What is buoyancy and why do I want it?
Buoyancy is the principle that explains why objects float and rise to the surface of water. If your game has any liquid surfaces (like water), you might consider trying out some buoyancy to make things a little more life like and immersive (this article will contain a splashing of buoyancy and water related puns - with no apologies).

Buoyancy is pretty simple and yields some good looking results! This physics concept has been used in many games, a good example being the Wave Race series. Now you can harness its power as well!

We're going to explore implementing buoyancy with any (or even no) physics API. We'll be making a number of simplifications and assumptions for our model. Because this is for a game, no one cares if it is scientifically accurate, all we care about is if it looks good. First we'll cover the forces involved and how to calculate them, then we'll do a quick run through the numbers, and finally we'll go over some sample source code.

What should I already know?
We assume that the reader is familiar with basic physics. This means you should know and understand the concepts of acceleration, velocity, and force.

Our Model of Buoyancy
To simplify things, we're going to ignore things like turbulence, Archimedes principle, wind resistance (if our object was above the fluid), and we are also only considering one axis of movement. What we are left with is a resolution of a few pretty simple forces. To lay the ground work for the coming math, imagine a ping pong ball immersed in water at a given depth. It has these forces acting upon it:

\[ F_g = \text{Force of Gravity} \] pulling down on the ball
**Fb** = **Force of Buoyancy** pushing up on the ball

**Fd** = **Force of Drag** from the water it must push through if it is moving up

We can see from the diagram that if the force of buoyancy is greater than the force of gravity and the force of drag, the ball will move up. Note that the drag force depends on the velocity of ball - the faster it's moving up, the more drag force will be acting on it.

To model the behavior of this ball or other buoyant objects, we're going to take these forces, and sum them up. That final, or net force, is the one we'll apply to the object. This net force will need to be updated and calculated each frame. We'll follow up with some physics api agnostic source code snippets that should help things sink in. Now let's dive in depth for each of our three forces.

**Force of Gravity**
This one is simple and one you probably already know how to calculate. The force of gravity is merely the mass of the ball multiplied by -9.8m/s^2 (the acceleration of gravity), which we will call 'g'.

\[
F_g = \text{mass} \times g
\]

**Force of Buoyancy**
This is a force you may not be familiar with, but it pretty much means there is a buoyancy force acting on the object that is equal to the weight of the fluid the object is displacing. So the bigger object you have, the more fluid its going to displace, and the larger the buoyancy force is going to be. Note that an object that is less dense than the fluid is guaranteed to float! But objects made of a more dense material can also float, provided the volume of fluid they are displacing weighs more than they do.

The force of buoyancy is calculated by the taking the fluid's density, the volume of fluid the object is displacing, and the negated acceleration of gravity, and multiplying them together. The equation looks like this:

\[
F_b = \text{fluidDensity} \times \text{displacedVolume} \times -g
\]

Finding the fluid's density is easy, it's just whatever you want it to be. For our fluid, we'll just say it has a density of 2. The volume is also easy, for a our ball, it's \((4/3)\pi \times r^3\). For more complicated shapes, just approximate the volume with a simpler shape.

**Force of Drag**
This one is slightly more complicated but not too bad. This force represents the water our ball will have to push through as it rises. This is a type of frictional force, very much like wind resistance, it depends on how fast the ball is moving. Also, this force always acts against the motion of the ball. If the ball goes up, this force acts down. If the ball goes down, this force acts up (this is covered again in the Code Overview, later on). The equation looks like this:

\[
F_d = \frac{-1}{2} \times \text{fluidDensity} \times (\text{velocity}^2) \times (\text{area}) \times \text{dragCoeff}
\]

We already know fluid density and should already know the velocity of our object, and the drag coefficient you can just make up - it really depends on the surface of the object (play with values between 0.1 and 1.1). The area refers to surface area of the object that pushes against the
fluid. This can sometimes be had to calculate, depending on your shape, and we can get by with approximations. For our purposes, cross sectional areas work well, so for a box shape, just use the width and depth of the box. For our ball, we'll just do a cross section of the sphere – a circle.

The Net Force
Now that we've got all those forces, we just add them together to get the final force acting on the object:

\[ F_{\text{net}} = F_g + F_b + F_d. \]

If you have an object whose density is less dense than the fluid, the buoyancy force is going to be greater than the force of gravity and drag, and will rise. If you have a boat made of some material more dense than water, just remember that the the volume of water its displacing must weigh more than it.

A Quick Run Through of the Numbers
If you're here for completeness of understanding, it might be useful to run through these equations with some real numbers, see what happens to our ball in this scenario. However, if you're in a hurry, just skip this section and go straight to the code overview. Let's refer to our previous picture, and assign some values. We'll assume the following:
- **Fluid Density** = 2
- **Ball Density** = 1
- **Ball Volume** = \((4/3)\pi \cdot (r \cdot 3) = (4/3) \cdot \pi \cdot (1 \cdot 3) = 4.188\)
- **Ball Mass** = volume \times density = 4.188

So first we calculate the **Force of Gravity**

\[ F_g = \text{mass} \cdot g \]
\[ = 4.188 \cdot -9.8 \]
\[ = -41 \text{ N} \]

This means gravity is pulling down on the ball with a force of 41 newtons

Next we'll calculate the **Force of Buoyancy**. In this calculation, since our ball is completely submerged, the displaced volume is equal to the volume of the ball.

\[ F_b = \text{fluid density} \times \text{displaced volume} \times -g \]
\[ = 2 \times 4.188 \times 9.8 \]
\[ = 82 \text{ N} \]

We've got a 82 N buoyancy force pushing up on the object

Now let's get into **Force of Drag**. At the moment of our calculation, we've just released the ball, so it's going to have a velocity of zero, making this whole force zero. We're going into it here because in the code section the velocity is not always going to be zero. Also, we're giving the ball a small drag coefficient. Finally, the area is just the area of a cross sectional circle of the ball (\(\pi \cdot r \cdot 2\)).
\[
\text{Fd} = (-1/2) \times \text{fluidDensity} \times (\text{velocity}^2) \times \text{(area)} \times \text{dragCoeff} \\
= (-1/2) \times 2 \times (0^2) \times (\pi \times 1^2) \times 0.1 \\
= 0 \text{ N}
\]

So at this moment, we have no drag force, because the ball is not yet moving. However, the next time we run these calculations, the velocity won't be 0 and we will have a drag force.

So we've got our 3 forces, let's sum them into the **Net Force**

\[
\text{Fnet} = \text{Fg} + \text{Fb} + \text{Fd} \\
= -41 \text{N} + 82 \text{N} + 0 \text{N} \\
= 41 \text{N}
\]

So our final net force is 41 newtons in the upward direction! The final force is positive, working against gravity. This means our ball will rise, which is great, because that's what we'd expect. If we pumped up the ball's density, our results might not be the same.

**Code Overview**

Let's take our new found physics understanding and put it to use in some sample code. Just like in the above example, we'll calculate each force individually, then sum them to get the final net force. There are a few extra steps we need to take in code, since we are running these calculations when the ball is fully submerged, partially submerged, and perhaps not submerged at all.

For this code example, we're going to assume a box shape instead of a sphere. This box shape will approximate the shape of a boat:

```cpp
//This would take place inside an Update(float delta) type function, where delta is the amount of time passed since the last frame:
float boatDensity = 0.65f;
float fluidDensity = 1;

//figure out the dimensions of our boat
BoundingBox boatBox = renderable.BBox;
float width = (boatBox.Max.X - boatBox.Min.X);
float depth = (boatBox.Max.Z - boatBox.Min.Z);
float height = (boatBox.Max.Y - boatBox.Min.Y);

// now we must find height of our water relative to the location of our object
// this is easy if your water doesn't move, the water height is just the height
// of your water plane in world space
float waterHeight = water.GetAvgHeight(this.Position, width, depth);

//now we calculate how much water our boat is displacing
float displacedVolume = width * depth * MoreMath.Clamp(waterHeight - (position.Y - height * 0.5f),0,height);

//calculate the boat's mass - we are only using part of the volume
// as we assume the rest is filled with air
```
boatMass = boatDensity * (width * depth * height * 0.75f);

//Now lets calculate the gravitational force
// Fg = mass  * g
float gravForce = boatMass * -9.8f;

//And Buoyancy!
// Fb = displacedVolume * fluidDensity * -g
float buoyancyForce = displacedVolume * fluidDensity * 9.8f;

//And finally drag force, but with some special considerations
//We only apply drag force if the boat is in the water
float dragforce = 0;
if (position.Y < waterHeight)
{
    // Fd = (1/2) * v^2 * fluidDensity * area * dragCoefficient
    dragforce = -0.5f * (yVel * yVel) * fluidDensity * (width * depth) * 1.33f;
    //we also want the drag force to operate in the opposite direction of the movement:
    // - if the object is sinking, it has to push through water as it sinks
    // - if the object is rising, it has to push through water going up
    if (yVel < 0)
        dragforce *= -1;
}

//just sum them up!
float netForce = gravForce + buoyancyForce + dragforce;

//So we’ve got a net force, now what?

//If you were using Havok or some other physics API, you would just apply it
//to your rigid body.

//If not, you need to extract acceleration from the force,
//use that to affect velocity, and then velocity to affect position!
//
// Force = mass * acceleration
// acceleration = (Force / mass)
float acel = (netForce / boatMass);

//apply the acceleration to the velocity
yVel += acel * delta;
//apply that to the position
position.Y += yVel * delta;

//wooo! that’s it!
Wrapping up
Play around with object mass, density, and the surface area constant, see what looks best. When you've got things looking good, you can try some advanced water rendering or simple fluid simulation to complete the effect, and the effect can be quite cool. The screenshot and video below use the exact principles and code presented above for the buoyant movement of the ships.

See the video here: [http://www.youtube.com/watch?v=vWEX0uZSDC0](http://www.youtube.com/watch?v=vWEX0uZSDC0)

For any questions or comments, improvements or criticisms, send them to pete.angst@gmail.com. Good luck and have fun!

References: