Creating a vpMarineShip

After you have created your ocean, you can now create a ship and simulate its movements on your new ocean. In this example we are working with the existing tugboat ship that is used in the `vpmarine_simple.acf`, included with the distribution.

Using the Vega Prime Marine **vpMarineShip** class, you can create a ship that responds to the oceans you have already created, as well as rendering various marine effects that respond to the ship's size, speed and maneuvering.

Creating a Ship

Using **vpMarine**, you can create your own ship for your ocean, but there are a few things you need to know before you begin. The following sections discuss the information you need to know to create your ship.

Creating a Ship?

For the advanced users, the term “creating” may be a little confusing. What are we actually creating? Are we going to create a model of a ship? Are we going to create a ship object?

We are actually “creating” two things: a ship definition and a ship object. The ship definition is a **vpMarineShip** instance that is associated with a ship object. The ship object is a **vpObject** configured to respond like a ship. The **vpObject** is tied to an actual model of a ship, defined by the `tugboat.flt` file.

Ship Coordinate System

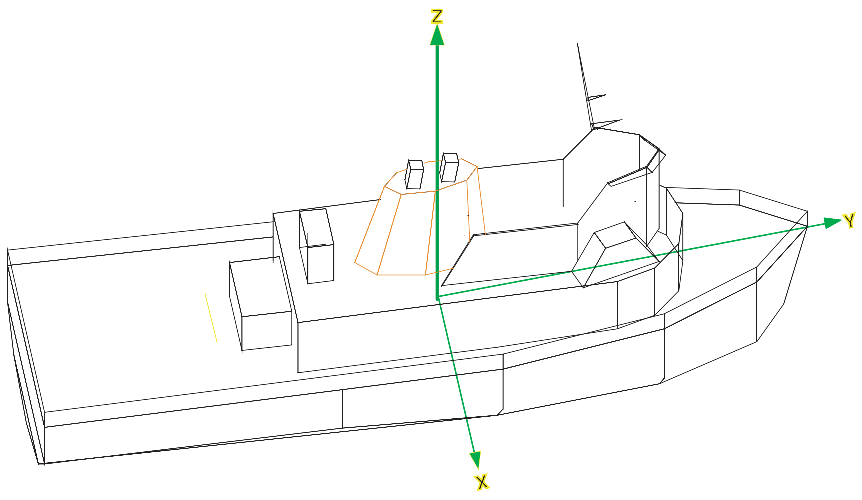
vpMarineShip assumes an X, Y, and Z coordinate system is applied to your ship. This coordinate system assumes that the front of your ship is on the positive Y-axis and the right side of the ship on the positive X-axis. The positive Z-axis is assumed to be coming straight up from the center of the ship.

Ship Terminology

In these sections describing the **vpMarineShip** API, informal terms are used. This is done so that the procedures are clear to those of us seeing a ship for the first time. However, there ARE correct nautical terms to describe the various parts of the ship, and you should be aware of them.

Bow	The forward end of the ship.
Stern	The back end of the ship.
Port	The left side of the ship.
Starboard	The right side of the ship
Fore	At or toward the bow of the vessel.
Aft	Close to or toward the stern of the vessel.

When you model your ship, the geometry of your ship should be positioned so that the ship is up, in relation to the water line. For our examples, we are using the tugboat model, included with the distribution. In the figure below you can see that the origin of the tugboat is at the intersection of the centerline of the tugboat (Y-axis), running from front to back, and the X-axis running right to left.



Note: The X-axis and Y-axis should be aligned on the water line of your ship. (Actually if you want to think of it in this manner, the Y-axis IS the waterline for your ship.) The Z-axis should be positioned at the origin of the ship. The intersection of the three axes is the origin of the ship. This is important because many other measurements of the ship are made in relation to this origin point.

Ship Dimensions & Attributes

In order to simulate a ship on your ocean, **vpMarineShip** needs to know certain dimensions about your ship. **vpMarineShip** also needs to know the ocean and **vpTransform** associated with your ship. All of these attributes influence any subsequent marine effects that you attach to your ship.

The **vpMarineShip** properties are described in the subsequent sections along with the specific **vpMarineShip** method used to set these properties. At the end of this section, you will find a sample code fragment showing how all of these methods are used to create a tugboat for use with your previously-created observer centered ocean.

The following properties and their values are used by **vpMarineShip**:

Ship attribute	vpMarineShip API method
Ocean	`setOcean();`
Transform	`setTransform();`
Beam	`setBeam();`
Width at Origin	`setWidthAtOrigin();`
Maximum Speed(knots)	`setMaxSpeed();`
Maximun Turn Rate	`setMaxTurnRate();`
Bow Flare Angle	`setBowFlareAngle();`
Bow Offset From Origin	`setBowOffset();`
Bow Width	`setBowWidth();`
Bow Length	`setBowLength();`
Bow Freeboard	`setBowFreeboard();`
Stempost Angle	`setBowStemPostAngle();`
Stempost Length	`setBowStemPostLength();`
Stern Offset From Origin	`setSternOffset();`
Stern Width	`setSternWidth();`
Stern Wake Angle	`setSternWakeAngle();`

Assigning an Ocean

In our previous example, you created an observer-centered ocean. Your ship will sail on this ocean; therefore, **vpMarineShip** needs to know about your previously defined **vpMarine** ocean.

Note: Your ship is not limited to sailing on only one ocean. As your ship moves, oceans can be switched underneath, to simulate crossing several oceans. Switching oceans can only be performed at the API level, however.

Using our previous example, where you created an observer-centered ocean, the following code shows how to associate this ocean with your ship.

```
m_pOceanObsCentered = new vpMarineOceanObserverCentered;
m_pOcean = m_pOceanObsCentered;
```

```
m_pShip = new vpMarineShip;
m_pShip->setOcean( m_pOceanObsCentered );
```

Assigning the vpTransform

A **vpObject** is a **vpTransform**, which inherently means the object contains a coordinate system that enables the object to be positioned and moved. Assigning a **vpTransform** or **vpTransform**-derived instance to your **vpMarineShip** enables the **vpMarineShip** to monitor the ship's speed and turn rate. Ship phenomena, such as bow waves and stern wakes, dynamically modify their geometry based upon these two measurements. The following code shows how to assign a **vpTransform** to our ship definition.

```
vpObject *m_pShipObj;
vpMarineShip *m_pShip;
m_pShip = new vpMarineShip;

m_pShip->setTransform( m_pShipObj );
```

Note: The **vpTransform** can be changed dynamically at runtime only through the API.

Maximum Speed & Turn Rate

Maximum speed is the maximum speed, in knots, that your ship or vessel can reach. The maximum speed is one of the most significant factors that influences the size and shape of ship phenomena.

Maximum turn rate is the maximum speed that your ship can turn, in a given time period of time. This value is expressed in degrees per second. The maximum turn rate is another significant factor influencing the size and shape of ship phenomena.

Maximum speed and turn rate are usually known values that can be determined about the ship you are going to simulate. These values can be found and are often ratings that can be accessed from your ship's manufacturer, for a given class of ship.

For our tugboat, we know that it has a maximum speed of 35 knots and a maximum turn rate of 5.5 degrees per second. The following code shows how to set these values.

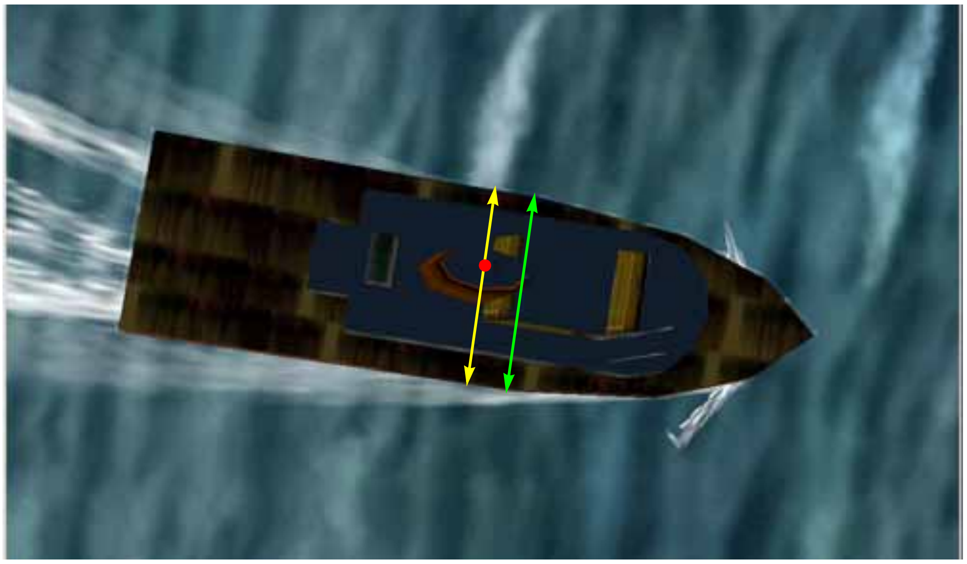
```
m_pShip->setMaxSpeed( 35.0f );
m_pShip->setMaxTurnRate( 5.5f );
```

Beam & Width at Origin

The beam is the extreme width of the ship at the waterline. Bow waves and hull wakes depend upon this property, in order to conform to the ship's shape.

The origin of the ship represents that ship's center of gravity along the centerline of the ship. The width at the origin enables the **vpMarineShip** API to compute two important locations along the ship's hull. The **vpMarineShipMotionStrategy** depends on these two locations. See "vpMarineShipMotionStrategy" on page 216.

In the figure below, the line to the right represents the tugboat's beam. This is the width of the ship at its widest point. The line to the left represents the width of the tug through the origin. You may notice a dot on this line. This is the origin of the ship.



For our tugboat example, we know that the tugboat's beam is 9 meters wide. The width of the tugboat at the origin is 7 meters. You would enter the following to set these values in code:

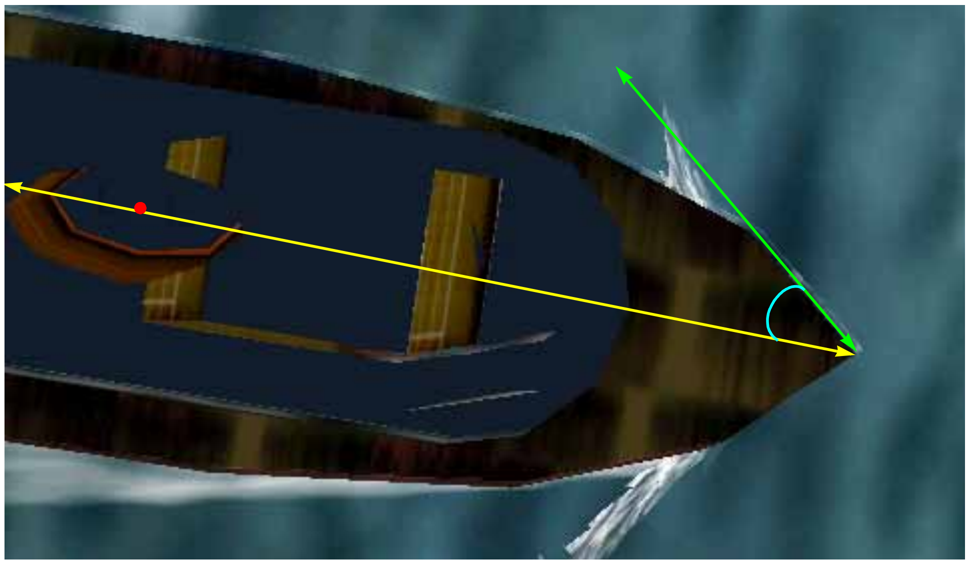
```
m_pShip = new vpMarineShip;
m_pShip->setBeam( 9.0f );
m_pShip->setWidthAtOrigin( 7.0f );
```

Bow Flare Angle

The bow flare angle is measured horizontally with respect to the centerline of the ship running fore and aft. The bow wave will follow the "flared out" shape of the bow.

Note: The bow flare angle can ONLY be within the range of 0 – 45 degrees or 90 degrees exactly. The 90 degree shape is for horizontal flare angles, that is, ships having a horizontal or level bow. Selecting the **Level** checkbox in LynX Prime sets this parameter to 90 degrees.

In the figure below, the line running down the center of the tugboat represents the centerline of the boat. The line running at an angle to the center line, is following the flare of the bow of the tugboat. The angle of these two lines represents the bow flare angle.

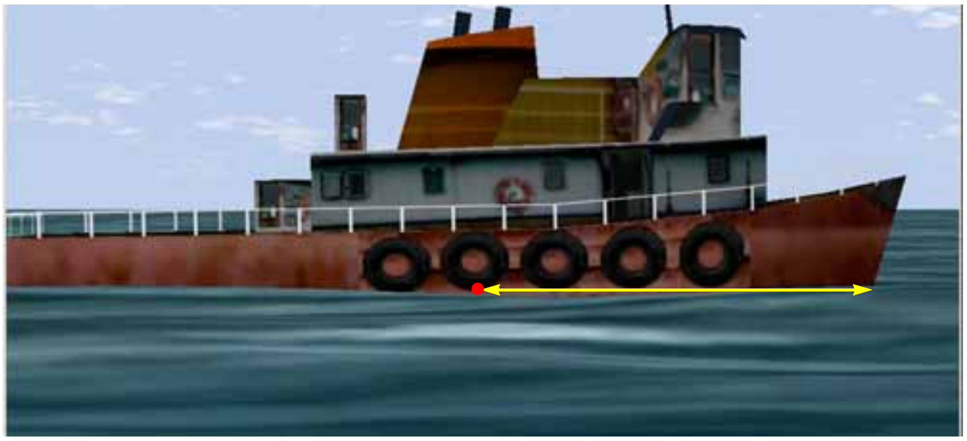


For out tugboat we determine that the flare angle is 44 degrees. The following code will set this value.

```
m_pShip->setBowFlareAngle( 44.0f );
```

Bow Offset

The bow offset is the distance from the ship's origin to the beginning of the ship's bow frame. This distance goes to the bow's origin and does not include the stempost. The figure below shows the bow offset. You may notice a dot on this line. This is the origin of the ship.



For our tugboat, we know that this bow offset distance is 13.7 meters. The following code sets this value.

```
m_pShip->setBowOffset( 13.7f );
```

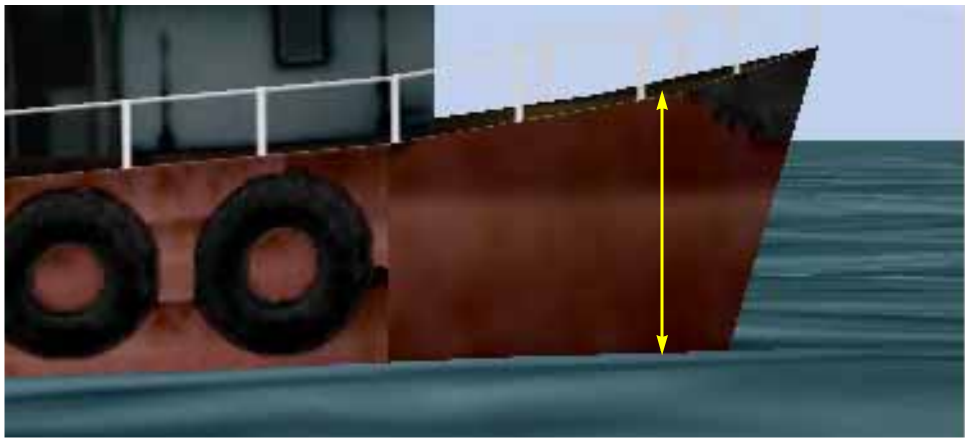
Ship Terminology (continued).

Stem	1. The timber at the extreme foward part of the boat secured to the forward end of the keel.
	2. The bow frame forming the apex of the intersection of the forward sides of a ship, that is rigidly connected at its lower end to the keel. The extension of the keel and the foremost part of the stem is referred to as the **stempost** or the **cutwater**. It may be rounded, streamlined, or straight, and it cuts the water as the vessel forges ahead.
Keel	1. The centerline of a ship running fore and aft. The backbone of a vessel.
	2. A steel beam or timber, or a series of beams or timbers joined together, extending along the center of the bottom of a ship from stem to stern and, to which the frames and hull plating are attached.,

Bow Freeboard

The bow freeboard is the vertical length of the side of the hull, near the bow's origin, that is above the water line. You need a freeboard measurement because medium and large ships produce a modest vertical (upward) bow wave wash. The height of this wash is a percentage of the available freeboard. However, a high speed patrol boat moving ahead at 45 knots will create a noticeable upward wash with spray ABOVE the freeboard at the bow, that moves aft and reduces along the waterline. In this case, you should specify the freeboard value in the context of the maximum achievable height of the spray.

The figure below shows the bow's freeboard.



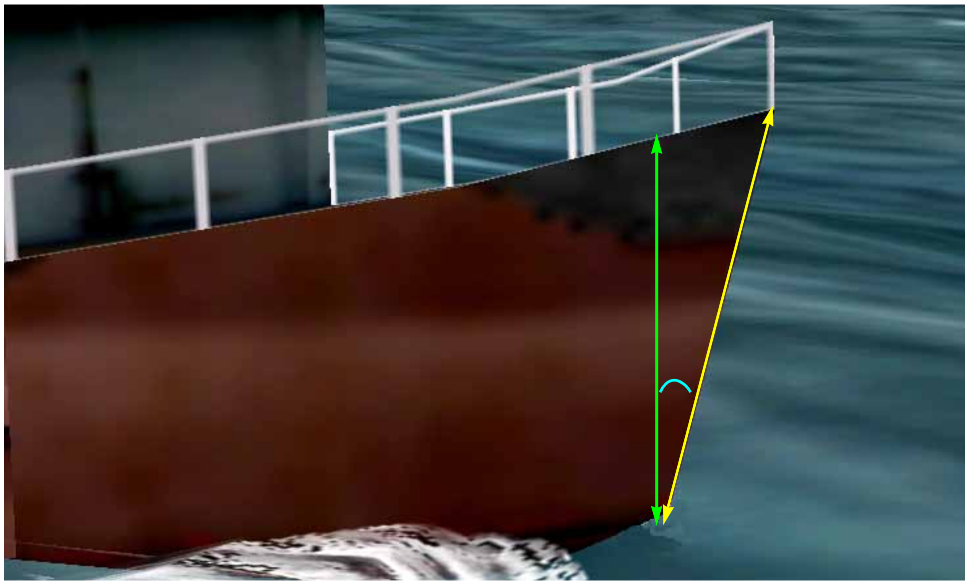
For our tugboat, we are going to set the freeboard to a value of 3.22 meters. The following code shows how to set this value.

```
m_pShip->setBowFreeboard( 3.22f );
```

Note: As you fine-tune your application, you may want to adjust this value so that the waves washing across the bow, as the ocean waves change height, look more natural.

Bow Stempost

The stempost angle is the angle between the ship's forward perpendicular plane and its stem post. A positive angle indicates that the top of the stem is angled away from the ship's origin. A negative angle indicates that the top of the stem is angled towards the ship's origin. The line on the left represents the location of the ship's forward perpendicular plane. The line on the right follows the stempost of our tugboat. The angle between these two lines is the bow's stempost angle.

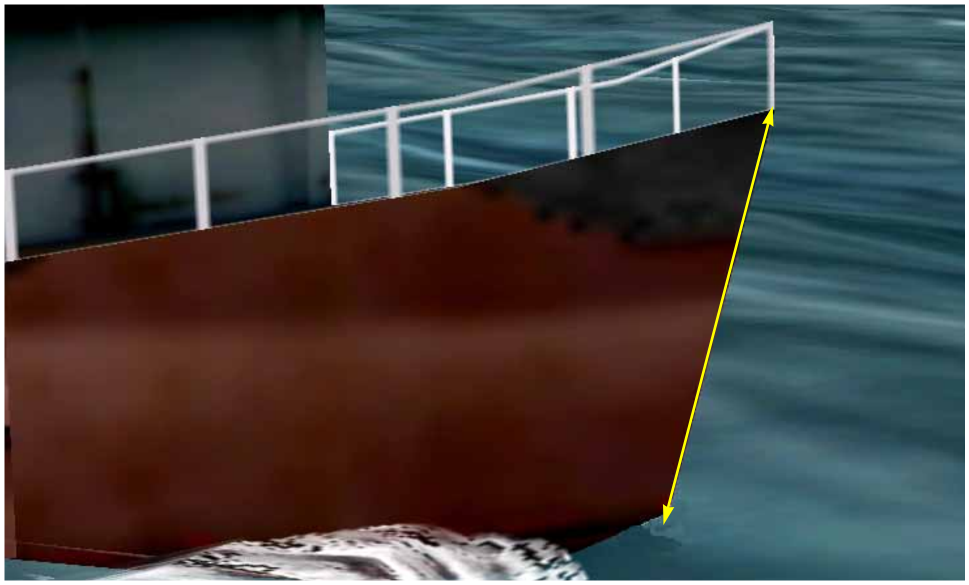


For our tugboat, this angle is 14 degrees. The following code shows how to set this value.

```
m_pShip->setBowStempostAngle( 14.0f );
```

Bow Stempost Length

The stempost length is the distance from the tip of the keel, in front, to the foremost part of the stem. While your stempost may be rounded, streamlined, or straight, **vpMarine** assumes the stempost to be straight. During runtime the **vpMarineShip** API computes the point of contact of the water with the stempost. This is because the stem is usually not straight and vertical. It usually angles outward. Therfefore, as the ship pitches down, the stem intersects the water farther out. As it pitches up, the intersection point is reduced, relative to the origin of the ship. The figure below shows the bow's stempost length.



For our tugboat, we know that the length of the bow's stempost is 1.8 meters. The following codes shows how to set this value.

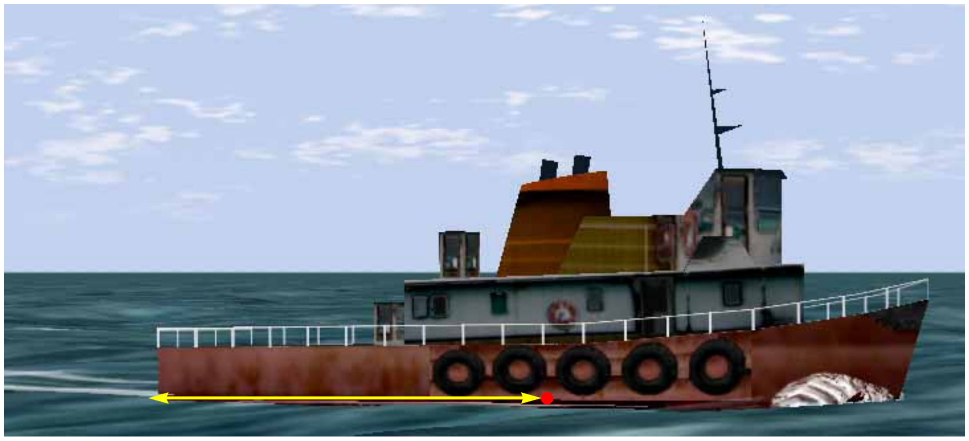
```
m_pShip->setBowStempostLength( 1.8f );
```

Note: The measurement of the bow's stempost length is from the end of the stem to the keel of your ship, not to the waterline.

The ship's bow wave is subsequently translated based upon the pitch of the vessel, the angle and length of the stem, and the ever-changing elevation of the water surface along the stempost.

Stern Offset

The stern offset is the distance from the origin of the ship to the stern of the ship. If you think of the origin of the ship starting at zero, and the distance toward the bow being in the positive direction, then the distance to the stern will always be a negative number. The figure below shows the stern offset. You may notice a dot on this line. This is the origin of the ship.

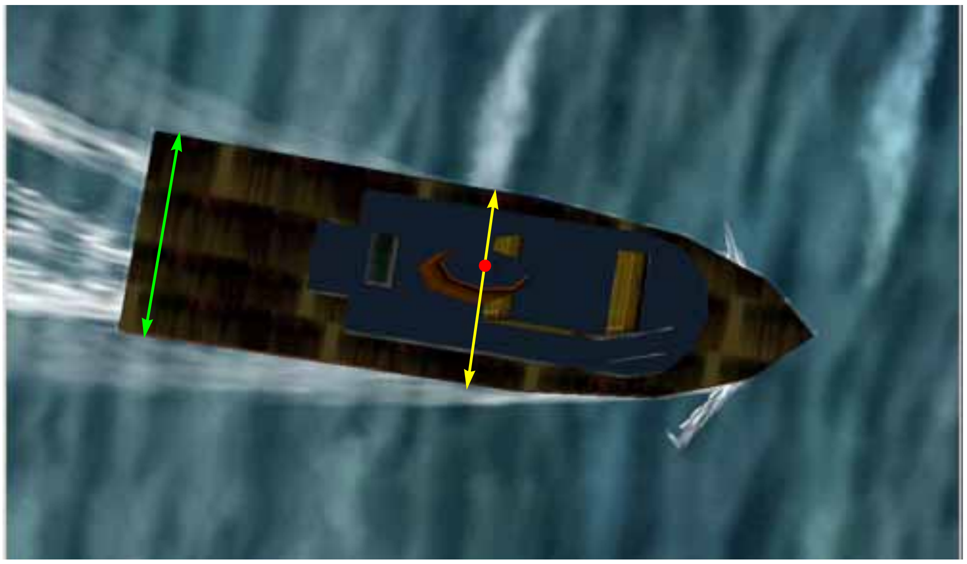


For our tugboat we know that the stern offset distance is 15.5 meters. Since we are moving from the origin of the ship, to the stern, in a negative direction, the distance is expressed as -15.5 meters. The following code shows how to set this value.

```
m_pShip->setSternOffset( -15.5f );
```

Stern Width

The stern width is the horizontal width of the ship's stern. In the figure below, the line to the left represents the tugboat's stern width. This is the horizontal width of the ship at the stern. The line to the right is for reference, and represents the width of the tug through the origin. (The width at the origin may be different than the width at the stern.) You may notice a dot on this line. This is the origin of the ship.



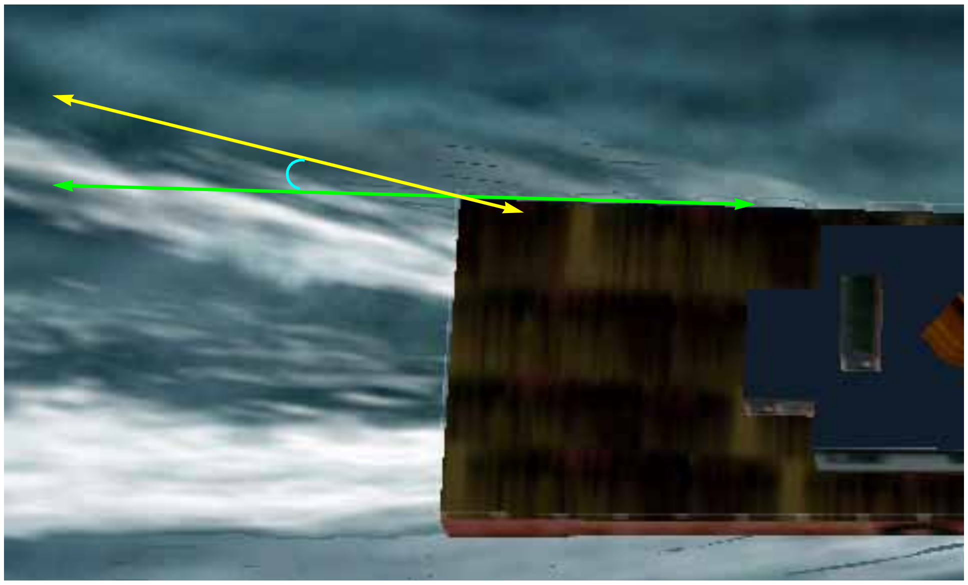
For our tugboat example, we know that the tugboat's stern width is 9 meters. The following code shows how to set this value.

```
m_pShip->setSternWidth( 9.0f );
```

Stern Wake Angle

The stern wake angle specifies what the angle of the wake should be, off the ship’s stern. This angle is measured with respect to the horizontal line, in degrees, on the port side of the ship, parallel to the centerline.

In the figure below, the angle specified is that angle between the line of the ship, parallel to the centerline and the outer edge of the stern wake.



For our tugboat, we want the stern wake angle to be 8 degrees off the port side of the ship. The following code shows how to set this value.

```
m_pShip->setSternWakeAngle( 8.0f );
```

The Complete Code

Now that we have discussed all the parameters needed to define your tugboat, here is all the code in one place needed to define your tugboat.

```
#include “vpMarineShip.h”

     //------------------------------------------------------
    // Define the parameters of the ship
    //----------------------------------------------------------
#include “vpMarineShip.h”


    vpMarineShip *m_pShip;

    m_pShip = new vpMarineShip;

    m_pShip->setName( "Tugboat Definition" );
    m_pShip->setTransform( m_pShipObj );
    m_pShip->setMaxSpeed( 35.0f );                    // knots
    m_pShip->setMaxTurnRate( 5.5f );
    m_pShip->setBeam( 9.0f );
    m_pShip->setWidthAtOrigin( 7.0f );

    m_pShip->setBowFlareAngle( 44.0f );
    m_pShip->setBowWidth( 1.5f );
    m_pShip->setBowLength( 2.0f );
    m_pShip->setBowOffset( 13.7f );
    m_pShip->setBowFreeboard( 3.22f );
    m_pShip->setBowStempostAngle( 14.0f );
    m_pShip->setBowStempostLength( 1.8f );

    m_pShip->setSternOffset( -15.5f );
    m_pShip->setSternWakeAngle( 8.0f );
    m_pShip->setSternWidth( 9.0f );
```

Note: You can find many code samples in your Vega Prime samples directory.

Using a LynX Prime .ACF file

For the LynX Prime user, you may have created some .ACF files that contain definitions and ship attributes that you want to use. This information can be accessed from your .ACF file and used to set up your ship, rather than hardcoding the ship attributes. You may want to do this if you want to change various attributes about your ship quickly, but need the flexibility of a separate application. The following code example shows how you can load an .ACF file to define your tugboat.

```
myApp *pMyAPP = new myApp;

// Use “mytug.acf” for defining my ship,
// or whatever I pass as the first argument.

if ( argc == 1 )
   pMyApp->define( “mytug.acf” )
else
   pMyApp->define( argv[1] );

vpObject *m_pShipObj;
vpMarineShip *m_pShip;

// Find the object I called “TugObj” and use this to define
// my ship object.
m_pShipObj = vpObject::find( “TugObj” );

// Find the vpMarineShip I called “TugDef” and use this to
// define my ship.
m_pShip = vpMarineShip::fine( “TugDef” );
```

vpMarine Effects

There are four different **vpMarine** effects that are available. The **vpMarine** effects are:

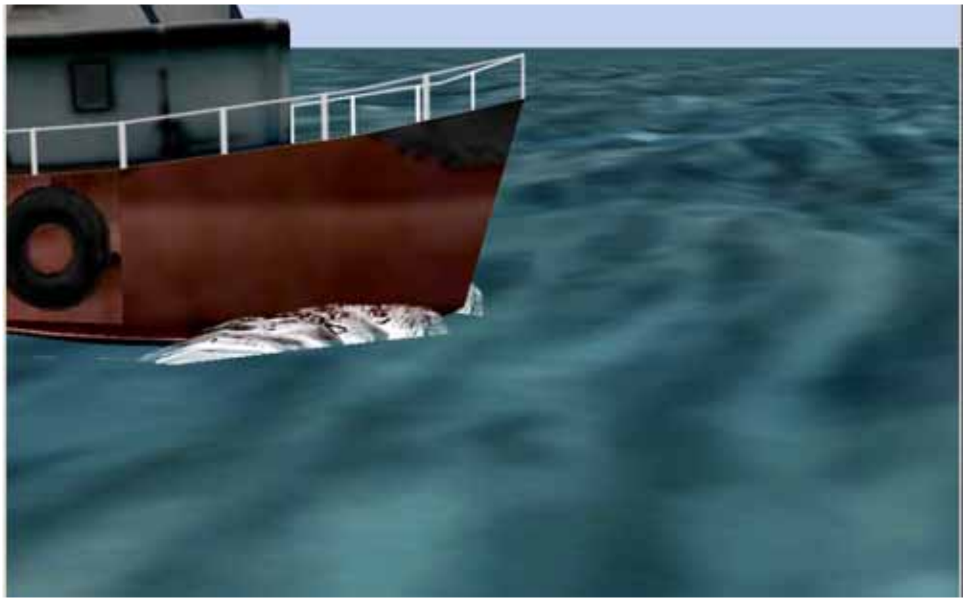
- Angled Bow Wave
- Level Bow Wave
- Stern Wake
- Hull Wake

These effects are described as well as the code and steps necessary to setup and apply these effects in the following pages.

Bow Waves

Bow waves are formed by the water displaced by the bow of a vessel cutting through the water. They represent the swell of water at the immediate bow of the ship, and indicate the speed and maneuvering of a vessel. **vpMarine** supports two types of bow waves: **Angled** and **Level**.

Angled Bow Wave



vpMarine generates geometry that conforms to bows that are not flat, or horizontal, but instead are angled. A **MarineFxBowWaveAngled** is automatically parented by the **vpTransform** that its **vpMarineShip** references.

The following **vpMarineShip** parameters strongly influence the size and shape of the angled bow wave when it is constructed:

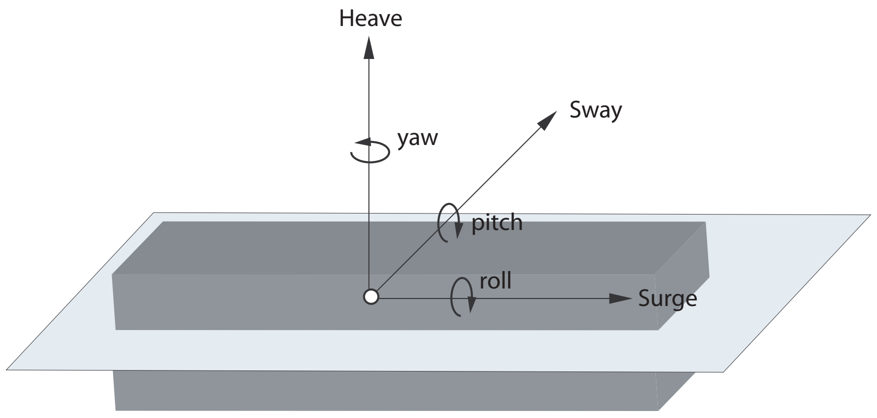
- Freeboard
- Beam
- Bow Width
- Bow Length
- Bow Flare Angle
- Bow Offset from the ship's Origin

- Waterline Length (calculated from the above parameters)

The following response by the ship's physical geometry, to the ocean waves encountered by the vessel, also strongly influence the size and shape of the bow wave:

- Actual speed relative to the maximum speed
- Actual turn rate relative to the maximum turn rate
- Pitch angle
- Roll angle
- Heave displacement

.



Creating an Angled Bow Wave

After you have created your ship and defined all the ship's properties, you can now add a **vpMarine** effect to be attached to your ship. For our example, we will assume that your ship has already been defined.

Setting the Ship Object, Ocean, and Motion Strategy

Here are the steps you will perform before you apply your **vpMarine** effect. These steps are:

- Associate a ship object with your ship definition.
- Add your ship object to your scene.
- Associate your ocean with your ship definition.
- Create a new motion strategy for your ship object.

Note: Motion strategies can only be applied to **vpTransforms**.

- Set the new motion strategy as the motion strategy for your ship object, unless there is already a motion strategy present. (If so, then set the new strategy as the next motion strategy for your ship object.)
- Set your motion strategy to compute the ship's yaw automatically.

The following code fragment shows how to accomplish these steps.

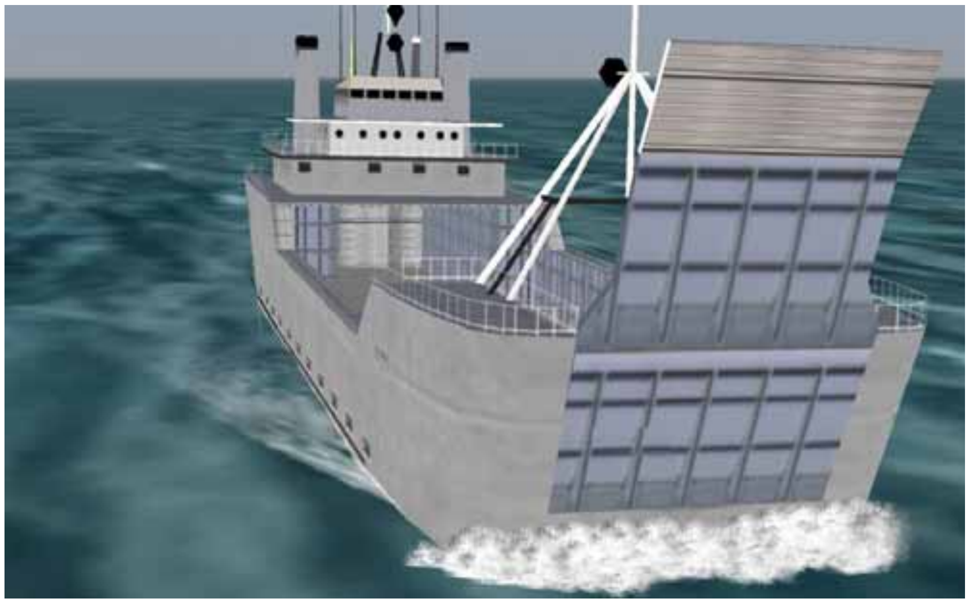
```
vpObject *m_pShipObj;
vpMarineShip *m_pShip;

m_pShip->setTransform( m_pShipObj);
m_pScene->addChild( m_pShipObj );
m_pShip->setOcean( m_pOcean );

m_pShipMotion = new vpMarineShipMotionStrategy;

if ( m_pShipObj->getStrategy() == NULL )
    m_pShipObj->setStrategy( m_pShipMotion );
else
    m_pShipObj->getStrategy()->setNextStrategy( m_pShipMotion );

m_pShipMotion->setShip( m_pShip );
m_pShipMotion->setComputeYawEnable( true );
```

Adding Your Effect

After these steps are performed, you can now set the **vpMarine** effect and the attributes of that effect. The following code shows the steps to create your angled bow wave.

```
m_pAngledBowWave = new vpMarineFxBowWaveAngled;

m_pAngledBowWave->setName( "Bow Wave" );
m_pAngledBowWave->setShip( m_pShip );
m_pAngledBowWave->setEnable( true );
m_pAngledBowWave->configure();
```

Level Bow Wave



Using the level bow wave effect, **vpMarineShip** generates geometry that conforms to bows that are flat horizontally, instead of angled. Level bow waves are simulated by a particle system effect. The effects size and shape are a function of ship velocity. Four **vpMarineShip** parameters can be configured to define the size and shape of the level bow wave:

- The maximum attainable speed by the vessel
- Bow width
- Bow offset from origin
- Freeboard

The **vpMarineShip** bow flare angle must be set to exactly 90.0 (indicative of a flat or horizontal bow.) A **MarineFxBowWaveLevel** is automatically parented by the **vpTransform** that its **vpMarineShip** references.

Creating a Level Bow Wave

After you have created your ship and defined all the ship's properties, you can now add a **vpMarine** effect to be attached to your ship. For our example, we will assume that your ship has already been defined.

Setting the Ship Object, Ocean, and Motion Strategy

Here are the steps you will perform before you apply your **vpMarine** effect. These steps are:

- Associate a ship object with your ship definition.
- Add your ship object to your scene.
- Associate your ocean with your ship definition.
- Create a new motion strategy for your ship object.

Note: Motion strategies can only be applied to **vpTransforms**.

- Set the new motion strategy as the motion strategy for your ship object, unless there is already a motion strategy present. (If so, then set the new strategy as the next motion strategy for your ship object.)
- Set your motion strategy to compute the ship's yaw automatically.

The following code fragment shows how to accomplish these steps.

```
vpObject *m_pShipObj;
vpMarineShip *m_pShip;

m_pShip->setTransform( m_pShipObj);
m_pScene->addChild( m_pShipObj );
m_pShip->setOcean( m_pOcean );

m_pShipMotion = new vpMarineShipMotionStrategy;

if ( m_pShipObj->getStrategy() == NULL )
    m_pShipObj->setStrategy( m_pShipMotion );
else
    m_pShipObj->getStrategy()->setNextStrategy( m_pShipMotion );

m_pShipMotion->setShip( m_pShip );
m_pShipMotion->setComputeYawEnable( true );
```

Adding Your Effect

After these steps are performed, you can now set the **vpMarine** effect and the attributes of that effect. This effect, as defined earlier, is for simulating a level bow wave.

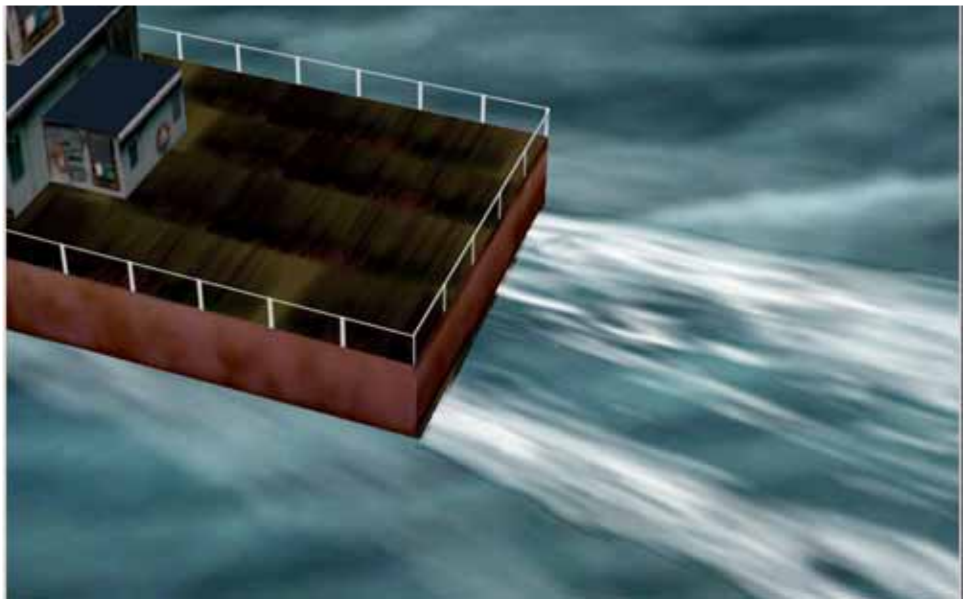
Note: This effect assumes that the bow of your ship is horizontal and that the bow's flare angle is exactly 90 degrees.

The following code shows the steps to create your level bow wave.

```
m_pLevelBowWave = new vpMarineFxBowWaveLevel;
m_pLevelBowWave->setName( "Bow Wave" );
```

```
m_pLevelBowWave->setShip( m_pShip );
m_pLevelBowWave->setEnable( true );
m_pLevelBowWave->configure();
```

Stern Wake



Stern wakes are generated by the cavitation of the propellers and the drag of the stern, leaving a trail which tracks the movement of the vessel. The greater the speed and power, the longer the dissipation time of the wake (specified in seconds). Stern wakes are simulated by draping a textured mesh onto the surface of an ocean. The wakes mesh conforms to the ship's motions and follows the ship's path (that is, as the ship turns, the wake curves to track the path of the vessel). Both the texture and the vertices of this mesh have an alpha component. Stern wakes grow wider and dissipate over time.

Creating a Stern Wake

After you have created your ship and defined all the ship's properties, you can now add a **vpMarine** effect to be attached to your ship. For our example, we will assume that your ship has already been defined.

Setting the Ship Object, Ocean, and Motion Strategy

Here are the steps you will perform before you apply your **vpMarine** effect. These steps are:

- Associate a ship object with your ship definition.
- Add your ship object to your scene.
- Associate your ocean with your ship definition.
- Create a new motion strategy for your ship object.

Note: Motion strategies can only be applied to **vpTransforms**.

- Set the new motion strategy as the motion strategy for your ship object, unless there is already a motion strategy present. (If so, then set the new strategy as the next motion strategy for your ship object.)
- Set your motion strategy to compute the ship's yaw automatically.

The following code fragment shows how to accomplish these steps.

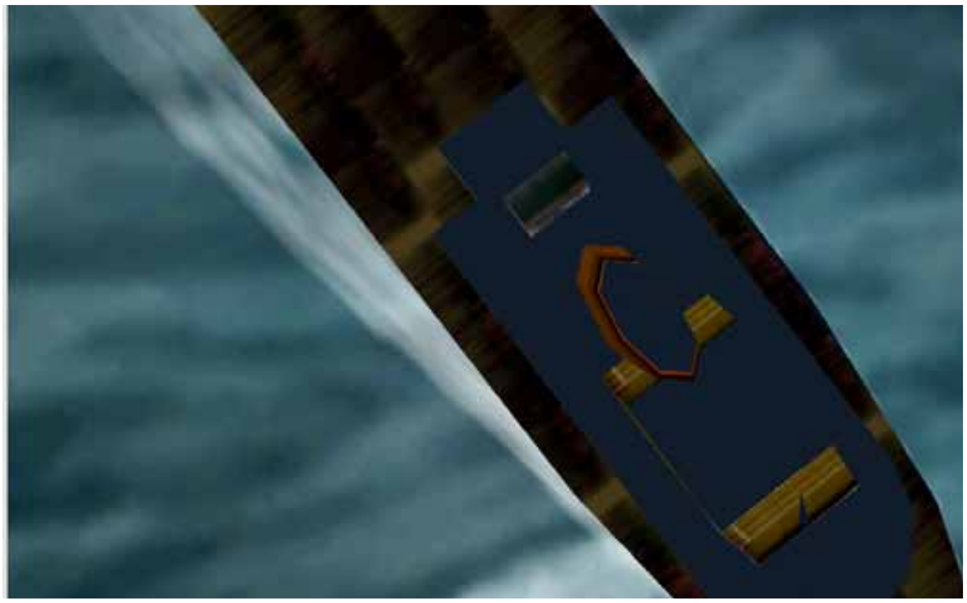
```
vpObject *m_pShipObj;
vpMarineShip *m_pShip;

m_pShip->setTransform( m_pShipObj);
m_pScene->addChild( m_pShipObj );
m_pShip->setOcean( m_pOcean );

m_pShipMotion = new vpMarineShipMotionStrategy;

if ( m_pShipObj->getStrategy() == NULL )
    m_pShipObj->setStrategy( m_pShipMotion );
else
    m_pShipObj->getStrategy()->setNextStrategy( m_pShipMotion );

m_pShipMotion->setShip( m_pShip );
m_pShipMotion->setComputeYawEnable( true );
```

Adding Your Effect

After these steps are performed, you can now set the **vpMarine** effect and the attributes of that effect. The following code shows the steps to create your stern wake.

```
m_pSternWake = new vpMarineFxSternWake;

m_pSternWake->setName( "Stern Wake" );
m_pSternWake->setShip( m_pShip );
m_pSternWake->setDissipationTime( 30.0f );
m_pSternWake->setEnable( true );
m_pSternWake->configure();
```

Hull Wake



Hull wakes are generated by the water displaced by the bow of a vessel cutting through the water, and the drag of the hull. The wake occurs when the swell of the water at the bow rises high enough that it separates from the water surface, breaks, and becomes foamy. Like stern wakes, hull wakes grow wider and dissipate over time.

Creating a Hull Wake

After you have created your ship and defined all the ship's properties, you can now add a **vpMarine** effect to be attached to your ship. For our example, we will assume that your ship has already been defined.

Setting the Ship Object, Ocean, and Motion Strategy

Here are the steps you will perform before you apply your **vpMarine** effect. These steps are:

- Associate a ship object with your ship definition.
- Add your ship object to your scene.
- Associate your ocean with your ship definition.

- Create a new motion strategy for your ship object.

Note: Motion strategies can only be applied to **vpTransforms**.

- Set the new motion strategy as the motion strategy for your ship object, unless there is already a motion strategy present. (If so, then set the new strategy as the next motion strategy for your ship object.)
- Set your motion strategy to compute the ship's yaw automatically.

The following code fragment shows how to accomplish these steps.

```
vpObject *m_pShipObj;
vpMarineShip *m_pShip;

m_pShip->setTransform( m_pShipObj);
m_pScene->addChild( m_pShipObj );
m_pShip->setOcean( m_pOcean );

m_pShipMotion = new vpMarineShipMotionStrategy;

if ( m_pShipObj->getStrategy() == NULL )
    m_pShipObj->setStrategy( m_pShipMotion );
else
    m_pShipObj->getStrategy()->setNextStrategy( m_pShipMotion );

m_pShipMotion->setShip( m_pShip );
m_pShipMotion->setComputeYawEnable( true );
```

Adding Your Effect

After these steps are performed, you can now set the **vpMarine** effect and the attributes of that effect. The following code shows the steps to create your hull wake.

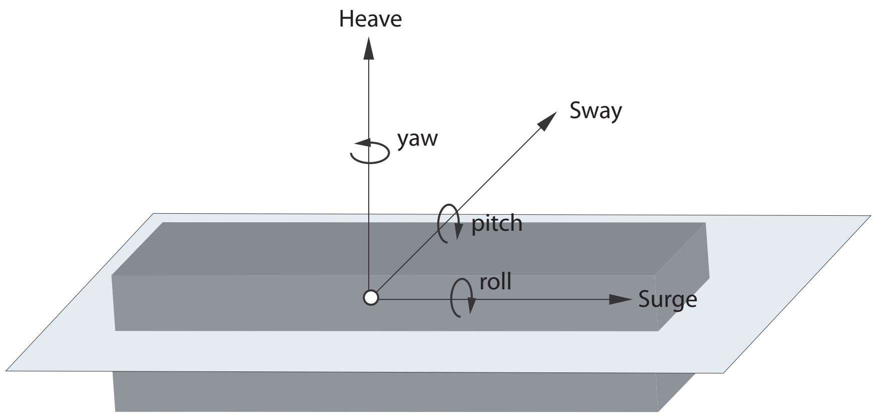
```
m_pHullWake = new vpMarineFxHullWake;

m_pHullWake->setName( "Hull Wake" );
m_pHullWake->setShip( m_pShip );
m_pHullWake->setDissipationTime( 8.0f );
m_pHullWake->setEnable( true );
m_pHullWake->configure();
```

vpMarine Effects Scaling

vpMarine effects do not respond to the scaling factors of their parent. The reason for this is that there is a one-to-one correspondence between the constructed geometry and the **vpMarineShip** definition. Thus, if you have specified that the stern width is 11 meters, and then scale your **vpObject** (SHIP) by a factor of two, then it is your responsibility to reset the stern width to 22 meters.

vpMarineShipMotionStrategy

Similar to the **vpGroundClamp**, which clamps a **vpPositionable::Strategy** to the ground, the **vpMarineShipMotionStrategy** is a **vpPositionable::Strategy** which animates the three-dimensional response (heave, pitch, and roll) between a ship's hull and the ocean waves it encounters..



Note: The motion described by the **vpMarineShipMotionStrategy** is only with regard to the ship's heave, pitch and roll. There is NO forward motion addressed by **vpMarineShipMotionStrategy**. This or any other motion must be addressed by some other seperate positioning strategy.

The **vpObject** can reference a **vpMarineShipMotionStrategy** instance by using `setStrategy()` or `setNextStrategy()`. You are responsible for setting the ship's horizontal position (X,Y) and yaw angle. The **vpMarineShipMotionStrategy** computes the correct heave, or Z-value, in addition to the pitch and roll. Optionally, you can enable the **vpMarineShipMotionStrategy** to compute the yaw angle automatically based upon the velocity in the X and Y axes.

```
void myApp::initializeShip()
{
   m_pShip = vpMarineShip::find( "ShipDef" );

   m_pShipMotionStrategy = new vpMarineShipMotionStrategy;
   m_pShipMotionStrategy->setComputeYawEnable( true );
   m_pShipMotionStrategy->setShip( m_pShip );
   if ( m_pShipObj->getStrategy() == NULL )
      m_pShipObj->setStrategy( m_pShipMotionStrategy );
   else
      m_pShipObj->getStrategy()->setNextStrategy( m_pShipMotionStrategy );

   m_pShipObj->setStrategyEnable( true );
}


void myApp::positionShip()
{
    x = .....;   y = ....;
    m_pShipObj->setTranslate( x, y, 0);
}
```

This ship position strategy uses four ocean height queries:

- Port and starboard heights to calculate the ship's roll
- Bow and stern heights to calculate the ship's pitch
- Bow and stern are averaged to produce the ship's overall heave

```
const vpTransform *getPortTransform( void ) const
const vpTransform *getStarboardTransform( void ) const
const vpTransform *getBowTransform ( void ) const
const vpTransform *getSternTransform ( void ) const
```