3D Beamforming for Matrix US probe

Klinisches Anwendungprojekt

14th October, 2019

1 Background

The Chair for Computer Aided Medical Procedures published with SUPRA an open source pipeline for fully software defined ultrasound processing for real-time applications [1]. Hence, SUPRA covers everything from beamforming to the output of B-mode images, running on consumer GPUs in real-time both in 2D and 3D.

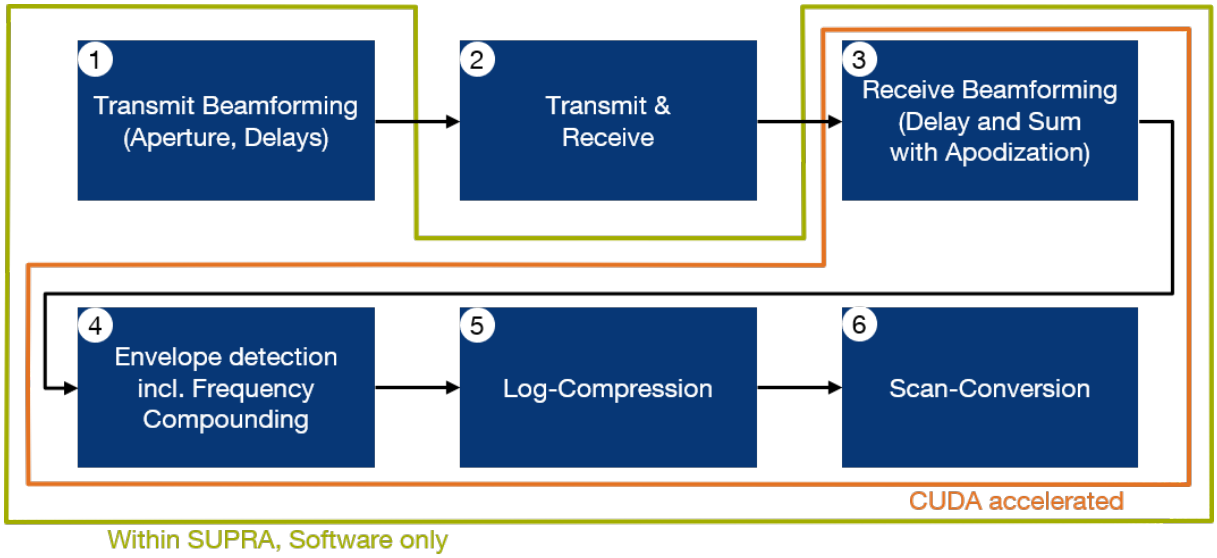

Figure 1: Pipeline of SUPRA

Figure 1 shows the concept behind SUPRA. The only part, not done by the software, is the transmission and reception of the ultrasound waves. Everything else takes place in the very complex implementation of SUPRA which can be found on GitHub[1].

[1] `https://github.com/IFL-CAMP/supra`

2 Project

The first part of the project would be to understand the complex system behind the already existing implementation and to see how the pipeline works. Moreover, it has to be figured out, where exactly the beamforming is done in order to be able to look more precisely at the implemented algorithms and find possible ways to improve them.
Secondly, the sudent is welcome to add several comments to already implemented methods in order to provide a small documentation for each method. The problem, SUPRA has until now, is the shortage of documentation and due to the complexity of the system, it is very difficult to feel one's way through all files. By adding some comments and providing a small documentation for some of the methods, future works can be made easier if not everybody has to start all over again and skim through all the files.

Last, the attempt of improving the already existing beamforming method for the 3D US transducer is the central part of this project. By reading several papers about beamforming, the student is also welcome to implement other beamforming methods (also for 2D beamforming), as sooner or later he will come across some versions which differ from the implemented ones. Until now, this algorithm has a high resolution of the created image in the sagittal plane but then has a low resolution in the longitudinal plane. Finding at least an idea how to achieve the reduction of the image artifacts due to cross-talk can be helpful for any future work with SUPRA.

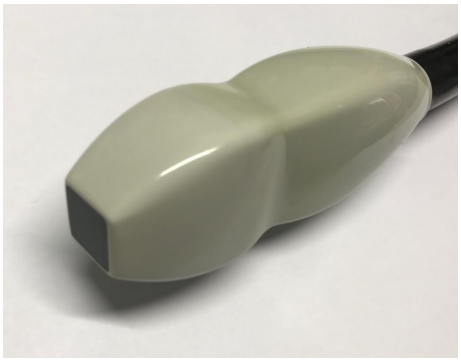

Figure 2: 2D matrix transducer

3 Implementation

3.1 Delay and Sum for 3D

For the implementation of the Delay and Sum beamformer refer to the files in src/SupraLib/Beamformer starting with RxSampleBeamformerDelayAndSum. The basic idea of this adaption of the existing algorithm is to use the raw data received from the transducer and to perform the beamforming in direction of the X axis and then in the direction of the Y axis. Both results are then added and returned.

- RxSampleBeamformerDelayAndSum.h:

  By using this implementation, the algorithm is designed to iterate along the X axis and for each element to iterate over the Y axis. For the summation of the values, the weights of the corresponding elements are considered. In general, the file has remained unchanged during the implementation of the beamformers.

  The implementation provides two methods, one for the beamforming in 2D and one for 3D.

  - sampleBeamform2D(): This method performs the Delay and Sum beamforming algorithm for 2D. This file has remained unchanged during the implementation of the changes of the 3D beamformer.
  - sampleBeamform3D(): This method performs the Delay and Sum beamforming algorithm for 3D. This file has remained unchanged during the implementation of the changes of the 3D beamformer. The algorithm is designed to iterate over the X axis. During this iteration, the values along the Y axis are summed up (multiplied with the corresponding weights).

- RxSampleBeamformerDelayAndSumYX.h:

  By using this implementation, the algorithm is designed like the algorithm of the beamformer above, just swapping the X and Y axis. The algorithm iterates along the Y axis and for each element it iterates over the X axis. For the summation of the values the weights of the corresponding elements are considered.

  The implementation provides two methods, one for the beamforming in 2D and one for 3D.

  - sampleBeamform2D(): Same function as sampleBeamform2D() in the class RxSampleBeamformerDelayAndSum.
  - sampleBeamform3D(): This method performs the Delay and Sum beamforming algorithm for 3D. Therefore, it call the function sampleBeamform3DYX() with the corresponding arguments.
  - sampleBeamform3DYX(): The algorithm is designed to iterate over the Y axis. During this iteration, the values along the X axis are summed up (multiplied with the corresponding weights). This method is called by sampleBeamform3D().

- RxSampleBeamformerDelayAndSumXYYX.h:

  By using this implementation, the algorithm unifies both algorithms from the two beamformers above. Again, for the summation of the values the weights of the corresponding elements are considered.

  The implementation provides two methods, one for the beamforming in 2D and one for 3D.

  - sampleBeamform2D(): Same function as sampleBeamform2D() in the class RxSampleBeamformerDelayAndSum.
  - sampleBeamform3D(): This method performs the Delay and Sum beamforming algorithm for 3D. Therefore, it call the function sampleBeamform3DXY() with the corresponding arguments, as well as the function sampleBeamform3DYX() with the corresponding arguments. The results are added and returned.
  - sampleBeamform3DXY(): The algorithm is designed to iterate over the X axis. During this iteration, the values along the Y axis are summed up (multiplied with the corresponding weights).
  - sampleBeamform3DYX(): The algorithm is designed to iterate over the Y axis. During this iteration, the values along the X axis are summed up (multiplied with the corresponding weights).

- RxSampleBeamformerDelayAndSumXYYXDivided.h:

  By using this implementation, the algorithm unifies the both algorithms from the two beamformers above and halves the result before returning. Again, for the summation of the values the weights of the corresponding elements are considered.

  The implementation provides two methods, one for the beamforming in 2D and one for 3D.

  - sampleBeamform2D(): Same function as sampleBeamform2D() in the class RxSampleBeamformerDelayAndSum.
  - sampleBeamform3D(): This method performs the Delay and Sum beamforming algorithm for 3D. Therefore, it call the function sampleBeamform3DXY() with the corresponding arguments, as well as the function sampleBeamform3DYX() with the corresponding arguments. The results are added, divided by two and returned.
  - sampleBeamform3DXY(): The algorithm is designed to iterate over the X axis. During this iteration, the values along the Y axis are summed up (multiplied with the corresponding weights).
  - sampleBeamform3DYX(): The algorithm is designed to iterate over the Y axis. During this iteration, the values along the X axis are summed up (multiplied with the corresponding weights).

3.2 Delay Multiply and Sum

For the implementation of the Delay Multiply and Sum beamformer (according to [2]) refer to the files in src/SupraLib/Beamformer starting with RxSampleBeamformerDelay-MultiplyAndSum.

- RxSampleBeamformerDelayMultiplyAndSum.h:

  The implementation provides two methods, one for the beamforming in 2D and one for 3D.

  - sampleBeamform2D(): Implementation of the 2D DMAS beamformer considering the weights of the elements, as well as the normalization of the result with the number of accumulated elements and multiplied with the accumulated weights of the elements. For performing the beamforming, the algorithm iterates over the receiving window choosing one index as the active element. For this active element each of the remaining elements on its right side are multiplied with the value of the active element and the corresponding weights and then summed up. For the accumulated weight, the product of both the weight of the actual element and the element, which is iterated over, is summed up. In general, the method refers to the elements of the element, being iterated over, with the name affix 'Shift'.
  - sampleBeamform3D(): Implementation of the 3D DMAS beamformer, adapted from the 2D DMAS beamformer, considering the weights of the elements, as well as the normalization of the result with the number of accumulated elements and multiplied with the accumulated weights of the elements. The algorithm is designed to iterate over the receiving window. Therefore, the algorithm picks an active element for which an subwindow is chosen, starting with at the next column and next row of the receiving window. Again, the value of active element, the element, which is iterated over and the corresponding weights are multiplies and summed. In general, the method refers to the elements of the element, being iterated over, with the name affix 'Shift'.

- RxSampleBeamformerDelayMultiplyAndSum2.h:

  The implementation provides just one usable method, the one for the beamforming in 2D.

  - sampleBeamform2D(): Implementation of the 2D DMAS beamformer considering the weights of the elements, but without the normalization of the result with the number of accumulated elements and multiplied with the accumulated weights of the elements. For performing the beamforming, the algorithm iterates over the receiving window choosing one index as the active element. For this active element each of the remaining elements on its right side are multiplied with the value of the active element and the corresponding weights and then summed up. For the accumulated weight, the product of both the weight of the actual element and the element, which is iterated over, is summed up. In general, the method refers to the elements of the element, being iterated over, with the name affix 'Shift'.

  - sampleBeamform3D(): Not implemented - use RxSampleBeamformerDelayMultiplyAndSum.

- RxSampleBeamformerDelayMultiplyAndSum3.h:

  The implementation provides just one usable method, the one for the beamforming in 2D.

  - sampleBeamform2D(): Implementation of the 2D DMAS beamformer considering the weights of the elements, as well as the normalization of the result with the number of accumulated elements and multiplied with the accumulated weights of the elements. For performing the beamforming, the algorithm iterates over the receiving window choosing one index as the active element. For this active element each of the remaining elements on its right side are multiplied with the value of the active element and then summed up. For the accumulated weight, the product of both the weight of the actual element and the element, which is iterated over, is summed up. The difference is, that for the summation of the multiplied values the weights are not considered. In general, the method refers to the elements of the element, being iterated over, with the name affix 'Shift'.
  - sampleBeamform3D(): Not implemented - use RxSampleBeamformerDelayMultiplyAndSum.

- RxSampleBeamformerDelayMultiplyAndSum4.h:

  The implementation provides just one usable method, the one for the beamforming in 2D.

  - sampleBeamform2D(): Implementation of the 2D DMAS beamformer. For performing the beamforming, the algorithm iterates over the receiving window choosing one index as the active element. For this active element each of the remaining elements on its right side are multiplied with the value of the active element and then summed up. In this algorithm, neither weights are considered, nor normalization is performed. In general, the method refers to the elements of the element, being iterated over, with the name affix 'Shift'.
  - sampleBeamform3D(): Not implemented - use RxSampleBeamformerDelayMultiplyAndSum.

3.3 Signed Delay Multiply and Sum

For the implementation of the Delay Multiply and Sum beamformer (according to [2]) refer to the header file src/SupraLib/Beamformer/RxSampleBeamformerSignedDelayMultiplyAndSum.h

- RxSampleBeamformerSignedDelayMultiplyAndSum.h:

  The implementation provides just one usable method, the one for the beamforming in 2D.

- sampleBeamform2D(): Implementation of the 2D SDMAS beamformer. For performing the beamforming, the algorithm returns the result of the Delay Multiply and Sum beamformer with the same sign as the result of the Delay and Sum beamformer would have.
- sampleBeamform3D(): Not implemented - use RxSampleBeamformerDelayMultiplyAndSum.

References

[1] Rüdiger Göbl, Nassir Navab, and Christoph Hennersperger. Supra: open-source software-defined ultrasound processing for real-time applications. *International Journal of Computer Assisted Radiology and Surgery*, Mar 2018.

[2] Thomas Kirchner, Franz Sattler, Janek Gröhl, and Lena Maier-Hein. Signed real-time delay multiply and sum beamforming for multispectral photoacoustic imaging. *Journal of Imaging*, 4, Oct 2018.