DOI: 10.1111/cgf.14100
ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2020
J. Bender and T. Popa
(Guest Editors)

Volume 39 (2020), Number 8

Making Procedural Water Waves Boundary-aware

S. Jeschke[1] C. Hafner[2] N. Chentanez[1] M. Macklin[1] M. Müller-Fischer[1] C. Wojtan[2]

[1]NVIDIA [2]IST Austria

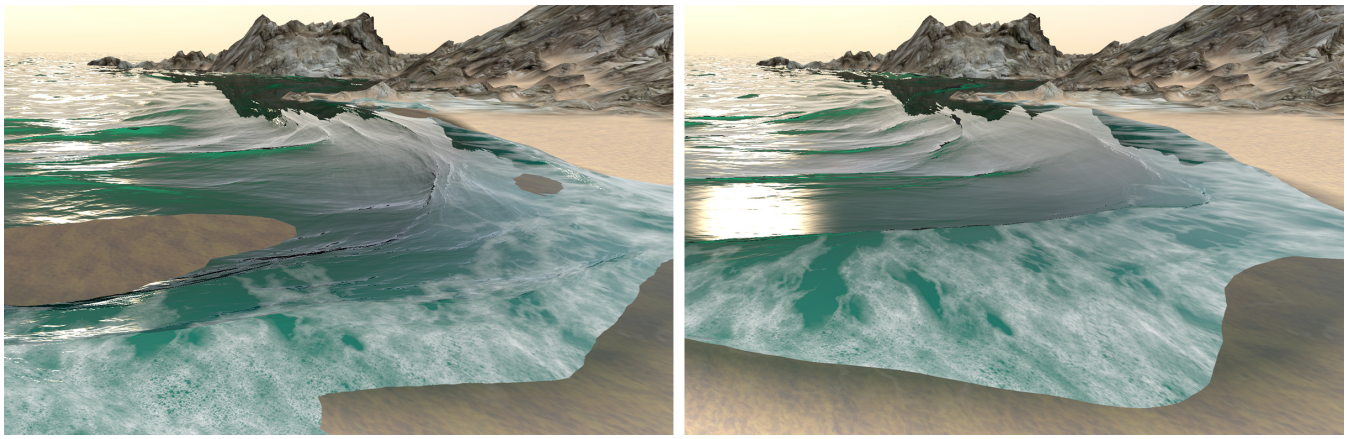


Figure 1: *The left part shows a conventional real-time ocean wave simulator: the wave model is unaware of the boundary conditions on the shore, so it unrealistically intersects the terrain. The right shows the same simulator when filtered through our model, avoiding terrain intersections.*

Abstract

The "procedural" approach to animating ocean waves is the dominant algorithm for animating larger bodies of water in interactive applications as well as in off-line productions — it provides high visual quality with a low computational demand. In this paper, we widen the applicability of procedural water wave animation with an extension that guarantees the satisfaction of boundary conditions imposed by terrain while still approximating physical wave behavior. In combination with a particle system that models wave breaking, foam, and spray, this allows us to naturally model waves interacting with beaches and rocks. Our system is able to animate waves at large scales at interactive frame rates on a commodity PC.

CCS Concepts

• ***Computing methodologies*** → ***Procedural animation;***

1. Introduction

Even after decades of algorithmic advances in computer animation, detailed simulations of large-scale bodies of water are extremely costly in terms of computational demand and memory. For such scenarios, waves are typically modeled using *linear wave theory* [Joh97], as it provides highly detailed simulations with comparably low computational demand [Tes99]. This solution works perfectly for ideal situations with periodic domains, and no boundaries or interacting obstacles. However, the behavior of procedural waves interacting with a sloped obstacle (like a beach) is still an open question. Previous procedural methods either emphasize realistic behavior *away from boundaries* (think deep water ocean waves), or model isolated phenomena like breaking waves. Furthermore, current procedural wave tools tend to fail in visually disturbing ways, like water penetrating solid obstacles and vice versa. In modern animation visual effects practice, these problems are manually covered by artists; for example, they hand-design specific wave primitives for beaches, use specific (steep) terrains at the water boundary, or use computationally expensive off-line simulators in problematic regions.

This paper presents a strategy for efficiently eliminating such visual artifacts from procedural wave models. We introduce a generalized transformation of wave trajectories called *wave cages*; these cages are designed to warp the original procedural wave motion in a way that satisfies solid boundary conditions and avoids interpenetration artifacts, while exactly preserving the original wave motions in open water. This strategy makes the popular procedural wave model applicable to more general scene configurations, and its low computational overhead preserves real-time performance even for large-scale scenes.

2. Related work

This section discusses a number of existing techniques for animating water waves interacting with obstacles.

Grid-based Navier-Stokes simulation methods Regular grid-based 3D fluid simulation [Bri15] can offer great accuracy and realism at the expense of vast computational resources. At large scales, the conservation of volume and other physical properties also remains a challenge. Attempts have been made by using adaptive grids such as octrees [LGF04, FWD14], tetrahedral meshes [CFL*07, ATW13], non-uniform rectilinear grids [ZLC*13], and sparse paged grids [AGL*17] to name a few. Despite constant progress, these methods are currently unsuitable for animating detailed ocean scale wave motion at interactive frame rates.

Particle-based methods An excellent recent survey about particle based liquid simulation with Smoothed Particle Hydrodynamics (SPH) can be found in [KBST19]. Position based fluids were proposed in [MM13]. For these methods, the number of particles required grows linearly with the volume of simulated liquid. A number of works use multi-resolution particles to try to address this issue [KAD*06, BG11]. Multiple scales of particles are used for simulating breaking waves in [DCG11]. While these methods can produce highly detailed liquid surfaces and small splashes, interactive performance is still limited to relatively small-scale scenes.

2.5D approaches Researchers have developed a number of *2.5D* methods which simplify the Navier-Stokes equations by assuming the water surface is a heightfield. For example, several researchers solve the wave equation on a 2D grid [KM90, Tes99] or utilize pipe models [OH95, vBBK08]. Other researchers employ Lagrangian representations, such as wave particles [YHK07] and wave packets [JW17]. The shallow water equations (SWE) are useful when the water wavelength is much larger than the water depth. SWE discretizations are usually Eulerian [CL95, LvdP02, HHL*05, WMT07, ATBG08], but some researchers have proposed Lagrangian 2D SPH discretizations as well [LH10, SBC*11]. [JSMF*18] use a wavelet representation that discretizes waves as functions of space, frequency, and direction to handle local interactions with wind and obstacles, and [SHW19] use fundamental solutions to animate wave-obstacle interactions. A recent method computes Lagrangian waves on top a moving water surface to add detail to an existing simulation [SSJ*20].

Procedural waves We use the term "procedural" wave models when referring to closed-form scripted behaviors of water waves. Computer graphics applications often accumulate many simple wave behaviors, typically with a fast summation algorithm like the FFT, in order to animate rich ocean details with little computational expense. Researchers tend to use circular wave trajectories for animating deep oceans [MWM87, Tes99] and ellipsoidal surface motions for animating waves on a sloped terrain [Pea86, FR86]. Tools for visual effects and game development, like NVIDIA's *WaveWorks* [Che20] typically combine these procedural wave motions with artist-friendly authoring tools in order to hand-tune wave behavior. Although some techniques allow for reflections and diffractions around obstacles [GLS00, JW15], none of these methods can actually prevent individual waves from intersecting the terrain.

3. Background

Linear wave theory [Joh97] provides the basis for modern ocean wave animation. Following [Tes99], the model describes the motion of a point $\boldsymbol{\eta}(\boldsymbol{x},t)$ on the water surface as a summation of many independent waves at different frequencies:

$$\boldsymbol{\eta}(\boldsymbol{x},t) = \sum_{i=1}^{N} \begin{pmatrix} -\hat{\boldsymbol{k}}_i A_i \sin(\boldsymbol{k}_i \cdot \boldsymbol{x} - \omega_i t + \phi_i) \\ A_i \cos(\boldsymbol{k}_i \cdot \boldsymbol{x} - \omega_i t + \phi_i) \end{pmatrix}. \tag{1}$$

Here, the wavevector $\boldsymbol{k}_i$ is a two-dimensional frequency vector with magnitude k_i and direction $\hat{\boldsymbol{k}}_i$. A_i is the wave amplitude, ω_i is the angular frequency, ϕ_i is the initial phase offset, and N is the total number of wave frequencies. Points on the wave surface evolve in circular motions, and a summation of these circles produces beautiful patterns known as "Gerstner" waves.

Subsequent work by Biesel and others [Bie52, CHH05, CHCH06] improved upon the Gerstner model by deriving the motion of waves above an infinitesimally sloped sea floor. Instead of summing over circular motions as in Eq. (1), this model produces a sum of *ellipsoidal* wave trajectories aligned with the sloping direction of the terrain. Although this is a significant improvement over the Gerstner model and has proved a great success in computer graphics [FR86], it still falls short of satisfying the demands of modern computer animation. The Biesel model assumes an infinitesimally small slope in the terrain, which causes the solution to blow up as the water depth tends to zero (i.e., the visually interesting point where the water surface splashes against the terrain). Furthermore, both Gerstner and Biesel models can exhibit virtually unbounded wave heights when all of the individual waves happen to constructively interfere with each other, practically guaranteeing unphysical terrain penetrations. Despite these problems, the Biesel model probably represents the theoretical limit of procedural wave models derived from first principles—any generalization beyond infinitesimal slopes would likely have to deal with refracting and breaking behaviors, which may be impossible to describe with a closed-form periodic trajectory. Consequently, we do not expect to find a closed-form wave solution for waves interacting with arbitrary terrain.

In this paper, we propose a geometric approach to remove the most problematic limitations of existing procedural wave solutions such as FFT methods or water surface wavelets [JSMF*18]—we bound procedural wave motions within ellipsoidal *wave cages* aligned with the slope of the terrain. In contrast to the procedural models discussed above, this simple constraint guarantees that the water surface never penetrates the sea floor, and it prevents unrealistic behaviors where the water meets the shore.

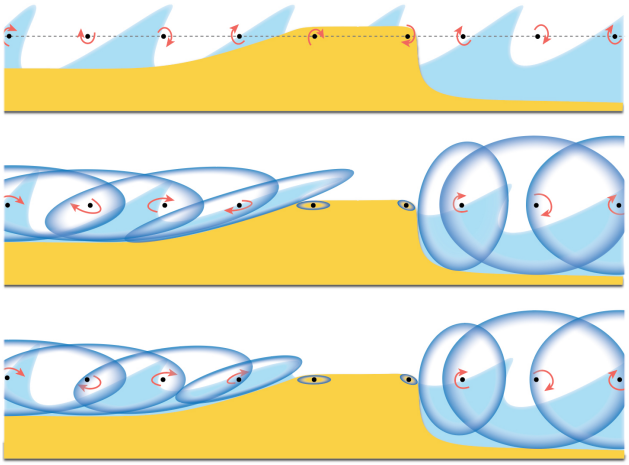


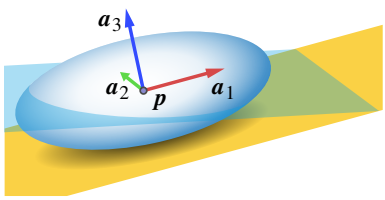
Figure 2: *Top: 2D example illustrating unconstrained Gerstner wave motion. Note the intersections with the underlying terrain. Middle: For each black sample point located at sea level, we fit a wave cage (blue ellipses) that is as large as possible without intersecting local terrain. The water surface is only allowed to move within the space of interpolated wave cages, preventing all intersections except at the water line. Bottom: Spatially smooth wave cages prevent unrealistic distortions near the terrain.*

4. Method Overview

To make waves boundary-aware, we define *wave cages* that allow us to locally constrain wave motion to lie within a prescribed ellipsoidal boundary (Section 5). Wave cages are dimensioned based on the proximity to the local terrain, which they are not allowed to intersect (Section 6). By confining wave motion to these fitted cages, we guarantee that waves do not penetrate the sea floor or any surrounding obstacles. Finally, we ensure that wave motions blend smoothly over space (Section 7), and explain how to reconstruct the new warped water surface (Section 8). The wave cage fitting process is outlined in Fig. 2.

5. Wave cages

This section introduces our concept of a *wave cage* which smoothly warps an unbounded, terrain-penetrating periodic wave trajectory into a bounded path that is guaranteed to avoid terrain intersections. We store cage parameters at each node $\boldsymbol{p}$ of a 2D regular grid located at sea level $z = 0$.



We define an orthonormal local coordinate frame of each wave cage $\boldsymbol{A}(\boldsymbol{x}) = [\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{a_3}]$ and align the cage with the terrain by setting $\boldsymbol{a_3}$ as the terrain surface normal, $\boldsymbol{a_1}$ pointing towards the closest water line, and $\boldsymbol{a_2}$ as the direction orthogonal to $\boldsymbol{a_1}$ and $\boldsymbol{a_3}$, as seen in the inset. We can define an ellipsoid aligned with this cage by associating a "radius" with each axis: $\boldsymbol{r}(\boldsymbol{x}) = [r_1, r_2, r_3]^T$ creates the ellipsoid $\hat{x}^2/r_1^2 + \hat{y}^2/r_2^2 + \hat{z}^2/r_3^2 = 1$

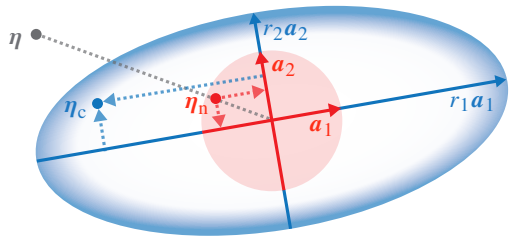


Figure 3: *Wave cage transformation example in 2D. The coordinates of a given point $\boldsymbol{\eta}$ are first scaled to fit in the unit circle (here depicted in red), giving the "normalized" point $\boldsymbol{\eta}_n$. Projecting $\boldsymbol{\eta}_n$ onto each ellipse axis and multiplying the resulting coordinates with the respective radii-scaled axes $r_1\boldsymbol{a_1}$ and $r_2\boldsymbol{a_2}$ provides the final point $\boldsymbol{\eta}_c$ within the wave cage.*

in frame-aligned coordinates $(\hat{x}, \hat{y}, \hat{z})$. We interpolate frame parameters at any point on the $(x, y, 0)$ plane using RoSy field interpolation [ZHT07]. Note that our ellipsoidal cages resemble the oriented solid particles of [MC11], who had a similar goal of warping physical simulation primitives to align with boundaries.

Our next task is to map a potentially unbounded motion to one that is guaranteed to lie within this ellipsoidal cage. As input to our method we assume that a procedural wave model has displaced every point of the planar water surface at rest by some displacement vector $\boldsymbol{\eta}$. We first choose to soft-clip the wave displacement $\boldsymbol{\eta}$ such that it is bounded by a unit sphere, and then deform the sphere into an ellipsoid. We start by applying a sigmoidal soft-clipping function

$$\boldsymbol{\eta}_{\mathrm{n}}(\boldsymbol{\eta}) = \begin{cases} \frac{\boldsymbol{\eta}}{\|\boldsymbol{\eta}\|} \tanh(s\|\boldsymbol{\eta}\|), & \text{if } \boldsymbol{\eta} \neq \boldsymbol{0}, \\ \boldsymbol{0}, & \text{otherwise}, \end{cases} \tag{2}$$

where s is a user parameter controlling the steepness of the nonlinear compression of the wave as it approaches the sphere boundary. We empirically choose $s = 0.5$ for all of our examples. This operation bounds the magnitude of $\boldsymbol{\eta}_n$ such that $\|\boldsymbol{\eta}_n\| < 1$.

We then transform this normalized displacement into the space occupied by the ellipsoidal wave cage using the map

$$\boldsymbol{\eta}_{\mathrm{c}}(\boldsymbol{x}, \boldsymbol{\eta}) = \boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^T \boldsymbol{\eta}_{\mathrm{n}}(\boldsymbol{\eta}), \quad \text{with} \quad \boldsymbol{R} = \mathrm{diag}(r_1, r_2, r_3), \tag{3}$$

where $\boldsymbol{A}(\boldsymbol{x}) = [\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{a_3}]$ is the matrix of local cage axes as defined above. Fig. 3 illustrates these steps. We note that the nonlinear cage mapping described here is not necessarily optimal, and other choices may be equally effective. We chose this particular transformation because it imposes guaranteed bounds while preserving excellent behavior for the most common smaller waves, and because it has analytic derivatives that prove useful in surface shading and additional physics/particle simulation (Appendix A). We also note that this mapping exhibits no singularities, so (unlike the Biesel model) it will generally have bounded velocities everywhere.

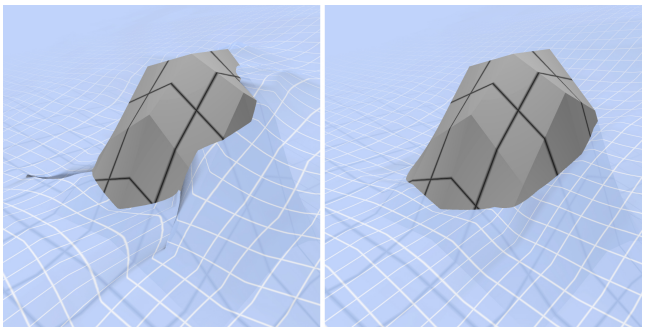


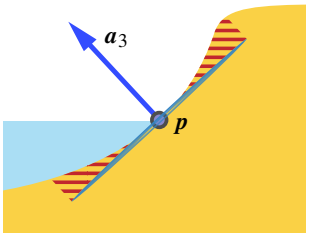
Figure 4: *Water surfaces (blue) resulting from a constant input wave displacement $\boldsymbol{u} = [0,0,1]^T$, filtered through terrain-fitted wave cages. Left: using unsmoothed cages from initial fitting (Section 6) distorts the surface considerably. Right: using smoothed cages (Section 7) removes distortions and results in a smooth surface. The black grid projected onto the terrain indicates the spacing of wave cage samples.*

6. Boundary-aware wave cages

Our goals in choosing wave cage radii $\boldsymbol{r}$ are that

- Gerstner wave motion is preserved as much as possible,
- points on the waterline may slide across the terrain, but no new water-terrain intersections appear (*cf.* Fig. 1),
- and the resulting motion is smooth.

Thus, $\boldsymbol{r}$ should be as large as possible, so water motion is not unduly restricted, but small enough to avoid terrain penetration of the water surface. Inspired by the interference-aware geometric modeling approach of [HPSZ11], we achieve these goals by growing a small ellipsoidal cage until it intersects the terrain.



We start from $\boldsymbol{r} = \boldsymbol{0}$, and iteratively increase the radii starting with r_1 and r_2. The expansion stops when either a maximal value is reached, or the penetration depth of the ellipsoid into the terrain exceeds a small user-defined threshold d, typically on the order of 10 cm. Without this threshold, curved terrain at the waterline (see inset) would immediately halt cage growth at radius zero, because even a small ellipsoid would intersect the terrain a little bit. This would result in the waterline "sticking" to the beach. The threshold d bounds the penetration depth (hatched red) without sacrificing the characteristic sliding behavior of waves washing up on a beach. We further investigate the effects of this threshold parameter in Section 8 and our supplemental video. Afterward, we analogously grow r_3 without additional thresholding. For points on the waterline, this yields $r_3 = 0$, effectively constraining waves to slide along planar terrain.

We choose to initially grow the two tangential directions r_1 and r_2 because typically $r_1, r_2 \gg r_3$ close to the waterline, so they account for most of the interesting wave motions. As shown in Fig. 2 (middle), cages are also fitted at grid points below the terrain—the same construction rules apply, but the roles of terrain and air are reversed.

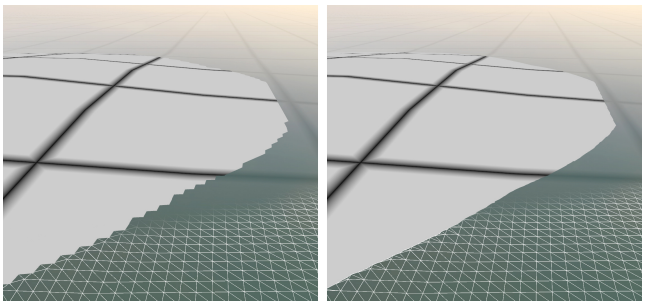


Figure 5: *Left: Jagged water boundary from linear interpolation after projecting vertices onto the terrain. Right: Problem solved by cutting and retessellating triangles that cross the waterline.*

7. Cage Smoothing

The cages constructed in the previous section vary smoothly over space, except near sharp terrain features and at the waterline—observe the cage size difference between points below and above the terrain in Fig. 2 (middle). This will make the water surface look distorted and introduce amplitude variations not present in the input waves. To mitigate this issue, we smooth $\boldsymbol{r}$ in a way that preserves the flat appearance of low-frequency waves.

In our wave cage model, waves with infinite wavelength will produce displacements of the form $\boldsymbol{\eta}_\mathrm{u}(\boldsymbol{x},\boldsymbol{r},\boldsymbol{u}) = \boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^T\boldsymbol{u}$, for some $\boldsymbol{u} \in \mathbb{R}^3$. This equation results from Eq. (3) after replacing the soft-clipped input wave $\boldsymbol{\eta}_\mathrm{n}$ by a constant vector $\boldsymbol{u}$. Fig. 4 shows a wave of this type for a fixed value of $\boldsymbol{u}$ before and after smoothing.

Our goal is to smooth the water surface $\boldsymbol{\eta}_\mathrm{u}$ for all such $\boldsymbol{u}$, so we solve the linearly constrained quadratic optimization problem

$$\begin{aligned} \min_{\boldsymbol{r}} \quad & \int_\Omega \left[(1-\alpha) \int_{S^2} \|\nabla \boldsymbol{\eta}_\mathrm{u}(\boldsymbol{x},\boldsymbol{r},\boldsymbol{u})\|_F^2 \,\mathrm{d}\boldsymbol{u} + \alpha \|\boldsymbol{r} - \bar{\boldsymbol{r}}\|^2 \right] \mathrm{d}\boldsymbol{x}, \\ \text{s.t.} \quad & r_3 = 0 \text{ on } \partial\Omega, \\ & 0 \le r_i \le \bar{r}_i \text{ in } \Omega, \forall i \in \{1,2,3\}, \end{aligned} \tag{4}$$

where $\|\cdot\|_F$ is the Frobenius norm. The first term in the energy penalizes high variations of $\boldsymbol{\eta}_\mathrm{u}$ by integrating over all possible input wave directions $\boldsymbol{u}$ with $\|\boldsymbol{u}\| = 1$. The second term keeps $\boldsymbol{r}$ close to the initial radii $\bar{\boldsymbol{r}}$ that were determined by the steps in Section 6. The weighting factor $\alpha = \alpha(\boldsymbol{x})$ is chosen close to 1 where $\int_{S^2} \|\nabla \boldsymbol{\eta}_\mathrm{u}\|_F^2 \,\mathrm{d}\boldsymbol{u}$ is already small prior to smoothing, and close to 0 where heavy smoothing is required.

At the waterline (where the sea level intersects the terrain), we apply the Dirichlet b.c. $r_3 = 0$ to retain sliding wave behaviors along the shore. The component-wise inequalities on $\boldsymbol{r}$ are imposed to ensure that ellipsoids may not increase in size and intersect the terrain after smoothing.

It can be shown that this problem is equivalent to a linearly

Figure 6: *Drone view of a beach scene, showing no interpenetration of water and beach.*

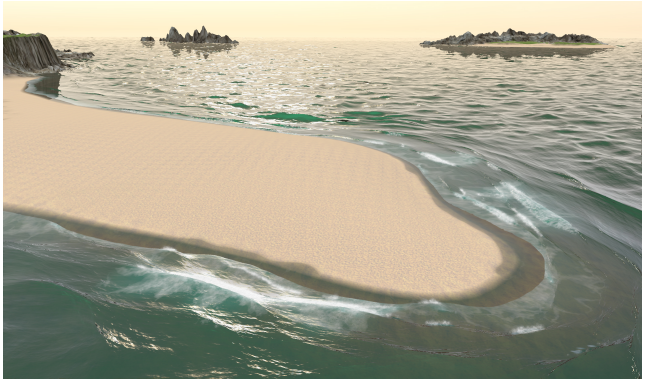


Figure 7: *Another view of the complex beach scene.*

constrained Dirichlet energy minimization problem of the form $\min_{\boldsymbol{r}} \int_\Omega (\|\nabla \boldsymbol{r}\|_F^2 + \boldsymbol{r}^T \boldsymbol{C}\boldsymbol{r} + \beta\|\boldsymbol{r} - \bar{\boldsymbol{r}}\|^2)\,\mathrm{d}\boldsymbol{x}$. Thus, it has a unique solution and can be discretized using standard finite differencing. We solve the system off-line using Jacobi iterations, as described in Appendix B. Fig. 2 (bottom) shows the result of this step in a 2D example.

8. Water surface representation

We model the water surface as a regular grid mesh with a spacing of 16 meters. It is then dynamically subdivided by the DirectX 11® Tessellation stage. We animate the water surface using a procedural wave model [JSMF*18] and filter it through our wave cages. The approach outlined so far gives satisfying results with only minor visible intersections, but two potential sources of visual artifacts remain. First, the penetration depth threshold d used in the initial cage fitting process (Section 6) can lead to small overlaps with the terrain. Second, the waterline may look jagged due to the finite resolution of the water mesh, as shown in Fig. 5 (left) and the supplemental video.

Both problems can be remedied by slightly modifying the tessellated water mesh. To fix any remaining intersections, water vertices below the terrain are displaced to lie above the terrain instead. We do this by sampling the tangent plane of the terrain at the location of the vertex, and projecting the vertex onto it. To prevent z-fighting, the vertex is further moved along the terrain normal by a small amount.

Jagged edges occur in Fig. 5 (left) because our implementation only evaluates the wave displacement $\boldsymbol{\eta}_\mathrm{c}$ at the *vertices* of the water mesh. Vertex positions below the terrain are then back-projected, and finally linearly interpolated across triangles. This order of operations—first projection, then interpolation—leads to the artifacts between projected and non-projected vertices. To remove them, we explicitly intersect the (undisplaced) water mesh with the terrain and retessellate boundary faces to generate vertices lying precisely on the waterline. Evaluating $\boldsymbol{\eta}_\mathrm{c}$ directly at these vertices without interpolation removes the artifacts as seen in Fig. 5 (right), even for a low-resolution water mesh. Note that using these mesh processing steps on Gerstner or Biesel waves without wave cages does not resolve their issues; this is illustrated in the supplemental video.

9. Implementation and Results

We base our implementation on water surface wavelets [JSMF*18] because they scale to large scenes and allow local control over wave orientation. Timings were taken on a laptop with NVIDIA® GeForce RTX 2080 MAX-Q graphics hardware.

The pre-processing steps from Sections 5–7 take a few seconds with timings given in Table 1. They need to be performed only once per scene, where our GPU shader implementation makes them suitable for execution at program startup. The parameters s and d are independent of grid resolution, and need not be adapted.

Table 1: *Timings in ms for the preprocessing steps that automatically compute the cages.*

Grid resolution	256^2	1024^2	4096^2
Initial cage fitting (Section 6)	0.2	1.5	15.1
Cage smoothing per iter. (Sec. 7)	1.9	20	324
Cage smoothing 100 iterations	191.9	1995.8	32436

The memory footprint of our method is minimal: we only need the local coordinate frame $\boldsymbol{A}$ and the cage radii $\boldsymbol{r}$ at every grid point of the terrain. Usually, the terrain normal is already in memory for shading, so a single extra scalar is required to define $\boldsymbol{A}$ uniquely. Due to the small overhead and ease of implementation of the regular grid data structure, we did not feel the need to optimize the data structure further. However, it should be possible to interpolate wave cage parameters on a sparser data structure if one wishes to do so.

Table 2 compares the rendering costs of Gerstner or Biesel waves with those incurred by using wave cages. The extra cost arises in the vertex shader, to evaluate the wave displacement according to Eq. (3), and in the pixel shader, to evaluate the water surface normal (see Appendix A). In both shader stages, the wave cage parameters

Figure 8: *Wave cages can also filter dynamically generated waves like this boat wake.*

are interpolated from neighboring points on the grid. Our unoptimized implementation renders the entire test scene, including water wave animation, particle system, and scene geometry, with 2xMSAA, at a speed of 70 fps at 720p, or 55 fps at 1080p.

Figures 1, 6, 7, and 9 as well as the supplemental video show how water animation with wave cages interacts with various types of environments, such as steep cliffs, gently sloped beaches, and narrow channels. Wave cages can also filter animations that are generated dynamically, making them a good fit for use in interactive applications like virtual reality and computer games. For example, Fig. 8 illustrates how wave cages enable boat wakes [JSMF*18] to roll up a beach naturally, without generating intersection artifacts, and the supplementary video shows the same scene in motion. Finally, the video also illustrates how:

- wave cage animation compares to Gerstner and Biesel waves (see the teaser image, Fig. 9, and from 0:10 in the video),
- letting the water wave amplitude fall off to zero towards the waterline does not yield plausible results (0:27),
- the compression factor s in Eq. (2) impacts the animation (1:40),
- the penetration depth threshold d (see Section 6) controls behavior at the waterline,
- cage smoothing improves visual quality (2:17),
- terrain back-projection and waterline re-tessellation remove remaining artifacts (2:33).

Our implementation is available as an executable binary that lets the user control the camera in the test scene shown in the video.

Extensions. In addition to the terrain that serves as an input to the wave-cage fitting process, the designer may optionally specify a "details" layer. This layer contains geometry that is too small to meaningfully interact with larger waves, such as small-scale rocks,

Table 2: *Timings in ms for the water surface rendering pass with and without our wave cages.*

Screen resolution (2x MSAA)	720p	1080p
Without wave cages	2.9	5.5
Wave cages	6.5	11.6

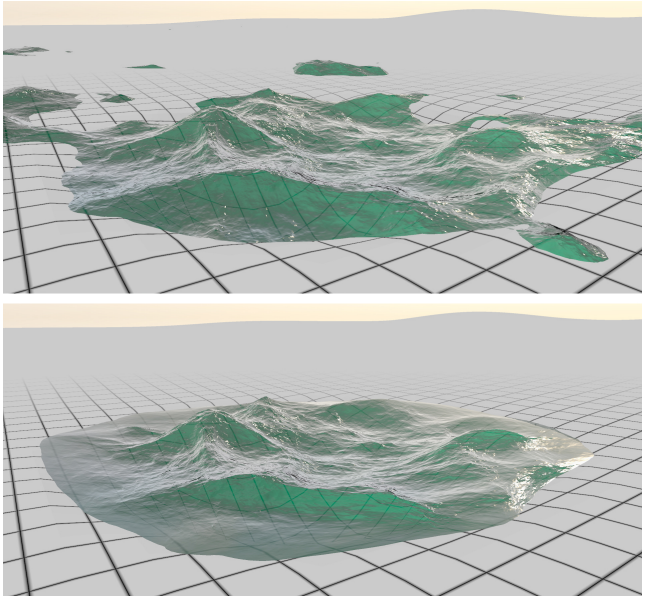


Figure 9: *Top: Gerstner waves interpenetrate the terrain. Bottom: the same animation frame filtered through our wave cages.*

very narrow columns, or beach debris. As such, it does not constrain the construction of wave cages, but interactions between waves and details are picked up by the particle system to improve visual fidelity. We illustrate this idea by animating waves colliding with rocks in our supplementary video.

The particle system is driven by the geometry, velocity, and acceleration of the water surface. It models spray from breaking waves, and foam from collisions between the water surface, the terrain, and the details layer. In addition, the particle system drives a textured foam layer on the water surface. Details on these extensions can be found in the supplemental document.

10. Limitations and Future Work

At the waterline, our wave cages compress the motion to lie on a plane; we currently rely on a depth offset variable d (Section 6) and projecting mesh vertices (Section 8) to enable interactions with curved terrains. However, we could imagine solving these problems with a more detailed curved cage instead. Our implementation also assumes the terrain is a heightfield, and it would be interesting to explore extensions to arbitrary 3D terrain geometry. Currently, we compute cages once as a pre-process and assume static terrain; we would like to explore dynamic cage computation in the future, which may allow interesting interactions with moving obstacles like boats.

Another limitation of the method, which we highlight in our video, is that our mesh processing takes place *per vertex*. Thus, we can guarantee that vertices of the water surface will never intersect the terrain, but that does not guarantee that all points within each triangle will be intersection-free. The problem is lessened if

we increase mesh resolution, but per-pixel computation might be required to fully solve the problem.

In addition, the approach presented here is purely geometric; it inherits the geometry of an existing procedural wave solver and warps it according to our constraints. It cannot currently prevent the tendency of Gerstner and Biesel waves from folding over themselves if their horizontal amplitude is too large. In the future, we would like to incorporate physical behaviors into the wave cages, to help remedy wave fold-overs, and to allow more realistic wave breaking and obstacle interactions.

Lastly, we note that we made no efforts to increase the efficiency of our spatial data structures. Although our video demonstrates that a regular grid implementation can simulate detailed waves interacting with large and complex island topography in real time, we believe this scaling can be improved even further in the future. Since many of the wave cage parameters are either constant or slowly-varying away from the waterline, we imagine that sparse data structures may be particularly effective at compressing data.

Acknowledgements

We wish to thank the anonymous reviewers and the members of the Visual Computing Group at IST Austria for their valuable feedback. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No. 638176 and No. 715767.

References

[AGL*17] AANJANEYA M., GAO M., LIU H., BATTY C., SIFAKIS E.: Power diagrams and sparse paged grids for high resolution adaptive liquids. *ACM Trans. Graph. 36*, 4 (July 2017). 2

[ATBG08] ANGST R., THÜREY N., BOTSCH M., GROSS M.: Robust and Efficient Wave Simulations on Deforming Meshes. *Computer Graphics Forum 27 (7)* (October 2008), 1895 – 1900. 2

[ATW13] ANDO R., THÜREY N., WOJTAN C.: Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph. 32*, 4 (2013). 2

[BG11] BARBARA S., GROSS M.: Two-scale particle simulation. *ACM Trans. Graph. 30*, 4 (2011), 1–8. 2

[Bie52] BIESEL F.: Study of wave propagation in water of gradually varying depth. In *Gravity Waves* (Nov. 1952), p. 243. 2

[Bri15] BRIDSON R.: *Fluid simulation for computer graphics*. AK Peters/CRC Press, 2015. 2

[CFL*07] CHENTANEZ N., FELDMAN B. E., LABELLE F., O'BRIEN J. F., SHEWCHUK J. R.: Liquid simulation on lattice-based tetrahedral meshes. In *Proceedings of SCA* (2007), pp. 219 – 228. 2

[CHCH06] CHEN Y.-Y., HSU H.-C., CHEN G.-Y., HWUNG H.-H.: Theoretical analysis of surface waves shoaling and breaking on a sloping bottom. part 2: Nonlinear waves. *Wave Motion 43*, 4 (2006), 339 – 356. 2

[Che20] CHEBLOKOV T.: Nvidia waveworks, 2020. URL: https://developer.nvidia.com/waveworks. 2

[CHH05] CHEN Y.-Y., HWUNG H.-H., HSU H.-C.: Theoretical analysis of surface waves propagation on sloping bottoms: Part 1. *Wave Motion 42*, 4 (2005), 335 – 351. 2

[CL95] CHEN J. X., LOBO N. D. V.: Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations. *Graph. Models Image Process. 57*, 2 (1995), 107–116. 2

[DCG11] DARLES E., CRESPIN B., GHAZANFARPOUR D.: A particle-based method for large-scale breaking wave simulation. *MG&V 20*, 1 (Jan. 2011), 3–25. 2

[FR86] FOURNIER A., REEVES W. T.: A simple model of ocean waves. *SIGGRAPH Comput. Graph. 20*, 4 (Aug. 1986), 75–84. 2

[FWD14] FERSTL F., WESTERMANN R., DICK C.: Large-scale liquid simulation on adaptive hexahedral grids. *IEEE Transactions on Visualization and Computer Graphics 20*, 10 (Oct 2014), 1405–1417. 2

[GLS00] GONZATO J.-C., LE SAËC B.: On modelling and rendering ocean scenes. *The Journal of Visualization and Computer Animation 11*, 1 (2000), 27–37. 2

[HHL*05] HAGEN T. R., HJELMERVIK J. M., LIE K.-A., NATVIG J. R., HENRIKSEN M. O.: Visual simulation of shallow-water waves. *Simulation Modelling Practice and Theory 13*, 8 (2005), 716–726. 2

[HPSZ11] HARMON D., PANOZZO D., SORKINE O., ZORIN D.: Interference-aware geometric modeling. *ACM Trans. Graph. 30*, 6 (2011), 137. 4

[Joh97] JOHNSON R. S.: *A Modern Introduction to the Mathematical Theory of Water Waves*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 1997. 1, 2

[JSMF*18] JESCHKE S., SKŘIVAN T., MÜLLER-FISCHER M., CHENTANEZ N., MACKLIN M., WOJTAN C.: Water surface wavelets. *ACM Trans. Graph. 37*, 4 (July 2018). 2, 5, 6

[JW15] JESCHKE S., WOJTAN C.: Water wave animation via wavefront parameter interpolation. *ACM Trans. Graph. 34*, 3 (May 2015), 27:1–27:14. 2

[JW17] JESCHKE S., WOJTAN C.: Water wave packets. *ACM Trans. Graph. 36*, 4 (July 2017). 2

[KAD*06] KEISER R., ADAMS B., DUTRÉ PH., GUIBAS L., PAULY M.: *Multiresolution particle-based fluids*. Technical Report 520, Department of Computer Science, ETH Zurich, 2006. 2

[KBST19] KOSCHIER D., BENDER J., SOLENTHALER B., TESCHNER M.: Smoothed particle hydrodynamics techniques for the physics based simulation of fluids and solids. In *Eurographics 2019 - Tutorials* (2019). 2

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. *ACM Trans. Graph. 24*, 4 (Sept. 1990), 49–57. 2

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 457 – 462. 2

[LH10] LEE H., HAN S.: Solving the shallow water equations using 2d sph particles for interactive applications. *The Visual Computer 26* (2010), 865–872. 2

[LvdP02] LAYTON A. T., VAN DE PANNE M.: A numerically efficient and stable algorithm for animating water waves. *The Visual Computer 18*, 1 (2002), 41–53. 2

[MC11] MÜLLER M., CHENTANEZ N.: Solid simulation with oriented particles. *ACM Trans. Graph. 30*, 4 (July 2011). 3

[MM13] MACKLIN M., MÜLLER M.: Position based fluids. *ACM Trans. Graph. 32*, 4 (July 2013). 2

[MWM87] MASTIN G. A., WATTERBERG P. A., MAREDA J. F.: Fourier synthesis of ocean scenes. *IEEE Comput. Graph. Appl. 7*, 3 (Mar. 1987), 16–23. 2

[OH95] O'BRIEN J. F., HODGINS J. K.: Dynamic simulation of splashing fluids. In *Proceedings of Computer Animation* (may 1995), vol. 95, pp. 198 – 205. 2

[Pea86] PEACHEY D. R.: Modeling waves and surf. *SIGGRAPH Comput. Graph. 20*, 4 (Aug. 1986), 65–74. 2

[SBC*11] SOLENTHALER B., BUCHER P., CHENTANEZ N., MÜLLER M., GROSS M. H.: Sph based shallow water simulation. In *VRIPHYS* (2011). 2

[SHW19] SCHRECK C., HAFNER C., WOJTAN C.: Fundamental solutions for water wave animation. *ACM Transactions on Graphics (TOG) 38*, 4 (2019), 1–14. 2

[SSJ*20] SKŘIVAN T., SÖDERSTRÖM A., JOHANSSON J., SPRENGER C., MUSETH K., WOJTAN C.: Wave curves: Simulating lagrangian water waves on dynamically deforming surfaces. *ACM Transactions on Graphics (TOG) 39*, 4 (2020). 2

[Tes99] TESSENDORF J.: Simulating ocean water. *SIGGRAPH course notes* (1999), 1–19. 1, 2

[vBBK08] ŠŤAVA O., BENEŠ B., BRISBIN M., KŘIVÁNEK J.: Interactive terrain modeling using hydraulic erosion. In *Proc. SCA* (2008), pp. 201–210. 2

[WMT07] WANG H., MILLER G., TURK G.: Solving general shallow wave equations on surfaces. In *Proc. SCA* (2007), pp. 229–238. 2

[YHK07] YUKSEL C., HOUSE D. H., KEYSER J.: Wave particles. *ACM Trans. Graph. 26*, 3 (July 2007), 99 – 107. 2

[ZHT07] ZHANG E., HAYS J., TURK G.: Interactive tensor field design and visualization on surfaces. *IEEE Transactions on Visualization and Computer Graphics 13*, 1 (Jan. 2007), 94–107. 3

[ZLC*13] ZHU B., LU W., CONG M., KIM B., FEDKIW R.: A new grid structure for domain extension. *ACM Trans. Graph. 32*, 4 (July 2013). 2

Appendix A: Derivatives within a wave cage

The final water surface is given by $\boldsymbol{\sigma}(\boldsymbol{x}) = x\boldsymbol{e}_1 + y\boldsymbol{e}_2 + \boldsymbol{\eta}_\mathrm{c}(\boldsymbol{x}, \boldsymbol{\eta}(\boldsymbol{x}))$, with $\boldsymbol{x} = (x, y)$ a point on the water surface at rest, and $\boldsymbol{\eta}_\mathrm{c}$ the cage-transformed wave displacement vector defined in Eq. (3). To shade $\boldsymbol{\sigma}$ in a rendering framework, we need to compute the surface normal, which is derived below. Additionally, we provide first and second time derivatives of $\boldsymbol{\eta}_\mathrm{c}$, which are useful for spawning foam particles or other secondary simulations.

The (unnormalized) surface normal is given by

$$\boldsymbol{n} = \frac{\partial \boldsymbol{\sigma}}{\partial x} \times \frac{\partial \boldsymbol{\sigma}}{\partial y} = \left(\boldsymbol{e}_1 + \frac{d\boldsymbol{\eta}_\mathrm{c}}{dx}\right) \times \left(\boldsymbol{e}_2 + \frac{d\boldsymbol{\eta}_\mathrm{c}}{dy}\right),$$

where the total derivatives are computed as

$$\frac{d\boldsymbol{\eta}_\mathrm{c}}{d\boldsymbol{x}} = \frac{\partial \boldsymbol{\eta}_\mathrm{c}}{\partial \boldsymbol{x}} + \boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^T \frac{\partial \boldsymbol{\eta}_\mathrm{n}}{\partial \boldsymbol{x}}.$$

The first term in the sum involves the spatial derivatives of $\boldsymbol{A}$ and $\boldsymbol{r}$, which are quantities stored on the simulation grid, and can be computed by sampling. However, neglecting this term does not lead to visual artifacts, because the characteristic look of light reflected on the waves is captured by the second term. Similarly,

$$\frac{\partial \boldsymbol{\eta}_\mathrm{c}}{\partial t} = \boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^T \frac{\partial \boldsymbol{\eta}_\mathrm{n}}{\partial t}, \quad \text{and} \quad \frac{\partial^2 \boldsymbol{\eta}_\mathrm{c}}{\partial t^2} = \boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^T \frac{\partial^2 \boldsymbol{\eta}_\mathrm{n}}{\partial t^2}$$

for time derivatives. We note that $\boldsymbol{\eta}_\mathrm{n}$ depends on $\boldsymbol{x}$ and t only indirectly through the Gerstner wave displacement $\boldsymbol{\eta}$ defined in Eq. (1). Thus we can compute first derivatives of $\boldsymbol{\eta}_\mathrm{n}$ as

$$\boldsymbol{\eta}'_\mathrm{n} = \begin{cases} \left(a_2 - \frac{a_1}{\|\boldsymbol{\eta}\|}\right) \frac{\boldsymbol{\eta}\cdot\boldsymbol{\eta}'}{\|\boldsymbol{\eta}\|^2}\boldsymbol{\eta} + \frac{a_1}{\|\boldsymbol{\eta}\|}\boldsymbol{\eta}', & \text{if } \boldsymbol{\eta} \neq \boldsymbol{0}, \\ s\boldsymbol{\eta}', & \text{otherwise.} \end{cases} \tag{5}$$

where $a_1 = \tanh(s\|\boldsymbol{\eta}\|)$, and $a_2 = s\,\mathrm{sech}^2(s\|\boldsymbol{\eta}\|)$. To compute the derivatives of $\boldsymbol{\eta}_\mathrm{n}$ wrt x, y, or t, replace $\boldsymbol{\eta}'$ with the corresponding derivatives $\partial\boldsymbol{\eta}/\partial x$, $\partial\boldsymbol{\eta}/\partial y$ or $\partial\boldsymbol{\eta}/\partial t$. This enables the computation of normals and velocities of surface points.

The acceleration can be computed from

$$\frac{\partial^2 \boldsymbol{\eta}_\mathrm{n}}{\partial t^2} = \begin{cases} \left(\frac{a_1}{\|\boldsymbol{\eta}\|} - a_2\right)\left[\left(3\left(\frac{\boldsymbol{\eta}\cdot\boldsymbol{\eta}'}{\|\boldsymbol{\eta}\|^2}\right)^2 - \frac{\boldsymbol{\eta}\cdot\boldsymbol{\eta}'' + \|\boldsymbol{\eta}'\|^2}{\|\boldsymbol{\eta}\|^2}\right)\boldsymbol{\eta} - 2\frac{\boldsymbol{\eta}\cdot\boldsymbol{\eta}'}{\|\boldsymbol{\eta}\|^2}\boldsymbol{\eta}'\right] & \\ \quad + \frac{a_1}{\|\boldsymbol{\eta}\|}\left(\boldsymbol{\eta}'' - 2sa_2\frac{(\boldsymbol{\eta}\cdot\boldsymbol{\eta}')^2}{\|\boldsymbol{\eta}\|^2}\boldsymbol{\eta}\right), & \text{if } \boldsymbol{\eta} \neq \boldsymbol{0}, \\ s\boldsymbol{\eta}'', & \text{otherwise,} \end{cases}$$

where $\boldsymbol{\eta}'$ and $\boldsymbol{\eta}''$ refer to the first and second time derivatives of $\boldsymbol{\eta}$.

Appendix B: Jacobi's Method for Cage Smoothing

Section 7 describes an optimization problem for smoothing the grid of wave cage radii $\boldsymbol{r}$ in order to preserve the behavior of waves with large wavelengths during animation. These types of waves have displacement fields $\boldsymbol{\eta}_\mathrm{u} = \boldsymbol{A}\boldsymbol{R}\boldsymbol{A}^T\boldsymbol{u}$ for some constant $\boldsymbol{u} \in \mathbb{R}^3$, so the goal is to smooth these surfaces on average for all $\boldsymbol{u}$ with $\|\boldsymbol{u}\| = 1$.

Of course, this optimization problem can be solved using standard methods, such as sequential quadratic programming. Here, we describe an alternative that requires no heavy-duty numerical solver and can be implemented in a few lines of code.

First, approximate the integral over S^2 with a sum over a finite set of samples $U = \{\boldsymbol{u}_i\}_{i=1}^n$ that are distributed uniformly over the sphere (we use $n = 20$). This transforms the integrated energy into a sum over the Dirichlet energies of n different surfaces $\boldsymbol{\eta}_\mathrm{u}(\boldsymbol{u}_i)$.

Jacobi's method for minimizing a Dirichlet energy on a regular grid proceeds by iteratively replacing the function value at a point by the average function value of its neighbors. This is a fixed-point iteration method, because the minimizer of the Dirichlet energy is harmonic, i.e., the function value at a point is *exactly* the average function value of its neighbors.

We adapt this method for our energy. In the following, let $\boldsymbol{x}_i$ be the position of the i-th node of the grid, N_i the indices of its neighbors, and $\boldsymbol{r}_i = \{r_{i,1}, r_{i,2}, r_{i,3}\}$ its cage radii in the current iteration. Then iterate the following steps:

1. For each node i of the grid:
   a. For each $k \in \{1, \ldots, n\}$, let
   $$\boldsymbol{p}_k := \frac{1}{|N_i|}\sum_{j\in N_i} \boldsymbol{\eta}_\mathrm{u}(\boldsymbol{x}_j, \boldsymbol{r}_j, \boldsymbol{u}_k),$$
   which is the average wave displacement of the node's neighbors associated with the wave $\boldsymbol{u}_k$.
   b. Compute $\tilde{\boldsymbol{r}}_i \in \mathbb{R}^3$ as the minimizer of
   $$\sum_{k=1}^{n} \|\boldsymbol{\eta}_\mathrm{u}(\boldsymbol{x}_i, \tilde{\boldsymbol{r}}_i, \boldsymbol{u}_k) - \boldsymbol{p}_k\|^2,$$
   i.e., choose $\tilde{\boldsymbol{r}}_i$ such that the wave displacements $\boldsymbol{\eta}_\mathrm{u}$ at node i are closest to the average wave displacements of its neighbors in a least-squares sense. Note that $\boldsymbol{\eta}_\mathrm{u}$ is linear in $\boldsymbol{r}$, so this can be done by solving a single 3-by-3 linear system.
   c. Set the new value of $\boldsymbol{r}_i$ to $(1-\alpha_i)\tilde{\boldsymbol{r}}_i + \alpha_i\bar{\boldsymbol{r}}_i$ and clamp its components to satisfy $0 \le r_{i,j} \le \bar{r}_{i,j}$ for $j \in \{1, 2, 3\}$.