1 QUICK START GUIDE

This document will explain you how to build you own floating entity script. If you want to setup a basic boat, you can directly use the pre-built scripts.

1.1 USING PREBUILD SCRIPTS

There is two prebuilt script. To setup a boat, you need to drag & drop the script on the model you want to simulate.
The two scripts are:

FloatingEntityFlat - This script simulate body on flat water surface.
FloatingEntityStorm – This script simulate body on a simple stormy ocean.

For the Storm, you can use the included shader.

1.2 MAKING YOUR OWN SCRIPT

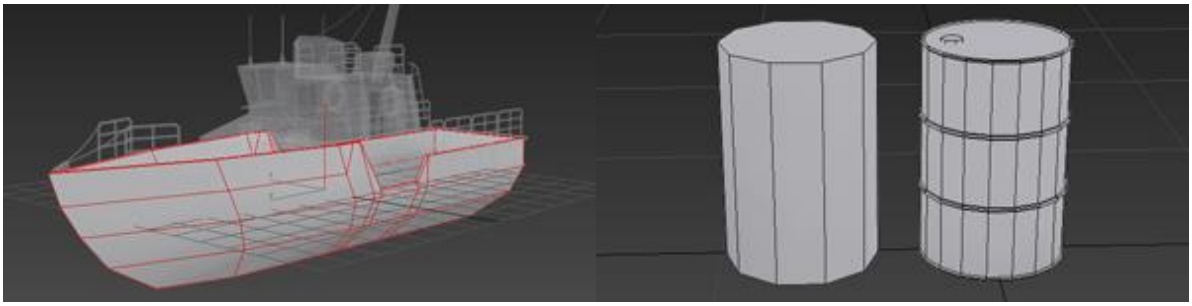
This little document should help you to setup a custom floating entity in less than 5 minutes.

1.2.1 Choose your floating entity

Take your 3D model. If this model contains more than 250 triangles, it's recommended to create another mesh, which will be used for the simulation only.

Here some arbitrary value recommended for a simple game:
Small props: 50 triangles
Small vehicle: 200 triangles
Large vehicle: 400+ triangles

So, for my example, I have two models, the NCGC VAKTA, a medium sized ship, and a metal drum.
Vakta's hull: 155 triangles.
Drum: 591 triangles -> I create a 'proxy', another mesh to simulate this drum -> 40 triangles.



1.2.2 Choose your water surface

If you have a custom river system, ocean system, that compute waves, you will have to tell this software how to get the height of the water at a position (X, -, Z).
If you only want to simulate a basic pound, or basic water surface, you can use built-in delegates.

For the boat, I will but it in a basic heavy storm, and use built-in delegate.
For the drum, I'll put it in a pool, with flat water, using a custom delegate.

Both scripts will be very similar. For the boat, the script will be:

```
using ArchimedsLab; //Include the DLL provided

//Boat script
public class FloatingGameEntity : MonoBehavior
{
    /* These 4 arrays are cache array, preventing some operations to be done each frame. */
    tri[] _triangles;
    tri[] worldBuffer;
    tri[] wetTris;
    tri[] dryTris;
    //These two variables will store the number of valid triangles in each cache arrays. They
are different from array.Length !
    uint nbrWet, nbrDry;
    Rigidbody rb; //Get the rigidbody
    Mesh m; //Mesh used for the simulation

    void Awake()
        rb = GetComponent<Rigidbody>();
        m = GetComponent<MeshFilter>().mesh;
        //Setting up the cache for the game. Here we use variables with a game-long lifetime.
        WaterCutter.CookCache(m, ref _triangles, ref worldBuffer, ref wetTris, ref dryTris);
    }

    void FixedUpdate(){

        /* It's strongly advised to call these in the FixedUpdate function to prevent some
weird behaviors */

        //This will prepare static cache, modifying vertices using rotation and position
offset.
        WaterCutter.CookMesh(transform.position, transform.rotation, ref _triangles, ref
worldBuffer);

        /*
            Now mesh ae reprensented in World position, we can split the mesh, and split tris
that are partially submerged.
            Here I use a very simple water model, already implemented in the DLL.
            You can give your own. See the example in Examples/CustomWater.
        */
        WaterCutter.SplitMesh(worldBuffer, ref wetTris, ref dryTris, out nbrWet, out nbrDry,
WaterSurface.simpleWater);
        //This function will compute the forces depending on the triangles generated before.
        Archimeds.ComputeAllForces(wetTris, dryTris, nbrWet, nbrDry, rb.velocity, rb);
    }
}
```

And for the drum, the script will be:

```
using ArchimedsLab; //Include the DLL provided

//Drum script
public class FloatingGameEntity : MonoBehavior
{
    tri[] _triangles;
    tri[] worldBuffer;
    tri[] wetTris;
    tri[] dryTris;
    uint nbrWet, nbrDry;
    Rigidbody rb;

    public Mesh m; //Mesh used for the simulation, set to our proxy in the editor

    //I create my own delegate. Here water will be flat, at Y=42.0
    WaterSurface.GetWaterHeight myWaterHeight = delegate(Vector3 a)
    {
        return 42F;
    };

    void Awake()
        rb = GetComponent<Rigidbody>();

        //Here, we send m, the mesh setup in the editor, our proxy.
        WaterCutter.CookCache(m, ref _triangles, ref worldBuffer, ref wetTris, ref dryTris);
    }

    void FixedUpdate(){
        WaterCutter.CookMesh(transform.position, transform.rotation, ref _triangles, ref worldBuffer);
        WaterCutter.SplitMesh(worldBuffer, ref wetTris, ref dryTris, out nbrWet, out nbrDry,
                              myWaterHeight); //And here, I send this new delegate.

        Archimeds.ComputeAllForces(wetTris, dryTris, nbrWet, nbrDry, rb.velocity, rb);
    }
}
```

1.3 NOTES

Your delegates MUST be fast. They are called several time per triangles, so think about that! It can be useful to have two water models: a complex one, using Gerstner waves and so, for the shader, and a simple one, just simulating the global shape of the sea, for the CPU.

This plugin has been made with a target of < 1ms / boat (~500 triangles). So if your delegate makes it greater than 5ms / boat, it may be time to optimize!

To simulate boat load, and have a realist boat, think about tweaking the center of mass. Unity doesn't put it every time in the perfect place.

1.4 FUTURE

Working on water occlusion system, and built-in payload simulation (cargo, empty/full drum, submarine). If a good performance level is reached, I will implement it in the next version.