Modern C++ for Computer Vision and Image Processing

Igor Bogoslavskyi

Outline

Generic programming

- Template functions
- Template classes

Iterators

Error handling

Program input parameters

OpenCV

- cv::Mat
- cv::Mat I/O
- SIFT Extraction
- FLANN in OpenCV
- OpenCV with CMake

Generic programming

https://vvvv.org/blog/generic-nodes-project

- **Generic programming:** separate algorithms from the data type
- `Cup` holds any type `T`, e.g. `Coffee` or `Tea`

Template functions

- Generic programming uses keyword `template`

```cpp
template <typename T, typename S>
T awesome_function(const T& var_t, const S& var_s) {
  // some dummy implementation
  T result = var_t;
  return result;
}
```

- `T` and `S` can be any type that is:
  - Copy constructable
  - Assignable
  - Is defined (for custom classes)

Explicit type

If the data type cannot be determined by the compiler, we must define it **ourselves**

```
// Function definition.
template <typename T>
T DummyFuncion() {
  T result;
  return result;
}
// use it in main function
int main(int argc, char const *argv[]) {
  DummyFuncion<int>();
  DummyFuncion<double>();
  return 0;
}
```

Template classes

- Similar syntax to template functions
- Use template type anywhere in class

```cpp
template <class T>
class MyClass {
 public:
  MyClass(const T& smth) : smth_(smth) {}
 private:
  T smth_;
};
int main(int argc, char const* argv[]) {
  MyClass<int> my_object(10);
  MyClass<double> my_double_object(10.0);
  return 0;
}
```

Template specialisation

- We can specialize for a type
- Works for functions and classes alike

```cpp
// Function definition.
template <typename T>
T DummyFuncion() {
  T result;
  return result;
}
template <>
int DummyFuncion() {
  return 42;
}
int main() {
  DummyFuncion<int>();
  DummyFuncion<double>();
  return 0;
}
```

Iterators

STL uses iterators to access data in containers

- Iterators are similar to pointers
- Allow quick navigation through containers
- Most algorithms in STL use iterators
- Access current element with `*iterator`
- Accepts `->` alike to pointers
- Move to next element in container
  `iterator++`
- Compare iterators with `==`, `!=`, `<`

```cpp
#include <iostream>
#include <map>
#include <vector>
using namespace std;
int main() {
  // Vector iterator.
  vector<double> x = {{1, 2, 3}};
  for (auto it = x.begin(); it != x.end(); ++it) {
    cout << *it << endl;
  }
  // Map iterators
  map<int, string> m = {{1, "hello"}, {2, "world"}};
  map<int, string>::iterator m_it = m.find(1);
  cout << m_it->first << ":" << m_it->second << endl;
  if (m.find(3) == m.end()) {
    cout << "Key 3 was not found\n";
  }
  return 0;
}
```

Error handling

- We can “throw” an exception if there is an error
- STL defines classes that represent exceptions. Base class: `exception`
- To use exceptions: `#include <stdexcept>`
- An exception can be “caught” at any point of the program (`try - catch`) and even “thrown” further (`throw`)
- The constructor of an exception receives a string error message as a parameter
- This string can be called through a member function `what()`

`throw` exceptions

Runtime Error:

```
// if there is an error
if (badEvent) {
    string msg = "specific error string";
    // throw error
    throw runtime_error(msg);
}
... some cool code if all ok ...
```

Logic Error: an error in logic of the user

```
throw logic_error(msg);
```

catch exceptions

- If we expect an exception, we can “catch” it
- Use try - catch to catch exceptions

```cpp
try {
   // some code that can throw exceptions z.B.
   x = someUnsafeFunction(a, b, c);
}
// we can catch multiple types of exceptions
catch ( runtime_error &ex )   {
   cerr << "Runtime error: " << ex.what() << endl;
} catch ( logic_error &ex )   {
   cerr << "Logic error: " << ex.what() << endl;
} catch ( exception &ex )   {
   cerr << "Some exception: " << ex.what() << endl;
} catch ( ... ) { // all others
   cerr << "Error: unknown exception" << endl;
}
```

Intuition

- Only used for **"exceptional behavior"**
- **Often misused**: e.g. wrong parameter should not lead to an exception
- GOOGLE-STYLE Don't use exceptions
- http://www.cplusplus.com/reference/exception/

https://google.github.io/styleguide/cppguide.html#Exceptions

Program input parameters

- Originate from the declaration of main function
- Allow passing arguments to the binary
- `int main(int argc, char const *argv[]);`
- `argc` defines number of input parameters
- `argv` is an array of string parameters
- By default:
  - `argc == 1`
  - `argv == "<program_name>"`

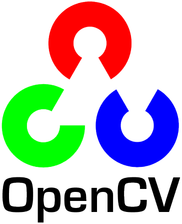
http://en.cppreference.com/w/cpp/language/main_function

Program input parameters

```cpp
#include <iostream>
#include <string>
using namespace std;
int main(int argc, char const *argv[]) {
  cout << "Got " << argc << " params\n";
  string program_name = argv[0];
  cout << "Program: " << program_name << endl;
  for (int i = 1; i < argc; ++i) {
    cout << "Param: " << argv[i] << endl;
  }
  return 0;
}
```

OpenCV

- Popular library for **Image Processing**
- We will be using **version 2** of OpenCV
- We will be using just a small part of it
- `#include <opencv2/core/core.hpp>` to use its core functionality
- Namespace `cv::`
- More here: http://opencv.org/



Basic Matrix Type

- Every image is a `cv::Mat`, for “Matrix”
- `Mat image(rows, cols, DataType, Value);`
- `Mat_<T> image(rows, cols, Value);`
- **I/O**:
  - Read image with `imread`
  - Write image with `imwrite`
  - Show image with `imshow`
  - Defined in `/usr/include/opencv2/highgui/highgui.hpp`

DataType

- OpenCV uses own types
- Names of types follow pattern `CV_<bit_count><itentifier><num_of_channels>`
- **Example**: RGB image is `CV_8UC3`: 8-bit unsigned char with 3 channels for RGB
- **Example**: Grayscale image is `CV_8UC1`: 8-bit unsigned char with 1 channel for intensity
- Better to use `DataType`
- **Example**: `DataType<uint>::type == CV_8UC1`
- Mixing up types in OpenCV is **extremely painful**!

imread

- Read image from file
- `Mat imread(const string& file, int mode=1)`
- Different modes:
  - unchanged: `CV_LOAD_IMAGE_UNCHANGED < 0`
  - 1 channel: `CV_LOAD_IMAGE_GRAYSCALE == 0`
  - 3 channels: `CV_LOAD_IMAGE_COLOR > 0`

```cpp
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;
int main() {
  Mat image = imread("../img/logo_opencv.png",
                     CV_LOAD_IMAGE_GRAYSCALE);
  Mat_<char> = imread("../img/logo_opencv.png",
                      CV_LOAD_IMAGE_GRAYSCALE);
  return 0;
}
```

imwrite

- Write the image to file
- Format is guessed from extension
- `bool imwrite(const string& file,`
`                const Mat& img);`

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;
int main() {
  Mat image = imread("../img/logo_opencv.png",
                     CV_LOAD_IMAGE_COLOR);
  imwrite("copy.jpg", image);
  return 0;
}
```

imshow

- Display the image on screen
- Needs a window to display the image
- `void imshow(const string& window_name,`
  `const Mat& mat)`

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;
int main() {
    Mat image = imread("../img/logo_opencv.png",
                       CV_LOAD_IMAGE_COLOR);
    // Create a window.
    namedWindow("Window name", WINDOW_AUTOSIZE);
    // Show image inside it.
    imshow("Window name", image);
    return 0;
}
```

OpenCV vector type

- OpenCV defines a class `cv::Vec<T, SIZE>`
- Many typedefs available: `Vec3f`, `Vec3b`, etc.
- Used to work with multidimentional images:
  `mat.at<Vec3b>(row, col);`

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace cv;
int main() {
    Mat mat(10, 10, CV_8UC3);
    std::cout << mat.at<Vec3b>(5, 5) << std::endl;
    Mat mat_char(10, 10, CV_8UC1);
    std::cout << mat_char.at<char>(5, 5) << std::endl;
    Mat_<float> matf(10, 10);
    std::cout << matf.at<float>(5, 5) << std::endl;
    Mat_<Vec3f> matf3(10, 10);
    std::cout << matf3.at<Vec3f>(5, 5) << std::endl;
}
```

SIFT Descriptors

- **SIFT**: **S**cale **I**nvariant **F**eature **T**ransform
- **Popular features**: illumination, rotation and translation invariant (to some degree)

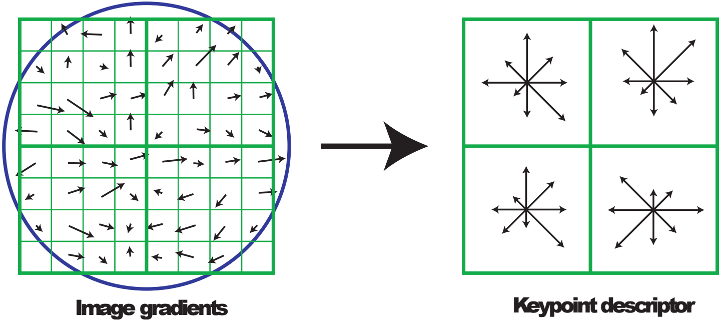


image courtesy of David G. Lowe

SIFT Extraction With OpenCV

- `SiftFeatureDetector` to detect the keypoints
- `SiftDescriptorExtractor` to compute descriptors in keypoints

```
// Detect key points.
SiftFeatureDetector detector;
vector<KeyPoint> keypoints;
detector.detect(input, keypoints);
// Show the keypoints on the image.
Mat image_with_keypoints;
drawKeypoints(input, keypoints, image_with_keypoints);
// Extract the SIFT descriptors.
SiftDescriptorExtractor extractor;
extractor.compute(input, keypoints, descriptors);
```

FLANN in OpenCV

- **FLANN**: **F**ast **L**ibrary for **A**pproximate **N**earest **N**eighbors
- build K-d tree, search for neighbors there

```cpp
// Create a kdtree for searching the data.
cv::flann::KDTreeIndexParams index_params;
cv::flann::Index kdtree(data, index_params);
...
// Search the nearest vector to some query
int k = 1;
Mat nearest_vector_idx(1, k, DataType<int>::type);
Mat nearest_vector_dist(1, k, DataType<float>::type);
kdtree.knnSearch(query, nearest_vector_idx,
                 nearest_vector_dist, k);
```

OpenCV 2 with CMake

- Install OpenCV 2 in the system

```
sudo add-apt-repository ppa:xqms/opencv-nonfree
sudo apt-get update
sudo apt-get install libopencv-nonfree-dev
```

- Find using `find_package(OpenCV 2 REQUIRED)`

```
find_package(OpenCV 2 REQUIRED)
```

- Include `${OpenCV_INCLUDE_DIRS}`
- Link against `${OpenCV_LIBS}`

```
add_library(some_lib some_lib_file.cpp)
target_link_libraries(some_lib ${OpenCV_LIBS})
add_executable(some_program some_file.cpp)
target_link_libraries(some_program ${OpenCV_LIBS})
```

Additional OpenCV information

- We are using **OpenCV version 2**
- Running version 3 will lead to errors
- Example project with additional information about using SIFT and FLANN can be found here:

  https://gitlab.igg.uni-bonn.de/teaching/example_opencv