

GPU-accelerated Interpolation for 3D Image Registration

Naveen Himthani
Institute for Computational
Engineering and Sciences
UT Austin

Amir Gholami
Department of Electrical
Engineering and Computer
Sciences
UC Berkeley

Andreas Mang
Department of Mathematics
University of Houston

George Biros
Institute for Computational
Engineering and Sciences
UT Austin

1. Introduction

Image registration is a key technology in image computing with numerous applications in medical imaging [7]. It is about establishing a *meaningful* spatial correspondence $\mathbf{y}$ between two images $\rho_R(\mathbf{x})$ (reference image) and $\rho_T(\mathbf{x})$ (template image) of the same object, such that the transformed template $\rho_T(\mathbf{y}(\mathbf{x}))$ is similar to $\rho_R(\mathbf{x})$. We consider $\mathbf{y}$ to be meaningful if it is a *diffeomorphism* (i.e., one-to-one, onto, and has a smooth inverse). We use an optimal control formulation, in which we parameterize $\mathbf{y}(\mathbf{x})$ with a smooth, stationary velocity field $\mathbf{v}(\mathbf{x})$ [2, 3]. Finding the optimal $\mathbf{v}$ requires solving a highly non-linear inverse problem that involves integro-differential operators, biharmonic operators, and pure advection operators both forward and backward in time. A major issue is the time to solution, which poses the demand for effective numerical schemes and the utilization of high performance computing resources.

We have previously presented a distributed memory algorithm for image registration on CPUs [3, 1]. In this poster, we extend [3, 1] and present a GPU-accelerated implementation of the computational kernels involved in registration. Our main *contribution* is a GPU accelerated implementation of the interpolation computational kernel.

2. Mathematical Formulation

We use an optimal control formulation. Given two scalar valued functions $\rho_T(\mathbf{x})$ and $\rho_R(\mathbf{x})$ (the template and reference image) we seek a stationary velocity field $\mathbf{v}(\mathbf{x})$ that minimizes (This is the simplest form; see [3, 1] and references for more complex formulations.)

$$\frac{1}{2} \int_{\Omega} (\rho_1(\mathbf{x}) - \rho_R(\mathbf{x}))^2 d\mathbf{x} + \frac{\beta}{2} \int_{\Omega} \Delta \mathbf{v} \cdot \Delta \mathbf{v} d\mathbf{x} \quad (1)$$

subject to the forward problem (i.e., state equation)

$$\partial_t \rho + \mathbf{v} \cdot \nabla \rho = 0 \quad \text{in } \Omega \times (0, 1], \quad \rho = \rho_T \quad \text{in } \Omega \times \{0\}. \quad (2)$$

The state variable $\rho(\mathbf{x}, t)$ in (2) represents the transported intensities of $\rho_T(\mathbf{x})$ subjected to $\mathbf{v}$. The final state $\rho_1(\mathbf{x}) := \rho(\mathbf{x}, 1)$ corresponds to $\rho_T(\mathbf{y}(\mathbf{x}))$. We measure the discrepancy between $\rho_1(\mathbf{x})$ and $\rho_R(\mathbf{x})$ based on a squared L^2 -distance (see (1)). The second term in (1) is a regularization model that is introduced to ensure that $\mathbf{v}$ is sufficiently smooth for a minimizer to exist. The forward problem has to be solved multiple number of times per image registration problem. This is costly. We present a fast and simple interpolation kernel to improve performance.

3. Computational Kernels

The main computational kernels for CLAIRE are the FFT used for spatial differential operators in our spectral approach and the interpolation in the semi-Lagrangian scheme for advection.

Semi-Lagrangian Time Stepping

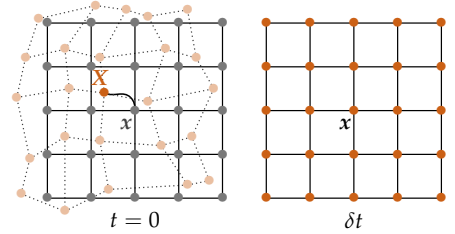


Figure 1: Illustration of the semi-Lagrangian scheme in 2-dimensions. Point x is a regular grid point and X is the material point at $t = 0$ that landed at x at time t . The semi-Lagrangian algorithm requires interpolation of v and m at these off-grid points

Interpolation

As explained in section 2, the transport equation in registration is solved using semi-Lagrangian scheme that requires costly interpolation of velocities and image data along backward characteristics as explained in [3]. We use a lagrange polynomial as a basis function for third-order interpolation which requires a different set of 64 nearest neighbours for each off-grid query point.

4. GPU accelerated semi-Lagrangian Solver

4.1 Approach

We present a fast semi-Lagrangian solver for the forward problem. The solver utilizes the hardware texture filtering feature of modern GPUs to interpolate a 3D scalar using a third-order lagrange basis polynomial. In [5, 4, 6, 8], splines are utilized for interpolation and requires prefiltering of spline coefficients for higher accuracy. Prefiltering is an expensive operation. Lagrange polynomials do not need any prefiltering and are much faster for given accuracy requirements. The novelty of our algorithm is that we are able to reduce 64 nearest neighbour fetches required for 3D

¹pf = prefiltering operation

²without prefiltering operation

Table 1: Computational performance of different gpu interpolation kernels in single precision for semi-lagrangian solver. We report relative errors and total execution timings for interpolation kernels for a benchmark forward problem (2) using a synthetic image ρ_s . The L_2 relative error is computed between ρ_s and the image obtained by solving (2) first using velocity $\mathbf{v}_s$ and then $-\mathbf{v}_s$. A "texture" prefix implies linear texture filtering is used. A "simple" prefix implies texture is used as a lookup instead. Runtimes are reported in milliseconds.

resolution	64 ³		256 ³	
kernel	rel. error	runtime	rel. error	runtime
haswell	2.16E-4	5.64	2.06E-4	344
skylake	2.16E-4	6	2.06E-4	321
ivyBridge	2.16E-4	5.18	2.06E-4	354
pf simple spline ¹	2.93E-4	5.77	2.05E-4	421
pf texture spline ¹	3.68E-4	4.27	2.08E-4	331
simple spline ²	2.17E-2	1.85	1.41E-3	115
simple lagrange	2.16E-4	1.79	2.06E-4	116
texture lagrange	3.88E-4	0.86	2.16E-4	56
texture spline ²	2.17E-2	0.36	1.41E-3	23

cubic interpolation to 1 trilinear, 6 bilinear, 12 linear texture interpolations and 8 point texture as shown in fig 2 table lookups which resulting in a significant speedup and at a slight cost of accuracy resulting from the low-precision storage of the interpolation weights of linear texture filtering.

Algorithm 1 Texture lagrange cubic interpolation

Input: query coordinates $\mathbf{x}_q$, data texture $\mathbf{tex}$ of function $f(\mathbf{x})$

Output: interpolated value $f(\mathbf{x}_q)$

- 1: $\mathbf{w} \leftarrow \text{lagrange_weights}(\mathbf{x}_q)$
- 2: $[\mathbf{g}, \mathbf{h}, \hat{\mathbf{x}}] \leftarrow \text{compute_texture_filter_coord}(\mathbf{w}, [\mathbf{x}_q])$
- 3: $\text{trilinear}[1] \leftarrow \text{tex3D}(\mathbf{tex}, \mathbf{h}, \hat{\mathbf{x}})$
- 4: $\text{bilinear}[6] \leftarrow \text{tex3D}(\mathbf{tex}, \mathbf{h}, \hat{\mathbf{x}})$
- 5: $\text{linear}[12] \leftarrow \text{tex3D}(\mathbf{tex}, \mathbf{h}, \hat{\mathbf{x}})$
- 6: $\text{point}[8] \leftarrow \text{tex3D}(\mathbf{tex}, \mathbf{h}, \hat{\mathbf{x}})$
- 7: $\mathbf{a} \leftarrow \text{linear_weighting_X}(\text{trilinear}, \text{bilinear}, \text{linear}, \text{point}, \mathbf{w}, \mathbf{g})$
- 8: $\mathbf{b} \leftarrow \text{linear_weighting_Y}(\mathbf{a}, \mathbf{w}, \mathbf{g})$
- 9: $f(\mathbf{x}_q) \leftarrow \text{linear_weighting_Z}(\mathbf{b}, \mathbf{w}, \mathbf{g})$

4.2 Results

We compare the speed and accuracy of our interpolation kernel with similar existing kernels [5, 4, 6, 8] for synthetic images of resolutions 64³ and 256³ in table 1. We perform the GPU benchmarks on the "Piz Daint" system at the Swiss National Supercomputing Center. Each XC50 compute node of "Piz Daint" has a NVIDIA Tesla P100 GPU. We also run these cases on the "Maverick" cluster at Texas Advanced Computing Center (TACC). Each GPU node of "Maverick" has a single NVIDIA Tesla K40 GPU. We report the runs on "Maverick" in the poster. We use a one-dimensional thread-block configuration and launch 256 threads per block for the compute kernel. We also compare the performance of our algorithm with a CPU based implementation [1]. The CPU benchmark is run on a single Haswell, Skylake and IvyBridge node with 20 MPI tasks each. After testing various configurations, 256 threads per block gives optimal performance. We launch a one-dimensional grid-block where each thread performs interpolation for a single query coordinate.

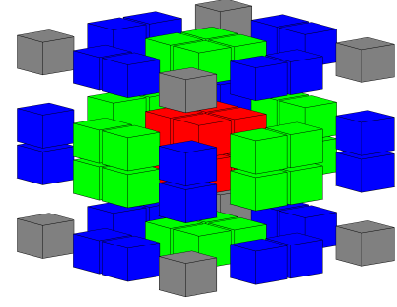


Figure 2: Illustration of lagrange texture interpolation using 64 nearest neighbours. The **center core** consisting of 8 data points is used for a single trilinear texture lookup. The **6 faces** each consisting of 4 points are used for bilinear lookups. The **12 edges** each consisting of 2 points are used for linear lookups and the remaining **8 points** are fetched using texture point lookup instead of accessing them from global memory. The resulting interpolated values are linearly weighted along x,y and z directions in that order.

4.2.1 Roofline Analysis

- We estimate the algorithmic Arithmetic Intensity (AI) of our kernel by computing the in-app Floating Point Operations (FLOPs) per byte of memory accessed perform a basic roofline model analysis with 'no cache' model.
- For a comparison, we also use the NVIDIA Visual Profiler to get the actual number of bytes moved from device memory to get a more accurate estimate of the Arithmetic Intensity. The analysis suggests that our kernel is memory bound. See figure 3

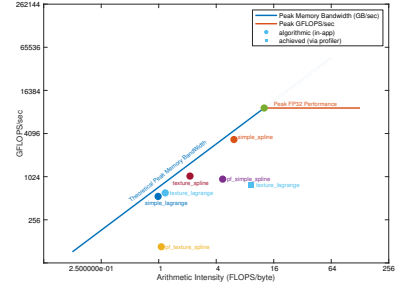


Figure 3: Roofline Analysis of different interpolation kernels to identify performance limiters.

Current and Future Work

We plan to combine texture memory accesses along with use of shared memory for good reuse of loaded data to further improve kernel performance. A simple experiment where we put all query coordinates close together such that they all use the same 64 grid values, the usage of shared memory provides high performance in such cases. We plan to extend this to more general scenarios.

5. References

- [1] A. Gholami, A. Mang, K. Scheufele, C. Davatzikos, M. Mehl, and G. Biros. A framework for scalable biophysics-based image analysis. In *Proc ACM/IEEE Conference on Supercomputing*, pages 19:1–19:13, 2017.

- [2] A. Mang and G. Biros. An inexact Newton–Krylov algorithm for constrained diffeomorphic image registration. *SIAM Journal on Imaging Sciences*, 8(2):1030–1069, 2015.
- [3] A. Mang, A. Gholami, and G. Biros. Distributed-memory large-deformation diffeomorphic 3D image registration. In *Proc ACM/IEEE Conference on Supercomputing*, number 72, 2016.
- [4] D. Ruijters, B. M. ter Haar Romeny, and P. Suetens. Efficient gpu-based texture interpolation using uniform b-splines. *Journal of Graphics Tools*, 13(4):61–69, 2008.
- [5] D. Ruijters and P. Thévenaz. Gpu prefilter for accurate cubic b-spline interpolation. *Comput. J.*, 55(1):15–20, Jan 2012.
<http://dx.doi.org/10.1093/comjnl/bxq086>.
- [6] C. Sigg and M. Hadwiger. Fast third-order texture filtering. pages 313–329, February 2005.
- [7] A. Sotiras, C. Davatzikos, and N. Paragios. Deformable medical image registration: A survey. *Medical Imaging, IEEE Transactions on*, 32(7):1153–1190, 2013.
- [8] P. Thévenaz, T. Blu, and M. Unser. Interpolation revisited [medical images application]. *Medical Imaging, IEEE Transactions on*, 19(7):739–758, 2000.