GIS++: Modern C++ for Efficient and Parallel In-Memory Spatial Computing

Martin Werner
martin.werner@unibw.de
Institute for Applied Computer Science & Forschungsinstitut CODE,
Bundeswehr University Munich, Germany

ABSTRACT

In this workshop, the participants get a deep dive to spatial computing on top of the Boost Geometry Library in C++. The workshop will highlight core concepts of generic programming, give hands-on experience in the area of OGC geometry and range and nearest neighbor queries based on arbitrary combinations of OGC and custom predicates. Finally, the tutorial gives an outlook how easy it has become to interface with covering libraries including GDAL, Python, and OpenGL.

CCS CONCEPTS

• **Information systems → Data structures**; • **Software and its engineering → Development frameworks and environments**; *Software creation and management.*

KEYWORDS

Spatial Computing, C++, Spatial Indexing, Spatial Queries, OGC Simple Feature Geometry

ACM Reference Format:
Martin Werner. 2019. GIS++: Modern C++ for Efficient and Parallel In-Memory Spatial Computing. In *1st ACM SIGSPATIAL International Workshop on Geospatial Data Access and Processing APIs (SpatialAPI'19), November 5, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3356394.3365587

1 INTRODUCTION

Spatial computing can be very challenging because of the variety and complexity of the topic. Many systems have been proposed that are optimized for performing certain operations with high performance. Other systems and frameworks have been proposed that can bridge a wide variety of topics, however, typically sacrificing efficiency by a significant margin.

However, there is a huge amount of high-quality, peer-reviewed software libraries that can bridge many worlds all being based in the same philosophy of generic programming and so-called concepts, which basically are abstractions of data types allowing careful implementations of different variants of algorithms for all reasonable situations and applying the right subset of these algorithms at runtime while performing the selection at compile time. The result is that a generic library with its many variants does not need any runtime code to select among the available implementations leading to exceptional performance gains in contrast to similarly flexible libraries, for example, based on object orientation.

In this tutorial, we want to expose interested participants to the beautiful world of modern C++ (to us, this means C++ since full adoption of the C++11 standard). In contrast to current mainstream solutions based on R, Python, Java, or similar low-profile languages, being able to implement code in C++ brings us many advantages in terms of

- efficiency and performance,
- immediate technology support including shared memory multiprocessing (SMP), supercomputing (MPI), GPU computing (OpenACC, NVIDIA thrust),
- source code portability,
- elegance, and
- library quality and support, and
- the ability to bind to a multitude of low-profile languages, most notably R and Python.

With these core features, it is surprising that the hard work behind the generic Boost libraries has not yet been adopted by the spatial computing community still sticking with old prejudice about C and C++ being difficult to learn and insecure to use. A first step to breaking the dominance of Java and Scala-based Big Data systems is, therefore, to show the power and simplicity of modern C++ for spatial computing research.

2 WHAT WILL BE COVERED?

This tutorial will be split into three parts, each consisting of a comparably short introductory presentation and a longer practical session. In the first part, Intro and Recap, we will collect the audience and bring everyone into the position of compiling sample code and knowing the main ideas of modern C++ (a bit). Then, we will start our deep-dive into Boost Geometry, an excellent, easy-to-use, complete, peer-reviewed, and highly performant library for doing spatial computing on top of your own already existing data types. A last section will show next steps and give hints on how to integrate with other libraries including GDAL (for data I/O), HDF5 (for parallel I/O and supercomputing), with Python (for easy usage, QGIS integration), with OpenGL and CUDA (vertex buffer objects) for real-time visualization or GPU computation, and reserve a few minutes for individual questions and discussions.

2.1 Intro and Recap (30 minutes)

In order to make a useful tutorial, we will expect all participants to be familiar with traditional C, C++, or Java. We will quite quickly explore how modern C++ is different from classical C and C++ and show some simple generic recipes including safe pointers, parallel loops and thread synchronization and give a tour of the three main containers in the standard template library: maps, lists, and vectors. It will be very helpful if participants have read about these concepts already.

With respect to applying these concepts, we will distribute access credentials to all users to the ICAML platform [2] and compile a "Hello World!" program with a simple Docker container.

2.2 The Generic Geometry Library (aka Boost.Geometry, 60 minutes)

Next, we will introduce the Boost Geometry Library [1]. This library is a generic library in which many aspects are controlled at compile time and the optimization stage of modern C++ toolchains gets away all the boilerplate code that is needed for full flexibility. Just to name a few features:

- fully independent of data representation,
- supports different coordinate systems (Euclidean, Spherical, Geographic) for many algorithms,
- all algorithms have the same and simplest name, for example, distance means distance in the associated coordinate system,
- full OGC Simple Feature support including operations and relations,
- excellent spatial indexing support with flexible query interfaces.

We will start learning how basic geometry (points, polygons, linestrings, and boxes) are declared and attached to already existing types and how coordinate system tags are attached and used for automatic algorithm selection. Then, we will show how to build container out of these representing MultiPoint, MultiLinestring, and MultiPolygon geometries.

We will continue showing how basic operations for simple geometry (e.g., ST_Repair or ST_Buffer) can be applied and how the dimensionality-enhanced nine intersection model (DE-9IM) relations between different geometries can be computed including parallel execution.

We will continue to explore the library boost::geometry::index bringing various spatial indexing trees including R* trees (with bulk loading support) to your system and how you can use them in a filter-and-refine manner or as well for exact queries including custom predicates. In this section, we will heavily rely on C++11 lambda expressions in order to reach exceptionally readable code with astonishing runtime behavior.

We will briefly cover how to read and write well-known text (WKT) and apply all that we have learned for assigning city names to points, e.g., for a simple spatial join operation including import and export from WKT CSV (for use with QGIS).

2.3 Outlook and Road Ahead (30 minutes)

In the last half an hour, we will show to the audience next steps illustrating how powerful this approach turns out in practice. We will show with minimal examples (show, not teach),

- how parallel file systems can be easily integrated using the HDF5 library,
- how GDAL can be used to load (nearly) arbitrary GIS files directly
- how implemented functionality can be exposed to Python in just a dozen lines of code (including NumPy)
- how the Boost Graph Library can be used together with Boost Geometry
- how a different coordinate system distance calculation can be instantiated and applied

In summary, we show the power of the given libraries and of the whole C++ ecosystem for current and future spatial computing needs and hope that we will encourage more and more people to learn that C++ (since C++11, but even more with C++17, and C++20) has become a better and better programming language while getting easier and easier to use.

3 PARTICIPANT PREPARATION

Participants should have written programs in one of the major imperative programming languages (Python, C++, C, Java, PHP, ...). Some knowledge of generic programming, functional programming, and parallel programming is helpful, but not expected.

We will provide a Web-based remote system for having an equal play ground for all participants through our Interdisciplinary Center for Applied Machine Learning (ICAML) [2]. Participants might want to have a simple GIS like Quantum GIS (QGIS) [3] installed on their local computer in order to easily inspect data and results during development as we skip a time-consuming introduction to how spatial data should be visualized in the modern C++ ecosystem.

Advanced participants can check out the web page of this tutorial at https://www.martinwerner.de/teaching/spatial-cpp to learn, how to set up a development system using a modern Linux or Docker.

ACKNOWLEDGMENTS

We acknowledge support from the Interdisciplinary Center for Applied Machine Learning (ICAML). The ICAML is funded by the Federal Ministry of Education and Research, Germany (Bundesministerium fur Bildung und Forschung, Forderkennzeichen 01IS17076).

REFERENCES

[1] Barend Gehrels, Bruno Lalande, Mateusz Loskot, Adam Wulkiewicz, and Menelaos Karavelas. 2019. The Generic Geometry Library (GGL) a.k.a. Boost::Geometry. https://www.boost.org/doc/libs/1_70_0/libs/geometry/doc/html/index.html.
[2] ICAML 2019. Interdisciplinary Center for Applied Machine Learning (ICAML). http://www.icaml.org.
[3] QGIS 2019. Quantum GIS (QGIS) . https://qgis.org.