SIMULATION OF DEFORMABLE ENVIRONMENT WITH HAPTIC FEEDBACK ON GPU

Marco Altomonte, Davide Zerbato, Debora Botturi, Paolo Fiorini

Laboratorio di Robotica, ALTAIR
Dipartimento di Informatica
Università di Verona

Overview

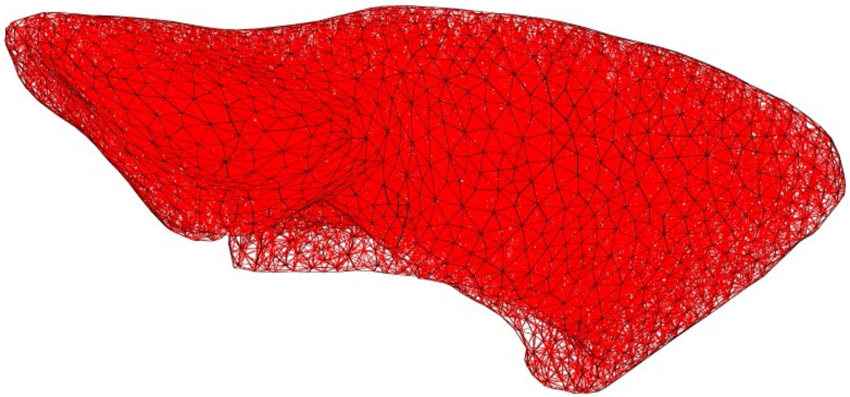
- Motivation
- Mass-spring systems
- Optimized Elastic force computation
- Collision detection
- Surgical gestures
- Haptic force computation
- Conclusions

Motivation

Mass-spring systems

To simulate a soft-body:

- discretize the volume in points, connected by springs
- compute forces with linear elastic model equations
- compute positions with a time integration scheme
- use tetrahdera to simulate volume preservation forces



Mass data structure

Mass data structure members:

- position at 3 contiguous time steps
- total acting force
- set of properties (mass, rendering flags, collision marks)

Due to hardware limitations:

- store each member in different arrays but at the same index
- compress scalar members in 4D floating-point vectors
- use bidimensional arrays

P31	P32	P33	P34	P35	P36
P25	P26	P27	P28	P29	P30
P19	P20	P21	P22	P23	P24
P13	**P14**	P15	P16	P17	P18
P7	P8	P9	P10	P11	P12
P1	P2	P3	P4	P5	P6

POSITION ARRAY 1

P31	P32	P33	P34	P35	P36
P25	P26	P27	P28	P29	P30
P19	P20	P21	P22	P23	P24
P13	**P14**	P15	P16	P17	P18
P7	P8	P9	P10	P11	P12
P1	P2	P3	P4	P5	P6

POSITION ARRAY 2

F31	F32	F33	F34	F35	F36
F25	F26	F27	F28	F29	F30
F19	F20	F21	F22	F23	F24
F13	**F14**	F15	F16	F17	F18
F7	F8	F9	F10	F11	F12
F1	F2	F3	F4	F5	F6

FORCE ARRAY

D31	D32	D33	D34	D35	D36
D25	D26	D27	D28	D29	D30
D19	D20	D21	D22	D23	D24
D13	**D14**	D15	D16	D17	D18
D7	D8	D9	D10	D11	D12
D1	D2	D3	D4	D5	D6

PROPERTIES ARRAY

Spring data structure

Store springs as mass data members in 4D vectors composed by:

- rest length
- elastic factor
- damping factor
- index of the other connected mass

To store the entire spring set:

- store a number of arrays equal to the maximum valence
- add null elements for masses with lower valences

M31	M32	M33	M34	M35	M36
M25	M26	M27	M28	M29	M30
M19	M20	M21	M22	M23	M24
M13	M14	M15	M16	M17	M18
M7	M8	M9	M10	M11	M12
M1	M2	M3	M4	M5	M6

MASS ARRAY

S30	S32	S21	S33	S34	S17
S26	S2	S29	S29	S7	S22
S23	S3	S18	S1	S20	S19
S15	S16	S18	S20	S21	S22
S10	S11	S5	S12	S4	S13
S1	S3	S5	S7	S8	S8

SPRING ARRAY 1

S31	S33	S24	NULL	NULL	S34
S28	NULL	NULL	S30	S28	S31
S24	S6	NULL	S25	S23	S26
S16	S17	S19	NULL	NULL	NULL
NULL	NULL	NULL	NULL	S13	S14
S2	S4	S6	NULL	S9	S11

SPRING ARRAY 2

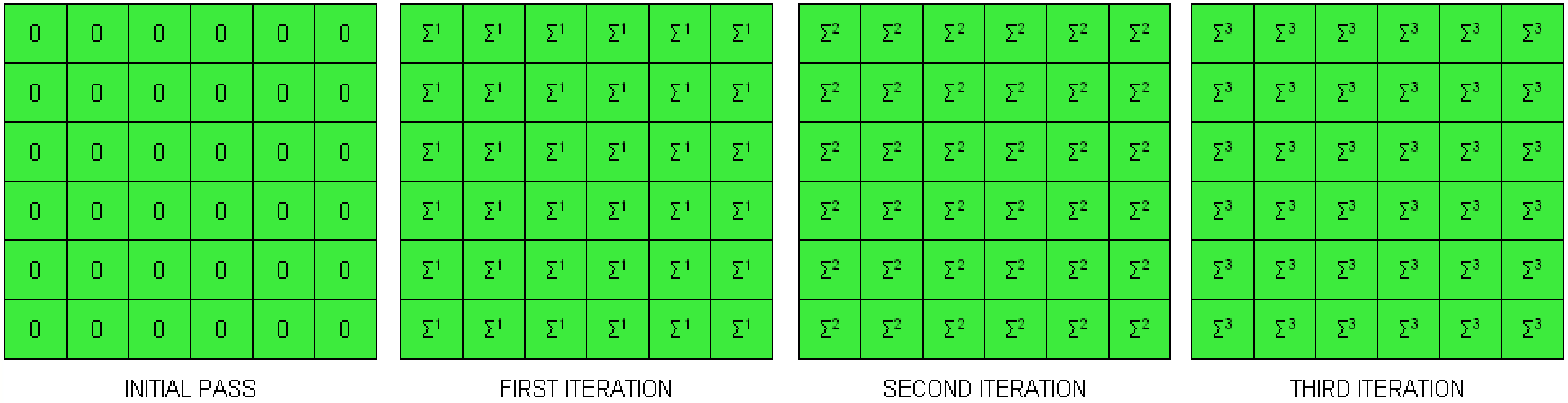
S32	S25	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	S27
NULL	S9	NULL	NULL	NULL	S27
NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	S14	S15
NULL	NULL	NULL	NULL	S10	S12

SPRING ARRAY 3

Elastic force computation

Accumulate elastic forces:

- clear the force array: apply a kernel that outputs (0,0,0,0)
- for each spring array apply a kernel on all elements
- for each element accumulate the elastic force of 1 spring
- kernel inputs are the mass position array and one spring array
- fetch the 2nd mass position using the index in the spring array
- detect and discard null springs



Performance optimization

We propose a method to reduce the null elements:

- sort the mass set by descending order valence
- for each spring array apply the kernel on non null springs only
- use a group of horizontal lines to specify the sub-array
- use a rectangular area to specify a complete array

OUR METHOD

M18	M21	M26	M27	M34	M35
M7	M8	M9	M10	M16	M17
M25	M28	M29	M33	M36	M4
M13	M14	M15	M19	M22	M23
M30	M31	M32	M1	M2	M3
M5	M6	M11	M12	M20	M24

MASS ARRAY

S22	S18	S2	S29	S33	S34
S10	S11	S5	S12	S20	S21
S28	S30	S28	S24	S34	S7
S16	S17	S19	S24	S25	S23
S27	S32	S25	S2	S4	S6
S10	S12	S14	S15	S9	S27

SPRING ARRAY 1

NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL
S26	S29	S7	S21	S17	NULL
S15	S16	S18	S23	S1	S20
S31	S31	S33	S1	S3	S5
S9	S11	S13	S14	S6	S26

SPRING ARRAY 2

NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL
S22	S30	S32	NULL	NULL	NULL
S8	S8	S4	S13	S3	S19

SPRING ARRAY 3

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

INITIAL PASS

Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1
Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1
Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1
Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1
Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1
Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1

FIRST ITERATION

Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1
Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1
Σ^2	Σ^2	Σ^2	Σ^2	Σ^2	Σ^1
Σ^2	Σ^2	Σ^2	Σ^2	Σ^2	Σ^2
Σ^2	Σ^2	Σ^2	Σ^2	Σ^2	Σ^2
Σ^2	Σ^2	Σ^2	Σ^2	Σ^2	Σ^2

SECOND ITERATION

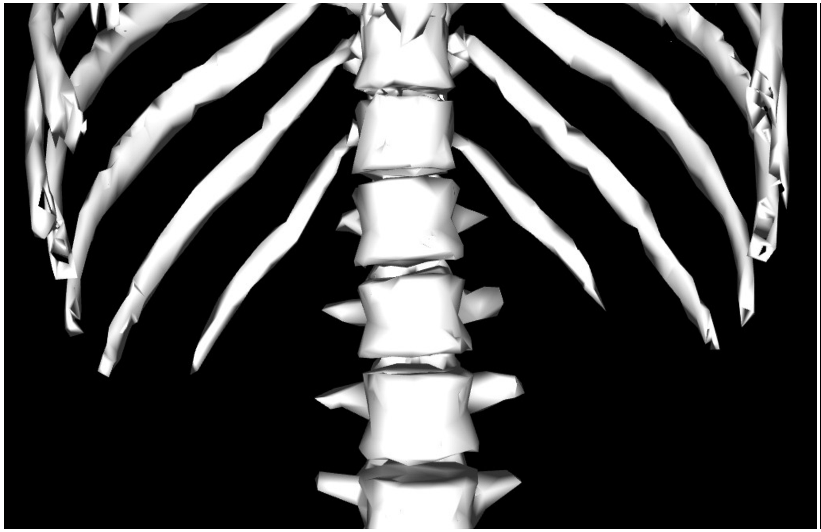
Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1
Σ^1	Σ^1	Σ^1	Σ^1	Σ^1	Σ^1
Σ^2	Σ^2	Σ^2	Σ^2	Σ^2	Σ^1
Σ^2	Σ^2	Σ^2	Σ^2	Σ^2	Σ^2
Σ^3	Σ^3	Σ^3	Σ^2	Σ^2	Σ^2
Σ^3	Σ^3	Σ^3	Σ^3	Σ^3	Σ^3

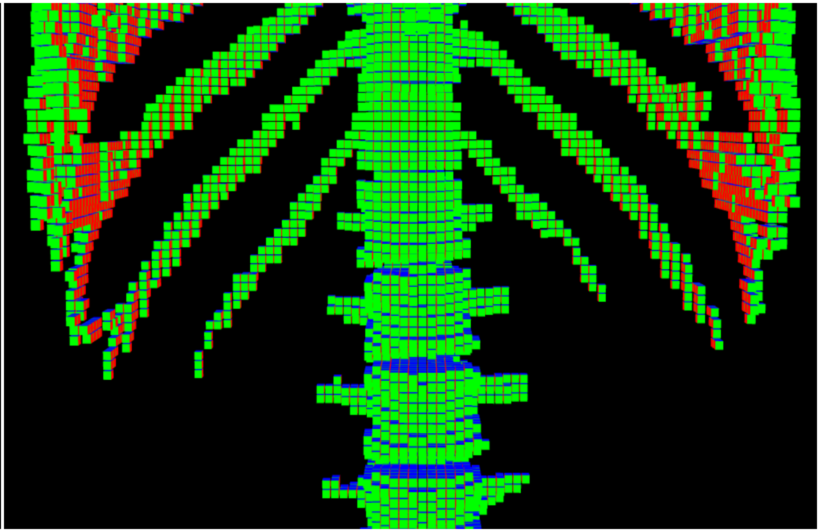
THIRD ITERATION

Collision detection with rigid bodies

We propose a method for collision detection with environment:

- discretize the workspace in a 3D array
- store a boolean value for each element (0=empty, 1=occupied)
- if mass is in a occupied element, resolve the collision



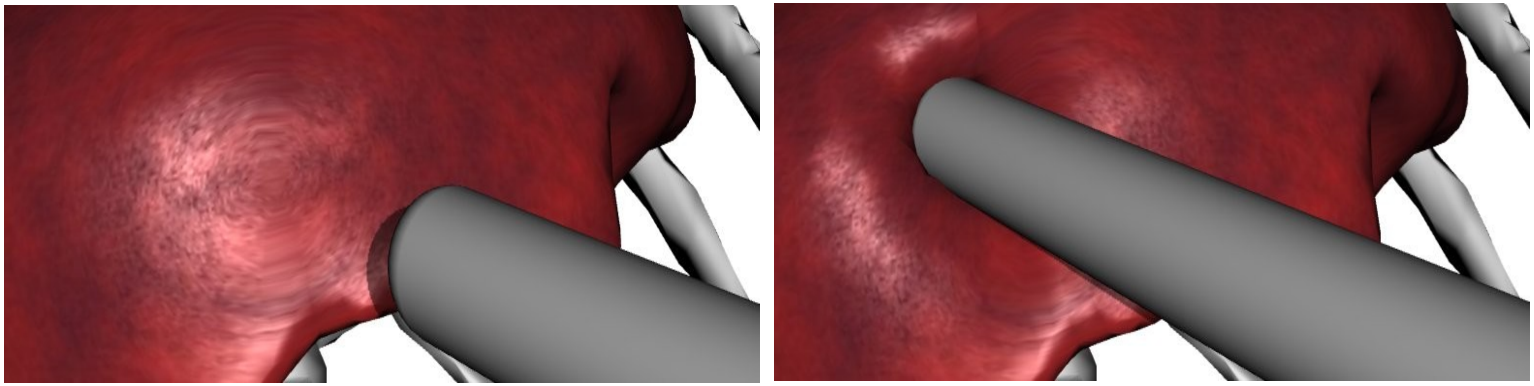


Probing gesture

Probing allows the surgeon to touch the surface of the organ.

Apply a set of kernels to:

- mark the masses colliding with tool surface
- approximate tool surface with ellipsoids or capsules
- resolve collisions translating marked masses on tool surface



Grabbing gesture

Grabbing allows the surgeon to grab parts of organ's surface.

- if tweezers are open apply the probing gesture only
- when the tweezers close, mark all masses in the closing area
- store these markings and the positions relative to the tool
- if tweezers are closed, displace marked masses

Haptic feedback

We propose a method to compute haptic forces on GPU:

- compute the force (or torque) applied by each mass on tools
- accumulate these results by applying the following kernel:

$$f_{i+1}(u,v)=f_i(2u,2v)+f_i(2u+1,2v)+f_i(2u,2v+1)+f_i(2u+1,2v+1)$$

- iterate the kernel halving the sub-array dimensions
- asyncronously transfer the result at (0,0) to the CPU

FORCE ON TOOL ARRAY

0	0	0	0	0	2
7	0	1	1	0	0
0	0	0	3	0	0
0	4	0	0	1	0
3	0	6	0	1	0
8	0	3	0	0	0

ACCUMULATED FORCE ON TOOL ARRAY

FIRST PASS

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
7	2	2	0	0	0
4	3	1	0	0	0
11	9	1	0	0	0

SECOND PASS

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
9	2	0	0	0	0
27	2	0	0	0	0

THIRD PASS

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
40	0	0	0	0	0

Conclusions and future work

We presented here:

- a method to optimize elastic force computation
- a fast method to do collision detection with environment
- two common surgical gestures
- a new and fast method to compute haptic forces entirely on GPU

The entire computation process takes about 0.7 msec on a GeForce8 with the used model (7750 masses, 38077 tetrahedra, 48254 springs).

Future work:

- CUDA porting
- add other surgical gestures

Thank you

Questions?