Summary of Select Activities

Dmitri Smirnov

Updated September 20, 2019

Inroduction

- **2014: Member of the STAR group**
  - Event reconstruction: Tracking, vertexing, geometry
  - Software peer reviews, general support
- **2010: Member of the RHIC Spin group**
  - Online support and operations for RHIC CNI polarimeters
  - Offline analysis for regular measurements of proton beam polarization
- **2006: D0 experiment at Fermilab**
  - Tracking detector expert support: Hardware, online controls, offline calibrations
  - Detector performance monitoring
  - Integration with event reconstruction software
- **2005: PhD on top quark properties at Femilab's CDF experiment**

Overview of STAR Software

- Similar to other big experiments STAR software can be split into two large parts—**offline** and **online**
  - **Offline** code is responsible for event reconstruction, geometry, simulation, calibration, database, ...
  - **Online** code is responsible for reading out detectors, creating raw data DAQ files, detector monitoring, ...
- My focus is on the **offline** part
- STAR code is located in a CVS repository. Total size is 26GB

```
$echo $CVSROOT
/afs/rhic.bnl.gov/star/packages/repository
```

  - Includes unpatched MC generators, patched Geant3, paper drafts, and user analysis code with binary data (23GB ☝)

Fun Facts and Statistics

Based on Git repo as of August 2019

- Overall **5M lines of code** in the most popular languages

```
$tokei --sort=code --type="FORTRAN Legacy,C,C Header,C++,C++ Header,Python,Perl" star-cvs/
```

Language	Files	Lines	**Code**
FORTRAN Legacy	3673	2400728	**1944928**
C Header	3657	2165256	**1869860**
C++	3630	1357126	**895478**
C	10405	475461	**329001**
Perl	340	120573	**81639**
C++ Header	974	76729	**40200**
Python	176	38364	**29307**
Total	22855	6634237	**5190413**

- About **3.7M lines of code** when exclude multiple versions of MC generators and Geant3

```
$tokei --exclude "StRoot/StarGenerator" --exclude "*geant3*" ...
```

Fun Facts and Statistics

Based on Git repo as of August 2019

	commits	files/c	lines++	lines--
fisyak	4844	8.2	5711940	1614370
didenko	4595	3.1	475511	128456
perev	3842	3.2	2733529	1433259
jeromel	3424	2.6	2098468	60738
fine	2880	2.6	617199	106234
jwebb	1457	4.2	5481898	41585
genevb	1210	2.3	283911	37494
kathy	988	2.3	63332	33938
ullrich	945	2.9	190422	32948
smirnovd	812	2.5	42806	26669
balewski	745	3.4	164257	31887
tonko	710	2.2	66361	16473
potekhin	578	6.6	1607303	6615
jcs	558	1.9	251117	16502
tai	550	7.0	303363	59626
caines	512	2.3	72928	18276
nevski	493	2.3	222354	120066
lasiuk	488	2.1	65275	18604
jml	477	3.0	104915	16943
pibero	472	2.7	52648	37253
mmiller	459	7.5	341599	36296
akio	448	4.6	347134	18082
pruneau	446	4.9	74694	58822
tull	440	4.9	162712	26052
calderon	431	3.0	56566	14644
sakrejda	373	1.6	31555	7118
suaide	339	3.1	109206	27500
ward	317	2.4	37787	6898
rfatemi	299	1.4	477356	6937
posk	291	3.6	59492	27624
laue	286	3.6	39720	10031
kocolosk	270	3.1	149353	20530
porter	255	8.1	71702	17809
dmitry	244	1.9	10679	1053
andrewar	243	1.3	17096	2498
avossen	230	2.5	43680	5147
geurts	221	1.8	28456	6281

- Legend

  `commits` : Total number of commits
  `files/c` : Files changed per commit
  `lines++` : Lines added
  `lines--` : Lines removed
  name : NPPS group members

- First commit

  `Author: starlib <> 1993-02-09`

- 44,300 total commits by 250 authors
- 4.6 commits per day
- No activity during 35% of the time

Statistics by [GitStats]

oldi	219	2.5	32010	11972
zhux	218	1.2	7358	1296
yepes	204	5.1	76010	16497
prindle	198	6.0	124399	35711
dongx	180	1.8	25669	4280
bum	180	3.1	11013	6635
hack	179	6.3	73366	8660
wenaus	177	2.5	13133	2404
		...		
panitkin	1	2.0	151	0
		...		

Git Repository

github.com/star-bnl/star-cvs

- After joining STAR realised that developing with CVS is not very efficient
  - There are no branches in the STAR CVS-based development workflow. The changes committed on the MAIN trunk and tested nightly
  - Without branches hard to test your changes in an isolated environment
  - Little chance to review or comment on changes. Peer reviews are done only for completely new code
- Decided to convert to a Git repository
  - Unfortunately, this activity is not widely accepted or supported in STAR
  - As a result the Git repository is read only and synced with CVS by a cron job a few times a day

Git Repository

github.com/star-bnl/star-cvs

- The import script is based on `cvs2git` converting selected CVS subdirectories to a Git repository
    - Several CVS subdirectories excluded resulting in the size of selected files of 1.7GB
    - We run BFG repo cleaner to correct imported history
        - E.g. CVS history contains a commit of a `.git` folder 😀 (Must be removed)
- Size of bare `.git` with full history is about 500MB and the full checkout is 900MB

```
$git clone ...
$git checkout master
$du --exclude .git -h -d1 star-cvs/ | sort -h
...
878M    star-cvs/
```

- Compare to ROOT repository sizes of 700MB and 1.7GB respectively
- The code of external packages (e.g. MC generators) still can be removed from the DVCS making the STAR software more compact and attractive

Building STAR Libraries

- Default STAR toolchain is based on Scientific Linux 7.4 (i.e. Red Hat 7) with upgraded gcc 4.8.5 with C++11 support
- Incomplete list of major external dependencies includes the following
  - **Packages:** ROOT v5.34/30 (libraries, rootcint), MC generators, Geant3
  - **Libraries:** CERNLIB, xml2, log4cxx, mysql
  - **Tools:** python2, perl, bison/yacc, flex
- `cons` is used to build the STAR libraries
  - Often claimed advantage of `cons` and CVS is the ability to checkout and compile individual files/subdirectories
  - In my opinion, this breaks the isolation of your local builds as you don't have control over the remote one

Motivation for CMake

- `cons` is the default build system for STAR
  - Implemented in Perl. `cons` home page last updated in 2001
    "cons has been decommissioned [...] scons has effectively replaced it"
- Building with `cons` does not easily support installation of STAR libraries and auxiliary files into multiple directories from same source
  - E.g. this is useful for comparison of different compiler options
  - No explicit "installation" step, libraries installed in checked out source directory 😒
- CMake has extensive online documentation, regular releases, reasonable defaults
  - Easy switch between different compilers, their versions, or generators
    (e.g. make vs ninja)
  - Generation of dependency trees which can be examined or used in an IDE
  - Close to becoming a "standard" for C++ projects

Building STAR Libraries with CMake

- Implementation of CMake builds is quite mature and close to a first public release
  - Minimum required version is 3.6
  - Development takes place at https://github.com/star-bnl/star-sw
- There are known differences in the final libraries produced with `cons` and CMake but most test jobs can run without a problem
- CMake installs all components in an isolated location (below example is for 32-bit build)

```
19M     star-install/.sl74_gcc485/bin
152M    star-install/.sl74_gcc485/lib
70M     star-install/.sl74_gcc485/include
86M     star-install/StarDb
6.6M    star-install/StRoot
361M    star-install/
```

Building STAR Libraries with CMake

- STAR libraries can be built from source in about 20 minutes on a four-core Intel Xeon E5 @ 3.00GHz with magnetic hard drive
  - For example, timing for 64-bit builds with
    `CMAKE_BUILD_TYPE=RelWithDebInfo`

```
$time make -j4
real    20m17.685s
```

    `CMAKE_BUILD_TYPE=MinSizeRel`

```
$time make -j4
real    16m44.444s
```

  - 32-bit builds are slightly slower (~5–10%)
  - Static library builds are slightly faster (~15%)

Building in a Docker Container

- Building STAR software inside a docker container can be used for quick test builds followed by test jobs. A possible general scenario:

  1. A docker image is created with all STAR software dependencies, i.e. ROOT, CERNLIB, and other libraries.
     Based on Ubuntu 16 the image size is 2GB
  2. A container with STAR code is built on top of that image and tests are executed.
     Image size increases to 2.3GB
  3. On success the newly created image with STAR libraries is tagged "latest" in the docker repository
  4. Subsequent builds can use the "latest" image as cache resulting in faster incremental build

- For example, see Best practices for building docker images with GitLab CI
- Dockerfiles are available in https://github.com/star-bnl/star-sw/docker
- A couple measurements with incremental builds:
  - When `touch` all files in `StRoot/StEvent` incremental build takes ~6m
  - When `touch` all files in `StRoot/Sti*` incremental build takes ~1m 30s

Additional Thoughts

- CMake can significantly reduce build times of STAR code base by utilizing multi-threading in `make`
  - Most of STAR software can be built from source in about 20 minutes
  - Comparable nightly builds with single-threaded `cons` take about 2 hours to build
- Slow nightly builds may not be the main problem of STAR software approaching its EOL
  - A practical use of multi-threaded builds may be applied to fast (incremental) builds to allow for a quick feedback (under 5 minutes) from automatically triggered CI tests
  - Such workflow would assume extensive use of branches which is not clear (to me) if can/should be implemented with CVS

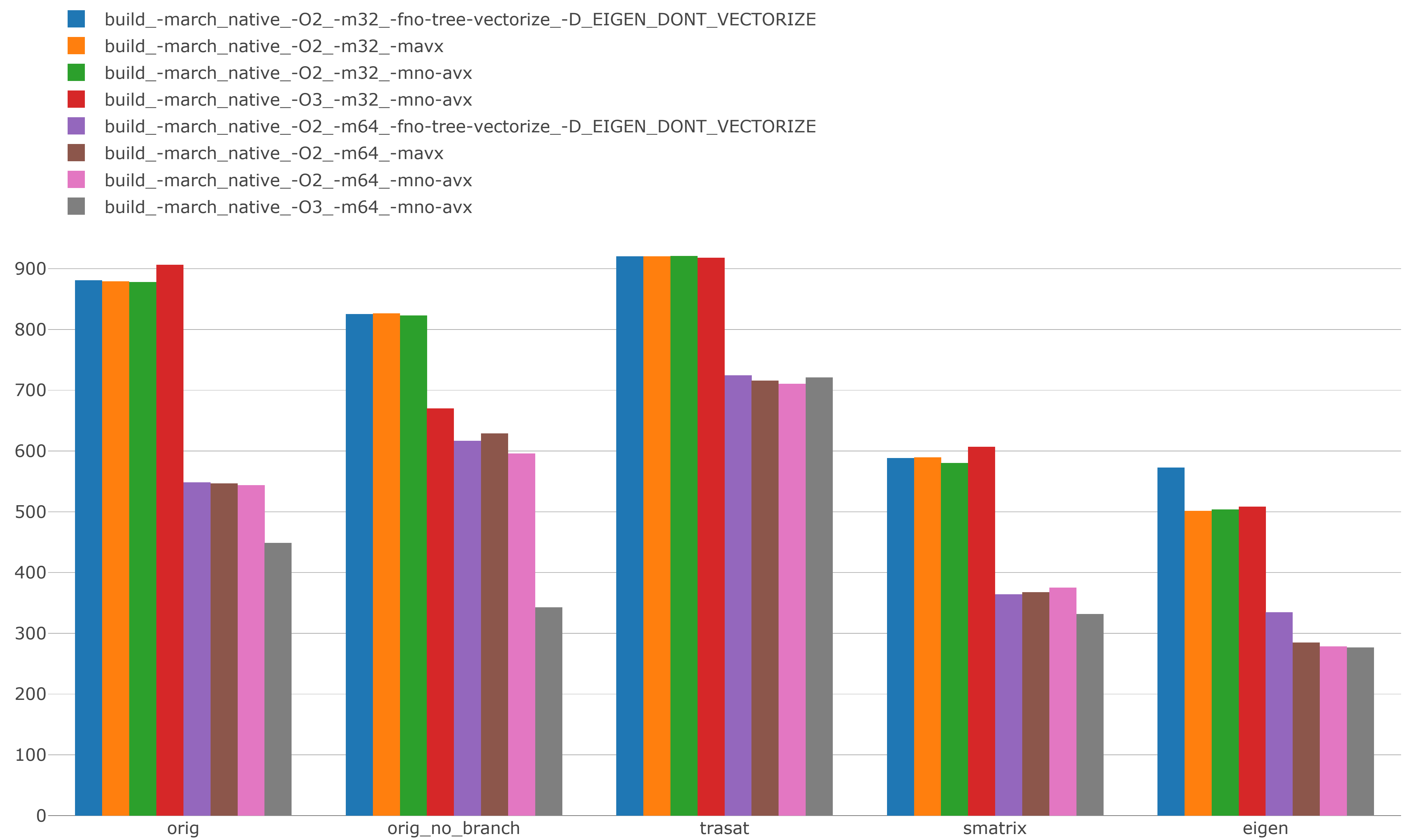
Reconstruction Code Optimization

- Using callgrind identified a couple of routines where tracking spends a significant fraction of time
- Not surprisingly the functions deal with matrix calculations. E.g.
    - Error propagation. A matrix product $B = ASA^T$ where A and S are 6×6 matrices, S is a symmetric covariance matrix
    - Weighted average and its covariance matrix of two multi-dimensional vectors

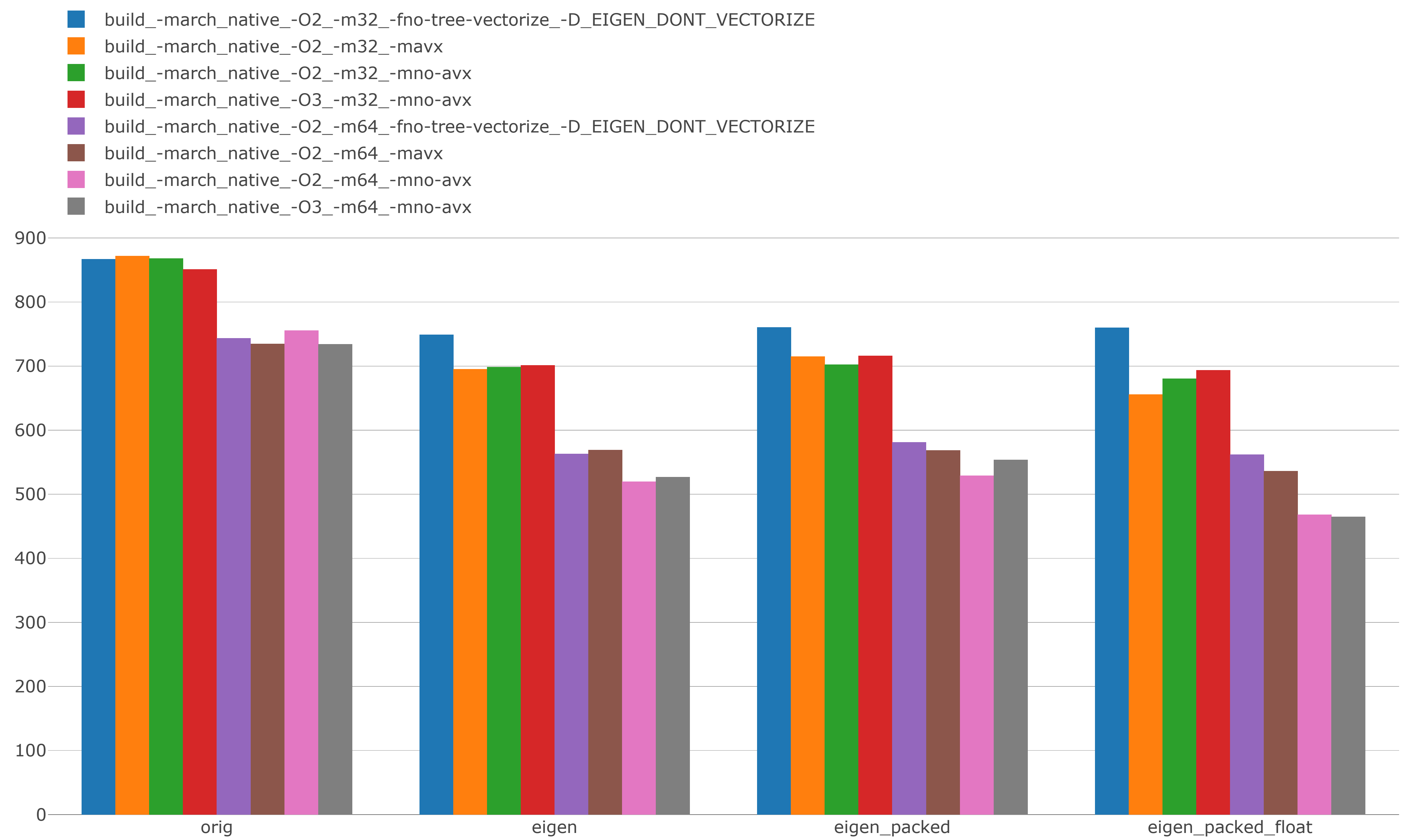
$$M = (W_1 + W_2)^{-1}(W_1 X_1 + W_2 X_2), \quad W = (W_1 + W_2)^{-1}$$

- Benchmarked various implementations including vectorized `Eigen`, `ROOT::Math::SMatrix`, `TCL::trasat`
- Tested different precisions and packing of input values, compiler flags -O2, -O3, -m32, -m64, sse, avx, auto vectorization

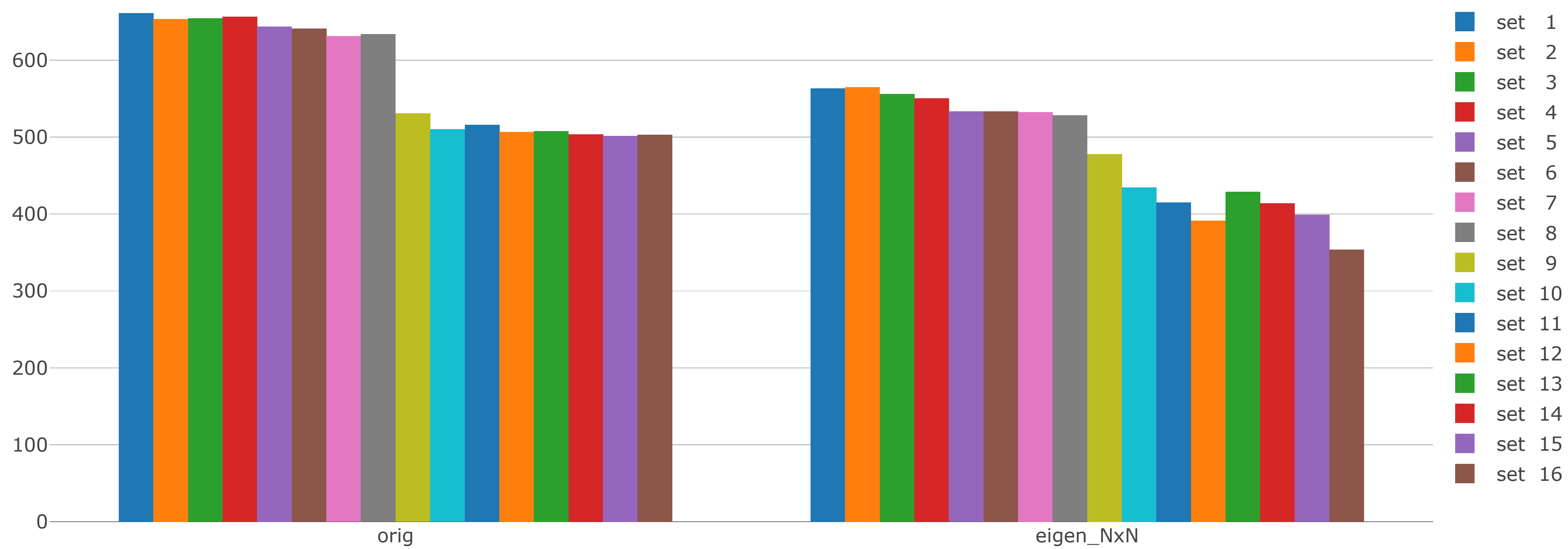
Alternative Implementations: ASA^T



Alternative Implementations: Weighted Average



Comparison of Options: Weighted Average




```
set1  = "-O2 -m32 -msse -mno-avx -ftree-vectorize"
set2  = "-O2 -m32 -msse -mno-avx -fno-tree-vectorize"
set3  = "-O2 -m32 -msse -mno-avx -ftree-vectorize -D EIGEN_DONT_VECTORIZE"
set4  = "-O2 -m32 -msse -mno-avx -fno-tree-vectorize -D EIGEN_DONT_VECTORIZE"
set5  = "-O3 -m32 -msse -mno-avx -ftree-vectorize"
set6  = "-O3 -m32 -msse -mno-avx -fno-tree-vectorize"
set7  = "-O3 -m32 -msse -mno-avx -ftree-vectorize -D EIGEN_DONT_VECTORIZE"
set8  = "-O3 -m32 -msse -mno-avx -fno-tree-vectorize -D EIGEN_DONT_VECTORIZE"
set9  = "-O2 -m64 -msse -mno-avx -ftree-vectorize"
set10 = "-O2 -m64 -msse -mno-avx -fno-tree-vectorize"
set11 = "-O2 -m64 -msse -mno-avx -ftree-vectorize -D EIGEN_DONT_VECTORIZE"
set12 = "-O2 -m64 -msse -mno-avx -fno-tree-vectorize -D EIGEN_DONT_VECTORIZE"
set13 = "-O3 -m64 -msse -mno-avx -ftree-vectorize"
set14 = "-O3 -m64 -msse -mno-avx -fno-tree-vectorize"
```

A Take Away from Optimization Studies

- The test are standalone, i.e. work with extracted individual routines
  - Made sure the input is realistic, in fact, sampled from real data
- Tests with Eigen implementation of matrix operations give up to 40%
  - This translates into at least 10% speed-up of the full reconstruction jobs
- Significant gain in speed going from 32-bit to 64-bit compilation with at least -O2 optimization flags
  - STAR has not switched to 64-bit builds yet
  - A 64-bit STAR libraries are build centrally with `cons` but in non-optimized mode

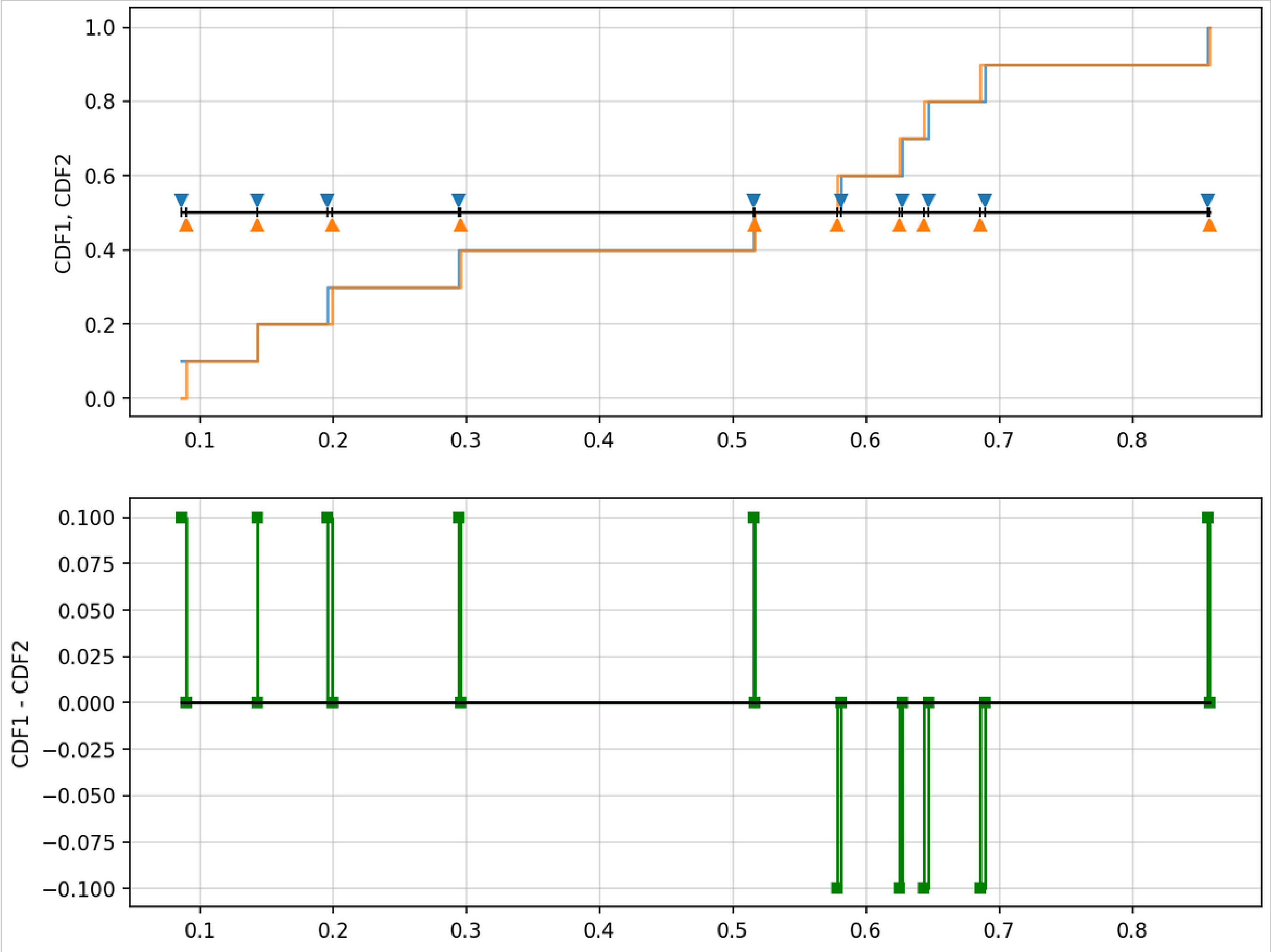
Switching from 32-bit to 64-bit Builds

- An extensive study has been performed to evaluate divergences in reconstruction chain output under assumption of only "technical" modifications
  - A change that does not affect the logic of the algorithm. E.g. change in precision of some internal calculations or compilation flags affecting certain CPU instructions
- In statistics and information theory, a **statistical distance** can quantify the difference between two samples
  - We considered the **Kolmogorov-Smirnov** (KS) and **Wasserstein** metrics (aka the earth mover's distance (EMD))

$$D = \sup_x |F(x) - G(x)|, \quad W = \int |F(x) - G(x)| dx$$

- More details available in the notebook launch binder Identifying Small Changes in Algorithm Output

Comparison of Cumulative PDFs of Two Samples


CDF1, CDF2
1.0
0.8
0.6
0.4
0.2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
CDF1 - CDF2
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

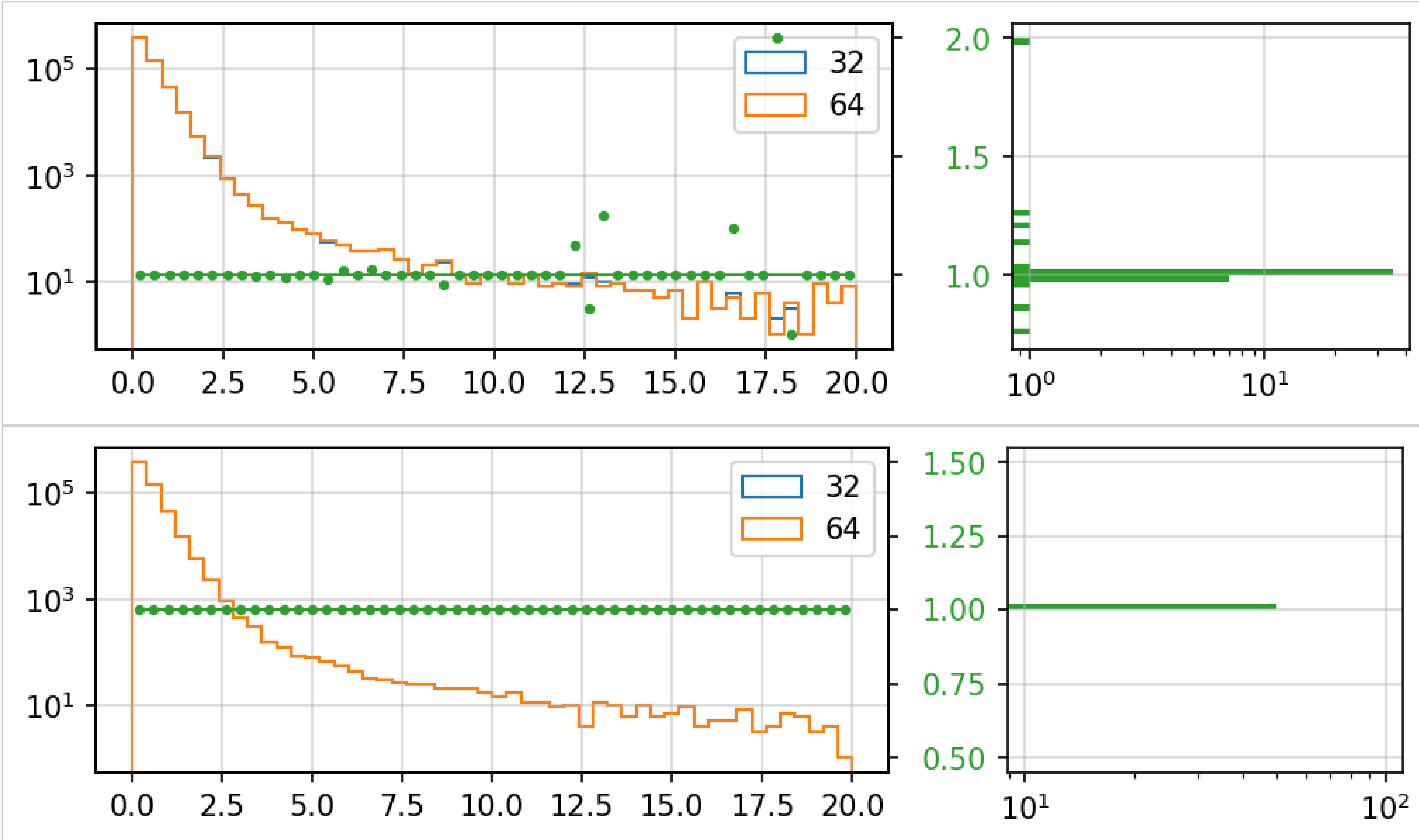
Additional Checks and Use Cases

- In addition to KS and EMD metrics we directly count the maximum number of consecutive values from same data set
- All the KS, EMD, and direct count give similar correlated results
- An additional effort can be made to formalize the requirements and limitations of the method
  - Improve documentation
  - Use in quick tests for refactoring and similar changes

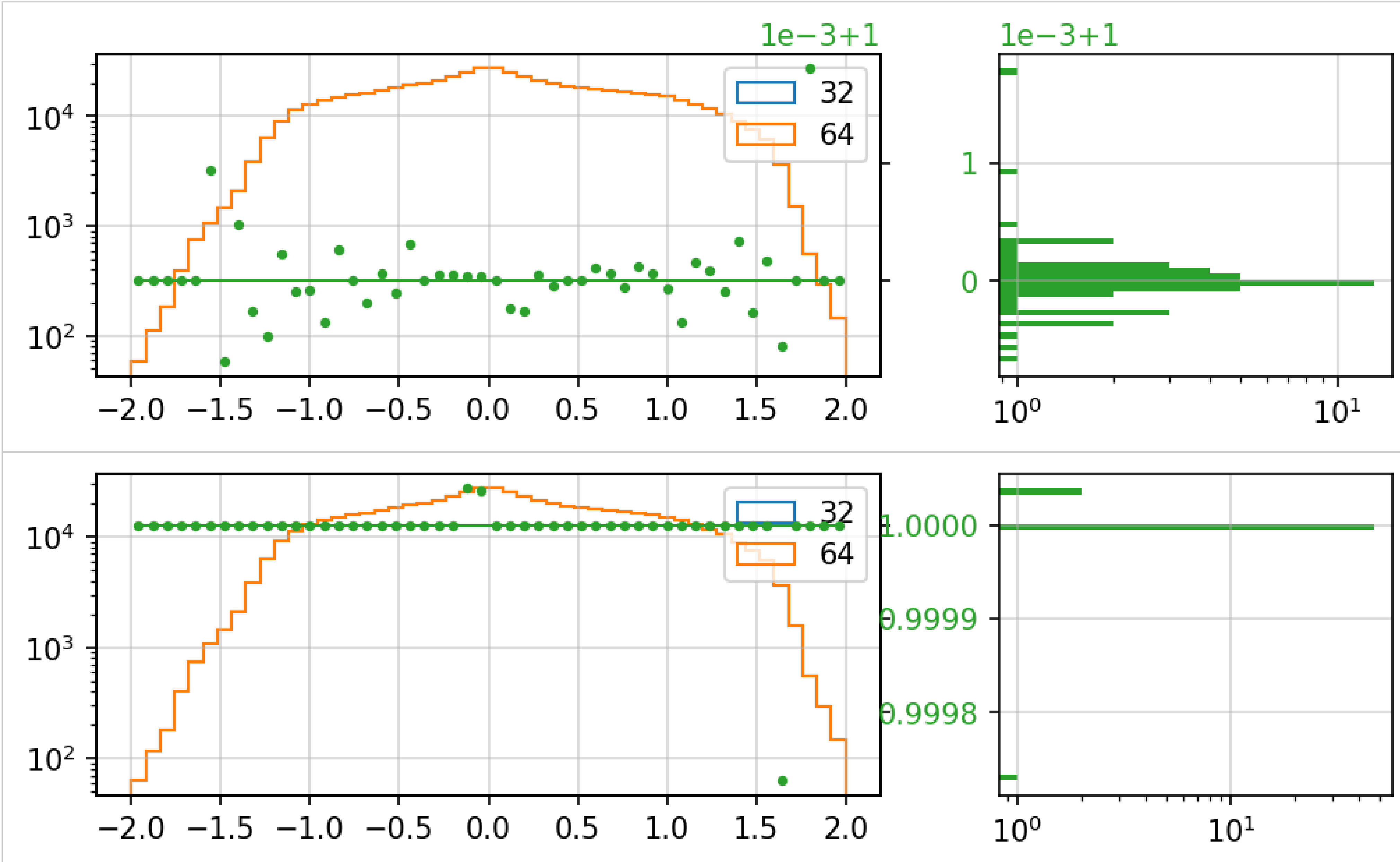
Switching from 32-bit to 64-bit Builds

- Some differences were confirmed by analyzing the assembly code
  - One example: https://gcc.godbolt.org/z/Diy0T8
  - Such rounding differences include cases where result of a calculation is cast to a single precision and then passed to other routines
- As a result all the differences between 32 and 64-bit results are well understood
- In fact, we confirmed that identical results can be produced
  - By fixing the above conversions in the code
  - By forcing SSE instruction set in 32-bit builds
  - Vice versa, switching to FPU in 64-bit is not feasible as it also requires a rebuild of the standard libraries

Switching from 32-bit to 64-bit Builds



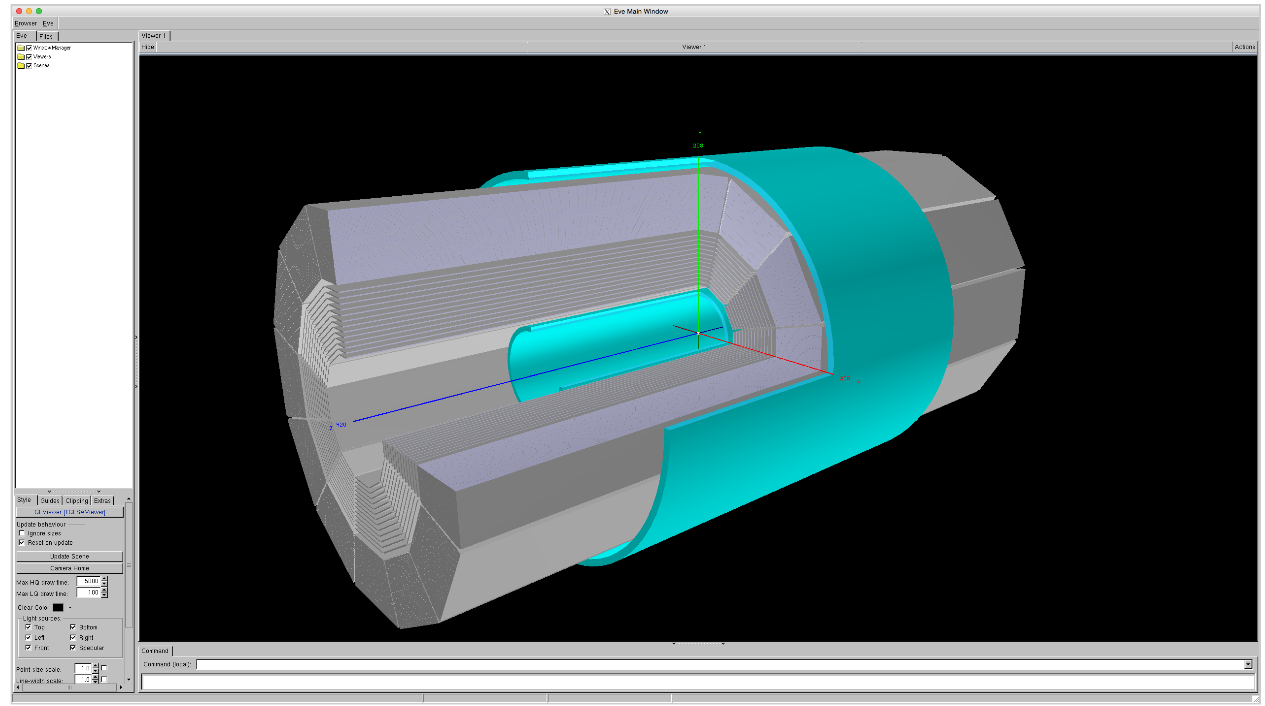
Switching from 32-bit to 64-bit Builds



Upgrade of STAR TPC Detector

- The Time Projection Chamber (TPC) is the primary tracking device of the STAR detector
  - Located inside a large solenoidal magnet that operates at 0.5 T
  - The TPC is 4 m long and 4 m in diameter
  - The paths of primary ionizing particles passing through the gas volume are reconstructed from the released secondary electrons which drift to the readout end caps at the ends of the chamber
  - 13 inner and 32 outer pad rows

- Below image represent TPC geometry as seen by the track reconstruction software

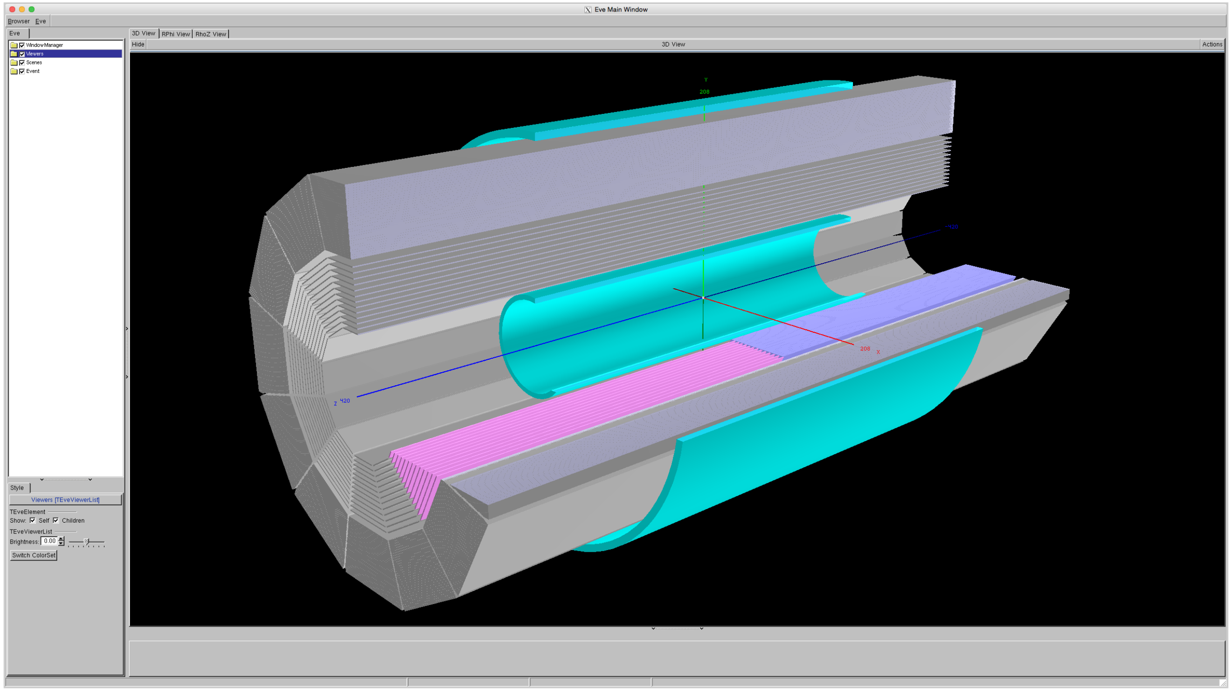


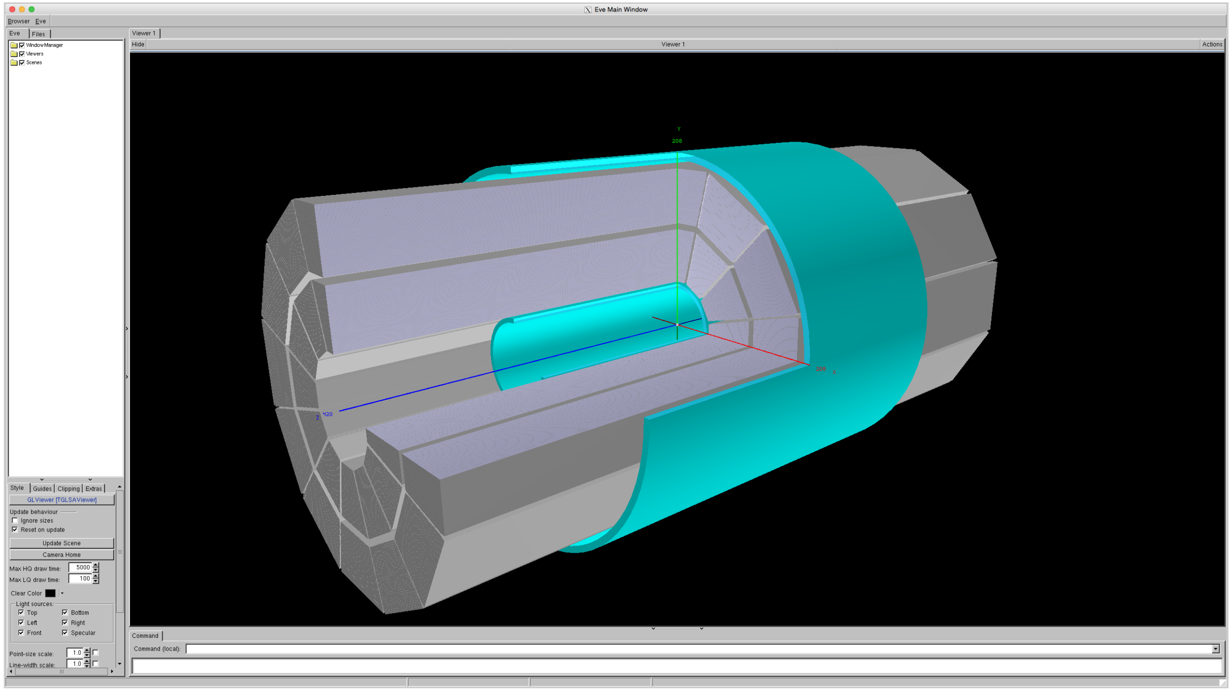
Note: Tracking layers extend beyond the nominal dimension from -200 to +200 cm to accommodate prompt hits

- TPC has undergone major upgrades in 2017 and 2018

Software Updates for TPC Upgrade

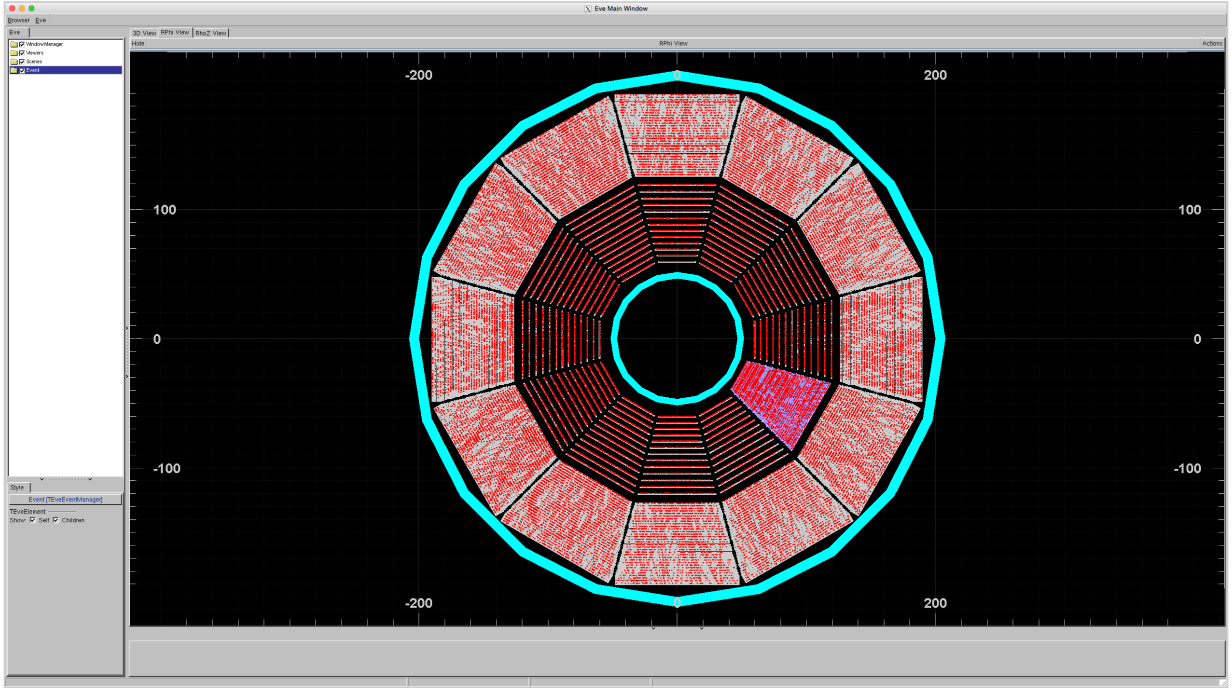
- In 2018 the number of tracking layers in TPC was tripled in a single inner sector
- In 2019 all inner sectors of TPC were upgraded

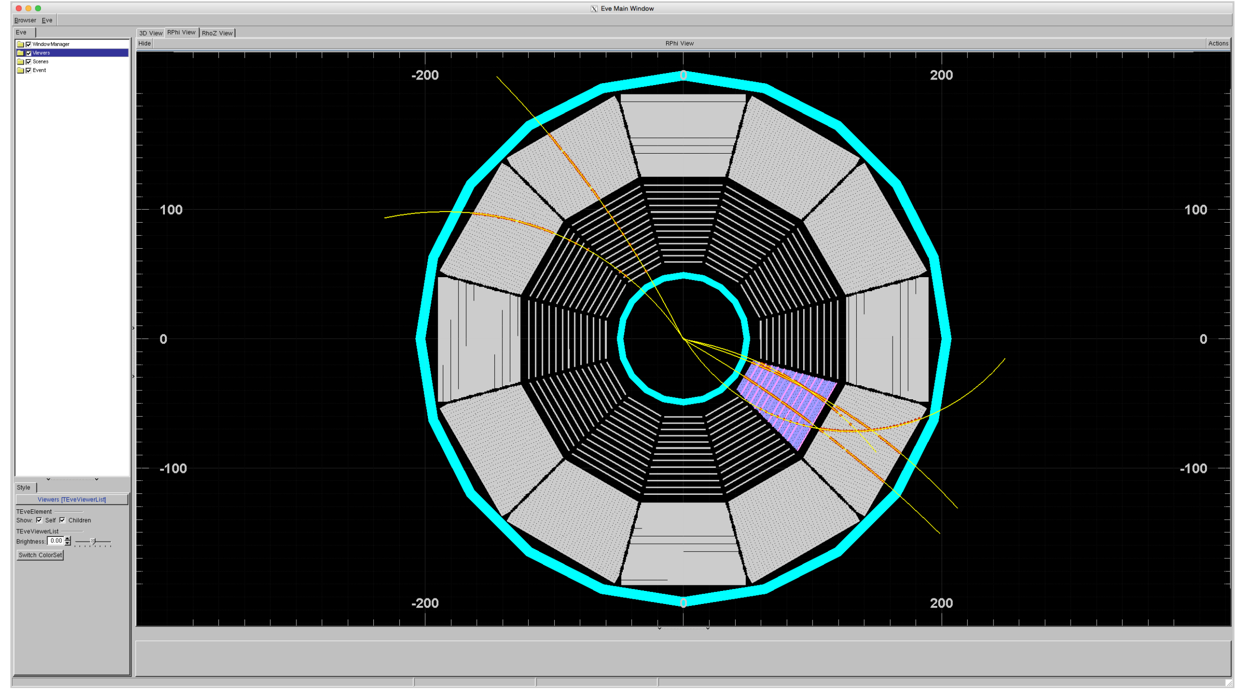


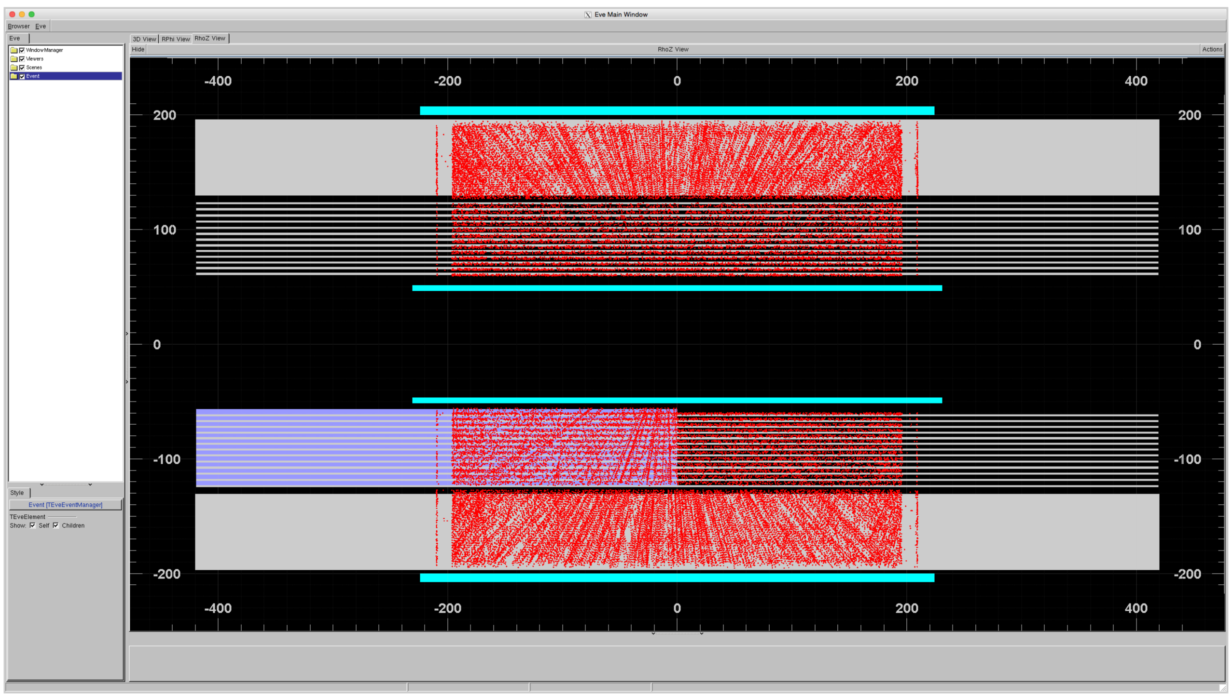


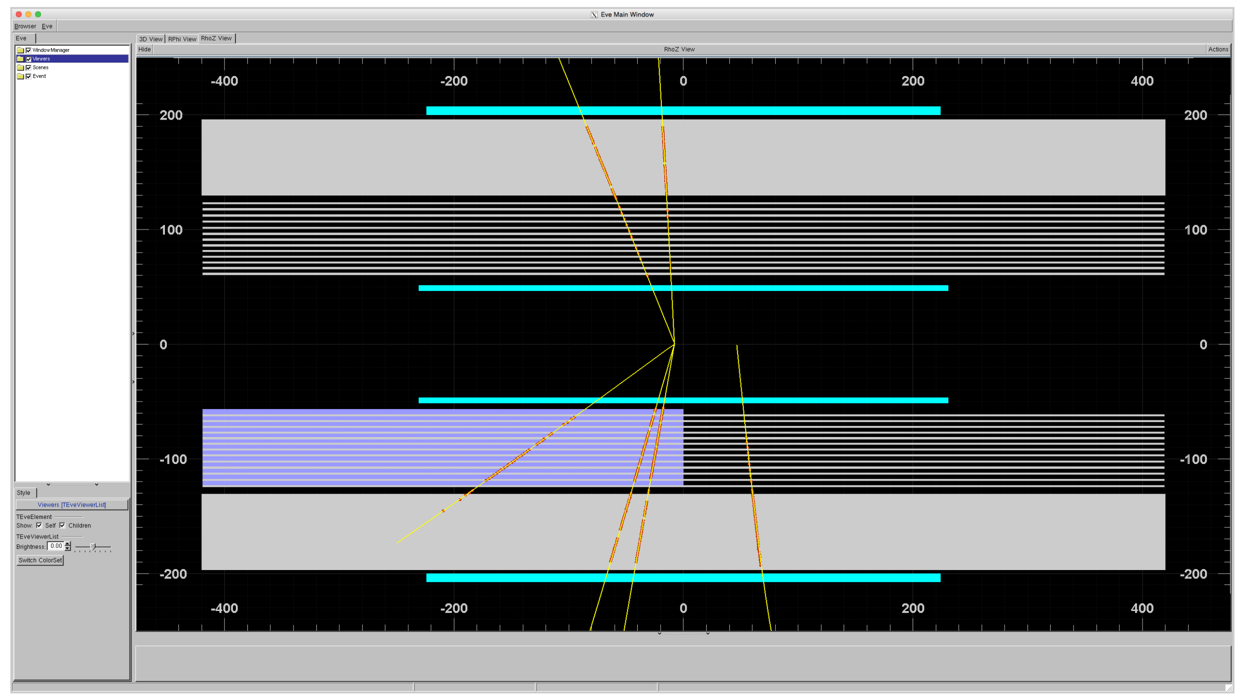
- Full-length layers shown in grey, half-length layers shown in magenta and purple
- Increased number of tracking layers required significant changes in reconstruction software
- We made sure the past results are fully reproducible after introduction of modifications

Reconstruction of Tracks in 2018 Instrumental Run


Eve Main Window
RPhi View
-200
200
100
100
0
0
-100
-100
-200
200


Eve Main Window
RPhi View
-200
200
100
100
0
0
-100
-100
-200
200


Eve Main Window
RhoZ View
-400
-200
0
200
400
200
200
100
100
0
0
-100
-100
-200
-200
-400
-200
0
200
400


Eve Main Window
RhoZ View
-400
-200
0
200
400
200
200
100
100
0
0
-100
-100
-200
-200
-400
-200
0
200
400

STAR `picoDST` Data Format

- In STAR `muDST` (aka μ`DST`, micro DST) is the primary output format used by all physics analyses
  - The format is based on objects of same type packed in ROOT's `TCloneArray`s attached to a common `TTree` as `TBranches`
  - Cross references between entities within event are implemented with raw indices
- With increasing luminosity the need for a more compact format for user analyses become apparent. The `picoDST` was proposed
  - In the initial version some observables were packed/unpacked in 16-bit integer with preset ranges and scale factors
  - There was lots of confusion on how `Double32_t` and `Float16_t` work so, a study with various packing options was carried out to verify the performance and confirm the usage benefits
  - Later a non-S&C maintainer was assigned. Missed opportunities due to no interest (or knowledge?)
    - Step away from `muDST` design by providing a more generic interface to add, activate/deactivate branches (i.e. by hiding raw pointers, utilizing template functions, std::tuples of branches)
    - Support backward compatible schema evolution (IMO, one of the powerful but underused features of IO in ROOT)

uproot

- Using Python visualization and data analysis libraries one quickly realizes the need for a Python reader for files in ROOT format
  - Among the available options `uproot` stands out because it does not depend on C++ ROOT, unlike `PyROOT` and `root_numpy`
- `uproot` is well supported and has become one of the most widely used Python packages made for particle physics with users from LHC experiments, theory, neutrino experiments, XENON-nT, MAGIC, and IceCube
- To be used with STAR files the package was missing the functionality to read `TTree` branches with truncated floating point numbers such as `Float16_t` and `Double32_t`
  - This functionality has been implemented along with the corresponding unit tests so, now STAR users can read files in both `muDST` and `picoDST` formats with `uproot`
  - https://github.com/scikit-hep/uproot