"


(def replay-results2 (time (replay-snapshots variation-name->info 8 snaps)))

(def rr-stats-85f1467
  (future
    ;; How this was computed, at commit 85f1467:
    (->> (replay-snapshots variation-name->info nil snaps)
         (replayed-results-stats)
         (vec)
         (time))))

(print-replayed-results-table @rr-stats-85f1467)
;; OLD
;;|              Fire name |             Variation |                          t0 |                                  t1 | n cells
;;|------------------------+-----------------------+-----------------------------+-------------------------------------+---------
;;|              ca-summit |          00) original | Wed Sep 21 10:15:00 UTC 2022 | Thu Sep 22 09:58:00 UTC 2022 (+24hr) |
;;|              ca-summit | 01) crowning disabled | Wed Sep 21 10:15:00 UTC 2022 | Thu Sep 22 09:58:00 UTC 2022 (+24hr) |
;;|            or-sturgill |          00) original | Wed Sep 14 09:54:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|            or-sturgill | 01) crowning disabled | Wed Sep 14 09:54:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|          mt-government |          00) original | Fri Sep 16 11:01:00 UTC 2022 | Sat Sep 17 11:20:00 UTC 2022 (+24hr) |
;;|          mt-government | 01) crowning disabled | Fri Sep 16 11:01:00 UTC 2022 | Sat Sep 17 11:20:00 UTC 2022 (+24hr) |
;;|               id-lemhi |          00) original | Sun Sep 25 21:12:00 UTC 2022 | Mon Sep 26 10:19:00 UTC 2022 (+13hr) |
;;|               id-lemhi | 01) crowning disabled | Sun Sep 25 21:12:00 UTC 2022 | Mon Sep 26 10:19:00 UTC 2022 (+13hr) |
;;|           wa-irving-pk |          00) original | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+12hr) |
;;|           wa-irving-pk | 01) crowning disabled | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+12hr) |
;;| id-isabella-lower-twin |          00) original | Sun Sep 18 08:37:00 UTC 2022 | Mon Sep 19 10:49:00 UTC 2022 (+26hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Sun Sep 18 08:37:00 UTC 2022 | Mon Sep 19 10:49:00 UTC 2022 (+26hr) |
;;|              ca-summit |          00) original | Thu Sep 22 21:18:00 UTC 2022 | Sat Sep 24 09:20:00 UTC 2022 (+36hr) |
;;|              ca-summit | 01) crowning disabled | Thu Sep 22 21:18:00 UTC 2022 | Sat Sep 24 09:20:00 UTC 2022 (+36hr) |
;;|          wa-goat-rocks |          00) original | Sun Sep 25 09:45:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+11hr) |
;;|          wa-goat-rocks | 01) crowning disabled | Sun Sep 25 09:45:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+11hr) |
;;| id-isabella-lower-twin |          00) original | Mon Sep 19 10:49:00 UTC 2022 | Mon Sep 19 21:25:00 UTC 2022 (+11hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Mon Sep 19 10:49:00 UTC 2022 | Mon Sep 19 21:25:00 UTC 2022 (+11hr) |
;;|              ca-summit |          00) original | Mon Sep 26 09:32:00 UTC 2022 | Tue Sep 27 09:11:00 UTC 2022 (+24hr) |
;;|              ca-summit | 01) crowning disabled | Mon Sep 26 09:32:00 UTC 2022 | Tue Sep 27 09:11:00 UTC 2022 (+24hr) |
;;|        or-double-creek |          00) original | Mon Sep 19 09:09:00 UTC 2022 | Tue Sep 20 08:52:00 UTC 2022 (+24hr) |
;;|        or-double-creek | 01) crowning disabled | Mon Sep 19 09:09:00 UTC 2022 | Tue Sep 20 08:52:00 UTC 2022 (+24hr) |
;;|            mt-billiard |          00) original | Wed Sep 14 09:52:00 UTC 2022 | Fri Sep 16 11:11:00 UTC 2022 (+49hr) |
;;|            mt-billiard | 01) crowning disabled | Wed Sep 14 09:52:00 UTC 2022 | Fri Sep 16 11:11:00 UTC 2022 (+49hr) |
;;|                ca-red |          00) original | Wed Sep 14 16:50:00 UTC 2022 | Thu Sep 15 10:28:00 UTC 2022 (+18hr) |
;;|                ca-red | 01) crowning disabled | Wed Sep 14 16:50:00 UTC 2022 | Thu Sep 15 10:28:00 UTC 2022 (+18hr) |
;;|             mt-cannon |          00) original | Thu Sep 15 09:34:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+59hr) |
;;|             mt-cannon | 01) crowning disabled | Thu Sep 15 09:34:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+59hr) |
;;|          wa-goat-rocks |          00) original | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+14hr) |
;;|          wa-goat-rocks | 01) crowning disabled | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+14hr) |
;;|             id-ross-fk |          00) original | Tue Sep 13 09:24:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+25hr) |
;;|             id-ross-fk | 01) crowning disabled | Tue Sep 13 09:24:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+25hr) |
;;|                ca-red |          00) original | Thu Sep 15 10:28:00 UTC 2022 | Thu Sep 15 17:34:00 UTC 2022 (+ 7hr) |
;;|                ca-red | 01) crowning disabled | Thu Sep 15 10:28:00 UTC 2022 | Thu Sep 15 17:34:00 UTC 2022 (+ 7hr) |
;;|              ca-summit |          00) original | Mon Sep 19 21:23:00 UTC 2022 | Tue Sep 20 08:54:00 UTC 2022 (+12hr) |
;;|              ca-summit | 01) crowning disabled | Mon Sep 19 21:23:00 UTC 2022 | Tue Sep 20 08:54:00 UTC 2022 (+12hr) |
;;|        or-double-creek |          00) original | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|        or-double-creek | 01) crowning disabled | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|                wa-kid |          00) original | Tue Sep 13 20:51:00 UTC 2022 | Wed Sep 14 09:52:00 UTC 2022 (+13hr) |
;;|                wa-kid | 01) crowning disabled | Tue Sep 13 20:51:00 UTC 2022 | Wed Sep 14 09:52:00 UTC 2022 (+13hr) |
;;|        wa-minnow-ridge |          00) original | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|        wa-minnow-ridge | 01) crowning disabled | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|                ca-red |          00) original | Fri Sep 16 20:40:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+12hr) |
;;|                ca-red | 01) crowning disabled | Fri Sep 16 20:40:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+12hr) |
;;|         mt-george-lake |          00) original | Tue Sep 13 09:22:00 UTC 2022 | Thu Sep 15 11:09:00 UTC 2022 (+50hr) |
```

```
;;|         mt-george-lake | 01) crowning disabled | Tue Sep 13 09:22:00 UTC 2022 | Thu Sep 15 11:09:00 UTC 2022 (+50hr) |
;;|               id-lemhi |          00) original | Mon Sep 26 10:19:00 UTC 2022 | Tue Sep 27 09:09:00 UTC 2022 (+23hr) |
;;|               id-lemhi | 01) crowning disabled | Mon Sep 26 10:19:00 UTC 2022 | Tue Sep 27 09:09:00 UTC 2022 (+23hr) |
;;|            wa-irving-pk |          00) original | Mon Sep 26 09:28:00 UTC 2022 | Tue Sep 27 09:07:00 UTC 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled | Mon Sep 26 09:28:00 UTC 2022 | Tue Sep 27 09:07:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original | Tue Sep 27 09:07:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Tue Sep 27 09:07:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:17:00 UTC 2022 (+11hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:17:00 UTC 2022 (+11hr) |
;;|          mt-government |          00) original | Sat Sep 17 11:20:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+ 9hr) |
;;|          mt-government | 01) crowning disabled | Sat Sep 17 11:20:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+ 9hr) |
;;|              ca-barnes |          00) original | Tue Sep 13 13:18:00 UTC 2022 | Tue Sep 13 19:12:00 UTC 2022 (+ 6hr) |
;;|              ca-barnes | 01) crowning disabled | Tue Sep 13 13:18:00 UTC 2022 | Tue Sep 13 19:12:00 UTC 2022 (+ 6hr) |
;;|         wa-minnow-ridge |          00) original | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+12hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+12hr) |
;;|         wa-minnow-ridge |          00) original | Wed Sep 21 09:22:00 UTC 2022 | Thu Sep 22 10:43:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Wed Sep 21 09:22:00 UTC 2022 | Thu Sep 22 10:43:00 UTC 2022 (+25hr) |
;;|            wa-irving-pk |          00) original | Mon Sep 19 10:00:00 UTC 2022 | Tue Sep 20 08:50:00 UTC 2022 (+23hr) |
;;|            wa-irving-pk | 01) crowning disabled | Mon Sep 19 10:00:00 UTC 2022 | Tue Sep 20 08:50:00 UTC 2022 (+23hr) |
;;|         or-double-creek |          00) original | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 08:33:00 UTC 2022 (+12hr) |
;;|         or-double-creek | 01) crowning disabled | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 08:33:00 UTC 2022 (+12hr) |
;;| id-columbus-bear-gulch |          00) original | Wed Sep 14 21:21:00 UTC 2022 | Sat Sep 17 10:36:00 UTC 2022 (+61hr) |
;;| id-columbus-bear-gulch | 01) crowning disabled | Wed Sep 14 21:21:00 UTC 2022 | Sat Sep 17 10:36:00 UTC 2022 (+61hr) |
;;|               id-lemhi |          00) original | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+14hr) |
;;|               id-lemhi | 01) crowning disabled | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+14hr) |
;;|            wa-irving-pk |          00) original | Wed Sep 28 10:30:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled | Wed Sep 28 10:30:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+24hr) |
;;|            wa-irving-pk |          00) original | Sat Sep 17 09:47:00 UTC 2022 | Mon Sep 19 10:00:00 UTC 2022 (+48hr) |
;;|            wa-irving-pk | 01) crowning disabled | Sat Sep 17 09:47:00 UTC 2022 | Mon Sep 19 10:00:00 UTC 2022 (+48hr) |
;;|         or-double-creek |          00) original | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|         or-double-creek | 01) crowning disabled | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|              ca-summit |          00) original | Sun Sep 25 09:49:00 UTC 2022 | Mon Sep 26 09:32:00 UTC 2022 (+24hr) |
;;|              ca-summit | 01) crowning disabled | Sun Sep 25 09:49:00 UTC 2022 | Mon Sep 26 09:32:00 UTC 2022 (+24hr) |
;;| id-caledonia-blackburn |          00) original | Wed Sep 14 09:52:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+72hr) |
;;| id-caledonia-blackburn | 01) crowning disabled | Wed Sep 14 09:52:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+72hr) |
;;| id-kootenai-ru-complex |          00) original | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 02:03:00 UTC 2022 (+ 6hr) |
;;| id-kootenai-ru-complex | 01) crowning disabled | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 02:03:00 UTC 2022 (+ 6hr) |
;;|              ca-summit |          00) original | Tue Sep 27 09:11:00 UTC 2022 | Wed Sep 28 09:43:00 UTC 2022 (+25hr) |
;;|              ca-summit | 01) crowning disabled | Tue Sep 27 09:11:00 UTC 2022 | Wed Sep 28 09:43:00 UTC 2022 (+25hr) |
;;|            mt-margaret |          00) original | Thu Sep 15 08:43:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+49hr) |
;;|            mt-margaret | 01) crowning disabled | Thu Sep 15 08:43:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+49hr) |
;;|            wa-irving-pk |          00) original | Thu Sep 15 10:26:00 UTC 2022 | Fri Sep 16 09:15:00 UTC 2022 (+23hr) |
;;|            wa-irving-pk | 01) crowning disabled | Thu Sep 15 10:26:00 UTC 2022 | Fri Sep 16 09:15:00 UTC 2022 (+23hr) |
;;|             wa-siouoxon |          00) original | Sun Sep 25 11:28:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+24hr) |
;;|             wa-siouoxon | 01) crowning disabled | Sun Sep 25 11:28:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+24hr) |
;;| id-caledonia-blackburn |          00) original | Sat Sep 17 09:47:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+11hr) |
;;| id-caledonia-blackburn | 01) crowning disabled | Sat Sep 17 09:47:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+11hr) |
;;|         or-cedar-creek |          00) original | Fri Sep 16 09:15:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+24hr) |
;;|         or-cedar-creek | 01) crowning disabled | Fri Sep 16 09:15:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+24hr) |
;;|            wa-irving-pk |          00) original | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|            wa-irving-pk | 01) crowning disabled | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|          id-trail-ridge |          00) original | Sat Sep 17 10:23:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+10hr) |
;;|          id-trail-ridge | 01) crowning disabled | Sat Sep 17 10:23:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+10hr) |
;;| id-kootenai-ru-complex |          00) original | Wed Sep 14 12:19:00 UTC 2022 | Fri Sep 16 20:02:00 UTC 2022 (+56hr) |
;;| id-kootenai-ru-complex | 01) crowning disabled | Wed Sep 14 12:19:00 UTC 2022 | Fri Sep 16 20:02:00 UTC 2022 (+56hr) |
;;|               id-katka |          00) original | Fri Sep 16 20:02:00 UTC 2022 | Sun Sep 18 12:12:00 UTC 2022 (+40hr) |
;;|               id-katka | 01) crowning disabled | Fri Sep 16 20:02:00 UTC 2022 | Sun Sep 18 12:12:00 UTC 2022 (+40hr) |
;;|           or-sturgill |          00) original | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|           or-sturgill | 01) crowning disabled | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|          wa-bolt-creek |          00) original | Thu Sep 22 09:54:00 UTC 2022 | Thu Sep 22 20:29:00 UTC 2022 (+11hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Thu Sep 22 09:54:00 UTC 2022 | Thu Sep 22 20:29:00 UTC 2022 (+11hr) |
;;| id-kootenai-ru-complex |          00) original | Fri Sep 16 20:02:00 UTC 2022 | Sun Sep 18 12:04:00 UTC 2022 (+40hr) |
;;| id-kootenai-ru-complex | 01) crowning disabled | Fri Sep 16 20:02:00 UTC 2022 | Sun Sep 18 12:04:00 UTC 2022 (+40hr) |
;;|             ca-rodgers |          00) original | Fri Sep 16 16:28:00 UTC 2022 | Sat Sep 17 09:49:00 UTC 2022 (+17hr) |
;;|             ca-rodgers | 01) crowning disabled | Fri Sep 16 16:28:00 UTC 2022 | Sat Sep 17 09:49:00 UTC 2022 (+17hr) |
;;|         wa-minnow-ridge |          00) original | Sun Sep 25 08:58:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+12hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Sun Sep 25 08:58:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+12hr) |
;;|          mt-government |          00) original | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 09:28:00 UTC 2022 (+13hr) |
```

```
;;|          mt-government | 01) crowning disabled | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 09:28:00 UTC 2022 (+13hr) |
;;|               id-katka |          00) original | Wed Sep 14 12:19:00 UTC 2022 | Fri Sep 16 20:02:00 UTC 2022 (+56hr) |
;;|               id-katka | 01) crowning disabled | Wed Sep 14 12:19:00 UTC 2022 | Fri Sep 16 20:02:00 UTC 2022 (+56hr) |
;;|          wa-goat-rocks |          00) original | Mon Sep 26 11:08:00 UTC 2022 | Tue Sep 27 10:00:00 UTC 2022 (+23hr) |
;;|          wa-goat-rocks | 01) crowning disabled | Mon Sep 26 11:08:00 UTC 2022 | Tue Sep 27 10:00:00 UTC 2022 (+23hr) |
;;|          id-deep-creek |          00) original | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:19:00 UTC 2022 (+11hr) |
;;|          id-deep-creek | 01) crowning disabled | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:19:00 UTC 2022 (+11hr) |
;;|              ca-summit |          00) original | Sun Sep 18 10:21:00 UTC 2022 | Sun Sep 18 16:30:00 UTC 2022 (+ 6hr) |
;;|              ca-summit | 01) crowning disabled | Sun Sep 18 10:21:00 UTC 2022 | Sun Sep 18 16:30:00 UTC 2022 (+ 6hr) |
;;| id-isabella-lower-twin |          00) original | Thu Sep 15 09:34:00 UTC 2022 | Sat Sep 17 08:56:00 UTC 2022 (+47hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Thu Sep 15 09:34:00 UTC 2022 | Sat Sep 17 08:56:00 UTC 2022 (+47hr) |
;;|             id-tenmile |          00) original | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|             id-tenmile | 01) crowning disabled | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|            ca-mosquito |          00) original | Wed Sep 14 22:01:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+12hr) |
;;|            ca-mosquito | 01) crowning disabled | Wed Sep 14 22:01:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+12hr) |

(def rr-stats-9689124
  (future
   ;; How this was computed, at commit 9689124:
   (->> (replay-snapshots variation-name->info nil snaps)
        (replayed-results-stats)
        (vec)
        (time))))

(print-replayed-results-table @rr-stats-9689124)
;; NEW
;;|               Fire name |             Variation |                          t0 |                                  t1 | n cells
;;|-------------------------+-----------------------+-----------------------------+-------------------------------------+--------
;;|               ca-barnes |          00) original | Tue Sep 13 13:18:00 UTC 2022 | Tue Sep 13 19:12:00 UTC 2022 (+ 6hr) |
;;|               ca-barnes | 01) crowning disabled | Tue Sep 13 13:18:00 UTC 2022 | Tue Sep 13 19:12:00 UTC 2022 (+ 6hr) |
;;|             ca-mosquito |          00) original | Wed Sep 14 09:05:00 UTC 2022 | Wed Sep 14 14:11:00 UTC 2022 (+ 5hr) |
;;|             ca-mosquito | 01) crowning disabled | Wed Sep 14 09:05:00 UTC 2022 | Wed Sep 14 14:11:00 UTC 2022 (+ 5hr) |
;;|             ca-mosquito |          00) original | Wed Sep 14 14:11:00 UTC 2022 | Wed Sep 14 22:01:00 UTC 2022 (+ 8hr) |
;;|             ca-mosquito | 01) crowning disabled | Wed Sep 14 14:11:00 UTC 2022 | Wed Sep 14 22:01:00 UTC 2022 (+ 8hr) |
;;|             ca-mosquito |          00) original | Wed Sep 14 22:01:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+12hr) |
;;|             ca-mosquito | 01) crowning disabled | Wed Sep 14 22:01:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+12hr) |
;;|             ca-mosquito |          00) original | Thu Sep 15 09:34:00 UTC 2022 | Fri Sep 16 09:17:00 UTC 2022 (+24hr) |
;;|             ca-mosquito | 01) crowning disabled | Thu Sep 15 09:34:00 UTC 2022 | Fri Sep 16 09:17:00 UTC 2022 (+24hr) |
;;|             ca-mosquito |          00) original | Fri Sep 16 09:17:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+24hr) |
;;|             ca-mosquito | 01) crowning disabled | Fri Sep 16 09:17:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+24hr) |
;;|             ca-mosquito |          00) original | Sat Sep 17 08:58:00 UTC 2022 | Sat Sep 17 22:26:00 UTC 2022 (+13hr) |
;;|             ca-mosquito | 01) crowning disabled | Sat Sep 17 08:58:00 UTC 2022 | Sat Sep 17 22:26:00 UTC 2022 (+13hr) |
;;|             ca-mosquito |          00) original | Sat Sep 17 22:26:00 UTC 2022 | Sun Sep 18 09:32:00 UTC 2022 (+11hr) |
;;|             ca-mosquito | 01) crowning disabled | Sat Sep 17 22:26:00 UTC 2022 | Sun Sep 18 09:32:00 UTC 2022 (+11hr) |
;;|                  ca-red |          00) original | Wed Sep 14 09:05:00 UTC 2022 | Wed Sep 14 16:50:00 UTC 2022 (+ 8hr) |
;;|                  ca-red | 01) crowning disabled | Wed Sep 14 09:05:00 UTC 2022 | Wed Sep 14 16:50:00 UTC 2022 (+ 8hr) |
;;|                  ca-red |          00) original | Wed Sep 14 16:50:00 UTC 2022 | Thu Sep 15 10:28:00 UTC 2022 (+18hr) |
;;|                  ca-red | 01) crowning disabled | Wed Sep 14 16:50:00 UTC 2022 | Thu Sep 15 10:28:00 UTC 2022 (+18hr) |
;;|                  ca-red |          00) original | Thu Sep 15 10:28:00 UTC 2022 | Thu Sep 15 17:34:00 UTC 2022 (+ 7hr) |
;;|                  ca-red | 01) crowning disabled | Thu Sep 15 10:28:00 UTC 2022 | Thu Sep 15 17:34:00 UTC 2022 (+ 7hr) |
;;|                  ca-red |          00) original | Thu Sep 15 17:34:00 UTC 2022 | Thu Sep 15 20:57:00 UTC 2022 (+ 3hr) |
;;|                  ca-red | 01) crowning disabled | Thu Sep 15 17:34:00 UTC 2022 | Thu Sep 15 20:57:00 UTC 2022 (+ 3hr) |
;;|                  ca-red |          00) original | Thu Sep 15 20:57:00 UTC 2022 | Fri Sep 16 09:17:00 UTC 2022 (+12hr) |
;;|                  ca-red | 01) crowning disabled | Thu Sep 15 20:57:00 UTC 2022 | Fri Sep 16 09:17:00 UTC 2022 (+12hr) |
;;|                  ca-red |          00) original | Fri Sep 16 09:17:00 UTC 2022 | Fri Sep 16 20:40:00 UTC 2022 (+11hr) |
;;|                  ca-red | 01) crowning disabled | Fri Sep 16 09:17:00 UTC 2022 | Fri Sep 16 20:40:00 UTC 2022 (+11hr) |
;;|                  ca-red |          00) original | Fri Sep 16 20:40:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+12hr) |
;;|                  ca-red | 01) crowning disabled | Fri Sep 16 20:40:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+12hr) |
;;|                  ca-red |          00) original | Sat Sep 17 08:58:00 UTC 2022 | Sat Sep 17 21:12:00 UTC 2022 (+12hr) |
;;|                  ca-red | 01) crowning disabled | Sat Sep 17 08:58:00 UTC 2022 | Sat Sep 17 21:12:00 UTC 2022 (+12hr) |
;;|              ca-rodgers |          00) original | Wed Sep 14 09:56:00 UTC 2022 | Thu Sep 15 21:07:00 UTC 2022 (+35hr) |
;;|              ca-rodgers | 01) crowning disabled | Wed Sep 14 09:56:00 UTC 2022 | Thu Sep 15 21:07:00 UTC 2022 (+35hr) |
;;|              ca-rodgers |          00) original | Thu Sep 15 21:07:00 UTC 2022 | Fri Sep 16 16:28:00 UTC 2022 (+19hr) |
;;|              ca-rodgers | 01) crowning disabled | Thu Sep 15 21:07:00 UTC 2022 | Fri Sep 16 16:28:00 UTC 2022 (+19hr) |
;;|              ca-rodgers |          00) original | Fri Sep 16 16:28:00 UTC 2022 | Sat Sep 17 09:49:00 UTC 2022 (+17hr) |
;;|              ca-rodgers | 01) crowning disabled | Fri Sep 16 16:28:00 UTC 2022 | Sat Sep 17 09:49:00 UTC 2022 (+17hr) |
;;|               ca-summit |          00) original | Sun Sep 18 10:21:00 UTC 2022 | Sun Sep 18 16:30:00 UTC 2022 (+ 6hr) |
;;|               ca-summit | 01) crowning disabled | Sun Sep 18 10:21:00 UTC 2022 | Sun Sep 18 16:30:00 UTC 2022 (+ 6hr) |
```

```
;;|              ca-summit |          00) original | Sun Sep 18 16:30:00 UTC 2022 | Mon Sep 19 10:02:00 UTC 2022 (+18hr) |
;;|              ca-summit | 01) crowning disabled | Sun Sep 18 16:30:00 UTC 2022 | Mon Sep 19 10:02:00 UTC 2022 (+18hr) |
;;|              ca-summit |          00) original | Mon Sep 19 10:02:00 UTC 2022 | Mon Sep 19 21:23:00 UTC 2022 (+11hr) |
;;|              ca-summit | 01) crowning disabled | Mon Sep 19 10:02:00 UTC 2022 | Mon Sep 19 21:23:00 UTC 2022 (+11hr) |
;;|              ca-summit |          00) original | Mon Sep 19 21:23:00 UTC 2022 | Tue Sep 20 08:54:00 UTC 2022 (+12hr) |
;;|              ca-summit | 01) crowning disabled | Mon Sep 19 21:23:00 UTC 2022 | Tue Sep 20 08:54:00 UTC 2022 (+12hr) |
;;|              ca-summit |          00) original | Tue Sep 20 08:54:00 UTC 2022 | Wed Sep 21 10:15:00 UTC 2022 (+25hr) |
;;|              ca-summit | 01) crowning disabled | Tue Sep 20 08:54:00 UTC 2022 | Wed Sep 21 10:15:00 UTC 2022 (+25hr) |
;;|              ca-summit |          00) original | Wed Sep 21 10:15:00 UTC 2022 | Thu Sep 22 09:58:00 UTC 2022 (+24hr) |
;;|              ca-summit | 01) crowning disabled | Wed Sep 21 10:15:00 UTC 2022 | Thu Sep 22 09:58:00 UTC 2022 (+24hr) |
;;|              ca-summit |          00) original | Thu Sep 22 09:58:00 UTC 2022 | Thu Sep 22 21:18:00 UTC 2022 (+11hr) |
;;|              ca-summit | 01) crowning disabled | Thu Sep 22 09:58:00 UTC 2022 | Thu Sep 22 21:18:00 UTC 2022 (+11hr) |
;;|              ca-summit |          00) original | Thu Sep 22 21:18:00 UTC 2022 | Sat Sep 24 09:20:00 UTC 2022 (+36hr) |
;;|              ca-summit | 01) crowning disabled | Thu Sep 22 21:18:00 UTC 2022 | Sat Sep 24 09:20:00 UTC 2022 (+36hr) |
;;|              ca-summit |          00) original | Sat Sep 24 09:20:00 UTC 2022 | Sun Sep 25 09:49:00 UTC 2022 (+24hr) |
;;|              ca-summit | 01) crowning disabled | Sat Sep 24 09:20:00 UTC 2022 | Sun Sep 25 09:49:00 UTC 2022 (+24hr) |
;;|              ca-summit |          00) original | Sun Sep 25 09:49:00 UTC 2022 | Mon Sep 26 09:32:00 UTC 2022 (+24hr) |
;;|              ca-summit | 01) crowning disabled | Sun Sep 25 09:49:00 UTC 2022 | Mon Sep 26 09:32:00 UTC 2022 (+24hr) |
;;|              ca-summit |          00) original | Mon Sep 26 09:32:00 UTC 2022 | Tue Sep 27 09:11:00 UTC 2022 (+24hr) |
;;|              ca-summit | 01) crowning disabled | Mon Sep 26 09:32:00 UTC 2022 | Tue Sep 27 09:11:00 UTC 2022 (+24hr) |
;;|              ca-summit |          00) original | Tue Sep 27 09:11:00 UTC 2022 | Wed Sep 28 09:43:00 UTC 2022 (+25hr) |
;;|              ca-summit | 01) crowning disabled | Tue Sep 27 09:11:00 UTC 2022 | Wed Sep 28 09:43:00 UTC 2022 (+25hr) |
;;|              ca-summit |          00) original | Wed Sep 28 09:43:00 UTC 2022 | Thu Sep 29 10:13:00 UTC 2022 (+25hr) |
;;|              ca-summit | 01) crowning disabled | Wed Sep 28 09:43:00 UTC 2022 | Thu Sep 29 10:13:00 UTC 2022 (+25hr) |
;;| id-caledonia-blackburn |          00) original | Wed Sep 14 09:52:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+72hr) |
;;| id-caledonia-blackburn | 01) crowning disabled | Wed Sep 14 09:52:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+72hr) |
;;| id-caledonia-blackburn |          00) original | Sat Sep 17 09:47:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+11hr) |
;;| id-caledonia-blackburn | 01) crowning disabled | Sat Sep 17 09:47:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+11hr) |
;;| id-columbus-bear-gulch |          00) original | Tue Sep 13 10:13:00 UTC 2022 | Wed Sep 14 09:05:00 UTC 2022 (+23hr) |
;;| id-columbus-bear-gulch | 01) crowning disabled | Tue Sep 13 10:13:00 UTC 2022 | Wed Sep 14 09:05:00 UTC 2022 (+23hr) |
;;| id-columbus-bear-gulch |          00) original | Wed Sep 14 09:05:00 UTC 2022 | Wed Sep 14 21:21:00 UTC 2022 (+12hr) |
;;| id-columbus-bear-gulch | 01) crowning disabled | Wed Sep 14 09:05:00 UTC 2022 | Wed Sep 14 21:21:00 UTC 2022 (+12hr) |
;;| id-columbus-bear-gulch |          00) original | Wed Sep 14 21:21:00 UTC 2022 | Sat Sep 17 10:36:00 UTC 2022 (+61hr) |
;;| id-columbus-bear-gulch | 01) crowning disabled | Wed Sep 14 21:21:00 UTC 2022 | Sat Sep 17 10:36:00 UTC 2022 (+61hr) |
;;| id-columbus-bear-gulch |          00) original | Sat Sep 17 10:36:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+10hr) |
;;| id-columbus-bear-gulch | 01) crowning disabled | Sat Sep 17 10:36:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+10hr) |
;;| id-columbus-bear-gulch |          00) original | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 10:17:00 UTC 2022 (+14hr) |
;;| id-columbus-bear-gulch | 01) crowning disabled | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 10:17:00 UTC 2022 (+14hr) |
;;|          id-deep-creek |          00) original | Sat Sep 17 21:14:00 UTC 2022 | Sun Sep 18 10:17:00 UTC 2022 (+13hr) |
;;|          id-deep-creek | 01) crowning disabled | Sat Sep 17 21:14:00 UTC 2022 | Sun Sep 18 10:17:00 UTC 2022 (+13hr) |
;;|          id-deep-creek |          00) original | Sun Sep 18 10:17:00 UTC 2022 | Mon Sep 19 10:00:00 UTC 2022 (+24hr) |
;;|          id-deep-creek | 01) crowning disabled | Sun Sep 18 10:17:00 UTC 2022 | Mon Sep 19 10:00:00 UTC 2022 (+24hr) |
;;|          id-deep-creek |          00) original | Mon Sep 19 10:00:00 UTC 2022 | Mon Sep 19 21:25:00 UTC 2022 (+11hr) |
;;|          id-deep-creek | 01) crowning disabled | Mon Sep 19 10:00:00 UTC 2022 | Mon Sep 19 21:25:00 UTC 2022 (+11hr) |
;;|          id-deep-creek |          00) original | Mon Sep 19 21:25:00 UTC 2022 | Tue Sep 20 09:41:00 UTC 2022 (+12hr) |
;;|          id-deep-creek | 01) crowning disabled | Mon Sep 19 21:25:00 UTC 2022 | Tue Sep 20 09:41:00 UTC 2022 (+12hr) |
;;|          id-deep-creek |          00) original | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:19:00 UTC 2022 (+11hr) |
;;|          id-deep-creek | 01) crowning disabled | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:19:00 UTC 2022 (+11hr) |
;;| id-isabella-lower-twin |          00) original | Tue Sep 13 10:11:00 UTC 2022 | Wed Sep 14 09:05:00 UTC 2022 (+23hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Tue Sep 13 10:11:00 UTC 2022 | Wed Sep 14 09:05:00 UTC 2022 (+23hr) |
;;| id-isabella-lower-twin |          00) original | Wed Sep 14 09:05:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Wed Sep 14 09:05:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;| id-isabella-lower-twin |          00) original | Thu Sep 15 09:34:00 UTC 2022 | Sat Sep 17 08:56:00 UTC 2022 (+47hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Thu Sep 15 09:34:00 UTC 2022 | Sat Sep 17 08:56:00 UTC 2022 (+47hr) |
;;| id-isabella-lower-twin |          00) original | Sat Sep 17 08:56:00 UTC 2022 | Sun Sep 18 08:37:00 UTC 2022 (+24hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Sat Sep 17 08:56:00 UTC 2022 | Sun Sep 18 08:37:00 UTC 2022 (+24hr) |
;;| id-isabella-lower-twin |          00) original | Sun Sep 18 08:37:00 UTC 2022 | Mon Sep 19 10:49:00 UTC 2022 (+26hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Sun Sep 18 08:37:00 UTC 2022 | Mon Sep 19 10:49:00 UTC 2022 (+26hr) |
;;| id-isabella-lower-twin |          00) original | Mon Sep 19 10:49:00 UTC 2022 | Mon Sep 19 21:25:00 UTC 2022 (+11hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Mon Sep 19 10:49:00 UTC 2022 | Mon Sep 19 21:25:00 UTC 2022 (+11hr) |
;;| id-isabella-lower-twin |          00) original | Mon Sep 19 21:25:00 UTC 2022 | Tue Sep 20 08:52:00 UTC 2022 (+11hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Mon Sep 19 21:25:00 UTC 2022 | Tue Sep 20 08:52:00 UTC 2022 (+11hr) |
;;| id-isabella-lower-twin |          00) original | Tue Sep 20 08:52:00 UTC 2022 | Tue Sep 20 22:43:00 UTC 2022 (+14hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Tue Sep 20 08:52:00 UTC 2022 | Tue Sep 20 22:43:00 UTC 2022 (+14hr) |
;;| id-isabella-lower-twin |          00) original | Tue Sep 20 22:43:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+11hr) |
;;| id-isabella-lower-twin | 01) crowning disabled | Tue Sep 20 22:43:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+11hr) |
;;|               id-katka |          00) original | Wed Sep 14 12:19:00 UTC 2022 | Fri Sep 16 20:02:00 UTC 2022 (+56hr) |
;;|               id-katka | 01) crowning disabled | Wed Sep 14 12:19:00 UTC 2022 | Fri Sep 16 20:02:00 UTC 2022 (+56hr) |
```

```
;;|               id-katka |          00) original | Fri Sep 16 20:02:00 UTC 2022 | Sun Sep 18 12:12:00 UTC 2022 (+40hr) |
;;|               id-katka | 01) crowning disabled | Fri Sep 16 20:02:00 UTC 2022 | Sun Sep 18 12:12:00 UTC 2022 (+40hr) |
;;|  id-kootenai-ru-complex |          00) original | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:05:00 UTC 2022 (+24hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:05:00 UTC 2022 (+24hr) |
;;|  id-kootenai-ru-complex |          00) original | Wed Sep 14 09:05:00 UTC 2022 | Wed Sep 14 12:19:00 UTC 2022 (+ 3hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled | Wed Sep 14 09:05:00 UTC 2022 | Wed Sep 14 12:19:00 UTC 2022 (+ 3hr) |
;;|  id-kootenai-ru-complex |          00) original | Wed Sep 14 12:19:00 UTC 2022 | Fri Sep 16 20:02:00 UTC 2022 (+56hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled | Wed Sep 14 12:19:00 UTC 2022 | Fri Sep 16 20:02:00 UTC 2022 (+56hr) |
;;|  id-kootenai-ru-complex |          00) original | Fri Sep 16 20:02:00 UTC 2022 | Sun Sep 18 12:04:00 UTC 2022 (+40hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled | Fri Sep 16 20:02:00 UTC 2022 | Sun Sep 18 12:04:00 UTC 2022 (+40hr) |
;;|  id-kootenai-ru-complex |          00) original | Mon Sep 26 10:17:00 UTC 2022 | Tue Sep 27 10:47:00 UTC 2022 (+25hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled | Mon Sep 26 10:17:00 UTC 2022 | Tue Sep 27 10:47:00 UTC 2022 (+25hr) |
;;|  id-kootenai-ru-complex |          00) original | Tue Sep 27 10:47:00 UTC 2022 | Wed Sep 28 03:16:00 UTC 2022 (+16hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled | Tue Sep 27 10:47:00 UTC 2022 | Wed Sep 28 03:16:00 UTC 2022 (+16hr) |
;;|  id-kootenai-ru-complex |          00) original | Wed Sep 28 03:16:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+ 6hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled | Wed Sep 28 03:16:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+ 6hr) |
;;|  id-kootenai-ru-complex |          00) original | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|  id-kootenai-ru-complex |          00) original | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 02:03:00 UTC 2022 (+ 6hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 02:03:00 UTC 2022 (+ 6hr) |
;;|               id-lemhi |          00) original | Sun Sep 25 21:12:00 UTC 2022 | Mon Sep 26 10:19:00 UTC 2022 (+13hr) |
;;|               id-lemhi | 01) crowning disabled | Sun Sep 25 21:12:00 UTC 2022 | Mon Sep 26 10:19:00 UTC 2022 (+13hr) |
;;|               id-lemhi |          00) original | Mon Sep 26 10:19:00 UTC 2022 | Tue Sep 27 09:09:00 UTC 2022 (+23hr) |
;;|               id-lemhi | 01) crowning disabled | Mon Sep 26 10:19:00 UTC 2022 | Tue Sep 27 09:09:00 UTC 2022 (+23hr) |
;;|               id-lemhi |          00) original | Tue Sep 27 09:09:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+24hr) |
;;|               id-lemhi | 01) crowning disabled | Tue Sep 27 09:09:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+24hr) |
;;|               id-lemhi |          00) original | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|               id-lemhi | 01) crowning disabled | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|               id-lemhi |          00) original | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+14hr) |
;;|               id-lemhi | 01) crowning disabled | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+14hr) |
;;| id-lynx-meadows-3-prong |          00) original | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+25hr) |
;;| id-lynx-meadows-3-prong | 01) crowning disabled | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+25hr) |
;;|  id-patrol-point-dismal |          00) original | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+25hr) |
;;|  id-patrol-point-dismal | 01) crowning disabled | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+25hr) |
;;|              id-ross-fk |          00) original | Tue Sep 13 09:24:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+25hr) |
;;|              id-ross-fk | 01) crowning disabled | Tue Sep 13 09:24:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+25hr) |
;;|              id-tenmile |          00) original | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|              id-tenmile | 01) crowning disabled | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|              id-tenmile |          00) original | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|              id-tenmile | 01) crowning disabled | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|              id-tenmile |          00) original | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 08:33:00 UTC 2022 (+12hr) |
;;|              id-tenmile | 01) crowning disabled | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 08:33:00 UTC 2022 (+12hr) |
;;|          id-trail-ridge |          00) original | Sat Sep 17 10:23:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+10hr) |
;;|          id-trail-ridge | 01) crowning disabled | Sat Sep 17 10:23:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+10hr) |
;;|          id-trail-ridge |          00) original | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 08:37:00 UTC 2022 (+12hr) |
;;|          id-trail-ridge | 01) crowning disabled | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 08:37:00 UTC 2022 (+12hr) |
;;|            mt-billiard |          00) original | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:52:00 UTC 2022 (+25hr) |
;;|            mt-billiard | 01) crowning disabled | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:52:00 UTC 2022 (+25hr) |
;;|            mt-billiard |          00) original | Wed Sep 14 09:52:00 UTC 2022 | Fri Sep 16 11:11:00 UTC 2022 (+49hr) |
;;|            mt-billiard | 01) crowning disabled | Wed Sep 14 09:52:00 UTC 2022 | Fri Sep 16 11:11:00 UTC 2022 (+49hr) |
;;|              mt-cannon |          00) original | Wed Sep 14 09:52:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|              mt-cannon | 01) crowning disabled | Wed Sep 14 09:52:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|              mt-cannon |          00) original | Thu Sep 15 09:34:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+59hr) |
;;|              mt-cannon | 01) crowning disabled | Thu Sep 15 09:34:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+59hr) |
;;|              mt-cannon |          00) original | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 08:37:00 UTC 2022 (+12hr) |
;;|              mt-cannon | 01) crowning disabled | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 08:37:00 UTC 2022 (+12hr) |
;;|          mt-george-lake |          00) original | Tue Sep 13 09:22:00 UTC 2022 | Thu Sep 15 11:09:00 UTC 2022 (+50hr) |
;;|          mt-george-lake | 01) crowning disabled | Tue Sep 13 09:22:00 UTC 2022 | Thu Sep 15 11:09:00 UTC 2022 (+50hr) |
;;|           mt-government |          00) original | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:05:00 UTC 2022 (+24hr) |
;;|           mt-government | 01) crowning disabled | Tue Sep 13 09:22:00 UTC 2022 | Wed Sep 14 09:05:00 UTC 2022 (+24hr) |
;;|           mt-government |          00) original | Wed Sep 14 09:05:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|           mt-government | 01) crowning disabled | Wed Sep 14 09:05:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|           mt-government |          00) original | Thu Sep 15 09:34:00 UTC 2022 | Fri Sep 16 11:01:00 UTC 2022 (+25hr) |
;;|           mt-government | 01) crowning disabled | Thu Sep 15 09:34:00 UTC 2022 | Fri Sep 16 11:01:00 UTC 2022 (+25hr) |
;;|           mt-government |          00) original | Fri Sep 16 11:01:00 UTC 2022 | Sat Sep 17 11:20:00 UTC 2022 (+24hr) |
;;|           mt-government | 01) crowning disabled | Fri Sep 16 11:01:00 UTC 2022 | Sat Sep 17 11:20:00 UTC 2022 (+24hr) |
;;|           mt-government |          00) original | Sat Sep 17 11:20:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+ 9hr) |
;;|           mt-government | 01) crowning disabled | Sat Sep 17 11:20:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+ 9hr) |
```

```
;;|           mt-government |          00) original | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 09:28:00 UTC 2022 (+13hr) |
;;|           mt-government | 01) crowning disabled | Sat Sep 17 20:21:00 UTC 2022 | Sun Sep 18 09:28:00 UTC 2022 (+13hr) |
;;|        mt-isabella-lake |          00) original | Wed Sep 14 03:40:00 UTC 2022 | Fri Sep 16 11:26:00 UTC 2022 (+56hr) |
;;|        mt-isabella-lake | 01) crowning disabled | Wed Sep 14 03:40:00 UTC 2022 | Fri Sep 16 11:26:00 UTC 2022 (+56hr) |
;;|             mt-margaret |          00) original | Wed Sep 14 09:52:00 UTC 2022 | Thu Sep 15 08:43:00 UTC 2022 (+23hr) |
;;|             mt-margaret | 01) crowning disabled | Wed Sep 14 09:52:00 UTC 2022 | Thu Sep 15 08:43:00 UTC 2022 (+23hr) |
;;|             mt-margaret |          00) original | Thu Sep 15 08:43:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+49hr) |
;;|             mt-margaret | 01) crowning disabled | Thu Sep 15 08:43:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+49hr) |
;;|             mt-margaret |          00) original | Sat Sep 17 09:47:00 UTC 2022 | Sun Sep 18 09:28:00 UTC 2022 (+24hr) |
;;|             mt-margaret | 01) crowning disabled | Sat Sep 17 09:47:00 UTC 2022 | Sun Sep 18 09:28:00 UTC 2022 (+24hr) |
;;|            mt-mill-lake |          00) original | Sat Sep 17 10:50:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+10hr) |
;;|            mt-mill-lake | 01) crowning disabled | Sat Sep 17 10:50:00 UTC 2022 | Sat Sep 17 20:21:00 UTC 2022 (+10hr) |
;;|          or-cedar-creek |          00) original | Wed Sep 14 03:20:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+ 7hr) |
;;|          or-cedar-creek | 01) crowning disabled | Wed Sep 14 03:20:00 UTC 2022 | Wed Sep 14 09:54:00 UTC 2022 (+ 7hr) |
;;|          or-cedar-creek |          00) original | Wed Sep 14 09:54:00 UTC 2022 | Wed Sep 14 21:18:00 UTC 2022 (+11hr) |
;;|          or-cedar-creek | 01) crowning disabled | Wed Sep 14 09:54:00 UTC 2022 | Wed Sep 14 21:18:00 UTC 2022 (+11hr) |
;;|          or-cedar-creek |          00) original | Wed Sep 14 21:18:00 UTC 2022 | Thu Sep 15 09:37:00 UTC 2022 (+12hr) |
;;|          or-cedar-creek | 01) crowning disabled | Wed Sep 14 21:18:00 UTC 2022 | Thu Sep 15 09:37:00 UTC 2022 (+12hr) |
;;|          or-cedar-creek |          00) original | Thu Sep 15 09:37:00 UTC 2022 | Fri Sep 16 09:15:00 UTC 2022 (+24hr) |
;;|          or-cedar-creek | 01) crowning disabled | Thu Sep 15 09:37:00 UTC 2022 | Fri Sep 16 09:15:00 UTC 2022 (+24hr) |
;;|          or-cedar-creek |          00) original | Fri Sep 16 09:15:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+24hr) |
;;|          or-cedar-creek | 01) crowning disabled | Fri Sep 16 09:15:00 UTC 2022 | Sat Sep 17 08:58:00 UTC 2022 (+24hr) |
;;|          or-cedar-creek |          00) original | Sat Sep 17 08:58:00 UTC 2022 | Sat Sep 17 21:14:00 UTC 2022 (+12hr) |
;;|          or-cedar-creek | 01) crowning disabled | Sat Sep 17 08:58:00 UTC 2022 | Sat Sep 17 21:14:00 UTC 2022 (+12hr) |
;;|          or-cedar-creek |          00) original | Sat Sep 17 21:14:00 UTC 2022 | Mon Sep 19 10:00:00 UTC 2022 (+37hr) |
;;|          or-cedar-creek | 01) crowning disabled | Sat Sep 17 21:14:00 UTC 2022 | Mon Sep 19 10:00:00 UTC 2022 (+37hr) |
;;|          or-cedar-creek |          00) original | Sun Sep 25 22:03:00 UTC 2022 | Mon Sep 26 11:10:00 UTC 2022 (+13hr) |
;;|          or-cedar-creek | 01) crowning disabled | Sun Sep 25 22:03:00 UTC 2022 | Mon Sep 26 11:10:00 UTC 2022 (+13hr) |
;;|          or-cedar-creek |          00) original | Mon Sep 26 11:10:00 UTC 2022 | Tue Sep 27 09:09:00 UTC 2022 (+22hr) |
;;|          or-cedar-creek | 01) crowning disabled | Mon Sep 26 11:10:00 UTC 2022 | Tue Sep 27 09:09:00 UTC 2022 (+22hr) |
;;|         or-double-creek |          00) original | Wed Sep 14 09:05:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|         or-double-creek | 01) crowning disabled | Wed Sep 14 09:05:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|         or-double-creek |          00) original | Thu Sep 15 09:34:00 UTC 2022 | Fri Sep 16 00:06:00 UTC 2022 (+15hr) |
;;|         or-double-creek | 01) crowning disabled | Thu Sep 15 09:34:00 UTC 2022 | Fri Sep 16 00:06:00 UTC 2022 (+15hr) |
;;|         or-double-creek |          00) original | Fri Sep 16 00:06:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+34hr) |
;;|         or-double-creek | 01) crowning disabled | Fri Sep 16 00:06:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+34hr) |
;;|         or-double-creek |          00) original | Sat Sep 17 09:47:00 UTC 2022 | Sat Sep 17 21:14:00 UTC 2022 (+11hr) |
;;|         or-double-creek | 01) crowning disabled | Sat Sep 17 09:47:00 UTC 2022 | Sat Sep 17 21:14:00 UTC 2022 (+11hr) |
;;|         or-double-creek |          00) original | Sat Sep 17 21:14:00 UTC 2022 | Mon Sep 19 09:09:00 UTC 2022 (+36hr) |
;;|         or-double-creek | 01) crowning disabled | Sat Sep 17 21:14:00 UTC 2022 | Mon Sep 19 09:09:00 UTC 2022 (+36hr) |
;;|         or-double-creek |          00) original | Mon Sep 19 09:09:00 UTC 2022 | Tue Sep 20 08:52:00 UTC 2022 (+24hr) |
;;|         or-double-creek | 01) crowning disabled | Mon Sep 19 09:09:00 UTC 2022 | Tue Sep 20 08:52:00 UTC 2022 (+24hr) |
;;|         or-double-creek |          00) original | Tue Sep 20 08:52:00 UTC 2022 | Tue Sep 20 20:19:00 UTC 2022 (+11hr) |
;;|         or-double-creek | 01) crowning disabled | Tue Sep 20 08:52:00 UTC 2022 | Tue Sep 20 20:19:00 UTC 2022 (+11hr) |
;;|         or-double-creek |          00) original | Tue Sep 20 20:19:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+13hr) |
;;|         or-double-creek | 01) crowning disabled | Tue Sep 20 20:19:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+13hr) |
;;|         or-double-creek |          00) original | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|         or-double-creek | 01) crowning disabled | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|         or-double-creek |          00) original | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|         or-double-creek | 01) crowning disabled | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|         or-double-creek |          00) original | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 08:33:00 UTC 2022 (+12hr) |
;;|         or-double-creek | 01) crowning disabled | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 08:33:00 UTC 2022 (+12hr) |
;;|             or-sturgill |          00) original | Wed Sep 14 09:54:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|             or-sturgill | 01) crowning disabled | Wed Sep 14 09:54:00 UTC 2022 | Thu Sep 15 09:34:00 UTC 2022 (+24hr) |
;;|             or-sturgill |          00) original | Thu Sep 15 09:34:00 UTC 2022 | Fri Sep 16 10:09:00 UTC 2022 (+25hr) |
;;|             or-sturgill | 01) crowning disabled | Thu Sep 15 09:34:00 UTC 2022 | Fri Sep 16 10:09:00 UTC 2022 (+25hr) |
;;|             or-sturgill |          00) original | Fri Sep 16 10:09:00 UTC 2022 | Fri Sep 16 20:42:00 UTC 2022 (+11hr) |
;;|             or-sturgill | 01) crowning disabled | Fri Sep 16 10:09:00 UTC 2022 | Fri Sep 16 20:42:00 UTC 2022 (+11hr) |
;;|             or-sturgill |          00) original | Fri Sep 16 20:42:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+13hr) |
;;|             or-sturgill | 01) crowning disabled | Fri Sep 16 20:42:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+13hr) |
;;|             or-sturgill |          00) original | Sat Sep 17 09:47:00 UTC 2022 | Sat Sep 17 16:47:00 UTC 2022 (+ 7hr) |
;;|             or-sturgill | 01) crowning disabled | Sat Sep 17 09:47:00 UTC 2022 | Sat Sep 17 16:47:00 UTC 2022 (+ 7hr) |
;;|             or-sturgill |          00) original | Sat Sep 17 16:47:00 UTC 2022 | Mon Sep 19 17:47:00 UTC 2022 (+49hr) |
;;|             or-sturgill | 01) crowning disabled | Sat Sep 17 16:47:00 UTC 2022 | Mon Sep 19 17:47:00 UTC 2022 (+49hr) |
;;|             or-sturgill |          00) original | Mon Sep 19 17:47:00 UTC 2022 | Tue Sep 20 09:41:00 UTC 2022 (+16hr) |
;;|             or-sturgill | 01) crowning disabled | Mon Sep 19 17:47:00 UTC 2022 | Tue Sep 20 09:41:00 UTC 2022 (+16hr) |
;;|             or-sturgill |          00) original | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 16:04:00 UTC 2022 (+ 6hr) |
;;|             or-sturgill | 01) crowning disabled | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 16:04:00 UTC 2022 (+ 6hr) |
```

```
;;|            or-sturgill |          00) original | Tue Sep 20 16:04:00 UTC 2022 | Tue Sep 20 20:19:00 UTC 2022 (+ 4hr) |
;;|            or-sturgill | 01) crowning disabled | Tue Sep 20 16:04:00 UTC 2022 | Tue Sep 20 20:19:00 UTC 2022 (+ 4hr) |
;;|            or-sturgill |          00) original | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|            or-sturgill | 01) crowning disabled | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|            or-sturgill |          00) original | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|            or-sturgill | 01) crowning disabled | Wed Sep 28 08:50:00 UTC 2022 | Wed Sep 28 20:17:00 UTC 2022 (+11hr) |
;;|            or-sturgill |          00) original | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 00:01:00 UTC 2022 (+ 4hr) |
;;|            or-sturgill | 01) crowning disabled | Wed Sep 28 20:17:00 UTC 2022 | Thu Sep 29 00:01:00 UTC 2022 (+ 4hr) |
;;|          wa-bolt-creek |          00) original | Sun Sep 18 05:10:00 UTC 2022 | Mon Sep 19 10:49:00 UTC 2022 (+30hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Sun Sep 18 05:10:00 UTC 2022 | Mon Sep 19 10:49:00 UTC 2022 (+30hr) |
;;|          wa-bolt-creek |          00) original | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:17:00 UTC 2022 (+11hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:17:00 UTC 2022 (+11hr) |
;;|          wa-bolt-creek |          00) original | Tue Sep 20 20:17:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+13hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Tue Sep 20 20:17:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+13hr) |
;;|          wa-bolt-creek |          00) original | Wed Sep 21 09:22:00 UTC 2022 | Thu Sep 22 09:54:00 UTC 2022 (+25hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Wed Sep 21 09:22:00 UTC 2022 | Thu Sep 22 09:54:00 UTC 2022 (+25hr) |
;;|          wa-bolt-creek |          00) original | Thu Sep 22 09:54:00 UTC 2022 | Thu Sep 22 20:29:00 UTC 2022 (+11hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Thu Sep 22 09:54:00 UTC 2022 | Thu Sep 22 20:29:00 UTC 2022 (+11hr) |
;;|          wa-bolt-creek |          00) original | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|          wa-bolt-creek |          00) original | Sat Sep 24 09:15:00 UTC 2022 | Sun Sep 25 09:45:00 UTC 2022 (+25hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Sat Sep 24 09:15:00 UTC 2022 | Sun Sep 25 09:45:00 UTC 2022 (+25hr) |
;;|          wa-bolt-creek |          00) original | Sun Sep 25 09:45:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+24hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Sun Sep 25 09:45:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+24hr) |
;;|          wa-bolt-creek |          00) original | Mon Sep 26 09:28:00 UTC 2022 | Tue Sep 27 09:07:00 UTC 2022 (+24hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Mon Sep 26 09:28:00 UTC 2022 | Tue Sep 27 09:07:00 UTC 2022 (+24hr) |
;;|          wa-bolt-creek |          00) original | Tue Sep 27 09:07:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+24hr) |
;;|          wa-bolt-creek | 01) crowning disabled | Tue Sep 27 09:07:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+24hr) |
;;|          wa-goat-rocks |          00) original | Sun Sep 25 09:45:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+11hr) |
;;|          wa-goat-rocks | 01) crowning disabled | Sun Sep 25 09:45:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+11hr) |
;;|          wa-goat-rocks |          00) original | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+14hr) |
;;|          wa-goat-rocks | 01) crowning disabled | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+14hr) |
;;|          wa-goat-rocks |          00) original | Mon Sep 26 11:08:00 UTC 2022 | Tue Sep 27 10:00:00 UTC 2022 (+23hr) |
;;|          wa-goat-rocks | 01) crowning disabled | Mon Sep 26 11:08:00 UTC 2022 | Tue Sep 27 10:00:00 UTC 2022 (+23hr) |
;;|          wa-goat-rocks |          00) original | Tue Sep 27 10:00:00 UTC 2022 | Tue Sep 27 21:25:00 UTC 2022 (+11hr) |
;;|          wa-goat-rocks | 01) crowning disabled | Tue Sep 27 10:00:00 UTC 2022 | Tue Sep 27 21:25:00 UTC 2022 (+11hr) |
;;|          wa-goat-rocks |          00) original | Tue Sep 27 21:25:00 UTC 2022 | Wed Sep 28 10:32:00 UTC 2022 (+13hr) |
;;|          wa-goat-rocks | 01) crowning disabled | Tue Sep 27 21:25:00 UTC 2022 | Wed Sep 28 10:32:00 UTC 2022 (+13hr) |
;;|           wa-irving-pk |          00) original | Tue Sep 13 09:22:00 UTC 2022 | Tue Sep 13 20:51:00 UTC 2022 (+11hr) |
;;|           wa-irving-pk | 01) crowning disabled | Tue Sep 13 09:22:00 UTC 2022 | Tue Sep 13 20:51:00 UTC 2022 (+11hr) |
;;|           wa-irving-pk |          00) original | Tue Sep 13 20:51:00 UTC 2022 | Thu Sep 15 10:26:00 UTC 2022 (+38hr) |
;;|           wa-irving-pk | 01) crowning disabled | Tue Sep 13 20:51:00 UTC 2022 | Thu Sep 15 10:26:00 UTC 2022 (+38hr) |
;;|           wa-irving-pk |          00) original | Thu Sep 15 10:26:00 UTC 2022 | Fri Sep 16 09:15:00 UTC 2022 (+23hr) |
;;|           wa-irving-pk | 01) crowning disabled | Thu Sep 15 10:26:00 UTC 2022 | Fri Sep 16 09:15:00 UTC 2022 (+23hr) |
;;|           wa-irving-pk |          00) original | Fri Sep 16 09:15:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+25hr) |
;;|           wa-irving-pk | 01) crowning disabled | Fri Sep 16 09:15:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+25hr) |
;;|           wa-irving-pk |          00) original | Sat Sep 17 09:47:00 UTC 2022 | Mon Sep 19 10:00:00 UTC 2022 (+48hr) |
;;|           wa-irving-pk | 01) crowning disabled | Sat Sep 17 09:47:00 UTC 2022 | Mon Sep 19 10:00:00 UTC 2022 (+48hr) |
;;|           wa-irving-pk |          00) original | Mon Sep 19 10:00:00 UTC 2022 | Tue Sep 20 08:50:00 UTC 2022 (+23hr) |
;;|           wa-irving-pk | 01) crowning disabled | Mon Sep 19 10:00:00 UTC 2022 | Tue Sep 20 08:50:00 UTC 2022 (+23hr) |
;;|           wa-irving-pk |          00) original | Tue Sep 20 08:50:00 UTC 2022 | Tue Sep 20 20:17:00 UTC 2022 (+11hr) |
;;|           wa-irving-pk | 01) crowning disabled | Tue Sep 20 08:50:00 UTC 2022 | Tue Sep 20 20:17:00 UTC 2022 (+11hr) |
;;|           wa-irving-pk |          00) original | Tue Sep 20 20:17:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+13hr) |
;;|           wa-irving-pk | 01) crowning disabled | Tue Sep 20 20:17:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+13hr) |
;;|           wa-irving-pk |          00) original | Wed Sep 21 09:22:00 UTC 2022 | Thu Sep 22 09:02:00 UTC 2022 (+24hr) |
;;|           wa-irving-pk | 01) crowning disabled | Wed Sep 21 09:22:00 UTC 2022 | Thu Sep 22 09:02:00 UTC 2022 (+24hr) |
;;|           wa-irving-pk |          00) original | Thu Sep 22 09:02:00 UTC 2022 | Thu Sep 22 20:29:00 UTC 2022 (+11hr) |
;;|           wa-irving-pk | 01) crowning disabled | Thu Sep 22 09:02:00 UTC 2022 | Thu Sep 22 20:29:00 UTC 2022 (+11hr) |
;;|           wa-irving-pk |          00) original | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|           wa-irving-pk | 01) crowning disabled | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|           wa-irving-pk |          00) original | Sat Sep 24 09:15:00 UTC 2022 | Sun Sep 25 08:58:00 UTC 2022 (+24hr) |
;;|           wa-irving-pk | 01) crowning disabled | Sat Sep 24 09:15:00 UTC 2022 | Sun Sep 25 08:58:00 UTC 2022 (+24hr) |
;;|           wa-irving-pk |          00) original | Sun Sep 25 08:58:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+12hr) |
;;|           wa-irving-pk | 01) crowning disabled | Sun Sep 25 08:58:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+12hr) |
;;|           wa-irving-pk |          00) original | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+12hr) |
;;|           wa-irving-pk | 01) crowning disabled | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+12hr) |
;;|           wa-irving-pk |          00) original | Mon Sep 26 09:28:00 UTC 2022 | Tue Sep 27 09:07:00 UTC 2022 (+24hr) |
;;|           wa-irving-pk | 01) crowning disabled | Mon Sep 26 09:28:00 UTC 2022 | Tue Sep 27 09:07:00 UTC 2022 (+24hr) |
```

```
;;|            wa-irving-pk |          00) original | Tue Sep 27 09:07:00 UTC 2022 | Wed Sep 28 10:30:00 UTC 2022 (+25hr) |
;;|            wa-irving-pk | 01) crowning disabled | Tue Sep 27 09:07:00 UTC 2022 | Wed Sep 28 10:30:00 UTC 2022 (+25hr) |
;;|            wa-irving-pk |          00) original | Wed Sep 28 10:30:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled | Wed Sep 28 10:30:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+24hr) |
;;|               wa-kalama |          00) original | Sat Sep 24 11:18:00 UTC 2022 | Sun Sep 25 10:50:00 UTC 2022 (+24hr) |
;;|               wa-kalama | 01) crowning disabled | Sat Sep 24 11:18:00 UTC 2022 | Sun Sep 25 10:50:00 UTC 2022 (+24hr) |
;;|               wa-kalama |          00) original | Sun Sep 25 10:50:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+24hr) |
;;|               wa-kalama | 01) crowning disabled | Sun Sep 25 10:50:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+24hr) |
;;|               wa-kalama |          00) original | Mon Sep 26 11:08:00 UTC 2022 | Tue Sep 27 10:02:00 UTC 2022 (+23hr) |
;;|               wa-kalama | 01) crowning disabled | Mon Sep 26 11:08:00 UTC 2022 | Tue Sep 27 10:02:00 UTC 2022 (+23hr) |
;;|                  wa-kid |          00) original | Tue Sep 13 10:13:00 UTC 2022 | Tue Sep 13 20:51:00 UTC 2022 (+11hr) |
;;|                  wa-kid | 01) crowning disabled | Tue Sep 13 10:13:00 UTC 2022 | Tue Sep 13 20:51:00 UTC 2022 (+11hr) |
;;|                  wa-kid |          00) original | Tue Sep 13 20:51:00 UTC 2022 | Wed Sep 14 09:52:00 UTC 2022 (+13hr) |
;;|                  wa-kid | 01) crowning disabled | Tue Sep 13 20:51:00 UTC 2022 | Wed Sep 14 09:52:00 UTC 2022 (+13hr) |
;;|     wa-mcallister-creek |          00) original | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|     wa-mcallister-creek | 01) crowning disabled | Tue Sep 27 10:49:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+22hr) |
;;|         wa-minnow-ridge |          00) original | Tue Sep 13 10:13:00 UTC 2022 | Wed Sep 14 11:32:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Tue Sep 13 10:13:00 UTC 2022 | Wed Sep 14 11:32:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge |          00) original | Wed Sep 14 11:32:00 UTC 2022 | Wed Sep 14 21:21:00 UTC 2022 (+10hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Wed Sep 14 11:32:00 UTC 2022 | Wed Sep 14 21:21:00 UTC 2022 (+10hr) |
;;|         wa-minnow-ridge |          00) original | Wed Sep 14 21:21:00 UTC 2022 | Fri Sep 16 10:06:00 UTC 2022 (+37hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Wed Sep 14 21:21:00 UTC 2022 | Fri Sep 16 10:06:00 UTC 2022 (+37hr) |
;;|         wa-minnow-ridge |          00) original | Fri Sep 16 10:06:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Fri Sep 16 10:06:00 UTC 2022 | Sat Sep 17 09:47:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original | Sat Sep 17 09:47:00 UTC 2022 | Sun Sep 18 10:17:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Sat Sep 17 09:47:00 UTC 2022 | Sun Sep 18 10:17:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge |          00) original | Sun Sep 18 10:17:00 UTC 2022 | Mon Sep 19 10:49:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Sun Sep 18 10:17:00 UTC 2022 | Mon Sep 19 10:49:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge |          00) original | Mon Sep 19 10:49:00 UTC 2022 | Mon Sep 19 21:25:00 UTC 2022 (+11hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Mon Sep 19 10:49:00 UTC 2022 | Mon Sep 19 21:25:00 UTC 2022 (+11hr) |
;;|         wa-minnow-ridge |          00) original | Mon Sep 19 21:25:00 UTC 2022 | Tue Sep 20 09:41:00 UTC 2022 (+12hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Mon Sep 19 21:25:00 UTC 2022 | Tue Sep 20 09:41:00 UTC 2022 (+12hr) |
;;|         wa-minnow-ridge |          00) original | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:17:00 UTC 2022 (+11hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Tue Sep 20 09:41:00 UTC 2022 | Tue Sep 20 20:17:00 UTC 2022 (+11hr) |
;;|         wa-minnow-ridge |          00) original | Tue Sep 20 20:17:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+13hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Tue Sep 20 20:17:00 UTC 2022 | Wed Sep 21 09:22:00 UTC 2022 (+13hr) |
;;|         wa-minnow-ridge |          00) original | Wed Sep 21 09:22:00 UTC 2022 | Thu Sep 22 10:43:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Wed Sep 21 09:22:00 UTC 2022 | Thu Sep 22 10:43:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge |          00) original | Thu Sep 22 10:43:00 UTC 2022 | Thu Sep 22 20:29:00 UTC 2022 (+10hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Thu Sep 22 10:43:00 UTC 2022 | Thu Sep 22 20:29:00 UTC 2022 (+10hr) |
;;|         wa-minnow-ridge |          00) original | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Thu Sep 22 20:29:00 UTC 2022 | Sat Sep 24 09:15:00 UTC 2022 (+37hr) |
;;|         wa-minnow-ridge |          00) original | Sat Sep 24 09:15:00 UTC 2022 | Sun Sep 25 08:58:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Sat Sep 24 09:15:00 UTC 2022 | Sun Sep 25 08:58:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original | Sun Sep 25 08:58:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+12hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Sun Sep 25 08:58:00 UTC 2022 | Sun Sep 25 21:14:00 UTC 2022 (+12hr) |
;;|         wa-minnow-ridge |          00) original | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+12hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Sun Sep 25 21:14:00 UTC 2022 | Mon Sep 26 09:28:00 UTC 2022 (+12hr) |
;;|         wa-minnow-ridge |          00) original | Mon Sep 26 09:28:00 UTC 2022 | Tue Sep 27 09:07:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Mon Sep 26 09:28:00 UTC 2022 | Tue Sep 27 09:07:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original | Tue Sep 27 09:07:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Tue Sep 27 09:07:00 UTC 2022 | Wed Sep 28 08:50:00 UTC 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original | Wed Sep 28 08:50:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled | Wed Sep 28 08:50:00 UTC 2022 | Thu Sep 29 10:11:00 UTC 2022 (+25hr) |
;;|             wa-siouoxon |          00) original | Sat Sep 24 20:42:00 UTC 2022 | Sun Sep 25 11:28:00 UTC 2022 (+15hr) |
;;|             wa-siouoxon | 01) crowning disabled | Sat Sep 24 20:42:00 UTC 2022 | Sun Sep 25 11:28:00 UTC 2022 (+15hr) |
;;|             wa-siouoxon |          00) original | Sun Sep 25 11:28:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+24hr) |
;;|             wa-siouoxon | 01) crowning disabled | Sun Sep 25 11:28:00 UTC 2022 | Mon Sep 26 11:08:00 UTC 2022 (+24hr) |
;;|             wa-siouoxon |          00) original | Mon Sep 26 11:08:00 UTC 2022 | Tue Sep 27 10:02:00 UTC 2022 (+23hr) |
;;|             wa-siouoxon | 01) crowning disabled | Mon Sep 26 11:08:00 UTC 2022 | Tue Sep 27 10:02:00 UTC 2022 (+23hr) |
;;|             wa-siouoxon |          00) original | Tue Sep 27 10:02:00 UTC 2022 | Wed Sep 28 10:32:00 UTC 2022 (+25hr) |
;;|             wa-siouoxon | 01) crowning disabled | Tue Sep 27 10:02:00 UTC 2022 | Wed Sep 28 10:32:00 UTC 2022 (+25hr) |
;; Disabling crowning removes most of the over-prediction, while only slightly inflating under-prediction:
(pprint-table-from-kv-lists
 ["Fire name"
  "Variation"
  "sim - real"
  "real - sim"
```

```
  "n cells really burned"
  "sim  real"
  "t0"
  "t1"]
 (replayed-results-table-entries
  replay-results))
;;|              Fire name |             Variation |      sim - real |    real - sim | n cells really burned |    sim  real |
;;|------------------------+-----------------------+-----------------+---------------+-----------------------+--------------+-----
;;|              ca-summit |          00) original |           0 (-) |         0 (-) |                   0.0 |          0 (-) | Wed
;;|              ca-summit | 01) crowning disabled |           0 (-) |         0 (-) |                   0.0 |          0 (-) | Wed
;;|            or-sturgill |          00) original |     210 (23.8%) |   681 (77.0%) |                 884.0 |   203 (23.0%) | Wed
;;|            or-sturgill | 01) crowning disabled |       66 (7.5%) |   681 (77.0%) |                 884.0 |   203 (23.0%) | Wed
;;|         mt-government |          00) original |         0 (0.0%) |   89 (100.0%) |                  89.0 |       0 (0.0%) | Fri
;;|         mt-government | 01) crowning disabled |         0 (0.0%) |   89 (100.0%) |                  89.0 |       0 (0.0%) | Fri
;;|              id-lemhi |          00) original |    2213 (123.6%) |   810 (45.2%) |                1791.0 |    981 (54.8%) | Sun
;;|              id-lemhi | 01) crowning disabled |      128 (7.1%) |   961 (53.7%) |                1791.0 |    830 (46.3%) | Sun
;;|           wa-irving-pk |          00) original |    2846 (419.8%) |   388 (57.2%) |                 678.0 |    290 (42.8%) | Sun
;;|           wa-irving-pk | 01) crowning disabled |       40 (5.9%) |   408 (60.2%) |                 678.0 |    270 (39.8%) | Sun
;;| id-isabella-lower-twin |          00) original |    1065 (179.0%) |   509 (85.5%) |                 595.0 |     86 (14.5%) | Sun
;;| id-isabella-lower-twin | 01) crowning disabled |     121 (20.3%) |   512 (86.1%) |                 595.0 |     83 (13.9%) | Sun
;;|              ca-summit |          00) original |         0 (0.0%) |   14 (100.0%) |                  14.0 |       0 (0.0%) | Thu
;;|              ca-summit | 01) crowning disabled |         0 (0.0%) |   14 (100.0%) |                  14.0 |       0 (0.0%) | Thu
;;|          wa-goat-rocks |          00) original |         0 (0.0%) |    9 (100.0%) |                   9.0 |       0 (0.0%) | Sun
;;|          wa-goat-rocks | 01) crowning disabled |         0 (0.0%) |    9 (100.0%) |                   9.0 |       0 (0.0%) | Sun
;;| id-isabella-lower-twin |          00) original |     313 (67.2%) |   133 (28.5%) |                 466.0 |    333 (71.5%) | Mon
;;| id-isabella-lower-twin | 01) crowning disabled |       26 (5.6%) |   146 (31.3%) |                 466.0 |    320 (68.7%) | Mon
;;|              ca-summit |          00) original |         0 (0.0%) |   49 (100.0%) |                  49.0 |       0 (0.0%) | Mon
;;|              ca-summit | 01) crowning disabled |         0 (0.0%) |   49 (100.0%) |                  49.0 |       0 (0.0%) | Mon
;;|        or-double-creek |          00) original |         0 (0.0%) | 1883 (100.0%) |                1883.0 |       0 (0.0%) | Mon
;;|        or-double-creek | 01) crowning disabled |         0 (0.0%) | 1883 (100.0%) |                1883.0 |       0 (0.0%) | Mon
;;|            mt-billiard |          00) original |  4757 (1336.2%) |   156 (43.8%) |                 356.0 |    200 (56.2%) | Wed
;;|            mt-billiard | 01) crowning disabled |    726 (203.9%) |   165 (46.3%) |                 356.0 |    191 (53.7%) | Wed
;;|                ca-red |          00) original |         0 (0.0%) |   96 (100.0%) |                  96.0 |       0 (0.0%) | Wed
;;|                ca-red | 01) crowning disabled |         0 (0.0%) |   96 (100.0%) |                  96.0 |       0 (0.0%) | Wed
;;|             mt-cannon |          00) original |  1545 (1095.7%) |     12 (8.5%) |                 141.0 |    129 (91.5%) | Thu
;;|             mt-cannon | 01) crowning disabled |   1122 (795.7%) |     12 (8.5%) |                 141.0 |    129 (91.5%) | Thu
;;|          wa-goat-rocks |          00) original |         0 (0.0%) | 1787 (100.0%) |                1787.0 |       0 (0.0%) | Sun
;;|          wa-goat-rocks | 01) crowning disabled |         0 (0.0%) | 1787 (100.0%) |                1787.0 |       0 (0.0%) | Sun
;;|             id-ross-fk |          00) original |    1229 (16.9%) |  5320 (73.0%) |                7284.0 |   1964 (27.0%) | Tue
;;|             id-ross-fk | 01) crowning disabled |     783 (10.7%) |  5325 (73.1%) |                7284.0 |   1959 (26.9%) | Tue
;;|                ca-red |          00) original |         0 (0.0%) |   97 (100.0%) |                  97.0 |       0 (0.0%) | Thu
;;|                ca-red | 01) crowning disabled |         0 (0.0%) |   97 (100.0%) |                  97.0 |       0 (0.0%) | Thu
;;|             ca-summit |          00) original |         2 (0.1%) |   250 (17.9%) |                1396.0 |   1146 (82.1%) | Mon
;;|             ca-summit | 01) crowning disabled |         2 (0.1%) |   250 (17.9%) |                1396.0 |   1146 (82.1%) | Mon
;;|        or-double-creek |          00) original |         0 (0.0%) | 17892 (87.6%) |               20434.0 |   2542 (12.4%) | Wed
;;|        or-double-creek | 01) crowning disabled |         0 (0.0%) | 18178 (89.0%) |               20434.0 |   2256 (11.0%) | Wed
;;|                wa-kid |          00) original |         0 (0.0%) |   78 (100.0%) |                  78.0 |       0 (0.0%) | Tue
;;|                wa-kid | 01) crowning disabled |         0 (0.0%) |   78 (100.0%) |                  78.0 |       0 (0.0%) | Tue
;;|        wa-minnow-ridge |          00) original |  16505 (755.7%) |     35 (1.6%) |                2184.0 |   2149 (98.4%) | Thu
;;|        wa-minnow-ridge | 01) crowning disabled |   3132 (143.4%) |    141 (6.5%) |                2184.0 |   2043 (93.5%) | Thu
;;|                ca-red |          00) original |         0 (0.0%) |   44 (100.0%) |                  44.0 |       0 (0.0%) | Fri
;;|                ca-red | 01) crowning disabled |         0 (0.0%) |   44 (100.0%) |                  44.0 |       0 (0.0%) | Fri
;;|         mt-george-lake |          00) original |   2248 (166.6%) |   571 (42.3%) |                1349.0 |    778 (57.7%) | Tue
;;|         mt-george-lake | 01) crowning disabled |     311 (23.1%) |   592 (43.9%) |                1349.0 |    757 (56.1%) | Tue
;;|              id-lemhi |          00) original |  15492 (773.8%) |    108 (5.4%) |                2002.0 |   1894 (94.6%) | Mon
;;|              id-lemhi | 01) crowning disabled |   9215 (460.3%) |    108 (5.4%) |                2002.0 |   1894 (94.6%) | Mon
;;|           wa-irving-pk |          00) original |    5493 (831.0%) |   235 (35.6%) |                 661.0 |    426 (64.4%) | Mon
;;|           wa-irving-pk | 01) crowning disabled |     191 (28.9%) |   253 (38.3%) |                 661.0 |    408 (61.7%) | Mon
;;|        wa-minnow-ridge |          00) original |    4018 (247.7%) |   259 (16.0%) |                1622.0 |   1363 (84.0%) | Tue
;;|        wa-minnow-ridge | 01) crowning disabled |     226 (13.9%) |   672 (41.4%) |                1622.0 |    950 (58.6%) | Tue
;;|        wa-minnow-ridge |          00) original |    1291 (59.9%) |   367 (17.0%) |                2157.0 |   1790 (83.0%) | Tue
;;|        wa-minnow-ridge | 01) crowning disabled |       69 (3.2%) |   722 (33.5%) |                2157.0 |   1435 (66.5%) | Tue
;;|          mt-government |          00) original |      81 (84.4%) |    36 (37.5%) |                  96.0 |     60 (62.5%) | Sat
;;|          mt-government | 01) crowning disabled |         0 (0.0%) |    43 (44.8%) |                  96.0 |     53 (55.2%) | Sat
;;|              ca-barnes |          00) original |         2 (0.2%) |   561 (63.2%) |                 888.0 |    327 (36.8%) | Tue
;;|              ca-barnes | 01) crowning disabled |         2 (0.2%) |   561 (63.2%) |                 888.0 |    327 (36.8%) | Tue
;;|        wa-minnow-ridge |          00) original |    3105 (135.1%) |    112 (4.9%) |                2298.0 |   2186 (95.1%) | Sun
;;|        wa-minnow-ridge | 01) crowning disabled |     598 (26.0%) |    150 (6.5%) |                2298.0 |   2148 (93.5%) | Sun
```

```
;;|        wa-minnow-ridge |          00) original |  13024 (419.5%) |     97 (3.1%) |          3105.0 |  3008 (96.9%) | Wed
;;|        wa-minnow-ridge | 01) crowning disabled |   1043 (33.6%) |   356 (11.5%) |          3105.0 |  2749 (88.5%) | Wed
;;|           wa-irving-pk |          00) original |    805 (49.7%) |  1016 (62.7%) |          1621.0 |   605 (37.3%) | Mon
;;|           wa-irving-pk | 01) crowning disabled |    407 (25.1%) |  1023 (63.1%) |          1621.0 |   598 (36.9%) | Mon
;;|        or-double-creek |          00) original |   4966 (19.2%) |  6099 (23.6%) |         25821.0 | 19722 (76.4%) | Wed
;;|        or-double-creek | 01) crowning disabled |    995 (3.9%) |  6810 (26.4%) |         25821.0 | 19011 (73.6%) | Wed
;;| id-columbus-bear-gulch |          00) original | 19703 (2976.3%) |   237 (35.8%) |          662.0 |   425 (64.2%) | Wed
;;| id-columbus-bear-gulch | 01) crowning disabled |   1357 (205.0%) |   292 (44.1%) |          662.0 |   370 (55.9%) | Wed
;;|              id-lemhi |          00) original |  13368 (362.9%) |    84 (2.3%) |          3684.0 |  3600 (97.7%) | Wed
;;|              id-lemhi | 01) crowning disabled |   5897 (160.1%) |   532 (14.4%) |          3684.0 |  3152 (85.6%) | Wed
;;|           wa-irving-pk |          00) original |   2802 (302.6%) |   271 (29.3%) |          926.0 |   655 (70.7%) | Wed
;;|           wa-irving-pk | 01) crowning disabled |    303 (32.7%) |   277 (29.9%) |          926.0 |   649 (70.1%) | Wed
;;|           wa-irving-pk |          00) original |   5712 (444.2%) |   942 (73.3%) |         1286.0 |   344 (26.7%) | Sat
;;|           wa-irving-pk | 01) crowning disabled |    702 (54.6%) |   980 (76.2%) |         1286.0 |   306 (23.8%) | Sat
;;|        or-double-creek |          00) original |   5861 (127.7%) |  3192 (69.6%) |         4588.0 |  1396 (30.4%) | Tue
;;|        or-double-creek | 01) crowning disabled |    573 (12.5%) |  3371 (73.5%) |         4588.0 |  1217 (26.5%) | Tue
;;|             ca-summit |          00) original |      0 (0.0%) |    85 (57.8%) |          147.0 |    62 (42.2%) | Sun
;;|             ca-summit | 01) crowning disabled |      0 (0.0%) |    85 (57.8%) |          147.0 |    62 (42.2%) | Sun
;;| id-caledonia-blackburn |          00) original |      0 (0.0%) |  188 (100.0%) |          188.0 |     0 (0.0%) | Wed
;;| id-caledonia-blackburn | 01) crowning disabled |      0 (0.0%) |  188 (100.0%) |          188.0 |     0 (0.0%) | Wed
;;| id-kootenai-ru-complex |          00) original |  10315 (198.9%) |  1282 (24.7%) |         5185.0 |  3903 (75.3%) | Wed
;;| id-kootenai-ru-complex | 01) crowning disabled |   1221 (23.5%) |  1438 (27.7%) |         5185.0 |  3747 (72.3%) | Wed
;;|             ca-summit |          00) original |        0 (-) |       0 (-) |            0.0 |         0 (-) | Tue
;;|             ca-summit | 01) crowning disabled |        0 (-) |       0 (-) |            0.0 |         0 (-) | Tue
;;|          mt-margaret |          00) original |   2611 (384.5%) |    31 (4.6%) |          679.0 |   648 (95.4%) | Thu
;;|          mt-margaret | 01) crowning disabled |   792 (116.6%) |    54 (8.0%) |          679.0 |   625 (92.0%) | Thu
;;|           wa-irving-pk |          00) original |      0 (0.0%) |   81 (100.0%) |           81.0 |     0 (0.0%) | Thu
;;|           wa-irving-pk | 01) crowning disabled |      0 (0.0%) |   81 (100.0%) |           81.0 |     0 (0.0%) | Thu
;;|           wa-siouoxon |          00) original |    271 (63.0%) |    90 (20.9%) |          430.0 |   340 (79.1%) | Sun
;;|           wa-siouoxon | 01) crowning disabled |    184 (42.8%) |   139 (32.3%) |          430.0 |   291 (67.7%) | Sun
;;| id-caledonia-blackburn |          00) original |      0 (0.0%) |  142 (100.0%) |          142.0 |     0 (0.0%) | Sat
;;| id-caledonia-blackburn | 01) crowning disabled |      0 (0.0%) |  142 (100.0%) |          142.0 |     0 (0.0%) | Sat
;;|         or-cedar-creek |          00) original |   1923 (10.1%) | 12723 (66.8%) |         19045.0 |  6322 (33.2%) | Fri
;;|         or-cedar-creek | 01) crowning disabled |   1646 (8.6%) | 13031 (68.4%) |         19045.0 |  6014 (31.6%) | Fri
;;|           wa-irving-pk |          00) original |   9202 (333.4%) |   580 (21.0%) |         2760.0 |  2180 (79.0%) | Thu
;;|           wa-irving-pk | 01) crowning disabled |   2470 (89.5%) |   701 (25.4%) |         2760.0 |  2059 (74.6%) | Thu
;;|         id-trail-ridge |          00) original |      0 (0.0%) |    2 (100.0%) |           2.0 |     0 (0.0%) | Sat
;;|         id-trail-ridge | 01) crowning disabled |      0 (0.0%) |    2 (100.0%) |           2.0 |     0 (0.0%) | Sat
;;| id-kootenai-ru-complex |          00) original |   8137 (228.1%) |   915 (25.7%) |         3567.0 |  2652 (74.3%) | Wed
;;| id-kootenai-ru-complex | 01) crowning disabled |   2989 (83.8%) |   960 (26.9%) |         3567.0 |  2607 (73.1%) | Wed
;;|              id-katka |          00) original |      0 (0.0%) |   12 (100.0%) |           12.0 |     0 (0.0%) | Fri
;;|              id-katka | 01) crowning disabled |      0 (0.0%) |   12 (100.0%) |           12.0 |     0 (0.0%) | Fri
;;|           or-sturgill |          00) original |  15066 (219.8%) |  1349 (19.7%) |         6854.0 |  5505 (80.3%) | Tue
;;|           or-sturgill | 01) crowning disabled |   1469 (21.4%) |  1948 (28.4%) |         6854.0 |  4906 (71.6%) | Tue
;;|         wa-bolt-creek |          00) original |    857 (20.5%) |  1004 (24.0%) |         4179.0 |  3175 (76.0%) | Thu
;;|         wa-bolt-creek | 01) crowning disabled |     14 (0.3%) |  1105 (26.4%) |         4179.0 |  3074 (73.6%) | Thu
;;| id-kootenai-ru-complex |          00) original |      0 (0.0%) |  488 (100.0%) |          488.0 |     0 (0.0%) | Fri
;;| id-kootenai-ru-complex | 01) crowning disabled |      0 (0.0%) |  488 (100.0%) |          488.0 |     0 (0.0%) | Fri
;;|            ca-rodgers |          00) original |      0 (0.0%) |   40 (100.0%) |           40.0 |     0 (0.0%) | Fri
;;|            ca-rodgers | 01) crowning disabled |      0 (0.0%) |   40 (100.0%) |           40.0 |     0 (0.0%) | Fri
;;|        wa-minnow-ridge |          00) original |    447 (18.6%) |   382 (15.9%) |         2408.0 |  2026 (84.1%) | Sun
;;|        wa-minnow-ridge | 01) crowning disabled |     30 (1.2%) |   509 (21.1%) |         2408.0 |  1899 (78.9%) | Sun
;;|         mt-government |          00) original |  1499 (1561.5%) |     9 (9.4%) |          96.0 |    87 (90.6%) | Sat
;;|         mt-government | 01) crowning disabled |    76 (79.2%) |   36 (37.5%) |          96.0 |    60 (62.5%) | Sat
;;|              id-katka |          00) original |   719 (463.9%) |   30 (19.4%) |          155.0 |   125 (80.6%) | Wed
;;|              id-katka | 01) crowning disabled |   205 (132.3%) |   36 (23.2%) |          155.0 |   119 (76.8%) | Wed
;;|         wa-goat-rocks |          00) original |   2110 (72.4%) |  675 (23.1%) |         2916.0 |  2241 (76.9%) | Mon
;;|         wa-goat-rocks | 01) crowning disabled |    879 (30.1%) |  712 (24.4%) |         2916.0 |  2204 (75.6%) | Mon
;;|         id-deep-creek |          00) original |     93 (64.1%) |    73 (50.3%) |          145.0 |    72 (49.7%) | Tue
;;|         id-deep-creek | 01) crowning disabled |     28 (19.3%) |    80 (55.2%) |          145.0 |    65 (44.8%) | Tue
;;|             ca-summit |          00) original |      0 (0.0%) |  1650 (67.0%) |         2461.0 |   811 (33.0%) | Sun
;;|             ca-summit | 01) crowning disabled |      0 (0.0%) |  1650 (67.0%) |         2461.0 |   811 (33.0%) | Sun
;;| id-isabella-lower-twin |          00) original |   1578 (426.5%) |   183 (49.5%) |          370.0 |   187 (50.5%) | Thu
;;| id-isabella-lower-twin | 01) crowning disabled |    210 (56.8%) |   204 (55.1%) |          370.0 |   166 (44.9%) | Thu
;;|            id-tenmile |          00) original |   775 (268.2%) |    47 (16.3%) |          289.0 |   242 (83.7%) | Tue
;;|            id-tenmile | 01) crowning disabled |   538 (186.2%) |    47 (16.3%) |          289.0 |   242 (83.7%) | Tue
;;|          ca-mosquito |          00) original |  47466 (43.6%) | 31656 (29.1%) |        108947.0 | 77291 (70.9%) | Wed
;;|          ca-mosquito | 01) crowning disabled |  15640 (14.4%) | 36248 (33.3%) |        108947.0 | 72699 (66.7%) | Wed
```

```
;; - IMPROVEMENT basic statistics across all rows - mean, stdev
;; filtered on ignition success
;; - pictures - Venn diagram ; 4 colors
;; - wind dir
;; -

(-> replay-results
    (->> (filter #(-> % ::sim-output ::simulations/result)))
    (first)
    :gridfire.lab.replay/t0-fire
    keys sort)

(->> snaps first (keys) (sort))

;; same set-logic stats on canonical tests.
;;

;; 1. Are we spreading too fast?
;; Experiment: compute the average (and maybe some quantiles) simulation-ToA in (sim  real). If it's very small compared to (t1-t0)
;; it's a sign that we're spreading too fast.
(->> replay-results
     (pmap toa-stats)
     (vec))

(def replay-results nil)

;; 2. Idea: find features informative about / correlated with prediction failure.
;; E.g angular distribution from the ignition-centroid -> actual spread centroid vector.

;; 2. Idea: calibrate based on some F-score

;; Idea: detect "failed-to-spread" regions:
;; - at the boundary of (real - sim) and (real  sim)
;; - ToA < t1 (did not fail to spread for lack of time)
;; We can then find features that make these failed-to-spread regions distinctive (non-burnable fuels?).

(def replay-results1 (time (replay-snapshots 20 snaps)))

(let [prof-maps (for [replay-res replay-results
                      [variation-name sim-output] (sort-by key (::sim-outputs replay-res))
                      rec-maps   (->> sim-output ::sampled-values (vals))]
                  (-> rec-maps
                      (->> (reverse) (into (sorted-map)))
                      (merge {"Fire name" (:pyrcst_fire_name replay-res)
                              "Variation" variation-name})
                      (as-> m
                            (let [submap-names ["surface-fire-min"
                                                "surface-fire-max"]]
                              (-> m
                                  (into (for [submap-name submap-names
                                              :let [submap (get m submap-name)]
                                              [k v] submap]
                                          [(str submap-name " " k) v]))
                                  (as-> m (apply dissoc m submap-names))))))))]
  (->> prof-maps (mapcat keys) (into (sorted-set)) (vec))
  (pprint/print-table
   ["Variation"
    "Fire name"

    "canopy-cover"
    "canopy-height"
    "canopy-base-height"
    "fire-line-intensity (kW/m)"
    "critical-intensity (kW/m)"
    "crown-bulk-density"
    "surface-fire-min :reaction-intensity"
```

```
    "residence-time"
    "surface-fire-min :residence-time"

    "slope"

    "temperature"
    "wind-speed-20ft"
    "wind-from-direction"
    "relative-humidity"
    "foliar-moisture (%)"

    "surface-fire-min :fuel-bed-depth"
    "surface-fire-min :heat-of-combustion"

    "aspect"
    "wind-from-direction"
    "max-spread-direction"

    "surface-fire-max :eccentricity"
    "surface-fire-max :effective-wind-speed"
    "surface-fire-max :max-spread-direction"
    "surface-fire-min :spread-rate"
    "surface-fire-max :max-spread-rate"
    "fire-line-intensity"
    "eccentricity"
    "max-spread-rate"]
   (->> prof-maps (sort-by hash) (take 200))))
;;|             Variation |              Fire name | canopy-cover |      canopy-height | canopy-base-height | fire-line-intensity
;;|-----------------------+------------------------+--------------+--------------------+--------------------+--------------------
;;|          00) original |               id-lemhi |          0.0 |                0.0 |                0.0 |
;;| 01) crowning disabled |        wa-minnow-ridge |         45.0 |   62.3390007019043 |   62.3390007019043 |  521.65472612
;;|          00) original |               id-lemhi |          0.0 |                0.0 |                0.0 |
;;|          00) original | id-isabella-lower-twin |         75.0 |  88.58699798583984 | 0.9843000173568726 |  627.4870608
;;|          00) original |            ca-mosquito |         68.0 | 114.83499908447266 |  6.561999797821045 | 505.11910258
;;|          00) original | id-kootenai-rv-complex |         55.0 |  75.46299743652344 |  2.624799966812134 | 145.72627998
;;|          00) original |         or-cedar-creek |         65.0 | 101.71099853515625 |  24.93560028076172 |  53.36342800
;;|          00) original |            ca-mosquito |         21.0 |  39.37200164794922 |  2.952899932861328 |
;;|          00) original |            ca-mosquito |         44.0 |   62.3390007019043 |  3.609100103378296 | 378.03830609
;;| 01) crowning disabled |        wa-minnow-ridge |         75.0 |  88.58699798583984 |  88.58699798583984 |  718.8990519
;;|          00) original | id-columbus-bear-gulch |         45.0 | 36.090999603271484 |  1.312399983406067 |  434.5291695
;;| 01) crowning disabled |            ca-mosquito |         47.0 |  85.30599975585938 |  85.30599975585938 |  512.6550568
;;| 01) crowning disabled |            ca-mosquito |         38.0 |  72.18199920654297 |  72.18199920654297 |
;;| 01) crowning disabled |            ca-mosquito |         67.0 | 111.55400085449219 | 111.55400085449219 |  657.5833756
;;| 01) crowning disabled |            ca-mosquito |          0.0 |                0.0 |                0.0 |
;;|          00) original |            ca-mosquito |         65.0 | 114.83499908447266 |  6.890100002288818 | 1057.1170272
;;|          00) original |        wa-minnow-ridge |         55.0 |  75.46299743652344 |  5.249599933624268 |  395.6676395
;;|          00) original | id-columbus-bear-gulch |         45.0 |  49.21500015258789 | 0.9843000173568726 |  144.5366096
;;|          00) original |            mt-billiard |         55.0 |  75.46299743652344 |  5.249599933624268 | 25.505772950
;;|          00) original | id-columbus-bear-gulch |         15.0 | 36.090999603271484 |  2.952899932861328 |
;;| 01) crowning disabled |            ca-mosquito |         29.0 |   68.9010009765625 |   68.9010009765625 |
;;|          00) original |            ca-mosquito |         67.0 |   82.0250015258789 | 3.9372000694274902 |  634.6987533
;;|          00) original |            ca-mosquito |          0.0 |                0.0 |                0.0 |
;;|          00) original |            ca-mosquito |         52.0 | 108.27300262451172 |  6.890100002288818 | 215.60237201
;;| 01) crowning disabled |               id-lemhi |         15.0 | 36.090999603271484 | 36.090999603271484 |
;;|          00) original |        wa-minnow-ridge |         65.0 | 101.71099853515625 | 7.8744001388549805 | 244.01288189
;;|          00) original |            ca-mosquito |         66.0 |  75.46299743652344 |  3.609100103378296 |  826.6591828
;;|          00) original |          wa-irving-pk |         65.0 |   62.3390007019043 |  2.624799966812134 |  505.5295341
;;|          00) original |            ca-mosquito |         65.0 | 104.99199676513672 |  5.905799865722656 | 1331.1254052
;;|          00) original |         or-cedar-creek |         65.0 | 101.71099853515625 |  7.218200206756592 |  54.87369020
;;|          00) original |            ca-mosquito |         38.0 |  72.18199920654297 |  4.921500205993652 |
;;| 01) crowning disabled |          wa-irving-pk |         55.0 |  88.58699798583984 |  88.58699798583984 |  235.4525107
;;| 01) crowning disabled |               id-lemhi |          0.0 |                0.0 |                0.0 |
;;| 01) crowning disabled |            or-sturgill |         45.0 |  75.46299743652344 |  75.46299743652344 |  58.40930615
;;|          00) original |        wa-minnow-ridge |         65.0 | 101.71099853515625 |  3.609100103378296 |  848.4813331
;;|          00) original |         or-cedar-creek |         65.0 | 127.95899963378906 |  32.48189926147461 | 20.571010698
;;|          00) original |            ca-mosquito |          0.0 |                0.0 |                0.0 |
;;|          00) original |        wa-minnow-ridge |         75.0 |  88.58699798583984 |  5.249599933624268 |  158.8199172
;;|          00) original | id-columbus-bear-gulch |         85.0 |  88.58699798583984 |  4.593400001525879 |  915.9213604
```

```
;;|          00) original |        wa-minnow-ridge |        65.0 |  88.58699798583984 |  5.905799865722656 |        964.0300908
;;|          00) original |               id-lemhi |        45.0 |  49.21500015258789 | 0.9843000173568726 |         423.811173
;;|          00) original |               id-lemhi |        25.0 | 36.090999603271484 | 1.9686000347137451 |
;;|          00) original |               id-lemhi |        55.0 |   62.3390007019043 | 3.9372000694274902 |         767.5179921
;;|          00) original |               id-lemhi |        35.0 |   62.3390007019043 | 7.8744001388549805 |
;;|          00) original |            ca-mosquito |        47.0 |  95.14900207519531 |   6.23390007019043 |         386.6322007
;;|          00) original |          wa-goat-rocks |        75.0 |  88.58699798583984 | 12.139699935913086 |         87.82275482
;;|          00) original |            mt-billiard |        65.0 |  88.58699798583984 |  5.905799865722656 |         858.2109787
;;| 01) crowming disabled |            ca-mosquito |         0.0 |                0.0 |                0.0 |
;;|          00) original | id-kootenai-rv-complex |        55.0 |   62.3390007019043 | 2.2967000007629395 |         623.8228235
;;|          00) original |            ca-mosquito |        19.0 |  65.62000274658203 |  5.249599933624268 |
;;| 01) crowming disabled |               id-lemhi |         0.0 |                0.0 |                0.0 |
;;| 01) crowming disabled |            ca-mosquito |        43.0 |   82.0250015258789 |   82.0250015258789 |        419.8543340
;;|          00) original | id-kootenai-rv-complex |        55.0 |   62.3390007019043 | 1.9686000347137451 |         70.74112932
;;|          00) original |              mt-cannon |        55.0 |  75.46299743652344 |  2.624799966812134 |         193.8835803
;;|          00) original |             or-sturgill |        55.0 |  75.46299743652344 | 1.6404999494552612 |         96.91183758
;;|          00) original | id-kootenai-rv-complex |        55.0 |  49.21500015258789 | 1.6404999494552612 |         380.4218259
;;|          00) original | id-columbus-bear-gulch |        75.0 |  75.46299743652344 | 3.9372000694274902 |         600.8851354
;;|          00) original |          wa-irving-pk |        75.0 | 101.71099853515625 |  6.890100002288818 |        148.90979232
;;|          00) original |               id-lemhi |         0.0 |                0.0 |                0.0 |
;;|          00) original |            ca-mosquito |         0.0 |                0.0 |                0.0 |
;;|          00) original |          wa-goat-rocks |        75.0 |   62.3390007019043 | 7.8744001388549805 |         40.95351745
;;|          00) original |          wa-irving-pk |        35.0 |  75.46299743652344 | 3.2809998989105225 |
;;|          00) original |            ca-mosquito |        51.0 |  78.74400329589844 |  4.593400001525879 |         518.5381117
;;|          00) original |          wa-irving-pk |        65.0 | 114.83499908447266 |  9.514900207519531 |        373.9944609
;;|          00) original |        wa-minnow-ridge |        65.0 |  88.58699798583984 |  4.593400001525879 |         437.0069031
;;|          00) original |             or-sturgill |        35.0 |   62.3390007019043 |  2.624799966812134 |
;;|          00) original |        wa-minnow-ridge |        55.0 |  75.46299743652344 |  5.249599933624268 |         475.1520097
;;|          00) original | id-kootenai-rv-complex |        85.0 |  88.58699798583984 |  4.593400001525879 |         220.1077179
;;| 01) crowming disabled |        or-double-creek |         0.0 |                0.0 |                0.0 |
;;|          00) original |        wa-minnow-ridge |        75.0 | 101.71099853515625 |  6.890100002288818 |        231.3515202
;;|          00) original |            ca-mosquito |        46.0 |  95.14900207519531 |   6.23390007019043 |         503.9229189
;;|          00) original |            ca-mosquito |         0.0 |                0.0 |                0.0 |
;;| 01) crowming disabled |         or-cedar-creek |        75.0 | 101.71099853515625 | 101.71099853515625 |         184.0809093
;;| 01) crowming disabled |             id-tenmile |        55.0 |   62.3390007019043 |   62.3390007019043 |         322.8767805
;;|          00) original |        wa-minnow-ridge |        65.0 |  88.58699798583984 |  5.905799865722656 |         768.5714309
;;|          00) original |        wa-minnow-ridge |        65.0 |  88.58699798583984 | 1.9686000347137451 |         686.279452
;;|          00) original |             id-tenmile |         0.0 |                0.0 |                0.0 |
;;| 01) crowming disabled |               id-lemhi |        15.0 | 22.966999053955078 | 22.966999053955078 |
;;|          00) original | id-columbus-bear-gulch |        55.0 |  75.46299743652344 |  5.249599933624268 |        11.11515242
;;|          00) original |            mt-billiard |        85.0 |  75.46299743652344 |  3.609100103378296 |         835.3285858
;;|          00) original |            ca-mosquito |        55.0 |  75.46299743652344 | 4.2652997970581055 |        510.6166187
;;| 01) crowming disabled |          wa-goat-rocks |        75.0 | 101.71099853515625 | 101.71099853515625 |         653.989808
;;|          00) original |            ca-mosquito |        43.0 | 104.99199676513672 |  17.717399597167397 |         198.7774267
;;|          00) original |        wa-minnow-ridge |        75.0 | 114.83499908447266 | 3.9372000694274902 |        1106.4829827
;;| 01) crowming disabled |               id-lemhi |         0.0 |                0.0 |                0.0 |
;;|          00) original |            ca-mosquito |        45.0 |   62.3390007019043 | 14.436400413513184 |         531.8789225
;;| 01) crowming disabled |            ca-mosquito |         0.0 |                0.0 |                0.0 |
;;| 01) crowming disabled |            ca-mosquito |         0.0 |                0.0 |                0.0 |
;;|          00) original |        or-double-creek |        45.0 |   62.3390007019043 | 1.6404999494552612 |        411.9939004
;;|          00) original |        wa-minnow-ridge |        45.0 |   62.3390007019043 | 2.2967000007629395 |        45.86060316
;;| 01) crowming disabled |            ca-mosquito |        67.0 | 118.11599731445312 | 118.11599731445312 |        2427.822711
;;|          00) original |            ca-mosquito |        36.0 |  59.05799865722656 |  1.312399983406067 |
;;|          00) original |        wa-minnow-ridge |        75.0 |  88.58699798583984 | 0.9843000173568726 |         47.4388216
;;|          00) original |        wa-minnow-ridge |        65.0 |  88.58699798583984 |  5.905799865722656 |         544.3497892
;;| 01) crowming disabled |            ca-mosquito |        58.0 | 111.55400085449219 | 111.55400085449219 |         182.5810740
;;|          00) original |            ca-mosquito |        58.0 |  95.14900207519531 |  5.577700138092041 |         247.9358542
;;|          00) original |               id-lemhi |        15.0 | 22.966999053955078 | 2.2967000007629395 |
;;|          00) original |            ca-mosquito |        61.0 | 111.55400085449219 |  6.890100002288818 |         813.7108607
;;|          00) original |        wa-minnow-ridge |        75.0 | 101.71099853515625 |  6.890100002288818 |         183.0685302
;;| 01) crowming disabled |          wa-irving-pk |        45.0 |  75.46299743652344 |  75.46299743652344 |         295.5272559
;;|          00) original | id-kootenai-rv-complex |        55.0 |   62.3390007019043 | 1.9686000347137451 |         46.70336698
;;|          00) original |          wa-irving-pk |        35.0 |  49.21500015258789 | 3.9372000694274902 |
;;|          00) original |        or-double-creek |        45.0 |  75.46299743652344 |  2.952899932861328 |         156.5239383
;;|          00) original |        wa-minnow-ridge |        75.0 |  88.58699798583984 |  5.249599933624268 |         461.6649959
;;|          00) original |        wa-minnow-ridge |        65.0 |  88.58699798583984 |  4.593400001525879 |        1022.9023931
;;|          00) original | id-columbus-bear-gulch |        35.0 | 36.090999603271484 | 1.6404999494552612 |
```

```
;;|          00) original |        wa-minnow-ridge |  75.0 |  88.58699798583984 |  5.249599933624268 |  417.71753548
;;|          00) original |              id-lemhi |  45.0 |   62.3390007019043 | 3.9372000694274902 |  256.7865924
;;|          00) original |           ca-mosquito |  57.0 | 108.27300262451172 |  6.561999797821045 |   259.662201
;;|          00) original |        wa-minnow-ridge |  45.0 |  75.46299743652344 |  2.952899932861328 |  59.150665496
;;| 01) crowning disabled |              id-lemhi |  15.0 | 36.090999603271484 | 36.090999603271484 |
;;| 01) crowning disabled |           ca-mosquito |  72.0 |  91.86799621582031 |  91.86799621582031 |  1240.760655
;;| 01) crowning disabled |              id-lemhi |  55.0 |   62.3390007019043 |   62.3390007019043 |  387.7699096
;;|          00) original |          wa-irving-pk |  35.0 |   62.3390007019043 |  7.218200206756592 |
;;| 01) crowning disabled |           ca-mosquito |  42.0 |  65.62000274658203 |  65.62000274658203 |  191.0296292
;;|          00) original | id-columbus-bear-gulch |  35.0 |   62.3390007019043 |  4.921500205993652 |
;;|          00) original |           or-sturgill |  45.0 |  75.46299743652344 |  2.952899932861328 |  225.9896494
;;|          00) original |              id-lemhi |   0.0 |                0.0 |                0.0 |
;;|          00) original | id-columbus-bear-gulch |  75.0 |  49.21500015258789 |  1.312399983406067 |  463.5319348
;;| 01) crowning disabled |           or-sturgill |  45.0 |   62.3390007019043 |   62.3390007019043 |  413.6209258
;;| 01) crowning disabled |        wa-minnow-ridge |  45.0 |   62.3390007019043 |   62.3390007019043 |  865.8049010
;;|          00) original |        wa-minnow-ridge |  65.0 |  88.58699798583984 |  5.905799865722656 |  843.3417167
;;|          00) original | id-kootenai-rv-complex |  45.0 |   62.3390007019043 | 1.6404999494552612 |  102.0947044
;;| 01) crowning disabled |           ca-mosquito |  49.0 | 118.11599731445312 | 118.11599731445312 |  1826.12075
;;|          00) original |            mt-cannon |  25.0 |  49.21500015258789 | 1.9686000347137451 |
;;| 01) crowning disabled |              id-lemhi |   0.0 |                0.0 |                0.0 |
;;|          00) original |         mt-george-lake |  45.0 |   62.3390007019043 | 2.2967000007629395 |  2.864780168
;;|          00) original |           ca-mosquito |  52.0 |  95.14900207519531 |  1.312399983406067 |  1074.982813
;;|          00) original |           ca-mosquito |  64.0 | 104.99199676513672 |  5.905799865722656 |  566.7345371
;;| 01) crowning disabled |        wa-minnow-ridge |  45.0 |  75.46299743652344 |  75.46299743652344 |  358.4513182
;;|          00) original |           ca-mosquito |  59.0 | 101.71099853515625 | 0.9843000173568726 |  522.2909635
;;|          00) original |           or-sturgill |  25.0 | 36.090999603271484 | 32.810001373291016 |
;;|          00) original |          wa-bolt-creek |  85.0 |  88.58699798583984 | 0.9843000173568726 |  130.1974655
;;|          00) original | id-isabella-lower-twin |  65.0 |   62.3390007019043 | 3.2809998989105225 |  935.9161923
;;| 01) crowning disabled |           ca-mosquito |  61.0 | 104.99199676513672 | 104.99199676513672 |  1734.6513983
;;|          00) original | id-kootenai-rv-complex |  55.0 |   62.3390007019043 | 0.9843000173568726 |  57.39865291
;;| 01) crowning disabled |        wa-minnow-ridge |  75.0 |  75.46299743652344 |  75.46299743652344 |  232.1658095
;;|          00) original | id-kootenai-rv-complex |  55.0 |   62.3390007019043 | 3.9372000694274902 |  1212.468854
;;|          00) original | id-kootenai-rv-complex |  55.0 |   62.3390007019043 | 2.2967000007629395 |  6.060497318
;;|          00) original |           ca-mosquito |  54.0 |  72.18199920654297 |  9.843000411987305 |  393.9669967
;;|          00) original |        or-double-creek |   0.0 |                0.0 |                0.0 |
;;|          00) original | id-columbus-bear-gulch |  85.0 | 101.71099853515625 |  5.905799865722656 |  524.2505799
;;|          00) original | id-columbus-bear-gulch |  75.0 |  88.58699798583984 |  5.249599933624268 |  725.9359851
;;| 01) crowning disabled |             id-ross-fk |  15.0 | 36.090999603271484 | 36.090999603271484 |
;;|          00) original |           or-sturgill |  45.0 |  75.46299743652344 |  5.905799865722656 |  8.045833795
;;|          00) original | id-kootenai-rv-complex |  55.0 |  88.58699798583984 |  6.561999797821045 |  20.28853351
;;|          00) original |              id-lemhi |  35.0 |  49.21500015258789 | 11.811599731445312 |
;;|          00) original |          wa-irving-pk |  65.0 |  49.21500015258789 | 0.9843000173568726 |  307.4816665
;;|          00) original |           ca-mosquito |  52.0 | 104.99199676513672 |  6.561999797821045 |  1158.935725
;;|          00) original |              id-lemhi |  25.0 |   62.3390007019043 | 11.811599731445312 |
;;| 01) crowning disabled |           ca-mosquito |  22.0 |   68.9010009765625 |   68.9010009765625 |
;;|          00) original |        wa-minnow-ridge |  45.0 |   62.3390007019043 | 2.2967000007629395 |  55.71846745
;;|          00) original |        wa-minnow-ridge |  65.0 | 101.71099853515625 | 7.8744001388549805 |  466.5421048
;;| 01) crowning disabled |              id-lemhi |  15.0 |  49.21500015258789 |  49.21500015258789 |
;;|          00) original |           ca-mosquito |   0.0 |                0.0 |                0.0 |
;;|          00) original | id-columbus-bear-gulch |   0.0 |                0.0 |                0.0 |
;;|          00) original | id-kootenai-rv-complex |  45.0 |  49.21500015258789 | 1.6404999494552612 |  93.10587368
;;|          00) original | id-kootenai-rv-complex |  55.0 |   62.3390007019043 | 2.2967000007629395 |  274.1983516
;;|          00) original |        wa-minnow-ridge |  65.0 |  88.58699798583984 |  2.952899932861328 |  891.6729053
;;| 01) crowning disabled |        or-double-creek |  45.0 |  75.46299743652344 |  75.46299743652344 |  137.3357763
;;| 01) crowning disabled | id-kootenai-rv-complex |  55.0 |   62.3390007019043 |   62.3390007019043 |  20.45895963
;;| 01) crowning disabled |           or-sturgill |  35.0 |   62.3390007019043 |   62.3390007019043 |
;;|          00) original |        wa-minnow-ridge |  55.0 |  75.46299743652344 |  5.249599933624268 |  406.3012818
;;|          00) original |         mt-george-lake |   0.0 |                0.0 |                0.0 |
;;|          00) original |           ca-mosquito |  66.0 | 108.27300262451172 |  6.23390007019043 |  535.7386056
;;|          00) original |        or-double-creek |  35.0 | 36.090999603271484 | 0.9843000173568726 |
;;| 01) crowning disabled |           ca-mosquito |  54.0 | 104.99199676513672 | 104.99199676513672 |  219.9863562
;;|          00) original |              id-lemhi |  35.0 |  49.21500015258789 | 3.2809998989105225 |
;;|          00) original | id-kootenai-rv-complex |  55.0 |  75.46299743652344 |  2.624799966812134 |  367.2571004
;;|          00) original |           ca-mosquito |  48.0 |  78.74400329589844 |  4.921500205993652 |  429.3471411
;;|          00) original |        wa-minnow-ridge |  75.0 |  88.58699798583984 |  5.249599933624268 |  565.5971541
;;|          00) original |           ca-mosquito |  51.0 |  95.14900207519531 |  5.905799865722656 |  203.9154300
;;|          00) original |        wa-minnow-ridge |  45.0 |  49.21500015258789 | 0.9843000173568726 |  316.3366650
;;|          00) original | id-kootenai-rv-complex |  75.0 |  88.58699798583984 |  5.249599933624268 |  255.6555643
```

```
;;| 01) crowning disabled |        wa-minnow-ridge |        35.0 |  75.46299743652344 |  75.46299743652344 |
;;|          00) original |            or-sturgill |        55.0 |  75.46299743652344 |  2.624799966812134 |        184.027510613
;;|          00) original |            ca-mosquito |        75.0 |  78.74400329589844 | 3.2809998989105225 |        1014.79794803
;;|          00) original |               id-lemhi |        35.0 |  49.21500015258789 | 11.811599731445312 |
;;|          00) original |        wa-minnow-ridge |        45.0 |  75.46299743652344 |  2.952899932861328 |        30.001054556
;;|          00) original |           wa-irving-pk |        75.0 |   62.3390007019043 | 4.2652997970581055 |         35.74199007
;;|          00) original |            mt-billiard |        75.0 |  88.58699798583984 |  5.249599933624268 |        192.41824577
;;|          00) original |           wa-irving-pk |        75.0 |   62.3390007019043 | 1.9686000347137451 |        227.69251129
;;|          00) original |            or-sturgill |        45.0 |   62.3390007019043 | 1.6404999494552612 |        207.25999318
;;|          00) original |            ca-mosquito |        56.0 |  85.30599975585938 |  4.921500205993652 |         616.3960885
;;|          00) original |           wa-irving-pk |        65.0 |  75.46299743652344 | 4.2652997970581055 |         339.4286063
;;| 01) crowning disabled |            or-sturgill |        55.0 |  75.46299743652344 |  75.46299743652344 |         5.492232441
;;|          00) original |        or-double-creek |        45.0 |  75.46299743652344 |  2.952899932861328 |         419.1809733
;;|          00) original |               id-lemhi |        35.0 |   62.3390007019043 |  4.921500205993652 |
;;|          00) original | id-columbus-bear-gulch |        85.0 |  75.46299743652344 |  3.609100103378296 |         734.6136999
;;|          00) original |            ca-mosquito |        59.0 | 118.11599731445312 |  7.218200206756592 |        286.50312609
;;|          00) original |            ca-mosquito |        70.0 |  91.86799621582031 |  4.593400001525879 |        205.28253743
;;|          00) original |           wa-irving-pk |        65.0 |   62.3390007019043 |  2.624799966812134 |        268.01702139
;;|          00) original |            ca-mosquito |        71.0 |   82.0250015258789 | 3.9372000694274902 |        303.71064597
;;|          00) original |            ca-mosquito |        66.0 | 118.11599731445312 |  6.890100002288818 |         1077.896712
;;|          00) original |        wa-minnow-ridge |        65.0 |  88.58699798583984 |  5.905799865722656 |         339.9155355
;;| 01) crowning disabled |               id-lemhi |        35.0 |   62.3390007019043 |   62.3390007019043 |
;;|          00) original |           wa-irving-pk |        65.0 |  49.21500015258789 | 0.9843000173568726 |        202.858514570
;;|          00) original | id-kootenai-ru-complex |        55.0 |  49.21500015258789 | 0.9843000173568726 |        102.251646222
;;|          00) original |        wa-minnow-ridge |        45.0 |   62.3390007019043 | 2.2967000007629395 |        37.147984238
(print-inputs-quantiles-tables
 [[:canopy-base-height-matrix "ft" zero?]
  [:canopy-height-matrix      "ft" zero?]]
 (->> replay-results
      (take 16)
      (map
       (fn [replay-res]
         [(-> replay-res :pyrcst_fire_name)
          (-> replay-res ::sim-outputs first val ::simulations/inputs)]))))

;;|      :pyrcst_fire_name |               :input-name |               :min |        :quartile-1 |            :median |         :qu
;;|-------------------------+---------------------------+--------------------+--------------------+--------------------+-----------
;;|              ca-summit | :canopy-base-height-matrix | 0.6561999917030334 | 2.2967000007629395 |  4.593400001525879 |  8.5305995
;;|              ca-summit |      :canopy-height-matrix |  9.843000411987305 |  52.49599838256836 |  75.46299743652344 |  91.867996
;;|            or-sturgill | :canopy-base-height-matrix | 0.9843000173568726 | 1.6404999494552612 |  2.952899932861328 |  4.593400
;;|            or-sturgill |      :canopy-height-matrix |  9.843000411987305 |  49.21500015258789 |   62.3390007019043 |  75.462997
;;|         mt-government | :canopy-base-height-matrix | 0.9843000173568726 | 1.6404999494552612 |  3.609100103378296 |  4.593400
;;|         mt-government |      :canopy-height-matrix |  9.843000411987305 |   62.3390007019043 |   62.3390007019043 |  75.462997
;;|              id-lemhi | :canopy-base-height-matrix | 0.9843000173568726 | 1.9686000347137451 | 3.2809998989105225 |  4.921500
;;|              id-lemhi |      :canopy-height-matrix |  9.843000411987305 |  49.21500015258789 |   62.3390007019043 |   62.3390
;;|           wa-irving-pk | :canopy-base-height-matrix | 0.9843000173568726 | 1.6404999494552612 | 4.2652997970581055 |   6.233900
;;|           wa-irving-pk |      :canopy-height-matrix |  9.843000411987305 |   62.3390007019043 |  75.46299743652344 |  88.586997
;;| id-isabella-lower-twin | :canopy-base-height-matrix | 0.9843000173568726 | 2.2967000007629395 | 3.9372000694274902 |  4.921500
;;| id-isabella-lower-twin |      :canopy-height-matrix |  9.843000411987305 |   62.3390007019043 |   62.3390007019043 |  75.462997
;;|              ca-summit | :canopy-base-height-matrix | 0.6561999917030334 | 2.2967000007629395 |  4.593400001525879 |  8.5305995
;;|              ca-summit |      :canopy-height-matrix |  9.843000411987305 |  52.49599838256836 |  75.46299743652344 |  91.867996
;;|         wa-goat-rocks | :canopy-base-height-matrix | 0.9843000173568726 |  3.609100103378296 |  5.249599933624268 |  6.8901000
;;|         wa-goat-rocks |      :canopy-height-matrix |  9.843000411987305 |   62.3390007019043 |  75.46299743652344 |  88.586997
;;| id-isabella-lower-twin | :canopy-base-height-matrix | 0.9843000173568726 | 2.2967000007629395 | 3.9372000694274902 |  4.921500
;;| id-isabella-lower-twin |      :canopy-height-matrix |  9.843000411987305 |   62.3390007019043 |   62.3390007019043 |  75.462997
;;|              ca-summit | :canopy-base-height-matrix | 0.6561999917030334 | 2.2967000007629395 |  4.593400001525879 |  8.858699
;;|              ca-summit |      :canopy-height-matrix |  9.843000411987305 |  52.49599838256836 |  75.46299743652344 |  91.867996
;;|       or-double-creek | :canopy-base-height-matrix | 0.9843000173568726 | 1.6404999494552612 |  2.952899932861328 |  4.593400
;;|       or-double-creek |      :canopy-height-matrix |  9.843000411987305 |  49.21500015258789 |   62.3390007019043 |  75.462997
;;|           mt-billiard | :canopy-base-height-matrix | 0.9843000173568726 | 2.2967000007629395 | 3.9372000694274902 |  5.249599
;;|           mt-billiard |      :canopy-height-matrix |  9.843000411987305 |   62.3390007019043 |  75.46299743652344 |  75.462997
;;|                ca-red | :canopy-base-height-matrix | 0.6561999917030334 | 1.9686000347137451 |  4.593400001525879 |  8.5305995
;;|                ca-red |      :canopy-height-matrix |  9.843000411987305 |  55.7770004272460 |  75.46299743652344 |  91.867996
;;|             mt-cannon | :canopy-base-height-matrix | 0.9843000173568726 | 1.6404999494552612 | 2.2967000007629395 | 3.9372000
;;|             mt-cannon |      :canopy-height-matrix |  9.843000411987305 |  49.21500015258789 |   62.3390007019043 |   62.3390
;;|         wa-goat-rocks | :canopy-base-height-matrix | 0.9843000173568726 |  3.609100103378296 |  5.249599933624268 |  6.8901000
;;|         wa-goat-rocks |      :canopy-height-matrix |  9.843000411987305 |   62.3390007019043 |  75.46299743652344 |  88.586997
;;|            id-ross-fk | :canopy-base-height-matrix | 0.9843000173568726 | 1.9686000347137451 |  2.624799966812134 |  4.593400
```

```
  ;;|            id-ross-fk |      :canopy-height-matrix |  9.843000411987305 | 36.090999603271484 |  49.21500015258789 |   62.3390

  (->> replay-results
       (map
        (fn [replay-res]
          [(-> replay-res :pyrcst_fire_name)
           (-> replay-res ::sim-outputs first val ::simulations/inputs :spotting)])))

  (def replay-results nil)

  *e)

(defn explore-variations
  [snaps variation-name->info]
  (->> (replayable-successive-t0t1-pairs snaps)
       (sort-by snapshot-pair-hash)
       (partition-all 10) ;; grouping into batches, processed in parallel
       (mapcat
        (fn replay-batch [t0+t1-snaps]
          (let [replay-results        (replay-snapshot-pairs variation-name->info t0+t1-snaps)
                summary-table-entries (replayed-results-table-entries replay-results)]
            (pprint-table-from-kv-lists summary-table-entries)
            summary-table-entries)))
       (vec)))

(defn index-explored-variations
  [kv-lists]
  (let [variation-name->summary-maps (->> kv-lists
                                          (map (fn kv-list->map [k+vs] (into {} k+vs)))
                                          (group-by #(get % "Variation")))]
    variation-name->summary-maps))

(defn weight-favoring
  "A weighting function that will put more weight on duration dt as it gets close to preferred-dt."
  [preferred-dt dt]
  ;; Gamma PDF with k=3 and mode at preferred-dt, re-scaled so that f(preferred-dt)=1.
  ;; There's no deep rationale for using the Gamma PDF - it just roughly has the right shape.
  (let [mode       preferred-dt
        k          3
        k-1        (- k 1.0)
        theta      (/ mode k-1)
        u          (/ (double dt) theta)]
    (* (Math/pow (* u
                    Math/E
                    (/ 1.0 k-1))
                 k-1)
       (Math/exp (- u)))))

(defn aggregate-explored-variations
  [variation-name->summary-maps]
  (letfn [(unformat-count [s] (-> s (str/split #"\s") (first) (read-string)))
          (weight [{dt-hr :delta-t-hr :as _m}]
            (weight-favoring 24.0 dt-hr))
          (weighted-avg [weight-fn score-fn coll]
            (/ (->> coll (map (fn [x] (* (weight-fn x) (score-fn x)))) (apply + 0.0))
               (->> coll (map weight-fn) (apply + 0.0))))
          (failed-ignition? [m]
            (zero? (+ (:sim-inter-obs-n-cells m) (:sim-minus-obs-n-cells m))))]
    (->> variation-name->summary-maps
         (map (fn [[variation-name sumry-maps]]
                (let [ms
                      (->> sumry-maps
                           (map (fn back-to-kws [sumry-map]
                                  (merge (select-keys sumry-map ["Variation" "t1"])
                                         {:pyrcst_fire_name (get sumry-map "Fire name")}
                                         {:observed-burn-n-cells (get sumry-map "n cells really burned")
                                          :sim-inter-obs-n-cells (-> sumry-map (get "sim  real") (unformat-count))
                                          :obs-minus-sim-n-cells (-> sumry-map (get "real - sim") (unformat-count))
```

```
                                        :sim-minus-obs-n-cells (-> sumry-map (get "sim - real") (unformat-count))
                                        :delta-t-hr            (when-some [[_ hrs] (re-matches #".*\(\+\s*(\d+)hr\)" (get sumry-m
                                                                 (-> hrs (Long/parseLong 10) (double)))}))
                        (vec))]
                {"Variation"          variation-name
                 "n replays"          (count ms)
                 "n failed ignitions" (->> ms (filter failed-ignition?) count)
                 "real - sim %-w-avg" (weighted-avg
                                       weight
                                       (fn [m]
                                         (let [real-n-cells (:observed-burn-n-cells m)]
                                           (if (zero? real-n-cells)
                                             0.0
                                             (-> (:obs-minus-sim-n-cells m)
                                                 (/ real-n-cells)
                                                 (* 100.0)))))
                                       (remove failed-ignition? ms))
                 "sim - real %-w-avg" (weighted-avg
                                       weight
                                       (fn [m]
                                         (let [real-n-cells (:observed-burn-n-cells m)]
                                           (if (zero? real-n-cells)
                                             0.0
                                             (-> (:sim-minus-obs-n-cells m)
                                                 (/ real-n-cells)
                                                 (* 100.0)))))
                                       (remove failed-ignition? ms))
                 "mixed error"        (weighted-avg
                                       weight
                                       (fn [m]
                                         (let [real-n-cells (:observed-burn-n-cells m)]
                                           (if (zero? real-n-cells)
                                             0.0
                                             (+
                                              (* 1.0 (-> (:sim-minus-obs-n-cells m) (/ real-n-cells)))
                                              (* 1.0 (-> (:obs-minus-sim-n-cells m) (/ real-n-cells)))))))
                                       ms)})))
        (sort-by #(get % "Variation")))))

(comment

  (def fut_explore-variations
    (future
     (let [variation-name->info  (into [["baseline" {}]]
                                       (for [eaf [1.0 0.5 1.5 0.8 1.2]]
                                         [(format "01) crowning disabled, EAF=%s" (str eaf))
                                          {::transform-inputs-fn (fn [inputs] (-> inputs
                                                                                  (assoc :canopy-base-height-matrix (:canopy-height
                                                                                  (update :ellipse-adjustment-factor-samples
                                                                                          (fn [eaf-samples] (mapv (constantly eaf)
       (explore-variations snaps variation-name->info))))

  (def variation-name->summary-maps (index-explored-variations @fut_explore-variations))

  (pprint/print-table (aggregate-explored-variations aggregate-explored-variations))
  ;;|                      Variation | n replays | n failed ignitions | real - sim %-w-avg | sim - real %-w-avg |        mixed error
  ;;|--------------------------------+-----------+--------------------+--------------------+--------------------+--------------------
  ;;| 01) crowning disabled, EAF=0.5 |       187 |                 58 |  38.43956966815156 |  90.82745015856521 | 1.1644699233103408
  ;;| 01) crowning disabled, EAF=0.8 |       187 |                 58 |  40.19882577624674 | 62.378055824764914 | 0.9784421793703703
  ;;| 01) crowning disabled, EAF=1.0 |       187 |                 58 |  41.06493197803133 |  50.96176395706256 | 0.9049083811943723
  ;;| 01) crowning disabled, EAF=1.2 |       187 |                 58 |  41.77630030878872 |  42.20147817521467 | 0.8488081863612352
  ;;| 01) crowning disabled, EAF=1.5 |       187 |                 58 | 42.697194089929376 |  33.75083916273461 | 0.7963265728909918
  ;;|                       baseline |       187 |                 58 |  35.72564690954984 | 310.89965530770434 | 2.6794366334053863

  ;; IMPROVEMENT: quantiles of baseline - treatment errors.

  ;; IMPROVEMENT new variations to try:
  ;; - Elmfire eccentricity formulas
```

```
  *e)

;; ------------------------------------------------------------------------
;; Images

(defn matrix-bounding-box                                    ;; TODO when support is empty?
  [elem-pred m]
  (let [dtype :double
        row-indices (t/compute-tensor (d/shape m)
                                      (fn [i _j] i)
                                      dtype)
        col-indices (t/compute-tensor (d/shape m)
                                      (fn [_i j] j)
                                      dtype)
        support-row-indices (d/emap
                             (fn [v i]
                               (if (elem-pred v) i Double/NaN))
                             dtype
                             m
                             row-indices)
        support-col-indices (d/emap
                             (fn [v j]
                               (if (elem-pred v) j Double/NaN))
                             dtype
                             m
                             col-indices)]
    [[(long (dfn/reduce-min support-row-indices))
      (long (dfn/reduce-min support-col-indices))]
     [(long (dfn/reduce-max support-row-indices))
      (long (dfn/reduce-max support-col-indices))]]))

(defn clip-matrix-to-bounding-box
  [m [[i0 j0] [i1 j1]]]
  (t/select m (range i0 (inc i1)) (range j0 (inc j1))))


(defn save-matrix-as-png-at-file
  [color-ramp nodata-value matrix file]
  (let [ret (io/file file)
        pixels-per-cell (max 1
                             (let [min-desired-width-px 800
                                   n-cols (-> (d/shape matrix) (nth 1) (long))]
                               (Math/round (double (/ (double min-desired-width-px) n-cols)))))]
    (io/make-parents ret)
    (save-matrix-as-png color-ramp pixels-per-cell nodata-value matrix (-> ret (.toPath) (.toString)))
    ret))


(defn venn-diagram-color-numbers
  [obs? sim? ign?]
  (if ign?
    4.0
    (if obs?
      (if sim? 2.5 1.0)
      (if sim? 5.0 0.0))))

(def <venn-diagram-colors-explanation>
  [:p "The colored map shows the areas that were ignited (yellow), over-predicted (red), under-predicted (blue) and correctly predi


(defn variation-hash
  "Creates a pseudo-id for a replayed variation."
  ^long [replay-res variation-name]
  (hash [(select-keys replay-res [:pyrcst_fire_name
                                  :pyrcst_fire_subnumber])
         (:pyrcst_snapshot_inst (::t0-fire replay-res))
         (:pyrcst_snapshot_inst (::t1-fire replay-res))
```

```
        variation-name]))

(defn generate-images-for-replay!
  [img-dir replay-res]
  (let [observed-burn-area   (::observed-burn-area replay-res)
        observed-burn-matrix (:matrix observed-burn-area)
        stop-inst            (-> replay-res ::t1-fire :pyrcst_snapshot_inst)]
    (reduce
     (fn [replay-res [path v]] (assoc-in replay-res path v))
     replay-res
     (for [[variation-name sim-output] (::sim-outputs replay-res)
           :let [sim-result               (::simulations/result sim-output)
                 sim-inputs               (::simulations/inputs sim-output)
                 sim-burn-area            (if (nil? sim-result)
                                            (:ignition-matrix sim-inputs)
                                            (simulated-burned-area-matrix stop-inst sim-inputs sim-result))
                 burn-venn-diagram-matrix (d/emap
                                           (fn [obs sim ign]
                                             (venn-diagram-color-numbers (pos? obs) (pos? sim) (= 1.0 ign)))
                                           :double
                                           observed-burn-matrix
                                           sim-burn-area
                                           (:ignition-matrix sim-inputs))
                 bbox                     (matrix-bounding-box pos? burn-venn-diagram-matrix)
                 variation-hash (variation-hash replay-res variation-name)
                 subdir (io/file img-dir (str variation-hash))
                 img-name->file           {::fuel-models       (save-matrix-as-png-at-file :gray
                                                                                          -1.0
                                                                                          (-> (:fuel-model-matrix sim-inputs)
                                                                                              (clip-matrix-to-bounding-box bbox)
                                                                                              (d/elemwise-cast :double))
                                                                                          (io/file subdir "fuel-models.png"))
                                           ::burn-venn-diagram (save-matrix-as-png-at-file :color 0.0
                                                                                           (-> burn-venn-diagram-matrix
                                                                                               (clip-matrix-to-bounding-box bbox))
                                                                                           (io/file subdir "burn-venn-diagram.png")
                                           ::toa-hr            (when-some [burn-time-matrix (:burn-time-matrix sim-result)]
                                                                 (let [nodata-value -1.0]
                                                                   (save-matrix-as-png-at-file :gray nodata-value
                                                                                               (-> burn-time-matrix
                                                                                                   (clip-matrix-to-bounding-box bbo
                                                                                                   (->> (d/emap (fn compute-toa-hr
                                                                                                                  (cond-> toa-min (
                                                                                                                nil)))
                                                                                               (io/file subdir "toa-hr.png")))))}]
           [img-k img-file] img-name->file]
       (let [path (conj [::sim-outputs variation-name ::generated-images] img-k)]
         [path img-file])))))

(defn <html-viz-page>
  [webviz-dir replay-results++]
  [:html
   [:body
    [:div
     [:p
      "This page is intended as a benchmark of GridFire misprediction: "
      "it shows the results of replaying PyreCast snapshots against more "
      "recent versions of themselves."]
     [:p "This analysis currently has " [:strong "some weaknesses:"]]
     [:ol
      [:li
       ;; IMPROVEMENT some under-predicted areas are far from the ignition-region:
       ;; in this case, under-predicting them is not GridFire's fault,
       ;; so we don't want those in the misprediction metrics.
       ;; Idea: buffering + graph components. Keep only connected components
       ;; of the really-burned area that intersect with the buffered ignition region.
       ;; Library: loom.
       "Some of the ignition regions computed upstream of GridFire are wrong, "
```

```
      "causing an underprediction which is no fault of GridFire. "
      "TODO: eliminate the influence of those underpredicted regions on misprediction metrics."]
     [:li "In particular, sometimes the ignition region is empty."]
     [:li "The (t0, t1) snapshot pairs are not always optimal. Too close increases the noise caused by poorly-estimated perimeters
   [:div <venn-diagram-colors-explanation>]
   (->> replay-results++
        (map
         (fn [replay-res]
           [:div
            (let [t0-inst (-> replay-res ::t0-fire :pyrcst_snapshot_inst)
                  t1-inst (-> replay-res ::t1-fire :pyrcst_snapshot_inst)]
              [:h2
               (:pyrcst_fire_name replay-res)
               " from "
               (-> t0-inst (pr-str))
               " to "
               (format-hours-between t0-inst t1-inst)])
            (->> replay-res ::sim-outputs
                 (map
                  (fn [[variation-name sim-output]]
                    (letfn [(<img> [img-k]
                              (when-some [img-file (-> sim-output ::generated-images (get img-k))]
                                [:img {:src   (subs (-> img-file (str))
                                                    (-> webviz-dir (str) (count) (inc)))
                                       :style "width: 400px;"}]))]
                      [:div {:style "padding-left: 100px;"}
                       [:h3 {:id (str (variation-hash replay-res variation-name))}
                        variation-name
                        [:a {:href  (str "#" (variation-hash replay-res variation-name))
                             :style "font-size: 8px; margin-left: 5px;"}
                         "link"]]
                       [:div
                        (<img> ::fuel-models)
                        (<img> ::burn-venn-diagram)
                        (<img> ::toa-hr)]])))
                 (doall))])))
        (doall))]])

(comment

  (def webviz-dir (io/file "../gridfire-replay-web"))
  (def img-dir (io/file webviz-dir "img"))

  *e)

(comment

  (def replay-res (nth replay-results2 2))
  (def sim-output (-> replay-res ::sim-outputs (get "00) original")))

  (-> sim-output keys sort)

  (-> sim-output ::simulations/inputs keys sort)

  (def webviz-dir (io/file "../gridfire-replay-web"))


  (-> replay-results2
      (->> (pmap (fn [replay-res] (generate-images-for-replay! img-dir replay-res))))
      (vec)
      (as-> replay-results++
            (spit (io/file img-dir webviz-dir "replay-results2-viz.html")
                  (html/html5
                   (<html-viz-page> webviz-dir replay-results++)))))


  ;; TODO :burn-time-matrix not starting at zero? (min= 21)
  (=
```

```
   #inst"2022-09-14T21:21:00.000-00:00"
   (-> replay-res ::t0-fire :pyrcst_snapshot_inst)
   (-> sim-output ::simulations/inputs :ignition-start-timestamps first))
  ;;=> true
  ;; 21 minutes is also the min of :burn-time-matrix...


  *e)

(comment


  ;; IMPROVEMENT sometimes it looks like the t1 active-fire fills holes inside the t0 active-fire region
  ;; (see e.g "wa-mcallister-creek from #inst "2022-09-27T10:49:00.000-00:00" to +22hr").
  ;; This should probably not be counted as under-prediction.

  (def fut_created-html
    (future
     (spit (io/file webviz-dir "replay-results-viz.html")
           (html/html5
            (<html-viz-page>
             webviz-dir
             (->> (replayable-successive-t0t1-pairs snaps)
                  (sort-by snapshot-pair-hash)
                  (partition-all 16)                        ;; grouping into batches, processed in parallel
                  (mapcat
                   (fn process-batch [t0+t1-snaps]
                     (->> (replay-snapshot-pairs variation-name->info t0+t1-snaps)
                          (pmap (fn [replay-res]
                                  (try
                                    (generate-images-for-replay! img-dir replay-res)
                                    (catch Exception err
                                      (pprint/pprint
                                       (ex-info
                                        (str "error for replay: " (pr-str (:pyrcst_fire_name replay-res)))
                                        {}
                                        err))
                                      nil))))
                          (remove nil?)
                          (doall)))))))))


  *e)
```

9.7.3 gridfire.lab.replay.compare-85f1467-9689124

```
;; FIXME LP coverage ???
(ns gridfire.lab.replay.compare-85f1467-9689124
  "Records a comparison of gridfire.lab.replay results for 2 version of GridFire,
  before and after directional crowning initiation."
  (:require [gridfire.lab.replay :as gflab-replay]))

(defn merge-rr-stats
  [lbl->rr-stats]
  (->> lbl->rr-stats
       (mapcat (fn [[label rr-stats-maps]]
                 (->> rr-stats-maps
                      (map (fn [rr-stats]
                             (update rr-stats
                                     :gridfire.lab.replay/variation-name->stats
                                     (fn rename-keys-w-lbl [vn->s]
                                       (into {}
                                             (map (fn [[variation-name s]]
                                                    [(str variation-name " " label)
                                                     s]))
                                             vn->s))))))))
```

```
      (group-by #(select-keys % [:pyrcst_fire_name
                                 :gridfire.lab.replay/t0-fire
                                 :gridfire.lab.replay/t1-fire]))
      (map (fn [[sim-id ms]]
             (merge sim-id
                    {:gridfire.lab.replay/variation-name->stats (->> ms
                                                                     (map :gridfire.lab.replay/variation-name->stats)
                                                                     (reduce into (sorted-map)))})))))

(comment

  (->> (merge-rr-stats {"OLD" @gflab-replay/rr-stats-85f1467
                        "NEW" @gflab-replay/rr-stats-9689124})
       (shuffle)
       (gflab-replay/print-replayed-results-table))
  ;;|               Fire name |                Variation |                          t0 |                                    t1 | n
  ;;|-------------------------+--------------------------+-----------------------------+---------------------------------------+--
  ;;|         wa-minnow-ridge |          00) original NEW | Wed Sep 14 13:32:00 CEST 2022 | Wed Sep 14 23:21:00 CEST 2022 (+10hr) |
  ;;|         wa-minnow-ridge |          00) original OLD | Wed Sep 14 13:32:00 CEST 2022 | Wed Sep 14 23:21:00 CEST 2022 (+10hr) |
  ;;|         wa-minnow-ridge | 01) crowning disabled NEW | Wed Sep 14 13:32:00 CEST 2022 | Wed Sep 14 23:21:00 CEST 2022 (+10hr) |
  ;;|         wa-minnow-ridge | 01) crowning disabled OLD | Wed Sep 14 13:32:00 CEST 2022 | Wed Sep 14 23:21:00 CEST 2022 (+10hr) |
  ;;|         wa-minnow-ridge |          00) original NEW | Sun Sep 18 12:17:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+25hr) |
  ;;|         wa-minnow-ridge |          00) original OLD | Sun Sep 18 12:17:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+25hr) |
  ;;|         wa-minnow-ridge | 01) crowning disabled NEW | Sun Sep 18 12:17:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+25hr) |
  ;;|         wa-minnow-ridge | 01) crowning disabled OLD | Sun Sep 18 12:17:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+25hr) |
  ;;|            mt-mill-lake |          00) original NEW | Sat Sep 17 12:50:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
  ;;|            mt-mill-lake |          00) original OLD | Sat Sep 17 12:50:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
  ;;|            mt-mill-lake | 01) crowning disabled NEW | Sat Sep 17 12:50:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
  ;;|            mt-mill-lake | 01) crowning disabled OLD | Sat Sep 17 12:50:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
  ;;|  id-kootenai-ru-complex |          00) original NEW | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 14:19:00 CEST 2022 (+ 3hr) |
  ;;|  id-kootenai-ru-complex |          00) original OLD | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 14:19:00 CEST 2022 (+ 3hr) |
  ;;|  id-kootenai-ru-complex | 01) crowning disabled NEW | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 14:19:00 CEST 2022 (+ 3hr) |
  ;;|  id-kootenai-ru-complex | 01) crowning disabled OLD | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 14:19:00 CEST 2022 (+ 3hr) |
  ;;|            wa-irving-pk |          00) original NEW | Sun Sep 25 10:58:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+12hr) |
  ;;|            wa-irving-pk |          00) original OLD | Sun Sep 25 10:58:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+12hr) |
  ;;|            wa-irving-pk | 01) crowning disabled NEW | Sun Sep 25 10:58:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+12hr) |
  ;;|            wa-irving-pk | 01) crowning disabled OLD | Sun Sep 25 10:58:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+12hr) |
  ;;|         wa-minnow-ridge |          00) original NEW | Fri Sep 16 12:06:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+24hr) |
  ;;|         wa-minnow-ridge |          00) original OLD | Fri Sep 16 12:06:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+24hr) |
  ;;|         wa-minnow-ridge | 01) crowning disabled NEW | Fri Sep 16 12:06:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+24hr) |
  ;;|         wa-minnow-ridge | 01) crowning disabled OLD | Fri Sep 16 12:06:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+24hr) |
  ;;|         or-double-creek |          00) original NEW | Sat Sep 17 23:14:00 CEST 2022 | Mon Sep 19 11:09:00 CEST 2022 (+36hr) |
  ;;|         or-double-creek |          00) original OLD | Sat Sep 17 23:14:00 CEST 2022 | Mon Sep 19 11:09:00 CEST 2022 (+36hr) |
  ;;|         or-double-creek | 01) crowning disabled NEW | Sat Sep 17 23:14:00 CEST 2022 | Mon Sep 19 11:09:00 CEST 2022 (+36hr) |
  ;;|         or-double-creek | 01) crowning disabled OLD | Sat Sep 17 23:14:00 CEST 2022 | Mon Sep 19 11:09:00 CEST 2022 (+36hr) |
  ;;|               wa-kalama |          00) original NEW | Sat Sep 24 13:18:00 CEST 2022 | Sun Sep 25 12:50:00 CEST 2022 (+24hr) |
  ;;|               wa-kalama |          00) original OLD | Sat Sep 24 13:18:00 CEST 2022 | Sun Sep 25 12:50:00 CEST 2022 (+24hr) |
  ;;|               wa-kalama | 01) crowning disabled NEW | Sat Sep 24 13:18:00 CEST 2022 | Sun Sep 25 12:50:00 CEST 2022 (+24hr) |
  ;;|               wa-kalama | 01) crowning disabled OLD | Sat Sep 24 13:18:00 CEST 2022 | Sun Sep 25 12:50:00 CEST 2022 (+24hr) |
  ;;|          or-cedar-creek |          00) original NEW | Mon Sep 26 13:10:00 CEST 2022 | Tue Sep 27 11:09:00 CEST 2022 (+22hr) |
  ;;|          or-cedar-creek |          00) original OLD | Mon Sep 26 13:10:00 CEST 2022 | Tue Sep 27 11:09:00 CEST 2022 (+22hr) |
  ;;|          or-cedar-creek | 01) crowning disabled NEW | Mon Sep 26 13:10:00 CEST 2022 | Tue Sep 27 11:09:00 CEST 2022 (+22hr) |
  ;;|          or-cedar-creek | 01) crowning disabled OLD | Mon Sep 26 13:10:00 CEST 2022 | Tue Sep 27 11:09:00 CEST 2022 (+22hr) |
  ;;|  id-isabella-lower-twin |          00) original NEW | Sat Sep 17 10:56:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+24hr) |
  ;;|  id-isabella-lower-twin |          00) original OLD | Sat Sep 17 10:56:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+24hr) |
  ;;|  id-isabella-lower-twin | 01) crowning disabled NEW | Sat Sep 17 10:56:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+24hr) |
  ;;|  id-isabella-lower-twin | 01) crowning disabled OLD | Sat Sep 17 10:56:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+24hr) |
  ;;|         wa-minnow-ridge |          00) original NEW | Tue Sep 13 12:13:00 CEST 2022 | Wed Sep 14 13:32:00 CEST 2022 (+25hr) |
  ;;|         wa-minnow-ridge |          00) original OLD | Tue Sep 13 12:13:00 CEST 2022 | Wed Sep 14 13:32:00 CEST 2022 (+25hr) |
  ;;|         wa-minnow-ridge | 01) crowning disabled NEW | Tue Sep 13 12:13:00 CEST 2022 | Wed Sep 14 13:32:00 CEST 2022 (+25hr) |
  ;;|         wa-minnow-ridge | 01) crowning disabled OLD | Tue Sep 13 12:13:00 CEST 2022 | Wed Sep 14 13:32:00 CEST 2022 (+25hr) |
  ;;|                id-lemhi |          00) original NEW | Tue Sep 27 11:09:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
  ;;|                id-lemhi |          00) original OLD | Tue Sep 27 11:09:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
  ;;|                id-lemhi | 01) crowning disabled NEW | Tue Sep 27 11:09:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
  ;;|                id-lemhi | 01) crowning disabled OLD | Tue Sep 27 11:09:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
  ;;|             or-sturgill |          00) original NEW | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 18:04:00 CEST 2022 (+ 6hr) |
  ;;|             or-sturgill |          00) original OLD | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 18:04:00 CEST 2022 (+ 6hr) |
  ;;|             or-sturgill | 01) crowning disabled NEW | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 18:04:00 CEST 2022 (+ 6hr) |
```

```
;;|            or-sturgill | 01) crowning disabled OLD | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 18:04:00 CEST 2022 (+ 6hr) |
;;|                 ca-red |          00) original NEW | Wed Sep 14 18:50:00 CEST 2022 | Thu Sep 15 12:28:00 CEST 2022 (+18hr) |
;;|                 ca-red |          00) original OLD | Wed Sep 14 18:50:00 CEST 2022 | Thu Sep 15 12:28:00 CEST 2022 (+18hr) |
;;|                 ca-red | 01) crowning disabled NEW | Wed Sep 14 18:50:00 CEST 2022 | Thu Sep 15 12:28:00 CEST 2022 (+18hr) |
;;|                 ca-red | 01) crowning disabled OLD | Wed Sep 14 18:50:00 CEST 2022 | Thu Sep 15 12:28:00 CEST 2022 (+18hr) |
;;|          id-trail-ridge |          00) original NEW | Sat Sep 17 12:23:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
;;|          id-trail-ridge |          00) original OLD | Sat Sep 17 12:23:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
;;|          id-trail-ridge | 01) crowning disabled NEW | Sat Sep 17 12:23:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
;;|          id-trail-ridge | 01) crowning disabled OLD | Sat Sep 17 12:23:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
;;|           wa-bolt-creek |          00) original NEW | Sun Sep 18 07:10:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+30hr) |
;;|           wa-bolt-creek |          00) original OLD | Sun Sep 18 07:10:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+30hr) |
;;|           wa-bolt-creek | 01) crowning disabled NEW | Sun Sep 18 07:10:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+30hr) |
;;|           wa-bolt-creek | 01) crowning disabled OLD | Sun Sep 18 07:10:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+30hr) |
;;|  id-kootenai-rv-complex |          00) original NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+24hr) |
;;|  id-kootenai-rv-complex |          00) original OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+24hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+24hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+24hr) |
;;|              ca-rodgers |          00) original NEW | Fri Sep 16 18:28:00 CEST 2022 | Sat Sep 17 11:49:00 CEST 2022 (+17hr) |
;;|              ca-rodgers |          00) original OLD | Fri Sep 16 18:28:00 CEST 2022 | Sat Sep 17 11:49:00 CEST 2022 (+17hr) |
;;|              ca-rodgers | 01) crowning disabled NEW | Fri Sep 16 18:28:00 CEST 2022 | Sat Sep 17 11:49:00 CEST 2022 (+17hr) |
;;|              ca-rodgers | 01) crowning disabled OLD | Fri Sep 16 18:28:00 CEST 2022 | Sat Sep 17 11:49:00 CEST 2022 (+17hr) |
;;|             or-sturgill |          00) original NEW | Mon Sep 19 19:47:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+16hr) |
;;|             or-sturgill |          00) original OLD | Mon Sep 19 19:47:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+16hr) |
;;|             or-sturgill | 01) crowning disabled NEW | Mon Sep 19 19:47:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+16hr) |
;;|             or-sturgill | 01) crowning disabled OLD | Mon Sep 19 19:47:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+16hr) |
;;|         wa-minnow-ridge |          00) original NEW | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|         wa-minnow-ridge |          00) original OLD | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|         or-double-creek |          00) original NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|         or-double-creek |          00) original OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|                 ca-red |          00) original NEW | Fri Sep 16 11:17:00 CEST 2022 | Fri Sep 16 22:40:00 CEST 2022 (+11hr) |
;;|                 ca-red |          00) original OLD | Fri Sep 16 11:17:00 CEST 2022 | Fri Sep 16 22:40:00 CEST 2022 (+11hr) |
;;|                 ca-red | 01) crowning disabled NEW | Fri Sep 16 11:17:00 CEST 2022 | Fri Sep 16 22:40:00 CEST 2022 (+11hr) |
;;|                 ca-red | 01) crowning disabled OLD | Fri Sep 16 11:17:00 CEST 2022 | Fri Sep 16 22:40:00 CEST 2022 (+11hr) |
;;|                 ca-red |          00) original NEW | Fri Sep 16 22:40:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+12hr) |
;;|                 ca-red |          00) original OLD | Fri Sep 16 22:40:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+12hr) |
;;|                 ca-red | 01) crowning disabled NEW | Fri Sep 16 22:40:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+12hr) |
;;|                 ca-red | 01) crowning disabled OLD | Fri Sep 16 22:40:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+12hr) |
;;|  id-isabella-lower-twin |          00) original NEW | Tue Sep 20 10:52:00 CEST 2022 | Wed Sep 21 00:43:00 CEST 2022 (+14hr) |
;;|  id-isabella-lower-twin |          00) original OLD | Tue Sep 20 10:52:00 CEST 2022 | Wed Sep 21 00:43:00 CEST 2022 (+14hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled NEW | Tue Sep 20 10:52:00 CEST 2022 | Wed Sep 21 00:43:00 CEST 2022 (+14hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled OLD | Tue Sep 20 10:52:00 CEST 2022 | Wed Sep 21 00:43:00 CEST 2022 (+14hr) |
;;|              ca-rodgers |          00) original NEW | Wed Sep 14 11:56:00 CEST 2022 | Thu Sep 15 23:07:00 CEST 2022 (+35hr) |
;;|              ca-rodgers |          00) original OLD | Wed Sep 14 11:56:00 CEST 2022 | Thu Sep 15 23:07:00 CEST 2022 (+35hr) |
;;|              ca-rodgers | 01) crowning disabled NEW | Wed Sep 14 11:56:00 CEST 2022 | Thu Sep 15 23:07:00 CEST 2022 (+35hr) |
;;|              ca-rodgers | 01) crowning disabled OLD | Wed Sep 14 11:56:00 CEST 2022 | Thu Sep 15 23:07:00 CEST 2022 (+35hr) |
;;|             or-sturgill |          00) original NEW | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|             or-sturgill |          00) original OLD | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|             or-sturgill | 01) crowning disabled NEW | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|             or-sturgill | 01) crowning disabled OLD | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|           wa-bolt-creek |          00) original NEW | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
;;|           wa-bolt-creek |          00) original OLD | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
;;|           wa-bolt-creek | 01) crowning disabled NEW | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
;;|           wa-bolt-creek | 01) crowning disabled OLD | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
;;|  id-patrol-point-dismal |          00) original NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;|  id-patrol-point-dismal |          00) original OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;|  id-patrol-point-dismal | 01) crowning disabled NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;|  id-patrol-point-dismal | 01) crowning disabled OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;|             or-sturgill |          00) original NEW | Fri Sep 16 12:09:00 CEST 2022 | Fri Sep 16 22:42:00 CEST 2022 (+11hr) |
;;|             or-sturgill |          00) original OLD | Fri Sep 16 12:09:00 CEST 2022 | Fri Sep 16 22:42:00 CEST 2022 (+11hr) |
;;|             or-sturgill | 01) crowning disabled NEW | Fri Sep 16 12:09:00 CEST 2022 | Fri Sep 16 22:42:00 CEST 2022 (+11hr) |
;;|             or-sturgill | 01) crowning disabled OLD | Fri Sep 16 12:09:00 CEST 2022 | Fri Sep 16 22:42:00 CEST 2022 (+11hr) |
;;|               ca-summit |          00) original NEW | Wed Sep 28 11:43:00 CEST 2022 | Thu Sep 29 12:13:00 CEST 2022 (+25hr) |
;;|               ca-summit |          00) original OLD | Wed Sep 28 11:43:00 CEST 2022 | Thu Sep 29 12:13:00 CEST 2022 (+25hr) |
;;|               ca-summit | 01) crowning disabled NEW | Wed Sep 28 11:43:00 CEST 2022 | Thu Sep 29 12:13:00 CEST 2022 (+25hr) |
```

```
;;|              ca-summit | 01) crowning disabled OLD | Wed Sep 28 11:43:00 CEST 2022 | Thu Sep 29 12:13:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge |          00) original NEW | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+12hr) |
;;|         wa-minnow-ridge |          00) original OLD | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+12hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+12hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+12hr) |
;;|                 wa-kid |          00) original NEW | Tue Sep 13 12:13:00 CEST 2022 | Tue Sep 13 22:51:00 CEST 2022 (+11hr) |
;;|                 wa-kid |          00) original OLD | Tue Sep 13 12:13:00 CEST 2022 | Tue Sep 13 22:51:00 CEST 2022 (+11hr) |
;;|                 wa-kid | 01) crowning disabled NEW | Tue Sep 13 12:13:00 CEST 2022 | Tue Sep 13 22:51:00 CEST 2022 (+11hr) |
;;|                 wa-kid | 01) crowning disabled OLD | Tue Sep 13 12:13:00 CEST 2022 | Tue Sep 13 22:51:00 CEST 2022 (+11hr) |
;;|           id-deep-creek |          00) original NEW | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+11hr) |
;;|           id-deep-creek |          00) original OLD | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+11hr) |
;;|           id-deep-creek | 01) crowning disabled NEW | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+11hr) |
;;|           id-deep-creek | 01) crowning disabled OLD | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+11hr) |
;;| id-lynx-meadows-3-prong |          00) original NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;| id-lynx-meadows-3-prong |          00) original OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;| id-lynx-meadows-3-prong | 01) crowning disabled NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;| id-lynx-meadows-3-prong | 01) crowning disabled OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;|             wa-siouxon |          00) original NEW | Sun Sep 25 13:28:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+24hr) |
;;|             wa-siouxon |          00) original OLD | Sun Sep 25 13:28:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+24hr) |
;;|             wa-siouxon | 01) crowning disabled NEW | Sun Sep 25 13:28:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+24hr) |
;;|             wa-siouxon | 01) crowning disabled OLD | Sun Sep 25 13:28:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+24hr) |
;;|               ca-summit |          00) original NEW | Thu Sep 22 23:18:00 CEST 2022 | Sat Sep 24 11:20:00 CEST 2022 (+36hr) |
;;|               ca-summit |          00) original OLD | Thu Sep 22 23:18:00 CEST 2022 | Sat Sep 24 11:20:00 CEST 2022 (+36hr) |
;;|               ca-summit | 01) crowning disabled NEW | Thu Sep 22 23:18:00 CEST 2022 | Sat Sep 24 11:20:00 CEST 2022 (+36hr) |
;;|               ca-summit | 01) crowning disabled OLD | Thu Sep 22 23:18:00 CEST 2022 | Sat Sep 24 11:20:00 CEST 2022 (+36hr) |
;;|  id-kootenai-rv-complex |          00) original NEW | Wed Sep 14 14:19:00 CEST 2022 | Fri Sep 16 22:02:00 CEST 2022 (+56hr) |
;;|  id-kootenai-rv-complex |          00) original OLD | Wed Sep 14 14:19:00 CEST 2022 | Fri Sep 16 22:02:00 CEST 2022 (+56hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled NEW | Wed Sep 14 14:19:00 CEST 2022 | Fri Sep 16 22:02:00 CEST 2022 (+56hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled OLD | Wed Sep 14 14:19:00 CEST 2022 | Fri Sep 16 22:02:00 CEST 2022 (+56hr) |
;;|                 wa-kid |          00) original NEW | Tue Sep 13 22:51:00 CEST 2022 | Wed Sep 14 11:52:00 CEST 2022 (+13hr) |
;;|                 wa-kid |          00) original OLD | Tue Sep 13 22:51:00 CEST 2022 | Wed Sep 14 11:52:00 CEST 2022 (+13hr) |
;;|                 wa-kid | 01) crowning disabled NEW | Tue Sep 13 22:51:00 CEST 2022 | Wed Sep 14 11:52:00 CEST 2022 (+13hr) |
;;|                 wa-kid | 01) crowning disabled OLD | Tue Sep 13 22:51:00 CEST 2022 | Wed Sep 14 11:52:00 CEST 2022 (+13hr) |
;;|         wa-minnow-ridge |          00) original NEW | Sat Sep 17 11:47:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge |          00) original OLD | Sat Sep 17 11:47:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Sat Sep 17 11:47:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Sat Sep 17 11:47:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+25hr) |
;;|              id-tenmile |          00) original NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 10:33:00 CEST 2022 (+12hr) |
;;|              id-tenmile |          00) original OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 10:33:00 CEST 2022 (+12hr) |
;;|              id-tenmile | 01) crowning disabled NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 10:33:00 CEST 2022 (+12hr) |
;;|              id-tenmile | 01) crowning disabled OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 10:33:00 CEST 2022 (+12hr) |
;;|            wa-irving-pk |          00) original NEW | Thu Sep 15 12:26:00 CEST 2022 | Fri Sep 16 11:15:00 CEST 2022 (+23hr) |
;;|            wa-irving-pk |          00) original OLD | Thu Sep 15 12:26:00 CEST 2022 | Fri Sep 16 11:15:00 CEST 2022 (+23hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Thu Sep 15 12:26:00 CEST 2022 | Fri Sep 16 11:15:00 CEST 2022 (+23hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Thu Sep 15 12:26:00 CEST 2022 | Fri Sep 16 11:15:00 CEST 2022 (+23hr) |
;;|          or-cedar-creek |          00) original NEW | Mon Sep 26 00:03:00 CEST 2022 | Mon Sep 26 13:10:00 CEST 2022 (+13hr) |
;;|          or-cedar-creek |          00) original OLD | Mon Sep 26 00:03:00 CEST 2022 | Mon Sep 26 13:10:00 CEST 2022 (+13hr) |
;;|          or-cedar-creek | 01) crowning disabled NEW | Mon Sep 26 00:03:00 CEST 2022 | Mon Sep 26 13:10:00 CEST 2022 (+13hr) |
;;|          or-cedar-creek | 01) crowning disabled OLD | Mon Sep 26 00:03:00 CEST 2022 | Mon Sep 26 13:10:00 CEST 2022 (+13hr) |
;;|             mt-margaret |          00) original NEW | Wed Sep 14 11:52:00 CEST 2022 | Thu Sep 15 10:43:00 CEST 2022 (+23hr) |
;;|             mt-margaret |          00) original OLD | Wed Sep 14 11:52:00 CEST 2022 | Thu Sep 15 10:43:00 CEST 2022 (+23hr) |
;;|             mt-margaret | 01) crowning disabled NEW | Wed Sep 14 11:52:00 CEST 2022 | Thu Sep 15 10:43:00 CEST 2022 (+23hr) |
;;|             mt-margaret | 01) crowning disabled OLD | Wed Sep 14 11:52:00 CEST 2022 | Thu Sep 15 10:43:00 CEST 2022 (+23hr) |
;;|  id-columbus-bear-gulch |          00) original NEW | Wed Sep 14 23:21:00 CEST 2022 | Sat Sep 17 12:36:00 CEST 2022 (+61hr) |
;;|  id-columbus-bear-gulch |          00) original OLD | Wed Sep 14 23:21:00 CEST 2022 | Sat Sep 17 12:36:00 CEST 2022 (+61hr) |
;;|  id-columbus-bear-gulch | 01) crowning disabled NEW | Wed Sep 14 23:21:00 CEST 2022 | Sat Sep 17 12:36:00 CEST 2022 (+61hr) |
;;|  id-columbus-bear-gulch | 01) crowning disabled OLD | Wed Sep 14 23:21:00 CEST 2022 | Sat Sep 17 12:36:00 CEST 2022 (+61hr) |
;;|  id-kootenai-rv-complex |          00) original NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|  id-kootenai-rv-complex |          00) original OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk |          00) original NEW | Fri Sep 16 11:15:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+25hr) |
;;|            wa-irving-pk |          00) original OLD | Fri Sep 16 11:15:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+25hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Fri Sep 16 11:15:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+25hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Fri Sep 16 11:15:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+25hr) |
;;|           wa-bolt-creek |          00) original NEW | Thu Sep 22 11:54:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+11hr) |
;;|           wa-bolt-creek |          00) original OLD | Thu Sep 22 11:54:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+11hr) |
;;|           wa-bolt-creek | 01) crowning disabled NEW | Thu Sep 22 11:54:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+11hr) |
```

```
;;|           wa-bolt-creek | 01) crowning disabled OLD | Thu Sep 22 11:54:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+11hr) |
;;|             mt-billiard |          00) original NEW | Wed Sep 14 11:52:00 CEST 2022 | Fri Sep 16 13:11:00 CEST 2022 (+49hr) |
;;|             mt-billiard |          00) original OLD | Wed Sep 14 11:52:00 CEST 2022 | Fri Sep 16 13:11:00 CEST 2022 (+49hr) |
;;|             mt-billiard | 01) crowning disabled NEW | Wed Sep 14 11:52:00 CEST 2022 | Fri Sep 16 13:11:00 CEST 2022 (+49hr) |
;;|             mt-billiard | 01) crowning disabled OLD | Wed Sep 14 11:52:00 CEST 2022 | Fri Sep 16 13:11:00 CEST 2022 (+49hr) |
;;|           mt-government |          00) original NEW | Fri Sep 16 13:01:00 CEST 2022 | Sat Sep 17 13:20:00 CEST 2022 (+24hr) |
;;|           mt-government |          00) original OLD | Fri Sep 16 13:01:00 CEST 2022 | Sat Sep 17 13:20:00 CEST 2022 (+24hr) |
;;|           mt-government | 01) crowning disabled NEW | Fri Sep 16 13:01:00 CEST 2022 | Sat Sep 17 13:20:00 CEST 2022 (+24hr) |
;;|           mt-government | 01) crowning disabled OLD | Fri Sep 16 13:01:00 CEST 2022 | Sat Sep 17 13:20:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk |          00) original NEW | Wed Sep 28 12:30:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk |          00) original OLD | Wed Sep 28 12:30:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Wed Sep 28 12:30:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Wed Sep 28 12:30:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+24hr) |
;;|  id-kootenai-ru-complex |          00) original NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 04:03:00 CEST 2022 (+ 6hr) |
;;|  id-kootenai-ru-complex |          00) original OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 04:03:00 CEST 2022 (+ 6hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 04:03:00 CEST 2022 (+ 6hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 04:03:00 CEST 2022 (+ 6hr) |
;;|          or-cedar-creek |          00) original NEW | Sat Sep 17 10:58:00 CEST 2022 | Sat Sep 17 23:14:00 CEST 2022 (+12hr) |
;;|          or-cedar-creek |          00) original OLD | Sat Sep 17 10:58:00 CEST 2022 | Sat Sep 17 23:14:00 CEST 2022 (+12hr) |
;;|          or-cedar-creek | 01) crowning disabled NEW | Sat Sep 17 10:58:00 CEST 2022 | Sat Sep 17 23:14:00 CEST 2022 (+12hr) |
;;|          or-cedar-creek | 01) crowning disabled OLD | Sat Sep 17 10:58:00 CEST 2022 | Sat Sep 17 23:14:00 CEST 2022 (+12hr) |
;;|          or-cedar-creek |          00) original NEW | Wed Sep 14 05:20:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+ 7hr) |
;;|          or-cedar-creek |          00) original OLD | Wed Sep 14 05:20:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+ 7hr) |
;;|          or-cedar-creek | 01) crowning disabled NEW | Wed Sep 14 05:20:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+ 7hr) |
;;|          or-cedar-creek | 01) crowning disabled OLD | Wed Sep 14 05:20:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+ 7hr) |
;;|           wa-bolt-creek |          00) original NEW | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 11:45:00 CEST 2022 (+25hr) |
;;|           wa-bolt-creek |          00) original OLD | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 11:45:00 CEST 2022 (+25hr) |
;;|           wa-bolt-creek | 01) crowning disabled NEW | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 11:45:00 CEST 2022 (+25hr) |
;;|           wa-bolt-creek | 01) crowning disabled OLD | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 11:45:00 CEST 2022 (+25hr) |
;;|             or-sturgill |          00) original NEW | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 12:09:00 CEST 2022 (+25hr) |
;;|             or-sturgill |          00) original OLD | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 12:09:00 CEST 2022 (+25hr) |
;;|             or-sturgill | 01) crowning disabled NEW | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 12:09:00 CEST 2022 (+25hr) |
;;|             or-sturgill | 01) crowning disabled OLD | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 12:09:00 CEST 2022 (+25hr) |
;;|         or-double-creek |          00) original NEW | Mon Sep 19 11:09:00 CEST 2022 | Tue Sep 20 10:52:00 CEST 2022 (+24hr) |
;;|         or-double-creek |          00) original OLD | Mon Sep 19 11:09:00 CEST 2022 | Tue Sep 20 10:52:00 CEST 2022 (+24hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Mon Sep 19 11:09:00 CEST 2022 | Tue Sep 20 10:52:00 CEST 2022 (+24hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Mon Sep 19 11:09:00 CEST 2022 | Tue Sep 20 10:52:00 CEST 2022 (+24hr) |
;;|             ca-mosquito |          00) original NEW | Thu Sep 15 00:01:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+12hr) |
;;|             ca-mosquito |          00) original OLD | Thu Sep 15 00:01:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+12hr) |
;;|             ca-mosquito | 01) crowning disabled NEW | Thu Sep 15 00:01:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+12hr) |
;;|             ca-mosquito | 01) crowning disabled OLD | Thu Sep 15 00:01:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+12hr) |
;;|         wa-minnow-ridge |          00) original NEW | Wed Sep 28 10:50:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge |          00) original OLD | Wed Sep 28 10:50:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Wed Sep 28 10:50:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Wed Sep 28 10:50:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+25hr) |
;;|               ca-summit |          00) original NEW | Wed Sep 21 12:15:00 CEST 2022 | Thu Sep 22 11:58:00 CEST 2022 (+24hr) |
;;|               ca-summit |          00) original OLD | Wed Sep 21 12:15:00 CEST 2022 | Thu Sep 22 11:58:00 CEST 2022 (+24hr) |
;;|               ca-summit | 01) crowning disabled NEW | Wed Sep 21 12:15:00 CEST 2022 | Thu Sep 22 11:58:00 CEST 2022 (+24hr) |
;;|               ca-summit | 01) crowning disabled OLD | Wed Sep 21 12:15:00 CEST 2022 | Thu Sep 22 11:58:00 CEST 2022 (+24hr) |
;;|  id-isabella-lower-twin |          00) original NEW | Wed Sep 21 00:43:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin |          00) original OLD | Wed Sep 21 00:43:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled NEW | Wed Sep 21 00:43:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled OLD | Wed Sep 21 00:43:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+11hr) |
;;|         or-double-creek |          00) original NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 10:33:00 CEST 2022 (+12hr) |
;;|         or-double-creek |          00) original OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 10:33:00 CEST 2022 (+12hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 10:33:00 CEST 2022 (+12hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 10:33:00 CEST 2022 (+12hr) |
;;|             wa-siouoxon |          00) original NEW | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:02:00 CEST 2022 (+23hr) |
;;|             wa-siouoxon |          00) original OLD | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:02:00 CEST 2022 (+23hr) |
;;|             wa-siouoxon | 01) crowning disabled NEW | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:02:00 CEST 2022 (+23hr) |
;;|             wa-siouoxon | 01) crowning disabled OLD | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:02:00 CEST 2022 (+23hr) |
;;|         wa-minnow-ridge |          00) original NEW | Mon Sep 19 12:49:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge |          00) original OLD | Mon Sep 19 12:49:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Mon Sep 19 12:49:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Mon Sep 19 12:49:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|  id-columbus-bear-gulch |          00) original NEW | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 23:21:00 CEST 2022 (+12hr) |
;;|  id-columbus-bear-gulch |          00) original OLD | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 23:21:00 CEST 2022 (+12hr) |
;;|  id-columbus-bear-gulch | 01) crowning disabled NEW | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 23:21:00 CEST 2022 (+12hr) |
```

```
;;|  id-columbus-bear-gulch | 01) crowning disabled OLD | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 23:21:00 CEST 2022 (+12hr) |
;;|            wa-irving-pk |          00) original NEW | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk |          00) original OLD | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|             wa-siouoxon |          00) original NEW | Tue Sep 27 12:02:00 CEST 2022 | Wed Sep 28 12:32:00 CEST 2022 (+25hr) |
;;|             wa-siouoxon |          00) original OLD | Tue Sep 27 12:02:00 CEST 2022 | Wed Sep 28 12:32:00 CEST 2022 (+25hr) |
;;|             wa-siouoxon | 01) crowning disabled NEW | Tue Sep 27 12:02:00 CEST 2022 | Wed Sep 28 12:32:00 CEST 2022 (+25hr) |
;;|             wa-siouoxon | 01) crowning disabled OLD | Tue Sep 27 12:02:00 CEST 2022 | Wed Sep 28 12:32:00 CEST 2022 (+25hr) |
;;|               mt-cannon |          00) original NEW | Thu Sep 15 11:34:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+59hr) |
;;|               mt-cannon |          00) original OLD | Thu Sep 15 11:34:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+59hr) |
;;|               mt-cannon | 01) crowning disabled NEW | Thu Sep 15 11:34:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+59hr) |
;;|               mt-cannon | 01) crowning disabled OLD | Thu Sep 15 11:34:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+59hr) |
;;|            wa-irving-pk |          00) original NEW | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 10:58:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk |          00) original OLD | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 10:58:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 10:58:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 10:58:00 CEST 2022 (+24hr) |
;;|               ca-summit |          00) original NEW | Mon Sep 19 23:23:00 CEST 2022 | Tue Sep 20 10:54:00 CEST 2022 (+12hr) |
;;|               ca-summit |          00) original OLD | Mon Sep 19 23:23:00 CEST 2022 | Tue Sep 20 10:54:00 CEST 2022 (+12hr) |
;;|               ca-summit | 01) crowning disabled NEW | Mon Sep 19 23:23:00 CEST 2022 | Tue Sep 20 10:54:00 CEST 2022 (+12hr) |
;;|               ca-summit | 01) crowning disabled OLD | Mon Sep 19 23:23:00 CEST 2022 | Tue Sep 20 10:54:00 CEST 2022 (+12hr) |
;;|  id-columbus-bear-gulch |          00) original NEW | Tue Sep 13 12:13:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+23hr) |
;;|  id-columbus-bear-gulch |          00) original OLD | Tue Sep 13 12:13:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+23hr) |
;;|  id-columbus-bear-gulch | 01) crowning disabled NEW | Tue Sep 13 12:13:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+23hr) |
;;|  id-columbus-bear-gulch | 01) crowning disabled OLD | Tue Sep 13 12:13:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+23hr) |
;;|           wa-goat-rocks |          00) original NEW | Tue Sep 27 12:00:00 CEST 2022 | Tue Sep 27 23:25:00 CEST 2022 (+11hr) |
;;|           wa-goat-rocks |          00) original OLD | Tue Sep 27 12:00:00 CEST 2022 | Tue Sep 27 23:25:00 CEST 2022 (+11hr) |
;;|           wa-goat-rocks | 01) crowning disabled NEW | Tue Sep 27 12:00:00 CEST 2022 | Tue Sep 27 23:25:00 CEST 2022 (+11hr) |
;;|           wa-goat-rocks | 01) crowning disabled OLD | Tue Sep 27 12:00:00 CEST 2022 | Tue Sep 27 23:25:00 CEST 2022 (+11hr) |
;;|              id-tenmile |          00) original NEW | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|              id-tenmile |          00) original OLD | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|              id-tenmile | 01) crowning disabled NEW | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|              id-tenmile | 01) crowning disabled OLD | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|                id-lemhi |          00) original NEW | Mon Sep 26 12:19:00 CEST 2022 | Tue Sep 27 11:09:00 CEST 2022 (+23hr) |
;;|                id-lemhi |          00) original OLD | Mon Sep 26 12:19:00 CEST 2022 | Tue Sep 27 11:09:00 CEST 2022 (+23hr) |
;;|                id-lemhi | 01) crowning disabled NEW | Mon Sep 26 12:19:00 CEST 2022 | Tue Sep 27 11:09:00 CEST 2022 (+23hr) |
;;|                id-lemhi | 01) crowning disabled OLD | Mon Sep 26 12:19:00 CEST 2022 | Tue Sep 27 11:09:00 CEST 2022 (+23hr) |
;;|                id-lemhi |          00) original NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+14hr) |
;;|                id-lemhi |          00) original OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+14hr) |
;;|                id-lemhi | 01) crowning disabled NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+14hr) |
;;|                id-lemhi | 01) crowning disabled OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 12:11:00 CEST 2022 (+14hr) |
;;|                id-katka |          00) original NEW | Wed Sep 14 14:19:00 CEST 2022 | Fri Sep 16 22:02:00 CEST 2022 (+56hr) |
;;|                id-katka |          00) original OLD | Wed Sep 14 14:19:00 CEST 2022 | Fri Sep 16 22:02:00 CEST 2022 (+56hr) |
;;|                id-katka | 01) crowning disabled NEW | Wed Sep 14 14:19:00 CEST 2022 | Fri Sep 16 22:02:00 CEST 2022 (+56hr) |
;;|                id-katka | 01) crowning disabled OLD | Wed Sep 14 14:19:00 CEST 2022 | Fri Sep 16 22:02:00 CEST 2022 (+56hr) |
;;|         or-double-creek |          00) original NEW | Tue Sep 20 10:52:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+11hr) |
;;|         or-double-creek |          00) original OLD | Tue Sep 20 10:52:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+11hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Tue Sep 20 10:52:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+11hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Tue Sep 20 10:52:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+11hr) |
;;|  id-kootenai-rv-complex |          00) original NEW | Mon Sep 26 12:17:00 CEST 2022 | Tue Sep 27 12:47:00 CEST 2022 (+25hr) |
;;|  id-kootenai-rv-complex |          00) original OLD | Mon Sep 26 12:17:00 CEST 2022 | Tue Sep 27 12:47:00 CEST 2022 (+25hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled NEW | Mon Sep 26 12:17:00 CEST 2022 | Tue Sep 27 12:47:00 CEST 2022 (+25hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled OLD | Mon Sep 26 12:17:00 CEST 2022 | Tue Sep 27 12:47:00 CEST 2022 (+25hr) |
;;|              id-tenmile |          00) original NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|              id-tenmile |          00) original OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|              id-tenmile | 01) crowning disabled NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|              id-tenmile | 01) crowning disabled OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|          or-cedar-creek |          00) original NEW | Sat Sep 17 23:14:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+37hr) |
;;|          or-cedar-creek |          00) original OLD | Sat Sep 17 23:14:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+37hr) |
;;|          or-cedar-creek | 01) crowning disabled NEW | Sat Sep 17 23:14:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+37hr) |
;;|          or-cedar-creek | 01) crowning disabled OLD | Sat Sep 17 23:14:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+37hr) |
;;|                 ca-red |          00) original NEW | Thu Sep 15 12:28:00 CEST 2022 | Thu Sep 15 19:34:00 CEST 2022 (+ 7hr) |
;;|                 ca-red |          00) original OLD | Thu Sep 15 12:28:00 CEST 2022 | Thu Sep 15 19:34:00 CEST 2022 (+ 7hr) |
;;|                 ca-red | 01) crowning disabled NEW | Thu Sep 15 12:28:00 CEST 2022 | Thu Sep 15 19:34:00 CEST 2022 (+ 7hr) |
;;|                 ca-red | 01) crowning disabled OLD | Thu Sep 15 12:28:00 CEST 2022 | Thu Sep 15 19:34:00 CEST 2022 (+ 7hr) |
;;|              mt-cannon |          00) original NEW | Wed Sep 14 11:52:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|              mt-cannon |          00) original OLD | Wed Sep 14 11:52:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|              mt-cannon | 01) crowning disabled NEW | Wed Sep 14 11:52:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
```

```
;;|              mt-cannon | 01) crowning disabled OLD | Wed Sep 14 11:52:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|           wa-bolt-creek |          00) original NEW | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|           wa-bolt-creek |          00) original OLD | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|           wa-bolt-creek | 01) crowning disabled NEW | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|           wa-bolt-creek | 01) crowning disabled OLD | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|            wa-irving-pk |          00) original NEW | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 12:30:00 CEST 2022 (+25hr) |
;;|            wa-irving-pk |          00) original OLD | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 12:30:00 CEST 2022 (+25hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 12:30:00 CEST 2022 (+25hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 12:30:00 CEST 2022 (+25hr) |
;;|            wa-irving-pk |          00) original NEW | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|            wa-irving-pk |          00) original OLD | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|  id-columbus-bear-gulch |          00) original NEW | Sat Sep 17 12:36:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
;;|  id-columbus-bear-gulch |          00) original OLD | Sat Sep 17 12:36:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
;;|  id-columbus-bear-gulch | 01) crowning disabled NEW | Sat Sep 17 12:36:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
;;|  id-columbus-bear-gulch | 01) crowning disabled OLD | Sat Sep 17 12:36:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+10hr) |
;;|         wa-minnow-ridge |          00) original NEW | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|         wa-minnow-ridge |          00) original OLD | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|               ca-barnes |          00) original NEW | Tue Sep 13 15:18:00 CEST 2022 | Tue Sep 13 21:12:00 CEST 2022 (+ 6hr) |
;;|               ca-barnes |          00) original OLD | Tue Sep 13 15:18:00 CEST 2022 | Tue Sep 13 21:12:00 CEST 2022 (+ 6hr) |
;;|               ca-barnes | 01) crowning disabled NEW | Tue Sep 13 15:18:00 CEST 2022 | Tue Sep 13 21:12:00 CEST 2022 (+ 6hr) |
;;|               ca-barnes | 01) crowning disabled OLD | Tue Sep 13 15:18:00 CEST 2022 | Tue Sep 13 21:12:00 CEST 2022 (+ 6hr) |
;;|         or-double-creek |          00) original NEW | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 23:14:00 CEST 2022 (+11hr) |
;;|         or-double-creek |          00) original OLD | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 23:14:00 CEST 2022 (+11hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 23:14:00 CEST 2022 (+11hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 23:14:00 CEST 2022 (+11hr) |
;;|             or-sturgill |          00) original NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|             or-sturgill |          00) original OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|             or-sturgill | 01) crowning disabled NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|             or-sturgill | 01) crowning disabled OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|               ca-summit |          00) original NEW | Sun Sep 25 11:49:00 CEST 2022 | Mon Sep 26 11:32:00 CEST 2022 (+24hr) |
;;|               ca-summit |          00) original OLD | Sun Sep 25 11:49:00 CEST 2022 | Mon Sep 26 11:32:00 CEST 2022 (+24hr) |
;;|               ca-summit | 01) crowning disabled NEW | Sun Sep 25 11:49:00 CEST 2022 | Mon Sep 26 11:32:00 CEST 2022 (+24hr) |
;;|               ca-summit | 01) crowning disabled OLD | Sun Sep 25 11:49:00 CEST 2022 | Mon Sep 26 11:32:00 CEST 2022 (+24hr) |
;;|                  ca-red |          00) original NEW | Thu Sep 15 22:57:00 CEST 2022 | Fri Sep 16 11:17:00 CEST 2022 (+12hr) |
;;|                  ca-red |          00) original OLD | Thu Sep 15 22:57:00 CEST 2022 | Fri Sep 16 11:17:00 CEST 2022 (+12hr) |
;;|                  ca-red | 01) crowning disabled NEW | Thu Sep 15 22:57:00 CEST 2022 | Fri Sep 16 11:17:00 CEST 2022 (+12hr) |
;;|                  ca-red | 01) crowning disabled OLD | Thu Sep 15 22:57:00 CEST 2022 | Fri Sep 16 11:17:00 CEST 2022 (+12hr) |
;;|          mt-government |          00) original NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+24hr) |
;;|          mt-government |          00) original OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+24hr) |
;;|          mt-government | 01) crowning disabled NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+24hr) |
;;|          mt-government | 01) crowning disabled OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+24hr) |
;;|           id-deep-creek |          00) original NEW | Mon Sep 19 12:00:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|           id-deep-creek |          00) original OLD | Mon Sep 19 12:00:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|           id-deep-creek | 01) crowning disabled NEW | Mon Sep 19 12:00:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|           id-deep-creek | 01) crowning disabled OLD | Mon Sep 19 12:00:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin |          00) original NEW | Sun Sep 18 10:37:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+26hr) |
;;|  id-isabella-lower-twin |          00) original OLD | Sun Sep 18 10:37:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+26hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled NEW | Sun Sep 18 10:37:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+26hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled OLD | Sun Sep 18 10:37:00 CEST 2022 | Mon Sep 19 12:49:00 CEST 2022 (+26hr) |
;;|                  ca-red |          00) original NEW | Thu Sep 15 19:34:00 CEST 2022 | Thu Sep 15 22:57:00 CEST 2022 (+ 3hr) |
;;|                  ca-red |          00) original OLD | Thu Sep 15 19:34:00 CEST 2022 | Thu Sep 15 22:57:00 CEST 2022 (+ 3hr) |
;;|                  ca-red | 01) crowning disabled NEW | Thu Sep 15 19:34:00 CEST 2022 | Thu Sep 15 22:57:00 CEST 2022 (+ 3hr) |
;;|                  ca-red | 01) crowning disabled OLD | Thu Sep 15 19:34:00 CEST 2022 | Thu Sep 15 22:57:00 CEST 2022 (+ 3hr) |
;;|         wa-minnow-ridge |          00) original NEW | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 10:58:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original OLD | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 10:58:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 10:58:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Sat Sep 24 11:15:00 CEST 2022 | Sun Sep 25 10:58:00 CEST 2022 (+24hr) |
;;|          or-cedar-creek |          00) original NEW | Thu Sep 15 11:37:00 CEST 2022 | Fri Sep 16 11:15:00 CEST 2022 (+24hr) |
;;|          or-cedar-creek |          00) original OLD | Thu Sep 15 11:37:00 CEST 2022 | Fri Sep 16 11:15:00 CEST 2022 (+24hr) |
;;|          or-cedar-creek | 01) crowning disabled NEW | Thu Sep 15 11:37:00 CEST 2022 | Fri Sep 16 11:15:00 CEST 2022 (+24hr) |
;;|          or-cedar-creek | 01) crowning disabled OLD | Thu Sep 15 11:37:00 CEST 2022 | Fri Sep 16 11:15:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original NEW | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+12hr) |
;;|         wa-minnow-ridge |          00) original OLD | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+12hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+12hr) |
```

```
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+12hr) |
;;|           mt-government |          00) original NEW | Sat Sep 17 13:20:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+ 9hr) |
;;|           mt-government |          00) original OLD | Sat Sep 17 13:20:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+ 9hr) |
;;|           mt-government | 01) crowning disabled NEW | Sat Sep 17 13:20:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+ 9hr) |
;;|           mt-government | 01) crowning disabled OLD | Sat Sep 17 13:20:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+ 9hr) |
;;|               ca-summit |          00) original NEW | Thu Sep 22 11:58:00 CEST 2022 | Thu Sep 22 23:18:00 CEST 2022 (+11hr) |
;;|               ca-summit |          00) original OLD | Thu Sep 22 11:58:00 CEST 2022 | Thu Sep 22 23:18:00 CEST 2022 (+11hr) |
;;|               ca-summit | 01) crowning disabled NEW | Thu Sep 22 11:58:00 CEST 2022 | Thu Sep 22 23:18:00 CEST 2022 (+11hr) |
;;|               ca-summit | 01) crowning disabled OLD | Thu Sep 22 11:58:00 CEST 2022 | Thu Sep 22 23:18:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk |          00) original NEW | Mon Sep 19 12:00:00 CEST 2022 | Tue Sep 20 10:50:00 CEST 2022 (+23hr) |
;;|            wa-irving-pk |          00) original OLD | Mon Sep 19 12:00:00 CEST 2022 | Tue Sep 20 10:50:00 CEST 2022 (+23hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Mon Sep 19 12:00:00 CEST 2022 | Tue Sep 20 10:50:00 CEST 2022 (+23hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Mon Sep 19 12:00:00 CEST 2022 | Tue Sep 20 10:50:00 CEST 2022 (+23hr) |
;;|             or-sturgill |          00) original NEW | Wed Sep 14 11:54:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|             or-sturgill |          00) original OLD | Wed Sep 14 11:54:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|             or-sturgill | 01) crowning disabled NEW | Wed Sep 14 11:54:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|             or-sturgill | 01) crowning disabled OLD | Wed Sep 14 11:54:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|           wa-bolt-creek |          00) original NEW | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|           wa-bolt-creek |          00) original OLD | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|           wa-bolt-creek | 01) crowning disabled NEW | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|           wa-bolt-creek | 01) crowning disabled OLD | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|             ca-mosquito |          00) original NEW | Sat Sep 17 10:58:00 CEST 2022 | Sun Sep 18 00:26:00 CEST 2022 (+13hr) |
;;|             ca-mosquito |          00) original OLD | Sat Sep 17 10:58:00 CEST 2022 | Sun Sep 18 00:26:00 CEST 2022 (+13hr) |
;;|             ca-mosquito | 01) crowning disabled NEW | Sat Sep 17 10:58:00 CEST 2022 | Sun Sep 18 00:26:00 CEST 2022 (+13hr) |
;;|             ca-mosquito | 01) crowning disabled OLD | Sat Sep 17 10:58:00 CEST 2022 | Sun Sep 18 00:26:00 CEST 2022 (+13hr) |
;;|               wa-kalama |          00) original NEW | Sun Sep 25 12:50:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+24hr) |
;;|               wa-kalama |          00) original OLD | Sun Sep 25 12:50:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+24hr) |
;;|               wa-kalama | 01) crowning disabled NEW | Sun Sep 25 12:50:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+24hr) |
;;|               wa-kalama | 01) crowning disabled OLD | Sun Sep 25 12:50:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk |          00) original NEW | Thu Sep 22 11:02:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk |          00) original OLD | Thu Sep 22 11:02:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Thu Sep 22 11:02:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Thu Sep 22 11:02:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge |          00) original NEW | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original OLD | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Mon Sep 26 11:28:00 CEST 2022 | Tue Sep 27 11:07:00 CEST 2022 (+24hr) |
;;|              ca-rodgers |          00) original NEW | Thu Sep 15 23:07:00 CEST 2022 | Fri Sep 16 18:28:00 CEST 2022 (+19hr) |
;;|              ca-rodgers |          00) original OLD | Thu Sep 15 23:07:00 CEST 2022 | Fri Sep 16 18:28:00 CEST 2022 (+19hr) |
;;|              ca-rodgers | 01) crowning disabled NEW | Thu Sep 15 23:07:00 CEST 2022 | Fri Sep 16 18:28:00 CEST 2022 (+19hr) |
;;|              ca-rodgers | 01) crowning disabled OLD | Thu Sep 15 23:07:00 CEST 2022 | Fri Sep 16 18:28:00 CEST 2022 (+19hr) |
;;|  id-isabella-lower-twin |          00) original NEW | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 10:52:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin |          00) original OLD | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 10:52:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled NEW | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 10:52:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled OLD | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 10:52:00 CEST 2022 (+11hr) |
;;|             mt-margaret |          00) original NEW | Sat Sep 17 11:47:00 CEST 2022 | Sun Sep 18 11:28:00 CEST 2022 (+24hr) |
;;|             mt-margaret |          00) original OLD | Sat Sep 17 11:47:00 CEST 2022 | Sun Sep 18 11:28:00 CEST 2022 (+24hr) |
;;|             mt-margaret | 01) crowning disabled NEW | Sat Sep 17 11:47:00 CEST 2022 | Sun Sep 18 11:28:00 CEST 2022 (+24hr) |
;;|             mt-margaret | 01) crowning disabled OLD | Sat Sep 17 11:47:00 CEST 2022 | Sun Sep 18 11:28:00 CEST 2022 (+24hr) |
;;|             or-sturgill |          00) original NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 02:01:00 CEST 2022 (+ 4hr) |
;;|             or-sturgill |          00) original OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 02:01:00 CEST 2022 (+ 4hr) |
;;|             or-sturgill | 01) crowning disabled NEW | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 02:01:00 CEST 2022 (+ 4hr) |
;;|             or-sturgill | 01) crowning disabled OLD | Wed Sep 28 22:17:00 CEST 2022 | Thu Sep 29 02:01:00 CEST 2022 (+ 4hr) |
;;|  id-isabella-lower-twin |          00) original NEW | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|  id-isabella-lower-twin |          00) original OLD | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled NEW | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled OLD | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|         or-double-creek |          00) original NEW | Fri Sep 16 02:06:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+34hr) |
;;|         or-double-creek |          00) original OLD | Fri Sep 16 02:06:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+34hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Fri Sep 16 02:06:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+34hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Fri Sep 16 02:06:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+34hr) |
;;|           wa-goat-rocks |          00) original NEW | Sun Sep 25 11:45:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+11hr) |
;;|           wa-goat-rocks |          00) original OLD | Sun Sep 25 11:45:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+11hr) |
;;|           wa-goat-rocks | 01) crowning disabled NEW | Sun Sep 25 11:45:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+11hr) |
;;|           wa-goat-rocks | 01) crowning disabled OLD | Sun Sep 25 11:45:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+11hr) |
;;|          or-cedar-creek |          00) original NEW | Wed Sep 14 11:54:00 CEST 2022 | Wed Sep 14 23:18:00 CEST 2022 (+11hr) |
;;|          or-cedar-creek |          00) original OLD | Wed Sep 14 11:54:00 CEST 2022 | Wed Sep 14 23:18:00 CEST 2022 (+11hr) |
;;|          or-cedar-creek | 01) crowning disabled NEW | Wed Sep 14 11:54:00 CEST 2022 | Wed Sep 14 23:18:00 CEST 2022 (+11hr) |
```

```
;;|          or-cedar-creek | 01) crowning disabled OLD | Wed Sep 14 11:54:00 CEST 2022 | Wed Sep 14 23:18:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge |          00) original NEW | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge |          00) original OLD | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Tue Sep 27 11:07:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+24hr) |
;;|           wa-goat-rocks |          00) original NEW | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+14hr) |
;;|           wa-goat-rocks |          00) original OLD | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+14hr) |
;;|           wa-goat-rocks | 01) crowning disabled NEW | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+14hr) |
;;|           wa-goat-rocks | 01) crowning disabled OLD | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 13:08:00 CEST 2022 (+14hr) |
;;|             ca-mosquito |          00) original NEW | Wed Sep 14 16:11:00 CEST 2022 | Thu Sep 15 00:01:00 CEST 2022 (+ 8hr) |
;;|             ca-mosquito |          00) original OLD | Wed Sep 14 16:11:00 CEST 2022 | Thu Sep 15 00:01:00 CEST 2022 (+ 8hr) |
;;|             ca-mosquito | 01) crowning disabled NEW | Wed Sep 14 16:11:00 CEST 2022 | Thu Sep 15 00:01:00 CEST 2022 (+ 8hr) |
;;|             ca-mosquito | 01) crowning disabled OLD | Wed Sep 14 16:11:00 CEST 2022 | Thu Sep 15 00:01:00 CEST 2022 (+ 8hr) |
;;|  id-kootenai-ru-complex |          00) original NEW | Fri Sep 16 22:02:00 CEST 2022 | Sun Sep 18 14:04:00 CEST 2022 (+40hr) |
;;|  id-kootenai-ru-complex |          00) original OLD | Fri Sep 16 22:02:00 CEST 2022 | Sun Sep 18 14:04:00 CEST 2022 (+40hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled NEW | Fri Sep 16 22:02:00 CEST 2022 | Sun Sep 18 14:04:00 CEST 2022 (+40hr) |
;;|  id-kootenai-ru-complex | 01) crowning disabled OLD | Fri Sep 16 22:02:00 CEST 2022 | Sun Sep 18 14:04:00 CEST 2022 (+40hr) |
;;|                ca-summit |          00) original NEW | Tue Sep 27 11:11:00 CEST 2022 | Wed Sep 28 11:43:00 CEST 2022 (+25hr) |
;;|                ca-summit |          00) original OLD | Tue Sep 27 11:11:00 CEST 2022 | Wed Sep 28 11:43:00 CEST 2022 (+25hr) |
;;|                ca-summit | 01) crowning disabled NEW | Tue Sep 27 11:11:00 CEST 2022 | Wed Sep 28 11:43:00 CEST 2022 (+25hr) |
;;|                ca-summit | 01) crowning disabled OLD | Tue Sep 27 11:11:00 CEST 2022 | Wed Sep 28 11:43:00 CEST 2022 (+25hr) |
;;|           wa-goat-rocks |          00) original NEW | Tue Sep 27 23:25:00 CEST 2022 | Wed Sep 28 12:32:00 CEST 2022 (+13hr) |
;;|           wa-goat-rocks |          00) original OLD | Tue Sep 27 23:25:00 CEST 2022 | Wed Sep 28 12:32:00 CEST 2022 (+13hr) |
;;|           wa-goat-rocks | 01) crowning disabled NEW | Tue Sep 27 23:25:00 CEST 2022 | Wed Sep 28 12:32:00 CEST 2022 (+13hr) |
;;|           wa-goat-rocks | 01) crowning disabled OLD | Tue Sep 27 23:25:00 CEST 2022 | Wed Sep 28 12:32:00 CEST 2022 (+13hr) |
;;|            wa-irving-pk |          00) original NEW | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 11:02:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk |          00) original OLD | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 11:02:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 11:02:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 11:02:00 CEST 2022 (+24hr) |
;;|           id-deep-creek |          00) original NEW | Sat Sep 17 23:14:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+13hr) |
;;|           id-deep-creek |          00) original OLD | Sat Sep 17 23:14:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+13hr) |
;;|           id-deep-creek | 01) crowning disabled NEW | Sat Sep 17 23:14:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+13hr) |
;;|           id-deep-creek | 01) crowning disabled OLD | Sat Sep 17 23:14:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+13hr) |
;;|          or-cedar-creek |          00) original NEW | Wed Sep 14 23:18:00 CEST 2022 | Thu Sep 15 11:37:00 CEST 2022 (+12hr) |
;;|          or-cedar-creek |          00) original OLD | Wed Sep 14 23:18:00 CEST 2022 | Thu Sep 15 11:37:00 CEST 2022 (+12hr) |
;;|          or-cedar-creek | 01) crowning disabled NEW | Wed Sep 14 23:18:00 CEST 2022 | Thu Sep 15 11:37:00 CEST 2022 (+12hr) |
;;|          or-cedar-creek | 01) crowning disabled OLD | Wed Sep 14 23:18:00 CEST 2022 | Thu Sep 15 11:37:00 CEST 2022 (+12hr) |
;;|         or-double-creek |          00) original NEW | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|         or-double-creek |          00) original OLD | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|             ca-mosquito |          00) original NEW | Fri Sep 16 11:17:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+24hr) |
;;|             ca-mosquito |          00) original OLD | Fri Sep 16 11:17:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+24hr) |
;;|             ca-mosquito | 01) crowning disabled NEW | Fri Sep 16 11:17:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+24hr) |
;;|             ca-mosquito | 01) crowning disabled OLD | Fri Sep 16 11:17:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+24hr) |
;;|             ca-mosquito |          00) original NEW | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 16:11:00 CEST 2022 (+ 5hr) |
;;|             ca-mosquito |          00) original OLD | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 16:11:00 CEST 2022 (+ 5hr) |
;;|             ca-mosquito | 01) crowning disabled NEW | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 16:11:00 CEST 2022 (+ 5hr) |
;;|             ca-mosquito | 01) crowning disabled OLD | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 16:11:00 CEST 2022 (+ 5hr) |
;;|             or-sturgill |          00) original NEW | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 18:47:00 CEST 2022 (+ 7hr) |
;;|             or-sturgill |          00) original OLD | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 18:47:00 CEST 2022 (+ 7hr) |
;;|             or-sturgill | 01) crowning disabled NEW | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 18:47:00 CEST 2022 (+ 7hr) |
;;|             or-sturgill | 01) crowning disabled OLD | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 18:47:00 CEST 2022 (+ 7hr) |
;;|                ca-summit |          00) original NEW | Sat Sep 24 11:20:00 CEST 2022 | Sun Sep 25 11:49:00 CEST 2022 (+24hr) |
;;|                ca-summit |          00) original OLD | Sat Sep 24 11:20:00 CEST 2022 | Sun Sep 25 11:49:00 CEST 2022 (+24hr) |
;;|                ca-summit | 01) crowning disabled NEW | Sat Sep 24 11:20:00 CEST 2022 | Sun Sep 25 11:49:00 CEST 2022 (+24hr) |
;;|                ca-summit | 01) crowning disabled OLD | Sat Sep 24 11:20:00 CEST 2022 | Sun Sep 25 11:49:00 CEST 2022 (+24hr) |
;;|            wa-irving-pk |          00) original NEW | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|            wa-irving-pk |          00) original OLD | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Thu Sep 22 22:29:00 CEST 2022 | Sat Sep 24 11:15:00 CEST 2022 (+37hr) |
;;|         wa-minnow-ridge |          00) original NEW | Wed Sep 14 23:21:00 CEST 2022 | Fri Sep 16 12:06:00 CEST 2022 (+37hr) |
;;|         wa-minnow-ridge |          00) original OLD | Wed Sep 14 23:21:00 CEST 2022 | Fri Sep 16 12:06:00 CEST 2022 (+37hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Wed Sep 14 23:21:00 CEST 2022 | Fri Sep 16 12:06:00 CEST 2022 (+37hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Wed Sep 14 23:21:00 CEST 2022 | Fri Sep 16 12:06:00 CEST 2022 (+37hr) |
;;|           mt-government |          00) original NEW | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|           mt-government |          00) original OLD | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|           mt-government | 01) crowning disabled NEW | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
```

```
;;|           mt-government | 01) crowning disabled OLD | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|              id-ross-fk |          00) original NEW | Tue Sep 13 11:24:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;|              id-ross-fk |          00) original OLD | Tue Sep 13 11:24:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;|              id-ross-fk | 01) crowning disabled NEW | Tue Sep 13 11:24:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;|              id-ross-fk | 01) crowning disabled OLD | Tue Sep 13 11:24:00 CEST 2022 | Wed Sep 14 11:54:00 CEST 2022 (+25hr) |
;;|           wa-bolt-creek |          00) original NEW | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 11:54:00 CEST 2022 (+25hr) |
;;|           wa-bolt-creek |          00) original OLD | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 11:54:00 CEST 2022 (+25hr) |
;;|           wa-bolt-creek | 01) crowning disabled NEW | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 11:54:00 CEST 2022 (+25hr) |
;;|           wa-bolt-creek | 01) crowning disabled OLD | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 11:54:00 CEST 2022 (+25hr) |
;;|           wa-bolt-creek |          00) original NEW | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|           wa-bolt-creek |          00) original OLD | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|           wa-bolt-creek | 01) crowning disabled NEW | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|           wa-bolt-creek | 01) crowning disabled OLD | Tue Sep 20 22:17:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|  id-caledonia-blackburn |          00) original NEW | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+11hr) |
;;|  id-caledonia-blackburn |          00) original OLD | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+11hr) |
;;|  id-caledonia-blackburn | 01) crowning disabled NEW | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+11hr) |
;;|  id-caledonia-blackburn | 01) crowning disabled OLD | Sat Sep 17 11:47:00 CEST 2022 | Sat Sep 17 22:21:00 CEST 2022 (+11hr) |
;;|             wa-siouoxon |          00) original NEW | Sat Sep 24 22:42:00 CEST 2022 | Sun Sep 25 13:28:00 CEST 2022 (+15hr) |
;;|             wa-siouoxon |          00) original OLD | Sat Sep 24 22:42:00 CEST 2022 | Sun Sep 25 13:28:00 CEST 2022 (+15hr) |
;;|             wa-siouoxon | 01) crowning disabled NEW | Sat Sep 24 22:42:00 CEST 2022 | Sun Sep 25 13:28:00 CEST 2022 (+15hr) |
;;|             wa-siouoxon | 01) crowning disabled OLD | Sat Sep 24 22:42:00 CEST 2022 | Sun Sep 25 13:28:00 CEST 2022 (+15hr) |
;;|           id-deep-creek |          00) original NEW | Sun Sep 18 12:17:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+24hr) |
;;|           id-deep-creek |          00) original OLD | Sun Sep 18 12:17:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+24hr) |
;;|           id-deep-creek | 01) crowning disabled NEW | Sun Sep 18 12:17:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+24hr) |
;;|           id-deep-creek | 01) crowning disabled OLD | Sun Sep 18 12:17:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+24hr) |
;;|         or-double-creek |          00) original NEW | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 02:06:00 CEST 2022 (+15hr) |
;;|         or-double-creek |          00) original OLD | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 02:06:00 CEST 2022 (+15hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 02:06:00 CEST 2022 (+15hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 02:06:00 CEST 2022 (+15hr) |
;;|               ca-summit |          00) original NEW | Sun Sep 18 12:21:00 CEST 2022 | Sun Sep 18 18:30:00 CEST 2022 (+ 6hr) |
;;|               ca-summit |          00) original OLD | Sun Sep 18 12:21:00 CEST 2022 | Sun Sep 18 18:30:00 CEST 2022 (+ 6hr) |
;;|               ca-summit | 01) crowning disabled NEW | Sun Sep 18 12:21:00 CEST 2022 | Sun Sep 18 18:30:00 CEST 2022 (+ 6hr) |
;;|               ca-summit | 01) crowning disabled OLD | Sun Sep 18 12:21:00 CEST 2022 | Sun Sep 18 18:30:00 CEST 2022 (+ 6hr) |
;;|  id-columbus-bear-gulch |          00) original NEW | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+14hr) |
;;|  id-columbus-bear-gulch |          00) original OLD | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+14hr) |
;;|  id-columbus-bear-gulch | 01) crowning disabled NEW | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+14hr) |
;;|  id-columbus-bear-gulch | 01) crowning disabled OLD | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 12:17:00 CEST 2022 (+14hr) |
;;|               ca-summit |          00) original NEW | Sun Sep 18 18:30:00 CEST 2022 | Mon Sep 19 12:02:00 CEST 2022 (+18hr) |
;;|               ca-summit |          00) original OLD | Sun Sep 18 18:30:00 CEST 2022 | Mon Sep 19 12:02:00 CEST 2022 (+18hr) |
;;|               ca-summit | 01) crowning disabled NEW | Sun Sep 18 18:30:00 CEST 2022 | Mon Sep 19 12:02:00 CEST 2022 (+18hr) |
;;|               ca-summit | 01) crowning disabled OLD | Sun Sep 18 18:30:00 CEST 2022 | Mon Sep 19 12:02:00 CEST 2022 (+18hr) |
;;|         wa-minnow-ridge |          00) original NEW | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 12:43:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge |          00) original OLD | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 12:43:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 12:43:00 CEST 2022 (+25hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Wed Sep 21 11:22:00 CEST 2022 | Thu Sep 22 12:43:00 CEST 2022 (+25hr) |
;;|          or-cedar-creek |          00) original NEW | Fri Sep 16 11:15:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+24hr) |
;;|          or-cedar-creek |          00) original OLD | Fri Sep 16 11:15:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+24hr) |
;;|          or-cedar-creek | 01) crowning disabled NEW | Fri Sep 16 11:15:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+24hr) |
;;|          or-cedar-creek | 01) crowning disabled OLD | Fri Sep 16 11:15:00 CEST 2022 | Sat Sep 17 10:58:00 CEST 2022 (+24hr) |
;;|     wa-mcallister-creek |          00) original NEW | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|     wa-mcallister-creek |          00) original OLD | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|     wa-mcallister-creek | 01) crowning disabled NEW | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|     wa-mcallister-creek | 01) crowning disabled OLD | Tue Sep 27 12:49:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+22hr) |
;;|            wa-irving-pk |          00) original NEW | Tue Sep 13 11:22:00 CEST 2022 | Tue Sep 13 22:51:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk |          00) original OLD | Tue Sep 13 11:22:00 CEST 2022 | Tue Sep 13 22:51:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Tue Sep 13 11:22:00 CEST 2022 | Tue Sep 13 22:51:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Tue Sep 13 11:22:00 CEST 2022 | Tue Sep 13 22:51:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk |          00) original NEW | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+12hr) |
;;|            wa-irving-pk |          00) original OLD | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+12hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+12hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Sun Sep 25 23:14:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+12hr) |
;;|                id-lemhi |          00) original NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|                id-lemhi |          00) original OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|                id-lemhi | 01) crowning disabled NEW | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|                id-lemhi | 01) crowning disabled OLD | Wed Sep 28 10:50:00 CEST 2022 | Wed Sep 28 22:17:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge |          00) original NEW | Sun Sep 25 10:58:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+12hr) |
;;|         wa-minnow-ridge |          00) original OLD | Sun Sep 25 10:58:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+12hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Sun Sep 25 10:58:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+12hr) |
```

```
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Sun Sep 25 10:58:00 CEST 2022 | Sun Sep 25 23:14:00 CEST 2022 (+12hr) |
;;|           mt-government |          00) original NEW | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 11:28:00 CEST 2022 (+13hr) |
;;|           mt-government |          00) original OLD | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 11:28:00 CEST 2022 (+13hr) |
;;|           mt-government | 01) crowning disabled NEW | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 11:28:00 CEST 2022 (+13hr) |
;;|           mt-government | 01) crowning disabled OLD | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 11:28:00 CEST 2022 (+13hr) |
;;|                id-katka |          00) original NEW | Fri Sep 16 22:02:00 CEST 2022 | Sun Sep 18 14:12:00 CEST 2022 (+40hr) |
;;|                id-katka |          00) original OLD | Fri Sep 16 22:02:00 CEST 2022 | Sun Sep 18 14:12:00 CEST 2022 (+40hr) |
;;|                id-katka | 01) crowning disabled NEW | Fri Sep 16 22:02:00 CEST 2022 | Sun Sep 18 14:12:00 CEST 2022 (+40hr) |
;;|                id-katka | 01) crowning disabled OLD | Fri Sep 16 22:02:00 CEST 2022 | Sun Sep 18 14:12:00 CEST 2022 (+40hr) |
;;|             ca-mosquito |          00) original NEW | Sun Sep 18 00:26:00 CEST 2022 | Sun Sep 18 11:32:00 CEST 2022 (+11hr) |
;;|             ca-mosquito |          00) original OLD | Sun Sep 18 00:26:00 CEST 2022 | Sun Sep 18 11:32:00 CEST 2022 (+11hr) |
;;|             ca-mosquito | 01) crowning disabled NEW | Sun Sep 18 00:26:00 CEST 2022 | Sun Sep 18 11:32:00 CEST 2022 (+11hr) |
;;|             ca-mosquito | 01) crowning disabled OLD | Sun Sep 18 00:26:00 CEST 2022 | Sun Sep 18 11:32:00 CEST 2022 (+11hr) |
;;|               ca-summit |          00) original NEW | Tue Sep 20 10:54:00 CEST 2022 | Wed Sep 21 12:15:00 CEST 2022 (+25hr) |
;;|               ca-summit |          00) original OLD | Tue Sep 20 10:54:00 CEST 2022 | Wed Sep 21 12:15:00 CEST 2022 (+25hr) |
;;|               ca-summit | 01) crowning disabled NEW | Tue Sep 20 10:54:00 CEST 2022 | Wed Sep 21 12:15:00 CEST 2022 (+25hr) |
;;|               ca-summit | 01) crowning disabled OLD | Tue Sep 20 10:54:00 CEST 2022 | Wed Sep 21 12:15:00 CEST 2022 (+25hr) |
;;|  id-isabella-lower-twin |          00) original NEW | Thu Sep 15 11:34:00 CEST 2022 | Sat Sep 17 10:56:00 CEST 2022 (+47hr) |
;;|  id-isabella-lower-twin |          00) original OLD | Thu Sep 15 11:34:00 CEST 2022 | Sat Sep 17 10:56:00 CEST 2022 (+47hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled NEW | Thu Sep 15 11:34:00 CEST 2022 | Sat Sep 17 10:56:00 CEST 2022 (+47hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled OLD | Thu Sep 15 11:34:00 CEST 2022 | Sat Sep 17 10:56:00 CEST 2022 (+47hr) |
;;|         or-double-creek |          00) original NEW | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|         or-double-creek |          00) original OLD | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Wed Sep 14 11:05:00 CEST 2022 | Thu Sep 15 11:34:00 CEST 2022 (+24hr) |
;;|          mt-george-lake |          00) original NEW | Tue Sep 13 11:22:00 CEST 2022 | Thu Sep 15 13:09:00 CEST 2022 (+50hr) |
;;|          mt-george-lake |          00) original OLD | Tue Sep 13 11:22:00 CEST 2022 | Thu Sep 15 13:09:00 CEST 2022 (+50hr) |
;;|          mt-george-lake | 01) crowning disabled NEW | Tue Sep 13 11:22:00 CEST 2022 | Thu Sep 15 13:09:00 CEST 2022 (+50hr) |
;;|          mt-george-lake | 01) crowning disabled OLD | Tue Sep 13 11:22:00 CEST 2022 | Thu Sep 15 13:09:00 CEST 2022 (+50hr) |
;;|           mt-government |          00) original NEW | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 13:01:00 CEST 2022 (+25hr) |
;;|           mt-government |          00) original OLD | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 13:01:00 CEST 2022 (+25hr) |
;;|           mt-government | 01) crowning disabled NEW | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 13:01:00 CEST 2022 (+25hr) |
;;|           mt-government | 01) crowning disabled OLD | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 13:01:00 CEST 2022 (+25hr) |
;;|            wa-irving-pk |          00) original NEW | Sat Sep 17 11:47:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+48hr) |
;;|            wa-irving-pk |          00) original OLD | Sat Sep 17 11:47:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+48hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Sat Sep 17 11:47:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+48hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Sat Sep 17 11:47:00 CEST 2022 | Mon Sep 19 12:00:00 CEST 2022 (+48hr) |
;;|             or-sturgill |          00) original NEW | Tue Sep 20 18:04:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+ 4hr) |
;;|             or-sturgill |          00) original OLD | Tue Sep 20 18:04:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+ 4hr) |
;;|             or-sturgill | 01) crowning disabled NEW | Tue Sep 20 18:04:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+ 4hr) |
;;|             or-sturgill | 01) crowning disabled OLD | Tue Sep 20 18:04:00 CEST 2022 | Tue Sep 20 22:19:00 CEST 2022 (+ 4hr) |
;;|  id-isabella-lower-twin |          00) original NEW | Mon Sep 19 12:49:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin |          00) original OLD | Mon Sep 19 12:49:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled NEW | Mon Sep 19 12:49:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|  id-isabella-lower-twin | 01) crowning disabled OLD | Mon Sep 19 12:49:00 CEST 2022 | Mon Sep 19 23:25:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk |          00) original NEW | Tue Sep 20 10:50:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk |          00) original OLD | Tue Sep 20 10:50:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Tue Sep 20 10:50:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Tue Sep 20 10:50:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge |          00) original NEW | Thu Sep 22 12:43:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+10hr) |
;;|         wa-minnow-ridge |          00) original OLD | Thu Sep 22 12:43:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+10hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Thu Sep 22 12:43:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+10hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Thu Sep 22 12:43:00 CEST 2022 | Thu Sep 22 22:29:00 CEST 2022 (+10hr) |
;;|  id-caledonia-blackburn |          00) original NEW | Wed Sep 14 11:52:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+72hr) |
;;|  id-caledonia-blackburn |          00) original OLD | Wed Sep 14 11:52:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+72hr) |
;;|  id-caledonia-blackburn | 01) crowning disabled NEW | Wed Sep 14 11:52:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+72hr) |
;;|  id-caledonia-blackburn | 01) crowning disabled OLD | Wed Sep 14 11:52:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+72hr) |
;;|  id-kootenai-rv-complex |          00) original NEW | Wed Sep 28 05:16:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+ 6hr) |
;;|  id-kootenai-rv-complex |          00) original OLD | Wed Sep 28 05:16:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+ 6hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled NEW | Wed Sep 28 05:16:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+ 6hr) |
;;|  id-kootenai-rv-complex | 01) crowning disabled OLD | Wed Sep 28 05:16:00 CEST 2022 | Wed Sep 28 10:50:00 CEST 2022 (+ 6hr) |
;;|         or-double-creek |          00) original NEW | Tue Sep 20 22:19:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|         or-double-creek |          00) original OLD | Tue Sep 20 22:19:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|         or-double-creek | 01) crowning disabled NEW | Tue Sep 20 22:19:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|         or-double-creek | 01) crowning disabled OLD | Tue Sep 20 22:19:00 CEST 2022 | Wed Sep 21 11:22:00 CEST 2022 (+13hr) |
;;|          id-trail-ridge |          00) original NEW | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+12hr) |
;;|          id-trail-ridge |          00) original OLD | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+12hr) |
;;|          id-trail-ridge | 01) crowning disabled NEW | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+12hr) |
```

```
;;|          id-trail-ridge | 01) crowning disabled OLD | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+12hr) |
;;|         wa-minnow-ridge |          00) original NEW | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge |          00) original OLD | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge | 01) crowning disabled NEW | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|         wa-minnow-ridge | 01) crowning disabled OLD | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|                id-lemhi |          00) original NEW | Sun Sep 25 23:12:00 CEST 2022 | Mon Sep 26 12:19:00 CEST 2022 (+13hr) |
;;|                id-lemhi |          00) original OLD | Sun Sep 25 23:12:00 CEST 2022 | Mon Sep 26 12:19:00 CEST 2022 (+13hr) |
;;|                id-lemhi | 01) crowning disabled NEW | Sun Sep 25 23:12:00 CEST 2022 | Mon Sep 26 12:19:00 CEST 2022 (+13hr) |
;;|                id-lemhi | 01) crowning disabled OLD | Sun Sep 25 23:12:00 CEST 2022 | Mon Sep 26 12:19:00 CEST 2022 (+13hr) |
;;|            wa-irving-pk |          00) original NEW | Tue Sep 13 22:51:00 CEST 2022 | Thu Sep 15 12:26:00 CEST 2022 (+38hr) |
;;|            wa-irving-pk |          00) original OLD | Tue Sep 13 22:51:00 CEST 2022 | Thu Sep 15 12:26:00 CEST 2022 (+38hr) |
;;|            wa-irving-pk | 01) crowning disabled NEW | Tue Sep 13 22:51:00 CEST 2022 | Thu Sep 15 12:26:00 CEST 2022 (+38hr) |
;;|            wa-irving-pk | 01) crowning disabled OLD | Tue Sep 13 22:51:00 CEST 2022 | Thu Sep 15 12:26:00 CEST 2022 (+38hr) |
;;|               ca-summit |          00) original NEW | Mon Sep 19 12:02:00 CEST 2022 | Mon Sep 19 23:23:00 CEST 2022 (+11hr) |
;;|               ca-summit |          00) original OLD | Mon Sep 19 12:02:00 CEST 2022 | Mon Sep 19 23:23:00 CEST 2022 (+11hr) |
;;|               ca-summit | 01) crowning disabled NEW | Mon Sep 19 12:02:00 CEST 2022 | Mon Sep 19 23:23:00 CEST 2022 (+11hr) |
;;|               ca-summit | 01) crowning disabled OLD | Mon Sep 19 12:02:00 CEST 2022 | Mon Sep 19 23:23:00 CEST 2022 (+11hr) |
;;|          id-deep-creek |          00) original NEW | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+12hr) |
;;|          id-deep-creek |          00) original OLD | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+12hr) |
;;|          id-deep-creek | 01) crowning disabled NEW | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+12hr) |
;;|          id-deep-creek | 01) crowning disabled OLD | Mon Sep 19 23:25:00 CEST 2022 | Tue Sep 20 11:41:00 CEST 2022 (+12hr) |
;;|       mt-isabella-lake |          00) original NEW | Wed Sep 14 05:40:00 CEST 2022 | Fri Sep 16 13:26:00 CEST 2022 (+56hr) |
;;|       mt-isabella-lake |          00) original OLD | Wed Sep 14 05:40:00 CEST 2022 | Fri Sep 16 13:26:00 CEST 2022 (+56hr) |
;;|       mt-isabella-lake | 01) crowning disabled NEW | Wed Sep 14 05:40:00 CEST 2022 | Fri Sep 16 13:26:00 CEST 2022 (+56hr) |
;;|       mt-isabella-lake | 01) crowning disabled OLD | Wed Sep 14 05:40:00 CEST 2022 | Fri Sep 16 13:26:00 CEST 2022 (+56hr) |
;;|                ca-red |          00) original NEW | Sat Sep 17 10:58:00 CEST 2022 | Sat Sep 17 23:12:00 CEST 2022 (+12hr) |
;;|                ca-red |          00) original OLD | Sat Sep 17 10:58:00 CEST 2022 | Sat Sep 17 23:12:00 CEST 2022 (+12hr) |
;;|                ca-red | 01) crowning disabled NEW | Sat Sep 17 10:58:00 CEST 2022 | Sat Sep 17 23:12:00 CEST 2022 (+12hr) |
;;|                ca-red | 01) crowning disabled OLD | Sat Sep 17 10:58:00 CEST 2022 | Sat Sep 17 23:12:00 CEST 2022 (+12hr) |
;;|             ca-summit |          00) original NEW | Mon Sep 26 11:32:00 CEST 2022 | Tue Sep 27 11:11:00 CEST 2022 (+24hr) |
;;|             ca-summit |          00) original OLD | Mon Sep 26 11:32:00 CEST 2022 | Tue Sep 27 11:11:00 CEST 2022 (+24hr) |
;;|             ca-summit | 01) crowning disabled NEW | Mon Sep 26 11:32:00 CEST 2022 | Tue Sep 27 11:11:00 CEST 2022 (+24hr) |
;;|             ca-summit | 01) crowning disabled OLD | Mon Sep 26 11:32:00 CEST 2022 | Tue Sep 27 11:11:00 CEST 2022 (+24hr) |
;;|                ca-red |          00) original NEW | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 18:50:00 CEST 2022 (+ 8hr) |
;;|                ca-red |          00) original OLD | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 18:50:00 CEST 2022 (+ 8hr) |
;;|                ca-red | 01) crowning disabled NEW | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 18:50:00 CEST 2022 (+ 8hr) |
;;|                ca-red | 01) crowning disabled OLD | Wed Sep 14 11:05:00 CEST 2022 | Wed Sep 14 18:50:00 CEST 2022 (+ 8hr) |
;;| id-isabella-lower-twin |          00) original NEW | Tue Sep 13 12:11:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+23hr) |
;;| id-isabella-lower-twin |          00) original OLD | Tue Sep 13 12:11:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+23hr) |
;;| id-isabella-lower-twin | 01) crowning disabled NEW | Tue Sep 13 12:11:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+23hr) |
;;| id-isabella-lower-twin | 01) crowning disabled OLD | Tue Sep 13 12:11:00 CEST 2022 | Wed Sep 14 11:05:00 CEST 2022 (+23hr) |
;;|              mt-cannon |          00) original NEW | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+12hr) |
;;|              mt-cannon |          00) original OLD | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+12hr) |
;;|              mt-cannon | 01) crowning disabled NEW | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+12hr) |
;;|              mt-cannon | 01) crowning disabled OLD | Sat Sep 17 22:21:00 CEST 2022 | Sun Sep 18 10:37:00 CEST 2022 (+12hr) |
;;|           ca-mosquito |          00) original NEW | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 11:17:00 CEST 2022 (+24hr) |
;;|           ca-mosquito |          00) original OLD | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 11:17:00 CEST 2022 (+24hr) |
;;|           ca-mosquito | 01) crowning disabled NEW | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 11:17:00 CEST 2022 (+24hr) |
;;|           ca-mosquito | 01) crowning disabled OLD | Thu Sep 15 11:34:00 CEST 2022 | Fri Sep 16 11:17:00 CEST 2022 (+24hr) |
;;| id-kootenai-rv-complex |          00) original NEW | Tue Sep 27 12:47:00 CEST 2022 | Wed Sep 28 05:16:00 CEST 2022 (+16hr) |
;;| id-kootenai-rv-complex |          00) original OLD | Tue Sep 27 12:47:00 CEST 2022 | Wed Sep 28 05:16:00 CEST 2022 (+16hr) |
;;| id-kootenai-rv-complex | 01) crowning disabled NEW | Tue Sep 27 12:47:00 CEST 2022 | Wed Sep 28 05:16:00 CEST 2022 (+16hr) |
;;| id-kootenai-rv-complex | 01) crowning disabled OLD | Tue Sep 27 12:47:00 CEST 2022 | Wed Sep 28 05:16:00 CEST 2022 (+16hr) |
;;|            or-sturgill |          00) original NEW | Sat Sep 17 18:47:00 CEST 2022 | Mon Sep 19 19:47:00 CEST 2022 (+49hr) |
;;|            or-sturgill |          00) original OLD | Sat Sep 17 18:47:00 CEST 2022 | Mon Sep 19 19:47:00 CEST 2022 (+49hr) |
;;|            or-sturgill | 01) crowning disabled NEW | Sat Sep 17 18:47:00 CEST 2022 | Mon Sep 19 19:47:00 CEST 2022 (+49hr) |
;;|            or-sturgill | 01) crowning disabled OLD | Sat Sep 17 18:47:00 CEST 2022 | Mon Sep 19 19:47:00 CEST 2022 (+49hr) |
;;|          wa-bolt-creek |          00) original NEW | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|          wa-bolt-creek |          00) original OLD | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|          wa-bolt-creek | 01) crowning disabled NEW | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|          wa-bolt-creek | 01) crowning disabled OLD | Tue Sep 20 11:41:00 CEST 2022 | Tue Sep 20 22:17:00 CEST 2022 (+11hr) |
;;|              wa-kalama |          00) original NEW | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:02:00 CEST 2022 (+23hr) |
;;|              wa-kalama |          00) original OLD | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:02:00 CEST 2022 (+23hr) |
;;|              wa-kalama | 01) crowning disabled NEW | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:02:00 CEST 2022 (+23hr) |
;;|              wa-kalama | 01) crowning disabled OLD | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:02:00 CEST 2022 (+23hr) |
;;|            mt-margaret |          00) original NEW | Thu Sep 15 10:43:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+49hr) |
;;|            mt-margaret |          00) original OLD | Thu Sep 15 10:43:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+49hr) |
;;|            mt-margaret | 01) crowning disabled NEW | Thu Sep 15 10:43:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+49hr) |
```

```
;;|            mt-margaret | 01) crowning disabled OLD | Thu Sep 15 10:43:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+49hr) |
;;|          wa-goat-rocks |          00) original NEW | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:00:00 CEST 2022 (+23hr) |
;;|          wa-goat-rocks |          00) original OLD | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:00:00 CEST 2022 (+23hr) |
;;|          wa-goat-rocks | 01) crowning disabled NEW | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:00:00 CEST 2022 (+23hr) |
;;|          wa-goat-rocks | 01) crowning disabled OLD | Mon Sep 26 13:08:00 CEST 2022 | Tue Sep 27 12:00:00 CEST 2022 (+23hr) |
;;|          wa-bolt-creek |          00) original NEW | Sun Sep 25 11:45:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+24hr) |
;;|          wa-bolt-creek |          00) original OLD | Sun Sep 25 11:45:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+24hr) |
;;|          wa-bolt-creek | 01) crowning disabled NEW | Sun Sep 25 11:45:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+24hr) |
;;|          wa-bolt-creek | 01) crowning disabled OLD | Sun Sep 25 11:45:00 CEST 2022 | Mon Sep 26 11:28:00 CEST 2022 (+24hr) |
;;|            or-sturgill |          00) original NEW | Fri Sep 16 22:42:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+13hr) |
;;|            or-sturgill |          00) original OLD | Fri Sep 16 22:42:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+13hr) |
;;|            or-sturgill | 01) crowning disabled NEW | Fri Sep 16 22:42:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+13hr) |
;;|            or-sturgill | 01) crowning disabled OLD | Fri Sep 16 22:42:00 CEST 2022 | Sat Sep 17 11:47:00 CEST 2022 (+13hr) |
;;|            mt-billiard |          00) original NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:52:00 CEST 2022 (+25hr) |
;;|            mt-billiard |          00) original OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:52:00 CEST 2022 (+25hr) |
;;|            mt-billiard | 01) crowning disabled NEW | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:52:00 CEST 2022 (+25hr) |
;;|            mt-billiard | 01) crowning disabled OLD | Tue Sep 13 11:22:00 CEST 2022 | Wed Sep 14 11:52:00 CEST 2022 (+25hr) |

;; Main observations:
;; (1) Under "00) original", the new version causes much less overprediction than the old version.
;; (2) Under "01) crowning disabled", the new version causes much MORE overprediction than the old version.
;; So in all likelihood, something has changed to make the surface fire spread more aggressive, at least in these simulations.

;; Here is a curated list of the new branches which might have had an impact:

;; £ git br --merged HEAD --no-merge 85f1467
;  conflicts-96
;  draft-vwaeselynck-smoothed-supergrid-perturbation
;  fix-suppression-fc-max0
;  hotfix-fuel-moisture-get-band
;  kcheung-fix-suppression-override-config
;  kcheung-sdi
;  kcheung-sdi-spec-docs
;  kcheung-use-fuel-varying-multipliers
;  vwaeselynck-29-no-boxing-in-get-value-fn
;* vwaeselynck-330-gridfire-historical-fires
;  vwaeselynck-352-disable-crowning
;  vwaeselynck-353-directional-crowning-initiation
;  vwaeselynck-395-new-fuel-models
;  vwaeselynck-398-burn-period-from-sun-per-ignition
;  vwaeselynck-405-e2g-ignition-csv-override-layers
;  vwaeselynck-410--simplify-spotting-sampling
;  vwaeselynck-434-compiler-options-for-production-jar
;  vwaeselynck-436-gridfire-optimizations
;  vwaeselynck-fix-NaN-clock
;  vwaeselynck-fix-create-new-burn-vectors_test
;  vwaeselynck-fix-elmfire-merge-override-config
;  vwaeselynck-fix-missing-test-suites-metadata
;  vwaeselynck-fix-process-output-layers
;  vwaeselynck-fix-test-suites
;  vwaeselynck-fix-valid-fuel-range-spec-WUI-max=303
;  vwaeselynck-hash-determined-pixel-perturbation
;  vwaeselynck-refactor-ignition-sites-sampling
;  vwaeselynck-update-magellan-deps-2022-12
;  vwaeselynck-upgrade-magellan-2022.10.21
;  vwaeselynk-336-burn-period-from-sunrise-sunset

*e)
```

9.8 Data Structures and Algorithmic Helpers

9.8.1 gridfire.grid-lookup

```
(ns gridfire.grid-lookup
  "Utils for efficiently resolving values at [b i j] coordinates in a grid.
```

```
  Coordinates [b i j] correspond to:
  - b: 'hourly Band', a 1h-resolution temporal-grid coordinate;
  - i: matrix row, a spatial-grid coordinate, discretizing the y axis (yes, you read that right);
  - j: matrix column, a spatial-grid coordinate, discretizing the x axis."
  (:require [gridfire.structs.tensor-meta :as tensor-meta]
            [tech.v3.datatype             :as d]
            [tech.v3.tensor               :as t])
  (:import (clojure.lang IFn$LD IFn$LLD IFn$LLLD)))
;; NOTE clj-kondo (2022.11.02) currently reports false-positive linting issues, not sure what to do about it: (Val, 28 Nov 2022)
;; £ clj-kondo --lint src/gridfire/grid_lookup.clj
;; src/gridfire/grid_lookup.clj:8:14: warning: namespace gridfire.structs.tensor-meta is required but never used
;; src/gridfire/grid_lookup.clj:9:14: warning: namespace tech.v3.datatype is required but never used
;; src/gridfire/grid_lookup.clj:10:14: warning: namespace tech.v3.tensor is required but never used
;; src/gridfire/grid_lookup.clj:175:12: warning: Unresolved namespace d. Are you missing a require?
;; src/gridfire/grid_lookup.clj:176:6: warning: Unresolved namespace t. Are you missing a require?
;; src/gridfire/grid_lookup.clj:202:12: error: Unresolved symbol: suitable-for-primitive-lookup?
;; src/gridfire/grid_lookup.clj:254:27: warning: Unresolved namespace tensor-meta. Are you missing a require?
;; src/gridfire/grid_lookup.clj:272:5: error: Unresolved symbol: double-at
;; linting took 51ms, errors: 2, warnings: 6

(defn suitable-for-primitive-lookup?
  "Whether the given value can be used as the 1st argument to (double-at v & coords)"
  [v]
  ;; HACK relying on clojure.lang primitive-signature interfaces;
  ;; that's convenient, but strictly speaking these are implementation details
  ;; of Clojure execution.
  (or
   (instance? IFn$LD v)
   (instance? IFn$LLD v)
   (instance? IFn$LLLD v)))

(defn double-at
  ;; NOTE turning this function into a macro only made simulations barely faster (455ms instead of 465ms) - not worth it IMHO. (Val,
  "Looks up a double-typed value at the given grid coordinates,
  avoiding boxing of the returned value."
  (^double [getter ^long b]
   (.invokePrim ^IFn$LD getter b))
  (^double [getter ^long i ^long j]
   (.invokePrim ^IFn$LLD getter i j))
  (^double [getter ^long b ^long i ^long j]
   (.invokePrim ^IFn$LLLD getter b i j)))

(comment
 ;; (double-at ...) does allow lookup without primitive boxing:
 (clj-java-decompiler.core/decompile
  (let [f
        ;; Let's make sure the Clojure compiler won't guess the signature of f:
        (get {2
              (fn my-getter ^double [^long i ^long j]
                (* 2. (+ i j)))}
             (+ 1 1))]
    (fn my-computation ^double []
      (+ 3.
         (double-at f 2 3)))))
 ;;public final class grid_lookup$fn__44754$my_computation__44757 extends AFunction implements D
 ;;{
 ;;    // ...
 ;;
 ;;    @Override
 ;;    public final double invokePrim() {
 ;;        return 3.0 + ((OLLD)grid_lookup$fn__44754$my_computation__44757.const__2.getRawRoot()).invokePrim(this.f, 2L, 3L);
 ;;    }
 ;;
 ;;    // ...
 ;;}

 ;; In the above Java code, we do see that double-at is called without boxing.
 ;; We still see the overhead of Var-lookup (.getRawRoot()),
```

```
 ;; but that's not necessarily very significant (thanks to CPU caching),
 ;; and could get eliminated by Clojure direct linking (https://clojure.org/reference/compilation#directlinking).

 *e)

(comment
 ;; NOTE Why not do this more cleanly with a Clojure protocol? As of Clojure 1.10,
 ;; (defprotocol ...) does not compile to a Java interface
 ;; with primitive signatures, and therefore cannot be used to avoid boxing:
 (in-ns 'user)
 (defprotocol IMyProtocol
   (get-me-a-double ^double [this ^long i ^long j]))
 (require 'clojure.reflect)
 (clojure.reflect/reflect user.IMyProtocol)
 ;=>
 {:bases   nil,
  :flags   #{:interface :public :abstract},
  :members #{#clojure.reflect.Method{:name            get_me_a_double,
                                     :return-type     java.lang.Object,
                                     :declaring-class user.IMyProtocol,
                                     :parameter-types [java.lang.Object java.lang.Object],
                                     :exception-types [],
                                     :flags           #{:public :abstract}}}}
 ;; Therefore, unfortunately, we have to use some other mechanism.
 *e)

(defn ensure-flat-jvm-tensor
  "Given a tensor, returns a tensor which is backed by a flat JVM array, copying if necessary."
  [m]
  (cond-> m
    (nil? (d/as-array-buffer-data m))
    (t/clone :container-type :jvm-heap)))

(defn- tensor-wrapped-array
  [unwrap-fn m]
  (let [arr (unwrap-fn m)]
    (assert (identical? arr (unwrap-fn m)))
    arr))

(defn indexes-out-of-bound-ex
  [max-b max-i max-j b i j]
  (ex-info (format "ND-array indexes [b i j] out of bounds (must be <= %s and > [0 0 0]): %s"
                   (pr-str [max-b max-i max-j])
                   (pr-str [b i j]))
           {::b+i+j [b i j]
            ::max-b+i+j [max-b max-i max-j]}))

(defmacro aget-3dim
  [arr n-per-row n-per-band default-v? default-v max-b max-i max-j b i j]
  `(let [out-of-bounds?# (or (< ~b 0) (> ~b ~max-b)
                             (< ~i 0) (> ~i ~max-i)
                             (< ~j 0) (> ~j ~max-j))]
     (if out-of-bounds?#
       (if ~default-v?
         ~default-v
         (throw (indexes-out-of-bound-ex ~max-b ~max-i ~max-j ~b ~i ~j)))
       (aget ~arr (-> ~j
                      (unchecked-add (unchecked-multiply ~n-per-row ~i))
                      ;; NOTE simplifying when b is known to be zero at compile-time. (Val, 24 Nov 2022)
                      ~@(when-not (= 0 b) [`(unchecked-add (unchecked-multiply ~n-per-band ~b))])
                      (int))))))

(defn- unwrap-short-tensor
  ^shorts [m]
  ;; HACK relying on an implementation detail of d/. Because d/->short-array will tend to cause copying for some reason.
  (.ary_data (d/as-array-buffer m)))

(defn tensor-cell-getter
```

```
  "Given a tensor `m`, returns a value suitable to be called by `double-at`.

  `m` must have gone through `ensure-flat-jvm-tensor`.
  Optionally, a non-nil `default-value` number may be supplied,
  which will be returned when indexes fall out of bounds."
  [m default-value]
  {:post [(suitable-for-primitive-lookup? %)]}
  (if (number? m)
    (let [v (double m)]
      (fn get0d
        (^double [^long _b] v)
        (^double [^long _i ^long _j] v)
        (^double [^long _b ^long _i ^long _j] v)))
    (let [shape       (d/shape m)
          n-dims      (count shape)
          [n-per-band
           n-per-row
           ubounds]   (case n-dims
                        2 (let [[n-rows n-cols] shape]
                            ;; NOTE we are treating the 2-dims tensors as a special case of the 3-dims tensors.
                            ;; This is because we want to return always the same type (JVM class) of function,
                            ;; to limit the creation of polymorphic call sites.
                            [0
                             n-cols
                             [1 n-rows n-cols]])
                        3 (let [[n-bands n-rows n-cols] shape]
                            [(* (long n-rows) (long n-cols))
                             n-cols
                             [n-bands n-rows n-cols]]))
          n-per-band  (long n-per-band)
          n-per-row   (long n-per-row)
          default-v?  (some? default-value)
          default-v   (if default-v? (double default-value) Double/NaN)
          [mb mi mj]  ubounds
          max-b       (dec (long mb))
          max-i       (dec (long mi))
          max-j       (dec (long mj))]
      (case (d/elemwise-datatype m)
        (:float64 :double)
        (let [arr (doubles (tensor-wrapped-array d/->double-array m))]
          (fn mget-doubles
            (^double [^long i ^long j] (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j 0 i j))
            (^double [^long b ^long i ^long j] (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j b i j))))
        :float32
        (let [arr (floats (tensor-wrapped-array d/->float-array m))]
          (fn mget-floats
            (^double [^long i ^long j] (double (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j 0 i j)))
            (^double [^long b ^long i ^long j] (double (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j b
        :int32
        (let [arr (ints (tensor-wrapped-array d/->int-array m))]
          (fn mget-ints
            (^double [^long i ^long j] (double (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j 0 i j)))
            (^double [^long b ^long i ^long j] (double (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j b
        :int16
        (let [arr (shorts (tensor-wrapped-array d/->short-array m))]
          (fn mget-shorts-int16
            (^double [^long i ^long j] (double (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j 0 i j)))
            (^double [^long b ^long i ^long j] (double (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j b

        :short
        (let [arr (shorts (tensor-wrapped-array unwrap-short-tensor m))]
          (fn mget-shorts-short
            (^double [^long i ^long j] (double (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j 0 i j)))
            (^double [^long b ^long i ^long j] (double (aget-3dim arr n-per-row n-per-band default-v? default-v max-b max-i max-j b

(defn add-double-getter
  "Attaches to tensor m (in its metadata) a cached return value of (tensor-cell-getter),
  so that we don't need to recompute-it each time."
```

```
  ([m] (add-double-getter m nil))
  ([m default-value]
   (vary-meta m (fn [met] (tensor-meta/map->GettersMeta (assoc met :grid-lookup-double-getter (tensor-cell-getter m default-value))

(defn mgetter-double
  "Given a tensor which has been accelerated through (add-double-getter),
  return a getter object g which can be looked up via (double-at g ...)."
  [m]
  (-> m
      (meta)
      (tensor-meta/double-getter)
      (or (throw (ex-info (format "This tensor has not been accelerated by going through %s." (pr-str `add-double-getter))
                          {::tensor m})))))

(defn mget-double-at
  "Like tech.v3.tensor/mget, but faster.
  The tensor m must have been accelerated by going through #'add-double-getter.
  (mget-double-at m i j) is equivalent to (let [my-getter (mgetter-double m)] (double-at my-getter i j));
  the latter form improve the performance of repeated invocations."
  (^double [m ^long b]
   (double-at (mgetter-double m) b))
  (^double [m ^long i ^long j]
   (double-at (mgetter-double m) i j))
  (^double [m ^long b ^long i ^long j]
   (double-at (mgetter-double m) b i j)))
```

9.8.2 gridfire.grid-lookup-test

```
(ns gridfire.grid-lookup-test
  (:require [gridfire.grid-lookup :as grid-lookup]
            [clojure.test         :refer [deftest is testing]])
  (:import (clojure.lang IFn$OLD IFn$OLLD IFn$OLLLD IFn$LD IFn$LLD IFn$LLLD)))

(deftest ^:unit double-at-primitiveness-test
  (testing "(double-at getter & coords)"
    (testing "accepts primitive long-typed coordinates, and returns a primitive double."
      (is (instance? IFn$OLD grid-lookup/double-at))
      (is (instance? IFn$OLLD grid-lookup/double-at))
      (is (instance? IFn$OLLLD grid-lookup/double-at)))))

(deftest ^:unit clojure-primitive-functions-test
  (testing "this example primitive-signature function"
    (let [my-getter (fn
                      (^double [^long b]
                       (* 2. b))
                      (^double [^long i ^long j]
                       (* 2. i j))
                      (^double [^long b ^long i ^long j]
                       (* 2. b i j)))]
      (testing "does indeed accept and return primitive values:"
        (let [implemented-types (-> my-getter (type) (ancestors))]
          (is (contains? implemented-types IFn$LD))
          (is (contains? implemented-types IFn$LLD))
          (is (contains? implemented-types IFn$LLLD))))
      (testing "may be called efficiently with (double-at ...)"
        (is (grid-lookup/suitable-for-primitive-lookup? my-getter))
        (testing "which returns correct results:"
          (is (=
               6.
               (grid-lookup/double-at my-getter 3)))
          (is (=
               42.
               (grid-lookup/double-at my-getter 3 7)))
          (is (=
               420.
               (grid-lookup/double-at my-getter 10 3 7)))))))))
```

9.8.3 gridfire.utils.gradient

```
(ns gridfire.utils.gradient)

(defn estimate-dF
  "Utility for gradient estimation: estimates the derivative dF := (F/u)*du  (F(u + du) - F(u))  (F(u + du) - F(u - du))/2,
  based on (potentially missing) values of F(u - du), F(u) and F(u + du).
  Returns 0.0 when too few of the 3 values of F are available,
  (which is signaled by their being equal to NA-Fvalue, typically Double/NaN or -1.0)."
  ^double [^double NA-Fvalue ^double Fu-du ^double Fu ^double Fu+du]
  (if (= NA-Fvalue Fu+du)
    (if (= NA-Fvalue Fu-du)
      0.0                       ; "spear head"
      (if (= NA-Fvalue Fu)
        0.0
        (- Fu Fu-du)))
    (if (= NA-Fvalue Fu-du)
      (if (= NA-Fvalue Fu)
        0.0
        (- Fu+du Fu))
      ;; This is the favored way of estimating the gradient: https://numpy.org/doc/stable/reference/generated/numpy.gradient.html
      (* 0.5 (- Fu+du Fu-du)))))
```

9.8.4 gridfire.utils.flow

```
(ns gridfire.utils.flow)

(defn- log2
  ^double [^double x]
  ;; TODO bump Clojure version then use clojure.math/log. (Val, 22 Nov 2022)
  (/ (Math/log x)
     (Math/log 2.0)))

(defn- h2
  "Binary Entropy function - the entropy of a Bernoulli random variable with parameter p, in bits."
  ^double [^double p]
  (cond
    (= 0.0 p) 0.0
    (= 1.0 p) 0.0
    :else     (let [q (- 1.0 p)]
                ;; NOTE we could instead use the natural logarithm,
                ;; which would return the entropy in nats instead of bits;
                ;; however, bits seem more natural in this case,
                ;; since bits can be viewed as an "expected number of binary tests required".
                (+ (* p (log2 (/ 1.0 p)))
                   (* q (log2 (/ 1.0 q)))))))

(defn- n*entropy
  "Computes the N * H[B], where N is (count ds),
  and B is the random variable (pred D), for D uniformly chosen among ds."
  ^double [ds pred]
  (let [n  (double (count ds))
        n+ (->> ds (filter pred) (count) (double))]
    (* n (h2 (/ n+ n)))))

(defn- total-information-gain
  "Computes N * I[X,Y], in which:
  - I[X,Y] is the Mutual Information: I[X,Y] = H[Y] - H[Y|X],
  - H[Z] is the Entropy of random variable Z,
  - N is the number of outcomes (count ds),
  - Y is the binary random variable (pred D),
  - X is the random variable (get d->v D),
  under the probability model that
  D is a random variable uniformly distributed among (keys d->v).

  I[X,Y] is a measure of how much uncertainty about X is eliminated by knowing Y (on average, and vice-versa)."
```

```
  ^double [pred d->v]
  ;; Background: https://vvvvalvalval.github.io/posts/2020-05-15_Using-Decision-Trees-for-charting-ill-behaved-datasets.html#Appendix
  (- (n*entropy (keys d->v) pred)
     (->> (keys d->v)
          (group-by d->v)
          (vals)
          (map (fn [ds] (n*entropy ds pred)))
          (reduce + 0.0)
          (double))))

(defn- double-bit-tester
 [^long bit-idx]
 (fn test-bit-of-double [^double v]
   (bit-test (Double/doubleToRawLongBits v) bit-idx)))

(defn- bit-information-gain
  ^double [d->ret ^long bit-idx]
  (/ (total-information-gain (double-bit-tester bit-idx) d->ret)
     (double (count d->ret))))

(defmacro case-double
  "Similar to (case ...) but optimized for a finite input set of doubles.
  match+rets must be a sequence of match clauses followed by return expressions,
  like in case, except that:
  1. a match clause can only be a number or list of numbers,
  2. there cannot be a fallback last expression,
  3. the caller MUST ensure that x will evaluate to be among the doubles enumerated in the match clauses;
  the behavior is unspecified if that assumption is not met."
  [x & match+rets]
  ;; Implementation: expands to a tree of bit tests on the raw bits of x;
  ;; the bits are chosen by eager entropy reduction (as in a Decision Tree),
  ;; assuming that all values in (keys double->ret) are equally probable.
  (let [double->ret  (->> match+rets
                          (partition 2)
                          (into {} (mapcat (fn match+ret->kvs [[match ret]]
                                             (cond
                                               (number? match) [[(double match)
                                                                 ret]]
                                               (list? match) (->> match
                                                                  (mapv (fn [d]
                                                                          [(double d)
                                                                           ret])))))))))
        raw-bits-sym (gensym 'raw-bits)]
    (letfn [(code-for [d->ret]
              (if (apply = (vals d->ret))
                ;; Leaf: all considered doubles have the same return expr, and so we emit that expr.
                (-> d->ret (first) (val))
                ;; We choose a most informative bit to test against, in the sense of Information Theory,
                ;; i.e. a bit that achieves the maximal entropy reduction.
                (let [most-informative-bit (->> (range 64)
                                                (apply max-key (fn [bit-idx] (bit-information-gain d->ret bit-idx))))]
                  `(if (bit-test ~raw-bits-sym ~most-informative-bit)
                     ~(code-for (select-keys d->ret (->> (keys d->ret)
                                                         (filter (double-bit-tester most-informative-bit)))))
                     ~(code-for (select-keys d->ret (->> (keys d->ret)
                                                         (remove (double-bit-tester most-informative-bit))))))))))]
      `(let [~raw-bits-sym (Double/doubleToRawLongBits ~x)]
         ~(code-for double->ret)))))

(comment

  (def x 45.0)
  (macroexpand '(case-double x
                  0.0   false
                  45.0  true
                  90.0  false
                  135.0 true
                  180.0 false
```

```
                225.0 true
                270.0 false
                315.0 true))
;;=> expands to something like:
(let [raw-bits15403 (Double/doubleToRawLongBits x)]
  (if (bit-test raw-bits15403 45)
    (if (bit-test raw-bits15403 52)
      (if (bit-test raw-bits15403 49)
        true
        false)
      true)
    (if (bit-test raw-bits15403 62)
      (if (bit-test raw-bits15403 53)
        false
        (if (bit-test raw-bits15403 52)
          false
          true))
      false)))
;; NOTE: it's not guaranteed that eager entropy reduction
;; achieves an optimal "depth" of decision tree;
;; even so, it's probably close to optimal efficiency,
;; as it will tend to be rewarded by CPU branch prediction.

;; Here's an example where the sign bit (64) is clearly the most informative,
;; and gets indeed tested:
(macroexpand '(case-double x
                (-1.0 -2.0 -3.0 -4.0) :negative
                ( 1.0  2.0  3.0  4.0) :positive))
;;=> expands to something like:
(let* [raw-bits2011 (Double/doubleToRawLongBits x)]
  (if (bit-test raw-bits2011 63)
    :negative
    :positive))

;; Here's a slightly more irregular variant of the above:
(macroexpand '(case-double x
                (0.0 -1.0 -2.0 -3.0 -4.0) :nonpositive
                (     1.0  2.0  3.0  4.0) :positive))
;;=> expands to something like:
(let* [raw-bits2016 (Double/doubleToRawLongBits x)]
  (if (bit-test raw-bits2016 63)
    :nonpositive
    (if (bit-test raw-bits2016 62)
      :positive
      (if (bit-test raw-bits2016 61)
        :positive
        :nonpositive))))

(macroexpand '(case-double x
                0.0 0
                45.0 1
                90.0 2
                135.0 3
                180.0 4
                225.0 5
                270.0 6
                315.0 7))
;;=>
(let [raw-bits15408 (Double/doubleToRawLongBits x)]
  (if (bit-test raw-bits15408 50)
    (if (bit-test raw-bits15408 53)
      (if (bit-test raw-bits15408 51)
        5
        4)
      (if (bit-test raw-bits15408 52)
        2
        1))
    (if (bit-test raw-bits15408 52)
```

```
        (if (bit-test raw-bits15408 49)
          7
          6)
        (if (bit-test raw-bits15408 62)
          3
          0))))

  *e)
```

9.9 Helpers for GridFire development

9.9.1 gridfire.structs.burn-vector

```
(ns gridfire.structs.burn-vector
  "Custom data structure for representing Burn Vectors, isolated in this namespace to wrap JVM interop.")

(defrecord BurnVector
  [^long   i
   ^long   j
   ;; The above i,j coordinates locate the origin of this burn vector.
   ;; The following direction (a multiple of 45°) locates its target cell:
   ^double direction
   ^double fractional-distance
   ^double spread-rate
   ^double terrain-distance
   ^double burn-probability])

(defmacro make-burn-vector
  "Fast constructor for BurnVector."
  [& args]
  ;; More efficient than ->BurnVector, in particular because it avoids boxing of arguments.
  `(BurnVector. ~@args))

(defn get-i
  ^long [^BurnVector bv]
  (.-i bv))

(defn get-j
  ^long [^BurnVector bv]
  (.-j bv))

(defn get-direction
  "The direction of the BurnVector, a double among #{0.0 45.0 90.0 ...}"
  ^double [^BurnVector bv]
  (.-direction bv))

(defn get-fractional-distance
  "Represents the progress of the Burn Vector along its travel line. In particular:
  - 0.5 is the starting point (center of the cell);
  - 1.0 is the crossing point to the target cell;
  - > 1.0 can only be an ephemeral state (calls for transition);
  - < 0.5 is in practice the 2nd half of the travel (after transitioning to target cell);
  - > 0.5 is in practice the 1st half of the travel (after igniting, before transitioning)."
  ^double [^BurnVector bv]
  (.-fractional-distance bv))

(defn get-spread-rate
  "The current speed of the Burn Vector."
  ^double [^BurnVector bv]
  (.-spread-rate bv))

(defn get-terrain-distance
  ^double [^BurnVector bv]
  (.-terrain-distance bv))

(defn get-burn-probability
```

```
  ^double [^BurnVector bv]
  (.-burn-probability bv))
```

9.9.2 gridfire.utils.geo

```
(ns gridfire.utils.geo
  (:import (org.geotools.referencing CRS)
           (org.opengis.geometry Envelope)))

(defn- envelope-lower-corner-lat+lon
  [^Envelope envelope]
  (let [crs-4326  (CRS/decode "EPSG:4326")                  ; NOTE I checked that this call is fast. (Val, 01 Nov 2022)
        lc        (-> envelope
                      ;; Transforming to a CRS that will give us latitude/longitude coordinates.
                      (CRS/transform crs-4326)
                      (.getLowerCorner))
        [lat lon] (.getCoordinate lc)]
    [lat lon]))

(defn resolve-simulation-lat+lon
  "Returns a [latitude longitude] (in degrees) pair for some point in the :envelope of the given GridFire inputs map.

  The returned latitude and longitude are in angular degrees."
  [inputs]
  ;; NOTE why use the Lower Corner? It doesn't matter much which point we use, (Val, 01 Nov 2022)
  ;; because we're not after high accuracy.
  ;; We could use the Median point, but that would require
  ;; a GeoTools upgrade.
  (envelope-lower-corner-lat+lon (:envelope inputs)))
```

9.9.3 gridfire.utils.geo-test

```
(ns gridfire.utils.geo-test
  (:require [clojure.test        :refer [deftest is testing use-fixtures]]
            [gridfire.core       :refer [load-config! load-inputs!]]
            [gridfire.utils.geo  :refer [resolve-simulation-lat+lon]]
            [gridfire.utils.test :refer [with-temp-directories]]))

(use-fixtures :once (with-temp-directories ["outputs/simple/"]))

(deftest ^:unit resolve-simulation-lat+lon-example-test
  (testing `resolve-simulation-lat+lon
    (testing "on our Canonical config"
      (let [config (load-config! "test/gridfire/resources/canonical_test/base-config.edn")
            inputs (load-inputs! config)]
        (is (= [36.144687673692644 -123.00000000000001]
               (resolve-simulation-lat+lon inputs))
            "returns an USA-typical [lat lon] pair."))))))
```

9.10 Code for Application-Specific Deployments of GridFire

9.10.1 gridfire.active-fire-watcher-test

```
(ns gridfire.active-fire-watcher
  (:require [clojure.core.async   :refer [<! >! go-loop timeout]]
            [clojure.edn          :as edn]
            [clojure.string       :as s]
            [gridfire.conversion  :refer [convert-date-string]]
            [nextjournal.beholder :as beholder]
            [triangulum.logging   :refer [log-str]])
  (:import java.util.Date))

(set! *unchecked-math* :warn-on-boxed)
```

```
;;=============================================================================
;; File Path -> Job
;;=============================================================================

(def file-name-regex #"[^/]*(?=[.][a-zA-Z]+$)")

(def fire-name-regex #"[a-zA-Z]*[-[a-zA-Z0-9]]*")

(def ignition-time-regex #"\d{8}_\d{6}")

(defn- build-job [path-str]
  (let [file-name     (re-find file-name-regex path-str)
        fire-name     (re-find fire-name-regex file-name)
        ignition-time (re-find ignition-time-regex file-name)]
    {:fire-name     fire-name
     :type          :active-fire
     :ignition-time (convert-date-string ignition-time
                                         "yyyyMMdd_HHmmss"
                                         "yyyy-MM-dd HH:mm zzz")}))

(defn- suppress? [{:keys [also-simulate-suppression? suppression-white-list]} file-path]
  (and also-simulate-suppression?
       (some #(s/includes? file-path %) (-> (slurp suppression-white-list)
                                            (edn/read-string)
                                            (:suppression-white-list)))))

;;=============================================================================
;; Server
;;=============================================================================

(defonce download-in-progress (atom {}))

(defn- now ^long []
  (inst-ms (Date.)))

(defn- still-waiting? [^long last-mod-time]
  (< (- (now) last-mod-time) 5000)) ; wait up to 5 seconds

(defn- count-down [config job-queue path-str]
  (go-loop [^long last-mod-time (get @download-in-progress path-str)]
    (if (still-waiting? last-mod-time)
      (do (<! (timeout 1000))
          (recur (get @download-in-progress path-str)))
      (do (log-str "SCP Complete: " path-str)
          (swap! download-in-progress dissoc path-str)
          (let [job (build-job path-str)]
            (>! job-queue job)
            (when (suppress? config path-str)
              (>! job-queue (assoc job :suppression {:suppression-dt          60.0 ;TODO remove hard code
                                                     :suppression-coefficient 2.0}))))))))) ;TODO remove hard code

(defn- handler [config job-queue]
  (fn [{:keys [type path]}]
    (let [path-str (.toString path)]
      (case type
        :create (do (log-str "Active Fire input deck detected: " path-str)
                    (swap! download-in-progress assoc path-str (now))
                    (count-down config job-queue path-str))
        :modify (swap! download-in-progress assoc path-str (now)) ; reset counter
        nil))))

(defonce directory-watcher (atom nil))

(defn start! [{:keys [active-fire-dir] :as config} job-queue]
  (when (and active-fire-dir (nil? @directory-watcher))
    (reset! directory-watcher
            (beholder/watch (handler config job-queue) active-fire-dir))))
```

```
(defn stop! []
  (when @directory-watcher
    (beholder/stop @directory-watcher)
    (reset! directory-watcher nil)))
```

9.10.2 gridfire.active-fire-watcher-test

```
(ns gridfire.active-fire-watcher-test
  (:require [gridfire.active-fire-watcher :refer [file-name-regex
                                                  fire-name-regex
                                                  ignition-time-regex]]
            [clojure.test :refer [deftest is]]))

(deftest ^:unit file-name-regex-test

  (is (= "nm-johnson_20210617_203600_001"
         (->>  "nm-johnson_20210617_203600_001.tar"
               (re-find file-name-regex))))

  (is (= "nm-johnson_20210617_203600_001"
         (->>  "/home/nm-johnson_20210617_203600_001.tar"
               (re-find file-name-regex))))

  (is (= "nm-johnson_20210617_203600_001"
         (->>  "/gridfire/data/nm-johnson_20210617_203600_001.tar"
               (re-find file-name-regex)))))

(deftest ^:unit fire-name-regex-test

  (is (= "nm-johnson"
         (->>  "nm-johnson_20210617_203600_001"
               (re-find fire-name-regex))))

  (is (= "ut-bear-bennion-creek"
         (->>  "ut-bear-bennion-creek_20210610_184300_001"
               (re-find fire-name-regex))))
  (is (= "nm"
         (->>  "nm_20210617_203600_001"
               (re-find fire-name-regex))))

  (is (= "ca-success-2"
         (->>  "ca-success-2_20210618_000000_001.tar"
               (re-find fire-name-regex)))))

(deftest ^:unit ignition-time-regex-test

  (is (= "20210617_203600"
         (->>  "nm-johnson_20210617_203600_001"
               (re-find ignition-time-regex)))))
```

9.10.3 PyreCast Server Interface

FIXME Deprecation notice.

(EDIT: For the purpose of PyreCast,) the GridFire system is also available for use in server mode. The GridFire server will listen for requests to launch fire sparead simulations. Upon completion of the simulation a set of post processing scripts will run to process binary outputs into a directory structure containing geoTIFF files and then packed into a tarball. This tarball is sent over scp into another server for processing.

1. Preprocessing Dependencies

The processing of active fire and match drop runs require the following package to work:

- babashka

2. Postprocessing Dependencies

   The Post processing scripts require the following packages to work:

   - ssh
   - pigz
   - mpirun
   - gdal
   - elmfire_post_¡elmfire_version¿ (for elmfire version see ‘resources/elmfire_post.sh‘)

3. Server Configuration

   The GridFire system is also available for use in server mode. The server is confgured by and edn file containing the following contents:

```
{:software-dir    "/gridfire/software"
 :incoming-dir    "/gridfire/incoming"
 :active-fire-dir "/gridfire/incoming/active_fires"
 :data-dir        "/gridfire/data"
 :log-dir         "/gridfire/log"}
```

   software-dir The directory the gridfire repo is cloned into.

   incoming-dir The directory where the server will look for match-drop input decks (which should be uploaded from the data provisioning server on wx.pyregence.org). The server keeps the latest 20 input decks. (see cleanup.sh in this directory)

   active-fire-dir The directory where the server will look for active-fire input decks (which are uploaded from the data provisioning server on wx.pyregence.org). The server's file watcher thread will automatically add a request onto the server's ‘standby-queue‘ whenever a new input deck is uploaded. The server keeps the latest 200 input decks. (see cleanup.sh in this directory)

   data-dir The directory into which input deck tarballs from incoming-dir are unpacked. The server keeps the latest 20 unpacked tarballs. (see cleanup.sh in this directory)

   log-dir The directory into which log files are written. The server keeps the last 10 days of logs.

   The GridFire server is launched by user ‘gridfire‘ with this command:

```
clojure -M:run-server -c server.edn
```

4. Implementing Code

   (a) gridfire.server.pyrecast-async

```
(ns gridfire.server.pyrecast-async
  "For exposing GridFire through a socket server, making it act as a worker process behind a job queue,
  which sends notifications to the client as the handling progresses.

  This is an in-process (not distributed), singleton (its state is namespace-anchored, being held in Vars),
  single-threaded server, which implies some usage limitations (you can't have several servers in the same JVM,
```

```
  there's no resilience to JVM crashes, and you can't scale out to several worker processes behind the job queue.)"
  (:require [clojure.core.async           :refer [>!! alts!! chan thread]]
            [clojure.data.json            :as json]
            [clojure.edn                  :as edn]
            [clojure.java.io              :as io]
            [clojure.java.shell           :as sh]
            [clojure.pprint               :refer [pprint]]
            [clojure.spec.alpha           :as spec]
            [clojure.string               :as str]
            [gridfire.active-fire-watcher :as active-fire-watcher]
            [gridfire.conversion          :refer [convert-date-string camel->kebab kebab->camel]]
            [gridfire.core                :as gridfire]
            [gridfire.simple-sockets      :as sockets]
            [gridfire.spec.server         :as server-spec]
            [gridfire.utils.server        :refer [nil-on-error throw-message]]
            [triangulum.logging           :refer [log log-str set-log-path!]]
            [triangulum.utils             :refer [parse-as-sh-cmd]])
  (:import java.text.SimpleDateFormat
           java.util.Calendar
           java.util.TimeZone))

(set! *unchecked-math* :warn-on-boxed)

;;=============================================================================
;; Request/Response Functions
;;=============================================================================

(def date-from-format "yyyy-MM-dd HH:mm zzz")
(def date-to-format   "yyyyMMdd_HHmmss")

(defn- build-geosync-request [{:keys [fire-name ignition-time suppression] :as _request}
                              {:keys [geosync-data-dir host] :as _config}]
  (let [updated-fire-name (if suppression (str fire-name "-suppressed") fire-name)
        timestamp         (convert-date-string ignition-time date-from-format date-to-format)]
    (json/write-str
     {"action"             "add"
      "dataDir"            (format "%s/%s/%s" geosync-data-dir updated-fire-name timestamp)
      "geoserverWorkspace" (format "fire-spread-forecast_%s_%s" updated-fire-name timestamp)
      "responseHost"       host
      "responsePort"       5555})))

(defn- send-geosync-request! [request {:keys [geosync-host geosync-port] :as config}]
  (sockets/send-to-server! geosync-host
                           geosync-port
                           (build-geosync-request request config)))

(defn- build-gridfire-response [request {:keys [host port] :as _config} status status-msg]
  (json/write-str (merge request
                         {:status        status
                          :message       status-msg
                          :response-host host
                          :response-port port})
                  :key-fn (comp kebab->camel name)))

(defn- send-gridfire-response! [{:keys [response-host response-port] :as request} config status status-msg]
  (when (spec/valid? ::server-spec/gridfire-server-response-minimal request)
    (sockets/send-to-server! response-host
                             response-port
                             (build-gridfire-response request config status status-msg))))

;;=============================================================================
;; Process override-config
;;=============================================================================

(defn- add-ignition-start-timestamp [config ignition-date-time]
  (assoc config :ignition-start-timestamp ignition-date-time))

(defn- calc-weather-start-timestamp [ignition-date-time]
```

```
  (doto (Calendar/getInstance (TimeZone/getTimeZone "UTC"))
    (.setTime ignition-date-time)
    (.set Calendar/MINUTE 0)))

(defn- add-weather-start-timestamp [config ignition-date-time]
  (assoc config :weather-start-timestamp (calc-weather-start-timestamp ignition-date-time)))

(defn- write-config! [output-file config]
  (log-str "Writing to config file: " output-file)
  (with-open [writer (io/writer output-file)]
    (pprint config writer)))

(defn- process-override-config! [{:keys [ignition-time] :as _request} file]
  (let [formatter          (SimpleDateFormat. "yyyy-MM-dd HH:mm zzz")
        ignition-date-time (.parse formatter ignition-time)
        config             (edn/read-string (slurp file))]
    (write-config! file
                   (-> config
                       (add-ignition-start-timestamp ignition-date-time)
                       (add-weather-start-timestamp ignition-date-time)))))

;;================================================================================
;; Process suppression-params
;;================================================================================

(defn- add-suppression [config {:keys [suppression-dt suppression-coefficient] :as _suppression-params}]
  (assoc config :suppression {:suppression-dt          suppression-dt
                              :suppression-coefficient suppression-coefficient}))

(defn- update-suppression-params! [gridfire-edn-path {:keys [suppression] :as _request}]
  (let [config (edn/read-string (slurp gridfire-edn-path))]
    (if suppression
      (write-config! gridfire-edn-path (add-suppression config suppression))
      config)))

;;================================================================================
;; Shell Commands
;;================================================================================

(def path-env (System/getenv "PATH"))

(defn- sh-wrapper [dir env verbose & commands]
  (sh/with-sh-dir dir
    (sh/with-sh-env (merge {:PATH path-env} env)
      (reduce (fn [acc cmd]
                (let [{:keys [out err]} (apply sh/sh (parse-as-sh-cmd cmd))]
                  (str acc (when verbose out) err)))
              ""
              commands))))

;;================================================================================
;; Request Processing Functions
;;================================================================================

(defn- run-post-process-scripts! [request config output-dir]
  (log-str "Running post-process scripts.")
  (let [commands [["./elmfire_post.sh" "Running elmfire_post."]
                  ["./make_tifs.sh" "Creating GeoTIFFs."]
                  ["./build_geoserver_directory.sh"]
                  ["./upload_tarball.sh"]
                  ["./cleanup.sh"]]]
    (doseq [[cmd response-msg] commands]
      (when response-msg
        (send-gridfire-response! request config 2 response-msg))
      (-> (sh-wrapper output-dir {} true cmd)
          (log :truncate? false :newline? false)))))

(defn- copy-post-process-scripts! [from-dir to-dir]
```

```
  (log-str "Copying post-process scripts into " to-dir)
  (sh-wrapper from-dir
              {}
              false
              (str "cp resources/elmfire_post.sh " to-dir)
              (str "cp resources/make_tifs.sh " to-dir)
              (str "cp resources/build_geoserver_directory.sh " to-dir)
              (str "cp resources/upload_tarball.sh " to-dir)
              (str "cp resources/cleanup.sh " to-dir)))

(defn- build-file-name [fire-name ignition-time]
  (str/join "_" [fire-name (convert-date-string ignition-time date-from-format date-to-format) "001"]))

(defn- unzip-tar!
  "Unzips a tar file and returns the file path to the extracted folder."
  [{:keys [fire-name ignition-time type suppression] :as _request}
   {:keys [data-dir incoming-dir active-fire-dir] :as _config}]
  (let [working-dir   (if (= type :active-fire) active-fire-dir incoming-dir)
        tar-file-name (str (build-file-name fire-name ignition-time) ".tar")
        out-file-name (build-file-name (if suppression (str fire-name "-suppressed") fire-name)
                                       ignition-time)
        out-file-path (.getAbsolutePath (io/file data-dir out-file-name))]
    (if (.exists (io/file working-dir tar-file-name))
      (do
        (sh/sh "mkdir" "-p" out-file-path)
        (sh-wrapper working-dir
                    {}
                    true
                    (format "tar -xf %s -C %s --strip-components 1"
                            tar-file-name
                            out-file-path))
        (.getPath (io/file data-dir out-file-name)))
      (throw (Exception. (str "tar file does not exist: " (.getAbsolutePath (io/file working-dir tar-file-name))))))))

;;TODO Try babashka's pod protocol to see if it's faster than shelling out.
(defn- process-request!
  "Runs the requested simulation using the supplied config.

  WARNING: because each simulation requires exclusive access to various resources (e.g all the processors),
  do not make several parallel calls to this function."
  [request {:keys [software-dir override-config backup-dir] :as config}]
  (try
    (let [input-deck-path     (unzip-tar! request config)
          elmfire-data-file   (.getPath (io/file input-deck-path "elmfire.data"))
          gridfire-edn-file   (.getPath (io/file input-deck-path "gridfire.edn"))
          gridfire-output-dir (.getPath (io/file input-deck-path "outputs"))
          {:keys [err out]}   (if override-config
                                (do
                                  (process-override-config! request override-config)
                                  (sh/sh "resources/elm_to_grid.clj" "-e" elmfire-data-file "-o" override-config)
                                  (update-suppression-params! gridfire-edn-file request))
                                (sh/sh "resources/elm_to_grid.clj" "-e" elmfire-data-file))]
      (if err
        (log-str out "\n" err)
        (log-str out))
      (send-gridfire-response! request config 2 "Running simulation.")
      (if (gridfire/process-config-file! gridfire-edn-file) ; Returns true on success
        (do (copy-post-process-scripts! software-dir gridfire-output-dir)
            (run-post-process-scripts! request config gridfire-output-dir)
            (when backup-dir
              (sh/sh "tar" "-czf"
                     (str backup-dir "/" (.getName (io/file input-deck-path)) ".tar.gz")
                     (.getAbsolutePath (io/file input-deck-path))
                     (str (.getName (io/file input-deck-path)) ".tar.gz") ))
            (sh/sh "rm" "-rf" (.getAbsolutePath (io/file input-deck-path)))
            [0 "Successful run! Results uploaded to GeoServer!"])
        (throw-message "Simulation failed. No results uploaded to GeoServer.")))
    (catch Exception e
```

```
      [1 (str "Processing error: " (ex-message e))])))

;;=============================================================================
;; Job Queue Management
;;=============================================================================

(defonce *job-queue-size      (atom 0))
(defonce *stand-by-queue-size (atom 0))

(defonce =job-queue=
         (chan 10 (map (fn [x]
                         (swap! *job-queue-size inc)
                         (delay (swap! *job-queue-size dec) x)))))

(defonce =stand-by-queue=
         (chan 10 (map (fn [x]
                         (swap! *stand-by-queue-size inc)
                         (delay (swap! *stand-by-queue-size dec) x)))))

(defonce *server-running? (atom false))

(defn- process-requests-loop!
  "Starts a logical process which listens to queued requests and processes them.

  Requests are processed in order of priority, then FIFO.

  Returns a core.async channel which will close when the server is stopped."
  [config]
  (reset! *server-running? true)
  (thread
    (loop [request @(first (alts!! [=job-queue= =stand-by-queue=] :priority true))]
      (log-str "Processing request: " request)
      (let [[status status-msg] (process-request! request config)]
        (log-str "-> " status-msg)
        (if (= (:type request) :active-fire)
          (send-geosync-request! request config)
          (send-gridfire-response! request config status status-msg)))
      (when @*server-running?
        (recur @(first (alts!! [=job-queue= =stand-by-queue=] :priority true)))))))

(defn- maybe-add-to-queue! [request]
  (try
    (if (spec/valid? ::server-spec/gridfire-server-request request)
      (do (>!! =job-queue= request)
          [2 (format "Added to job queue. You are number %d in line." @*job-queue-size)])
      [1 (str "Invalid request: " (spec/explain-str ::server-spec/gridfire-server-request request))])
    (catch AssertionError _
      [1 "Job queue limit exceeded! Dropping request!"])
    (catch Exception e
      [1 (str "Validation error: " (ex-message e))])))

(defn- parse-request-msg
  "Parses the given JSON-encoded request message, returning a request map (or nil in case of invalid JSON)."
  [request-msg]
  (-> request-msg
      (json/read-str :key-fn (comp keyword camel->kebab))
      (nil-on-error)))

;; Logically speaking, this function does (process-request! (parse-request-msg request-msg) config).
;; However, in order to both limit the load and send progress-notification responses before completion,
;; the handling goes through various queues and worker threads.
(defn- schedule-handling! [config request-msg]
  (thread
    (log-str "Request: " request-msg)
    (if-let [request (parse-request-msg request-msg)]
      (let [[status status-msg] (maybe-add-to-queue! request)]
        (log-str "-> " status-msg)
        (send-gridfire-response! request config status status-msg))
```

```
      (log-str "-> Invalid JSON"))))

;;=============================================================================
;; Start/Stop Servers
;;=============================================================================

(defn start-server! [{:keys [log-dir port] :as config}]
  (when log-dir (set-log-path! log-dir))
  (log-str "Running server on port " port)
  (active-fire-watcher/start! config =stand-by-queue=)
  (sockets/start-server! port (fn [request-msg] (schedule-handling! config request-msg)))
  (process-requests-loop! config))

(defn stop-server! []
  (reset! *server-running? false)
  (sockets/stop-server!)
  (active-fire-watcher/stop!))

;; TODO write spec for server
;; Sample config
#_{:software-dir               "/home/kcheung/work/code/gridfire"
 :incoming-dir               "/home/kcheung/work/servers/chickadee/incoming"
 :active-fire-dir            "/home/kcheung/work/servers/chickadee/incoming/active_fires"
 :data-dir                   "/home/kcheung/work/servers/chickadee/data"
 :override-config            "/home/kcheung/work/servers/chickadee/gridfire-base.edn"
 :log-dir                    "/home/kcheung/work/servers/chickadee/log"
 :backup-dir                 "/home/kcheung/work/servers/chickadee/backup-to-ftp"
 :suppression-white-list     "/home/kcheung/work/servers/chickadee/suppression-white-list.edn"
   :also-simulate-suppression? true}
```

(b) gridfire.server.pyrecast-async-test

```
(ns gridfire.server.pyrecast-async-test
  (:require [clojure.java.io                :as io]
            [clojure.java.shell             :refer [sh]]
            [clojure.test                   :refer [deftest is use-fixtures testing]]
            [gridfire.server.pyrecast-async :as server]
            [gridfire.utils.test            :refer [with-temp-directories]]))

;;-----------------------------------------------------------------------------
;; Paths
;;-----------------------------------------------------------------------------

(def data-dir        "test/gridfire/temp/data/")
(def incoming-dir    "test/gridfire/temp/incoming/")
(def active-fire-dir "test/gridfire/temp/active_fires/")

;;-----------------------------------------------------------------------------
;; Fixtures
;;-----------------------------------------------------------------------------

(use-fixtures :once (with-temp-directories [data-dir incoming-dir active-fire-dir]))

;;-----------------------------------------------------------------------------
;; Tests
;;-----------------------------------------------------------------------------

(deftest ^:unit unzip-tar-test
  (sh "cp" "test/gridfire/resources/server_test/unzip-fire-test_19700101_000000_001.tar" active-fire-dir)
  (let [request-base {:fire-name     "unzip-fire-test"
                      :ignition-time "1970-01-01 00:00 UTC"
                      :type          :active-fire}
        config       {:data-dir        data-dir
                      :incoming-dir    incoming-dir
                      :active-fire-dir active-fire-dir}]
    (testing "unsuppressed"
      (#'server/unzip-tar! request-base config)
      (is (.exists (io/file data-dir "unzip-fire-test_19700101_000000_001"))))
```

```
    (testing "suppressed"
      (#'server/unzip-tar! (assoc request-base :suppression  {:suppression-dt         1440.0
                                                              :suppression-coefficent 2.0})
                           config)
      (is (.exists (io/file data-dir "unzip-fire-test-suppressed_19700101_000000_001"))))))
```

(c) gridfire.elm-to-grid-test

```
(ns gridfire.elm-to-grid-test
  (:require [clojure.java.io      :as io]
            [clojure.java.shell   :as sh]
            [clojure.spec.alpha   :as s]
            [clojure.test         :refer [deftest is testing use-fixtures]]
            [gridfire.spec.common :refer [*check-files-exist?*]]
            [gridfire.spec.config :as spec]
            [gridfire.utils.files :as files]
            [gridfire.utils.test  :refer [with-temp-directories]]))

(def elm-to-grid-path "./resources/elm_to_grid.clj")

;;------------------------------------------------------------------------------
;; Fixtures
;;------------------------------------------------------------------------------

(defn- delete-file-if-exists
  [file]
  (let [f (io/file file)]
    (when (.exists f)
      (.delete f))))

(use-fixtures :once
  (with-temp-directories ["test/gridfire/resources/config_test/outputs"]))

;;------------------------------------------------------------------------------
;; Tests
;;------------------------------------------------------------------------------

(defn- is-emitted-gridfire-end-as-expected
  [elm-to-grid-sh-args emitted-gfr-conf-path expected-gfr-conf-path]
  (try
    (let [ret (apply sh/sh elm-to-grid-path elm-to-grid-sh-args)]
      (or (is (= 0 (:exit ret)) "The script completed successfully.")
          ;; Throwing to avoid continuing execution of the test.
          (throw (ex-info (format "elm_to_grid.clj returned exited with error status %d: %s"
                                  (:exit ret)
                                  (:err ret))
                          ret))))
    (comment
      ;; TIP: Run this when you want to update the expected config after changing elm_to_grid.
      (sh/sh "cp" emitted-gfr-conf-path expected-gfr-conf-path))
    (testing "The emitted GridFire config"
      (let [config (files/read-situated-edn-file emitted-gfr-conf-path)]
        (if-not (is (s/valid? ::spec/config config) "is valid.")
          (s/explain ::spec/config config)
          (is (= config
                 (files/read-situated-edn-file expected-gfr-conf-path))
              ;; This testing technique is sometimes known as 'Snapshot Testing'.
              "has not changed."))))
    (finally
      (delete-file-if-exists emitted-gfr-conf-path))))

(deftest ^:unit elm-to-grid-examples-test
  (testing "Basic example."
    (let [elm-data-path "test/gridfire/resources/config_test/sample-elmfire.data"
          gfr-conf-path "test/gridfire/resources/config_test/gridfire.edn"]
      (is-emitted-gridfire-end-as-expected ["--elmfire-config" elm-data-path]
                                           gfr-conf-path
```

```
                                          "test/gridfire/resources/config_test/expected-gridfire.edn")))
  (testing "FF AWS example."
    (binding [*check-files-exist?* false]
      (let [dir-path      "test/gridfire/resources/elmfire-examples/ff-aws-2022-10/"
            elm-data-path (str dir-path "elmfire.data")
            gfr-conf-path (str dir-path "gridfire.edn")]
        (testing "With minimal options."
          (is-emitted-gridfire-end-as-expected ["--elmfire-config" elm-data-path]
                                               gfr-conf-path
                                               (str dir-path "expected-0-gridfire.edn")))
        (testing "With maximal options."
          (is-emitted-gridfire-end-as-expected ["--elmfire-config"                    elm-data-path
                                                "--override-config"                   (str dir-path "gridfire_base.edn"
                                                ;; NOTE created manually by editing another file. (Val, 10 Nov 2022)
                                                "--elmfire-summary-csv"               (str dir-path "synthetic-elmfire-
                                                "--pyrome-spread-rate-adjustment-csv" (str dir-path "pyrome_adjustment_
                                                "--pyrome-calibration-csv"            (str dir-path "pyrome_calibration
                                               gfr-conf-path
                                               (str dir-path "expected-1-gridfire.edn")))))))
```

(d) gridfire.spec.server

```
(ns gridfire.spec.server
  (:require [clojure.spec.alpha    :as spec]
            [gridfire.utils.server :refer [hostname? port? fire-name? time?]]))

(spec/def ::response-host hostname?)
(spec/def ::response-port port?)
(spec/def ::fire-name     fire-name?)
(spec/def ::ignition-time time?)
(spec/def ::type          #{:active-fire}) ; TODO: Add more types as needed

(spec/def ::gridfire-server-request
  (spec/keys
   :req-un [::fire-name ::ignition-time]
   :opt-un [::response-host ::response-port ::type]))

(spec/def ::gridfire-server-response-minimal
  (spec/keys
   :req-un [::response-host ::response-port]))
```

(e) gridfire.utils.server

```
(ns gridfire.utils.server
  (:require [clojure.string :as s]))

(set! *unchecked-math* :warn-on-boxed)

;; TODO extend to more timezones
(def time-regex
  "regex for timestamp with the form YYYY-MM-DD HH:MM ZZZ
  (i.e. 2021-03-15 05:00 PST)"
  #"[0-9]{4}-(0[1-9]|1[0-2])-(0[1-9]|[1-2][0-9]|3[0-1]) (2[0-3]|[01][0-9]):[0-5][0-9] (PST|UTC|EST)")

(defmacro nil-on-error
  [& body]
  (let [_ (gensym)]
    `(try ~@body (catch Exception ~_ nil))))

(defn non-empty-string?
  [x]
  (and (string? x)
       (pos? (count x))))

(defn hostname?
  [x]
  (or (= x "localhost")
```

```
      (and (non-empty-string? x)
           (s/includes? x ".")
           (not (s/starts-with? x "."))
           (not (s/ends-with? x "."))))))

(defn port?
  [x]
  (and (integer? x)
       (< 0 x 0x10000)))

(defn fire-name?
  [x]
  (string? x))

(defn time?
  [x]
  (some? (re-matches time-regex x)))

(defn throw-message
  [msg]
  (throw (ex-info msg {})))
```

9.11 Build script

9.11.1 build

```
(ns build
  (:require [clojure.tools.build.api :as b])
  (:import java.util.Date))

(defn get-calendar-branch-version []
  (let [today  (Date.)
        commit (b/git-process {:git-args "rev-parse --short HEAD"})]
    (format "%d.%d.%d-%s"
            (+ 1900 (.getYear today))
            (+ 1 (.getMonth today))
            (.getDate today)
            commit)))

(def build-folder "target")
(def jar-content (str build-folder "/classes"))
(def basis (b/create-basis {:project "deps.edn"}))

(def app-name "gridfire")
(def version (get-calendar-branch-version))
(def uberjar-file-name (format "%s/%s-%s.jar" build-folder app-name version))

(defn clean [_]
  (b/delete {:path build-folder})
  (println (format "Build folder \"%s\" removed" build-folder)))

(defn uberjar [_]
  ;; clean up old files
  (clean nil)

  ;; copy resources to jar-content folder
  (b/copy-dir {:src-dirs ["src" "resources"]
               :target-dir jar-content})

  (b/compile-clj {:src-dirs     ["src"]
                  :class-dir    jar-content
                  :bindings     {#'clojure.core/*assert* false}
                  :compile-opts {:direct-linking true}
                  :basis        basis})

  ;; package jar-content into a jar
```

```
(b/uber {:class-dir jar-content
         :uber-file uberjar-file-name
         :basis     basis
         :main      'gridfire.cli
         :manifest  {"Specification-Title"    "Java Advanced Imaging Image I/O Tools"
                     "Specification-Version"  "1.1"
                     "Specification-Vendor"   "Sun Microsystems, Inc."
                     "Implementation-Title"   "com.sun.media.imageio"
                     "Implementation-Version" "1.1"
                     "Implementation-Vendor"  "Sun Microsystems, Inc."}})

(println (format "Uberjar file created: \"%s\"" uberjar-file-name)))
```

10 FIXME: Unfiled Namespaces

10.0.1 gridfire-unmerged.scale-assessment

```
;; (ns gridfire-unmerged.scale-assessment
;;   (:require [clojure.java.io :as io]
;;             [tech.v3.tensor :as t]
;;             [tech.v3.tensor.operators :as top]
;;             [clojure.core.reducers :as r]
;;             [clojure.data.csv :as csv]
;;             [incanter.core :as incanter]
;;             [incanter.charts :as charts]
;;             [incanter.stats :as stats]
;;             [matrix-viz.core :refer [save-matrix-as-png]]
;;             [gridfire.postgis-bridge :refer [postgis-raster-to-matrix]]
;;             [gridfire.fire-spread :refer [random-cell select-random-ignition-site burnable? burnable-neighbors?]]
;;             [gridfire.monte-carlo :refer [run-monte-carlo-fire-spread]]))

;; (def db-spec {:classname   "org.postgresql.Driver"
;;               :subprotocol "postgresql"
;;               :subname     "//localhost:5432/calfire"
;;               :user        "gjohnson"})

;; (def landfire-layers-by-tile
;;   {:tile205 {:elevation     (postgis-raster-to-matrix db-spec "validation.elevation_tile205"     nil nil)
;;              :slope         (postgis-raster-to-matrix db-spec "validation.slope_tile205"         nil nil)
;;              :aspect        (postgis-raster-to-matrix db-spec "validation.aspect_tile205"        nil nil)
;;              :fuel-model    (postgis-raster-to-matrix db-spec "validation.fuel_model_tile205"    nil nil)
;;              :canopy-height (postgis-raster-to-matrix db-spec "validation.canopy_height_tile205" nil nil)
;;              :canopy-cover  (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile205"  nil nil)}
;;    :tile210 {:elevation     (postgis-raster-to-matrix db-spec "validation.elevation_tile210"     nil nil)
;;              :slope         (postgis-raster-to-matrix db-spec "validation.slope_tile210"         nil nil)
;;              :aspect        (postgis-raster-to-matrix db-spec "validation.aspect_tile210"        nil nil)
;;              :fuel-model    (postgis-raster-to-matrix db-spec "validation.fuel_model_tile210"    nil nil)
;;              :canopy-height (postgis-raster-to-matrix db-spec "validation.canopy_height_tile210" nil nil)
;;              :canopy-cover  (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile210"  nil nil)}
;;    :tile281 {:elevation     (postgis-raster-to-matrix db-spec "validation.elevation_tile281"     nil nil)
;;              :slope         (postgis-raster-to-matrix db-spec "validation.slope_tile281"         nil nil)
;;              :aspect        (postgis-raster-to-matrix db-spec "validation.aspect_tile281"        nil nil)
;;              :fuel-model    (postgis-raster-to-matrix db-spec "validation.fuel_model_tile281"    nil nil)
;;              :canopy-height (postgis-raster-to-matrix db-spec "validation.canopy_height_tile281" nil nil)
;;              :canopy-cover  (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile281"  nil nil)}
;;    :tile310 {:elevation     (postgis-raster-to-matrix db-spec "validation.elevation_tile310"     nil nil)
;;              :slope         (postgis-raster-to-matrix db-spec "validation.slope_tile310"         nil nil)
;;              :aspect        (postgis-raster-to-matrix db-spec "validation.aspect_tile310"        nil nil)
;;              :fuel-model    (postgis-raster-to-matrix db-spec "validation.fuel_model_tile310"    nil nil)
;;              :canopy-height (postgis-raster-to-matrix db-spec "validation.canopy_height_tile310" nil nil)
;;              :canopy-cover  (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile310"  nil nil)}
;;    :tile564 {:elevation     (postgis-raster-to-matrix db-spec "validation.elevation_tile564"     nil nil)
;;              :slope         (postgis-raster-to-matrix db-spec "validation.slope_tile564"         nil nil)
;;              :aspect        (postgis-raster-to-matrix db-spec "validation.aspect_tile564"        nil nil)
;;              :fuel-model    (postgis-raster-to-matrix db-spec "validation.fuel_model_tile564"    nil nil)
```

```
;;              :canopy-height (postgis-raster-to-matrix db-spec "validation.canopy_height_tile564" nil nil)
;;              :canopy-cover  (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile564"  nil nil)}
;;    :tile643 {:elevation     (postgis-raster-to-matrix db-spec "validation.elevation_tile643"     nil nil)
;;              :slope         (postgis-raster-to-matrix db-spec "validation.slope_tile643"         nil nil)
;;              :aspect        (postgis-raster-to-matrix db-spec "validation.aspect_tile643"        nil nil)
;;              :fuel-model    (postgis-raster-to-matrix db-spec "validation.fuel_model_tile643"    nil nil)
;;              :canopy-height (postgis-raster-to-matrix db-spec "validation.canopy_height_tile643" nil nil)
;;              :canopy-cover  (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile643"  nil nil)}})

;; (defn load-extreme-weather-readings
;;   [csv-filename]
;;   (mapv (fn [[x y date hour percentile temp rh ws gust wd ffwi ffwig]]
;;           {:x          (read-string x)
;;            :y          (read-string y)
;;            :date       date
;;            :hour       (read-string hour)
;;            :percentile (read-string percentile)
;;            :temp       (read-string temp)
;;            :rh         (read-string rh)
;;            :ws         (read-string ws)
;;            :gust       (read-string gust)
;;            :wd         (read-string wd)
;;            :ffwi       (read-string ffwi)
;;            :ffwig      (read-string ffwig)})
;;         (with-open [in-file (io/reader csv-filename)]
;;           (doall (rest (csv/read-csv in-file))))))

;; (def extreme-weather-readings
;;   {:tile205 (load-extreme-weather-readings "resources/tile205_extreme_ffwig_percentiles.csv")
;;    :tile210 (load-extreme-weather-readings "resources/tile210_extreme_ffwig_percentiles.csv")
;;    :tile281 (load-extreme-weather-readings "resources/tile281_extreme_ffwig_percentiles.csv")
;;    :tile310 (load-extreme-weather-readings "resources/tile310_extreme_ffwig_percentiles.csv")
;;    :tile564 (load-extreme-weather-readings "resources/tile564_extreme_ffwig_percentiles.csv")
;;    :tile643 (load-extreme-weather-readings "resources/tile643_extreme_ffwig_percentiles.csv")})

;; (def global-cell-size 98.425) ; 30m
;; (def global-burn-duration 60.0) ; mins

;; (defn lattice-cell
;;   [total-rows total-cols lattice-rows lattice-cols lattice-number]
;;   (let [di (double (/ total-rows lattice-rows))
;;         dj (double (/ total-cols lattice-cols))
;;         i0 (long (/ di 2.0))
;;         j0 (long (/ dj 2.0))]
;;     [(+ i0 (long (* di (quot lattice-number lattice-rows))))
;;      (+ j0 (long (* dj (rem lattice-number lattice-rows))))]))

;; (defn select-lattice-ignition-site
;;   [fuel-model-matrix lattice-rows lattice-cols lattice-number]
;;   (let [num-rows        (-> (t/tensor->dimensions fuel-model-matrix) :shape first)
;;         num-cols        (-> (t/tensor->dimensions fuel-model-matrix) :shape second)
;;         ignition-matrix (doto (t/new-tensor num-rows num-cols) (mop/+= 1.0))]
;;     (if (and (burnable? ignition-matrix fuel-model-matrix ignition-site)
;;              (burnable-neighbors? ignition-matrix fuel-model-matrix num-rows num-cols ignition-site))
;;       ignition-site)))

;; (defn scenario-sites
;;   [fuel-model-matrix scenario]
;;   (case (int scenario)
;;     1 (repeatedly 92 #(select-random-ignition-site fuel-model-matrix))
;;     2 (filterv identity (r/map #(select-lattice-ignition-site fuel-model-matrix 7 7 %) (range 49)))
;;     3 (filterv identity (r/map #(select-lattice-ignition-site fuel-model-matrix 7 7 %) (range 49)))
;;     4 (filterv identity (r/map #(select-lattice-ignition-site fuel-model-matrix 14 14 %) (range 196)))
;;     5 (filterv identity (r/map #(select-lattice-ignition-site fuel-model-matrix 14 14 %) (range 196)))
;;     6 (let [ignition-matrix (doto (t/new-tensor 67 67) (mop/+= 1.0))]
;;         (filterv (fn [ignition-site]
;;                    (and (burnable? ignition-matrix fuel-model-matrix ignition-site)
;;                         (burnable-neighbors? ignition-matrix fuel-model-matrix 67 67 ignition-site)))
```

```
;;                 (for [i (range 67) j (range 67)] [i j])))
;;     7 (let [num-rows (-> (t/tensor->dimensions fuel-model-matrix) :shape first)
;;             num-cols (-> (t/tensor->dimensions fuel-model-matrix) :shape second)]
;;         (take 1000 (distinct (repeatedly #(random-cell num-rows num-cols)))))))
;;             num-cols (-> (t/tensor->dimensions fuel-model-matrix) :shape second)]
;; (def scenario-samples-per-site
;;   {1 #(vector (rand-int 92))
;;    2 (constantly [0 45 90])
;;    3 (constantly [0 8 16 24 32 40 48 56 64 72 80 88])
;;    4 (constantly [0 45 90])
;;    5 (constantly [0 8 16 24 32 40 48 56 64 72 80 88])
;;    6 #(range 92)
;;    7 #(vector (rand-int 92))})

;; (defn run-scenario-on-tile
;;   [scenario tile]
;;   (let [landfire-layers  (landfire-layers-by-tile tile)
;;         cell-size        global-cell-size
;;         ignition-sites   (let [fm  (landfire-layers :fuel-model)
;;                                fm' (m/submatrix fm 66 67 66 67)]
;;                            (mapv #(mop/+ [66 66] %) (scenario-sites fm' scenario)))
;;         weather-readings (mapv #(assoc % :mparam 0.04) (extreme-weather-readings tile))
;;         samples-per-site (scenario-samples-per-site scenario)
;;         burn-duration    global-burn-duration
;;         ellipse-adjustment-factor 1.0]
;;     (run-monte-carlo-fire-spread landfire-layers cell-size ignition-sites
;;                                  weather-readings samples-per-site burn-duration
;;                                  ellipse-adjustment-factor)))

;; (def wrf-cells
;;   {:tile205 [205  6 14]
;;    :tile210 [210 19 20]
;;    :tile281 [281  8 19]
;;    :tile310 [310 18 12]
;;    :tile564 [564  6  2]
;;    :tile643 [643  5  3]})

;; (defn postprocess-simulation-results
;;   [scenario tile {:keys [results-table burn-prob-matrix]}]
;;   (let [wrf-cell    (wrf-cells tile)
;;         plot-titles ["Scenario 1 - Random (Ignition,WRF) Sample Pairs"
;;                      "Scenario 2 - Coarse Lattice, Few WRF Samples"
;;                      "Scenario 3 - Coarse Lattice, Many WRF Samples"
;;                      "Scenario 4 - Fine Lattice, Few WRF Samples"
;;                      "Scenario 5 - Fine Lattice, Many WRF Samples"
;;                      "Scenario 6 - All 30m Cells, All WRF Samples"
;;                      "Scenario 7 - Random (Ignition,WRF) Sample Pairs"]]
;;     (with-open [out-file (io/writer (str "org/pics/scale_assessment/" (name tile) "_data-table-scenario-" scenario ".csv"))]
;;       (csv/write-csv out-file
;;                      (cons ["wrf_tile" "wrf_x" "wrf_y" "landfire_x" "landfire_y" "ffwig_percentile"
;;                             "ws_20ft_mph" "wdir" "mparam" "eaf" "fire_size_ac" "flame_length_mean"
;;                             "flame_length_stddev" "fire_volume" "fire_shape"]
;;                            (mapv (fn [{:keys [ignition-site weather-sample wind-speed-20ft wind-from-direction
;;                                               equilibrium-moisture eaf fire-size flame-length-mean
;;                                               flame-length-stddev fire-volume fire-shape]}]
;;                                    (let [local-site     (mop/- ignition-site [66 66])
;;                                          wrf-percentile (- 100.0 (/ weather-sample 36.53))]
;;                                      [(wrf-cell 0)
;;                                       (wrf-cell 1)
;;                                       (wrf-cell 2)
;;                                       (local-site 0)
;;                                       (local-site 1)
;;                                       wrf-percentile
;;                                       wind-speed-20ft
;;                                       wind-from-direction
;;                                       equilibrium-moisture
;;                                       eaf
;;                                       fire-size
```

```
;;                                      flame-length-mean
;;                                      flame-length-stddev
;;                                      fire-volume
;;                                      fire-shape]))
;;                                  results-table))))
;;     (incanter/save
;;      (charts/scatter-plot (mapv :fire-size results-table)
;;                           (mapv :flame-length-mean results-table)
;;                           :group-by (mapv #(Math/floor (- 5.0 (* 2.0 (/ (:weather-sample %) 36.53)))) results-table)
;;                           :title (plot-titles (dec scenario))
;;                           :x-label "Fire Size (ac)"
;;                           :y-label "Flame Length Mean (ft)"
;;                           :series-label "Percentile 0=97.5, 1=98, 2=98.5, 3=99, 4=99.5, 5=100"
;;                           :legend true)
;;      (str "org/pics/scale_assessment/" (name tile) "_scatterplot-scenario-" scenario ".png"))
;;     (spit (str "org/pics/scale_assessment/" (name tile) "_burn-prob-matrix-scenario-" scenario ".clj")
;;           (mapv vec (t/rows burn-prob-matrix)))
;;     (save-matrix-as-png :color 4 -1.0 burn-prob-matrix
;;                         (str "org/pics/scale_assessment/" (name tile) "_burn-prob-scenario-" scenario ".png"))))

;; (defn read-csv-simple
;;   [filename]
;;   (with-open [in-file (io/reader filename)]
;;     (let [rows (csv/read-csv in-file)]
;;       {:header (first rows)
;;        :rows   (doall (rest rows))})))

;; (defn write-csv-simple
;;   [filename csv]
;;   (with-open [out-file (io/writer filename)]
;;     (csv/write-csv out-file (cons (:header csv) (:rows csv)))))

;; (defn merge-csv-files-simple
;;   [csv-directory infile-pattern outfile]
;;   (let [csv-files (->> (io/file csv-directory)
;;                        (file-seq)
;;                        (filter #(re-matches infile-pattern (.getName ^java.io.File %)))
;;                        (map #(read-csv-simple %)))]
;;     (write-csv-simple (io/file csv-directory outfile)
;;                       {:header (:header (first csv-files))
;;                        :rows   (mapcat :rows csv-files)})))

;; ;; (merge-csv-files-simple "/data/statewide_results" #".*.csv£" "all-fires-combined.csv")

;; (defn merge-csv-files
;;   [csv-directory infile-pattern scenario-pattern outfile]
;;   (let [csv-files (->> (io/file csv-directory)
;;                        (file-seq)
;;                        (filter #(re-matches infile-pattern (.getName %)))
;;                        (map #(re-matches scenario-pattern (.getName %)))
;;                        (mapv (fn [[filename scenario]] (assoc (read-csv-simple (io/file csv-directory filename)) :scenario scenari
;;     (write-csv-simple (io/file csv-directory outfile)
;;                       {:header (cons "scenario" (:header (first csv-files)))
;;                        :rows   (mapcat (fn [{:keys [scenario rows]}] (mapv #(cons scenario %) rows)) csv-files)})))

;; (defn interpolate-fire-records
;;   [csv-mins csv-maxs {:keys [percentile ws wd :as wrf-sample]}]
;;   (mapv (fn [csv-min csv-max]
;;           (let [min-ws      (csv-min 6)
;;                 max-ws      (csv-max 6)
;;                 min-fsz     (csv-min 8)
;;                 max-fsz     (csv-max 8)
;;                 min-flm     (csv-min 9)
;;                 max-flm     (csv-max 9)
;;                 min-fls     (csv-min 10)
;;                 max-fls     (csv-max 10)
;;                 max-percent (max 0.0 (min 1.0 (/ (- ws min-ws) (- max-ws min-ws))))
;;                 min-percent (- 1.0 max-percent)
```

```
;;                rand-op1    (rand-nth [+ -])
;;                rand-op2    (rand-nth [+ -])
;;                rand-op3    (rand-nth [+ -])]
;;             [(csv-min 0) ; wrf_tile
;;              (csv-min 1) ; wrf_x
;;              (csv-min 2) ; wrf_y
;;              (csv-min 3) ; landfire_x
;;              (csv-min 4) ; landfire_y
;;              percentile  ; ffwig_percentile
;;              ws ; ws_20ft_mph
;;              wd ; wdir
;;              (max 0.0
;;                   (rand-op1
;;                    (+ (* min-percent min-fsz)
;;                       (* max-percent max-fsz))
;;                    (rand (* 0.1 max-fsz)))) ; fire_size_ac
;;              (max 0.0
;;                   (rand-op2
;;                    (+ (* min-percent min-flm)
;;                       (* max-percent max-flm))
;;                    (rand (* 0.1 max-flm)))) ; flame_length_mean
;;              (max 0.0
;;                   (rand-op3
;;                    (+ (* min-percent min-fls)
;;                       (* max-percent max-fls))
;;                    (rand (* 0.1 max-fls))))])) ; flame_length_stddev
;;         csv-mins
;;         csv-maxs))

;; (defn bridge-csv-samples
;;   [csv-min-dir csv-max-dir output-dir]
;;   (doseq [scenario [6] tile [:tile205 :tile210 :tile281 :tile310 :tile564 :tile643]]
;;     (let [csv-min-file (io/file csv-min-dir (str (name tile) "_data-table-scenario-" scenario ".csv"))
;;           csv-max-file (io/file csv-max-dir (str (name tile) "_data-table-scenario-" scenario ".csv"))
;;           csv-mins     (if (.exists csv-min-file)
;;                          (with-open [csv-min-reader (io/reader csv-min-file)]
;;                            (mapv #(mapv read-string %) (rest (csv/read-csv csv-min-reader)))))
;;           csv-maxs     (if (.exists csv-max-file)
;;                          (with-open [csv-max-reader (io/reader csv-max-file)]
;;                            (mapv #(mapv read-string %) (rest (csv/read-csv csv-max-reader)))))]
;;       (with-open [out-file (io/writer (io/file output-dir (str (name tile) "_data-table-scenario-" scenario ".csv")))]
;;         (csv/write-csv out-file
;;                        (cons ["wrf_tile" "wrf_x" "wrf_y" "landfire_x" "landfire_y" "ffwig_percentile" "ws_20ft_mph"
;;                               "wdir" "fire_size_ac" "flame_length_mean" "flame_length_stddev"]
;;                              (concat csv-maxs
;;                                      (apply concat
;;                                             (for [i (range 1 91)]
;;                                               (let [wrf-sample  (get-in extreme-weather-readings [tile i])]
;;                                                   (interpolate-fire-records csv-mins csv-maxs wrf-sample))))
;;                                      csv-mins)))))))

;; (defn plot-csv-results
;;   [csv-directory tile scenario]
;;   (let [plot-titles ["Scenario 1 - Random (Ignition,WRF) Sample Pairs"
;;                      "Scenario 2 - Coarse Lattice, Few WRF Samples"
;;                      "Scenario 3 - Coarse Lattice, Many WRF Samples"
;;                      "Scenario 4 - Fine Lattice, Few WRF Samples"
;;                      "Scenario 5 - Fine Lattice, Many WRF Samples"
;;                      "Scenario 6 - All 30m Cells, All WRF Samples"
;;                      "Scenario 7 - Random (Ignition,WRF) Sample Pairs"]
;;         results-table (with-open [in-file (io/reader (io/file csv-directory (str (name tile) "_data-table-scenario-" scenario ".cs
;;                         (mapv (fn [record] {:percentile        (read-string (record 5))
;;                                             :fire-size         (read-string (record 8))
;;                                             :flame-length-mean (read-string (record 9))})
;;                               (rest (csv/read-csv in-file))))]
;;     (incanter/save
;;      (charts/scatter-plot (mapv :fire-size results-table)
;;                           (mapv :flame-length-mean results-table)
```

```
;;                          :group-by (mapv #(Math/floor (- 5.0 (* 2.0 (- 100.0 (:percentile %))))) results-table)
;;                          :title (plot-titles (dec scenario))
;;                          :x-label "Fire Size (ac)"
;;                          :y-label "Flame Length Mean (ft)"
;;                          :series-label "Percentile 0=97.5, 1=98, 2=98.5, 3=99, 4=99.5, 5=100"
;;                          :legend true)
;;      (io/file csv-directory (str (name tile) "_scatterplot-scenario-" scenario ".png")))))

;; (defn generate-cumulative-csv-results
;;   [csv-directory tile scenario]
;;   (let [results-table (with-open [in-file (io/reader (io/file csv-directory (str (name tile) "_data-table-scenario-" scenario ".cs
;;                         (mapv (fn [record] {:percentile        (read-string (record 5))
;;                                             :fire-size         (read-string (record 8))
;;                                             :flame-length-mean (read-string (record 9))
;;                                             :fire-volume       (read-string (record 11))
;;                                             :fire-shape        (read-string (record 12))})
;;                               (rest (csv/read-csv in-file))))]
;;     (with-open [out-file (io/writer (io/file csv-directory (format "%s_cumulative-stats-scenario-%d.csv" (name tile) scenario)))]
;;       (csv/write-csv out-file
;;                      (cons ["samples" "fire_size_mean" "fire_size_stddev" "fire_size_skewness"
;;                             "flame_length_mean" "flame_length_stddev" "flame_length_skewness"
;;                             "fire_volume_mean" "fire_volume_stddev" "fire_volume_skewness"
;;                             "fire_shape_mean" "fire_shape_stddev" "fire_shape_skewness"]
;;                            (for [bin-size (range 100 1001 100)]
;;                              (let [results-table (take bin-size results-table)
;;                                    fire-sizes    (filterv pos? (mapv :fire-size         results-table))
;;                                    flame-lengths (filterv pos? (mapv :flame-length-mean results-table))
;;                                    fire-volumes  (filterv pos? (mapv :fire-volume       results-table))
;;                                    fire-shapes   (filterv pos? (mapv :fire-shape        results-table))
;;                                    get-stats     (juxt stats/mean stats/sd stats/skewness)]
;;                                (cons bin-size (mapcat get-stats [fire-sizes flame-lengths fire-volumes fire-shapes])))))))
;;     (doseq [bin-size (range 100 1001 100)]
;;       (let [results-table (take bin-size results-table)]
;;         (incanter/save
;;          (charts/scatter-plot (mapv :fire-size results-table)
;;                               (mapv :flame-length-mean results-table)
;;                               :group-by (mapv #(Math/floor (- 5.0 (* 2.0 (- 100.0 (:percentile %))))) results-table)
;;                               :title (str "Scenario 7-" bin-size " Fire Size vs Flame Length")
;;                               :x-label "Fire Size (ac)"
;;                               :y-label "Flame Length Mean (ft)"
;;                               :series-label "Percentile 0=97.5, 1=98, 2=98.5, 3=99, 4=99.5, 5=100"
;;                               :legend true)
;;          (io/file csv-directory (format "%s_scatterplot-scenario-%d-%04d.png" (name tile) scenario bin-size)))
;;         (incanter/save
;;          (charts/histogram (filterv pos? (mapv :fire-size results-table))
;;                            :nbins 20
;;                            :density true
;;                            :title (str "Scenario 7-" bin-size " Fire Size")
;;                            :x-label "Fire Size (ac)")
;;          (io/file csv-directory (format "%s_fire-size-scenario-%d-%04d.png" (name tile) scenario bin-size)))
;;         (incanter/save
;;          (charts/histogram (filterv pos? (mapv :flame-length-mean results-table))
;;                            :nbins 20
;;                            :density true
;;                            :title (str "Scenario 7-" bin-size " Flame Length Mean")
;;                            :x-label "Flame Length Mean (ft)")
;;          (io/file csv-directory (format "%s_flame-length-scenario-%d-%04d.png" (name tile) scenario bin-size)))
;;         (incanter/save
;;          (charts/histogram (filterv pos? (mapv :fire-volume results-table))
;;                            :nbins 20
;;                            :density true
;;                            :title (str "Scenario 7-" bin-size " Fire Volume")
;;                            :x-label "Fire Volume (ac*ft)")
;;          (io/file csv-directory (format "%s_fire-volume-scenario-%d-%04d.png" (name tile) scenario bin-size)))
;;         (incanter/save
;;          (charts/histogram (filterv pos? (mapv :fire-shape results-table))
;;                            :nbins 20
;;                            :density true
```

```
;;                            :title (str "Scenario 7-" bin-size " Fire Shape")
;;                            :x-label "Fire Shape (ac/ft)")
;;          (io/file csv-directory (format "%s_fire-shape-scenario-%d-%04d.png" (name tile) scenario bin-size)))))))

;; ;; Example commands for the 36 (scenario,tile) pairs
;; (comment
;;   (time (def scenario1-tile205 (run-scenario-on-tile 1 :tile205)))
;;   (time (def scenario2-tile205 (run-scenario-on-tile 2 :tile205)))
;;   (time (def scenario3-tile205 (run-scenario-on-tile 3 :tile205)))
;;   (time (def scenario4-tile205 (run-scenario-on-tile 4 :tile205)))
;;   (time (def scenario5-tile205 (run-scenario-on-tile 5 :tile205)))
;;   (time (def scenario6-tile205 (run-scenario-on-tile 6 :tile205)))
;;   (time (def scenario7-tile205 (run-scenario-on-tile 7 :tile205)))

;;   (postprocess-simulation-results 1 :tile205 scenario1-tile205)
;;   (postprocess-simulation-results 2 :tile205 scenario2-tile205)
;;   (postprocess-simulation-results 3 :tile205 scenario3-tile205)
;;   (postprocess-simulation-results 4 :tile205 scenario4-tile205)
;;   (postprocess-simulation-results 5 :tile205 scenario5-tile205)
;;   (postprocess-simulation-results 6 :tile205 scenario6-tile205)
;;   (postprocess-simulation-results 7 :tile205 scenario7-tile205)

;;   (time (def scenario1-tile210 (run-scenario-on-tile 1 :tile210)))
;;   (time (def scenario2-tile210 (run-scenario-on-tile 2 :tile210)))
;;   (time (def scenario3-tile210 (run-scenario-on-tile 3 :tile210)))
;;   (time (def scenario4-tile210 (run-scenario-on-tile 4 :tile210)))
;;   (time (def scenario5-tile210 (run-scenario-on-tile 5 :tile210)))
;;   (time (def scenario6-tile210 (run-scenario-on-tile 6 :tile210)))
;;   (time (def scenario7-tile210 (run-scenario-on-tile 7 :tile210)))

;;   (postprocess-simulation-results 1 :tile210 scenario1-tile210)
;;   (postprocess-simulation-results 2 :tile210 scenario2-tile210)
;;   (postprocess-simulation-results 3 :tile210 scenario3-tile210)
;;   (postprocess-simulation-results 4 :tile210 scenario4-tile210)
;;   (postprocess-simulation-results 5 :tile210 scenario5-tile210)
;;   (postprocess-simulation-results 6 :tile210 scenario6-tile210)
;;   (postprocess-simulation-results 7 :tile210 scenario7-tile210)

;;   (time (def scenario1-tile281 (run-scenario-on-tile 1 :tile281)))
;;   (time (def scenario2-tile281 (run-scenario-on-tile 2 :tile281)))
;;   (time (def scenario3-tile281 (run-scenario-on-tile 3 :tile281)))
;;   (time (def scenario4-tile281 (run-scenario-on-tile 4 :tile281)))
;;   (time (def scenario5-tile281 (run-scenario-on-tile 5 :tile281)))
;;   (time (def scenario6-tile281 (run-scenario-on-tile 6 :tile281)))
;;   (time (def scenario7-tile281 (run-scenario-on-tile 7 :tile281)))

;;   (postprocess-simulation-results 1 :tile281 scenario1-tile281)
;;   (postprocess-simulation-results 2 :tile281 scenario2-tile281)
;;   (postprocess-simulation-results 3 :tile281 scenario3-tile281)
;;   (postprocess-simulation-results 4 :tile281 scenario4-tile281)
;;   (postprocess-simulation-results 5 :tile281 scenario5-tile281)
;;   (postprocess-simulation-results 6 :tile281 scenario6-tile281)
;;   (postprocess-simulation-results 7 :tile281 scenario7-tile281)

;;   (time (def scenario1-tile310 (run-scenario-on-tile 1 :tile310)))
;;   (time (def scenario2-tile310 (run-scenario-on-tile 2 :tile310)))
;;   (time (def scenario3-tile310 (run-scenario-on-tile 3 :tile310)))
;;   (time (def scenario4-tile310 (run-scenario-on-tile 4 :tile310)))
;;   (time (def scenario5-tile310 (run-scenario-on-tile 5 :tile310)))
;;   (time (def scenario6-tile310 (run-scenario-on-tile 6 :tile310)))
;;   (time (def scenario7-tile310 (run-scenario-on-tile 7 :tile310)))

;;   (postprocess-simulation-results 1 :tile310 scenario1-tile310)
;;   (postprocess-simulation-results 2 :tile310 scenario2-tile310)
;;   (postprocess-simulation-results 3 :tile310 scenario3-tile310)
;;   (postprocess-simulation-results 4 :tile310 scenario4-tile310)
;;   (postprocess-simulation-results 5 :tile310 scenario5-tile310)
;;   (postprocess-simulation-results 6 :tile310 scenario6-tile310)
```

```
;;   (postprocess-simulation-results 7 :tile310 scenario7-tile310)

;;   (time (def scenario1-tile564 (run-scenario-on-tile 1 :tile564)))
;;   (time (def scenario2-tile564 (run-scenario-on-tile 2 :tile564)))
;;   (time (def scenario3-tile564 (run-scenario-on-tile 3 :tile564)))
;;   (time (def scenario4-tile564 (run-scenario-on-tile 4 :tile564)))
;;   (time (def scenario5-tile564 (run-scenario-on-tile 5 :tile564)))
;;   (time (def scenario6-tile564 (run-scenario-on-tile 6 :tile564)))
;;   (time (def scenario7-tile564 (run-scenario-on-tile 7 :tile564)))

;;   (postprocess-simulation-results 1 :tile564 scenario1-tile564)
;;   (postprocess-simulation-results 2 :tile564 scenario2-tile564)
;;   (postprocess-simulation-results 3 :tile564 scenario3-tile564)
;;   (postprocess-simulation-results 4 :tile564 scenario4-tile564)
;;   (postprocess-simulation-results 5 :tile564 scenario5-tile564)
;;   (postprocess-simulation-results 6 :tile564 scenario6-tile564)
;;   (postprocess-simulation-results 7 :tile564 scenario7-tile564)

;;   (time (def scenario1-tile643 (run-scenario-on-tile 1 :tile643)))
;;   (time (def scenario2-tile643 (run-scenario-on-tile 2 :tile643)))
;;   (time (def scenario3-tile643 (run-scenario-on-tile 3 :tile643)))
;;   (time (def scenario4-tile643 (run-scenario-on-tile 4 :tile643)))
;;   (time (def scenario5-tile643 (run-scenario-on-tile 5 :tile643)))
;;   (time (def scenario6-tile643 (run-scenario-on-tile 6 :tile643)))
;;   (time (def scenario7-tile643 (run-scenario-on-tile 7 :tile643)))

;;   (postprocess-simulation-results 1 :tile643 scenario1-tile643)
;;   (postprocess-simulation-results 2 :tile643 scenario2-tile643)
;;   (postprocess-simulation-results 3 :tile643 scenario3-tile643)
;;   (postprocess-simulation-results 4 :tile643 scenario4-tile643)
;;   (postprocess-simulation-results 5 :tile643 scenario5-tile643)
;;   (postprocess-simulation-results 6 :tile643 scenario6-tile643)
;;   (postprocess-simulation-results 7 :tile643 scenario7-tile643)

;;   (do
;;     (generate-cumulative-csv-results "org/pics/scale_assessment" :tile205 7)
;;     (generate-cumulative-csv-results "org/pics/scale_assessment" :tile210 7)
;;     (generate-cumulative-csv-results "org/pics/scale_assessment" :tile281 7)
;;     (generate-cumulative-csv-results "org/pics/scale_assessment" :tile310 7)
;;     (generate-cumulative-csv-results "org/pics/scale_assessment" :tile564 7)
;;     (generate-cumulative-csv-results "org/pics/scale_assessment" :tile643 7))

;;   (merge-csv-files "org/pics/scale_assessment/sequential_runs" #"tile.+data-table-scenario.+\.csv" #"tile.+-(\d+)\.csv" "data-tabl
```

10.0.2 gridfire-unmerged.fire-spread-test

```
;; (ns gridfire-unmerged.fire-spread-test
;;   (:require [clojure.java.io :as io]
;;             [clojure.core.matrix :as m]
;;             [clojure.core.matrix.operators :as mop]
;;             [clojure.core.reducers :as r]
;;             [clojure.java.jdbc :as jdbc]
;;             [clojure.data.csv :as csv]
;;             [clojure.set :as set]
;;             [matrix-viz.core :refer [save-matrix-as-png]]
;;             [gridfire.common :refer [burnable-fuel-model? burnable? burnable-neighbors?]]
;;             [gridfire.fuel-models :refer [fuel-models build-fuel-model moisturize]]
;;             [gridfire.surface-fire :refer [rothermel-surface-fire-spread-no-wind-no-slope
;;                                            rothermel-surface-fire-spread-max
;;                                            rothermel-surface-fire-spread-any
;;                                            anderson-flame-depth
;;                                            byram-fire-line-intensity
;;                                            byram-flame-length
;;                                            wind-adjustment-factor
;;                                            degrees-to-radians grass-fuel-model?]]
;;             [gridfire.fire-spread :refer [run-fire-spread rothermel-fast-wrapper]]
```

```
;;             [gridfire.monte-carlo :refer [cells-to-acres]]
;;             [gridfire.postgis-bridge :refer [postgis-raster-to-matrix]]
;;             [tech.v3.tensor         :as t]
;;             [tech.v3.datatype       :as d]
;;             [tech.v3.datatype.functional :as dfn])
;;   (:import org.postgresql.jdbc.PgArray))

;; (defn combine-into-map
;;   ([] {})
;;   ([r1] r1)
;;   ([r1 r2] (merge r1 r2)))

;; (def anderson-sample-values
;;   [[ 78.0  4.0]
;;    [ 35.0  6.0]
;;    [104.0 12.0]
;;    [ 75.0 19.0]
;;    [ 18.0  4.0]
;;    [ 32.0  6.0]
;;    [ 20.0  5.0]
;;    [  1.6  1.0]
;;    [  7.5  2.6]
;;    [  7.9  4.8]
;;    [  6.0  3.5]
;;    [ 13.0  8.0]
;;    [ 13.5 10.5]])

;; ;; Compare with tables in Anderson 1982
;; ;; Largest error is: +6.5 ft/min, -9.6 in
;; (for [fuel-model-number (range 1 14)]
;;   (let [[anderson-spread-rate anderson-flame-length] (anderson-sample-values (dec fuel-model-number))
;;         sample-fuel-moisture                         {:dead {:1hr 0.08 :10hr 0.08 :100hr 0.08} :live {:herbaceous 1.0 :woody 1.0}}
;;         spread-info-min                              (-> (build-fuel-model fuel-model-number)
;;                                                          (moisturize sample-fuel-moisture)
;;                                                          (rothermel-surface-fire-spread-no-wind-no-slope))
;;         spread-info-max                              (rothermel-surface-fire-spread-max spread-info-min 440.0 0.0 0.0 0.0 1.0)
;;         spread-rate                                  (:max-spread-rate spread-info-max)
;;         flame-depth                                  (anderson-flame-depth spread-rate (:residence-time spread-info-min))
;;         fire-line-intensity                          (byram-fire-line-intensity (:reaction-intensity spread-info-min)
;;                                                                                 flame-depth)
;;         flame-length                                 (byram-flame-length fire-line-intensity)]
;;     (format "%3d %5.1f %5.1f %4.1f(%4.1f) %4.1f %4.1f %4.1f(%4.1f)" fuel-model-number
;;             (/ spread-rate 1.1) anderson-spread-rate
;;             (- (/ spread-rate 1.1) anderson-spread-rate)
;;             (* 1.1 (- (/ spread-rate 1.1) anderson-spread-rate))
;;             flame-length anderson-flame-length
;;             (- flame-length anderson-flame-length)
;;             (* 12.0 (- flame-length anderson-flame-length)))))

;; (def scott-burgan-sample-values
;;   {101 [ :moderate       :low]
;;    102 [     :high  :moderate]
;;    103 [     :high  :moderate]
;;    104 [:very-high      :high]
;;    105 [:very-high      :high]
;;    106 [:very-high :very-high]
;;    107 [:very-high :very-high]
;;    108 [:very-high :very-high]
;;    109 [  :extreme   :extreme]
;;    121 [ :moderate       :low]
;;    122 [     :high  :moderate]
;;    123 [     :high  :moderate]
;;    124 [     :high :very-high]
;;    141 [ :very-low  :very-low]
;;    142 [      :low       :low]
;;    143 [      :low       :low]
;;    144 [     :high  :moderate]
;;    145 [:very-high :very-high]
```

```
;;    146 [     :high      :high]
;;    147 [     :high :very-high]
;;    148 [     :high      :high]
;;    149 [     :high :very-high]
;;    161 [      :low       :low]
;;    162 [ :moderate       :low]
;;    163 [     :high  :moderate]
;;    164 [ :moderate  :moderate]
;;    165 [ :moderate  :moderate]
;;    181 [ :very-low  :very-low]
;;    182 [ :very-low  :very-low]
;;    183 [ :very-low       :low]
;;    184 [      :low       :low]
;;    185 [      :low       :low]
;;    186 [ :moderate       :low]
;;    187 [      :low       :low]
;;    188 [ :moderate       :low]
;;    189 [ :moderate  :moderate]
;;    201 [ :moderate       :low]
;;    202 [ :moderate  :moderate]
;;    203 [     :high      :high]
;;    204 [:very-high :very-high]})

;; (defn qualify-spread-rate
;;   [spread-rate]
;;   (condp >= spread-rate
;;     2   :very-low
;;     5   :low
;;     20  :moderate
;;     50  :high
;;     150 :very-high
;;     300 :extreme))

;; (defn qualify-flame-length
;;   [flame-length]
;;   (condp >= flame-length
;;     1   :very-low
;;     4   :low
;;     8   :moderate
;;     12  :high
;;     25  :very-high
;;     100 :extreme))

;; ;; Compare with tables in Scott & Burgan 2005
;; (for [fuel-model-number (drop 18 (sort (keys fuel-models)))]
;;   (let [[sb-spread-rate sb-flame-length] (scott-burgan-sample-values fuel-model-number)
;;         sample-fuel-moisture             {:dead {:1hr 0.06 :10hr 0.07 :100hr 0.08} :live {:herbaceous 0.60 :woody 0.90}}
;;         spread-info-min                  (-> (build-fuel-model fuel-model-number)
;;                                              (moisturize sample-fuel-moisture)
;;                                              (rothermel-surface-fire-spread-no-wind-no-slope))
;;         spread-info-max                  (rothermel-surface-fire-spread-max spread-info-min 440.0 0.0 0.0 0.0 1.0)
;;         spread-rate                      (:max-spread-rate spread-info-max)
;;         flame-depth                      (anderson-flame-depth spread-rate (:residence-time spread-info-min))
;;         fire-line-intensity              (byram-fire-line-intensity (:reaction-intensity spread-info-min)
;;                                                                     flame-depth)
;;         flame-length                     (byram-flame-length fire-line-intensity)]
;;     (format "| %3d | %10s %2s %10s | %10s %2s %10s |" fuel-model-number
;;             (qualify-spread-rate (/ spread-rate 1.1))
;;             (if (= (qualify-spread-rate (/ spread-rate 1.1)) sb-spread-rate) "==" "!=")
;;             sb-spread-rate
;;             (qualify-flame-length flame-length)
;;             (if (= (qualify-flame-length flame-length) sb-flame-length) "==" "!=")
;;             sb-flame-length)))

;; (def low-fuel-models #{102.0 104.0 105.0 109.0 122.0 123.0 124.0 163.0})
;; (def error-fuel-models #{102.0 121.0 122.0 142.0 145.0 147.0 165.0 189.0})

;; ;; Compare with WAF paper Figure 4 (page 6)
```

```
;; (for [waf [0.2 0.4 0.7] wind-speed-20ft [4 8 12 16 20]]
;;   (let [midflame-wind-speed  (* waf 88 wind-speed-20ft)
;;         sample-fuel-moisture {:dead {:1hr 0.05 :10hr 0.05 :100hr 0.05} :live {:herbaceous 0.75 :woody 0.75}}
;;         spread-info-min      (-> (build-fuel-model 2)
;;                                  (moisturize sample-fuel-moisture)
;;                                  (rothermel-surface-fire-spread-no-wind-no-slope))
;;         spread-info-max      (rothermel-surface-fire-spread-max spread-info-min midflame-wind-speed 0.0 0.0 0.0 1.0)
;;         spread-rate          (:max-spread-rate spread-info-max)
;;         flame-depth          (anderson-flame-depth spread-rate (:residence-time spread-info-min))
;;         fire-line-intensity  (byram-fire-line-intensity (:reaction-intensity spread-info-min)
;;                                                         flame-depth)
;;         flame-length         (byram-flame-length fire-line-intensity)]
;;     (format "%2d %3.1f %4.1f %5.1f %4.1f"
;;             wind-speed-20ft waf (/ midflame-wind-speed 88) (/ spread-rate 1.1) flame-length)))

;; (def behaveplus5-surface-fire-values
;;   {1   [1 108.8 5.0]
;;    2   [2 50.6 7.8]
;;    3   [3 144.5 15.6]
;;    4   [4 109.2 24.5]
;;    5   [5 36.1 7.8]
;;    6   [6 41.7 7.1]
;;    7   [7 38.0 7.1]
;;    8   [8 2.5 1.3]
;;    9   [9 10.8 3.4]
;;    10  [10 12.0 6.4]
;;    11  [11 7.3 3.9]
;;    12  [12 16.4 9.4]
;;    13  [13 19.9 12.4]
;;    91  [91 0.0 0.0]
;;    92  [92 0.0 0.0]
;;    93  [93 0.0 0.0]
;;    98  [98 0.0 0.0]
;;    99  [99 0.0 0.0]
;;    101 [101 21.7 2.3]
;;    102 [102 54.8 5.8]
;;    103 [103 77.3 8.8]
;;    104 [104 110.3 10.8]
;;    105 [105 92.7 13.9]
;;    106 [106 121.1 17.4]
;;    107 [107 161.5 22.8]
;;    108 [108 169.6 29.9]
;;    109 [109 274.5 41.2]
;;    121 [121 23.0 4.2]
;;    122 [122 32.3 6.2]
;;    123 [123 50.1 10.1]
;;    124 [124 42.3 18.4]
;;    141 [141 5.4 1.5]
;;    142 [142 9.8 5.7]
;;    143 [143 5.4 2.9]
;;    144 [144 42.0 8.8]
;;    145 [145 77.3 16.4]
;;    146 [146 31.6 11.0]
;;    147 [147 51.0 15.6]
;;    148 [148 29.8 12.1]
;;    149 [149 55.2 20.0]
;;    161 [161 4.2 2.2]
;;    162 [162 15.5 4.2]
;;    163 [163 37.6 9.0]
;;    164 [164 16.0 6.7]
;;    165 [165 12.0 8.7]
;;    181 [181 1.1 0.7]
;;    182 [182 1.7 1.0]
;;    183 [183 2.1 1.2]
;;    184 [184 3.1 1.6]
;;    185 [185 5.6 2.5]
;;    186 [186 7.4 3.1]
;;    187 [187 3.6 2.3]
```

```
;;    188 [188 7.4 3.9]
;;    189 [189 11.1 5.6]
;;    201 [201 7.8 3.6]
;;    202 [202 18.9 7.0]
;;    203 [203 34.1 10.5]
;;    204 [204 65.6 14.7]})

;; (defn surface-fire-check [fuel-model-number fuel-moisture midflame-wind-speed slope]
;;   (let [spread-info-min     (second (rothermel-fast-wrapper fuel-model-number fuel-moisture))
;;         spread-info-max     (rothermel-surface-fire-spread-max
;;                              spread-info-min midflame-wind-speed 0.0 slope 0.0 1.0)
;;         flame-depth         (anderson-flame-depth (:max-spread-rate spread-info-max)
;;                                                   (:residence-time spread-info-min))
;;         fire-line-intensity (byram-fire-line-intensity (:reaction-intensity spread-info-min) flame-depth)
;;         flame-length        (byram-flame-length fire-line-intensity)
;;         trim-value          #(/ (Math/round (* % 10.0)) 10.0)]
;;     [(int fuel-model-number) (trim-value (/ (:max-spread-rate spread-info-max) 1.1)) (trim-value flame-length)]))

;; (for [fm-number (sort (keys fuel-models))]
;;   (let [gridfire-results   (surface-fire-check fm-number
;;                                                {:dead {:1hr 0.04 :10hr 0.04 :100hr 0.06}
;;                                                 :live {:herbaceous 0.40 :woody 0.70}}
;;                                                440.0
;;                                                0.1)
;;         behaveplus-results (behaveplus5-surface-fire-values (int fm-number))]
;;     (if (= gridfire-results behaveplus-results)
;;       [fm-number :matched]
;;       [fm-number :no-match gridfire-results behaveplus-results])))

;; (def test-duration 60.0) ; mins
;; (def test-cell-size 98.425) ; 30m
;; (def test-num-rows 200) ; 6km total
;; (def test-num-cols 200) ; 6km total
;; (def test-fuel-moisture {:dead {:1hr 0.04 :10hr 0.04 :100hr 0.06} :live {:herbaceous 0.40 :woody 0.70}})
;; (def test-wind-speed-20ft 25.0) ; mph
;; (def test-wind-direction 45.0) ; from NE
;; (def test-ignition-site [100 100])

;; (defn view-hfire-results
;;   [elevation-matrix slope-matrix fuel-model-matrix ignition-site ellipse-adjustment-factor outfile-base]
;;   (let [landfire-layers      {:elevation          elevation-matrix
;;                               :slope              slope-matrix
;;                               :aspect             (doto (t/new-tensor test-num-rows test-num-cols) (mop/+= 270.0)) ; downhill = we
;;                               :fuel-model         fuel-model-matrix
;;                               :canopy-height      (t/new-tensor test-num-rows test-num-cols)
;;                               :canopy-base-height (t/new-tensor test-num-rows test-num-cols)
;;                               :canopy-cover       (t/new-tensor test-num-rows test-num-cols)
;;                               :crown-bulk-density (t/new-tensor test-num-rows test-num-cols)}
;;         foliar-moisture      0.9
;;         hfire-results        (run-fire-spread test-duration test-cell-size landfire-layers
;;                                               test-wind-speed-20ft test-wind-direction test-fuel-moisture
;;                                               foliar-moisture ellipse-adjustment-factor ignition-site)
;;         fire-spread-outfile  (str "org/pics/" outfile-base "_hfire-fire-spread.png")
;;         flame-length-outfile (str "org/pics/" outfile-base "_hfire-flame-length.png")
;;         [i j]                ignition-site]
;;     (save-matrix-as-png :color 4 -1.0 (doto (:fire-spread-matrix  hfire-results) (t/mset! i j 2.0)) fire-spread-outfile)
;;     (save-matrix-as-png :color 4 -1.0 (doto (:flame-length-matrix hfire-results) (t/mset! i j 2.0)) flame-length-outfile)))

;; (defn elevation-west-aspect
;;   [elevation-change]
;;   (let [elev (t/new-tensor test-num-rows test-num-cols)]
;;     (doseq [i (range test-num-rows)
;;             j (range test-num-cols)]
;;       (t/mset! elev i j (* j elevation-change)))
;;     elev))

;; (defn make-slope-layer
;;   [elevation-change]
```

```
;;   (doto (t/new-tensor test-num-rows test-num-cols) (mop/+= (/ elevation-change test-cell-size))))

;; (defn make-fuel-model
;;   [fuel-model-number]
;;   (doto (t/new-tensor test-num-rows test-num-cols) (mop/+= fuel-model-number)))

;; (let [outfiles {[ 1   0] "grass_fire_slope_W000%"
;;                 [ 1  30] "grass_fire_slope_W030%"
;;                 [ 1  60] "grass_fire_slope_W060%"
;;                 [ 1 100] "grass_fire_slope_W100%"
;;                 [ 4   0] "chaparral_fire_slope_W000%"
;;                 [ 4  30] "chaparral_fire_slope_W030%"
;;                 [ 4  60] "chaparral_fire_slope_W060%"
;;                 [ 4 100] "chaparral_fire_slope_W100%"
;;                 [ 8   0] "timber_litter_fire_slope_W000%"
;;                 [ 8  30] "timber_litter_fire_slope_W030%"
;;                 [ 8  60] "timber_litter_fire_slope_W060%"
;;                 [ 8 100] "timber_litter_fire_slope_W100%"
;;                 [11   0] "logging_slash_fire_slope_W000%"
;;                 [11  30] "logging_slash_fire_slope_W030%"
;;                 [11  60] "logging_slash_fire_slope_W060%"
;;                 [11 100] "logging_slash_fire_slope_W100%"}]
;;   (doseq [fuel-model [1 4 8 11] elevation-gain [0 30 60 100]]
;;     (view-hfire-results (elevation-west-aspect elevation-gain)
;;                         (make-slope-layer elevation-gain)
;;                         (make-fuel-model fuel-model)
;;                         test-ignition-site
;;                         1.0
;;                         (str "fire_spread/" (outfiles [fuel-model elevation-gain])))))

;; (def db-spec {:classname   "org.postgresql.Driver"
;;               :subprotocol "postgresql"
;;               :subname     "//localhost:5432/calfire"
;;               :user        "gjohnson"})

;; ;; Create images of LANDFIRE inputs
;; (def validation-layers
;;   {:tile205 {:ignition             [46 154]
;;              :elevation            (postgis-raster-to-matrix db-spec "validation.elevation_tile205"     nil nil)
;;              :slope                (postgis-raster-to-matrix db-spec "validation.slope_tile205"         nil nil)
;;              :aspect               (postgis-raster-to-matrix db-spec "validation.aspect_tile205"        nil nil)
;;              :fuel-model           (postgis-raster-to-matrix db-spec "validation.fuel_model_tile205"    nil nil)
;;              :canopy-height        (postgis-raster-to-matrix db-spec "validation.canopy_height_tile205" nil nil)
;;              :canopy-cover         (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile205"  nil nil)
;;              :flammap-fire-spread  (postgis-raster-to-matrix db-spec "validation.fire_presence_tile205" nil nil)
;;              :flammap-flame-length (postgis-raster-to-matrix db-spec "validation.flame_length_tile205"  nil nil)}
;;    :tile210 {:ignition             [111 81]
;;              :elevation            (postgis-raster-to-matrix db-spec "validation.elevation_tile210"     nil nil)
;;              :slope                (postgis-raster-to-matrix db-spec "validation.slope_tile210"         nil nil)
;;              :aspect               (postgis-raster-to-matrix db-spec "validation.aspect_tile210"        nil nil)
;;              :fuel-model           (postgis-raster-to-matrix db-spec "validation.fuel_model_tile210"    nil nil)
;;              :canopy-height        (postgis-raster-to-matrix db-spec "validation.canopy_height_tile210" nil nil)
;;              :canopy-cover         (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile210"  nil nil)
;;              :flammap-fire-spread  (postgis-raster-to-matrix db-spec "validation.fire_presence_tile210" nil nil)
;;              :flammap-flame-length (postgis-raster-to-matrix db-spec "validation.flame_length_tile210"  nil nil)}
;;    :tile281 {:ignition             [79 124]
;;              :elevation            (postgis-raster-to-matrix db-spec "validation.elevation_tile281"     nil nil)
;;              :slope                (postgis-raster-to-matrix db-spec "validation.slope_tile281"         nil nil)
;;              :aspect               (postgis-raster-to-matrix db-spec "validation.aspect_tile281"        nil nil)
;;              :fuel-model           (postgis-raster-to-matrix db-spec "validation.fuel_model_tile281"    nil nil)
;;              :canopy-height        (postgis-raster-to-matrix db-spec "validation.canopy_height_tile281" nil nil)
;;              :canopy-cover         (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile281"  nil nil)
;;              :flammap-fire-spread  (postgis-raster-to-matrix db-spec "validation.fire_presence_tile281" nil nil)
;;              :flammap-flame-length (postgis-raster-to-matrix db-spec "validation.flame_length_tile281"  nil nil)}
;;    :tile310 {:ignition             [86 113]
;;              :elevation            (postgis-raster-to-matrix db-spec "validation.elevation_tile310"     nil nil)
;;              :slope                (postgis-raster-to-matrix db-spec "validation.slope_tile310"         nil nil)
;;              :aspect               (postgis-raster-to-matrix db-spec "validation.aspect_tile310"        nil nil)
```

```
;;              :fuel-model           (postgis-raster-to-matrix db-spec "validation.fuel_model_tile310"    nil nil)
;;              :canopy-height        (postgis-raster-to-matrix db-spec "validation.canopy_height_tile310" nil nil)
;;              :canopy-cover         (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile310"  nil nil)
;;              :flammap-fire-spread  (postgis-raster-to-matrix db-spec "validation.fire_presence_tile310" nil nil)
;;              :flammap-flame-length (postgis-raster-to-matrix db-spec "validation.flame_length_tile310"  nil nil)}
;;    :tile564 {:ignition             [109 92]
;;              :elevation            (postgis-raster-to-matrix db-spec "validation.elevation_tile564"     nil nil)
;;              :slope                (postgis-raster-to-matrix db-spec "validation.slope_tile564"         nil nil)
;;              :aspect               (postgis-raster-to-matrix db-spec "validation.aspect_tile564"        nil nil)
;;              :fuel-model           (postgis-raster-to-matrix db-spec "validation.fuel_model_tile564"    nil nil)
;;              :canopy-height        (postgis-raster-to-matrix db-spec "validation.canopy_height_tile564" nil nil)
;;              :canopy-cover         (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile564"  nil nil)
;;              :flammap-fire-spread  (postgis-raster-to-matrix db-spec "validation.fire_presence_tile564" nil nil)
;;              :flammap-flame-length (postgis-raster-to-matrix db-spec "validation.flame_length_tile564"  nil nil)}
;;    :tile643 {:ignition             [86 116]
;;              :elevation            (postgis-raster-to-matrix db-spec "validation.elevation_tile643"     nil nil)
;;              :slope                (postgis-raster-to-matrix db-spec "validation.slope_tile643"         nil nil)
;;              :aspect               (postgis-raster-to-matrix db-spec "validation.aspect_tile643"        nil nil)
;;              :fuel-model           (postgis-raster-to-matrix db-spec "validation.fuel_model_tile643"    nil nil)
;;              :canopy-height        (postgis-raster-to-matrix db-spec "validation.canopy_height_tile643" nil nil)
;;              :canopy-cover         (postgis-raster-to-matrix db-spec "validation.canopy_cover_tile643"  nil nil)
;;              :flammap-fire-spread  (postgis-raster-to-matrix db-spec "validation.fire_presence_tile643" nil nil)
;;              :flammap-flame-length (postgis-raster-to-matrix db-spec "validation.flame_length_tile643"  nil nil)}})

;; (def validation-num-rows (-> (t/tensor->dimensions (-> validation-layers :tile205 :elevation)) :shape first))
;; (def validation-num-cols (-> (t/tensor->dimensions (-> validation-layers :tile205 :elevation)) :shape second))

;; (doseq [tile  [:tile205 :tile210 :tile281 :tile310 :tile564 :tile643]
;;         layer [:elevation :slope :aspect :fuel-model :canopy-height :canopy-cover]]
;;   (let [[i j]  (-> validation-layers tile :ignition)
;;         matrix (-> validation-layers tile layer)]
;;     (save-matrix-as-png :color 4 -1.0 (doto (d/clone matrix) (t/mset! i j -1.0))
;;                         (str "org/pics/validation/" (name tile) "_" (name layer) ".png"))))

;; (def validation-outputs
;;   (into {}
;;         (for [tile [:tile205 :tile210 :tile281 :tile310 :tile564 :tile643]]
;;           (let [[i j]                    (-> validation-layers tile :ignition)
;;                 elev                     (d/clone (d/emap #(* % 3.28) nil (-> validation-layers tile :elevation)))
;;                 slp                      (d/clone (d/emap #(Math/tan (degrees-to-radians %)) nil (-> validation-layers tile :slop
;;                 asp                      (-> validation-layers tile :aspect)
;;                 fm                       (-> validation-layers tile :fuel-model)
;;                 ch                       (d/clone (d/emap #(* % 0.328) nil (-> validation-layers tile :canopy-height)))
;;                 cc                       (-> validation-layers tile :canopy-cover)
;;                 ffs                      (-> validation-layers tile :flammap-fire-spread)
;;                 ffl                      (-> validation-layers tile :flammap-flame-length)
;;                 cbd                      (t/new-tensor test-num-rows test-num-cols)
;;                 cbh                      (t/new-tensor test-num-rows test-num-cols)
;;                 landfire-layers          {:elevation          elev
;;                                           :slope              slp
;;                                           :aspect             asp
;;                                           :fuel-model         fm
;;                                           :canopy-height      ch
;;                                           :canopy-base-height cbh
;;                                           :canopy-cover       cc
;;                                           :crown-bulk-density cbd}
;;                 foliar-moisture          0.9
;;                 ellipse-adjustment-factor 1.0
;;                 hfire-output             (run-fire-spread test-duration test-cell-size landfire-layers
;;                                                           test-wind-speed-20ft test-wind-direction test-fuel-moisture
;;                                                           foliar-moisture ellipse-adjustment-factor [i j])
;;                 hfs                      (-> hfire-output :fire-spread-matrix)
;;                 hfl                      (-> hfire-output :flame-length-matrix)
;;                 hfl-global               (d/clone
;;                                           (d/emap (fn [fm-number slope aspect canopy-height canopy-cover]
;;                                                     (if-not (burnable-fuel-model? fm-number)
;;                                                       0.0
;;                                                       (if (fuel-models (int fm-number))
```

```
;;                                                  (let [[fuel-model spread-info-min] (rothermel-fast-wrapper fm-number tes
;;                                                        waf                          (wind-adjustment-factor (:delta fuel-m
;;                                                        midflame-wind-speed          (* test-wind-speed-20ft 88.0 waf)
;;                                                        spread-info-max              (rothermel-surface-fire-spread-max spr
;;                                                                                                                      mid
;;                                                                                                                      tes
;;                                                                                                                      slo
;;                                                                                                                      asp
;;                                                                                                                      1.0
;;                                                        flame-depth                  (anderson-flame-depth (:max-spread-ra
;;                                                                                                           (:residence-time
;;                                                        fire-line-intensity          (byram-fire-line-intensity (:reaction-
;;                                                        flame-length                 (byram-flame-length fire-line-intensi
;;                                                    flame-length)
;;                                                  nil
;;                                                  -1.0)))
;;                                              fm slp asp ch cc))
;;                ffl-clipped              (d/clone (d/emap #(if (pos? %1) %2 0.0) nil hfs ffl))
;;                fl-diff                  (d/clone (d/emap - nil ffl hfl-global))
;;                mfl                      (max (dfn/reduce-max ffl) (dfn/reduce-max hfl) (dfn/reduce-max hfl-global))
;;                low-fm                   (d/clone (d/emap #(cond (= -1.0 %)  -1.0 (low-fuel-models %) 1.0 :else 0.0) nil fm))
;;                error-fm                 (d/clone (d/emap #(cond (= -1.0 %2) -1.0 (> (Math/abs ^double %1) 5.0) %2 :else 0.0) nil
;;             (save-matrix-as-png :color 4 -1.0 (doto (d/clone hfs) (t/mset! i j -1.0))
;;                                 (str "org/pics/validation/" (name tile) "_hfire-fire-spread.png"))
;;             (save-matrix-as-png :color 4 -1.0 (doto (d/clone ffs) (t/mset! i j -1.0))
;;                                 (str "org/pics/validation/" (name tile) "_flammap-fire-spread.png"))
;;             (save-matrix-as-png :color 4 -1.0 (doto (d/clone hfl) (t/mset! i j mfl))
;;                                 (str "org/pics/validation/" (name tile) "_hfire-flame-length.png"))
;;             (save-matrix-as-png :color 4 -1.0 (doto (d/clone ffl) (t/mset! i j mfl))
;;                                 (str "org/pics/validation/" (name tile) "_flammap-flame-length.png"))
;;             (save-matrix-as-png :color 4 -1.0 (doto (d/clone hfl-global) (t/mset! i j mfl))
;;                                 (str "org/pics/validation/" (name tile) "_hfire-flame-length-global.png"))
;;             (save-matrix-as-png :color 4 -1.0 (doto (d/clone ffl-clipped) (t/mset! i j mfl))
;;                                 (str "org/pics/validation/" (name tile) "_flammap-flame-length-clipped.png"))
;;             (save-matrix-as-png :color 4 -1.0 fl-diff
;;                                 (str "org/pics/validation/" (name tile) "_flame-length-difference.png"))
;;             (save-matrix-as-png :color 4 -1.0 low-fm
;;                                 (str "org/pics/validation/" (name tile) "_low-fuel-models.png"))
;;             (save-matrix-as-png :color 4 -1.0 error-fm
;;                                 (str "org/pics/validation/" (name tile) "_error-fuel-models.png"))
;;             [tile {:hfire-fire-spread            hfs
;;                    :hfire-flame-length           hfl
;;                    :hfire-flame-length-global    hfl-global
;;                    :flammap-flame-length-clipped ffl-clipped
;;                    :flame-length-difference      fl-diff
;;                    :error-fuel-models            error-fm}])))

;; (doseq [tile [:tile205 :tile210 :tile281 :tile310 :tile564 :tile643]]
;;   (let [fm            (-> validation-layers  tile :fuel-model)
;;         ffs           (-> validation-layers  tile :flammap-fire-spread)
;;         hfs           (-> validation-outputs tile :hfire-fire-spread)
;;         ffs-cells     (set (filter (fn [[i j]] (pos? (d/mget ffs i j))) (m/index-seq ffs)))
;;         hfs-cells     (set (filter (fn [[i j]] (pos? (d/mget hfs i j))) (m/index-seq hfs)))
;;         agreement     (count (set/intersection ffs-cells hfs-cells))
;;         overpred      (count (set/difference hfs-cells ffs-cells))
;;         underpred     (count (set/difference ffs-cells hfs-cells))
;;         target        (count ffs-cells)
;;         fl-diff       (-> validation-outputs tile :flame-length-difference)
;;         fl-diff-mean  (/ (dfn/sum fl-diff) (m/ecount fl-diff))
;;         fl-diff-var   (/ (dfn/sum (d/emap #(Math/pow (- fl-diff-mean %) 2.0) nil fl-diff)) (m/ecount fl-diff))
;;         fl-diff-stdev (Math/sqrt fl-diff-var)]
;;     (println (format "| %s | %3d%%(%3d) | %3d%%(%3d) | %3d%%(%3d) | %4.1f | %4.1f |"
;;                      (subs (name tile) 4)
;;                      (int (* 100.0 (/ agreement target)))
;;                      agreement
;;                      (int (* 100.0 (/ overpred target)))
;;                      overpred
;;                      (int (* 100.0 (/ underpred target)))
```

```
;;                      underpred
;;                      fl-diff-mean
;;                      fl-diff-stdev))))

;; (doseq [tile [:tile205 :tile210 :tile281 :tile310 :tile564 :tile643]]
;;   (let [hfire-fire-size   (->> (-> validation-outputs tile :hfire-fire-spread)
;;                                (t/tensor->buffer)
;;                                (filter #(= % 1.0))
;;                                (count)
;;                                (cells-to-acres test-cell-size))
;;         flammap-fire-size (->> (-> validation-layers tile :flammap-fire-spread)
;;                                (t/tensor->buffer)
;;                                (filter #(= % 1.0))
;;                                (count)
;;                                (cells-to-acres test-cell-size))
;;         grass-area        (->> (-> validation-layers tile :fuel-model)
;;                                (t/tensor->buffer)
;;                                (filter grass-fuel-model?)
;;                                (count)
;;                                (cells-to-acres test-cell-size))
;;         tile-number       (subs (name tile) 4)]
;;     (println (format "| %s | %s | %s | %s | %s |" tile-number flammap-fire-size hfire-fire-size (/ flammap-fire-size hfire-fire-s

;; (for [tile [:tile205 :tile210 :tile281 :tile310 :tile564 :tile643]]
;;   (let [error-fm (-> validation-outputs tile :error-fuel-models)]
;;     [tile (sort (distinct (filter pos? (t/tensor->buffer error-fm))))]))

;; (sort (distinct (apply concat (vals (into {} (for [tile [:tile205 :tile210 :tile281 :tile310 :tile564 :tile643]]
;;                                                (let [fm (-> validation-layers tile :fuel-model)]
;;                                                  [tile (sort (distinct (filter pos? (t/tensor->buffer fm))))])))))))

;; (sort (distinct (apply concat (vals (into {} (for [tile [:tile205 :tile210 :tile281 :tile310 :tile564 :tile643]]
;;                                                (let [error-fm (-> validation-outputs tile :error-fuel-models)]
;;                                                  [tile (sort (distinct (filter pos? (t/tensor->buffer error-fm))))])))))))
```

10.0.3 gridfire.build-test-db

```
(ns gridfire.build-test-db
  (:require [clojure.java.shell :as sh]
            [clojure.string :as str]
            [triangulum.logging :refer [log-str]]))

(def path-env (System/getenv "PATH"))

(defn parse-as-sh-cmd
  "Split string into an array for use with clojure.java.shell/sh."
  [s]
  (loop [chars (seq s)
         acc   []]
    (if (empty? chars)
      acc
      (if (= \` (first chars))
        (recur (->> chars (rest) (drop-while #(not= \` %)) (rest))
               (->> chars (rest) (take-while #(not= \` %)) (apply str) (str/trim) (conj acc)))
        (recur (->> chars (drop-while #(not= \` %)))
               (->> chars (take-while #(not= \` %)) (apply str)
                    (str/trim) (#(str/split % #" ")) (remove str/blank?) (into acc)))))))

(defn sh-wrapper [dir env verbose & commands]
  (sh/with-sh-dir dir
    (sh/with-sh-env (merge {:PATH path-env} env)
      (reduce (fn [acc cmd]
                (let [{:keys [out err]} (apply sh/sh (parse-as-sh-cmd cmd))]
                  (str acc (when verbose out) err)))
              ""
              commands))))
```

```
(defn load-default-data [verbose]
  (log-str "Loading landfire rasters...")
  (log-str "Please enter the gridfire_test user's password:")
  (flush)
  (let [password (String/valueOf (.readPassword (System/console)))]
    (->> (sh-wrapper "./test/gridfire/resources"
                     {:PGPASSWORD password}
                     verbose
                     "sh ../../../resources/import_landfire_rasters.sh gridfire_test landfire 900914"
                     "sh ../../../resources/import_ignition_rasters.sh gridfire_test ignition 900914")
         (log-str))
    (->> (sh-wrapper "./test/gridfire/resources/weather-test"
                     {:PGPASSWORD password}
                     verbose
                     "sh ../../../../resources/import_weather_rasters.sh gridfire_test weather 900914 800x800")
         (log-str))))

(defn build-everything [verbose]
  (log-str "Building database...")
  (log-str "Please enter the postgres user's password:")
  (flush)
  (let [password (String/valueOf (.readPassword (System/console)))]
    (->> (sh-wrapper "./src/sql"
                     {:PGPASSWORD password}
                     verbose
                     "psql -h localhost -U postgres -f create_test_db.sql")
         (log-str)))
  (load-default-data verbose))

(defn -main [& args]
  (build-everything true)
  (shutdown-agents))
```