Hayate 2.0 Manual and Documentation

Hayate version 2.0

Document version 1.0

Hayate 2.0 is a new and improved version of previous Hayate versions. The source code has been completely rewritten and optimized to ensure maximum performance. Additionally a bunch of new features have been added.

Even though Hayate is pretty self-explanatory there are some mistakes to be made that can hopefully be solved with this document. If have a problem with Hayate, please contact me and/or check for the latest version of this document here.

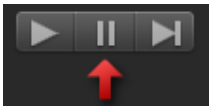
m4xproud@gmail.com

Content

General hint:

If the Hayate does not work in edit mode, make sure that the turbulence is enabled, ‘amplitude’ is big enough to make a visual change and the particle system is set to world simulation space. Also a turbulence type must be chosen before any effect will occur.

If everything is set up press the pause button twice to initiate the turbulence calculation.



1.0 Manual

1. Debugging

Debugging is a very important in order to create good looking and efficient results. That is the reason why “Hayate 2.0” comes with enhanced debugging features. It is now possible to create a 3D flow field that helps visualizing the evolution of the turbulence.

To start debugging you will have to enable the turbulence first. The default settings of the turbulence field are set to : Field size (5, 0, 5) & Step size (0.1, 0.15, 0.1).

Field size is the size of the turbulence debugging area in world units. If Field size is set to 5 on the x-axis it will range from -5 to +5.

Step size is the amount of points drawn to create the debug field. The smaller this value the more points will be drawn.

Troubleshooting:

-Be careful not to use too small values because it could freeze Unity3D.

-If you don’t see anything make sure that at least one axis (Turbulence settings per axis) is set to any turbulence type.

-Press the pause button twice in order to start turbulence calculations.

2. Turbulence settings

Turbulence settings per axis:
It is now possible to use a different turbulent type per axis. The different types are:

None
Sine
Cosine
Perlin
Precalculated Texture
Audio (precalculated)

The calculated turbulence per particle can be assigned to the particle’s velocity or position. It is also possible to use the absolute values or relative values. This is useful to change the shape of the turbulence field. The turbulence types are sorted by performance cost.

Velocity & **Relative**: Particle velocity += Calculated Turbulence

Velocity & **Absolute**: Particle velocity = Calculated Turbulence

Position & **Relative**: Particle position += Calculated Turbulence

Troubleshooting:

-In order to see turbulence of Velocity & Absolute you will have to use much bigger Amplitude values. **Absolute amplitude * 100 ≈ Relative amplitude**.

Turbulence variables:

Amplitude:

The Amplitude represents the strength of the Turbulence. The Higher this value is, the faster the particles will move.

Frequency:

This value clinches (smaller) or stretches (bigger) the turbulence field.

Offset:

The offset shifts the turbulence field in negative or positive direction.

Offset speed:

Offset speed is useful to change the evolution of the turbulence field. Use this value to make particles swirl around. The 2 buttons below allow you to lock the position of the offset to the GameObject or Randomize the Offset on Start().

Every Variable can be changed independently and per axis.

Troubleshooting:

-Amplitude and Frequency must not be 0. Disable turbulence instead. This also saves performance.

-Locking the Offset always locks to world space = local space Vector3.zero.

Texture turbulence settings:

These settings are only visible if at least one turbulence type is set to 'Precalculated texture'.

'Precalculated texture' uses a texture to create a turbulence field. Each color channel represents an axis (**R** = X, **G** = Y, **B** = Z, **A** = Lifetimeloss). Hayate's initial turbulence uses values between -1 and 1 which is then multiplied by Amplitude. The colors of the texture are remapped to fit those values: Black causes the particles to move into negative direction, white in positive direction. **50% grey** (0-256 = 128 or 0.5f in a color channel) equals 0 in turbulence strength.

Important: If you use the alpha channel of the texture to remove particles after they are spawed make sure to insert a custom color into the color channel. If you use the PNG file format **0% alpha** will trigger Photoshop to insert white into the color channel (Gimp will

insert black). Therefore it is better to use the **Targa** format because you have the possibility to create your own alpha mask while having full control over your other color channels.

Setup:

Just drag and drop any texture from the project view into the designated texture slot and hit 'Update turbulence'.

Note: You can only use read/write enabled textures. You can change those settings in the import menu of the texture.

If you want to use the alpha channel, simply click on the designated button. The Threshold can be adjusted to prevent spawned particles from popping up and disappearing. However, this might not always be possible. To fix this you will have to spawn particles with 100% alpha and then adjust the color over lifetime curve to match your effect. Please look at scene 5 of the demo.

Troubleshooting:

-To align the turbulence with a setting you have to change the Offset per axis to 'Texture width / 2'. Example: If the Texture is 512 pixel wide, change the emitter offset to 256. The Pixel are mapped to 1:100 if Frequency is 1. That means if Frequency = 100 each pixel has the size of 1 world unit. This will create a jagged turbulence field.

Audio turbulence settings:

These settings are only visible if at least one turbulence type is set to 'Precalculated texture'.

Using this turbulence type creates a turbulence field using the samples from a supplied AudioClip. Drag and drop an AudioClip into the designated slot and hit 'Update turbulence'.

Trouble shooting:

-Before you can use the AudioClip you have to set the Clip's 'Load type' import settings to 'Decompress on Load".

-If the turbulence is very small or not existing, change the 'At smaple' value before updating again to make sure the AudioClip is not completely silent at " At sample".

3. **Collision settings:**
   If there is a collider attached to the ParticleSystem's GameObject you can emit a certain amount of particles.

4. **Transform particles**
   If you want to use a transform instead of a billboard or a mesh as a particle you can do so now by enabling this option. Simply drag and drop a prefab into the designated slot and you

are good to go. Two more options allow you to tell Hayate what to do with the transform when it's dying: Kill it ('delete') or detach it and kill it after X seconds ('Time until death'). It is also possible to rotate the transform so that it always looks at its flight direction.

5. **Follow point**

To make all particles follow a single point in world space you can tell Hayate to follow a Transform or input the position yourself by supplying a Vector3. 'Strength' is used to accelerate particles.
If you want to make the particles to close in on their target simply limit their velocity over time, using Shuriken settings.

Troubleshooting:

-If the particles are not moving towards the point but somewhere else, make sure that the ParticleSystem's simulation space is set to 'World'.

6. **Mesh target**

This option allows you to move the particles towards a mesh of your choice. Add a GameObject with a MeshFilter component and select a approximation method:

By Distance will accelerate particles the closer they get.
By Time will slow down particles the closer they get.
Physical will keep the particles inertia.

Speed is used to calculate the speed of the particles. It has different properties for each approximation method:

By Distance: The higher the faster the particles will 'snap' to their position.
By Time: The Higher this value is, the closer they will come their position.
If it's too small it might not have any effect at all.

Physical: The higher the faster. In contrast to 'By distance' the particles inertia will be kept and the particles will pass their position and return.

Subdividing

For mesh target to work you need a vertex per particle. If you want more particles you can use Hayate's subdivision options to generate more vertices. This is done by telling Hayate the desired size of the smallest triangle. Hayate takes care and adds vertices where they're needed to make a more even distribution of particles all over the mesh.

Recommended workflow:

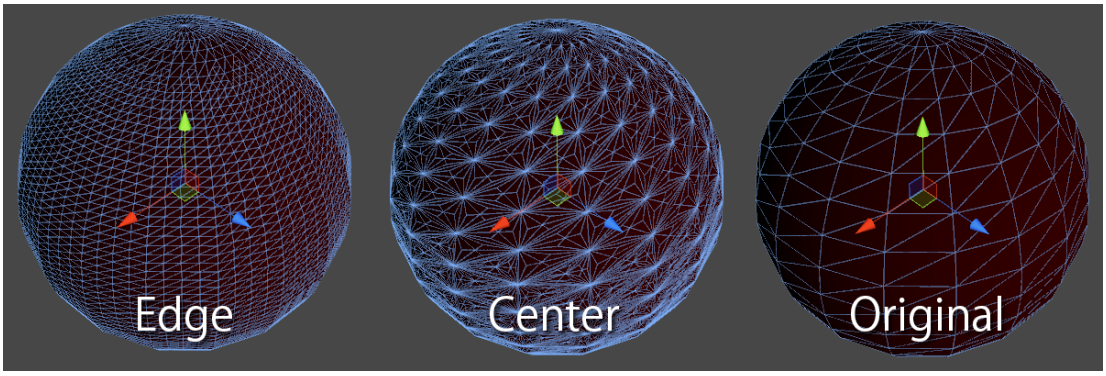
1. Get mesh info.
    a. Check the Console output.
    b. Subdivide your mesh using the 'smallest triangle' in your current mesh.
    c. Check the Console output.

Hayate will tell you how many perticles can be used.

2. Closing in.
    a. If there are too many reset the mesh. Then increase the 'smallest triangle'-value.

b. If there are too many particles/ vertices, reset the mesh and decrease the ‘smallest triangle’ -value.

To guarantee the best result possible try to change the subdivision option from center to edge or vice versa. Edge subdivision will add 3 vertices on the edges of the triangle to create 3 new triangles. Center subdivision will do the same but only adds one vertex in the center.



7. Threading options

This is used to minimize Hayate’s impact on the overall game performance. If Hayate runs ‘Threads’-amount of threads. If not altered it will be the amount of processors detected by the Application. Example: 4 physical cores on a CPU with hyperthreding will use 8 Threads. If safe mode is enabled particle calculations will be 100% on time when the frame is rendering but this might thwart the efficiency of Threading.

On some devices Threading helped to improve the frame rate by up to 150%.
Threading doesn’t work in conjunction with Mesh targets and Transform particles.

2.0 Public Variables

1. Enums / Main variables
    a. **UseCalculationMethodX** (type of **CalculationMethod**)
    b. **UseCalculationMethodY** (type of **CalculationMethod**)
    c. **UseCalculationMethodZ** (type of **CalculationMethod**)
    d. **UseRelativeOrAbsoluteValues** (type of **TurbulenceType**)
    e. **centerDivision** (type of **DivisionType**)
    f. **AssignTurbulenceTo** (type of **AssignTo**)
    g. **meshFollow** (type of **MeshFollow**)
    h. **bool** UseTurbulence
    i. **int** MaxParticles (default 100.000)
2. Debugging
    a. **bool** drawTurbulenceField
    b. **Vector3** fieldSize
    c. **Vector3** stepSize
    d. **Color** debugColor
3. Texture turbulence
    a. **Texture2D** Turbulence
    b. **bool** useAlphaMask
    c. **float** threshold
4. Audio turbulence
    a. **AudioClip** audioClip
    b. **int** amountOfSamples
    c. **int** atSample
5. Global settings
    a. **Vector3** Offset
    b. **Vector3** OffsetSpeed
    c. **bool** lockOffsetToEmitterPosition
    d. **bool** randomizeOffsetAtStart
    e. **Vector2** randomOffsetRange
6. Collision options
    a. **bool** burstOnCollision
    b. **int** burstNum
7. Transform particles
    a. **bool** useTransformParticle
    b. **bool** detachTransformParticleAfterParticleDeath
    c. **float** detachedObjectDestructionTimeAfter
    d. **bool** transformParticleLookTowardsFlightDirection
    e. **GameObject** transformParticle
8. Follow point
    a. **bool** followPoint
    b. **Transform** followTransform
    c. **Vector3** followPosition
    d. **float** followStrength
9. Mesh target

    a. **bool** moveToMesh
    b. **GameObject** meshTarget
    c. **float** particleSpeedToMesh
    d. **float** smallestTriangle

10. Threading
    a. **bool** useThreading
    b. **bool** safeMode
    c. **int** numThreads

Hayate methods

Void UpdateTurbulence() – Updates the texture turbulence at runtime.

Void calculateAudioTurbulence() – Updates the Audio turbulence at runtime.

Vector3 GetTurbulence(Vector3 Position) – Returns the turbulence at position.

3.0 Tutorials

Tutorials will be added later on. If you have suggestions please tell me: m4xproud@gmail.com

4.0 Other information

Music used in the demo:

Larmes by **Silence**

Usage free of charge under Creative Commons http://www.jamendo.com/de/track/5346/larmes

Skinned mesh target/emitter is not supported due to extreme performance cost of recalculating the vertex position. This feature might be added in the future if performance is somewhat reasonable.