UTube Model

CBP 23-07-22

Summary

The U-Tube is another DHO experiment. So we can use our existing code for the MAS22_MassSpring_Airtrack; yes, we don't have to code from scratch, all we have to do is to tweak our existing code. Almost. You will also learn something about Unreal – how to create your own Actor Class, how to import assets (static meshes) created in other software, such as Blender or 3DSMax, and how to assign materials to static meshes. Here's a summary of what we'll be doing:

1. Creating a new class **MAS22_UTube** using Unreal Editor (this will create **MAS22_UTube.cpp** and **MAS22_UTube.h** files for you).

2. Copying code from our existing **MAS22_MassSpring_Airtrack.cpp** and **MAS22_MassSpring_Airtrack.h** files. We'll rename all functions and re-purpose the code, so we don't have to code everything from scratch.

3. Adding in the new assets (static meshes, materials) into the Content Browser

4. Assigning materials to our assets using the Material Editor.

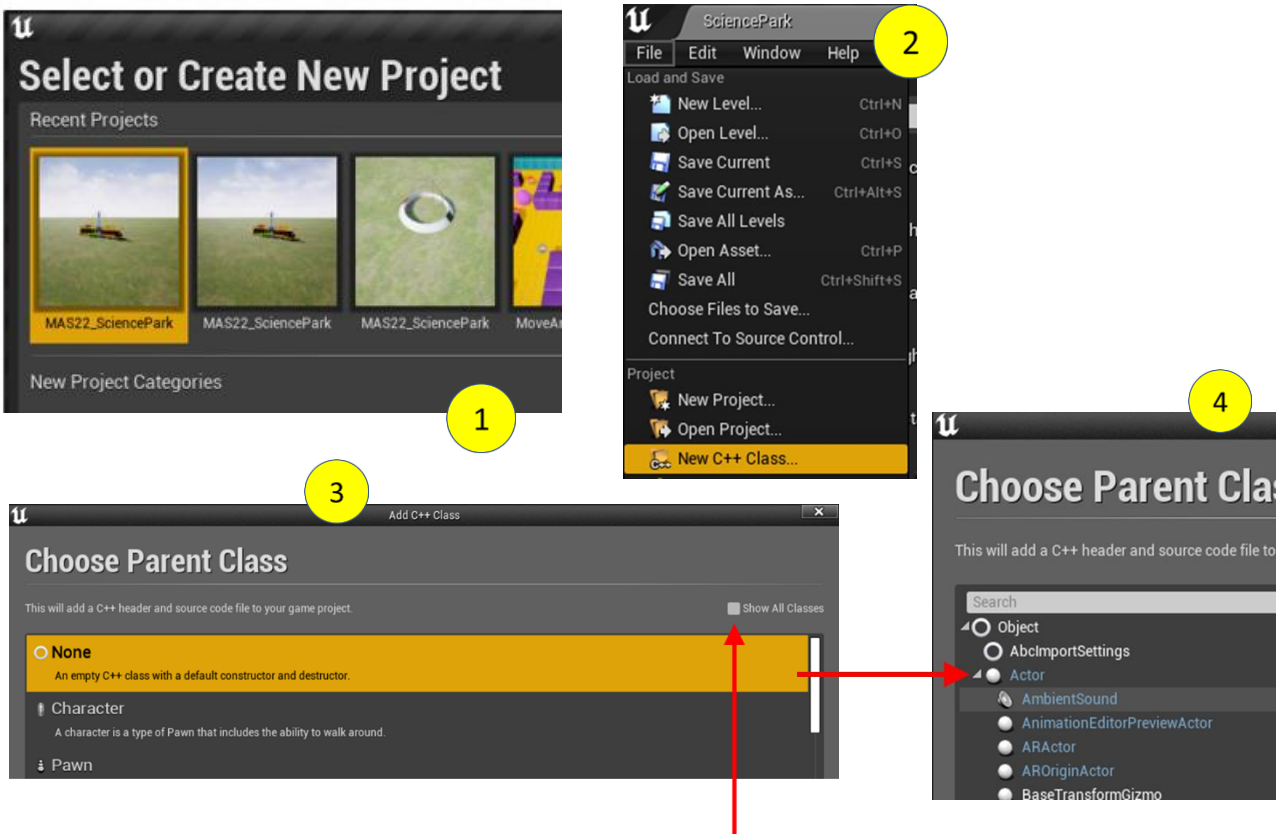
Stage 1 Creating a new MAS22_Utube Class (.cpp and .h) files.

(1) Open Unreal Editor and select our project.

(2) In **File** select **New C++ Class**

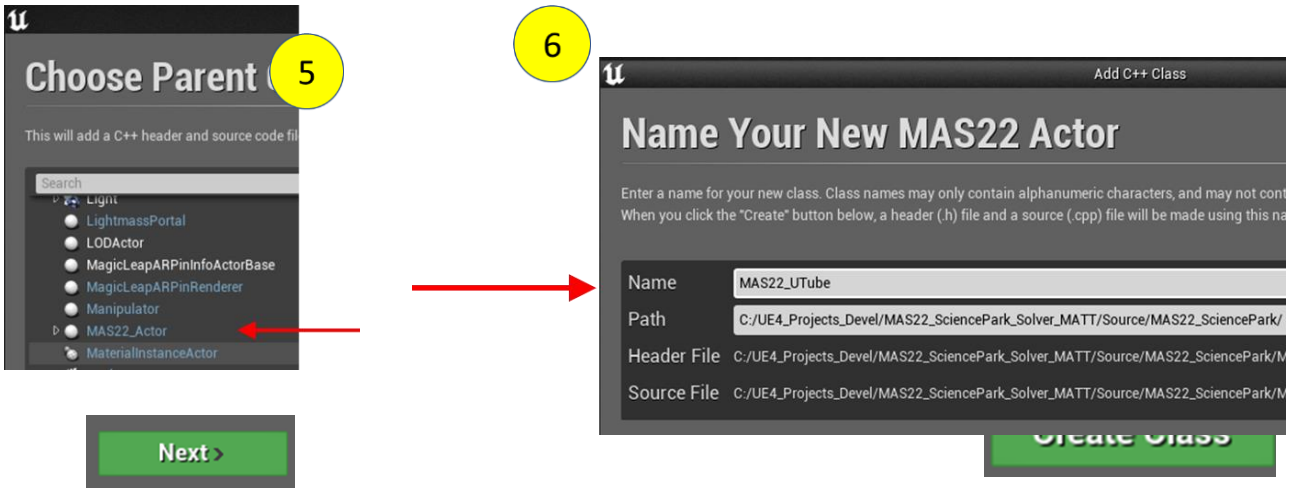
(3) In the Choose Parent Class dialogue, check **Show All Classes**

(4) Expand **Actor**

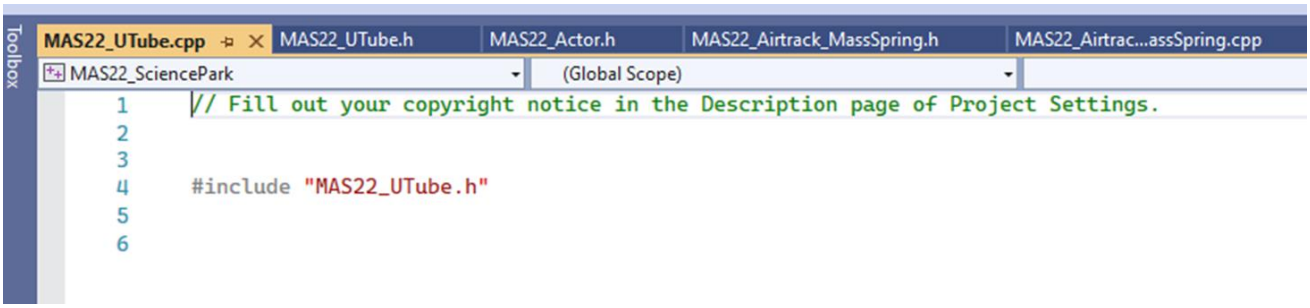


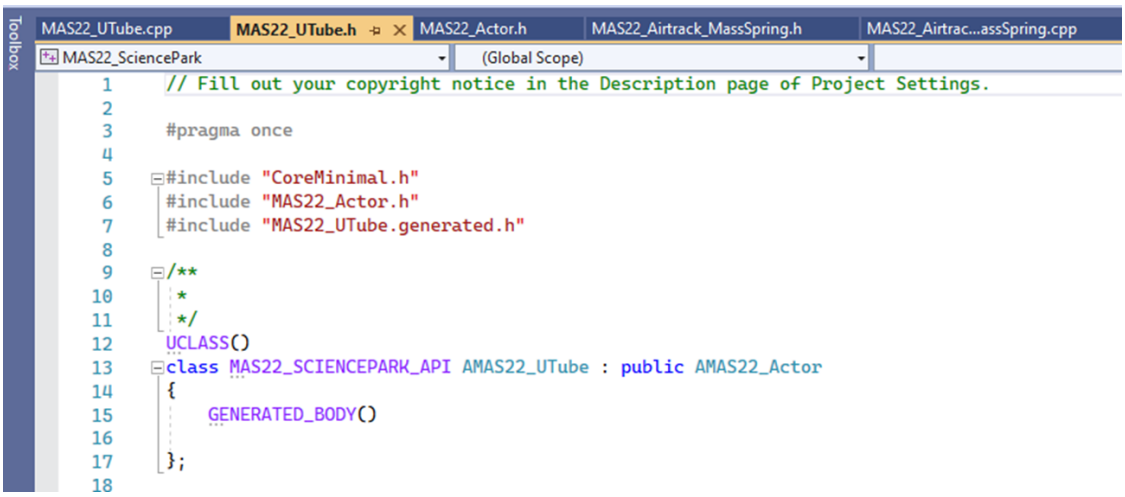
(5) Select **MAS22_Actor** (don't expand) and hit Next

(6) Give it a name, and hit Create Class



The following files will be automatically created for you and will appear in Visual Studio. Let's see what they tell us.





The **MAS22_UTube.cpp** file is a one-liner. Unreal knows we must include stuff from the associated header file. But of course we know we have to include other headers.

The **MAS22_UTube.h** file does a little more. First, there are three header files it includes:

```
#include "CoreMinimal.h"
#include "MAS22_Actor.h"
#include "MAS22_UTube.generated.h"
```

The first, **CoreMinimal.h** is the engine's best guess at the functions we shall require from the engine, although we know we may require more.

The second **MAS22_Actor.h** tells us to include all the info from our parent class **MAS22_Actor**, this is the info needed to make out UTube work within the context of the MAS22 Simulation Engine. Unreal knew we must include this, since we derived our current class from **MAS22_Actor**.

The third **MAS22_UTube.generated.h** includes all the info required by the Unreal Engine to make our code work, for example to get our assets, parameters and any other variables declared with **UPROPERTY()** linked up with the engine and especially the Editor. **This must be the last include. So, when we add any other includes they MUST COME BEFORE THIS.**

Next comes the macro **UCLASS()** This tells the compiler that we are creating a new C++ class, but also that it is an *Unreal engine* class. Again, this mysteriously links with the Unreal engine internals.

Then comes the class definition on line 13 (above). The **MAS22_SCIENCEPARK_API** bit reminds us that our new UTube experiment is part of ourMAS22_SciencePark project. The **AMAS22_UTube : public AMAS22_Actor** bit reminds us that our new **MAS22_UTube** actor is a child of the **MAS22_Actor**. Unreal adds an **A** (for Actor) because of its internal naming convention.

Stage 2 Copying and Pasting our MAS22_AirTrack code.

The basic idea is to copy and paste our airtrack code into MAS22_UTube.cpp and MAS22_UTube.h. But we must retain the integrity of these lines of code in MAS22_UTube.h which Unreal has provided for us. This will happen automatically if you follow the steps indicated below.

```
#include "MAS22_UTube.generated.h"

UCLASS()
class MAS22_SCIENCEPARK_API AMAS22_UTube : public AMAS22_Actor
{
	GENERATED_BODY()

};
```

1a) Copy the code in **MAS22_Airtrack_MassSpring.cpp** into over-witing everything.

1b) Copy the code in **MAS22_Airtrack_MassSpring.h** into **MAS22_UTube.h** over-writing everything.

2a) In **MAS22_UTube.h** replace all text **MAS22_AirTrack_MassSpring** with **MAS22_UTube**

2b) In **MAS22_UTube.cpp** replace all text **MAS22_AirTrack_MassSpring** with **MAS22_UTube**

3) Now make the following name changes

i) In the header, rename **DataRecord_ATMS** to **DataRecord_UTube** in the definition of struct (near line 60)

ii) Rename **FDataRecord_ATMS** to **FDataRecord_Utube** at the end of the header (near line 190)

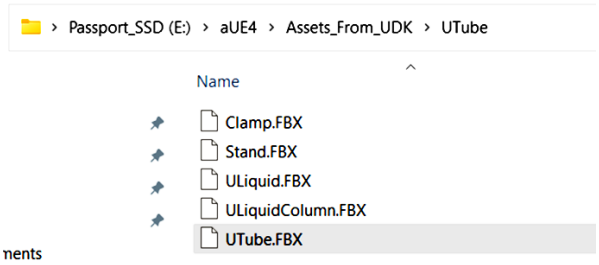
iii) In LogData() make the same name change

3) The code should now compile.

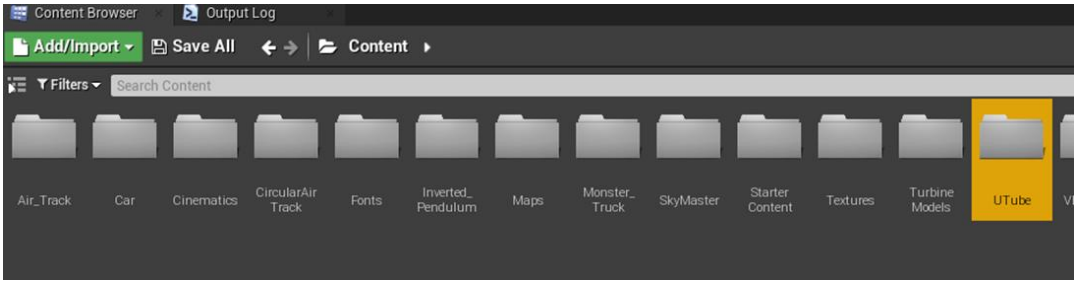
4) Drop a MAS22_UTube into your level. It will look like and air track because it is an air track. So now we must make it into a UTube.

Stage 3 – Importing New Assets

You are provided with a zipped folder containing some **fbx** files. These are static mesh assets which we have exported from our previous UDK engine. Now we must import these into Unreal 4.

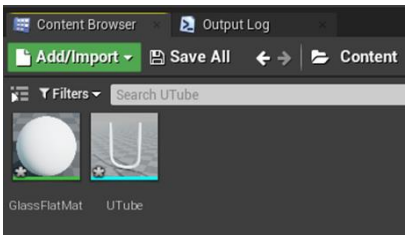


1) In the **Content Browser** right-click and create a new folder **UTube** like this

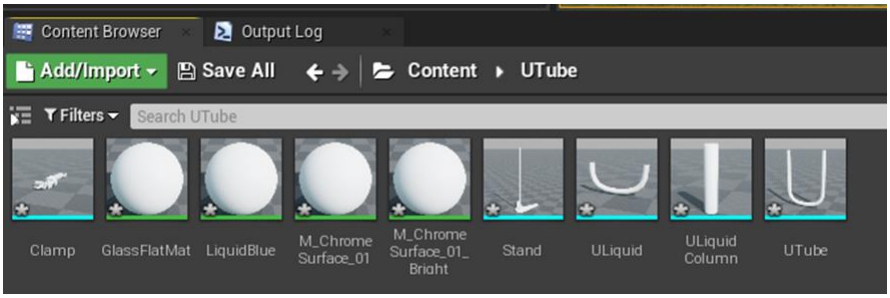


2) Still in the Content Browser open this folder (double left click).

3) Drag the **UTube.FBX** from your Windows folder into the UTube folder area in the Content Browser. Unreal will give you a dialogue box, just hit **Import All**. You should see this. The sphere is the material (set to default white – we shall change this later) and to the right you have the UTube static mesh.



4) Now import all the other **fbx** files and you will get this.



Stage 4 – Getting the Assets into Code

Make sure you are looking at **MAS22_Utube.cpp** in Visual Studio. Make the following changes in the *Constructor* **AMAS22_UTube::AMAS22_UTube()** function

1) Change **Asset1** to the following. *Refer to previous notes on how to do this.* Note, the following should be spread over two lines, each ending with the semicolon.

```
static ConstructorHelpers::FObjectFinder<UStaticMesh>
MeshAsset1(TEXT("StaticMeshStaticMesh'/Game/UTube/UTube.UTube'"));
	UStaticMesh* Asset1 = MeshAsset1.Object;
```

2a) Rename **Track** to **UTube** (Visual Studio: **Edit > Find and Replace > Quick Replace**)

```
	UTube = CreateDefaultSubobject<UStaticMeshComponent>("UTube");
	UTube->SetStaticMesh(Asset1);
	UTube->SetRelativeLocation(FVector(0, 0, 0));
	UTube->SetupAttachment(CapsuleCollider);
```

2b) And rename also in the header

```
	// 2(H) Add variable for the track ------------------------------------
	UPROPERTY(VisibleDefaultsOnly)
		class UStaticMeshComponent* UTube;
```

3) **Compile** and you should see the air-track has become a UTube! You might need to delete and replace the actor.

4) Now grab the asset for the **ULiquid** (Bend part of UTube)

```
	static
ConstructorHelpers::FObjectFinder<UStaticMesh>MeshAsset2(TEXT("StaticMeshStaticMesh'/Game/UTub
e/ULiquid.ULiquid'"));
	UStaticMesh* Asset2 = MeshAsset2.Object;
```

5a) Rename **Glider** to **ULiquid**

```
	ULiquid = CreateDefaultSubobject<UStaticMeshComponent>("ULiquid1");
	ULiquid->SetStaticMesh(Asset2);
	ULiquid->SetRelativeLocation(FVector(0, 0, 0));
	ULiquid->SetupAttachment(CapsuleCollider);
```

5b) and also in the header

```
	UPROPERTY(VisibleDefaultsOnly)
		class UStaticMeshComponent* ULiquid;
```

Don't try to compile since the code still refers to Glider and we just deleted Glider.

6) Now grab the liquid column **ULiquidColumn** reference.

```
	static
ConstructorHelpers::FObjectFinder<UStaticMesh>MeshAsset3(TEXT("StaticMeshStaticMesh'/Game/UTub
e/ULiquidColumn.ULiquidColumn'"));
	UStaticMesh* Asset3 = MeshAsset3.Object;
```

7a) Rename **LeftSpring** to **LeftColumn** and **RightSpring** to **RightColumn**

```
	LeftColumn = CreateDefaultSubobject<UStaticMeshComponent>("LeftColumn");
	LeftColumn->SetStaticMesh(Asset3);
	LeftColumn->SetRelativeLocation(FVector(-140, 0, 55));
	LeftColumn->SetupAttachment(CapsuleCollider);

	RightColumn = CreateDefaultSubobject<UStaticMeshComponent>("RightColumn");
	RightColumn->SetStaticMesh(Asset3);
	RightColumn->SetRelativeLocation(FVector(140, 0, 55));
	RightColumn->SetupAttachment(CapsuleCollider);
```

7b) and in the header

```
	UPROPERTY(VisibleDefaultsOnly)
		class UStaticMeshComponent* LeftColumn;

	UPROPERTY(VisibleDefaultsOnly)
		class UStaticMeshComponent* RightColumn;
```

8) Now grab the stand **Stand** reference

```
	static
ConstructorHelpers::FObjectFinder<UStaticMesh>MeshAsset4(TEXT("StaticMeshStaticMesh'/Game/UTub
e/Stand.Stand'"));
	UStaticMesh* Asset4 = MeshAsset4.Object;
```

9a) Rename **LeftStop** to **Stand**

```
	Stand = CreateDefaultSubobject<UStaticMeshComponent>("Stand");
	Stand->SetStaticMesh(Asset4);
	Stand->SetRelativeLocation(FVector(-279, 0, 0));
	Stand->SetupAttachment(CapsuleCollider);
```

9b) and fix up the header

```
	UPROPERTY(VisibleDefaultsOnly)
		class UStaticMeshComponent* Stand;
```

10) Finally grab the clamp. Note you have to create a new **MesgAsset5** and **Asset5**

```
static
ConstructorHelpers::FObjectFinder<UStaticMesh>MeshAsset5(TEXT("StaticMeshStaticMeshStaticMesh'
/Game/UTube/Clamp.Clamp'"));
	UStaticMesh* Asset5 = MeshAsset5.Object;
```

11a) Rename **RightStop** to **Clamp** and make sure you use the new **Asset5**.

```
	Clamp = CreateDefaultSubobject<UStaticMeshComponent>("Clamp");
	Clamp->SetStaticMesh(Asset5);
	Clamp->SetRelativeLocation(FVector(279, 0, 0));
	Clamp->SetRelativeRotation(FRotator(0, 180, 0).Quaternion());
	Clamp->SetupAttachment(CapsuleCollider);
```

11b) and fix the header

```
	UPROPERTY(VisibleDefaultsOnly)
		class UStaticMeshComponent* Clamp;
```

Good! So we've replaced the assets with the ones we need, but the above code translates and rotates assets for the airt-rack model; we must return and correct these. But let's move on to get something working.

Delete these lines which refer to Glider and springs

```
gliderStartLocation = Glider->GetRelativeLocation();
leftSpringStartLocation = LeftSpring->GetRelativeLocation();
rightSpringStartLocation = RightSpring->GetRelativeLocation();
```

Stage 5 – Start working on the Code

Write the **Visualization(…)** function, we'll see why this is so a little later on when we've taken a look at the theory.

```
void AMAS22_UTube::Visualization(float deltaTime) {

	FVector scaleVec;
	scaleVec = LeftColumn->GetRelativeScale3D();
	scaleVec.Z = 1.0 + u[0];
	LeftColumn->SetRelativeScale3D(scaleVec);

	scaleVec = RightColumn->GetRelativeScale3D();
	scaleVec.X = 1,0 - u[0];
	RightColumn->SetRelativeScale3D(scaleVec);

}
```

1) Compile. All should be OK.

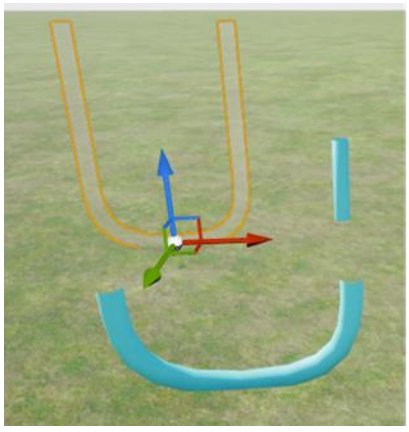
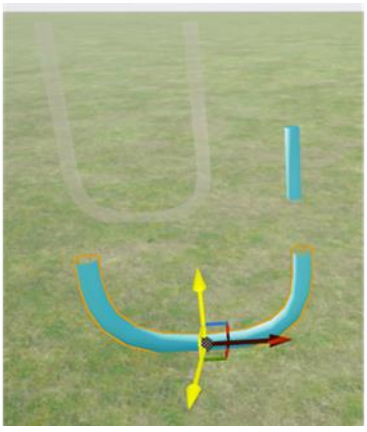
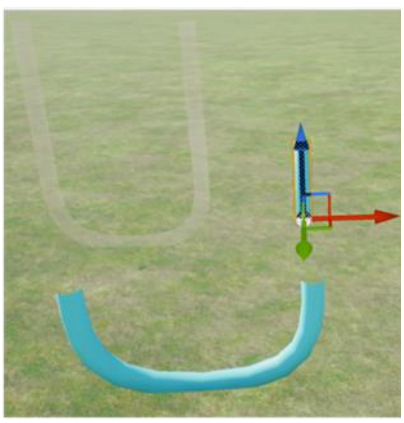
2) In the editor

(i) give a name for this model. How about "UTube" ?
(ii)Set the solver to RKF

3) Run. You will see the liquid columns oscillate. But not quite right. Also, the components need moving to their correct positions, and we need to assign materials to the components.

Stage 6 – Assembling the apparatus components.

You already know how to do this. The diagram below shows where the zero location of each component is located.

The actual tube, **UTube** and the liquid at the bottom **ULiquid** should be located at the same point (0,0,0). The **LeftColumn** and **RightColumn** need shifting in the X and Z directions. The shifts are easy to work out in the **left** viewport. The Stand and Clamp will give you a challenge.

Stage 7a – Coding the Right Hand Side – A Little Theory

Here we need a little theory. We can obtain the ODEs expressing the dynamics of the fluid column through two routes, as usual either the energy route or the force route. I find the force route less abstract.

But let's start along the energy route and assume there is no dissipation. The total fluid length is L and the tube has a constant cross section A. Therefore, the total mass of the column is ρAL. We can think of the situation shown below as taking a column of liquid length z from the left column and adding it to the right column. In other words, we have raised this column a distance z, and it therefore has been given an increase in gravy potential energy

$$(pAz)gz$$

Since energy is conserved, and since there is no dissipation, the total energy is

$$E = \frac{1}{2}(\rho AL)\left[\frac{dz}{dt}\right]^2 + \rho gAz^2$$

Differentiating w.r.t. time we have

$$0 = (\rho AL)\frac{d^2z}{dt^2} + 2\,\rho gAz$$

Cleaning up we have the ODE for SHM

$$\frac{d^2z}{dt^2} = -2\frac{g}{L}z$$

with period

$$T = 2\pi\sqrt{\frac{L}{2g}}$$

This looks like the linearized pendulum solution, but the UTube has the same period as a pendulum of length L/2. Also note that A and ρ do not enter into this expression. Of course, we need not tell our students this; they could find it out by doing some investigations *before the theory is presented.*

We could get the ODEs by taking the force route. First, a simple-minded approach. The force on the water is due to the weight of the displaced column on the right which is

$$2\rho gAz$$

and this force accelerates the entire mass of water ρAL. So we have,

$$\frac{d^2z}{dt^2} = -2\frac{g}{L}z$$

Without thinking about the mechanism of fluid damping in this system at the moment, we can take a sensible approach and make this proportional to the speed squared. So we end up with

$$\frac{d^2z}{dt^2} = -2\frac{g}{L}z - C\left(\frac{dz}{dt}\right)^2$$

To start *coding* this, we split it into two 1st-order ODEs as usual

$$\frac{dz}{dt} = v$$

$$\frac{dv}{dx} = -2\frac{g}{L}z - Cv^2$$

Now we must make choices of our variable names, both for physical variables and physical parameter. Here's a suggestion

Physical Variable	Code name	Physical Parameter	Code Name
z	dispZ	g	gravity
v	velyZ	L	totalLength
		C	frictionCoeff

This coding takes place in **MAS22_UTube.h**

1a) Make sure the rhs class is named correctly, this must be **class rhs_UTube**. Simplest way is to replace the text **AirTrack_MK** with **UTube.**

1b) Do the same replacement in **MAS22_UTube.cpp**. Do a Compile to check if this has worked OK.

2) Let's rename the parameters in the **rhs_UTube(…)** constructor. The constructor is found at the top of the class declaration.

After renaming, the constructor should look like this

```
double gravity, totalLength, frictionCoeff;

rhs_UTube(double _gravity, double _totalLength, double _frictionCoeff) :
	gravity(_gravity), totalLength(_totalLength), frictionCoeff(_frictionCoeff)
{}
```

Let's pause a moment and look at these variable names. Why are some preceded by an underscore ? Well the names inside the function (…..) with the underscores are its *parameters* which will ultimately be filled in when the function is called. The other names, without the underscores, are variables to be used in the code for this class. The compiler expects these to be different. In the constructor, the second line includes terms like `gravity(_gravity)`, bind these two together, so that **gravity = _gravity**;

3) While we are dealing with parameters, move down towards the middle of the file where you will see a section labelled **private:** where more parameters are declared! These are outside of the right hand side class, and these are the ones which will be used in **MAS22_UTube.cpp**. Why do they have a P at the end? Later!

```
private:

	// Parameters ----------------------------------------------------------
	// variables used within the .cpp file. Append P for parameter so that
	// actual names can appear in Editable variables.

	double gravityP, totalLengthP, frictionCoeffP;
```

4a) Now it's time to work on the ODEs which is found in the **operator(…)** function. Use the agreed code names for parameters and physical variables and change this code. Here is what I got.

```
void operator() (double t, double u[], double dudt[]) {
	dispZ = u[0];
	velyZ = u[1];
	dudt[0] = velyZ;
	dudt[1] = -2.0 * (gravity / totalLength) * dispZ - (frictionCoeff) * velyZ;
}
```

4b) Of course you will have to change the declarations of the local variables used in this rhs class; this happens just underneath the above function.

```
private:
	double dispZ;
	double velyZ;

}; // end of rhs Class    -------------
```

Now we have to propagate these changes to the .cpp file.

This coding takes place in **MAS22_UTube.cpp**

5) We have to update references to **rhs_UTube(…)** where the arguments need updating to correspond to the function parameters. So we need the following. Don't forget, it occurs in two places.

```
rightHandSide = new rhs_UTube(gravityP, totalLengthP, frictionCoeffP);
```

Now we have to change the parameter names

6) Change the names in the function **SetModelParameters()**

```
	line.val = gravityP;
	line.name = "gravity";
	modelParameters.Add(line);

	line.val = totalLengthP;
	line.name = "totalLength";
	modelParameters.Add(line);

	line.val = frictionCoeffP;
	line.name = "frictionCoeff";
	modelParameters.Add(line);
```

7) Now update the **UpdateModelParameters()** function

```
	gravityP = modelParameters[INDEX_GRAVY].val;
	totalLengthP = modelParameters[INDEX_TOTALLENGTH].val;
	frictionCoedffP = modelParameters[INDEX_FRICTION].val;
```

8) Note how the index names have been changed, so these need changing in the defines at the top of the code.

```
#define INDEX_GRAVY 0
#define INDEX_TOTALLENGTH 1
#define INDEX_FRICTION 2
```

9) Now update the **GetDefaultParameters()** function

```
void AMAS22_UTube::GetDefaultParameters() {
	gravityP = gravity;
	totalLengthP = totalLength;
	frictionCoeffP = frictionCoeff;
}
```

You will find that Visual Studio *IntelliSense* will indicate that it can't find the variables on the right. That's because they're not declared. So …

10) In the header file **MAS22_UTube.h** change the Editable Variables (the ones that appear in the Editor Details dialogue) to this.

```
	UPROPERTY(EditAnywhere, Config)
		double gravity = 9.8;

	UPROPERTY(EditAnywhere, Config)
		double totalLength = 1.0;

	UPROPERTY(EditAnywhere, Config)
		double frictionCoeff = 0.0;
```

11) Now we need to attack that magic code that creates our Octave script. For the moment, let's suspend this code and return to it later. So let's comment out most of it:

i) Start a block comment by placing **/*** after **logfile.Append(filename);**

ii) End the block comment by placing ***/** after **FFileHelper::**

12) More parameter fixes! This time in the function **ApplyConfig()**.

```
GConfig->GetDouble(TEXT("/Script/MAS22_SciencePark.MAS22_UTube"), TEXT("gravity"), gravity,
configFName);
printf("AMAS22_UTube::ActorInitialize() >>>>>Config File read: gravity set to %f", gravity);

GConfig->GetDouble(TEXT("/Script/MAS22_SciencePark.MAS22_UTube"), TEXT("totalLength"),
totalLength, configFName);
printf("AMAS22_UTube::ActorInitialize() >>>>>Config File read: totalLength set to %f",
totalLength);

GConfig->GetDouble(TEXT("/Script/MAS22_SciencePark.MAS22_UTube"), TEXT("frictionCoeff"),
frictionCoeff, configFName);
printf("AMAS22_UTube::ActorInitialize() >>>>>Config File read: frictionCoeff set to %f",
frictionCoeff);
```

13) At this point the code should Compile, though we're not quite done yet. We have to deal with the Initial Conditions. First in **MAS22_UTube.h** change the declarations from X to Z like this.

```
	UPROPERTY(EditAnywhere)
		double initDispZ = 0.5;

	UPROPERTY(EditAnywhere)
		double initVelyZ = 0;
```

14a) In **MAS22_UTube.cpp** update the variables in **SetModelICs()**

```
icLine.val = initDispZ;
icLine.name = "init DispZ";
modelInitialConditions.Add(icLine);

icLine.val = initVelyZ;
icLine.name = "init VelyZ";
modelInitialConditions.Add(icLine);
```

14b) and also in **UpdateModelICs()**

```
initDispZ = modelInitialConditions[INDEX_INITDISPZ].val;
initVelyZ = modelInitialConditions[INDEX_INITVELYZ].val;
```

and don't forget to change the defines

```
#define INDEX_INITDISPZ 0
#define INDEX_INITVELYZ 1
```

15) Now we must update the variables where we use them, in **InitializeSolver()**

```
u[0] = initDispZ;
u[1] = initVelyZ;
```

16) Time to change the HUD code so it shows **dispZ** and **velyZ.** In fact, it's easier to search and replace all the old variables **dispX** and **velyX** in the **.cpp** file with **dispZ** and **velyZ.** Here's the HUD

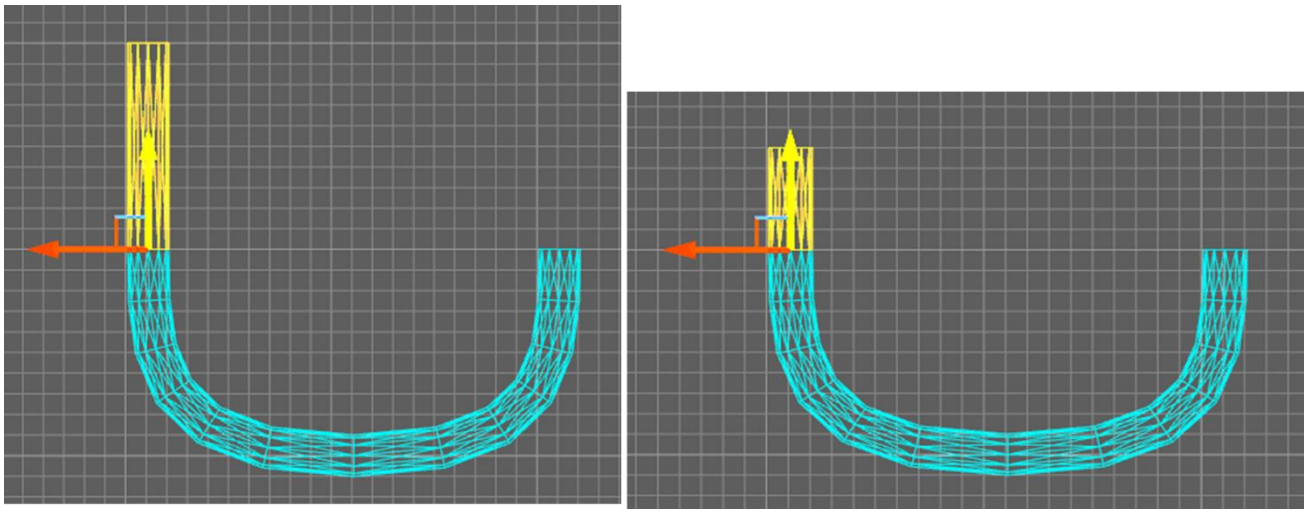
```
line.message = FString::Printf(TEXT("dispX = %f"), dispZ);
line.colour = VARIBLE_COLOR;
line.value = 0;
HUDLines.Add(line);

line.message = FString::Printf(TEXT("velyX = %f"), velyZ);
line.colour = VARIBLE_COLOR;
line.value = 0;
HUDLines.Add(line);
```

17) Try a Compile at this point. All should be OK

Stage 7c – Coding the Visualization

This is where we make the columns move, and we're going to use some magic. The ODE solver gives us **dispZ** which will oscillate around zero. So we could change the *location* of the left and right columns. But there's a better way; we

can change the *scaling* of the columns. The diagram below helps us to understand this. These were made in the Editor front (I think) view.



On the left, a **ULiquidColumn** has been placed manually above the **ULiquid** where it will be located by our code. The default location of the column is shown.

On the right, we have changed the Z-scaling in the Editor Details panel for the column to 0.5, and guess what? The column has shrunk to 0.5 of its original length. This is because the engine applies scaling relative to the default location of the component.

So you will understand how the following visualization code works.

```
void AMAS22_UTube::Visualization(float deltaTime) {

	FVector scaleVec;
	scaleVec = LeftColumn->GetRelativeScale3D();
	scaleVec.Z = 1.0 + dispZ;
	LeftColumn->SetRelativeScale3D(scaleVec);

	scaleVec = RightColumn->GetRelativeScale3D();
	scaleVec.Z = 1.0 - dispZ;
	RightColumn->SetRelativeScale3D(scaleVec);

}
```

Stage 8 – Writing the Configuration file

During development we can change parameters through the Editor Details dialogue, but in a packaged level this is not possible. Changes can be made through the parameter menu system, but default parameters need to be specified by the instructor. This happens through a *config* file, a text file which can be edited by the instructor which the MAS22 engine reads. The config file is provided.

Stage 9 – Fixing the Octave Logging system

More name changing and a little kludge are involved here.

1) In **MAS22_UTube.h** change the **FDataRecord_UTube** struct so it has fields **double time, dispZ, velyZ;**

2) In **MAS22_UTube.cpp** function **EndPlay(…)** make the following changes

(i) replace all **dispX** and **velyX** with **dispZ** and **velyZ**. (Take care to check camel case)

(ii) Modify the section “Get the parameters into Octave” to use our parameters

(iii) Do this also in the line which write the plot **title**

3) Now we're going to do a *kludge*. We're going to mis-appropriate the code which writes the Octave script to plot the theory. Comparing the ODEs we see the following correspondences:

$$m = 1, \qquad k = 2g/L\,, \qquad b = C$$

So insert the following code in the section "calc the theoretical solution"

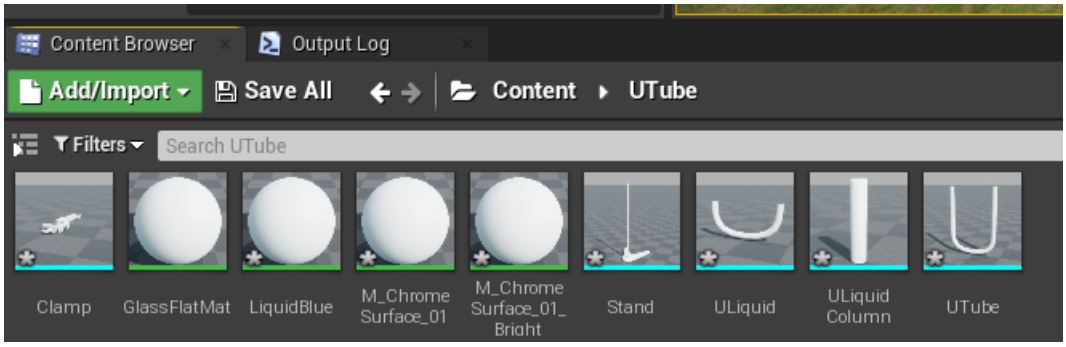
```
        stringToSave.Append("m=1;\n");
        stringToSave.Append("b=frictionCoeff;\n");
        stringToSave.Append("k=2*gravity/totalLength;\n");
```

Of course this is also an *approximation* since friction here is proportional to speed squared and not speed, as in the case of the air-track oscillator. Guess we shall see how good it is.

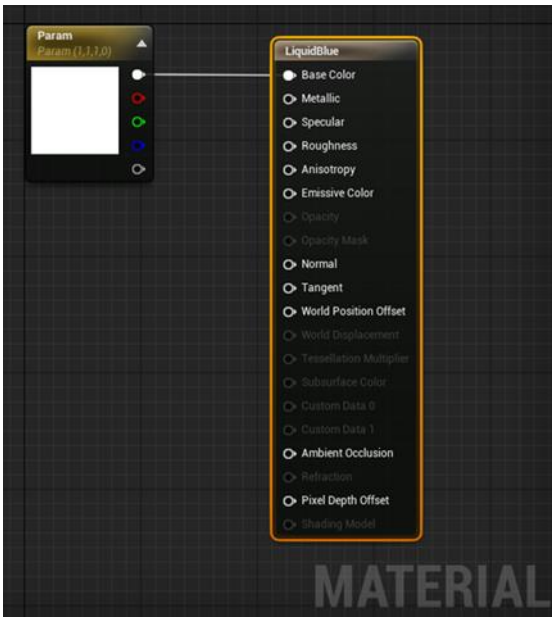
Stage 10 – Assigning Materials to the Static Meshes

We could do this in code, and there are times when we should do this in code. But this is not such a time. Unreal has a powerful Material system including a Material Editor which we shall get to know here. This will form a part of our arsenal of tools to create experiments in our PhysLab. Here we shall explore just a tiny bit of the Material Editor. We need to do two things; first to get the water blue and second to get the tube translucent.

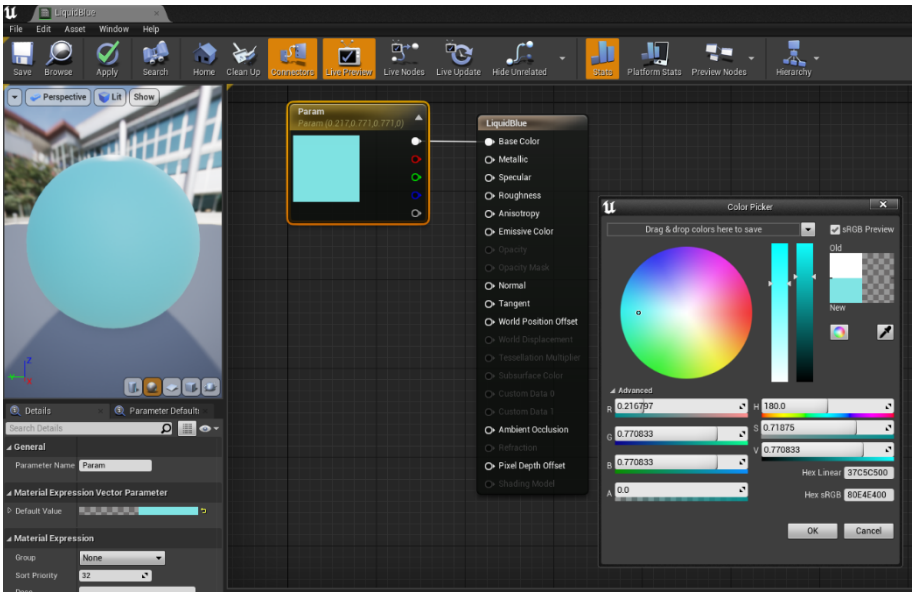
1) Make sure you have imported the .**fbx** objects into the UTube folder in the Content Browser.



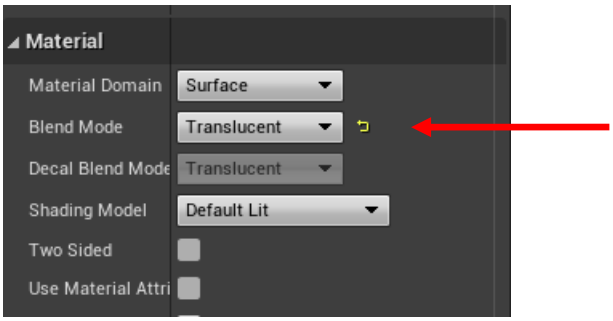
2) Let's get the liquid painted blue. Double left-click on the **LiquidBlue** material sphere to open this in the material editor. Drag the components apart so you get this



3) Double left-click on the Param and select a nice watery blue, like this. Then **Save** (screen top left) and close the LiquidBlue tab (top left). The water columns in your level should now be blue.



4) Now let's assign a material to the UTube. Open **GlassFlatMat** in the Material Editor and separate out the components. Set the **Blend Mode** to **Translucent** (Editor bottom left)



5) Drag a wire out of **Opacity** and select a **Constant** node like this (bottom left) then set it to 0.3 (bottom right) in the Details panel. Save. Your Utube should now be translucent. Play around with its color and opacity until you get something you are happy with.