3
Opacidade 3D

Existem trabalhos na literatura que falam sobre a função de transferência multidimensional [5, 12, 19, 20, 25] e em todos a grande dificuldade é encontrar uma função de transferência multidimensional que seja adequada para o tipo de dado que esteja sendo visualizado. O problema de se trabalhar com um objeto de dimensão maior que dois está na dificuldade em se criar uma interface com o usuário que é quase tão complicado que editar diretamente o dado. Com o aumento da dimensão aumenta-se a flexibilidade, porém aumenta-se também a dificuldade do usuário definir uma função de transferência que melhor se adeque ao dado que se deseja visualizar.

A interface com o usuário é extremamente importante, pois é esta que vai conduzir o usuário durante o processo de ajuste da função de transferência e, conseqüentemente, na geração da visualização final. Caso o usuário tenha dificuldades em ajustar a função de transferência, possivelmente não terá controle sobre o que está sendo visualizado.

3.1.
Teoria da Opacidade 3D

Neste trabalho, quando se trata de uma função de transferência (Opacidade 3D), trata-se de um objeto que tem a mesma dimensão do volume que se quer inspecionar. Desta forma, a tarefa de especificar uma função de Opacidade 3D com toda a sua liberdade pode ser tão difícil quanto manipular diretamente o volume de interesse. Neste trabalho sugere-se uma simplificação na especificação da Opacidade 3D e propõe-se restringir às funções de transferência 3D que podem ser obtidas através de uma combinação de três funções de transferência 1D. A opacidade 3D, desenvolvida neste trabalho, utiliza três tabelas de cores 1D para compor a chamada *lut 3D* (tabela de cores 3D). A idéia de dimensão é dada por cada tabela de cor 1D que compõe a tabela de cores 3D.

Optou-se por utilizar três tabelas de cores 1D, pois, conforme já mencionado, caso fosse criada uma interface 3D a dificuldade que o usuário teria em escolher uma região para marcar como visível seria a mesma que marcar no próprio dado volumétrico que se deseja visualizar.

Para a opacidade 3D foi criado um novo volume, que seguindo a nomenclatura dada por Silva[30], chamou-se de volume terciário. Seguindo a nomenclatura da opacidade 2D, os nomes dos volumes não têm nenhuma ligação com a importância do mesmo na visualização final.

Junto com as três tabelas de cores 1D descritas anteriormente, na seção 2.2.5.1, a tabela de cores 3D é composta por mais doze coeficientes, que serão utilizados para criar a função de transferência tridimensional. Esta, por sua vez, é responsável por criar uma tabela de cores resultante que será utilizada na visualização do dado sísmico. Esta tabela resultante não é apresentada ao usuário, pois sua criação é feita no *hardware* gráfico.

A função de transferência é dada através de uma combinação entre as três tabelas de cores utilizadas no processo e de acordo com os coeficientes (parâmetros de combinação) fornecidos pelo usuário. A combinação é feita com base nos canais *RGB* e da transparência das tabelas de cores, atendendo o grau de contribuição de cada canal no valor final da tabela de cor resultante. Neste trabalho explora-se duas maneiras de combinar as três cores provenientes de cada tabela 1D: uma por combinação linear e outra por produto.

Os doze parâmetros utilizados para definir a função de transferência que irá gerar a tabela de cores resultante são: C_{r_1}, C_{r_2}, C_{r_3}, C_{g_1}, C_{g_2}, C_{g_3}, C_{b_1}, C_{b_2}, C_{b_3}, C_{a_1}, C_{a_2} e C_{a_3}, onde os C_{r_i}, C_{g_i}, C_{b_i} e C_{a_i} representam os coeficientes do canal *R*, *G*, *B* e *A* de cada tabela de cores, respectivamente.

Uma possível função de transferência é dada por:

$$C_R = C_{r_1} \times R_1 + C_{r_2} \times R_2 + C_{r_3} \times R_3$$

$$C_G = C_{g_1} \times G_1 + C_{g_2} \times G_2 + C_{g_3} \times G_3$$

$$C_B = C_{b_1} \times B_1 + C_{b_2} \times B_2 + C_{b_3} \times B_3$$

$$C_A = C_{a_1} \times A_1 + C_{a_2} \times A_2 + C_{a_3} \times A_3 \qquad (1)$$

Onde C_R, C_G, C_B e C_A são os canais *R*, *G*, *B* e *A* da tabela de cores resultante, respectivamente. Os coeficientes podem variar de zero a um, sendo zero quando

este parâmetro não contribui em nada para a tabela de cores resultante e um para quando o parâmetro contribui com cem por cento para a tabela de cores resultante.

Os parâmetros R_i, G_i, B_i e A_i são os valores dos canais R, G, B e A, respectivamente, das tabelas de cores 1D que irão gerar a tabela de cores 3D.

Durante a execução deste trabalho foram feitos testes com outras funções de transferência e chegou-se à conclusão que para alguns casos a multiplicação dos canais é mais vantajosa do que a soma. A combinação de multiplicação dos canais é dada por:

$$C_R = C_{r_1} \times R_1 \times C_{r_2} \times R_2 \times C_{r_3} \times R_3$$

$$C_G = C_{g_1} \times G_1 \times C_{g_2} \times G_2 \times C_{g_3} \times G_3$$

$$C_B = C_{b_1} \times B_1 \times C_{b_2} \times B_2 \times C_{b_3} \times B_3$$

$$C_A = C_{a_1} \times A_1 \times C_{a_2} \times A_2 \times C_{a_3} \times A_3 \qquad (2)$$

No caso da função de transferência que utiliza a multiplicação dos canais de cores, a representação da função de transferência utilizada pode ser resumida a:

$$C_R = C_{res_R} \times R_1 \times R_2 \times R_3$$

$$C_G = C_{res_G} \times G_1 \times G_2 \times G_3$$

$$C_B = C_{res_B} \times B_1 \times B_2 \times B_3$$

$$C_A = C_{res_A} \times A_1 \times A_2 \times A_3 \qquad (3)$$

onde, $C_{res_R} = C_{r_1} \times C_{r_2} \times C_{r_3}$, $C_{res_G} = C_{g_1} \times C_{g_2} \times C_{g_3}$, $C_{res_B} = C_{b_1} \times C_{b_2} \times C_{b_3}$ e $C_{res_A} = C_{a_1} \times C_{a_2} \times C_{a_3}$.

Quando o usuário seleciona uma determinada região do atributo sísmico como sendo visível, há interesse em estudar um evento sísmico que ocorreu naquela era geológica. Gerhardt[11,12] verificou que com a opacidade 1D, muitas vezes, o usuário não consegue isolar por completo este evento (Gerhardt[11,12] e Silva[30]) e para tal ele tem à disposição a opacidade 2D. Esta se utiliza de dois atributos sísmicos para tentar isolar o evento sísmico de interesse, fazendo com que apenas o que está marcado como sendo visível nos dois atributos seja visto pelo usuário.

Quando a opacidade 3D utiliza a soma de canais para gerar a visualização, isto não acontece, pois ao invés de restringir-se uma área de interesse, a soma

funciona como uma “união” de áreas de interesse, fazendo com que mais dados sejam vistos. Já a multiplicação de canais restringe, somente, à área de interesse.

A Figura 15 mostra como a multiplicação dos atributos funciona como uma interseção. Somente na região em que ambos atributos estão visíveis é que o produto resulta um. Nas demais regiões este produto é zero e nada é visualizado.

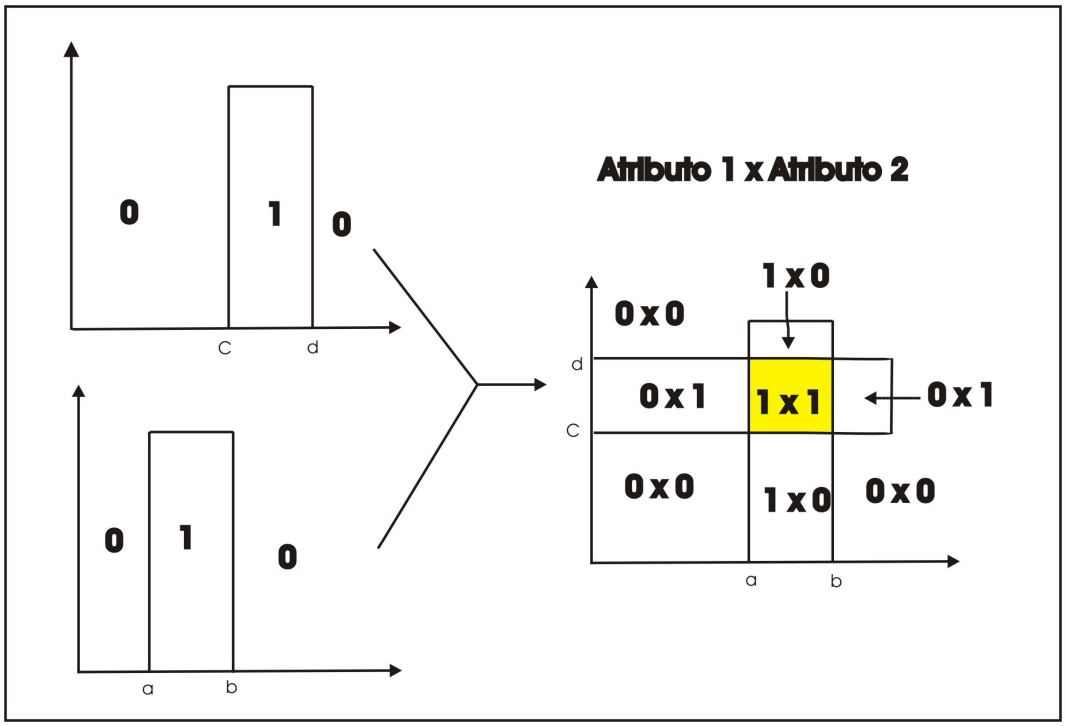

Figura 15 – Multiplicação do canal alfa na opacidade 2D.

Este trabalho tem por objetivo apresentar uma técnica que auxilie o trabalho de um intérprete, disponibilizando uma série de atributos sísmicos, onde cada um acrescente uma informação a mais para o estudo de um determinado bloco sísmico.

A interação com o usuário é de fundamental importância para a eficácia desta técnica. Para garantir uma renderização em tempo real, utiliza-se programação em placa de vídeo. Para executar o algoritmo de visualização é necessário que a estação de trabalho possua uma placa de vídeo com extensão GL_ARB_fragment_program. Esta extensão foi aprovada pelo comitê ARB em setembro de 2002 e incorporada ao *OpenGL* a partir da versão 1.3. Por utilizar uma extensão do *OpenGL* a implementação não é restrita a um fabricante específico de *hardware* gráfico, basta o mesmo ter a extensão mencionada.

3.2. Implementação

A implementação da opacidade 3D foi feita no *software* v3o2. No desenvolvimento deste trabalho foi necessária a criação de uma interface onde o usuário tem a oportunidade de editar a tabela de cores referente à opacidade 3D. Esta tabela de cores foi chamada de *lut 3D*. A seguir será apresentada a parte do código referente à programação em placa com uma descrição de cada linha do código. Esta parte do código foi implementada como um *fragment program*, que é a parte do código que será executada em *GPU*. O *fragment program* é executado para cada fragmento, ou seja, para cada *pixel* da imagem que encontra-se no *z-buffer* e que será renderizado no *frame buffer*.

```
(1)  "!!ARBfp1.0"
(2)  "TEMP rgba_frag;"
(3)  "TEMP lut1;"
(4)  "TEMP lut2;"
(5)  "TEMP lut3;"
(6)  "TEMP temp;"
(7)  "TEX rgba_frag, fragment.texcoord[0], texture[0], 3D;"
(8)  "TEX lut1, rgba_frag.g, texture[2], 2D;"
(9)  "TEX lut2, rgba_frag.b, texture[3], 2D;"
(10) "TEX lut3, rgba_frag.r, texture[4], 2D;"
(11) "MUL temp, program.local[1], lut1;"
(12) "MAD temp, program.local[2], lut2, temp;"
(13) "MAD temp, program.local[3], lut3, temp;"
(14) "MUL result.color, temp, 0.3333;"
(15) "END";
```

Esta seção será dividida em três partes e em cada uma, será comentado um conjunto de linhas do código. A primeira parte identifica as texturas que serão utilizadas no processo de visualização e mostra como estas são construídas. Em seguida é apresentada a parte onde são feitas as consultas das texturas (*fetch*). E por último é apresentada a combinação das cores, utilizando as funções de transferências que foram descritas na seção anterior.

Como pode ser visto o *fragment program* é uma *string*. Esta string é passada para a placa em tempo de execução. A placa fica encarregada de compilar este programa e de armazená-lo em memória (da *GPU*).

A primeira linha do código refere-se a versão do *fragment program* que está sendo utilizada, no caso a ARBfp1.0. As linhas (2), (3), (4), (5) e (6) são declarações de variáveis temporárias que estão sendo criadas pelo *fragment program*. Estas variáveis são vetores com quatro posições. Neste caso a palavra *TEMP* é uma palavra reservada do *fragment program.*

3.2.1. Construção das Texturas

Antes de explicar como ocorre a construção das texturas que são utilizadas na visualização, deve-se entender um detalhe de implementação do *v3o2*. Para compor a opacidade 2D, Silva[30] cria uma textura *RGB* tridimensional para guardar os valores dos atributos que compõem a opacidade 2D.

Ao visualizar um volume, no *v3o2*, é feita uma "transformação" do dado sísmico, pela qual os valores do atributo sísmico são convertidos para valores inteiros entre zero e duzentos e cinqüenta e cinco. Este processo é conveniente já que a visualização do dado sísmico é feita utilizando uma tabela com 256 cores. Este processo é comumente chamado de quantização. O valor do atributo quantizado refere-se a uma cor da tabela de cores 1D associada a este volume que será utilizado para visualização.

Para compor a textura *RGB* tridimensional, Silva[30] faz com que o canal R seja sempre zero, o canal G seja os valores quantizados do volume primário e o canal B como sendo os valores quantizados do volume secundário. A Figura 16 mostra o esquema de construção desta textura tridimensional.

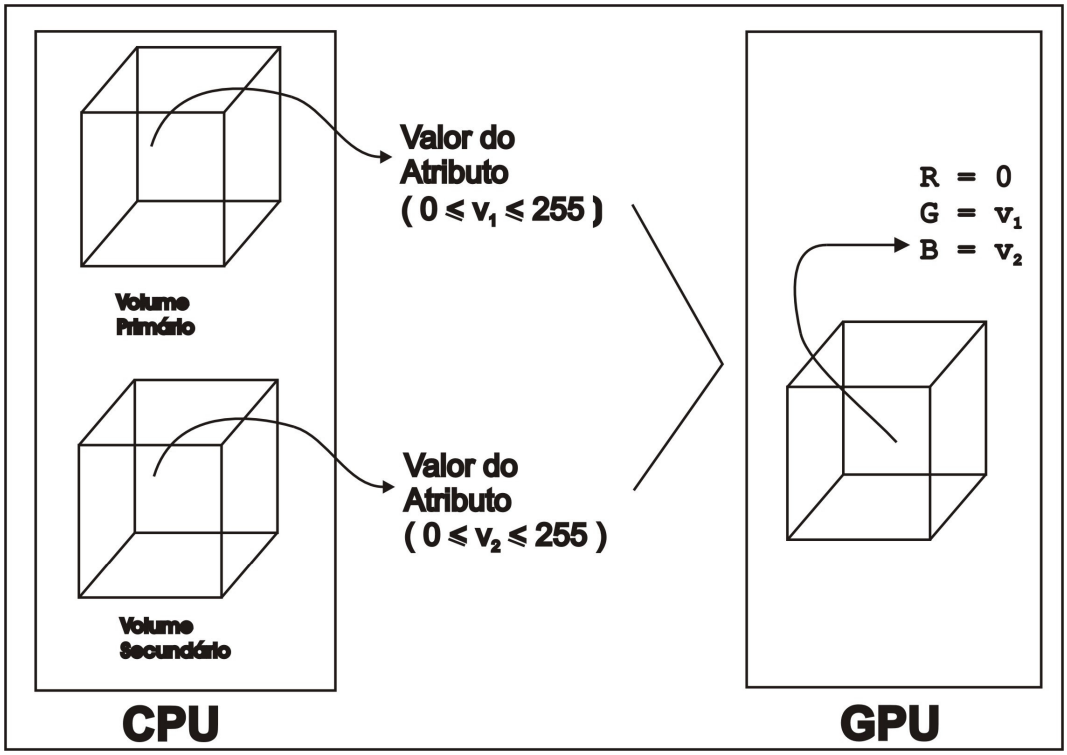

Figura 16 – Construção da textura tridimensional RGB (opacidade 2D).

A opacidade 3D faz uso desta textura tridimensional atribuindo os valores quantizados do volume terciário ao canal R da textura, que anteriormente tinha seus valores nulos. Os canais G e B não são alterados.

Antes de dar continuidade a análise do código, faz-se necessário identificar que texturas estão sendo criadas no programa. A primeira textura que aparece no código é a *texture[0]* que é uma textura 3D. Esta textura é a textura tridimensional que foi descrita anteriormente. A *texture[1]* não aparece em nenhum momento no código apresentado. Isto se deve ao fato desta textura estar sendo utilizada para a iluminação do dado sísmico, não sendo utilizada na opacidade 3D.

As texturas de número dois, três e quatro são as texturas do tipo 2D que estão associadas aos volumes primário, secundário e terciário, respectivamente. A criação destas texturas e o carregamento destas, em GPU, são feitos fora do *fragment program*.

3.2.2. Consulta de Textura

Para aplicar uma textura em um objeto, é preciso criar uma relação entre os vértices dos polígonos do objeto e os *texels* da textura. Esta relação é feita através

do mapeamento das coordenadas de textura. Cada vértice de um polígono que está sendo desenhado possui um vetor *(s, t, r, w)* de coordenadas de textura que identifica um ponto com seu valor *RGBA* na textura. Os vetores dos pontos são interpolados para todos os fragmentos do polígono durante a rasterização.

As linhas (7), (8), (9) e (10) do código apresentado começam com a instrução *TEX*, que é o mapeamento da coordenada de textura para a cor da textura. A instrução *TEX* possui a sintaxe

```
TEX v, u, t, tipo de textura
```

onde, `v` representa um vetor de saída com os valores de RGB. O segundo parâmetro `u` indica a identificação da imagem de textura. A textura "alvo" é indicada pelo parâmetro `t` e o `tipo de textura` refere-se à dimensão da textura.

A linha (7) do código apresentado usa a coordenada de textura (*s, t, r)* do fragmento que está em *fragment.texcoord[0]* para obter em *texture[0]* qual é o RGB que esta coordenada de textura aponta e associa ao vetor *rgba_frag*. Esta textura é uma textura dependente, pois utiliza os valores RGB como coordenada de textura. Isto pode ser visto ao analisar as linhas (8), (9) e (10).

Como foi descrito na seção 3.2.1 a textura zero possui os valores quantizados referentes aos volumes primário, secundário e terciário. Logo, o vetor *rgba_frac.g* representa o valor "quantizado" do atributo primário e este valor representa um índice da tabela de cores 1D que está associado a este volume.

Na linha de número (8), o valor que foi retornado da linha (7) e armazenado em *rgba_frac.g* é usado como coordenada de textura na consulta ao valor de *RGBA* que está em *texture[2]*. Este valor de RGBA é atribuído a variável *lut1*. Feito isso, a cor resultante em *lut1* é a cor definida pelo volume primário. O mesmo ocorre nas linhas (9) e (10), onde são atribuídas as cores definidas pelos volumes secundários e terciários às variáveis *lut2* e *lut3*, respectivamente.

A Figura 17 mostra o caminho que é feito pela consulta *(fetch)* de textura até a cor que contribuirá com a visualização.

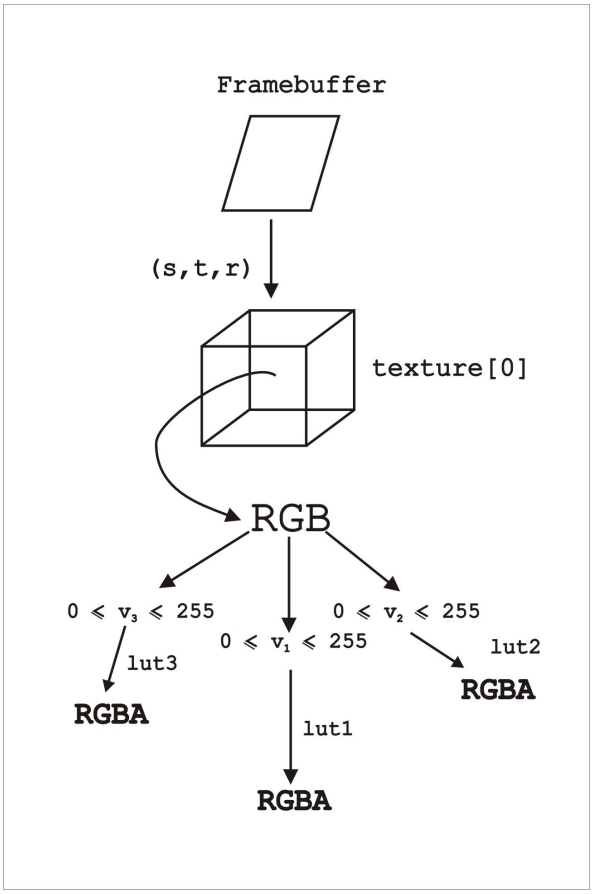

Figura 17 – Consulta (fetch) de textura até a cor.

Ao final da fase de consulta temos três vetores RGBA, um para cada volume (atributo). Esses três vetores são combinados para gerar a cor final do fragmento.

3.2.3. Combinação das cores

A instrução *MUL* aparece na linha (11). Esta instrução é uma palavra reservada do *fragment program* que tem por objetivo multiplicar duas variáveis, essa multiplicação é feita componente a componente. A sintaxe é *"MUL a, b, c;"*, onde pode-se entender como sendo $a = b \times c$. Ainda na linha (11), aparece uma variável chamada *program.local*. Esta variável refere-se aos coeficientes descritos na seção 3.1, que serão utilizados na função de transferência. Este parâmetro também é um vetor de quatro posições.

O *program.local[1]* refere-se aos coeficientes C_{r_1}, C_{g_1}, C_{b_1}, C_{a_1}, ou seja, os coeficientes que estão sendo aplicados à tabela de cores 1D associada ao volume primário. Estes parâmetros são passados para a *GPU* através do comando

glProgramLocalParameter4dARB, e toda vez que um parâmetro for atualizado, este comando deve ser chamado para informar ao *hardware* gráfico os novos valores a serem utilizados. As variáveis *program.local[2]* e *program.local[3]* são os coeficientes relacionados à tabela de cores associada ao volume secundário e à tabela de cores associada ao volume terciário, respectivamente. O resultado da linha (11) é que o vetor *temp* será representado por:

$$temp.r = C_{r_1} \times lut1.r$$

$$temp.g = C_{g_1} \times lut1.g$$

$$temp.b = C_{b_1} \times lut1.b$$

$$temp.a = C_{a_1} \times lut1.a\,.$$

Nas linhas (12) e (13) aparece a instrução *MAD (multiply and add)*, que é responsável por multiplicar e somar. A sintaxe de *MAD* é *"MAD a, b, c, d"* que pode ser escrita como sendo $a = b \times c + d$. Logo, na linha (12) o vetor *temp* será representado como sendo a multiplicação dos coeficientes associados a tabela de cores do volume secundário somado com os valores encontrados na linha (11), ou seja, *temp* será representado por:

$$temp.r = C_{r_2} \times lut2.r + temp.r$$

$$temp.g = C_{g_2} \times lut2.g + temp.g$$

$$temp.b = C_{b_2} \times lut2.b + temp.b$$

$$temp.a = C_{a_2} \times lut2.a + temp.a$$

Para a linha (13) tem-se que *temp* será descrito como sendo:

$$temp.r = C_{r_3} \times lut3.r + temp.r$$

$$temp.g = C_{g_3} \times lut3.g + temp.g$$

$$temp.b = C_{b_3} \times lut3.b + temp.b$$

$$temp.a = C_{a_3} \times lut3.a + temp.a$$

A linha de número (14) novamente representa uma multiplicação. Aqui o resultado da multiplicação é atribuído a variável *result.color*. Esta variável é uma palavra reservada do *fragment program* e representa a cor com que o fragmento será desenhado no *frame-buffer*. A multiplicação por 0,3333 tenta impedir a saturação da cor resultante do fragmento.

Para finalizar o código a linha de número (15) indica ao *fragment program* que este chegou ao fim.

Esta foi a análise para a parte do código referente ao processamento em *GPU* utilizando uma função de transferência com soma de canais. A seguir será apresentado o código com a função de transferência que utiliza a multiplicação dos canais de cores para compor a visualização.

```
(1)  "!!ARBfp1.0"
(2)  "TEMP rgba_frag;"
(3)  "TEMP lut1;"
(4)  "TEMP lut2;"
(5)  "TEMP lut3;"
(6)  "TEMP temp1;"
(7)  "TEMP temp2;"
(8)  "TEMP temp3;"
(9)  "TEMP temp4;"
(10) "TEX rgba_frag, fragment.texcoord[0], texture[0], 3D;"
(11) "TEX lut1, rgba_frag.g, texture[2], 2D;"
(12) "TEX lut2, rgba_frag.b, texture[3], 2D;"
(13) "TEX lut3, rgba_frag.r, texture[4], 2D;"
(14) "MUL temp1, program.local[1], lut1;"
(15) "MUL temp2, program.local[2], lut2;"
(16) "MUL temp3, program.local[3], lut3;"
(17) "MUL temp4, temp2, temp1;"
(18) "MUL result.color, temp4, temp3;"
(19) "END";
```

Analisando o código acima, pode-se notar que não existe nenhuma instrução nova. São criadas variáveis temporárias novas como *temp1, temp2, temp3 e temp4*. A maior diferença para a função de transferência que utiliza a soma dos canais é que neste caso as instruções *MAD* foram substituídas por *MUL*. As linhas (12) e (13) da função de transferência com soma de canais deu lugar às linhas (15) e (16) da função de transferência com multiplicação de canais. Nesta função de transferência não é necessário fazer a multiplicação por 0,3333 para evitar a saturação dos canais.

Na ferramenta que está sendo oferecida junto ao *software v3o2*, a interface de construção da tabela de cores 3D possui dois quadros de parâmetros. No

quadro da esquerda o usuário informa qual é a tabela de cores 1D que se deseja associar ao volume primário, a tabela de cores do volume secundário e a tabela do volume terciário. No quadro à direita tem-se a opção de editar os coeficientes que serão passados para o programa. Na Figura 18 tem-se uma visão da interface final.

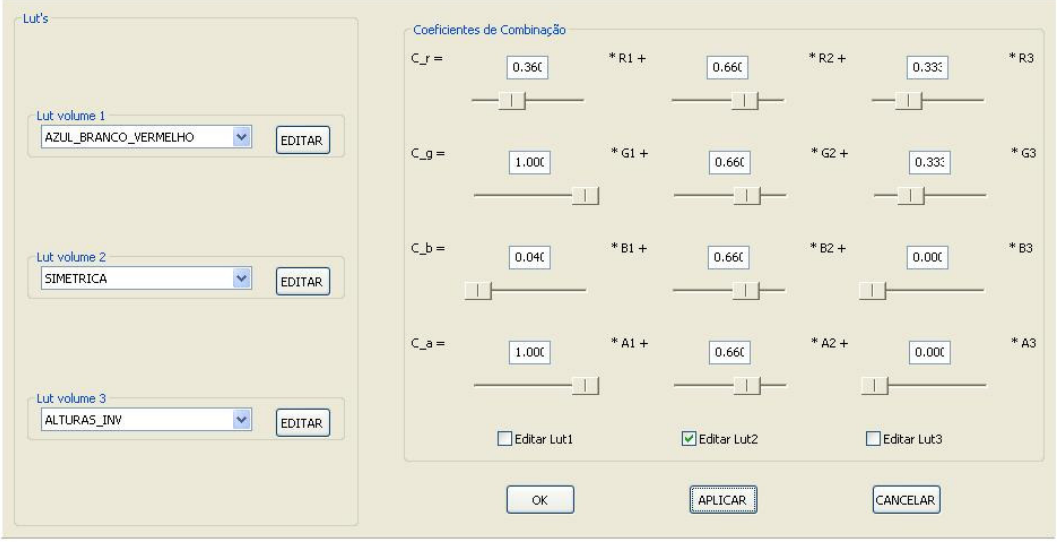

Figura 18 – Interface de edição da tabela de cores 3D com soma de canais

Percebe-se na figura que as tabelas de cores são associadas a volumes. Estes volumes são os volumes sísmicos de origem dos atributos, ou seja, são os dados volumétricos que compõem a combinação.

A tabela de cores que for associada a um volume pode ser editada como qualquer tabela de cores 1D. No quadro da direita há, ainda, três *check boxes* (caixas que podem ser marcadas como ativas ou inativas) que quando ativas, indicam que o usuário deseja editar os valores dos coeficientes referentes àquela tabela de cores de forma idêntica, ou seja, todos os quatro coeficientes referentes à tabela de cores em questão terão o mesmo valor. Caso o usuário desmarque a *check box* os coeficientes de combinação poderão ser editados separadamente.

No exemplo da Figura 18 pode-se notar que o *check box* referente a tabela de cores associada ao volume secundário está ativo e que os quatro coeficientes possuem valor 0.66. Os *check boxes* das outras tabelas de cores estão inativos e têm valores diferentes para seus coeficientes.