2
Trabalhos Relacionados

Este capítulo se concentra em resumir as idéias contidas nos artigos que serviram de base para este trabalho e em discutir brevemente artigos com idéias semelhantes porém executadas de maneira distinta.

É particularmente importante que se entenda o funcionamento do algoritmo de convolução por integral de linha (2) para melhor compreensão da técnica proposta.

2.1
Convolução por Integral de Linha

Proposto por Cabral e Leedom (2), o algoritmo de LIC apresenta uma estratégia densa para visualização por convolução de imagens de ruído branco com base em um campo vetorial bidimensional, de modo a gerar imagens como a da Figura 2.1.

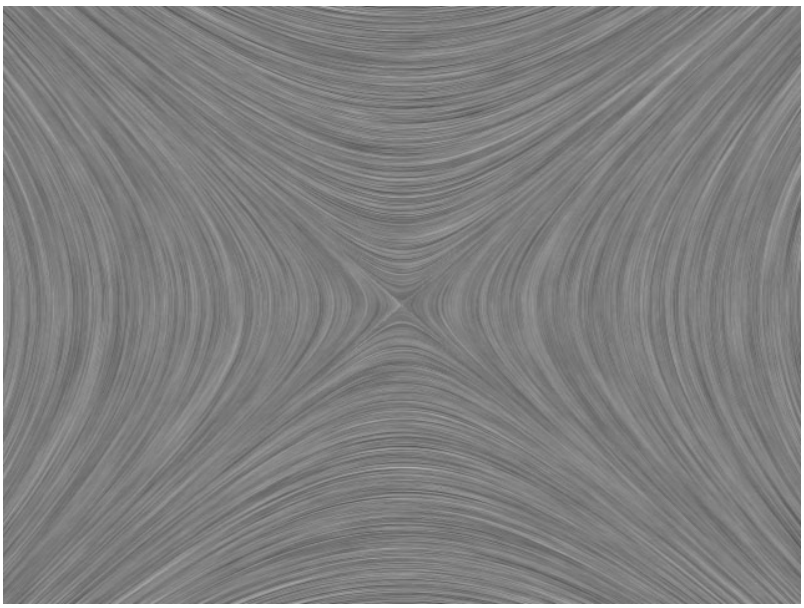


Figura 2.1: Imagem típica do algoritmo de convolução por integral de linha.

2.1.1
Algoritmo

Inicialmente, considera-se que, para cada pixel na imagem de saída, existe uma correspondente amostra do campo vetorial e um pixel na imagem de ruído branco.

Assumindo pixels como quadrados de lado $1u.m.$, a cor de cada pixel na imagem final é calculada considerando os seguintes passos: Partindo do centro do pixel, é traçado um segmento de reta acompanhando o vetor amostrado correspondente (advecção positiva) até que seja atingida uma borda de pixel. A partir do ponto onde o segmento parou, consideramos o vetor amostrado para o pixel atingido e traçamos um novo segmento de reta de maneira similar. O procedimento é repetido sucessivamente até que a soma do tamanho dos segmentos de reta atinja um valor pré-definido L.

O mesmo procedimento é executado considerando o sentido inverso dos vetores (advecção negativa). O conjunto de segmentos obtidos pelas duas advecções constitui uma linha de fluxo, como mostrado na Figura 2.2.

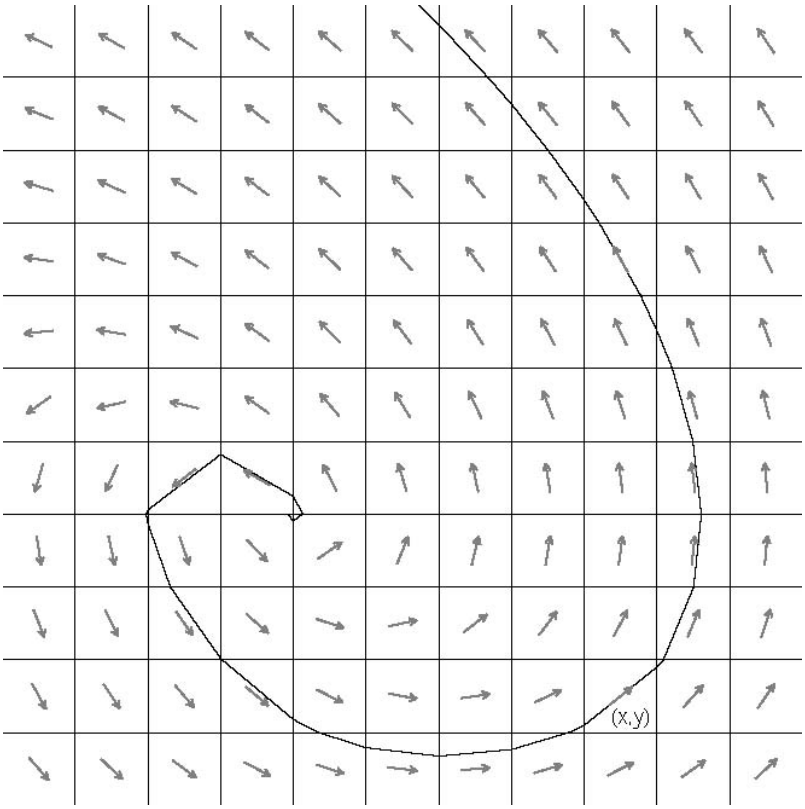


Figura 2.2: Linha de fluxo avaliada para um pixel no algoritmo de LIC. Imagem extraída do artigo original (2).

A cor final do pixel sendo avaliado é dada por uma convolução que pode ser entendida como uma média aritmética ponderada das cores de ruído branco correspondentes a cada pixel atingido pela linha de fluxo, utilizando como peso o tamanho do segmento da linha que passa por este pixel. A Figura 2.3 resume os passos para definição da cor de um pixel.

Para melhor entendimento do algoritmo, consideremos duas situações: o que acontece quando dois pixels adjacentes estão sobre uma mesma linha de fluxo? E quando se encontram em linhas paralelas? No primeiro caso, como

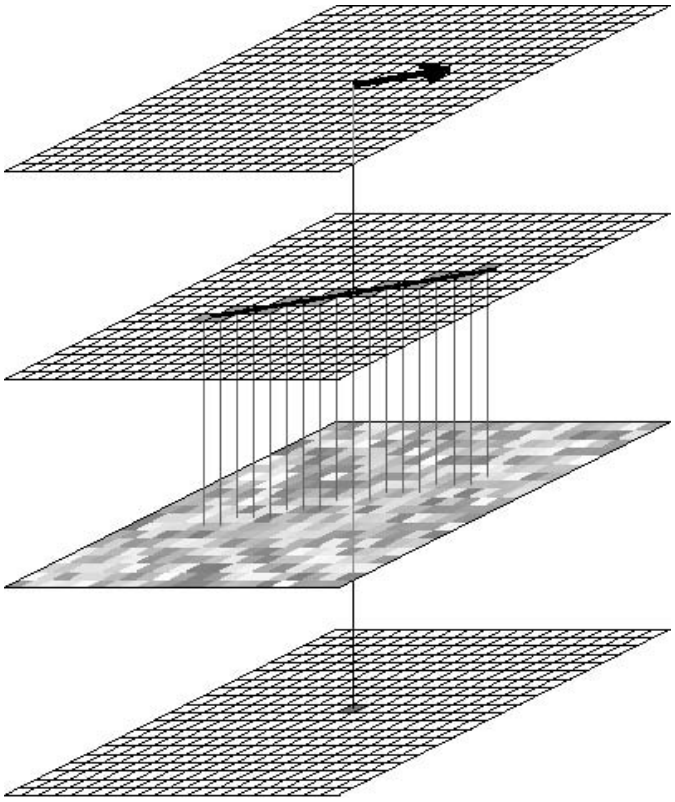


Figura 2.3: A partir da linha de fluxo e da imagem de ruído, é obtida a cor final por convolução. Imagem extraída do artigo original (2).

são adjacentes, ambos terão linhas de fluxo semelhantes, exceto pelos extremos. Desta forma, a cor final de ambos é muito próxima. Se analisarmos os demais pontos ao longo da streamline, perceberemos que a cor muda suavemente conforme nos afastamos do centro da linha de fluxo.

No segundo caso, os dois pixels geram streamlines paralelas, que acessam áreas completamente diferentes da imagem de ruído branco e, portanto, tendem a ter cores finais radicalmente diferentes. Desta forma, percebemos claramente na imagem final linhas ao longo do fluxo.

2.1.2 Considerações Adicionais

Algumas considerações devem ser feitas sobre este algoritmo. Em primeiro lugar, é possível que, devido à natureza do campo, novos segmentos de reta acrescentados à linha de fluxo tenham tamanho zero ou próximo de zero, o que poderia resultar em repetição infinita ou em um aumento desnecessário do tempo de computação. Esse tipo de situação pode ocorrer, por exemplo, em vórtices ou quando vetores adjacentes "apontam" para uma borda compartilhada. Caso seja identificado o problema, basta que se interrompa o cálculo da linha de fluxo naquele ponto.

É interessante notar que o cálculo da cor de cada pixel é feito de maneira independente, o que torna o uso de paralelismo trivial. No caso específico de programação em placa gráfica, um *fragment shader* atende perfeitamente às necessidades do algoritmo.

O tamanho L escolhido para a linha de fluxo tem impacto direto no desempenho do algoritmo. Quanto maior o valor, mais suave a imagem final e maior o tempo de processamento. Empiricamente, valores entre 20 e 30 mostraram-se suficientes para uma boa qualidade final. Valores abaixo de 10 dificultam a identificação das linhas de fluxo, enquanto para valores acima de 40 o ganho de qualidade é muito pequeno.

Inicialmente consideramos que existe um pixel na imagem de ruído branco para cada pixel na imagem final, o que implica em ambas terem a mesma resolução. Isso, no entanto, não é necessário. Podemos utilizar uma imagem de ruído de tamanho fixo (256×256 ou 512×512) e repeti-la tantas vezes quanto necessário para cobrir a imagem final. No caso de programação em GPU, as funcionalidades de textura resolvem rapidamente esta questão.

Finalmente, é possível aplicar filtros para criar um efeito de animação na imagem final. O artigo original (2) sugere o uso de um filtro de Hanning, com o qual a convolução seria:

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

Onde c e d são constantes do filtro e β indica a mudança de fase em radianos. Se novamente pensarmos na convolução como uma média ponderada, o peso w de um segmento de reta é dado por:

$$\begin{aligned} w(a,b) = \frac{1}{4}\Bigg(b - a + \frac{\sin(bc) - \sin(ac)}{c} + \frac{\sin(bd + \beta) - \sin(ad + \beta)}{d} \\ + \frac{\sin(b(c-d) - \beta) - \sin(a(c-d) - \beta)}{2(c-d)} \\ + \frac{\sin(b(c+d) + \beta) - \sin(a(c+d) + \beta)}{2(c+d)} \Bigg) \end{aligned} \tag{2-2}$$

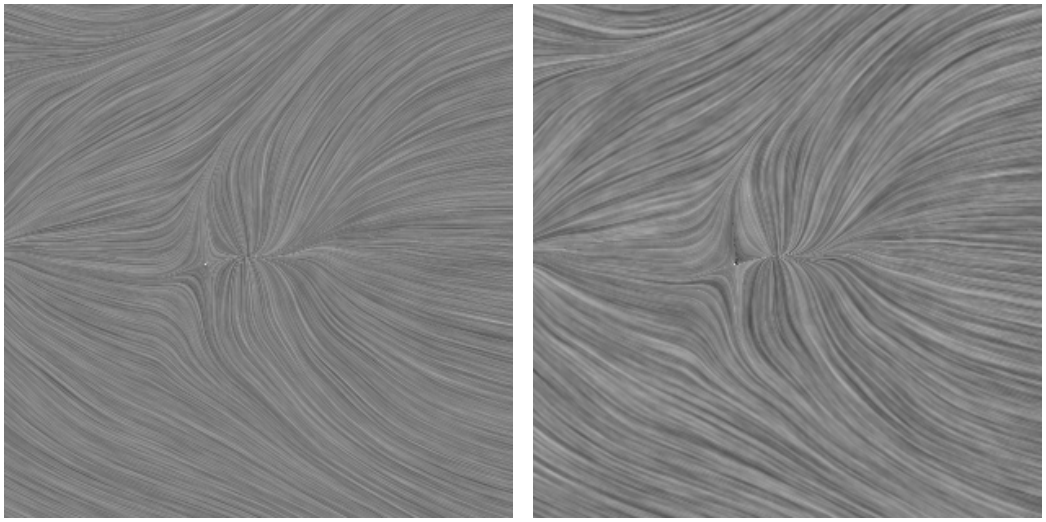
Considerando apenas a metade da linha de fluxo gerada durante a advecção positiva, a representa o tamanho da linha de fluxo calculada até o momento e b é dado pela soma de a com o tamanho do segmento. A mesma equação é válida para a advecção negativa (ver Seção 2.1.1) considerando tamanhos negativos.

2.1.3
Espessura das Linhas

Uma última consideração diz respeito à imagem de ruído branco. Conforme discutido na Seção 2.1.2, a imagem de ruído utilizada tem tamanho fixo e possui, como é de se esperar, pixels em tons de cinza aleatórios. Com isso, linhas de fluxo paralelas acessam regiões completamente diferentes do ruído e geram resultados drasticamente diferentes.

Se, no entanto, as regiões não fossem tão diferentes assim entre linhas de fluxo paralelas porém adjacentes, perceberíamos linhas mais espessas na imagem final. Uma maneira de se observar esse efeito é utilizando uma imagem de ruído onde ao invés de cada pixel ter um tom de cinza diferente, grupos de 4 ou 9 pixels possuem uma única cor.

Desta forma, a tendência é que grupos de 2 ou 3 linhas paralelas na imagem final se juntem para formar uma única linha mais grossa e, portanto, mais facilmente identificável. A Figura 2.4 evidencia este resultado.



2.4(a): Ruído Convencional 2.4(b): Ruído para Realce

Figura 2.4: Diferentes imagens de ruído aplicadas ao LIC.

2.2
LIC com Melhorias

Okada e Kao (6) propõem uma série de melhorias para tornar a imagem final do LIC (2) mais nítida. Duas delas foram incorporadas ao algoritmo desenvolvido neste trabalho.

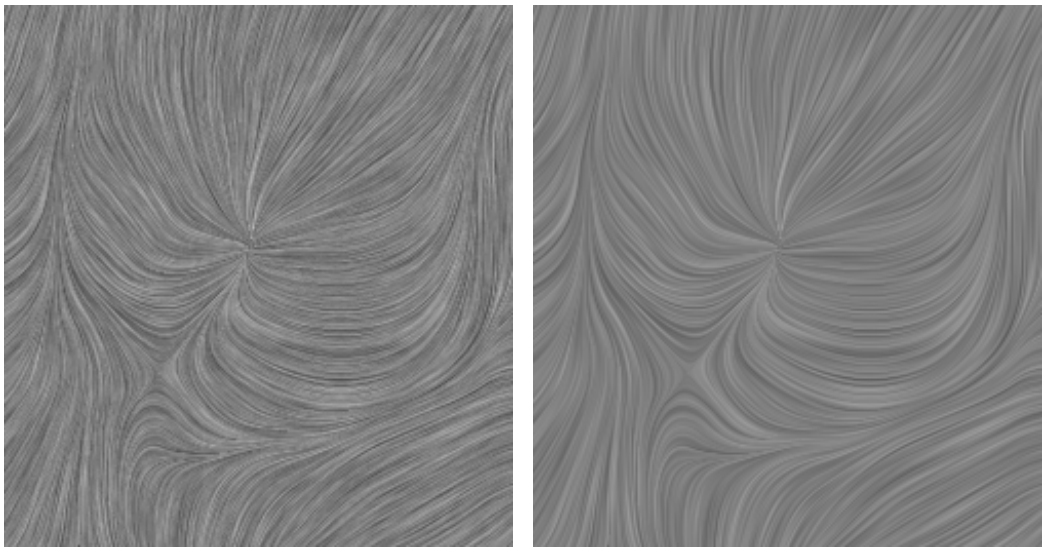
2.2.1
LIC Duplo

Uma melhoria discutida por Okada e Kao (6) consiste em executar o algoritmo original duas vezes. Na primeira, não há qualquer mudança e o LIC (2) é executado normalmente. Na segunda, ao invés de utilizar um ruído branco, é utilizada a imagem resultante da passada anterior.

Exceto pelo fato desta imagem ter a mesma resolução da imagem final e não ser necessário fazer as replicações discutidas na Seção 2.1.2, ela é utilizada no lugar de um ruído branco sem nenhuma modificação no algoritmo.

O resultado final é uma imagem com menos *aliasing* e com linhas mais suaves. A segunda passada tenta tornar as cores ao longo de uma linha de fluxo mais homogêneas.

Na prática, além de gerar imagens mais nítidas, tal estratégia permite que se use tamanhos de linhas de fluxo menores sem prejuízo para a qualidade e, portanto, tende a compensar a inclusão de uma nova passada do algoritmo. A Figura 2.5 evidencia os ganhos da segunda passada.



2.5(a): LIC Convencional 2.5(b): LIC Duplo

Figura 2.5: Comparação entre LIC convencional e LIC duplo.

2.2.2
Filtro Passa-Alta

O efeito de *anti-aliasing* provocado pela execução dupla do algoritmo permite que um filtro de passa-alta aplicado à imagem final seja capaz de efetivamente realçar as linhas de fluxo. A Figura 2.6 ilustra esta aplicação.

A aplicação do filtro se dá através do seguinte kernel 3×3:

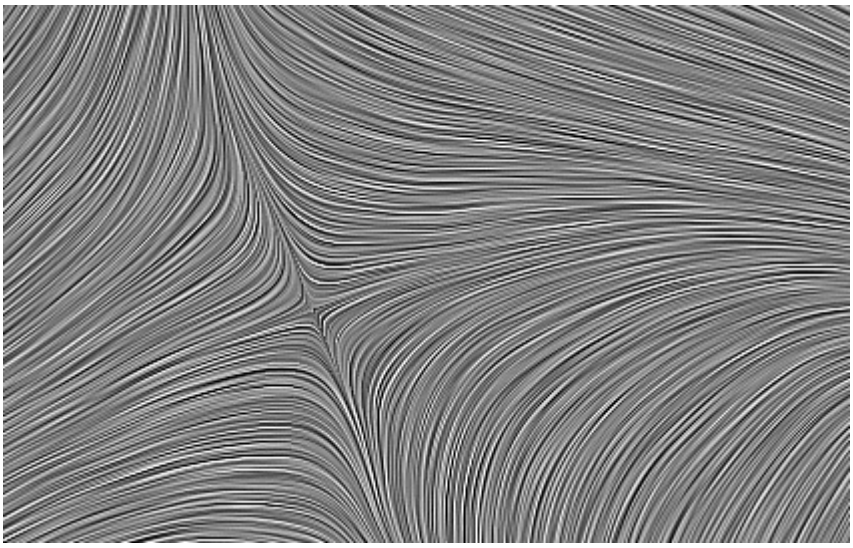


Figura 2.6: Filtro de passa-alta aplicado ao resultado de duas passadas de LIC.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Okada e Kao (6) argumentam que o uso de um kernel maior (5×5) não traria benefícios devido à natureza das linhas geradas pelo algoritmo.

2.3 Visualização de Fluxo Baseada em Imagens

Van Wijk (11) introduz uma técnica baseada em advecção de textura (Visualização de Fluxo Baseada em Imagens ou IBFV) capaz de gerar imagens semalhantes às do LIC (2) em menor tempo computacional.

O algoritmo consiste basicamente em realizar composições sucessivas de imagens, distorcendo a imagem corrente de acordo com o campo vetorial e retroalimentando o algoritmo com imagens de ruído branco.

O autor vai além e propõe (12) uma nova técnica (IBFV para Superfícies Curvas ou IBFVS), baseada na anterior, para visualizar campos vetoriais sobre superfícies de modelos 3D utilizando uma estratégia semelhante à apresentada neste trabalho, que consiste em trabalhar com projeções no espaço da tela.

Entretanto, a natureza do IBFV dificulta o estabelecimento de coerência entre quadros, o que provoca um efeito estranho e desconfortável para o usuário durante a manipulação do modelo. Ainda que o autor tenha encontrado maneiras de aliviar o efeito durante a aproximação ou afastamento do modelo, corrigir o problema durante rotações não é algo trivial. O autor argumenta, ainda, que em algumas situações a imagem gerada pode ser ambígua. Uma

discussão mais detalhada sobre o assunto será apresentada junto com uma proposta de solução na Seção 3.3.3.

2.4 Outros Trabalhos

Há muita pesquisa voltada para otimização do LIC, dentre as quais pode-se citar (8), (14) e (3). Em termos de performance, consideramos suficiente manter o algoritmo original, porém utilizando GLSL para aproveitar a capacidade de processamento de placas gráficas modernas. Qin et al. (7) utilizam também a GPU, porém optando pelo uso de CUDA. Hlawatsch et al. (4) exploram aceleração através de hierarquias de integração, de um modo paralelizável em placa gráfica. Teitzel et al. (9) e Battke et al. (1) propõem extensões para visualização em superficies que trabalham no espaço do objeto. Weiskopf e Ertl (13) realiza cálculos no espaço da tela, porém utiliza texturas sólidas para manter coerência entre quadros, método mais caro e complexo que o proposto neste trabalho. Laramee et al. (5) apresentam um amplo e interessante resumo sobre técnicas de visualização de campos vetoriais em geral. Finalmente, as principais idéias deste trabalho, com foco em visualização para reservatórios de petróleo, foram apresentadas em (10) como parte dos resultados desta pesquisa.