@tomoaki_teshima updated at 2020-12-08

OpenCV Advent Calendar 2020 Day 6

关于OpenCV舍入误差bit-exactness

OpenCV

入门

- 今天，我想谈谈OpenCV的bit-exactness。
- 本文是OpenCV 基督降临日历 2020，第 6 天的文章。
- 有关其他文章，请参阅目录。

TL;DR

- 基本上，OpenCV API 不保证 bit-exactness

舍入误差和 bit-exactness

- 通常，浮动小数运算不能用匹配的结果来表示。
- 计算结果通常落在离散表示的浮点数之间。
- 此时，无论是否舍入到任一值，都成为与实数不同结果，因此产生"舍入误差"

- 如果舍入误差足够小，则不需要担心舍入误差。 另外一方面，在舍入误差相对于目标较大情况下，需要正确地考虑舍入误差
- 另一方面，在 OpenCV 中成为热门话题的 bit-exactness 询问，如果平台（如体系结构/CPU/GPU/SIMD 操作）发生变化，是否以位为单位输出相同的值。
- 目前，OpenCV 正在讨论一项改进建议，标题为 OpenCV 进化。 其OE-15是关于OpenCV函数的bit-exactness的提案
- 顺便说一下，bit-exactness 和每次执行的结果模糊的行为是一个完全不同的问题。
  - 很多人经常误解它，但每次你执行它时，结果都会改变，这是一个完全不同的问题。 但是，我仔细观察那些与舍入误差混淆的人。
  - 无论程序是否包含浮点操作，程序的结果都应在执行多次时相同。
  - 与舍入误差不同，这称为 deterministic（确定性）行为。
  - 对于非 deterministic 行为，多线程控制可能会失败或踩踏未初始化的变量。

关于 OpenCV 中的 bit-exact

- 通常，当平台更改时，无法预期 bit-exact 结果。
- 当然，如果仅限于整数操作，则可以预期 bit-exact 的结果。 但是，由于输入和输出是整数（例如CV_8UC3），因此内部不存在浮点操作。
- 如前所述，舍入过程特定于平台，不幸的是，无法保证 bit-exact
- 实际上，从 OpenCV 3.3.1 引入了颜色转换 Lab2RGB，并在 3.4.0 中引入了 resize 中的 bit-exactness。
- 在这里，我将介绍一个案例，其中bit-exactness在检查的范围内是有保证的，有些是有保证的，有些是不能保证的。

在任何情况下都保证 bit-exact

- `transpose`
- `split / merge`
- `flip`
- `reshape`
- 当然，从内存中读取像素值的过程，但无需执行任何处理，即可将像素值存储在单独的排序和内存地址中。

- 无论它们是哪种输入或平台，它们都保证 bit-exactness。
- 反过头来，我想大概只有这五个。

有保证部分bit-exact的情况

- `cvtColor`（`RGB2GRAY` 一些颜色转换，如） `RGB2HSV`
- `imwrite` （取决于输出文件的扩展名，如 png 和 bmp）
- `resize` （指定插值模式或时）

  `cv::INTER_LINEAR_EXACT` `cv::INTER_NEAREST_EXACT`
- `GaussianBlur`
- 这是保证bit-exactness的情况，这是这次的主要重点。
- 积分仅是"在某些情况下有保证"，并且可能不保证，具体取决于输入或模式。
- CPU版本是bit-exact，但OpenCL（UMat）实现不是bit-exact，也有陷阱。

未验证（不保证bit-exact）

- 其他一切都进入这个流派。 基本上，bit-exact 是一个不保证的假设。

bit-exact的陷阱

- 因为我没有太小心，我写了一个陷阱，我迷上了bit-exact是什么。

`imwrite` bit-exact `bmp` `png`

- 当然，在 中保存时，编码过程充满了浮点操作。 `jpeg`
- 要导出 bit-exact 结果，请保存它。 `png` `bmp`

验证 bit-exactness 需要单独的体系结构

- https://github.com/opencv/opencv/pull/10921#pullrequestreview-99294475

> "tolerance = 0" is not enough to guarantee bit-exact results. At least not between platforms.

- 在测试中，"我实现了容差 0 的测试，并通过了 bit-exact！ 不能坚持。
- 你不能声称这是一个位exact，如果你必须尝试两个平台，具有不同的体系结构，如PC和Razpait。

bit-exact的工作原理

- 内容是所谓的定点操作。 这并不难。
- 例如，考虑灰度。
- OpenCV（master）执行以下操作：

$$gray = B \times 0.114 + G \times 0.587 + R \times 0.299$$

```
//constants for conversion from/to RGB and Gray, YUV, YCrCb according to BT.601
static const float B2YF = 0.114f;
static const float G2YF = 0.587f;
static const float R2YF = 0.299f;
```

- 但是，在当前 master 中，这不是浮点运算，而是整数的乘积。

```
num
{
    gray_shift = 15,
    yuv_shift = 14,
    xyz_shift = 12,
    R2Y = 4899, // == R2YF*16384
    G2Y = 9617, // == G2YF*16384
    B2Y = 1868, // == B2YF*16384
    RY15 =  9798, // == R2YF*32768 + 0.5
    GY15 = 19235, // == G2YF*32768 + 0.5
    BY15 =  3735, // == B2YF*32768 + 0.5
```

```
    BLOCK_SIZE = 256
};
```

```
    static const int BY = BY15;
    static const int GY = GY15;
    static const int RY = RY15;
    static const int shift = gray_shift;
    :
        const int coeffs0[] = { RY, GY, BY };
        for(int i = 0; i < 3; i++)
                coeffs[i] = (short)(_coeffs ? _coeffs[i] : coeffs0[i]);
        if(blueIdx == 0)
            std::swap(coeffs[0], coeffs[2]);

        CV_Assert(coeffs[0] + coeffs[1] + coeffs[2] == (1 << shift));
    :
        short cb = coeffs[0], cg = coeffs[1], cr = coeffs[2];
    :
            int b = src[0], g = src[1], r = src[2];
            ushort d = (ushort)CV_DESCALE((unsigned)(b*cb + g*cg + r*cr), shift); // ここ
            dst[0] = d;
```

- 展开最后一个宏时，它看起来像这样： CV_DESCALE

```
#define  CV_DESCALE(x,n)     (((x) + (1 << ((n)-1))) >> (n))
  :
ushort d = (ushort)(((unsigned)(b*cb + g*cg + r*cr) + (1 << ((shift)-1))) >> (shift));
```

- const 值，如下所示：

```
ushort d = (ushort)(((unsigned)(b*3735 + g*19235+ r*9798) + (16384)) >> (15));
```

- 由于最后一个右 15 位移位等效，因此上述过程可以用以下公式表示： / 32768

$$gray = b \times \frac{3735}{32768} + g \times \frac{19235}{32768} + r \times \frac{9798}{32768} + \frac{16384}{32768}$$

- 最后$\frac{16384}{32768} = 0.5$添加，因为四舍五入的操作，重写，以下公式。

$$gray = b \times 0.114 + g \times 0.587 + r \times 0.299$$

- 因此，我们得出了与第一个公式完全相同的形状。

$$gray = B \times 0.114 + G \times 0.587 + R \times 0.299$$

- 什么？ 那么，到目前为止，计算是什么？ 这一点是，代码上的所有计算都是整数类型。

```
ushort d = (ushort)(((unsigned)(b*cb + g*cg + r*cr)                + (1 << ((shift)-1))) >
                            //  ^      ^      ^ 全て整数型の掛け算 ^ 整数型の足し算
```

- 此过程没有浮点操作，并且由于所有操作都是整数类型，因此不会发生任何"舍入误差"。

softfloat

- 与上一节一样，某些部分使用定点操作，但在这种情况下，只有一个约束。 这意味着计算过程中的值区域不会溢出。
- 例如，在颜色转换中使用的系数是固定的，并且输入固定在无符号 8 位上，因此在编译时可以检查不会溢出。
- 另一方面，函数主要使用两个值进行浮点运算：图像大小和缩放百分比，并且两者都没有定义范围。 `resize`
- 定点操作需要固定范围，因此不能继续执行定点操作。
- 在 OpenCV 中实现 `softfloat` 类。
  - 此类在构造函数中接受，并保留 浮点 、`uint` 和 `int` 联合 结构中的值。

    `softfloat` `float` `Cv32suf`

softfloat.hpp

```
/** @brief Construct from float */
explicit softfloat( const float a ) { Cv32suf s; s.f = a; v = s.u; }
```

cvdef.h

```
typedef union Cv32suf
{
    int i;
    unsigned u;
    float f;
}
Cv32suf;
```

- 然后，在以后的运算中，所有操作都疯狂地在~~代码中模拟~~。 float

softfloat.cpp

```
// 浮動小数点演算の加算をエミュレートしたコード（抜粋）
static float32_t softfloat_addMagsF32( uint_fast32_t uiA, uint_fast32_t uiB )
{
    :
    /*------------------------------------------------------------------------
      第1引数と第2引数の符号ビットと指数部を取り出し
    *------------------------------------------------------------------------*/
    expA = expF32UI( uiA );
    sigA = fracF32UI( uiA );
    expB = expF32UI( uiB );
    sigB = fracF32UI( uiB );
    /*------------------------------------------------------------------------
      指数部の差分を計算
    *------------------------------------------------------------------------*/
    expDiff = expA - expB;
    if ( ! expDiff ) {
        /*--------------------------------------------------------------------
          指数部が同じ場合の処理
        *--------------------------------------------------------------------*/
         :
    } else {
        /*--------------------------------------------------------------------
          指数部が異なる場合の処理（＝桁あわせから始める）
        *--------------------------------------------------------------------*/
         :
        if ( expDiff < 0 ) {
            /*----------------------------------------------------------------
              第2引数の絶対値が第1引数より大きい場合
            *----------------------------------------------------------------*/
            :
```

```
        } else {
            /*------------------------------------------------------------------------
              第1引数の絶対値が第2引数より大きい場合
            *------------------------------------------------------------------------*/
            :
        }
         :
    }
    return softfloat_roundPackToF32( signZ, expZ, sigZ );
    /*------------------------------------------------------------------------
     結果がNaNになる場合
    *------------------------------------------------------------------------*/
 propagateNaN:
    uiZ = softfloat_propagateNaNF32UI( uiA, uiB );
 uiZ:
    return float32_t::fromRaw(uiZ);
}
```

- "过去，CPU 上没有浮点算术单元，因此出现了双处理器。 在那之前，我用代码模拟了所有浮点运算，"我在课本上读到过，我听过一些老人们说，但到了2020年，在竞争浮点运算的时代，我敢说，在一个时钟上可以计算多少FLOPs的时代，我敢说，我会在代码中模拟浮点运算!!! x87
  - 顺便说一下，OpenCV 在 2017 年被合并了。 softfloat
- 然而，与赚取大量的FLOPS相比，它更需要进行准确的计算，没有舍入误差，这是可以理解的。

总结

- 当四舍五入误差被卡在里面时，它看到黑暗。
- 我们介绍了一些 API，这些 API 在平台更改时不受舍入误差的影响。
- 我认为有一个漏洞，类型担心舍入误差免费，我认为最好看看OpenCV的实现内部。

补充

- 虽然某些函数保证跨平台的 bit-exactness，但某些参数（如灰度参数）在三个或四个系列中使用略有不同的参数，因此不能保证跨 OpenCV 版本的 bit-exactness。
- 这正是版本之间的差异。

参考 URL

- 我捕获了 OpenCV 日志，并拉出包含 bit-exact 实现的 PR，但可能存在泄漏。
- 将 `resize` 函数作为 bit-exact 的 commit
- 将 `resize` 函数作为 bit-exact 的 PR
- 将 `Lab` 和 `RGB` 颜色空间转换为 bit-exact 的 PR

Aki Teshima @tomoaki_teshima

http://tessy.org/wiki

Follow