6
Aspectos computacionais

6.1
Edição de filtros

Nesta seção detalhamos como é feito o mapeamento das frequências do som às frequências da malha em ambos os métodos.

6.1.1
Filtros em Harmônicos de Variedade

Os filtros da equação (2-16) são sensíveis, então o mapeamento das amplitudes das harmônicas do sinal para as amplitudes das harmônicas da variedade requer bastante cautela se for feito manualmente. Por isto, foi desenvolvida um sistema simples de galeria para ajudar o usuário a realizar este mapeamento.

Suponha um sinal de som $g(t)$, o qual queremos usar para deformar nossa malha. Queremos obter um filtro $\varphi(\nu)$ das amplitudes harmônicas $\tilde{g}(\xi)$ do sinal $g(t)$, tal que a relação $\varphi(\nu) = \Phi(\tilde{g})(\nu)$ seja não necessariamente linear. Além disso, o número de frequências $\#\xi$ do sinal quase sempre é diferente do número de harmônicas $\#\nu$ da malha.

Portanto, o mapeamento das amplitudes harmônicas do sinal para as amplitudes harmônicas da malha é realizado em duas etapas:

- Usamos uma função de transferência $t : \xi \mapsto \nu \in \{0, 1, \ldots, \#\nu - 1\}$.
- Usamos uma função de amplificação, $\mathrm{a} : \nu \mapsto \mathrm{a}(\nu) \in \mathbb{R}$ aplicadas nas amplitudes dos harmônicos da variedade.

Como queremos que cada harmônico da malha receba contribuição de todos os harmônicos do sinal, para por exemplo podermos associar uma harmônica a um instrumento, propomos o mapeamento entre harmônicos $\Phi_{t,\mathrm{a}} : \tilde{g} \mapsto \varphi$ como:

$$\Phi_{t,\mathrm{a}}(\tilde{g})(\nu) = \mathrm{a}(\nu) \cdot \left(\sum_{\xi \in t^{-1}(\nu)} \tilde{g}(\xi) \right) + 1. \tag{6-1}$$

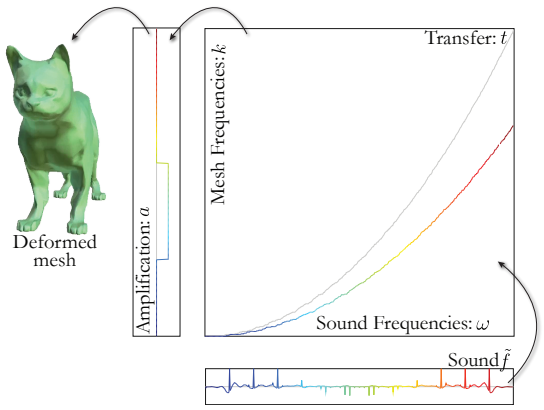


Figura 6.1: Filtro de combinações entre as funções de transferência e amplificação. A curva em cinza para a transferência corresponde ao mapeamento direto de $\xi = \sqrt{\Lambda}_{t(\xi)}$.

Na figura 6.1 ilustramos o mapeamento das funções de transferência e de amplificação. Desta forma, as frequências da malha irão receber contribuições de todas as frequências do som $t^{-1}(\nu)$. Note que adicionamos um na equação, para manter a intuição usual de amplificação. Amplificando todas as frequências da malha para 0 (isto é: a $\equiv$ 0) obtemos $\phi = 1$; logo não haverá deformação da malha. Note também que assim como as amplitudes do som podem ser negativas, a amplificação também pode ser negativa.

6.1.2
Filtros em Análise Modal

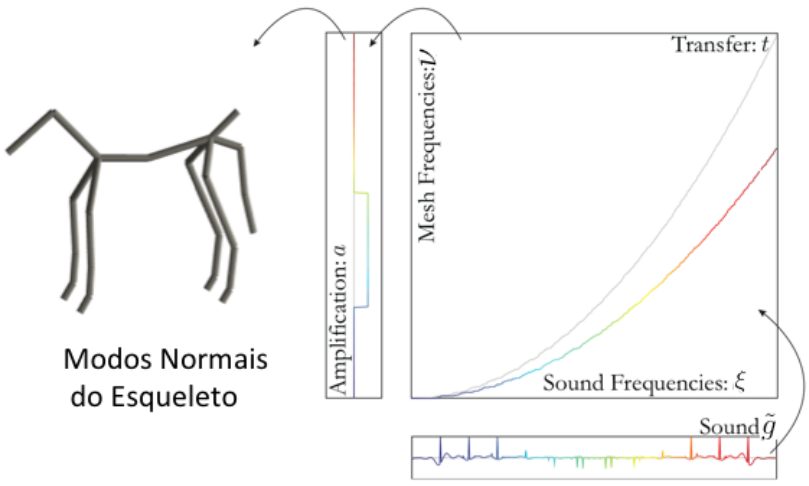


Figura 6.2: Filtro como a combinação das funções de transferência e de amplificação para os harmônicos do modo normal.

Analogamente aos Harmônicos de Variedade, dado um sinal de som $g(t)$, a relação $\varphi(\nu) = \Phi(\tilde{g})(\nu)$, entre o filtro e o sinal $\tilde{g}(\nu)$ não precisa ser necessariamente linear. Em geral o número de frequências $\#\xi$ do sinal é diferente do número de frequências $\#\nu$ do Modelo Modal, e seria interessante que cada corpo tivesse a influência de todas as harmônicas do sinal. Portanto podemos criar uma função de amplificação $\mathrm{a} : \nu \longrightarrow \mathrm{a}(\nu) \in \mathbb{R}$ e uma função de transferência $t : \xi \longrightarrow \nu$ tal que combinando estes construímos a aplicação $\Phi_{\mathrm{a}} : \tilde{g} \longrightarrow \varphi$ como:

$$\Phi_{\mathrm{a}}(\nu) = \mathrm{a}(\nu)\left(\sum_{\xi \epsilon t^{-1}(\nu)} \tilde{g}(\xi)\right) + 1. \tag{6-2}$$

Com isso, cada corpo terá a influência de todas as harmônicas possíveis do sinal, e se a amplificação for nula, $\mathrm{a} \equiv 0$, os corpos permanecerão em suas respectivas posições. Substituindo a função do filtro na equação original teremos que:

$$\begin{aligned} F(i) &= \sum_{\nu=0}^{n-1}\left(\mathrm{a}(\nu)\left(\sum_{\xi \epsilon t^{-1}(\nu)} \tilde{g}(\xi)\right) + 1\right)\tilde{F}(\nu)H_\nu \\ &= \sum_{\nu=0}^{n-1}\mathrm{a}(\nu)\left(\sum_{\xi \epsilon t^{-1}(\nu)} \tilde{g}(\xi)\right)\tilde{F}(\nu)H_\nu + \sum_{\nu=0}^{n-1}\tilde{F}(\nu)H_\nu. \end{aligned} \tag{6-3}$$

Observe que a amplificação "a" pode assumir valores negativos, dado que as amplitudes das harmônicas podem ser negativas.

6.2 Galeria

Na animação usando Harmônicos de Variedades, o mapeamento $\varphi_{t,a}$ é representado por dois vetores: $t \in \mathbb{N}^{\#\xi}$, e $\mathrm{a} \in \mathbb{R}^{\#\nu}$. Podemos facilmente misturar as aplicações harmônicas por combinações desses vetores. Usando o vocabulário de algoritmos genéticos, o mapeamento harmônico $\Phi_{t,\mathrm{a}}$ é representado por dois cromossomos a e t, os quais podem reproduzir pela combinação.

Diferentes mapeamentos harmônicos são propostos ao usuário que, então, seleciona aqueles que deseja. A partir da escolha dos mapeamentos é gerada uma nova galeria usando reprodução genética, até o usuário escolher apenas um mapeamento harmônico.

6.2.1
Geração de novas galerias por reprodução genética

Para reproduzir novas galerias, S pares de mapeamentos selecionados pelo usuário são aleatoriamente escolhidos. Como as funções t e a tem efeitos complementares, as reproduzimos independentemente. Isto reduz o tamanho inicial da galeria. Na prática, isto significa que primeiro decidimos se combinamos as funções de transferência do par usando um experimento de $\frac{1}{2}$−Bernoulli ("cara" ou "coroa"). De forma similar, decidimos se as funções de amplificação serão combinadas.

Para combinar as funções de transferência de frequências t' e t'', primeiro escolhemos um valor inteiro aleatoriamente n_ν^0, como uma variável aleatória geométrica em $\{1,\ldots,\#\nu\}$, e uma variável aleatória r^0 com distribuição uniforme em $[0,1]$. Selecionamos os n_ν^0 primeiros coeficientes do vetor t como os primeiros n_ν^0 coeficientes de $r^0 t' + (1-r^0)t''$. Novamente escolhemos $n_\nu^1 \in \{1,\ldots\#\nu\}$ e $r^1 \in [0,1]$, e ajustamos r^1 de tal forma que $n_\nu^0 + n_\nu^1 \leq \#\nu$ (o objetivo do processo aleatório geométrico é reduzir os efeitos deste truncamento). Determinamos então os n_ν^1 valores de t como descrito acima, e repetimos até completar todas as frequências. Realizamos as mesmas operações acima para a combinação das amplificações.

Este método de combinação evita produzir combinações que variem bruscamente, quando comparados a escolher aleatoriamente valores reais r em cada frequência.

6.2.2
Inicialização das galerias

Inicialmente geramos uma galeria que poderia teoricamente gerar qualquer mapeamento harmônico usando a reprodução citada na subseção 6.2.1. Como as reproduções das funções de transferência e de amplificação são independentes, podemos usar os S elementos da galeria inicial para gerar tanto as funções de transferência das frequências quanto as funções de amplificação. Isto reduz o tamanho da galeria inicial embora, geralmente seja preciso uma reprodução a mais para obtermos mapeamentos interessantes.

A primeira função de transferência da frequência é o mapeamento direto:

$$t_{ini}(\xi) = min\left\{\nu;\ tal\ que\ \frac{\xi}{\xi_{\#\xi}} \leq \sqrt{\frac{\Lambda_\nu}{\Lambda_{\#\nu}}}\right\}. \tag{6-4}$$

Isto nos assegura que, se existe um único ν tal que $\frac{\xi}{\xi_{\#\xi}} \leq \sqrt{\frac{\Lambda_\nu}{\Lambda_{\#\nu}}}$, então $t_{ini}(\xi) = \nu$. Esta função mapeia as baixas frequências do som nas baixas frequências da malha, similarmente para as altas. Podemos definir

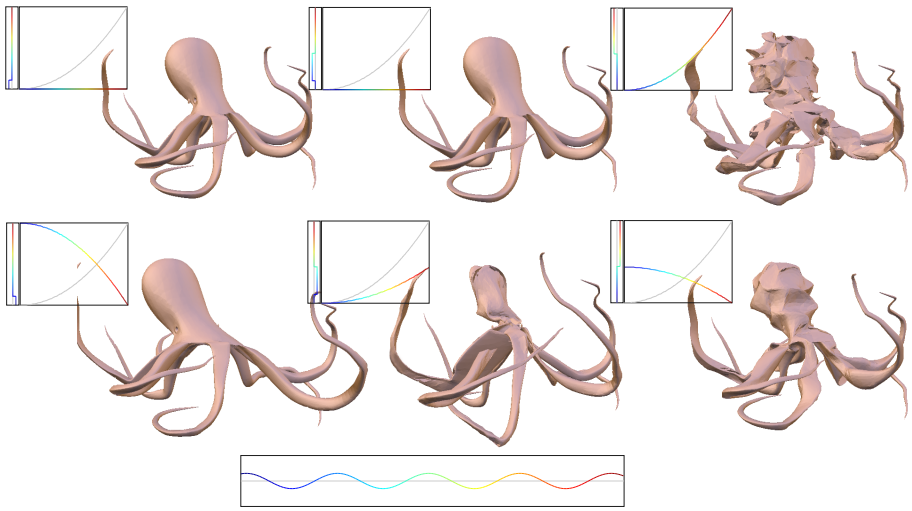


Figura 6.3: Galeria inicial do modelo polvo e suas respectivas funções de amplificação e transferência. A frequência do som é ilustrada na parte de baixo na figura.

$t_{rev} = \#\nu - t_{ini}$ que mapeia as altas frequências do som nas baixas frequências, e vice-versa.

Geralmente, a alteração das baixas frequências da malha nos dão efeitos mais visíveis. Portanto, definimos as funções de transferência de frequências das galerias iniciais como transferências concentradas nas baixas frequências: $t(\xi) = t_{ir}(\epsilon\xi)$, onde t_{ir} é t_{ini} ou t_{rev} e $\epsilon \in \left\{0, 1, \frac{2}{S}, 2\frac{2}{S}, 3\frac{2}{S}, \ldots\right\}$. O primeiro valor para ϵ, isto é, $\epsilon = 0$, é um mapeamento constante para a frequência mais baixa e a frequência mais alta. Isto nos assegura que qualquer função de transferência pode ser gerada pelas combinações.

As funções de amplificação da galeria inicial são simples filtros passa-baixa, com fatores negativos ou positivos. O intervalo das frequências harmônicas da variedade $0, \ldots, \#\nu - 1$ é dividido em intervalos I_ϵ para $\epsilon \in \left\{0, 1, \frac{S}{2}, 2\frac{S}{2}, 3\frac{S}{2}, \ldots\right\}$. Depois definimos a função de amplificação para a primeira metade da galeria $a_\epsilon(\nu) = M_a$ se $\nu \in I_\epsilon$ e $a_\epsilon(\nu) = 0$ caso contrário, onde M_a é o fator máximo de amplificação. A outra metade é definida similarmente usando $-M_a$.

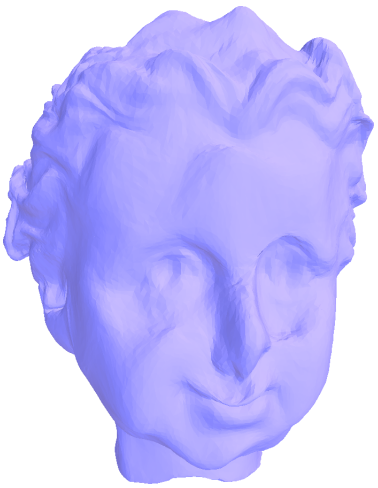
Se as amplitudes do som são normalizadas a $[-1, 1]$ e se a malha é razoavelmente suave, a ordem de grandeza de M_a é 5.000. Como queremos enfatizar as baixas frequências, definimos os intervalos I_ϵ em $\left[\epsilon^2, \left(\epsilon + \frac{2}{S}\right)^2\right]$.

6.3 Efeitos complementares

Para recalcular as normais em Harmônicos de Variedade, usamos um geometry shader que calcula, para cada triângulo, uma normal constante. Esta normal é usada em uma iluminação por pixel via o fragment shader. Entretanto, uma normal constante nos dá um shading facetado. Para obter resultados mais suaves, no geometry shader resolvemos ponderar a normal constante do triângulo com a normal original do vértice.

6.4(a): Modelo sem deformação com as normais originais.

6.4(b): Modelo deformado com as normais originais.



6.4(c): Modelo deformado com as normais deformadas, flat normals

6.4(d): Modelo deformado com normais ponderadas com peso $0,4$ para normais deformadas.

Outro efeito complementar em Harmônicos de Variedade, foi permitir o reuso de mapeamentos harmônicos. O principal obstáculo é que o número de frequências da variedade $\#\nu$ pode ser diferente de modelo para modelo. Como

solução, normalizamos os valores da imagem de t para um intervalo constante $[0,1] : \bar{t}(\xi) = \frac{t(\xi)}{\#\nu}$ e adaptamos o mapeamento $\Phi_{t,\mathrm{a}}$ para

$$\Phi_{t,\mathrm{a}}(\tilde{g})(\nu) = \mathrm{a}(\nu) \cdot \left(\sum_{\xi \epsilon \bar{t}^{-1}\left(\frac{\nu}{\#\nu}\right)} \tilde{g}(\xi) \right) + 1. \tag{6-5}$$

O cálculo das normais na animação com Análise Modal foi baseado no artigo (21), e detalhamos a seguir. O cálculo envolve apenas um geometry shader e um passo de "render-to-normal-buffer". O geometry shader recebe cada triângulo e emite três pontos, posicionados pelo índice de cada vértice do triângulo. A cor de cada ponto é dada por $d^{-1}(v)\mathbf{n_T}$, onde $d^{-1}(v)$ é a inversa do grau do vértice v e $\mathbf{n_T}$ é a normal do triângulo escalonado pela área do mesmo.

Os valores de $d^{-1}(v)$ são armazenados em textura, e escalonados para não acumular além de $\frac{1}{2}$ (ou seja, são divididos por duas vezes a maior área da malha). Assim a normal de cada vértice é dada pela média das normais das faces adjacentes. Esta soma final é obtida usando uma função de *blending* e armazenada em um *framebuffer*.

6.4 Implementação na GPU do filtro harmônico

A principal dificuldade em trabalhar com música em tempo real é calcular e renderizar cada elemento da galeria sincronizado com a música.

Se tivermos S elementos na galeria, cada qual é uma malha com n vértices com $\#\nu$ frequências harmônicas, e $\#\xi$ frequências de música, um único frame realiza $\mathrm{O}(S \cdot \#\nu \cdot \#\xi \cdot 3n)$ operações.

Portanto, propomos uma implementação do filtro harmônico da variedade na GPU, enquanto a decomposição harmônica é realizada na CPU.

- **Implementação** Para a implementação do código em placa gráfica foi usada a linguagem GLSL (3). Usamos um único fragment shader, que calcula o sinal filtrado F_φ para cada coordenada x, y, e z junto com um passo de "render-to-vertex-buffer" para copiar o resultado nas coordenadas dos vértices.

  Enviamos para GPU os harmônicos da variedade por textura: uma textura $\tilde{x}\tilde{y}\tilde{z}$ que contém as amplitudes harmônicas $\tilde{x}(\nu)$, $\tilde{y}(\nu)$, e $\tilde{z}(\nu)$ da malha original, uma textura d_{xyz} contendo a soma das contribuições das altas frequências para cada componente e uma textura H_ν contendo os autovetores dos harmônicos da variedade. O filtro φ deve ser enviado para GPU a cada galeria em cada frame. Como, φ possui um tamanho

menor que as frequências de som $\tilde{g}$ e os vetores t e a, nós o calculamos na CPU e o enviamos como uma textura 1D ϕ.

- **Armazenamento** Todas as texturas são armazenadas usando floats 32 bits para manter a precisão das coordenadas dos vértices. Como o número de vértices é geralmente maior que o tamanho máximo das texturas 1D, usamos duas coordenadas de textura em $\{0, \ldots, [\sqrt{n}] - 1\}$ como índices dos vértices. As contribuições das altas frequências são armazenadas como textura 2D RGB de tamanho $[\sqrt{n}] \times [\sqrt{n}]$, onde as coordenadas são mapeadas para componentes de RGB.

  Como o número de frequências da variedade $\#\nu$ cabe em uma linha de textura, as amplitudes harmônicas originais $\tilde{x}\tilde{y}\tilde{z}$ são armazenadas como textura 1D RGB de tamanho $\#\nu$, onde os componentes $\tilde{x}$, $\tilde{y}$, e $\tilde{z}$ são mapeados para RGB.

  Finalmente, o escalar H_ν e o ϕ do autovetor harmônico de n coordenadas e o filtro podem ser armazenados nos componentes RGBA para otimizar espaço: ϕ é então uma textura $\left[\frac{\#\nu}{4}\right]$ 1D e H_ν uma textura $[\sqrt{n}] \times [\sqrt{n}] \times \left[\frac{\#\nu}{4}\right]$ 3D RGBA.
- **Fragment Shader** Após carregar todas as texturas, a renderização de um único quadrado de tamanho $[\sqrt{n}] \times [\sqrt{n}]$ irá chamar o fragment shader para cada índice de vértice e recalcular as posições dos vértices como framer color, veja figura 6.4. O fragment shader renderiza para uma frame buffer as novas posições dos vértices, o qual é copiado para um vertex buffer na GPU. O shader recebe uma variável uniforme $\delta\nu = \frac{1}{(4\#\nu - 1)}$, a qual é o incremento normalizado para a iteração da frequência da variedade dentro de coordenadas de textura normalizada.

6.5 Implementação na GPU do filtro modal

Ao recalcular as posições dos corpos a partir dos modos normais em cada tempo, guardamos as transformações obtidas que serão usadas para o skinning de cada vértice. Enviamos então apenas duas texturas para a GPU: no pré-processamento a textura $\tilde{x}\tilde{y}\tilde{z}$ que contém as posições dos vértices com as suas respectivas transformações pré-calculadas para cada modo normal, e a cada quadro a textura ϕ contendo os produtos $\mu \sin(2\pi\nu t + \gamma)$ de cada modo normal.

```
uniform sampler1D x̃ỹz̃ ;
uniform sampler2D dxyz ;
uniform sampler1D Hk ;
uniform sampler1D φ ;
uniform float δk ;

void main()
{

    vec3 texcoord = gl_TexCoord[0].stp;
    vec3 pos = texture2D(dxyz, texcoord.st).xyz ;

    for( float k = 0.0; k ≤ 1.0; )
    {
        texcoord.p = k;
        vec4 H = texture3D( Hk, texcoord);
        vec4 f = texture1D( φ , k );

        vec3 x̃ỹz̃0 = texture1D( x̃ỹz̃ , k ).xyz; k += δk;
        vec3 x̃ỹz̃1 = texture1D( x̃ỹz̃ , k ).xyz; k += δk;
        vec3 x̃ỹz̃2 = texture1D( x̃ỹz̃ , k ).xyz; k += δk;
        vec3 x̃ỹz̃3 = texture1D( x̃ỹz̃ , k ).xyz; k += δk;

        pos += f[0] * H[0] * x̃ỹz̃0 + f[1] * H[1] * x̃ỹz̃1 +
               f[2] * H[2] * x̃ỹz̃2 + f[3] * H[3] * x̃ỹz̃3;
    }
    gl_FragColor.rgb = pos.xyz

}
```

Figura 6.4: Fragment Shader para o filtro dos harmônicos da variedade.

Cada elemento de ϕ é enviado a galeria correspondente ao seu modo e como, em geral, a quantidade de modos é pequena, eles são calculados diretamente na CPU.

- **Armazenamento das texturas.** Como cada transformação contém 3 componentes de translação (tx, ty, tz) e mais 3 de rotação (rx, ry, rz), além de termos que guardar a posição inicial do vértice, precisamos de um espaço de pelo menos $2 * \#\nu + 1$ para cada vértice, onde $\#\nu$ é o número de modos obtidos no Modelo Modal. Portanto, os escalares da textura $\tilde{x}\tilde{y}\tilde{z}$ são armazenados em uma textura 3D RGB, de dimensões $\sqrt{\#v} \times \sqrt{\#v} \times (2 * \#\nu + 1)$, onde $\#v$ corresponde ao número de vértices.

$$\tilde{x}\tilde{y}\tilde{z} = [x_0, y_0, z_0, rx_0, ry_0, rz_0, tx_0, ty_0, tz_0, x_1, y_1, \ldots x_{\#v}, y_{\#v}, z_{\#v}, rx_{\#v}, ry_{\#v}, rz_{\#v}, tx_{\#v}, ty_{\#v}, tz_{\#v}].$$

A textura ϕ é apenas um vetor de textura 1D de float, já que consta apenas os $\#\nu$ escalares $\mu \sin(2\pi\nu t + \gamma)$ de cada modo normal.

- **Fragment Shader** Após carregar todas as texturas, chamamos o fragment shader de cada vértice para recalcular as novas posições dos vértices como cores RGB, veja figura 6.5. O fragment shader processa então um frame buffer contendo as novas posições, o qual é copiado para um vertex buffer na GPU. O shader recebe uma variável uniforme que é um incremento normalizado $dm = \frac{1}{2*\#\nu+1}$ para iterar nas coordenadas da textura que são normalizadas.

6.6 Detalhes de implementação

Análise Harmônica em Variedades Para o método de Harmônicos em Variedade usamos a estrutura de dados Compact Half Edge (19) para armazenar a malha.

Para o cálculo dos autovetores e dos autovalores harmônicos da variedade usamos a biblioteca Scalable Library for Eigenvalue Problem Computations (SLEPc) (13). SLEPc além de apresentar um grande número de métodos para cálculo de autovetores, também apresenta interfaces para pacotes bem conhecidos de autovalores.

Para a decodificação do som, usamos a biblioteca FFmpeg(7) e a OpenAL (14) para a decodificação e monitoramento do som. Para a configuração do áudio nos baseamos no tutorial disponível em kcat.strangesoft.net/openal.html.

Análise Harmônica combinada com Modal em Modelos Articulados O ambiente em que desenvolvemos as animações usando Análise Modal e Harmônica para modelos articulados foi o QtCreator. No cálculo das matrizes e do problema de autovalores usamos a biblioteca GNU Scientific Library (GSL) (2). A GSL oferece um grande número de rotinas matemáticas em diversas áreas e desenvolvida para a linguagem C e C++.

```
uniform sampler3D x̃ỹz̃ ;
uniform sampler1D φ ;
uniform float dm ;

void main()
{

    vec3 texcoord = vec3(0.0,gl_TexCoord[0].st ) ;
    vec3 v = texture3D( x̃ỹz̃, texcoord).xyz ;

    vec3 tm = vec3(0.0, 0.0, 0.0);
    vec3 rm = vec3(0.0, 0.0, 0.0);

    for( texcoord.s = dm; texcoord.s ≤ 1.0; )
    {
        float f = texture1D( φ , texcoord.s ).x ;
        tm += f * texture3D( x̃ỹz̃, texcoord ).xyz ; texcoord.s += dm ;
        rm += f * texture3D( x̃ỹz̃, texcoord ).xyz ; texcoord.s += dm ;
    }

    v + = tm ;
    float theta = length( rm ) ;
    rm / = theta ;
    float c = cos(theta) ;
    gl_FragColor.rgb = c∗v + sin(theta) ∗ cross(rm,v) + (1.0-c)∗dot(rm,v)∗rm;

}
```

Figura 6.5: Fragment Shader para o filtro dos harmônicos no modal.

Assim como para Harmônicos em Variedade, para a decodificação do som, usamos a biblioteca FFmpeg (7) e a OpenAL (14) para a decodificação e monitoramento do som. Além disso, também usamos as mesmas configurações de som do Análise Harmônica em Variedades.

Como estrutura de dados usamos listas de instâncias de classes para juntas e corpos rígidos. Para o armazenamento e manipulação da malha, usamos a estrutura de dados Compact Half Edge (19).