Models of animated rivers for the interactive exploration of landscapes

Qizhi Yu

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

THESE

pour obtenir le grade de

DOCTEUR DE L'Institut polytechnique de Grenoble

Spécialité : *« Informatique »*

préparée au laboratoire Jean Kuntzmann

dans le cadre de **l'Ecole Doctorale** *« Mathématiques, Sciences et Technologies de l'Information »*

présentée et soutenue publiquement

par

Qizhi YU

le 17 Nov 2008

Modèles de rivières animées pour l'exploration interactive de paysages

Models of animated rivers for the interactive exploration of landscapes

DIRECTEURS DE THESE

Fabrice NEYRET

CO-DIRECTEUR(S) DE THESE EVENTUEL(S)

Eric BRUNETON

JURY

Mme. Marie-Paule Cani	Président
M. Jean-Pierre Jessel	Rapporteur
M. Eric Galin	Rapporteur
M. Fabrice Neyret	Directeur de thèse
M. Eric Bruneton	Co-encadrant
M. Jean-Christophe Gonzato	Examinateur

ACKNOWLEDGMENTS

First and foremost, I would like to thank my supervisor, Fabrice Neyret for his guidance, encouragement and patience. His enthusiasm for science and nature always inspired me in the past three years, and I believe this impact will last forever in my life. I would also like to thank my co-supervisor, Eric Bruneton for his kind advisement in the last year.

Many thanks to Jean-Pierre Jessel, Eric Galin and Jean-Christophe Gonzato for being the member of the jury.

I would like to thank Marie-Paul Cani for being the president of the jury and great leadership in the EVASION team. Special thanks to Romain Arcila for his help on the French abstracts of this dissertation, Sébastien Barbier and Christian Boucheny for their many helps on my life in Grenoble, and my officemates Jamie Wither and Mathieu Coquerelle. Thank you very much (in alphabetical order): Florence Bertails, Georges-Pierre Bonneau, Guillaume Bousquet, Antoine Bouthors, Philippe Decaudin, Julien Diener, François Faure, Franck Hètroy, Anne Pierson, Laks Raghupathi, Lionel Revéret, Damien Rohmer, Maxime Tournier and Xiaomao Wu.

I would also like to thank many people in the ARTIS team: Nicolas Holzschuch, Joëlle Thollot, Sylvain Lefebvre, Cyril Crassin and David Vanderhaeghe for their coauthoring, comments and discussions during this dissertation work.

I am very grateful for the Marie Curie Actions that gave me the opportunity to work with all the nice people mentioned above. Special thanks go to Carole Bienvenu for her great assistant work in the fellowship program.

Finally, I wish to thank my entire family, especially my mother, for all their support and confidence in me.

ABSTRACT

Rivers are ubiquitous in nature, and are thus an important visual component in the simulation of natural scenes. Because rivers are dynamic in nature, it is necessary to animate their flow in these visual simulations. Realistic animation of rivers is a challenging problem because convincing simulations must incorporate multi-scale surface details and flow motion, and many of the phenomena involved have complex underlying physical causes. River animation is particularly difficult in emerging interactive applications such as Google Earth or video games that allow users to explore a very large scene and to dynamically decide whether to observe rivers at very small or large scales. Controlling the design of water simulations is another hard problem.

The goal of this dissertation is to achieve real-time, scalable, and controllable river animation with a detailed and space-time continuous appearance. To achieve this realism, the river animation problem is broken down into macro-, meso-, and micro-scale subproblems ranging from coarse to fine. We propose appropriate models for each scale that capture relevant surface details and fluid motion. In the macro-scale, we propose a procedural method that can compute the velocities of rivers with curved banks, branchings and islands on-the-fly. In the meso-scale, we propose an improved phenomenological method for simulating the quasi-stationary waves that are made by obstacles. Moreover, we propose a method for constructing an adaptive and feature-aligned water surface. In the micro-scale, we propose the use of wave sprites, a sprite-based texture model, to represent advected details with stationary spectrum properties on flow surfaces. Armed with wave sprites and a dynamic adaptive sampling scheme, we can texture the surface of a very large or even unbounded river with scene-independent performance. In addition, we propose a spectrum-preserving texture advection method that has useful applications beyond river animation.

We demonstrate that the combination of our models across three scales helps us incorporate visually convincing animated rivers into a very large terrain in real-time interactive applications.

Keywords: image synthesis; natural phenomena; animation; rivers; real-time; dynamic texture

RESUME

Les rivières sont fréquentes dans la nature, et sont donc importantes dans une simulation de scènes naturelles 3D. Afin de reproduire l'écoulement des rivières réelles, ces rivières virtuelles doivent être animées. Mais ceci est un problème difficile. Il faut prendre en compte des détails de surface et des mouvements à plusieurs échelles, et la plupart des phénomènes impliqués ont des causes physiques sous jacentes complexes. L'animation de rivières est particulièrement difficile dans le contexte d'applications interactives émergentes telles que Google Earth ou certains jeux vidéos, qui permettent à l'utilisateur d'explorer une scène très vaste et d'y observer des rivières de très près ou de très loin, à n'importe quel moment. Le contrôle utilisateur des simulations de fluide est un autre problème difficile.

Le but de cette thèse est d'obtenir des rivières animées en temps-réel sur de très grand terrains, avec une animation contrôlable et un rendu détaillé et continu en temps et en espace. Pour atteindre ce but nous décomposons le problème en 3 sous-problèmes pour les grandes, moyennes et petites échelles. Nous proposons des modèles appropriés pour chaque échelle, qui représentent correctement les détails et les mouvements du fluide à l'échelle considérée. Pour les grandes échelles nous proposons une méthode procédurale pour calculer à la volée la vitesse d'écoulement de rivières de formes quelconques, avec des affluents et des îles. Pour les moyennes échelles nous proposons une méthode de simulation spécifique pour générer les vagues quasi stationnaires causées par les obstacles. Nous proposons également une méthode pour construire un maillage adaptatif aligné avec les lignes de crête de ces vagues. Pour les petites échelles, nous proposons une méthode à base de sprites texturés pour représenter les petits détails dont le spectre est stationnaire et qui sont advectés avec la rivière. Complétés par un échantillonnage adaptatif, cette méthode nous permet de texturer de très grandes rivières, les performances étant indépendantes de la complexité de la scène. Nous proposons également une méthode lagrangienne pour l'advection de texture, qui peut s'appliquer à d'autres domaines que l'animation de rivières.

Nous montrons que la combinaison de nos trois modèles nous permet d'ajouter des

rivières animées visuellement convaincantes dans de très grand terrains, dans des applications temps-réel interactives.

Mots-clés : synthèse d'image ; phénomènes naturels ; animation ; rivière ; temps-réel ; dynamique texture

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The modeling of rivers is an important topic in computer graphics. Nowadays, more and more graphics applications have the ambitious goal of presenting virtual but highly realistic nature scenes. In these scenes, rivers are often an indispensable feature. Examples of such application include video games, simulators, geographic information systems, landscape design and feature films. Furthermore, rivers are dynamic in nature, thus river animation is desirable to improve the realism of these applications.

However, the animation of rivers is still a challenging problem. The modeling of water or other fluids in computer animation has been studied for more than two decades, but there are few existing techniques applicable for river animation. One challenge is that the motion of river water has a wide range of scales. We would expect to see river water flowing continuously in the kilometer scale, while having surface waves in the millimeter scale. It is difficult to resolve all of the scales that affect the visual impression of river surfaces fully in any single model. Another challenge is that many phenomena relevant to river surfaces have very complicated underlying physical causes, such as turbulence. The physically-based simulation models used in computer graphics still cannot easily handle these causes. Therefore, even in off-line graphics, river animation is often limited to a small domain and neglects many distinct surface features of rivers, such as hydraulic jump.

This work targets emerging interactive applications like Google Earth [Goo08]. Such applications let users navigate in real-time even on a very large-scale global terrain, where users can examine any feature in a very close or very wide view at any moment. Thus, these applications introduce even more challenges for the animation

algorithm. In this context, a good solution should convince users that a river is animated with visually believable details everywhere while maintaining a continuous flow. Moreover, the algorithms must run in real-time with limited computational resources. In practice, however, animation of rivers in this type of applications is either unavailable (*e.g.,* Google Earth) or approximated by using simple tricks (*e.g.,* Crysis [Cry08]).

In many graphics applications, users need to design the scene and want to control the animation. However, controlling simulation behavior is another challenging problem. For example, in physically-based numerical fluid simulations, traditionally this is achieved by setting initial values or boundary conditions for the governing equations. But it is difficult to predict the simulation behavior from these input parameters. An ideal solution should provide designers with intuitive handles which are directly related to the simulation behavior.

1.1 Thesis Statement

The thesis of this research is presented below:

> *We can achieve real-time, scalable and controllable river animation by combining models appropriate to different scales of surface details and fluid motion on current desktop hardware.*

Our constraints are:

- *Performance*: We strive for 25 frames per second or more.
- *Scalability*: The rivers we handle can be very large and even unbounded, and users must be allowed to observe the rivers in a very narrow view or in a very broad view at any moment.
- *Controllability*: The models should provide intuitive handles for users to control the behavior or appearance of the rivers on-the-fly.
- *Quality*: The animation should convey visually convincing fluid motion and appearance, and ensure spatial and temporal continuity.

1.2 Overview

In this dissertation, we break down the river animation problem into macro-, meso-, and micro-scale subproblems from coarse to fine. We propose appropriate models for each scale that capture the relevant surface details and fluid motion. In the following, we detail the problems, challenges, and our strategies for each scale.

1.2.1 Macro-Scale

A distinct overall visual impression of ordinary rivers is a continuous and smooth flow through long streamlined channels, and past junctions and islands (Figure 1.1). To reconstruct this in computer graphics, we need to determine the overall geometry and velocity of a river. For everyday's river, it is justified to assume that surface fluctuations and vertical water motion do not contribute much to the visual impression in this scale. Therefore, we can model rivers as 2-D steady flow, in which neither surface elevation nor velocity varies with time.

Figure 1.1: Macro-scale: overall visual impression of rivers.

The main challenge here is to solve a large scale problem while limiting the computation and memory cost in the real-time context. Solving 2D shallow water equations numerically could generate the surface elevation and velocity that we want with reasonable boundary conditions [Mol95]. However, this process cannot run in real-time for large rivers. Moreover, for some graphics applications which aim at global-scale navigation (such as Google Earth), even relying on pre-computation is also impractical.

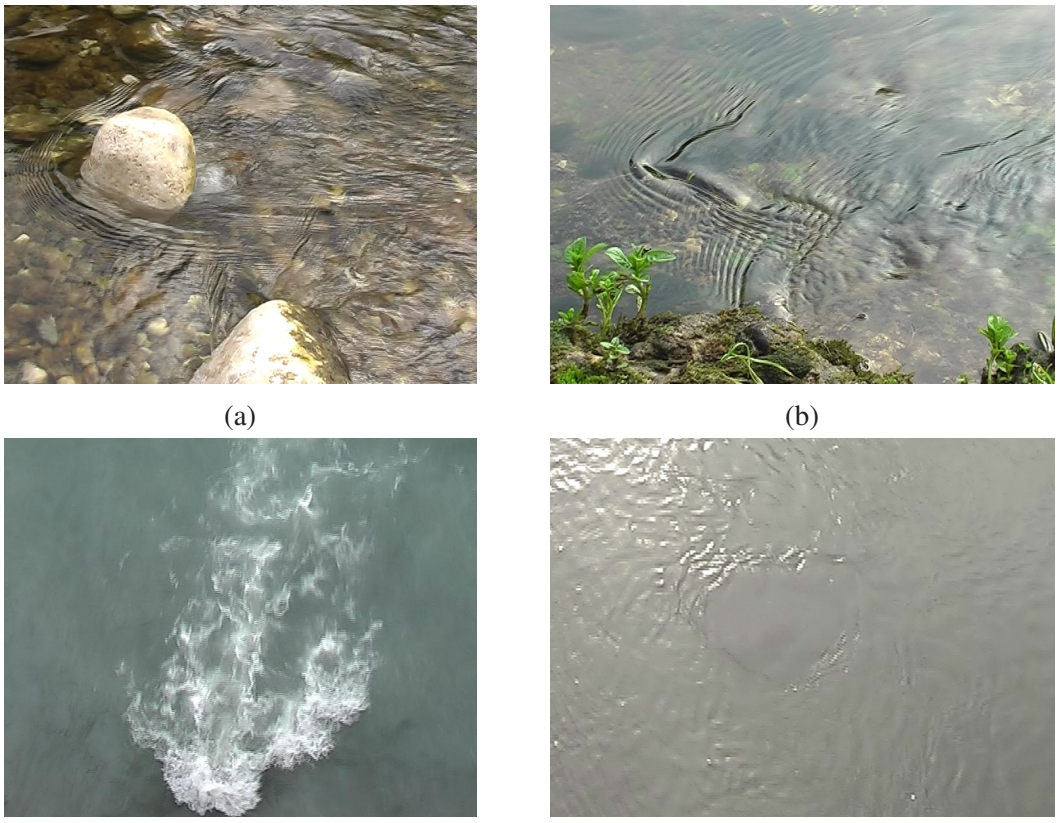
In this dissertation we focus on the animation problem. The shape of the river surface could be generated with existing methods such as [KMN88] and [Bru08]. For velocity, we resort to the procedural approach which is usually stable, fast, scalable and controllable. We propose a stream-function based method that can generate visually convincing channel flows on-the-fly. The evaluation of velocity only depends on the geometries of local flow boundaries (banks and islands). The method also lets users control the flow by flow rates, boundary conditions, and boundary geometries interactively, without interrupting the animation.

1.2.2 Meso-Scale

Many surface waves in rivers are local and well-structured (Figure 1.2). These phenomena are usually caused by the interaction between water flow and obstacles, or the interaction between upwelling turbulence and free surfaces. These waves could be stationary or advected with the flow. Essentially, these phenomena attribute to the water current, and thus they are characteristic features that make rivers look different from ponds or lakes.

Few existing models in computer graphics are dedicated to simulating these phenomena. Relying solely on traditional *computer fluid dynamics* (CFD) models is not practical due to the complicated underlying physical causes. Because strong flow discontinuity and turbulence are often involved in these types of phenomena, applying the CFD techniques directly would require very high-resolution discretization to capture detailed surface features. Worse, sometimes the cause of a phenomenon is not local, nor is it surface-based. For example, the phenomenon of boils shown in Figures 1.2d and 1.3 is caused by ejections coming from the river bed [Jac76]. In this case, we would have to simulate the whole water body, even though we are only interested in the deformation of the water surface.

Our idea is to separate the representation for animation and rendering: we simulate directly the features of a phenomenon using a vector representation, and then convert the vector features to another appropriate representation (*e.g.,* mesh) for high-quality rendering. We call this two-step approach *feature-based vector simulation*. In this dissertation we choose the stationary waves caused by obstacles as a case study to validate our approach (Figure 1.2a). This phenomenon has been explored in [NP01] with a phe-



(a) (b)

(c) (d)

Figure 1.2: Meso-scale: local and structured waves in running streams. (a) Quasi-stationary waves caused by obstacles. (b) Capillary ship waves caused by submerged obstacles. (c) Hydraulic jump caused by bed topography change. (d) Surface boils (circular waves in the center) caused by upwelling water.

nomenological method that fits well to our approach. We improve their model for better robustness and efficiency, and use it to simulate the dynamic vector feature of the target waves. For efficient and quality rendering, we propose a technique to construct adaptive and feature aligned water surfaces from the simulated vector information. Finally, we achieve real-time and realistic animation of the stationary waves caused by obstacles.

1.2.3 Micro-Scale

One can always find abundant small waves continuously distributed on river surfaces and advected with surface flow (Figure 1.4). Unlike the phenomena we have introduced

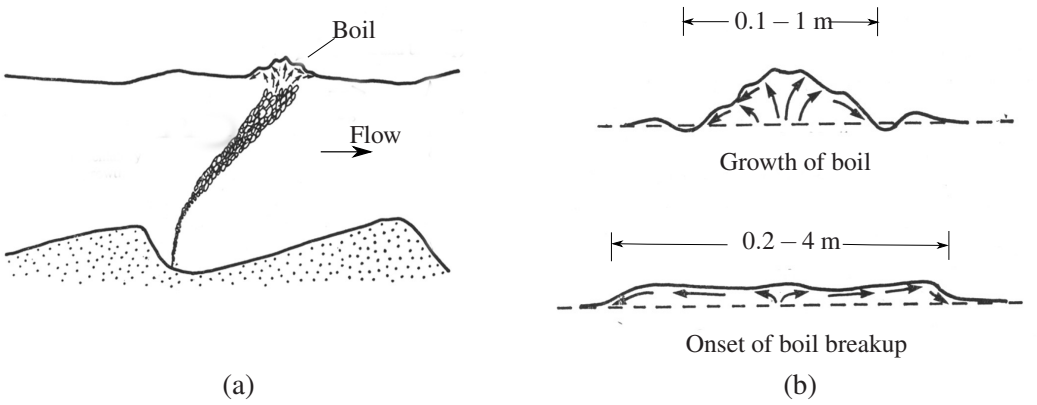


Figure 1.3: Schematic illustration of the mechanism of boils. (a) Boils are caused by the interaction of ejections coming from the river bed with the water surface. (b) Development of a typical boil. (Figures are reproduced from [Jac76])

in the macro-scale, these phenomena do not have distinct structural geometries, and we mostly see their statistical properties. Simulating this kind of phenomenon is also important to ensure the realism of river animation because the advection of these waves conveys the flow motion of rivers.

Figure 1.4: Micro-scale: small waves on river surfaces.

There are some methods in computer graphics that aim at simulating small waves similar to the phenomena mentioned above, including statistic-based methods [MWM87, Tes04] or procedural methods [Per85, YHK07]. These methods can produce nice waves for non-flowing water such as ocean waves. However, they neglect the advection of

waves caused by water flow. It is not clear how to adapt these methods for rivers where wave advection is a main surface characteristic.

One opportunity to solve the problem is to rely on texturing techniques because these wave features are similar both in time and spatial space. Then the problem is how to generate a texture sequence which confirms to a given flow field while preserving the texture property of a given reference texture. The reference texture can be specified either by high-level procedural parameters or a texture example. Therefore, our goal in this scale is to develop a flow-guided dynamic texture scheme adapted to large surfaces while running in real-time. Our basic idea is to solve the problem from the Lagrangian point of view. Our study in this category includes two parts. First, we propose a new sprite-based texturing technique called *wave sprites*. The technique runs in scene-independent time by using a dynamic adaptive sampling scheme. Second, by improving and generalizing the wave sprites model, we arrive at a spectrum-preserving texture advection method that has applications beyond river animation.

1.3 Contributions

Besides the three-scale framework itself, the contributions of this dissertation include:

- A procedural method for generating the visually convincing velocity of channel flow with complex boundaries, branchings, and obstacles on-the-fly. The method is real-time, scalable to very large terrain and interactively editable. (Chapter 3)
- An adaptive sprite-based texturing method, *wave sprites*, for representing advected details on flow surfaces. The method supports unbounded scenes, and runs with scene-independent performance. (Chapter 5)
- A Lagrangian texture advection method ensuring both spatial and temporal continuity and spectrum conservation in real-time. (Chapter 6)
- A dynamic Poisson-disk sampling scheme that runs in linear time. (Sections 6.2.1 and 5.2.1)
- An improved phenomenological model for simulating the waves made by obstacles in real-time. (Chapter 4.1)
- A method to construct an adaptive water surface with waves on top to ensure real-time and high-quality rendering. (Section 4.2)

1.4 Organization

The remainder of this dissertation is organized as follows. The next chapter provides a brief introduction to related work. Then, the following chapters present methods addressing each of the three scales: macro-scale (Chapter 3), meso-scale (Chapter 4), and micro-scale (Chapters 5 and 6). In Chapter 3 we present a procedural method that can generate visually convincing velocity on-the-fly for channel flow. In Chapter 4 we present a feature-based vector simulation approach and choose waves caused by obstacles in a stream as a case study. Chapters 5 and 6 both deal with detailed features on flow surfaces, but with different emphases. In Chapter 5 we present an adaptive sprite-based texturing method to handle large-scale animated surfaces efficiently. In Chapter 6 we present a spectrum-preserving texture advection method. Finally, Chapter 7 concludes and describes directions for future work.

1.5 French Introduction

Les résultats des recherches sont présentés ci-dessous :

> *Nous pouvons générer des animations de rivières en temps réel sur du matériel actuel, à différentes échelles et contrôlables par l'utilisateur, grâce à une combinaison de modèles adaptés à chaque échelle de détails et de mouvement du fluide.*

Nos contraintes sont les suivantes :

- *Performance* : Nous voulons au minimum 25 images par seconde ;
- *Echelle* : Les rivières que nous manipulons peuvent être très étendues, voire illimitées. L'utilisateur doit pouvoir visualiser la rivière soit de très près soit d'un point de vue très éloigné, et ce à n'importe quel moment ;
- *Contrôle* : Le modèle doit être aisément et rapidement (à la volée) paramétrable, tant pour le comportement que pour le rendu de la rivière ;
- *Qualité* : L'animation doit être plausible, pour le mouvement et l'apparence. Il doit y avoir une continuité spatio-temporelle.

Dans cette thèse, nous avons divisé le problème en trois sous problèmes : macro-, meso-, et micro-échelle, du plus grossier au plus fin. Nous proposons un modèle appro-

prié pour chaque échelle, où chaque modèle retranscrit les détails inhérents à la surface et à son mouvement, à l'échelle voulue.

En plus du modèle multi-échelle lui-même, nos contributions incluent :

- Une méthode procédurale pour générer une vitesse visuellement plausible du cours d'un fleuve avec des rives complexes, des embranchements et des obstacles, à la volée. Cette méthode est temps réel, fonctionne sur des terrains de grandes dimensions et avec des rivières qui peuvent être éditées interactivement. (Chapitre 3)
- Une méthode de texturage adaptative, basée sur les sprites, *wave sprites*, pour le transport de pertubations locales de la surface de la rivière. Cette méthode supporte des scènes extrêmement étendues, et les performances ne dépendent pas de la scène. (Chapitre 5)
- Une méthode d'advection de texture Lagrangienne assurant à la fois une continuité spatio-temporelle et une conservation du spectre en temps réel. (Chapitre 6)
- Un schéma d'échantillonnage dynamique basée sur une distribution de disques de Poisson mise à jour en temps linéaire. (Sections 6.2.1 et 5.2.1)
- Un modèle phénoménologique amélioré, basé sur une représentation vectorielle, pour l'animation temps réel des vagues formées par les obstacles (Chapitre 4.1).
- Une méthode de génération de la surface de l'eau où les vagues sont modelisées adaptativement, ce qui permet d'obtenir un rendu de bonne qualité en temps réel (Section 4.2).

Le reste de cette thèse est organisé de la façon suivante. Le chapitre suivant présente une courte introduction des travaux proches du nôtre. Dans le chapitre 3, nous présentons une méthode procédurale permettant de générer à la volée et de manière plausible la vitesse des courants des rivières. Dans le chapitre 4, nous présentons un modèle phénoménologique basé sur une représentation vectorielle pour simuler les vagues formées par les obstacles. Les chapitres 5 et 6 concernent l'animation des détails de la surface d'un flux avec différents compromis performance vs qualité. Dans le chapitre 5, nous présentons une méthode de texturage adaptative basée sur les sprites permettant de gérer des terrains de grande taille efficacement. Dans le chapitre 6, nous présentons une méthode d'advection de texture Lagrangienne, moins efficace mais donnant des résultats de meilleure qualité. Enfin, le chapitre 7 présente une conclusion ainsi que les différentes possibilités de travaux futurs.

Chapter 2

Related Work

In this chapter, we first review previous work on computer animation of water. After that, we cover previous work related to several individual components in our models.

2.1 Computer Animation of Water

2.1.1 Numerical Simulation

There are large literatures on simulating water by solving the governing equations numerically. Since these methods usually do not fit well with large-scale and real-time applications due to bad scalability, we review them briefly.

Early physically-based simulation work focused on waves on water surfaces represented by height fields. Under the assumptions of small wave amplitude and shallow water, Kass and Miller [KM90] used linear wave equation. Layton and van de Panne [LvdP02] simulated waves by solving 2D shallow water equations with a semi-Lagrangian time integration method. Chen *et al.* [CdVL95, CdVLHM97] used the pressure solved from 2D Navier-Stokes equations to modulate the height of water surfaces. In a small bounded domain, these methods could be used for generating waves in real-time interactive applications. Given sloping river beds, the shallow water model can simulate river flow [Mol95]. However, it is difficult to set boundary conditions for generating specific flow desired by users.

Solving full 3D Navier-Stokes equations on an Eulerian grid was pioneered in computer graphics by [FM96, Sta99]. This approach has been successfully used for of-

fline water animation by combining it with the level set method for tracking free surfaces [FF01, EMF02]. Though some efforts have recently succeeded in improving performance by using adaptive grids [LGF04, IGLF06], the performance of their methods is still far from real-time (Figure 2.1).

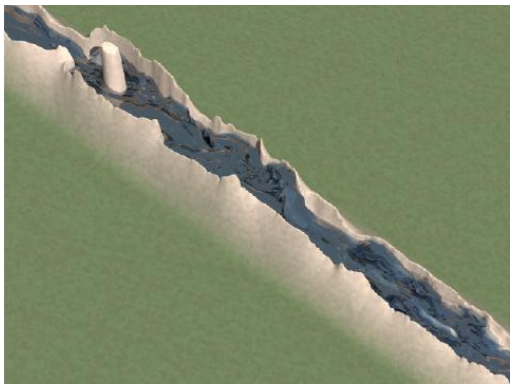
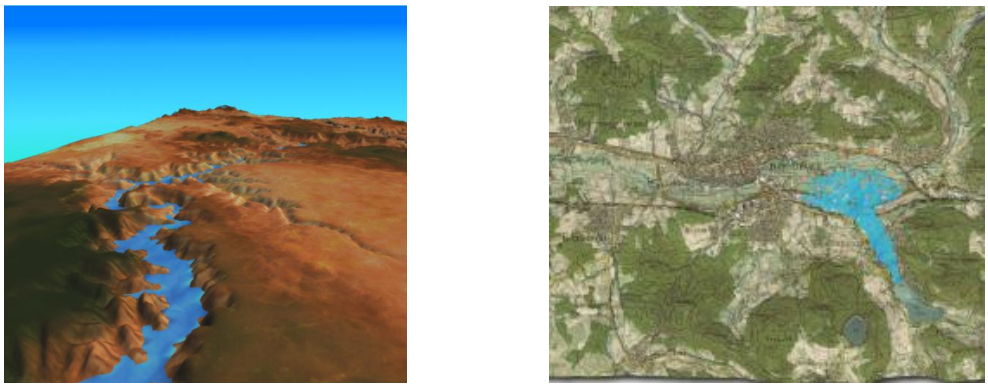


Figure 2.1: Simulation of a river by combining two-dimensional and three-dimensional simulation techniques (2000×200 horizontal resolution) [IGLF06]. The method is able to simulate complex water motion of rivers, however the computational cost is very expensive (approximately 25 minutes a frame using 20 processors).

Lagrangian particle methods are also popular for solving fluid flow. Müller *et al.* [MCG03] proposed a method based on smoothed particle hdrodynmaics (SPH) for simulating fluids. For rendering free surfaces, they used point splatting and the marching cubes method. Compared with the Eulerian grid approach, the Lagrangian particle approach tends to have better performance but suffers from the problem of reconstructing a smooth surface. Kipfer and Westermann [KW06] used the SPH method for simulating rivers in a bounded domain. They claimed interactive framerates while using a very low density of particles that leads to the loss of surface details (Figure 2.2).

2.1.2 Non-Numerical Simulation of Water Waves

Instead of numerically simulating fluid motion, some wave models reconstruct the water surfaces directly from the geometric or statistical properties of waves. This kind of approach appeared earlier than did the physically-based simulation in graphics communities, but it is still the first choice in many today's industry applications due to their advantages of fast and stable computation, low memory cost and high controllability.



(a) 8000 particles at 26 fps. (b) 20000 particles at 12.7 fps.

Figure 2.2: River simulation using SPH [KW06]. The method is able to simulate rivers on complex terrain. However, real-time performance is available only when using a limited number of particles. For a large scene, that leads to the loss of surface details.

Perlin noise [Per85] can be used for modeling random waves with very low computational cost. Classic procedural wave models approximate deep water waves by superimposing sinusoidal functions [Max81, FR86, Pea86]. By introducing an adaptive scheme, Hinsinger *et al.* [HNC02] achieved interactive animation of unbounded ocean surfaces. Instead, FFT wave models [MWM87, Tes04] perform the superposition in the spectrum domain, and thus the statistical wave spectrum of real ocean can be easily applied. Later, Mitchell [Mit04] demonstrated a GPU implementation of this method. While all these methods focus on ocean waves, some other procedural models deal with local wave phenomena, such as stationary waves caused by obstacles in streams [NP01] and ship waves [Gla02]. We revisit [NP01] and improve this work in Section 4.1. Most recently, Yuksel *et al.* [YHK07] proposed a concept called wave particles to efficiently simulate surface waves triggered by object-water interaction (Figure 2.3).

Note that all of the wave models mentioned above do not handle the effect of wave advection which is highly desired in river animation because of its ability to convey the underlying river flow. We tackle this problem in Chapters 5 and 6. In addition, we use Perlin noise and the FFT wave model mentioned above to generate texture examples required by our method.

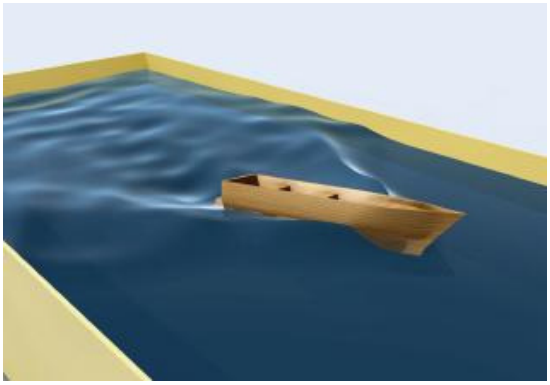


Figure 2.3: A result of wave particles [YHK07]. The method is able to simulate surface waves with object intersections in real-time for a limited domain. However, it is appropriate for situations that do not involve significant global flow.

2.1.3 Procedural Velocity Fields

Procedural methods can be used for constructing or enriching a flow field without numerical simulation. Wejchert and Haumann [WH91] constructed flow velocity fields by linearly superposing known analytic solutions of Laplacian equations. Chenney [Che04] proposed a tiling method for generating divergence-free velocity fields. Bridson *et al.* [BHN07] suggested to generate divergence-free velocity by taking the curl of a scalar function (see Section 2.2). They also demonstrated how to generate divergence-free velocity conforming to boundaries. Inspired from [BHN07], several models [KTJG08, SB08, NSCL08] have been proposed recently for generating small scale turbulent velocity generally based on some noise models [Per85, PN01, CD05] and texture advection method [Ney03]. Precisely, Kim *et al.* [KTJG08] advected wavelet noise [CD05]. Narain *et al.* [NSCL08] advected noise in the Lagrangian formalism. In addition, both [SB08] and [NSCL08] have taken into account the evolution of turbulence energy.

2.1.4 Particles and Sprites

Particle systems [Ree83, RB85, Sim90] are the procedural equivalent of Lagrangian CFD. They are currently often coupled with (animated) sprites to reconstruct a complex appearance, in games [Won] and even sometime in featured films [IC00]. Still, these methods are primarily used for splashing water. Note that flows based on video-clips [BSHK04] could be considered as part of the same category.

2.1.5 Mixed Models

Some groups have tackled water animation by treating visual features in different scales with different models. O'Brien and Hodgins [OH95] proposed a 3-part system which included a main volume, a fluid surface and the disconnected components of the fluid (spray). Thon *et al.* [TDG00] described ocean surfaces with two levels of details, including 2D trochoids and Perlin's turbulence functions. Thon and Ghazanfarpour [TG01] used the 2D N-S model for precomputing the horizontal velocity of river flow and Perlin's turbulence functions for generating vertical surface perturbations. Jensen and Golias [JG01] superimposed local shallow water waves onto the global water surface synthesized by Fourier approach [Tes04]. Thürey *et al.* [TSS$^+$07, TMFSG07] added small scale details such as foam, bubbles and breaking waves to the shallow water model. Cords [Cor07] combined low-resolution 3D SPH simulation for water volumes and high-resolution 2D wave equation for surface waves.

Though these work shares a common spirit with our own models, the ideas outlined here go beyond the previous findings in each of three scales. Moreover, among the above methods, only [TG01] can handle river flow, but the method relies on a precomputed numerical solution.

2.2 Stream Function

Fluid mechanics defines the *stream function* ψ for 2D incompressible flow in the xy plane as the scalar function which the flow velocity (u,v) is the curl of, *i.e.,* :

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}. \tag{2.1}$$

The isovalues of ψ are known as streamlines of the flow, which are tangent to the velocity field. Once we have the stream function values at any point in the flow, we can compute the velocity from the equation above by using the finite differences approach.

The divergence-free property, *i.e.,* the incompressibility, should be carefully considered when one want to generate a fluid-like velocity field. A nice property of the stream function is that it always yields a divergence-free velocity field, since the divergence of a curl is zero. Therefore, building velocity fields via stream functions is a way free of

the consideration for the divergence-free, *i.e.,* the incompressibility of fluids.

In computer graphics, some research work has used this property for constructing a divergence-free vector field. Chenney [Che04] used flow tiles to store precomputed stream function and construct a new velocity field by assembling these tiles. They presented an example of constructing a river flow, but the method can not get arbitrary complex boundaries since it relies on precomputed tiles. Von Funck *et al.* [vFTS06] constructed a velocity field for shape deformation from the co-gradient of a scaler field, which is a similar idea with deriving velocity from stream function. Recently, Bridson *et al.* [BHN07] demonstrated how to modulate stream function for constructing a velocity field conforming arbitrary boundaries. However, none of the above methods aims at generating channel-confined flow with branchings and obstacles. We tackle this problem in Chapter 3.

2.3 Dynamic Poisson Disk Distributions

A Poisson-disk distribution is a uniform point distribution in which the distance between any two points does not violate a minimum distance criterion. There are many studies on this topic. We refer the reader to [LD06] for a comprehensively survey. While previous work attempted to generate a static point set, Vanderhaeghe *et al.* [VBTS07] presented a dynamic Poisson-disk distribution algorithm in the context of stroke-based rendering, where the points are dynamic and temporally coherent while maintaining a Poisson-disk pattern. This algorithm generally includes three steps during each time frame: (1) move points forapplication-related calculations, (2) reserve points that do not violate minimum distance, and (3) insert new points in order to obtain a Poisson-disk pattern. In [VBTS07], the last two steps rely on the dart-throwing algorithm [MF92] because it is able to generate the Poisson-disk pattern incrementally. In indeed, it is possible to apply other incremental algorithms, such as the boundary sampling algorithm [DH06], to this problem. Note that the boundary sampling algorithm has the advantage of running at linear time though the quality of distribution is not perfect. We adopt it as the base of our adaptive dynamic sampling scheme (Sections 6.2.1 and 5.2.1).

2.4 Texturing Methods

In this section we review two lines of texturing research that are directly related to our work.

2.4.1 Sprite-Based Texturing

The ability to generate a large texture from a set of small texture samples is very useful because it can achieve high resolution with low memory overhead. Sprite-based texturing is one of the techniques used for this purpose. Neyret *et al.* [NHS02] introduced the idea of simulating sprites living in texture space in order to account for running drops on a curve surface. The update is achieved by sending to the GPU the list of modified areas in the texture, so this is not adapted to the overall flowing of details along the entire surface. Lefebvre and Neyret [LN03] proposed a scheme to dynamically compose texture patterns on the GPU in order to emulate a high resolution detailed texture. This method relies on an indirection grid which encodes the instance lying in each cell together with its transform (offset, scaling, rotation). However, the sprites cannot easily overlap, and their size should be constrained to the grid cell size. Lefebvre *et al.* [LHN05] introduced a GPU-enhanced adaptive structure which is able to efficiently manage overlapping sprites. Our texture composition scheme (Section 5.3) is inspired by [LHN05].

2.4.2 Flow Guided Dynamic Texture

Generating a sequence of textures that conforms to the underlying flow while retaining the texture properties is an effective way for enhancing details in fluid animation, or for visualization. Max and Becker [MB95] proposed a texture advection technique which advects texture coordinates and periodically resets the texture coordinates after a predefined latency for avoiding excessive stretching. Neyret [Ney03] pointed out that an optimal latency should be adaptive to local flow conditions and solves this issue by blending several texture layers using different latencies. This method still suffer the problems of destroying texture spectrum or not conforming to the flow in some cases. For this, we propose a new texture advection method from the Lagrangian point of view in Chapter 6. Recently, several methods based on texture synthesis have been proposed

for texturing dynamic flow surfaces [BSM+06, KAK+07, NKL+07]. These methods are computationally expensive, however, and are thus not suitable for real-time graphics.

2.5 Summary

In this chapter we have reviewed existing methods for handling water animation. Physically-based simulation of water is able to generate visually convincing results, but on current hardware, it is normally limited to off-line graphics, or real-time graphics for small domain. On the other hand, procedural methods are generally stable and fast, but no existing method targets rivers. Though some work has attempted to tackle the animation of large bodies of water by mixing several models, none of them could handle real-time animation of rivers. We also have introduced several existing techniques which have the potential to be used in our models for river animation. In the next chapter we will present our model in the macro-scale.

2.6 French Chapter Abstract

Dans ce chapitre, nous avons présenté les méthodes existantes d'animation de fluide. Les méthodes de rendu physique proposent des animations plausibles, mais sur les machines actuelles, ces méthodes ne permettent pas d'obtenir un rendu temps réel, ou seulement sur de petits domaines. Les méthodes de rendu procédurales sont généralement stables et rapides, mais aucune méthode existante ne permet de créer le flux d'une rivière complexe. Bien que des travaux précédents aient essayé de gérer l'animation de fluide à large échelle en combinant différents modèles, aucun de ces travaux ne peut générer des animations de rivières en temps réel. Nous avons également présenté plusieurs techniques existantes qui peuvent potentiellement être utilisées dans nos modèles d'animation. Le chapitre suivant présente notre modèle pour l'échelle macro.

Chapter 3

Macro-Scale: Procedural River Flow

The flowing of water is the most salient visual characteristic of everyday rivers. Our models for handling river surface features, which will be introduced in the following chapters, all require knowledge of flow velocity. Thus, it is important to compute river velocity for the purpose of realistic animation. Most of rivers in nature are turbulent flow, and, as a result, have very complicated velocity fields. In our framework, river velocity is decomposed into three scales. In the macro-scale, we assume the river flow as a 2D potential flow whose velocity is generated by the method presented in this chapter. In the meso-scale, we enrich the macro-scopic velocity by superposing local analytic velocity fields representing individual perturbations. These local velocity fields are advected with the macroscopic velocity. In the micro-scale, we do not explicitly generate velocity details. They are implicitly modeled by noise textures that are advected with the velocity provided by larger scales.

Though the river velocity can be simulated with CFD models, this approach has several disadvantages in the context of computer animation. First, the computation time for numerical simulations is usually unaffordable for interactive applications. In addition, numerical simulations usually compute and store data at a fixed resolution, which is unreasonable for cases in which the camera is permitted to move freely in a large scene. Furthermore, in many applications, the user wants to intuitively control the visual features of rivers (*e.g.,* main flow trajectory and qualitative flow speed). However, it is difficult to control the numerical simulation results by setting the initial values and the boundary conditions. Conversely, procedural methods can avoid these drawbacks.

In this chapter, we present a new procedural method for generating plausible river

velocity. The method is procedural because the velocity at any point is determined by the distances from the point to local boundaries and a few intuitive control parameters such as flow rates through channels. Though constructing fluid-like velocity fields with procedural methods has been explored in the past [BHN07], no existing method is able to generate a flow through a channel network with complex curved boundaries, branchings and obstacles.

3.1 Overview

Our goal is to generate plausible river flow velocity on-the-fly. The rivers we will model can be very large or even unbounded. They are described by channel networks where branchings and obstacles may exist (Figure 3.1). The boundaries of the channels and the obstacles are represented by curves. As for flow velocity, we are only interested in 2D horizontal fields. The velocity field we want may not be physically accurate but it should be visually convincing. This means that the flow should meet the following criteria: (1) incompressible, (2) boundary conformed, and (3) flowing through channels from source to sink.

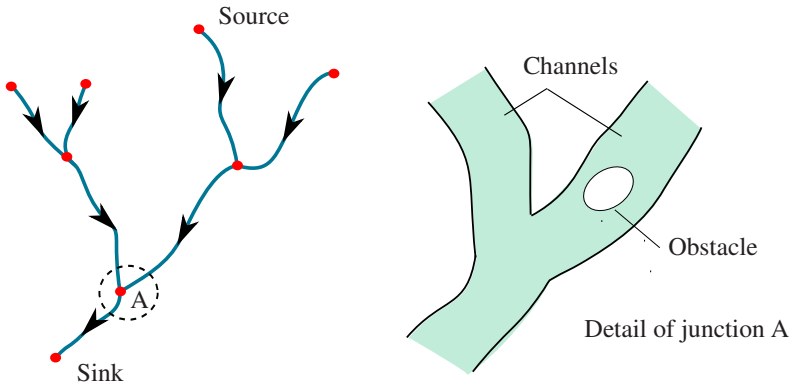


Figure 3.1: A river network.

As we mentioned in Section 2.2, there is a class of procedural methods that construct fluid-like velocity fields by taking the curl of the stream function. The merit of this approach is that it always satisfies the incompressibility criterion given an arbitrary stream function field. However, in order to meet the criteria (2) and (3) mentioned above, we must provide a reasonable stream function. For conforming to boundaries, [BHN07]

suggested ramping the stream function inside the influence region of a boundary to the value at that boundary. But, it is not clear how to extend this method to construct velocity fields for channel flow. We also found that this method has a flaw when treating the areas impacted by more than one influence region (Figure 3.2). In this case, if we still follow the method and simply modulate the stream function value according to the nearest boundary, the stream function will be discontinuous at the medial axis of several boundaries which take different boundary values. Artificially reducing the size of the influence region could avoid this flaw, while it may lead to unnatural velocity patterns.

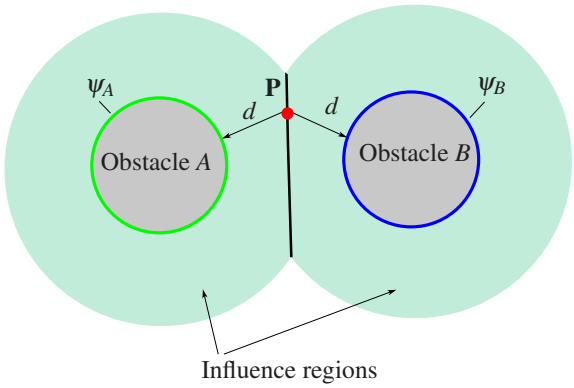


Figure 3.2: A flaw of the curl noise method [BHN07]. Given a point P at the medial axis of two close obstacles, we have $\Psi^P_{left} = f(\psi_A, d)$, $\Psi^P_{right} = f(\psi_B, d)$, where f is a ramp function. If $\psi_A \neq \psi_B$, we have $\Psi^P_{left} \neq \Psi^P_{right}$. Thus the stream function ψ is not continuous at the medial axis.

To bridge the gap described above, we propose a new stream-function based method for generating procedural flow. We also derive the flow velocity with the curl of a stream function ψ:

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}, \tag{3.1}$$

where ψ is the stream function. In order to construct a reasonable stream function field, we resort to an inverse distance weighted interpolation approach. The idea is inspired by two observations: first, the stream function at channel boundaries can be determined by the flow rates through the channels; and second, the interpolant of the inverse-distance weighted interpolation scheme is very similar to the solution of the Laplace's equation. Note that the Laplace's equation is exactly the governing equation of the 2D incompressible irrotational flow. Though river flow in nature is not irrotational flow, the difference

between them can be neglected in the context of macro-scale models.

3.2 Stream Function for Channel Flow

In this section, we describe a method for constructing a reasonable stream function field for a channel-confined flow with branchings and obstacles. We first introduce how to determine the stream function values at boundaries, and then describe an interpolation scheme for calculating the values inside the flow domain. Finally, we describe how to handle obstacles.

3.2.1 Boundary Values

An important characteristic of the stream function ψ is that the volume flow rate between any two points in a flow field is equal to the numerical difference in ψ between those points. The proof can be found in any Fluid Mechanics textbooks such as [Whi01]. Based on this relationship, the stream function value at a boundary should be constant because there is no flow through solid surfaces. Similarly, any two connected boundaries should take the same boundary value. Furthermore, by taking the two points on the two boundaries of a channel respectively, we have

$$Q = \psi_L - \psi_R, \tag{3.2}$$

where Q is the volume flow rate through the channel and ψ_L and ψ_R are stream function values respectively on the left and right boundary looking downstream of the channel (Figure 3.3).

Now let's analyze the case in a river network. First, we merge each pair of connected boundaries at junctions, because they should take the same stream function value (Figure 3.4). Supposing that a river network has n channels and m junctions, we will have $n-m+1$ unknown boundary values after merging connected boundaries. From Equation 3.2, we can obtain n equations for the channels. Considering the conservation of Q at junctions, in fact we only have $n-m$ independent equations. Therefore, given one of the boundary values, we can solve all the others. In practice, we can set one of the boundary values to an arbitrary value since the velocity only relates to the partial

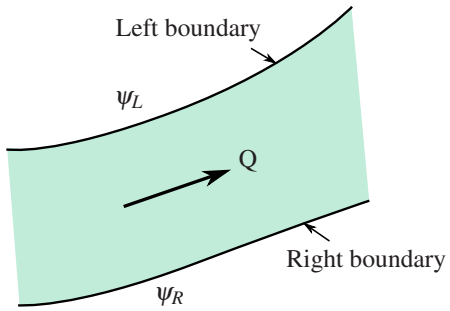


Figure 3.3: Stream function values ψ on channel boundaries relate to the volume flow rate Q by: $Q = \psi_L - \psi_R$.

derivatives of the stream function.

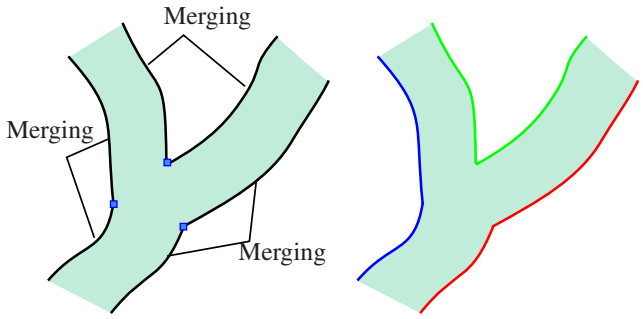


Figure 3.4: To ease the calculation of boundary values, we merge the connected boundaries at each junction. Left: there are six boundaries before the merging. Right: Three boundaries left after the merging.

The flow directions and the volume flow rates Q must be determined before applying the above method. Such information is usually available in a hydro database. In some applications, we can also see them as the parameters for controlling the animation. To do this, we note that the flow rates must satisfy the rule of volume balance at the junctions. In the following, we introduce a method for estimating flow rates that satisfy this constraint. We assume that there is only one downstream channel at each junction, which is a common assumption in Hydrology. Thus the volume balance equation at any junction is given as

$$Q_d = \sum_i Q_{u_i} \tag{3.3}$$

where subscript d denotes downstream and u denotes upstream. We also assume that the volume flow rate through each channel upstream the junction is proportional to the

width of that channel:

$$Q_{u_i} = \frac{w_{u_i}}{\sum_j w_{u_j}} Q_d \tag{3.4}$$

where w is channel width. Now if we set the volume flow rate of the most downstream channel and then perform an upstream traversal with applying Equation 3.4, we can get Q for all other channels.

3.2.2 Interpolation Scheme

Once the boundary values have been determined, the next task is to construct a reasonable stream function ψ inside the flow field. We resort to an interpolation method where the assumption is that the interpolated values should be influenced more by nearby boundaries and less by more distant boundaries. Thus our method can be viewed as a variant of Shepard's method (*i.e.,* inverse distance weighted interpolation method) [She68] originally proposed for the interpolation of scatter points. As we stated before, an interesting feature of Shepard's method is that the resulting interpolant is similar with the class of harmonic functions [GW78]. Recall that the governing equation with a single unknown variable ψ for a 2D irrotational flow is the Laplace equation. This explains why our method can generate plausible flow.

By replacing the distance to points with the distance to curves, the simplest version of the Shepard method has been adapted in [KKN+89] for contour interpolation:

$$h_s = \frac{\sum_i h_i/d_i}{\sum_i 1/d_i}, \tag{3.5}$$

where h_s is the value at the sample point P_s, h_i is the value of the i^{th} contour, and d_i is the distance from P_s to the i^{th} contour. The problem of contour interpolation is similar to ours. However, their scheme is not sufficient for us. First, we wish to query distances only in local regions, for performance reasons. Second, in order to imitate different boundary conditions, we need handles for controlling the gradient of the interpolated values.

Our interpolation scheme can be described as follows. Given a point P, let d_i be the distance from P to the boundary B_i on which the stream function value is ψ_i (Figure 3.5).

The interpolated stream function value is

$$\psi(P) = \frac{\sum_i w(d_i)\psi_i}{\sum_i w(d_i)}. \tag{3.6}$$

Here we define the weighting factor w as

$$w(d) = \begin{cases} d^{-p} \cdot f(1 - d/s), & \text{if } 0 < d \leq s, \\ 0, & \text{if } s < d, \end{cases} \tag{3.7}$$

where s is the influence size of boundaries, p is a positive real number and f is defined as

$$f(t) = 6t^5 - 15t^4 + 10t^3. \tag{3.8}$$

Notice that the function $\psi(P)$ is Lipschitz continuous and $\lim_{P \to B_i} \psi(P) = \psi_i$.

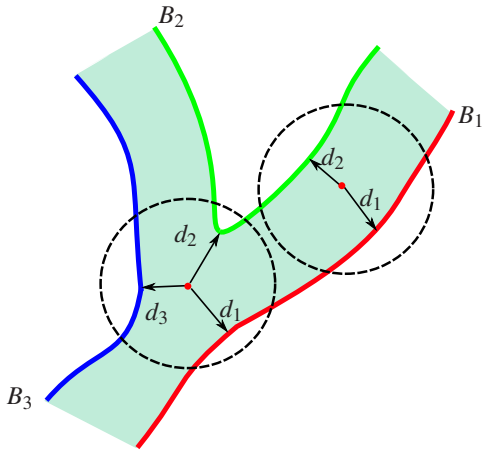


Figure 3.5: Distances to boundaries are used for interpolating the stream function. Given a point, only the boundaries that intersect with the circular search region are considered.

In Equation 3.7, for performance reasons we consider the influence size of boundaries s. Thus for a point in question P, we only need to query the boundaries which intersect with the circular search region of radius s around P (Figure 3.5). The boundaries outside of this region have zero weighting and may be excluded without effect. On the other hand, our experiments demonstrated that, for generating plausible channel flow, we should guarantee that any point in the flow is influenced by at least two

boundaries. Thus, s should be larger than the river width and indeed even larger so that branching regions are correctly interpolated. It can be set as a constant adapted to the worst case. However, for efficiency it is better to adapt it to the river width.

The power of distance p in Equation 3.7 is an important parameter for controlling the gradient of ψ. In the context of scattered data interpolation, [GW78] gave a proof that the resulting interpolant of Shepard's method has zero gradient at every data point when $p > 1$ and is not differentiable when $0 < p \leq 1$. Following that proof, we can get a similar result for $\psi(P)$. If $p > 1$,

$$\lim_{P \to B_i} \frac{\partial \psi(P)}{\partial d_i} = 0, \tag{3.9}$$

and if $0 < p \leq 1$, the partial derivative fails to exist at boundaries. Thus we can take $p > 1$ for imitating no-slip boundary, and $0 < p \leq 1$ for slip boundary.

3.2.3 Handling Obstacles

In this section we discuss the way in which obstacles are treated. Regarding the interpolation of the stream function, the boundaries of obstacles are not different from the boundaries of channels. Thus we simply need to include both the boundaries of obstacles and those of channels in Equation 3.6. Certainly, we need to determine the boundary values of obstacles before interpolating. Let C_i be the center of an obstacle O_i. We compute the stream function $\psi(C_i)$ by Equation 3.6 using only the boundaries of channels. Then we use $\psi(T_i)$ as an estimation of the boundary value of O_i.

In the weighting function Equation 3.7, we can assign independent influence size s for each obstacle. Thus we can make the velocity pattern around an obstacle adaptive to its size. Note that, in such cases, the search radius for the point in question should be the maximum s.

3.3 Implementation Details

Since our interpolation method relies heavily on the distance to boundaries, we need an efficient method for computing these distances. We achieve this by incorporating our velocity generation method into a tiling based terrain system such as [Bru08].

When a new terrain tile is created at run time, we associate it with a *hydro tile* which contains data structures for accelerating distance calculation. The concept of distance fields [FPRJ00] is a common data structure for this purpose. However, obtaining a single shortest distance to nearby boundaries is not enough for our interpolation scheme. For interpolating, we need to know several shortest distances respectively to different boundary curves that are characterized by their different boundary values (Figure 3.5). Thus, building a single distance field as [FPRJ00] is complicated and generally too expensive in the context of the real-time generation of hydro tiles. Instead, we build a quadtree in each hydro tile to store the segments of boundary curves which intersect with the tile (Figure 3.6). We construct the quadtree with a top-down approach. Cells are recursively subdivided if the number of segments inside cells is larger than a threshold. During distance queries for a given point, the segment quadtree allows us to exclude segments which do not intersect with the search region of the query point, thus fast distance calculation is possible.

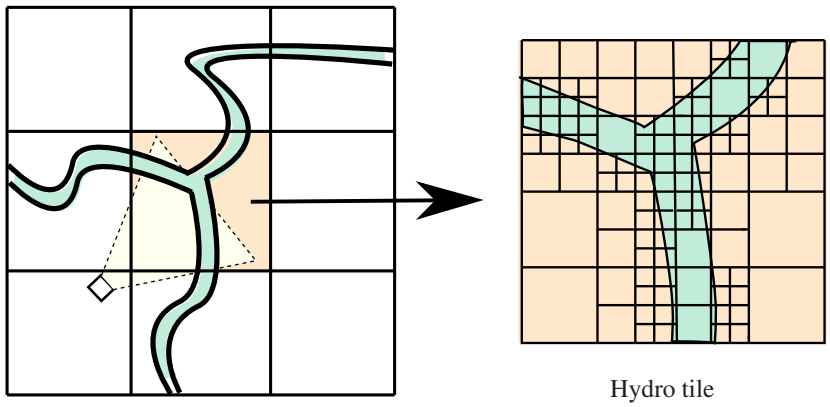


Figure 3.6: Terrain tiles and a hydro tile. We associate a hydro tile to a newly created terrain tile. In the hydro tile, a quadtree that stores the segments of boundary curves is constructed for accelerating distance calculation.

When moving obstacles are considered, we prefer to use an independent data structure for each obstacle. Thus we can avoid frequently updating the segment quadtree. For obstacles which can be approximated by simple shapes, we may use analytic methods to calculate distances. Otherwise, we can build a local adaptive distance field [FPRJ00] for each obstacle.

3.4 Results

To validate our method, we compared the streamlines generated by our method against the ones generated by a potential flow solver (Figure 3.7). The similarity between the two results shows that our method is a reasonable approximation. Figure 3.8 demonstrates that slip and no-slip boundary conditions can be simulated by using various distance power p.

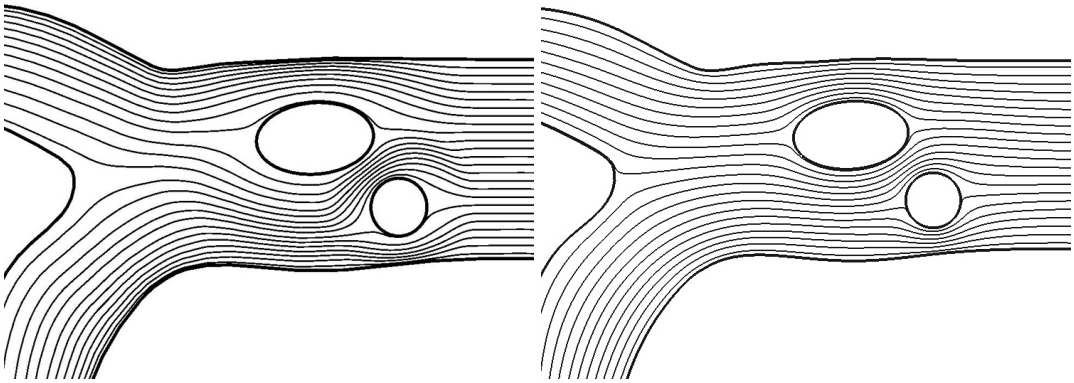


Figure 3.7: The streamlines of a flow past obstacles and through a junction. *Left:* Generated by our procedural method with distance power $p = 1.0$. *Right:* Generated by the potential flow solver in *OpenFOAM* [Ope08], a CFD software.

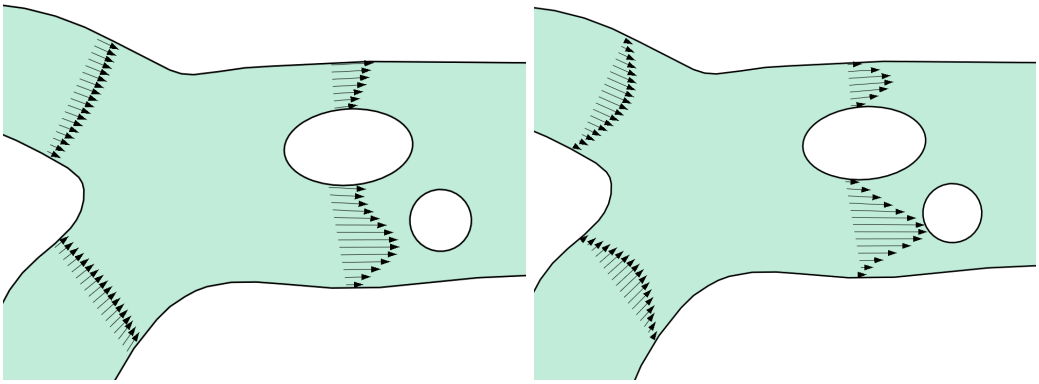
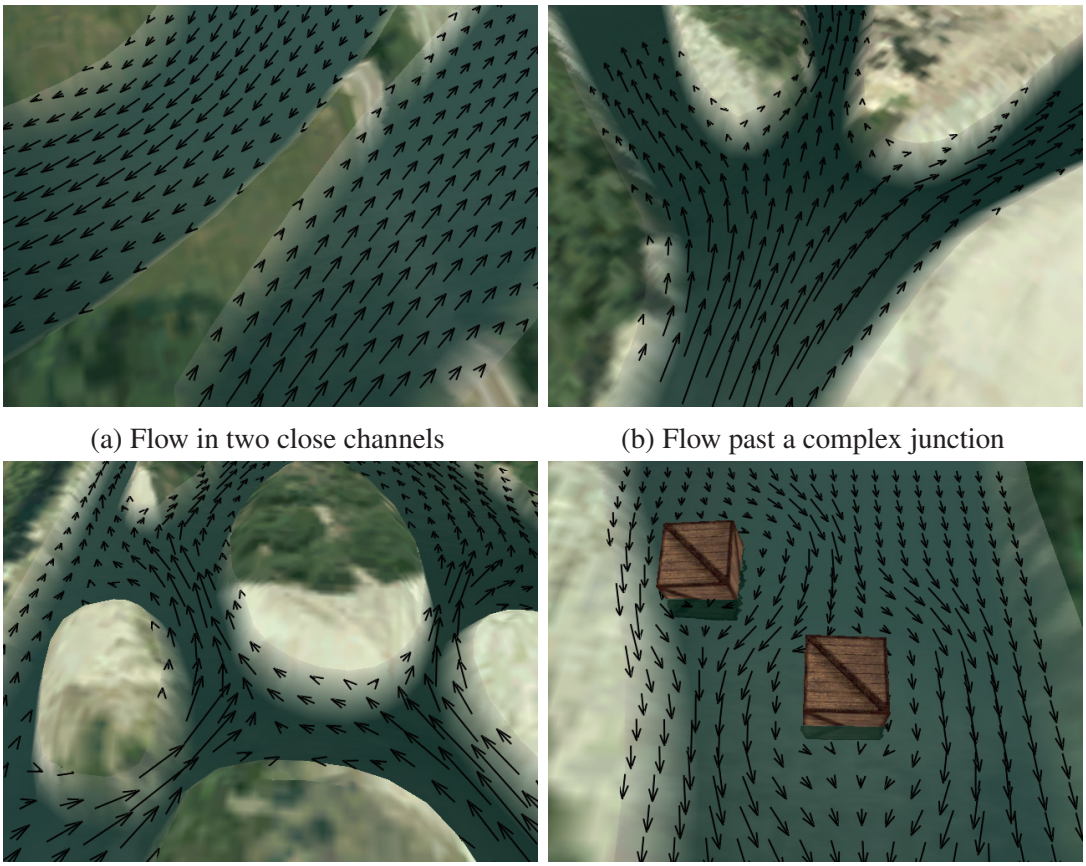


Figure 3.8: Velocity profiles generated by our method. Slip and no-slip boundary conditions can be simulated with $p = 0.9$ (*left*) or $p = 1.2$ (*right*), respectively.

To demonstrate the performance and robustness of our method, we applied it to a terrain application by combining the technique of waves sprites introduced in Chapter 5.

In the application, the procedural velocity was used for advecting water waves and floating objects. Figure 5.7 shows that the timing cost of our method (labeled with *particles advection*) holds a small portion of each animation step. Figure 3.9 shows several velocity fields generated by our method in some special or complicated cases. The visually convincing results demonstrate that our method is robust. In the accompanying video [1], we show that our method is fast enough to support on-the-fly river editing, *e.g.,* moving boundaries.



(a) Flow in two close channels

(b) Flow past a complex junction

(c) Flow past multi-islands

(d) Flow past floating boxes

Figure 3.9: Surface velocity vector plots from our method. It demonstrates that our method can yield visually convincing velocity fields even in various complicated cases.

[1] Available at http://evasion.inrialpes.fr/Membres/Qizhi.Yu/phd/

3.5 Discussion

3.5.1 2D Flow Hypothesis and Terrain Slope

Our 2D flow hypothesis is valid for constant water depth (and homogeneous velocity profile along each water column). To account for depth $h(x,y)$ variations we should simply consider $q(x,y) = v(x,y)h(x,y)$ instead of v in our derivation: $\nabla \cdot q = 0$. Similarly, we will have $q = \nabla \times \psi$. This means if we know the water depth h, we can get the velocity from $v = q/h$. In a steady flow, the Chézy formula provides a convenient approximation : $v = C\sqrt{Rs}$ with C the Chézy constant, s the slope, $v = Q/S$ the average velocity in a vertical section of surface S, perimeter P, and hydraulic radius $R = S/P$. Assuming the section has a known shape, e.g., a rectangle of known length l and height h, this yields h as a function of s, l, Q.

3.5.2 Unbounded or Dynamic Scenes

The fact that we can efficiently recompute the acceleration structure for distance computations yields various advantages. First, it allows our model to fit in a precomputation-free environment where visible tiles are generated on demand. This is a key condition to make our model amenable to very large terrains. Second, we can deal with interactive changes, including falling objects, or moving obstacles. Third, this property could make it possible to deal with flows with moving boundaries, such as flooding rivers, mud or lava flows.

3.6 Summary

In this chapter we have described a new stream-function based procedural method capable of calculating the flow velocity for rivers with branchings and obstacles. The method is stable and fast and is thus ready for real-time applications. Armed with tiling approaches, our method can be applied to a very large scene and even an unbounded scene. Furthermore, due to proceduralism, the result of our method can be intuitively modulated by users. In the next chapter, we turn to the study of meso-scale phenomena.

3.7 French Chapter Abstract

Dans ce chapitre, nous avons décrit une nouvelle méthode procédurale basée sur une fonction de flux, permettant de calculer la vitesse du courant pour des rivières en tenant compte des embranchements et des obstacles. Cette méthode est stable et rapide et elle peut donc être utilisée pour du temps réel. Utilisé conjointement avec des approches par tuile (*tile*), notre méthode peut gérer de très grandes scènes voire des scènes illimitées. De plus, du fait qu'elle soit procédurale, le résultat peut être facilement contrôlé par l'utilisateur. Dans le chapitre suivant, nous étudions les phénomènes de l'échelle méso, plus précisément les vagues stationnaires formées par des obstacles.

Chapter 4

Meso-Scale: Feature-Based Vector Simulation

This chapter focuses on simulating the local and structured waves on river surfaces which we classify as problems in the meso-scale (Figure 1.2). To better understand the challenges we face, let us first recall the global constraints imposed on this thesis work. First, the phenomena to be simulated happen in a large virtual scene, which requires good scalability of the solution. Second, we aim at real-time applications. Finally, the simulation model should provide intuitional handles for users to control the shape or appearance of the target wave phenomenon.

We choose a typical meso-scale phenomenon shown in Figure 4.1 as our case study. In a quasi-steady running stream, a stationary obstacle often generates a peculiar quasi-stationary wave pattern. The local water surface shows a regular and nearly sinusoidal shape, and the wave crests remain nearly in the same places, only with a little oscillating caused by small perturbations. Since this phenomenon is common in everyday streams, simulating it in a computer-generated virtual environment is very much desired.

Simulating the waves arising from obstacles is challenging. Relying on traditional CFD techniques to simulate this phenomenon requires very high-resolution discretization and thus can be prohibitively expensive to compute. It is even worth considering wide scenes such as long rivers. Instead, a phenomenological method, proposed by Neyret and Praizelin [NP01], can simulate the vector features (crest curves) of this kind of wave efficiently without numerical simulation. However, the results given in [NP01] are only schematic curves. An unsolved problem is how to construct water surfaces

Figure 4.1: Quasi-stationary wave patterns caused by obstacles in a running stream. *Left:* Note the wave upstream of a stone in the middle, and the ripples triggered upstream of it. Millimetric surface details are visible. *Right:* Intersection of the waves.

from the vector features for the rendering purpose. On the other hand, there is room for improving this method in terms of both robustness and performance.

In this chapter, we propose a feature-based vector simulation approach, and use it to simulate the waves caused by obstacles. The basic idea is to separate the representation for animation and rendering. Following the spirit of [NP01], we use a dynamic vector representation to capture the geometric features and the motion behavior of the target waves (Section 4.1). The concise vector representation allows low cost computation and provides good controllability. For rendering, we convert the vector information to a high resolution representation which is suitable for high-quality rendering (Section 4.2). Finally we superimpose the waves on the underlying mean flow which can be simulated by existing methods like shallow water model [Wu04] with relatively low resolution.

4.1 The Improved Phenomenological Model

In this section, we present our wave model for generating the geometric features of the quasi-stationary waves caused by obstacles. Given a horizontal flow velocity field and a depth field, the model is expected to output the vector features of the target waves. Note that the input fields could be much coarser than the resolution of wave details that we want to render. Since our model is mostly based on Neyret and Praizelin's work [NP01],

we first briefly review their method in Section 4.1.1. Then we detail our improvements in Section 4.1.2.

4.1.1 Review of Neyret and Praizelin's Method

Neyret and Praizelin's method assumes that the predominant wave caused by an obstacle in a shallow stream can be approximated by the shallow water theory [Sto57, page 22], which suggests that its wave speed is independent of its wavelength:

$$|\mathbf{c}| = \sqrt{gh}, \tag{4.1}$$

where g is the gravitational acceleration and h is the local water depth. Using the analogy of the shallow water theory with compressible gas dynamics, the target wave is called *shockwave* in [NP01]. In addition, Neyret and Praizelin's method assumes that the shockwave triggers a series of ripples upstream of it. Figure 4.2 gives a schematic illustration of these assumptions.

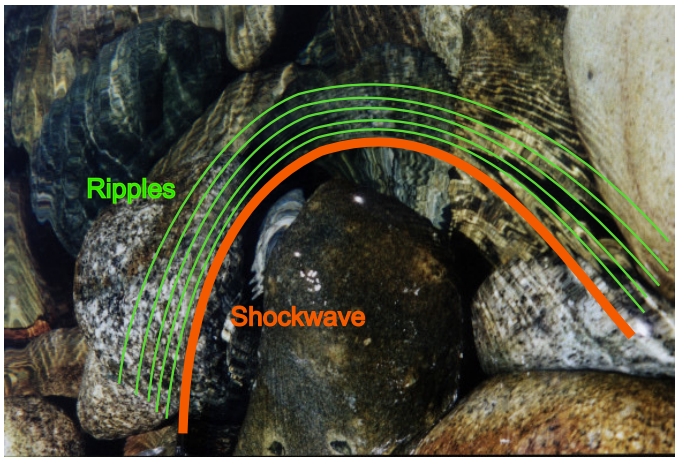


Figure 4.2: Neyret and Praizelin's method [NP01] assumes that the waves made by an obstacle is composed of a predominant shockwave and a series of ripples.

Neyret and Praizelin's method is derived by analyzing the geometric properties of the shockwave (Figure 4.3). First, in a running stream, a wave can be stationary provided the wave speed

$$|\mathbf{c}| = |\mathbf{v}| \cos \alpha, \tag{4.2}$$

where $\mathbf{v}$ is the local flow velocity and α is the propagation angle of the wave relative to the upstream direction. Therefore, the crest of a shockwave should lie at an angle

$$\alpha = \arccos \frac{\sqrt{gh}}{|\mathbf{v}|} \tag{4.3}$$

to the upstream direction. Next, let us examine where the most upstream point, the *starting point*, of a shockwave locate. At the starting point, the wave crest should be orthogonal to $\mathbf{v}$, *i.e.,* $\alpha = 0$. Substituting it into Equation 4.3, we get $|\mathbf{v}| = \sqrt{gh}$. To facilitate further discussion, we remind the Froude number:

$$F_r = \frac{|\mathbf{v}|}{\sqrt{gh}}. \tag{4.4}$$

A supercritical flow ($F_r > 1$) past an obstacle yields a subcritical area ($F_r < 1$) in front of the obstacle. This means we can expect to find a starting point at the boundary of transition ($F_r = 1$) in front of an obstacle [1].

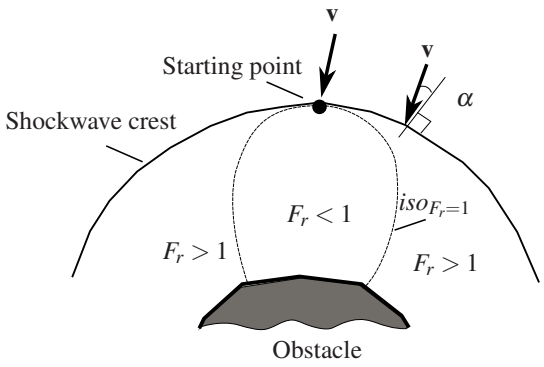


Figure 4.3: Illustration of Neyret and Praizelin's method [NP01] for simulating the waves in front of obstacles in a running stream.

Based on the above analysis, the algorithm for constructing a shockwave crest can be summarized as follows:

- Construct an $iso_{F_r=1}$ curve immediately upstream of an obstacle.

[1] As stated in [NP01], it is also possible to find the starting point downstream of an obstacle where $F_r = 1$. For clarity, we only consider the upstream case in this work, but the downstream case is mostly identical.

- Find the shockwave starting point on the $iso_{F_r=1}$ curve with the criterion of $\mathbf{v} \perp iso_{F_r=1}$.
- Construct the shockwave crest by iteratively creating line segments according to Equation 4.3.

For accounting for animation, a quasi-stationary velocity field is constructed following the sprit of [WH91]. First, a stationary velocity field $\mathbf{v}_s$ is precomputed by solving the Laplace equation on a 2D grid with the finite difference method. Next, the velocity fields of local perturbations $\mathbf{v}_{p_i}$ are superimposed on the stationary velocity field:

$$\mathbf{v} = \mathbf{v}_s + \sum_i \mathbf{v}_{p_i}. \tag{4.5}$$

For example, $\mathbf{v}_{p_i}$ can be a sink or a source with a small radius. Note that these perturbations should be advected with the stationary flow $\mathbf{v}_s$.

Since the velocity field $\mathbf{v}$ is time-varying, $iso_{F_r=1}$ curves and shockwave starting points need to be updated at each time step. Certainly, simply generating new shockwaves from new starting points at each time step would not yield a visually convincing result. It will lead to an instant global change of a whole wave, but in fact the perturbation information should not travel along the whole wave at once. For accounting this propagation, Neyret and Praizelin's method achieves the animation by updating the vertices of an existing shockwave (a segment list). It updates the position of the vertices not only according to local velocity variations but also according to the perturbations gradually propagated from upstream vertices.

4.1.2 Our Improvements

In order to increase performance and robustness, we improve Neyret and Praizelin's method as follows.

Updating Shockwave Starting Points

Neyret and Praizelin's method generates new $iso_{F_r=1}$ lines at each time step, which has several drawbacks. An abrupt velocity perturbation might occasionally lead to a discontinuity of an $iso_{F_r=1}$ line which will result in the popping of the corresponding shockwave. Furthermore, reconstructing $iso_{F_r=1}$ at each simulation step is time-consuming.

For obtaining better robustness and higher efficiency, we construct an $iso_{Fr=1}$ line only once, and directly update the shockwave starting point according to local velocity perturbations. For each shockwave starting point $\mathbf{x}$, we store its initial location $\mathbf{x}_0$ and a local steady velocity gradient $d\mathbf{v}_0 = \nabla|\mathbf{v}_s(\mathbf{x}_0)|$. As shown in Figure 4.4, at each time step we update $\mathbf{x}$ with an offset λ from $\mathbf{x}_0$ along the direction $d\mathbf{v}_0$:

$$\mathbf{x} = \mathbf{x}_0 + \lambda \frac{d\mathbf{v}_0}{|d\mathbf{v}_0|}, \tag{4.6}$$

to ensure $|\mathbf{v}(\mathbf{x})| = |\mathbf{c}|$, *i.e.,* $Fr = 1$. Now our goal is to solve λ at each time step. Using linear approximation, we can estimate stationary flow velocity $\mathbf{v}_s(\mathbf{x})$ with

$$\mathbf{v}_s(\mathbf{x}) \approx \mathbf{v}_s(\mathbf{x}_0) + \lambda d\mathbf{v}_0. \tag{4.7}$$

In addition, we assume that only the nearest perturbation $\mathbf{v}_d$ has a significant influence on the position of a shockwave starting point. Substituting Equations 4.6 and 4.7 in Equation 4.5 yields

$$|\mathbf{v}(\mathbf{x})| = |\mathbf{v}_s(\mathbf{x}_0) + \lambda d\mathbf{v}_0 + \mathbf{v}_p(\mathbf{x}_0 + \lambda \frac{d\mathbf{v}_0}{|d\mathbf{v}_0|})| = |\mathbf{c}| \tag{4.8}$$

Finally, we solve the above equation for λ by using a simple iterative scheme.

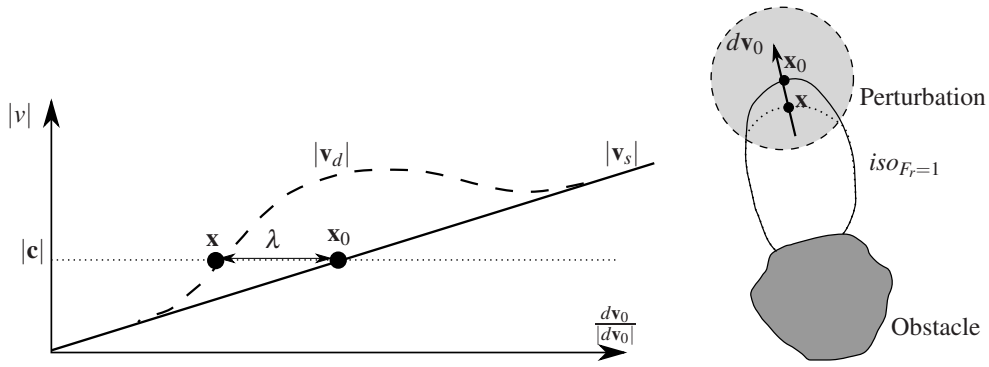


Figure 4.4: Updating a shockwave starting point $\mathbf{x}$ from its initial value $\mathbf{x}_0$ in the vicinity of a perturbation.

Tracing Shockwaves

Once having obtained a shockwave starting point, Neyret and Praizelin's method constructs a corresponding segment list which represents a shockwave by making use of Equation 4.3. However, this Eulerian scheme is not well suitable for animation, because it makes the propagation of perturbations along a segment list complicated and unstable. Instead, we rely on a Lagrangian representation which is more suitable for advecting physical quantity.

Before detailing our method, let us first investigate the motion of the elements of a shockwave (Figure 4.5). In a running stream, waves not only propagate at the wave velocity **c** but also advect with the current of velocity **v**. Hence the resultant velocity of a wave element is the vector sum of the two velocities:

$$\mathbf{v}_e = \mathbf{v} + \mathbf{c}. \tag{4.9}$$

Equation 4.2 tells us that the component $|\mathbf{v}|\cos\alpha$ of stream velocity at right angles to the crest cancels the crest's motion at the wave speed $|\mathbf{c}|$. Thus we have

$$|\mathbf{v}_e| = |\mathbf{v}|\sin\alpha = \sqrt{|\mathbf{v}|^2 - |\mathbf{c}|^2}, \tag{4.10}$$

and $\mathbf{v}_e$ is tangent to the wave crest.

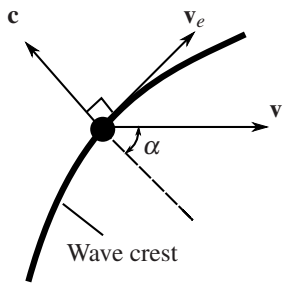


Figure 4.5: The velocity of a wave element $\mathbf{v}_e$ is the vector sum of the wave velocity **c** and the stream velocity **v**. Here the wave crest propagates at the angle of α to the upstream direction.

Our method uses particles to represent the wave elements discussed above. At each

time step, we generate two new particles respectively at the two sides of a shockwave starting point and update all existing particles with $\mathbf{v}_e$. In addition, in order to imitate the attenuation of wave energy, we associate an intensity value to each particle. The intensity value starts from 1 and linearly decrease to 0 in a given time period. We kill a particle once its intensity value vanishes. Finally, connecting existing particles with line segments gives us a shockwave crest (Figure4.6).

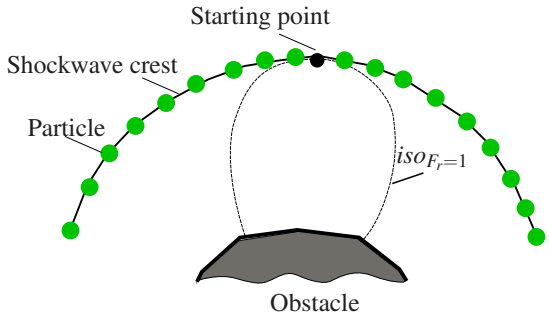


Figure 4.6: Using particles to trace the trajectory of a shockwave.

4.2 Efficient High Quality Rendering of Waves

We next turn to the problem of rendering a water surface on which the quasi-stationary waves caused by obstacles are present. We aim at quality rendering with minimum memory and computation cost. In real-time graphics, the surface of a large body of water is often represented by a height field. However, this regular grid representation is not optimal in our case. On one hand, the wave phenomenon we focus on is local, *i.e.,* it appears only at some specific locations but not everywhere in the flow domain. Therefore, a wanted representation scheme should be adaptive, *i.e.,* use higher resolution at wave locations and lower resolution elsewhere. On the other hand, as shown in Figure 4.7, using a mesh aligned with a feature line is preferable than using a regular grid for reducing geometric aliasing and normal noise [BK01].

To overcome these drawbacks, we propose an adaptive representation scheme by exploiting the vector wave information that results from our simulation model. We model the water surface by superposing *wave surfaces* on a *mean water surface* (Figure 4.8).

As has been stated in Section 1.2, the mean water surface should be handled in the macro-scale of our framework. Therefore, we treat it as known in the present section. The mean water surface can be represented by a coarse mesh or height field. Each wave surface is then represented by a fine mesh strip aligned to the feature curve of the corresponding wave. In the next subsection, we present how to construct these wave surfaces and superimpose them on the mean water surface. Then we treat wave intersection in Section 4.2.2.

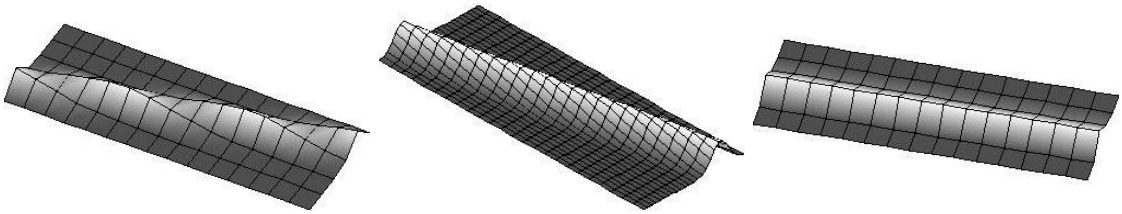


Figure 4.7: Comparison of the rendering results obtained with different meshing scheme. *Left*: Using a regular grid over the domain, quads as thin as a narrow feature can still generate very distracting geometric aliasing. *Middle*: The sampling rate should be even greater, given by Shannon's theorem. *Right*: Using a mesh aligned with the feature line prevents geometric aliasing with fewer quads.

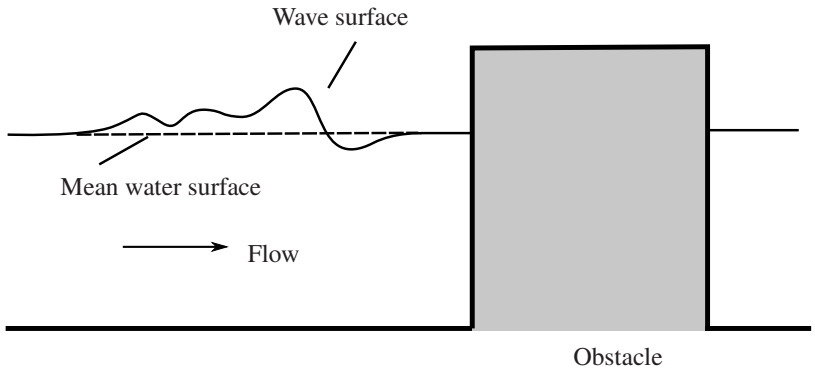


Figure 4.8: A river surface is modeled by superposing wave surfaces on a base (mean) water surface.

4.2.1 Constructing Wave Surfaces along Shockwave Crests

The waves caused by an obstacle in the flow consist of a dominating shockwave and a train of parasitic ripples upstream of it. To construct the surface, in theory we need

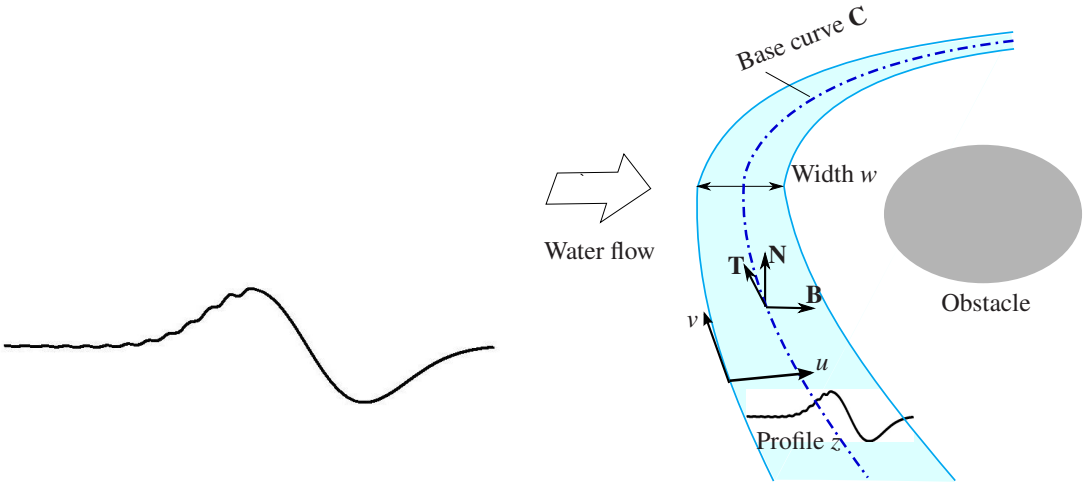


Figure 4.9: *Left:* Our custom wave profile $z(x)$ defined in Equation 4.16. *Right:* Sweeping the profile curve along a base curve (a shockwave crest) forms a wave surface.

to simulate the crests, profiles and amplitudes of both the shockwave and the ripples. However, what we can obtain now from the simulation model introduced in the previous section is only the shockwave crests. Fortunately, one important feature of the wave pattern we target is that all the wave crests are nearly regular and parallel. Therefore, it is reasonable to assume that the wave surface to be constructed is the resultant of sweeping a profile curve along a shockwave crest (Figure 4.9). Here, the profile curve represents the superposition of one shockwave and a series of ripples. With this assumption, we are able to define the wave surface based on simulated shockwave crests and user given profile curves.

We formulate the surface definition as follows. The sweeping operation uses a shockwave crest as its base curve $\mathbf{C}(u)$. Let $\langle \mathbf{T}, \mathbf{B}, \mathbf{N} \rangle$ be a local frame moving along the base curve, with $\mathbf{T}$ the unit tangent to the base curve, $\mathbf{N}$ the normal of the mean water surface, and $\mathbf{B} = \mathbf{T} \times \mathbf{N}$. Given a normalized wave profile $z(v)$, a wave amplitude function $a(u)$ and a wave width function $w(u)$, we define the parameterized wave surface as:

$$\mathbf{S}(u,v) = \mathbf{S}_b(\mathbf{x}(u,v)) + a(u) \cdot z(v) \cdot \mathbf{N}, \tag{4.11}$$

where S_b is the base water surface and

$$\mathbf{x}(u,v) = \mathbf{C}(u) + v \cdot w \cdot \mathbf{B}. \tag{4.12}$$

If we neglect the derivative of amplitude $a(u)$, the surface normal can be calculated by

$$\mathbf{N}_w(u,v) = -\frac{a}{w}\frac{\partial z}{\partial v}\mathbf{B} + \mathbf{N} \quad \text{renormalized.} \tag{4.13}$$

Note that this formula allows us to compute normals efficiently by using pre-computed derivatives of the wave profile z.

Our goal now is to tessellate the parameterized wave surface, $\mathbf{S}(u,v)$, to get a mesh strip that is able to minimize the geometric aliasing. As suggested in [BK01], an effective way to achieve this is to construct a quad mesh whose edges are aligned to iso-parameter lines (Figure 4.10). We build the quad mesh in two steps. *First*, we create rib curves uniformly sampled along the base curve. Meanwhile, we let the rib curves be orthogonal to the base curve. Note that the rib curves may intersect each other when the curvature radius of the base curve is less than $w/2$. In this case, we relax the the requirements of orthogonality. *Second*, we uniformly sample on the rib curves to create lines in another parameterization direction, *i.e.,* , v direction. The sampling density is controlled by the LOD scheme that will be introduced in Section 4.3.2.

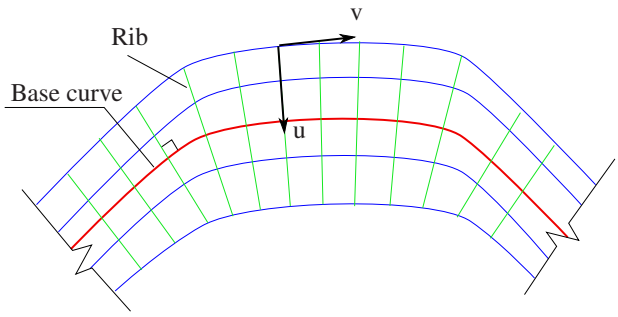


Figure 4.10: Wave surface is sampled by a quad mesh aligned with iso-parameter lines for reducing normal noise.

Once having tessellated the wave surfaces, we need to apply the mesh strips upon the mesh of the mean water surface. Drawing them separately would not give a correct result, since wave surfaces may have negative offset from the mean water surface (Figure 4.8). The mesh-stitching technique described in [L.P00] may work for merging these meshes. However, this method requires remeshing the two surfaces to be merged. Since the waves are dynamic, the remeshing would have to be done in each frame. It is not only computationally expensive but also leads to an extra time cost for uploading

the new surface data to the GPU memory.

To avoid remeshing the mean water surface, we solve the mesh composing problem by using the stencil buffer found on graphics hardware. We first draw all wave surfaces into not only a color buffer but also a stencil buffer. The stencil buffer is able to indicate which fragments are covered by the wave surfaces. Then we draw the mean water surface into the color buffer where the wave surfaces are not present. By using this method, we do not need to modify the mean water surface. Still, we need to ensure a perfect continuity between the wave surface and the base water surface. When the boundary of a wave surface intersects with the edges of the mean water surface mesh, geometry gaps may appear. To avoid this, we insert extra ribs by interpolation as shown in Figure 4.11.

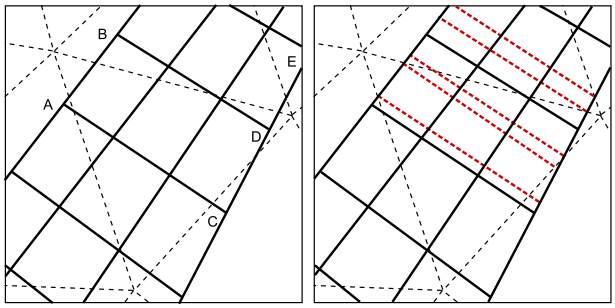
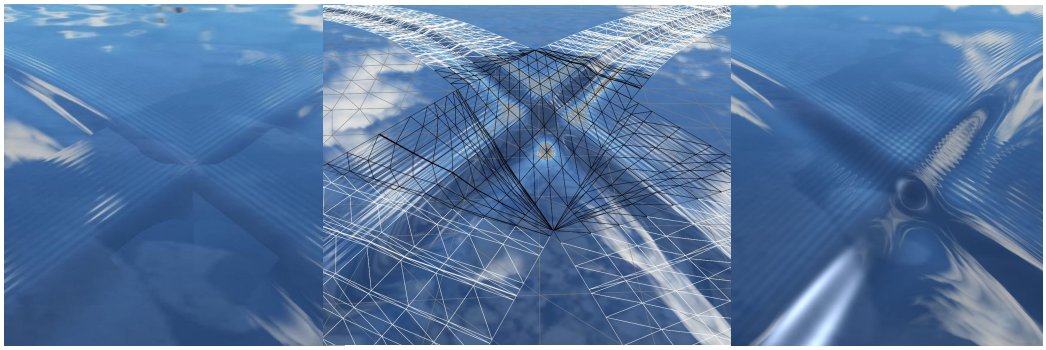


Figure 4.11: *Left:* Mean water surface mesh (dashed) and original mesh strip (plain). Boundary edges like AB, CD and DE must be split because they intersect with the edges of the mean water surface mesh. *Right:* The mesh strip with added extra ribs (red dashed liens) by interpolating.

4.2.2 Handling Wave Intersection

When two obstacles are close, the stationary waves may intersect (Figure 4.1). At the intersection part, waves caused by different obstacles are superposed. Simply drawing two wave surfaces without handling the intersection will lead to an un-smooth result (Figure 4.12a), and can not account for the addition of amplitudes. To properly superpose waves, we construct a dedicated mesh patch for the intersection part. Suppose that two wave surfaces $\mathbf{S}_1(u_1, v_1)$ and $\mathbf{S}_2(u_2, v_2)$ intersect as shown in Figure 4.13. The



(a) (b) (c)

Figure 4.12: The intersection of two wave patterns. (a) A simple Z-buffer rendering without handling crossing. (b) We generate a dedicated mesh patch for the intersection part. (c) Final rendering with a proper intersection treatment.

superposed surface of the intersection part is defined by:

$$\mathbf{S}(u_1,u_2,v_1,v_2) = \mathbf{S}_1(u_1,v_1) + \mathbf{S}_2(u_2,v_2) - \mathbf{S}_b(u_1,u_2). \tag{4.14}$$

The surface normal can be calculated by:

$$\mathbf{N}_w(u_1,u_2,v_1,v_2) = -\frac{a_1}{w_1}\frac{\partial z}{\partial v}(v_1)\cdot\mathbf{T}_2 + \frac{a_2}{w_2}\frac{\partial z}{\partial v}(v_2)\cdot\mathbf{T}_1 + \mathbf{B}_1\times\mathbf{B}_2 \quad \text{renormalized.} \tag{4.15}$$

We tessellate the intersection part with a structured grid aligned with the grid lines of $\mathbf{S}_1$ and $\mathbf{S}_2$. Moreover, we need to cut out this intersection part in both $\mathbf{S}_1$ and $\mathbf{S}_2$.

4.3 Implementation details

4.3.1 Wave Profile

The choice of the wave profile $z(x)$ is up to users except the constraint: $z(x) = z'(x) = z''(x) = 0$ at the two ends to ensure G^2 continuity between the wave surface and the base water surface. In our implementation, we used the normalized wave profile illustrated in Figure 4.9. It is defined as the sum of a gravity wave profile $z_g(x)$ and a capillary

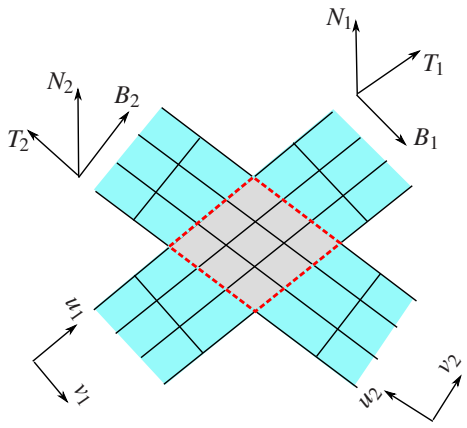


Figure 4.13: For handling wave intersection, we cut out this part in the two intersected mesh, and create a dedicated mesh patch (bounded by red dashed lines). strips.

wave profile $z_c(x)$ inspired by [FM98]:

$$z(x) = z_g(x) + z_c(x), \quad x \in [-1..1], \tag{4.16}$$

where

$$z_g(x) = 2(3x + \frac{1}{\sqrt{2}})e^{-(3x+\frac{1}{\sqrt{2}})^2}, \tag{4.17}$$

and

$$z_c(x) = \begin{cases} .045(e^{-2x}\cos(24\pi x) - 1 - x(e^{-2}\cos(24\pi) - 1)), & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{4.18}$$

4.3.2 Bump-mapping and LOD

Defining a wave surface by sweeping allows us to calculate accurate surface normals efficiently. We always determine per-pixel normals through the analytical bump computed from Equations 4.13 and 4.15. Problems may occur when we should filter the bumps themselves, *i.e.,* when the sampling frequency (i.e. pixel size) is smaller than the frequency of waves. As predicted by Shannon's theorem, this yields aliasing. In practice, this occurs (for the capillary wave component of the profile (Equation 4.16). So, these high frequency waves must be properly filtered. We progressively fade the

capillary ripples according to the view distance d. We rewrite Equation 4.16 to:

$$z(x) = z_g(x) + \beta z_c(x), \tag{4.19}$$

where β decreases from 1 to 0 as d increases. The filtered geometric information should be accounted for in the illumination model, which is a general and tough problem. We leave it as future work.

To ensure good performance, we determine the resolution of wave mesh strips according to the view distance d. We set the level of subdivision of a wave surface in the direction u as $\lfloor \log_2((1-\delta)N_{max}) \rfloor$, with N_{max} the maximum number of grid lines, and

$$\delta = \begin{cases} 0, & d \leq d_{near}, \\ \frac{d-d_{near}}{d_{far}-d_{near}}, & d_{near} < d < d_{far}, \\ 1, & d \geq d_{far}. \end{cases} \tag{4.20}$$

where d_{near} and d_{far} correspond to the finest and the coarsest LOD, respectively.

4.3.3 Shading Model

For realistic rendering of water surfaces, we rely on existing GPU techniques. We consider reflection (global and local, with Fresnel term) and refraction effects. Local reflection and refraction are achieved by projecting reflection and refraction maps onto the water surface [AVO02]. A cube map is used to account for global environment (*e.g.,* sky) where the projective texture does not provide pixels of close objects. Finally, we combine those colors in the pixel shader using a Fresnel term.

4.4 Results

We benched our method on a scene containing 35m long 3m wide river. All tests were done on an AMD Athlon 3000+ at 1.8Ghz with an NVIDIA GeForce 8800GTS.

Figure 4.14 shows very detailed waves generated by our method in real-time. The wave phenomenon we target is only one of many kinds of waves on river surfaces. Therefore, mixing our model with other wave models is necessary in real applications.

Figure 4.15 demonstrates that our method is well compatible with other wave models using bump mapping. Figure 4.16 shows waves disturbed by floating waves. Please consult the accompanying video to see more animation results [2].

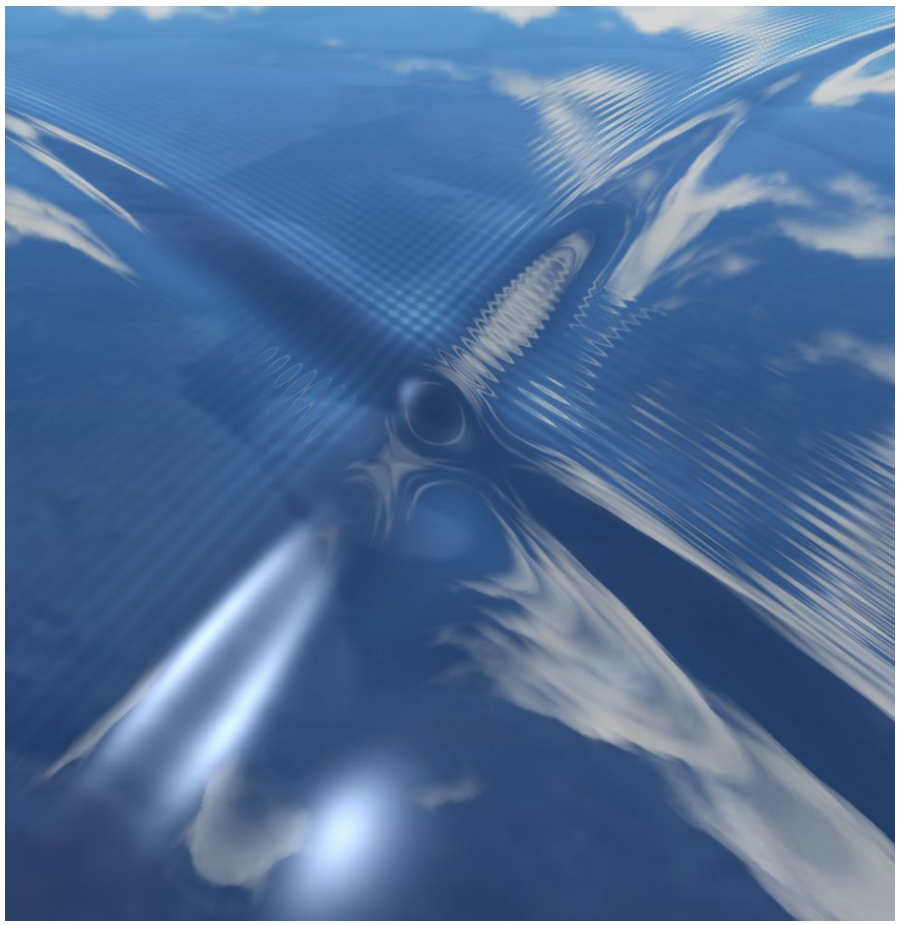
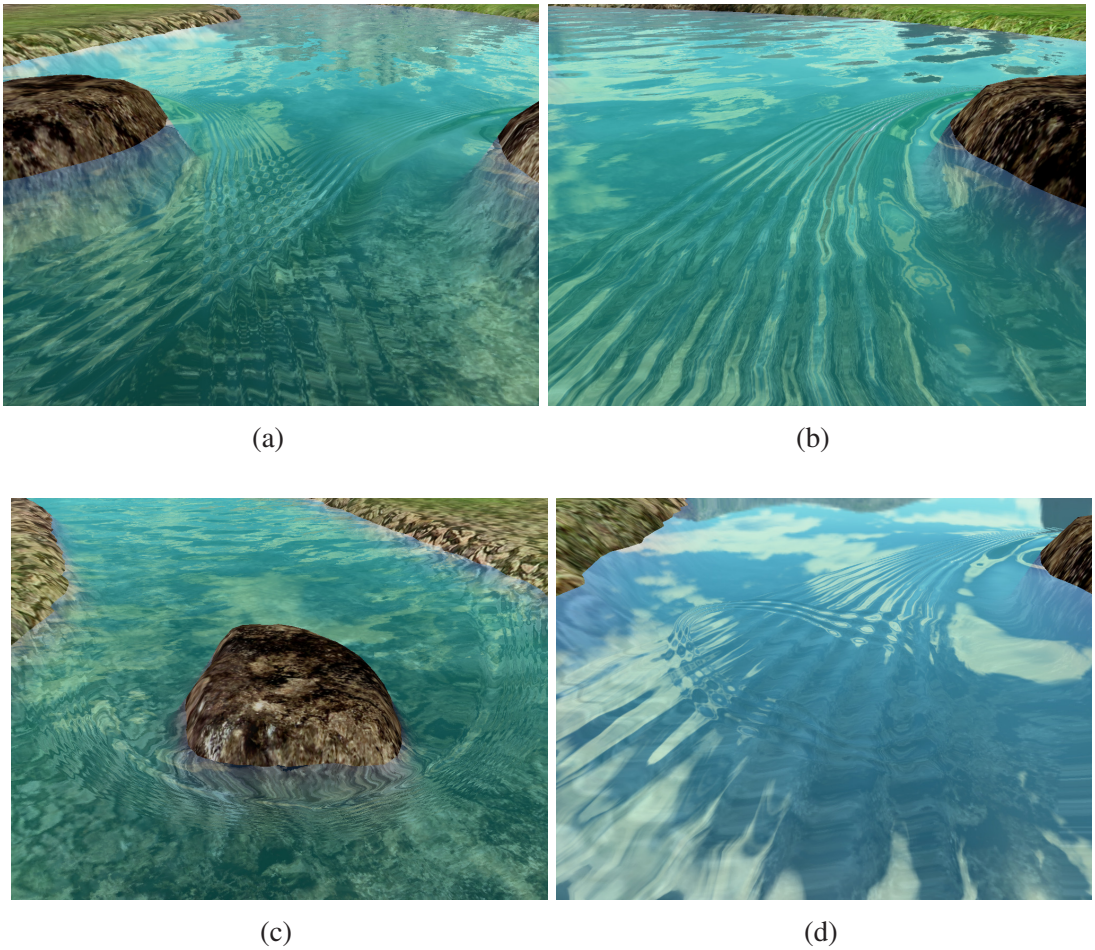


Figure 4.14: High resolution waves simulated in real-time.

To demonstrate the performance of our method, we tested it with various view distances (Figure 4.17). The simulation time includes two main parts: generating dynamic shockwave crests and constructing wave surface meshes. Table 4.4 demonstrates that our method is applicable for real-time applications.

[2] Available at http://evasion.inrialpes.fr/Membres/Qizhi.Yu/phd/



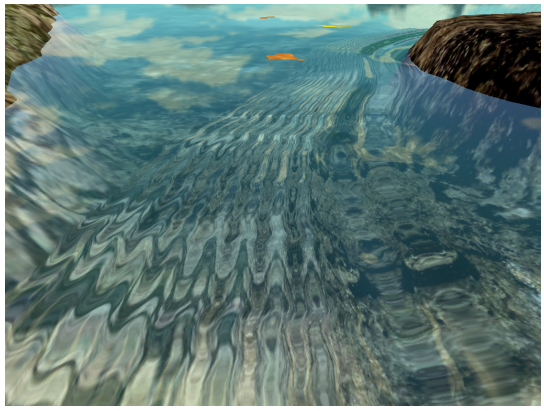
(a) (b)

(c) (d)

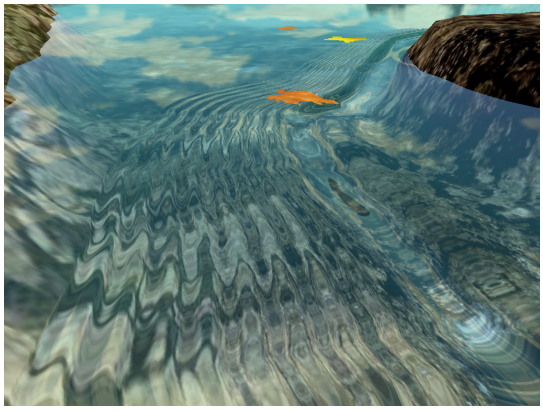
Figure 4.15: Results of superimposing noise waves (a)(b)(c) and boils (d) on our waves by using bump mapping.

4.5 Summary

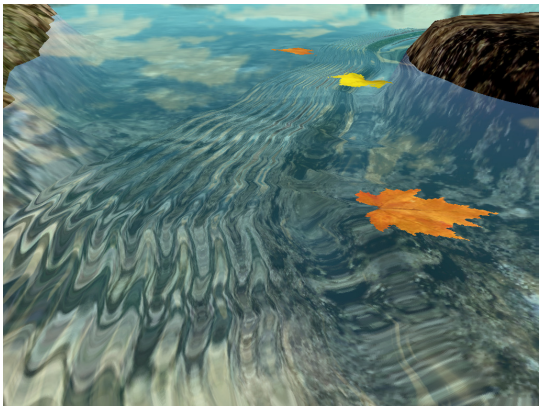
In this chapter, we have proposed a feature-based vector simulation approach for simulating local and structured waves in streams. We have chosen a typical meso-scale phenomenon: stationary waves caused by obstacles as a case study. For better performance and robustness, we have improved the phenomenological model proposed in [NP01] in two aspects: updating shockwave starting points according to local velocity perturbations and using particles to trace a shockwave crest (Section 4.1.2). For efficient and quality rendering, we have presented a method that is able to construct an adaptive and feature-aligned surface meshes according to the simulated vector features (Section 4.2).



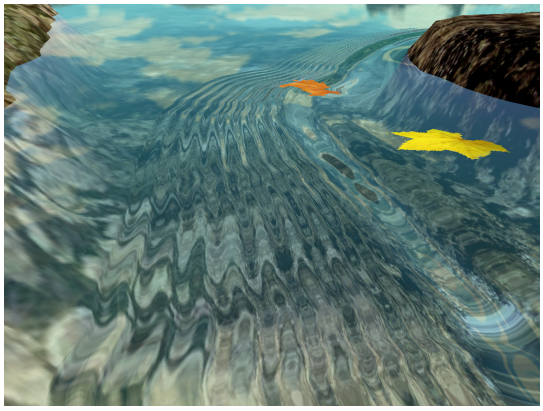
Frame 20



Frame 40



Frame 60



Frame 80

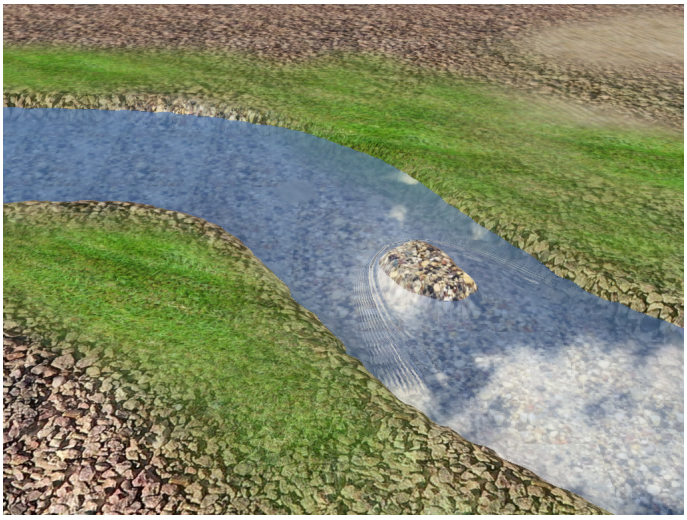
Figure 4.16: Waves disturbed by floating leaves. We obtain the quasi-stationary velocity field by superimposing small local velocity fields attached to the leaves on a stationary velocity field.

The results show that our approach permits high animation and rendering quality at cheap simulation cost. It also allows users to intuitionally control the animation, for example the shape of waves (Section 4.3.1). In the next two chapters we turn to handle micro-scale surface features of rivers with texturing methods.

4.6 French Chapter Abstract

Dans ce chapitre, nous avons présenté un phénomène représentatif de l'échelle méso : les vagues stationnaires formées par des obstacles. Nous poursuivons l'idée présentée

(a) Far view.



(b) Middle view.

(c) Close view.

Figure 4.17: Typical views considered in our performance test.

	near view	middle view	far view
shockwave crests	2.5	2.5	2.5
wave meshes	12	13	17

Table 4.1: Performance for wave simulation (unit: ms/frame)

dans [NP01], cherchant à simuler les caractéristiques géométriques des vagues. Afin d'obtenir de meilleurs performances ainsi qu'une meilleure robustesse, nous avons amélioré le modèle phénoménologique d'onde de choc proposé dans [NP01] de deux manières : en actualisant le point de départ des ondes de chocs en fonction des perturbations locales de la vitesse et en utilisant des particules pour dessiner la trajectoire des ondes de chocs (Section 4.1.2). Afin d'obtenir un rendu correct et efficace, nous avons introduit une méthode permettant de générer de manière adaptative la surface de l'eau par des maillages, en fonction de la trajectoire des vagues simulées (Section 4.2). Cette méthode permet de découpler la résolution utilisée pour la simulation de la résolution utilisée pour le rendu. Cela permet également de pouvoir contrôler aisément les caractéristiques de l'animation, comme la forme des vagues par exemple (Section 4.3.1). Dans les deux prochains chapitres, nous présentons les moyens utilisés pour gérer les caractéristiques de la surface des rivières à l'échelle micro avec différentes méthodes de texturage.

Chapter 5

Micro-Scale: Wave Sprites

This chapter and the next chapter are dedicated to handling the micro-scale visual phenomena of river surfaces. These phenomena include various small scale surface details such as waves and foams. Modeling these dynamic details are challenging because they often have complicated or even unknown physics causes. It is worse when we are under the constraints of real-time and large domain. Though there exist statistic-based methods or procedural methods for simulating small waves in deep water, they fail to simulate the advection of waves caused by the surface flow. Note that the advection of surface details is a distinct surface characteristic of rivers. Since these surface details often have similarity both in space and in time, we can consider the technique of texturing. To account for the advection, what we need is a method for generating texture sequences that can follow a given flow while preserving given statistic properties such as spectrum.

In the context of real-time graphics, the Eulerain texture advection methods proposed in [MB95, Ney03] have a similar purpose to ours. However, it is not clear how to extend them to be scalable for a very large domain. Apart from the scalability, they are not perfect in terms of spectrum preserving (which we will detail in the next chapter). To address the two issues, we explore the problem of advecting texture from the Lagrangian point of view in the following two chapters. In this chapter, we focus mainly on the issue of efficiency, while in the next chapter we will have a close look at the issue of quality.

The first task we face is to texture large river surfaces with high-resolution details but minimum memory overhead. Inspired from *texture sprites* [LHN05], we represent the

river surface by composing *wave sprites*: textured sprites which carry texture examples of waves (Sections 5.1 and 5.3). Since sprites can be updated easily, the sprite-based scheme is suitable for animation. Moreover, we need to consider following specific requirements of our problem:

- The motion of the sprites should convey the flow motion;
- The density of the sprites should be adaptive for performance reasons;
- The sprites should be well distributed (no holes and not too much overlap) to avoid biasing the global texture property after blending all sprites;
- The reconstructed global texture should ensure spatial and temporal continuity.

To meet these constraints, we propose a dynamic sampling method to maintain a set of particles on river surfaces which advect with the flow while keeping a Poisson-disk pattern in screen space (Section 5.2). Then we associate wave sprites to these particles with the consideration of spatial and temporal continuity (Sections 5.2.2 and 5.2.3). The sprite-based texture scheme together with the dynamic adaptive sampling technique provides a novel scalable texturing method, particularly suitable for advected surface details.

5.1 Wave Sprites

In this section, we introduce the basic concept of wave sprites. Figure 5.1 illustrates various coordinate systems mentioned in the following discussion. Wave sprites are texture elements that live on the river surface that can be parameterized as a mapping space (*e.g.,* top view of the scene). Their life cycles and locations are managed by the sampling method introduced in the next section. Each sprite has a circular influence region with a uniform radius r in screen space. The choice of r will be discussed in Section 5.2.2. We use wave sprites to carry surface details characterized by a tileable wave texture example T. For a newly created sprite, we initialize it with a random texture coordinates $\mathbf{u}_i$ which is constant during its life cycle. Given a wave sprite located

at $\mathbf{p}_i$ [1], we define its texture function in mapping space as

$$S_i(\mathbf{x}) = T(\mathbf{x} - \mathbf{p}_i + \mathbf{u}_i). \tag{5.1}$$

Then, the visual appearance of a river surface can be represented by a continuous texture function F reconstructed by composing all sprites:

$$F(\mathbf{x}) = \frac{\sum_i w_i \cdot S_i(\mathbf{x})}{\sum_i w_i}. \tag{5.2}$$

The weighting factor w_i involves both a kernel function k_i for ensuring spatial continuity and an intensity function f_i for ensuring temporal continuity:

$$w_i(\mathbf{x}, t_i) = k_i(\mathbf{x}) \cdot f_i(t_i), \tag{5.3}$$

where t_i is the age of a sprite. The functions k_i and f_i will be detailed in Sections 5.2.2 and 5.2.3, respectively.

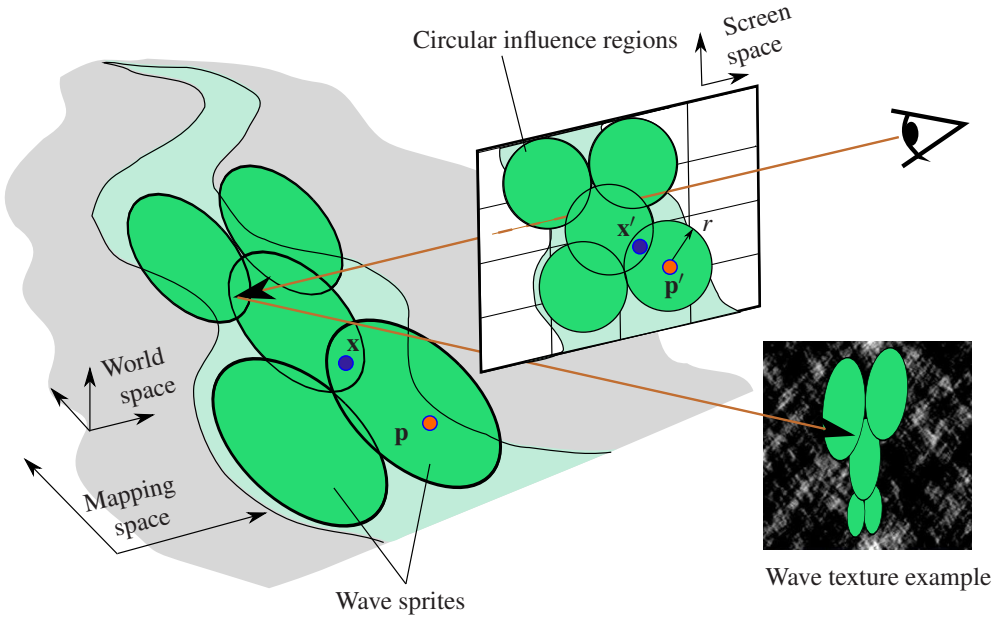


Figure 5.1: Illustration of wave sprites.

[1] Note that $\mathbf{p}'_i$, the screen space coordinate of $\mathbf{p}_i$, is the center of the corresponding circular influence region.

5.2 Dynamic Adaptive Sampling

To achieve efficient texture advection, the wave sprites defined above should advect with the river flow and be adaptive. In this section, we will introduce a sampling algorithm for addressing these problems. After that, we will discuss the considerations for ensuring the temporal coherence and spatial continuity of the reconstructed surfaces.

5.2.1 Sampling Algorithm

As we mentioned before, we rely on a set of dynamic adaptive particles to locate the sprites. The main idea is to maintain a Poisson-disk pattern in screen space. We summarize the algorithm of sampling in Algorithm 5.1.

Algorithm 5.1 Dynamic Adaptive Sampling

```
1: loop
2:     Advect particles with flow.
3:     Delete particles outside of the view frustum.
4:     Delete particles violating the minimum distance criterion d in screen space.
5:     Insert particles to keep the Poisson-disk pattern in screen space.
6:     for all new particles do
7:         Calculate world space coordinates.
8:     end for
9: end loop
```

In line 5 of the algorithm, we need a method for generating the Poisson-disk pattern from the existing particles which already respect the minimum distance criterion d. For this, we adapt an incremental method, *boundary sampling algorithm*, presented in [DH06]. This algorithm starts from a random point. The only adaptation we need is to replace the initial point with a set of existing points. Another advantage of using the boundary sampling algorithm is that it runs in $O(N)$ time.

So far we have introduced the basic steps of the sampling algorithm. Since the particles are finally used for carrying sprites, we have to consider the continuity issues during the reconstruction of water surfaces. We discuss them in the next two sections.

5.2.2 Spatial Continuity

For reconstructing a continuous surface, the sprites must fully cover the surface. It is more convenient to analyse this problem in screen space than in world space, since the influence regions of our sprites in screen space are disks with a uniform radius.

To ensure no holes in the reconstructed surface, the radius of the circular influence region of a sprite, r, must be large enough. Generally, for a Poisson-disk distribution with the minimum distance d, superimposing disks of radius no less than d centered at sampling points will lead to no holes in the sampling domain. This can be easily verified using proof by contradiction. Supposing there exists such a hole, we can take a point in the hole which is apart from any other point with a distance larger than d, which contradicts with the definition of the Poisson-disk distribution. In Algorithm 5.1, we take $d = r$ to ensure minimum overlap. However, the above analysis neglects the problem near boundaries.

Before introducing the problem near boundaries, we first explain our consideration of the sampling domain. The projected river surface in screen space usually does not fully occupy the window. Intuitionally, we should sample only inside the projected river surface. However, the projected surface may be composed of several unconnected irregular shapes. Thus we need to adapt the Poisson-disk distribution algorithm used in Algorithm5.1 to handle complex domains, which is not trivial. Instead, we simply choose to sample in the whole window. It is reasonable to do this, since our test has shown that the sampling procedure is not the performance bottleneck in our framework (Figure 5.7).

Sampling in the whole window results in two kinds of particle: outside rivers or inside rivers. Note that we should not associate sprites to the outside particles that are not advected by the flow. Otherwise we may have some fixed patches on the reconstructed surface, which looks unnatural.

Now we can return to the potential flaw near the boundaries and our method for addressing it. Figure 5.2 shows that we may face the risk of having uncovered parts near the flow boundaries. This is because some outside particles near the boundaries may invalidate some parts of the river surface for Poisson-disk sampling and they are not associated with sprites as we stated earlier. We tackle this problem by deleting those particles giving a greater chance for sampling inside the flow in the following frames.

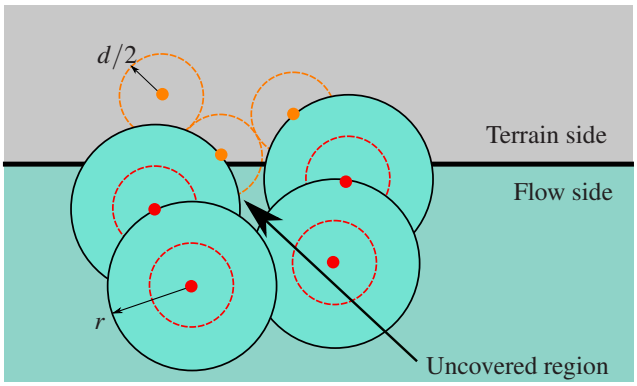


Figure 5.2: An uncovered region appears near the boundary in the flow side due to the presences of near boundary particles in the terrain side. Here, the small disks are Poisson-disks of radius $d/2$, and the large disks are the circular footprint of sprites of radius $r = d$.

We detect and delete the outside particles which have at least one inside particle falling in their circular neighborhoods of radius $2d$. The neighbor relations between particles are constructed as a byproduct of the Poisson-disk pass.

Finally, we need to ensure the continuity along the boundary of a sprite. For this, we define a kernel function that is used in the weighting factor (see Equation 5.3). For a sprite located at $\mathbf{p}$ in mapping space (Figure 5.1), the kernel function is:

$$k(\mathbf{x}) = h(1 - \frac{\|m(\mathbf{x}) - m(\mathbf{p})\|}{r}), \tag{5.4}$$

where m is the mapping from mapping space to screen space, and h is a smooth function which has zero first derivatives at both ends:

$$h(\alpha) = \begin{cases} 1, & \text{if } \alpha > 1, \\ 6\alpha^5 - 15\alpha^4 + 10\alpha^3, & \text{if } 0 \leq \alpha \leq 1, \\ 0, & \text{if } \alpha < 0. \end{cases} \tag{5.5}$$

5.2.3 Temporal Coherence

To avoid popping artifacts in the reconstructed texture, we fade in or fade out a sprite when it is inserted or deleted, respectively. Correspondingly, an intensity value f is associated with each particle. When a new particle is inserted, f is initialized to zero.

Then f gradually increases with the life time t during a predefined blending period T:

$$f(t) = \begin{cases} t/T, & \text{if } t < T, \\ 1, & \text{otherwise.} \end{cases} \tag{5.6}$$

When a particle is deleted by the sampling algorithm, it will not be considered as an existing sampling point anymore but still used for carrying the associated sprite until its intensity has decreased to zero. Let f_d be the intensity at the moment of deletion t_d. The decreasing intensity is

$$f(t) = \begin{cases} f_d(1-(t-t_d)/T), & \text{if } t - t_d < T, \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

In the texture reconstruction stage, the particle intensity is considered in the blending weight (see Equation 5.3).

5.3 Surface Reconstruction

After the treatment in the last section, we have sprites which totally cover the water surfaces visible in the view frustum, without too much overlapping. The problem is now to blend and render these sprites efficiently.

We rely on an indirection grid following the idea presented in [LHN05]. Unlikely to [LHN05], our grid is in screen space. We divide the window into a regular grid. Each cell of the grid stores a list of sprites which cover it. To avoid redundancy, we only store the indices of sprites in the grid while the parameters of sprites are stored separately in a *sprite parameters table* (Figure 5.3). The parameters of a sprite include location in mapping space, index of texture example, blending weight, and elapsed time. Both the indirection grid and the sprite parameters table are encoded into textures, called *indirection texture* and *parameters texture* respectively. This scheme allows us to treat our sprite set as a simple material described by a pixel shader and applied to an ordinary mesh rendered as a simple geometry.

The reconstruction algorithm performed in the pixel shader is summarized in Algorithm 5.2 (see also Figure 5.1).

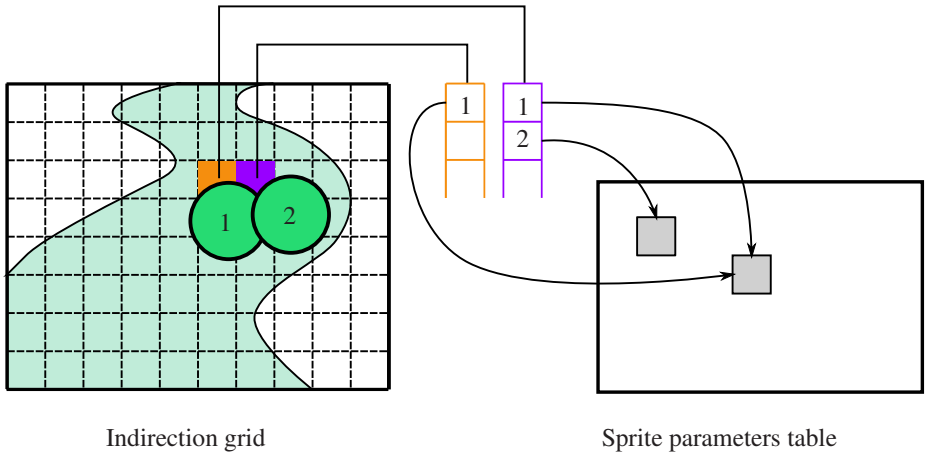


Figure 5.3: Storage scheme of sprites.

Algorithm 5.2 Surface Reconstruction Algorithm

1: **for all** pixels **do**
2: Access the indirection texture to find all the sprites covering current pixel.
3: **for all** sprites **do**
4: Access the parameters texture to get sprite parameters.
5: Calculate the contribution of current sprite by Equation 5.1.
6: Accumulate the current contribution.
7: **end for**
8: Apply ordinary shaders: bump map, reflection and refraction with Fresnel terms, and shading.
9: **end for**

5.4 Implementation Details

5.4.1 Wave Texture Examples

The surface details we focus on are wave patterns statistically populating the river surfaces, rather than local and well-structured waves such as stationary shockwaves or hydraulic jumps we treated in Chapter 4. In our implementation, we treat simple noise perturbations and wind waves, but many others could be incorporated by the user (*e.g.,* turbulences, wakes and foams).

We use a tileable texture sample to represent a kind of wave. For the time variation, one can rely on 3D textures or analytical evolution. Note that the texture data mainly represents parameters to be provided to the pixel shader, which may generate complex

results. In our implementation we store normalized heightfields and the pixel shader uses them for bump mapping and environment mapping. Precisely, for simple perturbations, we rely on Perlin noise which we prefer to precompute rather than to evaluate on-the-fly. For the wind waves, we use Fourier generation and rely on analytical time evolution as described in [Tes04].

Wave types can be carried by different sprites or shared by all sprites. In the second case, tuning parameters (amplitude, wavelength or wind direction) can be controlled at the scene level with a user-defined map or a procedural rule. Note that even wakes can be managed domain-wise using shaping and masking rules analog to the ones used in [BHN07].

5.4.2 From Screen Space to Mapping Space

When a new particle is generated by the sampling algorithm (Algorithm 5.1), we only know its coordinates in screen space. But later, we need to know its coordinates in world space for accessing flow velocity and texture addressing. Since the river surface is not flat, it is not trivial to unproject the particle onto the river surface. We solve this problem by rendering the river surface to a buffer, using the vertex coordinates as the vertex colors. Since the number of newly created particles at each frame is small , we only need to access a small set of pixels in the buffer. Instead of downloading the whole buffer to CPU, we read the corresponding pixels by individual query for each newly created particle.

5.5 Results and Discussion

In order to demonstrate the benefits of our method in real applications, we tested it in a scene of size 25×25 km^2 (Figure 5.4). The test was done on a AMD Athlon 3200 processor at 1.8 GHz with a GeForce 8800 GTS graphics board. We set the window size to be 800×600 and the radius of the Poisson-disks to be 20 pixels. The intermediate results (particles) and the final rendering results are shown in Figures 5.5 and 5.6, respectively. See also the accompanying video [2].

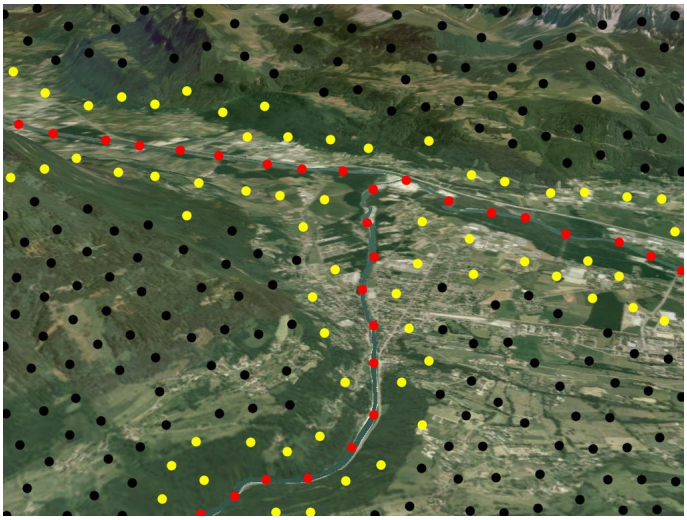
[2] Available at `http://evasion.inrialpes.fr/Membres/Qizhi.Yu/phd/`

Figure 5.4: Terrain with size 25×25 km^2.

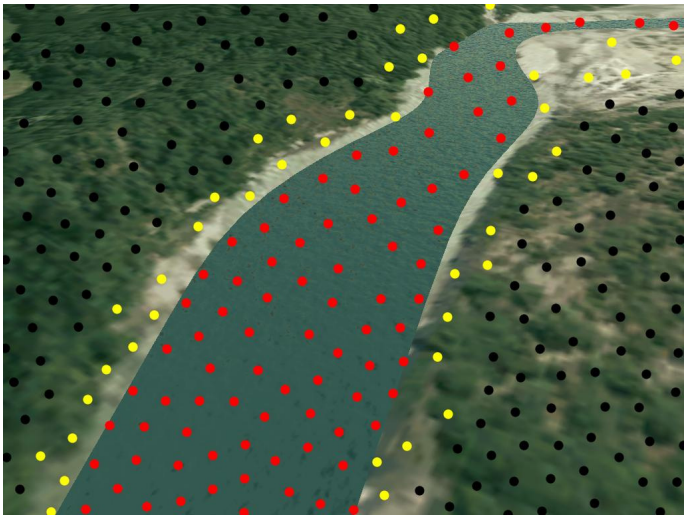
Figure 5.7 shows that given a Poisson-disk radius, the running time of our method depends linearly on the projected area of river surfaces in the window. Thus one significant feature of our method is that the performance is scene-size independent. In the test, we achieved real-time performance even in the worst case where the projected surfaces occupy the whole window. Certainly, the performance will fall down if we decrease the the Poisson-disk radius. However, a moderate value as we used in this test is sufficient due to the adaptivity of the particles and the sprite-based rendering scheme.

The accompany video shows that our method can ensure the continuity of water motion when the camera travels along a long river or zoom in and out at any moment. Furthermore, the continuity is also ensured when the user edits the rivers on the fly. The river appearances can be easily modified via the wave texture examples.

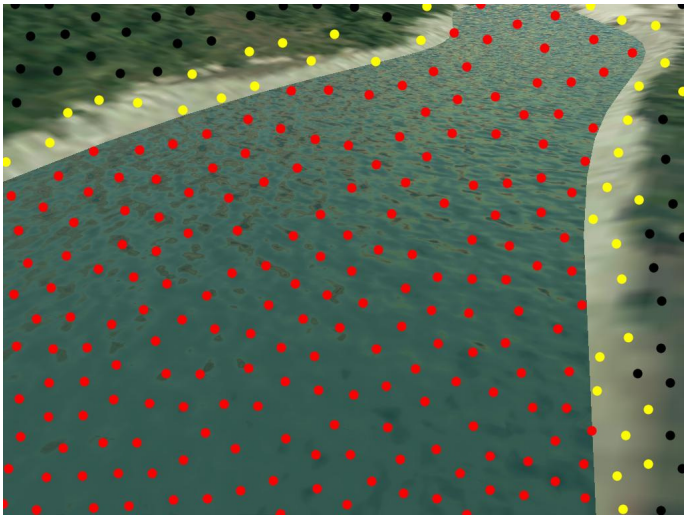
To demonstrate that advecting details with channel-confined flow is necessary for river animation, we compared our results against the animation of non-flowing water and uniform flow which can be handled by previous methods. The results demonstrate that our method shows considerable improvements (see the accompanying video).



(a) Far view.



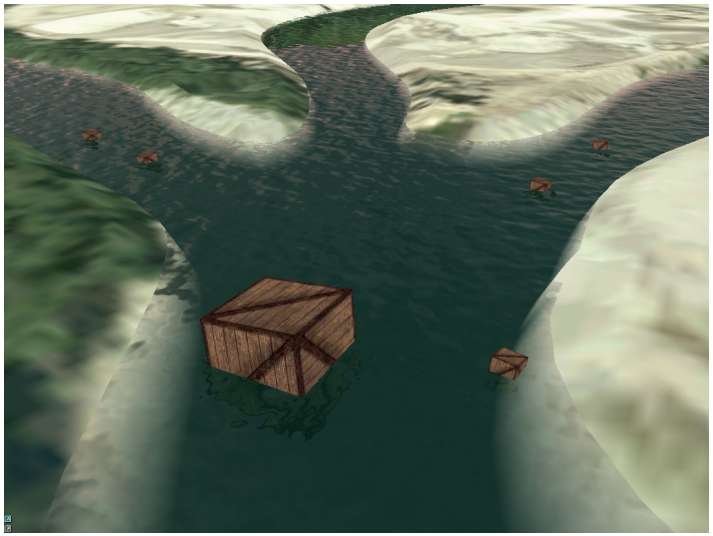
(b) Middle view.



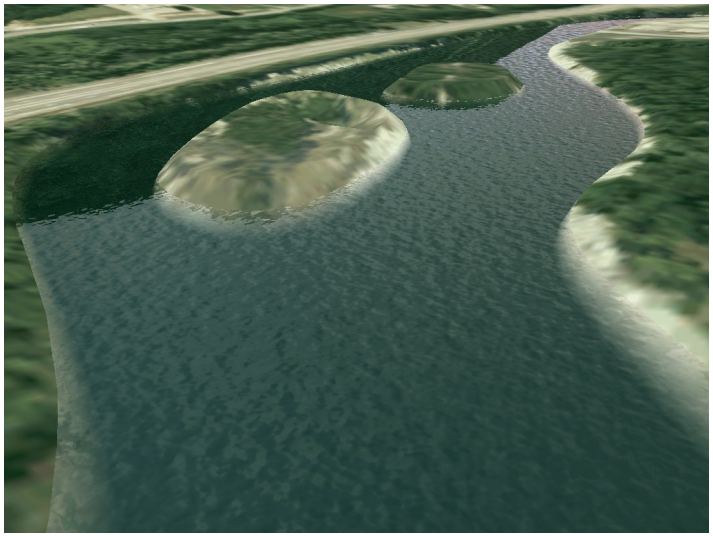
(c) Close view.

Figure 5.5: Particles maintain the Poisson-disk pattern in screen space. Here, the red are particles inside rivers, the black are particles outside rivers, and the yellow are particles outside rivers but near river boundaries.

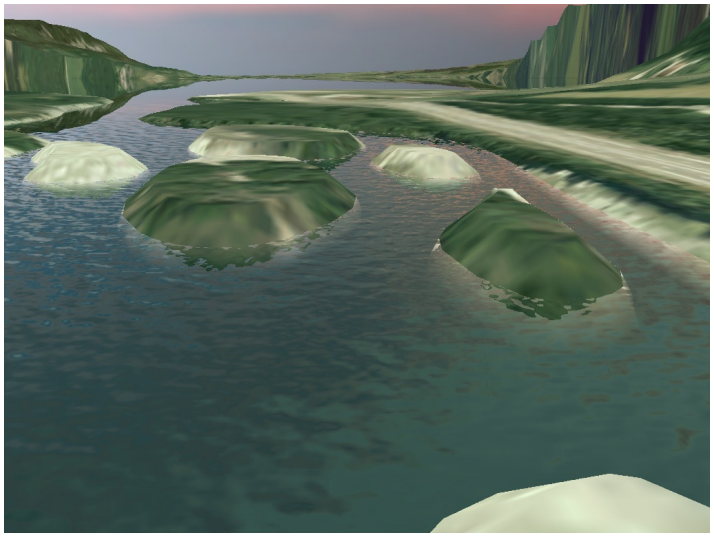
(a) Far view.



(b) Trifurcating junction with floating objects.



(c) River with islands.



(d) River with islands.

Figure 5.6: Screenshots of our river animation.

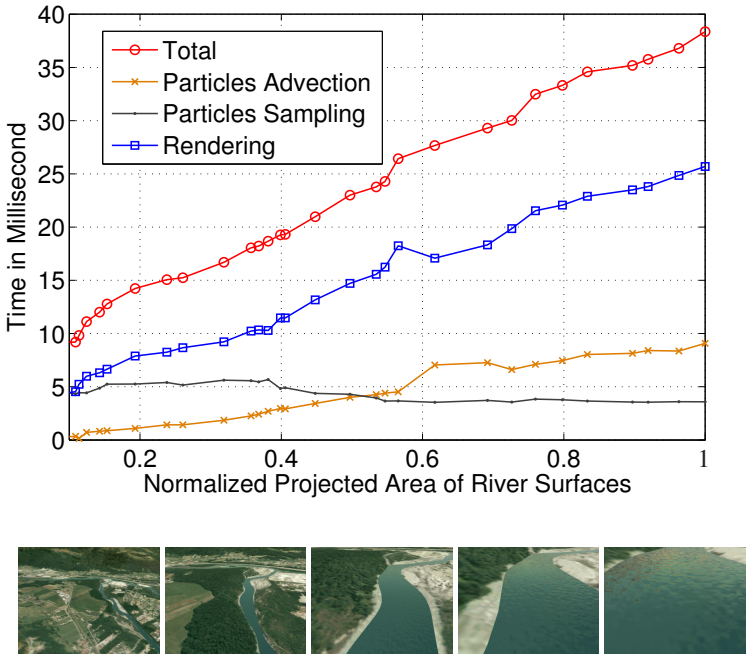


Figure 5.7: *Top:* The running time of our method in each frame. Note that the particles advection part includes the time cost for calculating the river velocity on-the-fly (Chap. 3). *Bottom:* From left to right, typical views with increasing projected areas of river surfaces.

5.6 Summary

In this chapter, we have proposed a scalable texturing scheme for representing advected details on flow surfaces. The phenomenon of wave advection has been achieved by advecting wave sprites with the flow. In addition, we proposed an adaptive dynamic sampling scheme to make the density of sprites adaptive to the view distance thus ensuring a high performance. The sampling scheme also ensures well-separated sprites and thus avoids biasing the texture property of the reconstructed surface as much as possible. With our texturing method, the visual appearance of a river surface is easily controllable by replacing or tuning texture examples.

One limitation of the present method is that it neglects the rotation and deformation of the sprites. This simplification is acceptable for the smooth flow through river channels, but it will leads visible artifacts in regions having high velocity variation. We will address this issue in the next chapter, where we present a Lagrangian texture advection

method.

5.7 French Chapter Abstract

Dans ce chapitre, nous avons proposé un nouveau schéma de texturage pour représenter des détails advectés à la surface des rivières. L'advection des vagues de surface a été obtenue en advectant des sprites de texture le long du flot. Nous avons également présenté une schéma d'échantillonnage permettant d'adapter le nombre de sprites en fonction de la distance au point de vue, ce qui permet d'obtenir de bonnes performances. Le schéma d'échantillonnage permet également d'avoir des sprites bien séparés, et donc de ne pas dégrader la surface reconstruite. De plus, l'apparence visuelle de la rivière peut être facilement modifiée en remplaçant le motif de la texture.

Cependant, cette méthode présente certaines limitations au niveau de la qualité. Nous avons en effet jusqu'à maintenant ignoré la rotation et la déformation des sprites. Cette simplification fonctionne efficacement pour des rivières dont le courant est régulier, mais produira des artefacts dans des cas plus généraux. Nous détaillons ce problème dans le prochain chapitre où nous présentons une méthode d'advection de texture Lagrangienne.

Chapter 6

Micro-Scale: Lagrangian Texture Advection

The common goal of the preceding chapter and this chapter is to find an efficient way to texture fluid surfaces with two contrary requirements: (1) the texture contents follow the underlying flow, and (2) the texture preserves its statistic properties. The results of the preceding chapter show that the technique of wave sprites, an adaptive sprite-based texturing method, scales well to large scenes in real-time applications. It can also meet the requirements of conserving the texture property. However, this method neglects local texture deformation, and thus fails to convey sub-sprite flow motion, which sometimes violates the first requirement mentioned above. The problem is particularly serious when we want to handle arbitrary flow fields which may have high velocity variation. In this chapter, we still employ the Lagrangian formalism as we did in the preceding chapter, but make a further step by accounting for the local texture deformation.

6.1 Motivation

In this section, we motivate our method by examining the quality issue of existing Eulerian texture advection methods [MB95, Ney03]. This kind of method maps a texture on a grid and advect the mapping coordinates with the given flow. Advecting texture coordinates will lead to texture stretching which continuously increases up to totally destroying the texture appearance. Therefore, an important step is to correct the texture distortion while enforcing temporal continuity. An ad hoc method is to periodically

reinitialize the texture mapping after some delay, called *latency*, and blend the two dephased textures. However, choosing a reasonable latency is not trivial. The latency used for controlling texture regeneration allows one to trade texture distortion for the quality of motion illusion. A higher latency permits larger texture distortion and thus the resulting texture sequences has a better consistence with the flow but a worse preservation of texture spectrum. A lower latency leads to a reverse result. To make balance between flow consistence and spectrum conservation, an optimal latency should vary with the location in a flow field, depending on the flow velocity and its gradient. However, only one global latency is used in [MB95]. As an improvement to this method, Neyret [Ney03] uses the latency that is adaptive to local flow conditions. His method advects several layers each of which is made by three textures. Each layer uses a different latency value. The local adaptation of latency is achieved by interpolating between the two closest layers according to a texture distortion metric in the spirit of MIP-mapping. This method works better than [MB95] but still suffers from the following drawbacks. First, the latency of the various layers is arbitrary. Quality will be broken for flows slower or faster than a given range. In particular, flows with parts at rest always cause problems because arbitrary long latency should be used at slow places. Second, the adaptation of latency is achieved by a simple interpolation between layers, which is not accurate. Finally, blending all these textures is costly and tends to smooth the resulting spectrum.

Besides the above problems related to the latency, the texture advection methods proposed in [MB95, Ney03] also share another drawback. In these methods, advection and blending of textures is done in an entire domain. This scheme is not optimal for phenomena which are visible only at some sparse places (*e.g.,* fire, smoke and clouds), especially in 3D cases.

We believe that most of the above drawbacks originate from the underlying Eulerian formalism: the requirement of adapting and regenerating locally does not match well with having a continuous mapping over the whole domain. Therefore, we propose a Lagrangian based method. The basic idea is to disperse texture distortion and regeneration over both space and time by the Lagrangian formalism while using local deformable patches to ensure continuity.

6.2 Our Method

Our method relies on a collection of particles each of which is associated with a *deformable* textured grid, called a *patch*. There is a one-to-one relationship between particles and patches. These patches will later be used to construct an advected texture by blending all of them. We develop our algorithm by considering the requirements of texture advection as follows.

First, texture features should move with a given flow. This can be easily done by advecting all particles and grid nodes of patches with the flow.

Second, the texture should conserve its statistics property which includes both static and dynamic aspects. On the static aspect, we let each patch initially map to a random portion of a given reference texture. Previous texture synthesis work [XGS00] has demonstrated that combining random patches of a texture example can yield a new texture which preserves the statistics of the texture example, especially for stochastic textures. In addition, we need to separate the patches well but still cover the texture domain fully. This is done by maintaining particles with a Poisson-disk distribution. On the dynamic aspect, patches undergo distortion because grid nodes are advected with the flow, which will gradually destroy the texture statistics. To avoid this, we kill a particle when the distortion of its patch exceeds some threshold.

Finally, the advected texture should preserve temporal continuity. To do so, when a particle is killed, we do not immediately discard it. Instead, we keep this particle in a *decay* list where each particle and its patch continue to be advected while fading out until the patch is totally invisible. The particles in the decay list are not considered in the Poisson-distribution anymore, but their associated patches are still used for reconstructing the global texture.

Putting the above considerations together, the full algorithm (Figure 6.1) in each time step can be summarized as follows:

- Advect particles and the grid nodes of each patch with the underlying flow.
- Kill a particle when the distortion of its associated patch exceeds some threshold (Sections 6.2.2 and 6.2.3).
- Maintain a Poisson-disk distribution of particles by killing or inserting particles (Section 6.2.1).
- Compute a weight map including temporal weights and spatial weights for each patch

(Section 6.2.4).

- Reconstruct a global texture by blending all the patches (Section 6.2.6).

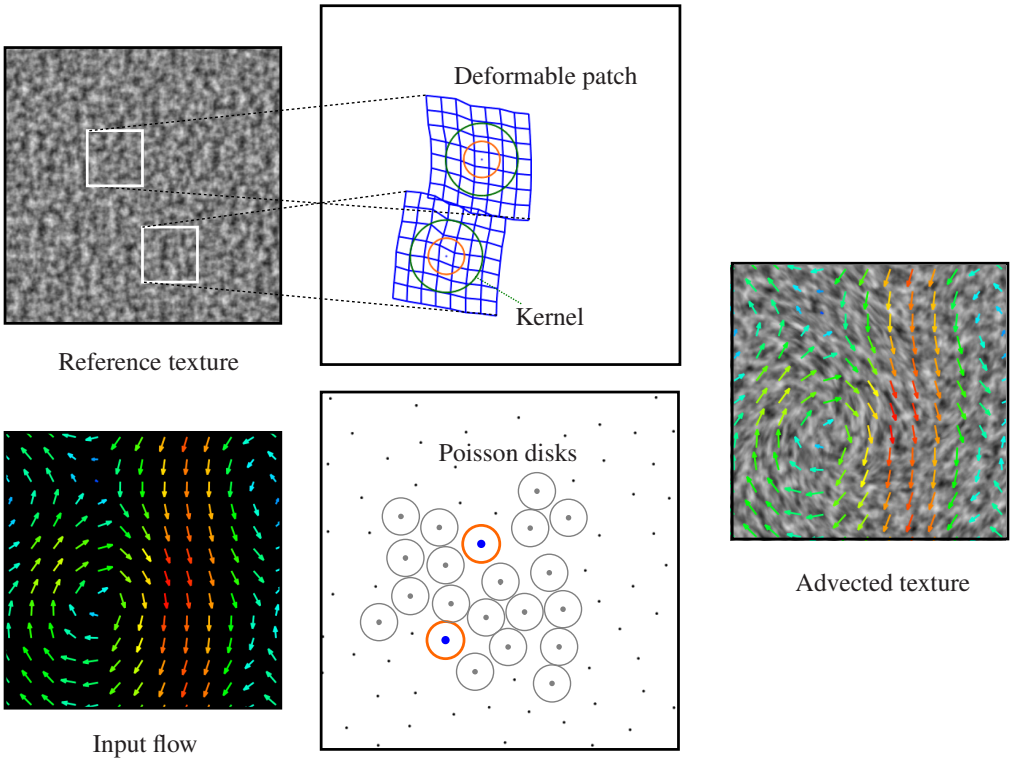


Figure 6.1: Overview of our algorithm. *Left*: Input a flow field and a reference texture. *Middle bottom*: Particles advected with the input flow while keeping the Poisson disk distribution. *Middle top*: A deformable patch is associated to each particle and also advected with the input flow. Each patch carries a random portion of the reference texture. *Right:* Blending all the patches yields an advected texture.

6.2.1 Dynamic Distribution of Particles

As we stated above, we have several requirements for placing patches:

- Patches should follow the underlying flow to convey the flow motion;
- Superimposing all patches should fully cover the texture space, to ensure spatial continuity;
- The patches should overlap as little as possible to avoid biasing the texture spectrum.

In fact, this problem is similar to the one we considered in Section 5.2. The only difference is that here the distribution is in texture space, but there the distribution is in screen-space. Thus the main part of the adaptive sampling algorithm proposed there can be re-used to address the current problem.

The idea is to maintain a set of particles that are advected with the underlying flow while keeping a Poisson-disk distribution by insertion or deletion. The well-separated particles resulting from the Poisson-disk distribution make the patches overlap as little as possible. As we stated in Section 5.2.2, given a Poisson-disk point set with radius r, the union of all disks of radius $2r$ centered at the Poisson-disk points can fully cover the sampling domain. This property helps satisfy the requirement of full-coverage in our problem, which will be detailed in the following section. In the following discussion, we call these disks of radius $2r$ *particle kernels*.

Our method for managing particles has two main stages. In the first stage, after updating particle positions by advection, we kill particles if their minimal distances to others are less than $2(1-\alpha)r$, where r is the Poisson-disk radius. We relax the radius by a factor α so that we kill as few as possible too young particles, because young particles tend to associate with patches with small distortion. α should not be too large in order to ensure that sprites separate well as a Poisson-disk distribution. In our implementation, we use $\alpha = 0.25$. Second, once we have achieved a set of particles respecting the relaxed minimal distance criterion, we use the *boundary sampling algorithm* [DH06] to insert new particles for maintaining a Poisson-disk pattern with radius of r. Here, we need a trivial adaption of the boundary sampling algorithm. The first step of the original algorithm is to generate a random point. Instead, we start the algorithm from an existing sampling points. It is reasonable to do this adaptation because the boundary sampling algorithm is essentially incremental.

6.2.2 Set-up of Patches

The dimension of patches should be set correctly to ensure the full coverage of the texture space. Thanks to the property of the Poisson-disk distribution we mentioned in the preceding section, we can guarantee full coverage by ensuring that each patch fully covers its corresponding particle kernel. Therefore, at particle creation we start with a square patch of side size $4(1+\beta)r$ centered at the particle (Figure 6.2).

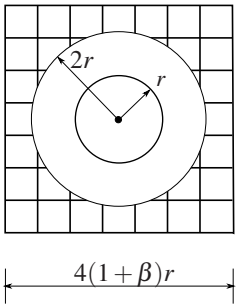


Figure 6.2: An initial patch is square and contains a regular grid (8×8 nodes in our implementation). The side size of the patch is $4(1+\beta)r$, where r is the Poisson-disk radius and $\beta > 0$, so that the patch can totally cover the particle kernel of radius $2r$.

We let $\beta > 0$ to ensure that a patch within a tolerable deformation (defined in the next section) still can fully cover its particle kernel. Still, a patch may finally fail to fully cover its particle kernel if one of the patch borders intersects with the kernel after an excessive distortion. In this case, we kill the corresponding particle. Though β corresponds to the maximum tolerable texture distortion, it is not used as a parameter for controlling the texture distortion in our method. In fact, we rely on a more precise method described in the next section for controlling the texture distortion. In this sense, β can be an arbitrary large value.

On the other hand, we should not use a too large β value for the performance reasons. The larger β, the more nodes needed to be advected. Therefore, we used a moderate value $\beta = 0.3$ in our implementation.

The patch is larger than the particle kernel, but only the portion of a patch that intersects with the corresponding particle kernel finally contributes to the reconstructed global texture. Thus we can achieve a minimal overlapping during the reconstruction.

6.2.3 The Distortion Metric

To limit the maximum distortion of an advected texture, we kill a particle when the distortion metric of its associated patch exceeds some threshold. In this section, we describe the way we use for measuring the distortion of a patch. Our basic idea is to triangulate a patch, and calculate the singular values of the Jacobian of the affine transformation between initial triangles and advected triangles (Figure 6.3). We derive our distortion metric in the follows.

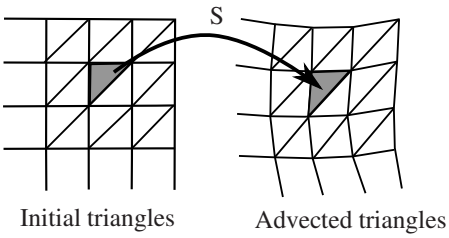


Figure 6.3: The distortion of a patch is measured by calculating the singular values of the Jacobian of the affine transformation between initial triangles and advected triangles.

Given an initial triangle $G = \triangle q_1 q_2 q_3$, and its corresponding advected triangle $\triangle p_1 p_2 p_3$, where $p_i = (x_i, y_i)$, the unique affine mapping $S(p) = q$ is

$$S(p) = (\langle p, p_2, p_3\rangle q_1 + \langle p, p_3, p_1\rangle q_2 + \langle p, p_1, p_2\rangle q_3)/\langle p_1, p_2, p_3\rangle, \tag{6.1}$$

and the partial derivates of S are

$$\begin{aligned} S_x &= \frac{\partial S}{\partial x} = \frac{q_1(y_2 - y_3) + q_2(y_3 - y_1) + q_3(y_1 - y_2)}{2\langle p_1, p_2, p_3\rangle}, \\ S_y &= \frac{\partial S}{\partial y} = \frac{q_1(x_2 - x_3) + q_2(x_3 - x_1) + q_3(x_1 - x_2)}{2\langle p_1, p_2, p_3\rangle}, \end{aligned} \tag{6.2}$$

where $\langle A, B, C\rangle$ denotes the area of triangle ABC. The singular values of the 2×2 Jacobian matrix $S = [S_x, S_y]$ are

$$\begin{aligned} \gamma_{min} &= \sqrt{1/2((a+c) - \sqrt{(a-c)^2 + 4b^2})}, \\ \gamma_{max} &= \sqrt{1/2((a+c) + \sqrt{(a-c)^2 + 4b^2})}, \end{aligned} \tag{6.3}$$

where $a = S_x \cdot S_x$, $b = S_x \cdot S_y$, and $c = S_y \cdot S_y$. They give the minimum and maximum length that a unit vector can get after the transformation due to advection. As in [SCOGL02], we use the following expression for measuring the distortion of the triangle G:

$$d(G) = max(\gamma_{max}, 1/\gamma_{min}). \tag{6.4}$$

Note that $d(G) \geq 1$ and the equality means no distortion.

Once we have defined the distortion metric of a single triangle, we are able to define

a distortion metric over a whole patch. In our case, we want to measure the worst case. Therefore, let $P = \{G_i\}$ be the triangles in a patch which intersect with its particle kernel, we use

$$D = \max_{G_i \in P} d(G_i) \tag{6.5}$$

as the distortion metric over the whole patch. To limit the maximum distortion of an advected texture, we kill a particle when $D > \delta_{kill}$. We use $\delta_{kill} = 2.5$ in our implementation.

Similarly, we can define the distortion metric at each grid node. Given a grid node $\mathbf{x}$, let $M = \{G_i\}$ be the adjacent triangles of the node. We define the distortion metric at the node as

$$\delta(\mathbf{x}) = \max_{G_i \in M} d(G_i). \tag{6.6}$$

This metric will later be used for determining the spatial weights of a patch discussed in the following section.

6.2.4 Computing Weight Maps

Enforcing temporal and spatial continuity is important in texture advection. To achieve this goal, we compute a weight map for each patch where weight values vary in time and space. Then, the weight maps are used to blend patches in the final texture reconstruction stage.

Temporal Weights

In order to avoid popping artifacts, we consider fade-in and fade-out associated with the distortion and life time of patches.

We fade in a newly created patch with a weight function F_{in} which increases linearly from 0 to 1 during a period of T_{in}. In our implementation, we used $T_{in} = 2$ seconds. Comparatively, the fade-out scheme is more complicated.

Fade-out weight is associated with the distortion of patches. For better continuity, this weight is counted from the birth time rather than the killing time. We fade out a patch continuously according to a function $F_{disto}(\mathbf{x})$ based on the distortion metric $\delta(\mathbf{x})$ at nodes, so that a node area becomes invisible when $\delta > \delta_{max}$, with $\delta_{max} > \delta_{kill}$. A

particle is removed from the decay list when all its patch nodes are invisible. We define

$$F_{disto} = 1 - \frac{\delta - 1}{\delta_{max} - 1}. \tag{6.7}$$

The threshold δ_{max}, together with δ_{kill} mentioned in the preceding section, is the main parameter in our model to balance between preserving texture spectrum and conveying flow motion. In our examples, we used $\delta_{max} = 5$ seconds.

The fade-out associated with distortion occasionally fails to remove a particle from the decay list in a limit period when the distortion of its patch does not gradually accumulate. On the other hand, we hope to limit the patches in the decay list to get as little overlapping as possible. Therefore, we introduce another fade-out weight F_{out} which is associated with the elapsed time accounted from the killing time. F_{out} decreases linearly from 1 to 0 during a period of T_{out} which is the maximum survival time of a particle in the decay list. Since T_{out} is mainly a security set but not the main quality control parameter in our model, it is not a problem to use a large value. We used 5 seconds in our implementation.

Spatial Weights

Recall that only the portion of the patch that intersects with the particle kernel contributes to the reconstruction. To ensure spatial continuity, we consider a kernel K_{partic} that fades from 1 at the center to 0 at the border of the particle kernel. In our implementation we use a linear kernel intensity.

For a particle in the decay list, its associated patch may stretch up to not totally covering the particle kernel. To ensure continuity at patch borders, we introduce a patch kernel K_{patch} fading to the borders. In our implementation we set it to 1 at the inner nodes and we bilinearly interpolate to 0 on the border cells.

Putting Together Weights

The total weight at a given location is then defined as the product of all the fading coefficients and spatial kernels:

$$w(\mathbf{x},t) = F_{in}(t)F_{out}(t)F_{disto}(\delta(\mathbf{x}))K_{partic}(\mathbf{x})K_{patch}(\mathbf{x}). \tag{6.8}$$

During the reconstruction, it is renormalized according to the sum of the weights of overlapping patches. This provides a resulting animated texture continuous in space and time.

6.2.5 Handling Solid Boundaries

As a Lagrangian approach, our method is intrinsically more flexible than Eulerian approaches for handling the flow confined by complex boundaries. Still, we need a specific treatment for patches overlapping solid boundaries. In these patches, grid nodes outside the flow domain fail to achieve the flow velocity. For no-slip boundaries, we can simply set zero velocity for the outside nodes. However, it is not reasonable to do this for slip boundaries. In the physical sense, a slip boundary condition can also be interpreted as a symmetry boundary. Thus we should extrapolate the outside velocities according to the velocities in nearby inside regions.

We build pyramid grids to estimate the velocities of the outside nodes in a patch (Figure 6.4). The idea is similar to the push-pull algorithm proposed in [GGSC96]. Given a patch overlapping boundaries with $2^n \times 2^n$ nodes, we set velocities to the inside nodes accordingly and mark the outside nodes as unknown. We build a coarser grid with $2^{(n-1)} \times 2^{(n-1)}$ nodes by averaging the known values of every 2×2 nodes in the current grid. Note that four nodes marked as unknown will still lead to an unknown node in the coarser level. We repeat building coarser grids until we reach a grid with no unknown node. If only the initial grid is not full of unknown nodes, we can always get such a grid without unknown nodes. Then, in reverse, we go down in the hierarchy from the coarsest grid, filling unknown nodes in a grid with the values of the corresponding nodes in the neighboring coarser grid. By using the above method, we can estimate velocities for the unknown nodes according to known values as local as possible.

6.2.6 Texture Reconstruction

So far we have had a set of patches that are well separated while fully covering the texture space. The advected texture at $\mathbf{x}$ can be computed by blending the textures $T_i(\mathbf{x})$

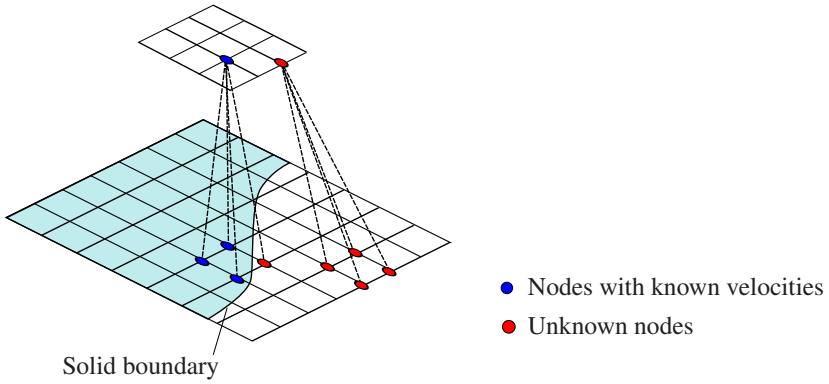


Figure 6.4: Pyramid grids for extrapolating the velocities of the outside nodes in a patch intersecting the solid boundary. We repeatedly build coarser grids by averaging known velocities of every 2×2 nodes until we reach a grid with no unknown node. Then, we go down in the hierarchy from the coarsest grid, filling unknown nodes in a grid with the values from the neighboring coarser grid.

of the patches covering this location with:

$$T(\mathbf{x}) = \frac{\sum w_i T_i}{\sum w_i}, \tag{6.9}$$

where T_i maps to a random portion of the reference texture, and w_i is the weight factor defined in Equation 6.8.

We propose two methods to blend the deformable textured patches:

- We can directly draw all patches, accumulating the $w_i T_i$ and w_i in separate channels. A second pass combines these channels to evaluate Equation 6.9.

- Alternatively, we can use an indirection structure as we did in Section 5.3. In a first pass we draw for each patch its local parameterization (u,v) and its weights $w(\mathbf{x},t)$ in an *FFD image*. We divide the fluid domain in tiles and compute for each tile the list of patches that intersect it. In a second pass a pixel shader uses these indirection maps to find all the patches that cover a given pixel, and get the (u,v) and $w(\mathbf{x},t)$ of each patch at this pixel which allows us to evaluate Equation 6.9.

The second method is more complex but is better adapted to the case of a sparse fluid in a large domain, such as a river. It computes only the visible pixels, and its memory usage is proportional to the number of grids, as opposed to the size of the

domain. On the contrary, the first method requires textures covering the whole domain at the maximum resolution, using floats (since the accumulated values are not bounded). It thus computes all pixels, even invisible ones.

In both methods the input texture can contain either final colors or input parameters for a complex procedural shader (clouds, fires, etc). In the second case we blend the parameters *before* applying the procedural shader to avoid ghosting effects [Ney03]. This also decreases the computation cost as the procedural shader is called only once per pixel instead of once per grid. Note that the procedural shader can use displacement mapping to create a 3D surface from the 2D advected texture (Figure 6.9).

6.3 Experimental Validation

To validate our method, we compared it with an Eulerian texture advection method. We first introduce the experimental workbench in Section 6.3.1, and then describe our observations in Section 6.3.2. Finally, we give the performance of our method in Section 6.3.3.

6.3.1 Workbench

Reference Method

As existing systems are too complex in terms of free parameters, we created a mixed method for meaningful comparison. This Eulerian method regenerates textures with global latency and blends three dephased textures. Thus the method represents an improvement of [MB95] and a portion of [Ney03]. As the formal method blends only two textures, its spectrum oscillates in time. Blending three dephased textures can overcome this problem, which corresponds to a single layer of the latter method. Since the latter method uses three such layers and local adaptive latency, it will work better than our reference method. However, using this method complicates the comparison. Most importantly, as we stated earlier, this method cannot solve the problem of latency completely. The valid range of flow is still limited. Therefore, we believe that using a single layer and tuning latency manually provides the best understanding of the intrinsic problem of the Eulerian approach.

Measurements

Texture advection has to meet two different requirements. On one hand, the texture sequence should be consistent with the input flow. On the other hand, the texture sequence should maintain the texture spectrum. Accordingly, our measurements include two aspects. To evaluate the flow consistency, we computed an optical flow from the texture sequence and compared it with the input flow. To evaluate the preservation of texture spectrum, we computed the Fourier transform of the advected texture and compared it with the spectrum of the reference texture.

Input Flow

We used various flow-fields as input (Figure 6.5):

- Simple analytic flows: constant flow, shear flow, rotating flow and source flow. These illustrate the algorithms' behavior on basic critical situations and these are the most meaningful in terms of spectrum.
- A free flow: user interacting with a 2D Navier-Stoke solver [Sta99]. This is a more typical heterogeneous flow.
- A flow with boundaries: using slip or no-slip conditions. It is challenging since the texture must obey extra constraints and the flow includes bifurcations.

Reference Texture

We used a 3 octave Perlin noise as the reference texture (Figure 6.1, *left top*). The smallest wavelength corresponds to 8 pixels and the largest wavelength is about $1/6^{th}$ of our particle kernel size.

6.3.2 Results

Please see the accompanying video for the comparison results [1]. The results show that our method can ensure both spectrum conservation and flow consistence but the Eulerain texture advection often miss one of the requirements. We give some detailed remarks in the following paragraphs.

[1] `http://evasion.inrialpes.fr/Membres/Qizhi.Yu/phd/`

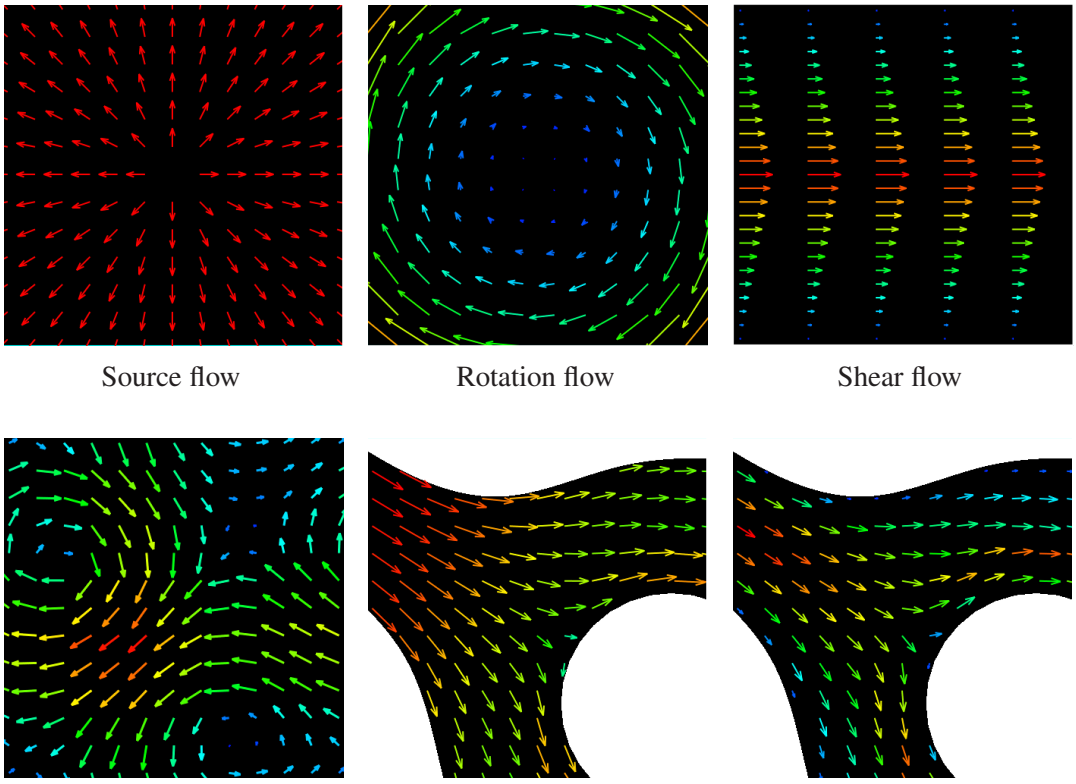


Figure 6.5: Various input flow used for evaluating our texture advection method.

For a constant flow, the FFT still shows slight oscillations for the reference Eulerian method, which does not exist at all in ours. Note also that the strip artifact in Perlin FFT [KKS08] is smoothed in our model.

For the shear flow examples, the velocity ramps from 0 at the boundary to a high value in the middle. This is a very demanding test of distortion and continuity, as well as the reconstruction of a continuous field from our discrete patches. For the reference method, it is easy to get either vanishing of flow motion or chewing-gum like overstretching (Figure 6.6). As seen in the accompanying video, our method preserves nicely both properties nicely. Moreover, our reconstructed texture is very homogeneous and our reconstructed motion is very accurate: even very slow velocity can be reproduced. The full [Ney03] method should be able to alleviate the flaws (within a given range), but it will introduce heterogeneous over-blurred bands corresponding to transition areas between layers.

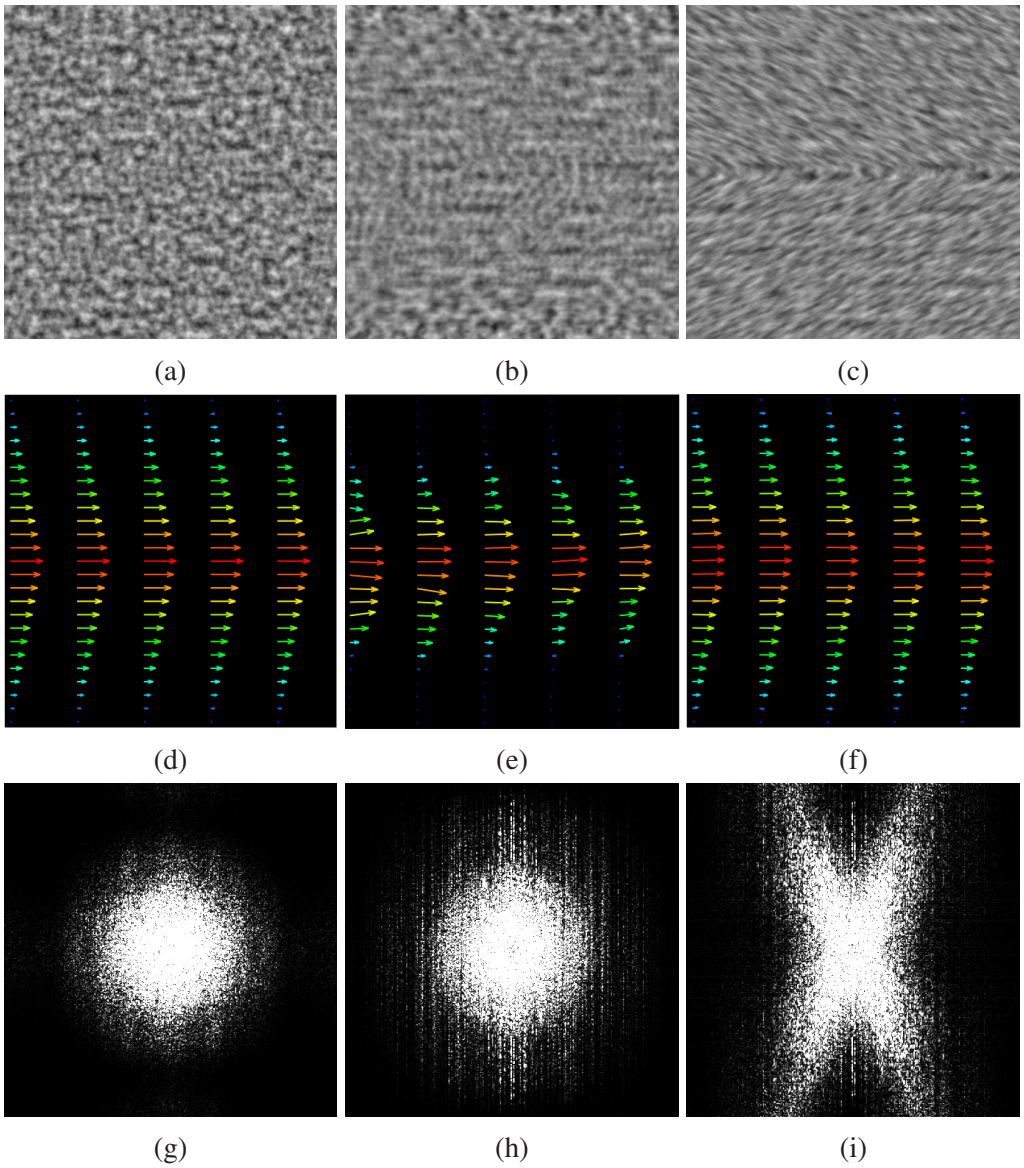


Figure 6.6: Problems of Eulerian texture advection. *Left*: input texture (a), input flow (d) and Fourier spectrum of input texture (g). *Middle*: with a short regeneration latency the advected texture (b) conveys an incorrect optical flow (e), but the Fourier spectrum is almost preserved (h). *Right*: with a long latency the texture is too stretched (c) and the Fourier spectrum is distorted (i), but the optical flow matches the input velocity field (f).

For the free flow, note that the velocity varies at a scale much smaller than the size of particle kernels. Still, our method easily accommodates the heterogeneity of the flow field (Figure 6.7), which the Eulerian method cannot do either. The same is true for the flow with boundaries and obstacle, despite the bifurcation and high distortions of the flow. Note that for the source flow, our method does not need to extrapolate (u,v) values at the source location.

6.3.3 Performances

We did our tests on an Athlon AMD 3000+ at 1.8GHz with an Nvidia GeForce 8800GTS. We measured performance on the free flow example. Image resolution is 1024×1024. The fluid grid is 64×64. We used $d = 100$ pixels which gave 150 particles on average. Patches have 8×8 nodes, so one patch cell is about 32×32 pixels. FFD images are 64×64 pixels. For reference texture we used a 1024×1024 image showing 3 octaves of Perlin noise (pseudo-wavelengths from 8 to 32 pixels).

The timing was 10 *ms* per frame (100 FPS). It decomposes into 1.6 *ms* for the stable fluid simulation and 8.1 *ms* for texture advection. In this last, advection of nodes represents 5.1 *ms* and final reconstruction 2.5 *ms*. Poisson disk treatment cost is negligible. In a real application the cost of the final color shader should be added.

In terms of memory, the FFD images $(u,v,w(x,t))$ cost $64 \times 64 \times N \times 3$ bytes for N particles, and the patches cost $8 \times 8 \times N \times 2$ floats. This makes about 2 MB in our example.

6.4 Discussion

Our new texture advection method is a balance that associates Lagrangian and local grids for the best. The Lagrangian formalism brings the decorrelation of texture mapping and regeneration events. Using local grids ensures continuous texture animation and provides an accurate way to measure texture distortion. Unlike the existing Eulerian methods, we do not rely on arbitrary latency values.

In addition, the Lagrangian formalism has intrinsic advantages in terms of performance. It allows us to easily avoid the computation of the empty region in a flow domain, which was not possible with [Ney03]. Furthermore, combining our method with

the adaptive sampling technique introduced in the preceding chapter, we will arrive at a scalable texture advection method that is suitable for very large and even unbounded scenes.

By connecting to various shaders, advected noise textures can generate abundant animated details. We have applied advected Perlin noise (or flow noise [PN01]) in several applications: clouds (Figures 6.8 and 6.9), fires (Figure 6.12) and rivers (Figures 6.11 and 6.10). See also the accompanying video.

Though our method mainly targets textures specified by spectrum or procedural parameters, experimental results show that our method also produces reasonable results for a wide variety of image textures (see Figures 6.13, 6.14 and 6.15, and the accompanying video). As expected, large-scale texture heterogeneities or anisotropy yields visible transition, but in practice they are often not annoying to the eye (*e.g.,* foam). Using an animated reference texture would have even better masked the transition. Note that for high-structured textures, our method suffers from ghosting artifacts since the patches simply take random portions from the reference texture. A promising future work is to choose a portion of texture that best matches the neighborhood for each newly created patch. Note that due to our deformable patches (and thus differently to other blended-sprite approaches [BSHK04]), two blended pixels of two blended patches will remain superimposed during advecting without relative sliding.

A concurrent work [NSCL08] constructs small scale turbulent flow velocities by superposing advected *noise particles*. The technique shares some similarity with ours, while a main difference is that they do not account for the deformation of the noise texture. This may be accepted in the case of constructing turbulent velocity fields but not in our case, where we need to avoid the sliding of texture content between overlapping patches.

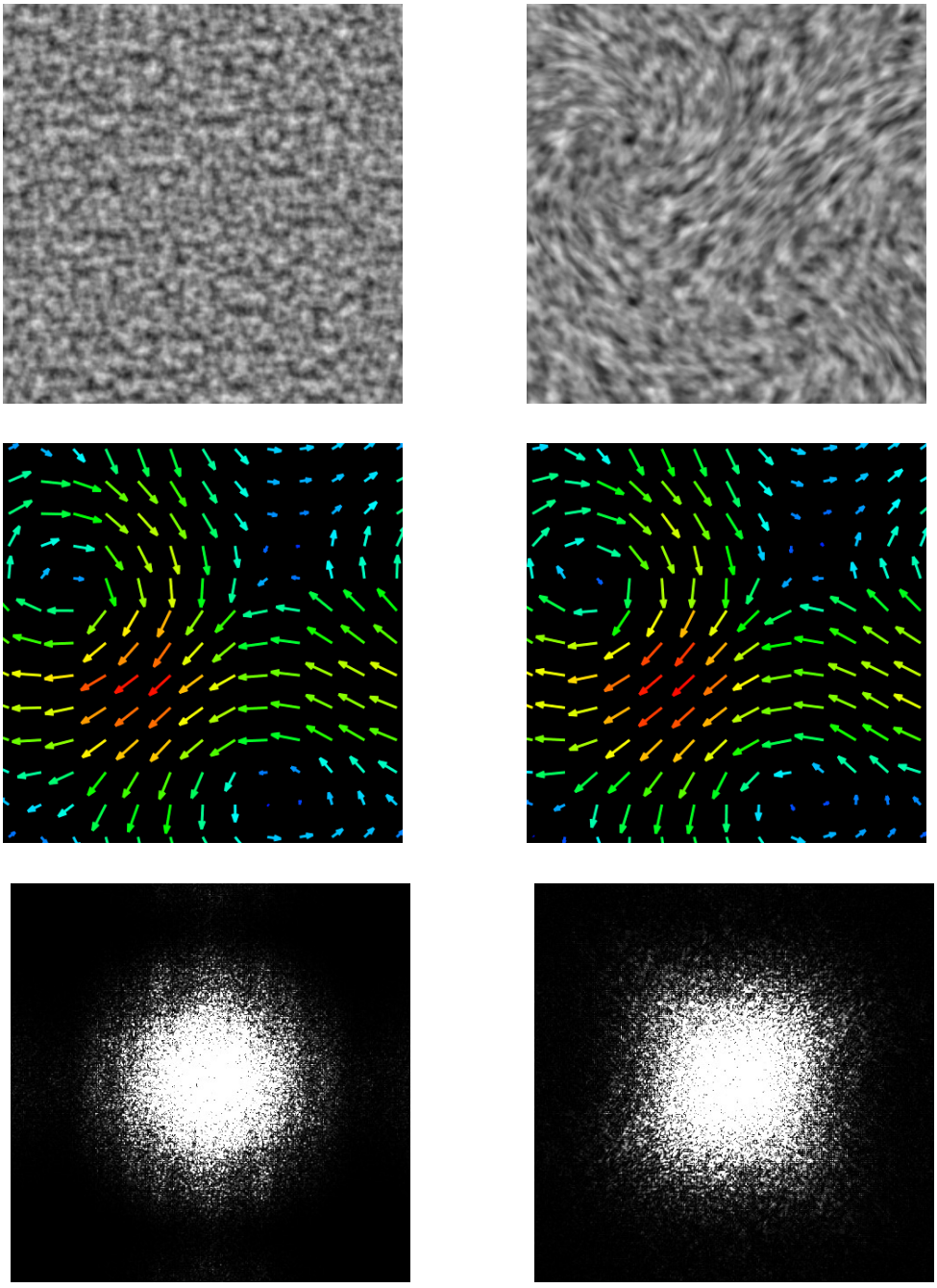
6.5 Summary

In this chapter we have presented a new texture advection method using the combination of particle advection and texture patch blending. The particles follow a birth and death process to maintain uniform density over the domain. Each particle is associated with a deformable textured patch. The patches are advected and distorted according to the underlying flow field. Each patch maps to a random portion of a reference texture. The

key benefit of our method is that texture stretching and correction (via regeneration) are dispersed over both space and time, which avoids the artifacts that the Eulerian method suffers from. Moreover, the method is well-suited for sparse or large domains. In the next chapter, we conclude with a summary of this dissertation work, including limitations and ideas for future work.

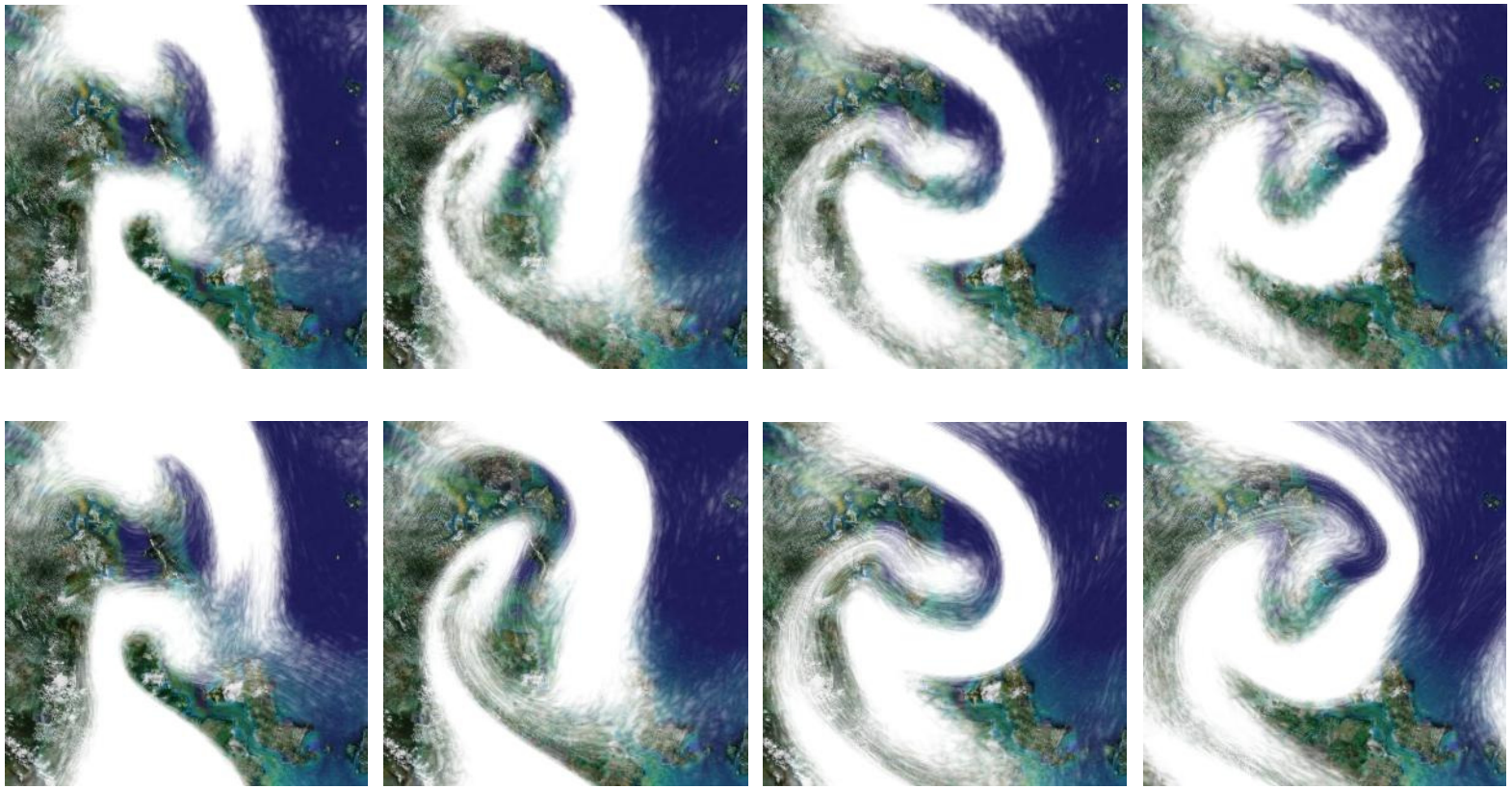
6.6 French Chapter Abstract

Dans ce chapitre, nous avons proposé un nouveau schéma de texturage utilisant des particules advectées déformables. Les particules naissent et meurent afin de garder une densité uniforme sur le domaine. A chaque particule est associé un *patch* texturé déformable. Ces patches sont advectés et déformés en fonction du courant sous jacent. Chaque patch correspond à une portion aléatoire d'une texture de référence. Le principal avantage de notre méthode provient du fait que l'étirement et la correction (par la régénération) de la texture sont répartis dans l'espace et le temps, ce qui permet d'éviter les artefacts présents avec la méthode Eulérienne. De plus, cette méthode convient bien pour les domaines clairsemés ou grands. Dans le chapitre suivant nous présentons une conclusion ainsi qu'un résumé de cette thèse. Nous présentons également les limitations de nos méthodes ainsi que plusieurs pistes pour les travaux futurs.



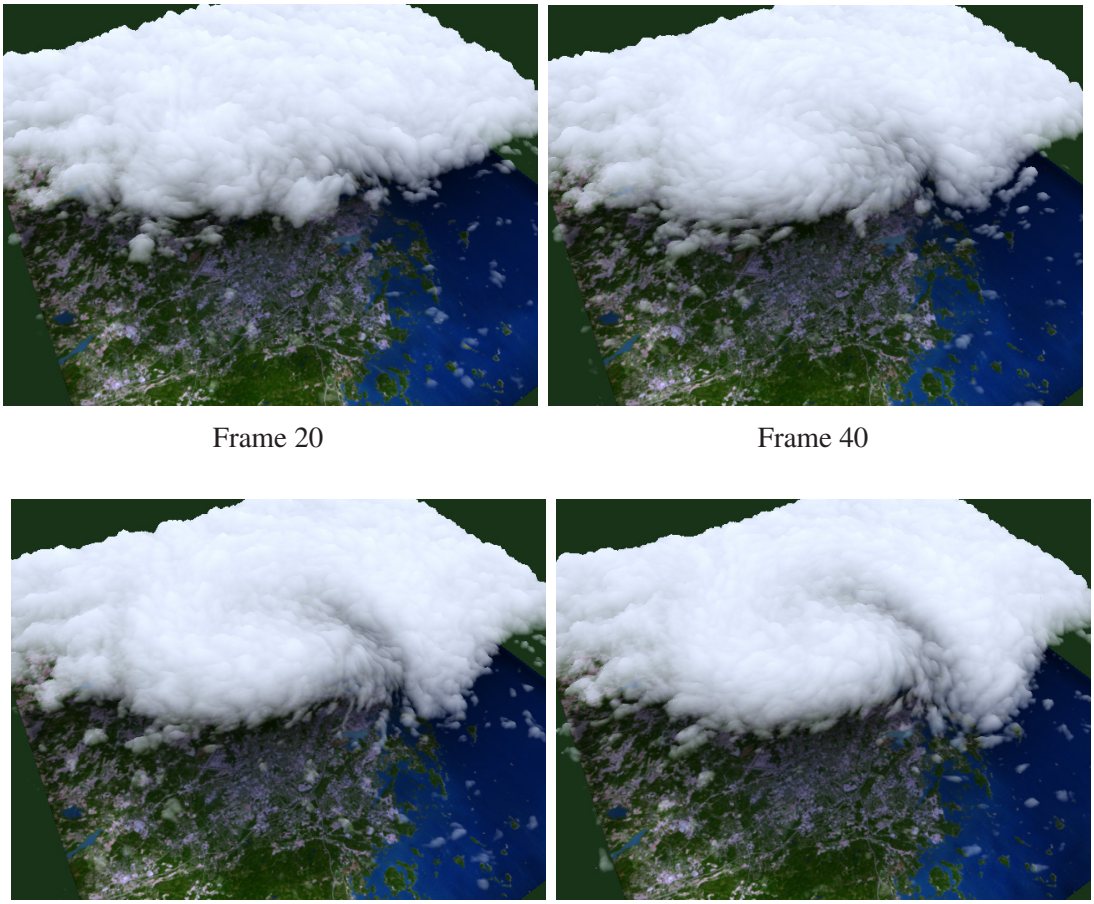
Input Our Results

Figure 6.7: Results of our method. *Left*: input texture , input flow, and Fourier spectrum of input texture. *Right*: our advected texture , its optical flow, and its Fourier spectrum.



Frame 20 Frame 40 Frame 60 Frame 80

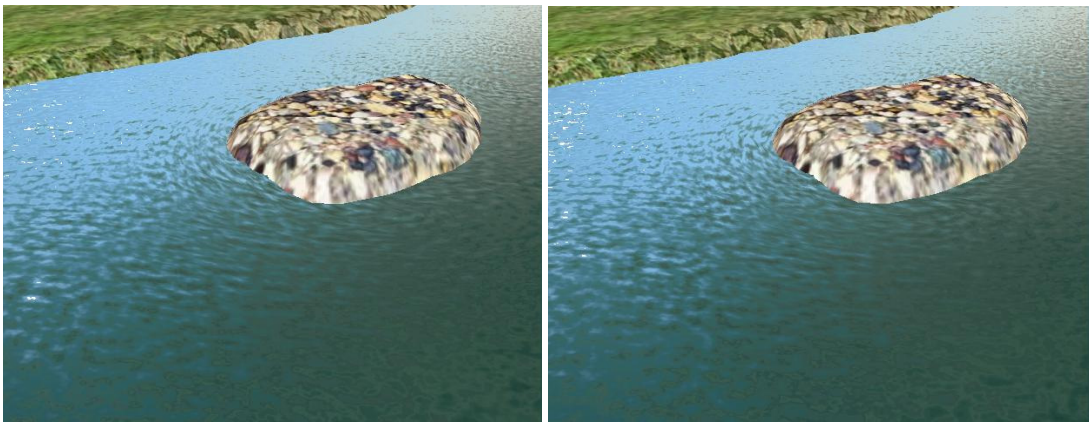
Figure 6.8: Comparison between our method (*top*) and Eulerian texture advection method (*bottom*) in an application where advected Perlin noise are applied to a cloud shader. Here, same reference noise texture and input flow are used in the two examples. It demonstrates that our method can conserve noise spectrum better than the Eulerian texture advection method.



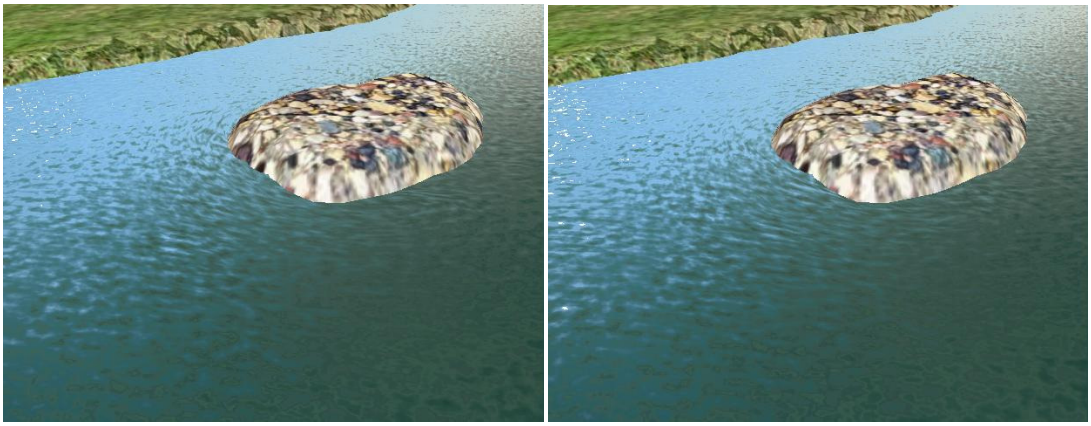
Frame 20 Frame 40

Frame 60 Frame 80

Figure 6.9: Animated clouds with advected details. It is generated in a number of steps. A 2D density field is advected with a low resolution velocity field. Flow noise [PN01] advected with our method is used to modulate the density field. Finally, the enriched density field is used for displacement mapping in a cloud shader.

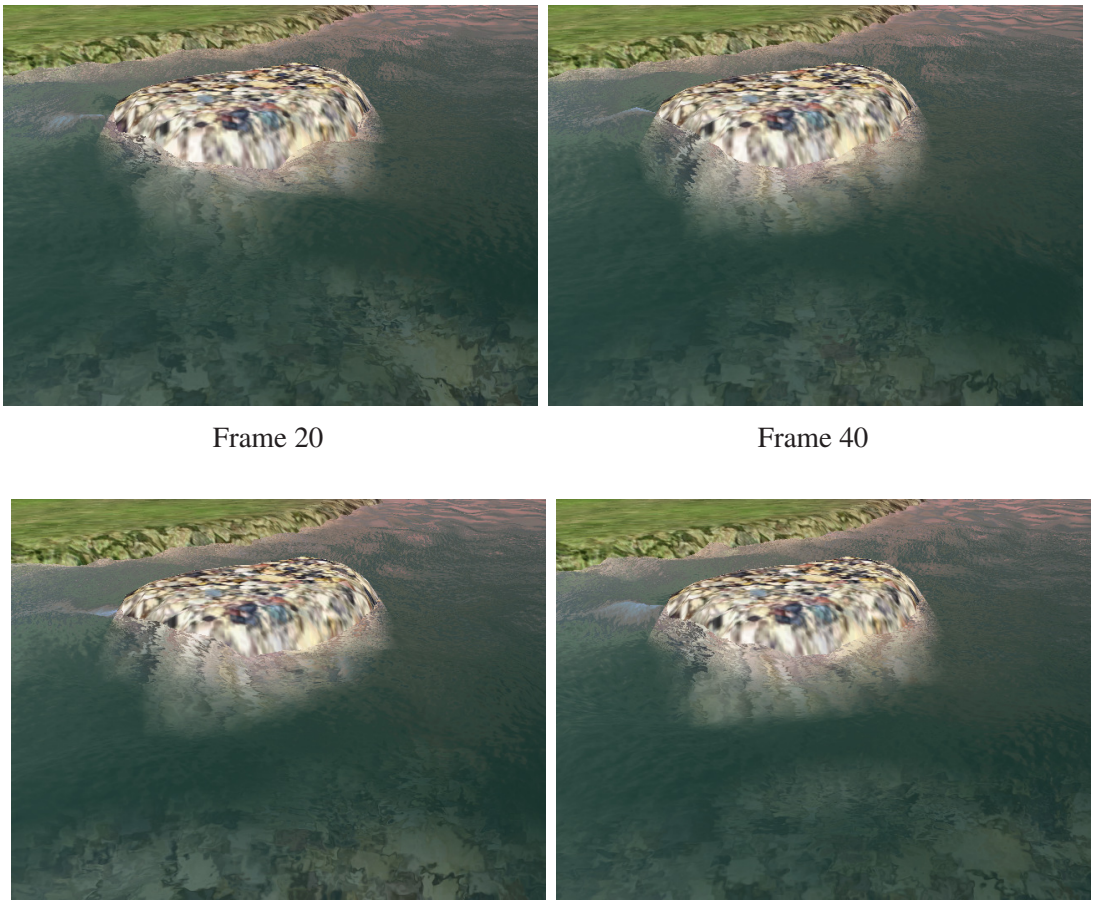


Frame 20 Frame 40



Frame 60 Frame 80

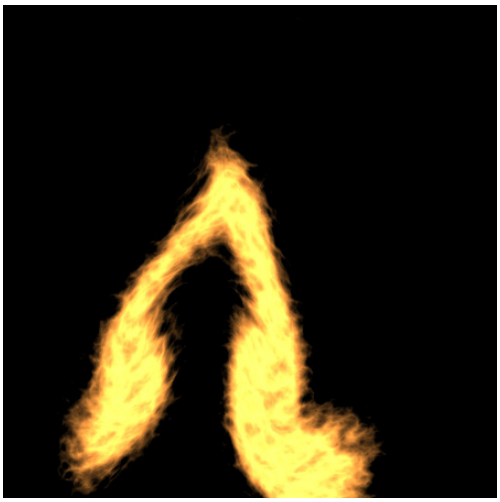
Figure 6.10: Flowing rivers using advected noise for bump mapping. It demonstrates that our texture advection method can conserve noise spectrum well even when the flow has a drastic variation (*e.g.,* the flow region in front of the island). Please see also the accompanying video.



Frame 20 Frame 40

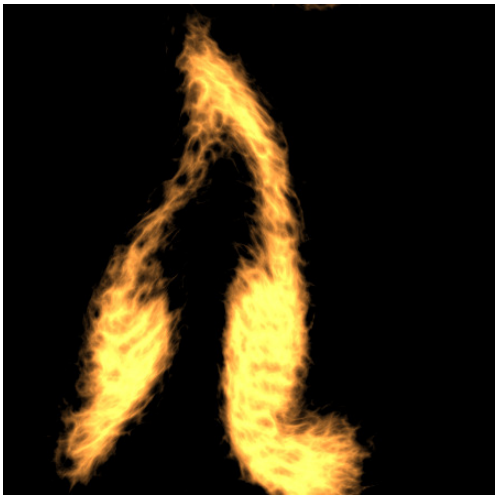
Frame 60 Frame 80

Figure 6.11: Flowing rivers using advected noise. The high-frequency component of the advected noise is used for bump mapping, and the the low-frequency component is used for displacement mapping. Please consult the accompanying video for a better demonstration.



Frame 20

Frame 40



Frame 60

Frame 80

Figure 6.12: Animated fires with abundant details. It is generated in several steps. A 2D density field is advected with a low resolution velocity field. Then, advected flow noise is used to modulate the density field. Finally, a fire shader uses the enriched density field to generate colors.

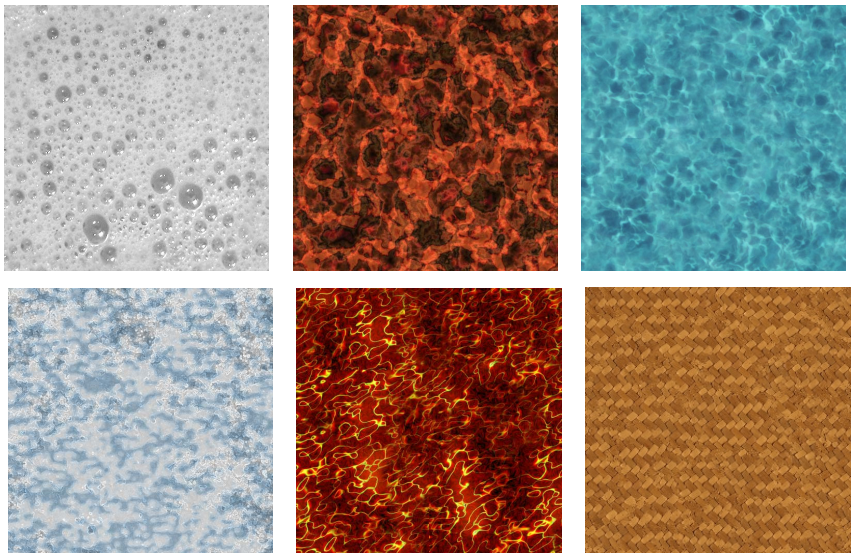


Figure 6.13: Non-noise textures used in our tests. The last one represents highly structured textures.

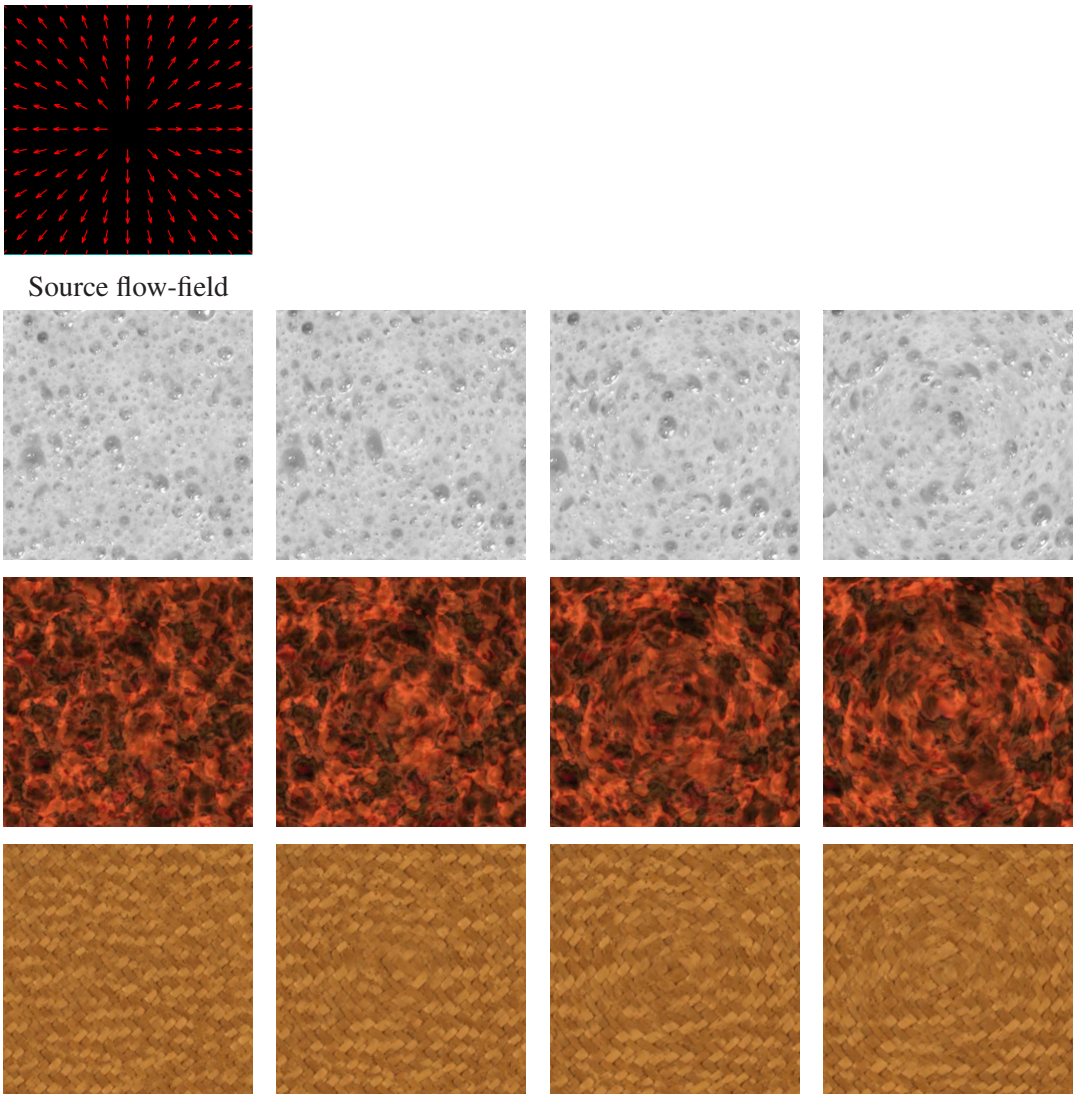


Figure 6.14: Advecting non-noise image textures using a flow-field. Shown are keyframes from texture sequences that follow a source flow-field. It demonstrates that our method works for some image textures (top and middle) but not for highly structured textures (bottom).

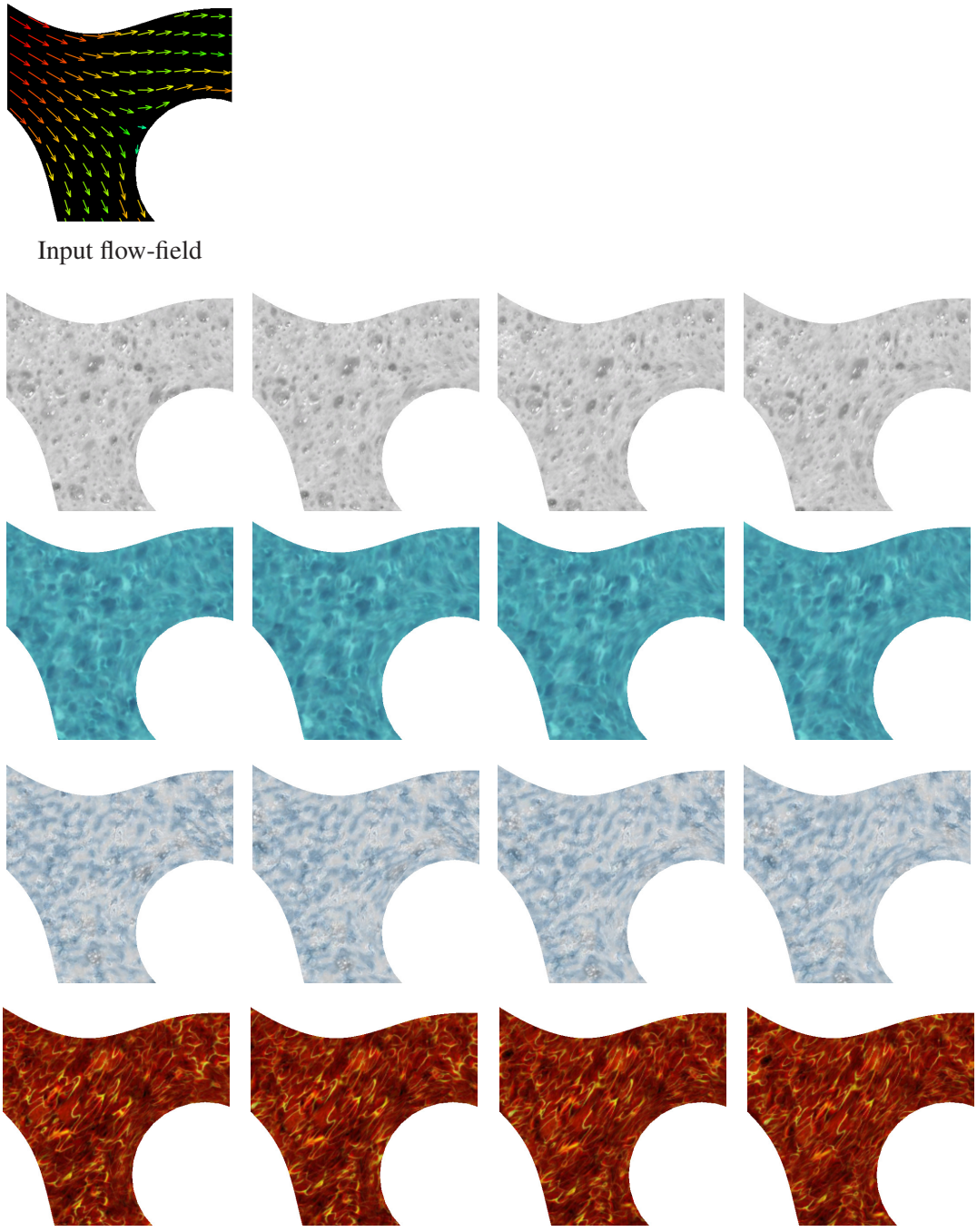


Figure 6.15: Advecting non-noise image textures using a flow-filed. Shown are keyframes from texture sequences that follow a boundary confined flow. It demonstrates that our method works for constrained flow and many image textures.

Chapter 7

Conclusion

The two sections below list the specific contributions of this dissertation and outline directions for future work in this area.

7.1 Summary of Contributions

We have proposed a three-scale framework for river animation, with novel models for each scale. In the macro-scale, we proposed a procedural method for generating plausible river flow on-the-fly. In the meso-scale, we proposed a feature-based vector simulation approach and applied it to simulating the waves caused by obstacles in a running stream. In the micro-scale, we proposed an adaptive method for texturing large animated flow surfaces with scene-independent performance. Moreover, we proposed a spectrum-preservation texture advection method based on the Lagrangian formalism. Both of the two texturing models rely on our efficient dynamic sampling scheme.

Combining these models together, we have achieved real-time, scalable and controllable river animation (see accompanying video [1]). The performance of our animation can meet the requirement of real-time applications. The procedural velocity and screen space sampling make our animation workable for landscapes and other very large scale scenes. Meanwhile, users are also allowed to observe a river closely and detailed surface waves are available. Our models ensure that all parts of the river are continuous both in space and time, even while the user is exploring, zooming or editing the river itself. In addition, the user can intuitively design and control the river flow (*e.g.,* trajectory and

[1] `http://evasion.inrialpes.fr/Membres/Qizhi.Yu/phd/`

velocity), and easily modify the surface details of rivers by changing reference textures or wave profiles.

7.2 Limitations and Future Work

In this section, we discuss the limitations and future work for the proposals in each scale.

7.2.1 Macro-Scale

The velocity profile of our procedural flow is influenced both by the interpolation parameter p and the search radius s in Equation 3.7. However, we still do not know the precise relationship between these parameters and the velocity profile. Future work needs to be done in order to develop an accurate handle that allows the user to manipulate the velocity profile.

In this work, we assumed that the river surface in the macro-scale is relatively unchanging. Further considerations will allow for the inclusion of time-varying long waves, and an efficient model is needed to simulate this unsteady river surface.

In addition, we have argued that it is possible to account for the slop of terrain in our model. We would like to add this feature in our implementation.

7.2.2 Meso-Scale

Besides the waves caused by obstacles, many other local and structured waves in running streams could be simulated by using featured-based vector simulation, such as hydraulic jumps, boils, ship waves, turbulent wakes, and cascades (Figure 1.2). For each phenomenon, one first needs to define appropriate vector primitives. and a physical model for constructing and animating the vector features. Then, the vector features must be converted to a representation suitable for high-quality rendering.

7.2.3 Micro-Scale: Wave Sprites

On the implementation side, it would be useful to use various wave examples for different sprites according to some domain-wise controls (*e.g.,* wind shadow, wakes). It

would also be interesting to replace the bump mapping with displacement mapping in our implementation.

Regarding the model itself, it would be interesting to adapt the distribution of particles to the stretching of the flow in order to better represent regions with high velocity variation. This is a form of importance sampling (*i.e.,* it is no longer a Poisson-disk homogeneous distribution). Note that the boundary sampling method [DH06] could be adapted for this purpose. The simplest way might be to use the method directly with different sized Poisson-disks. It does work as long as the ratio of their sizes is less than two.

7.2.4 Micro-Scale: Lagrangian Texture Advection

It would be useful to incorporate our Lagrangian texture advection method into the wave sprites model. By doing this, we would obtain an adaptive spectrum-preserving texture advection method.

With our method, advecting structured images sometimes does not yield good results because the texture content of each patch is chosen randomly from the reference texture. We could improve this by choosing for a newly born patch a portion of the reference texture that best matches the textures of its neighboring patches. Since all the patches strictly follow the underlying flow, the correlation between the patches will remain in the following frames.

To fully exploit the benefits of the Lagrangian formalism, we would like to extend the texture advection method for working on a 3D surface or in a volume. The fist step is to extend our dynamic poisson-disk sampling scheme to a higher dimension.

7.3 French Chapter Abstract

Dans cette thèse, nous avons proposé un modèle multi-échelle pour l'animation de rivière. Nous avons présenté un nouveau modèle pour chaque échelle. A l'échelle macro, nous avons proposé une méthode procédurale permettant de générer une rivière réaliste à la volée. A l'échelle méso nous avons amélioré un modèle phénoménologique basé sur une représentation vectorielle des ondes de choc près des obstacles, et proposé une methode pour la reconstruction adaptative de la surface de l'eau. A l'échelle mi-

cro, nous avons présenté une méthode adaptative pour texturer des surfaces de grande étendue avec des performances indépendantes de la scène. Nous avons également proposé une méthode d'advection de texture. Ces deux modèles reposent sur notre schéma d'échantillonnage adaptatif.

En combinant ces modèles, nous avons pu animer des rivières de taille mondiale en temps réel, tout en étant contrôlable. Les performances de notre système sont indépendantes de la scène. La vitesse procédurale et l'échantillonage en espace écran permettent à notre système de fonctionner sur des domaines illimités. Les utilisateurs peuvent observer la rivière de très près ou de très loin à tout moment. Des vagues très détaillées peuvent être affichées. Les différents parties des rivières sont continues dans l'espace et dans le temps, même lors de l'exploration ou de l'édition de la rivière par un utilisateur. Cela signifie que l'utilisateur peut éditer les lits des rivières ou ajouter des îles à la volée sans interrompre l'animation. La vitesse de la rivière change dès que l'utilisateur en édite les caractéristiques, et l'utilisateur peut auss modifier son apparence avec des textures.

Bibliography

[AVO02] John Isidoro Alex Vlachos and Chris Oat. Rippling reflective and refractive water. In *ShaderX : Vertex and Pixel shader Programming Tips and Tricks*. Addison-Wesley, 2002.

[BHN07] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 46, New York, NY, USA, 2007. ACM.

[BK01] Mario Botsch and Leif Kobbelt. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. *Computer Graphics Forum*, 20(3):402–410, September 2001.

[Bru08] Eric Bruneton. Real-time rendering and editing of vector-based terrains. *Computer Graphics Forum*, 27:311–320(10), April 2008.

[BSHK04] Kiran S. Bhat, Steven M. Seitz, Jessica K. Hodgins, and Pradeep K. Khosla. Flow-based video synthesis and editing. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 360–363, New York, NY, USA, 2004. ACM.

[BSM+06] Adam W. Bargteil, Funshing Sin, Jonathan E. Michaels, Tolga G. Goktekin, and James F. O'Brien. A texture synthesis method for liquid animations. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 345–351, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[CD05] Robert L. Cook and Tony DeRose. Wavelet noise. *ACM Trans. Graph.*, 24(3):803–811, 2005.

[CdVL95] Jim X. Chen and Niels da Vitoria Lobo. Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations. *Graph. Models Image Process.*, 57(2):107–116, 1995.

[CdVLHM97] Jim X. Chen, Niels da Vitoria Lobo, Charles E. Hughes, and J. Michael Moshell. Real-time fluid simulation in a dynamic virtual environment. *IEEE Comput. Graph. Appl.*, 17(3):52–61, 1997.

[Che04] Stephen Chenney. Flow tiles. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 233–242, 2004.

[Cor07] H. Cords. Mode-splitting for highly detailed, interactive liquid simulation. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 265–272, New York, NY, USA, 2007. ACM.

[Cry08] Crysis, 2008. `http://www.ea.com/crysis/`.

[DH06] Daniel Dunbar and Greg Humphreys. A spatial data structure for fast poisson-disk sample generation. *ACM Trans. Graph.*, 25(3):503–508, 2006.

[EMF02] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 736–744, New York, NY, USA, 2002. ACM Press.

[FF01] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, New York, NY, USA, 2001. ACM.

[FM96] Nick Foster and Demitri Metaxas. Realistic animation of liquids. In Wayne A. Davis and Richard Bartels, editors, *Graphics Interface '96*, pages 204–212. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1996.

[FM98] Alexey V. Federov and W. Kendal Melville. Nonlinear gravity-capillary waves with forcing and dissipation. *J. Fluid Mech.*, 354:1–42, 1998.

[FPRJ00] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *SIGGRAPH '00*, pages 249–254, 2000.

[FR86] Alain Fournier and William T. Reeves. A simple model of ocean waves. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 75–84, New York, NY, USA, 1986. ACM.

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1996. ACM.

[Gla02] Andrew Glassner. Duck! *IEEE Computer Graphics and Applications*, 22(4):88–97, 2002.

[Goo08] Google earth, 2008. `http://earth.google.com/`.

[GW78] William J. Gordon and James A. Wixom. Shepard's method of "metric interpolation" to bivariate and multivariate interpolation. *Mathematics of Computation*, 32(141):253–264, 1978.

[HNC02] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. Interactive animation of ocean waves. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 161–166, New York, NY, USA, 2002. ACM.

[IC00] Doug Ikeler and Jennifer Cohen. The use of spryticle in the visual FX for "the road to el dorado". In *ACM SIGGRAPH 2000 sketches*, 2000.

[IGLF06] Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 805–811, New York, NY, USA, 2006. ACM.

[Jac76] R. G. Jackson. Sedimentological and fluid-dynamic implications of the turbulent bursting phenomenon in geophysical flows. *Journal of Fluid Mechanics*, 77:531–560, 1976.

[JG01] Lasse Jensen and Robert Golias. Deep-water animation and rendering. In *Game Developers Conference Europe*, 2001. http://www.gamasutra.com/gdce/jensen/jensen_01.htm.

[KAK+07] Vivek Kwatra, David Adalsteinsson, Theodore Kim, Nipun Kwatra, Mark Carlson, and Ming Lin. Texturing fluids. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):939–952, 2007.

[KKN+89] K. Kaneda, F. Kato, E. Nakamae, T. Nishita, H. Tanaka, and Takao Noguchi. Three dimensional terrain modeling and display for environmental assessment. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 207–214, New York, NY, USA, 1989. ACM.

[KKS08] Andrew Kensler, Aaron Knoll, and Peter Shirley. Better Gradient Noise. Technical Report UUSCI-2008-001, SCI Institute, University of Utah, 2008.

[KM90] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, New York, NY, USA, 1990. ACM.

[KMN88] Alex D. Kelley, Michael C. Malin, and Gregory M. Nielson. Terrain simulation using a model of stream erosion. *SIGGRAPH Comput. Graph.*, 22(4):263–268, 1988.

[KTJG08] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–6, New York, NY, USA, 2008. ACM.

[KW06] Peter Kipfer and Rüdiger Westermann. Realistic and interactive simulation of rivers. In *GI '06: Proceedings of Graphics Interface 2006*,

pages 41–48, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[LD06] Ares Lagae and Philip Dutré. A comparison of methods for generating Poisson disk distributions. Report CW 459, Department of Computer Science, K.U.Leuven, Leuven, Belgium, August 2006.

[LGF04] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 457–462, New York, NY, USA, 2004. ACM.

[LHN05] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. Texture sprites: texture elements splatted on surfaces. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 163–170, New York, NY, USA, 2005. ACM.

[LN03] Sylvain Lefebvre and Fabrice Neyret. Pattern based procedural textures. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 203–212, New York, NY, USA, 2003. ACM.

[L.P00] Kobbelt L.P. An interactive approach to point cloud triangulation. *Computer Graphics Forum*, 19:479–487(9), September 2000.

[LvdP02] Anita T. Layton and Michiel van de Panne. A numerically efficient and stable algorithm for animating water waves. *The Visual Computer*, 18(1):41–53, 2002.

[Max81] Nelson L. Max. Vectorized procedural models for natural terrain: Waves and islands in the sunset. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques*, pages 317–324, New York, NY, USA, 1981. ACM.

[MB95] Nelson Max and Barry Becker. Flow visualization using moving textures. In *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data*, pages 77–87, 1995.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[MF92] M. Mccool and E. Fiume. Hierarchical poisson disk sampling distributions. In *Proc. of the Graphics Interface '92*, pages 94–105, Vancouver, Canada, 1992.

[Mit04] Jason L. Mitchell. Real-time synthesis and rendering of ocean water. In *ATI Research Technical Report*. 2004.

[Mol95] Thomas Molls. Depth-averaged open-channel flow model. *Journal of Hydraulic Engineering*, 121(6):453–465, 1995.

[MWM87] G. A. Mastin, P. A. Watterberg, and J. F. Mareda. Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Applications*, 7(3):16–23, March 1987.

[Ney03] Fabrice Neyret. Advected textures. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 147–153, 2003.

[NHS02] Fabrice Neyret, Raphael Heiss, and Franck Senegas. Realistic rendering of an organ surface in real-time for laparoscopic surgery simulation. *the Visual Computer*, 18(3):135–149, may 2002.

[NKL+07] Rahul Narain, Vivek Kwatra, Huai-Ping Lee, Theodore Kim, Mark Carlson, and Ming Lin. Feature-guided dynamic texture synthesis on continuous flows. In *Eurographics Symposium on Rendering 2007*, Grenoble, France, 2007.

[NP01] Fabrice Neyret and Nathalie Praizelin. Phenomenological simulation of brooks. In *Computer Animation and Simulation*, pages 53–64. Eurographics, Springer, Sep 2001. Eurographics Workshop on Animation and Simulation, Manchester.

[NSCL08] Rahul Narain, Jason Sewall, Mark Carlson, and Ming Lin. Fast animation of turbulence using energy transport and procedural synthesis. *ACM Transactions on Graphics, SIGGRAPH Asia*, 27(5), December 2008.

[OH95] J. F. O'Brien and J. K. Hodgins. Dynamic simulation of splashing fluids. In *CA '95: Proceedings of the Computer Animation*, page 198, Washington, DC, USA, 1995. IEEE Computer Society.

[Ope08] Openfoam, 2008. `http://www.opencfd.co.uk/openfoam/`.

[Pea86] Darwyn R. Peachey. Modeling waves and surf. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 65–74, August 1986.

[Per85] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 287–296, July 1985.

[PN01] Ken Perlin and Fabrice Neyret. Flow noise: textural synthesis of animated flow using enhanced Perlin noise. In *SIGGRAPH 2001 Technical Sketches and Applications*, August 2001. http://www-imagis.imag.fr/Publications/2001/PN01.

[RB85] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 313–322, July 1985.

[Ree83] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[SB08] Hagit Schechter and Robert Bridson. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation*, 2008.

[SCOGL02] Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 355–362, 2002.

[She68] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, New York, NY, USA, 1968. ACM.

[Sim90] Karl Sims. Particle animation and rendering using data parallel computation. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 405–413, New York, NY, USA, 1990. ACM.

[Sta99] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[Sto57] J. J. Stoker. *Water Waves: The Mathematical Theory*, volume IV of *Pure and Applied Mathematics*. Interscience Publishers, Inc., 1957.

[TDG00] Sebastien Thon, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Ocean waves synthesis using a spectrum-based turbulence function. In *Proceedings of Computer Graphics International 2000*, pages 65–72, 2000.

[Tes04] Jerry Tessendorf. Simulating ocean water. In *The elements of nature: interactive and realistic techniques*. ACM Press, 2004. SIGGRAPH 2004 Course Notes 31.

[TG01] Sebastien Thon and Djamchid Ghazanfarpour. A semi-physical model of running water. In *Eurographics UK*, pages 53–59, 2001.

[TMFSG07] Nils Thürey, Matthias Müller-Fischer, Simon Schirm, and Markus Gross. Real-time breakingwaves for shallow water simulations. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 39–46, Washington, DC, USA, 2007. IEEE Computer Society.

[TSS+07] Nils Thürey, F. Sadlo, S. Schirm, M. Müller-Fischer, and M. Gross. Real-time simulations of bubbles and foam within a shallow wa-

ter framework. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 191–198, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[VBTS07] David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François Sillion. Dynamic point distribution for stroke-based rendering. In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, pages 139–146, 2007.

[vFTS06] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, 2006.

[WH91] Jakub Wejchert and David Haumann. Animation aerodynamics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 19–22, New York, NY, USA, 1991. ACM.

[Whi01] Frank M. White. *Fluid Mechanics*, section 4.7, pages 238–245. McGraw-Hill, Columbus, OH, fourth edition, 2001.

[Won] Wonder touch. `http://www.wondertouch.com/`.

[Wu04] Weiming Wu. Depth-averaged two-dimensional numerical modeling of unsteady flow and nonuniform sediment transport in open channels. *Journal of hydraulics Engineering*, 130(10):1013–1024, 2004.

[XGS00] Y. Xu, B. Guo, and H.-Y. Shum. Chaos mosaic: Fast and memory efficient texture synthesis. Technical Report MSR-TR-2000-32, Microsoft Research, 2000.

[YHK07] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 99, New York, NY, USA, 2007. ACM.