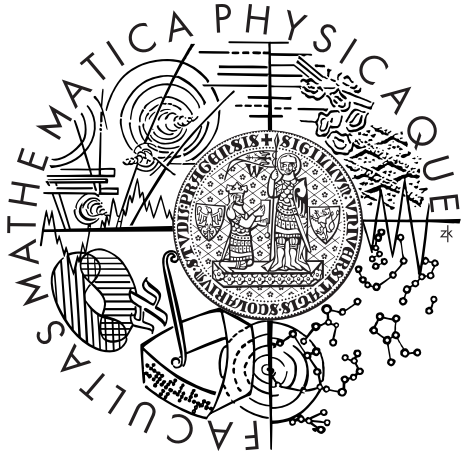
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Mária Vámošová

Wang Tiles for Frequency-Based Synthesis of Ocean Surfaces

Department of Software and Computer Science Education

Supervisor of the master thesis: doc. Dr. Alexander Wilkie
Study programme: Computer Science
Specialization: Software Systems

Prague 2013

I would like to thank Alexander Wilkie for his valuable advices as well as motivation and excitement about the topic. I would also like to thank all my friends for their support and encouragment.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, December 5, 2013 Mária Vámošová

Název práce: Wangovy dlaždice pro syntézu povrchu oceánu frekvenčními technikami

Autor: Mária Vámošová

Katedra: Kabinet software a výuky informatiky

Vedoucí diplomové práce: doc. Dr. Alexander Wilkie, Kabinet software a výuky informatiky

Abstrakt: Syntéza oceánu pomocí frekvenční analýzy je skupina metod, které modelují vlny v oceánu na základě frekvenčních spekter zkoumaných v rámci oceánografického výskumu. Výsledkem těchto metod je výškové pole, které je periodické a dá sa použít opakovaně na vydláždění větší plochy. Když ale použijeme dlaždic příliš mnoho, opakující sa vzory začnou být rušivé. Wangovo dláždění je metoda pro pokrytí roviny neperiodicky pomocí malého počtu předgenerovaných dlaždic. Cílem této práce je vytvořit sadu Wangových dlaždic. Každá dlaždice bude reprezentovat výškové pole oceánu vygenerované frekvenční metodou a pomocí těchto dlaždic potom můžeme vydláždit velkou plochu oceánu bez zjevné periodicity.

Klíčová slova: Wangovy dlaždice; syntéza oceánu frekvenčními technikami; procedurální modelování

Title: Wang Tiles for Frequency-Based Synthesis of Ocean Surfaces

Author: Mária Vámošová

Department: Department of Software and Computer Science Education

Supervisor: doc. Dr. Alexander Wilkie, Department of Software and Computer Science Education

Abstract: Frequency-based ocean synthesis is a class of methods for generating ocean waves in deep water using physically based ocean spectra provided by oceanographic research. The output of these methods is a heightfield which is periodic and can be used to tile a larger area. However, when using a larger number of tiles, a repetition pattern becomes apparent. Wang tiling is a method of tiling the plane non-periodically by using only a small set of input tiles with given constraints. The aim of this thesis is to create a set of Wang Tiles, each of which represents an ocean heightfield synthesised by a frequency-based method, so one can create large ocean scenes without apparent periodicity in the tiling pattern.

Keywords: Wang tiles; frequency-based ocean synthesis; procedural modelling

Obsah

1 Introduction **3**

2 Overview **5**
2.1 Deep Ocean Simulation 5
2.1.1 Spatial methods 5
2.1.2 Fourier domain methods 8
2.1.3 Hybrid methods 9
2.2 Tessendorf algorithm 10
2.3 Wang Tiles 12

3 Wang Tiles for Frequency Generated Ocean Surfaces **15**
3.1 Set of tiles 16
3.1.1 Wang Tiles generation 17
3.2 Interpolation of a sinusoid 18
3.2.1 1D case 18
3.2.2 2D case 21
3.2.3 Animation 22
3.2.4 Choppy waves 23
3.3 Wang Tiles 24
3.3.1 Basic tiles 26
3.3.2 More complex tiles 27

4 Algorithm **30**
4.1 Corners 30
4.1.1 Low Frequencies 31
4.1.2 High Frequencies 31
4.2 Complexity 32
4.3 GPU implementation 35

5 Implementation **36**
5.1 User Guide 36
5.1.1 Parameters 36
5.1.2 Navigation 38
5.1.3 Visualisation options 38
5.2 Programmer Documentation 38
5.2.1 Qt integration 38
5.2.2 Wang Tiles 39
5.2.3 Ocean rendering 40
5.2.4 Water shader 41

6 Experiments **42**
6.1 Ocean spectra 42
6.1.1 Phillips 42
6.1.2 JONSWAP 43
6.2 Wind speed in comparison to ocean tile area 45
6.3 Tile differentiation 47

6.4 Interpolation directions 48
6.5 Corner areas 48
6.6 Choppy waves 48
6.7 Suggested scenarios 49

7 Results **50**
7.1 Comparison of different techniques 50
7.2 Spectral analysis 51

8 Conclusion **53**

Bibliography **54**

1. Introduction

Realistic water simulation has been of great interest since the very beginning of computer graphics. Water is a natural phenomenon present in a great variety of scenes, ranging from a glass of water on a table through an animation of a sprinkling fountain, up to a large-scale overview of an ocean scenery. Water in some form appears almost everywhere. To produce realistically looking visuals, we have to deal with its highly dynamic properties - correctly simulate flowing, splashing, surface waves and ripples, and many other properties - as well as to compute all the light interactions: reflections, refractions, even light scattering occuring in the water volume, if we want to be truly realistic. All this makes the water simulation a challenging task.

In case of the ocean, we also have to deal with its massive size. While in some other fluid simulations we can rely on physics based approaches like Computational Fluid Dynamics, or Smoothed Particle Hydrodynamics, we cannot really use these techniques for the whole ocean mass. Therefore, in practical applications, sea is usually modelled as a heightfield by only considering its surface and not taking the volume underneath into account. More complicated physical computations are performed only in some regions: around the shoreline, when interacting with objects or when simulating breaking waves under heavy wind conditions.

The simplest case of an ocean scene is the ocean in deep water, far from the shore, without any object interaction and under reasonable weather conditions. In such an ocean, the surface does not interfere with the sea bottom and is not affected by fixed objects. Such sea is composed mainly by wind-generated gravity waves, and it can be simulated efficiently, producing visually plausible results.

One of the most popular approaches for this setting are the frequency based techniques. The outcome of these techniques is a heightfield of a fixed size, which is periodic and can be used repeatedly to tile a larger area. A major drawback of such tiling is its visible periodicity - one can easily see that it is the same heightfield repeating over and over. To cover the whole area, the ideal case would be to generate a large heightfield - however, that soon starts to be very costly as the area increases.

To address this problem, we propose to use a technique called Wang tiling. A Wang Tile is a square tile with four edges, each with an assigned color. A valid tiling is such tiling that all connected edges are of the same color. It has been found that with only a small set of Wang Tiles, a plane can be tiled aperiodically.

This has been of great use in computer graphics. Each Wang Tile can represent a piece of texture, and a particular color of an edge represents the pixel values of the texture in that area. Since only tiles with matching values can be laid next to each other, the result forms a seamless and aperiodic image.

We propose to use this technique for the ocean surface and create a large heightfield with only a small set of frequency-generated ocean tiles. The goal of this thesis is therefore to create a set of ocean heightfield tiles, each representing a Wang Tile, so that the height values would match at the borders, while at the same time, the overall appearance of the surface would remain undistorted.

The text is organised as follows. In Chapter 2 we provide an overview of existing algorithms for generating deep ocean surface and describe the previous

usage of Wang Tiles in computer graphics. In Chapter 3 we describe the process of creating a set of Wang Tiles for an ocean heightfield. In Chapter 4 we describe in more detail the exact algorithm for producing an aperiodic ocean surface and how to deal with additional problems that arise in the process. In Chapter 5 we describe the program we have created for the purpose of visual comparison of the results. In Chapter 6 we experiment with various settings, compare between different wave spectra and suggest the best usage scenarios. In Chapter 7 we evaluate the results and discuss the limitations of our algorithm. Chapter 8 concludes and suggest further work.

2. Overview

2.1 Deep Ocean Simulation

Realistic ocean simulation has been studied for a long time, therefore, many methods have been developed over time. In general, the algorithms use mathematical models as well as experimental oceanographic observations. They can be divided into three main categories: methods that evaluate the wave shape directly in the spatial domain, methods that represent the ocean surface in the Fourier domain and methods that combine both approaches.

2.1.1 Spatial methods

The base model of a single ocean wave is a sinusoidal function. This model is based on the underlying physical *linear wave theory*, the *Airy wave*, derived from the general Navier-Stokes equations on a free surface under certain assumptions that are reasonably realistic for an ocean in deep water.

The formula is

$$h(x, z, t) = A \cdot \cos(k_x x + k_z z - \omega t)$$

The vector (k_x, k_z) specifies the spatial frequency of the wave, while ω defines the speed in which the wave travels in time. A is the amplitude of the wave.

The surface of such a wave is a long-crested cosine fuction that is translated in the wave direction during the animation process.

The ocean is then composed of a number of such waves, and realistic effect is obtained by using waves of different directions and frequencies, moving at different speeds.

This method was first used in Computer Graphics by Schachter [47] to produce a simple bump map, and later by Max [35], creating a heightfield so that the waves can occlude each other and produce a more realistic image.

Perlin [42], on the other hand, does not use long crested waves, but waves originating from one point, as if a rock was thrown into a calm water. He uses 20 such origins, and combines the effect of all of them. This way, he was able to get rid of the repeating patterns long-crested waves produce.

For a simple deep water wave, the numbers (k_x, k_z) and its wavespeed ω are constant across all positions on the suface. However, to obtain a more realistic surface, Peachy [41] proposed to evaluate the phase shift at each point separately, and use a more complex formula that takes the actual sea depth at that point into account. One wave is therefore no longer homogenous over the surface, but has a different shape over different water depths and produces a more complex surface. Using this method, he was able to simulate waves as they approach shallow beach, including the refraction effect that causes waves coming from all directions to allign along the coast.

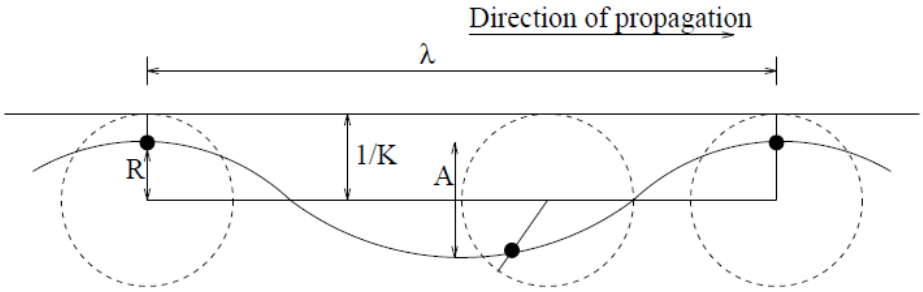
The waves computed using the sinusoid formula have rounded peaks which does not represent a real ocean wave well. To create a more realistic waves, we can use gerstner waves instead. Gerstner (or Rankine) wave was first described

by Gerstner in 1802 [16] as a solution to the physical problem, and was later re-descovered by Rankine [44]. The surface of the wave is computed as

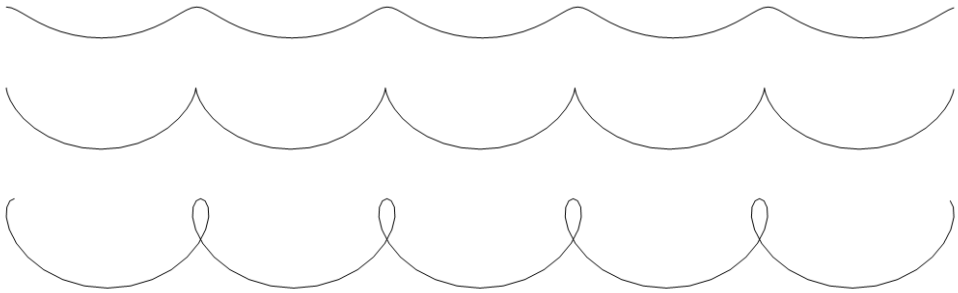
$$\begin{aligned}(x, z) &= (x_0, z_0) - \frac{(k_x, k_z)}{|(k_x, k_z)|} A \sin(k_x \cdot x_0 + k_z \cdot z_0 - \omega t) \\ y &= A \cos(k_x \cdot x_0 + k_z \cdot z_0 - \omega t)\end{aligned} \tag{2.1}$$

where $(k_x, k_z) = (2\pi/L_x, 2\pi/L_z)$, (L_x, L_z) is the wavelegth and ω again stands for the speed of the wave.

A Gerstner wave represents a curve described by a fixed point attached to a circle in some distance as the circle rolls along a straight line.



The surface shape is defined by the ratio between $k = |(k_x, k_z)|$ and the amplitude A. As the value kA approaches 1, the wave gets broader valleys and steeper tops. For $kA = 1$ the tops are perfectly sharp. For values $kA > 1$ the surface degenerates and no longer describes a water wave.



This formula was used by Fournier and Reeves [12]. They also introduced a series of modifications to incorporate wave refraction at changing water depths, modelling breaking waves at the shore with nice results. To model the wave correctly, upon reaching the shoreline, the water particle no longer moves around a circle, but rather an ellipse. This results in waves being stretched and realistically simulate waves approaching a beach.

Ts'o and Barsky [52] also work with refraction of waves - how do the waves change their direction when approaching a shoreline. They model the refraction in a way similar to the light refraction, using the Snell's law, and trace the wavetrains using *wave tracing* method. The wave surface of a single wave is then not a sine wave or a Gerstner wave, but rather a B-spline approximating the more complex surface created by a wave being refracted multiple times.

Other researchers build on this refraction approach, like Gonzato and Le Saëc [18]. Gamito and Musgrave [15] proposed a more accurate refraction com-

putations, also based on the wave tracing. The results are stored as a phase map over the terrain which is then applied to any geometric wave model.

Even though individual methods differ, the core remains the same. We compose the ocean as a sum of individual waves, and evaluate these waves at each heightpoint. This allows us to account for all sorts of forces influencing the wave shape and to produce realistic results under many conditions.

However, the number of these waves is crucial for the visual quality. This means the overall complexity gets to be problematic for larger scenes. This problem was addressed by proposed GPU oriented algorithms or LOD approaches.

Schneider and Westermann [48] provide a GPU implementation of an ocean surface generated by a turbulence gradient noise. Isidoro et al. [23] use a precomputed mesh perturbed by four sinusoids with low frequencies in the vertex shader. Small details are added through bump-mapping specified by a fixed texture.

Kryachko [28] uses vertex texture displacement and computes two sets of heightmaps: one with the heightmaps containing large waves, used over the whole surface for displacement, and one set with small details that are used as a normal map.

Many implementations simply use the techniques mentioned above and transfer the computations to the GPU. Salgado and Conci [46] provide an NVIDIA Cg implementation of the Gerstner wave model. Chen et al [6] create basic geometry by summing up sine waves in the vertex shader and adding the small details through bump-mapping.

As we are evaluating the height at each point separately, more complex phenomena can be simulated. Chou and Fu [4] decribed a GPU implemetation that combines a basic wave simulation with particle simulation around objects like ships. Particles not only react to the motion at each vertex, but also to general hydrodynamic forces.

A natural way of making the simulation realtime is to only simulate parts of the components, depending on distance. High frequencies or details can be ommited in distant parts and the simulation can focus on high quality in the areas close to the viewer.

Cui et al. [8] propose an adaptive tesselation, based on the approach by Hisinger et al. [21] (mentioned in the next section). The heightfield grid is adaptively tesselated so that it would be approximately evenly distributed in the screen space. A similar method, the grid approach, was proposed by Johanson [25]. A regular grid in the camera space is projected onto the ocean plane, so the world-space ocean grid is naturally coarser in the distance, and dense in the close areas.

Lee et al. [32], [33] futher improve this frustum technique and provide a realtime simulation usable in games or movie industry. They use Perlin noise as a wave generation model.

One of the most recent is a method by Bruneton et al. [1]. They propose an LOD approach where they simulate the geometry for the nearest points, then move the details to a bump map for the vertices futher away, and the areas in the distance get their lighting through BRDF. The method deals with how to make the transition seamless and fast and produces very good visual results.

To conclude, spatial methods produce very good results and when combined with GPU and acceleration techniques, can reach interactive simulation speed. As they deal with each point separately, they can easily incorporate more complex computations in areas around objects or in shallow water. However, many of these modifications slow down the simulation process.

There is also an issue with tuning up the wave parameters. The ocean is composed of many individual waves, each of which has to have a specified frequency, amplitude and phase shift. The setting of these parameters is crucial for the visual realism, and it can also reduce the computational speed if choosing the right set of waves. This is where the theoretical models of oceanography can be of great use.

2.1.2 Fourier domain methods

The ocean has been studied many years before the advent of computer graphics, from a physics viewpoint, either to understand certain phenomena, but also to be able to predict the ocean shape depending on atmospheric conditions. Such computations are useful in case we want to know when does a storm swell arrive to a coast, or to decide if the conditions are well enough for a ship to sail.

In the oceanographic research, the ocean surface is considered a random variable of horizontal position (x, z) and time t. Under this assumption, theoretical models have been built upon its Fourier transform. These models are based either on theoretical statistical formulas or have been created to match measured empirical data. The models have various parameters such as the wind speed, or the area over which the wind blows, mainly the most important atmospheric conditions we are interested in.

An *ocean spectrum* is a probablility density function which desribes the distribution of wave energy by its frequency.

This approach has been introduced to computer graphics by Mastin et al. [37]. They proposed to create a white noise field, compute its Fourier transform and filter the magnitudes by Pierson-Moskowitz empirical ocean spectrum. In addition, a direcional spreading function was added, to attenuate the waves travelling in various directions, as the original ocean spectra do not take directionality into account. The resulting maginitudes are then combined with the unfiltered phase shifts, and the heightfield is synthesised back. Similarily to the spatial methods, the animation is done by modifying the phase shift of the individual waves, each according to its speed, the computations performed in the Fourier domain. This method produced good results, as the wave amplitudes followed the real world data.

Premoze and Ashikhmin [40] later used the same idea, but with a different ocean spectrum to improve the visual quality.

Stam [50] also builds on this method, but performs the filtering of the noise in spatial domain, with a smaller kernel. Byt this approach he creates a set of tiles he then uses to tile a larger area. More details about the tiles are mentioned in the next section.

Probably the most famous Fourier based method is the one introduced by Tessendorf [54], used widely since then. Instead of filtering a white noise heightfield, Tessendorf creates the wave amplitudes and phase shifts directly in the Fourier

domain using the ocean spectra. During the animation, a spectrum for a given timeframe is obtained by phase shifting the basic spectrum, and the heightfield is synthesized by the inverse Fourier transform. He also introduced a modification to create a more realistic choppy waves, with broader troughs and sharper peaks, by displacing the positions of the points in the heightfield. This is done in a similar way as for the gerstner waves, with the displacement value being computed by the Fourier transform as well, so the computational speed is still fast.

However, methods based on a simple Fourier transform also have a drawback. They work with a grid with fixed resolution, therefore, either small details are missing, or the computational cost becomes too large. To address this issue, again, LOD modifications or GPU implementations have been introduced.

Jensen and Goliáš [24] divide the frequencies into two sets. Low frequencies creating a global motion are stored in a displacement vertex shader, and high frequencies are stored in a normal map. However, the FFT computations are still performed on the CPU and the heightfield has to be transferred at each step.

Mitchell [36] build on this approach and introduced a GPU implementation performing all the computations fully at the GPU, including a GPU FFT implementation, so they were able to reach interactive speeds. An implementation by Chiu and Chang [3] combines this approach with adaptive GPU-based ocean surface tessellation.

Hu et al. [22] use two ocean surfaces, one sampled with a fixed high resolution to represent the small details near the viewer, and the other sampled adaptively for the waves in greater distances.

2.1.3 Hybrid methods

Hybrid methods are a combination of the previous two approaches. They work in the spatial domain by evaluating individual waves, but they guide the wave parameters by the empirically based ocean spectra. Using this approach, the parameters such as the wind speed are incorporated into the simulation while keeping the benefits of the spatial approach like manipulating the wave shape depending on a particular spatial position.

Thon et al. [55] propose to create the ocean as a sum of trochoids adaptively sampled from an empirical ocean spectrum. The number of the trochoids is dependent on the distance, and only the areas of the spectrum with significant waves is being sampled. As the trochoids are periodic, to avoid repeating patters, they modify the phase of the trochoid at each point by the Perlin noise function. This way, each wave is no longer a simple trochoid, but has a more complex, spatially varying shape.

Hisinger et al. [21] then apply an LOD scheme to this approach which increases the computational speed. They sample the heightfield in such a way that the gridpoints are evenly distributed in the screen space.

Lee et al. [31] use a similar approach, except they use simple sinusoids and a more complex spectrum and also take the ocean depth into account. Xin et al. [58] incorporate also wind into the simulation.

A real-time ocean simulation framework was created by Lachman [29]. The framework is able to incorporate any wave model, spatial or frequency-based, and

provides an unified environment for the ocean simulation. The ocean surface is divided into grids of heightfields so an LOD scheme is used.

Robine and Frechot [45] use a fast sound synthesis method to perform a faster wave summation of individual Gerstner waves. This way, they are being able to speed up the animation in a similar way to the Fourier transform, but also keeping more freedom over the individual waves.

2.2 Tessendorf algorithm

As mentioned earlier, one of the most famous and most widely used algorithms for simulating the ocean surface is a frequency-based method proposed by Jerry Tessendorf [54]. In general, frequency-based ocean simulations are considered superior for many applications for their high visual realism and fast speed for large number of component waves based on the FFT principle. The Tessendorf ocean generation algorithm is the underlying algorithm of most recent frequency-based methods. Various LOD and GPU accelerations have been proposed, but the underlying wave-shape generator is usually this one.

Therefore, we also take this algorithm as the method for generating the ocean heightfield for the Wang Tiles. We will now describe the algorithm in more detail.

We are working on a discrete grid representing the heightfield. The grid is centralized around the position $[0, 0]$, with the positions ranging between $-\frac{L_x}{2} \leq x < \frac{L_x}{2}$, $-\frac{L_z}{2} \leq z < \frac{L_z}{2}$, respectively, where L_x and L_z are the physical dimensions of the tile. The equally sized spectrum grid then represents the set of waves. At each position, the wave has a defined *wave vector*. The vectors have values $(k, l) = (2\pi x/L_x, 2\pi z/L_z)$ with values x and z being the positions of the spatial grid points. Each vector represents the wavelength of the given wave - the ratio between its components is its direction. The value of the spectrum point then defines the amplitude and the phase shift of the wave. The spectrum is *centralized*, having large wavelengths around the middle, and high frequency waves at the edges.

At the beginning of the simulation, we generate values in the Fourier domain using an underlying physically based ocean spectrum. This will be our basic spectrum at time t_0.

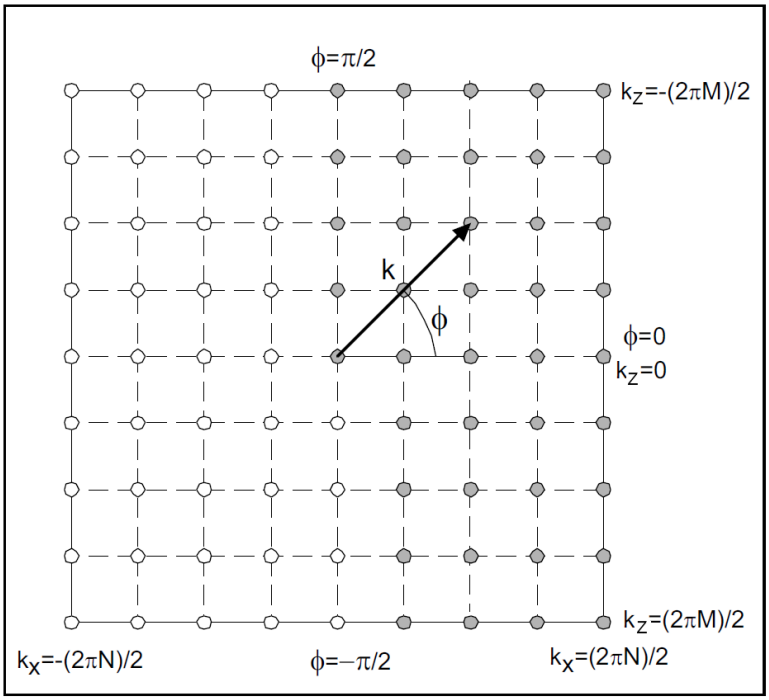
$$S_0(k, l) = \frac{1}{\sqrt{2}} \cdot (\xi_r + i\xi_i) \cdot \sqrt{P(k, l)}$$

ξ_r and ξ_i are Gaussian random numbers and $P(k, l)$ is the ocean spectrum - in case of Tessendorf's original algorithm, the *Phillips spectrum* (the exact formula is described in the implementation section). As Tessendorf himself suggests, we can use any other ocean spectrum, so we can choose a spectrum that suits our needs the best. More details about various spectra will be discussed later.

Every frame, the spatial heightfield is synthesised from a modified spectrum that incorporates the phase shift of the waves to simulate an animation.

$$S_t(k, l) = S_0(k, l) \cdot e^{i \cdot \omega(k,l) \cdot t} + S_0^*(-k, -l) \cdot e^{-i \cdot \omega(k,l) \cdot t}$$

where $S_0^*(k, l)$ is the conjugated value of the spectum value and $\omega(k, l)$ is the speed associated with this wave. The speed is computed by the physically based dispersion relationship as



Spectrum grid. Source: [51]

$$\omega(k,l) = \sqrt{g \cdot |(k,l)|}$$

where g is the gravitational constant of value 9.8.

A spatial heightfield is then computed by taking the real part of the inverse Fourier transform of the spectrum S_t. As a matter of fact, the result of the transform will have only pure real values because the input spectrum S_t is always symmetrical.

$$H(x,y,t) = \sum_{(k,l)} S_t(k,l) \cdot e^{i\cdot(kx+ly)}$$

To produce a more realistic result, Tessendorf suggests to use choppy waves with sharper peaks instead of simple cosine waves that have a too rounded shape. Despite being derived by a different approach, the resulting formula is a Gerster wave mentioned earlier.

The height of the surface is computed by the same procedure, but we also compute a displacement of the grid points.

$$D_t(x,y) = \sum_{(k,l)} -i\frac{(k,l)}{|(k,l)|} S_t(k,l) \cdot e^{i\cdot(kx+ly)}$$

The displacement field is again computed by the inverse FFT algorithm and taking the real part of the result. The final position of a point is then

$$(\tilde{x_t}, \tilde{y_t}) = (x,y) + \lambda D_t(x,y)$$

where λ is a parameter that specifies the *choppiness*. If set to 1, the waves will have the shape of the gerstner waves. But the sharpness of a gerstner wave is

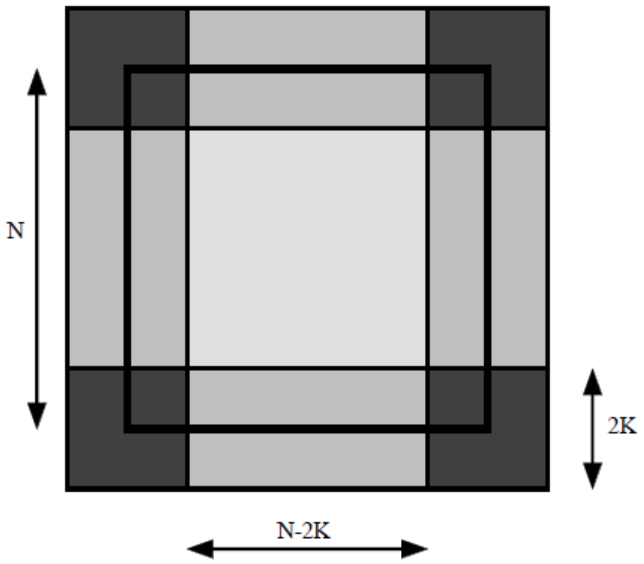
given by the ratio between the amplitude and the wavelength. So waves with low aplitudes might be still very rounded. Therefore, to enhance this effect, we can set λ to higher values, although we have to be careful not to degenerate the surface too much. On the other hand, as proposed by Tessendorf, if some positions of the surface dissect after the displacement, we can take it as an indication to produce a foam on the surface or other particle effects.

2.3 Wang Tiles

Wang Tiles is a logical formal system, proposed by a mathematician Hao Wang in 1961 [56]. A Wang Tile is an abstract tile with a specific value assigned to each border, often described as a color of the edge. A valid tiling is a tiling where all borders between the tiles match, i. e., have the same color. In addition, the tiles cannot be rotated or reflected. It has been found that using only a small set of predefined tiles, it is possible to tile the plane non-periodically.

The first one to propose using such tiling in computer graphics was Stam [50]. He used a deterministic tiling algorithm using 16 different tiles. He suggested to fill the tiles with a texture pattern in a way that the texture on adjacent edges will seamlessly match. The tiling then produces a large image using only these 16 pieces, and since the tiling is aperiodic, no visible artifacts are observed.

His method of creating the tiles was designated for textures created by filtering white noise. He proposes filling the inside of each tile with a random white noise, but to use a matching white noise in the area around each matching border. If the filter kernel is smaller than the border area, it assures that the resulting values on borders will also match, creating a seamless transition between the different values inside the tiles. The noise in the corner areas is the same for all tiles.



Obr. 2.1: Noise areas on the tile. K is the width of the kernel. Source [50]

Actually, he demonstated his method on creating an ocean surface, using the frequency filtering method by Mastin et al. [37]. However, the condition that

the filter kernel has to be small makes this method unusable for general ocean synthesis algorithms. Besides, as he performs the filtering in the spatial domain, the animation would be very costly, and he does not deal with that issue in his paper.

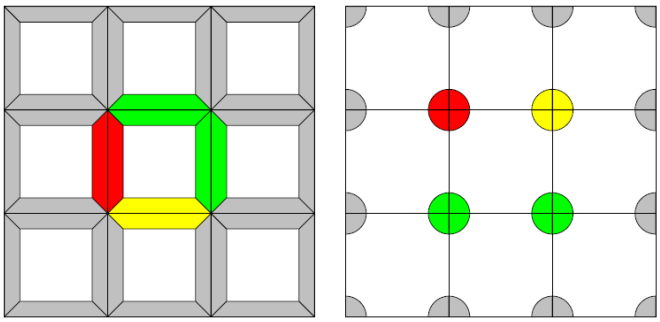
Neyret and Cani [38] proposed a similar method, but on a triangular mesh.

A more complex method was proposed by Cohen et al. [7] using only a set of 8 tiles and a new tiling algorithm. They synthesise the tiles from a given image texture by combining different patches and ensuring the seamless transition by minimal pixel cuts along the overlapping areas. The paper also deals with a problem that arises if a texture object overlaps the area around a corner, and proposes a method how to deal with corners in general, since the Wang tiling sets conditions only on the edges. However, this modification results in a much larger tileset.

Burke [2] proposes a variation of the method using a different tile synthesis approach based on the image quilting technique by Efros and Freeman [11].

The method of Cohen et al. was also used by Wei [57] who proposes a texture mapping algorithm based on Wang Tiles. Fu and Leung [14] extend the tiling to arbitrary topological surfaces.

A proposed solution to the corner problem are the *corner tiles*. Instead of matching the edges, the tiling must match the corners. This way, each tile controls all its neighbours, also the diagonal ones. Ng et al. [39] modified the method of Cohen et al. to work with the corner tiles.



Obr. 2.2: Wang tiles and corner tiles. Source [30]

Maung et al. [34] build on the method proposed by Stam and create a set of Perlin tiles. As perlin noise at a point is only dependent on a limited surrouning, if we match the guiding points at the borders, the resulting transition will also be seamless.

Wang tiles and corner tiles can also be used for object distribution, by creating tiles with poisson disc distributions that can then guide an object placement process. The first tile based method was presented by Shade et al. [49] and was a modification of a dart throwing algorithm.

The use of Wang tiles was suggested by Hiller et al. [20] who use Lloyd's relaxation for the disc distribution. This method was later adopted by Cohen et al. in their already mentioned article, where they also address this topic.

Kopf et al. [27] then presented a method to create Poisson distribution of given density in real time. The algorithm uses recursive Wang Tiles and they generate

the points progressively, so that a larger resolution is generated from the current in a smooth way.

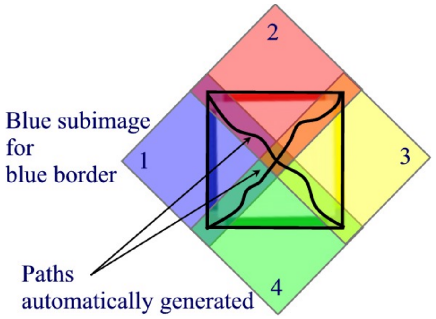
3. Wang Tiles for Frequency Generated Ocean Surfaces

As discussed in the overview of existing ocean simulation methods, one of the most popular approaches for deep ocean simulation are the frequency based techniques for their high visual quality and relatively fast computation speed based on the Fast Fourier Transform. The output of a frequency-based ocean synthesis is a heightfield that is periodic from the basic Fourier transform principle and can be used repeatedly to tile a larger area. However, such tiling suffers from visible periodicity artifacts. To address this problem, we aim to create a set of Wang Tiles, each representing an ocean heightfield. We then use these tiles to tile the given area aperiodically, producing a more realistic result with lesser cost than if we simply generated a large heightfield covering the whole area. The algorithm we chose for the wave generation is that of Tessendorf, described in detail in Section 2.2.

There are several problems we have to deal with. Each Wang Tile must have the overall wave features similar to those generated by the original algorithm, but at the same time, we also need to set fixed conditions on the boundaries so that the heightfields would seamlessly match over the edges.

The existing algorithms for creating Wang Tiles sets focus mainly on 2D image textures. They generate Wang Tiles from patches cut out from an input texture, and use pixel-cuts to blend these patches into one Wang Tile. Since the image texture on the tile is usually composed of non-continuous objects with boundaries and other edge-features, the cuts are performed along the edges to maintain these objects undistorted.

We cannot use this technique for the ocean waves. The surface of an ocean is continuous, with lower frequencies prevailing, and any height jump caused by a pixel-cut would be visually very visible. We have to change the surface slowly and continuously, to preserve its wave appearance, and to make the transition over the edges look natural.



An example of creating a Wang Tile from 4 input patches by a pixel cut. Source [7]

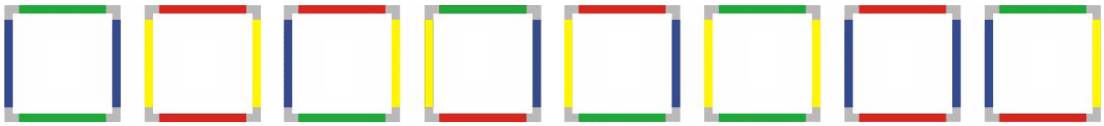
Another approach for Wang Tile generation is the one proposed by Jos Stam [50], described in detail in Section 2.3. However, as mentioned earlier, his method is only capable of producing surfaces where the value of a point is dependent only on its limited surrounding. The waves created by this method are limited in their wavelength, and the resulting surface looks less realistic than a surface produced by the Tessendorf's method.

In addition to that, we also have to keep in mind that we are producing an animation, and we have to generate the heightfields for every frame. Particulary for this reason, we want to keep the computations as effective as possible to be

able to reach interactive speeds.

3.1 Set of tiles

When talking about an aperiodic Wang Tiling, we have to specify which tiling algorithm we are using. Each algorithm is working with a predefined tileset, and describes a process of how to lay these tiles next to each other to produce a non-periodic tiling of a given area. The set of tiles we chose is the one proposed by Cohen et al. [7] consisting of 8 different tiles.



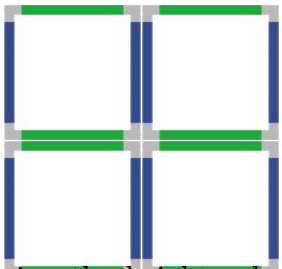
Obr. 3.1: Set of tiles. Source [7]

This set is small enough to keep the runtime speed fast. Unfortunately, troubles arise in the corner areas. However, using a more accurate *corner tiles* would result in a tileset of 16 tiles, twice as large. As we need to synthesise the heightfield every frame because of the animation, we decided to rather use less tiles and deal with the corners separately.

The tiling algorithm is very simple: As we can see, because the tiles cannot be rotated, each border has only two possible values. When creating a tiling, we tile the area from top to bottom, from left to right, so with every new tile, we have to meet two constraints: one given by the bottom border of the tile above, and one set by the right border of the tile at the left. The tileset is specific by the fact that for every combination of these two contraints there are exactly two tiles that can meet it. Therefore, when laying a new tile, we always have a choice from two possibilities. To make the tiling aperiodic, we can either choose between them by random, assuring that statistically no periodicity should appear, or we can simply follow a known aperiodic sequence.

What is more important, this tileset has other properties which make it particulary suitable for the ocean surface generation. The original frequency-based ocean tiles are periodic. If we take a look at the first two Wang Tiles, we can see that they correspond to two such ocean tiles. When put beside each other, not only do the border values match, but also the transitions over the edges are continuous and natural due to the Fourier transform periodicity.

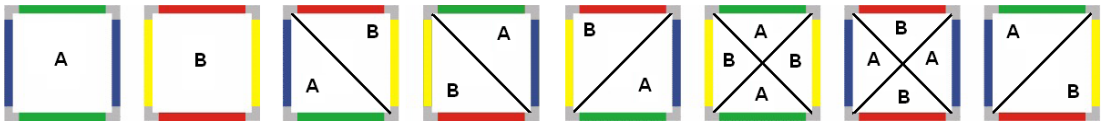
Periodic tiling using a simple ocean tile.



The tileset is also specific because of its property that each edge has only two possible values. Therefore, if we take two distinct ocean tiles as the first two Wang Tiles, the remaining tiles in the set then have borders defined by these two heightfields. Futhermore, we can imagine them as being direct combinations of the two original ocean surfaces.

This conception gives us more space to slowly change from the wave pattern around one boundary to the wave pattern on another boundary than only considering the height values at border points. Besides, some boudaries, like the green

Obr. 3.2: The other Wang Tiles as combination of two original ocean tiles.



and the blue one, are naturally connected because they belong to the same original ocean tile. Our aim is therefore to create the remaining 6 tiles as combinations of two input ocean tiles.

3.1.1 Wang Tiles generation

When creating a new tile, the simplest approach would be to combine it directly, using the heightfields of the original ocean tiles. However, we aim to preserve the overall frequency characteristics and wave appearance, so we need to take a look at individual waves.

From the principle of Tessendorf's algorithm, the ocean surface is created by computing the inverse Fourier transform of a spectrum at a given time. As we are only taking the real part of the result, each spectral component represents one long crested cosine wave. The spectrum at a given time is computed from the basic spectrum as

$$S_t(k,l) = S_0(k,l) \cdot e^{i\cdot\omega(k,l)\cdot t} + S_0^*(-k,-l) \cdot e^{-i\cdot\omega(-k,-l)\cdot t}$$

If we use this inside the inverse Fourier transform formula

$$\begin{aligned}
H(x,y) &= \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_t(k,l) \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\
H(x,y) &= \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} (S_0(k,l) \cdot e^{i\cdot\omega(k,l)\cdot t} + S_0^*(-k,-l) \cdot e^{-i\cdot\omega(-k,-l)\cdot t}) \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\
H(x,y) &= \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_0(k,l) \cdot e^{i\cdot\omega(k,l)\cdot t} \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\
&\quad + \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_0^*(-k,-l) \cdot e^{-i\cdot\omega(-k,-l)\cdot t} \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})}
\end{aligned}$$

we can see that the surface is composed of two sets of waves. One defined by the original spectrum, and one by its conjugated values. The frequency and the direction of a wave are defined by its position in the spectrum, the value (k,l). The amplitude and initial phase shift are specified by the value of the spectral component, $S_0(k,l)$. The two original ocean tiles have waves of the same frequencies and directions, but they each differ in amplitude and initial phase shift.

As we aim not to distort the wave characteristics, we will deal with each component wave separately, and create a wave preserving transition between the height constraints on the borders. More specifically, for each wave we will try

to interpolate between the two waves specified by two original spectral components $S_0^A(k,l)$ and $S_0^B(k,l)$ (or their conjugates), so that the values match on proper edges and the interpolated surface inbetween does not distort the wave appearance.

3.2 Interpolation of a sinusoid

3.2.1 1D case

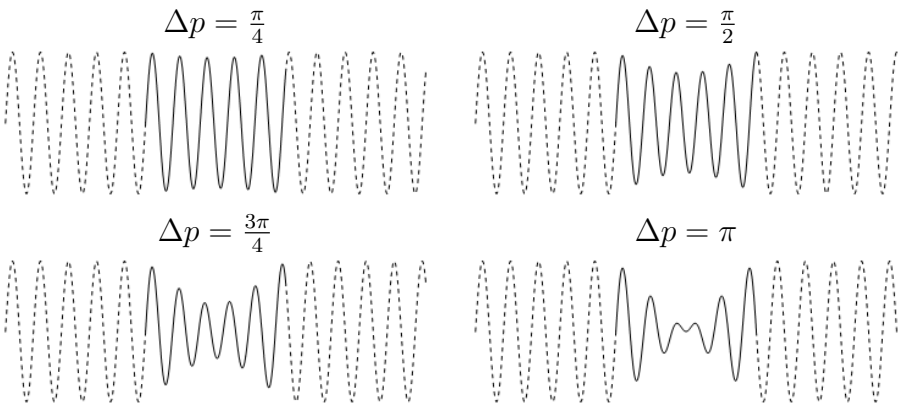
At first, we will try an interpolation of a simple 1D sinusoid. We will interpolate between two cosine waves of the same frequency and amplitude which differ only in the phase shift.

First, we use a simple linear interpolation.

$$H_s(x) = (1 - v_x) \cdot \cos(f \cdot x + p_1) + v_x \cdot \cos(f \cdot x + p_2)$$

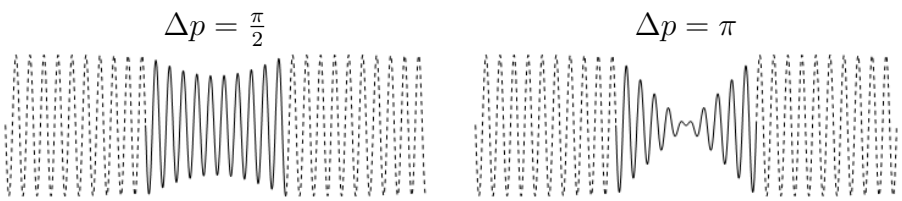
The result is again a sinusoidal surface. However, it is not a pure cosine wave as it suffers from a dampening effect when the phase shift difference gets larger - up to a point when the surface gets very distorted.

A dampening effect of the simple interpolation approach



For waves of equal amplitudes, the dampening minimum is located in the middle. In general, the position of the minimum is given by the ratio of amplitudes. The dampening effect is particulary visible for higher frequencies.

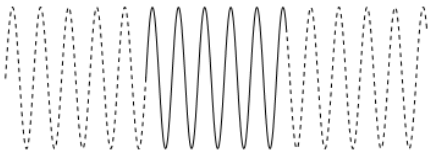
The dampening effect for higher frequencies



Let us now try a different approach. We will not interpolate the resulting values, but the phase shift instead, and evaluate the cosine function at each point.

$$H_f(x) = \cos(f \cdot x + (1 - v_x) \cdot p_1 + v_x \cdot p_2)$$

In principle, this approach means that at every point we are evaluating a slightly shifted cosine wave, and the shift continuously changes. The resulting function will therefore be another cosine wave of a different frequency. It is the original frequency stretched or folded so that the wave matches the values on both ends.



Obr. 3.3: $\Delta p = \frac{3\pi}{4}$.

The frequency of the new cosine depends on the phase shift difference and can be computed analytically. Therefore, it can be estimated for the worst case scenarios. For example, if our interpolation interval is $0 \leq x \leq 1$, i. e., $v_x = x$

$$\begin{aligned} H_f(x) &= \cos(f \cdot x + (1 - v_x) \cdot p_1 + v_x \cdot p_2) \\ H_f(x) &= \cos(f \cdot x + p_1 - x \cdot p_1 + x \cdot p_2) \\ H_f(x) &= \cos((f - p_1 + p_2) \cdot x + p_1) \\ H_f(x) &= \cos((f + \Delta p) \cdot x + p_1) \end{aligned}$$

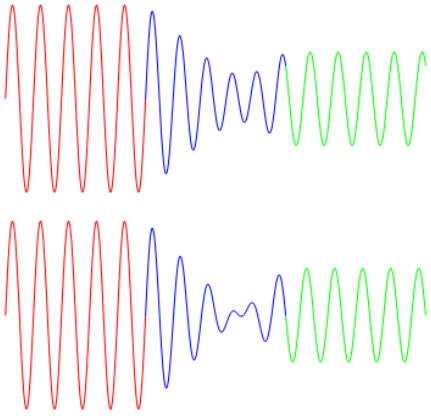
For a different interpolation interval is the formula analogical.

If we also take a difference in amplitudes into consideration, the first formula will remain practically unchanged, while the second one changes only slightly.

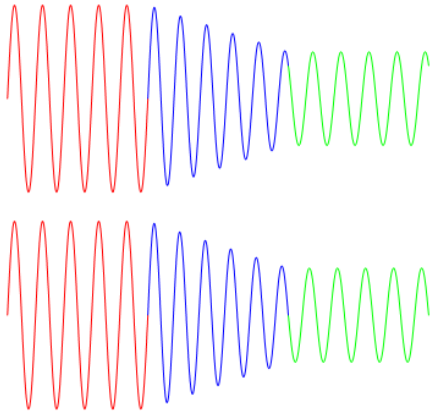
$$H_s(x) = (1 - v) \cdot A_1 \cdot \cos(f \cdot x + p_1) + v \cdot A_2 \cdot \cos(f \cdot x + p_2)$$

$$H_f(x) = ((1 - v) \cdot A_1 + v \cdot A_2) \cdot \cos(f \cdot x + (1 - v) \cdot p_1 + v \cdot p_2)$$

Simple interpolation

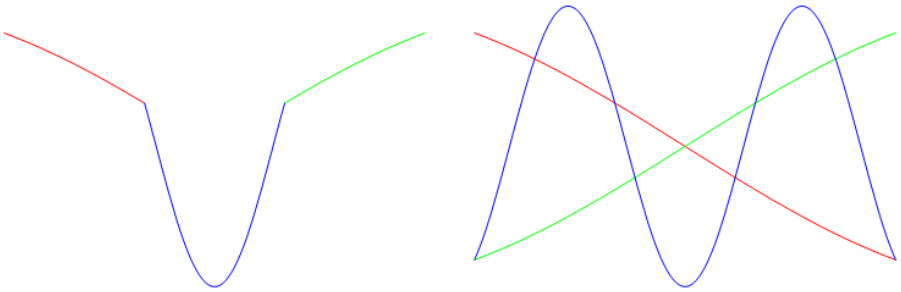


Phase-shift interpolation



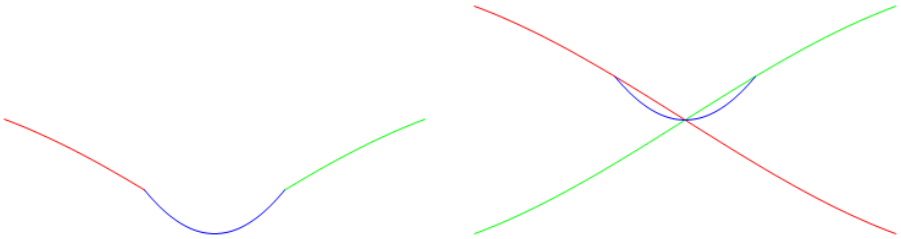
The surface of the phase-interpolated wave is again a sinusoidal wave, and the amplitude change forms a natural transition between the two surrounding cosine waves.

However, a problem arises when the wavelength of the cosine is greater than the interval of interpolation. Let us take a look at ends of two long waves and a phase-interpolated transition between them.



We can see that even though both waves have low amplitudes at the given interval, the transition created an unwanted peak, because it evaluated a cosine wave of interpolated phase shift, and by that shift the cosine value is large. Besides, the frequency of the resulting wave has changed rapidly.

On the other hand, in this case, the simple interpolation approach creates a much more plausible result as the resulting height can never exceed the original values.

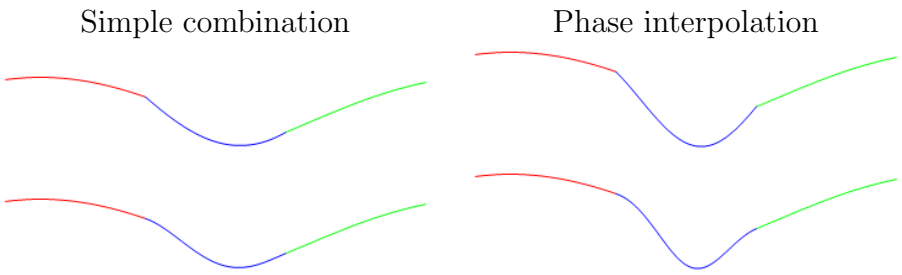


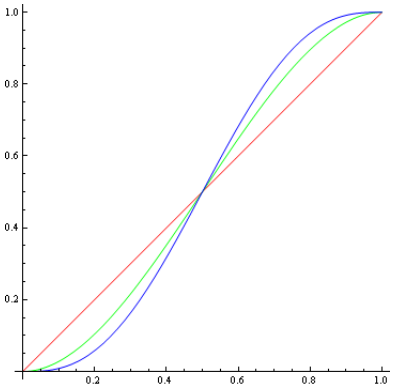
We have to take this into account when deciding which interpolation to choose. A phase-shift interpolation creates a more natural transition, however, it is unusable if the interpolation interval is too short. On the other hand, the surface of a simple interpolation gets visibly distorted if the phase shift difference exceeds $\frac{3}{4}\pi$.

A problem for a computer graphics application might arise from the fact that the derivation of the interpolated wave at its ends does not match the derivations of the original waves. Especially in case of water, a derivation step can cause a visible artifact.

This can be solved by using other intepolation function than a simple linear function. For example, the function $I_1(x) = -2x^3 + 3x^2$ preserves the first derivation at ends, while function $I_2(x) = 6x^5 - 15x^4 + 10x^3$ preserves also the second derivations. Instead of using the values v ranging from 0 to 1 in the interpolation functions, we use values $I(v)$, while v changes linearily from 0 to 1 over the interval of interpolation.

Using an interpolation preserving first derivations at ends





Shape of various interpolation functions.

However, such interpolations in case of the phase approach no longer keep the resulting surface a simple cosine wave. The wave will be stretched more at the ends than in the middle, and will not have a stable frequency. Still, especially when interpolating long waves over small interval, we have to preserve at least first derivations. This holds for both the simple interpolation approach as well as the phase interpolation.

3.2.2 2D case

We now want to use this approach for waves on a two dimensional surface. As mentioned earlier, the ocean surface is computed as a sum of long crested cosine waves, each of these waves having a different frequency, direction, amplitude, and initial phase shift. More specifically, one component wave is given by the following formula

$$H_{kl}(x,y) = S(k,l) \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})}$$

where (k,l) is computed from the position in the spectrum, and defines the frequency and direction of the wave. $S(k,l)$ is a complex number, the value of the spectrum component, and its polar form - the absolute value and the argument - specifies the amplitude and the phase shift of the wave.

Therefore, to interpolate between two waves at the same spectrum position, i. e., of the same frequency and direction, we interpolate between the complex values at this position. Again, we can either interpolate between the values directly

$$S_s = ((1-v) \cdot S_1 + v \cdot S_2)$$

or we can use the phase-interpolation approach, and interpolate between the amplitude and phase shift separately, in polar form of the complex number.

$$S_f = ((1-v) \cdot |S_1| + v \cdot |S_2|) \cdot e^{i((1-v)\cdot \arg(S_1)+v\cdot \arg(S_2))}$$

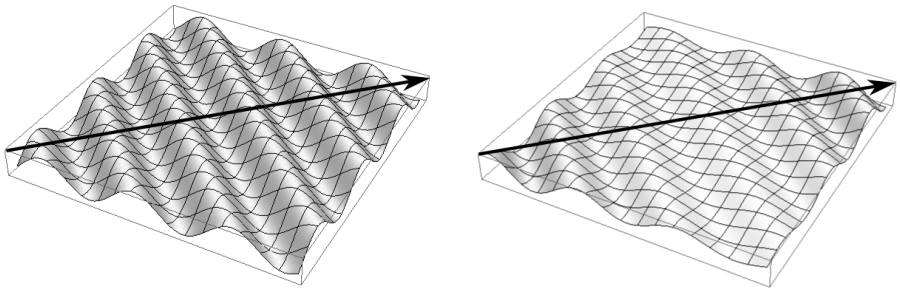
The resulting height of the wave at a point (x,y) is then computed by

$$H(x,y) = S_{xy}(k,l) \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})}$$

In addition to the 1D case, where the interpolation interval had a clear start and an end, on a two dimensional surface we also have to specify the interpolation arguments over the given area. The most simple case is to interpolate along a given direction. That means the interpolation value of a point is given as its orthogonal projection on the line - "direction" of interpolation, the beginning of the line having the value 0, and the end value 1.

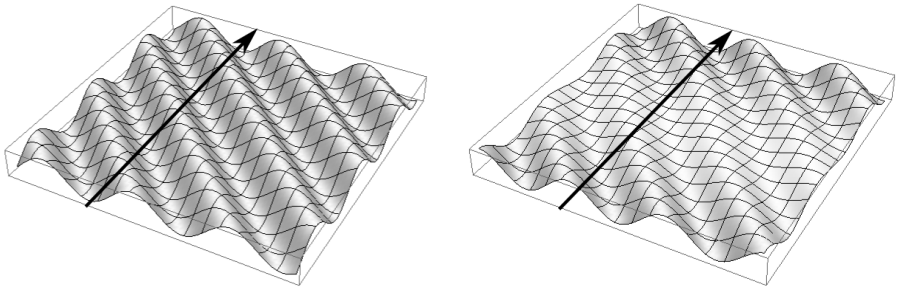
If we choose to interpolate in the direction of the wave, the long-crested wave will change exactly like its 1D case.

Phase-interpolation approach Simple combination approach



But we can choose other direction of interpolation. In such case, the dampening effect of the simple combination approach will happen along this direction.

Phase-interpolation approach Simple combination approach



Even more interesting is the effect of the phase-interpolation technique. In 1D case, the interpolation modified the frequency of the resulting wave. In 2D case, it also modifies the frequencies in both k and l components. If we interpolate in a direction that is not aligned with the wave, the resulting frequency ratio is not the same as of the original wave, i. e., the direction of the wave changes.

3.2.3 Animation

The interpolation approach works also for the animated surface. During the animation, each wave is shifted by multiplication by a constant $e^{i\cdot\omega(k,l)\cdot t} = e^{iC}$. We now show that we can perform the multiplication on the interpolated value and get the same result as if we interpolated the shifted values.

For simple combination approach is the formula after the animation shift as follows

$$S_s = ((1-v) \cdot S_1 \cdot e^{iC} + v \cdot S_2 \cdot e^{iC})$$
$$S_s = e^{iC} \cdot ((1-v) \cdot S_1 + v \cdot S_2)$$

For the phase-interpolation technique, the results again hold

$$S_f = ((1-v) \cdot |S_1| + v \cdot |S_2|) \cdot e^{i((1-v)\cdot(\arg(S_1)+iC)+v\cdot(\arg(S_2)+iC))}$$
$$S_f = ((1-v) \cdot |S_1| + v \cdot |S_2|) \cdot e^{i((1-v)\cdot\arg(S_1)+v\cdot\arg(S_2)+iC)}$$
$$S_f = ((1-v) \cdot |S_1| + v \cdot |S_2|) \cdot e^{i((1-v)\cdot\arg(S_1)+v\cdot\arg(S_2))} \cdot e^{iC}$$

This means that during the animation, we can simply move the interpolated surface and do not have to create it again for the shifted values.

3.2.4 Choppy waves

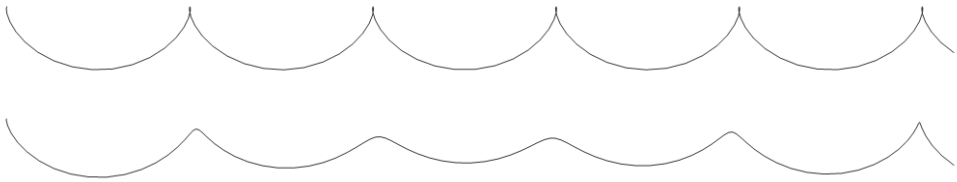
In the original ocean simulation algorithm, Tessendorf proposes a modification to obtain a more realistic choppy waves. Using this modification, in addition to the height interpolation we also interpolate the displacement vectors. The displacement for each component is given as

$$D_t(x,y) = -i\frac{(k,l)}{|(k,l)|}S_t(k,l) \cdot e^{i\cdot(kx+ly)}$$

again, taking the real part of the result. This corresponds to evaluating a long created sine wave, with the amplitude and the phase shift specified by the value $\frac{(k,l)}{|(k,l)|}S_t(k,l)$.

Let us take a look at a simplified 1D version of how the interpolated surfaces look like under this modification.

Choppy waves, phase-interpolation and simple combination approach.
$\Delta p = 3/4\pi$

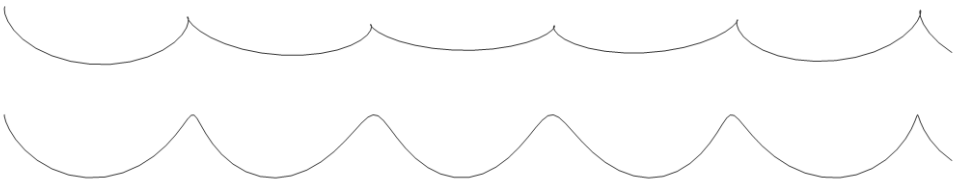


The results are analogical to the previous cases. Using a simple direct combination of both the heights and the displacement vectors results in a wave that is dampened as the phase difference gets bigger. Also, the choppy wave tops get smoother even for smaller values of phase difference.

The result of the phase-interpolation approach for both the height and the sine component of the displacement formula is again a choppy wave, stretched or folded so that the values at both ends match.

We can obtain interesting results by combining the two approaches. If we use a direct combination for heights, but phase-interpolation for the displacement

vector, the result will be a choppy dampened wave - in general a very unrealistic surface. Using the other variant, the result is a stretched wave, but the choppiness is not that sharp along its length, making tops lean on a side.



For a two dimensional wave the results are analogical.

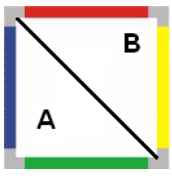
3.3 Wang Tiles

Let us now finally create a set of Wang Tiles using the techniques discussed above. We create our Wang Tiles as combinations of 2 ocean tiles, denoted A and B, with underlying basic spectra $S_0^A(k,l)$, $S_0^B(k,l)$. As explained earier, each value of a spectrum specifies two waves: one given by the original value of the spectrum component, and one given by its conjugate. Each of these waves then moves at a speed computed from its wavelegth; the conjugate wave in the negative direction.

To create a Wang Tile, we take one wave at a time, $2 \cdot |S_0|$ waves in total, and create a transition surface for each wave separately. These will be our basic surfaces at time t_0. The animation is done by changing the phase shift of the wave by multiplying it by a constant $e^{i \cdot \omega(k,l) \cdot t}$ ($e^{-i \cdot \omega(k,l) \cdot t}$ for conjugate waves).

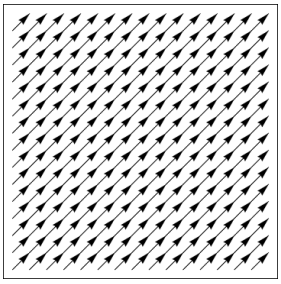
From the linear properties of the inverse Fourier transform, it is equal to perform the multiplication in the frequency domain before the transform, as in the original Tessendorf algorithm, or to multiply each point of the wave surface in the spatial domain (we then have to hold the whole complex number, not just the real part). To obtain the heightfield of a Wang Tile at a given time, we move each wave surface at the right speed, and add them all together. Using this approach, instead of a collection of long crested cosine waves specified by a spectrum, a Wang Tile is composed as a collection of waves created by combination of waves from spectra S_0^A and S_0^B.

For each wave we want to interpolate over the surface, we have to specify the interpolation argument for every point of the tile. For example, for a tile W_3 we want the wave to match the values of tile A at the left and bottom border, and the values of tile B at the right and top edge.



Wang Tile W_3

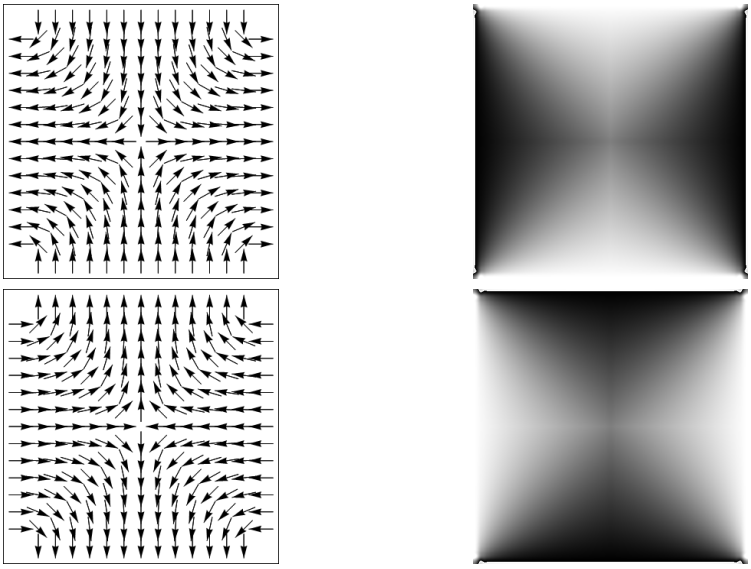
For this tile, the interpolation arguments are very straightforward. We can simply interpolate in the diagonal direction, always placing the start and the end of the interval on the edges. For the tile W_3, the interpolation directions look like this.



However, this approach does not work for tiles W_6 and W_7, as for these tiles there is no fixed direction we can interpolate along. Instead of coming up with with another interpolation function, we can take the direction for every point separately and change these directions continuously over the surface.

For tiles W_6 and W_7, the directions look like this, with the given interpolation arguments visualized as a greyscale image.

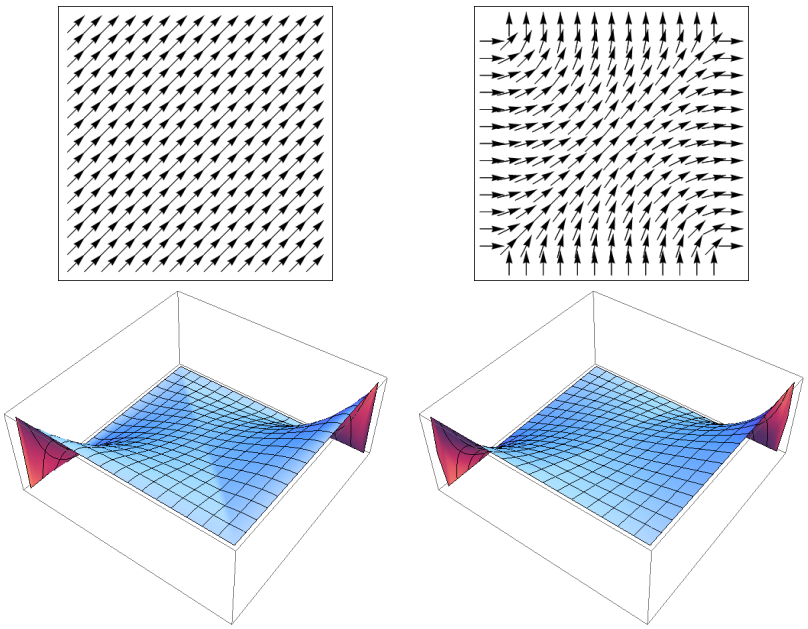
Interpolation directions Interpolation arguments over the surface



However, let us take a look at the interpolation arguments of the tile W_3 visualized as a 3D surface. We can notice a ridge going trough the middle. This can cause an edge on the interpolated wave surface, which can result in a visible artifact. If we want to make the interpolation arguments really smooth, we can use the bended direction approach even on the diagonal wang tiles. The surface directions will then look like this, and the interpolation values we get are smoother.

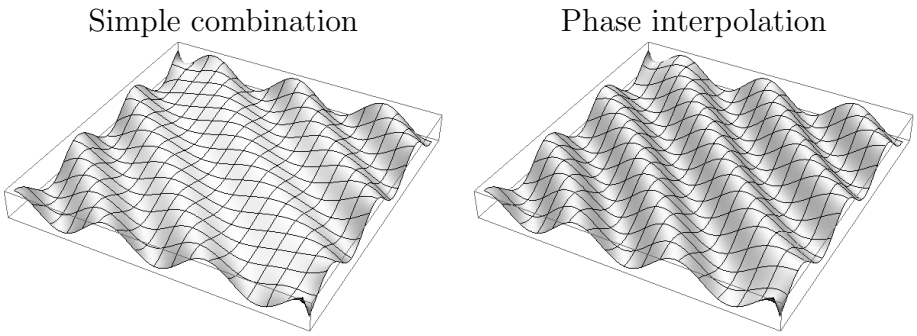
We can now observe the wave surface created by this method on the tile W_3. For the simple spatial combination, the attenuation is occuring around particular interpolation argument value which specifies the point where the two waves cancel each other out. The arguments are now spreaded, so the attenuation is not happening along one direction.

The effect of the phase-interpolation technique is different. As mentioned before, phase-interpolation along a fixed direction results in a change of the wave orientation. If we are changing the interpolation arguments directions during the



process, we end up with a bended wave. This forms a natural transition surface, but as in the 1D case, there are problems around the corner areas where the interpolation interval is very short compared to the wavelength.

Wave surface of tile W_3 $[\Delta p = 3/4\pi]$



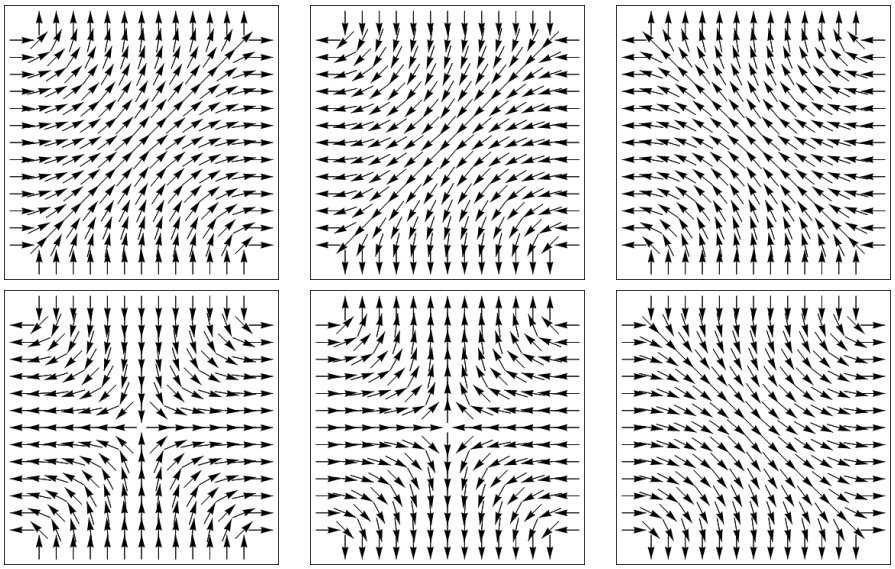
3.3.1 Basic tiles

Using the technique discussed above, each Wang Tile is defined by the set of directions we use for interpolating the component waves. In case of the Wang set we are using, the interpolation directions are visualised in the Figure 3.1.

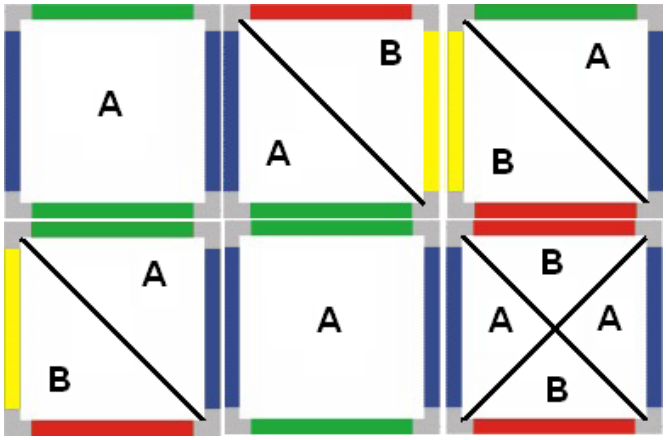
We can now create the tiles, and use them for tiling an ocean surface aperiodically. An example of such tiling is in the Figure 3.4.

However, as can be seen from the picture, the interpolation technique we are using has a major drawback. By the very definition of an interpolation, the surface is very similar to the 2 input ocean tiles. This causes the tiling to create larger

Tabuľka 3.1: Direction for Wang Tiles $W_3 - W_8$



Obr. 3.4: Tiling



areas of similar heightfield, so repeating patters appear almost as much as in the original periodic tiling.

This issue is a known problem of the Wang Tiles principle. The less input information we use to create the tile, the more it becomes visible that the covered area is composed of similar parts. Therefore, we need to create a more diverse set of tiles.

The borders of all tiles are defined by the 2 input tiles, but we want a more diverse heightfield inside.

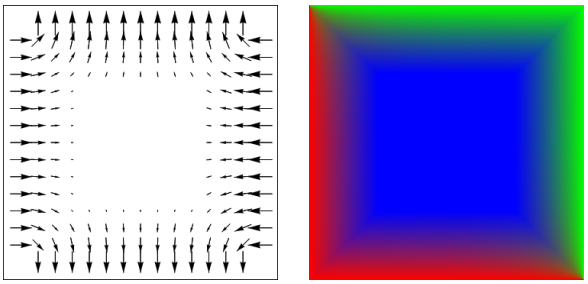
3.3.2 More complex tiles

For every tile, we take a new spectrum S_0^i that will specify the surface inside the tile. Again, we deal with each wave separately. There are multiple ways how to blend these waves with the border constraints.

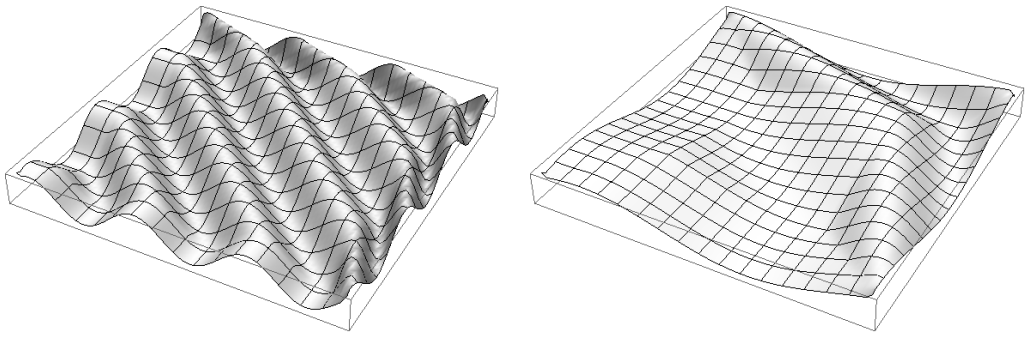
First, we can apply the classic, simple combination approach. We choose a *border size*, i. e., how large the border should be, and then interpolate on this border between the height of the wave defined by the inner spectrum S_0^i, and the height of the already interpolated S_0^A - S_0^B value, computed for the given tile by the previous approach. The inside of a tile will have waves defined entirely by the spectrum S_0^i, while the values on the border area will be a combination of all three spectra, S_0^i, S_0^A and S_0^B. However, the problem is that the interpolation interval has shortened to the size of the border area.

The resulting interpolation values can be visualised as follows. The red color represents a pure value A, the green color a pure value B, and blue the value specific for each tile. The colors on the graph show their combinations across the tile area.

Tabuľka 3.2: Direction for Wang Tiles $W_3 - W_8$



The resulting surface depends heavily on the size of the border area. If the border area is too small, we of course get very a distorted surface. Also, this approach is not suitable for long waves.

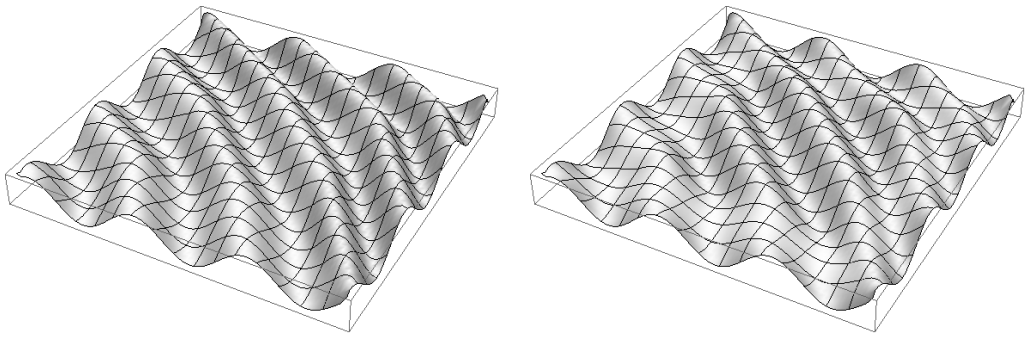


Examples of distorted surface due to a small border or a long wave

But for a moderately chosen size of the border area [we suggest 1/4 of the tile on each side], the results are very good. The phase-interpolation approach again being much better.

This is due to the fact that the heightfield inside the tile is a pure Tessendorf tile. The interpolated surface is only at the border, and as the simple interpolation method dampens the wave height, it is visible that the heightfield at the border us somewhat lower.

Phase interpolation and simple combination

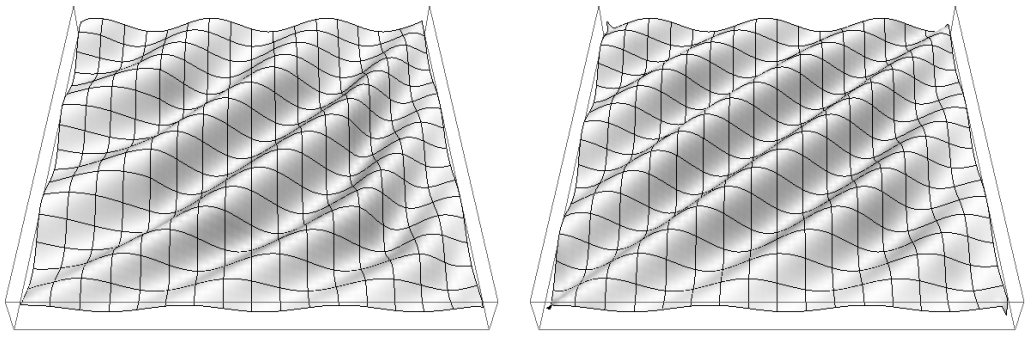


For the phase-interpolation approach, the bending of the wave also occurs only at the border. However, the bending is not as visible as the dampening.

We can also use the technique to interpolate the amplitude and the phase shift separately. Many problems arise from the fact that the interpolation interval is too short. Therefore, at every point, we can evaluate a wave that has the phase shift computed as a combination of only the two border values, given by the 2 input ocean tiles. The amplitude, however, will be computed as a combination of three values the same way as mentioned above, so amplitude inside of the tile will only depend on the separate spectrum S_0^i. Using this approach we will avoid the problems of having an interpolation interval that is too short for long waves with large phase difference.

On the other hand, the general shape of the ocean surface is mainly affected by the individual phase-shift values of the components, as the amplitudes are guided by the spectrum. These tiles will therefore be much less diverse than if we combined also the phase shift values of all three spectra. In practice, this results in a similar pattern as using just 2 input tiles, and it is therefore not practical.

Using also the phase information vs. taking the amplitudes only



4. Algorithm

Let us now describe in more detail the exact algorithm of how to use the final set of Wang Tiles to tile the plane aperiodically.

At first, we generate two basic spectra using the Tessendorf's algorithm as

$$S_0(k,l) = \frac{1}{\sqrt{2}} \cdot (\xi_r + i\xi_i) \cdot \sqrt{P(k,l)}$$

where ξ_r and ξ_i are two Gaussian random numbers, and $P(k,l)$ is a physical ocean spectrum dependent on the wind, the area size, or other parameters. We can choose any underlying ocean spectrum, but in our experiments, we implemented the *Phillips spectrum* as proposed by Tessendorf, and the empirical *JONSWAP* spectrum.

As we can see, the difference between the two spectra S_0^A and S_0^B is caused by the random numbers.

The first two Wang Tiles of the set, W_1, and W_2 are identical with two ocean tiles synthesised from these spectra by the Tessendorf method. The other Wang Tiles are then generated as follows.

For each tile, we generate a new basic spectrum S_0^i in the same way. This spectrum will specify the waves inside the tile.

We then create the Wang Tile surface at time t_0. For each spectrum component at the position (k,l), we combine the complex values of $S_0^A(k,l)$, $S_0^B(k,l)$ and $S_0^i(k,l)$ in any way described above, and evaluate the inverse fourier result for this wave at each point (x,y) at the border area of the tile. We do the same for the conjugate values of the spectra. We then store all these surfaces as the basic waves at time t_0. Note that we store the complex value of the fourier result, and not only its real part to be able to perform the animation.

A heightfield at a given time t is then computed by moving each basic surface at the right speed, depending on the wavelength of the wave it represents. We move the surface by multiplying each height point by $e^{i\cdot\omega(k,l)\cdot t}$ ($e^{-i\cdot\omega(-k,-l)\cdot t}$ for conjugate waves), i. e., changing the phase of the complex surface values. The final heightfield is then obtained by adding all waves together, and taking only the real part of the complex result.

We then use these tiles to tile the plane using the algorithm mentioned in Section 3.1.

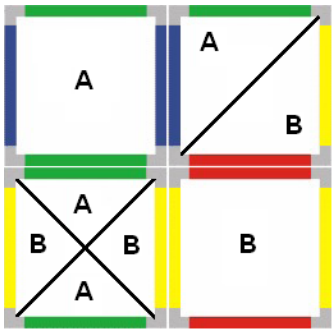
4.1 Corners

From the definition of Wang Tiles in general, the tiling algorithm does not cover the problem that might arise from the fact that two distinct tiles are touching at corners. A theoretical solution to this problem are the *corner tiles*, but the tileset of this approach would be twice as big, and we want to keep the number of different tiles low for the reason of computational speed.

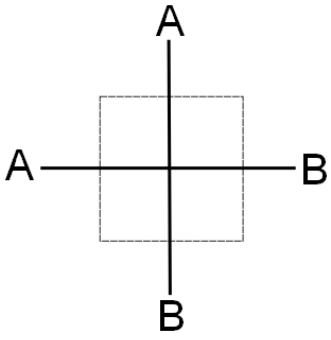
The corners are problematic also because in general, the value at most corner points is not well defined. Besides, the heightfield around the corner area is often

distorted because of the very limited interpolation interval. A surface that has to match the value of two different edges close to each other is problematic.

Tiles meeting like this cause problems



Corner with the given sides meeting

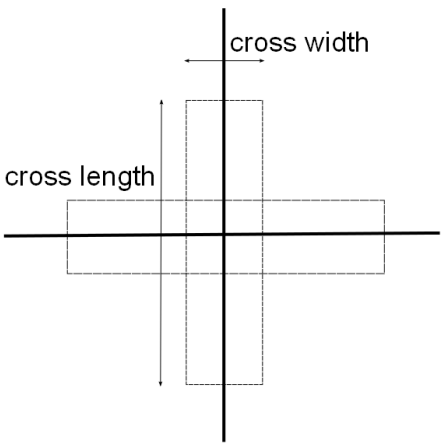


We have decided to resolve the issue in a post processing step. We work on a small area around the corners where the 4 tiles meet, and modify the heightfield to supress the distortions creating visible artifacts. We treat the long waves and the short waves separately, so that we can smooth out the surface of the big waves, while not supressing or destroying the ripples created by the short waves.

4.1.1 Low Frequencies

Low frequency components of the spectrum creating long waves cause several problems. First, around the corner areas, the surface is largely distorded since the interpolation interval is very small in comparison to the long wavelength. Also, for the waves running across the tile, there are additional problems when two distinct tiles touch at the corners, since the waves have no transition space at all.

Some of the long waves might have high amplitudes given by the underlying physical ocean spectra, so the problematic surface is even more visible. To deal with the this problem, we have decided to smooth out the height values produced by long waves in a cross shaped area around the corners. We blur the values by a convolution with a smoothing kernel.



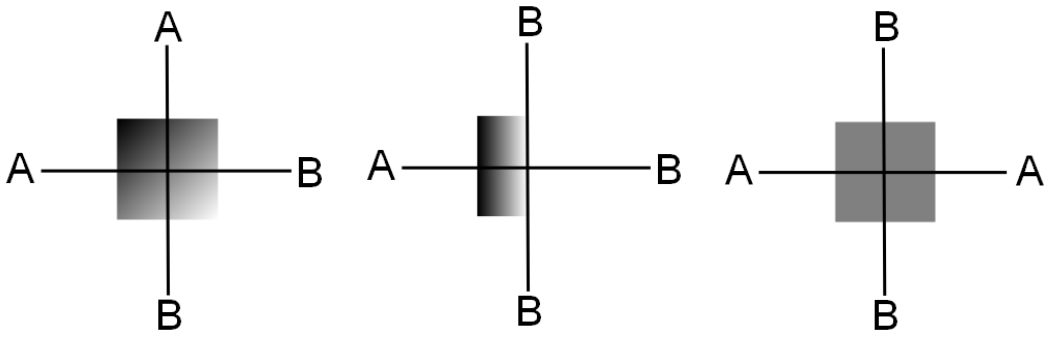
We are working on a cross shaped area so that we only consider the most distorted heights around the meeting area. We leave the rest of the heights as they are for a computational speed reasons.

4.1.2 High Frequencies

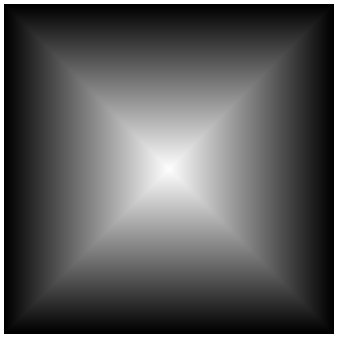
For high frequencies, the problematic area is smaller, but there still are troubles with undefined values at some corner points and again with the cross-tile waves. To handle this, we distinguish 16 cases of how the tiles can meet: all 2^4 combinations.

Two cases are simple - if all sides are homogenous, nothing has to be done. For other corner types, we again create interpolation arguments over the area, but in a very simplified way. The corners where the tiles meet diagonally, we merge them in the diagonal fashion. When the corner has only one side different, we interpolate in that direction. If there tiles meet in a cross shape, we simply use the mean value over the whole area. The basic concept is that we are prolonging the heightfield of one tile in the area of another, and blend it together.

Three basic types how the tiles can meet



At the rendering step, we set the corner height using the interpolation values and the current heights of tile A and tile B. We of course cannot use these values directly in the heightfield, since the heights at the edge of the corner area clearly do not match the values of the heightfield. Therefore, we blend the values in a pyramidal way.



This approach is of course far from ideal. The reason we use it is because it is very simple, and provides reasonably good results. However, the algorithm would definitely benefit from a more sophisticated approach.

4.2 Complexity

Let us now take a look at the complexity of the whole algorithm. First, consider a straightforward approach to compute the heightfield of a tile. For every spectrum component, we store the whole surface at the borders sepatately - in fact, two surfaces because of the conjugate waves. At every frame, we move each surface by multiplying every point by a given complex number, and add the surfaces together. Even though the inside of the tile can be computed by the FFT, the border has to be proportially big to the resultion, if we don't want to distort the surface too much. For a tile of size $N \times N$, with the border of width $N/4$, this still results in an overall complexity of $O(N^4)$. In comparison to Tessendorf's algorithm with a complexity of $O(N^2 \log N)$, this approach is very inefficient. In fact, the complexity is the same as if we created the surface from scratch at every frame, with input being the spectra of tiles A, B and the inner tile at the given time.

However, we can take a look at how the individual wave surfaces are composed. If we compose the surface simply by combining the complex value of the spectrum component directly

$$S_0^S(k,l) = (1-v) \cdot S_0^A + v \cdot S_0^B$$

and evaluate the surface, we can see that this interpolation can be done for all the waves together.

$$\begin{aligned}
H(x,y) &= \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_0^S(k,l) \cdot e^{i\cdot\omega(k,l)\cdot t} \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\
&+ \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_0^{S*}(-k,-l) \cdot e^{-i\cdot\omega(-k,-l)\cdot t} \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\
H(x,y) &= \sum_{k=0}^{N-1}\sum_{l=0}^{M-1}((1-v)\cdot S_0^A(k,l) + v \cdot S_0^B(k,l)) \cdot e^{i\cdot\omega(k,l)\cdot t} \\
&\cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\
&+ \sum_{k=0}^{N-1}\sum_{l=0}^{M-1}((1-v)\cdot S_0^{A*}(-k,-l) + v \cdot S_0^{B*}(-k,-l)) \cdot e^{-i\cdot\omega(-k,-l)\cdot t} \\
&\cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\
H(x,y) &= (1-v)\cdot[\sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_0^A(k,l) \cdot e^{i\cdot\omega(k,l)\cdot t} \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\
&+ \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_0^{A*}(-k,-l) \cdot e^{-i\cdot\omega(-k,-l)\cdot t} \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})}] \\
&+ v\cdot[\sum_{k=0}^{N-1}\sum_{l=0}^{M-1} B_0^A(k,l) \cdot e^{i\cdot\omega(k,l)\cdot t} \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\
&+ \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_0^{B*}(-k,-l) \cdot e^{-i\cdot\omega(-k,-l)\cdot t} \cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})}] \\
H(x,y) &= (1-v)\cdot H_A(x,y) + v \cdot H_B(x,y)
\end{aligned}$$

Using this approach, we really are interpolating directly the resulting height-fields of the ocean tiles A, B and a tile given by the spectrum S_i, respectively. This is the simplest approach possible, and for many cases gives reasonable results. As discussed above, the artifact it suffers from is the dampening effect of individual waves, and smoothing out the tops of the choppy waves. Also, if the waves are of opposite phase, there are areas where these waves are completely flattened. However, since the dampening occurs in different areas for different waves - given by the ratio of their amplitudes - in overall, for a heightfield composed of many waves of low amplitudes, the effect is not that noticable. We will discuss this in more detail in Chapter 6.

The phase-interpolation approach, i. e., combining the two spectral values in their polar form - amplitude and phase separately - eliminates these problems.

However, we have to perform the interpolation for each wave separately, as there is no easy way how to get the individual component values from the already summed up heightfield of tiles A and B.

This could mean that we really do have to create and store each component surface separately. But if we take a look at how we compute the surface for a given time during the animation

$$\begin{aligned} H(x,y) = &\sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_0^F(k,l)\cdot e^{i\cdot\omega(k,l)\cdot t}\cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \\ &+ \sum_{k=0}^{N-1}\sum_{l=0}^{M-1} S_0^{F*}(-k,-l)\cdot e^{-i\cdot\omega(-k,-l)\cdot t}\cdot e^{i2\pi(\frac{kx}{N}+\frac{ly}{M})} \end{aligned} \tag{4.1}$$

we can see that we can move all the component waves of the same speed together, by taking out the part $e^{i\cdot\omega(k,l)\cdot t}$ in front of summation. We can add the basic surfaces of the same speed together in advance, and during the animation move and sum only these partial heightfields, resulting in less computations per frame.

The speed of the wave is given by

$$\omega(k,l) = \sqrt{|(k,l)|\cdot g}$$

.

This means that there are always 8 component waves of the same speed, 4 original ones, and 4 of conjugate values. This does not change the complexity of the algorithm significantly. But if we cluster the speeds into less values, we will be able to compute the final heightfield more efficiently.

The speed is only dependent on the length of the wave. One proposal is to take the Manhattan metric of the wavelength computation instead of the Euclidean.

$$\tilde{w} = |(k,l)| = (k+l)$$

instead of

$$\tilde{w} = |(k,l)| = \sqrt{k^2+l^2}$$

In this metric, there are only $O(N)$ distinct values for a tile of size $N\times N$, instead of $O(N^2)$. The final complexity of the algorithm is then $O(N^2\cdot N) = O(N^3)$. In our experiments we saw no visible quality difference in the resulting animation in comparison with the proper Euclidean metric.

Another approach would be to use the Tessendorf's algorithm for creating a repeating animation, which clusters some speeds together. More specifically, to create a repeating animation, it is needed that all speeds are a multiply of the basic frequency

$$\omega_0 = \frac{2\pi}{T}$$

where T is the time after which the animation should repeat. The individual wave speeds are then computed as

$$\tilde{\omega}(k,l) = \lfloor \frac{\omega(k,l)}{\omega_0} \rfloor \cdot \omega_0$$

This creates a clustering of the continuous speed function into discrete intervals. The number of clusters is of course dependent on the time parameter T, which also affects the outcoming quality. If it is too large, the number of clusters will be large. If it is too small, the animation will repeat after a short time, producing a very negative visual effect.

4.3 GPU implementation

Another way of dealing with the issue of runtime speed is to use the advantages of parallel computations on GPU. We tried a simple straghtforward implementation, but the results were very unsatisfiying, being able to increment the speed only by approximately 20%.

The implementation was the following. First, we copied all the basic surfaces into the graphics card global memory. Then, at each simulation step, we created one core for each basic surface and position combination. The thread then fetched the value from the global memory, moved the wave surface according to the current time, and performed atomic add at the output field on its position.

The core clustering was done so that the global memory accesses will be coherent. Additionally, experiments showed that the atomic add did not slow down the computations significantly.

The reason the GPU implementation didn't gain a better performance raise is in our opinion the fact that at each simulation step, there are many fetches from the expensive global memory. Additionally, each value is used only once from the principle, so no computations can benefit from shared memory.

Another approach would be to create the heightfield entirely on GPU, computing the interpolated values every frame from scratch. This way, we would have to fetch only the three complex values used for generating the wave surface, and not having to look at each position separately. However, this means that we cannot pre-sum the component surfaces of the same speed, and we end up with a much larger complexity, in the end.

However, not being experienced in this area, there might be a proper GPU implementation what would be able to accelerate the algorithm much more significantly, and we encourage futher research in this area.

5. Implementation

As an environment for the ocean simulation visualisation we created a simple application where the user can choose simulation parameters and observe the visual properties of different approaches to the ocean Wang Tiles generation.

The user interface is written in Qt framework version 4.8.5 and the rendering is done by OpenGL. Also, we use the FFTW library [60] to compute the inverse Fast Fourier Transform. We provide binaries for Windows 32-bit systems, but the source code can be compiled under Linux and Mac OS as well, with the libraries being installed.

The Wang Tiles techniques implemented are simple combinations of the final heightfield values as well as the surface created by interpolating the phase shift of individual waves.

For both techniques, the Wang Tiles can be generated using three input spectra (two for the borders, and one for the inside) or only the two border-defining spectra. This option is there for showing that if we only generate the tiles as simple A/B combinations, repeating patterns appear.

We also implemented the original Tessendorf periodic tiling for visual comparison.

5.1 User Guide

The application is a simple desktop application. User chooses the parameters, and after pressing the *start* button, the visualisation begins. In case of the phase-interpolation technique, this means that the component surfaces have to be generated first, so it might take some time.

The user can also pause the visualisation at any moment.

Individual parameters either specify the general ocean simulation, or are used during the Wang Tiles generation. Navigation in the 3D space is done by keyboard and mouse. The user can also change specific visualisation options.

5.1.1 Parameters

1. Main Options

   A technique to use: either the original periodic tiling or one of the Wang Tiles techniques mentioned above.

2. Ocean Simulation Parameters

   The underlying ocean spectrum *Phillips* or *JONSWAP*.

   Grid size The size of a tile. The application only supports square grids of even sizes. For better performance, the value should be a power of 2, because of the inverse FFT we use to compute the basic heightfield.

   Water length The size of the ocean tile in meters.

   Wind speed Speed of wind in *m/s*.

Wind direction A vector specifing the direction of the wind.

choppiness A value specifying choppiness of waves, as in Tessendorf's algorithm.

phase-interpolation displacement / simple displacement Whether to compute the displacement vectors using the more expensive phase-interpolation that preserves the choppiness or just a simple combination.

3. Advanced Parameters

   Amplitude constant A constant specifying the amplitudes of waves in case we are using the Phillips spectrum.

   Fetch length The length of the fetch area in case we are using JONSWAP spectrum.

4. Wang Tiling Parameters

   Tiling sizes The tiling grid: how many tiles we put in each direction.

   Inner / SimpleAB Whether to use just simple 2 tiles combinations, or to incorporate the inner spectrum.

   Border size Size of the border, where we interpolate between the inner heightfield, and the values given by edges. The larger, the more coherent the result will be, but also, the less diverse the tiles will be.

   Wave cancel The treshold of the largest waves that will be removed (set to zero).

   Wave desynch The treshold of the largest waves that should be *out of phase*

5. Advanced Wang algorithm parameters

   Low frequency treshold The treshold specifying which frequencies are *low*, and should be smoothed at the cross areas around corners.

   Corner size The size of the corner square that blends the high frequencies around corners. Note that the implementation only supports corners of sizes less than the cross-length.

   Cross size The length and width of the cross area smoothing out the low frequencies around corners.

6. Animation

   Start button Start the simulation. In case of phase-interpolation technique, it generates the component surfaces first which might take some time.

   Pause button Pauses the animation at the given time for the user to observe certain features.

   Stop button Stops the animation. In case of phase-interpolation technique, the component surfaces are deleted, so restart would have to generate them again.

5.1.2 Navigation

Even thought the navigation is very simple, we have tried to make it as intuitive as possible. The user moves through the 3D space by either using the mouse, or with combination of keyboard. The basic panning over the place is done by keyboard arrows. The keys P and L move the camera in vertical direction.

The mouse is used to rotate the ocean surface. When using mouse while pressing *Ctrl* button, the user can drag and drop the ocean surface.

5.1.3 Visualisation options

There are three visualisation options user can change.

key Q Changes the surface shader. The user can choose between a diffuse surface, to better observe the little details, or a simple water shader.

key V Switches between a normal and a wireframe rendering.

key M Renders the lines showing the individual ocean tiles.

5.2 Programmer Documentation

The program is written in C++ language, in Visual Studio Express 2010 environment, using Qt framework for the user interface, and a combination of Qt and OpenGL for rendering. We use the FFTW library for the inverse FFT transform. We also use an external C++ source code for the *Gamma function* implementation [59].

We enclose the Qt .libs and .dlls as well as the ones of FFTW. It is important to use these .dlls and .libs, at least for Qt, as there was a bug in the official ones.

The code is well-commented, and the list of classes, their short description, and their methods is enclosed on CD as a Doxygen documentation. We now describe the most important implementation details of the application.

5.2.1 Qt integration

We have decided for the newest Qt 4.8.5 framework as the user interface platform for its portability. This version has a fully intergrated OpenGL support, so we do not have to use an additional framework for the visualisation. However, we did not want to create a fully integrated Qt application, so most of the classes are written in plain C++. The Qt related classes, such as the Main Window, or the rendering class, are all transformed to a simple C++ source codes using the Qt tools.

The Main Window layout was created using Qt designer. We then used the **uic** utility to generate a C++ header file with these layout informations we then included into the project. On the CD, we also provide the original .ui file, but it is not used for the compilation.

For internal reasons of Qt framework, any class containing the Q_OBJECT macro has to have an additional .cpp file generated by the Meta-Object compiler - **MOC**. This Qt utility handles the Qt C++ extensions, mainly the signal and

slot mechanism, runtime type information and the dynamic property system. The utility is executed on the header files containing the class declaration, and the output .cpp file is simply linked with the rest of application.

The OpenGL, along with the GLSL shaders is fully included in the Qt framework. In particular, in the QOpenGL wrapper based on the OpenGL Embedded Systems 2.0 specification. The framework provides special high-level wrappers to most of the OpenGL functions, but is capable of also calling the plain OpenGL.

However, during the developement, we have found a major bug in the Release version of QOpenGL 4.8.5 for Windows, namely in the pre-built development library we have downloaded from the official site. The problem which causes the application to simply crash, has been found a long time ago in earlier versions, and has been reported. However, it seems that the solution was still not incorporated to the pre-built libraries. As the solution was quite simple: it was only needed to redefine one macro in one header file, and some other minor changes, we have re-built the QOpenGL library, and provide the .lib and .dll files on the CD to replace the official version when using with our application. We have also reported the persisting problem to Qt.

5.2.2 Wang Tiles

Let us know describe the logic of the Wang Tiles implementation. All Wang Tiles are represented as a subclass of an abstract class *WangTile*. The first two wang tiles are implemented as *OriginalTiles* with heights computed by the original Tessendorf algorithm. The remaining 6 tiles then hold pointers to these tiles, and optionally also the third, "inner", tile representing the heightfield inside the tile.

Apart from that, each wang tile also holds the surface structure: a list of *basic surfaces*, to use in the phase-interpolation technique. The basic surfaces have to be precomputed in a way described earlier. We compute a transition surface for a each component wave, and sum up those of the same speed. The structure then holds the list of component surfaces for each speed.

During the animation, the computations depend on the technique the user has chosen. At first, we compute the heights of the two original tiles, A and B, by the original approach, computing the inverse FFT. Then we compute the heightfields of the rest of the Wang Tiles.

In case of a simple combination approach, we evaluate the heightfield of the ocean tile representing the inside area, and the final heightfield is then simply computed as a combination of heightfields of the three ocean tiles in a way decribed earlier. When using this technique, we do not precompute the basic surfaces structure, as this is only used for the phase-interpolation technique.

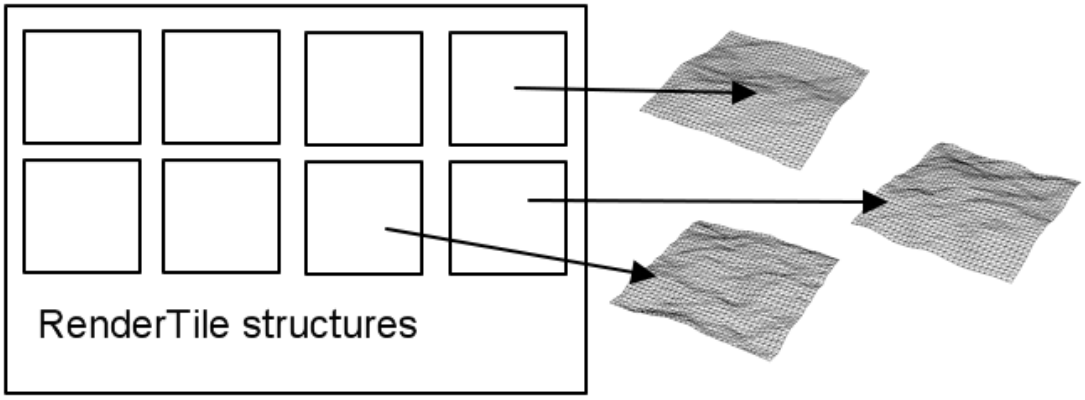
In case of the phase-interpolation approach, both the one where the inner spectrum only affects the amplitudes as well as the one where it represents the actual heightfield, the animation algorithm is the same. Before the animation, we precompute the basic surfaces: we combine individual waves in ways described earlier. At the animation, we then only move these values, and sum them all up in a heightfield. The precomputation step is very costly, and the algorithm is therefore halted at the beginning for some time, but in the user interface we provide a progress bar showing the process speed. Note that we only compute the height values at the border areas. The heightfield inside of the tile is computed

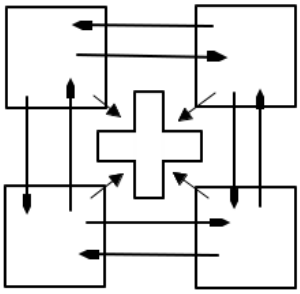
by the basic inverse FFT approach.

5.2.3 Ocean rendering

Since the principle of Wang Tiles is to be able to easily tile large areas, we do not want to create the vertex field for every single tile we are rendering. Instead, we have only one vertex buffer for each Wang Tile, and the rendering is done by first modifying values at some positions, then changing the *ModelView* matrix to incorporate a translation, and finally, rendering this particular part of the ocean.

For each Wang Tile, we create a structure *RenderTile*, holding the vertex buffer structure. There will only be 8 VBOs to create the whole ocean scene.





Four ParticularTiles meet. The cross-shaped area structure is created for every meeting.

For each ocean tile in our tiling, we create a structure *ParticularRenderTile*, representing tile already placed in the ocean. It holds the reference of the *RenderTile*, as well as pointers to all four neightbours (or less if it is on the border).

We also create 14 instances of corner areas, one for every non-trivial high frequency corner case. Individual ParticularRenderTiles then hold pointers to one of these instances for every corner, or the value *NULL* if that corner is a trivial case of homogeneous tiles meeting. For simpler implementation, we also avoid the cases when the tiles are on the border, so the corner is formed by only 2 tiles. In such case, we also represent this corner as *NULL* and ignore it during rendering.

The cross shaped areas over which we smooth out the low frequencies are created for every corner meeting. ParticularRenderTile structure holds a reference to the given cross shaped area at each corner.

The animation then proceeds as follows. At first, we compute the heightfield of all 8 Wang Tiles. We compute two heightfields: one given by the low frequency waves and one synthesised from the high frequency waves, their distinction given by the parameter **Low frequency treshold**. We update the values in VBO as a sum of these two heightfields. The normals are computed by a simple finite

differences approach, because we did not want to slow down the algorithm by computing normals analytically.

We then compute the heights in the corner areas using the heightfield of high frequencies of tiles A and B, evaluating the 14 cases. Also, we fill the current low frequency height values to each cross shaped area field and smooth them by filtering.

During the rendering step, we render each ocean tile separately. We modify the VBO in the corner areas to incorporate heightfield changes for this particular instance - both caused by the low frequency smoothing and the high frequency corner cases. We modify both heights and normals in these areas. In addition, we also have to modify the normals of vertices at the border because those can only be computed from the heightfield of the neighbouring tile borders.

In the end, we only change the *ModelView* matrix, render this vertex field, and move to another tile.

Since different ocean parts share the same vertex structures, we have to render them right after we changed the values for that particular part. So at every step, we modify the heightfield and render it straight away. If we modified two identical Wang Tiles tiles at once, we would distort the values of the first one, as even though they are *theoretically* the same, they each have different values in the corner areas.

Note that because of the clustering of wave surfaces according to their wave-speed, the *Low frequency treshold* holds also for the Manhattan metric, and not for the real wavelength. However, this does not introduce that large misconception and the advantage of working with the whole pre-summed surfaces pays off.

5.2.4 Water shader

To obtain a realistic looking ocean surface, we have implemented a simple water material fragment shader. Within the shader, we have access to the Camera position, and the World coordinate of the current fragment position. For each fragment, we compute the angle between the normal vector, and the eye-point vector, and use a Fresnel formula to compute the ratio between how much light is contributed from the reflection, and how much light is from the refracted ray.

The reflected light contributes with a sky color which we set to light grey as a default. To simulate the effect of a water volume, the refracted light contributes with color dependent on the refraction angle. If the angle is small, i. e., the ray travels near the sea surface, it is lighter because the light accumulated along the path near the surface. If the angle is larger, the color is darker, since there is not much light deep in the water. The exact value is computed by a linear interpolation between the light and a dark color of the water.

The resulting effect is of course not physically realistic, but is visually plausible.

6. Experiments

We can now take a look at various approaches for the Wang Tiles synthesis and their usability in practical applications.

We implemented two techniques mentioned earlier. A simple combination of the final heightfields of the original ocean tiles as well as the phase-interpolation approach.

It needs to be noted that during the phase interpolation, we still use the simple interpolation method around the corner areas. This is to prevent the unwanted peaks this interpolation otherwise produces.

We have tried the algorithm on tiles of sizes 32×32, and 64×64. For the simple combination approach is the runtime speed much faster than for the phase interpolation technique, so we have also tested the tiles of sizes 96×96. The physical resolution varies from grid points being between 1 and 2 meters apart. We did not work with a larger resolution, as we don't incorporate any LOD approach, so the high frequencies would create an alias when overlooking the large tiled area.

The tiling size we tried is 7×7. Larger tilings are of course also possible, but we soon had to deal with the problem of aliasing as the pixel resolution became too small for the amount of vertices.

6.1 Ocean spectra

The visual quality of the ocean surface, both the surface of individual tiles and the general appearance of the whole area, is heavily dependent on the underlying physical ocean spectrum we use for the wave synthesis. In our experiments we implemented two ocean spectra: *Phillips spectrum* and *JONSWAP*.

6.1.1 Phillips

Phillips spectrum is a theoretical spectrum used in the original algorithm proposed by Jerry Tessendorf. The spectrum is defined as

$$P(k,l) = A \cdot \frac{\exp(-1/(|(k,l)| \cdot L)^2)}{|(k,l)|^4} \cdot |(\tilde{k,l}) \cdot (\tilde{w_x,w_z})|^2$$

where L is the largest wave created by the specified wind speed V. g is the gravitational constant of value 9.8 m/s.

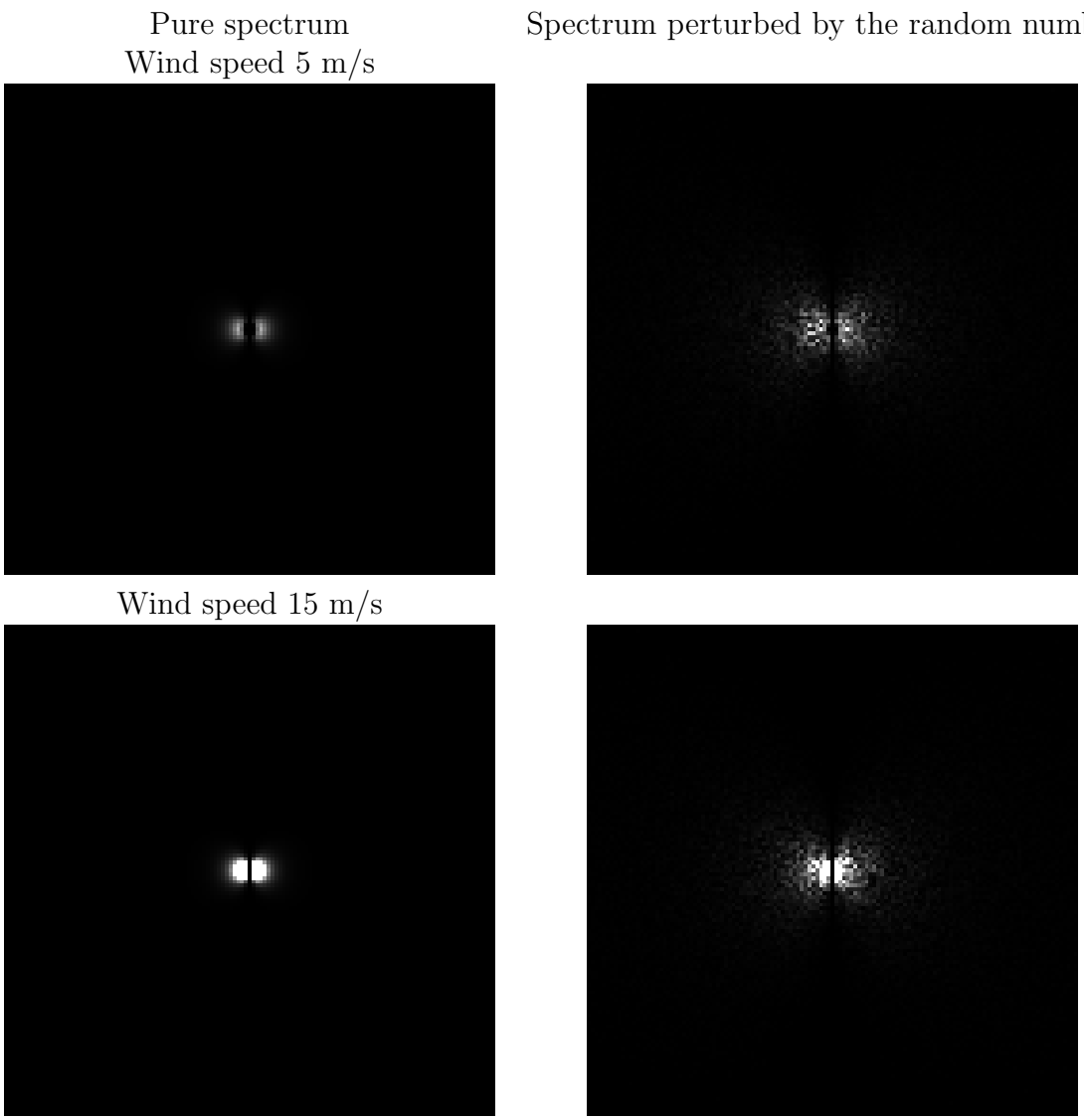
$$L = \frac{V^2}{g}$$

.

(w_x, w_z) is the wind direction and $(\tilde{k,l})$, $(\tilde{w_x,w_z})$ are the normalized vectors. A is a constant specifying amplitudes of waves. In the original paper no default value is provided and upon inspecting various implementation we came to a conclusion that it is up to user to define this value, to create the visual effect they are aiming for. This constant is therefore used to simply *stretch* the wave surface vertically.

The spectrum for a given wind speed looks like this.

The Phillips spectrum. The size of area is $64m^2$.

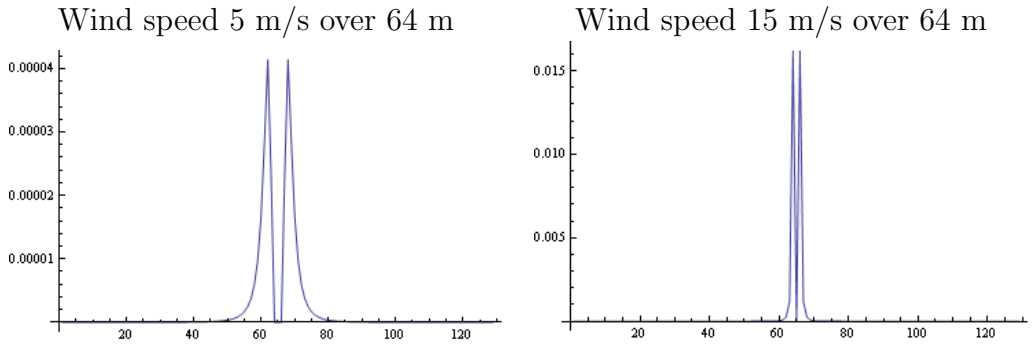


We can see that as the wind speed increases, most of the energy is embraced by few of the lowest frequencies, and the distribution rapidly drops as the waves get shorter. Another interesting aspect is that only the waves perpendicular to the wind get supressed, but the waves going directly opposite the wind are as strong as the waves running along the wind. The final effect for the animation is therefore the *standing wave* phenomenon, as the waves of opposite directions but same frequencies and same amplitudes move against each other. In practice, the waves are not perfect standing waves, since the amplitudes are being perturbed by noise, but the general appearance of the ocean is similar.

6.1.2 JONSWAP

JONSWAP spectrum is an empirical spectrum developed by the analysis of data collected during the Joint North Sea Wave Observation Project JONSWAP [19].

The Phillips spectrum cross section



The spectrum is a modification of a Pierson-Moskowitz spectum [43] introduced earlier, which worked with the concept of a *fully developed sea* - a sea created by a steady wind over a larger area. In such a sea, the waves stabilize after a given time and no longer change. However, during the JONSWAP analysis, they came to a conclusion that a sea is never fully developed, and the waves continue to change as the effect of wave to wave interaction. Additional factor is introduced to modify the Pierson-Moskowitz spectrum to incorporate this observed effect.

The spectrum is defined over frequency domain, and does not incorporate the wind direction. Therefore, an additional modification to the spectrum has to be done by a directional filter.

The formula for a given frequency, independent of the direction is

$$J(\omega) = \frac{\alpha g^2}{(2\pi)^4} \cdot \exp(-\frac{5}{4}(\frac{\omega_p}{\omega})^4) \cdot \gamma^r$$

$$r = \exp(-\frac{(\omega - \omega_p)^2}{2\sigma^2\omega_p^2})$$

where ω is the wave frequency, ω_p is the peak frequency given by the wind speed V as

$$\omega_p = \frac{3.5g}{V} \cdot (\frac{gF}{V^2})^{-0.33}$$

g is the gravitation constant of value 9.8 m/s, and F is the *fetch*: the area over which the wind blows.

α is a *Phillips constant* of value $8.1 \cdot 10^{-3}$, $\gamma = 3.3$, and σ is given by

$$\sigma = \begin{cases} 0.07 & \omega \leq \omega_p \\ 0.09 & \omega > \omega_p. \end{cases}$$

In addition to the wind speed and direction, this spectrum also incorporates the fetch area. This is particulary useful for the computer graphics application. When producing an ocean surface tile, we only work on a predefined tile area, and this helps us to produce a realistic surface the wind would create if it blew only over this area, instead of cutting out a small area of a large theoretical ocean surface, where the long waves would be prevailing.

The direcional filter is the following.

$$D(\omega, \theta_v) = \frac{\Gamma(s+1)}{2\sqrt{\pi}\Gamma(s+0.5)} \cdot (\cos^2(\frac{\theta_v}{2}))^s$$

where ω is the frequency, θ_v is the angle between the wave and the wind direction, Γ is the *Gamma function*, and s is given as

$$s = \begin{cases} s_m \cdot (\frac{\omega}{\omega_p})^{-2.5} & \omega \geq \omega_p \\ s_m \cdot (\frac{\omega}{\omega_p})^5 & \omega < \omega_p. \end{cases}$$

$$s_m = \begin{cases} 9.77 & \omega \geq \omega_p \\ 6.97 & \omega < \omega_p. \end{cases}$$

The resulting spectrum is then computed as the original spectrum multiplied pointwise by the directional filter. The spectrum range however is not the amplitude of the wave, but rather the energy. To evaluate the amplitude, we have to re-scale it by a proper factor. The formula is taken from [51]

$$A(k_x, k_z) = J(\omega)D(\omega, \theta)\frac{g2\pi}{8|(k_x, k_z)|KL}$$

where ω is the frequency of the wave, computed by the formula mentioned earlier, θ is the angle between the wave direction and wind, and K and L are the physical dimensions of the ocean area.

Shape of the final ocean spectrum is in Figure 6.1.

We can see that unlike the Phillips spectrum, this spectrum is highly directional due to the filter supressing the waves travelling in the opposite direction. This mean that during the animation, the surface seems to be translating in the direction of the wind, which simulates the real waves travelling. However, the result does seem a bit artificial. Therefore, a better directional filter would be useful to produce a similar effect to the standing wave phenomenon.

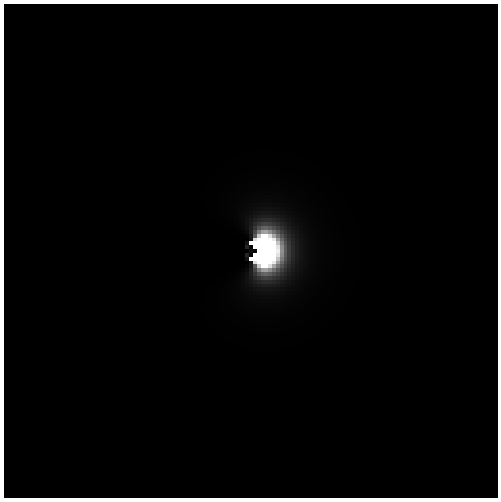
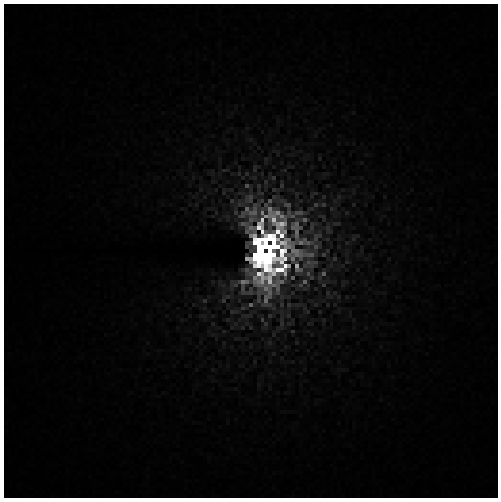
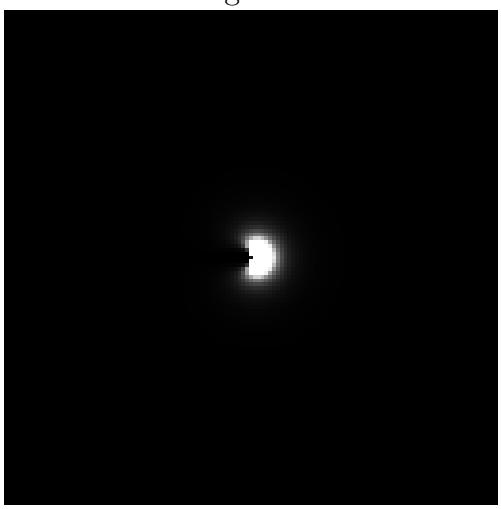
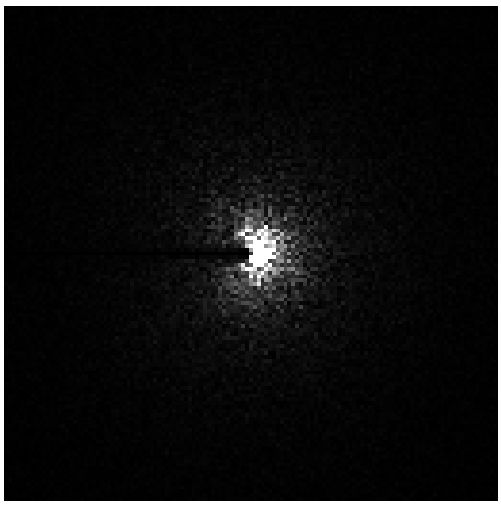
Since the spectrum incorporates the fetch area, the resulting wavelength distribution is much more suitable to our needs. If the fetch area is of the size of the tile, medium waves are prevailing, and the most long waves are supressed, which makes the wang tiling look natural.

6.2 Wind speed in comparison to ocean tile area

The wind speed is a parameter that affects the overall wave appearance of the tile due to specifying the distribution of the waves in the area. In general, creating the tiles from a spectrum where only the long waves are prevailing is a very hard task. First, trying to fit the long waves on the border constraints produces a highly distorted surface. Second, even if we fit the surface to be a natural transition, there is still a problem of the visual properties of such tiled area, since the main features would be of the size of one tile, and therefore, the apparent tiling will become visible.

It is necessary to set the wind speed to be *mild* for the given area. First, we take a look at how the ocean spectra look like given specific parameters. We can then evaluate which settings are suitable for the wang tiling method. The Phillips

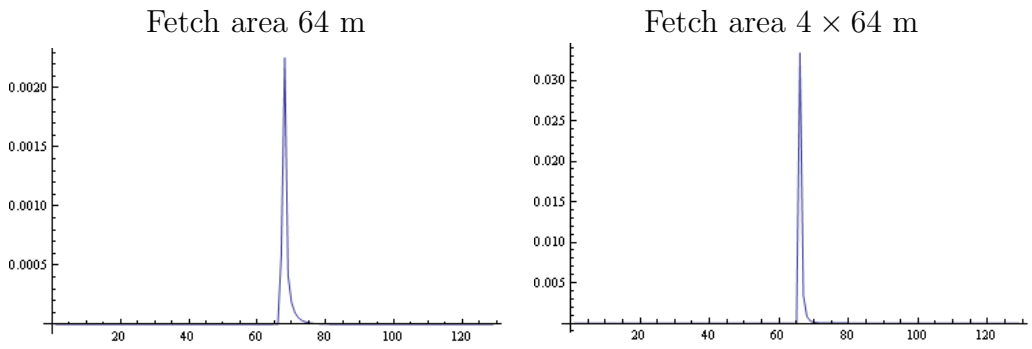
Tabuľka 6.1: The JONSWAP spectrum. The size of area is $64m^2$. The wind speed is 15 m/s.

Pure spectrum	Spectrum perturbed by the random numbers
Fetch length 64m	
	
Fetch length 4×64 m	
	

spectrum has only one parameter to set: the wind direction and its speed. The Jonswap spectrum has an additional parameter: the fetch area.

To obtain plausible results, we have to work with realistic parameters. A wind of speed 15 m/s over an area of 32 meters will produce almost solely long waves, and the algorithm to create the wang tiles will become unusable. However, a wind of speed 5 m/s (a *gentle breeze* on the Beaufort scale) on an area of size 64 meters produces a moderately distributed wave spectrum which is well suitable for the usage of the algorithm. In case of Jonswap, it is also very important to set the fetch area right. If we set it to a large value, it practically means that the wind is creating waves over a large area, and we are only looking at a small portion of that area. So again, most of the energy will be distributed among the long waves. We can set the fetch to be equal to the size of the tile. In such case, we will always have a spectrum that is moderately distributed, since the long wave

The JONSWAP spectrum cross section: Wind speed 15 m/s over 64 m.



cannot develop that much on such a small area. This is of course a physically unrealistic setting for an ocean area: in the ocean, we rarely get a wind blowing only over a 64 meter area. The setting can probably match lakes or closed bays. But most importantly, we are not trying to simulate the real nature - the tiling method itself breaks up all the physical basis. We simply want a visually plausible result.

6.3 Tile differentiation

An important feature of the Wang Tile set is its variability. If all the tiles are similar to each other, the resulting tiling will become apparent. As mentioned in the Section 3.3.2, the set created from just two ocean spectra is not diverse enough. We have to therefore also use an additional spectrum for each tile to specify the waves inside the tile area.

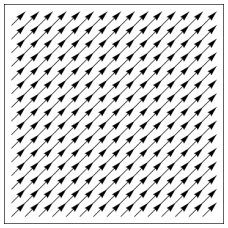
The size of the inside area is specified by the **Border Size** parameter. On the border, the waves are given as an interpolation between the edge constraints, and the inner spectrum, and inside the tile, the heightfield is only given by the inner spectrum itself. It is therefore important to set the border size to be large enough for the interpolated surface to give plausible results, but at the same time, small enough so that the area around edges would be different for each tile. The setting we found the best is the **Border Size** to je 1/5 to 1/4 of the grid size. For a stronger wind where we are expecting longer waves, we can increase this interval.

Another way to incorporate additional diversity into the tileset is to set some properties of the spectra explicitely. The area around the edges is mainly given by the spectra of the two original tiles, S_0^A and S_0^B. As mentioned above, the surface appearance is mostly dependent on the individual wave phase shifts. If we set the longest waves to have an opposite phase shift (randomly chosen from the interval of $\pi/4$ around an opposite phase), the area around edges will become more diverse for each tile. The optimal setting we have found is to set the **Wave Desynch** to $3 - 4$. For mild wind conditions, where the medium waves are prevailing, this provides a well diverse set of tiles.

For larger wind, where the long waves are prevailing it of course creates visible distortions. This can be avoided by explicitely removing the longest waves from the spectrum, their treshold being specified by the parameter **Wave remove**.

Although this approach goes against the physical principles, it might be handy in practical applications.

6.4 Interpolation directions



In Chapter 3, we proposed to construct the set of tiles using bended interpolation directions. To evaluate interpolation arguments over the tile area, at each point we change the interpolation direction. As mentioned in Chapter 3, this way we are able to produce interpolation arguments that are smooth over the inside of the area.

However, the arguments change more rapidly around the edges, which is the most problematic area. Visually, we don't mind the distortion within the tile as much as any disturbance around the edge.

Therefore, to avoid these problems, in practical application, we use only the simple directional interpolation for the diagonal tiles.

6.5 Corner areas

It is very important to set the corner settings for each situation. We divide the waves into two sets: the long waves, and the short ones. The long ones are smoothed in the cross-shaped areas specific for each tile-meeting case. The short ones are interpolated again, as if the waves continued in their direction, and we only evaluate 14 cases.

The treshold between the low frequencies and the high frequencies is set by the parameter **Frequency treshold**, and needs to be set with respect to the corner size and cross size so that no short wave will get smoothed out.

The smoothing in the cross-shaped areas is done by a 5×5 Gaussian kernel. Both for the cross-shaped areas, and the high-frequency corners, the resulting heightfield is again blended into the existing heightfield so that no height-steps appear at the edges of these areas.

6.6 Choppy waves

We also included the option for *choppy* waves, with sharper peaks and broader throughs. The waves are implemented by computing an additional heightfield positions displacement. By the simple combination technique, we simply interpolate between the resulting positions. But for a phase interpolation approach, we have two options: to compute both heights and displacement by the visually nicer phase interpolation, producing stretched or folded waves with a shape similar to the *Gerstner wave*. Or we can compute only the height by the phase interpolation, and compute the displacement as a simple combination of resulting displacements of the original surfaces. The other option produces a wave with skewed tops. However, the first option requires to hold the displacement information for each wave separately, with the resulting complexity being identical to the computation of heights. And as we need to compute the displacement both

in x and in z direction, the complexity increases three times. The other option, on the other hand, has a much lower complexity, while producing a slightly less quality results. The choice is therefore up to the user: whether to prefer a higher quality, or faster computations for the cost of quality decrease.

6.7 Suggested scenarios

As mentioned earlier, the most important parameter setting is the speed of the wind over the given area size. The one what works well for us is:

Spectrum Phillips

Resolution / Area 32/64

Wind speed $5 - 7$

choppiness $1 - 2$

Spectrum JONSWAP

Resolution / Area 32/64

Wind speed $10 - 15$

choppiness $1 - 2$

The JONSWAP spectrum has to have a larger wind speed because of the very small fetch area compared to a real ocean.

7. Results

Now that we have created the set of Wang Tiles and set up conditions for the usage of the method, we can evaluate the resulting visual appearance of such aperiodic tiling. The main criterium is of course visual plausibility. The tiling has to look like a seamless ocean area and no tiling artifacts should be present. In addition to that, we want the resulting surface to be a reasonable approximation of a real ocean under given conditions.

The main evaluating technique we used was simply to observe the resulting animation: watch the surface under different lighting and materials, and compare the tiled area to a large heightfield generated by the original Tessendorf method. In addition to that, we also provide a Fourier analysis of the resulting heightfields, and observe the differences in the spectral domain.

7.1 Comparison of different techniques

We now compare the results of the two different Wang Tile generation approaches we have implemented. The first one, a simple combination of heights of 3 original Tessendorf tiles [two for the borders, and one for the inside area] is clearly the fastest with the complexity of $O(N^2 \log N)$. The surface quality produced by this technique is highly dependent on the wave amplitudes. Since the drawback of a simple height combination is the wave dampening, it is clear that the larger the amplitudes of waves are, the more is this effect visible. As the interpolation only occurs in the border areas, the dampened waves will form a grid and the tiling will become apparent. On the other hand, for mild wind conditions the wave amplitudes are usually not that high and the grid is not visible.

However, the simple interpolation of the heights does produce a slightly lower waves in general. In addition to that, it rounds up the wave peaks if we use it for the choppy waves modification. The resulting surface is then still a visually plausible ocean area, but the properties are slightly different from the expected result. Still, if we set the parameters to describe mild weather conditions, and do not mind the fact that the amplitudes are slightly lower and the wave peaks slightly rounder, we can use this technique and profit from its simplicity and high speed.

The other technique we implemented is the phase-interpolation approach. At each point, both amplitude and the phase shift of the wave are computed as a combination of three spectra. The resulting height at the point is then computed by evaluating a wave with this amplitude and phase shift. In theory, this combination approach produces the most natural transition between the inside of the tile and the border constraints. In case of the choppy waves, it preserves the sharp peaks, and in general, preserves the overall wave features in a reliable way.

A major drawback of this method is its complexity. Since we are using the inner spectrum to fully define the waves inside the tile, we are only computing the height values in the border areas. Still, even with this optimization, as well as the optimization of clustering individual wave speeds, the method is much slower than the simple combination approach.

Still, this method produces the best results, and is recommended to use in

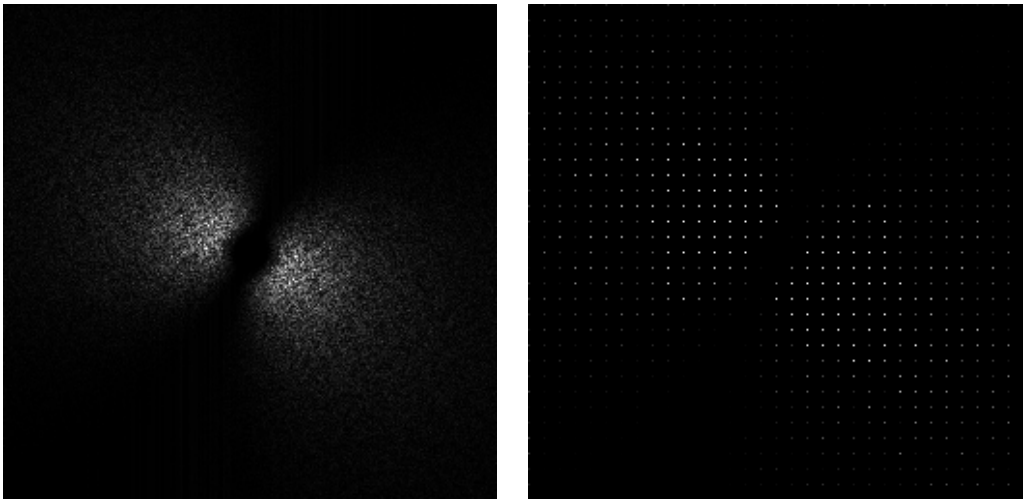
cases when the visual quality is dominant, and especially in cases of choppy waves with high amplitudes.

7.2 Spectral analysis

As another way of evaluating the results of the tiling is to compare the Fourier spectra of the area to an area generated as a large Tessendorf tile.

At first we can compare the spectrum of one large tile of size 256×256 on an area of 512×512 meters, and the original, periodic tiling with 8×8 tiles, each tile of size 32×32 points on an 64×64 area. The heightfield was generated using the Phillips spectrum with the wind speed of strength 5 m/s.

Spectra of a large tile, and the periodic tiling



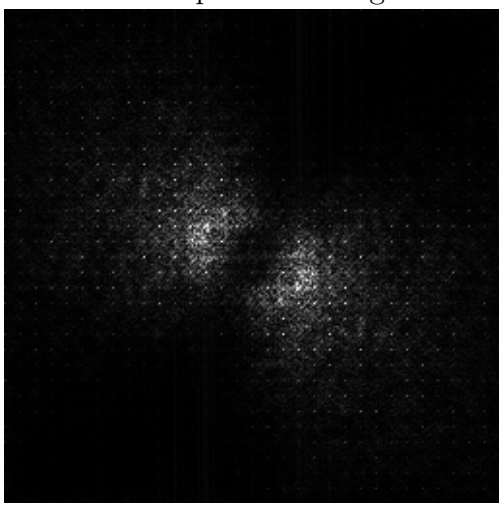
The shape of the periodic Fourier transform is expected. As we are copying the same instance of the tile over the whole area, we only cover frequencies dividable by 8 while the rest of the spectrum values are pure 0. The spectrum clearly shows the periodicity, which can also be seen in the spatial domain.

Let us now take a look at spectra produced by an aperiodic tiling, both using the tiles generated by the phase-interpolation technique, as the ones created by a simple combination of ocean heightfields.

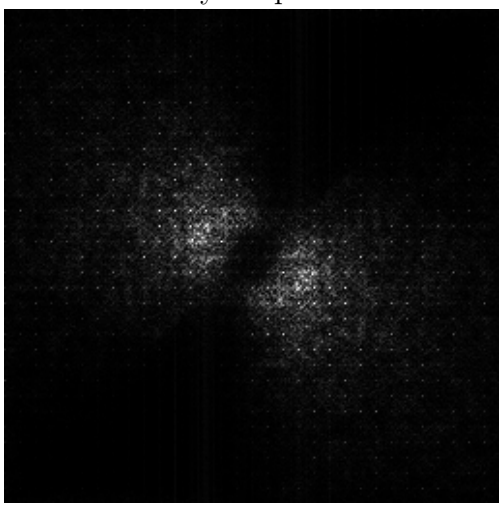
The resulting spectra are much more plausible. They still show the periodic patterns arising from the fact that each tile has the edge values strictly defined and these edges repeat over the whole tiling in given intervals. However, the heightfield now also contains frequencies inbetween introduced by the interpolated surface, the inner spectra, and overally from all various manipulations. The general shape of the spectrum resembles the spectrum generated by one large Tessendorf tile. There are differences of course, as the tiled surface cannot replicate the missing frequencies 100% well, but the fit is quite good.

Spectra of an aperiodic tiling

Phase-interpolated Wang Tiles



Tiles created by simple combinations



8. Conclusion

We have created a set of Wang Tiles each containing an ocean heightfield generated by a frequency-based method. We then use these tiles to aperiodically tile the plane, dealing with the corner-problems in a post-processing step. For a moderate wind conditions over a reasonably large area, the resulting ocean surface is a seamless heightfield with no visible artifacts present.

However, the method is unsuitable for large waves and atmospheric conditions in which low frequencies prevail. This is mainly due to the fact that if the main features of the heightfield are of size of one tile, the tiling becomes apparent. Futhermore, as the heightfield has set conditions on the edges, under some conditions it might become apparent that the exact values repeat over on a grid structure. However, this effect is not noticable under most conditions.

We also provide a simple implementation to demonstrate the results. The program provides an interface for setting various parameters and observe the visual quality of the animation. The ocean rendering is done is a very straightforward way and we do not implement any accelerating structures or a GPU implementation.

As a futher work we suggest an improved implementation and futher reasearch in making the animation realtime.

Literatúra

[1] BRUNETON E., NEYRET F., HOLZSCHUCH N.: Realtime realistic ocean lighting using seamless transitions from geometry to brdf. Computer Graphics Forum, 29, 2 (2010).

[2] BURKE R.: http://robburke.net/mle/wang/, visited December 5th, 2013.

[3] CHIU Y.-F., CHANG C.-F.: Gpu-based ocean rendering. In Proceedings of the IEEE International Conference on Multimedia and Expo (Toronto, Canada, 2006), pp. 2125–2128.

[4] CHOU C.-T., FU L.-C.: Ships on real-time rendering dynamic ocean applied in 6-dof platform motion simulator. In Proceedings of the CACS International Conference (Taichun, Taiwan, 2007).

[5] CIEUTAT J.-M., GONZATO J.-C., GUITTON P.: A general ocean waves model for ship design. In Virtual Concept, Conference Proceedings (Biarritz, France, 2003), pp. 187–194.

[6] CHEN H., LI Q.,WANG G.: An efficient method for real-time ocean simulation. Technologies for E-Learning and Digital Entertainment, 4469 (2007), 3–11.

[7] COHEN M., SHADE J., HILLER S., DEUSSEN O.: Wang Tiles for image and texture generation. ACM Transactions on Graphics (TOG), v.22 n.3, July 2003.

[8] CUI X., YI-CHENG J., XIU-WEN L.: Real-time ocean wave in multi-channel marine simulator. In Proceedings of the International Conference on Virtual RealityContinuum and its Applications in Industry (Singapore, 2004), pp. 332–335.

[9] DARLES E., CRESPIN B., GHAZANFARPOUR D.: Accelerating and enhancing rendering of realistic ocean scenes. In Proceedings of the WSCG (Plzen, Czech Republic,2007).

[10] DARLES E.,CRESPIN B.,GHAZANFARPOUR D.,GONZATO J.C.: A Survey of Ocean Simulation and Rendering Techniques in Computer Graphics. Computer Graphics Forum, 30, 1 (2011), pp. 43-60.

[11] EFROS A., FREEMAN W.: Image quilting for texture synthesis and transfer, Proceedings of the 28th annual conference on Computer graphics and interactive techniques, p.341-346, August 2001.

[12] FOURNIER A., REEVES W.: A simple model of ocean waves. In Proceedings of the ACM SIGGRAPH (Dallas, TX, USA, 1986), pp. 75–84.

[13] FRECHOT J.: Realistic simulation of ocean surface using wave spectra. In Proceedings of the GRAPP (Setubal, Portugal, 2006), pp. 76–83.

[14] FU C.-W., LEUNG M.-K., Texture tiling on arbitrary topological surfaces using wang tiles, Proceedings of the Sixteenth Eurographics conference on Rendering Techniques, June 29-July 01, 2005, Konstanz, Germany.

[15] GAMITO M., MUSGRAVE F.: An accurate model of wave refraction over shallow water. Computers & Graphics, 26, 2 (2002), 291–307.

[16] GERSTNER F.J.: Theorie der Wellen. Abh. d. k. bohm. Ges. d. Wiss. (now listed under Česká společnost nauk), Prague. Reprinted in Ann. der Physik 1809, 32, 412-440.

[17] GONZATO J.-C., SAEC B. L.: On modeling and rendering ocean scenes (diffraction, surface tracking and illumination). In Proceedings of the WSCG (Plzen, Czech Republic, 1999), pp. 93–101.

[18] GONZATO J.-C., SAEC B. L.: On modeling and rendering ocean scenes. Journal of Visualisation and Computer Simulation, 11 (2000), 27–37.

[19] HASSELMANN K., BARNETT T.P., E. BOUWS, H. CARLSON, D.E. CARTWRIGHT, K. ENKE, J.A. EWING, H. GIENAPP, D.E. HASSELMANN, P. KRUSEMAN, A. MEERBURG, P. MLLER, D.J. OLBERS, K. RICHTER, W. SELL, AND H. WALDEN: Measurements of wind-wave growth and swell decay during the Joint North Sea Wave Project (JONSWAP)' Ergnzungsheft zur Deutschen Hydrographischen Zeitschrift Reihe, A(8) (Nr. 12), p.95, 1973.

[20] HILLER S., DEUSSEN O., KELLER A.: Tiled Blue Noise Samples, Proceedings of the Vision Modeling and Visualization Conference 2001, p.265-272, November 21-23, 2001.

[21] HINSINGER D., NEYRET F., CANI M.-P.: Interactive animation of ocean waves. In SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (San Antonio, TX, USA, 2002), pp. 161–166.

[22] HU Y., VELHO L., TONG X., GUO B., SHUM H.: Realistic, real-time rendering of ocean waves. Computer Animated Virtual Worlds, 17, 1 (2006), 59–67.

[23] ISIDORO J., VLACHOS A., BRENNAN C.: Rendering ocean water. In Direct3D ShaderX: Vertex and Pixel Shader Tips and Tricks. W. Engel (Ed.). Wordware Publishing, Inc., Plano, TX, USA (2002).

[24] JENSEN L., GOLIÁŠ R.: Deep-water animation and rendering, In Proceedings of the Game Developper's Conference Europe '01, 2001.

[25] JOHANSON C.: Real-Time Water Rendering - Introducing the Projected Grid Concept. Master's thesis, Lund University, 2004.

[26] KINSMAN B.: Wind Waves: Their Generation and Propagation on the Ocean Surface. Dover Publications, 1984.

[27] KOPF J., COHEN-OR D., DEUSSEN O., LISCHINSKI D.: Recursive Wang tiles for real-time blue noise, ACM Transactions on Graphics (TOG), v.25 n.3, July 2006.

[28] KRYACHKO Y.: Using vertex texture displacement for realistic water rendering. GPU Gems, 2 (2005), 283–294.

[29] LACHMAN L.: An open programming architecture for modeling ocean waves. In Proceedings of the IMAGE Conference (Scottsdale, AZ, USA, 2007).

[30] LAGAE A., KAPLAN C.S., FU C.-W., OSTROMOUKHOV V., DEUSSEN O.: Tile-based methods for interactive applications. ACM SIGGRAPH 2008 classes, 2008.

[31] LEE N., BAEK N., RYU K.: Real-time simulation of surface gravity ocean waves based on the tma spectrum. In Proceedings of the International Conference on Computational Science (Beijing, China, 2007), pp. 122–129.

[32] LEE H.-M., GO C., LEE W.-H.: An efficient algorithm for rendering large bodies of water. Entertainment Computing - ICEC, 4161 (2006), 302–305.

[33] LEE H.-M., GO C., LEE W.-H.: A frustrum-based ocean rendering algorithm. Agent Computing and Multi-Agent Systems, 4088 (2006), 584–589.

[34] MAUNG D., YINXUAN S., CRAWFIS R.: Procedural textures using tilings with Perlin Noise. Computer Games (CGAMES), 2012 17th International Conference on, pp 60-65. DOI: 10.1109/CGames.2012.6314553.

[35] MAX N.: Vectorized procedural models for natural terrain: Waves and islands in the sunset. In Proceedings of the ACM SIGGRAPH (Dallas, TX, USA, 1981), pp. 317–324.

[36] MITCHELL J.: Real-Time Synthesis and Rendering of Ocean Water. Tech. Rep., ATI Research, 2005.

[37] MASTIN G., WATTERBERG P., MAREDA J.: Fourier synthesis of ocean scenes. IEEE Computer Graphics and Applications, 7, 3 (1987), 16–23.

[38] NEYRET F., CANI M.-P.: Pattern-based texturing revisited. Proceedings of the 26th annual conference on Computer graphics and interactive techniques, p.235-242, July 1999.

[39] NG T.-Y., WEN C., TAN T.-S., ZHANG X., KIM Y. J.: Generating an /spl omega/-tile set for texture synthesis. Proceedings of the Computer Graphics International 2005, p.177-184, June 02-04, 2005.

[40] PREMOZE S., ASHIKHMIN M.: Rendering natural waters. In PG '00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications (Washington, DC, USA, 2000), IEEE Computer Society, p. 23.

[41] PEACHEY D.: Modeling waves and surf. SIGGRAPH Computer Graphics, 20, 4 (1986), 65–74.

[42] PERLIN K.: An image synthesizer. SIGGRAPH Computer Graphics, 19, 3 (1985), pp. 287-296.

[43] PIERSON W., MOSKOWITZ L.: A proposed spectral form for fully developed wind seas based on similarity theory of s.a kilaigorodskii. Journal of Geophysical Research (1964), 5281–5190.

[44] RANKINE W. J. M.: On the exact form of waves near the surface of deep water. Phil. Trans. Roy. Soc. A 153, 127-138.

[45] ROBINE M., FRECHOT J.: Fast additive sound synthesis for real-time simulation of ocean surface. In Proceedings of the International Conference on Systems, Signals and Image Processing (IWSSIP) (Budapest, Hungary, 2006), pp. 223–226.

[46] SALGADO A., CONCI A.: On the simulation of ocean waves in real-time using the gpu. In Proceedings of the Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI) (Belo Horizonte, Brazil, 2007).

[47] SCHACHTER B.: Long crested wave models. CGIP, 12, 2 (1980), 187–201.

[48] SCHNEIDER J., WESTERMANN R.: Towards real-time visual simulation of water surfaces. In VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001 (2001), Aka GmbH, pp. 211–218.

[49] SHADE J., COHEN M. F., MITCHELL D. P.: Tiling layered depth images. Technical report, University of Washington, Department of Computer Science and Engineering, 2000.

[50] STAM J.: Aperiodic Texture Mapping. European Research Consortium for Informatics and Mathematics (ERCIM), 1997.

[51] STEPANEK R.: Photorealistic modelling of ocean surfaces. Master thesis, Vienna University of Technology, 2004.

[52] TS'O P., BARSKY B.: Modeling and rendering waves: wave-tracing using beta-splines and reflective and refractive texture mapping. ACM Transactions on Graphics, 6, 3 (1987), 191–214.

[53] THON S., DISCHLER J.-M., GHAZANFARPOUR D.: Ocean waves synthesis using a spectrum-based turbulence function. In CGI '00: Proceedings of the International Conference on Computer Graphics (Washington, DC, USA, 2000), IEEE Computer Society, p. 65.

[54] TESSENDORF J.: Simulating ocean waters. In SIGGRAPH course Notes (Course 47) (2001).

[55] THON S.,GHAZANFARPOUR D.:Oceanwaves synthesis and animation using real world information. Computers & Graphics, 26, 1 (2002), 99–108.

[56] WANG H.: Proving theorems by pattern recognition - II. Bell Systems Technical Journal, 40,1-42, 1961.

[57] WEI L.-Y.: Tile-based texture mapping on graphics hardware, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, August 29-30, 2004, Grenoble, France.

[58] XIN Z., FENGXIA L., SHOUYI Z.: Real-time and realistic simulation of large-scale deep ocean surface. In Proceedings of the Advances in Artificial Reality and Tele-Existence (Hangzhou, China, 2006), pp. 686–694.

[59] http://www.johndcook.com/stand_alone_code.html, visited December 5th, 2013.

[60] http://www.fftw.org/, visited December 5th, 2013.