Documentation for 2+1 dimension `globals.lisp`

November 30, 2011

Random Number generators

The random state for seeding the random number generator. This guarantees that we get a different sequence of random numbers each time we load the file. If you want the same sequence each time, which you would during testing to verify if a bug has been fixed, save the `*random-state*` to a file and load the state from that file.

```
(setf *random-state* (make-random-state t))
```

Counters for various simplices

The next four lines are counters for 3-simplicies, points and space-like 2-simplices. We recycle the ids for 3-simplices.

```
(defparameter *LAST-USED-3SXID* 0)
(defparameter *RECYCLED-3SX-IDS* '())
(defparameter *LAST-USED-POINT* 0)
(defparameter *LAST-USED-S2SXID* 0)
```

The following functions access these counters. `next-3simplex-id` returns a recycled id,

```
(defmacro next-pt ()
  `(incf *LAST-USED-POINT*))
(defmacro set-last-used-pt (pt)
  `(setf *LAST-USED-POINT* ,pt))
(defmacro next-s2simplex-id ()
  `(incf *LAST-USED-S2SXID*))
(defmacro next-3simplex-id ()
  `(if (null *RECYCLED-3SX-IDS*)
       (incf *LAST-USED-3SXID*)
       (pop *RECYCLED-3SX-IDS*)))
(defmacro recycle-3simplex-id (sxid)
  `(push ,sxid *RECYCLED-3SX-IDS*))
```

if possible, else increments the 3-simplex counter.

Hashtables for timelike subsimplices

timelike subsimplices and hashtables

```
(defun tlsubsx->id-hashfn (tlsx)
  (sxhash (sort (copy-list (third tlsx)) #'<)))
(defun tlsubsx->id-equality (tlsx1 tlsx2)
  (and (= (first tlsx1) (first tlsx2))
       (= (second tlsx1) (second tlsx2))
       (set-equal? (third tlsx1) (third tlsx2))))
(sb-ext:define-hash-table-test tlsubsx->id-equality tlsubsx->id-hashfn)
(defparameter *TL2SIMPLEX->ID* (make-hash-table :test 'tlsubsx->id-equality))
(defparameter *TL1SIMPLEX->ID* (make-hash-table :test 'tlsubsx->id-equality))
```

Hashtables for spacelike subsimplices

spacelike subsimplices and hashtables

```
(defun slsubsx->id-hashfn (slsx)
  (sxhash (sort (copy-list (second slsx)) #'<)))
(defun slsubsx->id-equality (slsx1 slsx2)
  (and (= (first slsx1) (first slsx2))
       (set-equal? (second slsx1) (second slsx2))))
(sb-ext:define-hash-table-test slsubsx->id-equality slsubsx->id-hashfn)
(defparameter *SL2SIMPLEX->ID* (make-hash-table :test 'slsubsx->id-equality))
(defparameter *SL1SIMPLEX->ID* (make-hash-table :test 'slsubsx->id-equality))
```