BEAUTY WAVES: AN ARTISTIC REPRESENTATION OF OCEAN WAVES USING BEZIER CURVES

A Thesis

by

JAY ALLEN FAULKNER

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2006

Major Subject: Visualization Sciences

BEAUTY WAVES: AN ARTISTIC REPRESENTATION OF OCEAN WAVES USING BEZIER CURVES

A Thesis

by

JAY ALLEN FAULKNER

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Ergun Akleman
Committee Members,	Karen Hillier
	John Keyser
Head of Department,	Mark Clayton

December 2006

Major Subject: Visualization Sciences

ABSTRACT

Beauty Waves: An Artistic Representation of
Ocean Waves Using Bezier Curves. (December 2006)
Jay Allen Faulkner, B.S., Texas A&M University
Chair of Advisory Committee: Dr. Ergun Akleman

In this thesis, we present a method for computing an artistic representation of ocean waves using Bezier curves. Wave forms are loosely based on procedural wave models and are designed to emulate those found in both art and nature. The wave forms are generated using a slice method which is user defined by structured input, thus providing the artist with full control over crest shape and placement. Wave propagation is obtained by interpolating between defined crest shapes and positions. We also present a method for computing a stylized representation of breaking crests in shallow water.

Artists may use our model to create many interesting wave forms, including basic sinusoidal waves and waves with breaking crests that have a rotation that is cyclical in time. The major drawbacks to our solution are that data entry can be tedious and it can be difficult to produce waves that animate with a natural appearance.

To Emily

ACKNOWLEDGMENTS

I would like to thank my committee chair Dr. Ergun Akleman for his guidance and patience, Prof. Karen Hillier for her critiques and for forcing me to use the right half of my brain, and Dr. John Keyser for his input and for being on my committee. I would also like to thank Cindy Hong, Margaret Lomas, the Stenner family, and my parents for their support.

Finally, I would like to thank God the Father, God the Son, and God the Holy Spirit without whose grace this work would not have been completed. Thanks also to Saint Mary, Saint Rita, Saint Bernadette, Saint Cecilia, Saint Romanus, and my Guardian Angel for their prayers and intercessions.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION . . . 1
A. Motivation . . . 1
B. Ocean Waves in Nature . . . 1
C. Ocean Waves in Art . . . 2

II PREVIOUS WORK . . . 5
A. The Quest for Photorealism . . . 5
B. Physically Based Methods . . . 5
C. Procedural Methods . . . 6

III METHODOLOGY . . . 10
A. Wave Models and Characteristics . . . 10
1. Gerstner Wave Model . . . 10
2. Beauty Wave Model . . . 12
B. X-Waves and Beauty Waves . . . 12
C. Crest Design and Control . . . 14
1. Breaking Waves . . . 15
2. Crest Windup . . . 15
3. Crest Unwind . . . 17
a. $\theta_c > 180°$ . . . 18
b. $0° \leq \theta_c \leq 180°$ . . . 21
D. Breaking Beauty Waves . . . 23

IV IMPLEMENTATION AND RESULTS . . . 27
A. Initialization and Structured Input . . . 27
1. Beauty Wave Control . . . 27
2. X-Wave Definition and Modulation . . . 28
B. Rendering . . . 29
C. Results . . . 29

CHAPTER Page

V CONCLUSION AND FUTURE WORK 33

REFERENCES 35

APPENDIX A 37

APPENDIX B 39

VITA 43

LIST OF FIGURES

FIGURE		Page
1	Breaking ocean waves in nature.	2
2	"The Great Wave at Kanagawa," Japanese woodblock print by Katsushika Hokusai [7].	3
3	Ocean waves in Japanese woodblock prints [6].	3
4	Ocean waves in Chinese ink painting [9].	4
5	Breaking waves in modern illustration.	4
6	Wave forms computed by Fournier and Reeves using the Gerstner wave model [4].	8
7	Transition to breaking crests computed using the method by Fournier and Reeves [4].	8
8	Wave characteristics.	11
9	Using Bezier curves to generate (A) one wavelength of a basic surface wave, and (B) modulated wave forms.	13
10	Bezier curve regions defining the wave shape between two crests.	15
11	(A) 90° and (B) 180° circular arcs computed using Bezier curves.	16
12	Effect of spiral factor, F_S, on crest windup region ($\theta_c = 360°$).	17
13	Effect of radius factor, F_R, on crest windup region ($\theta_c = 270°$).	18
14	Effect of thickness factor, F_T, on crest unwind region ($\theta_c = 360°$).	19
15	Effect of cusp tangent angle, θ_t, on crest unwind region when $\theta_c > 180°$.	20
16	Effect of crest core radius, F_{CR}, on crest unwind region when $\theta_c > 180°$ ($\theta_t = 180°$).	20

FIGURE Page

17 Transition of waveform with increasing cusp tangent angle, θ_t, when $\theta_c = 0°$. 22

18 Effect of crest core radius, F_{CR}, on crest unwind region when $\theta_c = 0°$ ($\theta_t = 180°$). 24

19 Breaking beauty wave at the crest rotation angle, θ_R, of 180°. 25

20 Formation of a breaking beauty wave. 26

21 Basic wave forms computed using beauty wave method using only one specified pose with (A) and without (B) added crest separation. . 30

22 Beauty waves using pose data shown in Appendix B. 30

23 More beauty waves. 31

24 Breaking beauty waves at two time samples. 31

CHAPTER I

INTRODUCTION

A. Motivation

The goal of this work is to provide a method for generating and animating stylized, two-dimensional wave forms with characteristics similar to waves found in nature and in art, particularly Asian land and seascapes. Artists will have direct control over a wide variety of crest shapes owing to our solution which does not rely on physically based equations of mass and momentum as do most ocean wave simulations.

Although not implemented, resulting wave forms could be spanned for construction and rendering of a three-dimensional ocean surface or used as a baseline from which to render ink strokes such as those found in Chinese ink painting or Japanese Sumi-e.

B. Ocean Waves in Nature

Ocean waves in nature exhibit a limitless variety of shapes due to the freedom of motion of fluids. One of the most interesting of these shapes is the breaking wave as shown in Figure 1.

As waves move from deep to shallow water, the wave begins to drag along the seafloor, thus slowing the velocity at the bottom of the wave as compared to the top. Eventually, gravity forces the faster moving wave crest to break and fall in front of the wave giving the profile view a rotational appearance. Larger wavelengths and velocities produce more dramatic results such as the tube wave (see Figure 1).

The journal model is *IEEE Transactions on Visualization and Computer Graphics*.

Fig. 1. Breaking ocean waves in nature.

Although our method can produce a common sinusoidal waveform, it is the shape and rotational motion of the breaking wave that are our primary focus.

C. Ocean Waves in Art

The representation of ocean waves in art was our greatest motivation for this work, particularly waves in Asian art. Probably the most famous example is "The Great Wave at Kanagawa," a woodblock print by the Japanese artist Katsushika Hokusai (see Figure 2). Several more examples of waves in Japanese woodblock prints are shown in Figure 3, while Figures 4 and 5 show two examples in Chinese ink painting and modern illustration, respectfully.

Note how the waves can be represented as a series of one or more lines with variation in form to give the appearance of a flowing surface. We take this same approach because it allows for a better view of the details generated by our wave model.

Fig. 2. "The Great Wave at Kanagawa," Japanese woodblock print by Katsushika Hokusai [7].

Fig. 3. Ocean waves in Japanese woodblock prints [6].

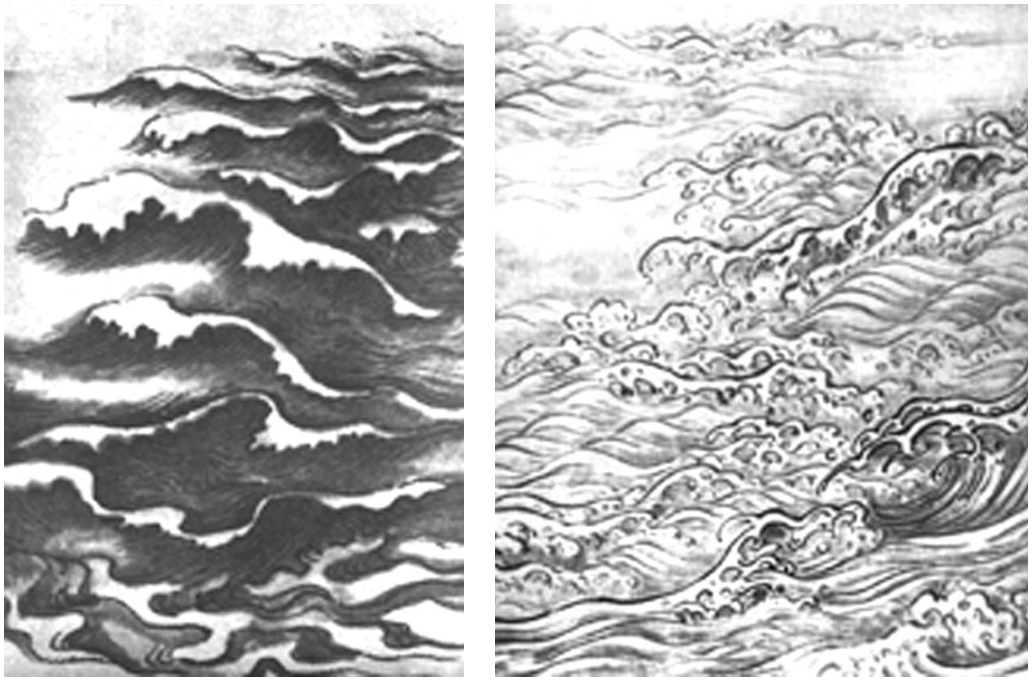


Fig. 4. Ocean waves in Chinese ink painting [9].

Fig. 5. Breaking waves in modern illustration.

CHAPTER II

PREVIOUS WORK

A. The Quest for Photorealism

The major focus in fluid visualization for computer graphics has been in achieving photorealistic behavior and appearance, including advances in computational efficiency and stability. When computing power was limited, solutions were computed using linear, procedural equations which were not photorealistic, but were stable, additive, and computationally inexpensive. Two procedural methods were used in particular. One produced periodic surface point displacements over time. A second used linear approximations to the non-linear, 3-D Navier-Stokes (N-S) equations for incompressible fluid flow. As computer technology advanced, physically-based approaches to the aforementioned non-linear equations became the norm. Because our work is derived from the older, procedural methods for fluid visualization, we will discuss them after the more recent work using physically based methods.

B. Physically Based Methods

To simulate the physical behavior of any general fluid, including ocean waves, one must solve the non-linear, inviscid, and incompressible Navier-Stokes equations. The equations of conservation of mass and momentum are

$$\mathbf{u_0} + (\mathbf{u} \cdot \nabla \mathbf{u}) + (\nabla p/\rho) - \mathbf{f} = 0 \tag{2.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.2}$$

where $\mathbf{u}$ is the velocity vector (u,v,w), ∇ is the gradient operator, p is the pressure, ρ is the density, and $\mathbf{f}$ is any external force such as gravity.

Early work using the non-linear, 3-D N-S equations involved iterative solutions using a regular grid of fluid cells with marker particles to track fluid flow [3]. Foster and Fedkiw [2] later introduced a hybrid approach for computing complex fluid interaction using massless marker particles to track an implicit surface. Enright et al. then improved on this method and included a 3-D simulation of a breaking wave in shallow water [1]. Although lacking in small scale features and computationally expensive, their solution produced a high level of realism.

Other work on breaking ocean waves was done by Mihalef et al. [11]. Their approach involved computing a full 3-D N-S solution on multiple 2-D slices of a breaking wave and then combining the slices to form a 3-D wave form. The initial conditions for their 2-D crest shapes could be defined by an animator from a pre-computed library, thereby eliminating the computations for wave propagation to the point of breaking and achieving greater control over wave shape than Enright et al. Work on interactive frame rates also came into focus when Mller et al. used a particle based approach with an interpolation method called smoothed particle hydrodynamics to simulate fluids [12]. Using particles allowed them to neglect mass conservation and the convection terms of the N-S equations which greatly simplified their computations and increased their frame rate. More recently, Irving et al. [8] used fluid cells to simulate large bodies of water by coupling a 3-D N-S solution for the air-water boundary while using a simpler 2-D height field approach for the remaining volume.

C. Procedural Methods

Using the full Navier-Stokes equations for fluid and ocean wave simulation, coupled with advanced rendering solutions, provides a very realistic appearance. These methods are also generally expensive in regards to computational time and effort, but

the trade-off is acceptable for their intended use in the gaming and motion picture industries. Our solution does not require such complexity and realism and is actually derived from earlier research on ocean wave simulation. Initially, ocean waves were computed using procedural methods which have the advantage of being linear. Two of the most common methods involved height fields or the Gerstner wave model.

Height fields are restricted to one vertical z position per (x,y) surface point, so they can only be used to compute waves without breaking crests. Peachey [13] used such a method to compute waves with refraction combined with a particle system for generation of foam and spray. Kass and Miller [10] also used a height field, but with a linear approximation of the N-S equations. The work of Fournier and Reeves [4] paralleled that of Peachey, but without height fields. They used the Gerstner wave model which describes a circular motion for the displacement of a particle around its rest position. A more detailed description of this model will be given later, but by controlling various parameters, the model allowed Fournier and Reeves to control the "sharpness" of crest peaks as shown in Figure 6. The procedural model fails at the point when the crest peaks begin to form loops. Fournier and Reeves were also able to generate the appearance of breaking crests by controlling the rate of circular rotation of surface points, as shown in Figure 7, and by flattening the circular orbits into ellipses in shallow water. Although somewhat convincing in appearance, the range of available crest shapes was extremely limited.

Ts'o and Barsky [15] later used the Gerstner wave model combined with wave-tracing to compute refracting waves, but approximated the appearance of crests using the tension property of beta-splines. Hinsinger et al. [5] computed deep water surface waves at interactive frame rates. They computed surface points on an adaptive mesh defined by a projection from the camera position which was then transformed into screen space. This allowed them to filter waves that would not be visible and to focus

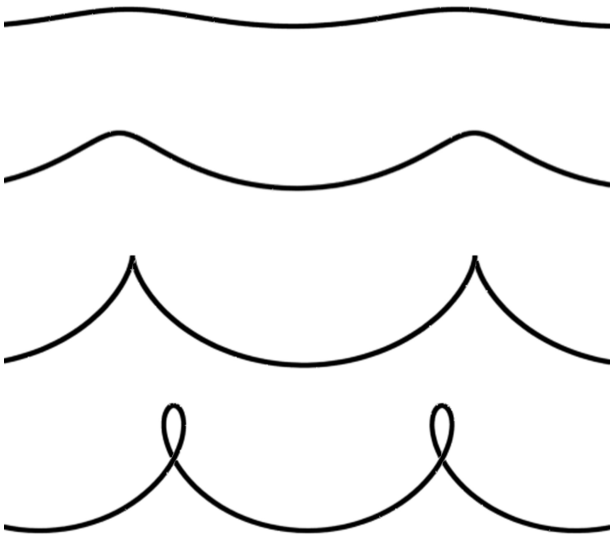


Fig. 6. Wave forms computed by Fournier and Reeves using the Gerstner wave model [4].

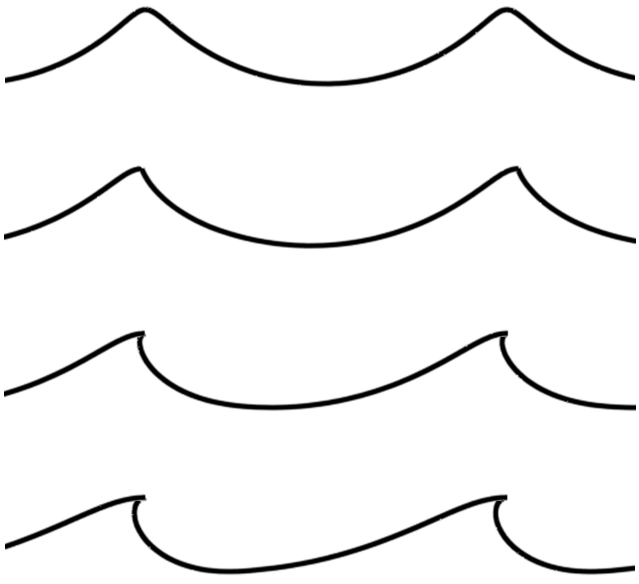


Fig. 7. Transition to breaking crests computed using the method by Fournier and Reeves [4].

sampling on any area of the computed surface for enhanced image quality in that region.

Procedural methods do not generate physically correct wave forms, but they do achieve the appearance of waves with simple and efficient computations. Our method is derived from the Gerstner wave model, but we dispense with the procedural computations. Our wave forms are generated based solely on appearance, given the fact that a particle tracing a circle over time produces the appearance of a gravity wave.

CHAPTER III

METHODOLOGY

In this work we develop a method for generating an artistic representation of ocean waves using cubic Bezier curves. Bezier curves are defined by four control points. The endpoints always lie on the curve, and the intermediate points control the rate of interpolation along the curve and define the tangents at the endpoints. Bezier curves were chosen because of their ease of control and because their convex hull properties can limit undesirable intersections [14]. We use cubic Bezier curves because we only need to approximate the general shape of wave forms. We are concerned with appearance only, and not with computing precise curves.

A. Wave Models and Characteristics

The basic features of a surface wave are crests and troughs. Wave crests represent the portion of the wave above the surface at rest, and troughs, the portion below. The crest and trough peaks are located at the highest and lowest vertical positions of the wave, respectfully. Figure 8 shows some of the basic characteristics of a wave. The wavelength, λ, is the distance between crests/troughs, the amplitude, a, is the height of the wave relative to the surface at rest, and the crest speed, c, is the distance of crest travel per unit time.

1. Gerstner Wave Model

As previously stated, our wave model is derived from the Gerstner model used by Fournier and Reeves [4]. The Gerstner model assumes uniform (no mass transport), incompressible (constant density), and irrotational flow which allows linear superposition of any number of wave trains at any point on a defined surface. The 2-D

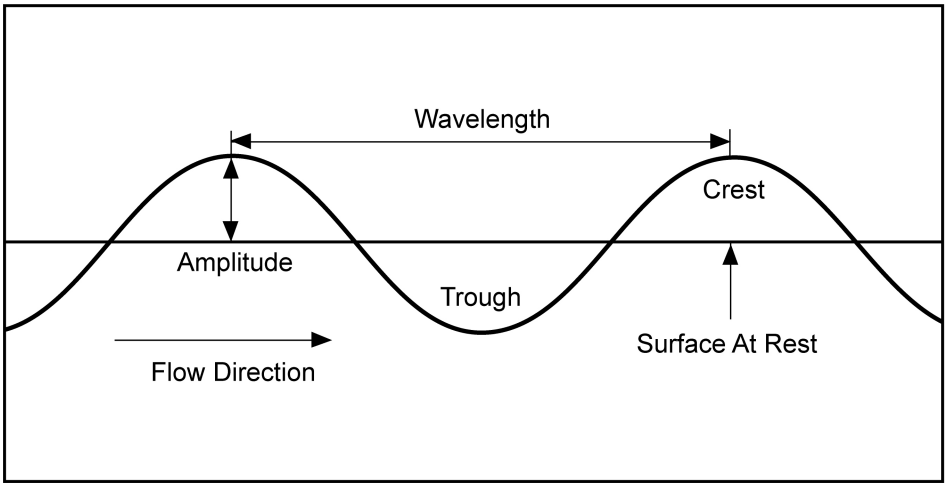


Fig. 8. Wave characteristics.

equations of motion for such a point are

$$x = x_0 + a \times \sin(\kappa x_0 - \omega t) \tag{3.1}$$

$$y = y_0 + a \times \cos(\kappa x_0 - \omega t) \tag{3.2}$$

where (x_0, y_0) is the position at rest, and t is the current time. The variable κ represents the number of cycles per unit time and is related to the wavelength by $\kappa = 2\pi/\lambda$. The variable ω is the angular frequency and is related to the period, T, by $\omega = 2\pi/T$, and to κ by $\kappa = \omega c$. The term in parenthesis represents the wave phase, ϕ ($\phi = \kappa x_0 - \omega t$).

Equations 3.1 and 3.2 describe a circular rotation which displaces a point at rest based on given values of initial position, amplitude, and phase. The product $\varsigma = \kappa a$ provides a measure of the sharpness of crest peaks (see Figure 6), where $\varsigma = 0$ gives the surface at rest and $\varsigma = 1$ gives a sharp vertical peak. Only at low values of ς does the circular motion give a wave form comparable to that of a sinusoid.

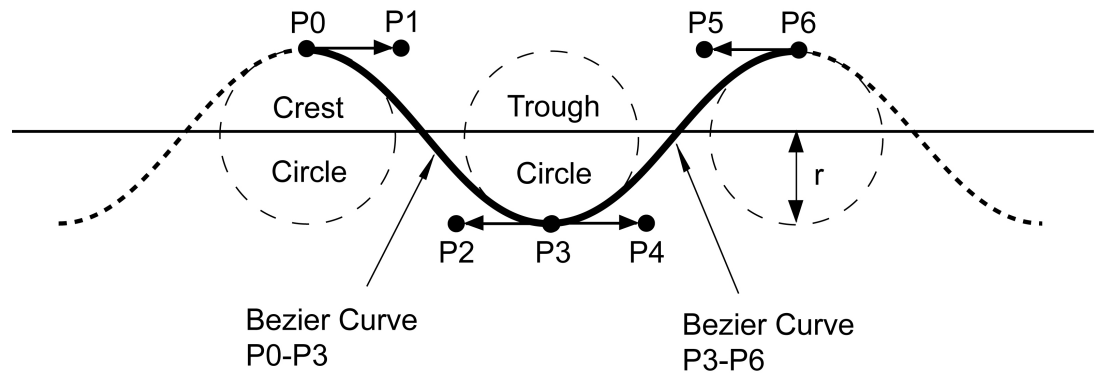
Wave forms computed using procedural models such as the Gerstner model only produce a limited range of shapes. Those computed from physically based equations are limited by physics and computational complexity. We wanted to create a model which would offer an artist more control and variety over crest shape with less computational effort. Bezier curves give us direct control, while circles and rotation allow us to create a wide variety of crest shapes and wave forms. We do make a trade-off for the added control, and that is in the increased effort required for wave form definition.

2. Beauty Wave Model

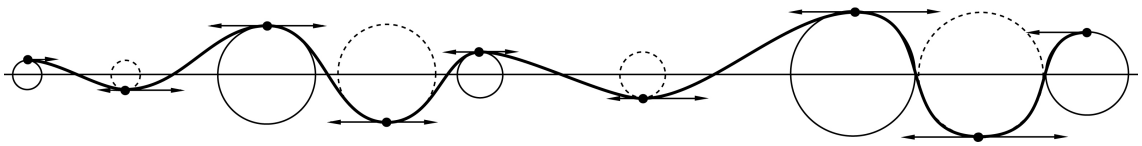
Figure 9·A shows how two Bezier curves can be used to generate one wavelength of a basic surface wave. The endpoints of both curves are located at crest and trough peaks which are defined by circles of a given radius, r. We call these circles wave circles or, more specifically, crest or trough circles. The radius represents the wave amplitude, and crest/trough circle separation represents the wavelength. Bezier tangent points are located at a given distance from the endpoints in a direction parallel to the surface at rest. A continuous, uniform wave can be generated by keeping equally spaced wave circles at a constant radius while using Bezier curves with constant tangent length and direction. Modulating these values can provide a wide variety of wave forms as shown in Figure 9·B, although larger variations begin to lose the appearance of ocean waves.

B. X-Waves and Beauty Waves

Our model provides flexibility in crest shape generation by allowing crest circles (not trough circles) to be controlled independently of each other. That is, each crest circle is considered to be an object with it's own set of descriptors called a pose. A 2-D



(A)



(B)

Fig. 9. Using Bezier curves to generate (A) one wavelength of a basic surface wave, and (B) modulated wave forms.

waveform called an x-wave is created by defining poses for one or more crests along the x-axis, and beauty waves consist of one or more x-waves positioned along the z-axis. Because rotation is a trivial matter, we restrict our computations to be aligned with the world space axes.

Trough circles are given a radius equal to the crest circle they follow and are centered between the inside edge of the surrounding two crest circles along the axis between crest circle centers. To prevent wavelengths from collapsing, adjacent crest circles must be separated by the diameter of the trough circle between them. Crest and trough circles may therefore touch, but not intersect. The right most wavelength in Figure 9·B shows this constraint.

C. Crest Design and Control

We define one wavelength of any given x-wave to be composed of three regions or sets of Bezier curves between two crest circles. The first and second regions of curves are discussed in the following sections and are called windup and unwind, respectfully. There is one exception when the windup region does not exist, and that is also discussed later. The third region is actually only one Bezier curve called the incline curve, and it runs from the bottom of a trough circle to the top of the following crest circle (see curve P3-P6 in Figure 9·A). However, depending on the pose of the leading crest of any particular wavelength, the beginning point of the incline curve may shift between the bottom of the trough circle to the bottom of the leading crest circle. Tangent lengths for the incline curve are computed relative to both the distance between endpoints and the ratio of the trough and following crest circle radii. Figure 10 shows the regions specified for each set of curves.

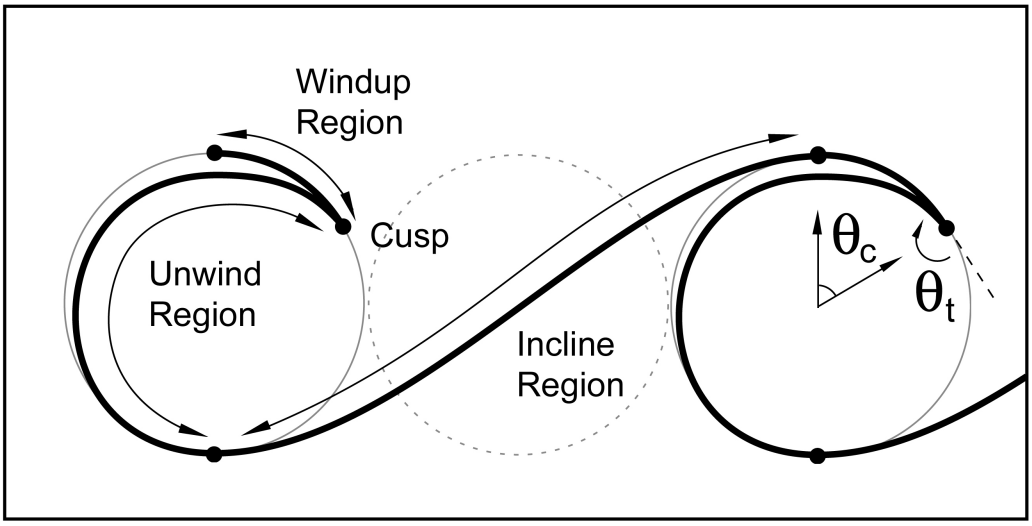


Fig. 10. Bezier curve regions defining the wave shape between two crests.

1. Breaking Waves

The wave form shown in Figure 10 has the shape of a breaking wave. To achieve this shape, the Bezier curve tangents defining the standard waveform in Figure 9·A must break such that the tangents forming the crest fail to lie on the same axis. The breaking point defines the separation between the windup and unwind regions of the crest and is called the crest cusp. We allow the outgoing cusp tangent, θ_t, to vary in direction by up to 180° from the axis of the incoming cusp tangent (see Figure 10). Coupling this feature with a circular curl of the crest allows us to achieve the basic shape of a breaking wave and more.

2. Crest Windup

The windup region of a crest defines the outer line of a wave from the top of a crest circle to the cusp and is specified by the crest curl angle, θ_c (see Figure 10). When $\theta_c = 0$ the windup region does not exist, leaving the unwind and incline regions

to form the wave between crest circles. As θ_c is increased, the end points of the Bezier curves defining the windup region are computed to be on 90° circular arcs, or quads. With one exception to be discussed in the next section, windup curves and quad curves are the same. Quad curves are computed using cubic Bezier curves with tangents having a length of 0.55 times the quad radius and being aligned with the axes of the circle as shown in Figure 11·A. The tangent multiplier 0.55 was chosen because it generates a cubic Bezier curve that approximates a circular arc. It is not necessary for the quad curves to be actual circles, as we are only using them to trace a path for the windup region of our wave forms.

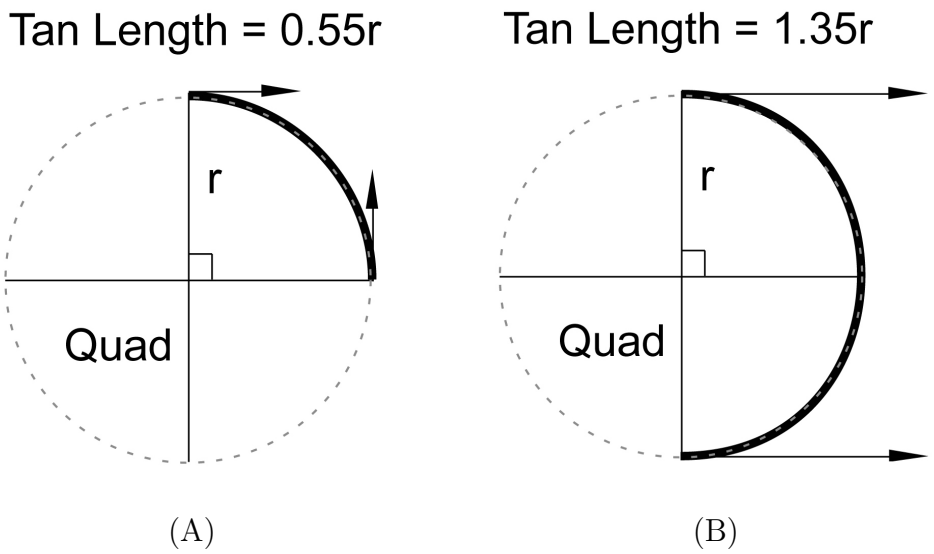
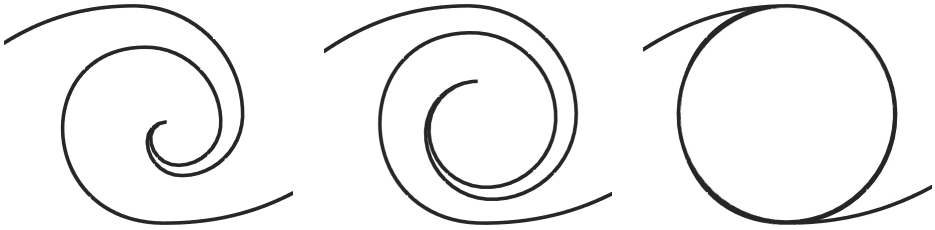


Fig. 11. (A) 90° and (B) 180° circular arcs computed using Bezier curves.

Because it will be of interest later, Figure 11·B shows how 180° circular arcs can be approximated in a similar manner using a radius multiplier of 1.35 for the Bezier tangent lengths.

The 90° quad radii diminish from the crest circle radius by a rate defined by a fractional parameter called the spiral factor, F_S. If $F_S = 1$, the Bezier curve endpoints

will always lie on the approximated crest circle. Otherwise, they will lie on the spiral of quad curves of diminishing radius. The minimum value of the spiral factor is clamped internally and will increase as necessary to accommodate the specified crest curl angle. Figure 12 shows the effect of F_S on the windup region.



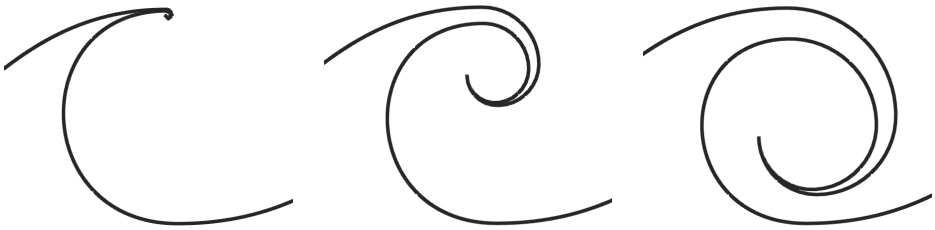
(A) $F_S = 0.0$ (B) $F_S = 0.5$ (C) $F_S = 1.0$

Fig. 12. Effect of spiral factor, F_S, on crest windup region ($\theta_c = 360°$).

Another fractional parameter affecting the windup region is the radius factor, F_R. F_R controls the radius of the first quad representing the windup region, where $F_R = 1$ gives a radius equal to the crest circle radius. As F_R decreases, it effectively moves the crest circle center toward the top which decreases the spiral radius of the crest. Again, we have clamped the minimum value internally to avoid computing curves that would be so small as to be unseen. Figure 13 shows the effect of F_R on the windup region.

3. Crest Unwind

The unwind region for any wavelength of our wave model is the region which allows for the most variation in crest shape. It also requires more computational effort because of the need to interpolate between various solutions. We will first discuss the unwind region when $\theta_c > 180°$, and then the region when $0° \leq \theta_c \leq 180°$.



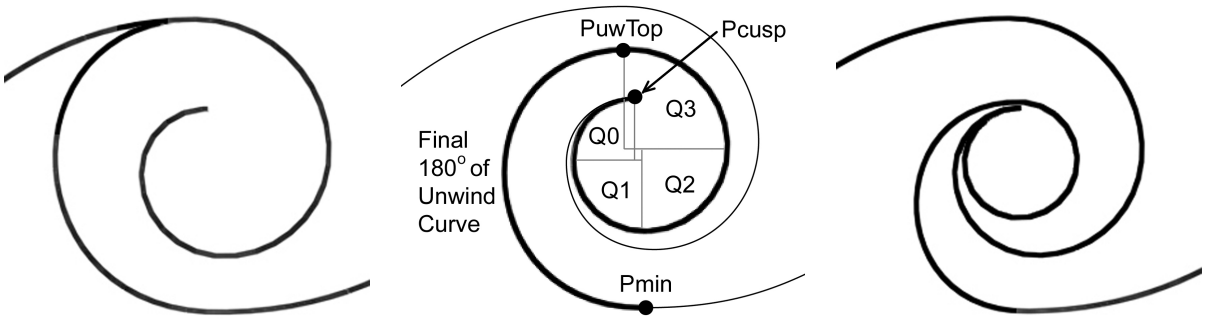
(A) $F_R = 0.0$ (B) $F_R = 0.5$ (C) $F_R = 1.0$

Fig. 13. Effect of radius factor, F_R, on crest windup region ($\theta_c = 270°$).

a. $\theta_c > 180°$

When $\theta_c > 180°$ and the cusp tangent angle, θ_t, is at its maximum angle of 180° as in Figure 13, the unwind curve forms a loop from the cusp to the beginning of the incline curve. In this case, baseline Bezier curves are computed in a similar manner as those for the windup region. The crest is unwound from the cusp to a position near or at the top of the crest circle by computing 90° quads of increasing radius. The top position of the unwind region, PuwTop, determines the rate of increase for quad radii and is defined by a fractional parameter called the thickness factor, F_T. When $F_T = 0$, the unwind curve is equal to the windup curve along that region. The maximum thickness at $F_T = 1$ is determined by the cusp position, Pcusp, when $\theta_c = 360°$. This allows the crest to curl without self-intersecting. Figure 14 shows the effect of increasing F_T on the unwind region. Figure 14·B also shows the 90° quads of increasing radius used to compute the baseline Bezier curves. It also shows the two parts of the unwind curve from Pcusp to PuwTop, and from PuwTop to the beginning of the incline curve, Pmin. Again, the Bezier curves we compute are only computed to give the appearance of 90° circular arcs.

The final portion of the unwind curve (PuwTop to Pmin) is computed using an

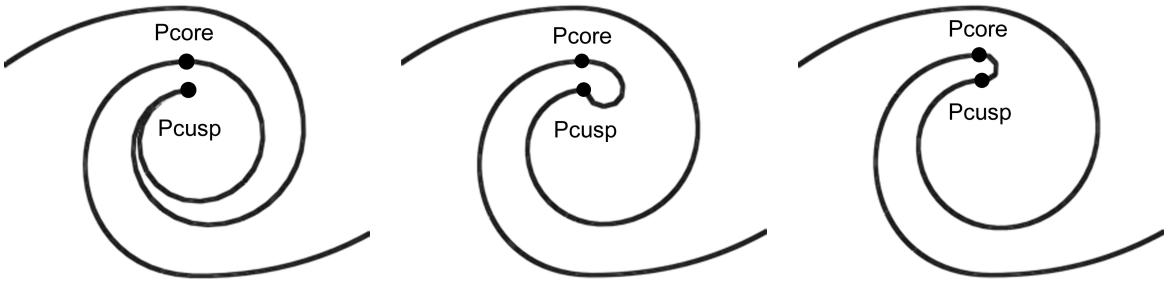


(A) $F_T = 0.0$ (B) $F_T = 0.5$ (C) $F_T = 1.0$

Fig. 14. Effect of thickness factor, F_T, on crest unwind region ($\theta_c = 360°$).

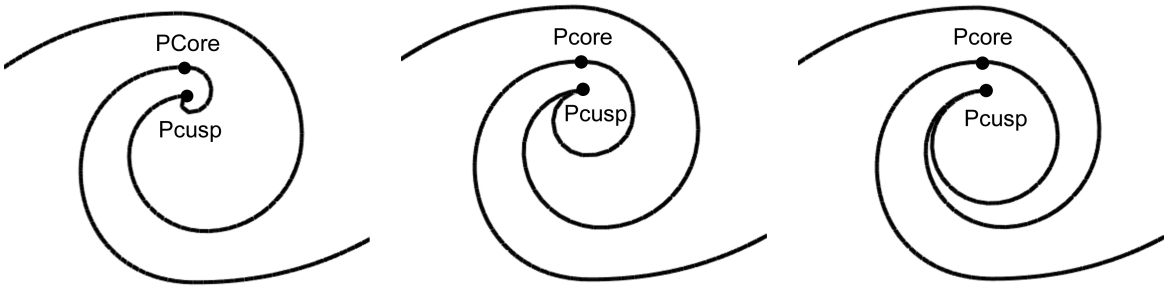
approximated, 180° Bezier curve similar to the one shown in Figure 11·B. The tangent lengths are modified depending on the skew of the endpoints from the vertical axis of the crest circle. Finally, as the cusp tangent angle θ_t decreases to 0, the 90° quad curves are diminished one in turn from the cusp. This collapses the crest core which we define as the portion of the unwind curve from the cusp to the point, Pcore, 360° from the cusp when $\theta_t = 360°$ (in Figure 14·B, the crest core would run from Pcusp to PuwTop/Pcore). When $\theta_t = 0°$, only a 180° Bezier curve remains to bridge the gap between Pcusp and Pcore. Figure 15 shows the crest core and the effect on the unwind region by decreasing θ_t when $\theta_c > 180°$.

Finally, the radius of the crest core can be modified by a fractional parameter called the core radius factor, F_{CR}. F_{CR} will have the greatest effect when the full crest core exists at $\theta_t = 360°$ and no effect when $\theta_t = 0°$. Figure 16 shows the effect on the crest core by increasing F_{CR} when $\theta_c > 180°$.



(A) $\theta_t = 180°$ (B) $\theta_t = 90°$ (C) $\theta_t = 0°$

Fig. 15. Effect of cusp tangent angle, θ_t, on crest unwind region when $\theta_c > 180°$.



(A) $F_{CR} = 0.0$ (B) $F_{CR} = 0.5$ (C) $F_{CR} = 1.0$

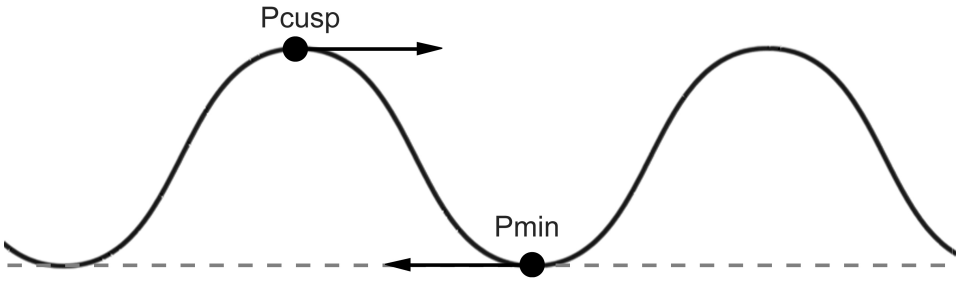
Fig. 16. Effect of crest core radius, F_{CR}, on crest unwind region when $\theta_c > 180°$ ($\theta_t = 180°$).

b. $0° \leq \theta_c \leq 180°$

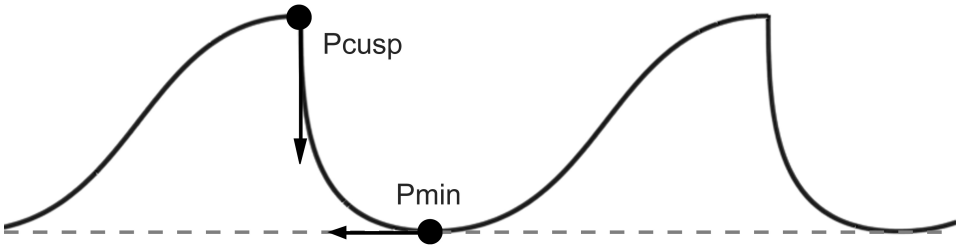
When $0° \leq \theta_c \leq 180°$, the transition from a standard, sinusoidal wave form into a breaking crest must be taken into account. For illustrative purposes, we will look at the case when $\theta_c = 0°$. In this case only the unwind and incline regions exist between crest circles, and the cusp is located at the top of the crest circle. If θ_t also equals zero, a sinusoidal wave form is generated. As the beginning tangent of the unwind curve (the cusp tangent) rotates with θ_t, the trough point, Pmin, moves from the bottom of the trough circle to the bottom of the leading crest circle. We use this line of movement to represent the lower limit of our wave forms in order to maintain a uniform appearance. We do this by keeping the Bezier curve tangents at Pmin aligned with the line of movement and by ensuring that the Bezier point defining the cusp tangent of the unwind curve never lies below the line of Pmin.

Figure 17 shows the movement of Pmin and the transition of our waveform with increasing cusp tangent angle, θ_t, for the case when $\theta_c = 0°$. The Bezier curve for the incline region retains its basic shape with the movement of Pmin, but the tangents are lengthened to account for the larger span between endpoints. The tangent direction at the cusp of the unwind curve is computed based on θ_t as usual. The tangent lengths for the unwind Bezier curve are computed by interpolation between those lengths when the tangents are both parallel with the line of travel of Pmin as in Figures 17·A and 17·C, and the lengths when they are perpendicular as in Figure 17·B. The perpendicular lengths are based on the intersection point between tangents. The case when $\theta_c = 0°$ was chosen for simplicity, but the same method applies as θ_c increases to 180°.

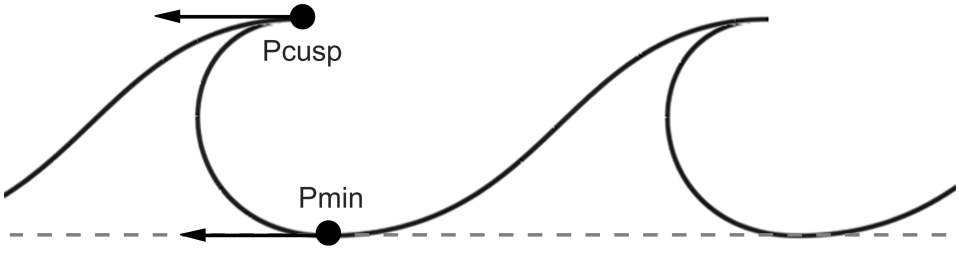
The same parameters controlling the unwind curve when $\theta_c > 180°$ still apply for the case when $0° \leq \theta_c \leq 180°$, however their effect may be negligible depending



(A) $\theta_t = 0°$



(B) $\theta_t = 90°$



(C) $\theta_t = 180°$

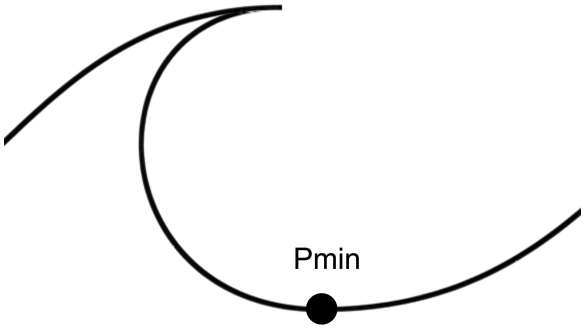
Fig. 17. Transition of waveform with increasing cusp tangent angle, θ_t, when $\theta_c = 0°$.

on the current values of θ_c and θ_t. For example, the thickness factor, F_T, and spiral factor, F_S, have no meaning when $\theta_c = 0°$ because no crest spiral or windup curve exists. Because the crest core is somewhat ambiguous when $\theta_c < 180°$, the effect of crest core radius, F_{CR}, on the crest unwind region is shown in Figure 18 when $\theta_c = 0°$. Note that Pmin shifts for this case.

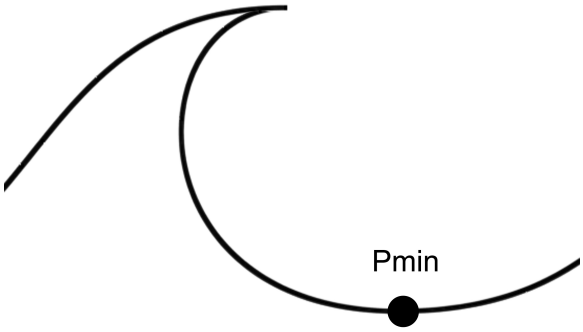
D. Breaking Beauty Waves

We wanted to extend our method of wave form computation to capture the appearance of the cyclical rotation of breaking crests similar to that of tube waves (see Figure 1). We do this by including another crest pose parameter called the crest rotation angle, θ_R. This parameter is currently just a switch where any value greater than zero will turn on the breaking crest feature for a particular pose. We wanted to keep the naming of the parameter generic for future updates. The crest rotation angle, currently forced to 180° for any given value, defines the crest curl angle (θ_c) when the crest cusp is desired to come into contact with the unwind curve. This will close the crest, and the unwind curve will form a loop or an air pocket similar to the profile of a crashing tube wave. Because gravity acts in the negative z direction, the logical place for our initial point of contact is at the lowest point of our wave form, Pmin. Setting θ_R to 180° sets the point of contact to Pmin and, with a lower value of F_S, works well for achieving the appearance of natural waves as shown in Figure 19.

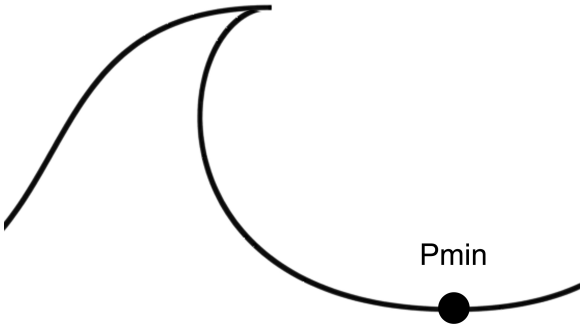
We get the cusp to contact the lower portion of the wave form by stretching the windup and unwind curves as θ_c goes from 0° to θ_R (180°). At this point we remove the series of unwind Bezier curves forming the air bubble from the curves forming the X-Wave proper. We then continue curling the crest until the cusp reaches the top position of the crest circle at $\theta_c = 360°$. We shrink the Bezier curves forming the



(A) $F_{CR} = 1.0$



(B) $F_{CR} = 0.5$



(C) $F_{CR} = 0.0$

Fig. 18. Effect of crest core radius, F_{CR}, on crest unwind region when $\theta_c = 0°$ ($\theta_t = 180°$).

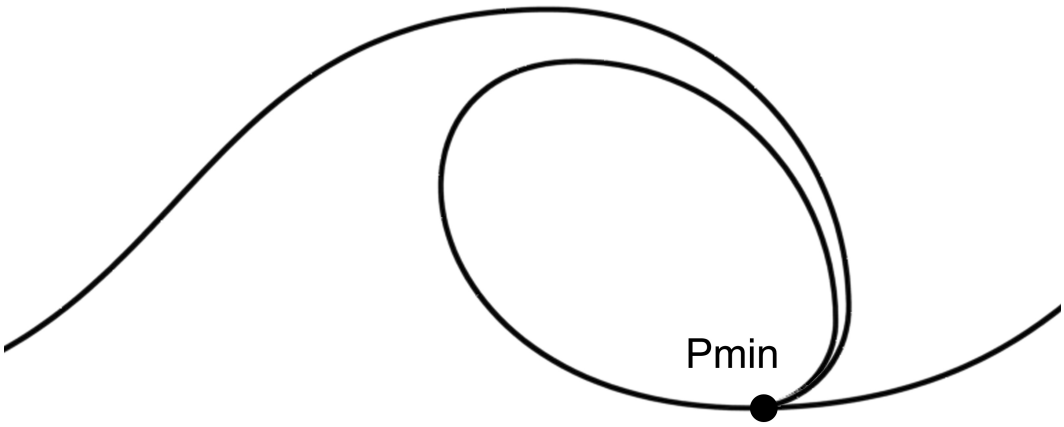
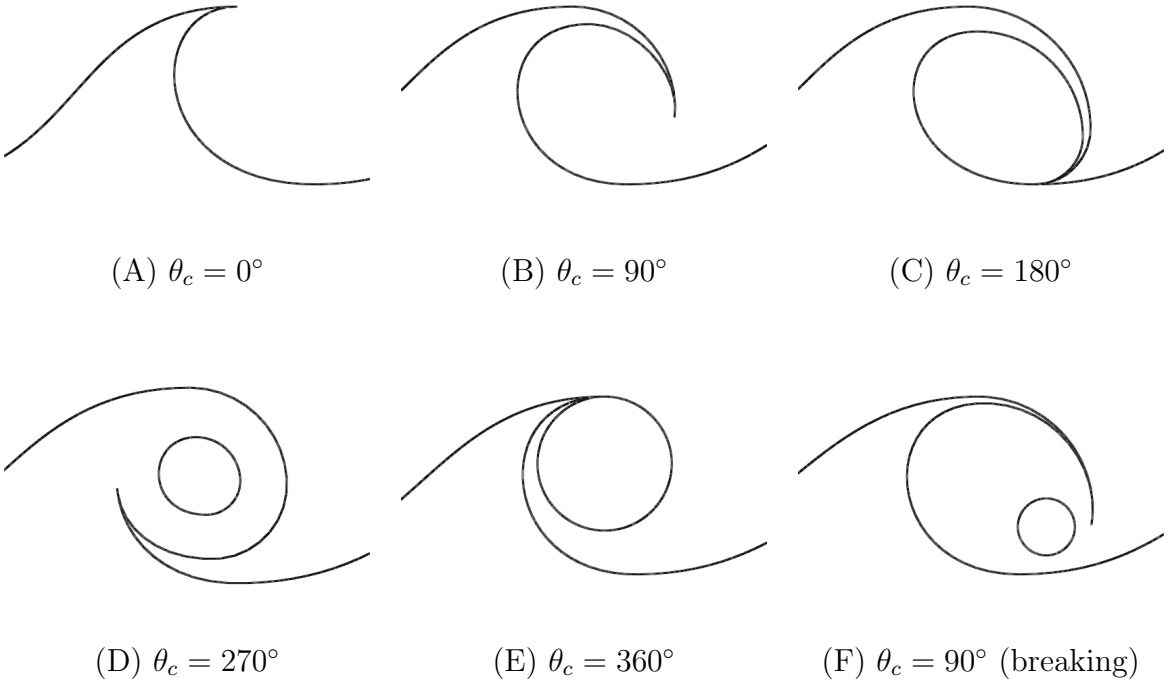


Fig. 19. Breaking beauty wave at the crest rotation angle, θ_R, of 180°.

air bubble and rotate them toward the top of the crest until they diminish to a point at a curl angle near or at 360°. The collapsing air bubble makes it appear as if it were being filled with water from the wave, thereby causing the formation of a water "drop" near the cusp of the crest. When the curl angle reaches 360°, we separate the water drop from the main Bezier curves of the X-Wave as we did for the bubble. In a similar fashion, the water drop diminishes to a point and rotates toward the bottom of the wave form as the curl angle approaches 180°. By having a water drop form as our air bubble collapses and vice versa, we give the illusion of conservation of mass and the cyclical rotation of a breaking wave. Figure 20 shows the formation of a breaking beauty wave. During the transition into and out of breaking waves, we vary the opacity of the bubble/drop Bezier curves from or to 0 depending on the starting value of θ_c. We require that there be at least two adjacent crests to be switched on with θ_R for breaking to be operational. We also simplify the transition into and out of breaking waves by requiring all breaking crests in any particular series and the non-breaking crests surrounding them to all use a crest curl angle of either $\theta_c = 0°$ or

$\theta_c = 180°$. These values are chosen because they are the angles where the main series of X-Wave Bezier curves for breaking crests match those for non-breaking crests.



(A) $\theta_c = 0°$ (B) $\theta_c = 90°$ (C) $\theta_c = 180°$

(D) $\theta_c = 270°$ (E) $\theta_c = 360°$ (F) $\theta_c = 90°$ (breaking)

Fig. 20. Formation of a breaking beauty wave.

CHAPTER IV

IMPLEMENTATION AND RESULTS

In this chapter, we discuss beauty wave initialization and program input structure. We will also show simple renderings of various beauty waves computed using our method of wave form generation.

A. Initialization and Structured Input

1. Beauty Wave Control

Our software is written in C++, with one class being devoted to the task of reading and verifying user input from pre-defined data files. The same class also handles the function calls necessary for computation, display, and data output, if any, of beauty waves. At least two input files are required for beauty wave control. The first is the control input file which contains the values necessary for program initialization. The parameters and formatting rules for this file are given in Appendix A. Most of the parameters are self explanatory; others will be discussed below.

The second input file type gives a description of the beauty wave state at a particular time as a formatted list of X-Waves and poses. If there are multiple state files and the time step is greater than zero, the input data will be interpolated based on a transition time between files given the current time. Interpolation between files allows for the animation of beauty waves. The parameters and formatting rules for beauty wave state input files are given in Appendix B.

2. X-Wave Definition and Modulation

Beauty wave state, or pose, input files are lists of pose data separated by X-Wave delimiters. The first pose of each X-Wave must specify all nine possible pose parameter values and only one pose is required for program operation. If the number of crests per X-Wave, exceeds the number given in an input file for any particular X-Wave, the same pose data will propagate along the wave for each remaining crest. The center positions of copied crests are computed as a shift of four times the radius of the previous pose (the minimum allowed separation between crest circles of constant radius) plus the end separation parameter provided in the control input file. Poses beyond the first for any X-Wave do not need to have all 9 parameters specified. Only those values an animator desires to change from the previous pose need to be included, although it is usually best to at least provide crest position data. If the radius of any crest approaches zero, we also fade the curl and cusp tangent angles to zero to flatten out the wavelength.

If the number of X-Waves to draw exceeds the number defined in beauty wave state files, the X-Waves will repeat along the z axis until the specified number of wave forms have been drawn. The lateral separation distance between wrapped X-Waves is specified in the control input file. If an animator wishes to allow wrapping of X-Waves, care should be taken when specifying the pose data for the beginning and ending X-Waves in order to avoid sudden changes in data at wrap positions. It also looks better to keep X-Waves evenly spaced along the z axis and to specify beginning and ending X-Wave poses with similar radii and x positions.

X-Waves are computed as a series of Bezier curves (including a separate series of four Bezier curves for each breaking crest), where curves are computed per wavelength based on interpolation of crest pose data between integer time values. So, if

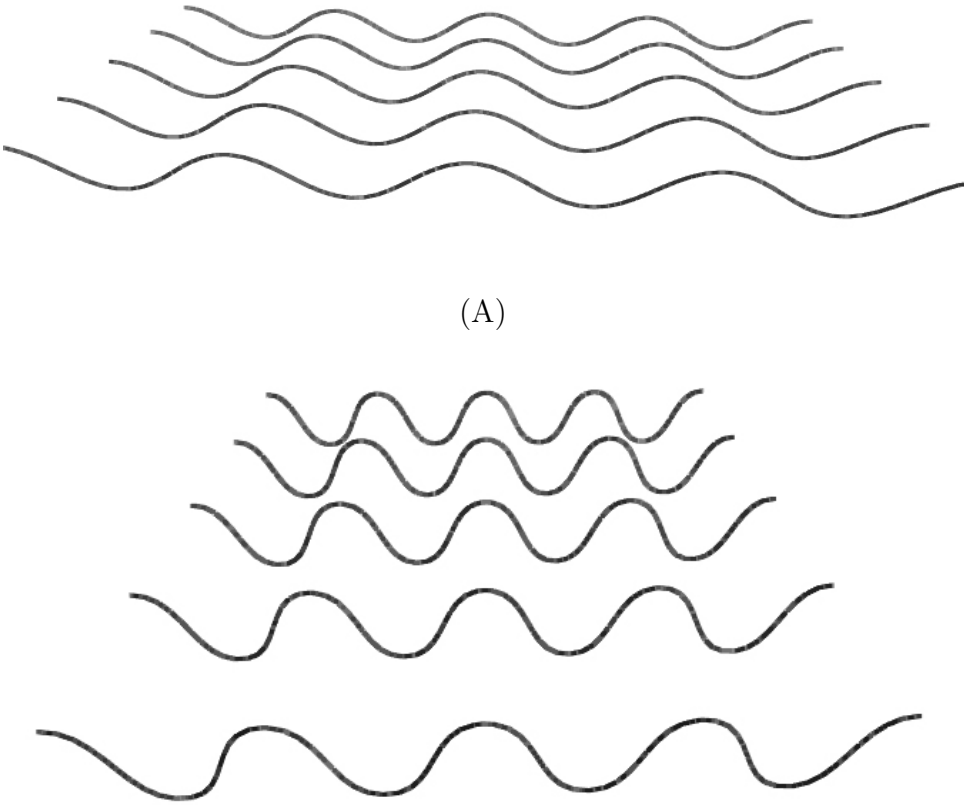
the time step is greater than zero, crests will move from one to the next as global time transitions from one integer value to the next. As mentioned, pose data is also interpolated between state input files (if more than one exists). Thus, we give artists complete control over nearly all wave design parameters, allowing them to generate a limitless variety of beauty waves and to animate them over time. The main drawback to this approach is the tedium involved in specifying parameters, which currently must be modified by hand.

B. Rendering

X-Waves are rendered using basic point and line segment primitives in openGL and displayed using GLUT. Users have the option of rendering either lines, points, or both for wave forms and/or Bezier curve hulls. Rendering options, including wave form opacity and color for lines and points, are specified in the input control file. Our display is rather primitive, as our primary focus was wave form generation. Improved rendering methods and 3-D surface generation are left for future work.

C. Results

Figures 21 through 24 are screen shots showing a variety of wave forms computed using our method. The state in Figure 21 has input where only one crest pose is specified for only one X-Wave. The same pose is used to represent four crests on five X-Waves with (21·A) and without (21·B) crest separation applied. Figure 22 uses the pose data shown in Appendix B, where the first X-Wave is a copy of the last (since the first X-Wave in Appendix B gives value ranges). Figure 23 shows a random sample of crest forms, and Figure 24 shows an example of beauty waves with breaking crests. Animated beauty waves show that in can be difficult to choose crest parameters that



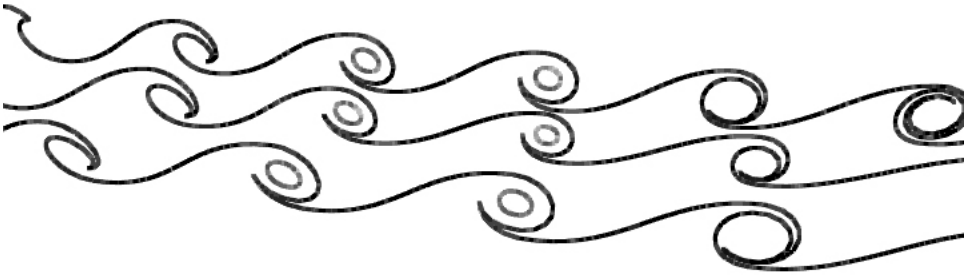
(A)

(B)

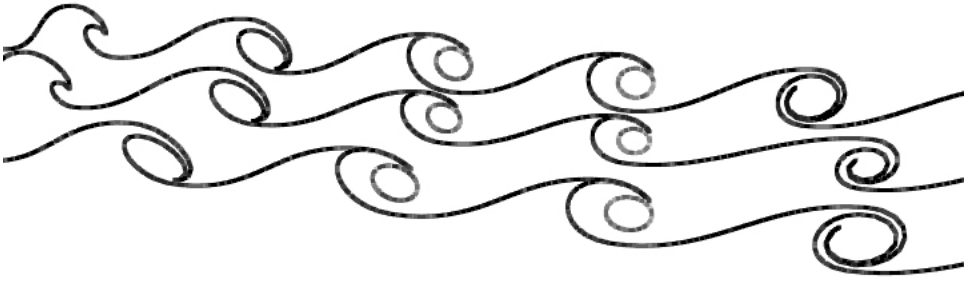
Fig. 21. Basic wave forms computed using beauty wave method using only one specified pose with (A) and without (B) added crest separation.

Fig. 22. Beauty waves using pose data shown in Appendix B.

Fig. 23. More beauty waves.



(A)



(B)

Fig. 24. Breaking beauty waves at two time samples.

produce smoothly flowing waves. It is necessary to take a trial by error approach in selecting proper values to give a desired output. As one would expect, parameters that vary uniformly along an X-Wave and between crests of adjacent X-Waves produce better results. Because we only interpolate between adjacent crest poses, it is also best to position crest poses of any particular X-Wave so that they are nearly evenly spaced along the x axis.

CHAPTER V

CONCLUSION AND FUTURE WORK

The main goal of this work is to provide a method of generating and animating wave forms to represent those found in both nature and in art, and to provide users with direct control over wave definition.

We derived our method from the Gerstner wave model which shows that a point tracing a circular path over time produces the general shape of a wave. We use circles at crest and trough locations coupled with Bezier curves to define wavelengths along the x axis. A series of wavelengths defines an X-Wave, and a series of X-Waves along the z axis defines beauty waves, or a representation of a 3-D fluid surface using 2-D slices. Crest shapes are governed by poses given as eight, user specified control parameters plus a position. Crest poses are interpolated based on the current time and the current integer time interval. Crest pose data is contained in one or more state input files which are interpolated over time (if more than one file) based on a specified file transition time. Main program initialization and control parameters are specified in a separate control input file.

Our method can produce a basic wave form and breaking crests with a rotation that is cyclical in time. Breaking crest shapes are created by computing a circular spiral of diminishing radius based on a given crest radius. Control is also given over crest core shape. Our model can be used to create many interesting wave forms, however, data entry can be tedious and it can be difficult to produce waves that animate as smoothly as one might desire.

Future work can include designing a method for automating user input while still allowing direct control over crest shape. A user interface could also be developed to aid in data entry and for direct manipulation of pose data during runtime.

X-Waves could be spanned to create a three-dimensional surface which could then be displayed using advanced rendering techniques, or ink strokes could be rendered on top of X-Waves to produce animated ink paintings similar to those found in Asian art. Future work can also include computation of breaking crests at larger angles of crest rotation, or in resolving the problems encountered when transitioning into and out of rotating crest forms. Although not implemented, our method can be capable of using recursion to produce wave forms on top of wave forms. This could be used to produce a sawtooth effect or the hook shaped forms found in Hokusai's painting "The Great Wave at Kanagawa" (Figure 2).

REFERENCES

[1] D. Enright, S. Marschner, and R. Fedkiw, "Animation and Rendering of Complex Water Surfaces," *ACM Trans. Graphics*, vol. 21, no. 3, pp. 736–744, July 2002.

[2] N. Foster and R. Fedkiw, "Practical Animation of Liquids," *Computer Graphics (SIGGRAPH '01 Proc.)*, E. Flume, ed., pp. 23–30, Aug. 2001.

[3] N. Foster and D. Metaxas, "Realistic Animation of Liquids," *Graphics Models and Image Processing*, vol. 58, no. 5, pp. 471–483, June 1996.

[4] A. Fournier and W.T. Reeves, "A Simple Model of Ocean Waves," *Computer Graphics (SIGGRAPH '86 Proc.)*, D.C. Evans and R.J. Athay, eds., vol. 20, no. 4, pp. 75–84, Aug. 1986.

[5] D. Hinsinger, F. Neyret, and M.-P. Cani, "Interactive Animation of Ocean Waves," *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation '02*, pp. 161–166, July 2002.

[6] A. Hiroshige, "Famous Views of the 60-odd Provinces (prints 15, 35, & 55)," Edo Period (1615-1868), c.a. 1853-56, Polychrome Woodblock Print, Ink and Color on Paper, Accessed October 7, 2006 from http://www.hiroshige.org.uk/index.html.

[7] K. Hokusai, "The Great Wave at Kanagawa (from a series of Thirty-Six Views from Mount Fuji)," Edo Period (1615-1868), c.a. 1830-32, Eijudo, Japan, Polychrome Woodblock Print, Ink and Color on Paper, H. O. Havemeyer Collection (JP1847), Accessed October 7, 2006 from http://www.metmuseum.org/Works_of_Art.

[8] G. Irving, E. Guendelman, Frank Losasso, and R. Fedkiw, "Efficient Simulation of Large Bodies of Water by Coupling Two and Three Dimensional Techniques," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 805–811, July 2006.

[9] S. Jia, *Lessons in Chinese Landscape Painting*, Wan Li Book Co. LTD, 1987.

[10] M. Kass and G. Miller, "Rapid, Stable Fluid Dynamics for Computer Graphics," *Computer Graphics (SIGGRAPH '90 Proc.)*, F. Baskett, ed., vol. 24, no. 4, pp. 49–57, Sept. 1990.

[11] V. Mihalef, D. Metaxas, and M. Sussman, "Animation and Control of Breaking Waves," *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation '04*, pp. 315–324, Aug. 2004.

[12] M. Müller, D. Charypar, and M. Gross, "Particle-Based Fluid Simulation for Interactive Applications," *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation '03*, pp. 154–159, July 2003.

[13] D.R. Peachey, "Modeling Waves and Surf," *Computer Graphics (SIGGRAPH '86 Proc.)*, D.C. Evans and R.J. Athay, eds., vol. 20, no. 4, pp. 65–74, Aug. 1986.

[14] D.F. Rogers and J.A. Adams, *Mathematical Elements of Computer Graphics*, second ed., WCB/McGraw-Hill, 1990.

[15] P. Ts'o and B. Barsky, "Modeling and Rendering Waves: Wave-Tracing Using Beta-Splines and Reflective and Refractive Texture Mapping," *ACM Trans. Graphics*, vol. 6, no. 3, pp. 191–214, July 1987.

APPENDIX A

BEAUTY WAVE CONTROL INPUT PARAMETERS

Following is the list of parameters contained in the control input file JBWinput.txt for beauty wave initialization. The order and number of parameters must be the same as listed below. C/C++ style comments //...// and /*...*/ may follow parameters as long as there is space between the parameter and comment. Parameters are listed here as a range of values such as 0->1 or as either of two values such as 0/1, where 1 is on and 0 is off. Parameter types are either integer, floating point, or string, and are listed here for reference only. They should not be included in the input file. Only one value should be provided for each parameter range, # (any number), or string. File names will be appended with a ".####.txt" extension at runtime, with output files beginning at 0 and the pose input file beginning at the specified beautyWave state input file number. All parameters must have a value provided even if they are not being used, such as output file names. There must also be a pound sign (#) on its own line at the end of the file. Note that improper file format will force the program to terminate.

```
(integer)   0/1     // Flag, write Bezier curve point output files
(integer)   0/1     // Flag, write wave pt output files (if # of wave pts >0)
(integer)   0/1     // Flag, pause time (stop computing)
(integer)   0->3    // Flag, xWave draw type
                    //   0: draw nothing; 1: draw lines only (default)
                    //   2: draw lines and points; 3: draw points only
(integer)   0->3    // Flag, Bezier curve draw type
```

```
                    //   0: draw nothing (default); 1: draw lines only
                    //   2: draw lines and points; 3: draw points only
(integer)   >=1     // Number of xWaves to draw
(integer)   >=0     // Number of wave points to compute per xWave
(integer)   >=1     // Number of crests (poses) per xWave
(integer)   >=1     // Number of beautyWave state input files
(integer)   >=0     // Starting beautyWave state input file number (no preceding 0's)
(float)     0->1    // Time step per compute cycle
(float)     >0      // Transition time between pose input files
(float)     >=0     // Added separation between crest poses
(float)     >0      // Lateral separation between repeated beautyWaves, if any
(float)     >0.3    // Minimum radius before transitioning pose to 0
(float)     0->1    // Line opacity for wave points
(float)     0->1    // Point opacity for wave points
(float)     0->1    // Line opacity for breaker air/drop
(float)     0->1 0->1 0->1      // Line color for wave points
(float)     0->1 0->1 0->1      // Point color for wave points
(float)     0->1 0->1 0->1      // Line color for breaker air/drop
(float)     # # #               // Camera position
(float)     # # #               // Camera look-at point
(string)    poseInputFileName   // Pose input filename (no # or extension)
(string)    bezierOutFileName   // Bezier output file name (no # or extension)
(string)    wavePtOutFileName   // Wave point output file name (no # or extension)
#                               // End of file flag
```

APPENDIX B

BEAUTY WAVE STATE INPUT PARAMETERS

Following is the list of input parameters contained in a beauty wave state input file, along with examples for formatting pose and X-Wave data. The pose parameters must be listed in order from 1 to 9 for the parameters specified, and the list number must precede the actual value on the same line with whitespace separating the two. Only the first pose of any particular X-Wave needs to have all nine parameters provided. If more than one pose is being computed, any parameter not specified for poses beyond the first will be copied from the previous pose. Copied position data will shift poses in the x-direction by a distance of four times the radius plus an end separation provided in the control input data. although only one pose need be included for any one X-Wave, it is a good idea to at least provide the position data. All z positions are verified to be identical to the first pose for any X-Wave. At least one X-Wave (with one or more poses) must be defined for any file.

C/C++ style comments //...// and /*...*/ may follow parameters as long as there is space between the parameter and comment. Parameters are listed here as a range of values such as 0->1 or as either of two values such as 0/180, where 180 is on and 0 is off. Parameter types are either integer or floating point and are listed here for reference only. They should not be included in the input file. Only one value should be provided for each parameter (plus the preceding list number). X-Wave definitions must begin with a '%' delimiter and poses must be separated by a '@' delimiter. There must also be a pound sign (#) on its own line at the end of the file, and the number of X-Waves defined in the file must equal or exceed the number provided at the top of the file. The state input file names need to have a

format of "filename.000#.txt", where the desired starting number is specified in the control input without the preceding zeros. Note that improper file format will force the program to terminate.

4	(integer)	>=1	// Number of xWaves defined
%			// X-Wave delimiter
1	(float)	0->1	// Crest radius fraction, F_R
2	(float)	0->1	// Crest thickness fraction, F_T
3	(float)	0->1	// Crest core radius fraction, F_{CR}
4	(float)	0->1	// Crest spiral fraction, F_S
5	(float)	>=0	// Cusp tangent angle, θ_t (input in degrees)
6	(float)	>=0	// Crest curl angle, θ_c (input in degrees)
7	(float)	0/180	// Crest rotation angle, θ_R (input in degrees)
8	(float)	0->$\#$	// Crest circle radius
9	(float)	$\#$ $\#$ $\#$	// Crest circle center
%			// X-Wave delimiter
1		0.7	// Crest radius fraction, F_R
2		0.3	// Crest thickness fraction, F_T
3		0.5	// Crest core radius fraction, F_{CR}
4		0.6	// Crest spiral fraction, F_S
5		180	// Cusp tangent angle, θ_t (input in degrees)

6	135.3	// Crest curl angle, θ_c (input in degrees)
7	0	// Crest rotation angle, θ_R (input in degrees)
8	1.7	// Crest circle radius
9	-0.3 0.3 5.5	// Crest circle center
@		// Pose delimiter
6	182	// Crest curl angle, θ_c (input in degrees)
9	8 -0.2 1.5	// Crest circle center
@		// Pose delimiter
5	140	// Cusp tangent angle, θ_t (input in degrees)
6	271	// Crest curl angle, θ_c (input in degrees)
9	16.3 0 8	// Crest circle center
%		// X-Wave delimiter
1	0.5	// Crest radius fraction, F_R
2	0.4	// Crest thickness fraction, F_T
3	0.8	// Crest core radius fraction, F_{CR}
4	0.6	// Crest spiral fraction, F_S
5	170	// Cusp tangent angle, θ_t (input in degrees)
6	175	// Crest curl angle, θ_c (input in degrees)
7	0	// Crest rotation angle, θ_R (input in degrees)
8	1.6	// Crest circle radius
9	0 0.2 11	// Crest circle center
@		// Pose delimiter
5	150	// Cusp tangent angle, θ_t (input in degrees)

6	182	// Crest curl angle, θ_c (input in degrees)
9	7 -0.2 11	// Crest circle center
@		// Pose delimiter
5	130	// Cusp tangent angle, θ_t (input in degrees)
9	14 -0.2 11	// Crest circle center
%		// X-Wave delimiter
1	0.9	// Crest radius fraction, F_R
2	0.5	// Crest thickness fraction, F_T
3	1.0	// Crest core radius fraction, F_{CR}
4	0.8	// Crest spiral fraction, F_S
5	160.4	// Cusp tangent angle, θ_t (input in degrees)
6	89.1	// Crest curl angle, θ_c (input in degrees)
7	0	// Crest rotation angle, θ_R (input in degrees)
8	1.45	// Crest circle radius
9	0 0 16.5	// Crest circle center
#		// End of file flag

VITA

Jay Allen Faulkner
Texas A&M University
Visualization Laboratory
Department of Architecture
Langford Center, C418
College Station, TX 77843-3137
jaf@viz.tamu.edu

Education

M.S. in Visualization Sciences	Texas A&M University, December 2006
B.S. in Aerospace Engineering	Texas A&M University, December 1995

Employment

Software Developer,	MultiGen-Paradigm Inc., October 2006 - Present
Technical Director Intern,	Pixar Animation Studios, January 2001 - August 2001
Test Director,	Lockheed-Martin, January 1996 - July 1999