Proceedings of the 6th Annual North American Conference on AI and Simulation in Games (GAMEON-NA), pp. 27-34, EUROSIS, Troy, NY, 2011.

VERY FAST REAL-TIME OCEAN WAVE FOAM RENDERING USING HALFTONING

Mary Yingst
Dept. of Computer
Science & Engineering
University of North Texas
Denton, TX, USA
maryyingst@my.unt.edu

Jennifer R. Alford
Digital Teapot, Inc.
Fort Worth, TX, USA
gralford@acm.org

Ian Parberry
Dept. of Computer
Science & Engineering
University of North Texas
Denton, TX, USA
ian@unt.edu

KEYWORDS

Physics and Simulation, Design

ABSTRACT

We introduce an efficient method for emulating sea foam dissipation suitable for use in real-time interactive environments such as video games. By using a precomputed dither array with controlled spectral characteristics adopted from halftone research as a control mechanism in the pixel shader, we can animate the appearance of foam bubbles popping in a random manner while allowing them to clump naturally.

INTRODUCTION

Real-time animation and rendering of ocean waves is often seen in video games, and adding foam to the waves lends an added level of realism. We describe a fast and effective method for rendering ocean wave foam by augmenting traditional texture based foam saturation methods with techniques from halftoning.

Takahashi et al. (2003) and Thürey et al. (2007) represent foam as a particle system. Although this is visually pleasing, it is computationally intensive. In large scale environments such as the ocean it is more practical to use faster texture based methods. Many methods of rendering foam rely on applying a texture of foam to the water surface. These methods apply a texture using a foam *saturation*, or *density* value to represent transparency of the texture which is applied to a mesh representing the water's surface (see, for example, Jensen and Golias (2001), Jeschke et al. (2003), and Kryachko (2005)). Li et al. (2008) similarly apply a foam color according to its density.

Real ocean foam consists of bubbles clumped together by surface tension on the surface of the water. Foam does not simply fade or become transparent as the bubbles dissipate. Traditional methods of foam generation ignore the active nature of foam density where bubbles pop over time. Since surface bubbles are either present or not in an area of water, this binary nature lends itself to the use of halftoning, a process used to reproduce images using patterns of black dots. Our use of halftoning with a saturation function that changes over time causes bubbles to appear to pop.

The remainder of this note is divided into five sections. First we give a high-level overview of our approach. Then we review in more depth our choice of foam saturation function, our use of a halftoning mask generated using methods from the halftoning literature, and how we apply that mask in a pixel shader. Finally we conclude with a discussion of our results.

OVERVIEW OF OUR APPROACH

To generate foam on the surface of the water using a foam saturation function, we must create the water surface as a mesh. Each location on the water's surface has a calculable saturation value using this function. The function must vary over time for the foam to animate and become more and less dense as waves pass and change. In figure 1 we see that by applying halftoning methods to a saturation function, we take an otherwise smooth area of the function and create the randomness expected when foam generates and dissipates.

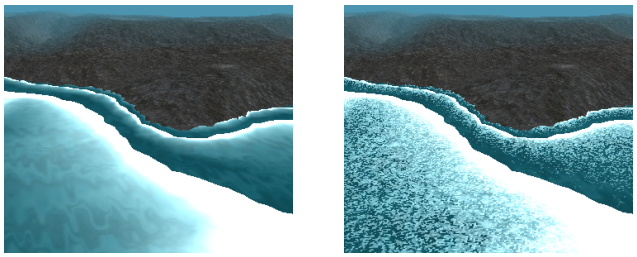


Figure 1: By replacing the application of a foam texture with a white tone we see that applying our method creates randomness on the right in the otherwise smooth saturation results pictured on the left.

Halftone masks, or *dither arrays*, are arrays of values that have a one-to-one correspondence with pixels in an image, or in our application, a texture. Each value of the halftone mask is used as a threshold against the corresponding texture

pixel to produce a binary output image that indicates, at each pixel position, whether the texture falls above or below the threshold. This process is commonly referred to as *thresholding*. Halftone masks are characterized by the binary pattern that results when thresholded against a constant image, or texture. Choosing threshold value values at each mask position is non-trivial. Ulichney (1987) provides a classic study of mask design and describes widely used metrics, based on the Fourier Transform, to characterize masks by their radially averaged power spectrum (RAPS), a measure of energy at different frequency bands, and anisotropy, a measure of radial symmetry. While halftoning can be accomplished with a variety of computational methods, we restrict ourselves to the use of masks because, as point operations, they are computationally efficient and naturally suited to pixel shader operations.

We depart from the traditional use of halftoning in printing and image display, which seeks to reduce visually objectionable noise in image reproduction, and instead we use a halftone mask to add noise. We draw on recent work in halftone mask design by observing that it is possible to design masks to produce lumping binary patterns which are reminiscent of the clumping of sea foam. We also observe that the binary nature of the threshold output is well-suited to simulate foam bubble popping when the mask is fixed per frame but the underlying image is not. In this work, we present a novel way to use halftone masks in conjunction with a saturation function and a texture to simulate foam and the popping behavior of foam. Further, we observe that the difference between the threshold value and an image or a texture provides a magnitude at each pixel position that we use as a transparency value for additional realism.

We use halftone masks that have been generated using a symmetric Gaussian function to filter white noise as described in Alford and Sheppard (2010). Gaussian filtering applies a two-dimensional Gaussian function to an image. σ is a value in the Gaussian function that denotes the width of the curve in the function; as σ increases, the width of the curve increases.

We simulate the effect of foam bubbles popping by finding the saturation of foam on the water's surface and applying a precomputed halftone mask to it. We use a modified version of the vertex shader outlined in a paper by Van Drasek III et al. (2010) to create parametric waves upon which to apply our foam. The next two sections will describe the saturation function and the halftone mask in more detail.

THE SATURATION FUNCTION

Kryachko (2005) uses the following foam saturation function which is dependent on ocean height. H_0 is base height, H is height, and $H_{\max}$ is height where foam is maximum.

$$f(x) = \frac{H - H_0}{H_{\max} - H_0}$$

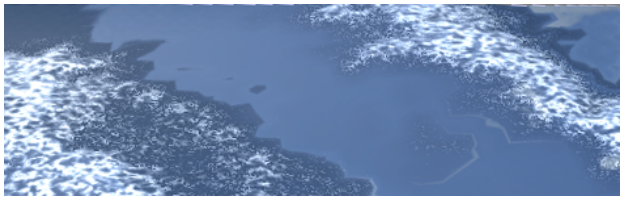


Figure 2: Foam with Kryachko's saturation function.

Although Kryachko's function achieves somewhat attractive results (see Figure 2 for example), the function results in a symmetric foam distribution, whereas we wish to model foam that is created by turbulence at the front of the wave and fades away behind it. Knowing the target foam density along the wave shape, we chose to apply $e^{\tan(x)}$ to the same vector and frequency used to determine wave shape.

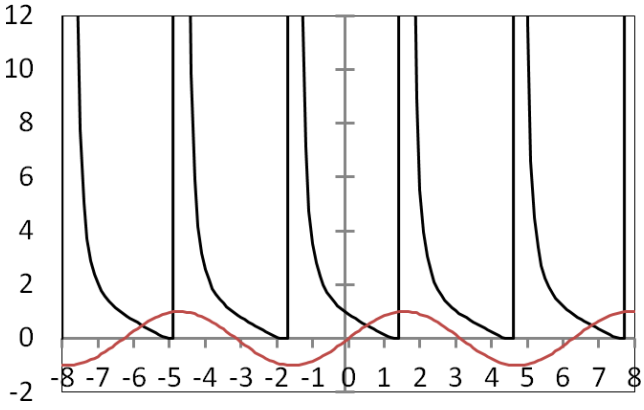


Figure 3: $e^{\tan(x)}, \sin(x)$

We use the following formulae from Van Drasek III et al. (2010) for the height y of the wave:

$$\begin{aligned} y &= A((\sin(\theta(x,z)) + 1)/2)^K \\ \theta(\vec{v}) &= (\vec{v} \cdot \vec{k})2\pi/\lambda_{adj} + \phi t \\ \phi &= 2s\pi/\lambda, \end{aligned}$$

and so we use $\theta(\vec{v})$ to also generate the periodic function.

$$f(\vec{v}) = e^{-\tan((\vec{v}\cdot\vec{k})2\pi/\lambda_{adj} + \phi t))},$$

where $\vec{v} = (x, z)$ is position, $\vec{k}$ is the wave direction, s is the speed of the wave, t is time, K is wave slope, A is wave amplitude, λ_{adj} is wavelength adjusted for ocean depth, and λ is original wavelength.

Since we are overlaying this function on the sine function that determines wave shape, we need to modify the formula slightly to align the foam with the waves. In Figure 3 we see that $e^{\tan(x)}$ is twice as frequent as $\sin(x)$, so we divide $\theta(x, z)$ by 2. Also to align the highest part of our function with the front part of the sine wave we add $\pi/2$. Our final formula is as follows, and gives a attractive saturation of

foam starting at the wave front and fading behind it.

$$f(\vec{v}) = (e^{-\tan((\vec{v}\cdot k)\pi/\lambda_{adj}+\phi t/2)+\pi/2)})/C,$$

where C is a user defined constant that governs the intensity of the foam. (We use $C = 4$ for convenience, but this value may be tuned by the designer.)

Saturation is computed as follows. Adjwavelength, and phaseC are calculated in the vertex shader and the values are interpolated for use in the pixel shader.

```
float getmysaturation(float2
  wavedirection, float2 xzposition,
  float Adjwavelength, float phaseC)
{
  float result =
    dot(wavedirection, xzposition)
    *6.28f/Adjwavelength;
  result = result + phaseC*gTimeNow;
  result = pow(2.718f,-1.0f*
  tan((result/2.0f)+ 1.07f))/4.0f;
  return result;
}
```

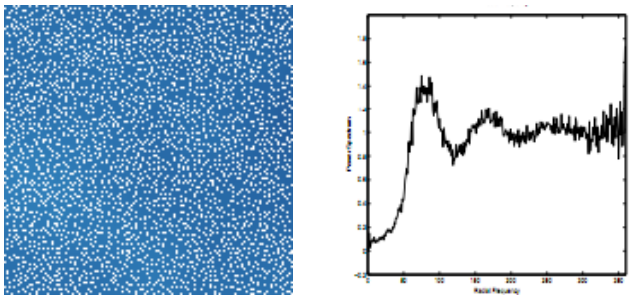
To pass values from the vertex shader we simply define an extra variable in the vertex output with a TEXCOORD semantic. Then the vertex shader sets the required values as follows:

```
struct VertexOutput
{
  ...
  float4 impVars : TEXCOORD4;
}

VertexOutput VS(...)
{
  VertexOutput OUT = (VertexOutput)0;
  ...
  OUT.impVars[1]=adjustedWavelength;
  OUT.impVars[0]=phaseConstant;
  OUT.impVars[2] = Po[0]; //xposition
  OUT.impVars[3] = Po[2]; //zposition
}

float4 PS(VertexOutput IN) : COLOR
{
  float saturated=getmysaturation
    (direction, float2(IN.impVars[2],
    IN.impVars[3]), IN.impVars[1],
    IN.impVars[0]);
  ...
}
```

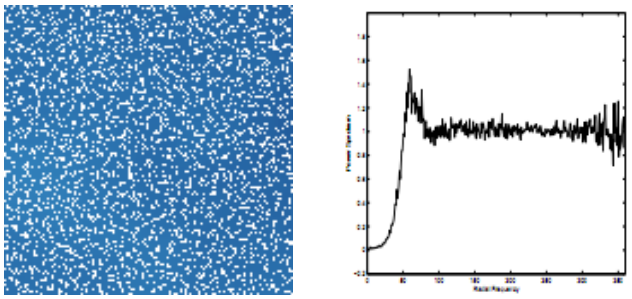
THE HALFTONE MASK

We use a halftone mask to threshold the saturation function to create dissipation through bubble popping. As saturation decreases over time at a specific location, the value will approach and pass the threshold used in our mask. While the saturation value is above the threshold, the foam will be present, but as time passes and the value decreases, eventually the foam will *pop* and dissappear. Since bubbles in foam clump, we must choose a halftone mask that produces clumps in the resulting dot patterns. Clumpiness, or clustering, can be seen in how close together some of the foam is while in other areas there are gaps.

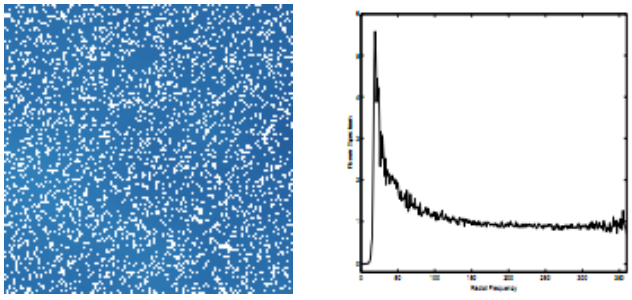
Alford and Sheppard (2010) show a variety of halftone masks created using radially symmetric Gaussian filters. We used their masks created using filters having σ ranging from 1.5 to 24 to produce the images in Figure 4 column 1. In Figure 4 we can see that the higher the σ, the closer together some of the dots are. By analyzing the RAPS we see that as σ increases, first oscillation is dampened in the high frequencies, then the values of the high frequency region is greatly reduced (Alford and Sheppard 2010). The results of this can be seen in the increased clustering and clumping behavior of the dot patterns. We found $\sigma = 24$ gives adequate visual clusters of foam.



(a) $\sigma = 1.5$



(b) $\sigma = 6$



(c) $\sigma = 24$

Figure 4: Halftone masks created by Gaussian filters having σ ranging from 1.5 to 24, with corresponding RAPS (images courtesy Alford and Sheppard (2010)).

APPLYING THE MASK

To create the halftoned saturation function $h(u, v)$ where u, v are texture coordinates and $h(u, v)$ is a `float4` RGBA color value at that position, we first create a texture to contain the mask information so that the data can be imported into the pixel shader. Given a 512×512 halftone mask, a 512×512 pixel texture is generated. This texture, when tiled across the surface of the water, has a corresponding u, v texture coordinate for each $\vec{v} = (x, z)$ position on the water. The mask value $m(u, v)$ can then be used to threshold the saturation function $f(x, z)$ as follows:

$$h(u,v) = \begin{cases} (0,0,0,1) & \text{if } f(x,z) \leq m(u,v) \\ (1,1,1,1) & \text{if } f(x,z) > m(u,v) \end{cases} \quad (1)$$

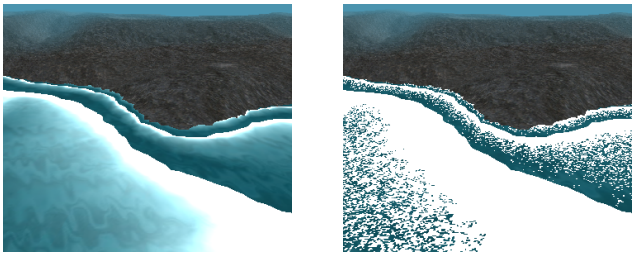


Figure 5: Applying Equation 1 to the saturation function at left gives the image at right.

We can then create a fading halftoned saturation function, $g(u, v)$, so the dots fade before they pop. We do this by taking the difference between saturation and mask number. Figure 1 shows the results of applying halftoning with fading to the saturation function. For all $g(u, v)$, $\alpha = 1$.

$$g(u,v).rgb = \text{clamp}(f(x,z)/2 - m(u/v), 0, 1) * h(u,v).rgb \quad (2)$$

Finally we apply $t(u, v)$, the foam texture to generate the final halftoned, textured, and faded image $j(u, v)$. For all $j(u, v)$, $\alpha = 1$.

$$j(u,v).rgb = \begin{cases} (0,0,0) & \text{if } f(x,z) \leq m(u,v) \\ g(u,v) * t(u,v).rgb & \text{if } f(x,z) > m(u,v) \end{cases} \quad (3)$$

Given a sampler for the halftone mask texture, MaskSampler; a sampler for the foam texture, SAMP_FoamTexture; and a sampler for the water surface texture, SAMP_WaterTexture; the following code finds the resulting color for the water's surface. The higher TEXscale or MASKscale is, the smaller the tiled texture will appear. A value of 400 for MASKscale gives suitably sized dots when using a 512×512 pixel mask.

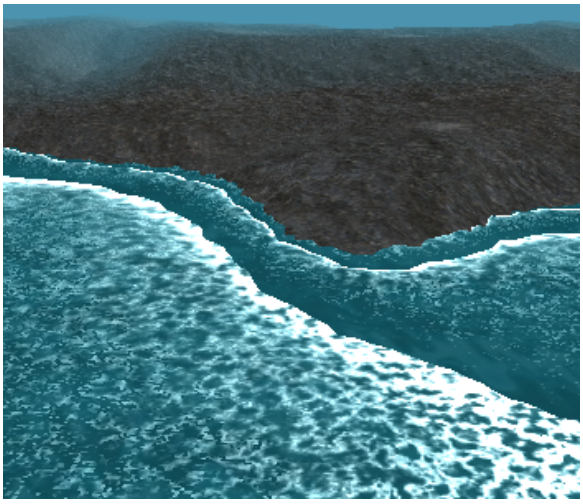


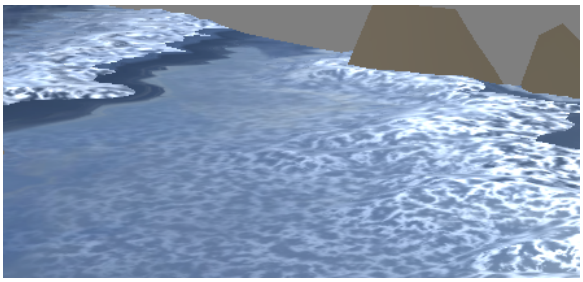
Figure 6: Coastline image using our new halftoning method, Equation 3.

```
//get water and foam texture color
float4 textureSamp = tex2D(
  SAMP_WaterTexture,
  IN.TexCoord1*TEXscale);
float4 foamSamp = tex2D(
  SAMP_FoamTexture,
  IN.TexCoord1*TEXscale);


//get the threshold from the mask
float masknumber=(tex2Dlod(
  MaskSampler, float4(IN.TexCoord1.xy
  *MASKscale,0,0)));

//threshold the saturation value
if(!((saturated)>(masknumber))){
        foamSamp[0] = 0;
        foamSamp[1] = 0;
        foamSamp[2] = 0;
}


//find the value for fading the foam
float difference = clamp(saturated
  -     masknumber, 0.15f, 3.0f);

//get the final foam value
foamSamp = difference * foamSamp;

//add the value to the water texture
//and clamp to a valid color
float4 result =clamp((textureSamp+
  foamSamp),0,1);
result[3] = 1.0f;
```

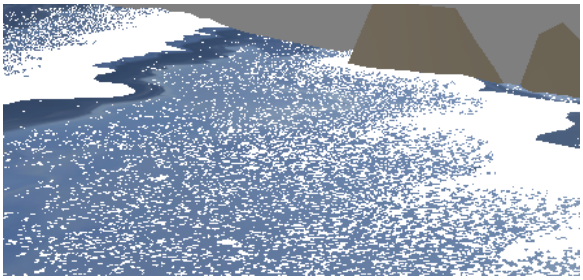
RESULTS

Figure 7(a) shows the traditional method of fading a foam texture according to a saturation function, similar to Kryachko (2005). Figure 7(b) shows the saturation halftoned us-

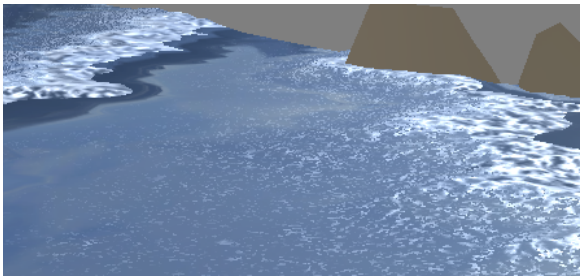
ing Equation 1 and no other functions applied. This method shows a realistic popping effect, but the foam is too harsh and white. Figure 7(c) shows our halftoning method in combination with a foam texture using Equation 3.



(a) Using a foam texture.



(b) Using a halftone mask to determine foam location.



(c) Using a halftone mask with a foam texture.

Figure 7: The results of using 3 different methods with the same settings (heightmap, wave speed, direction, and amplitude).

We performed some experiments to obtain a preliminary benchmark for the extra computation load required by our new halftoning technique (Figure 7(c)) to the traditional texturing technique (Figure 7(a)). We ran both algorithms for five minutes using NVidia Composer, using FRAPS to measure average frames-per-second. The scene rendered in all

Graphics Card	Textured	Halftoned
NVidia 8800	65.5fps	65.0fps
NVidia GeForce GT320m	80.4fps	78.2fps
Intel HD Graphics 3000	85.0fps	84.5fps

Table 1: Comparison of rendering frame rates in frames per second (fps).

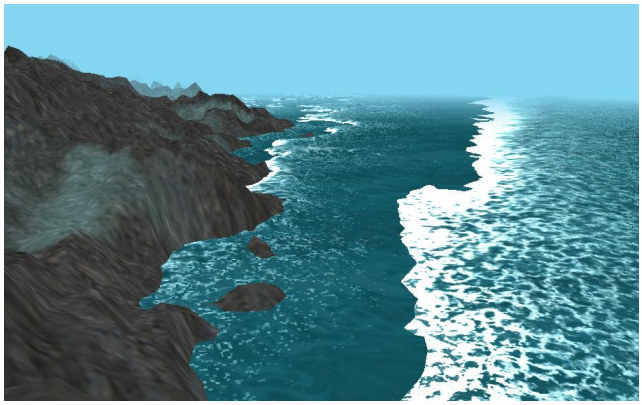


Figure 8: Scene used for measuring frame rates.

experiments is shown in Figure 8. The results are shown in Table 1. We conclude that the extra load on the video appears to be less that 3% higher than traditional texture-fading techniques, which is negligible.

Still pictures such as shown in Figure 7 and Figure 8 do not adequately capture the full effect of our algorithm. Figure 9 shows how foam bubbles fade and pop over time in the wake of each wave. This can be seen to best advantage in an animation such as the one we have placed online at Yingst et al. (2011).

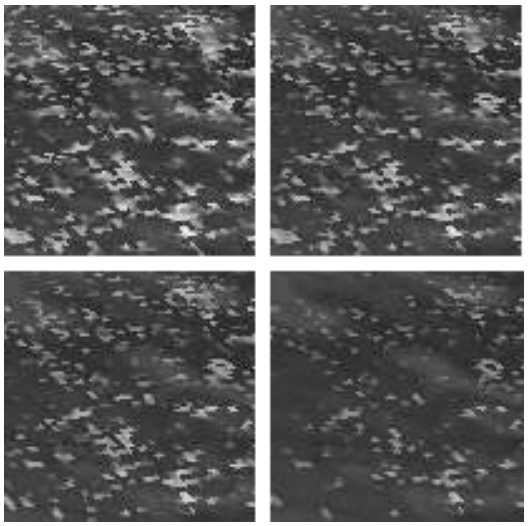


Figure 9: Close up view of foam bubbles fading and popping over time.

CONCLUSION AND FURTHER WORK

Not only does our halftoning technique achieve our goal of simulating foam dissipation in a real-time environment, but it also can be applied with little additional cost to traditional texture based methods that obtain foam saturation at the wa-

ter's surface. The saturation function used must vary over time for the bubble popping effect to occur using the halftoning method.

Our method currently produces pixelation at close range to the camera. One method for remedying this would be a second pass of a pixel shader to smooth the edges of the generated texture, which we leave as future work.

REFERENCES

Alford J.R. and Sheppard D.G., 2010. *Approximating Poisson Disk Distributions by Means of a Stochastic Dither Array*. In *EG UK Theory and Practice of Computer Graphics*.

Jensen L.S. and Golias R., 2001. *Deep-Water Animation and Rendering*. URL `www.gamasutra.com/gdce/2001/jensen/jensen_01.htm`. Presented at Game Developers Conference, Europe.

Jeschke S.; Birkholz H.; and Schmann H., 2003. *A Procedural Model for Interactive Animation of Breaking Ocean Waves*. In *Proceedings of WSCG 2003*. WSCG.

Kryachko Y., 2005. *Using Vertex Texture Displacement for Realistic Water Rendering*. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*.

Li Y.; Jin Y.; Yin Y.; and Shen H., 2008. *Simulation of shallow-water waves in coastal region for marine simulator*. In *Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*. ACM, 15:1–15:5.

Takahashi T.; Fujii H.; Kunimatsu A.; Hiwada K.; Saito T.; Tanaka K.; and Ueki H., 2003. *Realistic Animation of Fluid with Splash and Foam*. *Computer Graphics Forum*, 22, no. 3, 391–400.

Thürey N.; Sadlo F.; Schirm S.; Müller-Fischer M.; and Gross M., 2007. *Real-time simulations of bubbles and foam within a shallow water framework*. In *Proceedings of the 2007 ACM SIGGRAPH Eurographics Symposium on Computer Animation*. Eurographics Association.

Ulichney R., 1987. *Digital Halftoning*. Cambridge, Mass: The MIT Press.

Van Drasek III J.; Bookout D.; and Lake A., 2010. *Real-Time Parametric Shallow Wave Simulation*. URL `http://software.intel.com/sites/billboard/article-archive/real-time-parametric/`.

Yingst M.; Alford J.R.; and Parberry I., 2011. *Sea Foam*. URL `http://larc.unt.edu/ian/research/seafoam/`.

BIOGRAPHY

MARY YINGST was born and raised in Texas. She received her BS in computer science in 2007, and is a recent graduate of the University of North Texas, College of Engineering with a MS in Computer Science. She has participated in the game development program at UNT for several years, fostering her research interests which include graphics for game development and real-time simulation. Her Erdös number is 4. Her home page is `http://maryingst.net`.

JENNIFER (GINGER) ALFORD is Director of Computer Sciences at Trinity Valley School and President of Digital Teapot, Inc, (www.digitalteapot.org). She holds a PhD in electrical and computer engineering from the University of Iowa with a specialization in image processing and particular expertise in halftoning. Her 25 years experience in image processing and computer graphics include industrial research and software development, serving as an technical consultant and expert witness for intellectual property attorneys, college and graduate level teaching, and several academic publications. Dedicated to research and education, she has partnered with the Laboratory for Recreational Computing at the University of North Texas as a Research Associate.

IAN PARBERRY was born in London, England and emigrated as a child with his parents to Brisbane, Australia. After obtaining his undergraduate degree there from the University of Queensland he returned to England for a PhD from the University of Warwick. He has worked in academia in the US ever since. He is currently a full Professor in the Department of Computer Science and Engineering at the University of North Texas where he recently stepped down from a 2-year term as Interim Department Chair. A pioneer of the academic study of game development since 1993, his undergraduate game development program was ranked in the top 50 out of 500 in North America by The Princeton Review in 2010. He is on the Editorial Boards of the *Journal of Game Design and Development Education*, *IEEE Transactions on Computational Intelligence and AI in Games*, and *Entertainment Computing*, and he serves as the Secretary of the Society for the Advancement of the Science of Digital Games, which organizes the Annual Foundations of Digital Games conference. He is the author of 6 books and over 80 articles over 30 years' experience in academic research and education. His *h*-index is 18 and his Erdös number is 3. He can be contacted at `ian@unt.edu` or on Facebook. His home page is `http://larc.unt.edu/ian`.