Zeszyty Naukowe WWSI, No 25, Vol. 15, 2021, pp. 57-67
DOI: 10.26348/znwwsi.25.57

3D Medical Segmentation Visualization in Julia with MedEye3d

Jakub Mitura* [1] and Beata E. Chrapko[2]

[1]Medical University of Lublin
[2]Chair and Department of Nuclear Medicine, Medical University of Lublin

Abstract

MedEye3d is a Julia language package designed to simplify visualizations of segmentation in three dimensional setting. Motivation to develop this application was to provide to rapidly growing Julia language scientific community tool for research in three dimensional medical images. Package is based on multiple open source software packages, yet most prominent is utilization of OpenGl specification to enable GPU acceleration. Application was tested both on Linux and Windows platforms and in both cases latency observed by the user in most common interaction like scrolling, annotation and change of displayed plane was very small. Thanks to utilization of many modern packages and methodologies developed package is providing convenient visualization in rapid prototyping with medical image segmentation algorithms. Application also is easily extendable and will be included in medical image segmentation framework that is currently in development.

***Keywords* —** OpenGl, Computer Tomagraphy, PET/CT, medical image annotation, medical image visualization

1 Introduction

Image segmentation in the medical domain has multiple use cases. Most importantly it enables delineation of physiological and pathological structures, in order to confirm or reject some diagnostic hypothesis. In case of all segmentation problems, a very important step in evaluation of the segmentation algorithm output is visual inspection. Such inspection enables researchers that are responsible for creating and or evaluating developed algorithms to easily spot problems, and compare different algorithms in more detailed ways than can be achieved by usage of segmentation metrics alone. However in order for such in development visual evaluation to be useful it needs to meet some usage criteria:

*E-mail: jakub.mitura14@gmail.com

Manuscript received August 30, 2021

- It needs to be easily integrable to the programming language and libraries used by researchers.
- Performance of the tool must be adequate in order to suit the iterative process of algorithm development and refinement.
- Representation accuracy must be sufficient for the task at hand. It should not require an excessive amount of computational resources, in order to minimize its influence on running algorithms.
- Support for in memory data structures (arrays) should be convenient.
- Needs to provide possibility of simple manual annotations, on given mask and ability to control visibility and visual representation of given mask.
- Should provide also the possibility to display some metadata in text format like segmentation metrics for example Dice score [1].
- Ideally it should be also open source and well documented in order to enable users to modify it according to the task at hand.

In order to address all of those issues in the medical domain and Julia language ecosystem the described below package was developed.

2 Related software

There are already multiple applications providing the possibility to open and visualize medical data like computer tomography scans like for example Amide [2] and 3D Slicer [3]. These tools provide feature rich, mature and performant medical image visualization, yet to the best knowledge of the author none of them provides any inbuilt integration with Julia language [4]. Although it is possible to easily call either Python or C functions from Julia such usage leads to some performance penalty, and makes visualization tool modification and debugging highly inconvenient.

Another problem common for many of present tools is slow start up time related to added multiple features, that are important in precise diagnosis and annotation by clinicians, but are not so important in case of rapid development phase, the same qualities also frequently leads to extensive usage of computational resources what leads to problems in visualize the algorithm progress when it is running.

Further problems may arise in case the user tries to visualize data stored in arrays, passed directly from program in development rather than files of appropriate characteristics.

3 Package goal and contribution to the research community

Important practical question remains whether development in Julia language has any benefits over using more established languages and tools. First Julia language is posed to solve the two language problem, where rapid prototyping with dynamic language like Python needs to

be translated later to highly performant language like C++. From the practical perspective of the user of the developed tools it leads to many problems with compatibility between different software packages and tools. This problem in the Julia language ecosystem is highly limited because of both some design choice, well developed package manager, and relatively young code bases that limits the amount of the legacy code. Because of it and the multitude of research oriented packages that span most areas of engineering and mathematics implementing or using nearly all popular algorithms in vast majority of settings is greatly simplified. As a response to the limitations of existing tools in a very specialized use case for visualization in the rapid development phase of algorithm development in the Julia language, MedEye3d was developed.

4 Material methods and Experiments

Multiple freely available software packages were used in implementation. Most of the graphics display is controlled by OpenGL [5], with some julia helper libraries like ModernGl [6], FreeTypeAbstractions [7], Glutils [8]. Rocket.jl [9] (reactive functional package) was used to manage user Interaction. In order to increase productivity multiple utility packages were used like Dictionaries.jl [10], Parameters.jl [11], Setfield.jl [12], Match.jl [13]. In order to provide a simpler way of package modifications, the DrWatson.jl [14] package is used throughout the program. Program was developed using Julia 1.6 on both Ubuntu and Windows systems by a single developer in publicly available GitHub repository.

All experiments were conducted on two machines Windows PC (10th Generation Intel Core i9 processor, GeForce RTX 3080) and Ubuntu laptop (9th generation Intel Core i7 processor, GeForce GTX 150). The goal of utilizing two different software and hardware setups was to prove possibility of utilizing the package in most frequent configurations. Datasets utilized in experiments originated from publicly available dataset for CT scan SILVER07 dataset [15] (example of displayed data can be seen on Figure 1), and from Head-Neck-PET-CT (Positron Emission Tomography with Computer Tomography) dataset from Cancer Imaging Archive [16]. Because of the asynchronous nature of execution of many parts of the algorithm and different resource utilization by other running tasks the results may vary. In order to measure actual time to render the GPU synchronization was performed using OpenGl glFinish() function. Cross Section plane translation is defined as changing the single coordinate which defines the plane in the chosen cross section. Time needed to accomplish it is measured in milliseconds between rendering commands using BenchamrkTools.jl [17].

Response to mouse left click time was measured as a difference of time reported in the callback registered to GLFW window and end of the execution of a rendering command designed to render the visible change on the chosen texture.

Time required for cross section plane change (for example from transverse to sagittal) was defined as a difference in time from invocation of GLFW callback (which itself will be invoked on appropriate user interaction) to end of rendering function execution. Results of experiments mentioned above are summarized in Figure 2. What is clearly visible time required for completion of functions in case of the Ubuntu laptop was longer, yet considering far inferior hardware characteristic it was to be expected, also reduced performance did not affected user experience.

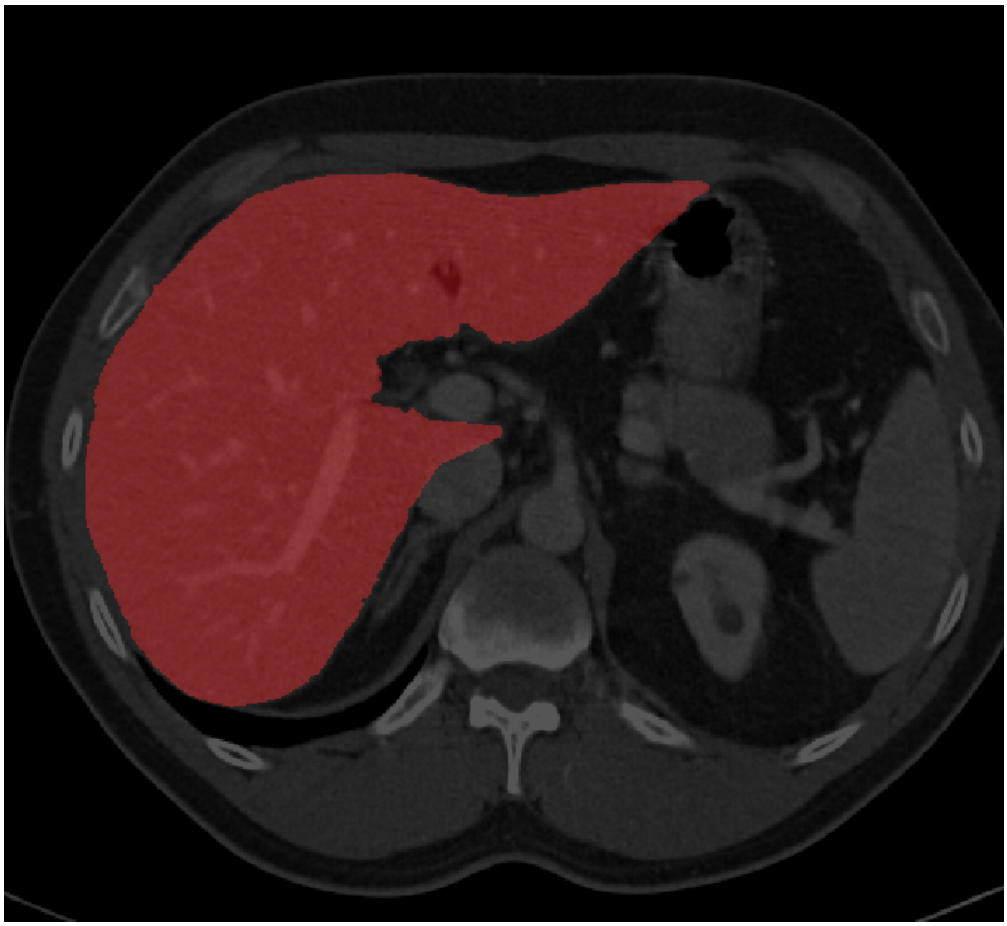

Figure 1: Transverse image, soft tissue window CT scan and gold standard liver segmentation

Memory usage was tested using actual data from PET/CT. Data uploaded to display constituted 3 Matricies of Float32, UInt8 and Int16 dataTypes, additionally 2000x8000 matrix holding text data was displayed. In case of Windows 10 system Dedicated GPU memory usage was evaluated using Details section of Task Manager and was oscillating between between 38K and 39.5K. In case of Ubuntu Laptop application used to estimate GPU memory usage was Nvidia System Monitor and dedicated GPU memory usage oscillated around 21MB.

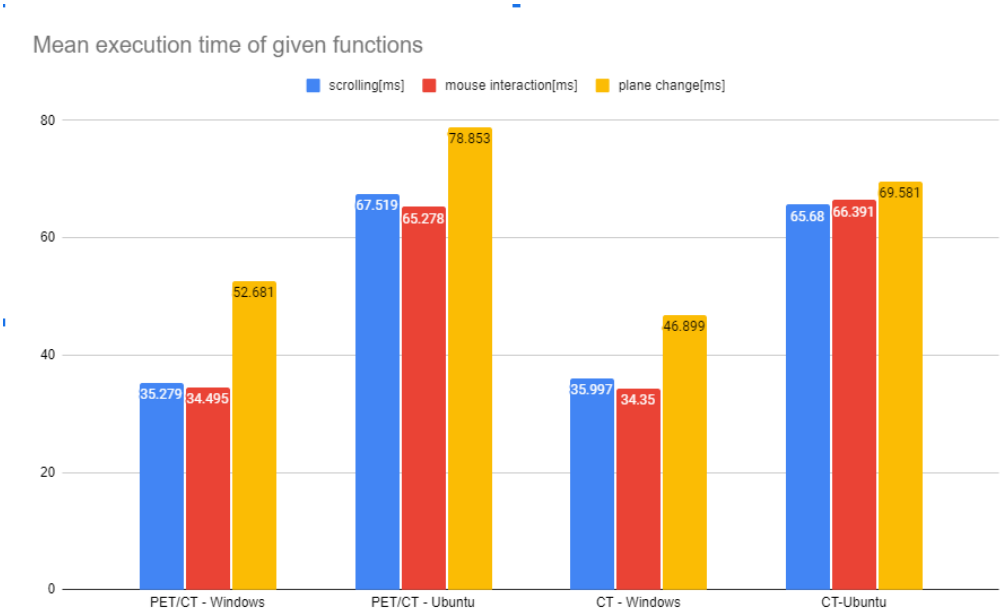

Figure 2: Execution times depending on function and machine used

5 Discussion

In order to address all of the criteria set in the introductions that are needed to met the goal of this package specific design choices were made. First requirement of integrability to the programming language and libraries used by researchers is obviously satisfied for Julia language users as the package is fully written in Julia code.

In order to provide sufficient performance rendering is done with GPU acceleration thanks to OpenGL [5]. Thanks to contiguous memory allocation of Julia arrays there is little or no need of preprocessing data before sending it to GPU memory. In order to minimize memory requirement but at the same time take advantage of fast texture GPU memory all binary mask data is stored and sent to GPU as an unsigned bytes array. Fast asynchronous user input processing is achieved thanks to reactive functional architecture implemented with help of Rocket.jl [9] package.

As most algorithms in digital processing can be accelerated on GPU it is of paramount importance to reduce GPU memory consumption by the viewer. This goal was achieved by minimizing data required by the given texture using minimal size integer or floating point textures (avoiding RGB textures), and sending to the GPU only a single slice (2 dimensional matrix form 3 dimensional image). The last decision may in some cases reduce the speed of switching between slices, yet in most cases it will not lead to change visible for the user. Another decision that limits resource usage and that fits well to the goal of this package is complete lack of the

Figure 3: Transverse image, soft tissue window only CT scan

graphical user interface – all interactions are either mouse interactions with the image itself or controlled by keyboard shortcuts. All functions can be also invoked via, for example, the REPL interface.

In order to represent 3D medical images properly some typical for this domain problems need to be addressed. First In order for the image viewer to be fully functional it needs to enable visualization of the medical image in coronal, sagittal and transverse planes. In case of the transverse plane it usually does not require much modifications, yet because of the variable thickness of slices, visualization in the coronal and sagittal plane needs some adjustment which is addressed in this package by dynamically setting vertices positions which are defining rectangular surface for texture display. Hence the height to width ratio that can be inferred from medical image metadata (voxel dimensions) will be preserved.

Data structures used in the described package are operating on arrays themselves or in order to reduce memory usage on their views. As OpenGl is characterized by some specific data types the package has mechanisms to automatically infer the optimal OpenGl types needed for a texture at hand on the basis of supplied Julia type.

Although purposefully no graphical user interface was developed, multiple keyboard and mouse interactions were developed. In order to increase ease of interaction each texture with which we wish to interact with needs to have some associated number. Using the mentioned number then we can set this mask to invisible by pressing *Ctrl + number* or visible by *Shift + number*. Example is visible in Figure 3. where only CT data is visible.

We can also define the difference between two masks which will show us the pixels where value in one maskA is bigger then in mask B. Activating the view of this difference is done by pressing *Shift + maskA number + "m" + maskB number*, analogically when we want to make this mask invisible we just need to press *Ctrl + "m"*. Example visible in Figure 4 where mask we evaluated cover bigger area than gold standard (visible in Figure 1), only difference pixels are displayed. In order to activate fast scrolling *"f"* needs to be pressed and in order to return to slower scrolling *"s"*.

In order to annotate given mask we activate it to be modified by pressing *Alt+number*. We set the integer value that will be set to this mask by pressing *Tab+number*, default value is *1*. In order to erase value we can just set value to *0*. Value chosen as a new value for pixels where the user would click will be seen at the bottom of the text panel. Then we can add chosen integer values to the mask by just left click and drag. In order to controll the area of the surface annotated surface related to single pixel point (where user clicked) we can use *Tab + "+"* or *Tab + "–"*. Such operation will modify the underlying 3 dimensional data in the slice we are currently on.

Thanks to the ability of setting different numbers in a mask we get a simple way of discerning separate structures of the same type (all are in the same mask) but different instances (different numbers set onto those masks). Other possibilities to use this system also exist. Further processing like for example defining volume from annotated points (for example using convex hull algorithms) is on a user side, as it may mean multiple things – surface, volume, marking artifact, saving point for further reference and for example expert consultation etc.

In order to change the plane of view (transverse, coronal, sagittal) we press *Space + 1* or *2* or *3* depending on the dimension that we want the plane to be in (example of transverse plane display can be seen on Figure 5). For display different window (bone, soft tissue, lungs) we can press *F1*, *F2*, *F3*.

Keys *F4* and *F5* with combination of *"+"* and *"–"* signs are used to increase or decrease given threshold. In case of continuous colors it will clamp values – so all above max will be equaled to max, and min if smaller than min. In case of main CT mask – it will control minimum voxel value where it is displayed as white and maximum voxel value when it is shown as black (for standard windows we can just use *F1*, *F2*, *F3*). In case of masks with single color associated we will use step functions so if data is outside the range it will return 0 – so will not affect display.

Metadata like segmentation metrics (like Dice, Jaccard etc. scores) [1] can be assigned to the whole three dimensional scrolling data and to each slice, given data will be presented in the text panel on the right. In case of the data associated with the whole data it will be displayed constantly on the top, slice defined data only when a given slice is displayed. Additionally some data related to current action like pointing to what number is used in manual modifications will be displayed at the bottom.

Figure 4: Transverse image, soft tissue window, only difference between masks

6 Conclusions and further developments

Developed package can be a useful tool in setting the rapid development of medical image segmentation. It is not intended as a replacement to already developed software tools, but a useful addition in the setting mentioned above. In order to maximize the usability in the rapid development setting multiple design decisions were made like independence from file system, GPU acceleration with limiting GPU memory usage, possibility of very simple yet expressive annotation and control system. Tool is also fully controllable from the code enabling any automatization that would be useful to the user.

MedEye3d is intended as a part of a bigger framework developed by the author that includes, segmentation evaluation package, data persistence package (using HDF5 filesystem),

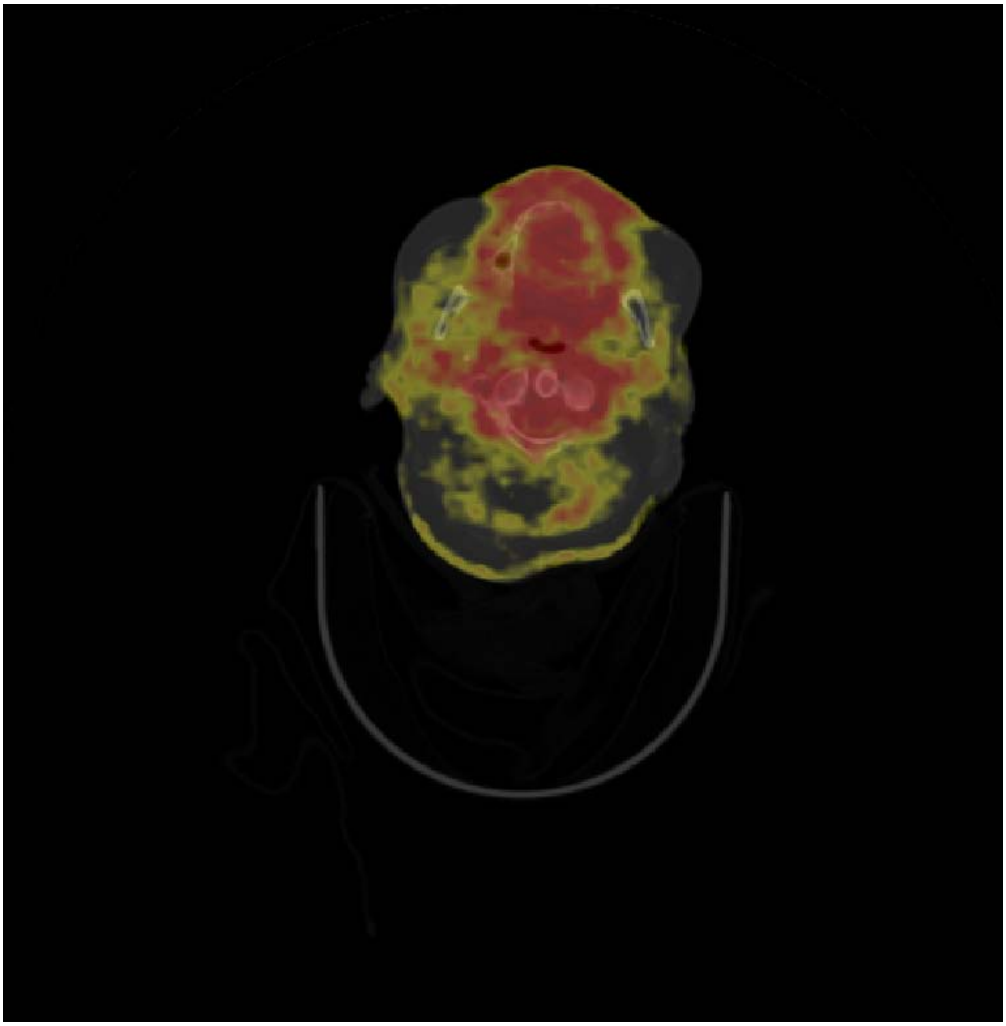

Figure 5: transverse Display FDG PET/CT

and segmentation pipeline package (which will connect all mentioned elements in coherent framework). All packages are developed in Julia, and are in the early development phase at the time of writing. MedEye3d package is intended to be lightweight, yet in case of features requests those will be provided if possible, most further work is intended to be in optimization understood as increased responsiveness and reduced computational resources demand.

Acknowledgement

The package is open source and available on GitHub[1] under the Apache license. Thorough documentation of all present functions was developed, and examples presented in the readme section of GitHub repository. In case of any feature request, bug reporting, contribution propositions contact the author via LinkedIn[2] or in the issues section of GitHub repository. Video tutorial representing usage and code structure (to enable custom modifications by the user) was published via YouTube and embedded in GitHub readme file[3].

References

[1] J. Bertels, T. Eelbode, M. Berman, D. Vandermeulen, F. Maes, R. Bisschops, and M. B. Blaschko, "Optimizing the Dice Score and Jaccard Index for Medical Image Segmentation: Theory and Practice," in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, D. Shen, T. Liu, T. M. Peters, L. H. Staib, C. Essert, S. Zhou, P.-T. Yap, and A. Khan, Eds. Cham: Springer International Publishing, 2019, pp. 92–100. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-32245-8_11

[2] A. M. Loening and S. S. Gambhir, "AMIDE: A Free Software Tool for Multimodality Medical Image Analysis," *Molecular Imaging*, vol. 2, no. 3, pp. 131–137, 2003. [Online]. Available: https://doi.org/10.1162/15353500200303133

[3] R. Kikinis, S. D. Pieper, and K. G. Vosburgh, *3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support*. New York, NY: Springer New York, 2014, pp. 277–289. [Online]. Available: https://doi.org/10.1007/978-1-4614-7657-3_19

[4] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017. [Online]. Available: https://doi.org/10.1137/141000671

[5] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[6] SimonDanisch, rennis250, and o jasper, "ModernGL.jl," 2021. [Online]. Available: https://github.com/JuliaGL/ModernGL.jl

[7] SimonDanisch and aaalexandrov, "FreeTypeAbstraction.jl," 2021. [Online]. Available: https://github.com/JuliaGraphics/FreeTypeAbstraction.jl

[8] jorge brito, "Glutils.jl," 2021. [Online]. Available: https://github.com/jorge-brito/Glutils.jl

[9] D. Bagaev, "Rocket.jl," 2021. [Online]. Available: https://github.com/biaslab/Rocket.jl

[1] `https://github.com/jakubMitura14/MedEye3d`
[2] `https://linkedin.com/in/jakub-mitura-7b2013151/`
[3] `https://youtu.be/tv7-nGiik-w`

[10] A. Ferris, "Dictionaries.jl," 2021. [Online]. Available: https://github.com/andyferris/Dictionaries.jl

[11] Mauro, "Parameters.jl," 2021. [Online]. Available: https://github.com/mauro3/Parameters.jl

[12] J. Weidner, "Setfield.jl," 2021. [Online]. Available: https://github.com/jw3126/Setfield.jl

[13] K. Squire, "Match.jl," 2021. [Online]. Available: https://github.com/kmsquire/Match.jl

[14] G. Datseris, J. Isensee, S. Pech, and T. Gál, "DrWatson: the perfect sidekick for your scientific inquiries," *Journal of Open Source Software*, vol. 5, no. 54, p. 2673, 2020. [Online]. Available: https://doi.org/10.21105/joss.02673

[15] T. Heimann, B. van Ginneken, and M. Styner, "SILVER07," 2021. [Online]. Available: http://www.sliver07.org/

[16] M. Vallieres, E. Kay-Rivest, L. J. Perrin, X. Liem, C. Furstoss, H. J. W. L. Aerts, N. Khaouam, P. F. Nguyen-Tan, C.-S. Wang, K. Sultanem *et al.*, "Radiomics strategies for risk assessment of tumour failure in head-and-neck cancer," *Scientific reports*, vol. 7, no. 1, pp. 1–14, 2017. [Online]. Available: https://doi.org/10.1038/s41598-017-10371-5

[17] J. Revels, "BenchmarkTools.jl," 2021. [Online]. Available: https://github.com/JuliaCI/BenchmarkTools.jl