Vision, Modeling, and Visualization (2011)
Peter Eisert, Konrad Polthier, and Joachim Hornegger (Eds.)

RITK: The Range Imaging Toolkit – A Framework for 3-D Range Image Stream Processing

J. Wasza[1], S. Bauer[1], S. Haase[1], M. Schmid[2], S. Reichert[4], J. Hornegger[1,3]

[1] Pattern Recognition Lab, Dept. of Computer Science
[2] Hardware/Software Co-Design, Dept. of Computer Science
[3] Erlangen Graduate School in Advanced Optical Technologies (SAOT)
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
[4] softgate GmbH, Erlangen, Germany

Abstract

The recent introduction of low-cost devices for real-time acquisition of dense 3-D range imaging (RI) streams has attracted a great deal of attention. However, to date, there exists no open source framework that is explicitly dedicated to real-time processing of RI streams. In this paper, we present the Range Imaging Toolkit (RITK). The goal is to provide a powerful yet intuitive software platform that facilitates the development of range image stream applications. RITK puts emphasis on real-time processing of range image streams and proposes the use of a dedicated pipeline mechanism. Furthermore, we introduce a powerful and convenient interface for range image processing on the graphics processing unit (GPU). Being designed thoroughly and in a generic manner, the toolkit is able to cope with the broad diversity of data streams provided by available RI devices and can easily be extended by custom range imaging sensors or processing modules. RITK is an open source project and will be made publicly available at `http://www5.cs.fau.de/ritk`.

Categories and Subject Descriptors (according to ACM CCS): I.4.9 [Image Processing and Computer Vision]: Applications—D.2.13 [Software Engineering]: Reusable libraries—

1. Introduction

In the past, the acquisition of dense three-dimensional range imaging (RI) data was both tedious, time consuming and expensive, hence, hindering a widespread application. Lately, technological advances in RI sensor design have rendered metric 3-D surface acquisition at high resolutions (up to 300k points) and real-time frame rates (up to 40 Hz) possible. The advent of Microsoft's Kinect, with a mass market retail price of $150 a unit and more than 10 million sales within a few months, has caused a furor in the field of consumer electronics. With the introduction of affordable hardware, 3-D perception is gaining popularity and importance across a wide range of domains. In particular, we note an explosion of innovation in applications that benefit from the fact that state-of-the-art RI sensors deliver dense and dynamic range data streams in real-time. Among others, real-time capable RI sensors hold potential for human computer interaction, augmented reality, surveillance and security applications, biometrics [BWMH11] , medical engineering [MBWH11, BBHR11], or automotive and industrial applications [RMBD08]. Among range imaging sensors, structured light and Time-of-Flight (ToF) based devices are of particular interest and popularity. Unlike conventional RI approaches such as passive stereo vision, the recovery of depth from texture-less regions or repetitive patterns is not an issue with these novel modalities. Regardless of fundamentally different physical principles [GZ08, KBKL09], both technologies are capable of delivering dense and metric 3-D surface information at real-time frame rates. However, the acquisition and streaming of RI data typically implies an immense amount of data (in the scale of 500 MBit/s) to be propagated and processed. This poses a challenge in terms of throughput for subsequent data processing and analysis algorithms. Despite the advances in sensor technology, there exists no open source framework dedicated to real-time processing of RI streams.

1.1. Contributions

In this paper, we introduce the range imaging toolkit (RITK). It is a cross-platform and object-oriented toolkit explicitly dedicated to the processing of data streams from modern RI devices. The toolkit can support developers in two ways: First, it can be used as an independent library for RI stream processing within existing software. Second, it provides a comprehensive software platform for the development of range imaging solutions at an application level. In particular, we propose an easy-to-use interface infrastructure that allows the developer to incorporate individual processing modules. An individual selection of modules from the RITK pool of building blocks can then be assembled into an application-specific RI processing pipeline. This approach facilitates the distribution and reuse of existing modules and thus provides a powerful yet intuitive rapid prototyping environment for the creation of standalone applications. The toolkit is explicitly dedicated to real-time processing of high-bandwidth data streams. In particular, we present an user-friendly interface for general purpose computing on modern many-core graphics processing units (GPUs). The interface facilitates the usage of dedicated hardware with a collection of convenience methods and macros. Furthermore, RITK takes advantage of the interoperability of general purpose computing on the GPU and rendering for real-time visualization of dynamic 3-D point cloud data.

Along with this publication, RITK will be released as open source software, providing a hands-on range image stream processing framework for the fast growing RI community. The release will include a collection of sample modules for acquisition, pre-processing, analysis and visualization of range image streams.

The paper is organized as follows. The remainder of this section compares the toolkit to related work. Section 2 gives an overview of the framework's architecture and underlying design principles. Subsequently, in Section 3, the key concepts of RITK and implementation issues are described in more detail. Section 4 gives examples of applications using RITK and illustrates how the concepts described before are used therein. In Section 5, we draw conclusions and give a brief outline of the future development plans for the presented platform.

1.2. Related Work

In the computer vision community, several open source software libraries for general purpose 2-D and 3-D image processing exist today [ISNC05, SML06, Bra00, RC11]. Most of these libraries provide some basic functionality for the processing of static 3-D point clouds or surfaces. However, the state-of-the-art lacks a library that explicitly addresses real-time processing of 3-D range image or point cloud streams.

To date, few platforms support ToF sensors [ART11, BIA11] and/or Microsoft's Kinect device [BIA11, SLR*11, RC11]. In terms of ToF imaging, we are aware of three libraries: The *action recognition and tracking based on time-of-flight sensors (ARTTS) toolbox* [ART11] is dedicated to ToF range data processing and divides into packages for general signal processing, object tracking and action recognition. However, the toolbox is based on MATLAB. The *medical imaging interaction toolkit (MITK)* [SYM*11] is a platform that exclusively addresses developers in the field of interactive medical image processing. Basically, it combines two established libraries for image processing and visualization (ITK, VTK, see below). Since ToF imaging holds potential for medical applications [MBWH11, SPH08], a submodule for acquisition and processing of ToF range data was added recently but functionality is limited to basic I/O and CPU-based filtering yet. The *basic image algorithms (BIAS)* library [BIA11] provides a code basis for image processing, motion estimation and 3-D reconstruction algorithms. It supports both ToF devices and Microsoft's Kinect, but the library is not explicitly dedicated to range image processing. The *flexible action and articulated skeleton toolkit (FAAST)* [SLR*11] supports the Kinect device, but is intended and designed in particular for full-body control in gaming and virtual reality applications. Hence, let us conclude that both MITK and FAAST are not explicitly dedicated to range image and point cloud processing, respectively.

Only recently, the *Point Cloud Library (PCL)* [RC11] was introduced. To our knowledge, it is the first library that first and foremost addresses the processing and interpretation of 3-D point clouds. However, in contrast to RITK, the library's focus is on providing a common repository of algorithms. Furthermore, and similar to the libraries presented before, PCL is designed for processing static range image or point cloud data, respectively. This contradicts the fact that modern RI devices stand out due to its capability of dynamic 3-D scene acquisition at real-time frame rates. These issues are addressed by the toolkit presented in this paper, being dedicated to the processing of a sequence or stream of range images on-the-fly and in real-time. These aspects point out that RITK and PCL are no competitors but can complement each other.

Being of particular relevance w.r.t. the design of RITK, below, let us briefly depict one of the most established and maintained open source libraries in the scientific computer vision community. The *Insight Segmentation and Registration Toolkit* (ITK) [ISNC05] is a powerful open source library of algorithms for 2-D and 3-D image processing, especially for segmentation and registration. In particular, in the field of medical image processing, it is regarded as the de-facto standard. For RITK, we take advantage of ITK's established programming paradigms for image processing. For instance, we adopt the powerful pipeline- and update-mechanism for the processing of RI data streams (see Sec. 3.2). As a consequence, developers being familiar with ITK will rapidly get accustomed to the RITK implementation scheme.

2. Architecture and Design Principles

RITK is an object-oriented, cross-platform toolkit written in C++, that can be used as a regular library as well as an application framework for rapid development. The application layer of RITK is designed with a focus on dynamically loadable plugins, facilitating the distribution and reuse of existing modules. Besides its rapid prototyping capability, this design principle eventually eases the assembly of standalone RI solutions.

The basic design strategy pursued during development was to build the toolkit upon existing and established libraries, extending their functionality to fit the needs for real-time range image streaming, and combine the libraries' individual advantages and unique characteristics to form a user-friendly framework, yet satisfying scientific and industrial demands. In particular, RITK is written with performance in mind, supporting modern multi-core CPU and many-core GPU architectures, as well as enabling the integration of dedicated hardware such as FPGAs.

2.1. Conceptual Modules

Functionality in RITK is spread across different modules. This ensures a high degree of cohesion and loose coupling, eventually providing the basis for the usage of RITK as a regular library as well as a standalone application.

- **ritkCommon**: This module defines the basic range imaging data and processing types and implements an ITK-style pipelining concept specially tailored to the needs of real-time range image streaming. In addition, the interface for seamless integration of dedicated hardware such as GPUs into ITK is provided.
- **ritkCore**: The core engine of RITK on the application layer is provided in this module. In particular, this includes a dynamically configurable range image acquisition pipeline, plugin management, event handling and broadcasting, as well as a graphical user interface (GUI) front-end to the aforementioned tasks. Here, the well established Qt framework [BS08] is used for both the GUI and as backbone for the modular plugin system and multi-threading on the application layer.
- **ritkVisualization**: Methods for the graphical representation of range data are implemented in this module. RITK provides a high-performant visualization system that is specifically designed for generic range image types. The visualization system is based on OpenGL and the OpenGL Shading Language (GLSL).
- **ritkVTK**: This optional module contains a convenient interface to the well-established *Visualization Toolkit* [SML06]. Usage of VTK is mainly confined to rendering and interaction, however, the module also enables utilization of VTK's rich set of mesh manipulation routines.
- **ritkCUDA**: This optional module contains the interface for general purpose computing on the GPU using Nvidia's CUDA [NVI10] architecture. The main task of this module is to facilitate CUDA usage and to provide the infrastructure for high-performance GPU based image processing using an ITK-style pipeline.

The conceptual modules of RITK and their dependencies are illustrated in Fig. 1, see Sec. 3 for a detailed description of functionality and concepts in RITK.

2.2. Application Layer and Plugin Architecture

The application layer of RITK is designed with a strong focus on modularity by dynamically loadable plugins/modules. The philosophy behind this approach is that RITK can easily be extended by custom RI sensors or data types and processing modules to assemble generic processing pipelines at run-time. This obviates the need to recompile the whole framework when adding new features. Due to the standardized, yet generic and modality independent range image format in RITK, compatibility and reusability of existing modules with custom extensions is granted. Going one step further, the plugin system of RITK is explicitly designed for dynamically loadable custom modules that run inside the application layer of RITK. This allows for rapid development of small sandbox tools as well as comprehensive standalone RI solutions that exploit RITK's generic custom sensor concept and real-time streaming oriented data acquisition and processing infrastructure. For this purpose, RITK provides an easy to use plugin creator and employs an XML-based key-and-value configuration system that eases automatic assembly and parameterization of applications at run-time.

The plugin concept in RITK is schematically illustrated in Fig. 2. Besides the aforementioned generic acquisition concept and run-time configurability, we point out RITK's broadcast mechanism and support for concurrently executed modules, each running in its own individual thread. Thus, reusability and integration of existing application modules is also given, which is beneficial for several comprehensive

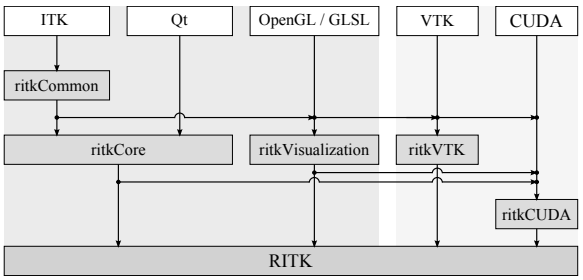

Figure 1: *The RITK modules. RITK is mainly based on ITK, the Qt framework, and OpenGL / GLSL for visualization. The VTK front-end and GPU support using CUDA is optional, yet included in the open source release.*

RI solutions (see the example in Sec. 4.2). Being important in industry driven cooperations and commercial research projects, such a plugin system comes with the advantage that no source code has to be provided. By providing pre-built binaries equipped with a license of limited duration, reproducible research or module testing is granted while retaining full control with regard to copyright issues or unauthorized disclosure and usage.

3. Implementation Details

In this chapter, we review the most important implementation details of RITK. In particular, we rediscover ITK's pipelining concept - originally developed for static medical image processing - for real-time image streaming and GPU support.

3.1. Range Image Format

The plurality of existing RI sensors (e.g. structured light, ToF) and its different or incompatible data structures implies the need for a standardized internal data format being flexible for future devices. Therefore, RITK comes with its own range image base class that is derived from `itk::ImageBase`, enabling full usage of ITK's smart-pointer, memory management and pipelining concept. By design, this also enables multi-threaded image processing to exploit the full potential of modern multi-core CPU architectures.

Range images contain a unique timestamp for detailed analysis and proper chronological reconstruction of recorded scenes. Furthermore, ITK's origin, orientation, and spacing concept can be used to provide multiple viewpoint functionality and for composite RI sensors and multi-camera setups.

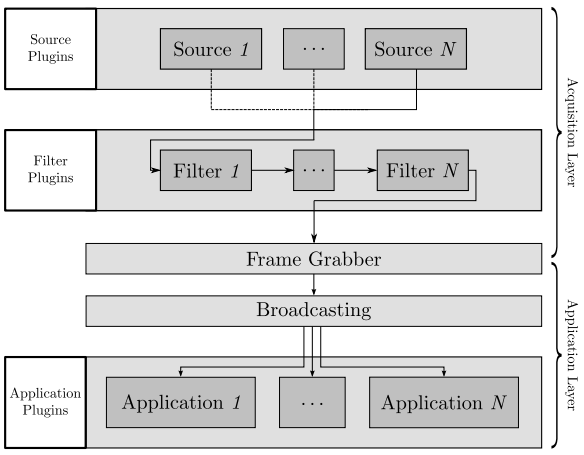

Figure 2: *The RITK plugin principle. RITK is explicitly designed for dynamically loadable and interchangeable modules, both on the acquisition and application layer.*

Besides the pure range information, RI sensors commonly provide additional data such as photometric or validity information. Thus, the actual data payload of an range image may consist of several values per pixel that can internally be stored as individual ITK images. This concept allows to take advantage of ITK's rich set of existing image processing functionalities and eases development of novel algorithms using state-of-the-art programming paradigms. In order to enable the integration of non-image data, RITK employs generic payload containers that can hold arbitrary data such as body landmarks (see Sec. 4.1). Given the object oriented approach in RITK, future sensors and additional payload can thus be easily integrated.

3.2. Processing Pipeline

Consequently, pipeline propagation of range images coincides with ITK, i.e. an RI source derived from `itk::ImageSource` is located at the pipeline's head and several RI filters derived from `itk::ImageToImageFilter` are connected downstream. In this context, RI filters can be seen as composite ITK filters or mini-pipelines that encapsulate processing of a range image's generic payload. Naturally, this allows to use the same pipeline with different sensors and image types. However, by default, ITK does not support pipeline propagation of custom image types that are unknown at compile-time. Regarding RITK's design philosophy to extend the framework by custom modules and to assemble application modules at run-time, this is a critical restriction. Therefore, the base class for RI filtering is specifically designed to handle this task in a fully automatic manner, by dynamically matching a filter's output type to fit its input type and to propagate this type information to all downstream elements.

A key concept pursued for data processing in RITK is the interface for so-called generic pixel containers. These pixel containers are intended to replace regular `itk::ImportImageContainers` dynamically at run-time and enable image data to be stored and processed in regular RAM as well as on dedicated hardware such as GPUs (see Sec. 3.4). Through the usage of sophisticated, yet fully automatic synchronization mechanisms, development efforts are minimized and potentially time consuming memory transfers between devices can be prevented. This is a crucial prerequisite for real-time capability, especially for streaming large data. Addressing the problem of pixel storage at run-time has the advantage, that special hardware can be seamlessly integrated into an ITK pipeline and combined with existing RITK modules.

3.3. Visualization

The graphical representation of range images is a key issue for understanding and analysis of 3-D scenes as well as an important feedback when designing processing algorithms. RITK offers OpenGL-based visualization methods that are specially tailored to high-performance rendering of streaming range data. In particular, RITK's visualization module puts emphasis on rendering of 3-D point clouds and surface meshes computed from the range or depth information. The visualization of a range image's additional payload such as photometric information is done by utilizing the OpenGL Shading Language. This enables continuous and interactive blending of different generic payload components in one scene. In order to support the interoperability of general purpose computing on the GPU and rendering, RITK's visualization module employs the concept of vertex buffer objects.

In addition, RITK provides a direct interface and easy-to-use convenience modules for 3-D surface and point cloud rendering in VTK [SML06]. However, VTK is not designed for processing and visualization of real-time streaming data. Besides run-time issues due to the computational overhead of an additional abstraction layer, VTK natively lacks synchronization methods for critical sections. This is a crucial drawback, as for performance reasons, data acquisition and rendering in RITK is usually performed in coherent threads. Thus, the visualization module of RITK extends VTK functionality to synchronize data while streaming, eventually preventing race conditions and invalid data states.

3.4. GPU Interface

RITK's GPU module contains a convenience interface for general purpose computing on the GPU using Nvidia's CUDA architecture. The main task of this module is to facilitate CUDA usage and to provide the infrastructure for high-performance GPU based image processing. This is done by providing concrete implementations of the generic pixel container concept (see Sec. 3.2) that is part of RITK's run-time configurable processing pipeline. In particular, these containers facilitate conversion between the different memory types in the CUDA architecture and automate host/device transfer in a way that minimizes copy overhead. Given their inheritance hierarchy, the CUDA containers naturally comply with resource (de-)allocation and storage in ITK. Thus, an ITK-style processing pipeline running entirely on the GPU as well as a heterogeneous pipeline using CPU and GPU processing elements is enabled.

As for CUDA versions < 4.0 memory allocation, access, and deallocation must be performed within the same thread context, convenience methods and macros to ease these tasks are provided.

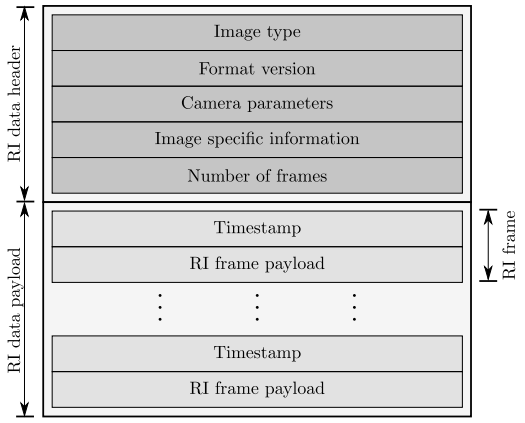

Figure 3: *The data format of a range image sequence (.ris) file, dividing into data header and payload. The image type is used to allocate dedicated I/O routines.*

3.5. File Format

There exist several standardized file formats (e.g. *.stl, .ply, .vtk*) for static point cloud and surface data, respectively. In contrast, there is not even one established file format for range image stream data. As a consequence, along with the introduction of RITK, we propose a new file format for dynamic range image stream data. In particular, RITK discriminates between a single static range image (*.ri*) and range image sequences (*.ris*). Both file formats divide into a compact header and the actual RI data payload, see Fig. 3.

The header holds the respective image type, the version of the file format (for backward compatibility), device-specific parameters w.r.t. camera calibration and 3-D reconstruction, image-specific information and the total number of frames in case of range image streams. In fact, the image type header field serves as an identifier for device-specific payload interpretation at runtime. Consequently, RITK is capable of allocating dedicated I/O routines for different RI sensors. Instead of using a separate parameter file, camera-specific parameters are stored as part of the header. This is motivated by the fact that the payload can be interpreted solely with the individual parameters of the actual sensor that was used for acquisition. The payload holds the RI data on a per-frame basis. For each single range image of a sequence, a unique timestamp is stored in addition to the captured sensor data. Hence, recorded scenes can be reconstructed, processed and analyzed in a chronologically accurate way. The type and size of the payload depends on the device-specific raw data. For instance, Microsoft's Kinect delivers orthogonal depth and the corresponding RGB data, whereas ToF sensors may provide either radial range and amplitude data or a sequence of measured phase images [KBKL09]. With respect to efficient I/O for range image sequences, data is stored as binary streams.

4. Applications and Examples

Due to its generic design, RITK can be used for a broad variety of real-time applications. Below, we briefly outline four practical examples that are based on and built with the presented toolkit. Thereby, we demonstrate the capabilities of RITK to be used as a software framework and rapid prototyping platform for assembling (standalone) range imaging solutions. Please note that not all examples and modules presented in the following sections will be included in the open source release of RITK.

4.1. Kinect Interface

The open source release of RITK comes with a ready-to-use interface for Microsoft's Kinect device using the OpenNI and NITE frameworks [Ope11]. The generic payload concept in RITK (Sec. 3.1) seamlessly enables the integration of Kinect's range information, RGB data and skeleton tracking functionality into RITK solutions. Fig. 4 shows a screenshot of a skeleton tracking application module that was implemented using RITK's plugin creator and VTK frontend.

4.2. Respiratory Motion

The management of respiratory motion is an important factor in fractionated radiotherapy (RT). The basic idea behind the application of real-time RI sensors in RT is that the patient's respiration curve can be acquired in a markerless and non-obtrusive manner. The reconstructed respiration curve can be employed in gated radiotherapy [SPH08], for instance. An RITK based implementation of this use-case is illustrated in Fig. 5. The example demonstrates RITK's support for the development of powerful standalone solutions. Here, we explicitly note the concept of application modules that run in parallel: in RT, a concurrently running recording module can be used for clinical documentation of the treatment fraction.

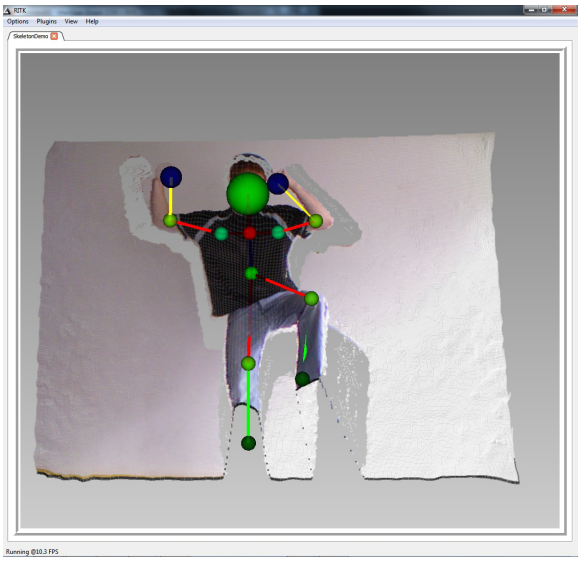

Figure 4: *Microsoft Kinect integration in RITK. Skeleton tracking is performed using the NITE framework. The estimated 3-D body landmark positions are propagated through the processing pipeline and to the skeleton tracking application module using RITK's generic payload concept.*

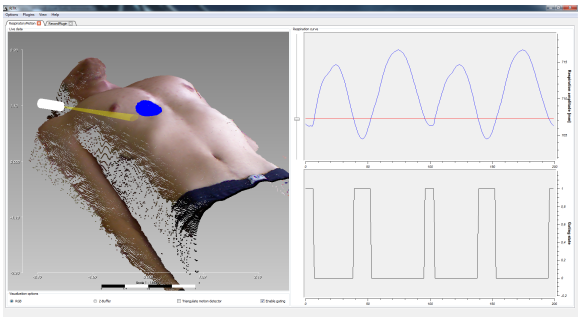

Figure 5: *Respiratory motion management in RITK using Microsoft's Kinect device. The patient's respiration curve can be accurately reconstructed. Note the concurrently running recording module that can be used for the purpose of clinical documentation.*

4.3. Range Image Stream Simulator

This example demonstrates the generic range image source concept pursued in RITK (Sec. 3.1), providing the fundamental basis to code modality independent algorithms that can be applied to data from different RI devices or synthetic sources. Here, we have implemented a range image stream simulator that produces range data based on the z-buffer representation of a 3-D scene. The scene can be individually composed. Using the simulator allows to experiment with different noise characteristics and sensor resolutions that occur with cameras from different manufacturers, while providing an absolute ground truth for evaluation purposes. Due to the modality independent processing pipeline in RITK, this inherently allows for benchmarking both accuracy and run-times of pre-processing filters in a reproducible manner [WBH11]. A screenshot of the simulator application is given in Fig. 6. In this example, the simulator is set up to produce synthetic range data that resembles ToF measurements including its unique noise and artifact characteristics [LSKK10]. The employed pre-processing pipeline was assembled from the pool of filter modules and effectively produces a reliable output surface while exploiting RITK's GPGPU convenience interface to achieve real-time frame rates. Besides benchmarking of pre-processing algorithms, this RI simulator can also be used to create virtual 3-D test environments. This, for instance, allows to evaluate segmen-

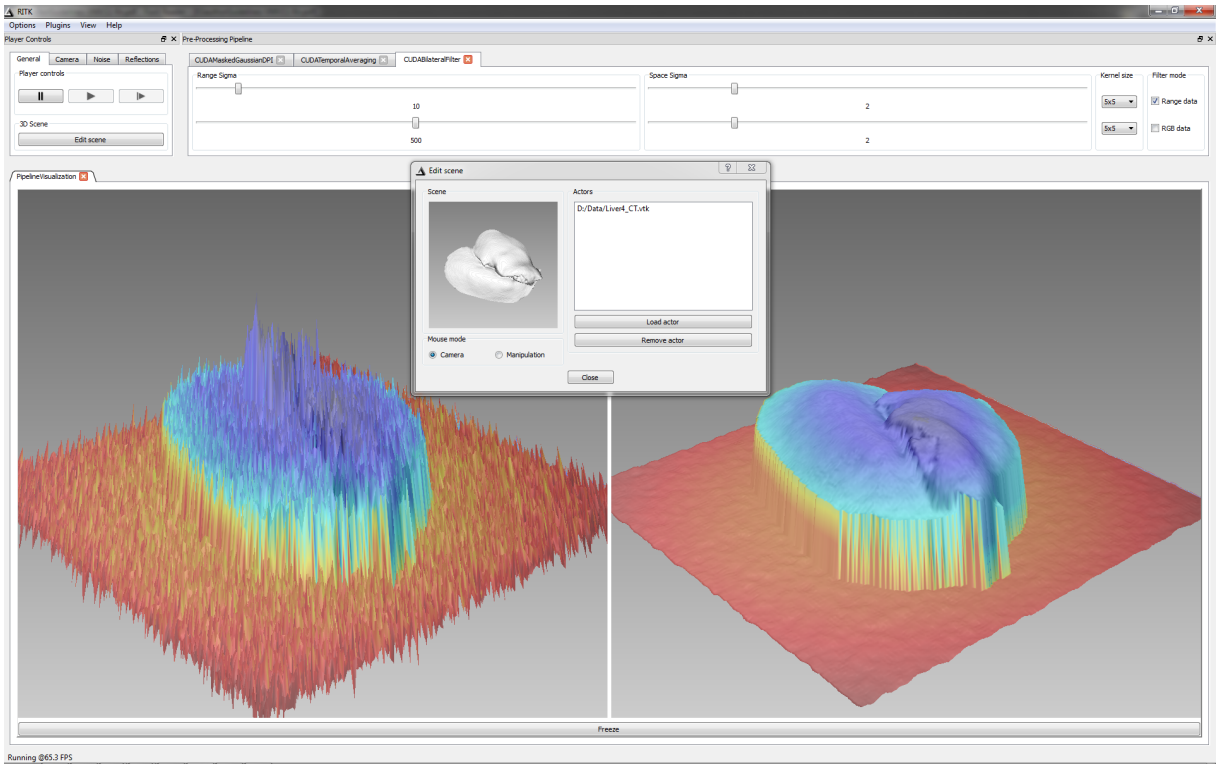

Figure 6: *Range image stream simulation for a liver mesh, used for benchmarking of pre-processing filters for real-time interventional range imaging. Left: Noisy and corrupted output of the simulator source module (synthetic sensor resolution: 200×200 px), providing the input for the pre-processing pipeline. Right: Surface data after pre-processing. The CPU-based implementation of the simulator takes 10 ms to generate one single output image. Using RITK's GPGPU interface, a pre-processing pipeline running entirely on the GPU can be implemented. For example, a pipeline that includes filters for normalized convolution, temporal averaging and edge-preserving bilateral denoising has a total run-time of ∼3 ms on an Nvidia Quadro FX 2800M GPU.*

tation and registration algorithms w.r.t. its robustness against occlusion or partial visibility. The basic RI simulator will be included in the open source release of RITK.

4.4. FPGA Filter Plugin

In this example, we demonstrate how external computing resources can be deployed within RITK. In particular, let us consider the integration of an FPGA for hardware acceleration of range image pre-processing and filtering. Using a Xilinx University Program Virtex-5-LX110T development board [Xil11], we have decided to connect the FPGA via Gigabit Ethernet to a host running RITK. This allows to employ the external board with different kind of host computers, including notebooks. However, this connection type is not mandatory, as, for example, PCI-Express might provide higher transfer rates. We use standard user datagram protocol (UDP) sockets to achieve low latency and minimum transmission overhead. On the FPGA, an open-source UDP/IP core optimized for direct PC-FPGA connectivity [ABS10] is employed in conjunction with a frame buffer. See Fig. 7 for an illustration. Concerning the RI data itself, the on-chip block memory (BRAM) might not be sufficient, as modern RI sensors feature resolutions of up to 640×480 pixels. Hence, the FPGA architecture distinguishes between image data stored in off-chip DDR RAM, and setup information kept in on-chip BRAM.

5. Conclusions

In this paper, we have presented an open source toolkit explicitly dedicated to real-time processing of 3-D data streams from modern RI devices. In addition to its usage as a range image stream processing library for existing software, RITK supports developers at an application level with a powerful development and rapid prototyping infrastructure for the creation of application-specific RI solutions. The toolkit can cope with the broad diversity of data streams provided by available RI devices and can be extended by custom sensor interfaces and processing modules. Due to its generic de-

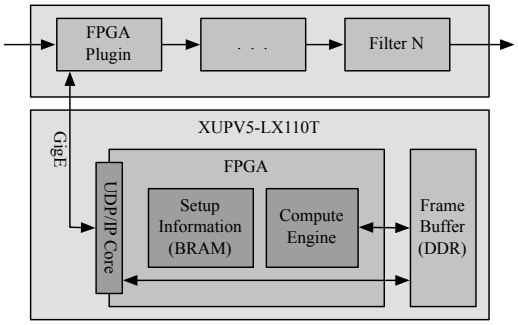

Figure 7: *Illustration of an FPGA integration into RITK's processing pipeline. In contrast to the GPU interface, communication ports for host/device data transfer are not provided as an RITK core module but within a filter plugin.*

sign, existing modules can be reused to assemble individual RI processing pipelines at run-time. With an emphasis on real-time capability, we have introduced a powerful pipeline concept and convenient interfaces for RI processing on modern multi-core CPUs and GPUs.

In our internal experience, RITK proved to greatly reduce the time required to develop range image stream applications. Hence, we feel confident that other researchers in the rapid growing community will also benefit from it. The similar structure should make it easy for developers already familiar with ITK to get accustomed to RITK.

The toolkit as presented is released under a free software license at `http://www5.cs.fau.de/ritk`.

Acknowledgments

J. Wasza, S. Bauer, M. Schmid and S. Reichert gratefully acknowledge the support by the European Regional Development Fund (ERDF) and the Bayerisches Staatsministerium für Wirtschaft, Infrastruktur, Verkehr und Technologie (StMWIVT), in the context of the R&D program IuK Bayern under Grant No. IUK338. S. Haase is supported by the Deutsche Forschungsgemeinschaft (DFG) under Grant No. HO 1791/7-1.

References

[ABS10] ALACHIOTIS N., BERGER S. A., STAMATAKIS A.: Efficient PC-FPGA Communication over Gigabit Ethernet. In *Proceedings of IEEE International Conference on Computer and Information Technology* (2010), pp. 1727–1734. 7

[ART11] ARTTS, The Action Recognition and Tracking based on Time-of-Flight Sensors (ARTTS) project, 2011. http://www.artts.eu/publications/artts_toolbox. 2

[BBHR11] BAUER S., BERKELS B., HORNEGGER J., RUMPF M.: Joint ToF Image Denoising and Registration with a CT Surface in Radiation Therapy. In *Proceedings of the International Conference on Scale Space and Variational Methods in Computer Vision* (2011). In press. 1

[BIA11] BIAS, The Basic Image AlgorithmS Library, http://www.mip.informatik.uni-kiel.de, 2011. 2

[Bra00] BRADSKI G.: The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000). 2

[BS08] BLANCHETTE J., SUMMERFIELD M.: *C++ GUI Programming with Qt 4*. Prentice Hall, 2008. 3

[BWMH11] BAUER S., WASZA J., MÜLLER K., HORNEGGER J.: 4D Photogeometric Face Recognition with Time-of-Flight Sensors. In *Proceedings of IEEE Workshop on Applications of Computer Vision* (2011), pp. 196–203. 1

[GZ08] GARCIA J., ZALEVSKY Z.: Range mapping using speckle decorrelation. US patent No.7433024, 2008. 1

[ISNC05] IBANEZ L., SCHROEDER W., NG L., CATES J.: *The ITK Software Guide*, second ed., 2005. 2

[KBKL09] KOLB A., BARTH E., KOCH R., LARSEN R.: Time-of-Flight Sensors in Computer Graphics. In *Eurographics - State of the Art Reports* (2009), Eurographics, pp. 119–134. 1, 5

[LSKK10] LINDNER M., SCHILLER I., KOLB A., KOCH R.: Time-of-Flight sensor calibration for accurate range sensing. *Computer Vision and Image Understanding 114*, 12 (2010), 1318 – 1328. Special issue on Time-of-Flight Camera Based Computer Vision. 6

[MBWH11] MÜLLER K., BAUER S., WASZA J., HORNEGGER J.: Automatic Multi-modal ToF/CT Organ Surface Registration. In *Proceedings of Bildverarbeitung für die Medizin* (2011), pp. 154–158. 1, 2

[NVI10] NVIDIA: *CUDA C Programming Guide 3.2*. NVIDIA, 2010. 3

[Ope11] OpenNI, Open Natural Interaction, 2011. http://www.openni.org. 6

[RC11] RUSU R. B., COUSINS S.: 3D is here: Point Cloud Library (PCL). In *Proceedings of IEEE International Conference on Robotics and Automation* (2011). 2

[RMBD08] RAPUS M., MUNDER S., BARATOFF G., DENZLER J.: Pedestrian recognition using combined low-resolution depth and intensity images. In *Proceedings of IEEE Intelligent Vehicles Symposium* (2008), pp. 632–636. 1

[SLR*11] SUMA E., LANGE B., RIZZO A., KRUM D., BOLAS M.: FAAST: The Flexible Action and Articulated Skeleton Toolkit. In *Proceedings of IEEE Virtual Reality* (2011), pp. 247–248. 2

[SML06] SCHROEDER W., MARTIN K., LORENSEN B.: *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics*. Kitware, Inc., 2006. 2, 3, 5

[SPH08] SCHALLER C., PENNE J., HORNEGGER J.: Time-of-Flight Sensor for Respiratory Motion Gating. *Medical Physics 35*, 7 (2008), 3090–3093. 2, 6

[SYM*11] SEITEL A., YUNG K., MERSMANN S., KILGUS T., GROCH A., DOS SANTOS T., FRANZ A., NOLDEN M., MEINZER H.-P., MAIER-HEIN L.: MITK-ToF - Range data within MITK. *International Journal of Computer Assisted Radiology and Surgery* (2011), 1–10. 2

[WBH11] WASZA J., BAUER S., HORNEGGER J.: High Performance GPU-based Preprocessing for Time-of-Flight Imaging in Medical Applications. In *Proceedings of Bildverarbeitung für die Medizin* (2011), pp. 324–328. 6

[Xil11] XILINX: *ML505/ML506/ML507 Evaluation Platform User Guide*, 2011. 7