HACKT IPPLE

An interactive physics-driven placement-engine manual

David Fang

This manual describes the usage and operation of HACKT's `ipple` placement engine.
This document can also be found online at http://www.csl.cornell.edu/~fang/hackt/ipple.
The main project home page is http://www.csl.cornell.edu/~fang/hackt/.

Short Contents

Table of Contents

1 Introduction

`ipple` is a physics simulator implementation of a general placement engine. `ipple` is part of the HACKT tool set, but is actually an entirely generic tool for use in any design toolchain. The placement engine uses various methods of mechanical simulation to find the minimum potential energy resting state of a system. The simulation however is user-driven and can be scripted to produce custom simulated annealing schedules.

2 Usage

FYI: the documentation here is extracted from source file '`main/ipple.cc`'.

`-a` *file* [User Option]
: Automatically save checkpoint *file* upon exit, regardless of the exit status. Useful for debugging and resuming simulations.

`-b` [User Option]
: Batch mode. Run non-interactively, suppressing the prompt and disabling tab-completion (from readline or editline). Useful for scripted jobs. Opposite of '`-i`'.

`-d` *N* [User Option]
: Not yet supported. Simulate in *N*-dimension space. Current fixed to 3D only.

`-h` [User Option]
: Print command-line options help and exit.

`-H` [User Option]
: Print list of all interpreter commands and exit.

`-i` [User Option]
: Interactive mode. Show prompt before each command. Enable tab-completion if built with readline/editline. Opposite of '`-b`'.

`-I` *path* (repeatable) [User Option]
: Append *path* to the list of paths to search for sourcing other command scripts in the interpreter.

`-v` [User Option]
: Print version and exit.

3 Global Parameters

This section describes all of the global parameters that affect simulation. All of these parameters are set by the `parameter` command with `parameter key=value ...`.

Simulation parameters:

`time_step` *val* [User Option]
: Sets the step-size for simulation and numerical integration.

Convergence parameters:

`position_tolerance` *val* [User Option]
: Set the threshold for maximum delta in position for considering convergence.

`velocity_tolerance` *val* [User Option]
: Set the threshold for maximum delta in velocity for considering convergence.

`energy_tolerance` *val* [User Option]
: Set the threshold for maximum delta in energy for considering convergence.

Physics parameters:

`damping` *val* [User Option]
: Viscous damping coefficient that applies a linearly dependent force in the opposite direction of the velocity vector.

`temperature` *val* [User Option]
: Temperature is used to randomly perturb the position of all objects after each iteration. Random perturbation is means by which annealing is simulated.

`repulsion_coeff` *val* [User Option]
`repulsion_constant` *val* [User Option]
: Object proximity is modeled as spring repulsion. '`repulsion_coeff`' is the repulsion spring coefficient for a compressed spring, which models the collision of soft objects. '`repulsion_constant`' is a constant additive force term for the spring that is compressed to model the hardness of the boundary.

Gravity parameters:

`x_gravity_coeff` *val* [User Option]
`y_gravity_coeff` *val* [User Option]
`z_gravity_coeff` *val* [User Option]
: The linear spring coefficient for gravity wells in the x,y,z direction.

`x_gravity_constant` *val* [User Option]
`y_gravity_constant` *val* [User Option]
`z_gravity_constant` *val* [User Option]
: The constant force term for gravity wells in the x,y,z directions. Setting this to non-zero helps attract objects closer to the well.

Feedback parameters:

`precision` *val* [User Option]
Sets the precision of floating-point values that are printed.

`watch_objects` *val* [User Option]
Set to 1 to print out object position updates with every iteration.

`watch_energy` *val* [User Option]
Set to 1 to print out object position updates with every iteration.

`report_iterations` *val* [User Option]
Set to 1 to report number of iterations in numerical convergence routines.

4 Commands

This chapter documents the various commands available in the interpreter. Commands are organized into categories.

FYI: the command documentation has been extracted from source file '`PR/pr-command.cc`'.

4.1 Built-in Commands

The following commands are listed in the `builtin` category.

`help` *cmd* [Command]
: Help on command or category *cmd*. '`help all`' gives a list of all commands available in all categories. '`help help`' tells you how to use `help`.

`echo` *args ...* [Command]
: Prints the arguments back to stdout.

`#` *...* [Command]
`comment` *...* [Command]
: Whole line comment, ignored by interpreter.

`exit` [Command]
`quit` [Command]
: Exit the simulator.

`abort` [Command]
: Exit the simulator with a fatal (non-zero) exit status.

`repeat` *n cmd...* [Command]
: Repeat a command *cmd* a fixed number of times, *n*. If there are any errors in during command processing, the loop will terminate early with a diagnostic message.

`interpret` [Command]
: Open an interactive subshell of the interpreter, by re-opening the standard input stream. This is useful when you want to break in the middle of a non-interactive script and let the user take control temporarily before returning control back to the script. The `exit` command or *`Ctrl-D`* sends the EOF signal to exit the current interactive level of input and return control to the parent. The level of shell is indicated by additional `>` characters in the prompt. This works if `ipple` was originally launched interactively and without redirecting a script through stdin.

```
$ ipple
ipple> !cat foo.iplrc
# foo.iplrc
echo hello world
interpret
echo goodbye world
ipple> source foo.iplrc
hello world
```

```
ipple>> echo where am I?
where am I?
ipple>> exit
goodbye world
ipple> exit
$
```

The following command is useful for showing each executed command.

`echo-commands` *arg* [Command]
: Enables or disables echoing of each interpreted command and tracing through sourced script files. *arg* is either "on" or "off". Default off.

The following commands pertain to command aliases.

`alias` *cmd args* [Command]
: Defines an alias, whereby the interpreter expands *cmd* into *args* before interpreting the command. *args* may consist of multiple tokens. This is useful for shortening common commands.

`unalias` *cmd* [Command]
: Undefines an existing alias *cmd*.

`unaliasall` [Command]
: Undefines *all* aliases.

`aliases` [Command]
: Print a list of all known aliases registered with the interpreter.

Shell commands may be executed by prefixing a line with '!'. For example, '`!whoami`'.

New: Block comments are pseudo C-style, using `/*` and `*/` to enclose comments. It is recommended to start use block-comment delimiters on their own lines to avoid confusion. The line parser is very crude. Nested comments are supported. Files with unterminated comments will be reported as errors. `#`-comments are allowed within block comments.

4.2 General Commands

The following commands are listed in the `general` category.

`source` *script* [Command]
: Loads commands to the interpreter from the *script* file. File is searched through include paths given by the [`-I`], page 3 command-line option or the [`addpath`], page 8 command.

`addpath` *path* [Command]
: Appends *path* to the search path for sourcing scripts.

4.3 `simulation` Commands

`seed48` [*int int int*] [Command]
Corresponds to libc's seed48 function. With no argument, print the current values of the internal random number seed. With three (unsigned short) integers, sets the random number seed. Note: the seed is automatically saved and restored in checkpoints. The seed value is reset to 0 0 0 with the `reset` command, but not with the `initialize` command.

`place` *obj loc* [Command]
Manually move object indexed *obj* to location *loc*, a 3D vector.

`pin` *obj* [Command]
`unpin` *obj* [Command]
Fix object *obj* at its current location. This is usually done for boundary terminals. `unpin` allows an object to move freely.

`scatter` [Command]
Relocates every object to some random location within the bounding box. This is often done at the start of the simulation.

`step` [Command]
Advances the simulation one iteration using the current `time_step` (parameter). Each iteration evaluates the forces (spring, repulsion, gravity) acting upon all objects, and then updates the position and velocity by numerical integration.

`snap-gravity-wells` [Command]
Forces all objects to re-locate to the nearest gravity well in each hyperplane. Typically, this is used for final placement legalization.

`kill-momentum` [Command]
Resets velocities of all objects to 0.

`shake-all` [*maxdist*] [Command]
Randomly perturbs positions of all objects in random direction, uniformly random distance bounded by *maxdist*. This helps the system get unstuck from some local minima. If *maxdist* is omitted, the global `temperature` parameter is used instead.

Simulation and convergence routines.

`simple-converge` [Command]
Runs `step` iteratively until some convergence criterion is met. The convergence criterion used here is when the relative maximum change in position and velocity fall below the thresholds specified in the global parameters, `position_tolerance` and `velocity_tolerance`.

`descend-gradient` [Command]
Accelerates all objects in a straight line (constant acceleration determined by an initial force calculation), until potential energy no longer decreases monotonically. At the start of this routine, all velocity/momentum is reset to 0.

`descend-gradient-converge` [Command]
Repeatedly runs `descend-gradient` until local minimum in potential energy is found. This is very similar to the conjugate-gradient minimization method.

`descend-potential` [Command]
At the start of this routine, all velocity/momentum is reset to 0. Simulates transiently `step` until while potential energy is monotonically decreasing. Unlike `descend-gradient` the force is continuously updated with every iteration, so the paths taken by each object may curve. This stops early to avoid overshooting a local minimum due to kinetic energy.

`descend-potential-converge` [Command]
Repeatedly runs `descend-potential` until local minimum in potential energy is found.

4.4 `info` Commands

`dump-state` [Command]
Print information for the entire state of the system: parameters, objects, etc...

`dump-parameters` [Command]
Print global system parameters.

`dump-objects` [Command]
Print state and location of all objects.

`dump-positions` [Command]
Print location of all objects.

`dump-energy` [Command]
Print detailed kinetic and potential energy breakdown for the whole system.

Finally, `ipple` can export placements to other visualization tools. (It's the least I can do until a GUI is added!)

`emit-dot` [*file*] [Command]
Print to stdout or *file* in the dot language (from graphviz), suitable for generating a diagram of the current state of objects. Recommendation: run the output through the `fdp` (force-directed placement) command: `fdp -Tpdf input -o output.pdf`.

`emit-fig` [*file*] [Command]
Print to stdout or *file* in the fig language (xfig), suitable for generating a diagram of the current state of objects. Recommendation: run the output through the `fig2dev` export command, such as: `fig2dev -Lpdf input.fig -o output.pdf`.

4.5 `tracing` Commands

Checkpointing is useful for saving the state of the simulator, which allows one to interrupt and resume long simulations, and also examine points of failure in detail.

`save` *ckpt* [Command]
: Saves the current state of the production rules and nodes into a checkpoint file *ckpt*. The checkpoint file can be loaded to resume or replay a simulation later.

`load` *ckpt* [Command]
: Loads a `ipple` checkpoint file into the simulator state. Loading a checkpoint will not overwrite the current status of the auto-save file, the previous autosave command will keep effect. Loading a checkpoint, however, will close any open tracing streams.

4.6 `debug` Commands

This section is reserved for commands that are only useful for debugging the simulator. Some commands that end in `-debug` have already been mentioned in the other sections.

Command Index

Concept Index