3
Técnica Proposta

Neste capítulo, o algoritmo LIC (2) será revisitado e questões relativas à manipulação dos dados de entrada serão consideradas. Serão também apresentadas particularidades e melhorias da implementação adotadas. Em seguida, será introduzida a técnica proposta, em sua versão mais simples. Problemas resultantes serão discutidos e suas respectivas soluções serão apresentadas. Finalmente, a técnica será revista em sua versão completa, como um resumo das soluções propostas.

3.1
LIC Revisitado

Como discutido nas Seções 2.1 e 2.2, podemos considerar um algoritmo 2D de visualização de campos vetoriais contendo três partes. Inicialmente, o LIC (2) é executado com base no campo vetorial e uma imagem de ruído branco. Executa-se novamente o LIC, utilizando a imagem gerada anteriormente no lugar da imagem de ruído. Finalmente, é executado um filtro passa-alta sobre esta segunda imagem gerada.

No contexto específico de programação em placa gráfica, consideramos que tanto o campo vetorial quanto a imagem de ruído estão armazenadas em texturas. Para gerar as imagens intermediárias, podemos utilizar renderização direta em textura e, na última etapa, renderizamos diretamente na tela.

Na primeira etapa, podemos utilizar mecanismos de repetição de textura na imagem de ruído, de forma a permitir o uso de imagens de ruído de menor resolução. Na segunda etapa, no entanto, cada pixel na "imagem de ruído" (resultado do passo anterior) deve corresponder a um único pixel na imagem final.

A textura representando o campo vetorial deve, também, estar alinhada com a imagem final e os vetores representados devem ser bidimensionais (estamos, afinal, usando um algoritmo 2D). Entretanto, para melhor calcular os segmentos da linha de fluxo (ver Seção 2.1.1), é necessário que esse vetor 2D seja normalizado.

Se tal normalização fosse realizada durante a execução do LIC, estaríamos repetindo trabalho desnecessariamente, já que cada pixel normalizaria tantos vetores quantos pixels forem atingidos por sua linha de fluxo. Evitamos esta situação normalizando as amostras do campo no momento da geração da textura de campo vetorial. Para ser possível a aplicação posterior de uma

escala de cores, armazenamos a magnitude do vetor em outro canal de cor desta textura.

3.1.1 Magnitude e animação

A primeira modificação de fato no algoritmo se dá no cálculo da animação. Como visto na Seção 2.1.2, esta é regida pelo seguinte kernel:

$$k(w) = \frac{1 + \cos(cw)}{2} \times \frac{1 + \cos(dw + \beta)}{2} \tag{3-1}$$

Sendo β o fator a ser variado de quadro para quadro. Repare que, durante todo o algoritmo, a magnitude dos vetores é irrelevante, importando somente sentido e direção.

De fato, a animação resultante é uniforme. Todo o campo se move com uma mesma velocidade, ditada pelo fator β. Já que possuímos informação sobre a magnitude e cada pixel é calculado de forma independente, poderíamos aplicar um fator de escala λ em β para utilizar a magnitude como um indicador de velocidade da animação. Teríamos desta forma o seguinte kernel:

$$k'(w) = \frac{1 + \cos(cw)}{2} \times \frac{1 + \cos(dw + \lambda\beta)}{2} \tag{3-2}$$

Inicialmente utilizamos λ considerando a magnitude normalizada ($\bar{m}$). Ou seja, em um dado ponto i, sendo m a magnitude da amostra correspondente, teríamos:

$$\lambda_i = \bar{m}_i = \frac{m_i}{m_{\max}} \tag{3-3}$$

No entanto, com este fator pixels vizinhos ao longo de uma dada linha de fluxo sairiam de fase e, após alguns frames, a animação teria seu sentido invertido, em particular em regiões de alta magnitude. Para manter tais pixels em fase ao ajustar a velocidade de animação, propomos um fator de velocidade proporcional à magnitude relativa *quantizada*. Nos testes que realizamos foi utilizado o seguinte fator λ_i:

$$\lambda_i = 0.1 \left\lfloor \frac{\bar{m}_i}{0.1} \right\rfloor \tag{3-4}$$

Desta forma, áreas de maior magnitude se movem mais rapidamente, mas todos os pixels em uma mesma área se movem em fase a uma mesma velocidade. Naturalmente, é criada uma descontinuidade nos limites entre regiões. No algoritmo original de LIC (2) e mesmo na versão com duas passadas (6), esta descontinuidade não é facilmente percebida no resultado. A aplicação do filtro de passa-alta, porém, tende a realçá-las.

Para tornar as descontinuidades menos aparentes e não prejudicar a qualidade da imagem final, é possível usar um filtro passa-alta dinâmico. Isto é, ele deve desconsiderar pixels onde λ_i é diferente de λ_{centro}.

Sendo O um pixel de fator diferente, X um pixel de mesmo fator e C o centro do kernel, teríamos, por exemplo:

$$\begin{bmatrix} X & X & X \\ O & C & X \\ O & O & X \end{bmatrix} \Longrightarrow \begin{bmatrix} -1 & -1 & -1 \\ 0 & 6 & -1 \\ 0 & 0 & -1 \end{bmatrix}$$

Ou seja, o peso de um pixel de mesmo fator é -1; de fator diferente, 0; e o pixel central é um a mais que a quantidade de pixels de mesmo fator. Com isso, as descontinuidades na imagem final tornam-se mais suaves, ainda que não sejam totalmente eliminadas. No entanto, esta suavização é o suficiente para que tais descontinuidades se tornem menos aparentes e menos incômodas durante a animação.

3.1.2 Modelo

Outro ponto a ser considerado diz respeito à taxa de amostragem do campo utilizada. Espera-se que exista uma amostra do campo por pixel da imagem de saída, mas normalmente temos amostras distribuídas por elementos da geometria da superficie. Por exemplo, pode existir uma amostra por vértice ou por célula do modelo. Por maior que seja a quantidade de amostras, se considerarmos a possibilidade de ampliação da imagem, eventualmente será necessário obter valores intermediários a partir da interpolação de amostras.

Como o algoritmo espera que o campo esteja em uma textura, a própria placa gráfica pode funcionar como interpoladora. Na prática, basta renderizar o modelo normalmente, informando um vetor por vértice (caso o campo esteja definido por célula, é necessário um pré-processamento para suavizar valores por vértice).

Durante a geração da imagem de campo vetorial, é necessário armazenar amostras normalizadas, por imposição da nossa implementação algoritmo LIC, e manter informação sobre magnitude em um canal de cor separado, para permitir coloração e animação com velocidade variável.

3.2 Trabalhando no Espaço da Tela

Até o momento, não nos preocupamos em momento algum com a tridimensionalidade do modelo e do campo. Inclusive definimos, na Seção 3.4.3,

que o campo vetorial considerado para aplicação do LIC é bidimensional.

Para que possa ser usado como entrada no algoritmo de LIC, o campo de entrada é processado em duas etapas. Inicialmente, como estamos utilizando a geometria da superfície como suporte geométrico, optamos por utilizar a componente do campo vetorial tangencial à superfície. O segundo passo consiste em projetar o campo tangente para o espaço da tela, de modo a obter um campo tangente bidimensional, como é esperado pelo algoritmo de LIC.

Dado um determinado ponto do modelo, assumimos que $\vec{f}$ é a amostra do campo, $\hat{n}$ é a normal, $\mathbf{v}$ é o vértice do modelo e $\mathbf{M}$ é a matriz de Model-View-Projection corrente. A Figura 3.1 ilustra as duas etapas discutidas.

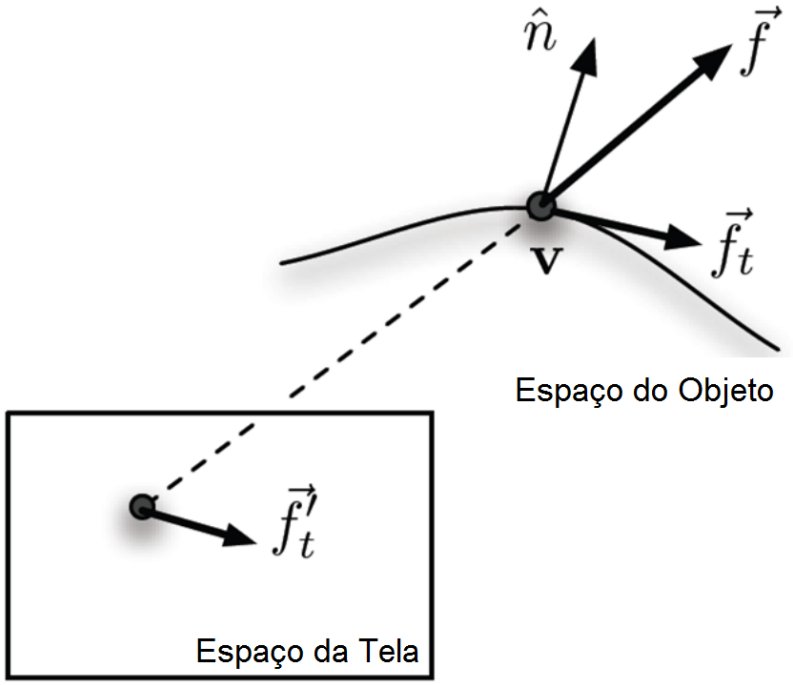

Figura 3.1: Projeção de um vetor do campo no espaço da tela.

A partir de $\vec{f}$, obtemos $\hat{f}$ e $||\vec{f}||$ e desejamos inicialmente obter um vetor $\vec{f}_t$ tangente à superfície:

$$f_n = \hat{f} \cdot \hat{n} \tag{3-5}$$

$$\vec{f}_t = \hat{f} - f_n \hat{n} \tag{3-6}$$

Em seguida, devemos transportar $\vec{f}_t$ para o espaço da tela:

$$\vec{f}'_t = \mathbf{M}(\mathbf{v} + \hat{f}_t) - \mathbf{M}\mathbf{v} \tag{3-7}$$

Finalmente, caso a janela de desenho não seja quadrada, a matriz de projeção incluída em $\mathbf{M}$ aplica a $\vec{f}'_t$ uma escala não uniforme que deve ser

corrigida. Sejam w e h respectivamente a largura e altura da janela de desenho, fazemos:

$$\vec{f_t'} = (x, y) \tag{3-8}$$

$$\vec{f_t''} = (x\frac{w}{h}, y) \tag{3-9}$$

Onde $\hat{f_t''}$ é o vetor a ser usado pelo algoritmo de LIC (2).

3.3 Problemas e Soluções

A técnica proposta até este ponto é capaz de gerar as imagens desejadas, mas há ainda algumas melhorias e correções a serem feitas, que serão discutidas nesta seção.

3.3.1 Escala de cores para magnitude

Uma das idéias proposta em (2) é utilizar uma escala de cores para representar a magnitude. No nosso caso, este mapeamento de cores seria um indicador complementar à velocidade variável de animação discutida na Seção 3.2.

Um dos primeiros subprodutos do processo de projeção descrito na Seção 3.2 é a magnitude da amostra do campo, $||\vec{f}||$. Como se trata de uma magnitude absoluta, é necessário definir uma magnitude global máxima $f_{\max}$ para a aplicação da escala. Desta forma, podemos fazer:

$$\bar{m} = \frac{||\vec{f}||}{f_{\max}} \tag{3-10}$$

Em seguida utilizamos $\bar{m}$ para indexar uma escala de cores 1D. O valor obtido da escala pode ser diretamente multiplicado pela cor final obtida pelo processamento do LIC. A Figura 3.2 ilustra a aplicação de uma escala branco-verde a um modelo de reservatório de petróleo.

3.3.2 Escala Bidimensional de Cores

Como discutido na Seção 3.2, utilizamos no algoritmo de LIC (2) a componente do campo tangencial à superfície. Em outras palavras, a componente normal à superfície é descartada, o que é um problema em regiões onde o campo não está alinhado à superfície.

Um dos subprodutos da etapa de projeção é f_n (Equação 3-5) que, por ser calculado a partir de $\hat{f}$, é um valor em $[-1, 1]$ que indica o quão desalinhado

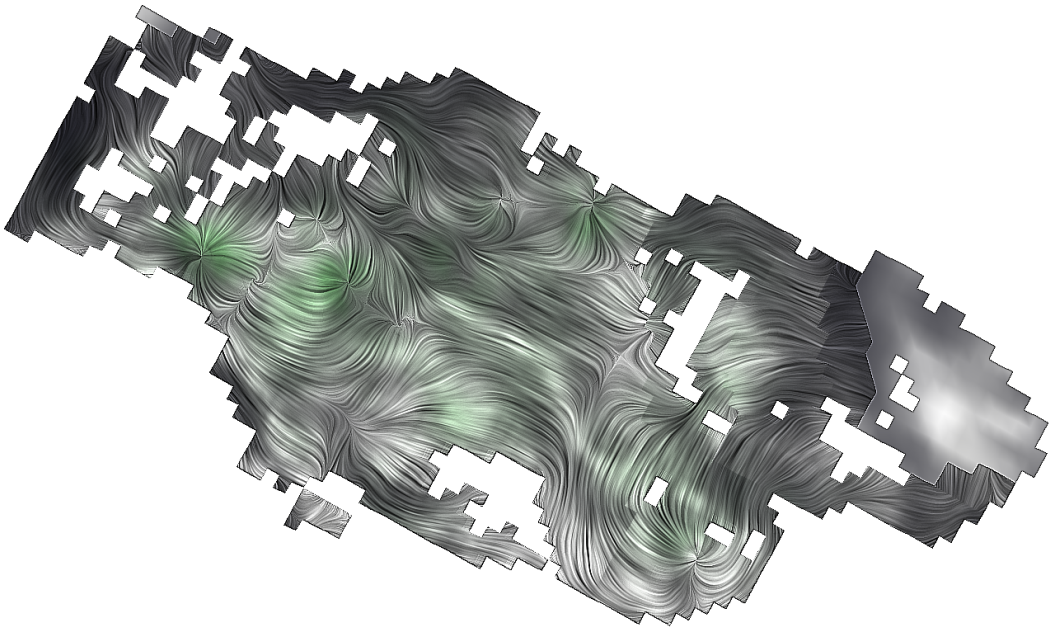

Figura 3.2: Aplicação de escala de cores (Branco-Verde) representando magnitude.

o vetor está e o sentido que possui em relação à normal. Valores positivos indicam que há uma componente "saindo" da superfície; valores negativos indicam vetores "entrando" na superfície; um valor zero indica que o vetor está inteiramente alinhado com a superfície.

A mesma estratégia discutida na Seção 3.3.1 poderia ser aplicada aqui, indexando uma escala de cores a partir de $\bar{f}_n$, calculado da seguinte forma:

$$\bar{f}_n = \frac{f_n + 1}{2} \tag{3-11}$$

A Figura 3.3 ilustra a aplicação de uma escala Azul-Branco-Vermelho representando a componente normal à superfície.

Desta maneira, no entanto, estaríamos restritos a representar apenas magnitude ou apenas a componente normal. Para resolver este problema, podemos utilizar uma escala bidimensional de cores, onde o eixo x indica componente normal e o eixo y, magnitude. Uma escala 2D está ilustrada na Figura 3.4.

Aplicando tal escala podemos condensar a informação contida nas Figuras 3.2 e 3.3 em uma única imagem. A Figura 3.5 ilustra este resultado.

3.3.3
Fluxo não-alinhado e ambiguidade

Van Wijk (12), ao aplicar uma estratégia de projeção semelhante à deste trabalho, chegou à conclusão de que em alguns casos um campo vetorial 3D

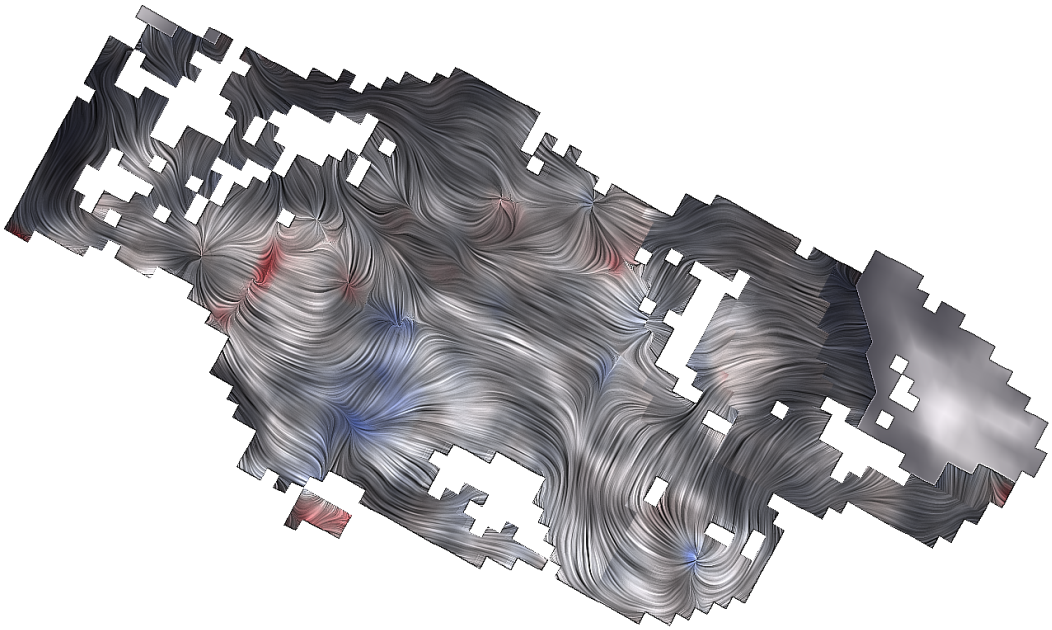

Figura 3.3: Aplicação de escala de cores (Azul-Branco-Vermelho) representando componente normal.

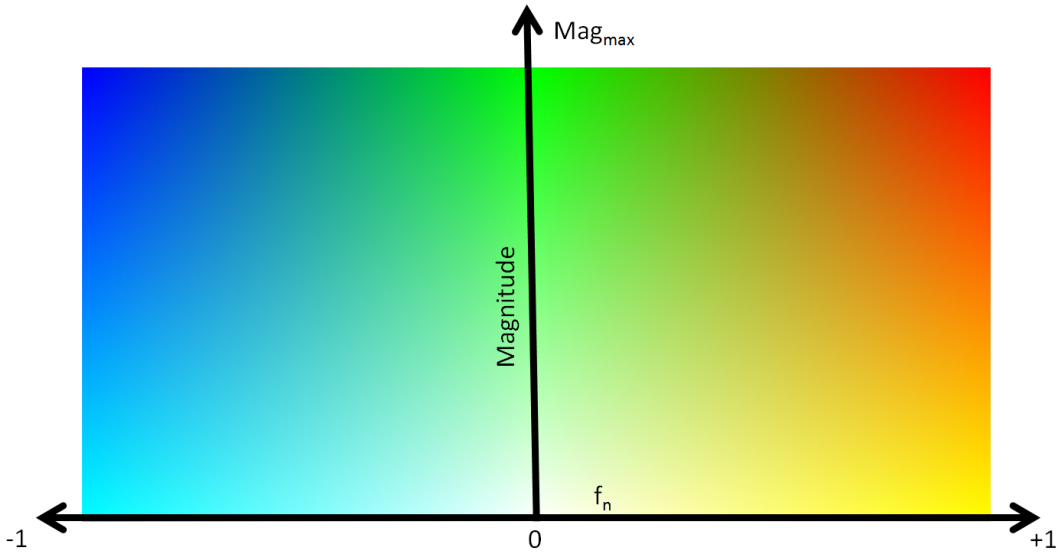

Figura 3.4: Escala bidimensional de cores para representar magnitude e componente normal.

pode ser melhor representado sem nenhum tipo de alinhamento.

Imagine, por exemplo, um campo vetorial homogêneo $\hat{f} = (0, 1, 0)$ representado sobre um toro, como ilustrado na Figura 3.6.

Compare as imagens na Figura 3.7, geradas respectivamente com e sem alinhamento. Van Wijk argumenta que a imagem sem alinhamento melhor representa o campo, por mostrar mais eficientemente que se trata de um campo

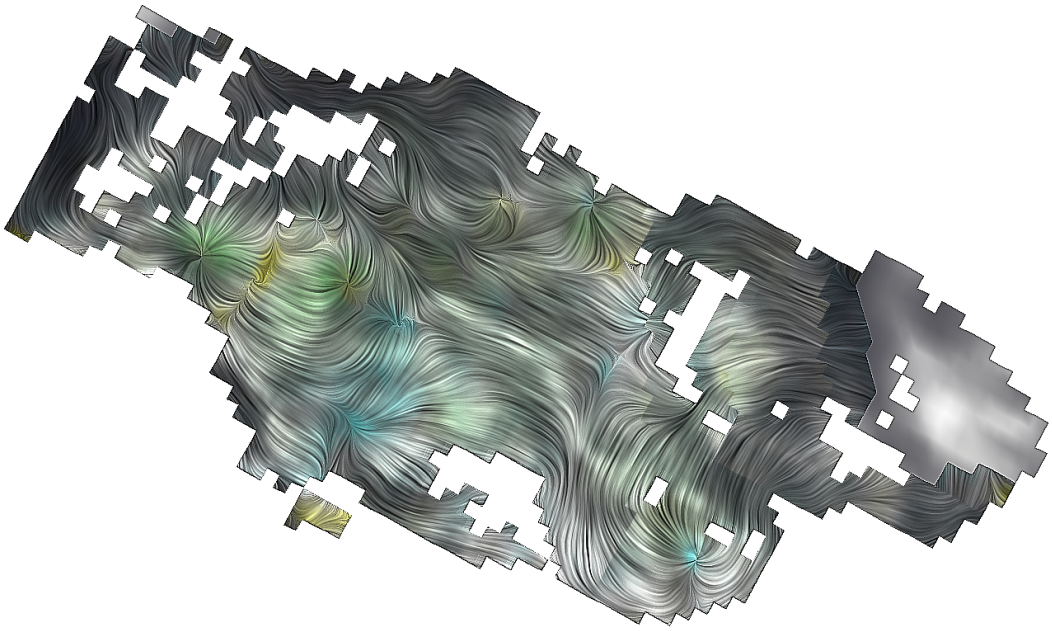

Figura 3.5: Aplicação de paleta de cores representando componente normal e magnitude.

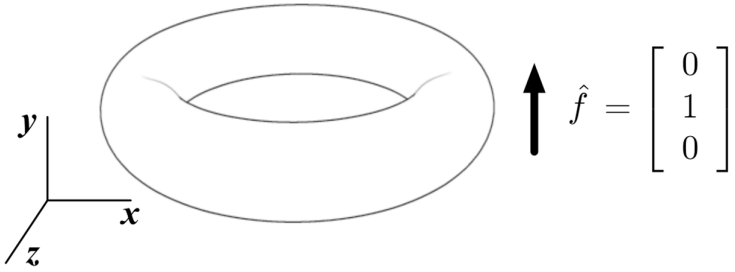

Figura 3.6: Campo vetorial homogêneo representado sobre um toro.

homogêneo.

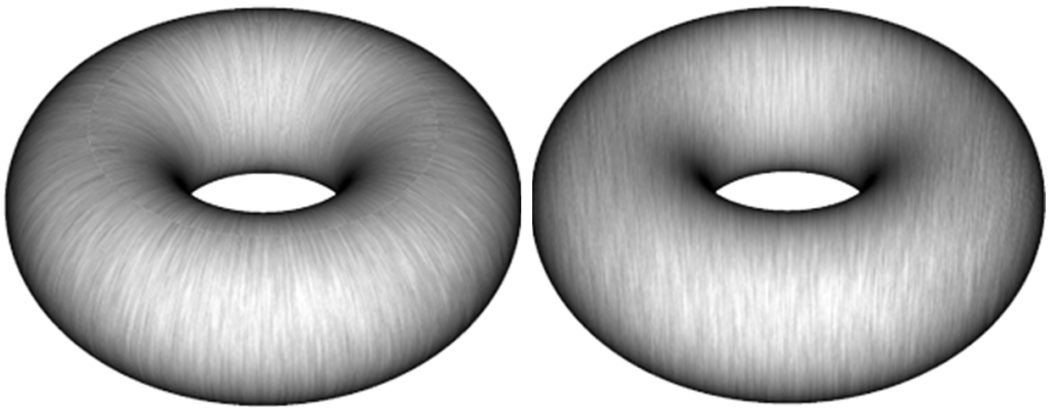

Figura 3.7: Campo homogêneo $\hat{f} = (0, 1, 0)$ sobre toro com e sem alinhamento, respectivamente.

Se, no entanto, repetirmos o experimento considerando um campo homogêneo onde $\hat{f} = (0, \sqrt{2}, -\sqrt{2})$, obteremos os resultados da Figura 3.8. Repare que os dois campos geraram imagens não projetadas similares, provocando uma ambiguidade. Nossa proposta é usar o campo alinhado em conjunto com a escala bidimensional de cores descrita na Seção 3.3.2. A existência de fluxo não alinhado será facilmente identificada sem que se crie nenhum tipo de ambiguidade. A Figura 3.9 ilustra estes resultados.

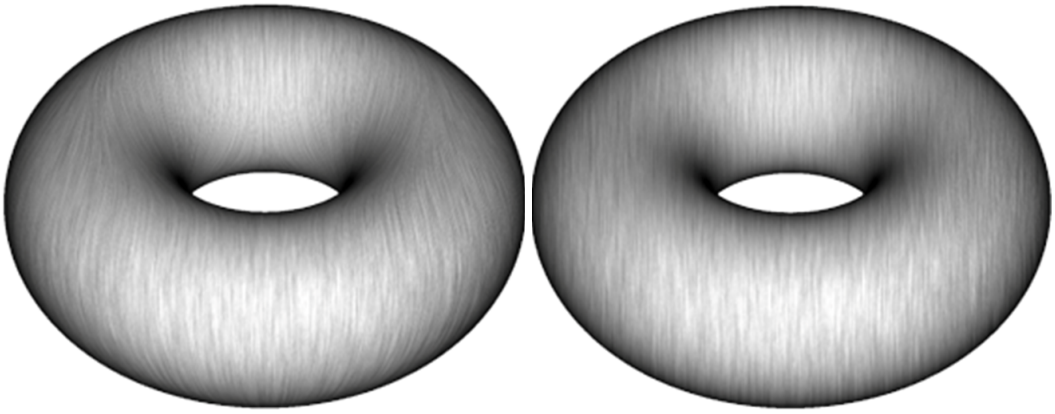

Figura 3.8: Campo homogêneo $\hat{f} = (0, \sqrt{2}, -\sqrt{2})$ sobre toro com e sem alinhamento, respectivamente.

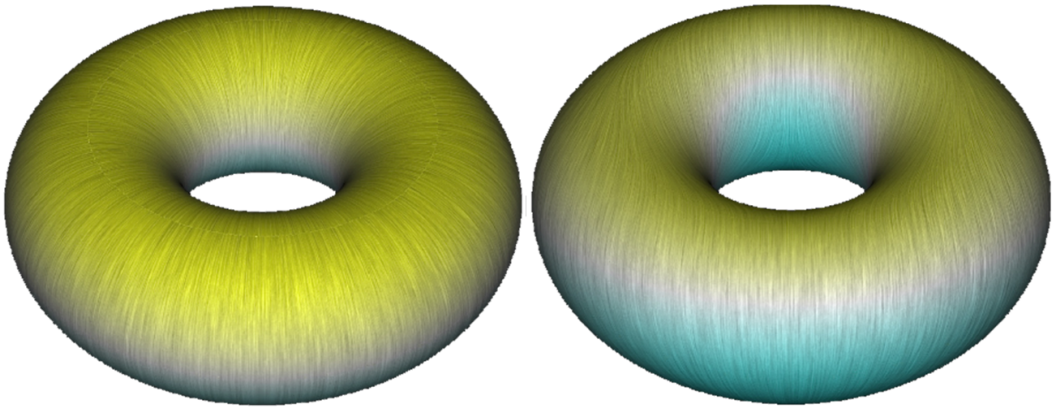

Figura 3.9: Campos homogêneos $\hat{f} = (0, 1, 0)$ e $\hat{f} = (0, \sqrt{2}, -\sqrt{2})$, respectivamente, alinhados e com paleta de cores.

3.3.4
Coerência entre quadros

Outro problema decorrente de se trabalhar no espaço da tela diz respeito à coerência entre quadros durante a manipulação do modelo. Como foi discutido na Seção 2.1.1, um dos motivos para o funcionamento do LIC (2) é o fato

de que linhas de fluxo paralelas acessam regiões completamente diferentes na imagem de ruído branco e, portanto, adquirem cores diferentes.

A região acessada depende da posição do ponto na imagem de ruído. Se deslocarmos o modelo 1 pixel para a direita, a cor de potencialmente todos os pontos irá mudar, criando um efeito de *flickering*. Este efeito, por vezes, cria também a indesejada ilusão de que o campo vetorial está mudando conforme se manipula o modelo.

A idéia para resolver este problema é simples: um ponto do modelo deve corresponder sempre a um mesmo ponto na imagem de ruído branco, independentemente da posição da câmera. Em 2D, se não nos preocuparmos com aproximações e rotações, podemos facilmente aplicar a idéia. Basta que fixemos um ponto qualquer e levemos em conta o deslocamento do mesmo durante o acesso à imagem de ruído.

Em 3D, uma solução possível seria o uso de uma textura 3D sólida, mapeando o ruído branco no espaço do objeto e realizando a convolução no espaço da tela. No entanto, o uso de texturas 3D ocuparia um grande espaço de memória e poderia ter impacto negativo no desempenho do algritmo. Para evitar tais problemas, propomos uma nova estratégia para atribuir coordenadas de textura aos vértices do modelo permitindo o uso de uma imagem de ruído 2D sem *flickering*.

Inicialmente, atribuímos coordenadas de textura aleatórias para cada vértice. Como resultado, triângulos do modelo serão mapeadas em regiões arbitrárias no espaço da textura. A probabilidade de um mapeamento de textura degenerado, isto é, vértices do triângulo acessarem um exato mesmo ponto do ruído, é pequena o suficiente para assumirmos que não acontece.

Após a projeção, triângulos são mapeados para tela e podem sofrer magnificação ou redução de textura, conforme necessário. Nós tentamos evitar a magnificação de texturas atraves do uso de uma imagem de ruído branco grande (1024×1024 se mostrou o suficiente em nossos experimentos). A redução é resolvida com o uso de *mipmapping*, onde cada nível da pirâmide de *mipmapping* contém uma imagem de ruído branco totalmente diferente. O acesso ao mipmapping é feito através de interpolação linear entre níveis diferentes e um determinado nível. Assim, garatimos uma transição suave através dos níveis mas preservamos a distribuição aleatória das amostras de ruído usadas.

Com a câmera posicionada, já havíamos gerado uma imagem com dados de campo vetorial, a ser utilizada pelo LIC. Nesta mesma etapa será gerada uma segunda textura, contendo os valores de coordenada de textura (s, t) gerados. Como tais valores estão associados a vértices do modelo, cada vértice

nessa imagem terá sempre um mesmo par, independentemente da posição da câmera. Chamaremos esta nova textura de *imagem de coordenada de textura*.

A aplicação desta estratégia garante coerência entre quadros durante a manipulação do modelo, o que permite que o usuário final tenha claramente a sensação de estar manipulando algo tridimensional.

3.3.5 Armazenando o Z-Buffer

Utilizando um canal livre da coordenada de textura descrita na Seção 3.3.1, podemos armazenar os valores de Z-Buffer relativos ao modelo. Este armazenamento é necessário porque não há informação de profundidade durante a renderização da imagem final de LIC, visto que ela é gerada a partir de imagens e não de um modelo.

A primeira utilidade do dado de profundidade diz respeito à possibilidade de descarte de fragmentos externos ao modelo, diferenciando-os de regiões pertencentes ao campo mas de magnitude zero. Regiões onde o campo vetorial é nulo podem, por exemplo, ser identificadas por uma cor padrão. Além disso, é possivel identificar descontinuidades no modelo e interromper a linha de fluxo caso a diferença de profundidade ultrapasse um valor pré-determinado.

Em algumas situações pode também ser interessante posicionar no modelo objetos como poços de petróleo ou algum texto relevante. Para isso é interessante que a renderização final possua informações de profundidade.

3.3.6 Iluminação

Devido à natureza do algoritmo e ao fato da última etapa do algoritmo ocorrer em 2D, pode ser difícil perceber na imagem final que se trata de um modelo 3D. Para que o usuário perceba claramente se tratar de um modelo tridimensional, é desejável acrescentar iluminação à imagem final.

Como a aplicação da iluminação deve ocorrer após o algoritmo de LIC, etapa na qual não há informação sobre o modelo, é necessário gerar uma imagem de luminância intermediária a ser aplicada no final. Esta imagem pode ser criada na etapa de geração da textura de campo vetorial utilizando.

3.4 Técnica Final

Podemos resumir a técnica, com todas as melhorias propostas nas seções anteriores, em um algoritmo de 4 etapas.

3.4.1
Pré-processamento

O único pré-processamento necessário é a criação de uma imagem de ruído branco para uso no algoritmo de LIC (2). Para melhor funcionamento com a estratégia de acesso indireto discutida na Seção 3.3.4, sugerimos que esta imagem tenha tamanho de 1024×1024. É necessário ainda construir uma pirâmide de *mipmapping* para a imagem.

Caso se deseje linhas mais grossas na imagem final, o ruído gerado deve ser esparso, isto é, ao invés de definir valores aleatórios para cada pixel de forma independente, devem ser atribuidos valores aleatórios para cada grupo de 4 ou 9 pixels vizinhos.

Além disso, espera-se que o modelo possua para cada vértice uma amostra do campo vetorial, uma normal e um par de coordenadas de textura geradas aleatoriamente. Conforme a necessidade, pode ser considerado como parte da etapa de pré-processamento garantir tais condições.

3.4.2
Projeção

Em seguida, o modelo é renderizado para gerar três texturas intermediárias: Campo Vetorial, Coordenada de Textura e Iluminação. A Figura 3.10 resume ilustra as duas primeiras etapas.

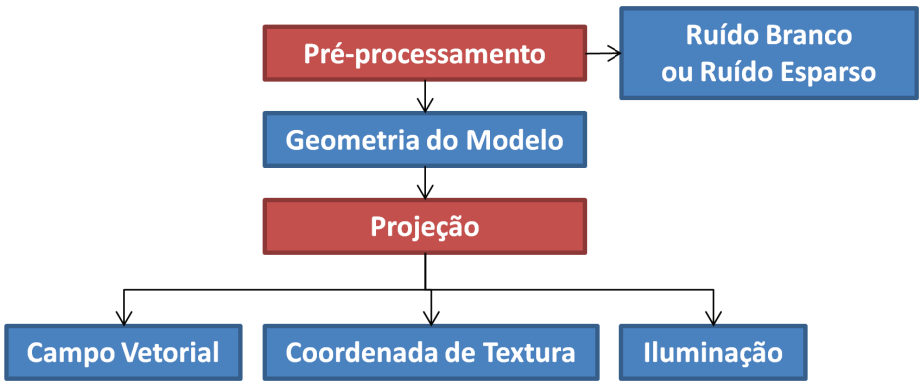

Figura 3.10: Etapas de pré-processamento e projeção.

A Tabela 3.1 mostra o conteúdo das texturas geradas nesta etapa.

3.4.3
Convolução por Integral de Linha

Nesta etapa, as texturas de campo vetorial e coordenada de textura são utilizadas na execução das duas passadas do algoritmo de LIC (ver Seção 2.2.1).

Textura	R	G	B	A
Campo Vetorial	$\hat{f}'_t$		f_n	$\|\|\vec{f}\|\|$
Coord. Textura	Coord. Tex. Aleatória		Z-Buffer	–
Iluminação	Luminância			

Tabela 3.1: Conteúdo das texturas geradas na etapa de projeção.

A primeira passada do algoritmo é tal como descrita na Seção 2.1.1, a única diferença sendo o acesso indireto à textura de ruído. O resultado deve ser armazenado em uma textura.

Na segunda passada, o resultado da etapa anterior deve ser utilizado como ruído. Não há, portanto, necessidade de mecanismos de repetição de textura nem de acesso indireto. A imagem é novamente armazenada em textura. A Figura 3.11 esquematiza esta etapa e seus produtos.

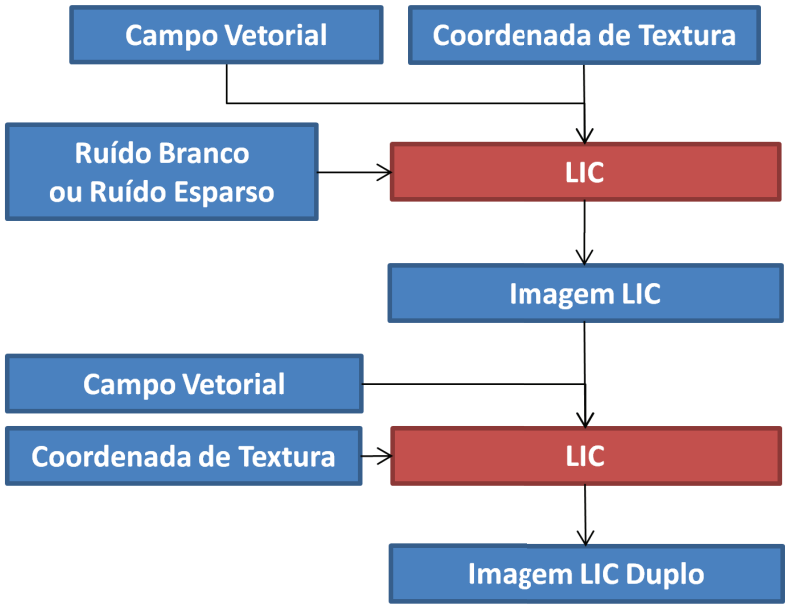

Figura 3.11: Etapa de Convolução por Integral de Linha.

3.4.4 Filtro Passa-Alta

Finalmente, sobre a imagem da etapa anterior é aplicado um filtro de passa-alta. Caso um quadro de animação esteja sendo renderizado, cuidados especiais devem ser tomados (ver Seção 3.1.1). De forma semelhante à animação, devem ser excluídos do filtro regiões externas ao modelo.

Após o filtro, é aplicada a coloração (a partir de uma escala de cores ou da paleta descrita na Seção 3.3.2) e, em seguida, a iluminação. Por último, são

preenchidos dados de profundidade. A imagem gerada neste ponto é final e pode ser renderizada diretamente para a tela. A Figura 3.12 apresenta a etapa final da ténica.

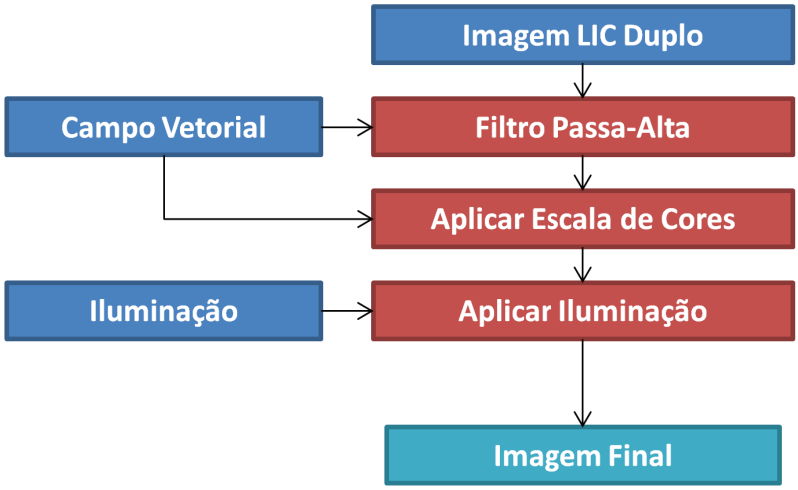

Figura 3.12: Etapa de Aplicação do Filtro Passa-Alta.