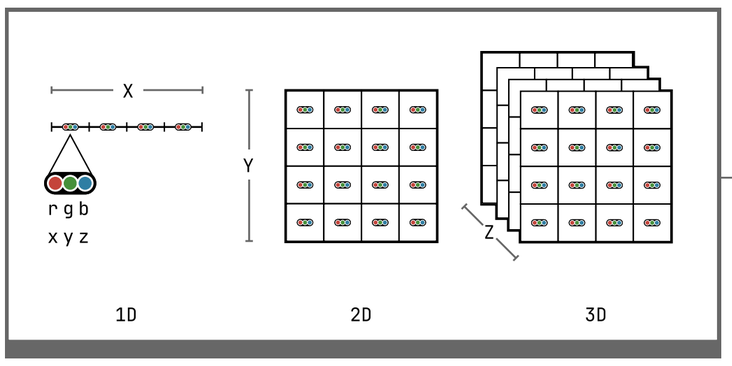
Utomata

Instrument for field-based computation

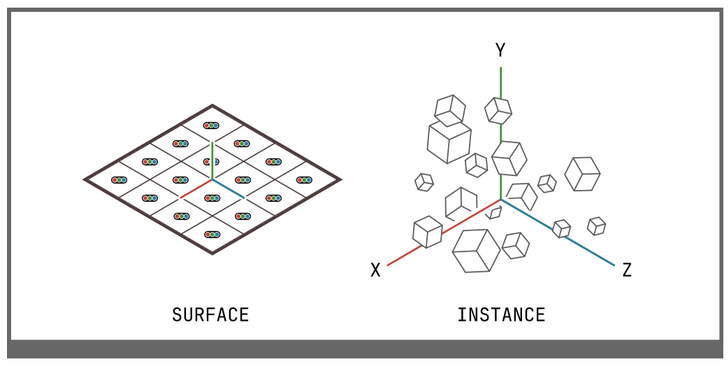
LANGUAGE SPEC

Version 0.0.1
Utomata Version 0.1.1

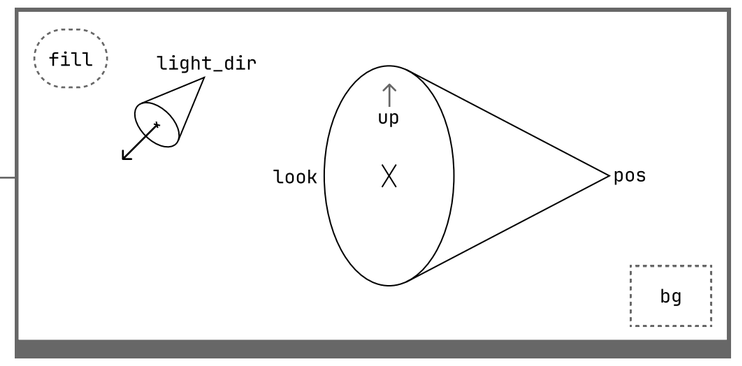
utomata.net



compute



project



render

FIELDS

```
#voxels{
   dim = (32,32,32);
   set = rand(1,2,3);
   run = frc(add(#, 0.01));
   val_dom = UNIT;
   every = 4;
}
```

PROPERTIES

dim	**(1, 1, 1)**
set	**(0)**
run	**#**
every	**1**
coord_dom	**SIGNED** | UNIT
val_dom	**SIGNED** | UNIT | REAL
sample	**SOURCE** | DEST
mode	**F32** | I32 | F16 | U8
bound	**WRAP** | CLAMP | ZERO
scale	**65536**
eps	**0.0001**

BUILT-IN VARIABLES

C	.xyz cell coordinate
step	current tick (0, 1, …, ∞)

UNITS

SIGNED	-1.0 to 1.0
UNIT	0.0 to 1.0
REAL	-∞ to ∞

REDUCERS

```
#A.SUM.BOX((-1,-1), (1, 1)){
   mlt(lrg(V, 0.5), v)
}
#A.SUM.RAD( (2) ){
   mlt(lrg(V, 0.5), v)
}
```

OPERATORS

operator	**SUM**, AVG, MIN, MAX
enumeration	**BOX**, RAD
coordinate	one or two vectors
body	expression over sample

BUILT-IN VARIABLES

V	current int lattice point
I	displacement from P
U	random signed vector at P

FORMATIONS

```
~cubes{
   dim = (32,32,32);
   shape = CUBE;
   col = #voxels;
   i_pos = C;
   i_scl = lrg(#voxels, 0.2);
}
```

PROPERTIES

dim	**(1, 1, 1)**
type	**SURFACE** | INSTANCE
col	**(1, 1, 1)**
disp	**(C.x, C.y, C.z)**
pos	**(0, 0, 0)**
rot	**(0, 0, 0)**
scl	**(1, 1, 1)**
rot_unit	**TAU** | SIGNED | UNIT
coord_dom	**SIGNED** | UNIT
shade	**FACE** | SMOOTH
filter	**NEAREST** | INTERP
sample	**SOURCE** | DEST
cull	**AUTO** | NONE | BACK | FRONT

INSTANCE PROPERTIES

shape	**QUAD** | CUBE | ICO | CONE
i_pos	**(C.x, C.y, C.z)**
i_rot	**(0, 0, 0)**
i_scl	**(1, 1, 1)**
i_piv	**(0, 0, 0)**
i_geo	**(1, 0, -0.4615385)**
i_res	**(0, 2, 0)**

LATT

```
// cubic Perlin noise
LATT.CUBE.QUNITIC[ mlt(C, 8) ]{
   dot(nrm(R), D)
}
```

PROPERTIES

geometry	**RECT** | CUBE
blend	**QUINTIC** | LINEAR | CUBIC
body	expression over point
P	current int lattice point
D	displacement from P
R	random signed vector at P

VIEWPORTS

```
~camera1{
   pos   = (5,5,-5);
   look  = (0,0,0);
   proj  = (0.1, 20, 0);
   bg    = (0.2, 0.6, 0.2);
   light = (0.8, 0.5, 0.5);
}
```

PROPERTIES

pos	**(0, 0, 2.4142)**
look	**(0, 0, 0)**
up	**(0, 1, 0)**
proj	**(0.1, 100, 0.5)**
light	**(0, 0, 0)**
light_dir	**(1, -1, -0.5)**
fill	**(1,1,1)**

TRIGGERS

```
@freeze(#A.step == 200){
   stop(#A, #B);
   log(#A[0,0], output_chan);
   set(#C);
}
```

PREDICATES

step == N	global step
#A.step == N	field step
step % N == P	modulo test
#A[x,y].x > V	cell thresh trigg
#A[x,y].x < V	cell thresh trigg

ACTIONS

set(#A, …)	reset field(s)
stop(#A, …)	pause field(s)
run(#A, …)	run if paused
step(#A, …)	single step
log(#A[x,y])	print cell value
log(#A, key)	save to output
log(#A[x,y], key)	save value text
halt()	pause all
flush()	flush text/json
next(<stream>)	advance input src
end()	batch complete

BINDINGS

```
$setting = (0.1, 0.2, 0.3);
```

UNIFORMS

Global single vector definitions. Can be set via the API without triggering recompilation.

```
&new = add(#pos, mlt(#vel, &drag));
```

MACROS

Reusable expressions. Must be complete. can be used anywhere and can contain other macros (but not self)

```
run = <runExpression>;
```

INPUT

enumerable source segments that can be set to a provider via the API. scriptable via @triggers

```
log(#A[0,0], output_chan);
```

OUTPUT

Data export channel can be set to a provider via the API. scriptable via @triggers

LOOKUP SEMANTICS

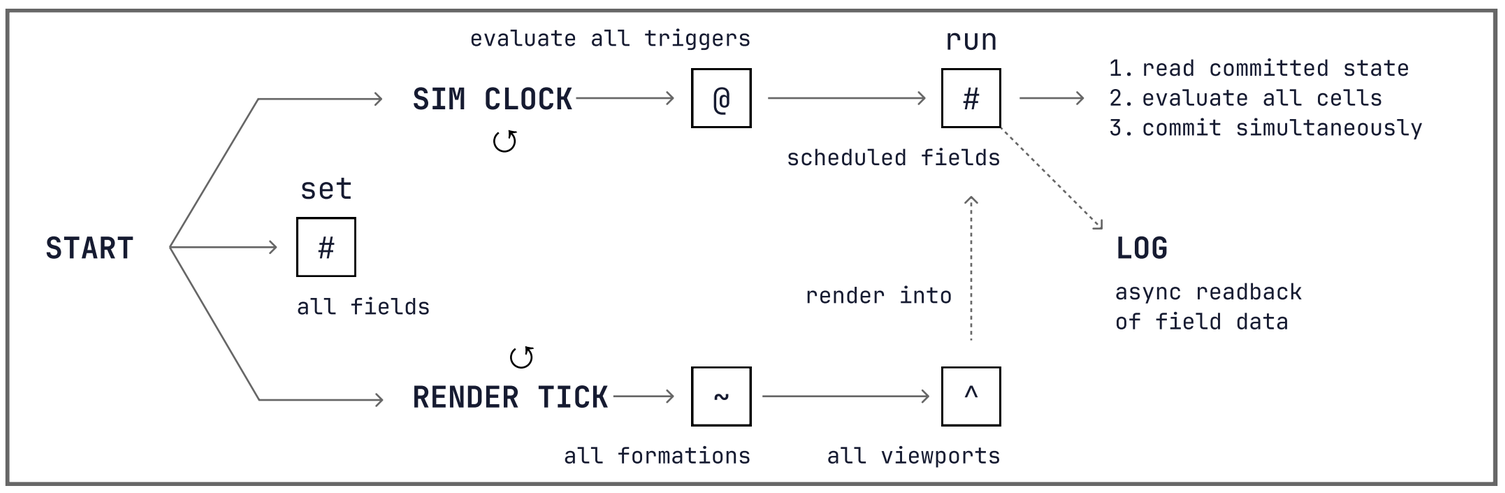
```
// 1D neighbors top-left/middle/right
&TL = #(-1,-1);
&TM = #(0,-1);
&TR = #(1,-1);

// 2D Moore neighborhood (box)
&V9 = #.SUM.BOX((-1,-1), (1,1));
&V8 = sub(&V9, #); // excl

// 2D Von-Neumann neighborhood (radial)
&V5 = #.SUM.RAD(1.25);
&V4 = sub(&V5, #); // excl

// filtered reducer
#vel.SUM[(-1,-1), (1,1)]{
   mlt(
      lrg(dst(#pos[U], #pos), 0.1),
      V
   )
}
```

EXECUTION MODEL



EXPRESSION EVALUATION

```
sgn(
   add(                 // OR-like
      eql(#A, &v),
      mlt(              // AND
         lrg(#B, &lo),
         sml(#B, &hi)
      )
   )
)

// ((#A==&v)||((#B>&lo)&&(#B<&hi)))
```

OPERATORS

LIFTING AND SWIZZLING

a	→	(a, a, a)
(a)	→	(a, a, a)
(a).y	→	(a, a, a)
(a, b)	→	(a, b, b)
(a, b, c).xyz	→	(a, b, c)
(a, b, c).z	→	(c, c, c)
(a, b, c).zxy	→	(c, a, b)
(a, b, c).xz	→	(a, c, c)

ARITHMETIC

add(a, b, …)	return sum of inputs
mlt(a, b, …)	return product of inputs
sub(a, b)	subtract b from a
div(a, b)	divide a by b
pow(a, b)	a to the power of b
mod(a, b)	modulo of a by b
sqt(a)	square root
log(a)	logarithm
abs(a)	absolute value of a
sgn(a)	sign of a (-1 | 0 | 1)

FRACTIONAL

flr(a)	closest lower integer
cil(a)	closest higher integer
rnd(a)	closest integer
frc(a)	fractional part of a

VECTOR ALGEBRA

dot(a, b)	return sum of inputs
dst(a)	return product of inputs
len(a)	return sum of inputs
nrm(a)	return product of inputs
aim(a)	return sum of inputs

VECTOR REDUCERS

sum(a)	sum of channels
sum(a, b, …)	sum of vectors
avg(a)	average of channels
avg(a, b, …)	average of vectors
min(a)	minimum channel value
min(a, b, …)	minimum per-channel
max(a)	maximum channel value
max(a, b, …)	maximum per-channel

TRIGONOMETRY

sin(a)	sine of a
cos(a)	cosine of a
tan(a)	tangent function of a
asn(a)	arc sine of a
acs(a)	arc cosine of a
atn(a)	arc tangent of a
atn(a, b)	two param arc tangent

BOOLEAN

eql(a, b)	1 if a == b, otherwise 0
lrg(a, b)	1 if a > b, otherwise 0
sml(a, b)	1 if a < b, otherwise 0

DOMAIN

dom_idx(a, b)	floor(a ÷ b)
dom_phs(a, b)	phase of a within b
dom_del(a, b, c)	disp across span

COLOR

hsl(r, g, b)	RGB to HSL
rgb(h, s, l)	HSL to RGB

CONSTANTS

PI	≈	3.14159
HALF_PI	≈	6.283192
TAU	≈	6.283192

UTILITIES

fof(a, b)	falloff
clp(a, b, c)	clamp a between b and c
mix(a, b, c)	mix a and and b by c
rmp(a, b, …)	?

RANDOM

rand(sx, sy, sz)	seeded random UNIT
srand(sx, sy, sz)	seeded rand SIGNED