OTIMIZAÇÕES PARA ETAPA DE PRÉ-PROCESSAMENTO NA CODIFICAÇÃO DE VÍDEOS HDR NO PADRÃO HEVC

ALEX BORGES; JOÃO BARTH; LUCIANO AGOSTINI;
BRUNO ZATT; MARCELO PORTO

Universidade Federal de Pelotas – Grupo de Arquiteturas e Circuitos Integrados
{amborges, jhpbarth, agostini, zatt, porto}@inf.ufpel.edu.br

1. INTRODUÇÃO

O *High Dynamic Range* (HDR) é o próximo salto em qualidade de vídeos, pois captura imagens com maior precisão de contraste (LUTHRA et. al., 2014). Contudo, não há hoje tecnologia para codificar (comprimir) vídeos neste formato (LUTHRA et. al., 2014), exigindo-se uma etapa de preparação do vídeo HDR para torná-lo acessível a tecnologia atual, chamada de pré-processamento de vídeos HDR. O pré-processamento é importante, pois busca adaptar o vídeo HDR de forma a manter a sua principal característica, que é o contraste dinâmico, mesmo perdendo mais de 95% da informação original do vídeo (LUTHRA et. al., 2014). Essa perda se dá pois cada bit que é removido do pixel, 50% de informação é perdida, os vídeos HDR são armazenados em 16 bits e é preciso reduzir eles para 8 bits para adaptarem-se aos reprodutores atuais.

Em um trabalho anterior (BORGES et. al., 2015) foi apresentado o fluxo do pré-processador de vídeos HDR e as principais motivações da área de pesquisa. O fluxo pode ser apresentado através dos softwares definidos em Mandel (2014), que são o *CTLRender*, cujo objetivo é realizar a transformação dos pixels, e o *tiff2ydzdx*, que adapta o arquivo utilizado pelos vídeos HDR para o formato de entrada do codificador de vídeos HEVC (BROSS et. al., 2015), já que o *CTLRender* não oferece um arquivo de saida compatível com a entrada do HEVC.

CTLRender é um interpretador de códigos *Color Transform Language* (CTL), onde é inserido a matemática para transformações dos pixels do vídeo HDR (MANDEL et. al., 2014). No código CTL original há muitas operações utilizando constantes, isso se justifica pela precisão dos resultados dos cálculos em tempo de execução, a nível de bits (RUGIERO; LOPES, 2008), que pode sofrer diferenças entre harwares diferentes, inclusive de um número pré-calculado.

Como um vídeo capturado exige muito espaço de armazenamento devido a seu tamanho puro (AGOSTINI, 2007), normalmente a codificação é feito simultaneamente a captura, assim um codificador completo em hardware é importante, e não há no atual estágio da dos vídeos HDR um hardware dedicado para isso, logo, uma arquitetura que trabalhe em conjunto com o codificador HEVC se faz relevante. E existem diversos aspectos a serem resolvidos no software-referência antes de propor uma arquitetura de hardware eficiente para esta etapa de pré-processamento. Um destes aspectos é a redução do uso de constantes no pré-processador, assim como em cálculos matematicamente desnecessários nesse contexto, este trabalho visa apresentar os resultados obtidos com essas simplificações matemáticas no software-referência.

2. METODOLOGIA

Como dito, o *CTLRender* possui diversos cálculos utilizando constantes numéricas a fim de gerar novas constantes numéricas, essas sim usadas sobre os pixels do vídeo. Em nível de hardware, esses cálculos se resolvidos antes da

implementação, resultam num código com equivalência matemática ao original. Nos parágrafos seguintes será abordado o exemplo do `RRT_SAT_MAX`, ela é utilizada na multiplicação pelo pixel a fim de corrigir a sua saturação, já no fim do processo. No entanto, calcular essa matriz exige alguns procedimentos, a começar pela matriz de cromacidade `AP1`, fórmula 1, representando as cores primárias do padrão ACES no espaço de cores CIE XYZ (SMITH; GUILD, 1931).

$$AP1 = \begin{bmatrix} 0.713 & 0.293 \\ 0.165 & 0.830 \\ 0.128 & 0.044 \\ 0.32168 & 0.33767 \end{bmatrix} \quad (1)$$

Dessa cromacidade obtemos uma matriz 4x4 chamada `XYZ_2_AP1_MAT`, conforme mostramos na fórmula 2, com variáveis substituídas pelos valores da cromacidade já apresentada.

$$\begin{aligned} X &= \frac{0.32168 * 1.0}{0.33767} \\ Z &= \frac{(1.0 - 0.32168 - 0.33767) * 1.0}{0.33767} \\ d_1 &= 0.713 * (0.044 - 0.830) \\ d_2 &= 0.128 * (0.830 - 0.293) \\ d_3 &= 0.165 * (0.293 - 0.044) \\ d &= d_1 + d_2 + d_3 \\ Sr_1 &= X * (0.044 - 0.830) \\ Sr_2 &= 0.165 * (1.0 * (0.044 - 1) + 0.044 * (X + Z)) \\ Sr_3 &= 0.128 * (1.0 * (0.830 - 1) + 0.830 * (X + Z)) \end{aligned}$$

$$\begin{aligned} Sr &= \frac{Sr_1 - Sr_2 + Sr_3}{d} \\ Sg_1 &= X * (0.293 - 0.044) \\ Sg_2 &= 0.293 * (Y * (0.044 - 1) + 0.044 * (X + Z)) \\ Sg_3 &= 0.128 * (Y * (0.293 - 1) + 0.293 * (X + Z)) \\ Sg &= \frac{Sg_1 + Sg_2 - Sg_3}{d} \\ Sb_1 &= X * (0.830 - 0.293) \\ Sb_2 &= 0.713 * (Y * (0.830 - 1) + 0.830 * (X + Z)) \\ Sb_3 &= 0.165 * (Y * (0.293 - 1) + 0.293 * (X + Z)) \\ Sb &= \frac{Sb_1 - Sb_2 + Sb_3}{d} \end{aligned}$$

$$XYZ\_2\_AP1\_MAT = \begin{bmatrix} Sr * 0.713 & Sr * 0.293 & Sr * (1 - 0.713 - 0.293) \\ Sg * 0.165 & Sg * 0.830 & Sg * (1 - 0.165 - 0.830) \\ Sb * 0.128 & Sb * 0.044 & Sb * (1 - 0.128 - 0.044) \end{bmatrix} \quad (2)$$

Destes, apenas os elementos da segunda coluna da matiz são usados para compor o vetor `rgb2Y[3]`. Esse vetor é então processado para resultar na matriz alvo, a `RRT_SAT_MAX`, como podemos observar na fórmula 3:

$$\begin{aligned} h &= 1.0 - 0.96 \\ RRT\_SAT\_MAX &= \begin{bmatrix} h * rgb2Y[0] + 0.96 & h * rgb2Y[1] & h * rgb2Y[2] \\ h * rgb2Y[0] & h * rgb2Y[1] + 0.96 & h * rgb2Y[2] \\ h * rgb2Y[0] & h * rgb2Y[1] & h * rgb2Y[2] + 0.96 \end{bmatrix} \end{aligned} \quad (3)$$

No fim do processo, obteremos a matriz desejada, `RRT_SAT_MAX`, conforme apresentada na fórmula 4. Devido ao fator interpretador do CTLRender, esse e outros cálculos são realizados a cada novo quadro do vídeo. Casos de simplificação também podem ser encontrados com as matrizes `AP0_2_AP1_MAT` e `AP1_2_AP0_MAT`e, inclusive, otimizações feitas sobre operações básicas entre constantes também foram realizadas para construção do sofware otimizado.

$$RRT\_SAT\_MAX = \begin{bmatrix} 0.970889 & 0.0108892 & 0.0108892 \\ 0.0269633 & 0.986963 & 0.0269633 \\ 0.00214758 & 0.00214758 & 0.962148 \end{bmatrix} \quad (4)$$

Para testar o impacto dessas alterações no código, foram realizadas duas simulações, uma com o código CTL original e outra com o código CTL otimizado, os vídeos gerados pelo pré-processador foram então codificados pelo HEVC (HM versão 16.2) a fim de comparar os resultados de PSNR e Bitrate. O vídeo HDR utilizado para testes foi o “Tears of Steal” em resolução UHD (*Ultra High Definition*) 4K (BLENDER, 2012), foram utilizados os quadros 100 à 249 da

sequência 04_3E desse vídeo. O *tiffydzdx* e o HEVC foram configurados conforme as recomendações em (MENDEL, 2014), com saida do vídeo para 10 bits.

3. RESULTADOS E DISCUSSÃO

As otimizações realizadas no código CTL foram feitas visando eliminar a quantidade excessiva de cálculos sobre constantes, e elas próprias, a fim de permitir o desenvolvimento de um hardware de pré-processamento sem consumos energéticos, temporais e de área desnecessários. Inicialmente considerando os ganhos obtidos com a simplificação do código, conforme mostra na Tabela 1, é possível observar a quantidade de constantes utilizadas em ambas as versões do código, além da quantidade de operações básicas envolvidas. Obervando esses resultados é possível perceber que há mais de 50% de redução em diversos aspectos do código, sem alterações significativas matematicamente falando. Vale destacar que existe uma redução de 60% no número de constantes, o que implica em uma menor memória dedicada na proposta de uma arquitetura de hardware para o pré-processamento.

TABELA 1. Consumo de recursos entre versões do arquivos CTLs original e otimizado

	Versão Original	Versão Otimizada	Redução (%)
Constantes	*256*	*100*	*60.93*
Adições	*59*	*26*	*55.93*
Subtrações	*39*	*10*	*74.35*
Multiplicações	*93*	*39*	*58.06*
Divisões	*31*	*15*	*51.61*

Para averiguar o impacto dessas otimizações sobre o vídeo, é preciso comparar os resultados com os obtidos pelo fluxo original, ambos podem ser vistos na Tabela 2 Nessa tabela, podemos observar os resultados obtidos no HEVC com diferentes parametros de quantização (QP), função que regula a qualidade de saída do vídeo e quanto maior o seu valor, menor é a qualidade, e em ambos os casos, tanto código original como otimizado, mostraram poucas diferenças em PSNR e Bitrate, mostrando que ambos códigos são equivalentes. A maior diferença está nos resultados de crominância, devido a extensão do HEVC para vídeos com alta profundidade de bits somente trabalhar com vídeos subamostrados em 4:2:0, ou seja, apenas um quarto das informações originais das camadas de cor são armazenadas (AGOSTINI, 2007), esse também é o motivo das melhorias alcançadas em destaque na Tabela 2.

TABELA 2. Resultados da codificação de vídeo HDR com as duas versões do pré-processador

	QP				QP			
	22	27	32	37	22	27	32	37
	Código Original				Código Otimizado			
Y_PSNR (db)	*42.9338*	*42.5386*	*41.8828*	*40.8340*	*42.9338*	*42.5384*	***41.8830***	*40.8340*
U_PSNR (db)	*48.6482*	*47.8513*	*47.0016*	*46.2023*	*48.6477*	*47.8521*	*47.0010*	*46.2023*
V_PSNR (db)	*51.5706*	*49.9035*	*48.2415*	*46.8729*	*51.5725*	*49.9063*	*48.2381*	*46.8729*
Bitrate (bits/sec)	*16,161.7331*	*9,057.4285*	*5,236.3098*	*3,044.7475*	*16,163.9066*	***9,055.9565***	***5,235.2627***	*3,044.7475*

4. CONCLUSÕES

Neste artigo foram apresentadas otimizações na etapa de pré-processametode vídeos HDR para a compressão segundo o padrão HEVC, com o

intuito de propiciar o desenvolvimento de uma arquitetura de hardware eficiente para este processo. O foco das otimizações foi a redução do número de constantes, pois o uso excessivo de constantes, bem como de operações sobre elas, geram um hardware menos eficiente. No trabalho foi demonstrado um exemplo de otimização utilizando como base a matriz `RRT_SAT_MAX`, mostrando o processo exigido para a criação dessa matriz e seu resultado final. Outras otimizações básicas foram feitas nos códigos do *CTLRender*, visando sempre eliminar o uso excessivo de constantes, isso resultou em um código contendo apenas operações envolvendo variáveis, e essas otimizações não impactaram o resultado objetivo da codificação de um vídeo HDR usando o HEVC, além disso foi possível alcançar uma redução de 60% em uso de constantes, o que implica em uma menor memória para armazenar esses valores. Também foi obtida uma redução de pelo menos 50% no número de operações matemáticas simples. Essas otimizações irão permitir a construção de um hardware com a mesma eficiência do software referência para pré-processamento e com menores custos de área e consumo de energia em relação a implementação original.

5. REFERÊNCIAS BIBLIOGRÁFICAS

AGOSTINI, L. V. **Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas à Compressão de Vídeo Segundo o Padrão H.264/AVC**. Porto Alegre: Universidade Federal do Rio Grande do Sul, 2007. Disponível em http://hdl.handle.net/10183/12425.

BLENDER. **Mango Project**: Tears of Steel. 2012. Disponível em https://mango.blender.org.

BORGES, A.; BARTH, J.; ZATT, B.; AGOSTINI, L.; PORTO, M. Hardware Architecture Proposal for the High Dynamic Range Video Pre-Processing. In: SOUTH MICRO-ELECTRONICS SYMPOSIUM, 30, Santa Maria, 2015. **Anais...** Santa Maria: EMICRO/SIM, 2015.

BROSS, B.; HAN, W.-J.; OHM, J.R.; SULLIVAN, G. J.; WANG, Y.K.; WIEGAND, T. **High Efficiency Video Coding (HEVC) text specification draft 10 (for FDIS & Consent)**. 2014. Disponível em http://phenix.it-sudparis.eu/jct/doc_end_user/current_document.php?id=7243.

LUTHRA, A.; FRANÇOIS, E.; HUSAK, W. **Draft Requirements and Explorations for HDR/WCG Content Distribution and Storage**. Disponível em http://mpeg.chiariglione.org/standards/exploration/high-dynamic-range-and-wide-colour-gamut-content-distribution/n14510-draft.

MANDEL, B.; FOGG, C.; HELMAN, J. High Dynamic Range / Wide Color Gamut workflow. **JCT-VC Meeting Valencia**, Valencia, Abril 2014. Disponível em http://phenix.int-evry.fr/jct/doc end user/current document.php?id=8888.

RUGGIERO, M. A.; LOPES, V. L. d. R. **Cálculo Numérico**: Aspectos Teóricos e Computacionais. Segunda ed.. São Paulo: Pearson - Makron Books, 2008. ISBN: 9788534602044.

SMITH, T.; GUILD, J. The C.I.E. Colorimetric standards and their use. **Transactions of the Optic Society**, [S.l.], v.33, n.3, p.73, 1931. DOI: 10.1088/1475-4878/33/3/301.