Better Collisions and Faster Cloth for Pixar's *Coco*

David Eberle
Pixar Animation Studios

ABSTRACT

Among the many technical challenges of Pixar's *Coco* was the need to handle cloth simulation for a densely populated city of skeleton characters. Skeletons posed new challenges to the collision algorithms of our in-house cloth system, Fizt. Continuous collision detection and response is an obvious solution to handling fast motion of thin geometry, but it presents us with a serious problem. In our production pipeline, geometry often starts in intersection. Animation also frequently causes kinematic surfaces to pinch the cloth between them and drive the cloth through itself. We present a solution for robustly allowing intersection recovery while employing standard continuous detection techniques.

Coco also demanded more cloth than any previous Pixar film. To keep up with demand, Fizt needed to run much faster. We share our techniques for gaining performance in linear system assembly and solution, which should be applicable to most implicit solvers.

CCS CONCEPTS

• **Computing methodologies → Physical simulation**;

KEYWORDS

Cloth simulation, continuous collision, sparse-matrix assembly

ACM Reference format:
David Eberle. 2018. Better Collisions and Faster Cloth for Pixar's *Coco*. In *Proceedings of SIGGRAPH '18 Talks, Vancouver, BC, Canada, August 12-16, 2018*, 2 pages.
https://doi.org/10.1145/3214745.3214801

1 HISTORICAL APPROACHES

Before *Coco*, the Fizt simulator (based on Baraff and Witkin [1998]) had been used for over fifteen films at Pixar. Collision detection was discrete, and one of the most important features of Fizt was its ability to gracefully recover from common unwanted intersections. Typical human limbs are much thicker than the skeleton geometry on *Coco*. Previously, if penetrations occurred, they were usually not very deep relative to the limb thickness and recovery was often possible by the end of a frame. Residual visual artifacts were handled by manual post-sim fixes or by making the character geometry invisible beneath slightly penetrating cloth regions. In rare cases of extremely fast motion, the number of simulation time steps could be increased to resolve the issue.

These prior techniques would not scale for *Coco*. Besides the amount of simulation that would probably require fixes, fast motion was very common. This, combined with the fact that skeleton geometry was so thin, made it possible for cloth to pass entirely through a limb in one time step and then never recover. Art direction required that the audience "feel" the bones through the cloth, so using large collision offsets was not an option.

Continuous collision detection **(CCD)** [Bridson et al. 2002; Provot 1995] was added to Fizt at the end of each time step to handle the thin, fast-moving geometry of *Coco*'s skeletons. To our knowledge, all CCD approaches assume that geometry starts in an intersection-free state and then strives to maintain it. Substantial work [Brochu et al. 2012; Tang et al. 2014; Wang 2014] has been devoted to providing numerically robust solutions to this task. Our artists were accustomed to starting with intersections and having them recover. Requiring a clean state or repairing body intersections over the shots would have been a costly disruption to their workflows.

2 OUR COLLISION SOLUTION

We needed a way to robustly allow CCD to prevent intersections while not interfering with recovery. For cloth/kinematic collisions, we tried a variety of reasonable heuristic techniques, but numerical issues would sometimes lead to discarding valid collisions.

The Global Intersection Analysis **(GIA)** algorithm presented in Baraff and Witkin [2003] provided us with a superior solution. The coloring from the flood fill of the GIA algorithm labels the intersecting mesh features. When CCD is performed between two mesh features, we can ignore ones already in intersection. This gives us the desired "one way membrane" behavior. When no intersection exists, CCD is performed and conversely is disabled between regions of intersection to allow recovery. We perform GIA between the positional states of all kinematic mesh pairs at the end of a time step. This information is stored for the next step to provide GIA information at the current position state. This way we know how the intersection status between kinematic geometry changes over the step. We disable CCD between cloth that collides with kinematic geometry that is pinched, becoming pinched or becoming unpinched. We also compute and utilize GIA information between all cloth/cloth and cloth/kinematic pairs at the current position state.

We also use GIA to better inform cloth/body collision constraint response from proximity queries. Particles too deep inside body geometry can be ejected out the wrong side by response. We examine the GIA coloring data of an inside particle and that of the face it would exit through. If both are not part of a dynamic/kinematic coloring, we omit the response. Particles closer to the actual contour of intersection will get response and, over time, the cloth will exit from the correct side. This was useful to correctly handle initially penetrating configurations on thin skeleton limbs. We've concluded that enabling communication among Intersection/GIA, Proximity Query, and CCD algorithms can enable better response decisions.

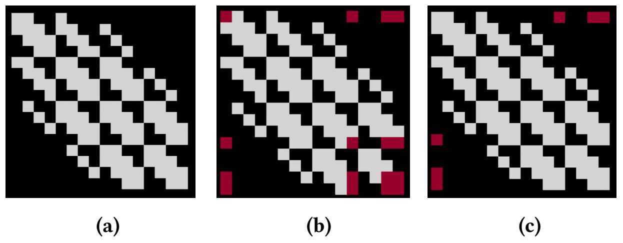

Figure 1: (a) Fixed sparsity pattern (FGSM), (b) Red entries of a transient contact between vertex 0 and face(13,14,15), (c) Red entries remaining in TGSM after merging with FGSM.

3 SOLVER PERFORMANCE

To improve linear system assembly performance, we attempted to reduce data traffic and make our parallel algorithm lock-free. We exploit the fact that our cloth meshes do not change connectivity during simulation and have a fixed sparsity pattern. Forces with Jacobians that persist over the simulation are called FixedForceElements **(FFE)**. FFEs have their own local storage for the forces and Jacobians. This allows them to be computed in parallel without any locks. The accumulation of the local Jacobians to a fixed pattern global sparse matrix **(FGSM)** happens in another lock-free parallel pass over the global blocks. This is a fairly standard procedure in parallel finite element codes. To reduce the number of elements that need to be traversed during these different phases we found inspiration in Curtis et al. [2008]. In our system, an FFE can index up to four particles and can therefore aggregate many different forces, such as stretch and shear. Some FFEs may cover two triangles and can also support a dihedral bending force. Our per-particle forces are also uniquely assigned to the FFEs. Our assignment procedure is simple and reduces the total number of elements needed by approximately one third, thereby reducing the memory bandwidth demands of both passes. Transient forces between particles with Jacobians, like contact penalty forces or toggled user springs, are called TransientForceElements **(TFE)**. The force gradients from TFEs are accumulated into a transient global sparse matrix **(TGSM)**. Each TGSM row has a per-thread structure for accumulating the transient entries in lock-free fashion. A serial reduction pass is performed on each TGSM row in parallel over all the rows. As the number of threads increases, this reduction has the potential to create a new bottleneck, but when running with 4-8 threads, we didn't observe adverse effects.

We perform another parallel pass over the rows of our TGSM to transfer any overlapping entries from it to the block entries of the FGSM (Figure 1). The FGSM and TGSM structures are converted to the standard block compressed row storage **(BCRS)** format. We cast the values to single precision in the BCRS structures to reduce the memory bandwidth during the solver's sparse matrix-vector multiply. Instead of the traditional linear system in Baraff and Witkin [1998], we use the pre-filtered form in Tamstorf et al. [2015].

We explored a variety of techniques to gain solver performance. Our global particle indices are reindexed once by a Reverse Cuthill-McKee ordering at the start of our simulation, using the fixed connectivity. This improves memory access for the solver by reducing the matrix bandwidth of the FGSM. This ordering also allows the column indices in the BCRS of the FGSM to be stored as `short` offsets from the diagonal. The column indices of the BCRS of the TGSM are `unsigned int` to accommodate transient interaction between any particles. Our iterative solver traverses the rows of our system in parallel many times. Only some rows have contributions from the TGSM. At each time step, we establish an array of appropriate function pointers to traverse the entries for each row. We also employ loop unrolling to process four blocks at a time when traversing each row. The indices for the end and remainder portions of these unrolled traversals are computed and stored once for the FGSM BCRS and at each time step for the TGSM BCRS.

Table 1: Average performance of Fizt's system assembly and solver before and after our work. Times are in seconds and all examples were run with 6 threads at 10 steps per frame.

Example	Build	Solve	Sim Per Frame
Miguel	2.07 vs. 0.60	3.57 vs. 0.65	8.50 vs. 3.26
Nana Coco	2.45 vs. 0.88	1.77 vs. 0.46	9.47 vs. 4.97
Skeleton In Dress	4.84 vs. 1.74	6.86 vs. 1.51	15.9 vs. 6.76
Miguel's Mother	2.98 vs. 1.10	4.90 vs. 1.23	12.0 vs. 5.32

4 RESULTS

We achieved a ~3X improvement on linear system assembly and a ~4x increase in the solver performance (Table 1). The original system assembly traversed elements individually and employed locks when building the global matrix. The addition of continuous collisions and more GIA computation took back some of these gains, leaving us with a ~2X gain to the average time per-frame. The combined robustness and speed improvements made the complexity and volume of cloth simulation more manageable for *Coco*. These enhancements also made it tractable for our animators to run simulations where a character's performance dictated expressive interaction with their garments.

ACKNOWLEDGMENTS

Thanks to Theodore Kim and Andrew Butts for their valuable feedback on this abstract, Hayley Iben for her support in this endeavor, and the Coco Sim/Tailoring team for their outstanding work.

REFERENCES

D. Baraff and A. Witkin. 1998. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM, 12.

D. Baraff and A. Witkin. 2003. Untangling Cloth. In *ACM SIGGRAPH 2003 Papers (SIGGRAPH '03)*. ACM, 9.

R. Bridson, R. Fedkiw, and J. Anderson. 2002. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '02)*. ACM, 594–603.

T. Brochu, E. Edwards, and R. Bridson. 2012. Efficient Geometrically Exact Continuous Collision Detection. *ACM Trans. Graph.* 31, 4, Article 96 (2012), 7 pages.

S. Curtis, R. Tamstorf, and D. Manocha. 2008. Fast Collision Detection for Deformable Models Using Representative-triangles. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games (I3D '08)*. ACM, 9.

X. Provot. 1995. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface* (1995), 12.

R. Tamstorf, T. Jones, and S. McCormick. 2015. Smoothed Aggregation Multigrid for Cloth Simulation. *ACM Trans. Graph.* 34, 6, Article 245 (2015), 13 pages.

M. Tang, R. Tong, Z. Wang, and D. Manocha. 2014. Fast and Exact Continuous Collision Detection with Bernstein Sign Classification. *ACM Trans. Graph.* 33, 6, Article 186 (2014), 8 pages.

H. Wang. 2014. Defending Continuous Collision Detection Against Errors. *ACM Trans. Graph.* 33, 4, Article 122 (2014), 10 pages.