EE4FE 13.46

m55362

ADDENDUM: Division-free fastRAHTHaar

Alexandre Zaghetto, Danillo Graziosi, Ali Tabatabai

SONY

Division-free fastRAHT/fastHaar

Division-dependent implementation:

```
fRahtKernel(...)
{
        _wSum = weightLeft + weightRight;

        _b.val = (weightRight << _b.kFracBits/_wSum);
(...)
}
```

$$w_1/(w_0 + w_1)$$

Division-free fastRAHT/fastHaar

Division-free implementation:

```
fRahtKernel(...)
{
        _wSum = weightLeft + weightRight;

        _b.val = divApprox(weightRight << _b.kFracBits, _wSum, 0);
(...)
}
```

$$w_1/(w_0 + w_1)$$

Division-free fastRAHT/fastHaar

Division-dependent implementation:

$$\hat{\Delta} = \sqrt{(\Delta^2(w_0 + w_1))/(w_0 \times w_1)}$$

$$\hat{\Delta} = \Delta \frac{\sqrt{w_0 + w_1}}{\sqrt{w_0 w_1}}$$

```
int64_t adjutedStepSize = isqrt (
        (_stepSize*_stepSize*adjustedNumFactor) / adjustedDenFactor
);
```


Division-free fastRAHT/fastHaar

$$\hat{\Delta} = \Delta \frac{\sqrt{w_0 + w_1}}{\sqrt{w_0 w_1}}$$

```
inline int64_t
divApprox(const int64_t a,...)
{
  assert(abs(a) < (1ull << 46));

(...)

}
```

Division-free implementation:

$$\hat{\Delta} = \sqrt{(\Delta^2 \times ((w_0 + w_1) \times 2^b)/(w_0 \times w_1))/2^b}$$

$$\hat{\Delta} = \Delta \frac{\sqrt{w_0 + w_1}}{\sqrt{w_0 w_1}}$$

```
int64_t adjutedStepSize = isqrt(

        (_stepSize*_stepSize*divApprox(
        adjustedNumFactor << kFracDiv, adjustedDenFactor, 0)) >> kFracDiv

);
```


Division-free fastRAHT/fastHaar

Example:

b = 24
numPoints = 1100000
$w_0 = w_1 = 550000$
QP = 51➔ Δ = 58368

$$\hat{\Delta} = \sqrt{(\Delta^2 \times ((w_0 + w_1) \times 2^b)/(w_0 \times w_1))/2^b}$$

$(w_0 + w_1) \times 2^b$= 1.8454938e+13

$1.8454938e+13/(w_0 \times w_1) = 61$

$(\Delta^2 \times 61) = 207843681667$

$\hat{\Delta} = 207843681667/2^b = 12388$

Division-free fastRAHT/fastHaar

```
//fastRAHT quantization
inline void
Quantizer::quantize(FixedPoint* x,
uint64_t adjustedNumFactor,
uint64_t adjustedDenFactor) const
{

(...)
int kFracDiv = 24;
int64_t adjutedStepSize = isqrt((_stepSize*_stepSize*divApprox(
        adjustedNumFactor << kFracDiv, adjustedDenFactor, 0)) >> kFracDiv);

int64_t offset = adjutedStepSize / 3;

if (x->val < 0)
        x->val = -divApprox((offset-x->val), adjutedStepSize, 0);
else
        x->val = divApprox(offset+x->val), adjutedStepSize, 0);
(...)
}
```

```
inline int64_t
divApprox(const int64_t a,...)
{
  assert(abs(a) < (1ull << 46));

(...)

}
```

$$\hat{\Delta} = \Delta \frac{\sqrt{w_0 + w_1}}{\sqrt{w_0 w_1}}$$

Division-free fastRAHT/fastHaar

```
//fastRAHT scaling
inline void
Quantizer::scale(FixedPoint* x,
uint64_t adjustedNumFactor,
uint64_t adjustedDenFactor) const
{
int kFracDiv = 21;

// quantization step needs to be adjusted according to the weights of input
points
int64_t adjutedStepSize = isqrt((_stepSize*_stepSize*divApprox(
          adjustedNumFactor << kFracDiv, adjustedDenFactor, 0)) >> kFracDiv);

    x->val *= adjutedStepSize;

}
```

```
inline int64_t
divApprox(const int64_t a,...)
{
  assert(abs(a) < (1ull << 46));

(...)

}
```

$$\hat{\Delta} = \Delta \frac{\sqrt{w_0 + w_1}}{\sqrt{w_0 w_1}}$$

