x265 码率控制重要函数

刘小杰(QQ:472249968)

rateEstimateQscale()获取当前 qscale

{

1：更新滑动窗口总复杂度，为加权值，m_movingAvgSum；

2：B帧情况为加权值，跟参考帧的平均qp和距离有关，得出的qp，再相加m_pbOffset / 2 或者m_pbOffset（根据是否为B帧），在根据是否是场景切换，调整qp，得到qScale，重新调整qScale=qScale*overflow（根据bits情况，已经编码和码率情况），返回qScale；

3：非B帧情况下，

A：two Pass

初始q = rce->newQScale;然后根据Bits重新调整q；

B：one Pass

CRF情况下初始qScale，

getQScale内容为

若cuTree启用

q = pow(BASE_FRAME_DURATION / CLIP_DURATION(2 * timescale), 1 - m_param->rc.qCompress);

否则

q = pow(rce->blurredComplexity, 1 - m_param->rc.qCompress);

然后重新调整q /= rateFactor;

若CRF未启用

initialQScale = getQScale(rce, m_wantedBitsWindow / m_cplxrSum);

getQScale内容为

若cuTree启用

q = pow(BASE_FRAME_DURATION / CLIP_DURATION(2 * timescale), 1 - m_param->rc.qCompress);

否则

q = pow(rce->blurredComplexity, 1 - m_param->rc.qCompress);

然后重新调整q /= rateFactor; （rateFactor= m_wantedBitsWindow / m_cplxrSum）

重新调整tunedQScale = tuneAbrQScaleFromFeedback(initialQScale);

主要根据qScale=qScale*overflow（根据bits情况，已经编码和码率情况）

重新调整qScale，根据是否为I帧，GOP情况，scenechanges等。

返回qScale//3：非B帧情况下

}

blurredComplexity 的计算 （非 B 帧的复杂度），m_shortTermCplxSum 和 m_shortTermCplxCount 初始为 0

```
m_shortTermCplxSum *= 0.5;
m_shortTermCplxCount *= 0.5;
m_shortTermCplxSum += m_currentSatd / (CLIP_DURATION(m_frameDuration) / BASE_FRAME_DURATION);
m_shortTermCplxCount++;
rce->blurredComplexity = m_shortTermCplxSum / m_shortTermCplxCount;
```

predictRowsSizeSum(Frame* curFrame, RateControlEntry* rce, double qpVbv, int32_t& encodedBitsSoFar) 返回当前已经编码的bits+未编码Bits估计值

```
{
   encodedBitsSoFar为已经编码的实际比特数（当前帧所有）;
   rowSatdCostSoFar = curEncData.m_rowStat[row].rowSatd;
   uint32_t satdCostForPendingCus = curEncData.m_rowStat[row].satdForVbv -                     rowSatdCostSoFar;
   satdCostForPendingCus >>= X265_DEPTH - 8;
   satdCostForPendingCus >>= X265_DEPTH - 8;
   对于每一行
   如果satdCostForPendingCus  > 0
   {
       double pred_s = predictSize(rce->rowPred[0], qScale, satdCostForPendingCus);
       qScale = x265_qp2qScale(qpVbv);
      {
        非I帧情况下
           统计当前行未编码vbvCost和totalBits
            refRowSatdCost += refEncData.m_cuStat[cuAddr].vbvCost;
            refRowBits += refEncData.m_cuStat[cuAddr].totalBits;
            refRowSatdCost >>= X265_DEPTH - 8;
            refQScale = refEncData.m_rowStat[row].rowQpScale;
       }
       {
         I帧情况
             totalSatdBits += (int32_t)pred_s;
          非I帧情况且qScale >= refQScale
           {
                         if (abs((int32_t)(refRowSatdCost - satdCostForPendingCus)) < (int32_t)satdCostForPendingCus / 2)
                    {
                        double predTotal = refRowBits * satdCostForPendingCus / refRowSatdCost * refQScale / qScale;
                        totalSatdBits += (int32_t)((pred_s + predTotal) * 0.5);
                    }
            }
            P帧情况qScale <refQScale
            {
                  intraCostForPendingCus = curEncData.m_rowStat[row].intraSatdForVbv - curEncData.m_rowStat[row].rowIntraSatd;
                    intraCostForPendingCus >>= X265_DEPTH - 8;
                /* Our QP is lower than the reference! */
                  double pred_intra = predictSize(rce->rowPred[1], qScale, intraCostForPendingCus);
                  totalSatdBits += (int32_t)(pred_intra + pred_s);
            }
            B帧情况
                 totalSatdBits += (int32_t)pred_s;

       }
```

```
    }

     return totalSatdBits + encodedBitsSoFar;
}


rowVbvRateControl(Frame* curFrame, uint32_t row, RateControlEntry* rce, double& qpVbv, uint32_t* m_sliceBaseRow, uint32_t sliceId)
{
  1：updatePredictor(rce->rowPred[0], qScaleVbv, (double)rowSatdCost, encodedBits);//
 主要更新每行的一些参数，这些参数用来估计当前帧的bits
  2：非I帧情况下调用
     updatePredictor(rce->rowPred[1], qScaleVbv, (double)intraRowSatdCost, encodedBits);
  3：根据VBV调整qpVBV，qpMin<qpVBV<qpMax,且估算bits和计划bits相符；
}



rateControlEnd(Frame* curFrame, int64_t bits, RateControlEntry* rce, int *filler)
{
  1：重新计算curEncData.m_avgQpAq；
  2：检查是否出现场景切换，如果出现重新设置m_shortTermCplxSum= rce->lastSatd / (CLIP_DURATION(m_frameDuration) /
     BASE_FRAME_DURATION)和m_shortTermCplxCount = 1;
  3：在X265_RC_CRF模式下，更新curEncData.m_rateFactor;
  4：Abr模式下，
     {
          重新调整bits；
          更新参数m_cplxrSum
                 m_cplxrSum += (bits * x265_qp2qScale(rce->qpaRc) / rce->qRceq) - (rce->rowCplxrSum);//非B帧
                 或
                   m_cplxrSum += (bits * x265_qp2qScale(rce->qpaRc) / (rce->qRceq * fabs(m_param->rc.pbFactor))) -
                       (rce->rowCplxrSum);//B帧
          更新参数
          m_wantedBitsWindow += m_frameDuration * m_bitrate;
          m_totalBits += bits - rce->rowTotalBits;
          m_encodedBits += actualBits;
          curFrame->m_rcData->wantedBitsWindow = m_wantedBitsWindow;
          curFrame->m_rcData->cplxrSum = m_cplxrSum;
          curFrame->m_rcData->totalBits = m_totalBits;
          curFrame->m_rcData->encodedBits = m_encodedBits;

      }
  5：vbv情况下更新参数
         *filler = updateVbv(actualBits, rce);
```

```
        curFrame->m_rcData->bufferFillFinal = m_bufferFillFinal;
        for (int i = 0; i < 4; i++)
        {
            curFrame->m_rcData->coeff[i] = m_pred[i].coeff;
            curFrame->m_rcData->count[i] = m_pred[i].count;
            curFrame->m_rcData->offset[i] = m_pred[i].offset;
        }
}

initPass2()
{
    allCodedBits 用来统计上次编码实际bits
    1：若为CRF模式，否则转2
       diffQp += int (m_rce2Pass[endIndex].qpaRc - m_rce2Pass[endIndex].qpNoVbv);
       当diffQp>1,且（endIndex-startIndex+1>=fps）
        for (int start = endIndex + 1; start <= endIndex + fps && start < m_numEntries; start++)
        {
                        RateControlEntry *rce = &m_rce2Pass[start];
                        targetBits += qScale2bits(rce, x265_qp2qScale(rce->qpNoVbv));
                        expectedBits += qScale2bits(rce, rce->qScale);
         }
       当expectedBits < 0.95 * targetBits重新调整
      调整rce->newQScale，调整VBV调用vbv2Pass

    2：ABR模式下
     调用  analyseABR2Pass

}
vbv2Pass(uint64_t allAvailableBits, int endPos, int startPos)
{
  调整m_rce2Pass[i].newQScale
  和
  m_rce2Pass[i].expectedVbv = m_bufferSize - fills[i];//fill为编码帧i的累计bits和以前加和

}
analyseABR2Pass(uint64_t allAvailableBits)
{
  目标为：调整rce->newQScale，使得bit相符（一个窗口内的bits）
  1：首先估计复杂度
   m_rce2Pass[m_encOrder[i]].blurredComplexity = cplxSum / weightSum;其中cplxSum / weightSum为当前i前后窗口的加权平均。
  2：然后得到qScale
       double q = getQScale(rce, 1.0);（根据复杂度得到）；
  3：然后根据stepMult = allAvailableBits / expectedBits调整rce->newQScale
     rce->newQScale = clipQscale(NULL, rce, blurredQscale[i]);
  4：调整VBV
```

调用vbv2Pass
}