Simulating Air Bubbles for Computer Generated Fluids

Joel Green
National Centre for Computer Animation
Bournemouth University, UK

Abstract

Most of the available literature on fluid simulations has not included air bubbles. This paper presents a technique to add air bubbles to a pre-computed or dynamically calculated fluid simulation. Bubbles form naturally in splashing water so the omission of bubbles detracts from the believability and visual realism of a fluid simulation. In this paper I propose a particle-based system that can be implemented with any particle-based simulation of a fluid. The bubble particles are only created where needed and their movement is then dynamically simulated according to the flow of the fluid into which they are introduced.

1.0 Introduction

Fluid simulations have become a major part of the computer graphics industry, with higher expectations on physical as well as visual realism.

Although the realistic simulation of fluids has been a major topic of research in recent years, investigation into simulating the air bubbles that get trapped in these fluids has received considerably less attention. The majority of previous work in the field of fluid dynamics has concentrated on the simulation of the fluid itself using the Navier-Stokes equations and very few have included any work regarding bubbles.

Fluid simulations are growing more advanced and the realism is approaching very high levels. However, many fluid simulations still lack the inclusion of trapped air bubbles which can be quite detrimental to the overall effect. The addition of bubbles to a fluid simulation can significantly enhance the visual realism.

This paper presents a number of different approaches to create air bubbles in a fluid system. One of the main examples used in the literature is that of water pouring into a glass. The practical assessment of this bubble system concentrates on this example of pouring water into a glass but the system is flexible enough to work in any situation involving splashing water. The ultimate aim was to produce a system that could automatically add the creation of bubbles into a fluid simulation within Alias Maya.

1.1 Previous Work

There exists a substantial body of work on fluid simulation and only a small amount on bubbles. Unfortunately most of the work with fluid simulations concentrates solely on producing a realistic looking fluid motion and tends to ignore the considerable contribution that air bubbles can make towards a convincing model of water. Most fluid dynamics make use of the equations developed by Claude Navier in 1822 and George Stokes in 1845 (Navier-Stokes equations) [FOS96]. A lot of work has been done over the past 50 years to implement these equations using computers and fluid movement can now be replicated to a high degree of physical accuracy. Foster *et al.* [FOS96] introduced some of the new approaches to using the Navier-Stokes equations for the modelling of fluid simulations giving greater control and subtlety by coupling the velocity and pressure forces more tightly.

There has been much less research concerning bubbles and how they interact with fluids. Similar areas of research have received some study such as the representation of the geometry of bubbles [GLA90] but there is little work concentrating on the interaction between bubbles and fluids. Greenwood *et al.* [GRE04] created a relatively simple method featuring passive bubbles that are dynamically generated and then simulated using the Eulerian velocity taken from the fluid. The bubbles are simply represented by spheres and do not have the ability to merge or split as real bubbles do. Hong *et al.* [HON03] make use of the volume of fluid method (VOF) to simulate individual bubbles. These bubbles can merge and deform depending on the surrounding forces and their own surface tension. However this detailed simulation is only used for the larger bubbles, spherical particles are used for the smaller bubbles which are advected in a similar way to those in Greenwood. Müller *et al.* [MÜL05] recently published their work which uses multiple fluids to achieve a full simulation of both the water and the air. The fluid-fluid interactions are simulated using particle systems. Air particles are dynamically spawned where pockets of air are likely to be trapped by the surrounding fluid, avoiding the need for a full volume of air. This approach features full interactive simulation between the fluid and bubbles. Müller's work is probably the most complete work to date featuring some clever techniques and impressive results, building on the previous work to refine and improve techniques. Despite the quality of the end result, again the methods do not have too much scientific basis. Approximate equations and models are used throughout with little reference to the real physical nature of the phenomena.

1.2 Simulating Air Bubbles In Maya

The system presented here is aimed to be a more immediate tool that, unlike some of the previous systems, can be used to add bubbles to a pre-computed fluid simulation. The system is designed to work with the popular 3D software, Alias Maya and offers an efficient way to improve the aesthetic of fluid simulations with the addition of bubbles. The bubble system is approached completely separately from the water simulation which allows the water simulation to be finalised and the bubbles added at a later stage. This could be advantageous as it negates the need to re-calculate the water in order to change the bubble motion.

2.0 Method Overview

This system is not designed to produce a simulation of the fluid itself, but instead works from a dynamic or cached particle system representing the flow of the fluid. It is not a physically accurate simulation of the bubbles; the focus was put on the visual result as opposed to using detailed mathematical models which are potentially slow to calculate.

A clean way of solving the problem of bubble simulation would be to simulate the air as a completely separate fluid system and have it interact with the water fluid system [MÜL05]. To create such a system for the air would require a great number of particles which would be unworkably slow. Instead here, to speed up the bubble simulation, bubble particles are generated on the fly only in locations where the air is likely to be trapped by the water and therefore where the bubbles need to be created.

The bubbles in this model have no effect on the fluid simulation as this did not seem to be a necessary inclusion. The mass of air bubbles is so small in comparison to the water that the effect they would have on the water flow would be inconsequential. An advantage of this approach is that the bubbles can be added subsequently to the water being simulated. Therefore, the water simulation may be finalised to get the desired flow without having to be recalculated if the settings for the bubbles are incorrect.

The bubble system is fundamentally a MEL (Maya Embedded Language) expression that is executed for each frame and calculates the positions and movements of the individual bubble particles. The motion and shape of the pouring water are represented by a different particle system. Bubble particles are emitted into this separate particle system at positions where it seems likely that air will be trapped by the water. These bubble particles are then assigned velocity values depending on the velocities of the surrounding water particles. In this way the bubbles are transported through the fluid in a realistic manner. A buoyancy value is also added to the upward y-velocities of the bubble particles in order to recreate the phenomenon of the rising bubbles. In order to simplify the process of analysing the positions of the different particles and to help averaging various values, it was necessary to use a grid that was able to divide the total volume into smaller local areas that could be scrutinized separately.

2.1 Realflow/Fluid Grids

The fluid simulations for the water in this project were created using a demo version of the program Realflow [NEX06]. Realflow creates fluid simulations of superior realism compared to that achievable in Maya. After creating the water simulations in Realflow the particle system could be imported directly into Maya and integrated with the bubble system. Using Realflow to produce the water simulations makes them more accurate and saves time as it considerably simplifies the process. Attempting to create a fluid simulation of a similar standard without the help of a program such as Realflow would have been beyond the scope of this project.

One of the main features allowing the bubble system to work was the introduction of Maya fluid grids. It was important to divide the volume into smaller, more manageable sections and fluid grids offered a convenient way of doing this. It is possible to query positions in relation to a fluid grid voxel and each voxel can be assigned individual values such as velocity and density. This means it is possible to average the velocities of a number of particles within a defined area. These averaged values can then be assigned to the bubble particles dependent on their position.

2.2 Bubble Creation

Bubbles are formed in a fluid when a pocket of air is enclosed by the liquid. This air is trapped inside the water and forms bubbles which rise to the surface where they burst.

The method used for the creation of the bubble particles was inspired by the work of Muller *et al.* [MÜL05]. Bubble particles are created according to the formation of the water particles. The bubble particles are needed in positions where air is likely to be engulfed inside the water. In order to find these positions, the fluid grid is used to analyse the positions of water particles compared to surrounding empty areas. By calculating whether the individual voxels of the fluid grid are empty or contain water particles, the fluid grid can then be assigned density, effectively generating a low resolution grid of ones and zeros set dependent on the presence of water particles.

The solution found to offer the most effective results for the creation of new bubble particles was quite simple. Obviously it is necessary to ensure the bubbles are not created in random locations or actually inside the water. If a voxel is completely empty of both water and bubble particles then it becomes a candidate to have a bubble particle created in its position. The

voxel directly above is then checked and if there are any water particles present there, the voxel is deemed to be a good location to create a bubble particle.

Once a suitable position is located for a bubble particle a new particle is created in the bubble particle system at this position and then initialised with a velocity depending on the surrounding fluid's velocity (see below).

The sizes of the bubbles are randomly set on creation. Maximum and minimum values can easily be tweaked to ensure the bubbles are of the right size for different types and scales of fluid simulation. Although this is obviously not an accurate way of replicating the variation in sizes between the bubbles it looks perfectly adequate and a system that calculates more genuine sizes for the bubbles would involve merging between bubbles. Including a deviation between the sizes of the bubbles significantly adds to the aesthetic quality of the simulations.

2.3 Deleting Bubbles

As a precaution against rogue bubble particles being created, a function has been included that removes erroneously placed bubble particles. After the bubbles have been simulated on each frame, part of the expression is executed checking for any escaped particles. If a bubble is out of bounds or gets isolated and is not within a certain distance of any water particles, its lifespan is set to zero and it is deleted. Each bubble particle is checked to see if there are any water particles in its general vicinity to get a rough idea of its location compared to the water. If the surrounding area is empty, it can be assumed that the particle has got removed from the rest of the simulation, is no longer contributing and can therefore be deleted. There are measures in place to attempt to avoid isolated particles but due to the nature of the simulation their occurrence is unavoidable in certain cases.

2.4 Simulate Bubble Motion

Numerous reasons affected the decision that the bubbles should not have an effect on the water's simulation. Ensuring the bubble system works with pre-computed fluids was an important feature of this project and also this level of interactivity between the water and bubbles would significantly slow it down. This allows the bubble particles to have their motion controlled directly by the water simulation. The bubbles' velocity simply corresponds to the average velocity of the surrounding water. The fluid grid came in useful again for this process as it gave convenient divisions within the 3d space that could be individually analysed.

The water particles each have their own velocity, calculated during the initial Navier-Stokes fluid simulation in Realflow. The fluid grid enables the system to find local particles within a set volume. The average velocity of all of the water particles within a single voxel is calculated and then that value is assigned to the velocity for all of the bubble particles located within that voxel.

If a bubble particle is positioned inside a voxel that contains water particles, the velocity from that voxel is transferred directly to the bubble particle. However if the bubble particle is in a voxel empty of water particles then the velocities from the surrounding voxels with water in them are averaged and that averaged value is assigned to be the velocity of that bubble particle. In this way the bubble particles essentially follow the movement of the fluid, being carried along by the water's motion as would happen in real life.

Another factor that had to be considered for the bubble's motion was the buoyancy of air in water. Due to the difference in density between water and air (water is roughly 1000 times denser than air), pockets of trapped air will always rise to the surface of water. The buoyant forces on the bubbles have been recreated simply by adding an extra value to the y-velocity. This added velocity causes the bubbles to rise to the surface as they would in real life. The speed at which they rise is also affected by the size of the air bubbles. Larger bubbles rise to the surface faster in this system due to the radius of the bubbles being included in the equation that dictates how much to add onto the y-velocity. Due to the difficulty of using an automatically computed process, the majority of previous work on bubbles also uses an artificial buoyancy force to make the bubbles rise to the surface.

2.5 Bubbles at Surface

In real life an air bubble will pop if its surface gets too thin and breaks. This issue is attempted with limited success by Hong et al. [HON03] but is quite complicated and does not integrate itself well with the approach taken here. An estimation of this phenomenon is included by simply bursting bubbles at random. It is not possible for bubbles to burst when surrounded by fluid so only bubbles that have reached the fluid's surface are considered for removal. Another factor to be considered here is the size of the bubble as larger bubbles have larger surface areas and are therefore more likely to pop than smaller bubbles. Each frame, any bubbles that appear to be at the surface of the fluid risk being removed with those of a larger size having a higher chance.

2.6 Converting Process to 3D

The initial work on this project was carried out using a two-dimensional simulation as this makes it clearer to see what is happening and the extent to which the bubble simulation is effective. It made the process easier to get working to begin with and simpler for error checking.

In order to convert the system to work for three-dimensional fluid simulations the information in the z-axis had to be considered. This process was generally a case of adding variables to store data relating to the extra dimension. The main rework was needed to allow the velocities of the fluid grid to be stored in a three-dimensional array. These values had to be stored when averaging the velocity within a certain area. Unfortunately MEL is unable to handle multi-dimensional arrays so it was necessary to store all the data in a one-dimensional array (see below).

This task turned out to be more taxing than anticipated. Although the general logic appears to be correct a lot of time had to be spent at the end trying to clear up a few glitches.

2.7 Other areas

One of the main problems that emerged during development was to ensure that the bubble particles reacted appropriately to the edges of the container and did not go out of bounds. A number of failsafe checks were introduced in order to avoid this problem. There was no way to simply dictate the boundaries and let the bubble particles automatically react to them as they are not dynamically simulated in that way. One technique used to overcome this was to check the estimated position of each bubble particle on the next frame and to stop it moving outside the boundaries.

Another tricky part of the programming was utilising the limited scope of MEL effectively. The entire task of programming the system was further complicated by the poor error handling and erratic behaviour exerted by Maya when executing the expression. One task that was harder to accomplish due to MEL was the averaging of the water particles velocities within a certain fluid grid voxel. Storing values in a multi-dimensional array is not possible in MEL so it was necessary to create an index that could take the x, y and z coordinates for a voxel in the fluid grid and calculate which entry this related to in a one-dimensional array.

3.0 Results

In order to better understand the movement and visual impact of air bubbles in pouring water, some video reference was taken (Figure 1). A container was built that was quite flat which made it much easier to discern what was happening when the water was poured. These tests show how prominent the bubbles are within the water motion and how they provide a visual representation of the waters flow.

When comparing the bubble simulations (Figure 2) against the filmed footage of real bubbles (Figure 1) this system proves itself to be a good approximation. This is particularly true when comparing it with the two-dimensional versions as it is clearer what is happening (Figure 3). The real bubbles move so fast that the warping and merging that is taking place is almost unnoticeable making its omission form this system less significant.

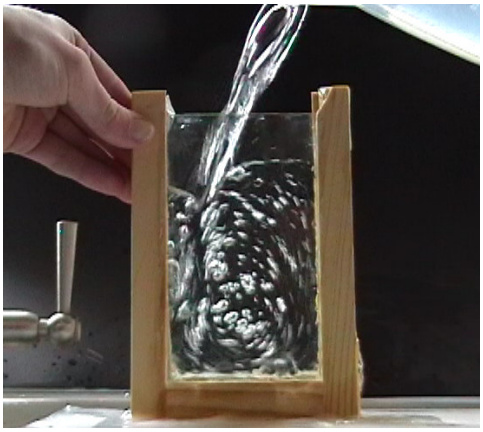

Figure 1: ***Still frame from a reference video taken to study the bubbles motion. The Container was built to give a better representation of the motion in a 2d simulation.***

The results with the two-dimensional simulation turned out very well but there were a few issues with the three-dimensional simulations. The final simulations sometimes suffer from small defects where bubbles get stuck or clump together.

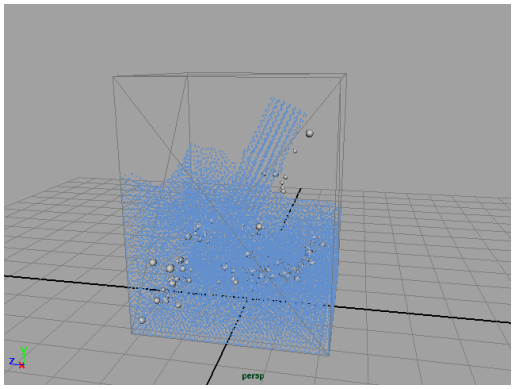

Figure 2: A 3d example of the bubble system automatically creating bubbles to a pre-computed fluid simulation.

In the two-dimensional simulations roughly 3500 particles were used in the water simulations and about 750 bubble particles were created to imitate the trapped air bubbles. The three-dimensional system made use of 25000 particles for the water and created around 5500 bubble particles. For the two-dimensional simulations the fluid grid was 35 by 35 voxels and for the three-dimensional simulations it was 25 by 25 by 25.

4.0 Conclusions and Future Work

The simulations resulting from this system validate the theory that adding air bubbles enhances the visual realism and aesthetic appeal of a water simulation. These air bubbles are naturally created in splashing

water so their inclusion in computer generated approximations seems an obvious choice that is often overlooked.

The bubble simulations that can be created using this system have various imperfections. This work is a simplification of a very complicated problem which requires a great deal of time and complex maths to solve realistically. Despite the fact that my method is less physically precise than some of the previously completed works on the subject the end product is still successful. The bubbles flow with the fluid in a realistic manner and they do appear to be emitted in suitable locations where air would be trapped by the water.

Although the effect that the air bubbles have on the movement of the water is minimal in reality, including this interaction would obviously enhance the overall realism of the water flow.

A more scientifically based simulation would be something I would be interested in pursuing but I think I would prefer to optimise a system similar to the one described here

Presently there is no user interface and in order to tweak the results it is necessary to change the actual code but it would be a simple process to add a GUI and make the system more user-friendly, easy to use and customisable so that an artist can tweak the way the bubbles look. It could be developed into a useful tool for a production environment despite its currently limited scope.

It might be a worthwhile extension to the project to look into the actual shading and rendering of the bubbles and water. Especially fascinating phenomena occur with the light at the boundaries between bubbles as considered by Kück *et al.* [KÜI02].

The bubbles in this model are only modelled using spheres and there was insufficient time to implement a system with merging or warping. They still look moderately convincing and a higher level of detail would possibly only be necessary for close up shots or slow motion viewing of the bubbles.

It would be interesting to extend this project with more dynamic bubbles similar to those of Hong *et al.* [HON03]. This might be possible if a much finer grid was used for storing the position and velocities of the fluid particles. A more physically based study of bubble motion is covered by Bunner *et al.* [BUN99].

The problems encountered with the three-dimensional simulations are only small glitches and could no doubt be solved with some more time. I am slightly disappointed not to have smoothed out these issues but they were unexpected and were not encountered until the last minute.

The simulation ended up being much slower than initially desired. The code could possibly be slightly optimised but I think the main problem is that the code is written using Maya's internal scripting language MEL. This interpreted scripting language executes much slower than a similar system would if written in a natively compiled language such as C. Considering time constraints, the approach I took was the most feasible for achieving satisfying results but a C based native implementation might have ultimately been more flexible.

The decision to use a particle based system was definitely the correct choice. Initially Maya fluids were considered but it was soon discovered they are not flexible enough to handle what would be asked of them. Particles worked well as they can be very high resolution, and can have extra particles added to or removed from a system arbitrarily.

On a personal level I am pleased with the outcome of the project. There were numerous occasions when I felt that I would not be able to get a result resembling bubbles at all and I encountered numerous problems with the general strangeness of MEL syntax and errors.

In terms of actual innovation I feel this project was a reasonable success. I had never done any work with either particle systems or with fluids so I had lots to learn coming into the project. I now feel quite capable with both areas and feel much more knowledgeable about computer graphics research and study in general. Similar works have been completed elsewhere within the last couple of years, however I couldn't find any Maya integrated systems that would do a similar job. Also, I approached the problem in a different manner by looking to simulate the bubbles only adding the result to a pre-existing water simulation.

5.0 Acknowledgements

Eike Anderson

6.0 References

[BUN99]
BUNNER B., TRYGGVASON G.: Direct numerical simulations of three-dimensional bubbly flows. *Physics of Fluids 11, 8* (1999)

[ENR02]
ENRIGT D., MARSCHNER S., FEDKIW R.: Practical animation of liquids. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 736-744

[FOS02]
FOSTER N., FEDWICK R.: Practical animation of liquids. In *proceedings of the 28th annual conference*

on Computer graphics and interactive techniques (2002)

[FOS96]
FOSTER N., METAXAS D.: Realistic animation of liquids. *Graphical Models and Image Processing 58, 5* (1996)

[GLA90]
GLASSNER A.: Andrew Glassner's notebook: Soap bubbles. Part 2. *IEEE Computer Graphics Principles and Practice.* Addison-Wesley, Reading, Mass. (1990)

[GRE04]
GREENWOOD S. T., HOUSE D. H.: Better with bubbles: enhancing the visual realism of simulated fluid. In *SCA '04: Proceedings of 2004 ACM SIGGRAPH/Eurographics symposium on Computer Animation* (2004), pp.163-169

[HON03]
HONG J. M., CHANG-HUN K.: Animation of Bubbles in liquid. *In Proceedings of Eurographics '03* (2003)

[KÜI02]
KÜIK H., VOGELGSANG C., GREINER G.: Simulation and rendering of liquid foams. *In Proceedings of Graphics Interface '02* (2002)

[MÜL03]
MÜLLER M., SOLENTHALER B., KEISER R., GROSS M.: Particle-based fluid simulation for interactive applications. *Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation* (2003)

[MÜL05]
MÜLLER M., SOLENTHALER B., KEISER R., GROSS M.: Particle-based fluid-fluid interaction. In *Proceedings of 2005 ACM SIGGRAPH/Eurographics symposium on Computer Animation* (2005)

[NEX06]
NEXT LIMIT TECHNOLOGIES, 2005. Welcome to Realflow.com. Madrid, Spain. Available from: http://www.nextlimit.com/realflow/index.html [Accessed 7 March 2006]

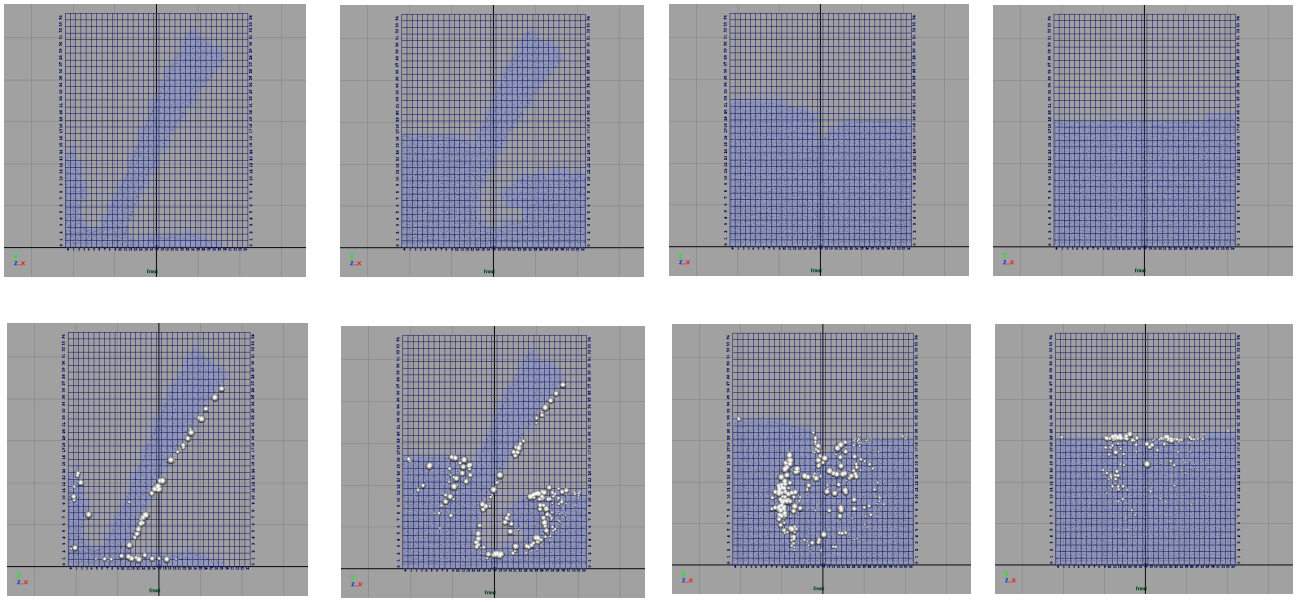

Figure 3: Two-dimensional simulation of water pouring into a glass. The top row shows the water particles pouring into the glass. The blue fluid indicates the fluid grid used for dividing the total area allowing for more localised calculations. The bottom row shows the air bubbles as they are added into the water model and then simulated.

Appendix

MEL Source Code

```
//#########################################################################################################################################
//#                                                                                                                                       #
//#                                                  Bubble Simulation Script                                                             #
//#                                                        Joel Green                                                                     #
//#                                                       Version 1.0                                                                     #
//#                                                                                                                                       #
//#########################################################################################################################################

//SETUP FOR USE:
//
//This system requires two particle systems and two fluid grids of equal resolution. The fluid grids should be just big enough to encompass
//the simulation of the pouring water. This code is setup for objects with specific names so it might be necessary to either re-name objects
//or alter the code
//
//The fluid grid for the water should have both density and velocity set to "static grid" and the fluid grid for the bubbles should have
//density and fluid set to "dynamic grid"
//
//The bubble particle system must have a radiusPP attribute added and have its lifespan mode set to "lifespanPP only".
//
//To cache the bubble particle data simply tick the "cache Data" button in the Attribute editor. It is important to clear or delete the expression
//before scrubinng the timeline or pressing play.

string $particle = "particleShape";
string $bubParticle = "bubParticleShape";
string $fluid = "fluidShape1";
string $bubFluid = "bubFluidShape";

main();

/*#########################################################################################################################################
for each fluid particle add density and velocity to fluid grid
#########################################################################################################################################*/

proc main()
{
	string $particle = "particleShape";
	string $bubParticle = "bubParticleShape";
	string $fluid = "fluidShape1";
	string $bubFluid = "bubFluidShape";

	float $res[] = `getAttr ($fluid + ".resolution")`;
	int $xRes = $res[0];
	int $yRes = $res[1];
	int $zRes = $res[2];

	int $max = ($xRes*$yRes*$zRes);

	global int $numPart[];
	global float $totVel[];

	//reset arrays to zero
	for($k=0;$k<$max;$k++)
	{
		$totVel[$k]=0;
		$numPart[$k]=0;
	}

	int $pCount;// = `getAttr ($particle + ".count")`;
	$pCount = `particle -ct -q $particle`;
	int $vX, $vY, $vZ, $cX, $cY, $cZ;
	float $pos[], $vel[], $fluidVelo[], $pPos[], $new[], $vel[], $fluV[];
	float $velX, $velY, $velZ;

	float $res[] = `getAttr ($fluid + ".resolution")`;
	int $xRes = $res[0];
	int $yRes = $res[1];
	int $zRes = $res[2];

	setFluidAttr -at "density" -cl fluid1; //remove all fluid
	setFluidAttr -at "velocity" -cl fluid1;		//remove all fluid
	vector $voxel;

	for ($i = 0; $i< $pCount; $i++)			//loop through the fluid particles
	{
		$pos = `getParticleAttr -at position -array true ($particle + ".pt[" + $i + "]")`;
		$vel = `getParticleAttr -at rfVelocity -array true ($particle + ".pt[" + $i + "]")`;

		$velX = $vel[0];
		$velY = $vel[1];
		$velZ = $vel[2];

		$px = $pos[0];
```

```
		$py = $pos[1];
		$pz = $pos[2];

		//get which voxel particle is in
		$voxel = `fluidVoxelInfo -voxel $px $py $pz fluid1`;

		$vX = $voxel.x;
		$vY = $voxel.y;
		$vZ = $voxel.z;

		//add density to fluid
		setFluidAttr -at "density" -fv 1 -xi $vX -yi $vY -zi $vZ fluid1;

		$fluV = `getFluidAttr -at "velocity" -xi $vX -yi $vY -zi $vZ fluid1`;	//1 or 0 if fluid or not

		//set velocity in fluids
		fluidVelocity($vX, $vY, $vZ, $velX, $velY, $velZ);		//average fluid velocity
	}

	//loop through all voxels and assign average velocities

	for($cX=0;$cX<$xRes;$cX++)
	{
		for($cY=0;$cY<$yRes;$cY++)
		{
			for($cZ=0;$cZ<$zRes;$cZ++)
			{
				averageVel($cX, $cY, $cZ);
			}
		}
	}

	simulate();  //simulate bubble motion (probably badly)
}

/*#########################################################################################################################
adding velocity to fluid with averaging - use 3 arrays to store av. velocities for each voxel and extra array to store num of part in each voxel
#########################################################################################################################*/

proc fluidVelocity(int $vX, int $vY, int $vZ, float $velX, float $velY, float $velZ)
{
	string $fluid = "fluidShape1";
	float $res[] = `getAttr ($fluid + ".resolution")`;
	float $temp = 0, $x, $y, $z;
	int $xRes = $res[0];
	int $yRes = $res[1];
	int $zRes = $res[2];
	int $xindex, $yindex, $zindex, $numP, $thing;
	int $max = ($xRes*$yRes*$zRes);

	global float $totVel[];
	global int $numPart[];

	float $xtot;
	float $ytot;
	float $ztot;

	//get array index
	$thing = (($xRes*$xRes*$vX)+(($xRes*$vY)+$vZ));

	$xindex = $thing;
	$yindex = $thing + 1;
	$zindex = $thing + 2;

	//add 1 to appropriate part of matrix indicating particle inside corresponding voxel
	$temp = $numPart[$thing];
	$numPart[$thing] = $temp + 1;

	$numP = $numPart[$thing];

	$xtot = (($totVel[$xindex])+$velX);
	$ytot = (($totVel[$yindex])+$velY);
	$ztot = (($totVel[$zindex])+$velZ);


	$totVel[$xindex] = $xtot;
	$totVel[$yindex] = $ytot;
	$totVel[$zindex] = $ztot;

}

/*#########################################################################################################################
average the velocities in fluid grid voxels
#########################################################################################################################*/

proc averageVel(int $vX, int $vY, int $vZ)
{
	string $fluid = "fluidShape1";

	float $res[] = `getAttr ($fluid + ".resolution")`;
```

```
	int $xRes = $res[0];
	int $yRes = $res[1];
	int $zRes = $res[2];
	int $max = ($xRes*$yRes*$zRes);

	global float $totVel[];
	global int $numPart[];


	int $xindex, $yindex, $zindex, $numP, $thing;
	float $x, $y, $z, $tX, $tY, $tZ;

	$thing = (($xRes*$xRes*$vX)+(($xRes*$vY)+$vZ));

	$xindex = $thing;
	$yindex = $thing + 1;
	$zindex = $thing + 2;

	$numP = $numPart[$thing];


	if($numP!=0)
	{
		$tX = ($totVel[$xindex]);

		$tY = ($totVel[$yindex]);
		$tZ = ($totVel[$zindex]);

		$x = ($tX/$numP);
		$y = ($tY/$numP);
		$z = ($tZ/$numP);


		//assign average velocities to voxels
		setFluidAttr -at "velocity" -vv $x $y $z -xi $vX -yi $vY -zi $vZ fluid1;
	}
	else
	{
		setFluidAttr -at "velocity" -vv 0 0 0 -xi $vX -yi $vY -zi $vZ fluid1;
	}

}

/*#########################################################################################################################################
delete rubbish bubble particles
#########################################################################################################################################*
/

proc bubbleDelete()
{
	string $particle = "particleShape";
	string $bubParticle = "bubParticleShape";
	string $fluid = "fluidShape1";
	string $bubFluid = "bubFluidShape";
	int $pCountBub = `particle -ct -q $bubParticle`;
	vector $bubVoxel;
	int $vX, $vY, $vZ, $vwX, $vwY, $vwZ, $topY, $botY;
	float $fluidFull[], $active[], $topEmpty[], $botEmpty[], $rad[], $randDel;
	int $delete, $topDel, $botDel;
	float $res[] = `getAttr ($fluid + ".resolution")`;
	int $xRes = $res[0];
	int $yRes = $res[1];
	int $zRes = $res[2];

	for ($i = 0; $i< $pCountBub; $i++)  //cycle thorugh bubble particles
	{
		//DELETE RUBBISH BUBBLE PARTICLES

		$active =  `getParticleAttr -at lifespanPP -array true ($bubParticle + ".pt[" + $i + "]")`;
		$rad = `getParticleAttr -at radiusPP -array true ($bubParticle + ".pt[" + $i + "]")`;

		$fluidFull[0] = 0;
		$fluidFull = `getFluidAttr -at "density" -xi $vX -yi $vY -zi $vZ fluid1`;  //1 or 0 if fluid or not

		$pos = `getParticleAttr -at position -array true ($bubParticle + ".pt[" + $i + "]")`;

		$px = $pos[0];
		$py = $pos[1];
		$pz = $pos[2];

		//get which voxel
		$bubVoxel = `fluidVoxelInfo -cb -voxel $px $py $pz bubFluid`;

		$vX = $bubVoxel.x;
		$vY = $bubVoxel.y;
		$vZ = $bubVoxel.z;

		if($active[0] != 0)		//if particle not dead
		{
			if($pos[1] < 0)			//if below bottom of area -> KILL
```

```
{
	select ($bubParticle + ".pt[" + $i + "]");
	setParticleAttr -at lifespanPP -fv 0;  //set lifespan to 0
}
else if($pos[0]<-2.2 || $pos[0]>2.2)
{
	select ($bubParticle + ".pt[" + $i + "]");
	setParticleAttr -at lifespanPP -fv 0;  //set lifespan to 0
}
else if($pos[2]<-2.2 || $pos[2]>2.2)
{
	select ($bubParticle + ".pt[" + $i + "]");
	setParticleAttr -at lifespanPP -fv 0;  //set lifespan to 0
}
else
{

	if($fluidFull[0]!=0)	//if does have fluid in square
	{

	}
	else
	{

		$delete = 0;
		for($wX=-1;$wX<=1;$wX++)	//cycle x from either side
		{
			for($wY=-1;$wY<=1;$wY++)	//cycle y either side
			{
				for($wZ=-1;$wZ<=1;$wZ++)//cycle z either side
				{
					$vwX = ($vX+$wX);
					$vwY = ($vY+$wY);
					$vwZ = ($vZ+$wZ);

					//1 or 0 if fluid or not
					$fluidFull = `getFluidAttr -at "density" -xi $vwX -yi $vwY -zi $vwZ fluid1`;

					if($fluidFull[0]!=1)	//if doesn't have fluid in square
					{
						$delete += 1;
					}
				}
			}
		}

		if($delete >= 20)  //if no fluid in any surrounding squares
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at lifespanPP -fv 0;  //set lifespan to 0
		}
		else		//check if on surface - no fluid above, all fluid below
		{
			$topDel = 0;
			$botDel = 0;
			for($wX=-1;$wX<=1;$wX++)	//cycle x from either side
			{
				for($wZ=-1;$wZ<=1;$wZ++)//cycle z either side
				{
					$vwX = ($vX+$wX);
					$vwZ = ($vZ+$wZ);
					$topY = ($vY+1);
					$botY = ($vY-1);

					$topEmpty = `getFluidAttr -at "density" -xi $vwX -yi $topY -zi $vwZ fluid1`;
					$botEmpty = `getFluidAttr -at "density" -xi $vwX -yi $botY -zi $vwZ fluid1`;

					if($topEmpty[0]!=1)	//if doesn't have fluid in square
					{
						$topDel += 1;
					}
					if($botEmpty[0]!=0)	//if doesn't have fluid in square
					{
						$botDel += 1;
					}
				}
			}

			if($topDel >= 6 && $botDel >= 6)		//if at surface
			{
				$randDel = rand($rad[0], 0.1);		//radii is between 0.02 and 0.07
				//0.09 is abritary value that might give nice percentage deleted
				if($randDel>0.09)
				{
					select ($bubParticle + ".pt[" + $i + "]");
					setParticleAttr -at lifespanPP -fv 0;  //set lifespan to 0
				}
			}
		}
	}
```

```
                }
            }
        }
        bubbleCreate();
}

/*##########################################################################################################################################
creation of new bubble particles
##########################################################################################################################################*/

proc bubbleCreate()
{
        string $particle = "particleShape";
        string $bubParticle = "bubParticleShape";
        string $fluid = "fluidShape1";
        string $bubFluid = "bubFluidShape";

        int $pCountBub = `particle -ct -q $bubParticle`;
        float $res[] = `getAttr ($fluid + ".resolution")`;
        int $xRes = $res[0];
        int $yRes = $res[1];
        int $zRes = $res[2];
        float $fluidEmpty[];
        float $bubEmpty[];
        float $fluidVel[], $fluVelAve[];
        vector $voxelPos;
        float $vpX, $vpY, $vpZ, $rad;
        int $yup=0, $wX, $wY, $wZ, $aX, $aY, $aZ;
        int $vX, $vY, $vZ, $vwX, $vwY, $vwZ, $avwX, $avwY, $avwZ;
        int $div, $above;

        for($vX=1;$vX<($xRes-2);$vX++)          //loop through (nearly) all voxels (not edge ones) cos don't want to create bubbles right at edges
        {
                for($vY=1;$vY<($yRes-2);$vY++)
                {
                        for($vZ=1;$vZ<($zRes-2);$vZ++)
                        {
                                $voxelPos = `fluidVoxelInfo -vc -xIndex $vX -yIndex $vY -zIndex $vZ fluid1`;        //position of middle of voxel
                                $fluidEmpty = `getFluidAttr -at "density" -xi $vX -yi $vY -zi $vZ fluid1`;//1 or 0 if fluid or not
                                $bubEmpty = `getFluidAttr -at "density" -xi $vX -yi $vY -zi $vZ bubFluid`;//1 or 0 if bubble or not

                                //put velocity of current voxel in $fluidVel
                                $fluidVel = `getFluidAttr -at "velocity" -xi $vX -yi $vY -zi $vZ -lf fluid1`;

                                $div = 0;

                                //MAKE BUBBLES
                                if(($fluidEmpty[0] != 1) && ($bubEmpty[0] != 1))      //if square is empty of fluid and bubble
                                {
                                        $yup = 0;
                                        $above=0;

                                        //check the 9 voxels above
                                        for($aX=-1;$aX<=1;$aX++)
                                        {
                                                for($aZ=-1;$aZ<=1;$aZ++)
                                                {
                                                        $avwX = ($vX+$aX);
                                                        $avwY = ($vY+1);
                                                        $avwZ = ($vZ+$aZ);

                                                        $fluidEmpty = `getFluidAttr -at "density" -xi $avwX -yi $avwY -zi $avwZ fluid1`;

                                                        if($fluidEmpty[0]!=0)   //if there is fluid in voxel
                                                        {
                                                                $above +=1;
                                                        }

                                                }
                                        }

                                        if($above>=6)      //if more than 6 of 9 voxels above have fluid can create bubble
                                        {
                                                //$fluidEmpty = `getFluidAttr -at "density" -xi $vwX -yi $vwY -zi $vwZ fluid1`;

                                                //reset values
                                                $fluVelAve[0] = 0;
                                                $fluVelAve[1] = 0;
                                                $fluVelAve[2] = 0;


                                                $yup = 1;

                                                //get average velocity of surrounding voxels
                                                for($aX=-1;$aX<=1;$aX++)  //cycle x from either side
                                                {
                                                        for($aY=-1;$aY<=1;$aY++)  //cycle y either side
                                                        {
                                                                for($aZ=-1;$aZ<=1;$aZ++)    //cycle z either side
                                                                {
```

```
                                                //fluid co-ord of voxel to test
                                                $vwX = ($vX+$aX);
                                                $vwY = ($vY+$aY);
                                                $vwZ = ($vZ+$aZ);

                                                $fluidEmpty = `getFluidAttr -at "density" -xi $vwX -yi $vwY -zi $vwZ fluid1`;

                                                if($fluidEmpty[0]!=0)   //if there is fluid in voxel
                                                {
                                                        $div +=1;
                                                        $fluVel = `getFluidAttr -at "velocity" -xi $vwX -yi $vwY -zi $vwZ fluid1`;
                                                        $fluVelAve[0] += $fluVel[0];
                                                        $fluVelAve[1] += $fluVel[1];
                                                        $fluVelAve[2] += $fluVel[2];
                                                }
                                        }
                                }
                        }


                        if($yup==1)     //if good square for making of bubble eg/ if matches criteria
                        {
                                //position of middle of voxel
                                $voxelPos = `fluidVoxelInfo -vc -xIndex $vX -yIndex $vY -zIndex $vZ fluid1`;

                                //square will always have no fluid therefore need av velocity from surrounding voxels
                                $vpX = $voxelPos.x;
                                $vpY = $voxelPos.y;
                                $vpZ = $voxelPos.z;

                                //Average surrounding velocities for x y and z components
                                $fluVelAve[0] /= $div;
                                $fluVelAve[1] /= $div;
                                $fluVelAve[2] /= $div;

                                //emit new particle
                                $rad = rand(0.02,0.1);
                                emit -object bubParticle -position $vpX $vpY $vpZ -at lifespanPP -fv 1000 -at velocity -vv $fluVelAve[0]
$fluVelAve[1] $fluVelAve[2] -at radiusPP -fv $rad;
                        }
                }
        }
    }
    bubbleFluid();
}

/*##########################################################################################################################
cycle through bubble particles and set density to 1 for each voxel with particle
##########################################################################################################################*/

proc bubbleFluid()
{
    string $particle = "particleShape";
    string $bubParticle = "bubParticleShape";
    string $fluid = "fluidShape1";
    string $bubFluid = "bubFluidShape";
    int $pCountBub;
    $pCountBub = `particle -ct -q $bubParticle`;

    setFluidAttr -at "density" -cl bubFluid;
    vector $bubVoxel;
    int $vX, $vY, $vZ;
    float $pos[];

    for ($i = 0; $i< $pCountBub; $i++)  //cycle thorugh bubble particles
    {
        float $active[] =  `getParticleAttr -at lifespanPP -array true ($bubParticle + ".pt[" + $i + "]")`;
        if($active[0]!=0)  //if not already killed
        {
            //SET DENSITY FOR BUBBLE FLUID WHERE BUBBLE PARTICLE FOUND

            $pos = `getParticleAttr -at position -array true ($bubParticle + ".pt[" + $i + "]")`;

            $px = $pos[0];
            $py = $pos[1];
            $pz = $pos[2];

            //get which voxel
            $bubVoxel = `fluidVoxelInfo -cb -voxel $px $py $pz bubFluid`;

            $vX = $bubVoxel.x;
            $vY = $bubVoxel.y;
            $vZ = $bubVoxel.z;

            //add density
            setFluidAttr -at "density" -fv 1 -xi $vX -yi $vY -zi $vZ bubFluid;
        }
    }
```

```
	select -cl;	//deselects most recent bub particle at end of frame
}

/*#########################################################################################################################################
simulate bubbles
#########################################################################################################################################*/

proc simulate()
{
	string $bubParticle = "bubParticleShape";
	int $i, $vXb, $vYb, $vZb, $pxb, $pyb, $pzb;
	int $pCountBub;
	$pCountBub = `getAttr ($bubParticle + ".count")`;
	vector $bubVoxel, $voxelPos;
	float $fluVelb[], $fluDens[], $posb[], $velSur[], $radius[];
	float $fluVelAve[];
	float $xCheck, $yCheck, $zCheck, $xAveCheck, $yAveCheck, $zAveCheck, $yBuoyancy, $xBuoyancy, $aVel, $vpX, $vpY, $vpZ, $vwX, $vwY,
$vwZ;

	for ($i = 0; $i< $pCountBub; $i++)  //cycle thorugh bubble particles
	{
		//work out what voxel particle is in
		$posb = `getParticleAttr -at position -array true ($bubParticle + ".pt[" + $i + "]")`;

		$pxb = $posb[0];
		$pyb = $posb[1];
		$pzb = $posb[2];

		//get which voxel
		$bubVoxel = `fluidVoxelInfo -cb -voxel $pxb $pyb $pzb bubFluid`;

		//x and y voxel components holding current particle
		$vXb = $bubVoxel.x;
		$vYb = $bubVoxel.y;
		$vZb = $bubVoxel.z;

		//reset values
		$fluVelAve[0] = 0;
		$fluVelAve[1] = 0;
		$fluVelAve[2] = 0;
		$div=0;

		$fluVelb = `getFluidAttr -at "velocity" -xi $vXb -yi $vYb -zi $vZb fluid1`;
		$fluDens = `getFluidAttr -at "density" -xi $vXb -yi $vYb -zi $vZb fluid1`;	//1 or 0 if fluid or not

		if($fluDens[0]!=0)	//if there is fluid in voxel
		{
			$xCheck = ($posb[0] + ($fluVelb[0]/12.5));	//pos + 2*velo per frame because velocity moves particle before checked
			$yCheck = ($posb[1] + ($fluVelb[1]/12.5));	//pos + 2*velo per frame
			$zCheck = ($posb[2] + ($fluVelb[2]/12.5));

			int $surround =0;
			$velSur[0]=0;
			$velSur[1]=0;
			$velSur[2]=0;

			for($wX=-1;$wX<=1;$wX++)	//cycle x from either side
			{
				for($wY=-1;$wY<=1;$wY++)	//cycle y from either side
				{
					for($wZ=-1;$wY<=1;$wY++)	//cycle z
					{
						//x, y and z voxel components surrounding current particle
						$vwX = ($vXb+$wX);
						$vwY = ($vYb+$wY);
						$vwZ = ($vZb+$wZ);

						//1 or 0 if fluid or not
						$fluDens = `getFluidAttr -at "density" -xi $vwX -yi $vwY -zi $vwZ fluid1`;
						if($fluDens[0]!=0)
						{
							$surround +=1;
						}
					}
				}
			}

			//BUOYANCY

			if($surround>=20)	//if almost totally surrounded by fluid - arbitary value at the mo
			{
				$radius = `getParticleAttr -at radiusPP ($bubParticle + ".pt[" + $i + "]")`;
				//$aVel = (abs($velSur[1])/9);
				//x is arbritary value to make bubbles rise. radius means bigger = faster
				//$yBuoyancy = $fluVelb[1]+((25/$aVel)*$radius[0]);
				$yBuoyancy = $fluVelb[1]+(100*$radius[0]);
				$xBuoyancy = (1.5*$fluVelb[0])+(0.1*(sin(frame + $i)));

				if($posb[0]<-2.0 || $posb[0]>2.0)	//IF TOWARD EDGES IN X
				{
```

```
		if($posb[2]<-2.0 || $posb[2]>2.0)	//IF TOWARD EDGES IN Z
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv -0.5 $yBuoyancy -0.5;  //set velocity
		}
		else
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv 0 $yBuoyancy $fluVelb[2];   //set velocity
		}
	}
	else if($posb[2]<-2.0 || $posb[2]>2.0)		//IF TOWARD EDGES IN Z
	{
		select ($bubParticle + ".pt[" + $i + "]");
		setParticleAttr -at velocity -vv $fluVelb[0] $yBuoyancy 0;   //set velocity
	}
	else if($posb[1]<0.2)			//IF NOT NEAR EDGES IN X OR Z BUT NEAR BOTTOM
	{
		if($fluVelb[1]>-0.5)	//if y-vel is not too downward strong then buoyancy it
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv $fluVelb[0] $yBuoyancy $fluVelb[2];	//set velocity
		}
		else
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv $fluVelb[0] 0.5 $fluVelb[2];  //set velocity
		}
	}
	else	//IF NOT IN VERY EDGE VOXELS
	{
		//IF GOING TO BE OUTSIDE BOUNDS ON NEXT FRAME
		if(($xCheck<-2.2)||($xCheck>2.2))  //x component outside boundaries
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv ($fluVelb[0]/2) $yBuoyancy $fluVelb[2];//set velocity
		}
		if($yCheck<0)	//y component
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv $fluVelb[0] ($fluVelb[1]/2) $fluVelb[2];  //set velocity

		}
		if(($zCheck<-2.2)||($zCheck>2.2))
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv $xBuoyancy $yBuoyancy ($fluVelb[2]/2);		//set velocity
		}
		else	//NOT NEAR EDGES - NO REAL RISK OF GOING OUT OF BOUNDS
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv $xBuoyancy $yBuoyancy $fluVelb[2];	//set velocity
		}
	}

}
//NON-BUOYANT BEHAVIOUR

else
{

	if($posb[0]<-2.0 || $posb[0]>2.0)
	{
		select ($bubParticle + ".pt[" + $i + "]");
		setParticleAttr -at velocity -vv 0 $fluVelb[1] 0;  //set velocity
	}
	else if($posb[1]<0.2)
	{
		select ($bubParticle + ".pt[" + $i + "]");
		setParticleAttr -at velocity -vv $fluVelb[0] 0 0;  //set velocity
	}
	else
	{
		//checking for boundaries against position for next frame
		if(($xCheck<-2.2)||($xCheck>2.2))  //x component outside boundaries
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv ($fluVelb[0]/2) $fluVelb[1] $fluVelb[2]; //set velocity
		}
		if(($zCheck<-2.2)||($zCheck>2.2))
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv $fluVelb[0] $fluVelb[1] ($fluVelb[2]/2); //set velocity
		}
		if($yCheck<0)	//y component
		{
			select ($bubParticle + ".pt[" + $i + "]");
			setParticleAttr -at velocity -vv $fluVelb[0] ($fluVelb[1]/2) $fluVelb[2]; //set velocity
		}
		else
```

```
                {
                    select ($bubParticle + ".pt[" + $i + "]");
                    setParticleAttr -at velocity -vv $fluVelb[0] $fluVelb[1] $fluVelb[2];       //set velocity
                }
            }
        }
    }
    else   //not in voxel with fluid
    {
        for($wX=-1;$wX<=1;$wX++)        //cycle x from either side
        {
            for($wY=-1;$wY<=1;$wY++)        //cycle y from either side
            {
                for($wZ=-1;$wZ<=1;$wZ++)
                {
                    //x and y voxel components surrounding current particle
                    $vwX = ($vXb+$wX);
                    $vwY = ($vYb+$wY);
                    $vwZ = ($vZb+$wZ);

                    //1 or 0 if fluid or not
                    $fluDens = `getFluidAttr -at "density" -xi $vwX -yi $vwY -zi $vwZ fluid1`;

                    if($fluDens[0]!=0)        //if there is some density in voxel then add 1 to $div
                    {
                        $div +=1;

                        $fluVelb = `getFluidAttr -at "velocity" -xi $vwX -yi $vwY -zi $vwZ fluid1`;
                        //print ("fluVelsurround " + $i + "= "+$fluVelb[0]+"\n");

                        $fluVelAve[0] += $fluVelb[0];
                        $fluVelAve[1] += $fluVelb[1];
                        $fluVelAve[2] += $fluVelb[2];
                    }
                }
            }
        }

        //avoid dividing by 0 errors

        if($div==0)
        {
            //delete - should never happen but prob will
        }
        else
        {
            //Average surrounding velocities for x y and z components
            $fluVelAve[0] /= $div;
            $fluVelAve[1] /= $div;
            $fluVelAve[2] /= $div;

            $xAveCheck = ($posb[0] + ($fluVelAve[0]/12.5));
            $yAveCheck = ($posb[1] + ($fluVelAve[1]/12.5));
            $zAveCheck = ($posb[2] + ($fluVelAve[2]/12.5));

            select ($bubParticle + ".pt[" + $i + "]");
            setParticleAttr -at velocity -vv 0 $fluVelAve[1] $fluVelAve[2];     //set velocity

            if($posb[0]<-2.0 || $posb[0]>2.0)
            {
                select ($bubParticle + ".pt[" + $i + "]");
                setParticleAttr -at velocity -vv 0 $fluVelAve[1] $fluVelAve[2];     //set velocity
            }
            else if($posb[1]<0.2)
            {
                select ($bubParticle + ".pt[" + $i + "]");
                setParticleAttr -at velocity -vv $fluVelAve[0] 0 $fluVelAve[2];     //set velocity
            }
            else
            {
                if(($xAveCheck<-2.2)||($xAveCheck>2.2))       //x component outside boundaries
                {
                    if($yAveCheck>0)//y component
                    {
                        select ($bubParticle + ".pt[" + $i + "]");
                        setParticleAttr -at velocity -vv ($fluVelAve[0]/2) $fluVelAve[1] $fluVelAve[2];       //set velocity
                    }
                }
                if($yAveCheck<0)//y component
                {
                    if(($xAveCheck>-2.2)||($xAveCheck<2.2))       //x component outside boundaries
                    {
                        select ($bubParticle + ".pt[" + $i + "]");
                        setParticleAttr -at velocity -vv $fluVelAve[0] ($fluVelAve[1]/2) $fluVelAve[2];       //set velocity
                    }
                }
                else if((($xAveCheck<-2.2)||($xAveCheck>2.2)) && (($zAveCheck<-2.2)||($zAveCheck>2.2)) && ($yAveCheck<0))
                {
                    select ($bubParticle + ".pt[" + $i + "]");
                    setParticleAttr -at velocity -vv 0 0 0;         //set velocity
```

```
                                        }
                                        else
                                        {
                                                //$fluVelAve[1]+=4;     //buoyancy
                                                select ($bubParticle + ".pt[" + $i + "]");
                                                setParticleAttr -at velocity -vv $fluVelAve[0] $fluVelAve[1] $fluVelAve[2];   //set velocity
                                        }
                                }
                        }
                }
        }
        bubbleDelete();
}
```