Development of GPU-efficient visualization and segmentation algorithms for 3D medical data

Doctoral-Meeting initiative

Julián Lamas-Rodríguez

Centro de Investigación en Tecnoloxías da Investigación
University of Santiago de Compostela
PhD advisors: Francisco Argüello and Dora B. Heras

Contents

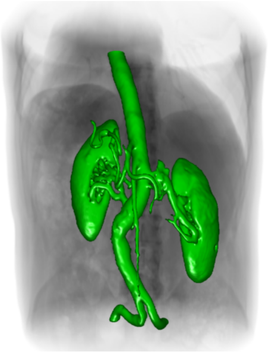
Contents

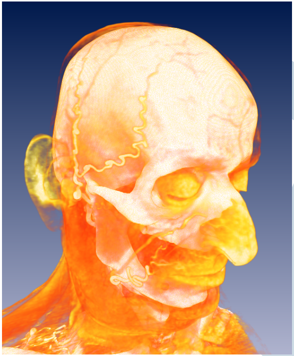
CITIUS

Processing of medical data

Typical applications with medical images:

- Manual and automatic segmentation
- Visualization of 3D volumes





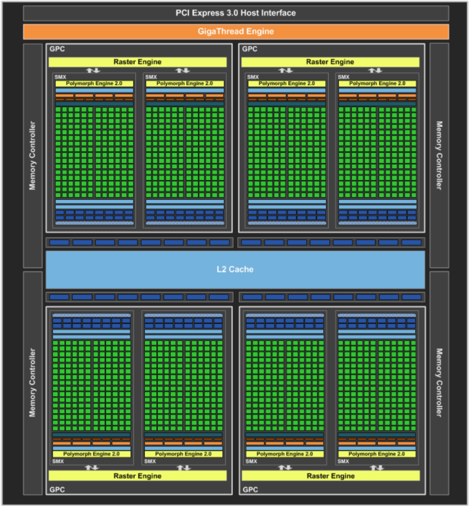
Processing of medical data

Main challenges

- Very computing-intensive tasks that require thousands of computations
- Results used for diagnosis should be obtained as quick as possible
- Use a high-efficient low-cost computing platform

How general-purpose GPUs can help

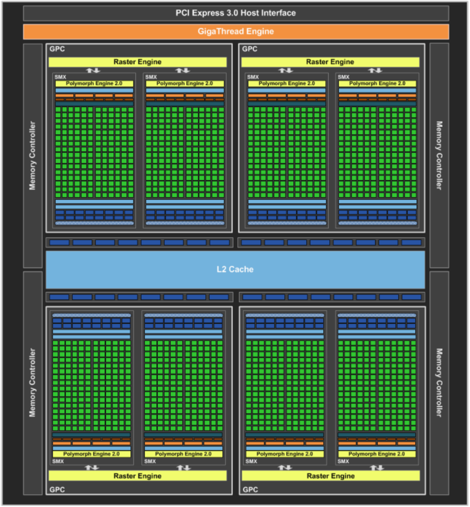
1. GPUs are nowadays commodity massive parallel processors which can efficiently execute thousands of threads in parallel
2. GPUs can be used to speedup computing-intensive algorithms



NVIDIA GTX 680 with Kepler GK104 architecture

Strengths

- Supports executing thousands of threads in parallel thread blocks that work independently
- A two-level cache hierarchy speedups accesses to global memory
- Besides the register space, the programmer can use the shared memory space and the texture cache



NVIDIA GTX 680 with Kepler GK104 architecture

Weaknesses

- The programmer must take into account the locality of global memory accesses to maximize throughput
- Accesses to shared memory space must be organized to avoid bank conflicts
- Minimize the use of registers and thread divergence ⇒ GPU code does not benefit from a complex logic

How suitable is an algorithm for a GPU?

Suitable

- High level of **data** parallelism
- No data dependencies
- Small number of R/W operations on memory

Unsuitable

- High level of **task** parallelism
- Lots of data dependencies: between neighbors, between stages, etc.
- Lots of R/W on memory

- Many algorithms are at an intermediate point
- Algorithms must be adapted to GPU to overcome the difficulties

How suitable is an algorithm for a GPU?

Suitable

- High level of **data** parallelism
- No data dependencies
- Small number of R/W operations on memory

Unsuitable

- High level of **task** parallelism
- Lots of data dependencies: between neighbors, between stages, etc.
- Lots of R/W on memory

- Many algorithms are at an intermediate point
- Algorithms must be adapted to GPU to overcome the difficulties

Contents

Hypothesis and objectives

Hypothesis

Is it possible to develop efficient schemes of segmentation and visualization of large 3D medical datasets in commodity GPUs?

Objectives

- Develop GPU-efficient solutions for volume segmentation based on level sets
- Develop a complete pipeline for visualization of large volumetric datasets on the GPU

Methodology: fundamental steps

1. Analyze the current state of the art, the theory behind each algorithm, the mathematical/physical fundamentals, etc.
2. Select algorithms in the literature that are suitable for GPU
3. Evaluate results by comparing with other implementations in the literature or implementations in CPU

Develop a GPU-efficient scheme for segmenting and visualizing large volumes of 3D data

Methodology: fundamental steps

1. Analyze the current state of the art, the theory behind each algorithm, the mathematical/physical fundamentals, etc.
2. Select algorithms in the literature that are suitable for GPU
3. Evaluate results by comparing with other implementations in the literature or implementations in CPU

Develop a GPU-efficient scheme for segmenting and visualizing large volumes of 3D data

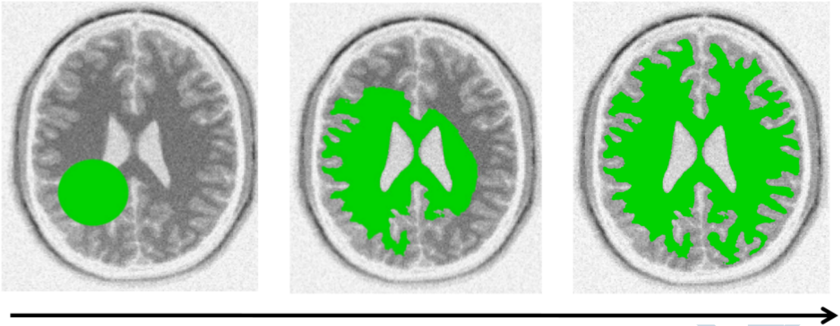
Contents

CITIUS

1. Implementation of segmentation algorithms on GPU

Image segmentation using level sets

Iteratively deform a surface in the direction of its normal based on surface and image properties



Level-set solutions considered in this work

Integer-based level set by Shi and Karl

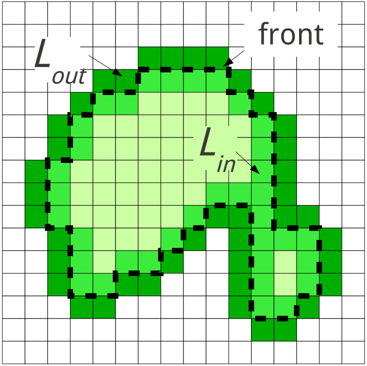
Approximate level-set computations using integer operations

Implicit active contour by Weickert and Kühne

Each iteration requires the resolution of a system of linear equations that can be decomposed into two tridiagonal systems

Integer-based level set

Front evolution

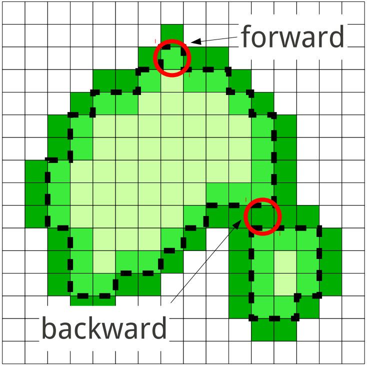


- Two lists of points are defined: L_{in} and L_{out}
- Each step makes adds and/or removes points from those lists to move the front

Premises

- Minimize the number of computations
- Focus on integer operations

Front evolution

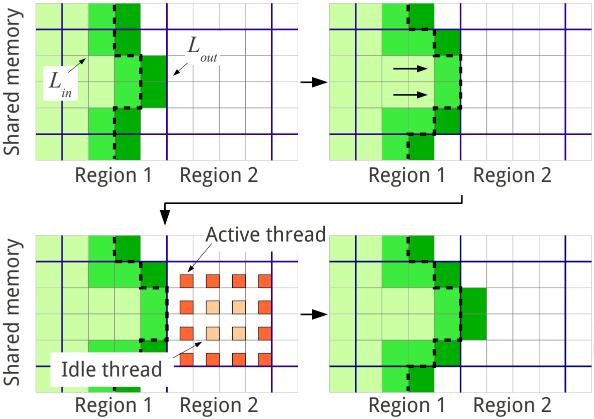


- Two lists of points are defined: L_{in} and L_{out}
- Each step makes adds and/or removes points from those lists to move the front

Premises

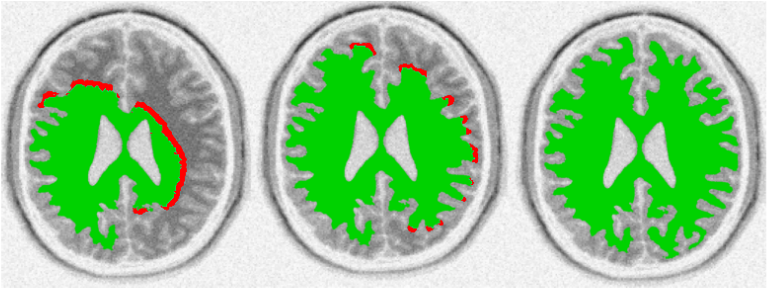
- Minimize the number of computations
- Focus on integer operations

GPU implementation: inconsistencies between regions



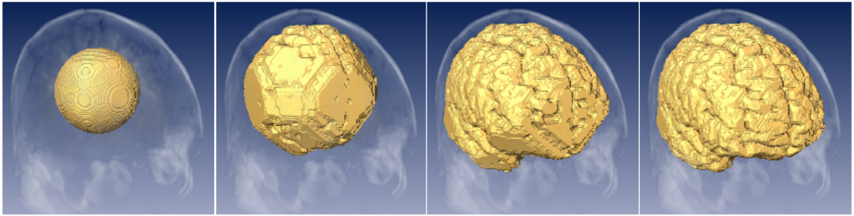
- ▷ Load elements in the outer borders into shared memory
- ▷ Fix inconsistencies using at the beginning of every iteration

GPU implementation: identify active regions



- ▷ Avoid unnecessary computations
- ▷ At the end of an iteration, a thread in each block checks for:
  - If its region must be active for the next iteration
  - If neighboring regions must be active for the next iteration

Results



Algorithm	Time	Speedup	Dice coefficient
CPU	2.4	1.0x	0.95
GPU	0.6	3.8x	0.96

Results in seconds for an NVIDIA GTX 580

Segmentation in action

Video demo

Implicit active contour level set

Parallel tridiagonal-system resolution

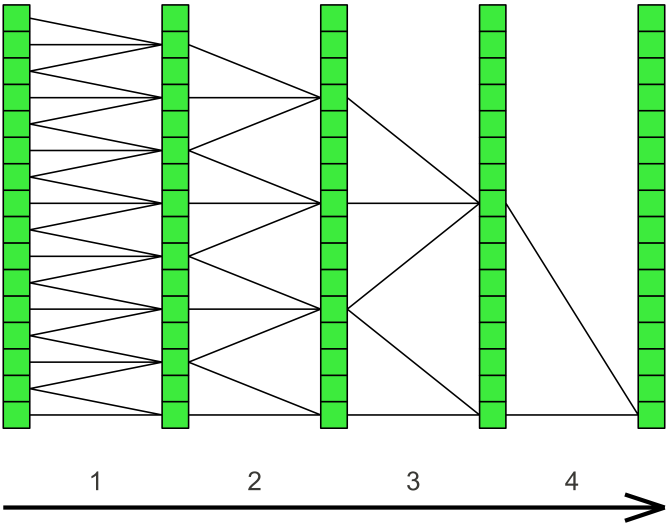
- In this method, the resolution of the level-set requires solving tridiagonal systems:

$$\begin{pmatrix} b_1 & c_1 & & 0 \\ a_2 & b_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ 0 & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} \tag{1}$$

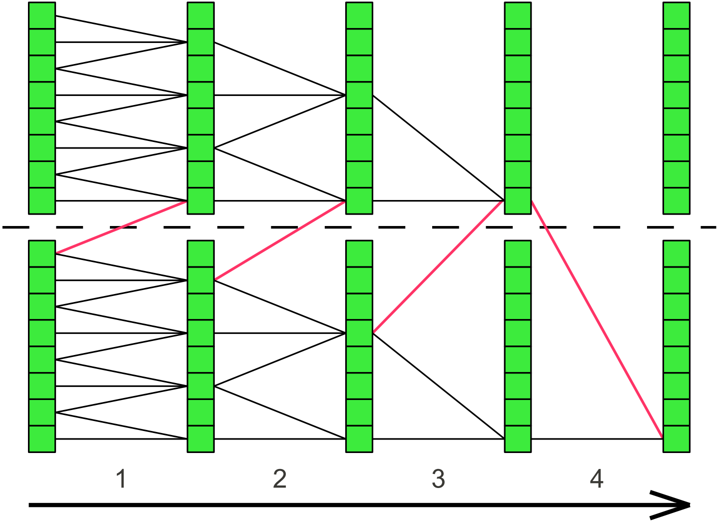
  Thus, we must consider how to efficiently solve these systems on GPU

- Two non-iterative parallel algorithms that have been studied:
  - Cyclic reduction
  - Recursive doubling

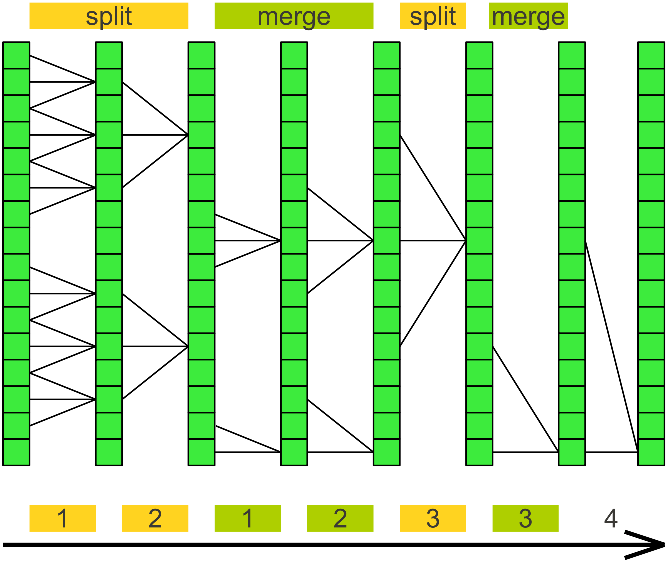
Split-and-merge technique



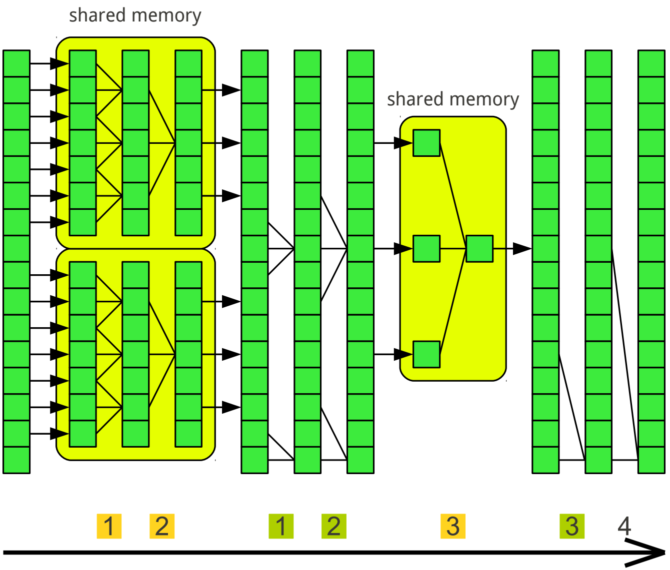
Split-and-merge technique



Split-and-merge technique



Split-and-merge technique



Keys and results

- A large tridiagonal systems cannot be solved by a single thread block
- Equations are distributed between thread blocks, and partial results are combined to obtain the global solution
- The split-and-merge technique exploits the shared memory space

	Single precision		Double precision	
Algorithm	***Time***	***Speedup***	***Time***	***Speedup***
OpenMP Bondeli's alg.	27583	1.0x	43435	1.0x
Cyclic reduction	1808	15.3x	2524	17.2x
Recursive doubling	6518	4.2x	10625	4.0x

Results in μs for a one-million equation system in an NVIDIA GTX 580

2. Implementation of visualization algorithms on GPU

Large volume rendering

Keys

- Volume rendering is a compute-intensive task which has already been successfully implemented on GPU
- In the recent years, improvements to data-acquisition methods have increased the size of volumetric datasets

Main challenge

- Render volumetric datasets within the limited memory restrictions of a GPU

Tasks to undertake on GPU

- Compression: wavelet transform and Ihm & Park's encoding
- Rendering: using bricking and texture mapping

Compression

Wavelet transform

Splits the volume data in low and high-frequency coefficients

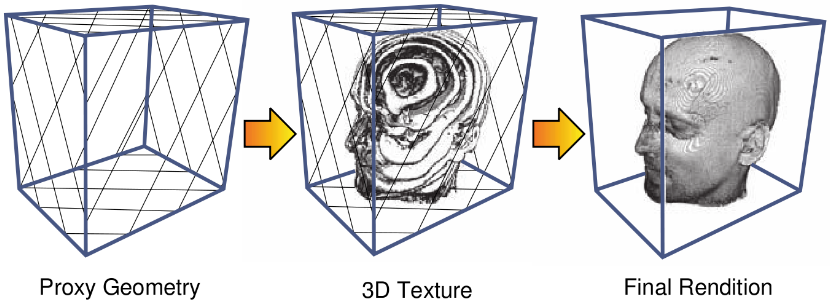
- High-frequency coefficients have close-to-zero values, and can be ignored without supposing an important loss of information
- High-frequency coefficients are stored together, which reduces the final compressed volume size

Encoding

- It allows quick access to arbitrary voxels
- It has already been used in large volume rendering

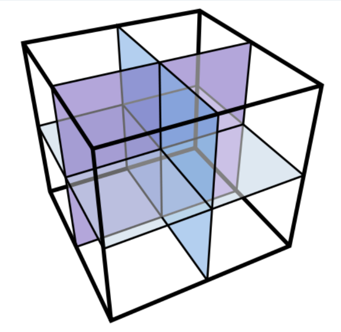
Rendering

Texture mapping



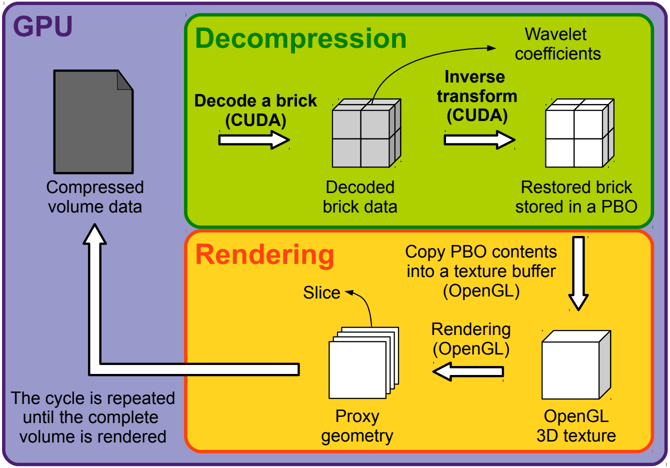
Rendering

Bricking



GPU visualization pipeline for large volumetric datasets

- **Decoding** and **inverse transform** steps are parallelized in GPU
- The complete pipeline (**decompression** and **rendering**) is executed on GPU, instead of a CPU-GPU hybrid approach



GPU decompression

Decoding

- A non compute-intensive task with thread divergence and low spatial locality
- Compressed volume is stored in texture memory to speedup read operations

Inverse wavelet transform

- The transform can be computed in shared memory
- Shared memory is very limited, so not all coefficients are stored on it
- Data has to be organized to achieve the maximum throughput from the memory operations

Visualization in action

Video demo

Contents

CITIUS

Benefits

- Speeding up common tasks in the processing of medical data, such as segmentation and visualization, can provide quicker diagnoses from which doctors and patients can benefit

- Solutions that can be executed in commodity hardware reduce costs and increase their accessibility to a wide range of users

Future work

Segmentation

- Complete GPU implementation of active contour level-set segmentation
- Integrate complete segmentation solution into Amira

Visualization

- Improve the efficiency of the compression+rendering pipeline
- Optimize visualization for large volumes by skipping regions that don't add to the final rendering

Collaborations

GPU segmentation

Implemented in Amira, a visual tool developed in the Zuse Institute Berlin

In collaboration with Dagmar Kainmüller, Stefan Zachow, and the Dept. of Vis. and Data Anal.

GPU visualization

Implemented in Volv, a visual tool developed in the ICCAS (University of Leipzig)

In collaboration with Daniella Wellein, Silvia Born, Matthias Pfeiffle and Oliver Burgert

Thank you!