CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Multi-level, Parallel Algorithms for Shape Interpolation on Modern Architectures
Student:	Šimon Hrabec
Supervisor:	Patrick Sanan, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2017/18

Instructions

Interpolation between shapes is a common task in geometry processing, fundamentally involving non-trivial geometric considerations. Aiming towards real-time, robust applications, we investigate how state-of-the-art methods may be accelerated and made robust with the introduction of multi-level methods and modern parallel and hybrid computer architectures.
1. Get familiar with the sub-field of shape interpolation within the field of geometry processing.
2. In C/C++, implement a variant of a shape-interpolation method from the literature and assess its performance.
3. Produce a prototype interactive application relying on this implementation.
4. Implement a multi-level extension of the method and assess its performance in terms of
   a) scalability of time to solution for a family of relevant interpolation problems,
   b) robustness to mesh complexity.
5. Implement one of the computational kernels for execution on a GPU.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague October 20, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Bachelor's thesis

Multi-level, Parallel Algorithms for Shape Interpolation on Modern Architectures

Šimon Hrabec

Supervisor: Dr. Patrick Sanan

9th January 2018

Acknowledgements

I would like to thank my supervisor Patrick Sanan for his support, guidance and the time he has devoted to me. I also want to give gratitude to Tomáš Kalvoda and his altruistic effort to explain the concepts related to my work and provide useful examples.

Declaration

In Prague on 9th January 2018 ....

Citation of this thesis

Hrabec, Šimon. *Multi-level, Parallel Algorithms for Shape Interpolation on Modern Architectures.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

Tato práce zkoumá téma interpolace tvarů. Zahrnuje úvod do souvisejících témat - numerických metod, paralelních výpočtů & CUDA. Práce obsahuje implementaci jednoho z algoritmtů pro interpolaci tvarů s víceúrovňovým rozšířením včetně implementace v CUDA.

Klíčová slova Interpolace tvarů, paralelní výpočty, CUDA, Grafika, Výpočetní geometrie, Numerické metody, Sít̆

Abstract

This work examines the problem of shape interpolation. It contains an overview of related topics - numerical methods, parallel computing & CUDA. It provides an implementation of an shape interpolation algorithm with multi-level extension including an implementation in CUDA.

Keywords Shape Interpolation, Parallel computing, CUDA, Graphics, Computational Geometry, Numerical Methods, Mesh

Contents

List of Figures

Chapter 1

Introduction

With the rapid progress during last few decades in the area of computer hardware, devices with high computational power have become widespread and hand in hand with that goes an increased demand for visually appealing graphics, including 3D graphics. In such area, objects are commonly represented as a set of polygons, formed by a connected network of points, forming a mesh. Despite the exponential (as it has been predicted by Moore and his law [1]) growth of performance efficient algorithms are still required. Shape interpolation is a problem where a sequence of shapes - represented as meshes - is given and intermediate ones are desired to obtain.

1.1 Motivation

For a long time there has been an exponential growth of performance where frequency increase has played a major role. However, with the exponential growth of frequency the power consumption grew in similar manner. The unsuccessful NetBurst architecture of INTEL showed us that forever going frequency increase is not an option and during the last decade we could observe a stall in the frequencies of CPUs [2]. This phenomenon has led to a paradigm shift in the area of computing. The limitation set by the infeasible frequency cap has been overcome by supplying multiple computational cores, allowing multiple things to be computed at the same time. Intel reported that lowering a frequency of a core by 20 percent saved about 50 percent of the energy. The aspect of power consumption is empathised in HPC, where the power required to run such computer can be over years higher than the computer itself [3] and also in the area of mobile computing, where power supply is limited.

But having a different paradigm at the side of hardware requires a similar change in the software that is run on such architecture. In last decade GP-GPU computing has become a popular option - using GPU as a highly parallel SIMD device [4]. CPUs are designed in a way to have low latency and for execution of various code due to which a significant area of the CPU is designated for control flow (branch prediction) and caching. GPUs used for GPGPU computing do not have sophisticated caching and branch prediction, but rather high number of computational units, due to which they provide high throughput with higher delay, making them optimal for homogeneous computation - where one instruction is applied on multiple data (also called single instruction multiple data - SIMD). The different architecture requires problems of interests to be formulated in different, parallel manner. Parallel and scalable algorithms are of interest.

1.2 Application - Use cases

Application domain of shape interpolation includes various areas. Starting with cinematography, where designer animates a character. He or she then can model just several poses in a second and then rely on a shape interpolation to model the rest. In such scenario a robustness of the algorithm might be the primary concern over a computational speed or power to performance ration. On the other side are mobile devices, which are restricted in term of power supply and for which power efficiency is of a great importance. A virtual reality game can receive information about position of other players and their body parts and interpolate the received information for smoother effect. A more general mesh interpolation for meshes with different connectivity can be used in graphics for interesting animation of object transformation.

Chapter 2

Analysis

2.1 Problem description

Shape interpolation is a problem where given two meshes (or possibly more) we want to obtain a series of intermediate meshes that best represent the movement or transformation from one to another. In fact the shape interpolation problem consists of 2 parts - vertex correspondence problem, where we need mark which vertices correspond to each other between the two given meshes (we can also interpolate between two same meshes), which can be done in a user assisted manner or automatically. Second problem is vertex path problem - computing the location of the vertices of the intermediate shapes.

For shapes that are under the effect of translation, scale or shear a simple linear interpolation can be used, however such approach might lead to a shape distortion for rotation due to its non-Euclidian nature. As shown on figure 2.1 a linear interpolation for rotating object will result in shrinkage of the object, which is not desired [1]. For shape interpolation algorithms we are interested how well the algorithm interpolates - how an object is distorted, how a human would perceive such shapes as natural - but also the computational speed. In the movie industry a high quality solution is desired, but in the case of mobile computing a real-time interpolation is desired despite its worse quality. There is a trade-off between the quality and processing time or number of shapes interpolated per second.

[1]Mesh on the picture has been created by Martin Kilian and is available on `http://graphics.stanford.edu/~niloy/research/shape_space/shape_space_sig_07.html`

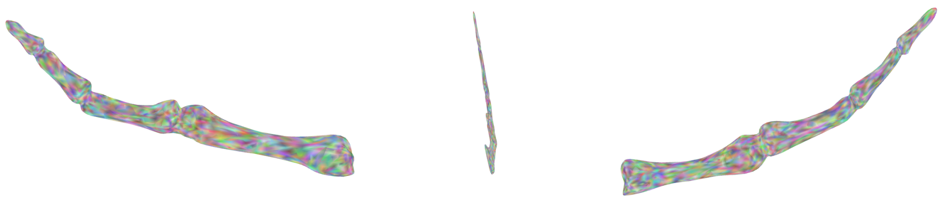

Figure 2.1: Mesh deformation caused by linear interpolation on rotating object

2.2 Related and previous work

The shape interpolation problem is quite flexible in term of possible way how to approach it and get the desired outcome. Marras et al.[5] do not treat the shape as a single object, but decompose the shape into rigid parts that do not require special treatement and are interpolate lineary and joints, that are then an object of non-linear, edge-based interpolation in order to speed up the interpolation computation. Alexa et al.[6] percieve the mesh interpolation of the volume, the interior of the object rather than its boundary and try to identify local affine transformations. Robert W. Sumner and Jovan Popović[7] interpolate a shape by a deformation transfer from another, semantically correspondent shape. T Winkler et al.[8] present an approach that lineary interpolates the intrinsic properties of the mesh (edge lengths and dihedral angles), but also consider the interpolation between more than 2 shapes and extrapolation. Martin Kilian et al. [9] introduce a Riemannian geometry based framework that works in the space of isometric deformations. Stefan Fröhlich and Mario Botsch[10] focus on combining the physics-based and example-driven techniques.

2.3 Energy function

For the purpose of shape interpolation it is useful to have a metric of how much the interpolated shape has been distorted. Such function can be called an energy function and will be denoted E.

$$E : P \to R$$

It takes the whole serie of interpolated meshes and produces a single number - energy, an indicator of the distortion. For the given metric we are trying to find a series of shapes that minimises the energy.

2.4 GPGPU Computing

The evolution of computer hardware does not bring only increased frequencies and higher density of transistors, but also architectural shifts and new concepts. In the past all the computation in the computer was done by CPU and possibly its specialised coprocessors. With the shift from text interface to graphical interfaces more performance was needed for operations related to graphics and specialised unit dedicated to graphics has emerged. Both CPU and GPU serve different purposes and therefore differ significantly in their architectures and design.

The CPU is a general-purpose processing unit, that executes arbitrary code. It is designed for execution of serial code and it tries to minimize or hide delays that an execution of code could cause - as this would efficiently make the process to stall and finish later. Due to that modern CPUs have hierarchy of cache memories, branch prediction, speculative execution, out-of-order execution. The ratio of cache memory and circuits to ALUs is much higher than for GPU.

Contrary to CPU, GPU is designed to process 3D graphics, which includes a lot of computation that can be done in parallel. An example might be a rotation of an object, which can be expressed as vector-matrix multiplications. Due to the nature of such computations GPUs are equipped with a lot of small and simple processing units. They are mainly focused on throughput. The difference can be understood when inspecting the number of cores. Intel Core i7-8700K has 6 cores, where NVIDIA GTX 1080 has 2560 cores. These cores are not comparable, as they differ significantly in complexity and capabilities.

Nowadays it is possible to utilise GPU not only to render graphics, but also to run user code. NVIDIA has it proprietary platform CUDA, another widespread possibility is OpenCL, which is contrary to NVIDIA's CUDA available on other platforms too. In the past it was necessary to model calculation to OpenGL or DirectX calls as explained in [11]. The reason why we do not use GPUs with huge performance (88731 GFLOPS for NVIDIA GTX 1080 [2]) is that in order to exploit its theoretical peak performance we need to express our problem in parallel manner, which is completely different concept than writing code in the traditional, sequential fashion. Moreover, not all problems can be computed in parallel as their nature is sequential. NVIDIA CUDA, which is used in this work, is an extension to C++. It takes special functions (kernels) marked with special keyword *__global__*, that is then run by every instantiated core. This concept is called SIMD - Single Instruction, Multiple

[2]NVIDIA specifications - `https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf`

Data. Such concept is also applied to CPUs - instructions sets such as SEE (Intel) or 3DNow! (AMD) can perform an operation on multiple registers, but at much smaller scale.

```
__global__
void vecAddKernel(double *inputA, double *inputb,
        double *output, int count){
    int idx = blockIdx.x*blockDim.x + threadIdx.x;

    if (idx < count)
        c[idx] = a[idx] + b[idx];
}

void vecAddGPU(double *h_inputA, double *h_inputb,
        double *h_output, int count){
    double *d_inputA;
    double *d_inputB;
    double *d_output;

    size_t arrayByteSize = count*sizeof(double);

    //Allocate memory on the device (GPU)
    cudaMalloc(&d_inputA, arraySize);
    cudaMalloc(&d_inputB, arraySize);
    cudaMalloc(&d_output, arraySize);

    //Copy input to the device (GPU)
    cudaMemcpy(d_inputA, h_inputA, arrayByteSize,
        cudaMemcpyHostToDevice);
    cudaMemcpy(d_inputB, h_inputB, arrayByteSize,
        cudaMemcpyHostToDevice);

    int blockSize = 1024;
    int gridSize = gridSize = (count-1)/blockSize+1;

    //Run the kernel
    vecAddKernel<<<gridSize, blockSize>>>(d_inputA,
        d_inputB, d_output, count);

    //Copy result back to host
    cudaMemcpy(h_output, d_output, arrayByteSize,
        cudaMemcpyDeviceToHost);

    //Free alocated memory on the device (GPU)
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
}
```

Listing 2.1: Vector addition CUDA example

The code snippet listing 2.1 shows a simple example of vector (1D arrays) addition executed on GPU. Each thread will execute the same function (*vecAddKernel*). What differ is the data that is accessed based on the index that is calculated from the thread and block ID. By using these each thread can access different data. Contrary to conventional paralellism on CPUs, where we can run multiple threads with different code, here we are restricted to the same code (function), at least for one kernel call. We can surely overcome this by an if statement, however this is not the proper usage of the CUDA paradigm. Threads are grouped to warps (details are omited) and when an if statement result in different paths to be taken by threads in the same group (warp), a code divergence will occur. This effectivelly makes the two divergent sets of threads be executed sequentialy rather than at the same time. Therefore, thread divergence shall be avoided wherever possible. NVIDIA, as an author with a commercial interest in the adoption of the technology, provides documentation and guides, where details about the CUDA architecture, including memory hierarchy, blocks, warps, memory synchronisation and compute capability are explained[12].

Another approach that can be taken is an adoption of higher-level libraries such as Thrust that allows a develop to abstract from the details of GPU programming and use STL-like looking parallel datastructures and algorithms (including sorting and reduction) without writing any kernel function.[3]

2.5 Implemented method

In this work I take a variation of method Kilian et al.[9] proposed. They define the metric for a mesh in the following way:

$$E(P) := \sum_{i=0}^{n} (\langle\langle X_i, X_i \rangle\rangle_{P_i} + \langle\langle X_i, X_i \rangle\rangle_{P_{i+1}}) \tag{2.1}$$

Given the following definitions of the riemannian metric,

$$\langle\langle X, Y \rangle\rangle_{M,\lambda} := \langle\langle X, Y \rangle\rangle_M + \lambda \langle\langle X, Y \rangle\rangle_M^{L^2} \tag{2.2}$$

the regularisation term,

$$\langle\langle X, Y \rangle\rangle_M^{L^2} := \sum_{p \in M} \langle X_P, Y_P \rangle A_P \tag{2.3}$$

[3]NVIDIA's developer website about the Thrust library - `https://developer.nvidia.com/thrust`

the semi Riemannian part of the metric (as isometric as possible),

$$\langle\langle X, Y\rangle\rangle_M^I := \sum_{(p,q)\in M} \langle X_p - X_q, p-q\rangle\langle Y_p - Y_q, p-q\rangle \tag{2.4}$$

we got an equation:

$$\begin{aligned} E(P) := \sum_{i=0}^{n}(\sum_{(p,q)\in P_i} \langle X_p - X_q, p-q\rangle^2 + \lambda \sum_{p\in P_i} \langle X_P, X_P\rangle A_P + \\ \sum_{(p,q)\in P_{i+1}} \langle X_p - X_q, p-q\rangle^2 + \lambda \sum_{p\in P_{i+1}} \langle X_p, X_p\rangle A_P) \end{aligned} \tag{2.5}$$

The purpose of creating a metric is not only to define how the meshes are distorted in the interpolation problem, but also to be able to minimise it and make the resulting interpolated shapes as least deformed as possible (and the definition of distorted is given by the metric).

For the minimisation (optimisation) we will use an iterative method (improving the solution with every step), which will operate with the derivative of the metric.

The advantage of this metric is that it consists of a lot of summations, for which can use the derivative sum rule - that a derivative of sum is sum of derivatives. Meshes P_0 and P_n are the input meshes and their positions are fixed - we are looking only for the positions for the poses in between. Therefore for the gradient calculation we can drop the gradient assigned to these 2 meshes.

After expanding all sums in the equations, it is possible to determine that a gradient of a mesh is determined only by itself and its neighbouring meshes. To determine the derivative of mesh P_i we can drop all terms that do not contain vertices from the mesh (as their derivative is zero). We get:

$$\frac{E(x)}{\partial x_i} = (\langle\langle X_{i-1}, X_{i-1}\rangle\rangle_{P_i} + \langle\langle X_i, X_i\rangle\rangle_{P_i} + \langle\langle X_i, X_i\rangle\rangle_{P_{i+1}} + \langle\langle X_{i+1}, X_{i+1}\rangle\rangle_{P_{i+1}})' \tag{2.6}$$

We can expand further to get:

$$\begin{aligned} \frac{E(x)}{\partial x_i} = & \sum_{(v,q)\in P_{i-1}} \langle X_p - X_q, p-q\rangle^2 + \lambda \sum_{p\in P_{i-1}} \langle X_P, X_P\rangle A_P + \\ & \sum_{(p,q)\in P_i} \langle X_p - X_q, p-q\rangle^2 + \lambda \sum_{p\in P_i} \langle X_P, X_P\rangle A_P + \\ & \sum_{(p,q)\in P_i} \langle X_p - X_q, p-q\rangle^2 + \lambda \sum_{p\in P_i} \langle X_P, X_P\rangle A_P + \\ & \sum_{(p,q)\in P_{i+1}} \langle X_p - X_q, p-q\rangle^2 + \lambda \sum_{p\in P_{i+1}} \langle X_p, X_p\rangle A_P \end{aligned} \tag{2.7}$$

2.5.1 Minimisation method

We now have defined a metric - a function that that takes a series of meshes and gives a real number that denotes a level of deformation. We also have a way to compute the gradient (partial derivative for every vertex). Now wan use these in an iterative method that, in steps, modify the mesh in a way that its energy decreases. A simple iterative optimisation method is gradient descent. It is simple and requires only the function and its gradient, but the cost is slow rate of convergence. It works with an idea to take a direction in which a function decreases the most and make a step in this way:

$$X_{n+1} = X_n - \gamma \nabla f(x)$$

In this equation γ represents a step size. It can be set in advance (it is guaranteed that there exist enoug small γ that the method converges) or we can adjust it as we iterate - lower the step until $X_n < X_n - gamma \nabla f(x)$.

More sophisticated method is Netwon's method. It works with Hessian matrix, which is a matrix of second-order partial derivatives. It has faster rate of convergence, but it requires more and it need more computation to apply the Hessian matrix. For such purpose quasi-Newton's method might come in use. Such method do not use Hessian matrix directly but only approximate it in some way. An example of a quasi-Newton's method is BFGS [13].

2.6 Multilevel approach

Multilevel approach in the problem of shape interpolation is utilised with the intention of speeding the computation up. It is built on the principle that vertex positions are not random or independent, instead being quite the contrary - highly dependent on each other, especially on the adjascent vertices. We can exploit such relation and create a simplified mesh that will omit some vertices or merge some vertices together. Such mash will be faster to operate with and later we can add the vertices back, estimating their positions relatively to their neighbours.

Let's demonstrate this approach on example. The figure 2.2 shows a pose of a simple 2D mesh, which has 9 vertices (A-I) and 8 faces. Second image shows a simplified version of the same mesh. It consists of only 4 vertices and 2 faces. We can then modify the mesh. The third pictures shows the mesh moved, rotated, scaled and slightly deformed with respect to the first pose. Fourth picture shows interpolation back to the finer grid and estimates a location of the vertices that have been added back. We can expect E to be between A,C,G,I, or D to be between A, G, E and similarly for other vertices.

In fact, we can we can take this further and not make a one simpler mesh, but rather make a hierarchy of meshes from the original fine mesh to the most

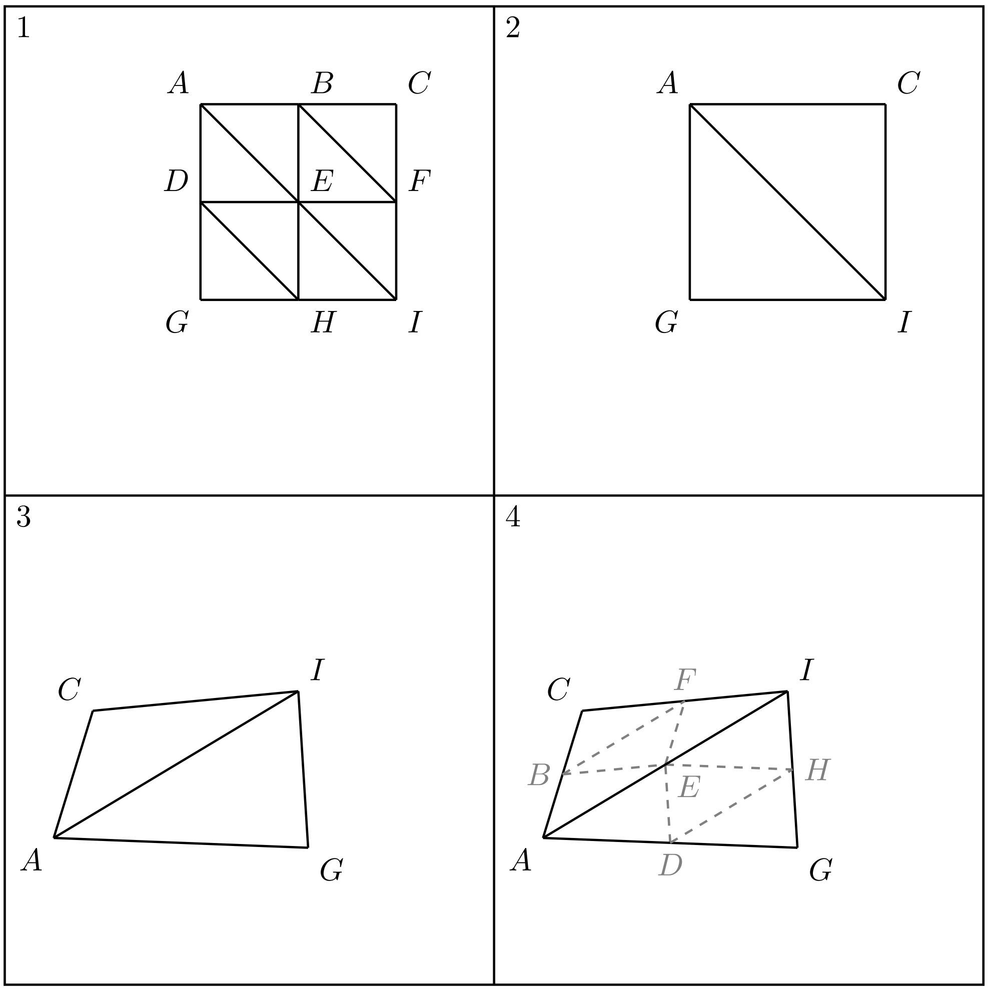

Figure 2.2: Demonstration of restriction and interpolation on a simple 2D mesh

simplified coarse one. We then start to work with the coarsest mesh and use the result as a starting point for the interpolated, finer level. Such interpolated mesh will then require less iterations of optimisation

Such approach is similar to multigrid methods [14] that are used to solve partial differential equations by applying steps such as:

- Smoothing - applying several iterations of an iterative optimisation method to reduce error
- Restriction - Reducing to a coarser grid

- Interpolation - Interpolating the computed values on a finer grid

In this work a multilevel approach is possible only on uniform meshes with recurring structure with external information and what vertices do merge. The paper written by Kilian at el. [9] explain briefly the interpolation step, how the initial positions of the newly added vertices are set - linear combination of the relative position to the neighbouring vertices of the corresponding vertices in the fixed bordering meshes (proportional to distance to these two). However, this relies heavily on how the mesh simplification (or sometimes called decimation) is executed, which is not mentioned in their work.

Chapter 3

Proposed solution

3.1 Scope of work

This thesis contains a codebase that might be divided into 3 parts - Matlab code, C++ code and a CUDA kernel.

Matlab has been used in this work due to its qualities in quick prototyping, flexibility and easy visualisations. A first version of the algorithm is implemented in Matlab also with simple gradient descent method as well as with naive BFGS method [13] (C++ and CUDA part relies on third party libraries). The Matlab code also contains a mesh generator. A function that can generate a bar-looking mesh with specified attributes including length of the bar (number of layers or floors), level of details, rotation and curvature. It is possible to use some of the publicly accessible meshes, but having the option to generate a mesh (or rather a series of meshes) with desired properties offers more flexibility. Figure 3.1 shows how such mesh can be generated and respective steps. A validation of the gradient is verified by the finite differences method, which is also implemented in Matlab.

The C++ part contains interactive graphical application which allows user to examine the whole process of the interpolation step by step, run simple linear interpolation to observe the mesh deformation, run the optimisation step and do time or space refinement.

CUDA kernel is able to perform the mesh interpolation on GPU.

3.2 Used libraries

Several third-party libraries have been used in this work:

- **OpenFrameworks** [15] - It is relatively easy to use framework for 2D

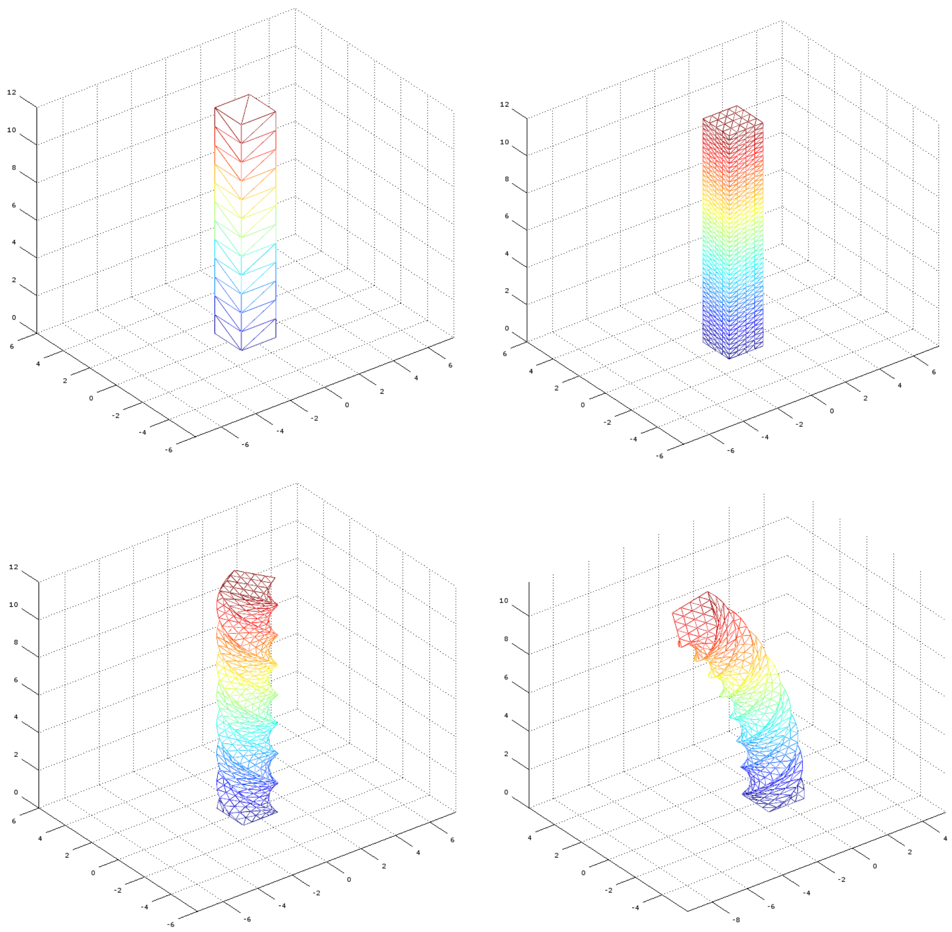

Figure 3.1: Steps in which an example bar mesh is generated

and 3D graphics. It based on OpenGL and free to use (being distributed under the MIT license). It is written in C++ and it is possible to be used only in supported IDEs, which makes portability and development across multiple platforms (e.g. Windows and OS X with Visual Studio and Xcode respectively) complicated.

- **Eigen** [16] - C++ library for linear algebra. It is made as templates, distributed only as header files, does not have any external dependencies except the standard library and has a strong emphasis on cross-platform compatibility. Eigen is being widely used across academia as well as industry. Eigen is being distributed under the MPL2 license.

- **CppOptimizationLibrary** [17] - C++ library with implementation of several numerical methods (such as Gradient Descent, Conjugate Gradient or BFGS) with emphasis on performance and simplicity of usage. Despite the statement that the Optimisation library is not dependent on additional dependencies it is built on Eigen, which it uses as for its type definitions (Matrix, Vector, Infinity).

- **CUDA L-BFGS** [18] - CUDA implementation of the L-BFGS method. It runs as a black box - user makes a derived class of the *cost_function* class, define two methods (the cost function itself and its gradient). Last update to this library has been done in the year 2012. It is distributed under CC BY 3.0 license (Creative Commons, Attribution).

Chapter 4

Implementation

4.1 Implementation details

Derivation validity - It is desired to verify that the equations (derivation) are correct, otherwise we would not obtain a correct gradient and the minimisation could not work. For such purpose we compare the derivation (analytic, explicit formula) with an approximation calculated with the finite difference method.

$$\frac{E(x)}{\partial x_i} = \lim_{h \to 0} \frac{E(x + hv_i) - E(x)}{h} \approx \frac{E(x + \epsilon v_i) - E(x)}{\epsilon}$$

In the equation above v_i represents a vector of the same size as x, full of zeros except the ith position, which is one. ϵ can be set to square root of a machine precision.

All the equation used in the gradient generation procedure has been tested by looking at the difference between analytic gradient and the one obtained by finite difference method and its result has members smaller than $1e - 7$ which lies below the numerical precision.

4.2 Test methodology and used meshes

Tests are partially run on a local machine and on a cluster of Institute of Computation Science of Università della Svizzera italiana, more specifically on a compute node with GPU.

ICS cluster GPU node specifications:

- **CPU**: 2x Intel E5-2650 v3, 20 (2 x 10) cores
- **GPU**: NVIDIA GeForce GTX 1080, 2560 CUDA cores
- **Memory**: 128GB RAM

Local Machine specifications:

- **CPU**: AMD Ryzen 5 1600, 6 cores (12 threads)
- **Memory**: 32GB RAM

Versions of software, libraries and frameworks used:

- **Matlab**: R2014b (8.4.0 150421)
- **Visual Studio**: Community 2015, 14.0.25431.01 Update 3
- **OpenFrameworks**: 0.9.8 (project for Visual Studio 2015)
- **Eigen**: 3.3.4
- **CUDA L-BFGS**: 1.0.3 (commit 7a9f786b23817e3c43bf0cb90ee3cb2978b0d702)
- **CppOptimizationLibrary**: commit e94e71c1e885ccec8c17500cde5c7a9eb2edb88b
- **Nvidia CUDA Compiler (NVCC)**: 8.0.26

A bar-shaped mesh has been generated for testing purposes with several levels of refinement with the following properties:

Mesh name	Refinement level	#vertices	#triangles
Testmesh0	0	44	84
Testmesh1	1	170	336
Testmesh2	2	674	1344
Testmesh3	3	2690	5376
Testmesh4	4	10754	21504
Testmesh5	5	43010	86016
Testmesh6	6	172034	344064
Testmesh7	7	688130	1376256

Figure 4.1: Number of vertices and faces for generated test meshes

In Test 1 I compare only time refinement. Testmesh0 is refined 13 times. Matlab code is also tested as a refference point.

Test 2 is running the complete C++ algorithm. It repeat 3x the following steps: Smoothing, smoothing, time refinement, space refinement.

4.3 Test results

Test 1 - time refinement:

Step	Matlab	C++
1	0.0117023	0.00125195
2	0.0174346	0.00319447
3	0.0280488	0.00479267
4	0.0356849	0.00895311
5	0.0639783	0.01635160
6	0.1257640	0.03412160
7	0.2454310	0.06654880
8	0.5263180	0.14157400
9	1.0076800	0.29367100
10	1.9933400	0.58383800
11	4.3546600	1.12278000
12	9.8749600	2.20736000
13	21.440200	4.34765000

Figure 4.2: Test 1, time refinement - measured in seconds

Figure 4.2 shows that growth rate appears to be constant, and that implementation in Matlab is 4-5 times slower, which is expectable due to the nature of the two languages.

Test 2 - complete algorithm:

Initial Mesh	Time
Testmesh3	1.10679
Testmesh4	3.96666
Testmesh5	26.91700

Figure 4.3: Test 2, complete algorithm - measured in seconds

figure 4.3 shows how well the algoritm scale for bigger meshes. It is important to note that the growth of vertices in the test meshes is exponential (e.g. Testmesh4 is has 4 times more vertices than Testmesh3).

Chapter 5

Conclusion

5.1 Evaluation

I presented an implementation of a modern shape interpolation algorithm including a prototype in Matlab and GPU kernel written in CUDA. I provided additional matlab code for generation of test meshes and verification of the implemented gradient calculation. The C++ impelemntation of the algorithm is wrapped in interactive graphical application which allows user to examine the method in an easily understood way.

5.2 Encountered problems

In this section I will shortly explain what problems have I encountered when working on the thesis. First and ever-present problem was numerical stability of the calculation. When optimising the mesh sequence I observed improvement of the mesh shapes and decrease of the energy, however when the convergence slowed down, it sometimes happened that the result came as a matrix of *NaN* (IEEE 754 floating point special value that represents a result of an operation that is not defined such as zero to the power of zero, sum of positive and negative infinity etc.)[19]. This would require more deliberate analysis to find the root cause, which is made harder by using third party libraries as black boxes and especially CUDA due to its parallel nature is hard to debug.

Another problem showed to be compatibility and support of used libraries. The Graphics framework OpenFrameworks has only limited number of supported IDEs. On a Windows machine I decided to use Visual Studio, which has only one supported version - 2015. Porting this codebase then to MacBook running OS X or PC running a Linux distribution has showed to be a complicated task. Also, the CUDA L-BFGS library is written as a CMake project, which can be imported to Visual Studio, but only it is more recent 2017 version. Even thou Visual Studio 2015 supports CUDA extended

C++ files (ending with .cu) and can compile them using the NVCC (Nvidia CUDA Compiler), an attempt to import the library into the project manually has failed and Visual Studio does not provide explanatory error messages for NVCC compiled files.

The numerical methods used in this work rely on the derivative of the energy function. If an incorrect derivative is provided, it might not be easy to detect. If a minor mistake is done, the optimisation method can still converge (however more slowly). A correctness of the derivative has been verified using the finite differences method, however the used C++ numerical library has a function to check if the solution is probably correct (using the same method), but this function claims the derivative is wrong. Therefore, there is either a mistake in the library, flaw in the verification using finite differences or the port of the code from Matlab to C++.

The implemented method has 3 steps - smoothening, time and space interpolation. When running the whole algorithm as a procedure rather than in interactive mode it is important to decide how many smoothening steps to apply before interpolation. Since the very general definition of the problem we can have meshes of various sizes (tens to hundreds thousands of vertices) and we might want to have a mesh sequence with different number of meshes as a result. The energy function returns a number that differs based on these factors and generally does not provide good information whether the current mesh sequence is good enough. Thus, a proper number of smoothening steps with respect to computational time and quality of the result is hard to determine.

The implemented method is described in the paper[9]. It defines its notation for vertecies, using top and bottom index to indicate position of shape in a sequence and position of a vertex within a shape. However sometimes the indices are switched, which causes confusion as the method explanation is dense.

5.3 Possible improvements

In order to solve the problem with platform and IDE dependence an option of replacing the OpenFrameworks with different alternative to do the visualisation might be considered. Currently the calculation itself relies on the *ofVec3f* class - an implementation of a vector of 3 numbers in single precision - and its methods as cross product and overloaded operators including multiplication with scalar and addition. This could be replaced with a class based on current Eigen functionality.

A general mesh decimation algorithm could be integrated or a proper bring-your-own-algorithm interface for mesh decimation and mesh hierarchy creation could be adopted in order not to be restricted to a special subclass of meshes with uniform recurring structure with additional vector reduction information.

A sophisticated suite of regression tests could be incorporated. Precomputed results for given meshes could be present for test comparrison. Currently it is hard to verify whether the algorithm is working properly as this is an optimisation problem.

As CUDA is proprietary to NVIDIA and restricted only to their GPUs. A kernel written in OpenCL would allow people to use the software regardless on their platform of use.

Bibliography

[1] Schaller, R. R.: Moore's law: past, present and future. *IEEE spectrum*, ročník 34, č. 6, 1997: s. 52–59.

[2] Ross, P. E.: Why cpu frequency stalled. *IEEE Spectrum*, ročník 45, č. 4, 2008: s. 72–72.

[3] Balladini, J.; Suppi, R.; Rexachs, D.; aj.: Impact of parallel programming models and CPUs clock frequency on energy consumption of HPC systems. In *Computer Systems and Applications (AICCSA), 2011 9th IEEE/ACS International Conference on*, IEEE, 2011, s. 16–21.

[4] Rahim, M. N. I. A.; Mazalan, L.; Adnan, S. F. S.: Analysis on parallelism between CPU and GPGPU processing on cluster computing. In *Computer Applications and Industrial Electronics (ISCAIE), 2014 IEEE Symposium on*, IEEE, 2014, s. 203–207.

[5] Marras, S.; Cashman, T. J.; Hormann, K.: A Mixed Shape Space for Fast Interpolation of Articulated Shapes. 2012.

[6] Alexa, M.; Cohen-Or, D.; Levin, D.: As-rigid-as-possible shape interpolation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 2000, s. 157–164.

[7] Sumner, R. W.; Popović, J.: Deformation transfer for triangle meshes. *ACM Transactions on Graphics (TOG)*, ročník 23, č. 3, 2004: s. 399–405.

[8] Winkler, T.; Drieseberg, J.; Alexa, M.; aj.: Multi-Scale Geometry Interpolation. In *Computer graphics forum*, ročník 29, Wiley Online Library, 2010, s. 309–318.

[9] Kilian, M.; Mitra, N. J.; Pottmann, H.: Geometric modeling in shape space. In *ACM Transactions on Graphics (TOG)*, ročník 26, ACM, 2007, str. 64.

[10] Fröhlich, S.; Botsch, M.: Example-Driven Deformations Based on Discrete Shells. In *Computer graphics forum*, ročník 30, Wiley Online Library, 2011, s. 2246–2257.

[11] Göddeke, D.: *Gpgpu-basic math tutorial*. Univ. Dortmund, Fachbereich Mathematik, 2005.

[12] NVIDIA, C.: NVIDIA CUDA C Programming Guide. 2012.

[13] Fletcher, R.: A new approach to variable metric algorithms. *The computer journal*, ročník 13, č. 3, 1970: s. 317–322.

[14] Wesseling, P.: Introduction to multigrid methods. *ICASE Report*, 1995.

[15] Lieberman, Z.; Watson, T.; Castro, A.: OpenFrameworks. 2016. Dostupné z: `http://openframeworks.cc`

[16] Guennebaud, G.; Jacob, B.; aj.: Eigen v3. 2017. Dostupné z: `http://eigen.tuxfamily.org`

[17] Wieschollek, P.: CppOptimizationLibrary. `https://github.com/PatWie/CppNumericalSolvers`, 2017.

[18] Wetzl, J.; Taubmann, O.: CUDA L-BFGS. `https://github.com/jwetzl/CudaLBFGS`, 2012.

[19] Kahan, W.: IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, ročník 754, č. 94720-1776, 1996: str. 11.

Appendix **A**

Acronyms

CUDA Compute Unified Device Architecture

BFGS Broyden–Fletcher–Goldfarb–Shanno

CPU Central Processing Unit

GPU Graphics Processing Unit

3D Three-dimensional

SIMD Single Instruction, Multiple Data

GPGPU General Purpose Computing on Graphics Processing Units

IDE Integrated Development Environment

MIT Massachusetts Institute of Technology

CC Creative Commons

MPL Mozilla Public License

IEEE Institute of Electrical and Electronics Engineers

NVCC Nvidia CUDA Compiler

ALU Arithmetic logic unit

GFLOPS Giga-Floating Point Operations Per Second

AMD Advanced Micro Devices

Appendix **B**

Contents of enclosed DVD

```
Code/OF_Application/apps/myApps/ThesisProject/src.Directory with
C++ source codes
 |
 |__ CudaLBFGS/include.....Additional custom header files for CUDA kernel
 |__ CudaLBFGS/projects ........... Directory with CUDA kernel source file
 |__ Latex ......... Directory with files related to LaTeX and PDF generation
 |   |__ BP_Hrabec_Simon_2017.tex ................. Template for final thesis
 |   |__ BP_simple.tex .................. Template for more compact version
 |   |__ images.....Directory with image files that are contained in the thesis
 |   |__ thesis.pdf..........................the thesis text in PDF format
 |   |__ thesis.ps.............................the thesis text in PS format
 |__ Matlab................................Directory with all matlab files
 |   |__ testbar0-7 .................................... Generated shapes
 |   |__ Mesh_generation ... Directory with code that creates example meshes
 |   |__ derivation_test.....Directory with code that implements test using
 |       finite differences
 |__ Output................................Directory with generated PDFs
 |   |__ BP_Hrabec_Simon_2017.pdf..........Proper version of thesis in PDF
 |   |__ BP_simple.pdf......................More compact version of thesis
 |__ Text...........................Directory with actual text of the thesis
 |__ Makefile.................Makefile that runs LaTeX and generate PDFs
```