Dynamic Deformation Textures: GPU-accelerated Simulation of Deformable Models in Contact

Nico Galoppo[+] Miguel A. Otaduy[⋆] Paul Mecklenburg[+] Markus Gross[⋆] Ming C. Lin[+]

[+]{nico,prm,lin}@cs.unc.edu, UNC Chapel Hill [⋆]{motaduy,grossm}@inf.ethz.ch, ETH Zurich

1 Introduction

We present an efficient algorithm for simulating contacts between deformable bodies with high-resolution surface geometry using *dynamic deformation textures*, which reformulate the 3D elastoplastic deformation and collision handling on a 2D parametric atlas to reduce the extremely high number of degrees of freedom arising from large contact regions and high-resolution geometry. Such computationally challenging dynamic contact scenarios arise when objects with rich surface geometry are rubbed against each other while they bounce, roll or slide through the scene, as shown in Figure 1.

We simulate real-world deformable solids that can be modeled as a rigid core covered by a layer of deformable material [Terzopoulos and Witkin 1988], assuming that the deformation field of the surface can be expressed as a function in the parametric domain of the rigid core. Examples include animated characters, furniture, toys, tires, etc.

We have developed novel and efficient solutions for physically-based simulation of dynamic deformations, as well as for collision detection and robust contact response, by exploiting the layered representation of the models and decoupling the degrees of freedom between the core and the deformation layers.

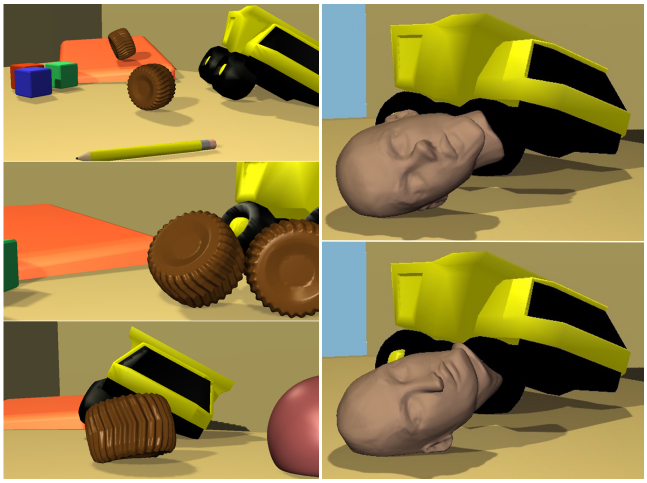

Figure 1: Soft Object Interaction in a Dynamic Scene.

2 Overview

Our mathematical formulation of dynamic simulation and contact processing, along with the use of dynamic deformation textures, is especially well suited for realization on commodity SIMD or parallel architectures, such as graphics processing units (GPU), Cell processors, and physics processing units (PPU). More in particular, the following key concepts contribute to the mapping of our algorithm to the GPU architecture, resulting in the effectiveness and efficiency of our algorithm:

- We reformulate the 3-dimensional elastoplastic deformations and collision processing on 2-dimensional *dynamic deformation textures*. This mapping is illustrated in Figures 2 and 3, with the 2D computational domains indicated by T and D. There is a natural mapping between the computational domains and graphics hardware textures.
- Using a two-stage collision detection algorithm for parameterized layered deformable models, our proximity queries are scalable and output-sensitive, i.e. the performance of the queries does not directly depend on the complexity of the surface meshes. We perform high resolution collision detection with an image based collision detection algorithm, implemented on the GPU.
- By decoupling the parallel update of surface displacements and parallel constraint-based collision response from the update of the core DoFs, we provide fast and responsive simulations under large time steps on heterogeneous materials. We have implemented this parallel implicit contact resolution method on the GPU, thereby exploiting the inherent parallellism of the GPU architecture.

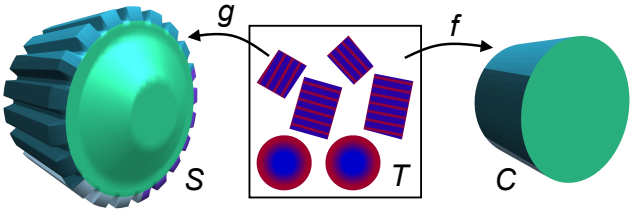

Figure 2: Deformable Object Representation. Deformable surface S (52K triangles) and core C (252 triangles) of a gear, showing the patch partitioning of its parameterization. The common color coding reflects the mapping $g \circ f^{-1} : C \rightarrow S$. The dynamic deformation texture T (256×256) stores the displacement field values on the surface. The gear contains 28K simulation nodes on the surface and 161K tetrahedra, allowing the simulation of highly detailed deformations.

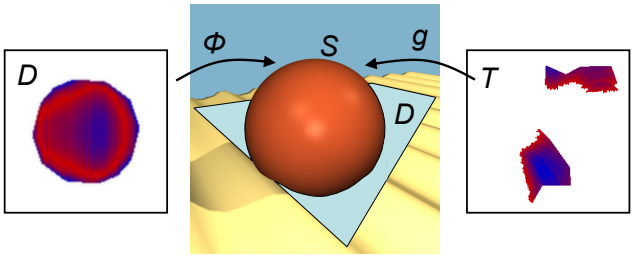

Figure 3: Texture-Based Collision Detection Process. Center: A sphere S collides with a textured terrain. Left: Contact plane D for texture-based collision detection, and mapping $\phi : D \rightarrow S$. The contact plane shows the penetration depth. Right: Dynamic deformation texture T, and mapping $g : T \rightarrow S$. The penetration depth is projected from D to T, and is available for collision response.

3 Algorithm and Parallel Implementation

The implicit formulation of the dynamic motion equations and collision response yields linear systems of equations with dense coupling between the core and elastic velocities. However, we can formulate the velocity update and collision response in a highly parallelizable manner [Galoppo et al.]. In Fig. 4 we outline the entire algorithm for simulating deformable objects in contact using dynamic deformation textures. We refer the interested reader to [Galoppo et al.] for details of the equations. Let s denote the opera-

tions that are performed on small-sized systems (i.e., computations of core variables, and low resolution collision detection). The remaining operations are all executed in a parallel manner on a large number of simulation nodes. Specifically, T refers to operations to be executed on all simulation nodes in the dynamic deformation texture T, D refers to operations to be executed on texels of the contact plane D, and T_D refers to operations to be executed on the colliding nodes.

	COLLISION-FREE UPDATE	
1.	Evaluate forces	T
2.	Solve the sparse linear systems $\tilde{\mathbf{M}}_e\mathbf{y}=\tilde{\mathbf{F}}_e$ and $\tilde{\mathbf{M}}_e\mathbf{Y}=\tilde{\mathbf{M}}_{ec}$ ([Galoppo et al.]), using a Conjugate Gradient solver [Golub and Van Loan 1996]	T
3.	Update core velocities $\mathbf{v}_c^-$	s
4.	Update elastic velocities $\mathbf{v}_e^-$	T
5.	Perform a position update $\mathbf{q}^- = \mathbf{q}(t)+\Delta t\mathbf{P}^+\mathbf{v}^-$	T
	COLLISION DETECTION	
6.	Execute low-resolution collision detection	s
7.	Compute penetration depth and contact normal	D
8.	Map contact information to the dynamic deformation textures	T
	COLLISION RESPONSE	
9.	Invert the block-diagonalized full-rank matrix $\mathbf{J}_e\tilde{\mathbf{M}}_e^{-1}\mathbf{J}_e^T$	T_D
10.	Solve for Lagrange-multipliers λ using the Sherman-Morrison-Woodbury formula	T_D
11.	Repeat steps 3 and 4 to obtain the collision impulse $\delta\mathbf{v}$	
12.	Compute friction impulse	T_D
13.	Perform a position update $\mathbf{q}(t+\Delta t) = \mathbf{q}^- + \Delta t\mathbf{P}^+(\delta\mathbf{v})$	T
	CONSTRAINT CORRECTION	
14.	Repeat collision detection steps 6 to 8	
15.	Apply constraint correction	T_D

Figure 4: Summary of Our Simulation Algorithm

As highlighted in the algorithm above, all operations to be executed on simulation nodes can be implemented with parallelizable computation stencils. Moreover, due to the regular meshing of the deformable layer, the computation stencils are uniform across all nodes. Boundaries between different texture patches, however, require special treatment in order to allow the use of uniform stencils. We adapt a method by Stam [2003] for providing accessible data in an 8-neighborhood to all nodes located on patch boundaries. Before every sparse matrix multiplication step in the algorithm, we fill a $\sqrt{2}$-texel-width region on patch boundaries by sampling values on the adjacent patches. Note that this sampling step removes the perfect symmetry of the large sparse linear systems to be solved, but in practice we have only found a slight decrease in the convergence of the Conjugate Gradient solver.

Due to its parallellizable nature, we were able to accelerate major parts of the algorithm on the GPU. In particular, all parts indicated by T, T_D and D were implemented on the GPU. Dynamic deformation textures map naturally to graphics memory textures, thus the elastic displacements $\mathbf{q}_e$ and velocities $\mathbf{v}_e$ associated with FEM nodes in the deformable layer are stored and updated directly in the GPU. Similarly, we exploit the implicit FEM mesh defined by the regular sampling of dynamic deformation textures for storing the stiffness coefficients local to the nodes in auxiliary GPU textures. The updates of elastic displacements and velocities (marked with T and T_D in the algorithm in Fig. 4) are executed by performing shader operations on fragment programs, exploiting Frame Buffer Objects for direct computations on textures. The updates of core velocities, on the other hand, are executed in the CPU after gathering intermediate computations performed in parallel on all nodes. Note that the communication between CPU and GPU is reduced to issuing commands and small data packets whose size is determined by the number of DoFs of the core. The efficient GPU-implementation of gather operations and sparse matrix multiplications, also required in the solution of linear systems using the Conjugate Gradient method, has been explored before [Kruger and Westermann 2003]. We exploit the Multiple Render Target extension to implement the Sherman-Morrison-Woodbury update, enabling the output of 3×3 matrices in one pass.

For the execution of collision detection, we also exploit image-based computations on the GPU. The computations of per-texel penetration depth and contact normal are performed by orthonormal projection of the low-resolution core geometry onto the contact plane D, and by using texture mapping to map the positions of the high-resolution surfaces to D. This projection is also used in shadow mapping-alike technique to obtain the inverse mapping, from the contact plane D back to the dynamic deformation texture T for contact response.

4 Results

In Figure 1, we show a scene where deformable tires with high-resolution features on their surfaces roll, bounce, and collide with each other. This simulation consists of 324K tetrahedra and 62K surface simulation nodes. Such high resolution enables the simulation of rich deformations, as shown in the accompanying video. All contacts on the surface have global effect on the entire deformable layer, they are processed simultaneously and robustly. Without any precomputation of dynamics or significant storage requirement, we were able to simulate this scene, processing over 15,000 contacts per second, on a 3.4 GHz P4 with NVidia GeForce 7800.

Our approach is considerably faster than other methods that enable large time steps, such as those that focus on the surface deformation and corotational methods that compute deformations within the entire volume, with more stable collision response. Our approach can also handle many more contact points than novel quasi-rigid dynamics algorithms using LCP [Pauly et al. 2004], while producing richer deformations, between moving objects (Figure 5).

Acknowledgements: This work was partly funded by the following agencies: NSF, DARPA, RDECOM, ARO and ONR.

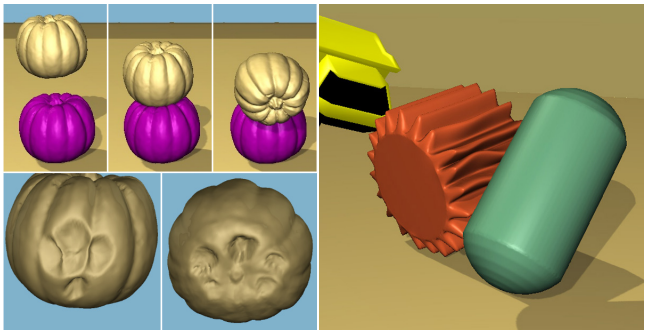

Figure 5: Rich Deformation of Detailed Geometry. *Bottom-left corner: views from below the upper pumpkin as it collides with the bottom pumpkin and deforms.*

References

GALOPPO, N., OTADUY, M. A., MECKLENBURG, P., GROSS, M., AND LIN, M. C. Fast simulation of deformable models in contact using dynamic deformation textures. Technical Report, UNC Chapel Hill - ETH Zurich.

GOLUB, G. H., AND VAN LOAN, C. F. 1996. *Matrix Computations*, 3rd ed. Johns Hopkins University Press.

KRUGER, J., AND WESTERMANN, R. 2003. Linear algebra operators for GPU implementation of numerical algorithms. In *Proc. of ACM SIGGRAPH*.

PAULY, M., PAI, D. K., AND GUIBAS, L. J. 2004. Quasi-rigid objects in contact. In *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

STAM, J. 2003. Flow on surfaces of arbitrary topology. In *Proc. of ACM SIGGRAPH*.

TERZOPOULOS, D., AND WITKIN, A. 1988. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications 8*, 6.