Um Estudo de Aplicação do iPACKMAN para Compressão de Texturas

Bruno Marques **Rafael Santos**,
Centro de Informática, UFPE
E-mail: bfm@cin.ufpe.br, rbcs@cin.ufpe.br

Marcelo Walter **Marcília Campos**
Centro de Informática, UFPE
E-mail: marcelow@cin.ufpe.br, mac@cin.ufpe.br

Resumo: Neste trabalho apresentamos uma investigação do algoritmo iPACKMAN para compressão de imagens, propondo alterações que o tornam mais simples de ser implementado em hardware e agilizam a compressão das imagens. Os nossos resultados experimentais demonstram que, para um grupo específico de imagens muito utilizadas em aplicações de computação gráfica que são as texturas, a nossa solução mais simples não compromete significativamente a qualidade da compressão, sendo portanto uma alternativa viável e mais eficiente para a tarefa de compressão.

Palavras-chave: Texturas, Compresão de Imagens, Computação Gráfica.

1. Introdução

Compressão de imagens, particularmente para dispositivos móveis com recursos limitados como celulares, torna-se uma necessidade que não pode ser ignorada. Uma solução atraente para estes casos é o armazenamento das imagens em formato comprimido, economizando memória, portanto, no momento de utilização a imagem é descomprimida conforme necessário. Esta solução economiza tanto espaço em memória quanto largura de banda na comunicação entre a memória e a CPU, pois a transferência se dá com imagem comprimida [5, 6].

A principal utilização de imagens em tarefas de computação gráfica recai sobre texturas. Texturas diferem de imagens em geral por duas propriedades: localidade e estacionaridade [3]. Para entender estas propriedades pode-se sugerir o seguinte. Considerando uma janela retangular pequena e móvel sobre uma imagem qualquer, ao examinarmos o conteúdo visível desta janela, observamos que para texturas, diferentes janelas apresentam conteúdo similar, enquanto para imagens em geral isto não é verdadeiro. Esta é a propriedade estacionaridade. A propriedade de localidade explica que, para texturas em geral, conseguimos estabelecer alguma relação entre um determinado pixel da textura e a uma vizinhança pequena.

Texturas são muito utilizadas em computação gráfica há bastante tempo [4], pois permitem o aumento do realismo visual sem o custo computacional de modelar cada detalhe. Captura-se do mundo real o efeito visual desejado e utiliza-se esta imagem nas aplicações.

Neste trabalho, propomos uma modificação no algoritmo para compressão de texturas iPACKMAN recentemente proposto por Strom [1]. Mostramos que, com uma pequena modificação do algoritmo conseguimos simplificar o mesmo sem comprometer a qualidade da compressão para texturas.

2. Trabalhos anteriores

Nessa seção apresentamos os trabalhos já realizados e diretamente relacionados com a nossa proposta.

2.1 PACKMAN

O iPACKMAN é uma modificação do PACKMAN e por esta razão apresentamos inicialmente o PACKMAN. Ele é um algoritmo de compressão de imagens criado por J. Ström e T. Akenine-Möller [1], visando uma baixa complexidade para que possa ser implementado em hardware. Nesse algoritmo, a imagem é dividida em blocos de 2 x 4 pixels, onde cada bloco é representado por 32 bits. Para cada bloco é armazenada uma cor base média de 12 bits, com 4 bits para componente R, G e B (abreviado $RGB444$), e os 20 bits restantes modulam a luminescência para cada pixel do bloco. A idéia básica da técnica é separar a informação de cor da informação de intensidade dos pixels. Em seguida, é escolhido um valor constante, de uma tabela, e essa constante é somada a cada componente da cor base. Para cada pixel devemos selecionar uma constante da tabela, por isso precisamos de 2 bits por pixel do bloco, o que nos dá um total de 16 bits. Os 4 bits restantes são utilizados para indexar uma *codeword* dentre as 16 que formam o *codebook*. A Tabela 1 mostra a primeira metade do *codebook* do PACKMAN, a segunda metade é obtida multiplicando-se cada vetor da primeira metade por 2.

Na descompressão, a cor base é expandida, de 12 para 24 bits ($RGB888$), e o valor constante

Tabela 1: Primeira metade do *codebook* do PACKMAN

cw	0	1	2	3	4	5	6	7
	-8	-12	-31	-34	-50	-47	-80	-127
	-2	-4	-6	-12	-8	-19	-28	-42
	2	4	6	12	8	19	80	42
	8	12	31	34	50	47	28	127

selecionado na tabela é adicionado à cor base expandida. Finalmente, os valores das componentes de cor devem ser truncados ao intervalo [0,255].

2.2 iPACKMAN

Em imagens onde a crominância não varia suavemente de um bloco para outro, o PACKMAN não apresenta um resultado muito satisfatório, pois mesmo a menor variação de crominância significa um intervalo muito grande, quando se usa uma representação de 12 bits. O iPACKMAN (improved PACKMAN) surgiu na tentativa de resolver esse problema. Ao invés de codificar uma cor base para cada bloco de 2 x 4, são agrupados 2 blocos, formando um bloco de 4 x 4, e a cor base desses blocos é calculada de acordo com a diferença entre eles. Para justificar esta modificação, foram realizados testes com um conjunto de 20 imagens, que mostraram que a grande maioria, cerca de 88% dos blocos, podem ser codificados diferencialmente usando 3 bits por componente de cor. Para isso, é calculada a cor média para os blocos 2 x 4 adjacentes e ambas são quantizadas para RGB555, melhorando a representação da crominância. Como em alguns casos os blocos não podem ser codificados dessa forma, foi inserido um bit para decidir se vai ser usada a codificação diferencial ou não, a esse bit daremos o nome de *diffbit*. Esse bit foi extraído do *codebook*, que antes tinha 4 bits (16 *codewords*), que passa a ter 3 bits (8 *codewords*). Dessa forma teremos um bit sobrando, que será utilizado para decidir se os blocos serão 2 x 4 ou 4 x 2, chamaremos esse bit de *flipbit*. O *codebook* do iPACKMAN é mostrado na Tabela 2.

Tabela 2: *Codebook* do iPACKMAN

cw	0	1	2	3	4	5	6	7
	-8	-17	-29	-42	-60	-80	-106	-183
	-2	-5	-9	-13	-18	-24	-33	-47
	2	5	9	13	18	24	33	47
	8	17	29	42	60	80	106	183

A descompressão funciona de maneira análoga ao PACKMAN, porém são levadas em consideração o *diffbit* e o *flipbit*. Tanto o codebook do PACKMAN quanto o codebook do iPACKMAN foram obtidos utilizando o algoritmo LBG para minimização do erro para o conjunto de 20 imagens teste já mencionado [7].

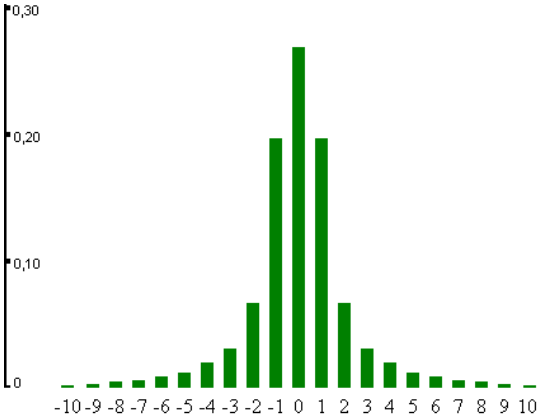

Figura 1: De acordo com o histograma 88% dos blocos estão entre -4 e 3, portanto, nesses casos o iPACKMAN é utilizado.

3. Uma nova abordagem sobre o iPACKMAN

Nesta seção apresentamos a fundamentação e os detalhes das modificações propostas.

3.1 Fundamentação

O iPACKMAN resolveu um importante problema do PACKMAN que era o pequeno espaço para armazenar a cor base dos blocos, dando-lhes um bit a mais, em cada componente de cor RGB, para realizar esta tarefa. Porém criou uma outra inconveniência: a resolução de luminescência dos blocos tornou-se baixa se comparada à do PACKMAN. Isto porque cada sub-bloco pode, no caso do iPackman, endereçar apenas 8 *codewords* possíveis (aquela que minimiza seu erro) no *codebook*.

Quando quisemos duplicar a quantidade de *codewords* possíveis na *codebook* para aumentar a qualidade da luminescência, passamos a necessitar de 2 bits a mais em cada bloco, um para cada sub-bloco. Para manter a taxa de compressão em 2 bpp, precisamos suprimir outros 2 pixels no bloco comprimido. A hipótese de J. Strom [1] na proposta do iPACKMAN era de que a média de um componente de cor de um sub-bloco poderia ser escrita como a soma da média do componente do sub-bloco adjacente com um número de 3 bits. Para mostrar isto ele testou um conjunto de imagens e obteve o histograma da Figura 1. Este histograma nos mostra que o iPACKMAN é usado em, aproximadamente, 88% dos blocos. Assim, poderíamos excluir este flag e utilizarmos o iPACKMAN em todos os casos, pois mesmo que os erros sejam grandes, eles aparecerão em apenas 12% dos casos.

Podemos mostrar ainda que o flag *flipbit* não é tão relevante quanto parece executando o algoritmo duas vezes: fixando todos os *flipbits* em '1' e depois

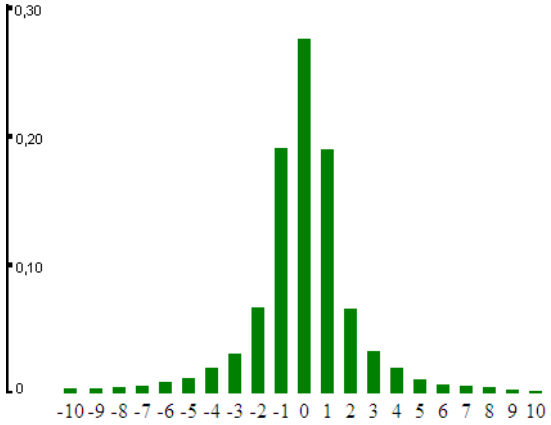

Figura 2: 87% dos blocos estão entre -4 e 3.

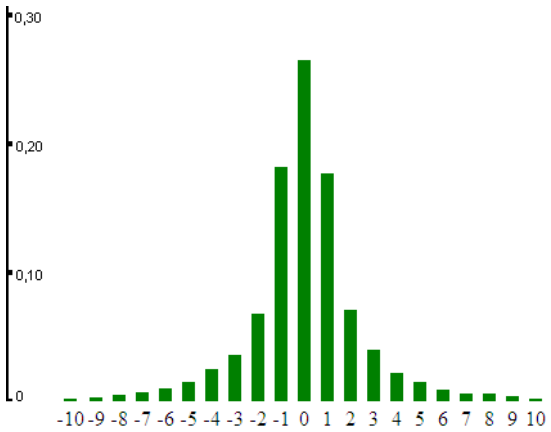

Figura 3: 89% dos blocos estão entre -4 e 3.

fixando todos os *flipbits* em '0', e comparando os resultados. Desta maneira teríamos que supor um determinado estado para o *flipbit*, que em tempo de compressão, seria extendida a todos os blocos da imagem. Para fazer isto testamos a hipótese do iPACKMAN em ambos os casos.

O histograma da Figura 2 representa o referido teste fixando o *flipbit* em '1'. O histograma da Figura 3 representa o mesmo teste fixando o *flipbit* em '0'. Os resultados são semelhantes, o que nos leva à conclusão de que a influência do *flipbit* pode ser considerada pequena e portanto este bit poderia ter outra utilização.

Embora a extinção destes flags incorra em algum erro, este pode ser um bom preço a se pagar por mais qualidade de luminescência e também simplicidade e eficiência na descompressão.

3.2 Descompressão direta

A descompressão no iPACKMAN pode ser implementada com facilidade em hardware, o algoritmo foi implementado com esta preocupação de projeto. Porém, podemos simplificar ainda mais a lógica computacional fazendo com que o algoritmo tome uma única seqüência possível de passos. A proposta é que com a supressão dos flags *flipbit* e *diffbit*, o descompressor terá apenas o trabalho de somar os campos de cor aos campos de luminescência adequados, economizando toda a lógica adicional do iPACKMAN.

A compressão consiste em encontrar a *codeword* do *codebook* que melhor representa o bloco de 2 x 4 pixels e a coordenada da *codeword* que melhor representa cada pixel. Isto é feito para cada um dos blocos da imagem. A mensuração de erro adotada é utilizada para decidir qual *codeword* melhor representa um dado pixel.

Tanto no PACKMAN quanto no iPACKMAN a compressão é feita de forma exaustiva, ou seja, são realizadas 2^{11} (2^3 *codewords* x 2^2 coordenadas x 2^2 modificadores x 2^4 pixels) interações para cada bloco de 4 x 4 pixels em uma imagem. O número de interações seria reduzido a metade em nossa forma alternativa de implementação ($2^{10} = 2^4$ *codewords* x 2^2 coordenadas x 2^4 pixels).

4. Resultados

Nessa sessão apresentamos os resultados experimentais obtidos utilizando imagens e métricas de erro.

Para o iPACKMAN, os resultados foram expressos em *Peak Signal to Noise Ratio* (PSNR) :

$$PSNR = 10\log_{10}(\tfrac{3\times 255^3}{RMSE^2}) \ ,$$

onde RSME, *root mean square error*, é a métrica usada por Fenney [2] para expressar os seus resultados:

$$RSME = \sqrt{\tfrac{1}{w\times h}\sum_{x,y} \Delta R_{xy}^2 + \Delta G_{xy}^2 + \Delta B_{xy}^2} \ ,$$

sendo w e h a largura e altura da imagem respectivamente e ΔR_{xy}, ΔG_{xy} e ΔB_{xy} as diferenças do pixel (x,y) entre a imagem original e a comprimida para o respectivo componente de cor.

Nos nossos testes, utilizamos o iPACKMAN original e a nossa versão para várias coleções de imagens e comparamos os resultados visuais e os respectivos PSNR's . Nestes testes consistentemente a nossa proposta apresentou resultados visuais e quantitativos satisfatórios para uma coleção de imagens de texturas. Ou seja, de todas as coleções de imagens que executamos nossos testes, as texturas apresentaram resultados satisfatórios.

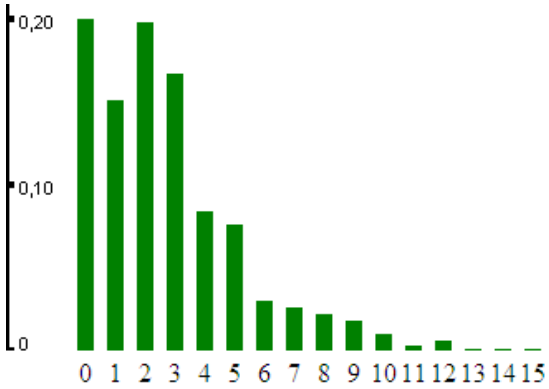

Figura 4: Uso das codewords nas texturas.

Essa diferença em favor das texturas é justificável porque, em geral para outros tipos de imagens, o uso do *codebook* está concentrado no início da tabela, que são os índices de menor valor. Os histogramas da Figura 4 e da Figura 5 mostram a distribuição dos índices usados na compressão de texturas e de outros tipos de imagens, respectivamente. Analisando os histogramas percebemos que para as texturas, a distribuição do uso dos índices é mais uniforme se comparado aos demais grupos de imagens.

Esse resultado nos leva a conclusão que, para alguns tipos de imagens, não é vantagem termos um número maior de vetores no *codebook*, já que apenas alguns poucos são usados na maioria dos casos. Para esses grupos de imagens, ganha-se mais com o uso dos flags *flipbit* e *diffbit*, do que usando um *codebook* maior. Porém, para as texturas o aumento do *codebook* pode ser uma proposta válida.

Tabela 3: Diferenças do PSNR

Textura	iPACK	Solução	Dif.	Dif. Perc.(%)
1	45.20	45.07	0.13	0.29
2	32.79	32.26	0.53	1.61
3	34.14	33.50	0.64	1.87
4	32.00	31.04	0.96	3.00
5	31.04	30.01	1.03	3.32
6	31.23	30.20	1.03	3.30

Sabendo desses resultados, direcionaremos nosso estudo apenas para o grupo das texturas. Analisaremos os PSNR's da compressão de imagens desse tipo, usando o iPACKMAN original e a nossa versão. A Tabela 3 mostra os valores dos PSNR's das imagens de teste para as duas versões, apresentando também a diferença entre eles. Percebemos que existe uma diferença quantitativa com relação à compressão de ambos os algoritmos, porém, é uma diferença relativamente pequena.

Na Figura 6 temos um comparativo visual da qualidade da compressão dessa nova versão do iPACKMAN, comparada com a do iPACKMAN

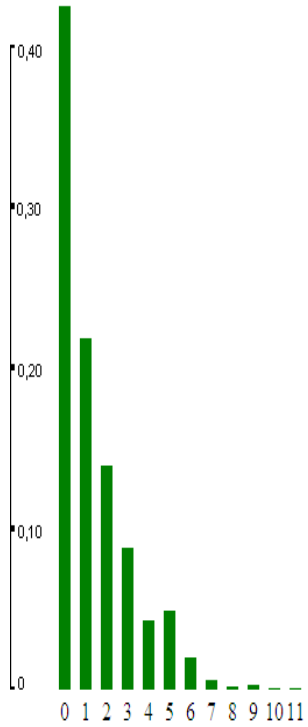

Figura 5: Uso das codewords nas imagens não-textura.

original. Nesta imagem fica mais evidente que os valores quantitativos não são significativos. Visualmente, para todos os fins práticos pode-se afirmar que as imagens são visualmente idênticas.

5. Conclusões e trabalhos futuros

Este trabalho apresentou uma modificação no algoritmo para compressão de texturas iPACKMAN [1]. A idéia principal foi explorar dois bits da compressão do iPACKMAN (*diffbit* e *flipbit*) para simplificar o algoritmo. Trocamos a utilização original destes bits para aumentar o número de *codewords* disponíveis. Esta modificação permite um algoritmo de compressão das imagens muito mais eficiente, sem perda significativa na qualidade para imagens de textura, as mais utilizadas em aplicações de computação gráfica. Tanto no PACKMAN quanto no iPACKMAN a compressão é feita de forma exaustiva, ou seja, são realizadas 2^{11} (2^3 *codewords* x 2^2 coordenadas x 2^2 modificadores x 2^4 pixels) interações para cada bloco de 4 x 4 pixels em uma imagem. A nossa solução reduz este número de interações à metade ($2^{10} = 2^4$ *codewords* x 2^2 coordenadas x 2^4 pixels).

Há uma grande possibilidade de trabalhos futuros

Figura 6: Na primeira coluna estão as imagens originais, na segunda as comprimidas com o iPACKMAN e na terceira estão as imagens comprimidas com a nossa versão do algoritmo.

na linha do que exploramos aqui. Os testes realizados apontam que um único *codebook* universal, não é a melhor solução em alguns casos. A concentração de uso das entradas do *codebook* em algumas partes aponta para uma solução com *codebooks* individualizados, ou por imagem, ou por grupos de imagens agrupadas por algum critério, por exemplo, imagens da natureza, imagens de pessoas, etc.

6. Agradecimentos

Gostariamos de agradecer aos autores do iPACKMAN a disponibilidade da imagem benchmark utilizada e a disponibilidade do código original do algoritmo.

Referências

[1] J. Ström, T. Akenine-Möller: 'iPACKMAN: High-Quality, Low Complexity Texture Compression for Mobile Phones', *Graphics Hardware* (2005), ACM Press, pp. 63-70.

[2] S. Fenney: 'Texture Compression using Low-Frequency Signal Modulation', *Graphics Hardware* (2003), ACM Press, pp. 84-91.

[3] Li-Yi Wei, M. Levoy: 'Fast texture synthesis using tree-structured vector quantization', *SIGGRAPH 2000 - Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press, pp. 479-488.

[4] P. Heckbert: 'Survey of Texture Mapping', *GI 1986 - Proceedings of the Graphics Interface* (1986), Canadian-Human Computer Communications Society, pp. 207-212.

[5] A. Beers et al: 'Rendering from Compressed Textures', *SIGGRAPH 1996 - Proceedings of Siggraph 1996* (1996), ACM Press, pp. 373-378.

[6] G. Knittel et al: 'Hardware for Superior Texture Performance', *Computers and Graphics* (1996), vol. 20, n. 4, pp. 475-481.

[7] Y. Linde, A. Buzo, R. Gray: 'An Algorithm for Vector Quantizer Design', *IEEE Transactions on Communications* (1980), vol. 28, n.1, pp. 84-94.