



SUMMER SCHOOL REPORT 2017  
18 AUGUST - 7 SEPTEMBER

---


# Simulations of 2D pixel detectors at European XFEL

## Extending X-CSIT to SimEx

---

Author: Jan-Philipp Burchert  
Email: [j.burchert@stud.uni-goettingen.de](mailto:j.burchert@stud.uni-goettingen.de)  
University of Göttingen  
Supervisor: Dr. Carsten Fortmann-Grote  
Email: [carsten.grote@xfel.eu](mailto:carsten.grote@xfel.eu)  
Group: XFEL WP-84 (Scientific Instrument SPB/SFX)

DESY  
Notkestraße 85  
22607 Hamburg

 +49 40 8998-0  
 +49 40 8998-3282  
 [desyinfo@desy.de](mailto:desyinfo@desy.de)  
 <http://www.desy.de/>

European XFEL GmbH  
Holzkoppel 4  
22869 Schenefeld

 +49-40-8998-6006  
 +49-40-8994-1905  
 [contact@xfel.eu](mailto:contact@xfel.eu)  
 <https://www.xfel.eu/>

# Contents

<b>1</b>	<b>Motivation</b>	<b>1</b>
<b>2</b>	<b>Background and Theory</b>	<b>1</b>
2.1	X-ray Free Electron Lasers . . . . .	1
2.2	Detector Effects on Signal . . . . .	3
2.3	Basics of Object Oriented Programming Languages . . . . .	4
<b>3</b>	<b>Existing Software</b>	<b>5</b>
3.1	Geant4/ X-CSIT . . . . .	5
3.2	SimEx . . . . .	6
3.3	Extending C++ to Python . . . . .	7
<b>4</b>	<b>Design</b>	<b>7</b>
4.1	C++ classes . . . . .	9
4.2	Python calculators . . . . .	10
4.3	Cmake . . . . .	11
<b>5</b>	<b>Application</b>	<b>12</b>
<b>6</b>	<b>Remaining Issues</b>	<b>14</b>
<b>7</b>	<b>Conclusion</b>	<b>14</b>
<b>8</b>	<b>Acknowledgements</b>	<b>15</b>
	<b>References</b>	<b>16</b>

## **Abstract**

The following report describes design and implementation of a detector simulation software, based on the existing software packages Geant4 and X-CSIT. X-CSIT describes the photon-matter interaction in the active layer of the detector and the propagation of created charges to the detector readout electronics. We make use of principles of object oriented programming to expose the X-CSIT library in the start-to-end simulation platform SimEx. C++ classes are compiled and linked to a shared object and made available in python by virtue of boost.python. Furthermore, cmake configuration and build instruction sets facilitate compiling this project. A test calculation for the AGIPD detector at the SPB-SFX instrument at European XFEL concludes the project.

**Keywords:** Geant4, X-CSIT, SimEx, detector simulation, single particle imaging, boost python, cmake, C++, python

# 1 Motivation

The European X-ray Free Electron Laser (XFEL) facility at Schenefeld is currently providing the brightest source of X-ray radiation with the highest pulse rate worldwide [Ebe17a]. For this purposes, it uses a large superconducting linear accelerator. However, like in every research facility, the quality of the X-ray data suffers from imperfections of its components such as optical elements, radiation damage or the detector response to incoming radiation. To improve experimental outcome the influence of these disturbances and noise need to be understood and minimized. For this purpose, simulations of the experiments are performed that allow to estimate this influence on the data both quantitatively and qualitatively.

For this reason, the software package `SimEx` ([https://github.com/eucall-software/simex\\_platform](https://github.com/eucall-software/simex_platform)) is developed which enables start-to-end simulations of the experiments. This project, `py_detector_interface`, the simulation of a photon detector is assigned to `SimEx`. The aim of `SimEx` is to simulate components of a X-ray facility as well as the entire facility to investigate the error propagation. So far the detector implementation has only implemented POISSON statistics but detectors usually have more fluctuations and effects to take into account. This project aims to integrate these additional features into the detector simulation of `SimEx`.

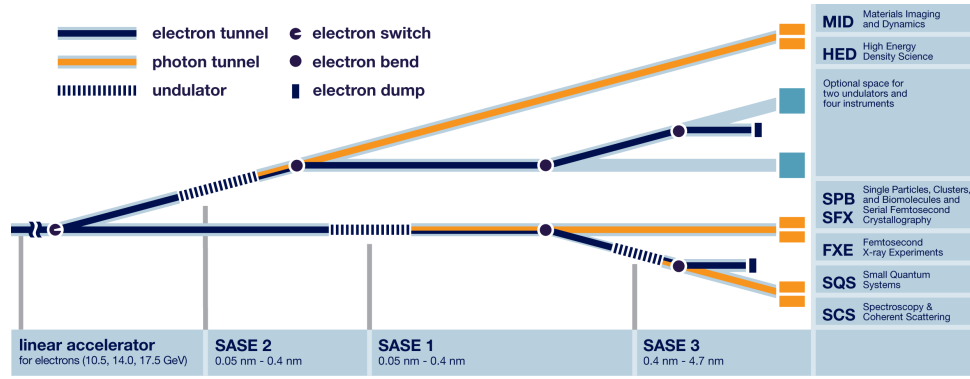
For this purpose, the software packages `Geant4` (<http://geant4.cern.ch/>) and `X-CSIT` (<https://git.xfel.eu/gitlab/karaboDevices/xcsit>) are utilized that have already been successfully implemented in the data analysis and control framework of the European XFEL `karabo` (<https://git.xfel.eu/gitlab/Karabo/Framework>). In contrast to `karabo`, where all the simulation is performed with devices which are integrated into the framework processing pipe, `SimEx` is a collection of classes. These so-called calculators are except for the input and output data independent of each other but all dependent on an `SimEx` abstract interface defining their layout and functionality. Each calculator simulates a certain component of the beam path. Consequently, installation as well as maintenance and usage of new components are more convenient in `SimEx`.

Another difference to `karabo` and `X-CSIT` is the programming language used: In contrast to `karabo` and `X-CSIT` which are written in C++, `SimEx` is written in python. Since `X-CSIT` is the basis of the simulation also in this project, the interface defined needs to be made accessible from python. To integrate it into `SimEx` an additional calculator needs to be written that utilizes the extended functions from `X-CSIT`. To achieve this, an interface between C++ written `X-CSIT` and python written `SimEx` source code is designed and implemented. This includes writing source code in C++ and python as well as creating a build procedure with `cmake`. Furthermore, an appropriate documentation and similar coding style like the one used for other `SimEx` calculators is required.

## 2 Background and Theory

### 2.1 X-ray Free Electron Lasers

XFELs produce high intensity X-ray pulses featuring peak brilliances of approximately  $10^{33}$  photons/s/mm<sup>2</sup>/mrad<sup>2</sup>/0.1%BW and short pulses of less than 100 fs duration (European XFEL according to [Gra09], [Ebe17a]). The peak brilliance is approx. nine orders of magnitude higher and the pulse length approx. three orders shorter than those created



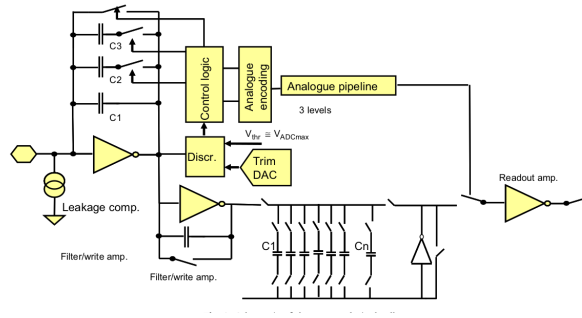
**Figure 1:** Schema of the European XFEL beamlines with undulators, beamlines and experimental stations as well as their research topic. Source [Ebe17b].

from synchrotron radiation [Gra09]. Such light sources have multiple components. The electron gun produces 27000 electron bunches per second [Ebe17a]. These are injected into a linear accelerator which increases the speed of those electrons very close to the speed of light [SDR08, S. 121]. To accelerate the European XFEL uses superconducting niobium cavities [Ebe17a].

After being accelerated the electron bunches are injected into undulators, where alternating strong magnetic fields urge the bunches to radiate their energy as X-rays. This process is highly non-linear [SDR08, p. 70] and is called **Self-Amplified Spontaneous Emission** (SASE). In principle, the electron bunch and the X-ray light interact leading to an exponential gain in the light intensity (for details see ref. [SDR08, p. 103ff]). However, the first photons in the undulators are emitted randomly spontaneous. For this reason, no shot is equal to another one which requires a lot of statistics when performing experiments and analysis [SDR08]. However, the light produced has the same properties than laser light making it usable for diffraction experiments [SDR08, p. 8]: nearly monochromatic, polarized, extremely bright, tightly collimated and highly spatially coherent.

The currently brightest light source of this type is the European XFEL (Schenefeld, Germany). It possesses a large superconducting accelerator (2,1 km) and produces a bright X-ray beam at a very high frequency [Ebe17b]. Electrons are accelerated to 17,5 GeV before entering the undulators where they radiate X-rays with wavelength down to 0.1 nm. There are five beam lines planned each providing two experimental stations. Currently, the experimental stations SPB-SFX and FXE (see figure 1) are completely equipped [Ebe17b]. They use hard X-rays for their experiments. However, when all planned experimental stations are built there will be also two stations SQS and SCS operating with soft X-rays. Additional information can be found in reference Altarelli [Alt11].

Experiments performed at SPB-SFX deal with single particle imaging and imaging of clusters. If high temporal resolution is required, experiments can also be performed at FXE. XFELs are well suited for resolving structures with single particle imaging methods e.g. of biomolecules which cannot be crystallized [FGBJ<sup>+</sup>17]. Because of their high photon flux and ultra-short pulses, images of the sample can be taken, before the sample suffers too much from radiation damage. The time molecule can withstand X-ray photons of 5 keV is according to Fortmann-Grote et al.  $\mathcal{O}(1)$ fs [FGBJ<sup>+</sup>17]. Furthermore, due to



**Figure 2:** Application Specific Integrated Circuit of a single pixel of the AGIPD detector with storage pipeline. Source: [HBD<sup>+</sup>11].

the high intensity even weakly scattering materials such as biological material can produce a sufficient signal [FGBJ<sup>+</sup>17].

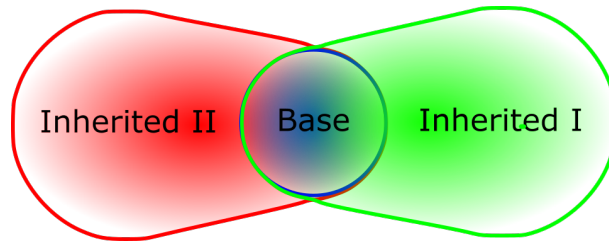
## 2.2 Detector Effects on Signal

Detector noise results from inevitable fluctuations and imperfections of the detector components [PG11]. However, it can be reduced. Understanding the performance of the detectors with respect to realistic data via simulation of the physical processes allows us to reduce the noise to the level required for a clear signal in the recorded experimental data. Each optimization is a trade off between noise reduction and performance. Every action to reduce noise will downgrade other detector parameters such as speed [PG11]. On the basis of the **Adaptive Gain Integrating Pixel Detector** (AGIPD) of the SPB beamline of XFEL, many noise sources can be identified.

Each detector possesses electronic components. These range from simple transistors to complex **Application Specific Integrated Circuits** (ASIC) [PG11] [SDH<sup>+</sup>10]. Due to imperfections in the material those circuits bear a natural source of fluctuations. Especially, the influence of leakage currents on operation amplifiers that increase with radiation intensity can result in increased noise levels [SDH<sup>+</sup>10]. Despite that, the resulting noise is often still smaller than e.g. the statistical fluctuations in of the photon number in each pixel [SDH<sup>+</sup>10].

Furthermore, leakage currents from another origin are much more serious. Since the XFEL source produces more images than can be processed in one of its 600  $\mu$ s pulse trains, the detector needs to store those image values previous to processing in a pipeline for each pixel (see figure 2). After each of those pulse trains there is a gap of approx. 100 ms, where the detector reads the pixel values of all stored images, digitizes them and store them on a hard disk. This pipelines consist of capacities and switches that suffer from leakage currents, as well. This not only affect the output values of intensity but also the noise level of the readout and digits [SDH<sup>+</sup>10].

Detectors at European XFEL have to cope with a very complex parameter space. They have to cover a broad range of energy of many keV for every pixel independently of each other. Furthermore, the pulse rate of the recording is quite high. Additionally, they need to be sensible enough to detect single photons but robust enough not to be destroyed at high intensities. This requires protection of the electronics leading to additional readout noise [PG11]. Despite optimization detectors have a finite dynamic range of values, which once exceeded, lead to noise in the shape of cut-off signals [PG11].



**Figure 3:** Sketch of polymorphism and inheritance. The coloured area can be understood as the amount of functions, methods and attributes defined in the class.

Additionally to the noise from electronic components and digitalization, there are also physical effects producing noise. Thermal fluctuations can be reduced by cooling. However, high intensity radiation can create plasmas in detector pixels that effect neighbouring pixels due to electron drift [PG11]. The process is called **blooming** and is taken into account in the charge simulation of this detector simulation project. Additional information can be found in references: Joy et al. [JWH<sup>+</sup>15], Rüter et al. [RHK<sup>+</sup>15], Shi et al. [SDH<sup>+</sup>10] and Potdevin et al. [PG11].

Last but not least, the number of photons arriving at a pixel and the number of charges created from an interacting photon are statistical values. They are uncertain and follow the Poisson statistics. Together with the blooming this makes up the most important source for noise. Nevertheless, in many experiments the noise resulting from a well optimized detector is much less than the fluctuations in the signal resulting e.g. from optical elements[PG11].

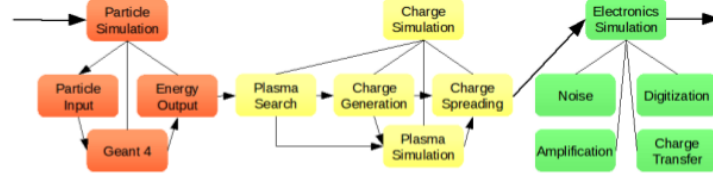
## 2.3 Basics of Object Oriented Programming Languages

In object oriented programming languages related source code is gathered in classes defining an abstract data type. From those classes instances can be created and linked to variables of the same type of the class. For this reason, in object oriented languages previous to usage of a variable the type of this variable needs to be declared and an instance of this type needs to be connected to the variable.

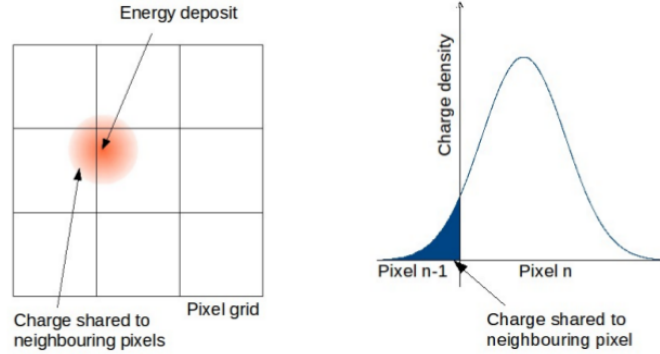
This structure allows special programming techniques. The type of a variable specifies the functions, methods and attributes that can be called upon this variable. For this reason, also instances of classes derived from that type can be linked to the variable. Since deriving from a class means inheriting its functions and attributes as well as adding additional ones, all the functions that can be called on the base instance can also be called on the child instance.

More generally, the instances linked to a variable of a certain type must have at least the function, methods and attributes of instances of that specific type. Consequently, many different instances can be linked to one variable to be declared on the base type of all the instances. This concept is called polymorphism (see figure 3 and allows for instance to use software without changing or knowing its source code.





**Figure 4:** Schema of the different parts of the detector simulation software X-CSIT. Source: [JWH<sup>+</sup>15].



**Figure 5:** Sketch how a charge cloud produced from a single interaction can spread to neighbouring pixels. Source: [JWH<sup>+</sup>15].

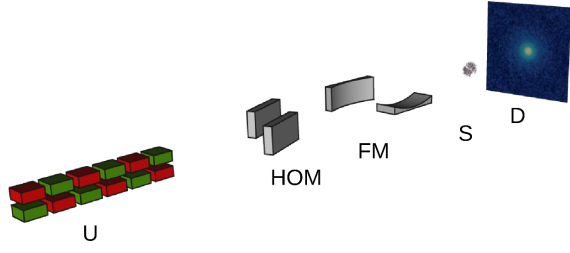
## 3 Existing Software

### 3.1 Geant4/ X-CSIT

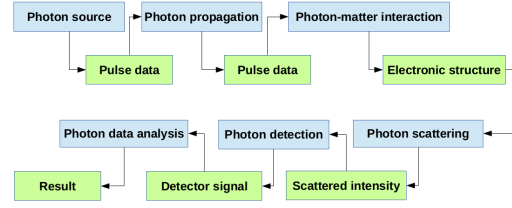
For the simulation of detectors Joy et al. (see ref. [JWH<sup>+</sup>15]) have created a software written in C++. X-CSIT is an object oriented approach to simulate the behaviour of 2D semiconductor pixel detectors. Due to the object orientation not only the initially implemented LPD, AGIPD, DCCS, pnCCD and FastCCD detectors but also derived detectors can be supported [JWH<sup>+</sup>15]. Initially written for being integrated into the **karabo** framework, the software is universal enough to be stand-alone. For this reason, utilization in this project is possible.

X-CSIT consists of three parts [JWH<sup>+</sup>15] as can be seen in figure 4. The first one, the particle simulation simulates, how in this case photons interact with the active layer of the detector. For this purpose X-CSIT acts as a wrapper of the interaction simulation software **Geant4** [JWH<sup>+</sup>15]. **Geant4** covers the physical models of a broad range of energy ranging from keV to TeV [AAA<sup>+</sup>03]. The models included can handle electromagnetic processes such as the photo electric effect and fluorescence but also hadronic and optical processes. The standard processes such as photo electric effect, Compton and Rayleigh scattering as well as Auger processes are also included [JWH<sup>+</sup>15] [AAA<sup>+</sup>03]. For this part X-CSIT has the task to manage the data transfer from and to **Geant4**.

The second part, the charge simulation, deals with the propagation of the charges and plasmas created by interactions. Their behaviour is mainly governed by drift and diffusion of electrons which can be described with a Gaussian normal distribution (see figure 5)



(a) Sketch of the experimental setup for single particle imaging with undulator (U), offset mirrors (HOM), focussing mirrors (FM), the sample (S) and the diffraction pattern captured at the detector (D).



(b) Representation of the experimental setup of 6a as a **SimEx** simulation.

**Figure 6:** Comparison of the setup between experimental and simulation in **SimEx**. Source: [FGAB<sup>+</sup>16].

with the following standard derivation [JWH<sup>+</sup>15]:

$$\sigma_d = \sqrt{\frac{2k_B T}{qE}} \cdot d. \quad (1)$$

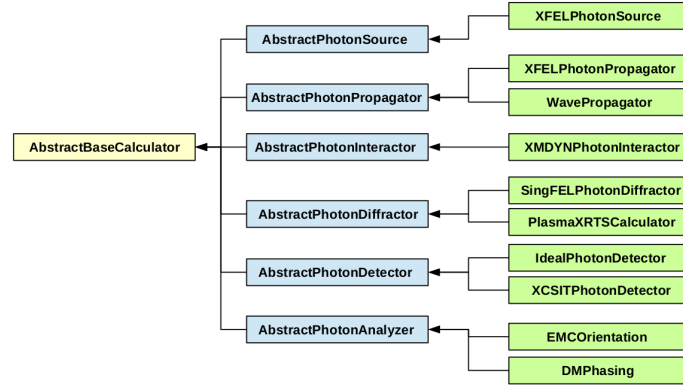
Here  $d$  is the depth in the material,  $T$  the temperature,  $q$  the drifting total charge and  $E$  the electrical field that pulls the charge to the readout electronics of the detector.

The last part of **X-CSIT** is made up by the detector readout electronics. The electronic simulation simulates the electronic components of the detector and their behaviour. For this purpose, **X-CSIT** offers modules which are combined to represent the circuits of the detector [JWH<sup>+</sup>15]. This simulation is not included in this project.

## 3.2 SimEx

**SimEx** is capable to simulate an entire experiment from the light source to the detector (see figure 6). Consequently, it is a tool aiming to improve the preparation, execution and analysis of experiments e.g. at the European XFEL [FGAB<sup>+</sup>16]. The platform has a hierarchical structure as can be seen in figure 7. The two topmost layers are abstract python classes defining the general structure of the the implementations. The last layer, depicted in green colour, are the implementations of those abstract interfaces [FGAB<sup>+</sup>16]. They are called calculators since here the simulation is run. The calculators are responsible for running the calculations [FGAB<sup>+</sup>16]. For this purpose, they include a method called **backengine**. Despite the fact, that they are written in python (<https://www.python.org/>), they often use software written in other languages such as C++ to perform the simulation. In this cases, the calculators act like a translator between input/output data from **SimEx** to the simulation application and visa versa. The communication between different calculators, different steps of the beam propagation, is implemented by exchanging **Hierarchical Data Format** (hdf5) files. Each calculator accepts one or more of them as input and creates one or more as output [FGAB<sup>+</sup>16].

Each calculator can cover a wide range of setups and experiment. Calculators simulating the source simulate the spectral, temporal and spatial characteristics of XFEL pulses.



**Figure 7:** Class hierarchy of the calculators in SimEx. Colours: white/ blue  $\hat{=}$  virtual, green  $\hat{=}$  implementation. Source: [FGAB<sup>+</sup>16].

Other ones simulate optical elements in the beam path such as apertures, mirrors or gratings. There is also a calculator that simulates the interaction between sample and photons and produces a diffraction pattern [FGAB<sup>+</sup>16]. Like the other calculators the detector simulation calculator receives input data, process them and creates output data, too [FGAB<sup>+</sup>16].

### 3.3 Extending C++ to Python

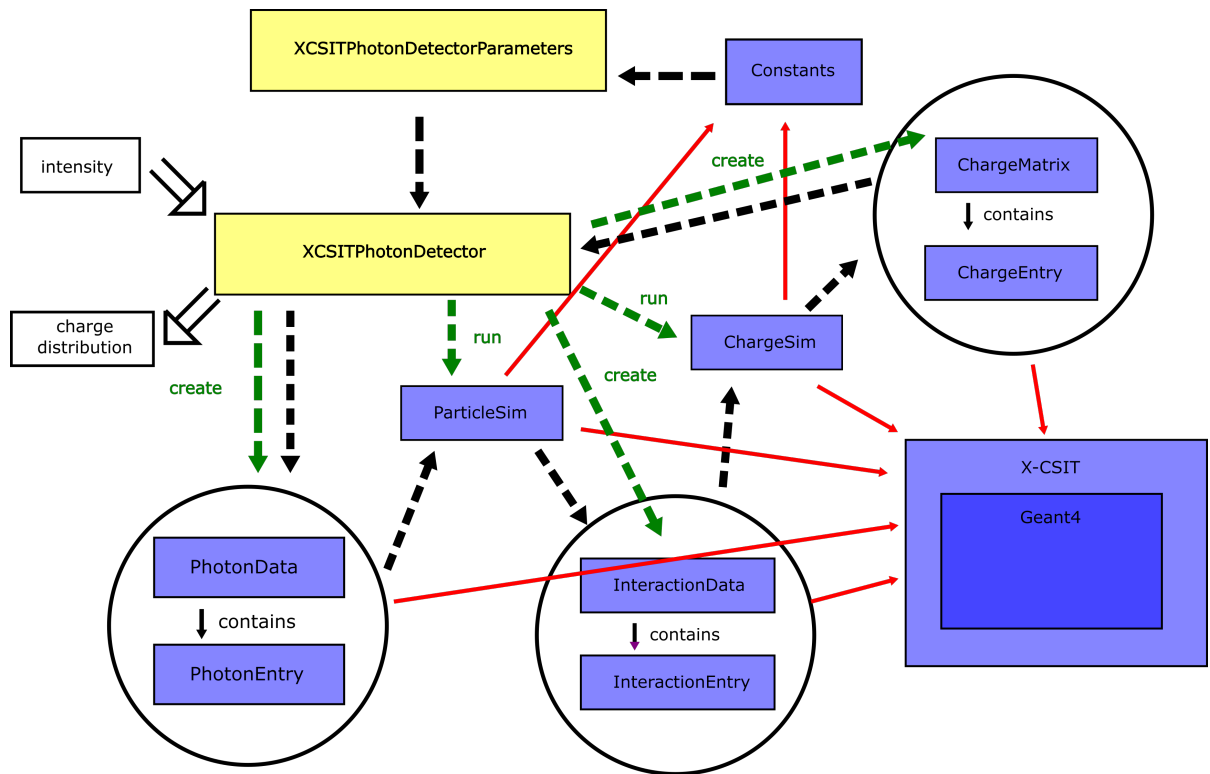
Since X-CSIT is written in C++ and the calculators of the **SimEx** are written in python, there is a need of extending C++ classes to python. The opposite process of using python code in C++ is called embedding [AS17]. To extend the classes of this project to python, **boost.python** ([http://www.boost.org/doc/libs/1\\_65\\_0/libs/python/doc/html/index.html](http://www.boost.org/doc/libs/1_65_0/libs/python/doc/html/index.html)) and **Swig** (<http://www.swig.org/>) were taken into consideration.

Swig is essentially a parser of the C++ header files that produces according to the set of desired options a shared object (Unix equivalent of Windows dynamic linked libraries) extending the header files. For this reason, it requires very little additional programming work of the developer and is quite efficient. However, not all features of C++11 standard are currently included (see <http://www.swig.org/>).

Additionally, X-CSIT already includes boost C++ libraries (<http://www.boost.org/>) and many users report that **Swig** can fail on projects including boost libraries. For this reason, **boost.python** was used to extend the C++ classes to python. **Boost.python** consists of header files that need to be added to the C++ implementations. Additionally, in the implementing source code a `BOOST_PYTHON_MODULE` needs to be defined. This module creates the equivalent of a python module and needs to define all the extended classes, their functions as well as their return and input parameters explicitly [AS17]. Consequently, **boost.python** can be seen as a module including abstract types that can be linked to C++ instances as well as to python instances.

## 4 Design

One of the major design ideas of this project is the usage of the already existing software X-CSIT. It already proved to be capable to perform a task like the desired one. For this



**Figure 8:** Sketch showing the dependencies and inheritance structure of the `py_detector_interface`. Colour code: yellow  $\hat{=}$  classes written in python, blue  $\hat{=}$  classes written in C++, red arrows  $\hat{=}$  inheritance, green arrows  $\hat{=}$  manipulation of other instances, black arrows  $\hat{=}$  data flow.

reason, this software package has the task to condense **X-CSIT** to its essential components necessary for the simulation in **SimEx**.

Since C++ and python programming languages belong to the object oriented programming languages, the concept of inheritance and polymorphism are well suited to achieve this. Polymorphism is the key concept for using self written classes of this design in **X-CSIT**. It allows also to minimize redundant source code that is always a potential source of error and very difficult to maintain. However, the required features remain accessible. For this reason, all the created classes except **Constants** are derived from **X-CSIT** or **SimEx** base classes and interfaces as can be seen in figure 8.

Another important aspect of this design is the design of the python class. In the end, this is the class that is accessed and used for performing the simulation. For this reason, it should have control over input and output as well as control over running the simulation. As can be seen in figure 8, the **XCSITPhotonDetector** calculator is given control over each step of the simulations. This includes creating data containers, initiate and run the simulations as well as reading the input file and creating the output file.

Nevertheless, the simulation itself should be triggered from C++ code. One reason for this is that tunnelling through the **boost.python** layer is assumed to be slow. Additionally, python code is slower than C++ and there where additional features needed such as a changed function signature in **ChargeSim**. Consequently, setting up and running simulations is programmed in C++ and only calling these functions is extended to python.

Last but not least, the entire project needs to be integrated into **SimEx**. With regard to the source code style and function this can easily be achieved by applying inheritance. However, one does not want to compile **X-CSIT** each time you compile also this project. For this reason, for both **X-CSIT** and **py\_detector\_interface** **cmake** is used to compile and link the compiled classes to shared objects. Shared objects are the Unix equivalents of Windows' dynamical linked libraries (.dll) offering a comfortable way to release and use applications.

## 4.1 C++ classes

Since **X-CSIT** was already implemented in the **karabo** framework there has already been a template how to write this classes. This template is the **xdsp/karabo framework** project by Tonn Rüter (see ref. [RHK<sup>+</sup>15]). Unfortunately, for that project **karabo** is required. For this reason, only the procedures suggested there could be used and had to be implemented again in a less restrictive and in a **karabo** independent manner. However, the order how certain **X-CSIT** classes are initialized and functions are called is adapted from those classes.

There are essentially two groups of C++ classes written for this projects. The first one consists of the input and output containers. They are derived from abstract interfaces located in **X-CSIT**, which have already an implementation in **X-CSIT**. However, at design time it was finally clear, if the implementation of **XCSITPhotonDetector** requires additional functionalities. Furthermore, the question if **boost.python** can extend classes and functions that are already compiled and stored in a shared object could not be answered. For these reasons, the data containers were implemented again using the abstract **X-CSIT** interfaces: **XPhotonEntry**, **XPhotonData**, **XInteractionData**,

category	options
DetectorType	pnCCD, LPD, AGIPD, AGIPDSPB, CAD
PlasmaSearch	BLANK
PlasmaSim	BLANKPLASMA
PointSim	FULL, FANO, LUT, BINNING

**Table 1:** This table show the options to choose from when selecting a mode for the simulations.

`XInteractionEntry`, `XChargeEntry` and `XChargeData`. Due to polymorphism other X-CSIT functions can deal with classes derived from them.

The second group of C++ classes deal with the simulations. There is a simulation of the photons interacting with the matter of the detector and a simulation dealing with the propagation of created charges in the detector. Both simulations have a parent class in X-CSIT. Their task is to behave like a filter. To run a simulation with the X-CSIT parent classes certain functions with certain formal parameters in their signature have to be called in a specific sequence. In order to avoid the need to extending all those types from X-CSIT possible to use as these formal parameters the simulation classes are necessary.

The functions of `ParticleSim` and `ChargeSim` receive strings to choose which instances of X-CSIT need to be instantiated and bound to a formal parameter of a X-CSIT simulation call. Furthermore, this make addition of e.g. detectors easier because they need to be added to the C++ simulation classes and `Constants` only. There is no need to export them to python. Currently the following options are included:

Furthermore, the simulation characterisation options specified in table 1 are needed in various classes. For instance, the „DetectorType“option is required for both `ParticleSim` and `ChargeSim`. Additionally, all their constants need to be accessible from the `XCSITPhotonDetectorParameters` class as well. For this reason, the constants are stored in an own class. Exploiting the capacities of C++ classes to inherit from many parent classes, `ParticleSim` and `ChargeSim` are not just inheriting from their X-CSIT parents but also from `Constants`. In principle, it would also be possible to make `XCSITPhotonDetectorParameters` inherit the constants from `Constants`. Since this is much more complicated due to the nature of the attributes of `Constants` (arrays of strings) than adding functions to `Constants` that return the values, the latter was implemented.

## 4.2 Python calculators

For this project two python classes were written. The first one, `XCSITPhotonDetectorParameters`, implements the abstract python class `AbstractCalculatorParameters`. Its purpose is to gather and check all the input parameters. If an parameter is set which is not specified in `Constants` an exception is raised. Nevertheless, instances of this class are essentially containers with property getter and setter functions. The properties are the same as the options in table 1.

The second class is the calculator itself. It implements `AbstractPhotonDetector` which itself is derived from `AbstractBaseCalculator`. For this reason, the way the simulation is performed is already predetermined:

1. After instantiation python calls immediately the `init` function. This function possesses three formal parameters: a `XCSITPhotonDetectorParameters` instance, a

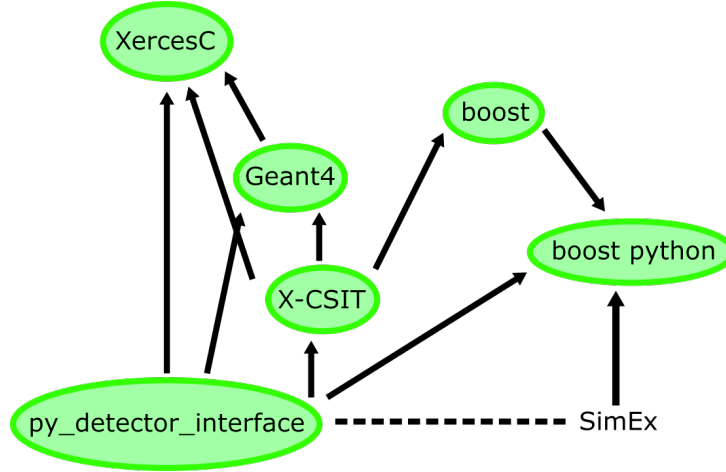
variable to store the path for the input file and a variable to store the path for the output file. Since python variables do not have types, the `init` function has to check if the inserted actual parameters fit with the required instances. Furthermore, `init` deals with incomplete input.

2. The next method to call is `readH5`. Since the input to this calculator is different to the input of `ParticleSim` and `ChargeSim`, the data from the hdf5 input file has to be translated: The matrix of intensities is read and transformed into instances of `PhotonEntry` stored in an instance of `PhotonData`. The instances of `PhotonEntry` store for each photon the following attributes: energy, normalized vector of flight direction, current position. Those values were calculated from the input data by applying geometry.
3. For running the simulation the `backengine` method has to be called. It consists of two parts:
  - a) The `PhotonData` instance is passed into `ParticleSim` which transfers the container into `XCSIT::XGeant4ParticleSim` where interactions of the photons with the detector material are simulated. The output container `InteractionData` is also passed to those classes. During the simulation it is filled with instances of type `InteractionEntry` that contain for each interaction the deposited energy in the material at a given site of the detector and the time when that happens after start.
  - b) The instance of `InteractionData` is handed to the instance of `ChargeSim` which transfers it into `XCSIT::XPlasmaPointChargeSim` and the `Geant4` classes respectively. Since the readout electronics cannot be at the surface of a detector an electrical field is applied to pull the created charges in the material to the readout electronics. During this propagation charge clouds resulting from e.g. plasmas can broaden and effect neighbouring pixels. This is simulated in `ChargeSim`. The output is an instance of `ChargeMatrix`, where each element, `ChargeEntry`, represents a pixel of the detector and each element contains the number of charges recorded in that detector pixel. Please note, that the perspective to look at the matrix is parallel to the z-axis/ propagation direction of the light.
4. Last but not least, the data containers and their content are written to the hdf5 output file at the location specified by the output path.

The structure of the input and output file can be found in the wiki of this project ([https://github.com/JPBGoe/py\\_detector\\_interface/wiki](https://github.com/JPBGoe/py_detector_interface/wiki)).

## 4.3 Cmake

On Unix systems the classical triad „configure“, „make“ and „make install“ is used to build software from source. Nevertheless, this can only be applied if there are not too many other applications to link to since otherwise creating the makefile would be a tough task. For this project, this classical approach does not work due to linking e.g. to X-CSIT. For this reason, a modern way to create the makefile is chosen. Using `cmake` (<https://cmake.org/>) allows to create a makefile including compilation and linking from



**Figure 9:** Depicted are the dependencies of the `py_detector_interface` project. The green circled names represent shared objects (.so) on Unix operating systems.

other libraries to a library. Additionally, `cmake` offers functions to create `bash` (<https://www.gnu.org/software/bash/>) files necessary for setting environmental variables. Furthermore, `SimEx` already uses `cmake` to compile its C++ sources and to download and install the entire software package. For these reasons, `cmake` was also used in this project. During this project CMakeLists.txt files, the source code files for `cmake`, have been created for this project as well as for `X-CSIT`. This allows to control the dynamic linking of the shared objects more efficiently, especially, since the linking dependencies of this project are not simple as can be seen in figure 9. In the end this project is also exported as `libpy_detector_interface.so`.

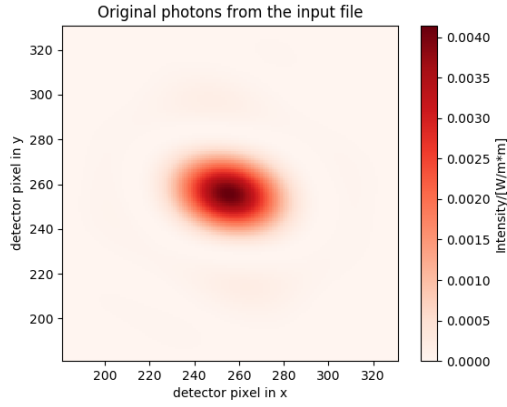
## 5 Application

To test the termination and as a proof of principle the `SimEx` tutorial ([https://github.com/eucall-software/simex\\_platform/wiki/SimEx-Tutorial](https://github.com/eucall-software/simex_platform/wiki/SimEx-Tutorial)) has been used to create a sample diffraction pattern of Nitrogenase Iron protein from the Protein Data Base (see PDB 2NIP [BWF<sup>+</sup>00]). The used detector quadrant of the „AGIPD“ detector has 512 x 512 pixel, each of a size of 200 x 200  $\mu\text{m}^2$ . All the photons are simulated with an energy of 4960 keV and the detector is 13 cm away from the origin of diffraction. To have enough photons at reasonable time the intensity  $I$  was non physically transformed into  $num$  photons as follows:

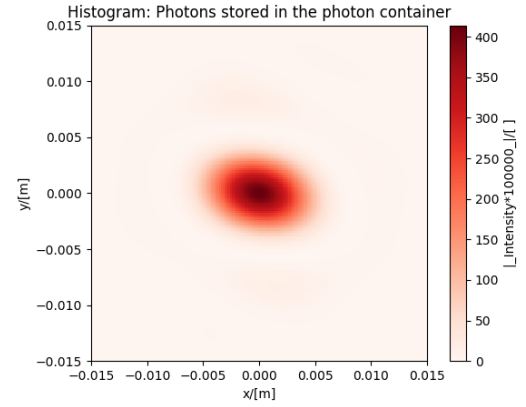
$$num = \lfloor I \cdot 100000 \rfloor.$$

As can be seen in the resulting images of this simulation (see figure 10) the chosen design is in principle capable of running such simulations. As expected for X-ray of such energies the interaction with the sample is very weak. However, one would expect at least a few scattered photons of the sample which cannot be observed in figure 10c). However, the charges after propagation meet the positions of the interactions. One cannot observe any „blooming“ events.

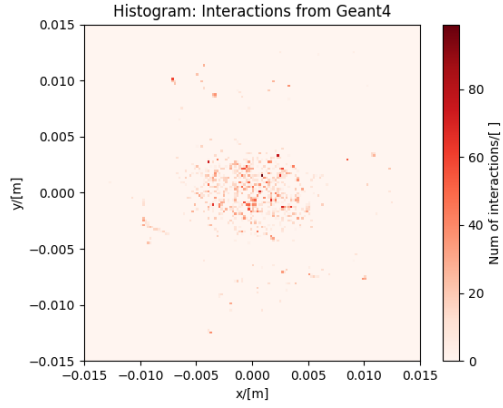




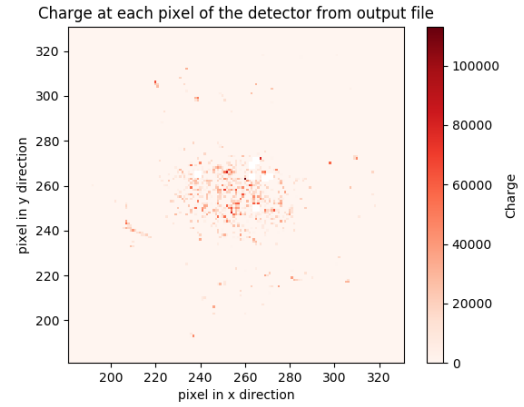
**(a)** Input intensity calculated by the diffraction calculator.



**(b)** Photonlist created from the the input (see 10a).



**(c)** Interactions of the photons with the detector material calculated by X-CSIT and Geant4.



**(d)** Output of the charge propagation simulation.

**Figure 10:** This figure shows the state of the data containers at intermediate steps of the simulation.

## 6 Remaining Issues

Apart from this proof of principle there are some issues in the python classes that need to be settled. The following list should give an overview. The following issue regards `XCSITPhotonDetectorParameters`:

- Inheritance: make it inherit from `Constants` to achieve a uniform inheritance structure between C++ and python classes respectively
- Test: add additional tests to the class

The following unsettled issues regard `XCSITPhotonDetector`:

- PointSim option LUT causes problems in X-CSIT: In this project running an simulation with this option will raise an exception
- Search: If there are too many photons (factor 100000) the charge simulation creates some e.g. 2 x 2 areas of infs or nans.
- Check: in the simulation there are no diffracted photons. Is this due to statistic or an program error?
- Multiple input files: At the moment only one input file is accepted. However, the diffraction calculator can produce multiple files at once that are necessary for the whole simulation
- Intensity photon transition: Include an appropriate intensity to photons calculation
- The charge matrix is turned by pi with respect to the recorded interactions. In the current simulation this has been corrected by a left-right and top-down flip of the charge matrix before saving it to the file.
- Single pattern treatment: So far all the incoming intensity matrices are simply summed up and the result is used as one single pattern. Better would be to run the simulation on each pattern and maybe sum up afterwards.
- Test of IO files: Especially, the input and output files need to be tested
- Test of physical properties: Does the number of interaction agree with the interaction cross section for this experiment? Does the charge simulation also agree?
- Include electronics: Include the electronics simulation into this project

## 7 Conclusion

Taking everything into consideration, there are lots of issues that remain to be settled. Especially, testing the results for consistency with the expectations and getting rid of smaller problems are task for future work. Furthermore, additional source code needs to be added to handle the interaction-to-photon transition in the `XCSITPhotonDetector` calculator more appropriately. Simulating each pattern and its intensity matrix on its own instead of summing them up and treating all the patterns as one would also be a valuable

feature that can be added. Nevertheless, it was shown, that the current design based on inheritance and polymorphism and by applying boost.python libraries and `cmake` makefile production is capable of accessing the X-CSIT detector simulation from `SimEx`.

## 8 Acknowledgements

First of all I would like to thank Ute Micheelsen, Olaf Behnke, Rainer Gehrke and the other organization team members for organizing this exciting summer school. Without their commitment and preparation this summer school would have been less informative, less fun and maybe also less productive than it was. Especially, their quick response to questions and the well done organization and provision of information was very helpful to concentrate on the contents of lectures and the project.

Furthermore, I would like to thank Adrian Mancuso, head of the XFEL Scientific Instruments SPB/SFX group, for accepting a summer student in his group. I am aware that students often mean additional bureaucracy and little scientific results/ progress. This is especially worth mentioning, since the summer student program took place at the time of probably additional workload due to the near opening of the XFEL facility. For this reason, I am grateful for being given this opportunity.

Special thanks go to my supervisor Carsten Fortmann-Grote for selecting the project and taking care of my progress. It was a demanding project to work on and I hope it can contribute to `SimEx` package a little bit. I enjoyed working at it and overcoming those obstacles on the way. Whenever there was a problem I was given hints, advice or new techniques for figuring out and solving the challenge. My impression is that I really learned a lot during the project. This ranges from programming languages like python via error prevention strategies such as unit tests to debugging tools such as `gdb`. Without his dedication and his advises this project would doubtlessly not have evolved half to the state it is.

Last but not least, I am very grateful to be allowed to participate in this summer school and to work at the XFEL site. It have been exciting weeks with a challenging project and interesting lectures. Furthermore, working at the fascinating XFEL facility only weeks before it officially opens was a really exciting experience. I am looking forward, for the next opportunity to participate in such a great project.

## References

- [AAA<sup>+</sup>03] AGOSTINELLI, Sea ; ALLISON, John ; AMAKO, K a. ; APOSTOLAKIS, J ; ARAUJO, H ; ARCE, P ; ASAI, M ; AXEN, D ; BANERJEE, S ; BARRAND, G u. a.: GEANT4—a simulation toolkit. In: *Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506 (2003), Nr. 3, S. 250–303
- [Alt11] ALTARELLI, M: The European X-ray free-electron laser facility in Hamburg. In: *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms* 269 (2011), Nr. 24, S. 2845–2849
- [AS17] ABRAHAMS, David ; SEEFELD, Stefan: *boost.python documentation*. [http://www.boost.org/doc/libs/1\\_65\\_0/libs/python/doc/html/index.html](http://www.boost.org/doc/libs/1_65_0/libs/python/doc/html/index.html). Version: Spt 2017
- [BWF<sup>+</sup>00] BERMAN, H.M. ; WESTBROOK, J. ; FENG, Z. ; GILLILAND, G. ; BHAT, T.N. ; WEISSIG, H. ; SHINDYALOV, I.N. ; BOURNE, P.E.: *The Protein Data Bank*. [www.rcsb.org](http://www.rcsb.org). Version: 2000. – Nucleic Acids Research, 28: 235–242
- [Ebe17a] EBELING, Bernd: *Facts and Figures*. online. [https://www.xfel.eu/facility/overview/facts\\_amp\\_figures/index\\_eng.html](https://www.xfel.eu/facility/overview/facts_amp_figures/index_eng.html). Version: 2017
- [Ebe17b] EBELING, Bernd: *XFEL website*. <https://www.xfel.eu/facility/>. Version: September 2017
- [FGAB<sup>+</sup>16] FORTMANN-GROTE, C ; ANDREEV, AA ; BRIGGS, R ; BUSSMANN, M ; BUZMAKOV, A ; GARTEN, M ; GRUND, A ; HÜBL, A ; HAUFF, S ; JOY, A u. a.: SIMEX: Simulation of Experiments at Advanced Light Sources. In: *arXiv preprint arXiv:1610.05980* (2016)
- [FGBJ<sup>+</sup>17] FORTMANN-GROTE, Carsten ; BUZMAKOV, Alexey ; JUREK, Zoltan ; LOH, N-TD ; SAMOYLOVA, L ; SANTRA, R ; SCHNEIDMILLER, EA ; TSCHENTSCHER, T ; YAKUBOV, S ; YOON, CH u. a.: Start-to-end simulation of single-particle imaging using ultra-short pulses at the European X-ray Free-Electron Laser. In: *IUCrJ* 4 (2017), Nr. 5, S. 560–568
- [Gra09] GRAAFSMA, Heinz: Requirements for and development of 2 dimensional X-ray detectors for the European X-ray Free Electron Laser in Hamburg. In: *Journal of Instrumentation* 4 (2009), Nr. 12, S. P12011
- [HBD<sup>+</sup>11] HENRICH, B ; BECKER, J ; DINAPOLI, R ; GOETTLICHER, P ; GRAAFSMA, H ; HIRSEMANN, H ; KLANNER, R ; KRUEGER, H ; MAZZOCCO, R ; MOZZANICA, A u. a.: The adaptive gain integrating pixel detector AGIPD a detector for the European XFEL. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 633 (2011), S. S11–S14
- [JWH<sup>+</sup>15] JOY, Ashley ; WING, Matthew ; HAUF, Steffen ; KUSTER, Markus ; RÜTER, Tonn: X-CSIT: a toolkit for simulating 2D pixel detectors. In: *Journal of Instrumentation* 10 (2015), Nr. 04, S. C04022

- [PG11] POTDEVIN, Guillaume ; GRAAFSMA, Heinz: Analysis of the expected AGIPD detector performance parameters for the European X-ray free electron laser. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 659 (2011), Nr. 1, S. 229–236
- [RHK<sup>+</sup>15] RUTER, Tonn ; HAUF, Steffen ; KUSTER, Markus ; JOY, Ashley ; AYERS, Ruth ; WING, Matthew ; YOON, Chun H. ; MANCUSO, Adrian P.: X-ray detector simulation pipelines for the European XFEL. In: *Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2015 IEEE* IEEE, 2015, S. 1–4
- [SDH<sup>+</sup>10] SHI, X ; DINAPOLI, R ; HENRICH, B ; MOZZANICA, A ; SCHMITT, B ; MAZZOCCO, R ; KRÜGER, H ; TRUNK, U ; GRAAFSMA, H ; CONSORTIUM, Agipd u. a.: Challenges in chip design for the AGIPD detector. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 624 (2010), Nr. 2, S. 387–391
- [SDR08] SCHMÜSER, Peter ; DOHLUS, Martin ; ROSSBACH, Jörg: *Ultraviolet and soft X-ray free-electron lasers: introduction to physical principles, experimental results, technological challenges*. Bd. 229. Springer Science & Business Media, 2008