OpenCv datatype (Large Size Objects)

① cv::Mat class

⇒ It is an n-dimensional dense array class.

⇒ The data layout of array M is defined by the array M.step[], so that the address of element $(i_0, \ldots, i_{M.dims-1})$ where $0 \leq i_k < M.size[k]$. is computed as

$$addr(M_{i_0,\ldots i_{M.dims-1}}) = M.data + M.step[0] * i_0 + \cdots + M.step[M.dims-1] * i_{M.dims-1}$$

⇒ For 2-dim array,

$$addr(M_{ij}) = M.data + M.step[0] * i + M.step[1] * j$$

⇒ The data contained in cv::Mat is not required to be simple primitives.

- Each element of the data in a cv::Mat can itself be either a single number or multiple number.
- In the case of multiple numbers, this is what the library refers to as a multichannel array.

⇒ Each matrix contains :-

- flags ⇒ Signaling the contents of the array.
- dims ⇒ Number of dimensions
- rows ⇒ Number of rows
- cols ⇒ Number of columns.

{not valid if dims > 2}

a data pointer to where data is stored
a refcount reference counter.

{ cv::Mat behaves very much like a smart pointer for the data contained in data. }

⇒ Creating an Array

Method 1

```
cv::Mat m;
m.create(row, col, type)
```

↓

CV_{8U, 16S, 16U, 32S, 32F, 64F} C{1, 2, 3}

Method 2

```
cv::Mat m(row, col, type)
```

↑ ③

⇒ OpenCV allows for arrays with more than three channels, but to construct one of these, you will have to call one of the function CV_{8U, 16S, 16U, 32S, 32F, 64F} C( ).

↳ There is no macro for CV_8UC7, to get this you would have to call CV_8UC(7).

Method 3

⇒ Creating a multi-dimensional array

```
int sz[3] = {100, 100, 100}
cv::Mat bigCube(3, sz, CV_8UC1, Scalar::all(0));
```

⇒ Accessing Array Elements Individually

⇒ The basic means of direct access is the (template) member function. at<>()

Example

```
cv::Mat m = cv::Mat::eye(10, 10, CV_32FC2);

m.at<cv::Vec2f>(3, 3)[0];
m.at<cv::Vec2f>(3, 3)[1];
```

② cv::SparseMat

(TODO)