4
Implementação

O principal objetivo da implementação é construir um framework geral o suficiente para ser aplicado em diversos tipos de visualizador científico com o mínimo de esforço. É parte do escopo deste projeto a aplicação do framework desenvolvido em dois sistemas comerciais distintos, como será discutido na Seção 5.

Este capítulo, no entanto, não pretende descrever cada minúcia da biblioteca desenvolvida ou mesmo explicar o funcionamento de cada classe. Ao invés disso, a intenção é discutir decisões de projeto relevantes e pontos de otimização na implementação.

4.1
Visão Geral da Biblioteca

A biblioteca foi desenvolvida com C++ e OpenGL, utilizando GLSL em todas as etapas de processamento. A responsabilidade da biblioteca, no entanto, começa apenas a partir da primeira etapa de LIC (descrita na Seção 3.4.3), não incluindo, portanto, as etapas de pré-processamento e projeção.

Esta decisão é, em um primeiro momento, estranha. Tanto a projeção quanto a definição de coordenadas de textura aleatórias são feitas de maneira padronizada e deveriam fazer parte da biblioteca. Ela se justifica, entretanto, quando se considera que abstrair o modelo diretamente dificultaria o processo de instanciação do framework, já que é a organização dos dados do modelo pode ser radicalmente diferente de um sistema para outro. O esforço de adequação a uma abstração imposta pelo framework, além de poder se tornar difícil para o cliente, pode ter impacto negativo no desempenho.

Sendo assim, optamos por gerar texturas com informações sobre o modelo e realizar a projeção a partir delas. O framework automatiza todo o processo a partir da etapa de LIC, descrita na Seção 3.4.3. Basta que o cliente execute as etapas de pré-processamento e projeção e forneça as três imagens de entrada. Todo o resto é feito de maneira transparente pela biblioteca.

4.2
Instanciando um Campo

A biblioteca espera que o cliente implemente uma interface simples, responsável simplesmente por processar as três texturas de entrada do algoritmo: campo vetorial, coordenadas de textura e iluminação.

Como todas as imagens dependem de renderização do modelo, é recomendado que sejam todas processadas em uma única passada, utilizando mecanismos de *multiple render targets* (MRT).

A principal preocupação do cliente da biblioteca é ser capaz de renderizar seu modelo atribuindo a cada vértice uma normal, um vetor do campo e uma coordenada de textura.

A coordenada de textura pode ser gerada aleatoriamente, mas valores não aleatórios podem funcionar bem em algumas situações. O principal requisito é que vértices vizinhos possuam coordenadas distintas de modo que cada pixel na imagem final acesse regiões diferentes da imagem de ruído. Desta forma, se houver alguma outra variável que prontamente atenda a tal requisito, ela pode ser usada. Caso contrário, valores aleatórios devem ser usados. O procedimento de projeção para a criação da textura de campo vetorial é simples e pode ser realizado com o trecho de código GLSL na listagem 4.3.

A partir deste ponto, basta informar à biblioteca os identificadores das texturas geradas e solicitar a renderização do campo vetorial. No caso de uso de escalas de cor, é necessário que se informe, além da própria escala a ser usada, os valores mínimo e máximo a serem considerados.

4.3 A Biblioteca

Tendo sido fornecidas pelo cliente as imagens de campo vetorial, de coordenada de textura e de iluminação, a biblioteca se torna responsável pela aplicação do LIC em duas passadas e pela aplicação do filtro de passa-alta. A aplicação é tal como descrito nas Seções 3.4.3 e 3.4.4. Para um melhor desempenho, todos os cálculos são realizados inteiramente em *fragment shaders*.

```
1  // f, normal e vertex são propriedades fornecidas por vértice
2  // Este processamento só faz sentido se ||f|| > 0
3
4  // Magnitude e Normalização
5  float mag = length(f);
6  f /= mag;
7
8  // Projeção para Superfície
9  float fn = dot (normal, f);     // Componente Normal
10 vec3 ft = vecIn - fn * normal; // Vetor tangente à superfície
11
12 // Projeção para Espaço da Tela
13 vec4 vp = gl_ModelViewProjection * vec4 (vertex, 1.0);
14 vec4 ap = gl_ModelViewProjection * vec4 (vertex + ft, 1.0);
15
16 vec2 fs = (ap.x - vp.x, ap.y - vp.y); // Equivalente a f't
17 fs.x = width/height;  // Ajustando aspecto da tela
18 fs = normalize(fs);   // LIC exige campo normalizado
19
20 // Resultado
21 output = (fs, fn, mag);
```

Tabela 4.1: Código em GLSL para projeção de um vetor do modelo.