What is OpenCV?

OpenCV stands for Open-Source Computer Vision (Library). It is the most common and popularly used, well-documented Computer Vision library. OpenCV is an open-source library that incorporates numerous computer vision algorithms. OpenCV increases computational efficiency and assists with real-time applications. One of the major goals of OpenCV is to provide an accessible and easy-to-use computer vision infrastructure that helps people build sophisticated computer vision applications quickly.

Features of OpenCV

Main features of OpenCV-Python include:

1. **Cross-platform**: Allows installation for different environments (operating systems)
2. **Portable**: Transferable to any machine that can run C.
3. **Open source**: OpenCV is free for use under the open-source Apache 2 License.
4. **Fast**: OpenCV is highly optimized and makes use of NumPy functions.
5. **Vast algorithms**: OpenCV packages contain more than 2500 algorithms.
6. **Fast prototyping:** Implemented in the development of real-time applications.
7. **Extensive use**: Used across various organizations and companies.

Stakeholder And the Specific Concerns

User

The people who use openCV to develop machine vision application or study machine vision.

Main Concerns

1. Whether necessary functionality about machine vision is being delivered.
2. Whether openCV is easy to use.
3. Whether openCV is high-performanced.

Project Manager

Responsible for planning, sequencing, scheduling, and allocating resources to develop software components and deliver components to integration and test activities.

Main Concerns

1. Whether openCV can be delivered on time.
2. Whether openCV has efficient image processing capability.

Tester

Creating tests based on the behavior and interaction of the software elements.

Main Concerns

1. Whether openCV is correctly developed.

Context

OpenCV was initiated at Intel in the year 1999 by Gary Bradsky. The first release of OpenCV came later in the year 2000. It was originally developed by Intel employees in a research center in Russia, but the project was taken over by a non-profit organization in 2012. OpenCV supports a multitude of Computer Vision and Machine Learning algorithms that are ever-growing.

In the initial stages of OpenCV, the goals of the company were as follows:

1. They sought to make advanced computer vision more accessible by providing open and optimized code for computer vision implementation.
2. The company aimed to essential open-source computer vision knowledge by providing a common platform that developers could build on, to make the code readily readable and transferable.
3. Deliver advanced commercial applications of computer vision by making portable and optimized code available for development. The OpenCV's license provided the developers with the freedom to choose whether to open-source their product or not.

Quality attribute

1. Usability . OpenCV supports a wide range of programming languages which include C++, Java, Python, etc. This is a cross-platform library that supports Windows, Linux, macOS, Android, and iOS.
2. Performance. OpenCV is a highly optimized library with focus on real-time applications.
3. Scalability. OpenCV now supports many algorithms related to computer vision and machine learning, and it is expanding every day.
4. Cross-platform. OpenCV has C++, Python and Java interfaces support Linux, MacOS, Windows, iOS, and Android.

Key Driver

1. Cross-platform

   The key design idea is to keep pipeline code itself platform-neutral while specifying which kernels to use and which devices to utilize using extra parameters at graph compile (configuration) time.

2. Performance

   It plays a major role in real-time image processing and computer vision tasks which is a necessity for modern applications. OpenCV makes use of NumPy, which is a highly optimized python library for numerical computations. All of the OpenCV array structures are converted to and from NumPy arrays.

3. Usability

   Users can use algorithm in openCV quickly and efficiently through calling function and giving parameters.

Early Decision

1. To make openCV Cross-platform, openCV was written in C and this makes OpenCV portable to almost any commercial system.
2. OpenCV has many component: core, imgproc, object detection, features2d, calib3dstereo, and other modules.
3. To make openCV easier to use, openCV design class MAT to achieve reference counting mechanism to automate memory management.

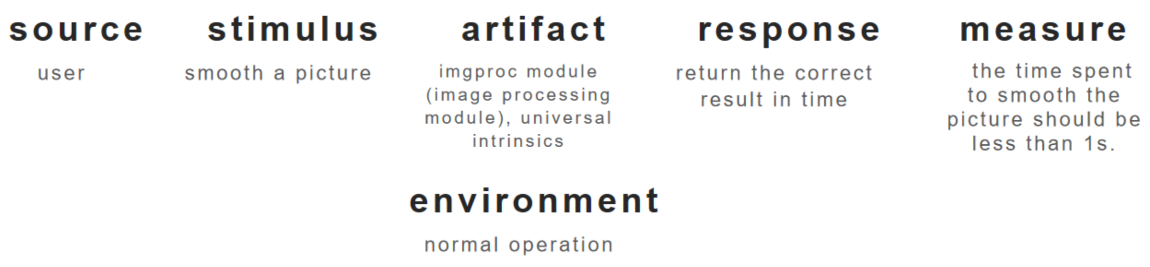
Technical Context

1. Usability. The lack of a computer vision library makes the development process require additional development.
2. Scalability. More and more computer vision algorithms have been proposed, which demands the scalability of the software. In commercial software, due to limitation of the interface, and it is difficult to expand flexibly according to the requirements.

Key Driver Scenario

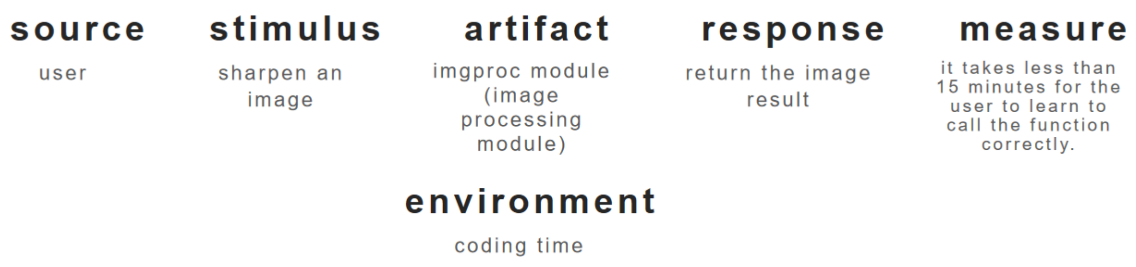
1. Performance

   source: user
   stimulus: smooth a picture
   environment: normal operation
   artifact: imgproc module (image processing module), universal intrinsics
   response: return the correct result in time
   response measure: the time spent to smooth the picture should be less than 1s.


source
user
stimulus
smooth a picture
artifact
imgproc module (image processing module), universal intrinsics
response
return the correct result in time
measure
the time spent to smooth the picture should be less than 1s.
environment
normal operation

2. usability

   source: user
   stimulus: sharpen an image
   artifact: imgproc module (image processing module)
   environment: coding time
   response: return the image result
   response measure: it takes less than 15 minutes for the user to learn to call the function correctly.

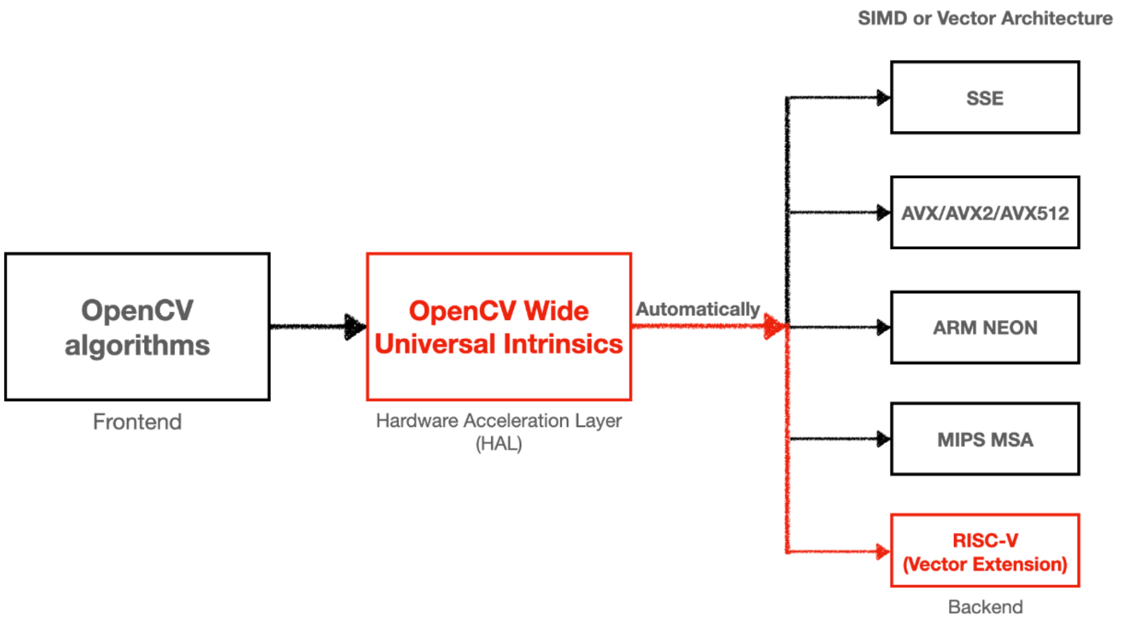

source
user
stimulus
sharpen an image
artifact
imgproc module (image processing module)
response
return the image result
measure
it takes less than 15 minutes for the user to learn to call the function correctly.
environment
coding time

Architecture Tactics

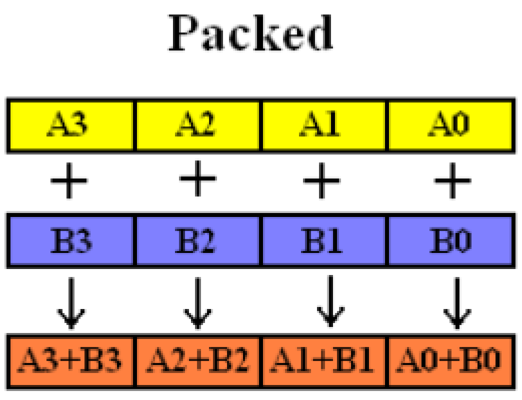
1. Performance and Interoperability : Use concurrent and tailor interface

   OpenCV has designed a uniform set of vector instructions(universal intrinsics), which can mask the differences between the different instruction sets. It allows you to write a piece of code for vector acceleration on different platforms and instruction sets.

OpenCV Universal Intrinsic encapsulates the vectors (vector) of the different SIMD instructions into a unified data structure, overloading the various operators, and describes the vector width as a variable that automatically changes with the compiled environment.

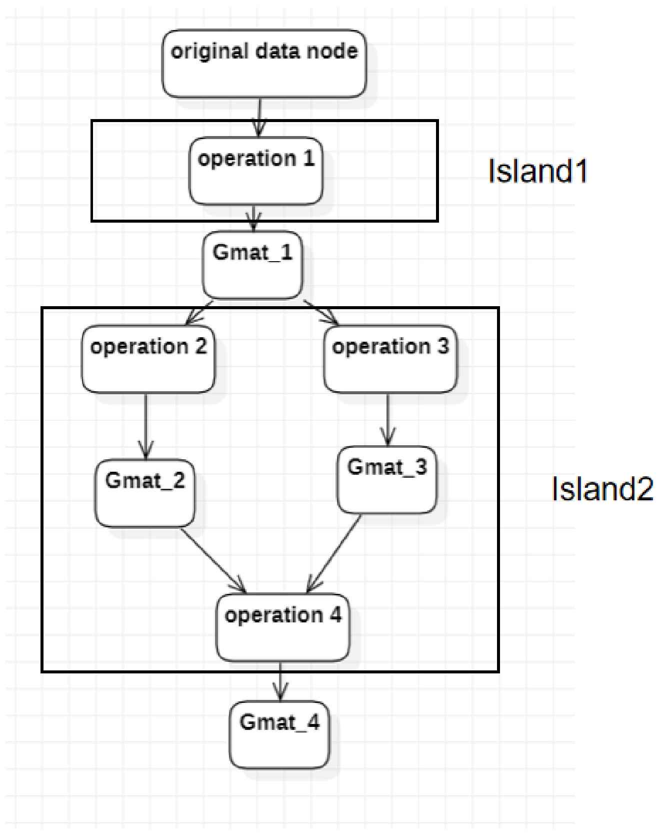

SIMD or Vector Architecture
OpenCV algorithms
Frontend
OpenCV Wide Universal Intrinsics
Hardware Acceleration Layer (HAL)
Automatically
SSE
AVX/AVX2/AVX512
ARM NEON
MIPS MSA
RISC-V (Vector Extension)
Backend

SIMD, aka single instruction multiple data, means that a once-run operational instruction can perform more than one data stream. For example, an addition instruction cycle can only calculate a set of numbers (one-dimensional vector addition). Using SIMD, an addition instruction cycle can calculate multiple data concurrently (n-dimensional vector addition), which greatly improves the operation efficiency.



2. Performance: Reduce Overhead

OpenCV designed class GMat. Instead of storing the actual data, it records what the user does on the GMat and eventually combines multiple GMat to generate a computational graph to handle the real calculation. GMat can provide cross-function optimization that cannot be provided by decentralized functions internally, such as the merger of arithmetic operations, multiplexing of cache and avoid multiple allocation of buffers. For example, in the following figure, logically, the result is output after 4 operations. However, OpenCV can improve performance by cross-function optimization. Depending on the affinity between operations or certain user-specified rules, operations 2 and 3 are incorporated into an 'island', which may reduce memory allocation or merger arithmetic operations.



Architecture Patterns

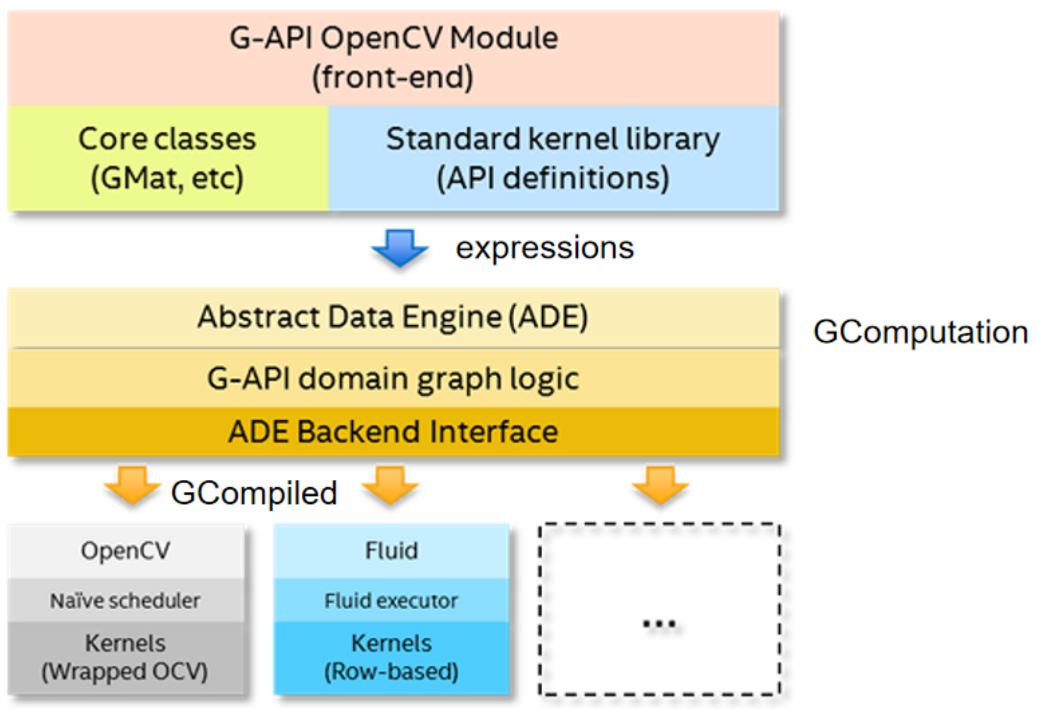
Layer Pattern

There are three layers in GAPI:

- **API Layer** – this is the top layer, which implements G-API public interface, its building blocks and semantics. When user constructs a pipeline with G-API, he interacts with this layer directly, and the entities the user operates on are provided by this layer.
- **Graph Compiler Layer** – this is the intermediate layer which unrolls user computation into a graph and then applies a number of transformations to it. This layer is built atop of ADE Framework.
- **Backends Layer** – this is the lowest level layer, which lists a number of Backends. In contrast with the above two layers, backends are highly coupled with low-level platform details, with every backend standing for every platform. A backend operates on a processed graph (coming from the graph compiler) and executes this graph optimally for a specific platform or device.

The classes used for interaction between layers are as follows:

- **GComputation** class represents a captured computation graph. GComputation objects form boundaries for expression code user writes with G-API, allowing to compile and execute it.
- **GCompiled** represents a compiled computation (graph). Objects of this class actually do data processing, and graph execution is incapsulated into objects of this class. At the same time, two different GCompiled objects produced from the single GComputation are completely independent and can be used concurrently.



The interaction procedure between the layers is as follows:

The user calls the function of the GAPI , which produces the expressions. These expressions will be passed to the Graph Compiler layer.

Initially, a bipartite graph is generated from expressions captured by API layer. This graph contains nodes of two types: *Data* and *Operations*. Graph always starts and ends with a Data node(s), with Operations nodes in-between. Every Operation node has inputs and outputs, both are Data nodes. This bipartite graph is saved by GComputation.

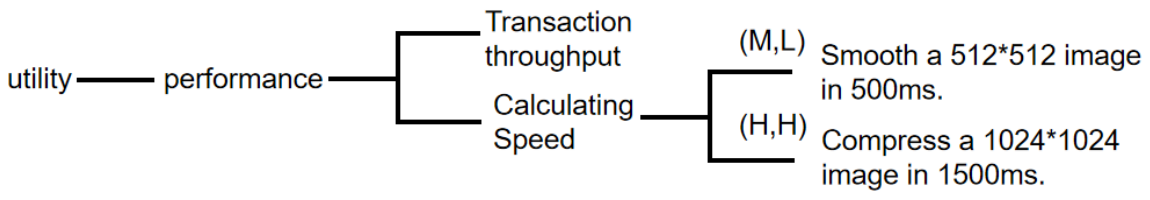
After the initial graph is generated, it is actually processed by a number of graph transformations, called *passes*. ADE Framework acts as a compiler pass management engine, and passes are written specifically for G-API. There are different passes which check graph validity, refine details on operations and data, organize nodes into clusters ("Islands") based on affinity or user-specified regioning and more.

Result of graph compilation is a compiled object, represented by class GCompiled. GCompiled will be executed in Backends layer later.

The way graph executed is defined by backends selected for compilation. In fact, every backend builds its own execution script as the final stage of graph compilation process, when an executable object is being generated.

Architecture Evaluation

Utility Tree


utility
performance
Transaction throughput
Calculating Speed
(M,L)
Smooth a 512*512 image in 500ms.
(H,H)
Compress a 1024*1024 image in 1500ms.

Architecture Approach

Scenario	Compress a 1024*1024 image			
Q-Attribute	Performance			
Environment	Normal operation			
Stimulus	User calls compression function			
Response	Image is compressed in 1500 ms.			
Architectural Decisions	Sensitivity	Tradeoff	Risk	Non-risk
Use Concurrent by SIMD	S1		R1	
Reduce Overhead by GMat	S2			N1
Reasoning	Worst-case estimate			
Arch. model	GMat and universal intrinsics			

Note:

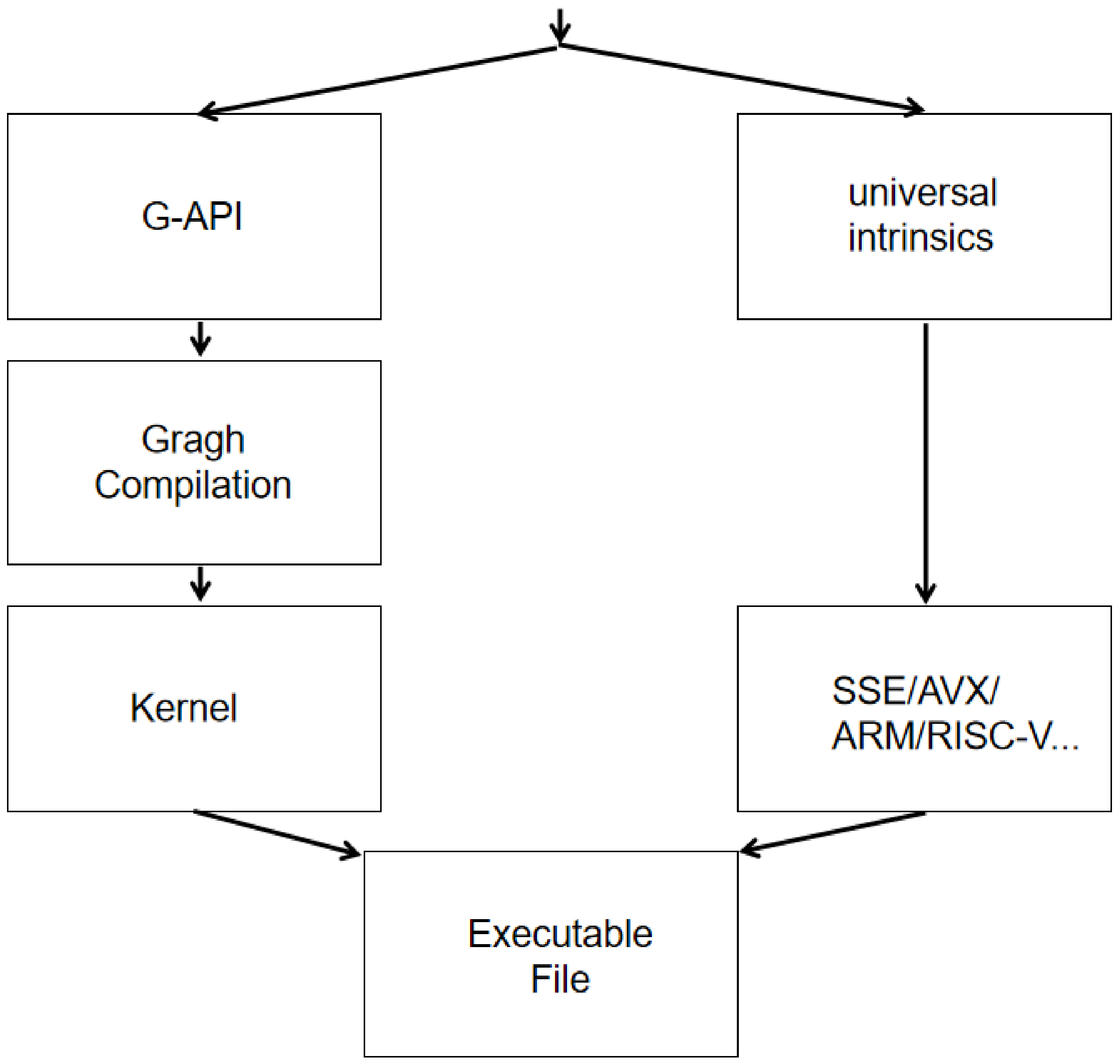
- S1 : SIMD can improve compression speed exponentially, but is highly related to instruction sets.
- R1 : New instruction sets may not be supported by universal intrinsics.
- S2 : GMat can optimize calculating progress, but only functions defined in GMat module can be optimized.
- N1 : GMat is reliable solution.

Reasoning:

- Compression consists of constructing n*n subimages, forwarding transform, quantizer, symbol encoder. Quantizer costs most of time.
- Constructing n*n subimages and forwarding transform is negligible.
- Using SIMD can fasten quantizer , and improve performance highly.

Architectural model:

G-API and universal intrinsics are two separate modules. The following figure shows the process accelerated with two modules.



Division of The Project

Work	Name
Introduction	刘骁，杨俊奇，王鼏宇
Features	刘骁，杨俊奇，王鼏宇
Stakeholder and Concerns	刘骁
Context	王鼏宇
Quality Attribute	刘骁，杨俊奇，王鼏宇
Key Driver	刘骁，杨俊奇
Early Decision	刘骁，杨俊奇，王鼏宇
Technical Context	杨俊奇
Key Driver Scenario	刘骁，杨俊奇
Architecture Tactics	Use concurrent and tailor interface：杨俊奇，Reduce Overhead：刘骁
Architecture Patterns	刘骁，杨俊奇
Architecture Evaluation	刘骁，杨俊奇
PPT	中期报告：刘骁，王鼏宇；终期报告：杨俊奇