Final: AlienGLRenderer: Cloth Simulation

Arrian Chi

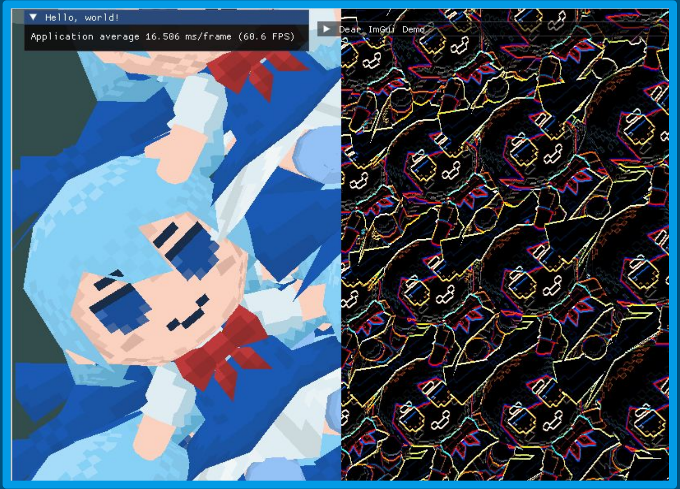
Link to presentation: https://docs.google.com/presentation/d/1kTidNlouo0Z4JUA38zkX3oHS2g0vJ7p8UtcZK0oS1vM/
Link to project: https://github.com/dinoplane/gl_alien_renderer

Inspiration

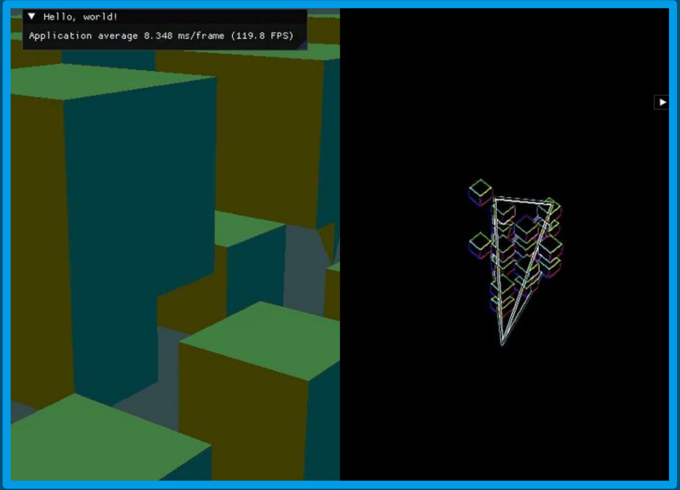
AlienGLRenderer

- A sandbox renderer in OpenGL
  - Streamline creation of simple scenes
  - Help me reinforce CG concepts
- Current features:
  - Post-processing
  - Instancing
  - Frustum culling
  - Gltf Model Loading
  - Scene Loading

I occasionally post my progress here: https://x.com/isacsac2017



```
▼ Hello, world!
Application average 16.506 ms/frame (60.6 FPS)
```



```
▼ Hello, world!
Application average 8.348 ms/frame (119.8 FPS)
```

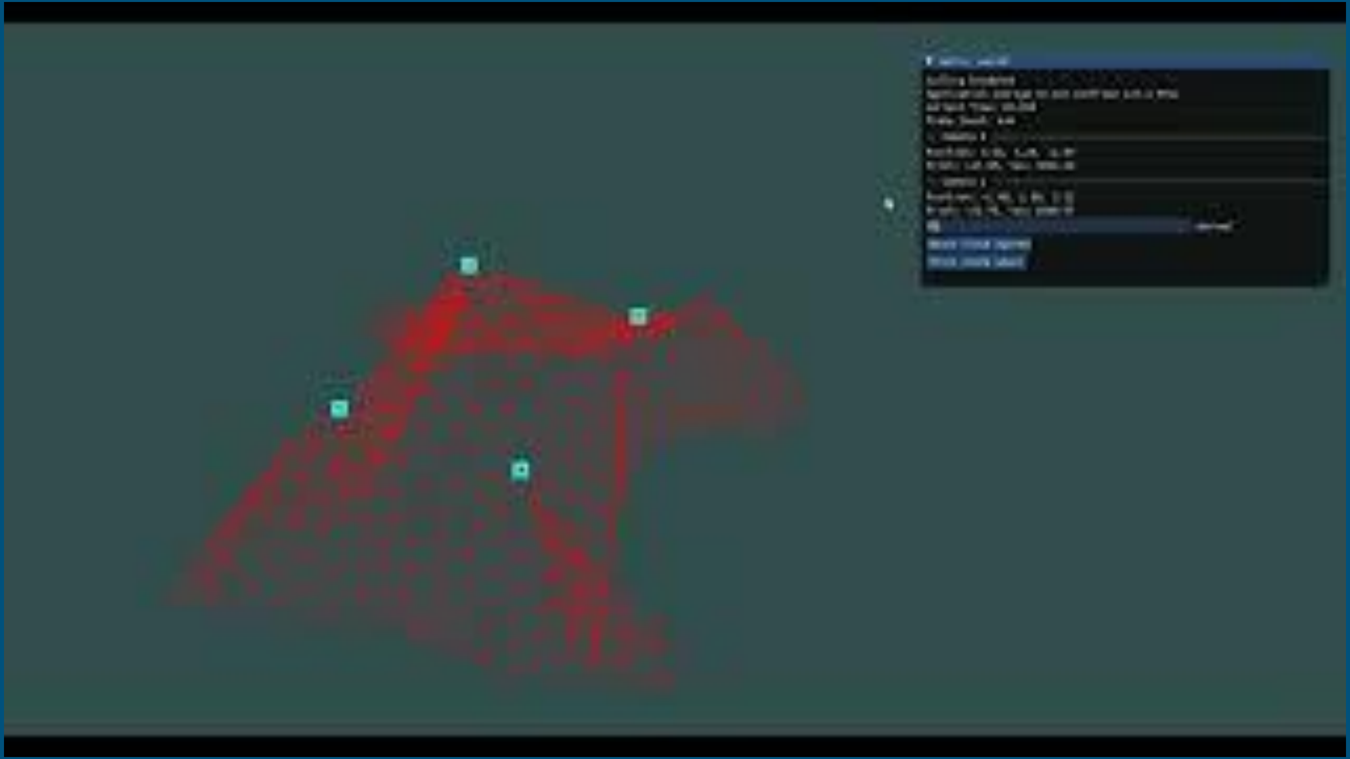
Questions

- How efficiently can we render a cloth with the DEP algorithm?
- How do we implement this?
- What problems arise when implementing the simulation in C++ / OpenGL?

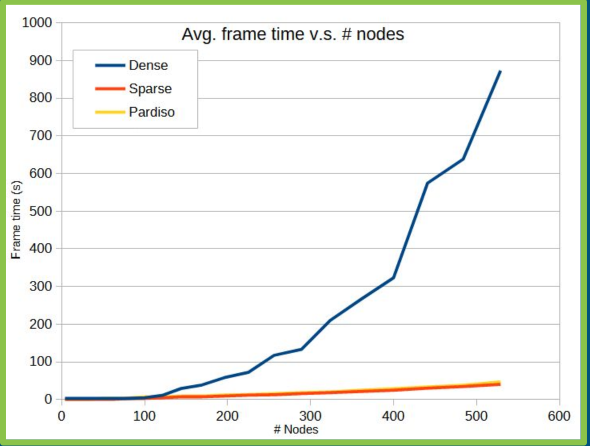
Questions

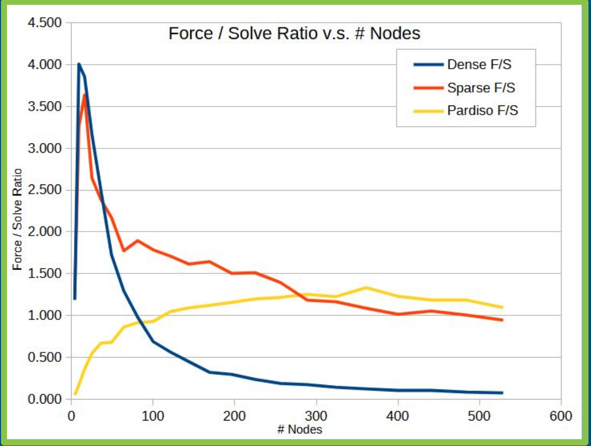
- How efficiently can we render a cloth with the DEP algorithm?
  - Current Best: 256 - 289 nodes (768 - 867 DOFs) at ~30FPS
  - With a slight caveat maybe more?
- How do we implement this?
  - Use sparse matrices
  - Use faster solvers (ONLY IF THEY ARE THE BOTTLENECK)
- What problems arise when implementing the simulation in C++ / OpenGL?
  - Memory alignment
  - API design issues
  - Numerical Degeneracies

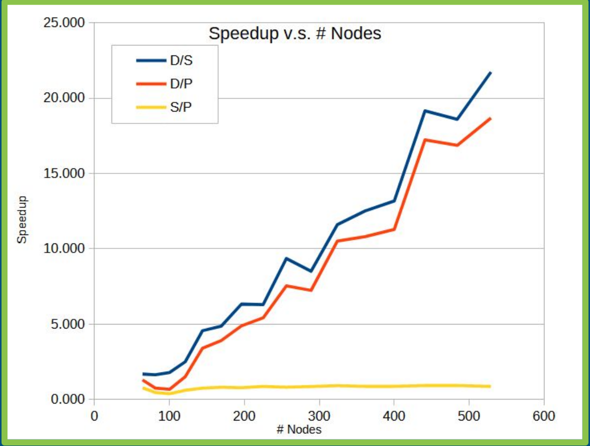
Results



Results - Cont.







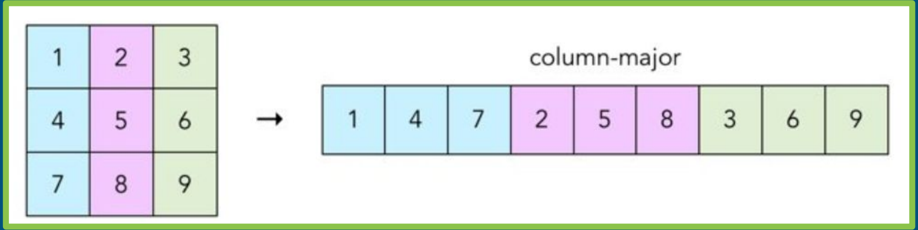
Implementation #1

Transcription of python version

- Numpy -> Eigen
- Dense Matrices
- Double -> floats

~30 FPS: 169 - 196 nodes (507 - 588 DOFs)

529 Nodes: 1.145 FPS



1	2	3
4	5	6
7	8	9

→

column-major

1	4	7	2	5	8	3	6	9

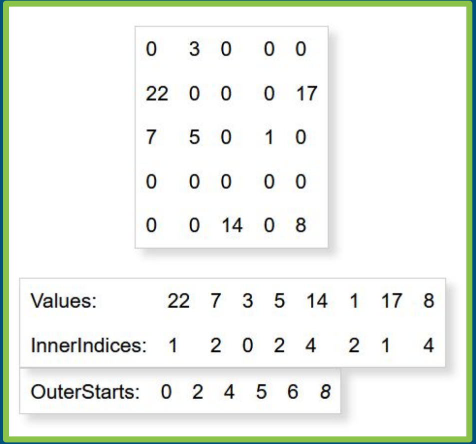
Implementation #2

Use Sparse Matrices

- Floats -> Doubles
- Dense Matrices -> Sparse Matrices
- Not Multithreaded
  - Actually didn't help that much
  - Cache coherency (false sharing)
- Fixed a major bug

~30 FPS: 484 nodes (1575 DOFs)

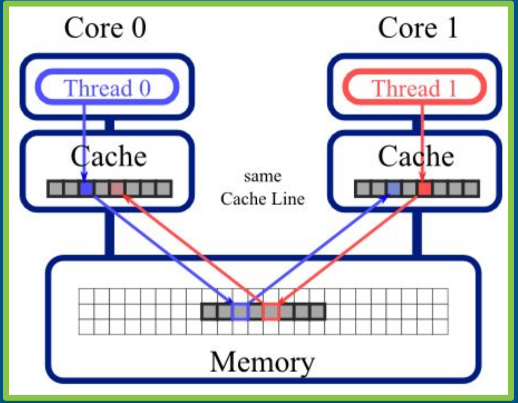
1764 Nodes: 4.098 FPS



0	3	0	0	0
22	0	0	0	17
7	5	0	1	0
0	0	0	0	0
0	0	14	0	8

Values:	22	7	3	5	14	1	17	8
InnerIndices:	1	2	0	2	4	2	1	4

OuterStarts:	0	2	4	5	6	8



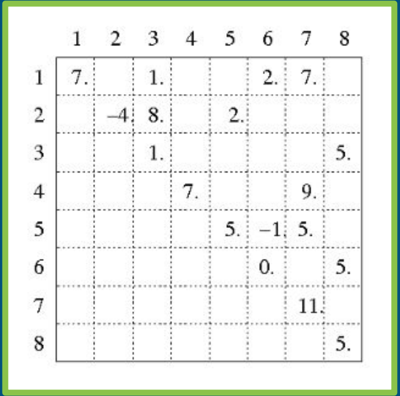
Implementation #3

Third implementation: using PARDISO

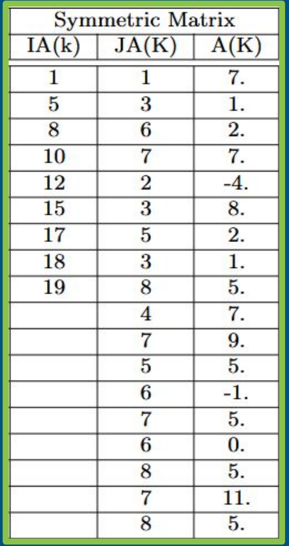
- Full Matrix -> Upper triangular
- Column major -> Row major

~30 FPS: 484 nodes (1575 DOFs)

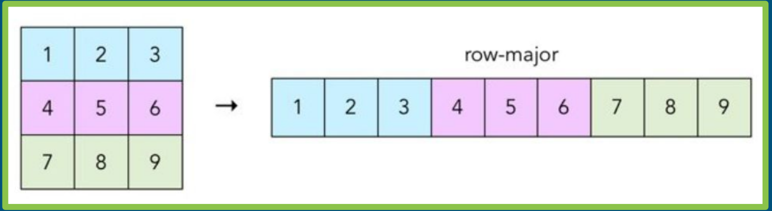
1764 Nodes: 4.435 FPS



	1	2	3	4	5	6	7	8
1	7.		1.			2.	7.	
2		−4	8.		2.			
3			1.					5.
4				7.			9.	
5					5.	−1	5.	
6						0.		5.
7							11.	
8								5.



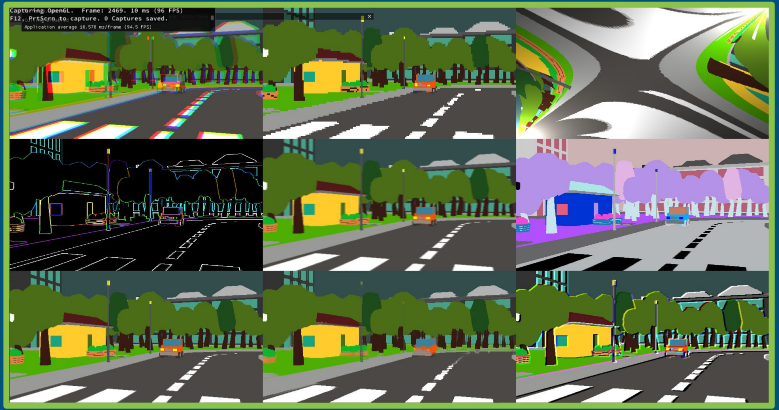
Symmetric Matrix		
IA(k)	JA(K)	A(K)
1	1	7.
5	3	1.
8	6	2.
10	7	7.
12	2	-4.
15	3	8.
17	5	2.
18	3	1.
19	8	5.
	4	7.
	7	9.
	5	5.
	6	-1.
	7	5.
	6	0.
	8	5.
	7	11.
	8	5.



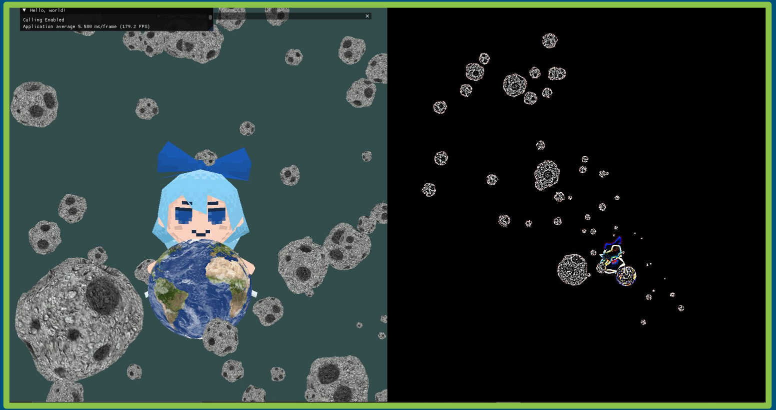
Comments

- OpenGL kinda sucks
  - Memory Alignment
- C++ is hard
  - Template metaprogramming is hard to debug
- Degeneracies
  - Arise when the young modulus is too large, the # nodes too large, the timestep too large etc.
  - Tweaking parameters is needed
- Eigen (reasonably) doesn’t have sparse matrix slicing
  - Develop intuition about how matrix slicing works.

Thank you!



```
Capturing OpenGL.  Frame: 2469. 10 ms (96 FPS)
F12, PrtScrn to capture. 0 Captures saved.
Application average 10.578 ms/frame (94.5 FPS)
```



```
▼ Hello, world!
Culling Enabled
Application average 5.580 ms/frame (179.2 FPS)
```

References

1] D. Baraff and A. Witkin, Large Steps in Cloth Simulation. New York, NY, USA: Association for Computing Machinery, 1 ed., 1998.

[2] E. Grinspun, A. N. Hirani, M. Desbrun, and P. Schr¨oder, “Discrete shells,” in Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA ’03, (Goslar, DEU), p. 62–67, Eurographics Association, 2003.

[3] R. Tamstorf and E. Grinspun, “Discrete bending forces and their jacobians,” Graph. Models, vol. 75, p. 362–370, Nov. 2013.

[4] J. Weil, “The synthesis of cloth objects,” ACM Siggraph Computer Graphics, vol. 20, no. 4, pp. 49–54, 1986.

[5] X. Provot et al., “Deformation constraints in a mass-spring model to describe rigid cloth behaviour,” 1995.

[6] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, “Elastically deformable models,” SIGGRAPH Comput. Graph., vol. 21, p. 205–214, Aug. 1987.

[7] S. MK Jawed, Lim, “Discrete simulation of slender structures,” 2023.

References

[8] G. Guennebaud, B. Jacob, et al., Eigen: A C++ template library for linear algebra. Eigen Contributors, 2024. Accessed: 2024-11-20.

[9] L. K. Contributors, False Sharing in the Linux Kernel, 2024. Accessed: 2024-11-20.

[10] D. Pasadakis, M. Bollh¨ofer, and O. Schenk, “Sparse quadratic approximation for graph learning,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 9, pp. 11256–11269, 2023.

[11] A. Eftekhari, D. Pasadakis, M. Bollh¨ofer, S. Scheidegger, and O. Schenk, “Block-enhanced precision matrix estimation for large-scale datasets,” Journal of Computational Science, vol. 53, p. 101389, 2021.

[12] L. Gaedke-Merzh¨auser, J. van Niekerk, O. Schenk, and H. Rue, “Parallelized integrated nested laplace approximations for fast bayesian inference,” 2022.

[13] C. Caruth, “There are no zero-cost abstractions.” CPPCon 2019, 2019.

[14] Khronos Group, OpenGL Documentation. Khronos Group. Accessed: 2024-11-20.