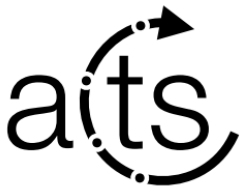
Introduction to algebra-plugins

Beomki Yeo
UC Berkeley



algebra-plugins

- algebra-plugins provides vector and matrix algebras required for track reconstruction
  - Various backends: cmath (home-brew), Eigen, SMatrix

- Users can configure the followings at compile-time:
  - Single or double precision
  - Which backends to use

Backend	CPU	CUDA	SYCL
cmath	✅	✅	✅
Eigen	✅	✅	🟡
SMatrix	✅	⚪	⚪

✅ Natively supported

🟡 Natively supported, but not tested

⚪ No support

Storage backend for vector and matrix

cmath storage

```
/// size type for Array storage model
using size_type = std::size_t;
/// Array type used in the Array storage model
template <typename T, size_type N>
using storage_type = std::array<T, N>;
/// Matrix type used in the Array storage model
template <typename T, size_type ROWS, size_type COLS>
using matrix_type = storage_type<storage_type<T, ROWS>, COLS>;
```

eigen storage

```
/// Eigen array type
template <typename T, int N>
class array : public Eigen::Matrix<T, N, 1, 0, N, 1> {

 public:
  /// Inherit all constructors from the base class
  using Eigen::Matrix<T, N, 1, 0, N, 1>::Matrix;

};  // class array

/// size type for Eigen storage model
using size_type = int;
/// Array type used in the Eigen storage model
template <typename T, size_type N>
using storage_type = array<T, N>;
/// Matrix type used in the Eigen storage model
/// If the number of rows is 1, make it RowMajor
template <typename T, size_type ROWS, size_type COLS>
using matrix_type = Eigen::Matrix<T, ROWS, COLS, (ROWS == 1), ROWS, COLS>;
```

Algorithm backend

- Vector algebra
  - Arithmetic operations
  - cross, dot products
  - normalization
- Matrix algebra
  - Arithmetic operations
  - Transpose
  - Determinant
  - Inversion
- Affine matrix representation (4x4 matrix) for local ↔ global transformation in cartesian coordinate

Frontend

- A frontend is the combination of the storage and algorithm backend
- For example, **eigen_cmath** frontend applies cmath algorithm to eigen storage backend

storage backend

```
// matrix actor
template <typename scalar_t, typename determinant_actor_t,
          typename inverse_actor_t>
using actor = cmath::matrix::actor<size_type, array_type, matrix_type, scalar_t,
                                   determinant_actor_t, inverse_actor_t,
                                   element_getter, block_getter>;
```

algorithm backend

Example usage of matrix algebra

Column-wise cross product

```
// Define matrix operator with cmath backend and single precision
using matrix_operator = cmath::matrix::actor<float>;

// Column-wise cross product between matrix (m) and vector (v)
__host__ __device__
inline matrix_type<3, 3> cross(const matrix_type<3, 3>& m,
                               const vector3& v) const {
    matrix_type<3, 3> ret;

    auto m_col0 = matrix_operator().template block<3, 1>(m, 0, 0);
    auto m_col1 = matrix_operator().template block<3, 1>(m, 0, 1);
    auto m_col2 = matrix_operator().template block<3, 1>(m, 0, 2);

    matrix_operator().set_block(ret, vector::cross(m_col0, v), 0, 0);
    matrix_operator().set_block(ret, vector::cross(m_col1, v), 0, 1);
    matrix_operator().set_block(ret, vector::cross(m_col2, v), 0, 2);

    return ret;
}
```

Remaining Issues

- Add benchmark tools to compare the performance of each frontend
- Add more algorithms
  - Matrix inversion (Currently there is only cofactor method)
- Migrate some of detray algebras into algebra-plugins
- Reimplement vectorization plugin (vc library)