Cracking program

The directory 2D contains couple of program that generate the surfaces, then simulate crack formation on those surfaces, and programs to view the generated data.

Requirements to compile and run these:

- C++ compiler
- OpenGL dev files
- qhull program (http://www.qhull.org/)
  - I was able to find a compiled package for my Ubuntu in the standard repository
  - the excecutables **`qhull`** and **`rbox`** from this packages should be in your PATH

To compile the programs:

- run **`make`** in the source directory

The following executables should be produced:

`relax`	Used to uniformly spread points using simple particle repell model inside a rectangular bounding box. This executable is used by the surface generation programs (gen*), but can be run manuall as well.
`relax-anim`	Similar to relax, but can produce PNG frames as the particles repell each other. Good for making movies.
`genplane`	Creates a flat rectangular model of a surface.
`genplane2`	Similar to genplane but with more options to control surface thickness and some simulation parameters.
`genplane3`	I think this is genplane2 + ability to define your own bounding boxes, i.e. Non-rectangular surface patches should be possible.
`genplane-bump`	I have a feeling this is an obsolete file.
`gensphere`	Can create a spherical model.
`gencylinder`	Creates a cylindrical model, with controllable radii thickness at topand bottom.
`gencyliner2`	Probably an older version of gencyliner.
`perlin`	Can produce perlin noise images on standard output, in PPM format. E.g. **`./perlin 100 100 1 10 1 1.5 gray > /tmp/x`**
`surface2fem`	Can be used to create models for the crack simulators from a simple triangle mesh file. Read the source for the format – it is trivial (points & triangles). Beware of degenerate surface models, especially if thickness is large and surface has crevices.
`viewer`	Use this to view models, for both initial and cracked surfaces. It produces nicer output than the simulator. It can also export to PDF and Povray.
`viewer-cyl`	I think this can be used to display a flat model as a cylinder. Not sure what was the purpose of this.
`main`	This is the main crack generator. It can display the models too, as they are generated.

Example

Here is a simple example to illustrate the complete process of generating some fracture pattern from scratch.

Step 1:

Generate the input model and view it.

```
./genplane 1000 1 r 1 1 > test.dat
```

This will create a file called test.dat that will contain the surface. You can edit the file to adjust some simulation constants. The file will looke something like this:

```
# Requested number of wedges: 1000
# Number of wedges:  1888


sim_time_total = 1.0
min_time_step = 0.001
max_time_step = 1.0000
sim_time_curr = 0.0
growth_x = 0.1
growth_y = 0.1
growth_z = 0.0
shrink_top_t0 = 0.0
shrink_top_t1 = 0.0
shrink_top_val = 0.0
shrink_bot_t0 = 0.0
shrink_bot_t1 = 0.0
shrink_bot_val = 0.0
shrink_height_t0 = 0.0
shrink_height_val0 = 0.0
shrink_height_t1 = 0.0
shrink_height_val1 = 0.0
gravity_x = 0.0
gravity_y = 0.0
gravity_z = 0.0

fracture_tip_inertia = 1
max_break_size = 0.0001
min_refine_size = 10.01
crack_tip_element_size = 0.1
crack_tip_sub_element_size = 0.001
min_element_size = -0.01
max_element_size = 0.1
precision = 1e-6

error_radius = 1
min_error_radius = 0.1

progressive_save = true
progressive_save_fmask = "/scratch/federl/%N-
%T.res"
progressive_save_skip = 0
```

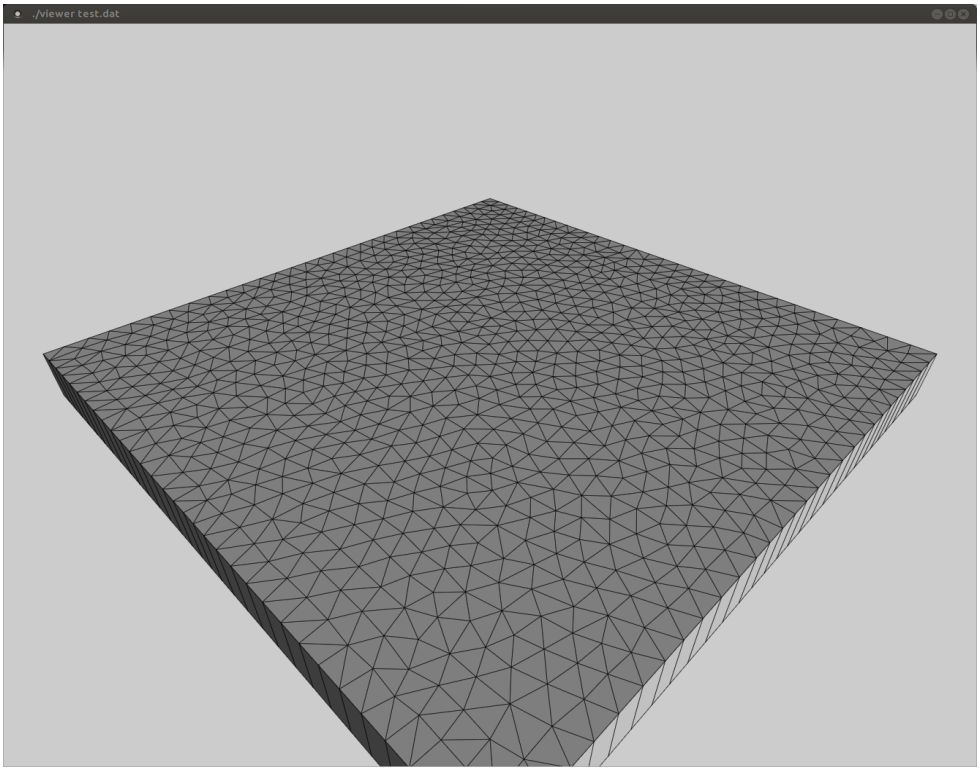
```
# description of how to randomize material
properties
BEGIN material_properties
        young_modulus constant 1000.0
        poisson_ratio constant 0.3
        yield_stress constant 70.0
        fracture_toughness constant 0.01
END material_properties

2012 # number of nodes
-0.500 -0.500 0.000 0.000 0.000 0.000 -0.500
-0.500 0.000 1
-0.500 -0.500 0.085 0.000 0.000 0.000 -0.500
-0.500 0.000 0
-0.356 -0.467 0.000 0.000 0.000 0.000 -0.356
-0.467 0.000 1
-0.356 -0.467 0.085 0.000 0.000 0.000 -0.356
-0.467 0.000 0
... rest of the 2012 nodes ...

1888 # number of elements

fivewall 0 277 115 1377 276 114 1376
fivewall 1 1187 1525 1531 1186 1524 1530
fivewall 2 1237 1323 63 1236 1322 62
fivewall 3 1963 1237 63 1962 1236 62
... rest of the 1888 elements ...

0 # number of fracture tips
```

Step 2:

To view the model, run the viewer on it:

```
./viewer test.dat
```

which should open a window that looks similar to this:



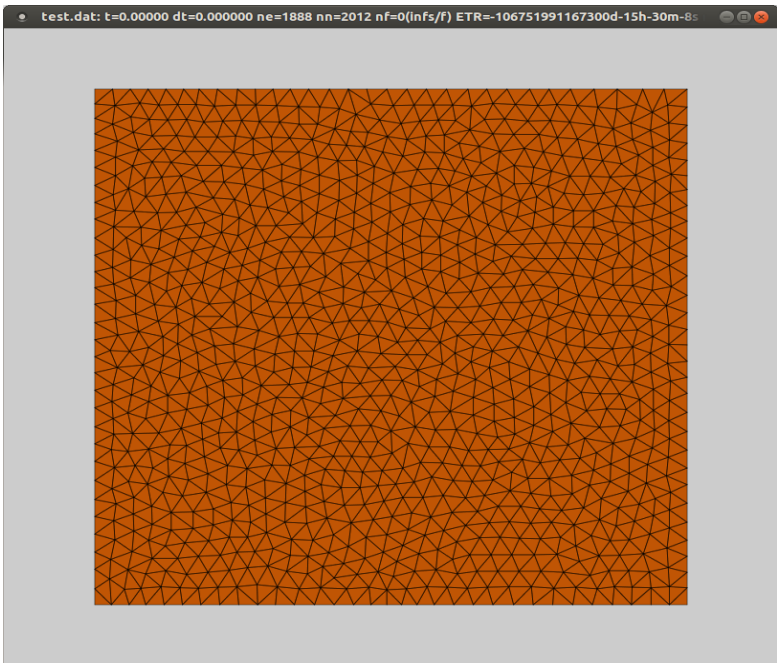
You can use the mouse (left & middle) buttons to rotate & zoom the object. Right click gives you a popup dialog with some options of exporting the image. I believe there is only one key that does anything, the letter 'T'. It changes the display type (wireframe, polygons, ...).

Step 3:

To run the simulator on this file, execute:

```
./main test.dat
```

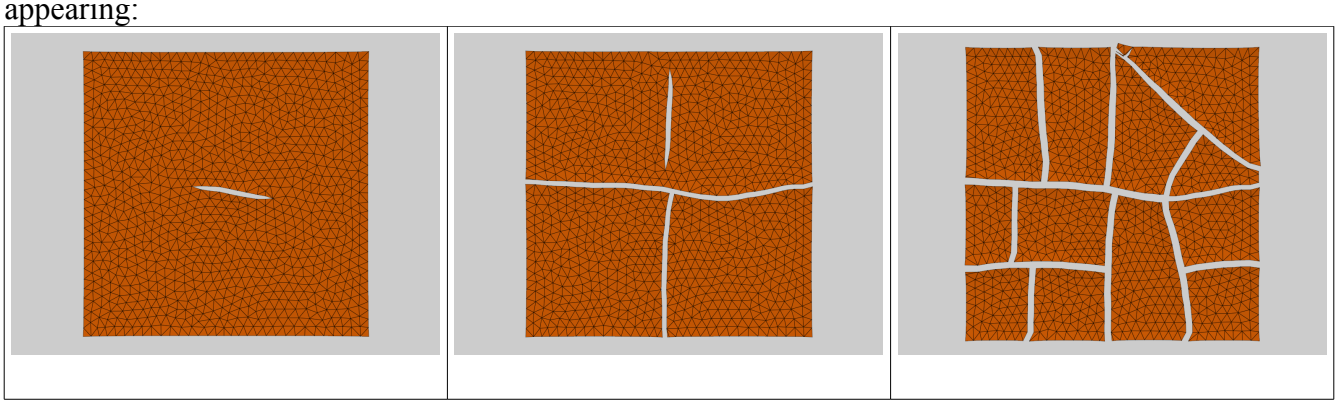
This will open up a window like this:



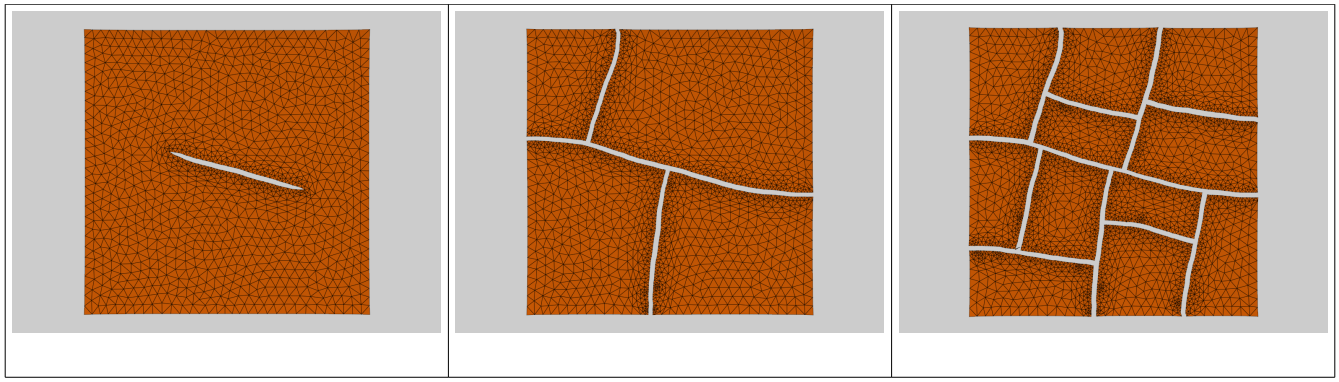
You can use the mouse to rotate/zoom the object. There is also a popup menu to start/stop simulation, as well as to save the current model to a file. Multiple keyboard shurtcuts are also available:

```
<space> toggles an element
<0> relax
<1> prettify mesh
<2> debug stuff
<3> peel
<4> split node
<k> insert sink
<s> subdivides an element
<h> displays this help
<i> prints some debuggin infor
<a> toggles display of axes
<w> toggles wireframe
<l> toggles wall display
<c> toggles colors used to draw elements
    normal -> stress -> yield_stree
<p> show stress planes in the nodes
<q> show stress planes in the elements
<z> print all stresses (nodes & elements) on
    stderr
<n> toggle display of node numbers
<-> halve the size of nodal cubes
<=> increase (10%) the size of nodal cubes
<f> toggle font
<g> toggle fog
<r> toggle disp. of background growth vectors
<o> draw original shapes toggle
<`> draw deformed/undeformed model toggle
<y> show plastic zones
<j> calculate precise stress at node
<k> fracture at node
```

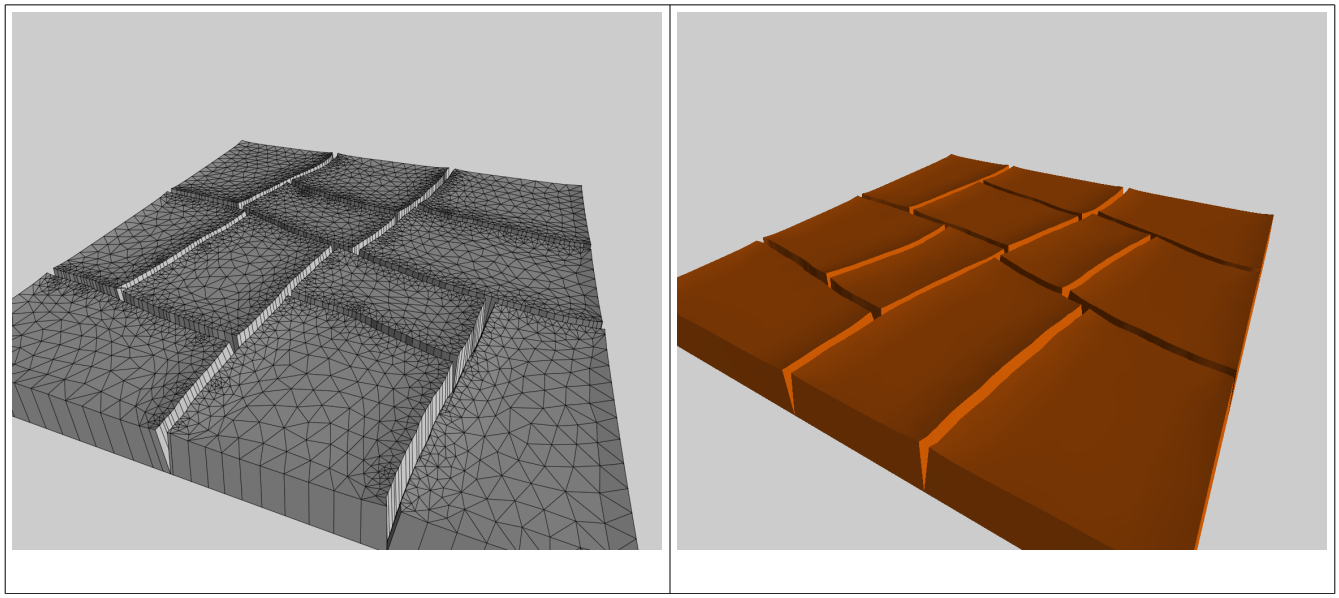
When you run the simulation (via the popup menu), you should see the surface growing and cracks appearing:



Youc an modify input file to see the dynamic subdivision (refinement around fractures). Set the value **`crack_tip_element_size = 0.02`** and re-run the simulation:



You can view the cracked model in the viewer:



Sample models and results

There are many different models and results in the directory 2D/Models. Most files there have 1 of two extensions **`.dat`** or **`.res`**. Both types of files are in the same file format, but by convention the **`.dat`** files are the original files, and **`.res`** files are the result files. You can load all of them into the simulator and into the viewer as well.

More advanced random point generator & triangulator.

In the directory **`Relax`** you can find some code to generate random points bounded by a non-convex polygon and optionally generate a triangular mesh for them (also bounded by the same polygon.

Requirements to compile and run:

- C++ compiler
- OpenGL dev files
- optionally 2 programs called **`triangle`** and **`showme`** , described here:
  - http://people.sc.fsu.edu/~jburkardt/c_src/triangle/triangle.html
  - I found a compiled and ready to install binary package called triangle-bin for my Ubuntu in the standard repository.
  - The binaries triangulate and showme from this package should be in your PATH.

To compile the code:

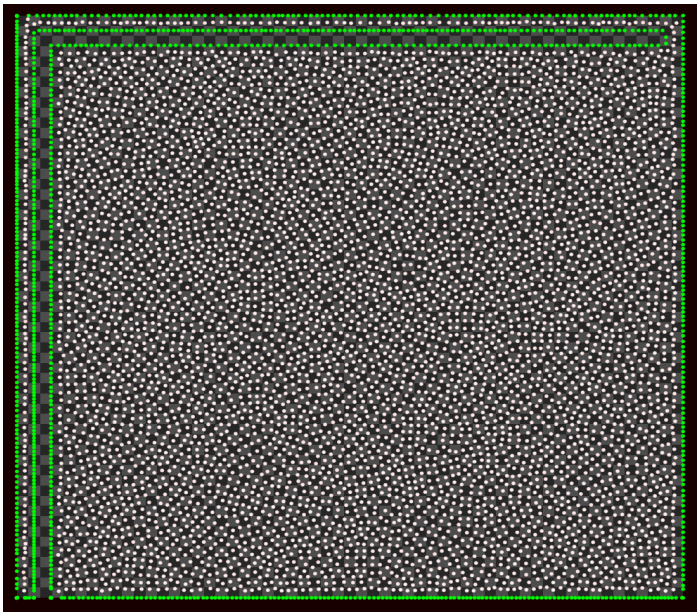
- run **`make`** in the source directory

To run the code you need an input file. The input file format looks something like this:

Input file x.dat	Notes
10 3 1 40 1 40 40 1 40 1 1 2 1 2 39 39 39 39 38 3 38 10000	Number of points defining the bounding polygon coordinates for point 1 coordinates for point 2 ... number of points to distribute

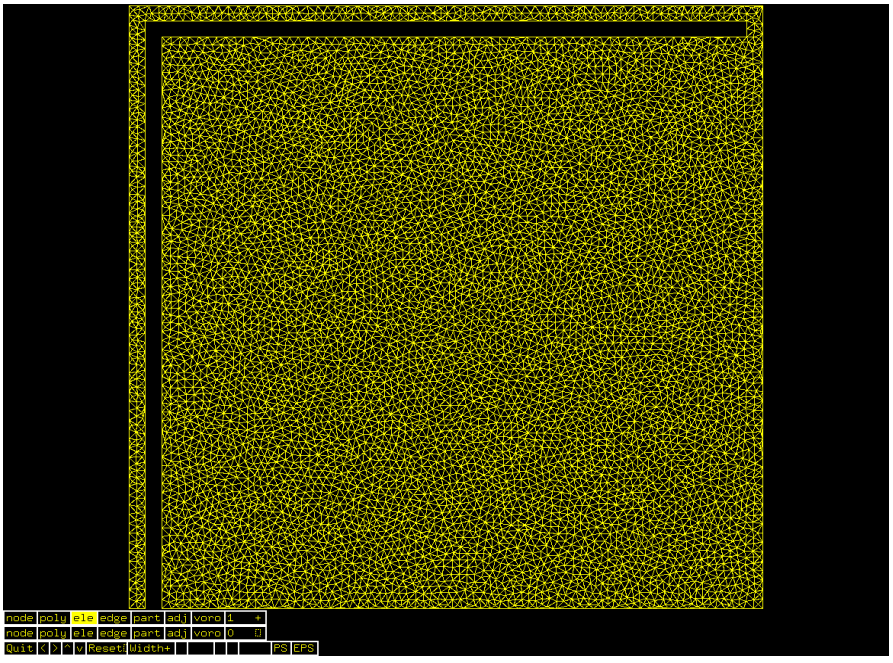
To generate the points based on this file (and display graphically what's going on), do this:

- `./relax -g < x.dat`
- A window should pop up, with a single point displayed. It has a popup menu with a couple of options (via right click).
- Select 'Run' to start the point generation. You should soon see something like this:



You can save the generated points to a file by choosing the 'Save' option from the popup menu. The output will be saved in the current directory as **`out.poly`**. The format is fairly straightforward and is described on the web page for the triangle program.

You can also triangulate the resulting points by chosing the 'Triangulate' option from the popup. This will automatically run the **`triangle`** program and immediately after a **`showme`** program to display results. The results of triangle are saved in **`/tmp/relax-data-<random string>.<extension>`** files. Again, read the docs for triangle to understand the format. The showme program outputs something like this:



The relax program has some additional options, you can see them by running:

```
relax -h
```

```
Usage: relax [-h] [-g] [-i n_iterations]
             [-v] [-b] [-t] [-j jitter] [-r radius perturbation]
             [-s thr] [-d thr] [-w weight]

       -h displays this help
       -g produces graphical result
       -i sets the number of iterations
       -v verbose
       -j radius (relative to desired_d)
       -r repulsion radius perturbation (0..1, default 0.2)
       -s threshold for splitting
       -d threshold for dying
       -w sensitivity to the force (default 1)
```

You will probably want to look into the sources for the exact meaning of these parameters.

There is also an executable called relax-anim in the directory. This can be used to produce movies of the animation. Individual frames are saved into a hard-coded directory as PNG files.

General algorithm for distributing n points inside a polygon

- Read in the polygon
- Initialize n points somewhere inside the polygon
  - Couple of choices here:
    - find some point in the polygon and initialize all points to this same location, or
    - initialize all points to be on a random vertex on the polygon, or
    - generate random points within the bounding box of the polygon, and accept the first n that are within the polygon
  - associate each point with a perturbed force field
  - the field attenuates to 0 with radius r
- repeat forever:
  - find out the total force acting on each point
    - to make this fast, the points are divided by a grid of size <r>. That way, the only points that can contribute any force to a given point are the points in the surrounding 8 grid tiles
  - perturb the total force by a random angle
    - this avoids having too many points being 'stuck' on edges
  - for each point
    - calculate a displacement based on the total force acting on the point
    - if this displacement would cross a boundary of the polygon, clip it
    - apply the displacement to the point
    - remember the largest displacement among all points
  - add the largest displacement from above to a moving exponential average of largest displacements
  - if the exponential average is below some threshold, break out of the loop