Houdini Ocean Toolkit for MODO 701, 801

Version 1.0

Philip Stopford - July 31, 2014

Table of Contents

Introduction

Houdini Ocean Toolkit (HOT) was developed by Drew Whitehouse, using algorithms from Jerry Tessendorf that are described in the 2004 SIGGRAPH course notes. These used to be available at http://www.tessendorf.org/reports.html, but the site is defunct. There doesn't appear to be an official resource for them, but a Google search turns up a few copies here and there.

HOT for MODO is based on a port started by Mark Wilson of www.ef9.com, but which was never completed. This version consists of three separate plugins, each with their own advantages and disadvantages. Note that, due to the way the plugins work in MODO, identical settings may not give the exact same results between plugins (e.g deformer has different operating conditions than the texture system).

Note that HOT is not an ocean shader.

This is an open source project; you can find the entire source code, including dependencies and project files at http://github.com/philstopford/HOT_MODO. It is made available under the terms of the GPL v2 license.

Requirements

HOT for MODO is built in its default configuration for MODO 701 and 801. It's available for Mac OS X and Linux. Windows support is subject to someone helping to resolve the linker issues affecting the project file. Requests for assistance in this area have gone unanswered at the time of writing.

Installation

HOT for MODO is delivered as a kit. There are two kit versions - one for MODO 701 and one for MODO 801. Install to the MODO kit folder by following the instructions below.

1. Launch MODO.
2. From MODO's 'System' menu, select 'Open Content Folder'
3. A file browser opens in the Content Folder. Open the 'Kits' folder in this location.
4. Drag and drop the 'hotmodo' folder from the archive to this location.
5. Restart MODO.

Linux - Additional Steps

Under Linux, there are some additional dependencies that are required. These should be available from your distribution's package manager.

libfftw3f (and libfftw3f_threads if it exists, but, distribution-dependending, this may be included in the libfftw3f package)

libIlmBase

For example, Ubuntu 12.04 has these as libilmbase6 and libfftw3-3.

General HOT Information

HOT works, behind the scenes, in a pretty general manner. Common parameters are discussed here. Outputs are discussed under each respective implementation because the implementation defines the available outputs and how they can be used.

Inputs

Gain : This provides a control for increasing/decreasing the displacement/deformer outputs of the HOT plugin. Raising this above 100% is not advised because you'll run into clamping.

Output Type : 0 is for displacement; 1 activates Jacobian mode (enabling spray, foam, etc. outputs in those plugins that support it)

Resolution : Supports values from 1 to 12. This is internally used as a power of 2 (e.g. a value of 12 results in 2^12 'pixels' on each side of the map used to calculate the Ocean). Larger values will take significantly longer to evaluate and require more memory. They may also give the impression that MODO has frozen. Give it time. See additional notes for the deformer with regard to this setting.

Ocean size : This has different meanings in different contexts. Refer to the specific sections for more information.

Wind Speed : This is not actually a wind, but more of a smoothing function - higher values will smooth the ocean surface.

Wind Direction : Direction of the wind, in degrees (along the surface).

Wind Align : How strongly the waves align to the wind.

Chop : This affects the profile of the wave peaks. This can also result in folding (self-intersections)

Wave height : Implementation dependent. For textures, raising this above 1.0 will result in clamping.

Shortest Wave Divisor : The larger you make this value, the more detail is added in the form of short waves. This is inverted from other implementations of HOT because MODO's user interface does not support very small values (rounding to zero).

Ocean Depth : This can be changed, but has a relatively minor effect on the ocean.

Seed : This is used to initialize the random number generator. You can use this to change the Ocean; if you find a value you like, you can use this in the same context in a different scene to get the exact same result.

Damping : Amount of attenuation applied to waves traveling against the prevailing direction.

Deformer

The deformer works against any mesh item that contains vertices. It has the benefit of providing immediate feedback in the viewport, working with the MODO deformer stack and also enabling vertex-based constraints of items (e.g. buoys, boats). It does require a dense mesh to provide convincing close-ups. You can use this with subdivision surfaces - the deformer will affect the cage.

Implementation Notes for Inputs

Output Type : In the deformer, the Jacobian setting of 1 is currently not useful. It is hoped to make it more so in a later update.

Resolution : in this mode, the final resolution of your ocean will of course depend on the number of vertices in your mesh. The resolution value only defines the granularity of the ocean map used to calculate the deformation. Increasing the ocean resolution on a low density mesh will not improve the output.

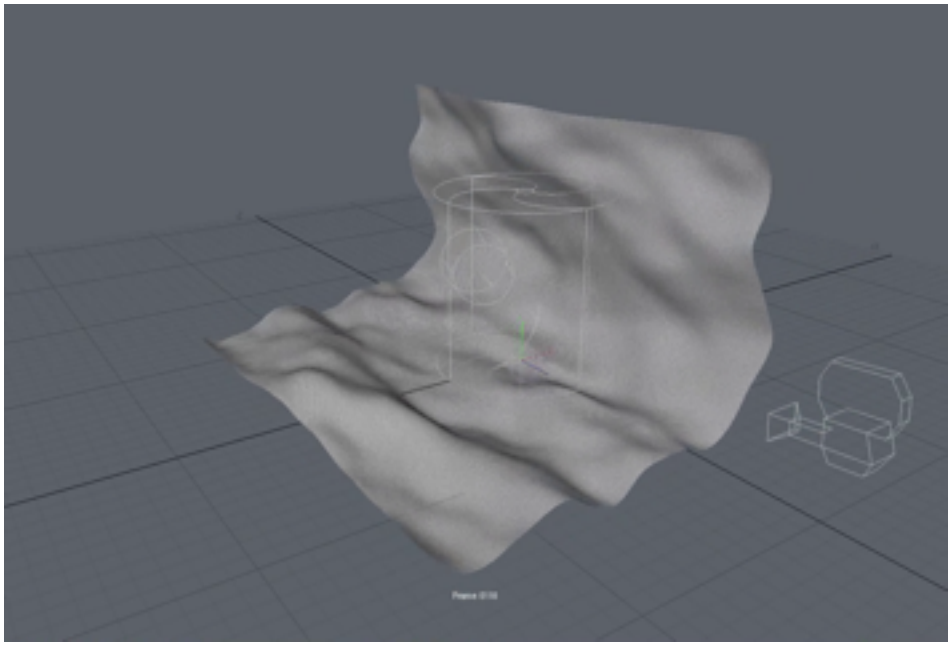
Ocean size : this defines the real world ocean size from which the vertices on your mesh are processed.

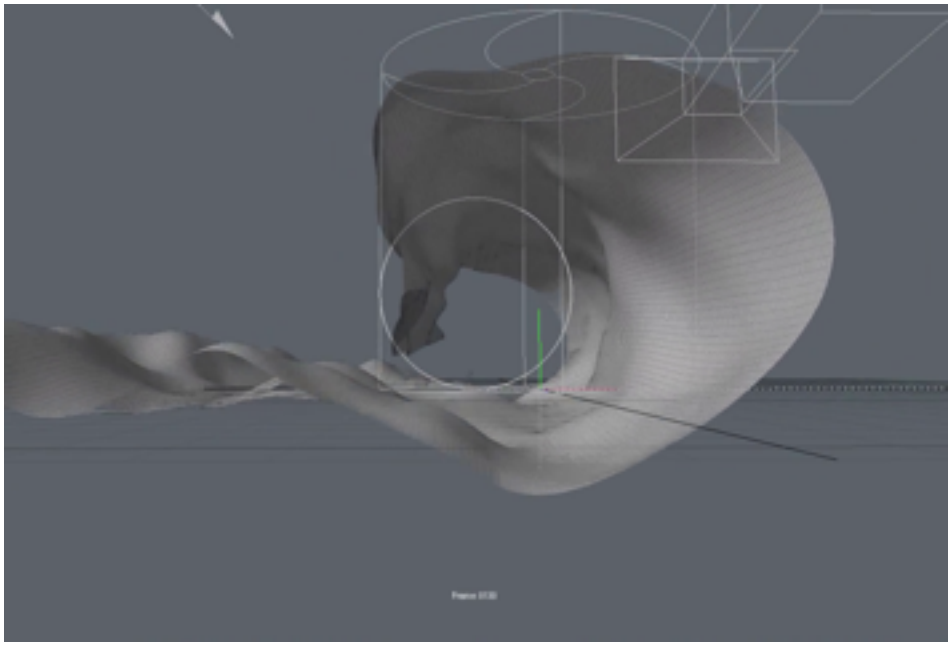
Wave height : defines the real world wave height of the deformer.

Outputs

The deformer only uses the displacement outputs of the HOT library. There is no output support for extended features (e.g. foam, spray, normals, etc.) due to architectural limitations in MODO. Whilst the interface permits a change in mode from deformer to Jacobian, the benefit of changing this setting is minimal in this implementation.

HOT for MODO allows you to take advantage of MODO's order of operations system to stack deformers. In this way, you can use a bend effector to create looming waves, a vortex effector to create whirlpools, etc.





Texture

The texture system is the fastest way of generating displacement for the ocean. You can take a simple quad poly and generate enormous oceans with compelling detail. You can use displacement or vector displacement, but note that the vector displacement can sometimes result in artifacts due to the high velocities resulting from wave folding.

HOT requires the projection type to be Solid. This is usually the default.

In Jacobian mode, you have the option of choosing the information output in the RGB channels of the color texture. For the full set of outputs, you will need to instance (or duplicate) the texture multiple times and this will incur additional memory and computation expense.

Note also that you won't see any viewport updates from this texture in displacement mode - you'll need to have Preview running. Be aware, Preview can make modo unstable in some situations, so be sure to save often in case of trouble.

Implementation Notes for Inputs

Jacobian Output mode : (**Valid in Jacobian mode**) Used to select what is output in the texture's red, green and blue channels :

- 0 : Red : Jplus; Green : Jminus; Blue : Zero
- 1 : Normal map
- 2 : Foam map
- 3 : Spray map
- 4 : Eigenminus
- 5 : Eigenplus

Ocean size : in this mode, the ocean size refers to the UVW dimensions of the surface. Note that this is affected by resolution. You can use this to force tiling of the ocean on the surface. This works in combination with the size of the texture locator for the texture. Higher values stretch the ocean over the surface, smaller values shrink it (leading to tiling)

Wave height : At 1.0, in a displacement texture, this works with material displacement height value to define the maximum wave height. For a displacement texture, setting this above 1.0 will result in clamping of the displacement. In other texture modes, this clamping will not be seen in the output.

Outputs

The texture output depends first of all on the way that the texture layer is being used in the ShaderTree.

In all modes of the texture layer, the value and alpha outputs are processed. The alpha is always set to 1.0 and the value output is always the Y displacement of the Ocean.

In color/vector modes, the texture layer output varies based on the Output Type set for the ocean texture :

In displacement mode (0), the red/green/blue channels are set to the respective displacement amounts in XYZ. In Jacobian mode (1), the output is defined by the Jacobian Output mode value.

- A note about displacement is needed here. MODO expects all displacement output from a texture to lie between 0 and 1. 0.5 represents no displacement, 0 is a full negative and 1 is full positive. This value is used with the displacement height of the material to define the actual displacement at each spot on the surface during rendering.

Channel Modifier

This is the plugin that gives you access to everything available in HOT : spray, foam, displacement, normals, etc. This comes with a cost, though. Due to architectural issues, it's much slower to evaluate than the deformer or texture, so only use it if you need the extended feature set.

Note also that you won't see any viewport updates from this plugin if you're driving displacement textures - you'll need to have Preview running. Be aware that Preview can make MODO unstable in some situations, so be sure to save often in case of trouble.

Implementation Notes for Inputs

This mode is much more general. It is generally expected that the user input the World Position (X, Z) from Shader Inputs into the 'InputX' and 'InputZ' channels of the channel modifier. This will sample across the surface and generate the values for each point.

Outputs

Note that the MODO SDK forces all inputs and outputs to be shown even when invalid for the mode of operation. Invalid outputs will be set to zero for the current mode.

Displacement : Outputs the XYZ displacement vector directly. Can be used to directly drive a vector displacement ShaderTree layer.

Normals : Calculates the surface normal that could be used to drive normal inputs on a surface, allowing for a more detailed render without the need to displace points.

Foam : (**Valid in Jacobian Mode**) This outputs a vector to indicate how the peaks of waves are folding in each of X, Y and Z.

Spray : (**Valid in Jacobian Mode**) This output is generally more useful than foam. It generates a vector to indicate spray in each of X, Y and Z.

Additional outputs are available for those interested :

Jminus

Jplus

Eigenminus

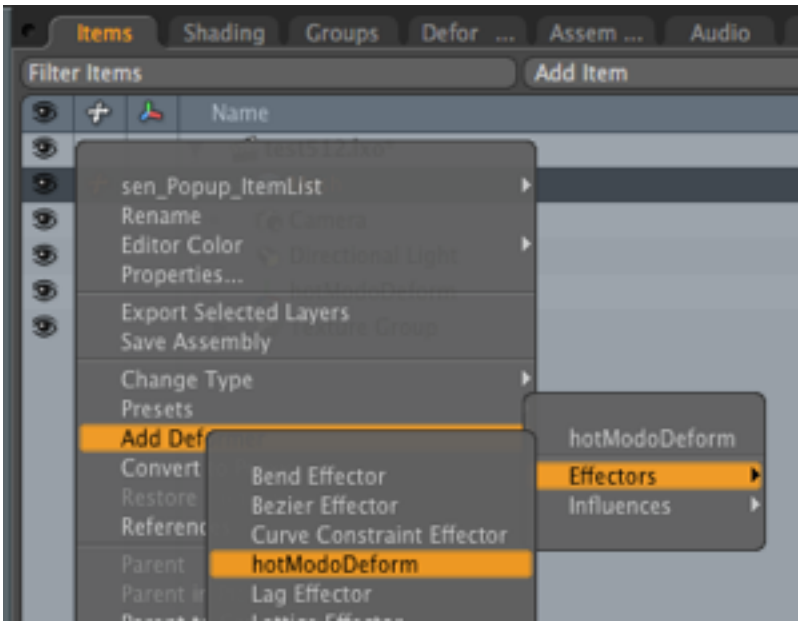
Eigenplus

Let's Get Our Feet Wet

Deformer

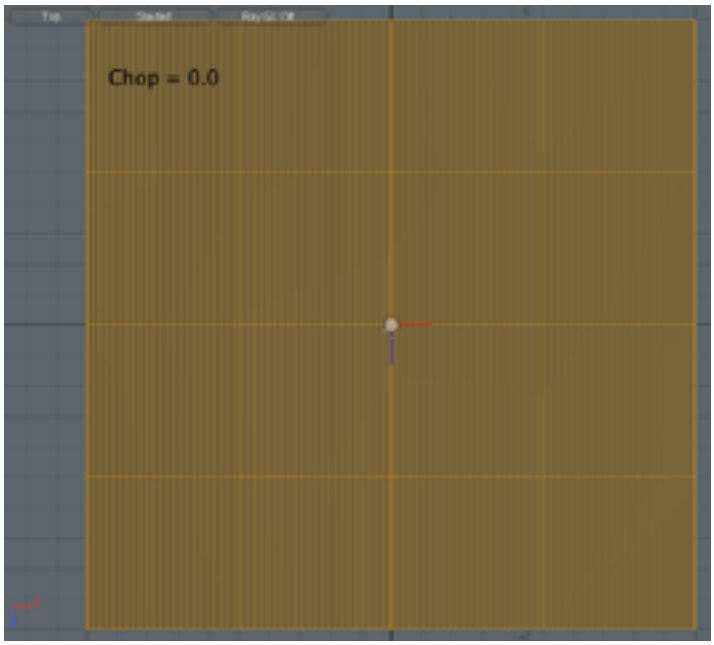
We're going to start with the deformer system because it has the advantage of displaying its results in the viewport and the ocean works based on the mesh size.

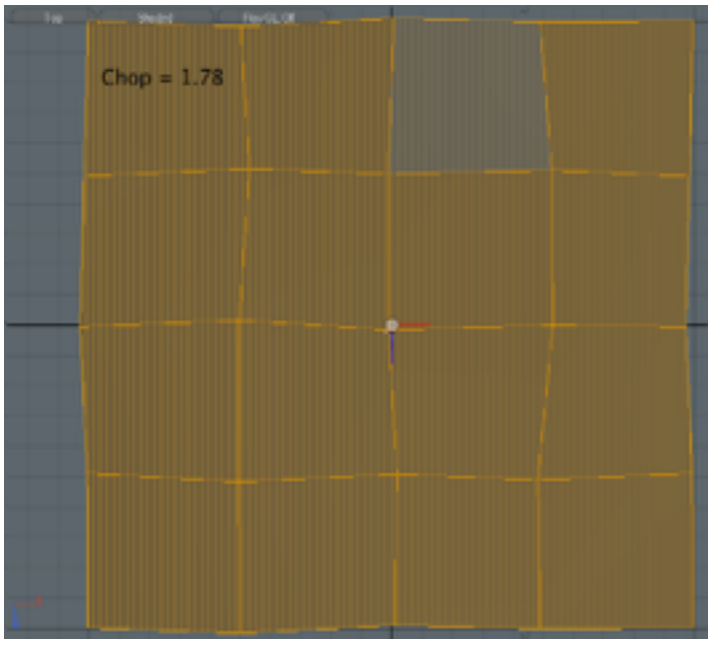
1. Add a unit plane to a mesh item. Scale it up to be 200 m on each side, centered on the world origin. If you sized the mesh item rather than the geometry itself, freeze the transforms on the mesh item.
2. Right click the mesh item in the item list and the Use the 'Add Item' drop down menu in the item list to add a hotModoDeform from the Deformers submenu.



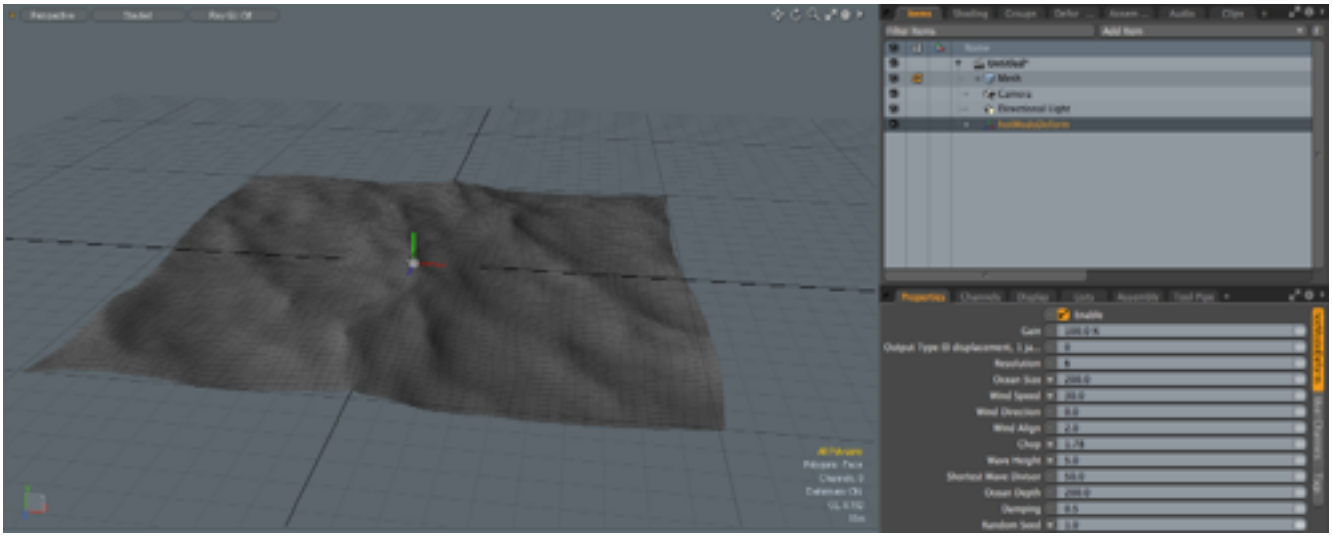
3. You should see a small shift in the viewport. It's not very convincing as an ocean, so let's subdivide the plane a few times.
4. In polygon mode, select your quad, hit Shift-D and hit return. This will turn your single quad into 4 polygons. You'll begin to see the ocean deform. Keep subdividing and increase amounts of detail will show up, but of course your mesh will become very dense.
5. Scrub the timeline to show how the ocean characteristics change over time.
6. Change the wave height to, say, 5 m and you should begin to more clearly see the ocean effect.

7. Let's quickly review the chop effect. By default, it's set to 1.78. Switch to a top view of your mesh. You'll see that the edges are all irregular. If you set chop to 0, the edges will become orthogonal. Chop, basically, adds lateral movement to the displacement of the vertices. No chop means only vertical displacement is applied to each vertex.





8. At 256 polygons, the ocean starts taking some basic shape. Let's keep subdividing to 4096 polygons. A reference is below.



9. If you scrub the timeline now, you will be able to see the wave interactions (forward and reverse).

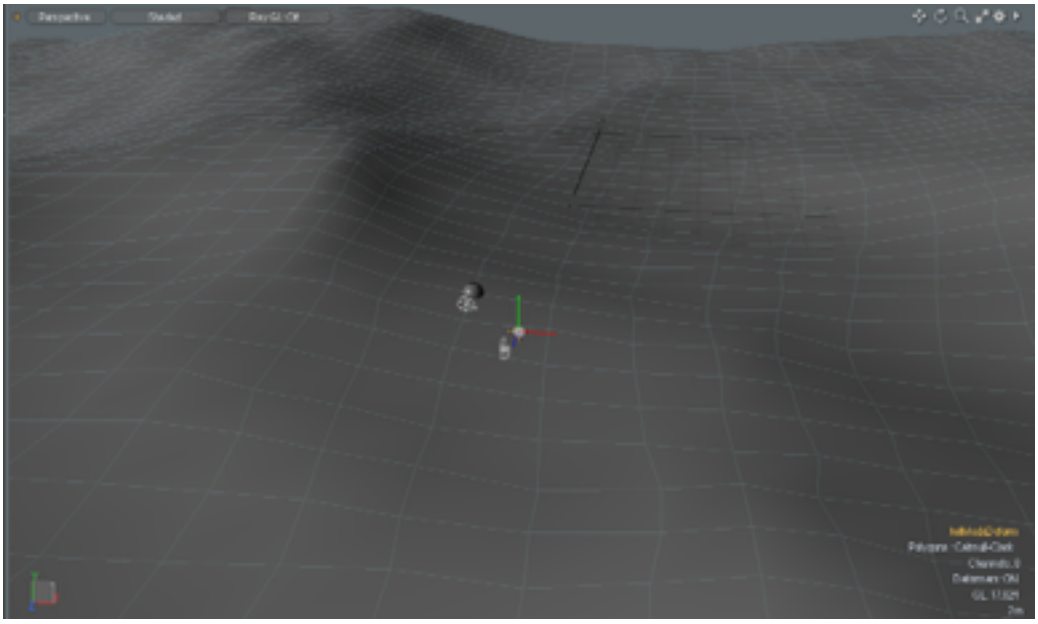
10. Let's add a point of interest. Add a unit sphere to the scene in a new mesh item.

11. Select the hotModoDeform entry in the item list and deselect 'Enable' to temporarily turn off the deformer.

12. Select the Sphere. Shift-select the ocean mesh.

13. From the 'Setup' workspace, under 'Modifiers' choose 'Vertex Constraint'. This will set up a constraint from the ball to the nearest vertex on the ocean.

14. Re-enable the hotModoDeform deformer. You should see something like this :



15. As you scrub the timeline, the ball will follow the deformed ocean. This is one of the key benefits to the deformer mode for HOT. It's very easy to set up and it's trivial to create constraints for objects.

16. Now that we have a lightly detailed ocean mesh thanks to the subdivision, let's explore how the resolution parameter affects the look and feel. Go ahead and change it from the default. Lower values evaluate extremely quickly, but lack any subtle motions that impart realism. At the higher end, impressive detail is added, but the evaluation slows down markedly.

17. Try also adjusting the ocean size. This is where you have to pay some attention to your settings. The ocean size relates directly to your ocean mesh size. For our example here, the default value of 200 matches with our geometry because of the sizing we performed in step 1. If you change the ocean size to 100, you will see tiling in the pattern because the ocean map needs to affect the entire mesh and so it has to repeat twice in both directions. If you make the ocean size 400, the ocean will appear blurred because a significant portion of the map is never applied to the mesh because it falls outside the mesh itself. In general, matching this value to your mesh size should be the first step in setting up your ocean. Detail is added using the other settings.

- Note that scaling the mesh item without freezing it will not change the effect of the deformer on the ocean mesh item.

18. Let's note in passing that the gain and height work identically in this implementation in terms of the final result. Changing wave height from 5 to 10 meters has the same effect as leaving the height alone and changing gain from 100% to 200%.

Let's take a look at the wind settings.

19. Set your ocean size back to 200.0. Set resolution at the highest value you can live with, e.g. 10. Set wave height to 7.0 and chop to 0.0 for the moment. Set wind speed to 50 and wind align to 0.0. If you scrub the timeline now, you'll see that the waves behave as though there is no wind in the scene. Gradually increase 'Wind Align' and that will change. You may well need to go above the 1.0 value. In this case, going as high as 10.0 might be needed. Remember that the wind works to push and smooth waves rather than as a genuine directional force. You can also see the effect of the Wind Direction control to steer the waves

20. Damping affects the amplitude of the waves traveling against the prevailing flow. At a setting of 0.0, the waves have the same amplitude in both directions; for a value of 1.0, there are no waves traveling against the prevailing flow. You can adjust this during playback to get an idea of the response to this control.

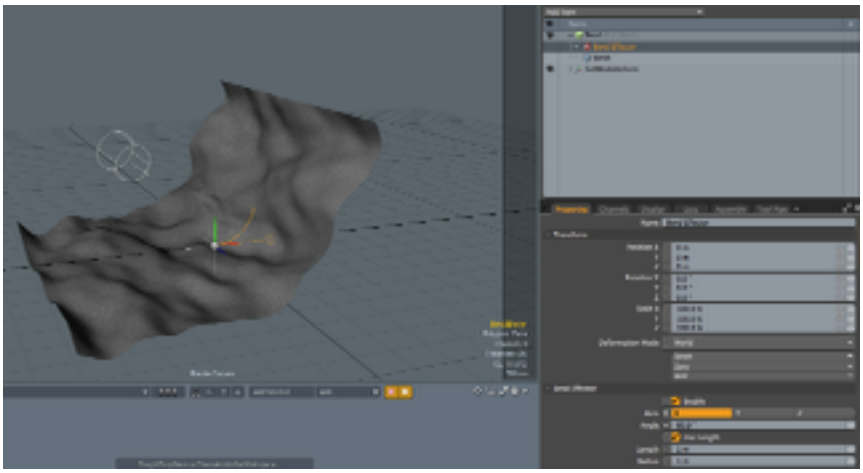
21. The seed value can be used to define the input to the ocean's internal random number generator to change the ocean look for the given set of parameters.

22. The last parameter to explore is the Shortest Wave Divisor. For this to be really noticeable, we'll need a denser mesh. Select the ocean mesh item and, in polygon mode, hit shift-D and return several times until you have 65,536 or more polygons. Now go back to the hotModoDeform item and change the shortest wave divisor value to 1.0. You'll immediately (more or less) see all of the fine detail waves vanish in the viewport. The value entered here defines the amount of short length detail on the ocean. The higher the number, the more fine detail will be added.

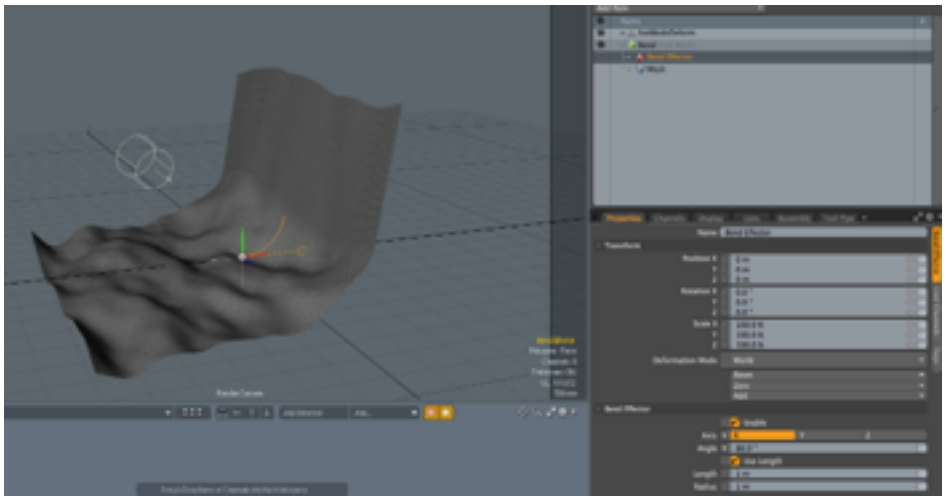
- The Shortest Wave Divisor control is more difficult to clearly see on the deformer without a sufficiently dense mesh. Its effect is far more obvious in the texture and channel modifier modes where the mesh density adapts to the camera's view of the ocean, but you'll of course only see it with Preview enabled (or by rendering a frame).

23. Now, let's explore the deformer stack options for HOT. Go back to the item list and right click on your ocean mesh. Add a bend effector from 'Add Deformer -> Effectors' from this menu. Hop over to the Deformers list. If all went well, the Bend effector will be at the top of the listing, with hotModoDeform at the bottom.

24. Let's adjust the bend settings on the Bend Effector (you can expand the Bend group in the Deformer list, or hop over to the Item List and select the Bend Effector directly). The Properties panel will then show the controls we want to adjust. Choose the X axis and increase the angle.



25. Now had back over to the Deformers list and move hotModoDeform to the top of the list. You'll see something like below. This is because HOT uses the X and Z positions of vertices to define the deformation in the Y axis.



26. You can also use falloffs with the HOT deformer. Just right click the hotModoDeform to add any falloffs that you would like to use.

Remember that you can also use Subdivision or Catmull-Clark Subdivision surfaces with the deformed mesh.

FAQ

I don't see any displacement in my render (using displacement or vector displacement). What's up?

Check that you have a material affecting the surface and ensure that the displacement height on this material is appropriately set. Due to MODO's architecture, we can't set this from the texture plugins.

My texture layers aren't working correctly. Why?

HOT needs the projection type to be set to 'Solid'. Check that that's the case. Then also check whether the displacement height is set appropriately on the material linked to your surface.

Linux : MODO won't start after installing this. Why?

This is likely the result of at least one missing dependency. On Mac, this doesn't appear to prevent MODO starting up, but Linux installs seem to be less tolerant. Check that you have the libraries installed (see Installation section, Linux : Additional Steps). Note that these libraries will need to be installed on Linux render nodes for the plugin to work there as well.

Linux : MODO renderslaves are having trouble. Why?

This is likely the result of at least one missing dependency. On Mac, this doesn't appear to prevent MODO starting up, but Linux installs seem to be less tolerant. Check that you have the libraries installed (see Installation section, Linux : Additional Steps).

Known Issues

The (vector) displacement texture can occasionally result in flat spots on large surfaces. This can be reduced by increasing the resolution value. Consider a 128x128 map spread over a large area - any pixel hitting the maximum value will affect a large portion of the surface, giving the impression of a flat spot; with more pixels (higher resolution), this spreading effect is reduced.

Vector displacement can result in artifacts on the surface due to high rate of change. This seems to be coming from HOT itself, but is under investigation. Displacement doesn't show these problems.

Development Notes

This section is purely for those interested in the code and approach taken. It is not intended for end-users; it's more of a lessons learned and MODO SDK section for developers. Mark's original source has been preserved in an archive in the repository for reference.

MODO SDK

Default is to expect MODO 801 SDK. Set the MODO701 preprocessor directive to target MODO 701 SDK.

OpenEXR

OpenEXR is a pain to build on Windows. You only need to build ilmbase for this project. There's a note to guide the undertaking in the source tree.

Mac Library Dependencies

Xcode makes it a little frustrating because it sets the library paths in the binaries to the wrong location. You need to use install_name_tool to fix the library references manually in the packaged hot modo.lx, Iex and Imath dylibs.

Windows Build

Despite significant investment of time and energy, I've been unable to resolve myriad linker related issues with the project on this platform. I have been looking for assistance with this, but no-one has come forward at the time of writing to help. As such, the 'solution' is in the source tree, but is not functional.

Linux Build

MODO is built with quite an old toolchain, GCC/G++ 4.1, due to the target of RHEL 5.4. This toolchain is generally not available on newer systems, so I opted to build the plugin on Ubuntu 10.04. The reason for this is that the package manager makes it quite easy. You need to install the following dev packages from the package manager (building these dependencies from source can be rather painful)

- libblitz-dev
- libfftw3f-dev
- libfftw3f_threads-dev
- libIlmBase-dev
- libloki-dev

With these packages installed, you will need to adjust the LXSDK location in the Makefile in the main source folder. You can then just run 'make'. The plugin will be available at Linux/build/hotmodo.lx

The resulting binary should work fine as-is on other systems as long as the library dependencies (libfftw3f, libfftw3f_threads, libilmbase) are met.

Threading

MODO uses threads on deformers in an interesting way. Up to ~512 vertices, no threading takes place. Beyond that point, threading is utilized. This doesn't appear to be documented and so it tripped me up in the early stages because I couldn't understand the cause for artifacts randomly popping up in the larger scenes I was testing.

With the help of various developers at The Foundry, the simplest solution was to use the per-thread local variables to avoid the headache that the MODO threadslot (thread local storage) approach seemed to indicate. When I originally looked at Mark's code (which only had the MODO-related source; none of the dependencies), I found this call which looked wrong. Ocean.h only has, in its original form, a method like this :

```
void eval2_xz(float x,float z)
```

where Mark's code was calling it with this, where result is float[3] :

```
cur.m_context->eval2_xz(p[0],p[1], result)
```

I assumed that this was a typo in the absence of the Ocean.h in the source archive. Of course, overloading the method in Ocean.h to set result[] then gives the per-thread protection we need. As the plugin developed, this overloading extended to allow us to retrieve per-thread values for all of the output values from HOT. This overloading extends down a few

levels in Ocean.h because some aspects are evaluated in later method calls (e.g. the Jacobian methods that provide foam, spray, etc. values)

There is a caveat here, though. Not all plugins allow the same approach. In the case of the channel modifier, there's no equivalent to the methods in the deformer and the texture system, so we end up creating the ocean per-thread. This takes a fairly long time, leading to performance headaches for that particular implementation.

Extended Information - Implementation Limitations

There are some implementation limitations for plugins in MODO that limit the ability to output the broad range of values from HOT in every mode. The deformer can only deform (and it cannot set vertex map values); the texture plugin can only output to the standard texture fields, hence the Jacobian output mode to try and make the extended features available. This is the reason for the channel modifier which is, sadly, quite slow due to the internal design of this part of MODO.

A custom material is being considered as a possibility.

Acknowledgements

- Drew Whitehouse for creating the original Houdini Ocean Tool-kit and for making it independent and easily portable to other 3D applications.
- David Ballesteros for allowing me to use aspects of his documentation from the LW port, and for the open source LW implementation of HOT that provided a useful reference regarding implementation details.
- I've also used his HOT for LW Microsoft Visual C++ project as a basis for the HOT for MODO Win64 project after hitting snags creating a new project directly.
- Mark Wilson, for starting the original port, and for sending the code on to Greg Duquesne. Greg wasn't able to work on it, but kindly let me have the source to revive the effort.
- Erwin Zwart for gently goading me into action.
- Michael Wolf (of db&w) for supporting the HOT for LW port that provided me with such a good reference.
- Greg Duquesne (of The Foundry), for initial guidance about the threading issues.
- Gwynne Reddick (of The Foundry), for help with the kit structure.
- Pete Segal (of The Foundry), for extensive and patient guidance during the effort to pull this up to date, for both the deformer and the texture work.
- Matt Cox (of The Foundry), for guidance in the channel modifier implementation.

Copyrights and Licenses

- HOT for MODO is licensed under GPLv2. It is copyright © 2014 Philip Stopford, copyright © 2012-2013 Mark Wilson.
- The Houdini Ocean Toolkit is copyright © 2005 Drew Whitehouse, ANU Supercomputer Facility. It is licensed under GPLv2 or later. (https://github.com/eloop/hot)
- MODO is copyright © 2001-2014 The Foundry Visionmongers Ltd. (http://www.thefoundry.co.uk)
- Houdini is copyright © 2014 Side Effects Software Inc. (http://www.sidefx.com)

HOT uses the following libraries :

- OpenEXR 2.1 is BSD licensed (http://www.ilm.com/opensource, http://www.openexr.org)
- FFTW is Copyright © 2003, 2007-11 Matteo Frigo, Copyright © 2003, 2007-11 Massachusetts Institute of Technology. It is licensed under GPLv2 or later. (http://fftw.org)
- Blitz++ is dual licensed under its own artistic license and GPLv2. (http://blitz.sourceforge.net/)
- Loki is licensed under the MIT license. (http://loki-lib.sourceforge.net/)