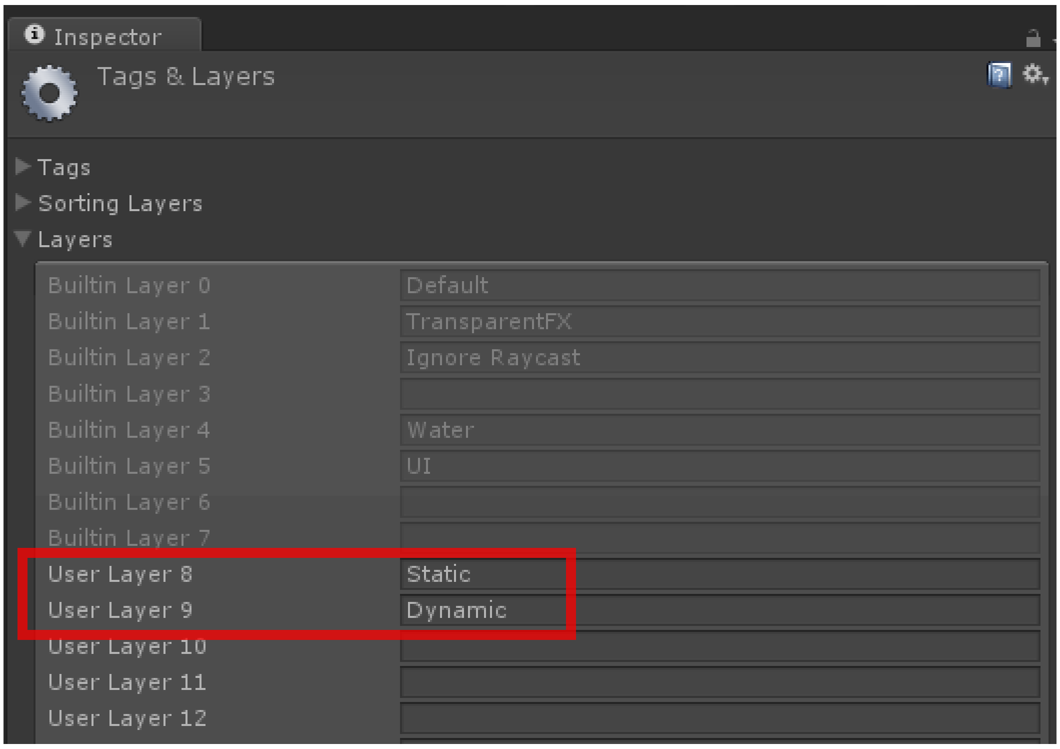
The Force Engine 0.03 Documentation

1 Project Setup

In Project Settings, script execution order is important. TFEngine and TFGUI should execute before other scripts.

The following layers should be reserved:



Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
Builtin Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	UI
Builtin Layer 6	
Builtin Layer 7	
User Layer 8	Static
User Layer 9	Dynamic
User Layer 10	
User Layer 11	
User Layer 12	

2 Scene Setup

The scene should contain a GameObject with TFEngine and TFSurfaceParameters scripts.

For stability, it is recommended not to set friction to infinite value.

The following demo scenes are provided:

- **Demo_basket**: serves as a test for the sphere vs trimesh collider
- **Demo_boxstack**: serves as a test for collision between all colliders
- **Demo_buggy**: demonstrates vehicle simulation
- **Demo_chain**: serves as a test scene for the ball joint
- **Demo_friction**: serves to test the Coulomb friction approximation
- **HelloWorld**: empty scene with just a floor, TFEngine and TFSurfaceParameters to start playing with the engine. The user can drag and drop in this scene prefabs from the Prefabs folder
- **Profile**: serves as a scene for performance profiling

3 GameObject Setup

In order for a GameObject to be part of the simulation, the following behaviours should be attached:

3.1 UnityEngine Collider and Rigidbody

These are used for first pass collision detection, to detect that two GameObjects are potentially colliding.

TFCollider will automatically set the Collider to trigger and the Rigidbody to kinematics.

3.2 TFResetScale

The engine partly supports scaling and nesting; this allows for example easy creation and attaching of boxes of various sizes.

This behaviour will bake the scale into the MeshFilter mesh vertices and into the Collider, then set the scale of the GameObject to {1, 1, 1}.

3.3 MeshFilter and MeshRenderer

Used for rendering

3.4 TFCollider

Similar to UnityEngine Collider. This is used to generate detailed collision information.

3.5 TFRigidbody

Similar to UnityEngine Rigidbody. Control of an object's position through TF physics simulation.

3.6 TFMassBehaviour

This behaviour is used to set the mass of TFRigidbody, according to the TFCollider shape. If final mass is set to zero, then the density is used to compute the mass.

3.7 ShowAutodisabled

This behaviour is used to change the colour of a GameObject, according to if the associated TFRigidbody is sleeping or not.

4 TFTrimesh

It is a triangle mesh collider that uses TFTrimeshData as data.

TFTrimeshData contains all triangles information in an optimized way. It is a ScriptableObject (belongs to the Unity project, not the scene) and can be used by many instances of TFTrimesh.

Current optimization is the use of an octree, where the root is level 0

TFTrimeshData can be constructed from a Mesh. A lot of the processing is done offline.

1. It is first created using **Tools/The Force Engine/Create TFTrimeshData**,
2. Assign a Mesh
3. Set Min Level and Max Level of recursion in the octree
4. Press **Create from Mesh**

Then you can assign the TFTrimeshData to a TFTrimesh in the scene.

5 What is new in version 0.03 ?

Starting from version 0.03, The Force Engine can act as a wrapper over native Open Dynamic Engine library, to benefit from the performances and large set of features of native ODE.

Not all ODE functionalities are currently wrapped.

The wrapping has been done for Windows x86 and x64 platforms. It could in theory easily be done for other platforms as long as you know how to compile ODE and create native unity plugins for these platforms.

6 How to Build Ode.dll ?

1. Download Ode 0.13 https://sourceforge.net/projects/opende/files/ODE/
2. Unzip
3. Unzip provided Ode_Delta.zip
4. Copy content of Ode_Delta into your ODE 0.13 folder and overwrite existing files
5. Open Ode.sln with Visual Studio 2015
6. Build

7 Stability

To improve the stability of the simulation:

- Decrease fixed time steps
- Increase number of iterations of the solver
- Avoid infinite friction in TFSurfaceParameters

8 Performance

Performance has improved and memory leakage has reduced in **The Force Engine 0.03**.

To measure performance, it is recommended to build a Standalone app.

9 How to create a new TFCollider (The Force Engine, The Force Engine PRO)

To create a new TFCollider, you should derive from TFCollider.

See provided TFCapsuleCollider source code as an example.

Methods worth overriding:

9.1 InitAfterResetScales()

Called after the scale has been reset. Good time to save the final world size of the collider and use it to adjust the mass of the attached TFRigidbody.

9.2 Bool CanHandleCollision(TFCollider other)

Managed implementation only.

This method should return true if this TFCollider knows how to handle collision against other TFCollider. For collision to take place between two TFColliders, at least one TFCollider should know how to collide against the other one.

9.3 Collide(TFCollider other, List<TFContactGeom> contactGeoms)

Managed Implementation only.

This method should generate detailed contact information for collision with other TFCollider.

10 How to create a new TFJoint (The Force Engine, The Force Engine PRO)

To create a new TFJoint, you should derive from TFJoint.

See provided TFHinge2 source code as an example.

Methods to override:

10.1 InitAfterResetScales()

Called after the scale has been reset. Good time to save anchor positions.

10.2 getInfo1(ref Info1 info), getInfo2(ArraySegment<Constraint> J)

Please have a look at article *How to make new joints in ODE* from Russell Smith, 2002.

http://www.ode.org/joints.pdf

11 How to translate a C++ file into C#

1. Change your source file extension from cpp to cs
2. Insert the file into your Unity project or Visual Studio solution
3. Fix errors
4. Repeat previous step until all errors fixed
5. Done !

12 Copyrights

The Force Engine inspired by

http://www.ode.org

http://www.ode.org/ode-license.html

13 Support

support@jstarlab.com