GPU-Based Contact-Aware Trajectory Optimization Using A Smooth Force Model (Supplementary Document)

ZHERONG PAN*, University of North Carolina at Chapel Hill, USA
BO REN, Nankai University, China
DINESH MANOCHA, University of Maryland at College Park, USA

CCS Concepts: • **Computing methodologies** → **Physical simulation**.

Additional Key Words and Phrases: trajectory optimization, articulated bodies, deformable bodies, position-based dynamics

ACM Reference Format:
Zherong Pan, Bo Ren, and Dinesh Manocha. 2019. GPU-Based Contact-Aware Trajectory Optimization Using A Smooth Force Model (Supplementary Document). 1, 1 (May 2019), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

A DERIVATIVES OF THE FORWARD DYNAMICS FUNCTION

In this section, we summarize the three-stage adjoint method to compute $\mathbf{FD}_i$ and $\partial\mathbf{FD}_i/\partial(\mathbf{x}_i,\mathbf{x}_{i-1},\mathbf{x}_{i-2})$. Note that since we use [Pan and Manocha 2018], our dynamic system is discretized in the Cartesian space using finite differences (Equation 4), unlike a previous method that discretizes the dynamic system in the configuration space via the chain rule (Equation 3). Also note that our external force model (Equation 6) and self collision force model $(-\partial\mathcal{P}_{\text{self}}(\mathbf{x}_i)/\partial\mathbf{x}_i)$ are also functions of Cartesian coordinates only. However, the joint limit force model $(-\partial\mathcal{P}_{\text{joint}}(\mathbf{x}_i)/\partial\mathbf{x}_i)$ is not a function of Cartesian coordinates and we compute it in a separate algorithm. Assuming no joint limits, we can rewrite the forward dynamics function as a function of the rigid transformations:

$$\mathbf{FD}_i\left(\mathbf{x}_i,\mathbf{x}_{i-1},\mathbf{x}_{i-2}\right) =$$
$$\mathbf{FD}_i\left(\mathbf{R}_k(\mathbf{x}_i),\mathbf{t}_k(\mathbf{x}_i),\mathbf{R}_k(\mathbf{x}_{i-1}),\mathbf{t}_k(\mathbf{x}_{i-1}),\mathbf{R}_k(\mathbf{x}_{i-2}),\mathbf{t}_k(\mathbf{x}_{i-2})\right).$$

This rewriting is the key feature of our forward dynamics function that allows us to compute all the derivatives analytically using a three-stage algorithm.

In the first stage, we use a forward pass to compute $\mathbf{R}_k(\mathbf{x}_i),\mathbf{t}_k(\mathbf{x}_i)$ for all k and i. The computations for different k and i can run in parallel. If we have KN threads, then the computation takes $O(K)$. Further optimization can reduce this cost to $O(\mathbf{log}(K))$ [Yang et al. 2017] but we have not implemented this strategy yet. In the second stage, we compute:

$$\mathbf{FD}_i \text{ and } \frac{\partial\mathbf{FD}_i}{\partial(\mathbf{R}_k(\mathbf{x}_i),\mathbf{t}_k(\mathbf{x}_i),\mathbf{R}_k(\mathbf{x}_{i-1}),\mathbf{t}_k(\mathbf{x}_{i-1}),\mathbf{R}_k(\mathbf{x}_{i-2}),\mathbf{t}_k(\mathbf{x}_{i-2}))},$$

which involves GPU-based collision detection [Lauterbach et al. 2009], the complexity of which is hard to bound, but the computations for different $\mathbf{FD}_i$ can run in parallel if we have N threads. In the third stage, we use a backward pass to compute:

$$\partial\mathbf{FD}_i/\partial(\mathbf{x}_i,\mathbf{x}_{i-1},\mathbf{x}_{i-2}) =$$
$$\frac{\partial\mathbf{FD}_i}{\partial(\mathbf{R}_k(\mathbf{x}_i),\mathbf{t}_k(\mathbf{x}_i),\mathbf{R}_k(\mathbf{x}_{i-1}),\mathbf{t}_k(\mathbf{x}_{i-1}),\mathbf{R}_k(\mathbf{x}_{i-2}),\mathbf{t}_k(\mathbf{x}_{i-2}))}$$
$$\frac{\partial(\mathbf{R}_k(\mathbf{x}_i),\mathbf{t}_k(\mathbf{x}_i),\mathbf{R}_k(\mathbf{x}_{i-1}),\mathbf{t}_k(\mathbf{x}_{i-1}),\mathbf{R}_k(\mathbf{x}_{i-2}),\mathbf{t}_k(\mathbf{x}_{i-2}))}{\partial(\mathbf{x}_i,\mathbf{x}_{i-1},\mathbf{x}_{i-2})}.$$

Again, the computations for different $\mathbf{FD}_i$ can run in parallel. For each $\mathbf{FD}_i$, if we have $3K$ threads, the complexity of the backward pass is $O(K)$ using the parallel algorithm in [Pan and Manocha 2018]. In summary, the third stage has complexity $O(K)$ if we have $3KN$ threads.

B ILLUSTRATION OF THE MATRIX OPERATIONS

In this section, we illustrate the compact storage format for **BLT** and **SBT** matrices in Figure 1. We illustrate the even-odd permutation and division of the BCR algorithm in Figure 2. Finally, in Figure 3, we illustrate the permutation to make the left-hand-side of the KKT system Equation 12 in $\mathbf{SBT}(4(2|\mathbf{x}|-|\mathbf{u}|))$.

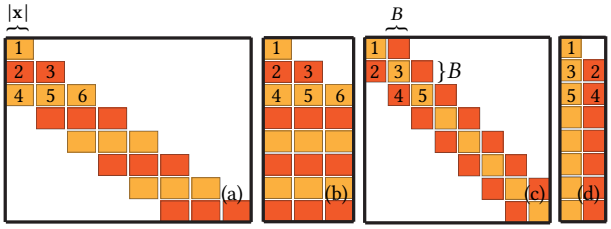

Fig. 1. Illustrations of the two kinds of sparse matrices with special structures. (a): $\mathbf{BLT}(3)$ in its sparse form. (b): $\mathbf{BLT}(3)$ in its compact form where each row has at most 3 blocks and each block has $|\mathbf{x}|$ columns. (c): $\mathbf{SBT}(B)$ in its sparse form where each block is of size $B\times B$. (d): $\mathbf{SBT}(B)$ in its compact form where we only store the lower blocks. The first few blocks are labeled for correspondence.

C PARALLEL MATRIX VECTOR COMPUTATION

We take the following steps to compute all the matrices and vectors required by Line 3 of Algorithm 1:

- We compute $\mathbf{FD}_i$ and $\partial\mathbf{FD}_i/\partial(\mathbf{x}_i,\mathbf{x}_{i-1},\mathbf{x}_{i-2})$ for all i in parallel.
- We assume objective functions are sum-of-squares so that $\mathbf{E}(\mathcal{T}_\mathbf{x})=\mathbf{J}(\mathcal{T}_\mathbf{x})^T\mathbf{J}(\mathcal{T}_\mathbf{x})$ and $\frac{\partial\mathbf{J}(\mathcal{T}_\mathbf{x})}{\partial\mathcal{T}_\mathbf{x}}\in\mathbf{BLT}(3)$. We compute each

*corresponding author

Authors' addresses: Zherong Pan, University of North Carolina at Chapel Hill, Sitterson Hall, Chapel Hill, NC, 27514, USA, zherong@cs.unc.edu; Bo Ren, Nankai University, 94 Weijin Rd, Tianjin, 300071, China, rb@nankai.edu.cn; Dinesh Manocha, University of Maryland at College Park, A.V. Williams Building, 8223 Paint Branch Drive, College Park, MD, 20742, USA, dm@cs.unc.edu.

2019. XXXX-XXXX/2019/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

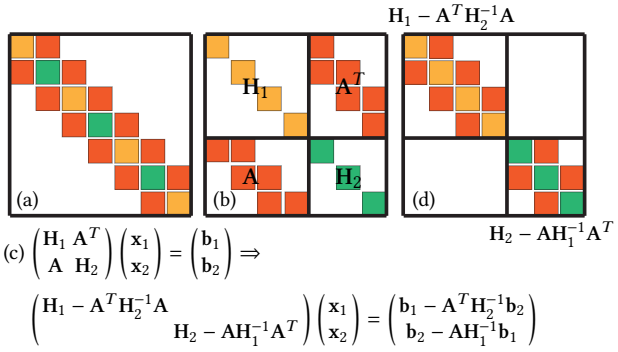

Fig. 2. An illustration of the BCR algorithm for an $\mathbf{SBT}(B)$ matrix with $D = 7$ diagonal blocks (a). The BCR algorithm permutes the rows and columns of the matrix and groups even blocks (yellow) and odd blocks (green) together (b). After the block Gaussian elimination shown in (c), the permuted system is divided into two subsystems with $\mathbf{SBT}(B)$ on the left-hand-side (d).

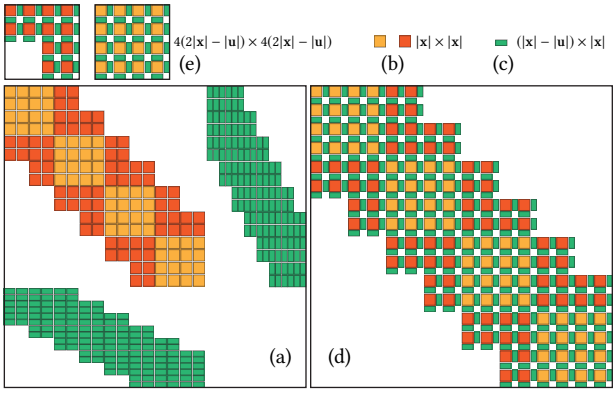

Fig. 3. We illustrate the sparsity pattern of Equation 12 when $Z = 8$ and $|\mathcal{T}_\mathbf{c}| = 16|\mathbf{x}|$. The left-hand-side matrix has a sparsity pattern shown in (a). Here $\mathbf{H} \in \mathbf{SBT}(4|\mathbf{x}|)$ with diagonal $|\mathbf{x}| \times |\mathbf{x}|$ block shown in yellow and off-diagonal $|\mathbf{x}| \times |\mathbf{x}|$ block is shown in red (b). $\mathbf{A} \in \mathbf{BLT}(6)$ with $(|\mathbf{x}| - |\mathbf{u}|) \times |\mathbf{x}|$ block is shown in green (c). After a permutation that interleaves blocks of $\mathbf{H}$ and blocks of $\mathbf{A}$, we get the sparsity pattern shown in (d), which is $\mathbf{SBT}(4(2|\mathbf{x}| - |\mathbf{u}|))$ with diagonal and off-diagonal blocks shown in (e).

element of $\mathbf{J}(\mathcal{T}_\mathbf{x})$ and each row of $\frac{\partial \mathbf{J}(\mathcal{T}_\mathbf{x})}{\partial \mathcal{T}_\mathbf{x}}$ in parallel. Then we compute $\mathbf{E}(\mathcal{T}_\mathbf{x})$ from a dot-product.

- $\mathbf{J}^T\mathbf{J}$-approximate Hessian of $\mathbf{E}(\mathcal{T}_\mathbf{x})$ without spline interpolation is denoted by $\mathbf{H}(\mathcal{T}_\mathbf{x}) = \frac{\partial \mathbf{J}(\mathcal{T}_\mathbf{x})}{\partial \mathcal{T}_\mathbf{x}}^T \frac{\partial \mathbf{J}(\mathcal{T}_\mathbf{x})}{\partial \mathcal{T}_\mathbf{x}}$ and we have $\mathbf{H}(\mathcal{T}_\mathbf{x}) \in \mathbf{SBT}(2|\mathbf{x}|)$. We compute each $|\mathbf{x}| \times |\mathbf{x}|$ block of $\mathbf{H}(\mathcal{T}_\mathbf{x})$ in parallel.
- Gradient of $\mathbf{E}(\mathcal{T}_\mathbf{x})$ without spline interpolation is denoted by $\mathbf{b}(\mathcal{T}_\mathbf{x}) = \frac{\partial \mathbf{J}(\mathcal{T}_\mathbf{x})}{\partial \mathcal{T}_\mathbf{x}}^T \mathbf{J}(\mathcal{T}_\mathbf{x})$ and we compute each element of $\mathbf{b}(\mathcal{T}_\mathbf{x})$ in parallel.
- $\mathbf{H} = (\mathbf{S} \otimes \mathbf{I}^{|\mathbf{x}|})^T \mathbf{H}(\mathcal{T}_\mathbf{x})(\mathbf{S} \otimes \mathbf{I}^{|\mathbf{x}|})$ is the $\mathbf{J}^T\mathbf{J}$-approximate Hessian of $\mathbf{E}(\mathcal{T}_\mathbf{c})$ with spline interpolation used by Algorithm 1 and $\mathbf{H} \in \mathbf{SBT}(4|\mathbf{x}|)$. We compute each $|\mathbf{x}| \times |\mathbf{x}|$ block of $\mathbf{H}$ in parallel.
- $\mathbf{b} = (\mathbf{S} \otimes \mathbf{I}^{|\mathbf{x}|})^T \mathbf{b}(\mathcal{T}_\mathbf{x})$ is the gradient of $\mathbf{E}(\mathcal{T}_\mathbf{c})$ with spline interpolation used by Algorithm 1 and we compute each element of $\mathbf{b}$ in parallel.
- $\mathbf{C}$ is the constraint with spline interpolation used by Algorithm 1. We compute each element of $\mathbf{C}$ in parallel.
- Jacobian of $\mathbf{C}(\mathcal{T}_\mathbf{x})$ without spline interpolation is denoted by $\mathbf{A}(\mathcal{T}_\mathbf{x})$ and $\mathbf{A}(\mathcal{T}_\mathbf{x}) \in \mathbf{BLT}(3)$. We compute each $(|\mathbf{x}| - |\mathbf{u}|) \times |\mathbf{x}|$ block of $\mathbf{A}(\mathcal{T}_\mathbf{x})$ in parallel.
- $\mathbf{A} = (\mathbf{S}^T \otimes \mathbf{U}_/)\mathbf{A}(\mathcal{T}_\mathbf{x})(\mathbf{S} \otimes \mathbf{I}^{|\mathbf{x}|})$ is the Jacobian of $\mathbf{C}(\mathcal{T}_\mathbf{c})$ with spline interpolation used by Algorithm 1 and $\mathbf{A} \in \mathbf{BLT}(8)$. We compute each $(|\mathbf{x}| - |\mathbf{u}|) \times |\mathbf{x}|$ block of $\mathbf{A}(\mathcal{T}_\mathbf{x})$ in parallel.
- $\mathbf{A}\mathbf{A}^T \in \mathbf{SBT}(6(|\mathbf{x}| - |\mathbf{u}|))$ and we compute each $(|\mathbf{x}| - |\mathbf{u}|) \times (|\mathbf{x}| - |\mathbf{u}|)$ block of $\mathbf{A}\mathbf{A}^T$ in parallel.

D LEARNING LOCOMOTIONS FROM PHYSICS-BASED LOSS

In Section 7, we jointly optimize RNN and $\mathcal{T}_\mathbf{x}$. However, if the main application is to train the recurrent neural network, it is also possible to eliminate $\mathcal{T}_\mathbf{x}$ and directly optimize $w_{\mathbf{RNN}}$. To do this, we note that a RNN defines a complete locomotion trajectory by unrolling: $\mathbf{c}_{i+1} = \mathbf{RNN}^i(\mathbf{c}_1, w_{\mathbf{RNN}})$. However, it is not generally possible for the RNN to satisfy the hard constraints in Equation 8, so we transform them to soft constraints, giving the following form:

$$\underset{w_{\mathbf{RNN}}}{\mathbf{argmin}}\ \mathbf{E}(\mathcal{T}_\mathbf{c}) + \|\mathbf{C}(\mathcal{T}_\mathbf{c})\|^2 \quad \mathbf{s.t.}\ \mathbf{c}_{i+1} = \mathbf{RNN}^i(\mathbf{c}_1, w_{\mathbf{RNN}}), \qquad (1)$$

which directly optimizes the RNN from a physics-based loss function. Here we fix the initial frame $\mathbf{c}_1$ and have the $\mathbf{RNN}$ derive $\mathbf{c}_2, \cdots, \mathbf{c}_Z$. Note that this application is not possible using previous trajectory optimization formulations because they depend on additional variables such as Lagrangian multipliers to determine contact forces. We use the GPU-based LBFGS algorithm [Nocedal and Wright 2006, Chapter 9.2] to solve this optimization with function gradient computed using back-propagation. As illustrated in Figure 4, we can find the novel walking gaits for an 18-DOF 4-legged spider after a few minutes of computation.

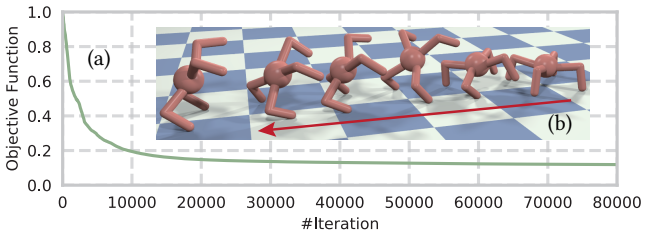

Fig. 4. We use Equation 1 to search for RNN that can represent the trajectory of a 4-legged spider walking. We show the convergence history of the LBFGS algorithm (a) and the optimized trajectory (b), where the spider is performing a stunt and uses only its two front legs to walk forward. The trajectory meets convergence criterion after 30000 iterations and 11(min) of computation.

REFERENCES

Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. 2009. Fast BVH construction on GPUs. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 375–384.

Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization, second edition*. World Scientific.

Zherong Pan and Dinesh Manocha. 2018. Position-Based Time-Integrator for Frictional Articulated Body Dynamics. In *Algorithmic Foundations of Robotics XIV*. Springer.

Y. Yang, Y. Wu, and J. Pan. 2017. Parallel Dynamics Computation Using Prefix Sum Operations. *IEEE Robotics and Automation Letters* 2, 3 (July 2017), 1296–1303.