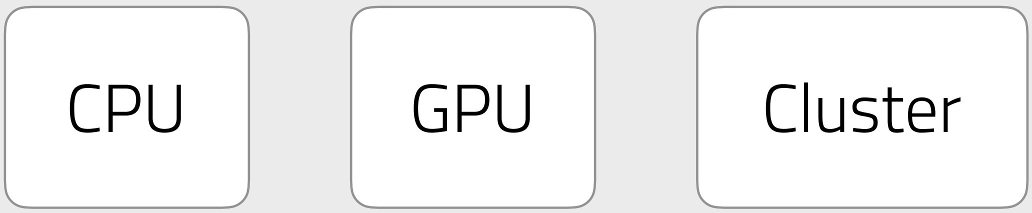
A DSL for Physical Simulation on GPUs and CPUs

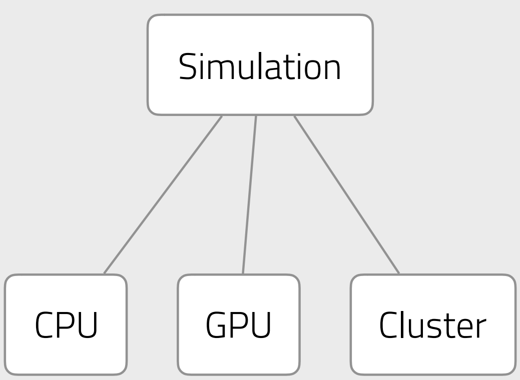
Gilbert Bernstein
Chinmayee Shah
Crystal Lemire
Zach DeVito
Matthew Fisher
Philip Levis
Pat Hanrahan

CPU

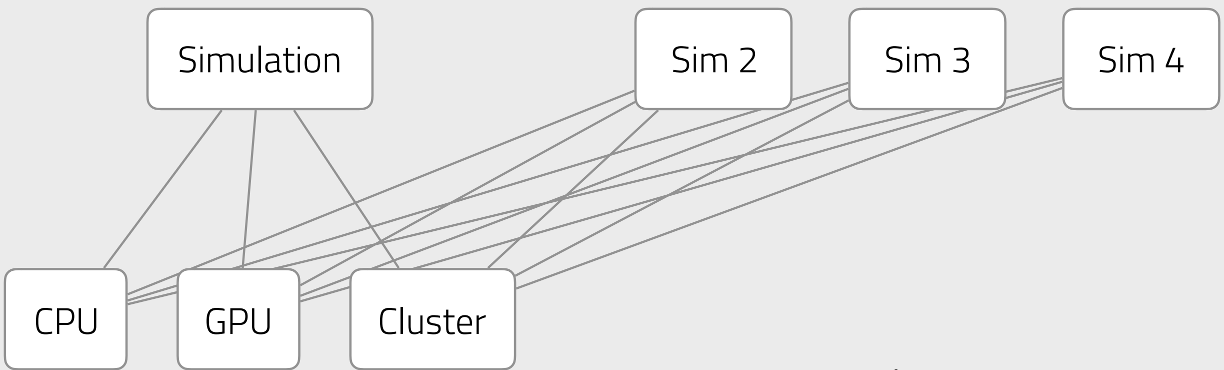
GPU



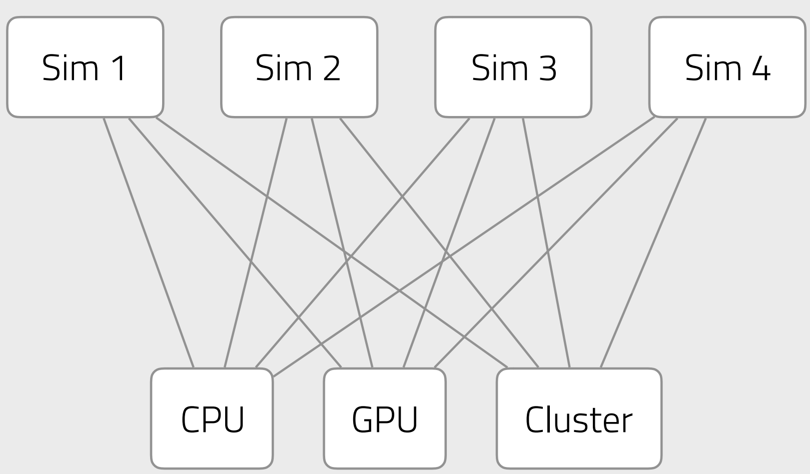
Porting Code is Expensive



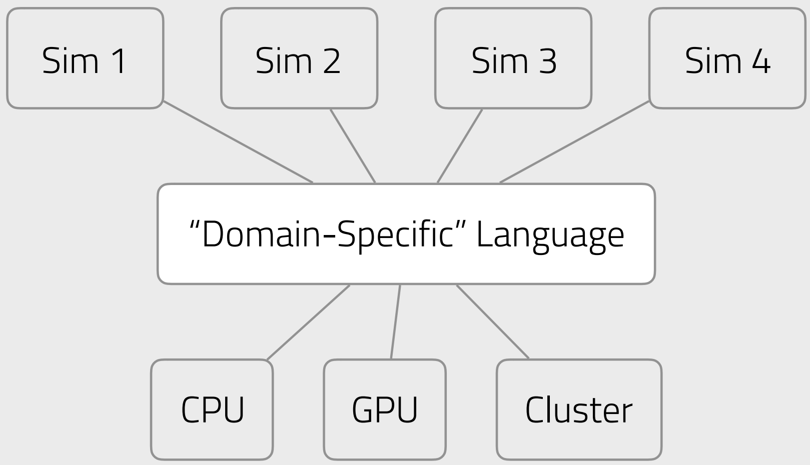
Porting Code is Expensive



Porting Code is Repetitive



Languages Abstract Hardware



Existing Languages in Graphics

PIXAR's RenderMan

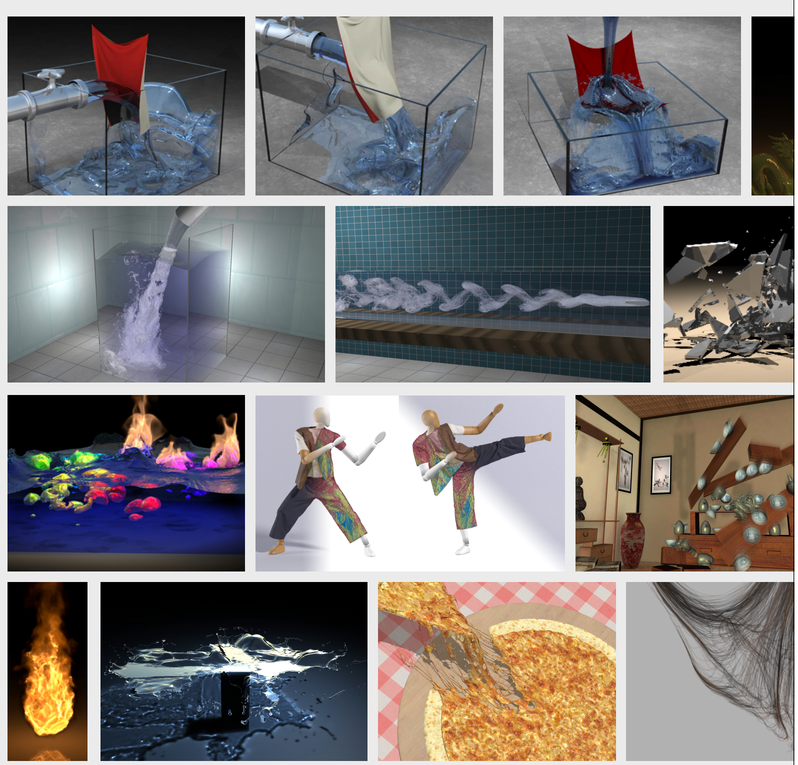
GLSL

Halide

Darkroom

What's tricky about designing languages for Simulation?

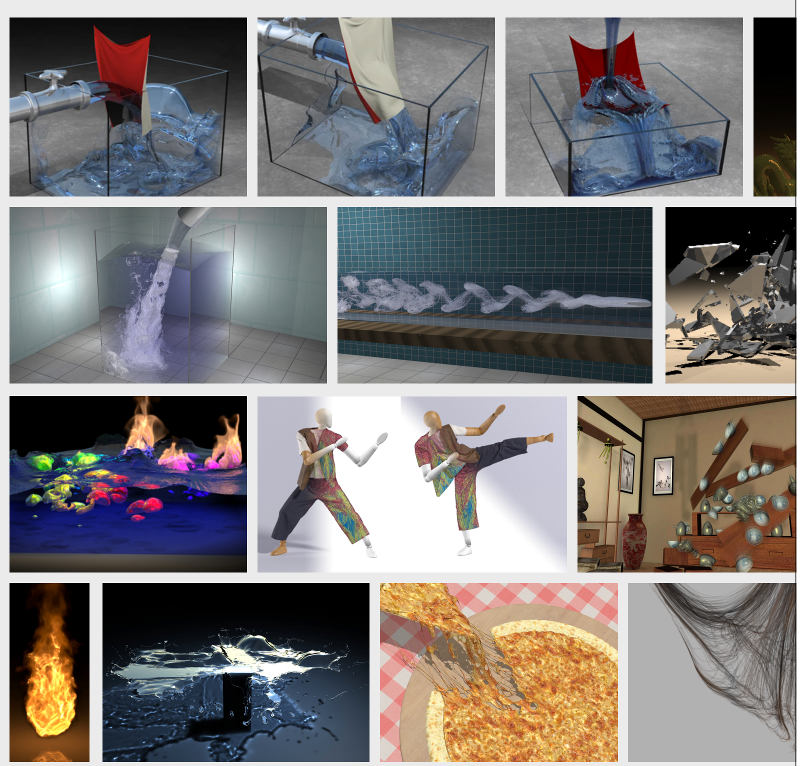
Simulations of Diverse Phenomena



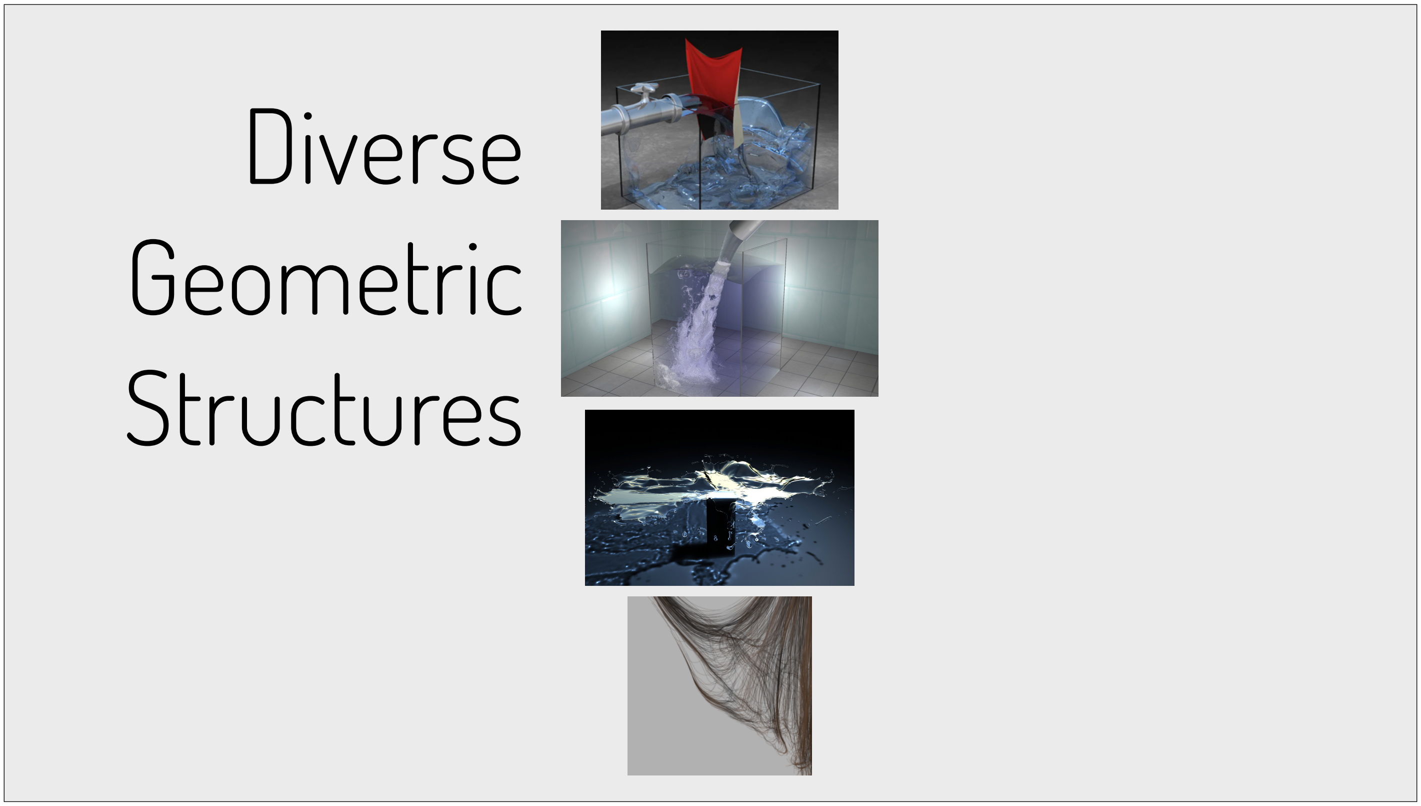
images © Ron Fedkiw, Doug James, Chris Wojtan, Rahul Narain, Andrew Selle

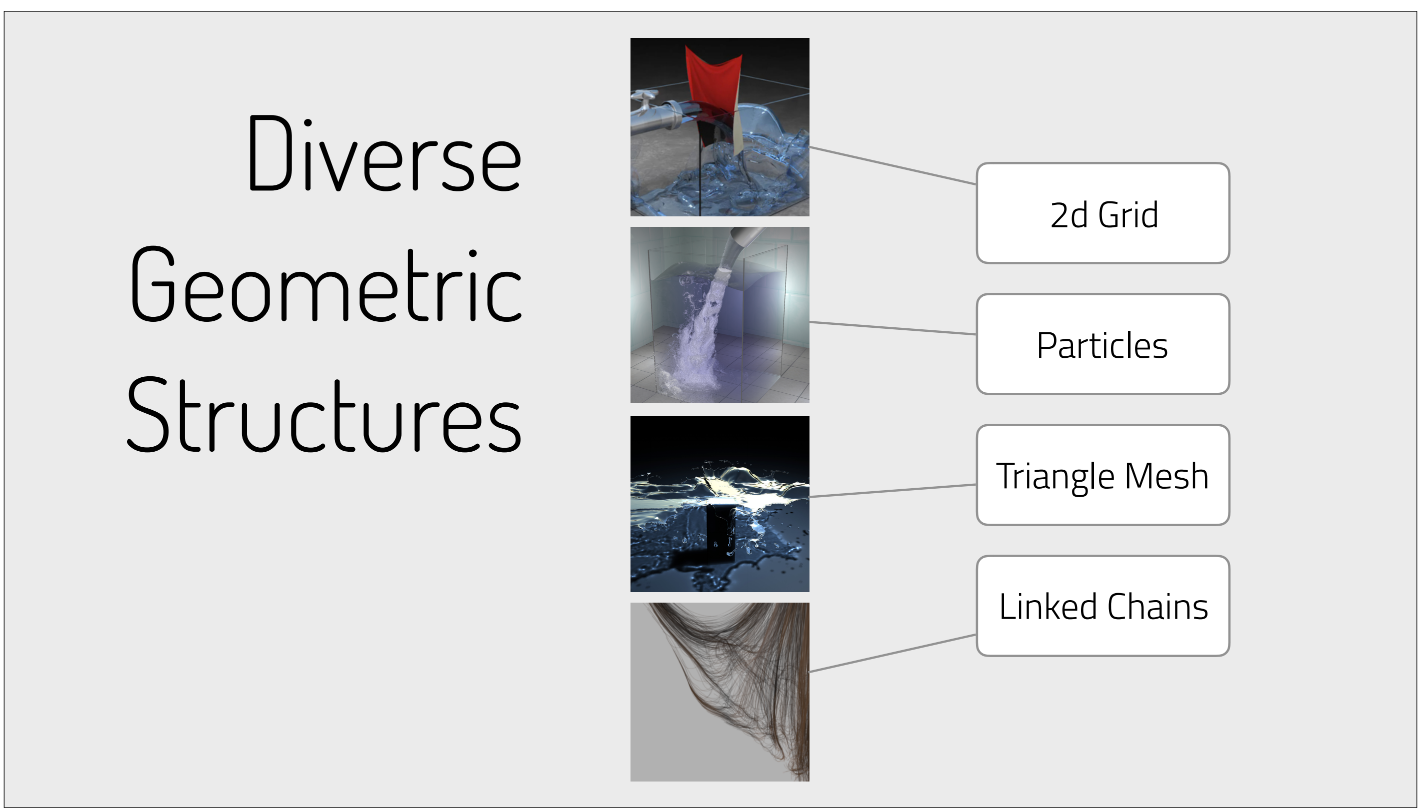
Simulations **Couple** Phenomena

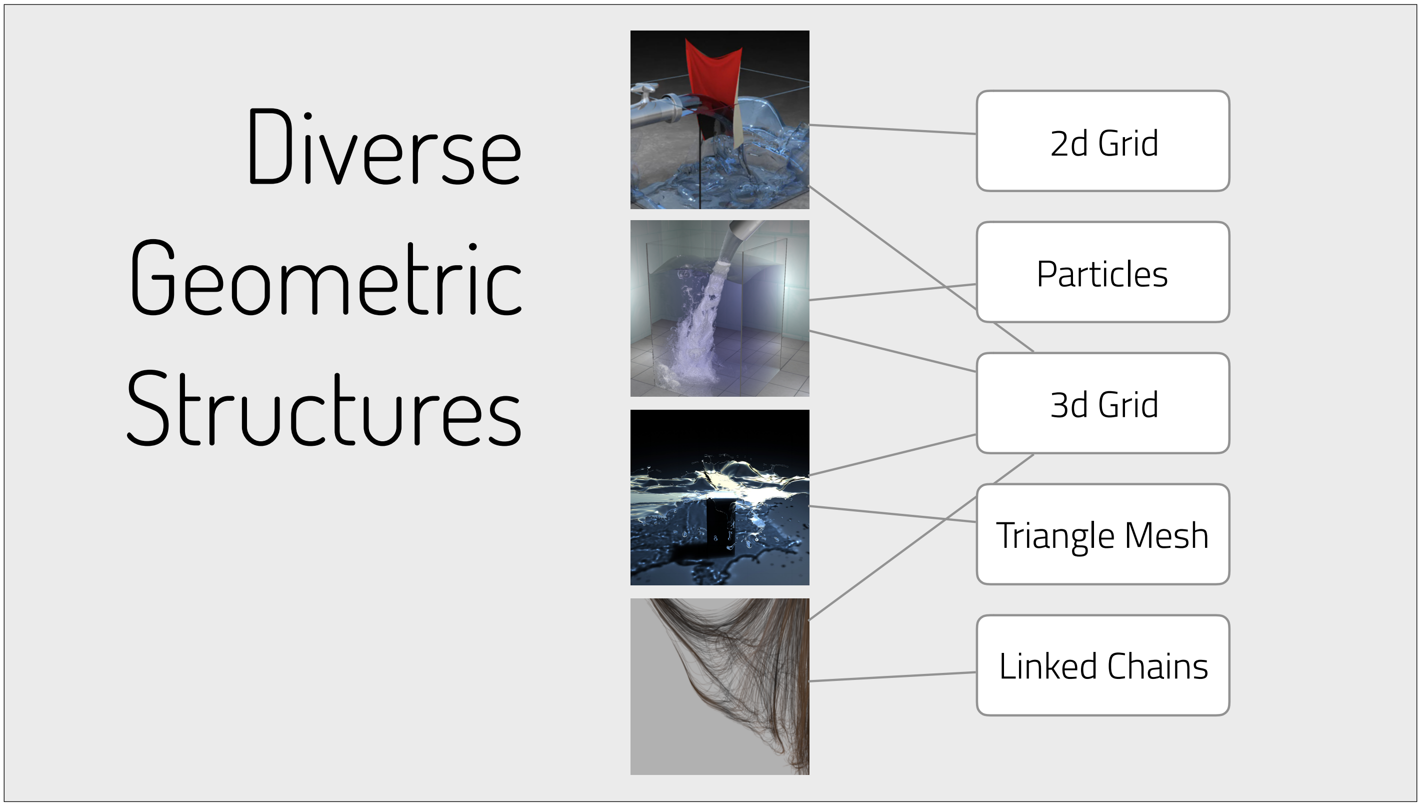
Simulations use Diverse **Geometric Structures**

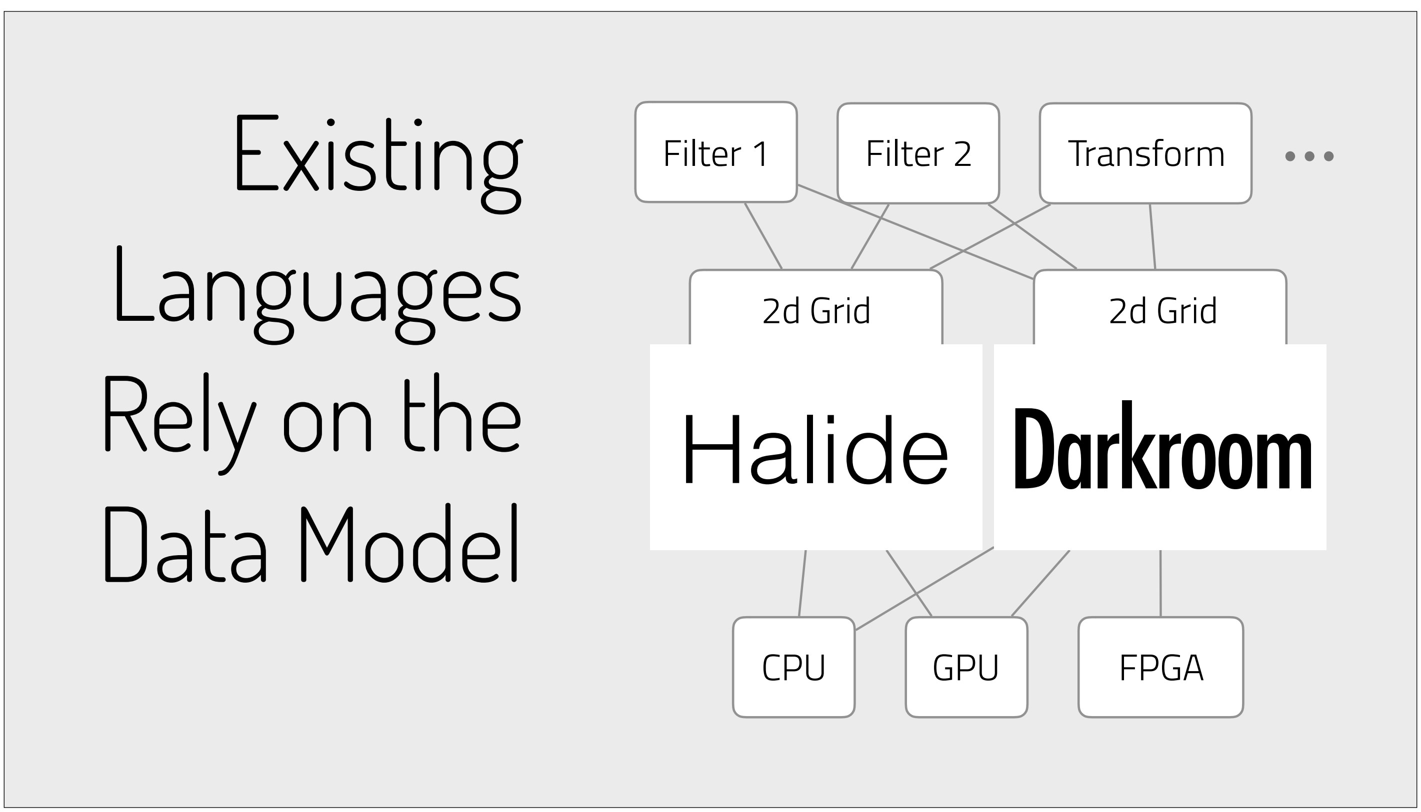


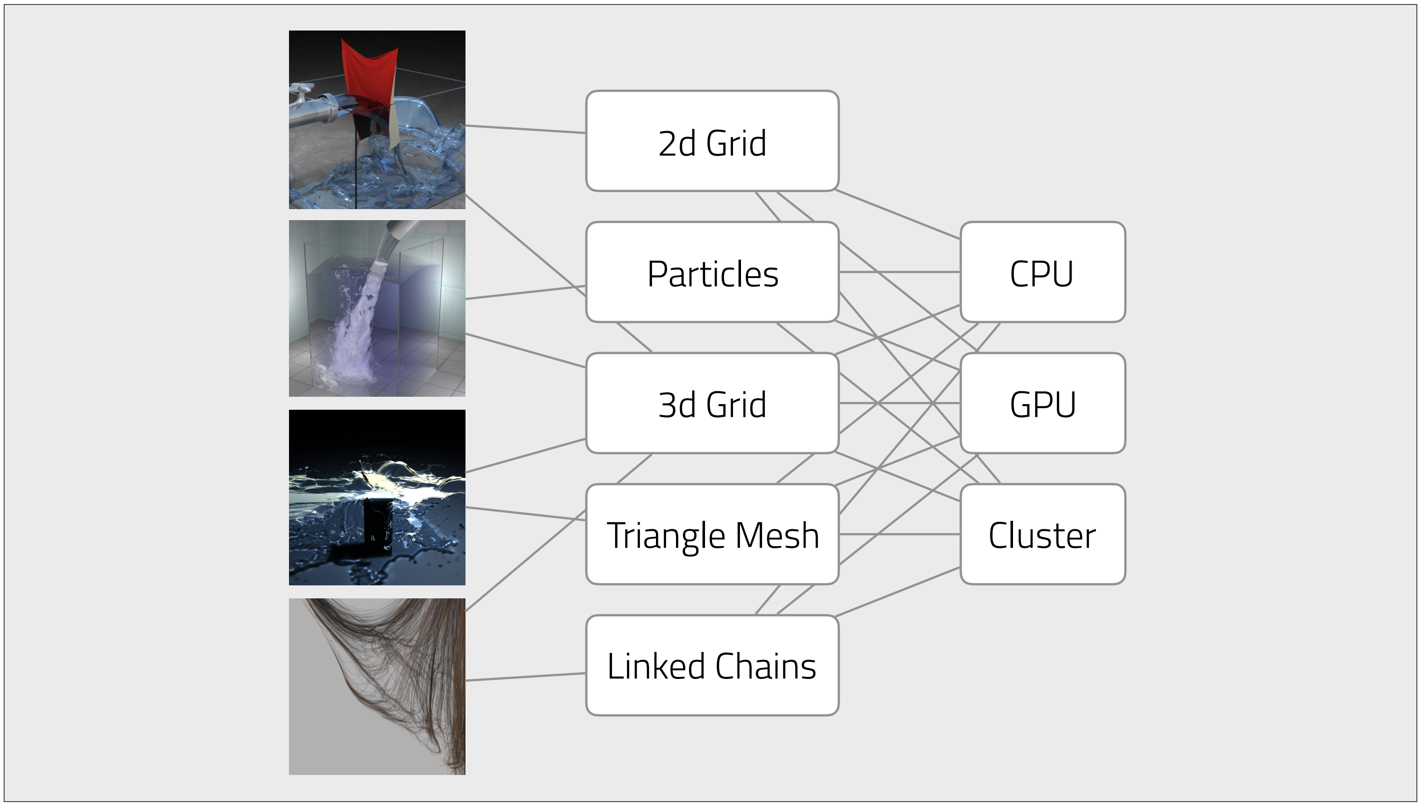
images © Ron Fedkiw, Doug James, Chris Wojtan,
Rahul Narain, Andrew Selle

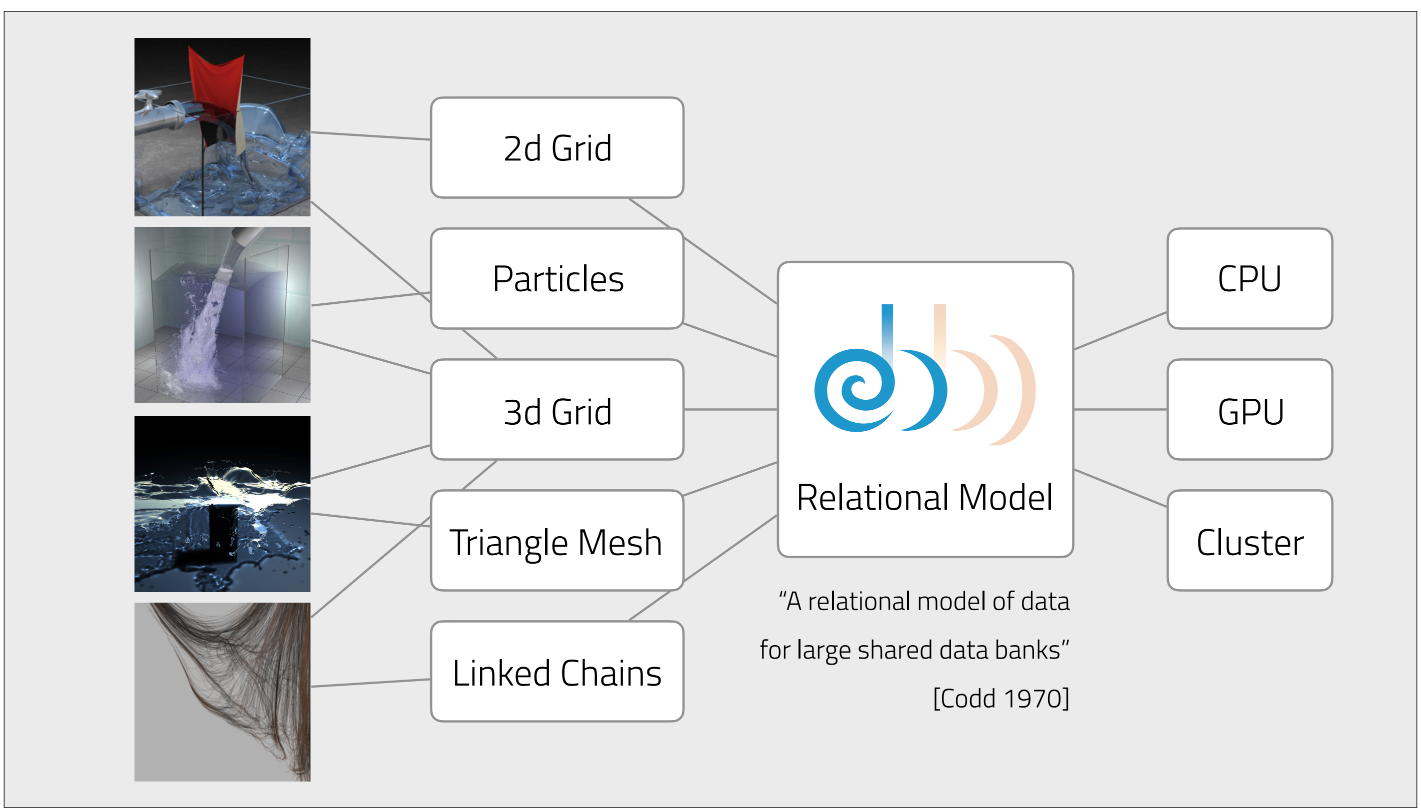

Diverse
Geometric
Structures

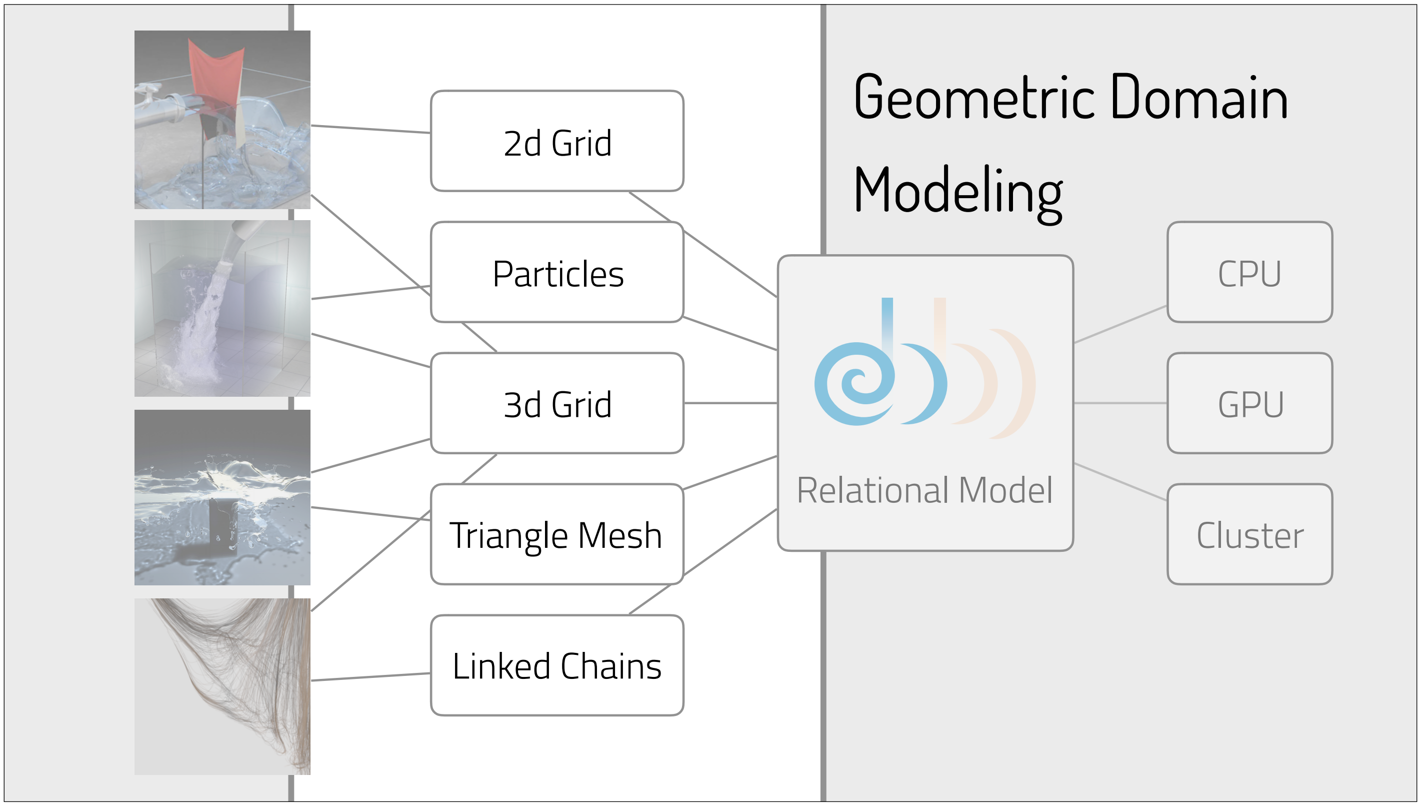

Diverse
Geometric
Structures
2d Grid
Particles
Triangle Mesh
Linked Chains


Diverse Geometric Structures
2d Grid
Particles
3d Grid
Triangle Mesh
Linked Chains

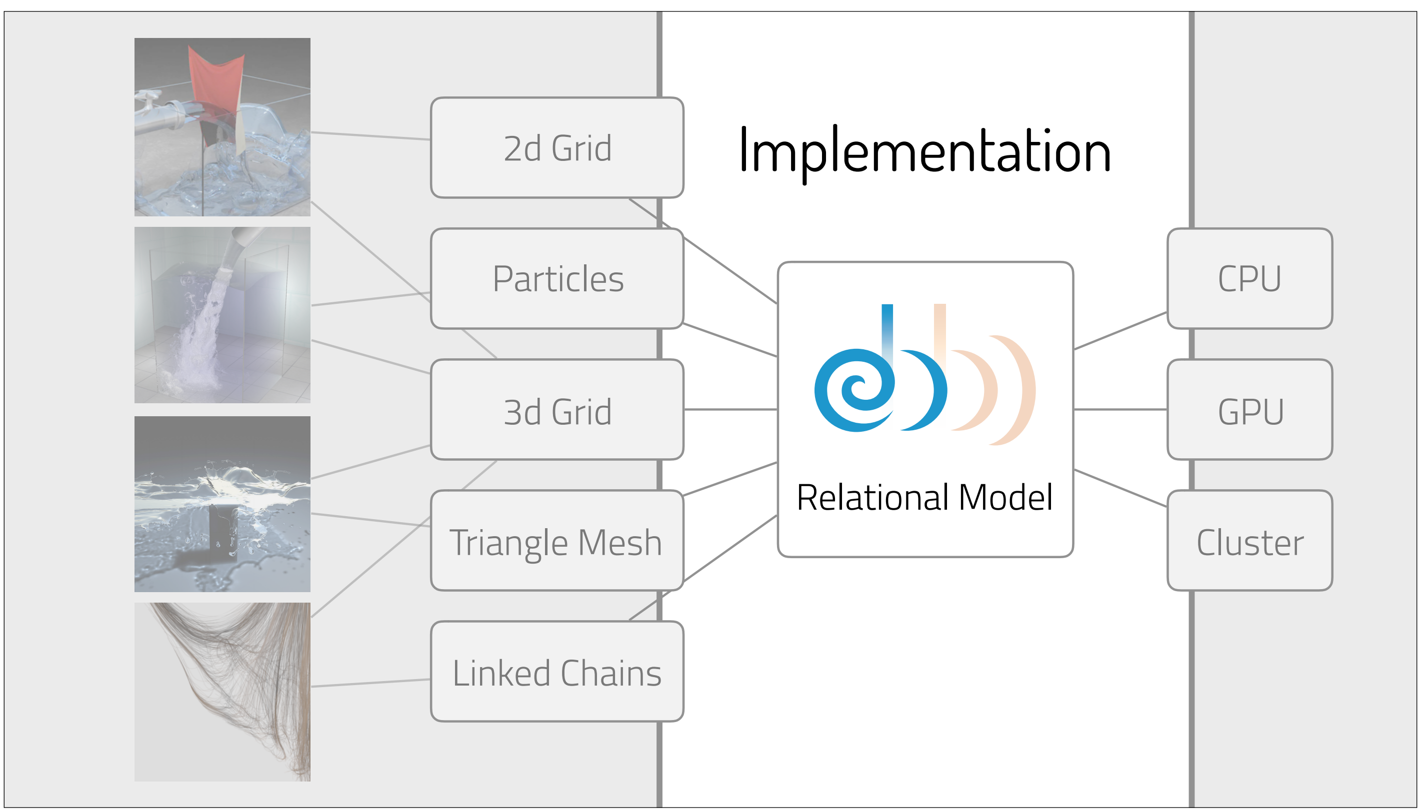

Existing Languages Rely on the Data Model
Filter 1
Filter 2
Transform
...
2d Grid
2d Grid
Halide
Darkroom
CPU
GPU
FPGA


2d Grid
Particles
3d Grid
Triangle Mesh
Linked Chains
CPU
GPU
Cluster


2d Grid
Particles
3d Grid
Triangle Mesh
Linked Chains
Relational Model
"A relational model of data
for large shared data banks"
[Codd 1970]
CPU
GPU
Cluster



Geometric Domain Modeling
2d Grid
Particles
3d Grid
Triangle Mesh
Linked Chains
Relational Model
CPU
GPU
Cluster



Implementation
2d Grid
Particles
3d Grid
Triangle Mesh
Linked Chains
Relational Model
CPU
GPU
Cluster

Example

Code

```
import "ebb"
local L = require "ebblib"
```

Import the language into a Lua Script

```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.domains.TetLib'
local dragon = TetLib.LoadTetmesh('dragon.veg')
```

Load the dragon mesh using the Tetrahedral Mesh Library

```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.domains.TetLib'
local dragon = TetLib.LoadTetmesh('dragon.veg')

local dt     = L.Constant(L.double, 0.0002)
local K      = L.Constant(L.double, 4.0)
local maxvel = L.Global(L.double, 0)
```

Setup Simulation Constant Values & Global Values

```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.domains.TetLib'
local dragon = TetLib.LoadTetmesh('dragon.veg')

local dt     = L.Constant(L.double, 0.0002)
local K      = L.Constant(L.double, 4.0)
local maxvel = L.Global(L.double, 0)

dragon.vertices:NewField('vel', L.vec3d):Load({0,0,0})
dragon.vertices:NewField('nxt_vel', L.vec3d):Load(...)
dragon.vertices:NewField('nxt_pos', L.vec3d):Load(...)
dragon.edges:NewField('rest_len', L.double):Load(0)
```

Declare & Initialize

Fields of data

```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.domains.TetLib'
local dragon = TetLib.LoadTetmesh('dragon.veg')

local dt     = L.Constant(L.double, 0.0002)
local K      = L.Constant(L.double, 4.0)
local maxvel = L.Global(L.double, 0)

dragon.vertices:NewField('vel', L.vec3d):Load({0,0,0})
dragon.vertices:NewField('nxt_vel', L.vec3d):Load(...)
dragon.vertices:NewField('nxt_pos', L.vec3d):Load(...)
dragon.edges:NewField('rest_len', L.double):Load(0)

local ebb init_rest_len ( e : dragon.edges )
  var diff = e.head.pos - e.tail.pos
  e.rest_len = L.length(diff)
end
```

Ebb functions

define per-element

computations

```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.domains.TetLib'
local dragon = TetLib.LoadTetmesh('dragon.veg')

local dt     = L.Constant(L.double, 0.0002)
local K      = L.Constant(L.double, 4.0)
local maxvel = L.Global(L.double, 0)

dragon.vertices:NewField('vel', L.vec3d):Load({0,0,0})
dragon.vertices:NewField('nxt_vel', L.vec3d):Load(...)
dragon.vertices:NewField('nxt_pos', L.vec3d):Load(...)
dragon.edges:NewField('rest_len', L.double):Load(0)

local ebb init_rest_len ( e : dragon.edges )
  var diff = e.head.pos - e.tail.pos
  e.rest_len = L.length(diff)
end
```

Read

Write

Ebb understands how fields are accessed

```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.domains.TetLib'
local dragon = TetLib.LoadTetmesh('dragon.veg')

local dt     = L.Constant(L.double, 0.0002)
local K      = L.Constant(L.double, 4.0)
local maxvel = L.Global(L.double, 0)

dragon.vertices:NewField('vel', L.vec3d):Load({0,0,0})
dragon.vertices:NewField('nxt_vel', L.vec3d):Load(...)
dragon.vertices:NewField('nxt_pos', L.vec3d):Load(...)
dragon.edges:NewField('rest_len', L.double):Load(0)

local ebb init_rest_len ( e : dragon.edges )
  var diff = e.head.pos - e.tail.pos
  e.rest_len = L.length(diff)
end
dragon.edges:foreach(init_rest_len)

-- initialize vel and acc --
```

Functions launch over sets of elements

```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.domains.TetLib'
local dragon = TetLib.LoadTetmesh('dragon.veg')

local dt     = L.Constant(L.double, 0.0002)
local K      = L.Constant(L.double, 4.0)
local maxvel = L.Global(L.double, 0)

dragon.vertices:NewField('vel', L.vec3d):Load({0,0,0})
dragon.vertices:NewField('nxt_vel', L.vec3d):Load(...)
dragon.vertices:NewField('nxt_pos', L.vec3d):Load(...)
dragon.edges:NewField('rest_len', L.double):Load(0)

local ebb init_rest_len ( e : dragon.edges )
  var diff = e.head.pos - e.tail.pos
  e.rest_len = L.length(diff)
end
dragon.edges:foreach(init_rest_len)
-- initialize vel and acc --
```

```
local ebb compute_max_vel ( v : dragon.vertices )
  maxvel max= L.length(v.vel)
end
```

Functions can reduce global values


```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.do
local dragon = TetLib.LoadTetm

local dt     = L.Constant(L.do
local K      = L.Constant(L.do
local maxvel = L.Global(L.doub

dragon.vertices:NewField('vel'
dragon.vertices:NewField('nxt_
dragon.vertices:NewField('nxt_
dragon.edges:NewField('rest_le

local ebb init_rest_len ( e :
  var diff = e.head.pos - e.ta
  e.rest_len = L.length(diff)
end
dragon.edges:foreach(init_rest
-- initialize vel and acc --

local ebb compute_max_vel ( v
  maxvel max= L.length(v.vel)
end
```


```
local ebb compute_acc ( v : dragon.vertices )
  var force = { 0.0, 0.0, 0.0 }

  -- Spring Force
  var mass = 0.0
  for e in v.edges do
    var diff  = e.head.pos - v.pos
    var scale = (e.rest_len / L.length(diff)) - 1.0
    mass     += e.rest_len
    force    -= K * scale * diff
  end

  v.nxt_pos = v.pos + dt        * v.vel
                    + 0.5*dt*dt * force/mass
  v.nxt_vel = v.vel + dt        * force/mass
end
```

```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.do
local dragon = TetLib.LoadTetm

local dt     = L.Constant(L.do
local K      = L.Constant(L.do
local maxvel = L.Global(L.doub

dragon.vertices:NewField('vel'
dragon.vertices:NewField('nxt_
dragon.vertices:NewField('nxt_
dragon.edges:NewField('rest_le

local ebb init_rest_len ( e :
  var diff = e.head.pos - e.ta
  e.rest_len = L.length(diff)
end
dragon.edges:foreach(init_rest
-- initialize vel and acc --

local ebb compute_max_vel ( v
  maxvel max= L.length(v.vel)
end
```

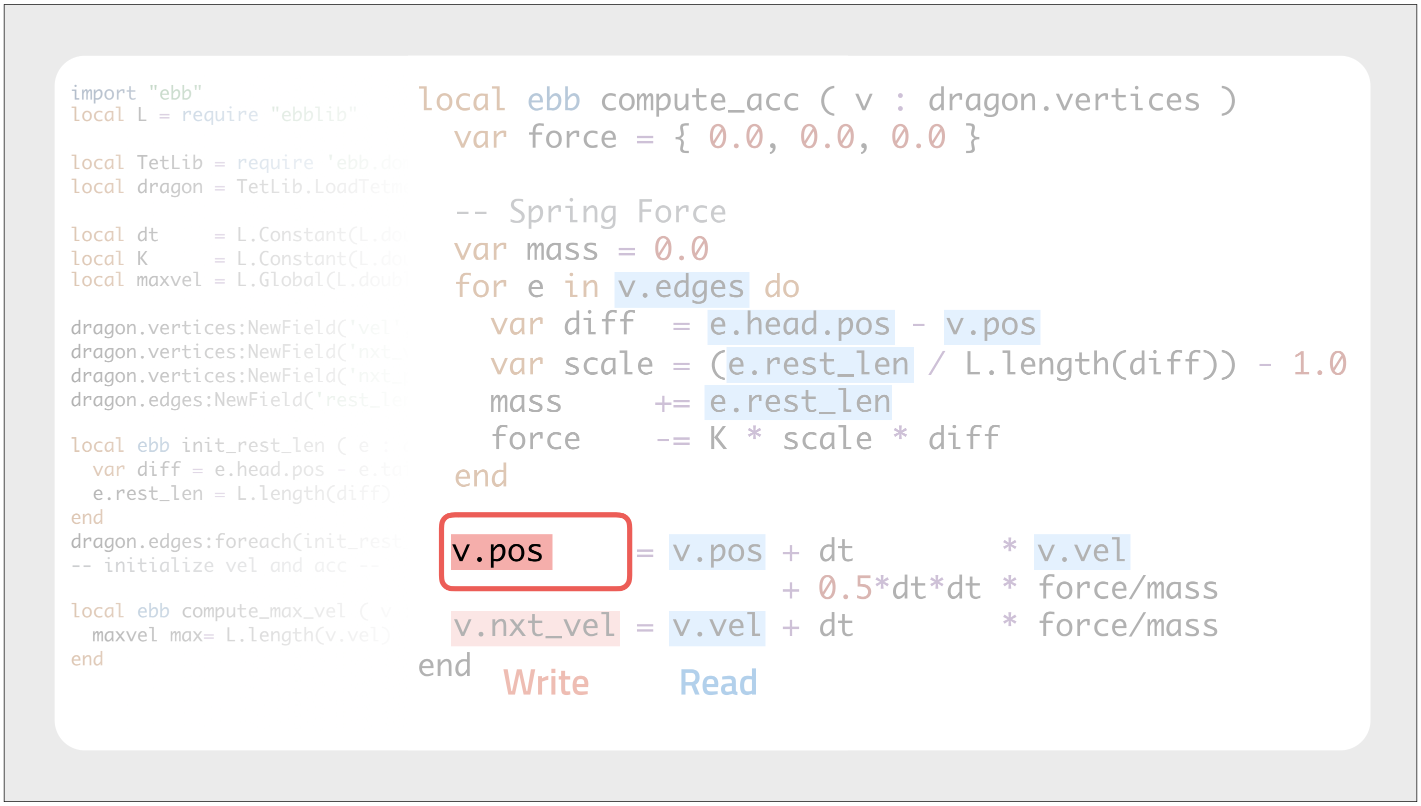
```
local ebb compute_acc ( v : dragon.vertices )
  var force = { 0.0, 0.0, 0.0 }

  -- Spring Force
  var mass = 0.0
  for e in v.edges do
    var diff  = e.head.pos - v.pos
    var scale = (e.rest_len / L.length(diff)) - 1.0
    mass     += e.rest_len
    force    -= K * scale * diff
  end

  v.nxt_pos = v.pos + dt          * v.vel
                    + 0.5*dt*dt * force/mass
  v.nxt_vel = v.vel + dt          * force/mass
end
```

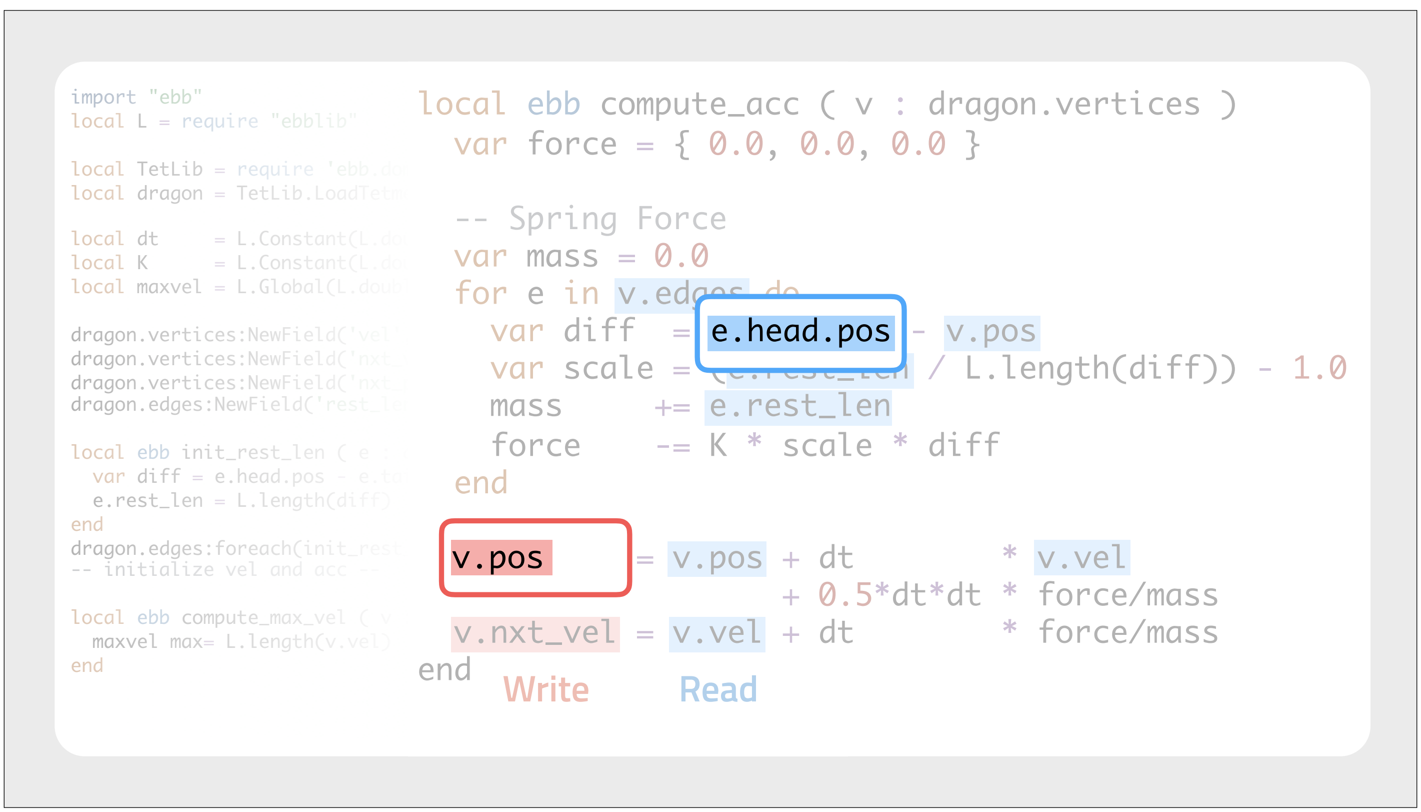
Write Read


```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.do
local dragon = TetLib.LoadTetm

local dt     = L.Constant(L.do
local K      = L.Constant(L.do
local maxvel = L.Global(L.doub

dragon.vertices:NewField('vel'
dragon.vertices:NewField('nxt_
dragon.vertices:NewField('nxt_
dragon.edges:NewField('rest_le

local ebb init_rest_len ( e :
  var diff = e.head.pos - e.ta
  e.rest_len = L.length(diff)
end
dragon.edges:foreach(init_rest
-- initialize vel and acc --

local ebb compute_max_vel ( v
  maxvel max= L.length(v.vel)
end
```

```
local ebb compute_acc ( v : dragon.vertices )
  var force = { 0.0, 0.0, 0.0 }

  -- Spring Force
  var mass = 0.0
  for e in v.edges do
    var diff  = e.head.pos - v.pos
    var scale = (e.rest_len / L.length(diff)) - 1.0
    mass     += e.rest_len
    force    -= K * scale * diff
  end

  v.nxt_pos = v.pos + dt          * v.vel
                    + 0.5*dt*dt * force/mass
  v.nxt_vel = v.vel + dt          * force/mass
end
```

Write Read


import "ebb"
local L = require "ebblib"
local TetLib = require 'ebb.do
local dragon = TetLib.LoadTetm
local dt = L.Constant(L.dou
local K = L.Constant(L.dou
local maxvel = L.Global(L.doub
dragon.vertices:NewField('vel
dragon.vertices:NewField('nxt_
dragon.vertices:NewField('nxt_
dragon.edges:NewField('rest_le
local ebb init_rest_len (e :
var diff = e.head.pos - e.ta
e.rest_len = L.length(diff)
end
dragon.edges:foreach(init_rest
-- initialize vel and acc --
local ebb compute_max_vel (v
maxvel max= L.length(v.vel)
end
local ebb compute_acc (v : dragon.vertices)
var force = { 0.0, 0.0, 0.0 }
-- Spring Force
var mass = 0.0
for e in v.edges do
var diff = e.head.pos - v.pos
var scale = (e.rest_len / L.length(diff)) - 1.0
mass += e.rest_len
force -= K * scale * diff
end
v.pos = v.pos + dt * v.vel
+ 0.5*dt*dt * force/mass
v.nxt_vel = v.vel + dt * force/mass
end
Write
Read


import "ebb"
local L = require "ebblib"
local TetLib = require 'ebb.do
local dragon = TetLib.LoadTetm
local dt = L.Constant(L.dou
local K = L.Constant(L.dou
local maxvel = L.Global(L.doub
dragon.vertices:NewField('vel
dragon.vertices:NewField('nxt_
dragon.vertices:NewField('nxt_
dragon.edges:NewField('rest_le
local ebb init_rest_len (e :
var diff = e.head.pos - e.ta
e.rest_len = L.length(diff)
end
dragon.edges:foreach(init_rest
-- initialize vel and acc --
local ebb compute_max_vel (v
maxvel max= L.length(v.vel)
end
local ebb compute_acc (v : dragon.vertices)
var force = { 0.0, 0.0, 0.0 }
-- Spring Force
var mass = 0.0
for e in v.edges do
var diff = e.head.pos - v.pos
var scale = (e.rest_len / L.length(diff)) - 1.0
mass += e.rest_len
force -= K * scale * diff
end
v.pos = v.pos + dt * v.vel
+ 0.5*dt*dt * force/mass
v.nxt_vel = v.vel + dt * force/mass
end
Write
Read

```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.domains.TetLib'
local dragon = TetLib.LoadTetmesh('dragon.veg')

local dt     = L.Constant(L.double, 0.0002)
local K      = L.Constant(L.double, 4.0)
local maxvel = L.Global(L.double, 0)

dragon.vertices:NewField('vel', L.vec3d):Load({0,0,0})
dragon.vertices:NewField('nxt_vel', L.vec3d):Load(...)
dragon.vertices:NewField('nxt_pos', L.vec3d):Load(...)
dragon.edges:NewField('rest_len', L.double):Load(0)

local ebb init_rest_len ( e : dragon
  var diff = e.head.pos - e.tail.pos
  e.rest_len = L.length(diff)
end
dragon.edges:foreach(init_rest_len)
-- initialize vel and acc --

local ebb compute_max_vel ( v : drag
  maxvel max= L.length(v.vel)
end
```

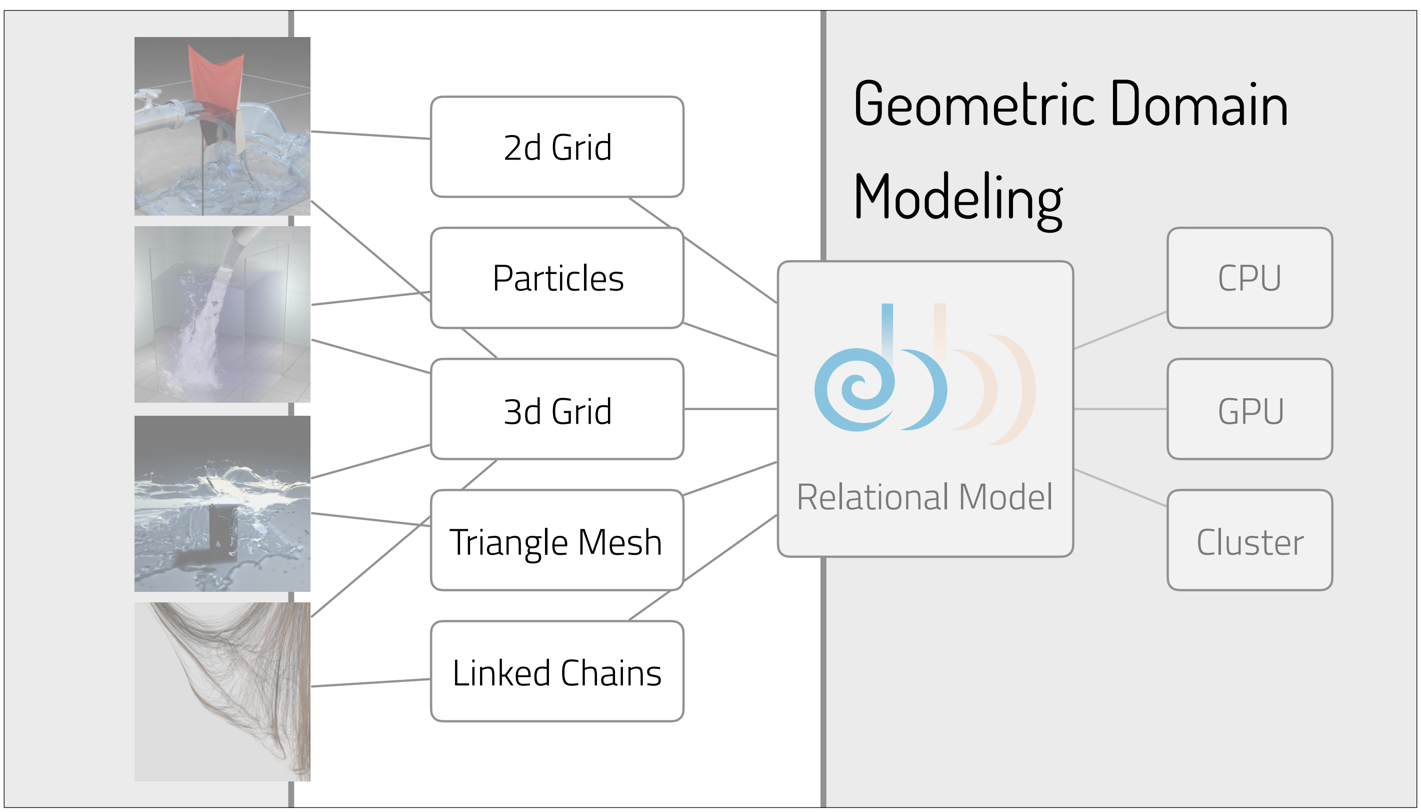
```
local ebb compute_acc ( v : dragon.vertices )
  var force = { 0.0, 0.0, 0.0 }

  -- Spring Force
  var mass = 0.0
  for e in v.edges do
    var diff  = e.head.pos - v.pos
    var scale = (e.rest_len / L.length(diff)) - 1.0
    mass     += e.rest_len
    force    -= K * scale * diff
  end

  v.nxt_pos = v.pos + dt         * v.vel
                    + 0.5*dt*dt  * force/mass
  v.nxt_vel = v.vel + dt         * force/mass
end
```

```
-- Sim Loop
for i=1,40000 do
  dragon.vertices:foreach(compute_acc)
  dragon.vertices:swap('pos', 'nxt_pos')
  dragon.vertices:swap('vel', 'nxt_vel')
end
```

```
...
local particles = L.NewRelation {
  name = 'particles', size = M,
}

particles:NewField('d_cell',grid.dual_cells):Load(...)
particles:NewField('pos', L.vec3f):Load(...)
particles:NewField('vel', L.vec3f):Load(…)

local ebb update_particle_vel ( p : particles )
  var x1 = fmod( p.pos[0] - 0.5f )
  var y1 = fmod( p.pos[1] - 0.5f )
  var x0 = 1.0f - x1
  var y0 = 1.0f - y1

  p.vel = x0 * y0 * p.dual_cell.cell(0,0).vel
        + x1 * y0 * p.dual_cell.cell(1,0).vel
        + x0 * y1 * p.dual_cell.cell(0,1).vel
        + x1 * y1 * p.dual_cell.cell(1,1).vel
end

...

for i=1,10000 do
  update_particle_vel(particles)
  update_particle_pos(particles)
   grid.dual_cells:point_locate(particles.dual_cell,
                                particles.pos)
end
```

More examples at:

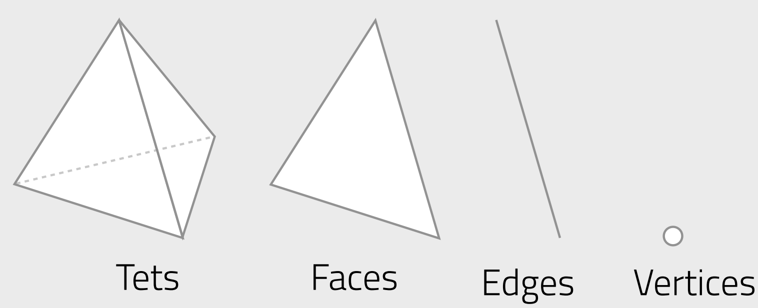
http:// ebblang.org



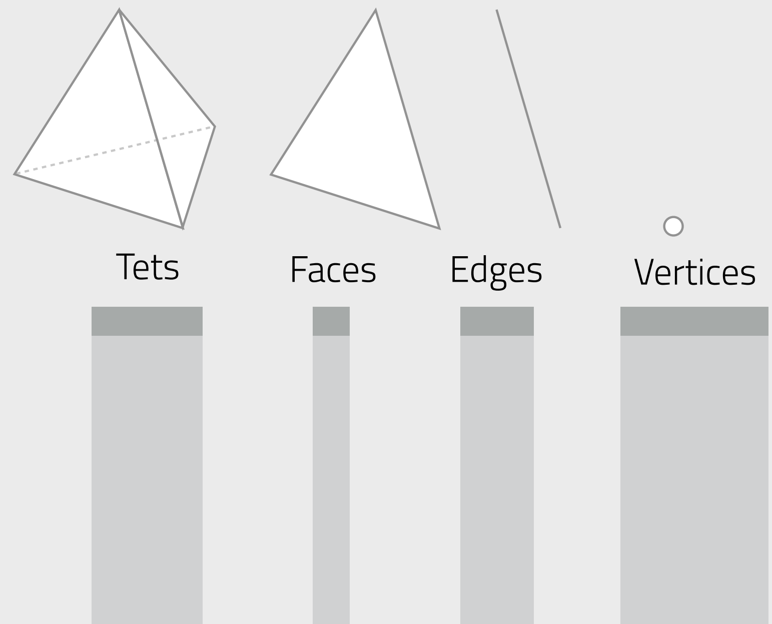
```
import "ebb"
local L = require "ebblib"

local TetLib = require 'ebb.domains.TetLib'
local dragon = TetLib.LoadTetmesh('dragon.veg')
```

How was the Tetrahedral Mesh Library Implemented?

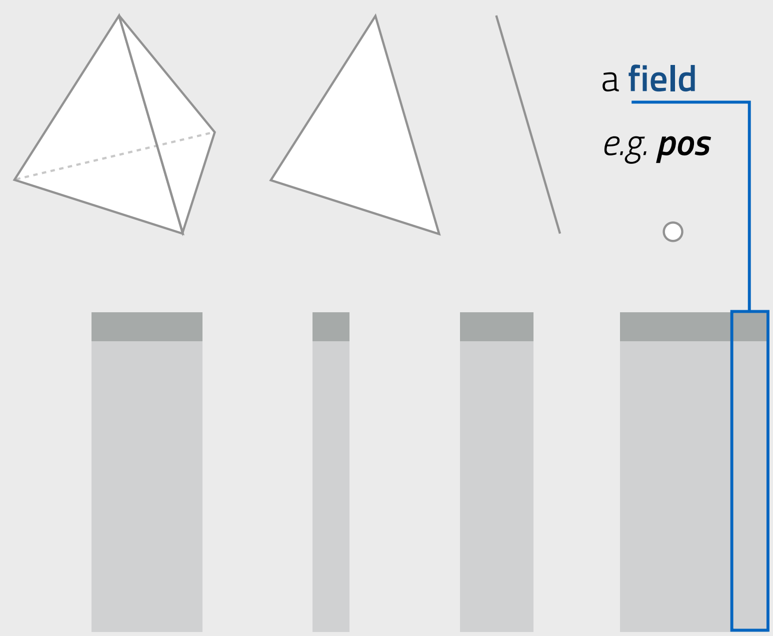
Modeling a TetMesh with Relations



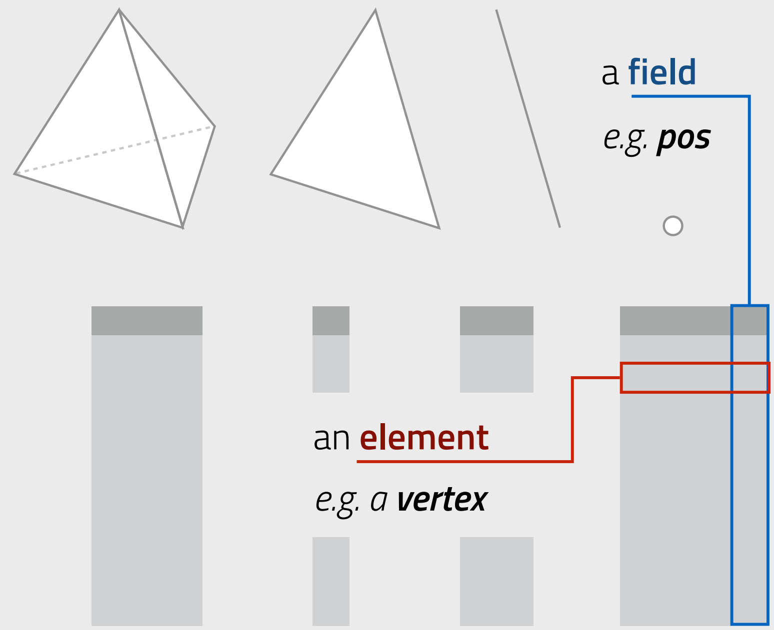
Modeling a TetMesh with Relations



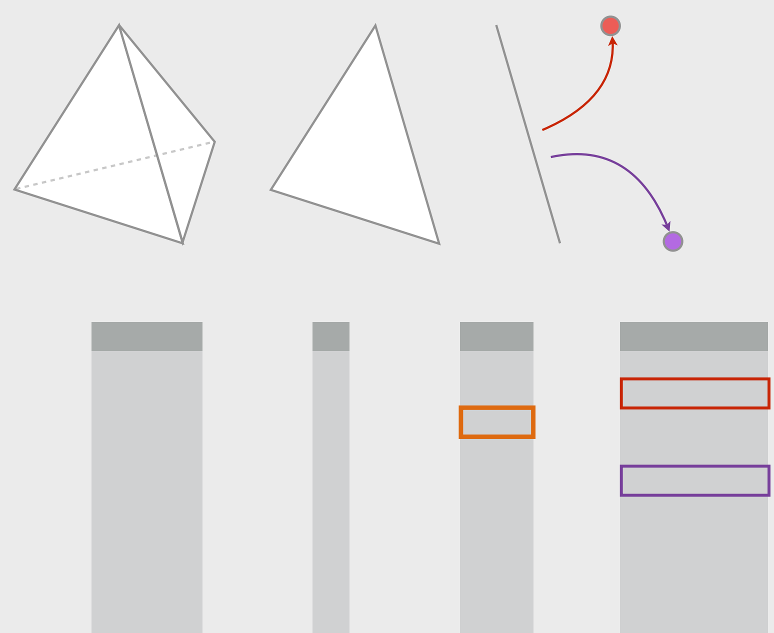
Modeling a TetMesh with Relations



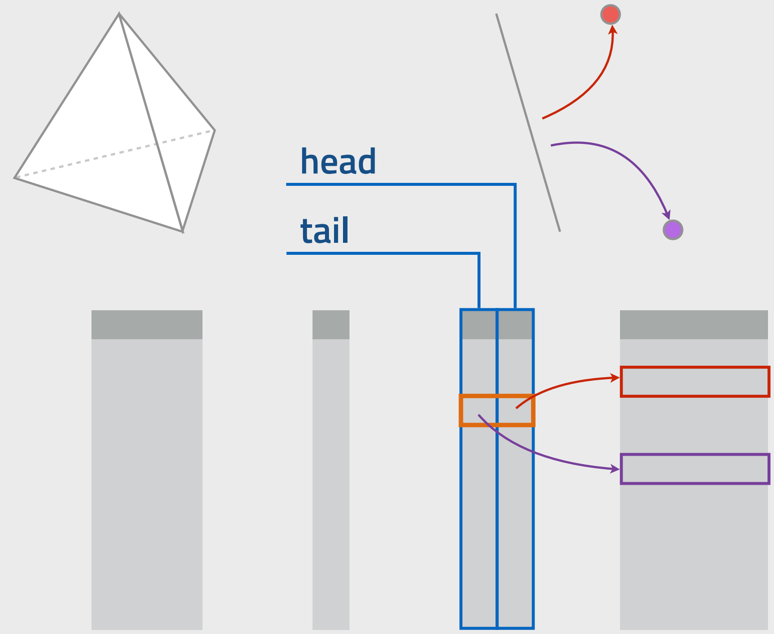
Modeling a TetMesh with Relations



Topology: Connecting Elements

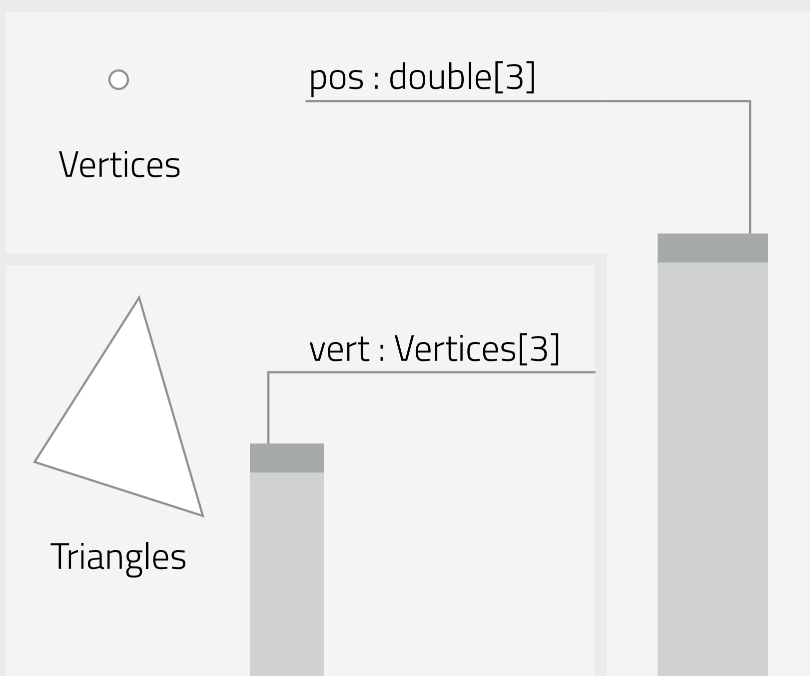


Topology: Connecting Elements



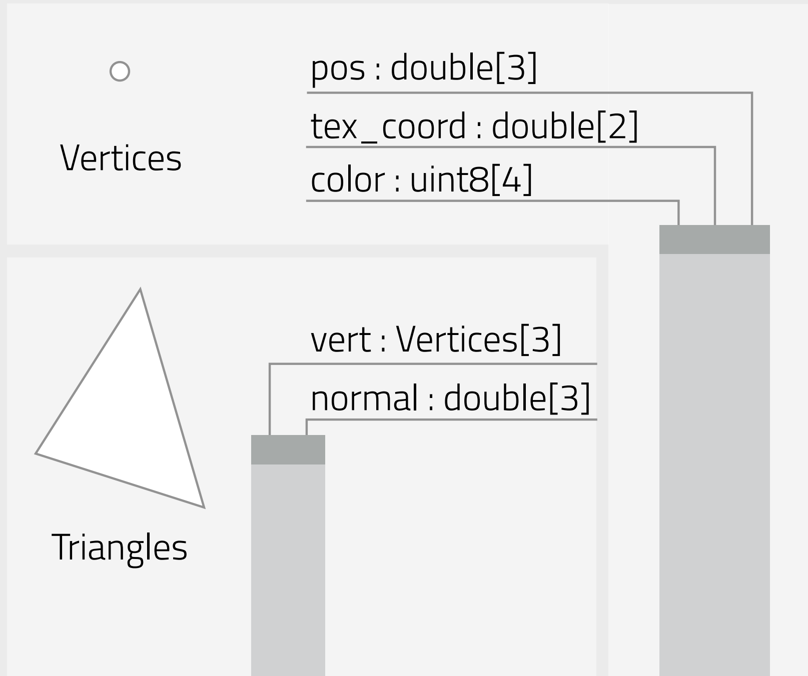
example

OpenGL Pipeline Input



example

OpenGL Pipeline Input



Indices ?

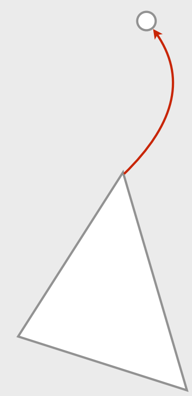
Triangles.vert : **GLuint**[3]

Pointers ?

Triangles.vert : **Vert***[3]

Keys ?

Triangles.vert : **Vertices**[3]



Key-Fields

```
ebb foo( e : edges )
  var diff = e.head.pos - e.tail.pos
  ...
```

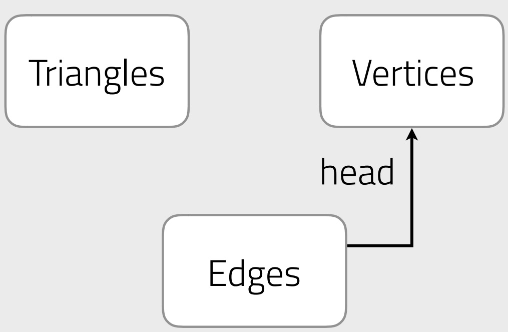
Key-Fields

```
ebb foo( e : edges )
  var diff = e.head.pos - e.tail.pos
  ...
```

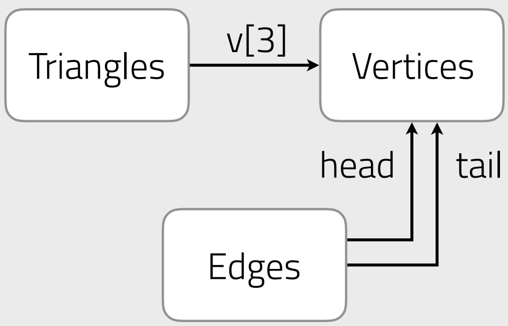
```
edges:NewField('head', vertices)
```

```
edges )
e.head.pos - e.tail.pos
```


```
edges:NewField('head', vertices)
```

```
edges )
e.head.pos - e.tail.pos
```




```
edges:NewField('head', vertices)
```

Key-Fields

```
ebb foo( e : edges )
  var diff = e.head.pos - e.tail.pos
  ...
```

Query-Loops

```
ebb bar( v : vertices )
  for e in L.Where(edges.tail, v) do
    v.sum_t += e.head.t
    ...
```

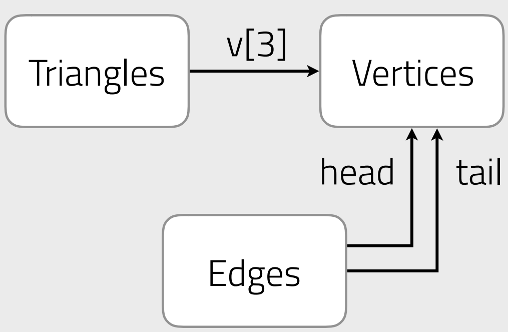
Key-Fields

```
ebb foo( e : edges )
  var diff = e.head.pos - e.tail.pos
  ...
```

Query-Loops

```
ebb bar( v : vertices )
  for e in L.Where(edges.tail, v) do
    v.sum_t += e.head.t
    ...
```

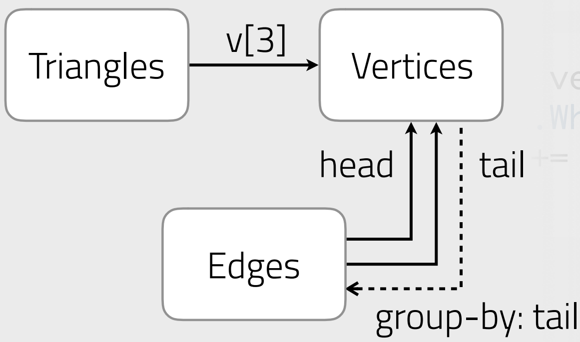
```
edges:GroupBy('tail')
```




```
edges )
e.head.pos - e.tail.pos
```

```
vertices )
.Where(edges.tail, v) do
+= e.head.t
```

```
edges:GroupBy('tail')
```




```
edges )
e.head.pos - e.tail.pos
```

```
vertices )
.Where(edges.tail, v) do
+= e.head.t
```

```
edges:GroupBy('tail')
```

Key-Fields

```
ebb foo( e : edges )
  var diff = e.head.pos - e.tail.pos
  ...
```

Query-Loops

```
ebb bar( v : vertices )
  for e in v.edges do
    v.sum_t += e.head.t
    ...
```

Affine-Indices

```
ebb baz( c : cells )
  c.sum_p = c(1,0).p
          + c(0,1).p
    ...
```

```
Affine(cells, {{1,0,1},
                 {0,1,0}}, c).p +
```

Key-Fields

```
ebb foo( e : edges )
  var diff = e.head.pos - e.tail.pos
  ...
```

Query-Loops

```
ebb bar( v : vertices )
  for e in v.edges do
    v.sum_t += e.head.t
    ...
```

Affine-Indices

```
ebb baz( c : cells )
  c.sum_p = L.Affine(cells,
                      {{1,0,1},
                       {0,1,0}}, c).p +
```

$$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}c.x\\c.y\end{bmatrix}+\begin{bmatrix}1\\0\end{bmatrix}$$

Key-Fields

```
ebb foo( e : edges )
  var diff = e.head.pos - e.tail.pos
  ...
```

Query-Loops

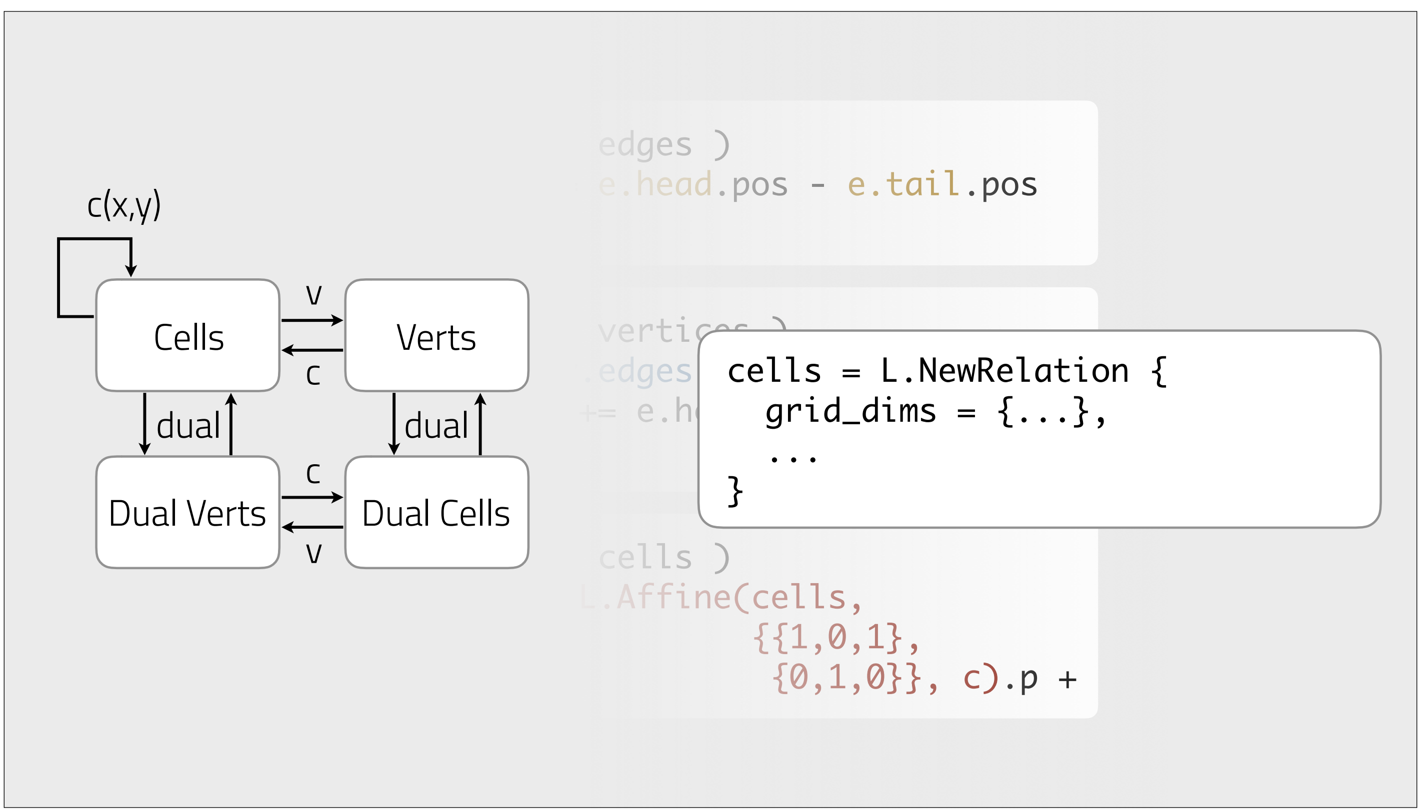
```
ebb bar( v : vertices )
  for e in v.edges
    v.sum_t += e.h
    ...
```

```
cells = L.NewRelation {
  grid_dims = {...},
  ...
}
```

Affine-Indices

```
ebb baz( c : cells )
  c.sum_p = L.Affine(cells,
                      {{1,0,1},
                       {0,1,0}}, c).p +
```


c(x,y)
Cells
edges)
e.head.pos - e.tail.pos
cells = L.NewRelation {
grid_dims = {...},
...
}
cells)
.Affine(cells,
{{1,0,1},
{0,1,0}}, c).p +


c(x,y)
Cells
v
c
Verts
dual
dual
c
v
Dual Verts
Dual Cells
edges)
e.head.pos - e.tail.pos
cells = L.NewRelation {
grid_dims = {...},
...
}
cells)
.Affine(cells,
{{1,0,1},
{0,1,0}}, c).p +

Key-Fields

```
ebb foo( e : edges )
  var diff = e.head.pos - e.tail.pos
  ...
```

Query-Loops

```
ebb bar( v : vertices )
  for e in v.edges do
    v.sum_t += e.head.t
    ...
```

Affine-Indices

```
ebb baz( c : cells )
  c.sum_p = c(1,0).p + c(-1,0).p
          + c(0,1).p + c(0,-1).p
    ...
```

Key-Fields

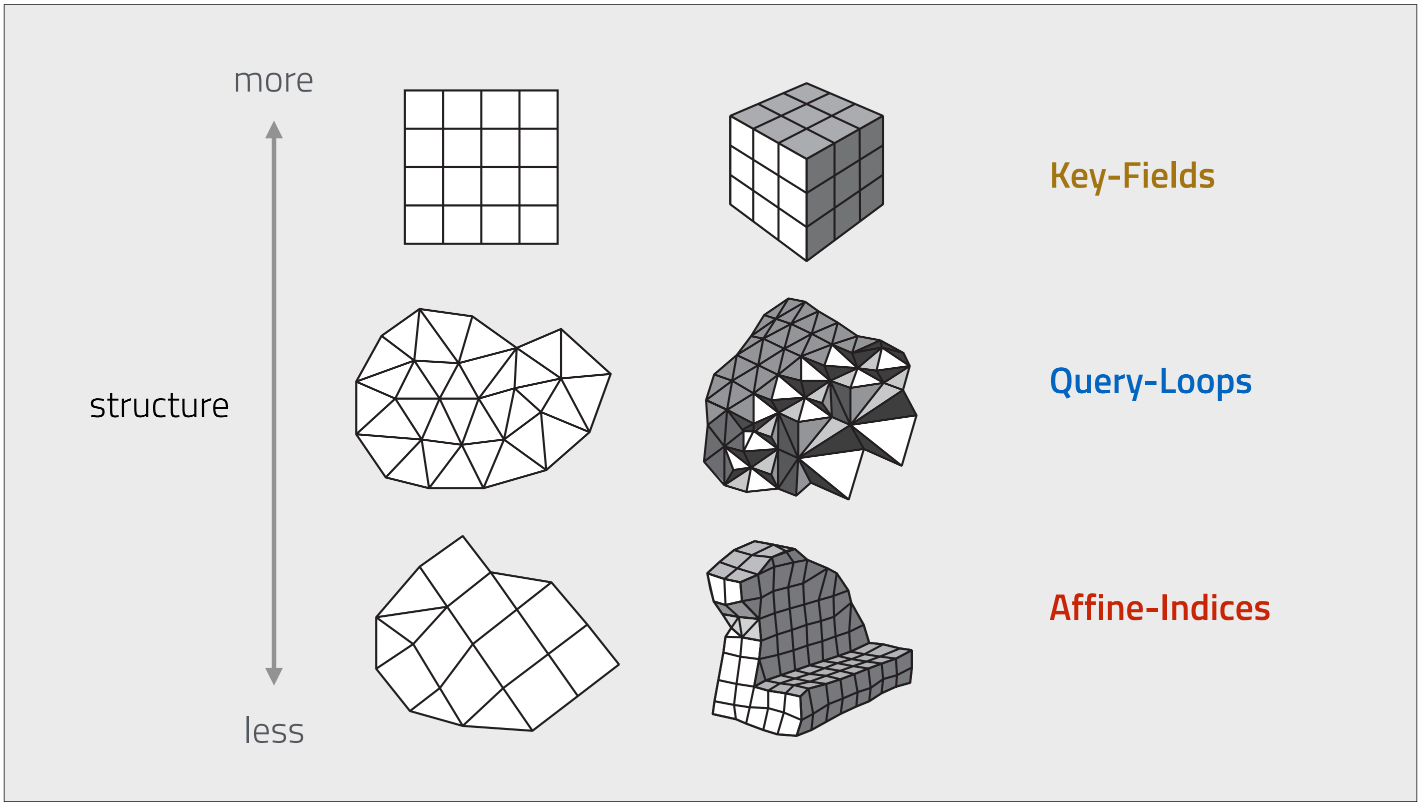
1 memory read per-access

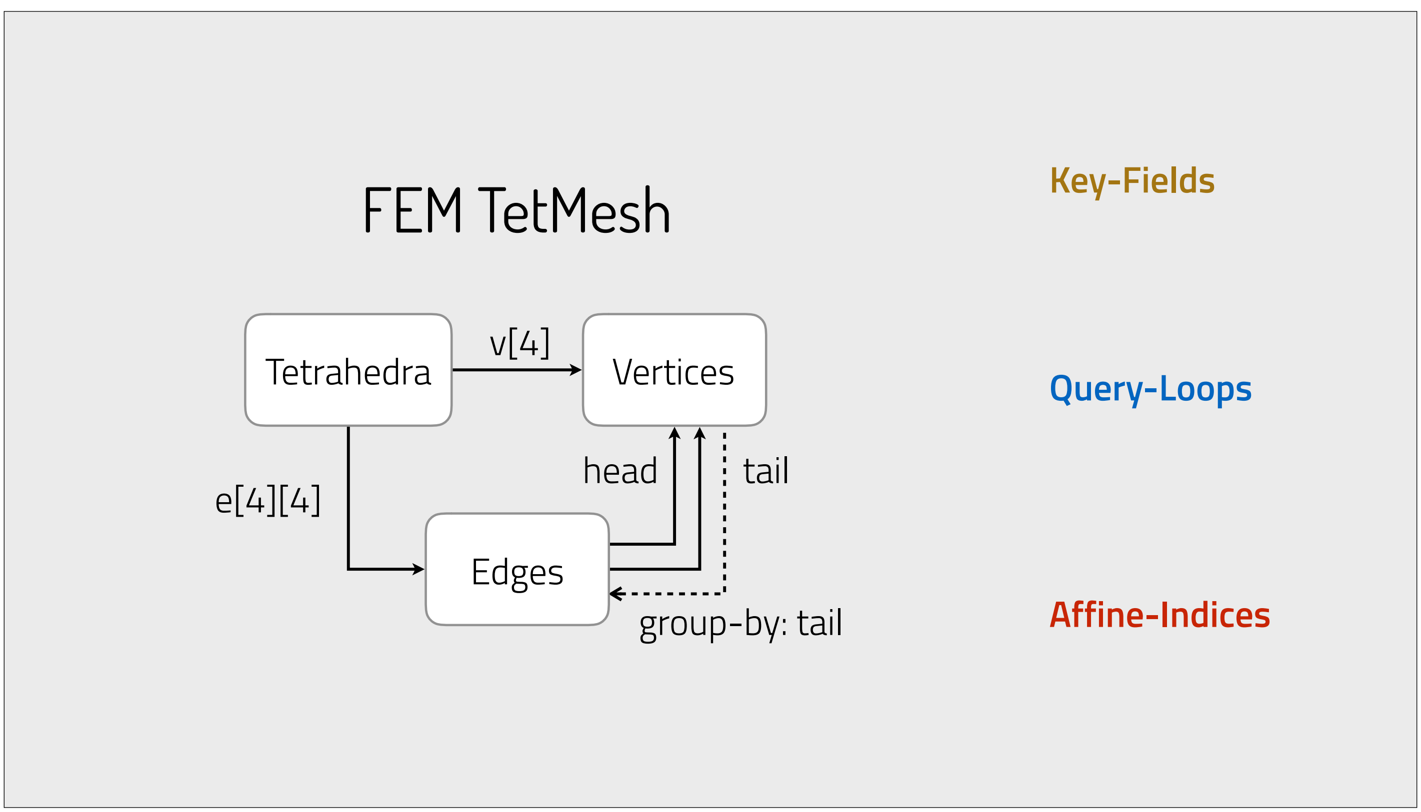
Query-Loops

2 memory reads per-loop

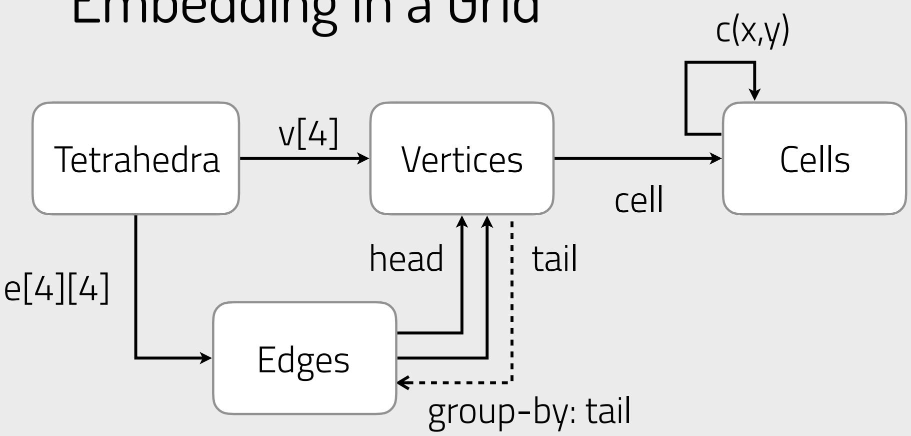
Affine-Indices

0 memory reads


more
structure
less
Key-Fields
Query-Loops
Affine-Indices


FEM TetMesh
Tetrahedra
v[4]
Vertices
e[4][4]
head
tail
Edges
group-by: tail
Key-Fields
Query-Loops
Affine-Indices

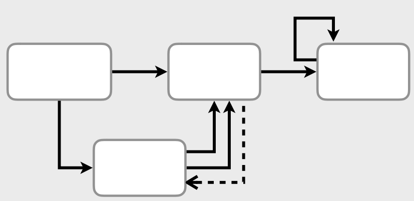
Embedding in a Grid



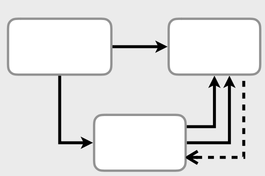
Key-Fields

Query-Loops

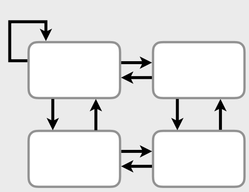
Affine-Indices



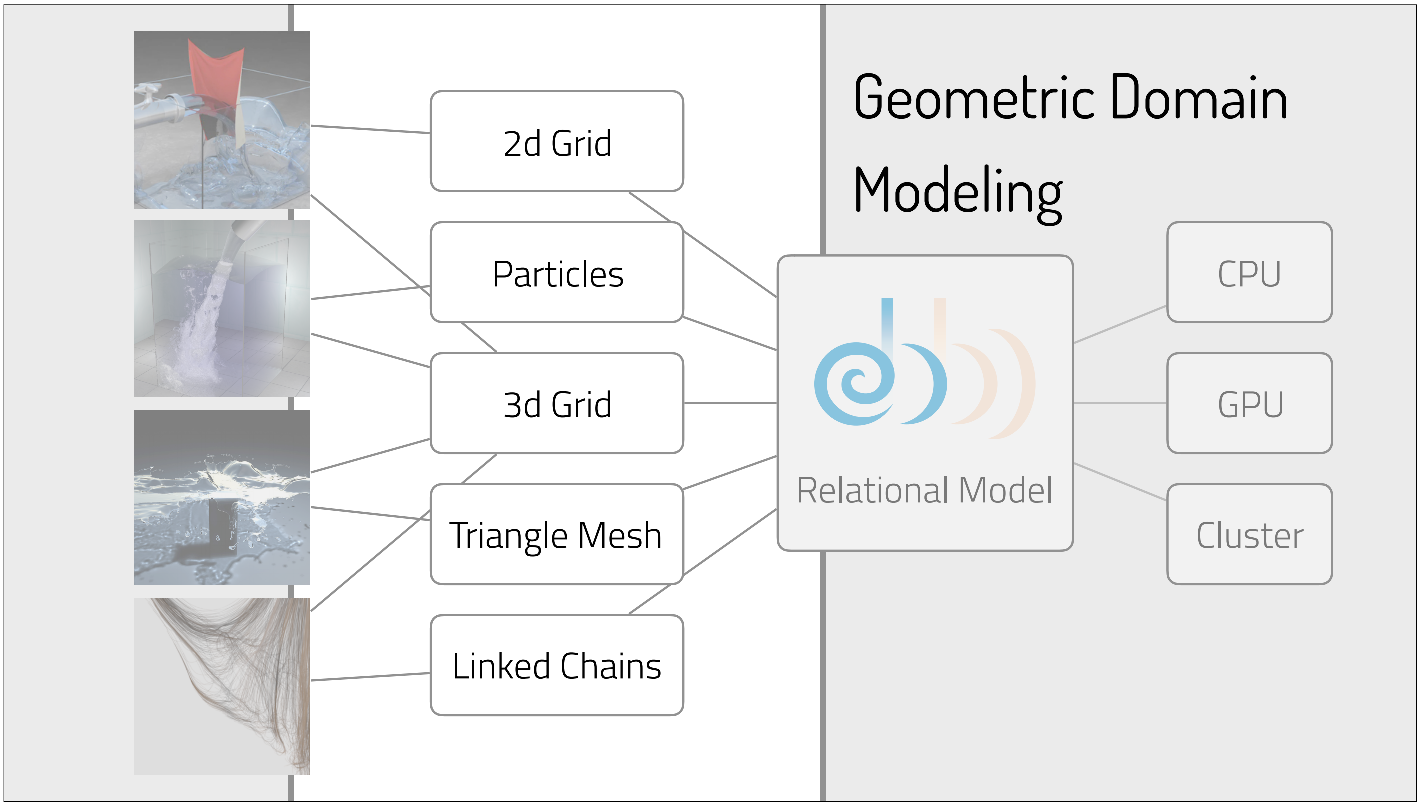
Key-Fields

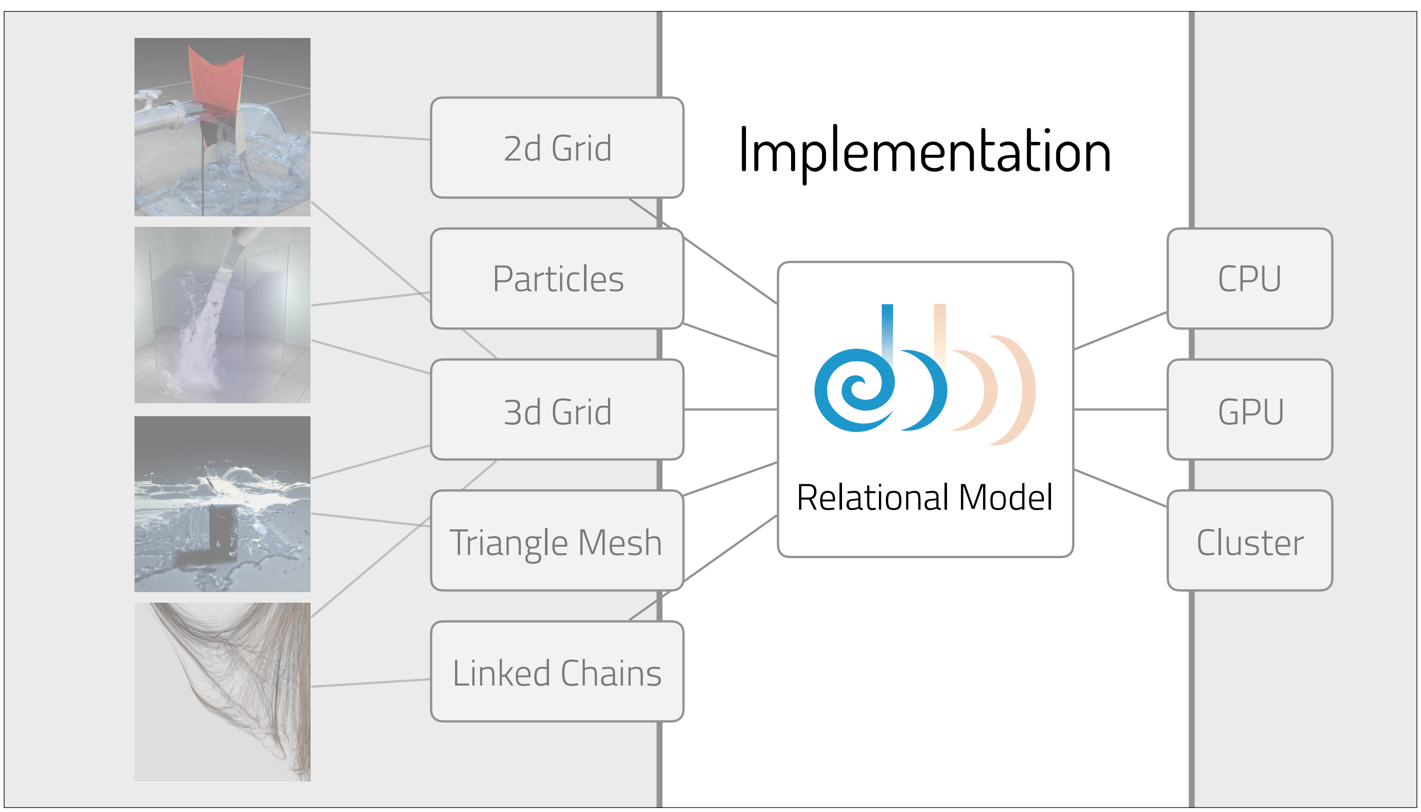


Query-Loops



Affine-Indices


Geometric Domain Modeling
2d Grid
Particles
3d Grid
Triangle Mesh
Linked Chains
Relational Model
CPU
GPU
Cluster


Implementation
2d Grid
Particles
3d Grid
Triangle Mesh
Linked Chains
Relational Model
CPU
GPU
Cluster

Phase Analysis

```
local ebb compute_acc ( v : dragon.vertices )
  var force = { 0.0, 0.0, 0.0 }

  -- Spring Force
  var mass = 0.0
  for e in v.edges do
    var diff  = e.head.pos - v.pos
    var scale = (e.rest_len / L.length(diff)) - 1.0
    mass     += e.rest_len
    force    -= K * scale * diff
  end

  v.nxt_pos = v.pos + dt          * v.vel
                    + 0.5*dt*dt * force/mass
  v.nxt_vel = v.vel + dt          * force/mass
end
```

Loop Generation

```
dragon.vertices:foreach(compute_acc)
```

```
for i=0,vertices.size do
  ...
end
```

```
CUDA_Launch(
  n_vertices/block_size,
  1, 1,
  block_size, 1, 1,
  kernel_code, ...
)
```

CPU

GPU

Instruction Generation

ebb

Terra

LLVM

x86 PTX

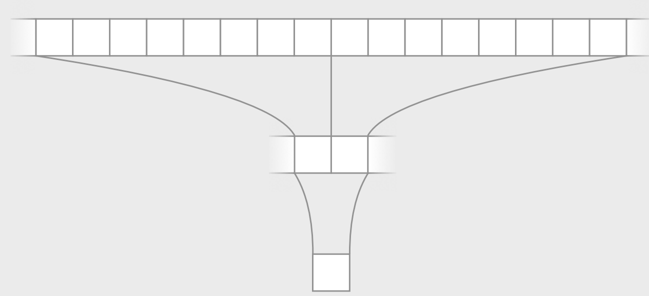
Reductions

```
local ebb compute_max_vel ( v : vertices )
  maxvel max= L.length(v.vel)
end
```

```
for i=0,vertices.size do
  ...
end
```

trivial, given sequential access

CPU

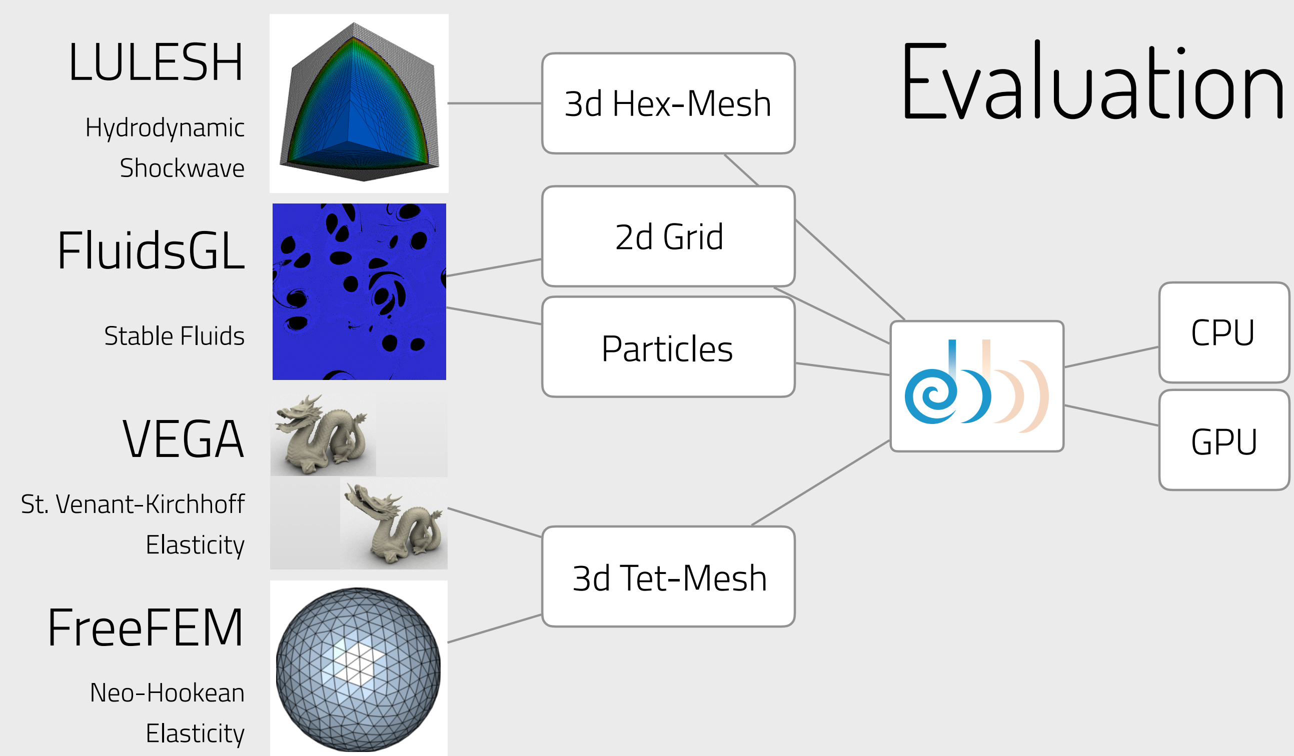


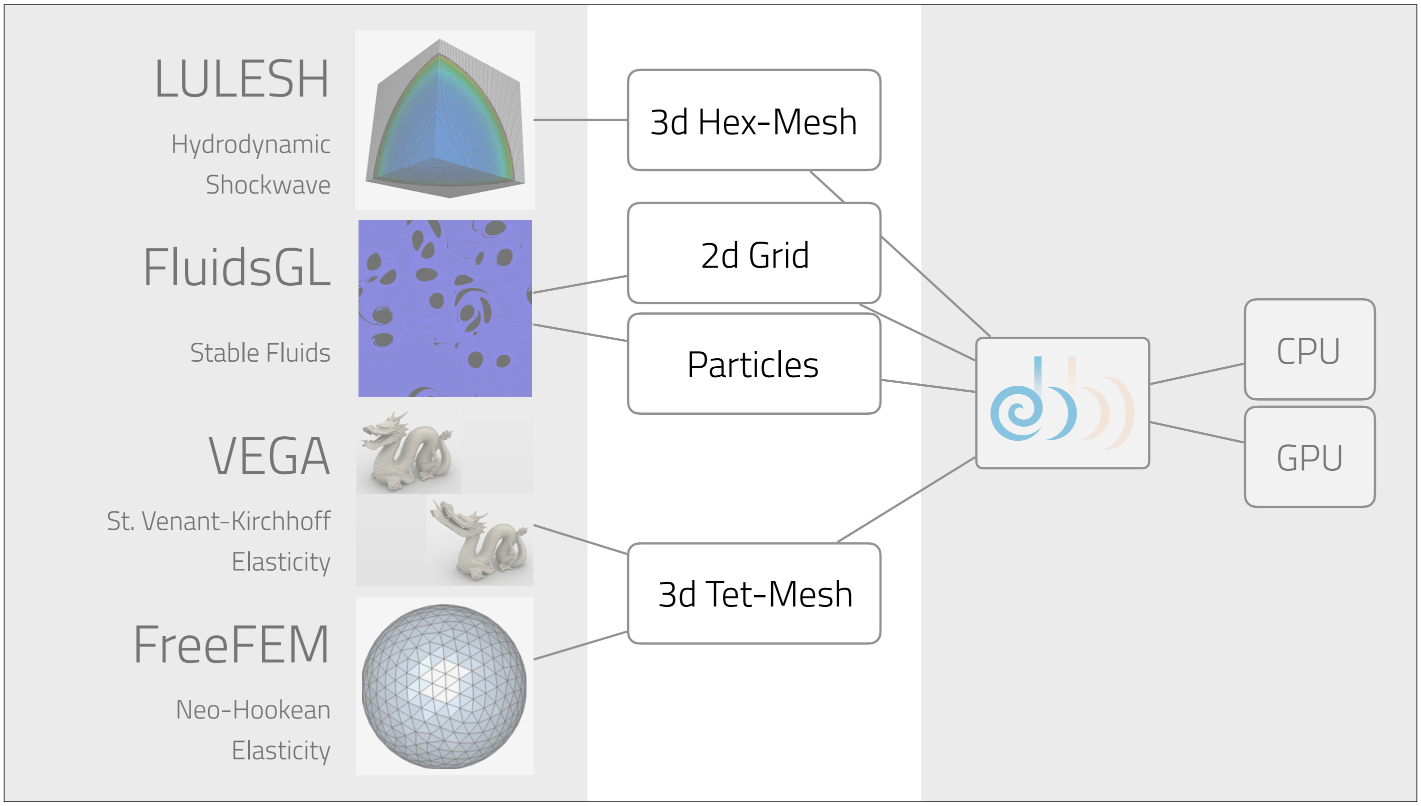
GPU

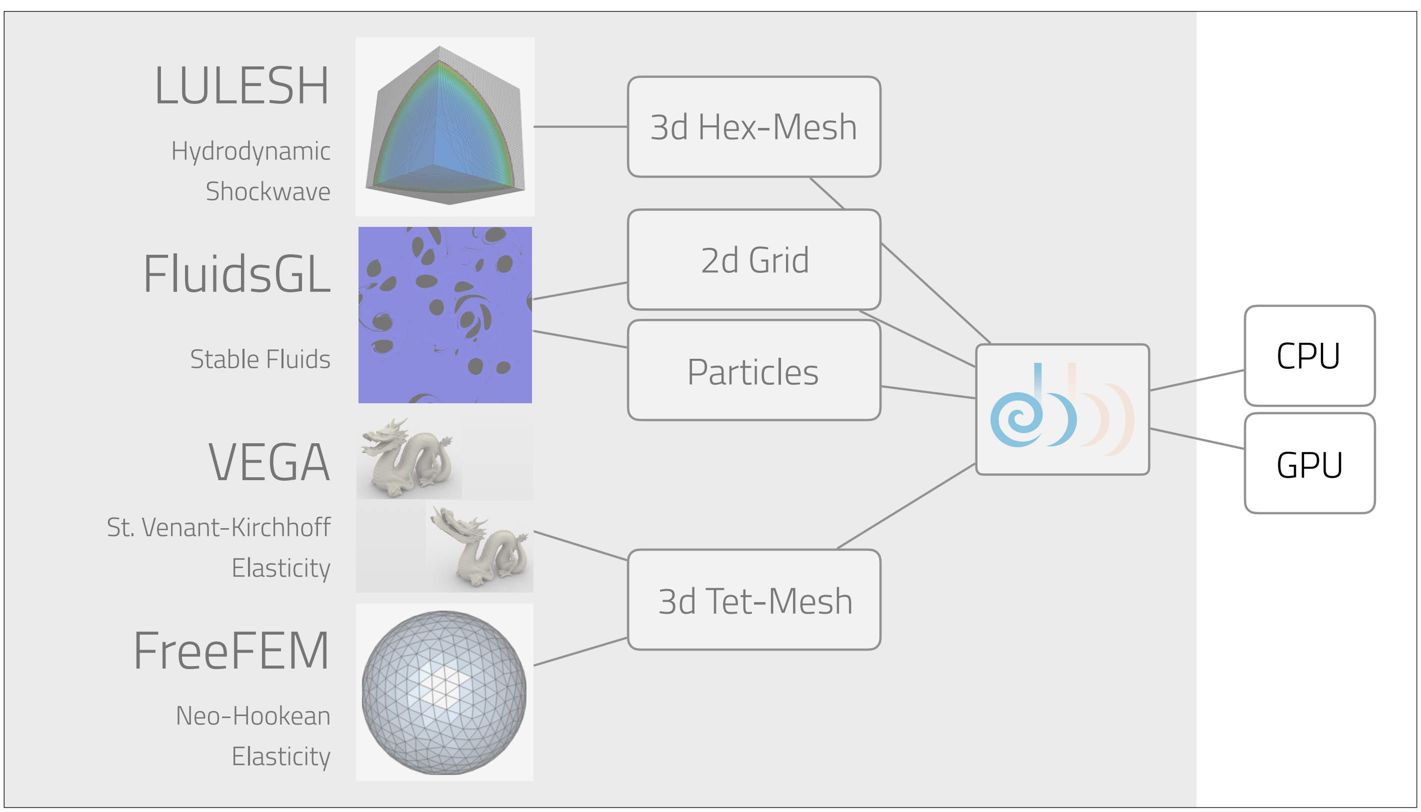
More Implementation Details

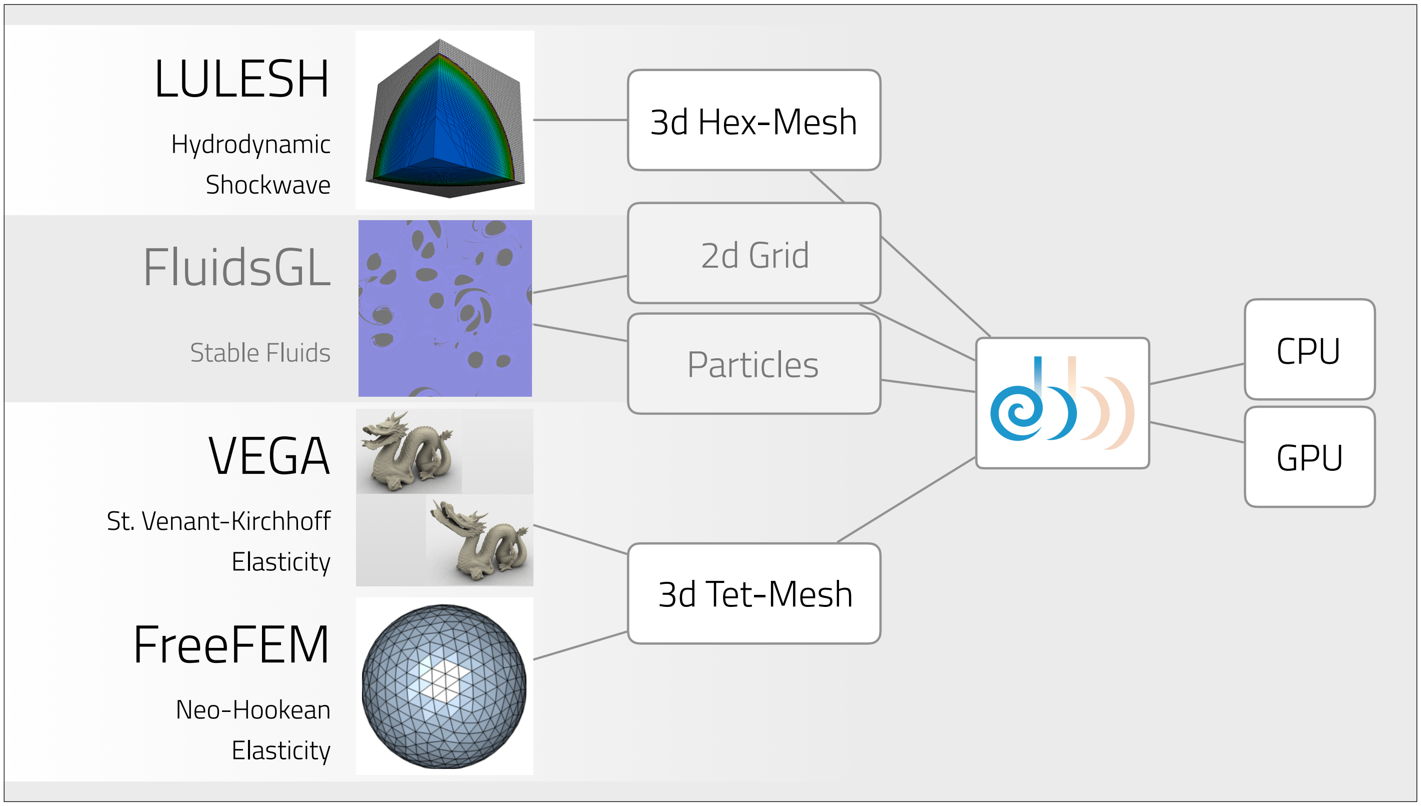
- automated data movement and layout
- data indexing for fast-access
- code-path specialization
- subset representation

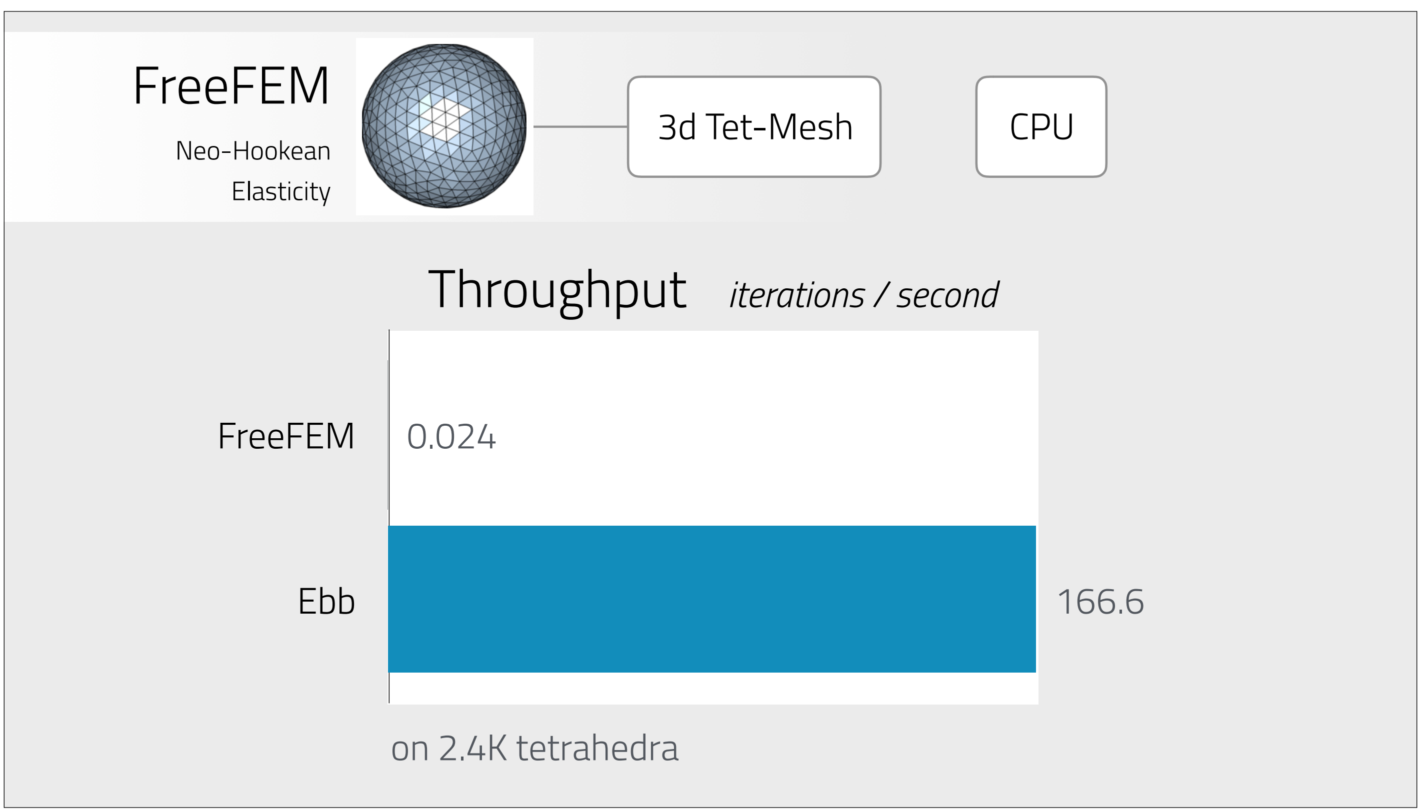
See paper & code for more details...

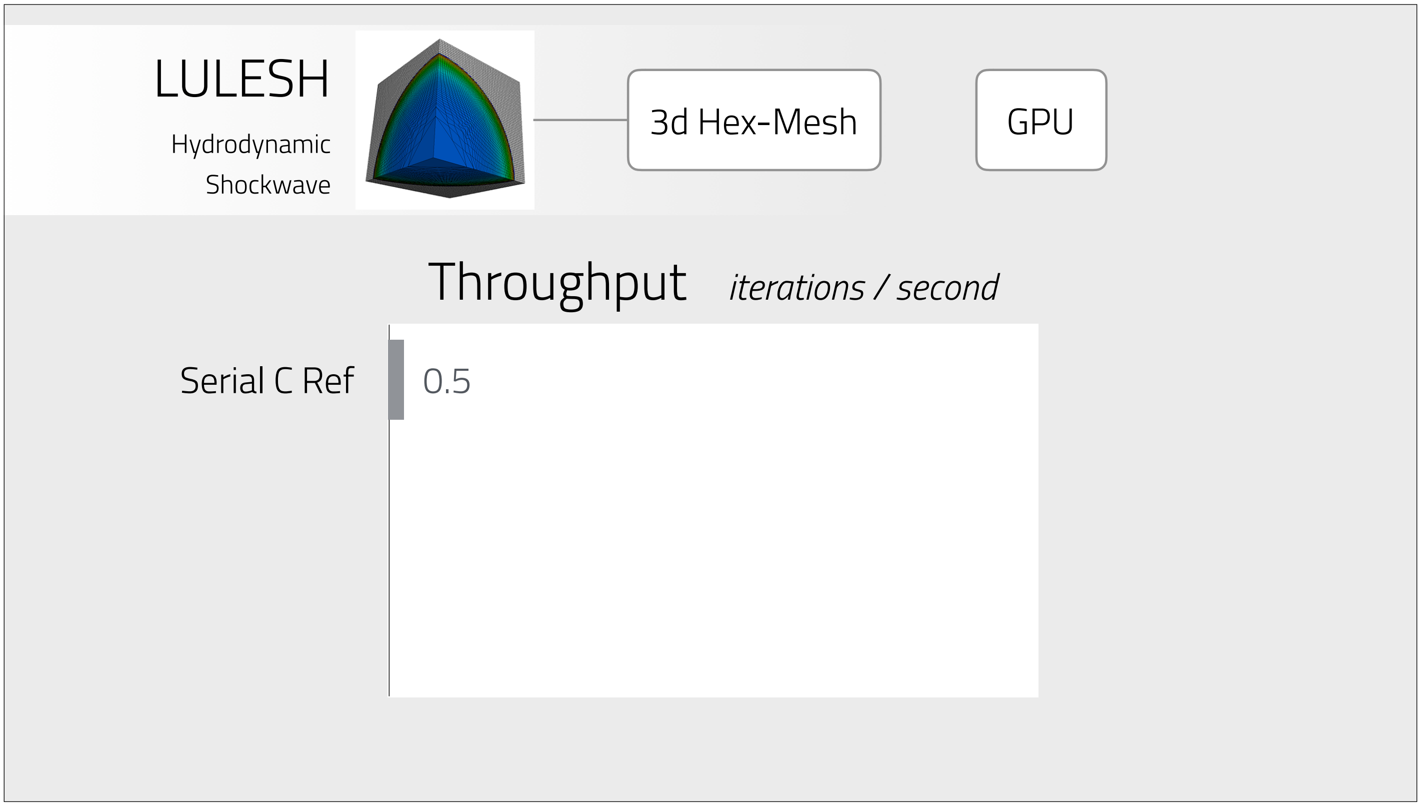


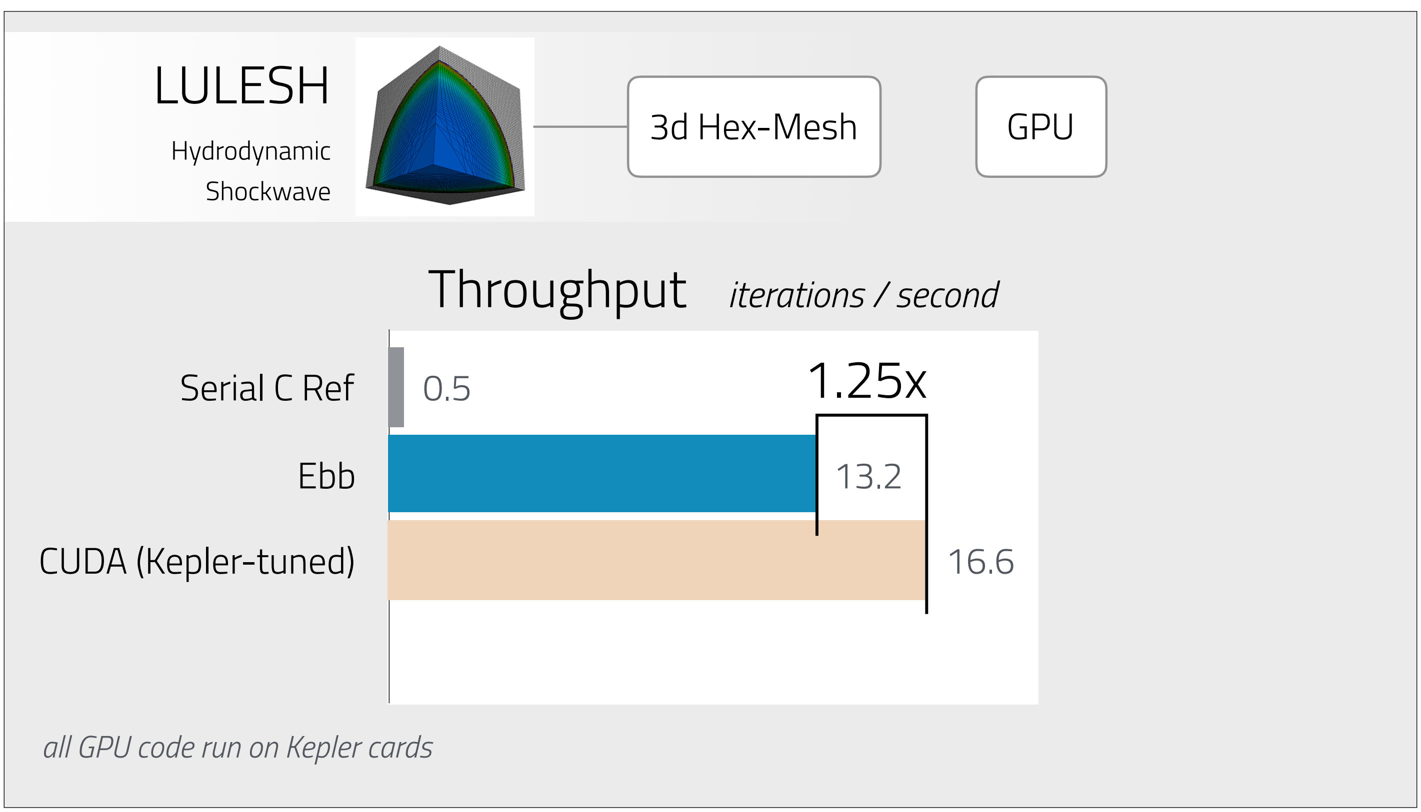

LULESH
Hydrodynamic
Shockwave
FluidsGL
Stable Fluids
VEGA
St. Venant-Kirchhoff
Elasticity
FreeFEM
Neo-Hookean
Elasticity
3d Hex-Mesh
2d Grid
Particles
3d Tet-Mesh
CPU
GPU

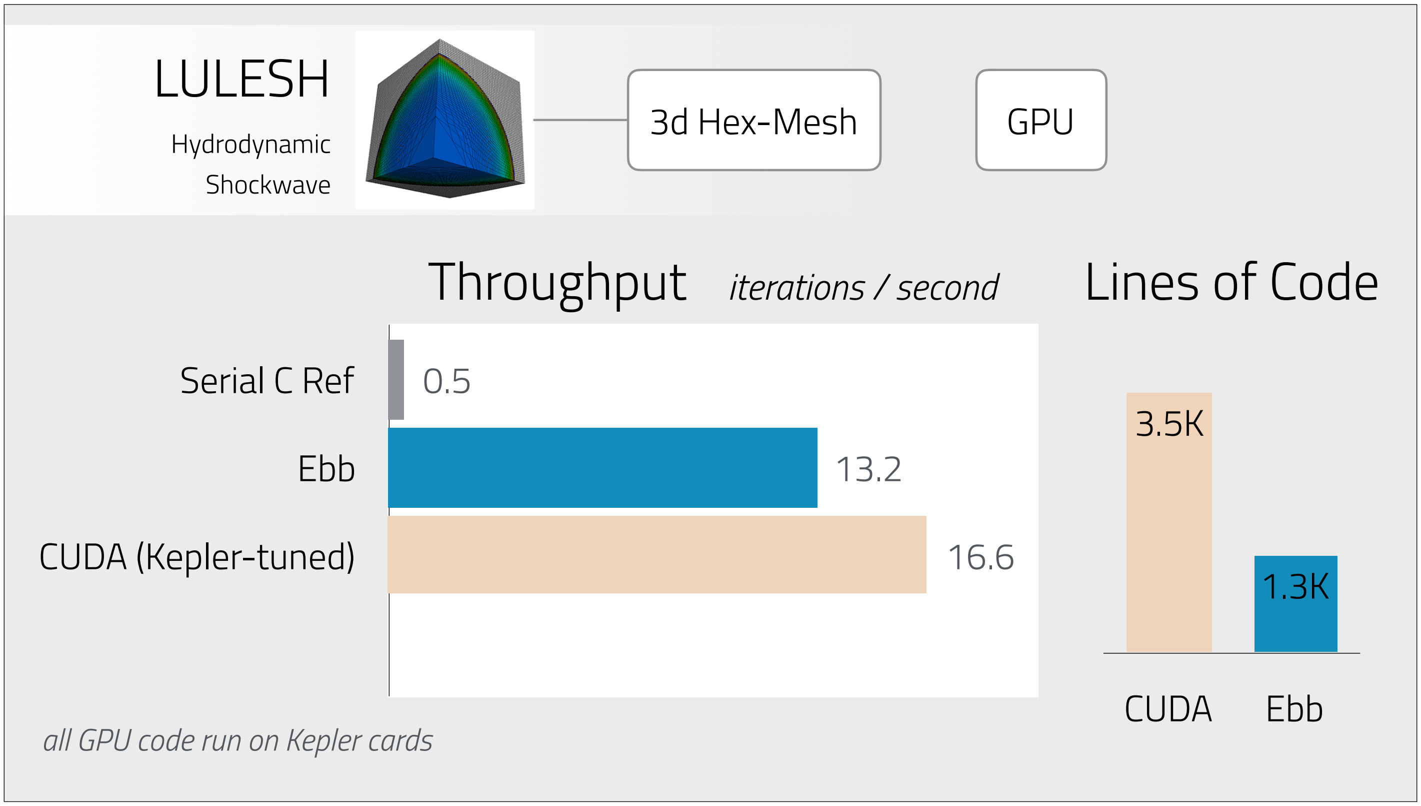

LULESH
Hydrodynamic
Shockwave
FluidsGL
Stable Fluids
VEGA
St. Venant-Kirchhoff
Elasticity
FreeFEM
Neo-Hookean
Elasticity
3d Hex-Mesh
2d Grid
Particles
3d Tet-Mesh
CPU
GPU

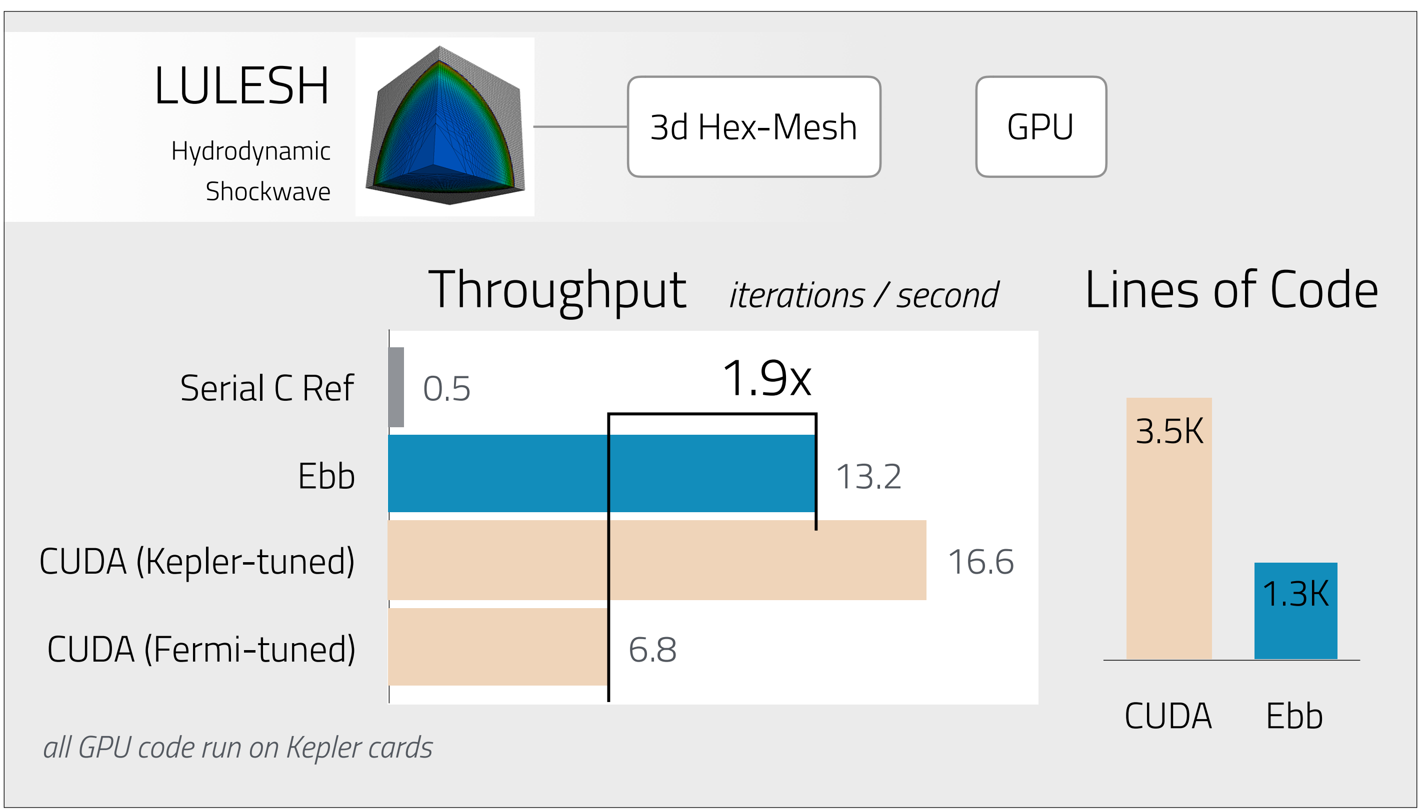

LULESH
Hydrodynamic
Shockwave
FluidsGL
Stable Fluids
VEGA
St. Venant-Kirchhoff
Elasticity
FreeFEM
Neo-Hookean
Elasticity
3d Hex-Mesh
2d Grid
Particles
3d Tet-Mesh
CPU
GPU

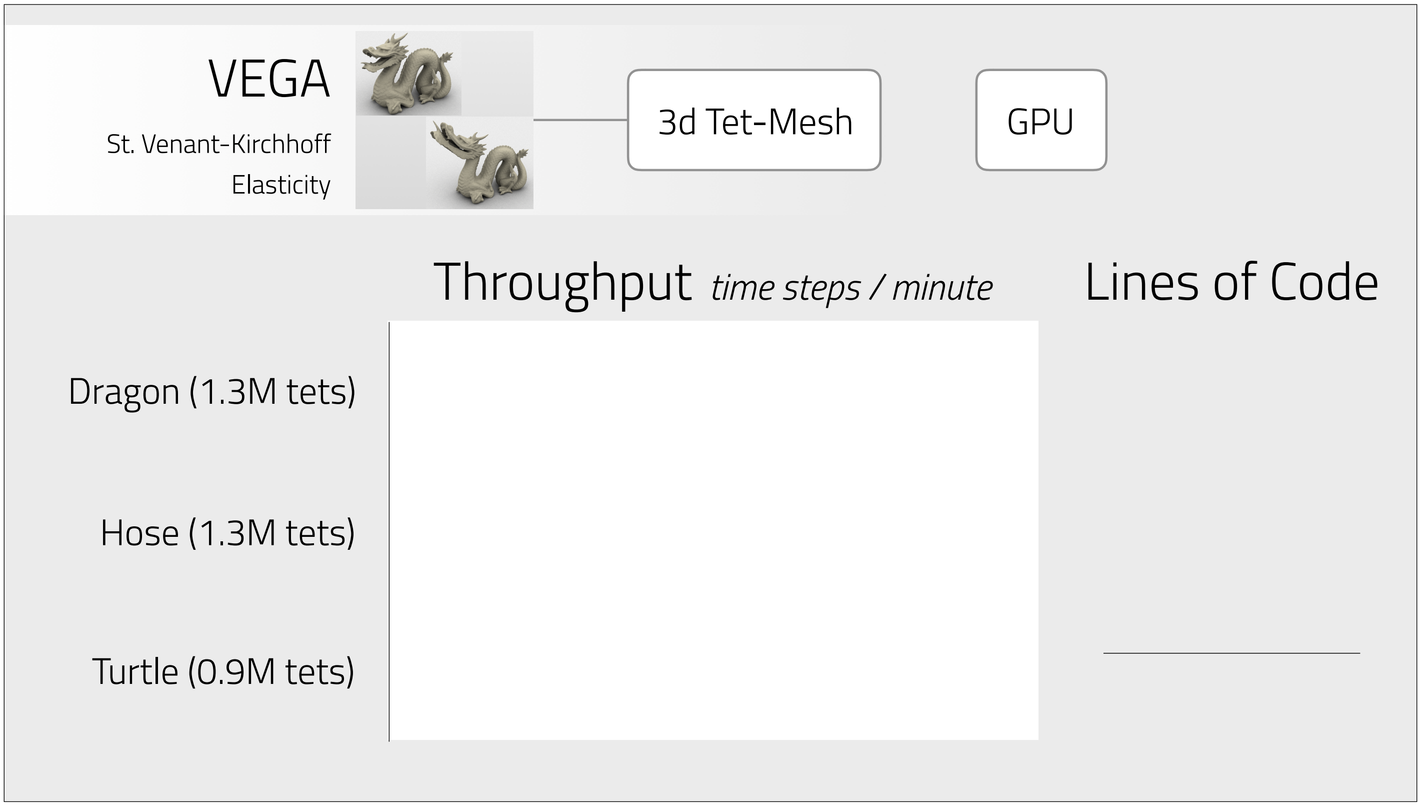

FreeFEM
Neo-Hookean
Elasticity
3d Tet-Mesh
CPU
Throughput
iterations / second
FreeFEM
0.024
Ebb
166.6
on 2.4K tetrahedra

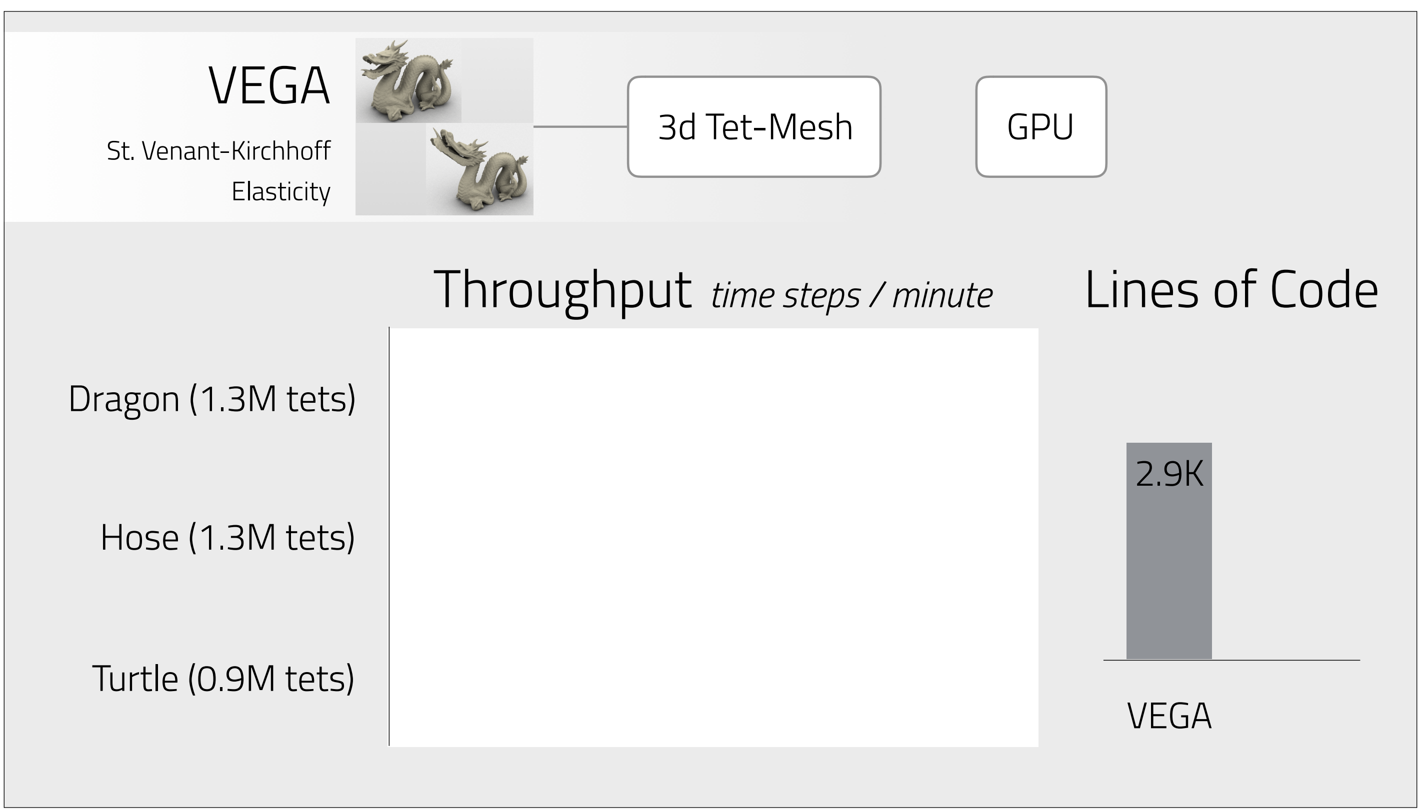

LULESH
Hydrodynamic
Shockwave
3d Hex-Mesh
GPU
Throughput
iterations / second
Serial C Ref
0.5

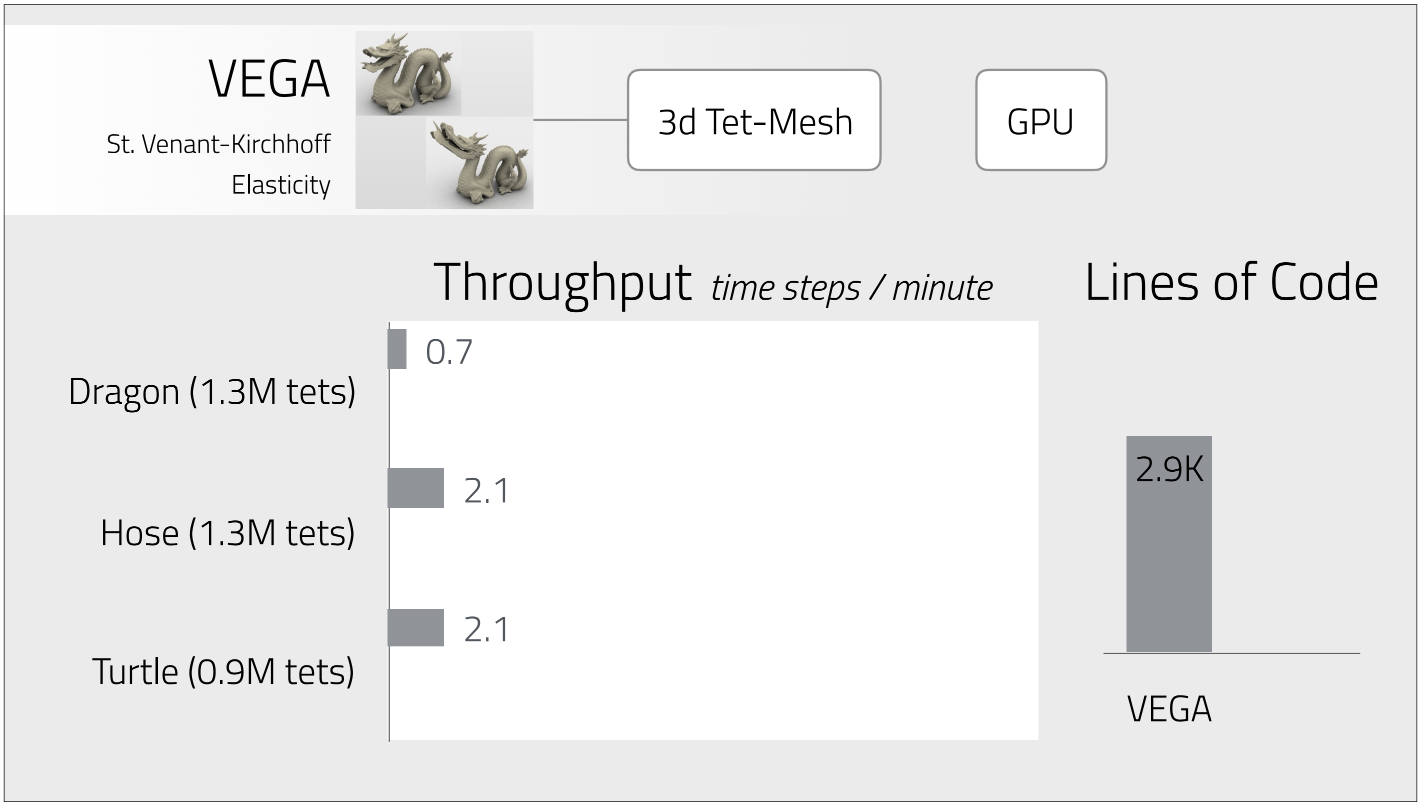

LULESH
Hydrodynamic
Shockwave
3d Hex-Mesh
GPU
Throughput
iterations / second
Serial C Ref
0.5
1.25x
Ebb
13.2
CUDA (Kepler-tuned)
16.6
all GPU code run on Kepler cards

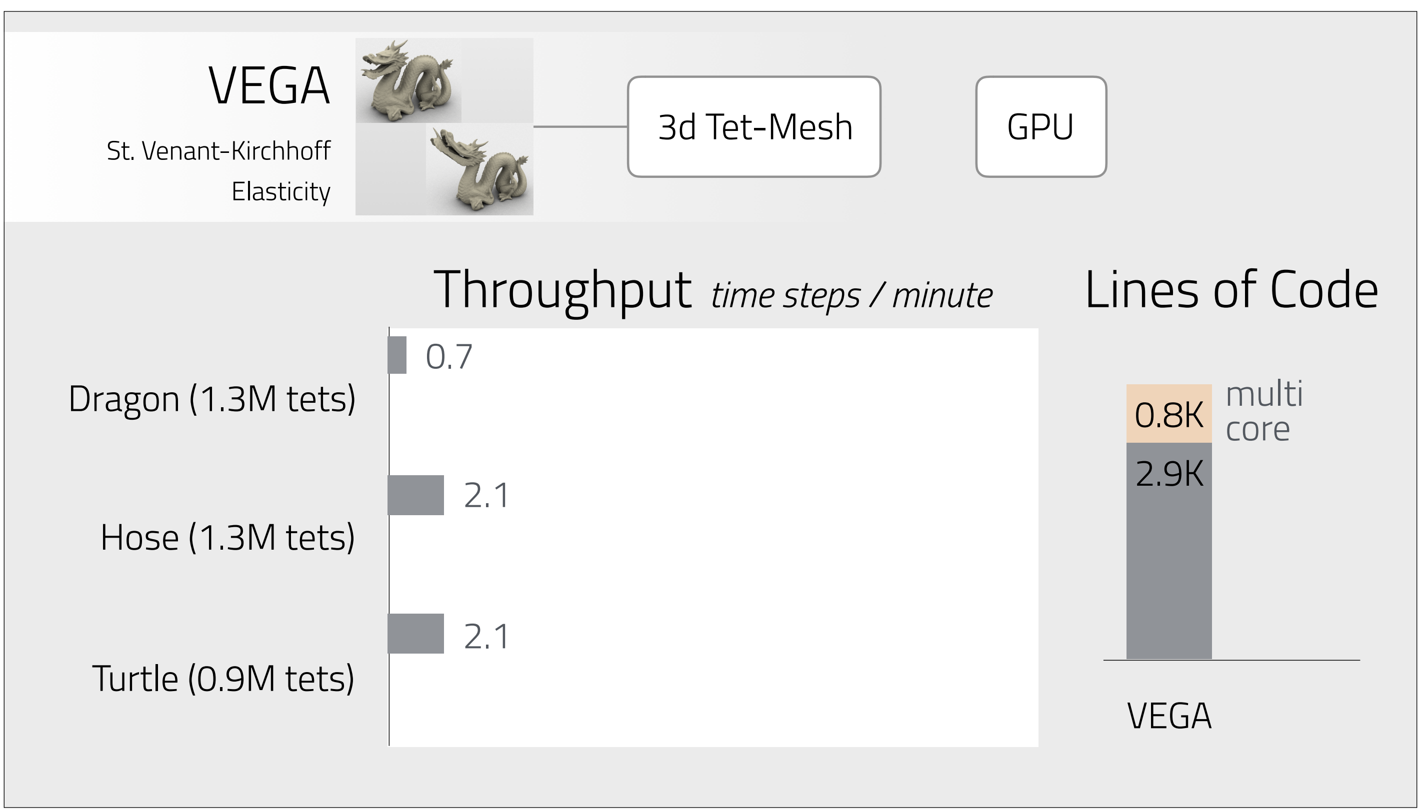

LULESH
Hydrodynamic
Shockwave
3d Hex-Mesh
GPU
Throughput
iterations / second
Lines of Code
Serial C Ref
0.5
Ebb
13.2
CUDA (Kepler-tuned)
16.6
3.5K
1.3K
CUDA
Ebb
all GPU code run on Kepler cards

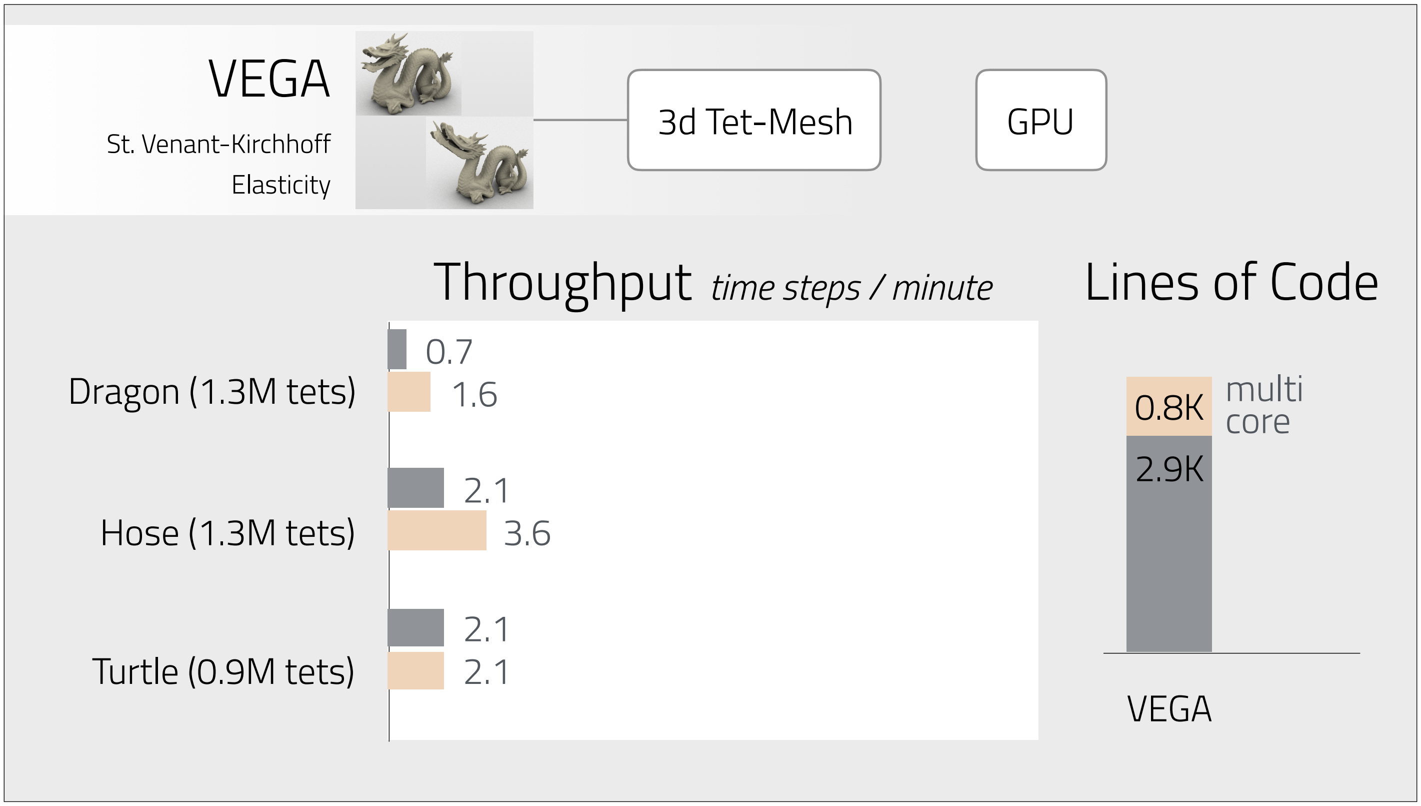

LULESH
Hydrodynamic
Shockwave
3d Hex-Mesh
GPU
Throughput
iterations / second
Lines of Code
Serial C Ref
0.5
1.9x
Ebb
13.2
CUDA (Kepler-tuned)
16.6
CUDA (Fermi-tuned)
6.8
3.5K
1.3K
CUDA
Ebb
all GPU code run on Kepler cards

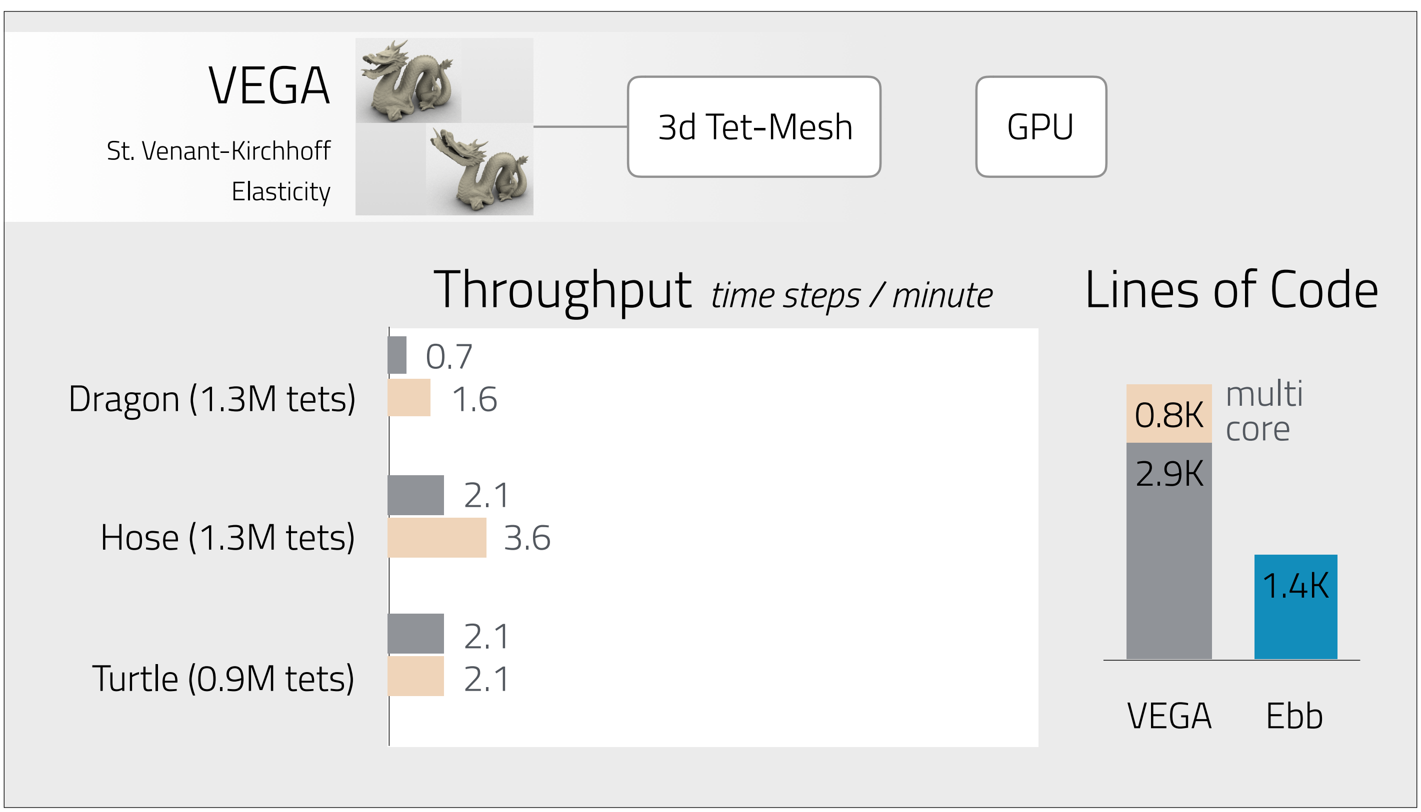

VEGA
St. Venant-Kirchhoff
Elasticity
3d Tet-Mesh
GPU
Throughput *time steps / minute*
Lines of Code
Dragon (1.3M tets)
Hose (1.3M tets)
Turtle (0.9M tets)

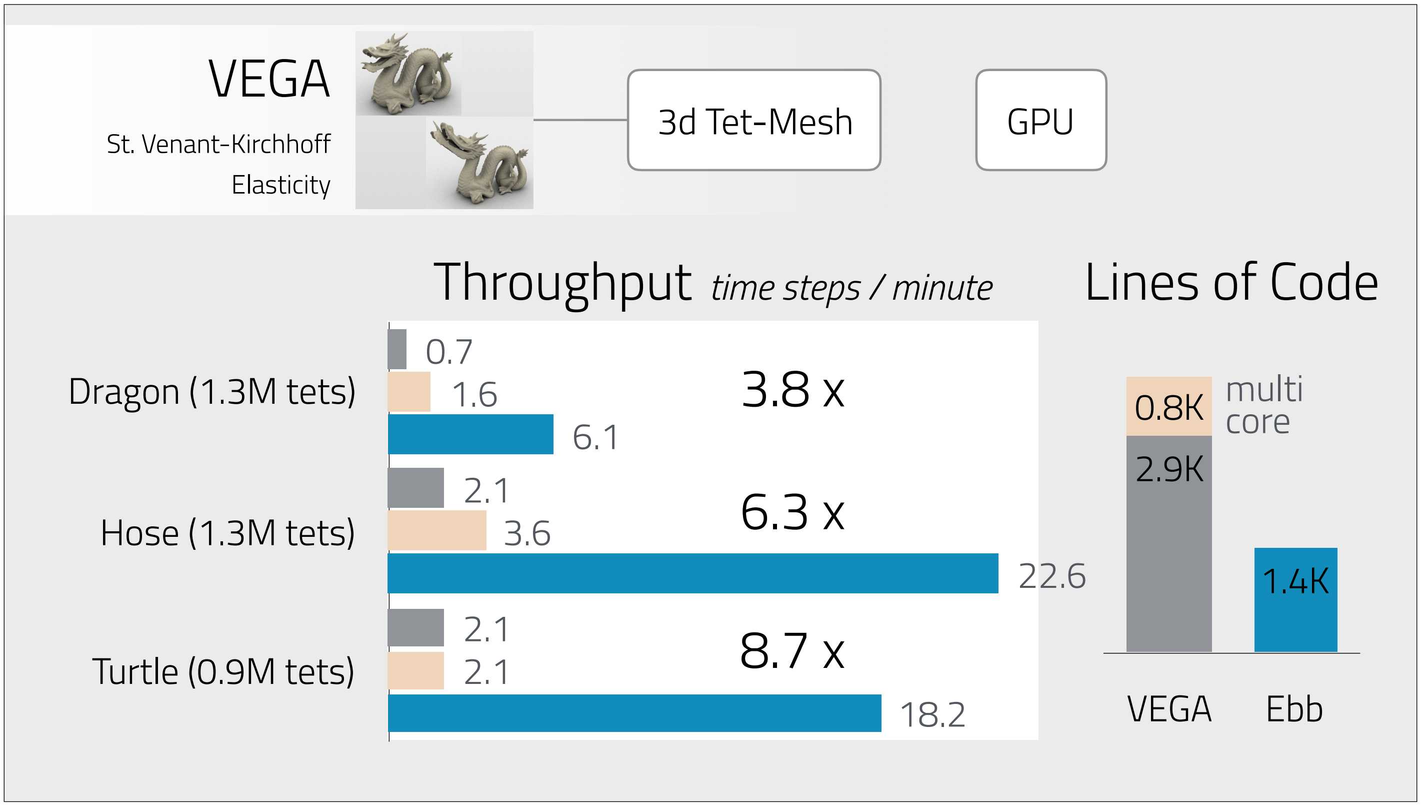

VEGA
St. Venant-Kirchhoff
Elasticity
3d Tet-Mesh
GPU
Throughput *time steps / minute*
Lines of Code
Dragon (1.3M tets)
Hose (1.3M tets)
Turtle (0.9M tets)
2.9K
VEGA


VEGA
St. Venant-Kirchhoff
Elasticity
3d Tet-Mesh
GPU
Throughput *time steps / minute*
Dragon (1.3M tets)
0.7
Hose (1.3M tets)
2.1
Turtle (0.9M tets)
2.1
Lines of Code
2.9K
VEGA


VEGA
St. Venant-Kirchhoff
Elasticity
3d Tet-Mesh
GPU
Throughput *time steps / minute*
Dragon (1.3M tets)
0.7
Hose (1.3M tets)
2.1
Turtle (0.9M tets)
2.1
Lines of Code
0.8K
multi
core
2.9K
VEGA


VEGA
St. Venant-Kirchhoff
Elasticity
3d Tet-Mesh
GPU
Throughput time steps / minute
Dragon (1.3M tets)
0.7
1.6
Hose (1.3M tets)
2.1
3.6
Turtle (0.9M tets)
2.1
2.1
Lines of Code
0.8K
multi
core
2.9K
VEGA


VEGA
St. Venant-Kirchhoff
Elasticity
3d Tet-Mesh
GPU
Throughput time steps / minute
Dragon (1.3M tets)
0.7
1.6
Hose (1.3M tets)
2.1
3.6
Turtle (0.9M tets)
2.1
2.1
Lines of Code
0.8K
multi
core
2.9K
1.4K
VEGA
Ebb



http:// ebblang.org

GETTING STARTED TUTORIALS MANUAL PUBLICATIONS CONTACT DOWNLOAD

View On Github

a part of the Liszt project and PSAAP II center at Stanford University

Ebb

is a programming language for writing physical simulations. Ebb programs are *performance portable*: they can be efficiently executed on both CPUs and GPUs. Ebb is embedded in the Lua programming language using Terra.

http:// ebblang.org

Tutorials

Introduction

These tutorials provide a tour through all o
sufficient to get started writing your own si
domains.

01: Hello, 42!

The basics of an Ebb program; print out 42 for ea

02: Domain Loading From Files

How to use a domain library to load in a mesh fro
statistics and computations on that mesh; We us

03: Visualizing Simulations

Basic usage of VDB to generate visual output fro
vertices of the octahedron.

04: User-defined Fields and Globals

05: Accessing Neighbors

How to access data at neighboring element
heat-diffusion on the surface of the Stanfor

06: Phases, Reads, Writes, Reductions

A key feature of Ebb is that all functions ar
and show alternative ways of writing the he

07: Using Standard Grids

Some features of the standard grid domain;
grid, handling both periodic and normal bo

08: Relations

Relations are the basic data structure in Ebl
build a torus from scratch and simulate hea

09: Particle-Grid Coupling

How to connect and update the relationshi
tracer particles in an evolving heat gradien

Interoperability

These tutorials introduce the features th
code, including how to write custom hig
(familiarity with the introduction tutoria

10: Data Layout Descriptors (DLDs)

DLDs give us raw access to the simulation mer
integrate a piece of unsafe code into an Ebb pr

11: Calling C-code

DLDs can also be used from C code written en
unsafe C function into an Ebb simulation.

12: File I/O

Using DLDs, we can efficiently load data into a
simulation; we write code to load and write OI

Domain Modeling

These tutorials explain the features that

should be prepared to start developing th
domains. (familiarity with the introductio

13: Group-By and Query-Loops

Grouping and Querying lets us invert simple re
simulate heat diffusion on a graph encoded wi

14: Join Tables

A common pattern that enables us to represent
it to enable access to the triangles around a ver

15: Macros

Macros let us hide unintuitive encodings behin
the join-table example using macros.

16: Grid Relations

How to use relations to represent data from a g
two-scale coupled grid-to-grid domain for simu

17: Subsets

http:// ebblang.org

OVERVIEW
- Overview
- The Ebb Language
  - Ebb functions
  - Types, Literals, and Casting
  - Expressions
  - Declarations and Assignments
  - Field/Global Writes, Reads, and Reductions
  - Phase Checking
  - Control Flow
- The Ebb API
  - Types
  - Summary of Types
  - Builtins
  - External C functions in Ebb code
  - Globals
  - Constants and Literals
  - Relations
  - Grouping and Query-Loops
  - Affine Indexing and Grids
  - Fields
  - Subsets
  - Data Layout Descriptors
  - Load and Dump (File I/O)
  - Macros and Quotes

Ebb Manual

Overview

Ebb consists of two parts: an embedded language, and a Lua API. The language proper is used to define Ebb *functions*, while the Lua API is used to construct and interrogate the data structures, as well as launch functions via `foreach` calls. For instance, in the `hello42` sample program, the `printsum()` function is written in the Ebb language, while the rest of the program makes calls to the API.

In addition to these two parts, a set of standard domain and support libraries are provided, which this documentation will also discuss.

The remainder of the manual will assume a passing familiarity with the structure of Ebb programs. For a more intuitive introduction to the language, please see the tutorials.

Additionally this manual assumes a passing familiarity with the Lua language. Specifically, Ebb is embedded in Lua 5.1, via Terra. You can find a number of good tutorials, manuals and documentation online, which we will not repeat here.

The Ebb Language

The Ebb language is used to define Ebb functions, which can either be used in other
Ebb functions, or executed for each element of some relation.

Why New Programming Languages for Simulation?

GILBERT BERNSTEIN
Stanford University
and
FREDRIK KJOLSTAD
Massachusetts Institute of Technology

Ebb: A DSL for Physical Simulation on CPUs and GPUs

GILBERT LOUIS BERNSTEIN and CHINMAYEE SHAH and CRYSTAL LEMIRE and ZACHARY DEVITO and MATTHEW FISHER and PHILIP LEVIS and PAT HANRAHAN
Stanford University

Designing programming environments for physical simulation is challenging because simulations rely on diverse algorithms and geometric domains. These challenges are compounded when we try to run efficiently on heterogeneous parallel architectures. We present Ebb, a domain-specific language (DSL) for simulation, that runs efficiently on both CPUs and GPUs. Unlike previous DSLs, Ebb uses a three-layer architecture to separate (1) simulation code, (2) definition of data structures for geometric domains, and (3) runtimes supporting parallel architectures. Different geometric domains are implemented as libraries that use a common, unified, relational data model. By structuring the simulation framework in this way, programmers implementing simulations can focus on the physics and algorithms for each simulation without worrying about their implementation on parallel computers. Because the geometric domain libraries are all implemented using a common runtime based on relations, new geometric domains can be added as needed, without specifying the details of memory management, mapping to different parallel architectures, or having to expand the runtime's interface.

We evaluate Ebb by comparing it to several widely used simulations, demonstrating comparable performance to hand-written GPU code where available, and surpassing existing CPU performance optimizations by up to 9× when no GPU code exists.

This paper describes Ebb[1], a domain-specific language (DSL) for developing physical simulations of fluids and deformable meshes that is designed to run efficiently on both CPUs and GPUs. Ebb is motivated by the successes of prior DSLs, such as the RenderMan shading language [Hanrahan and Lawson 1990], the Halide image processing language [Ragan-Kelley et al. 2012], and the Liszt [DeVito et al. 2011] language for solving partial differential equations on unstructured meshes. These DSLs use abstractions (lights and materials for rendering, functional images, and meshes/fields, respectively) that allow simulation programmers to write code at a higher-level. Even though DSL code is higher-level, the DSLs can be compiled to a wide range of computer platforms and perform as well as code written in a low-level language.

Each of these existing DSLs are designed around one geometric domain (e.g. Liszt's unstructured meshes) whereas simulations often need to use a variety of geometric domains (triangle meshes, regular grids, tetrahedral volumes, etc.). In order to support multiple geometric domains, we propose a three-layer architecture for Ebb. In the top layer, users write application code, such as a fluid simulator, FEM library, or multi-physics library, in the Ebb language using its geometric domain libraries; Similar to shader languages, this

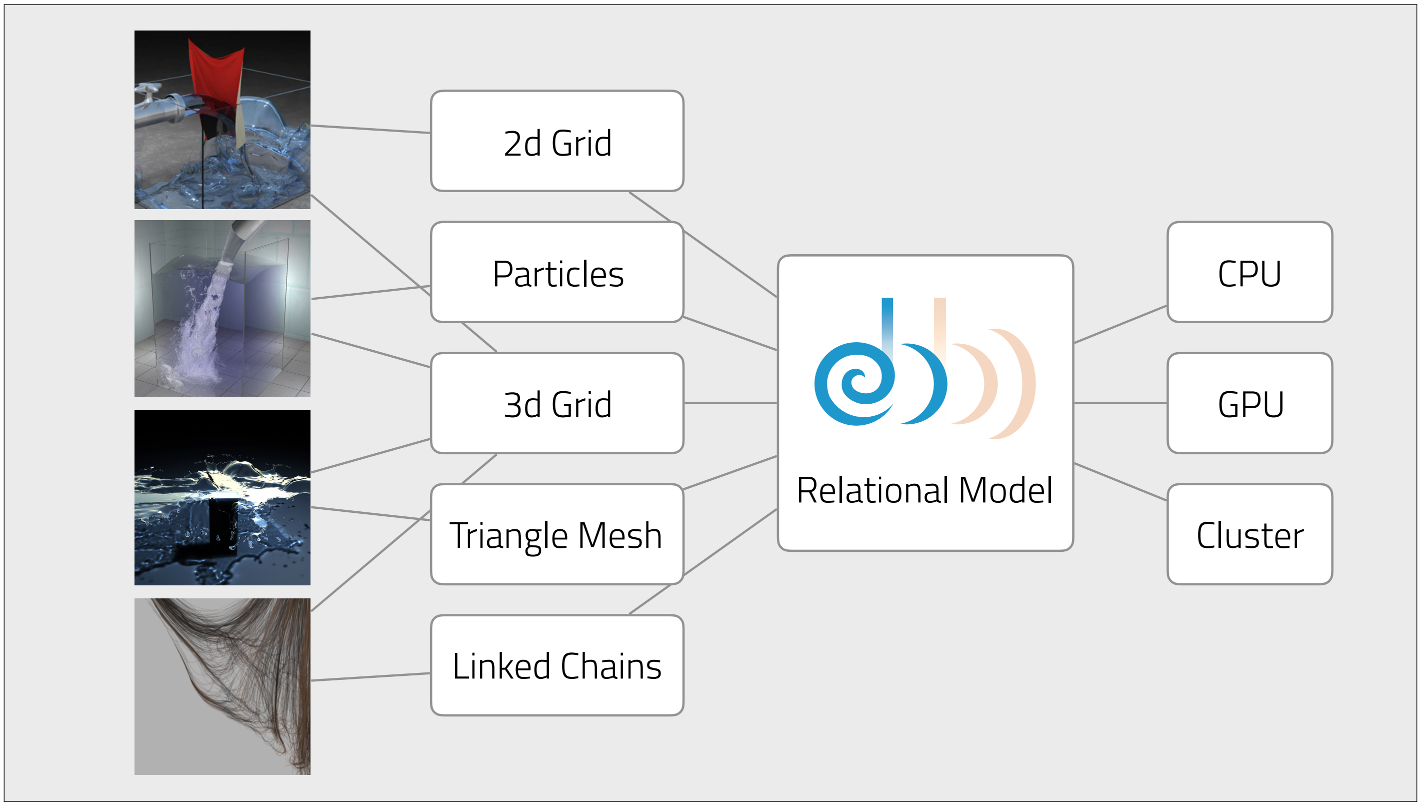
Simit: A Language for Physical Simulation

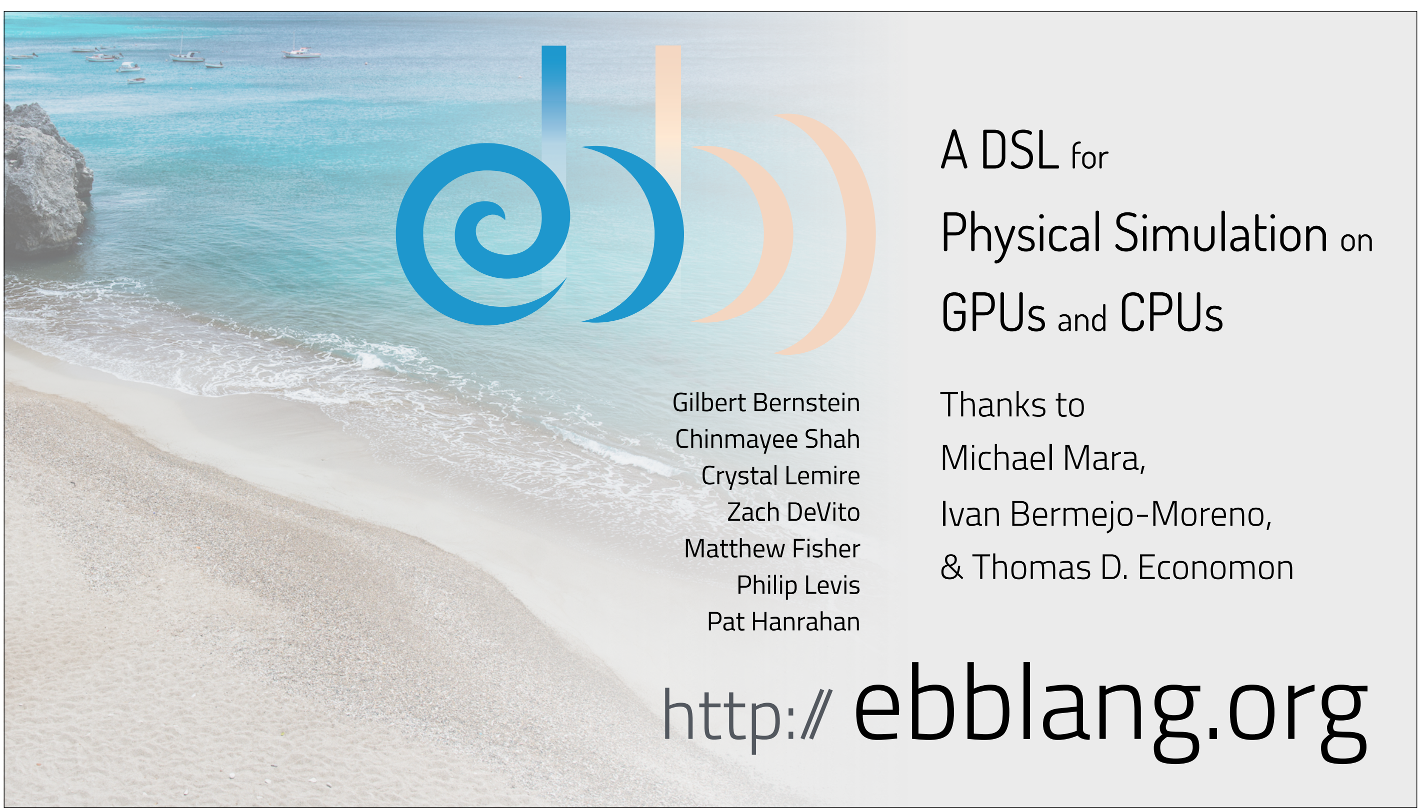
FREDRIK KJOLSTAD
Massachusetts Institute of Technology
SHOAIB KAMIL
Adobe
JONATHAN RAGAN-KELLEY
Stanford University
DAVID I.W. LEVIN
Disney Research
SHINJIRO SUEDA
California Polytechnic State University
DESAI CHEN
Massachusetts Institute of Technology
ETIENNE VOUGA
University of Texas at Austin
DANNY M. KAUFMAN
Adobe
and
GURTEJ KANWAR, WOJCIECH MATUSIK, and SAMAN AMARASINGHE
Massachusetts Institute of Technology

Future Directions in Simulation Languages

Simit

- Collisions, Remeshing, non-trivially parallelizable algorithms
- Distributed Machines (cloud, cluster, etc.)
- New Data Layout & Code Optimizations
- Simulation-Specific Debugging & Support Tools


2d Grid
Particles
3d Grid
Triangle Mesh
Linked Chains
Relational Model
CPU
GPU
Cluster


A DSL for
Physical Simulation on
GPUs and CPUs
Gilbert Bernstein
Chinmayee Shah
Crystal Lemire
Zach DeVito
Matthew Fisher
Philip Levis
Pat Hanrahan
Thanks to
Michael Mara,
Ivan Bermejo-Moreno,
& Thomas D. Economon
http:// ebblang.org