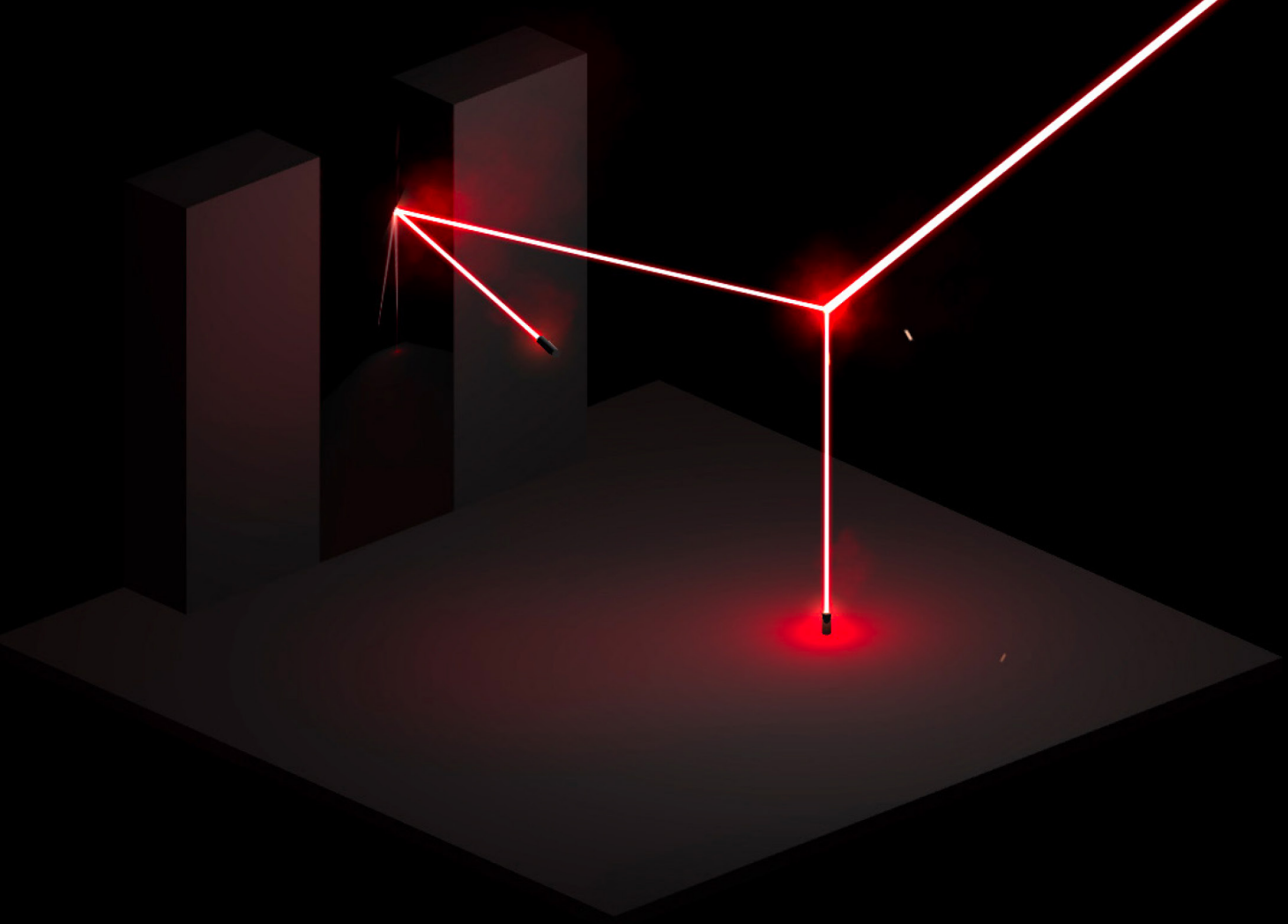
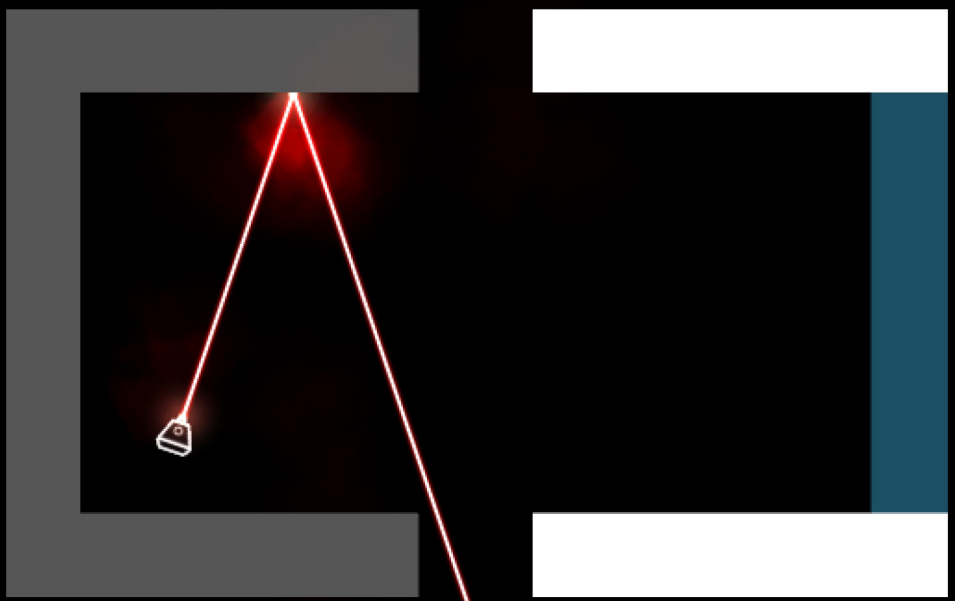


BEAMTIME

A DYNAMIC LASER BEAM SUITE



EAGER AMOEBA®

CONTENTS

INTRO

Hello and thank-you for purchasing Beamtime!

Beamtime is a pair of scripts aimed at creating interactive and realistic laser beams within Unity 4.5+.
Each laser object is identified by the laser script then it's processed by the laser system script in order to aid performance.
Beamtime is also a term in particle physics. It's shorthand for the time allocated to a researcher to research particle beams from a particular source
We named our asset Beamtime after it was suggested to us by arcosapphire on reddit. We liked the term and it's meaning so it stuck!
With this asset, you can have your own beams to research.
Enjoy your Beamtime!

Package Contains

-Example scenes
-Sample laser shader, effects and texture
-Laser script
-Laser System script
-Billboard script
-DestroyFX script
-Prefabs
    -FX objects
    -Laser object
    -Laser system
-README.txt

Limitations

The product is limited by the complexity of your simulation, as the calculations are quite complex you need to take care to not overload the system with too many objects (we've tested this system with 20+ lasers and errors/performance hits show when these lasers all collide with each other, if this can be mitigated we recommend you doing so).
The effects we've provided are incredibly basic, while these are usable we recommend creating your own shaders for emission and global illumination. These can be found on the asset store for next to nothing so this shouldn't be an issue.
With regards to multi-player, due to the systems involved for network communication being complex and specialised, we have only tested this locally. We recommend test and integrating this asset carefully into your own networking/multi-player solution. There's information on how we would recommend doing this is the multi-player section.

Version differences

We have made the asset compatible with Unity 4.5+
All versions are almost identical bar the effects provided.
In Unity 4, the effects are far more basic than those found in 5 due to the integration of the advanced particle system. The 2D example scene is also non-existent in this version.

Quick-start

1. Place the LaserSystem prefab into your scene, this is used to manage the system.
2. Place the Laser prefab into your scene at the position you want a laser.
3. Play and enjoy!

FEATURES

This is a list of features and functions this asset contains.

Block Laser Type

This allows the laser(s) to block other lasers as opposed to intersection. This laser will only intersect with other block lasers.

Charge Laser Type

This allows the laser(s) to charge other lasers, increasing the power of any it comes into contact with.

Intersection

Allows the laser(s) to intersect with other lasers, ending this laser at the point of intersection and reflecting if this option is enabled (see reflection section).

Hit

Allows the laser(s) to hit with colliders and end at the point of hit.

Reflection

This option makes the laser(s), reflect from hits or intersections. This is done be creating a child rotated toward the reflection angle. For hits, this angle will be reflected along the normal of the hit. For intersection, this will be the cross-section of both lasers that created the child.

Material Reflection

This allows hit reflection to be enabled by materials. This will work for mesh collider and shape colliders. For mesh colliders it will look at the mesh and check if the hit point contains the material. For shape colliders, it will simple check if the renderer contains the material in it's material array due to the limitation of shape colliders. This also allows you to set the power increment and hit fx per material.

Passthrough Material

This allows the laser(s) to not collide with certain materials, particularly useful for glass. This also allows you to set the power increment per material. Does not create hit fx.

Power

This is the power of a laser, you can set the increment value for certain conditions as-well as a max level and starting level.

Object Pooling

The system includes a simple object pooling script, for more information see page 14 'simpleGOPool'.

EXPLANATION

Laser System

The laser system works by scanning for all gameobjects with the laser script on them, then applying the effects as configured in the laser system script.
Options have been included for length, intersection, hit, FX gameobjects, power and more!

In order for the script to work, you are required to create a laser object with the laser script attached. This is used to detect laser objects and is required when creating their reflection/ intersection.

Laser Object

The laser object is used to let the system know where a laser is and if any local variables are required.

Guideline

You need to attach the laser script to your object, a line renderer is also recommended but can be circumvented with your own scripts.
Your own scripts can use the public variables that contain passed information from the system (more info in the scripts section).

FX Object

FX objects can be found on each script as:
Start cap
End cap
Intersect object
Hit object

These are objects that are created when certain conditions are met (and follow the position of these conditions until they are not met).

Guideline

Start caps are created at the start of the laser whilst it's enabled.
End caps are created at the end of a laser when it has reached the maximum length it possible can (so if the laser is interrupted by an intersection or hit, these will not spawn).
Intersection objects are spawned when two lasers intersect and follow the intersection point.
Hit objects are spawned when a laser hits an object and follow the hit point.

When creating the FX Object, we recommend using the destroyFX script to improve these. Largely it doesn't matter what you create for these as long as you are happy with the end result.

USAGE GUIDELINES

Namespace

In order to access any classes from this asset you will need to include the namespace at the top of your own scripts. Like so:

```
using eageramoeba.Lasers;
```

Turn On/Off System

In order to turn the system on, you will need to access the lasersystem script via the static instance and set the correct variable to true or false and reset the system. Like so:

```
laserSystem.instance.onSys = false;
laserSystem.instance.reset();
```

Turn On/Off Laser

In order to turn a laser on or off, you will need to access the laser script and set the correct variable to true or false. True for on and false for off, like so:

```
[YOUR LASER OBJECT].GetComponent<laser>().onM = false;
```

2D/3D

The system has two modes, 2D and 3D. 2D is only used when the 2D mode checkbox is ticked.
The difference between the two modes is how hits and intersections are handled.
In 2D modes spriterenderer materials are referenced along with meshrenderers when using hit passthrough/reflection materials, whereas in 3D only meshrenderers are referenced.
In 2D mode only 2D colliders are used and in 3D modes only 3D colliders are used.
In 2D mode passthrough materials are disabled due to the way 2D colliders work, use the layermask instead to get the same effect.
Lastly, in 3D mode the laser is emitted from the forward of the object. In 2D, the up is used.

Material Properties

When using reflect materials or passthrough in 3D, it's recommended you use a mesh collider with materials mapped to the mesh being used. If you do, the location of the material will be scanned and used for this.
If you don't the system will just check if the material array contains the material.
Please note, material scan is quite heavy. It is recommended you use the tag mode for reflect and a hitmask with "ignore raycast" off for passthrough to increase performance.

Line Renderer

When using the laser system line renderer system, ensure you're using world space and the line renderer is on the same object as the laser script.

Multi-player

We have not built in multi-player support out-of-the-box due to the wide range of solutions and the complexity involved.
In the future we may look at producing our own multi-player solution, this will probably be in a version 2 of Beamtime.

Here are a two solution ideas to go with as a starting point:
1. Peer-to-peer, where original lasers are sent and reflected/hit lasers are simulated on the client.
2. Create a custom client to process only laser effects via information sent from the server.

Scene Creation

When creating a scene, you need to ensure you have an object in place with the LaserSystem script attached. Make sure there is only one instance of the script in place or you will encounter bugs.

If you do not place the system script in the scene, the laser system will not be activated as this script powers the system.

Laser Object Creation/Deletion

In order to create a laser object, attach the laser script and a line render (if using the line renderer part of the system).

Configure as necessary.

If using the pooling system, remember to reset the laser by calling the following function on it:

```
resetVals();
```

In order to destroy a laser properly, please call the following function on the laser:

```
destroyLas();
```

This function will automatically reset the laser.

FX Object Creation/Deletion

FX objects can be anything, as long as they suit the art style of your game (or don't your choice).

However, in order to make them work a little better with the system, make sure to only nest particle systems in the object once so the system can modify their properties..
As the system is built to make sure particle systems are destroyed smoothly.
When the object is destroyed, all systems emission values are set to zero and the objects are destroyed on a timer. This means particles finish their lifetime as opposed to disappearing suddenly.

We've also created the destroy FX script to help make the destruction of these objects smoother, place any object that is required to be destroyed instantly when the prefab is to be destroyed in the "Light Obj" field. If pooling is enabled, this object will just be disabled and re-enabled accordingly.

If using the pooling system, remember to reset the fx object by calling the following function on it:

```
startFX();
```

In order to destroy a fx object properly, please call the following function on the fx object:

```
DestroyFX();
```

This function will automatically reset the laser.

Laser Materials

Creating effective lasers is an integral part of game immersion.
When creating a shader/material for your lasers, make sure backface culling is off if you use reflections. As by default, line-renderers will only render on one side of the line due to backfaces not being rendered.

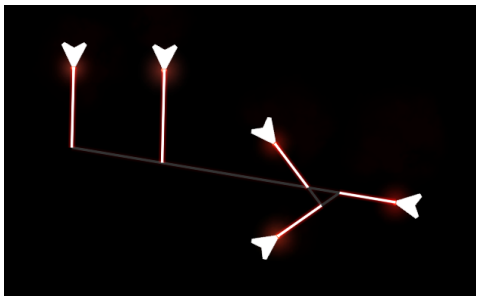
We've provided a sample material/shader for lasers as an example.

INFINITE LOOP PROTECTION

Some situations (where intersection is enabled) can produce infinite loops in the system, it's unavoidable in certain situations.
To avoid anomalies, an element of infinite loop protection has been added, this cannot be turned off unlike 'self infinite loop protection' (see page 9).
There are several modes, we personally recommend flicker as it reflects what these lasers might do if possible in the real world. Please note, all modes are purely cosmetic, for all intents and purposes any laser that has a possible intersection with any infinite loop will be cut short at the maximum intersection/hit point.
The modes and their effects are:

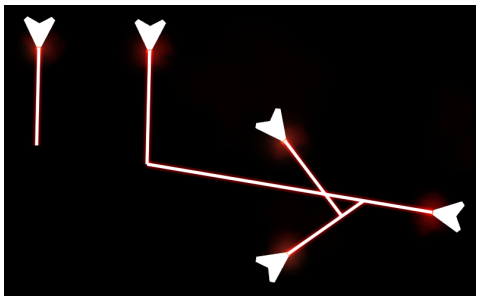
Flicker

Flicker between the first and last intersection.



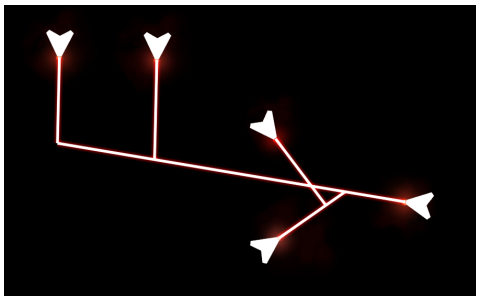
Specify

Specify the position you want to be used, if this position is over the max, the last will be used. In this example, I have used the 3rd position and set the Specify Position field to 2.



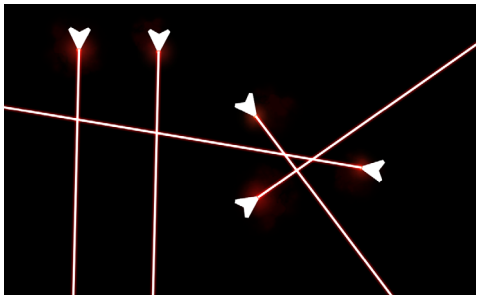
Last

User the last viable intersection.



Ignore

Ignores the protection (sort of) and makes any infinite looping lasers use their max possible length.

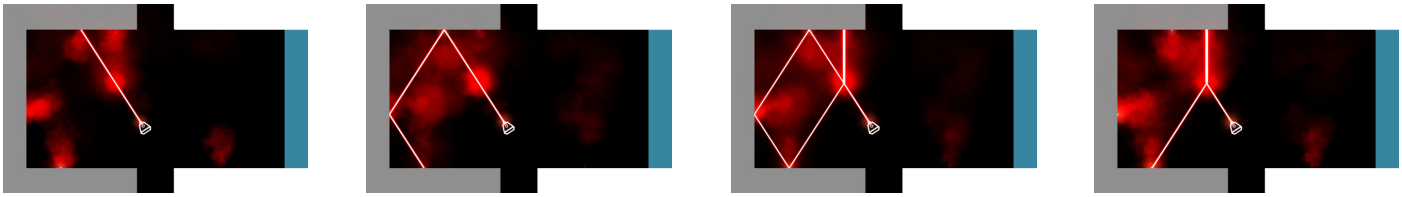


SELF INFINITE LOOP PROTECTION

Some situations can produce self infinite loops in the system. Take the following level layout:

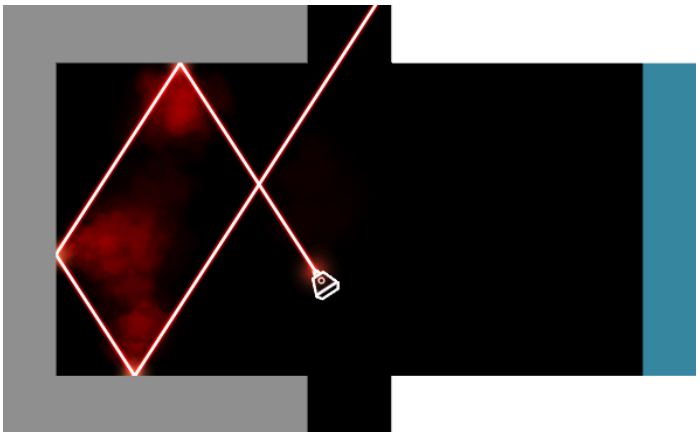
The grey area reflects the laser, the blue area lets it pass through and the white area blocks it. The ship turns on it's laser when the space bar is pressed. (This layout is provided in the asset as the 2D example scene)

When self infinite loop protection is off, the following sequence happens multiple times per second when the ship's laser is activated:



The laser rebounds off of all three walls and back into itself, casing the creation laser to end before it created the other laser objects. This means these objects are destroyed and the laser then rebounds off of the walls again.
This could create some interesting mechanics and be a non-issue providing your pooling system is capable of handling the creation of extra objects (more on that in pooling section.

To this end, we have provided a self infinite loop protection mode, when activated it will allow children of lasers to pass through the origin lasers (and those created from them) without interacting with them. Like so:



To help facilitate custom effects with this eventuality, such as a "crossing the streams" effect, we've provided variables in the laser object script to log the origin laser for each child laser (as well as is the laser is a child).

SCRIPTS

This asset contains 4 scripts (plus editor companions), when modifying these scripts, it's recommended you make copies of all four and use those. As updates can override your changes. These scripts are as follows:

billboard

This is a simple script for creating a billboard effect with 2D sprites/textures/images.

DestroyFX

This is a simple script that will help enhance when FX objects are destroyed/created, this script is placed on the fx object and is accessed by the script laser system when an effected needs to be destroyed. It executes DestroyFX() when a fx object is supposed to be destroyed (immediately and before the delay).
It also executes startFX() when the FX object is created via pooling.

Editor Display	Public Variable	Description
Light Obj	`lightObj`	This is an object that is destroyed/disabled immediately as opposed to the delay that you can set via laser system. This is particularly useful for lights.

Laser

This is the laser script, used to indicate this object is a laser and set local variables for this laser. When looking at each variable with a toggle (checkbox) on it's left, this is a local variable. Tick this box to NOT use the global variable for this laser. In combination with the general global option, this is a powerful way to choose which local variables you wish to use.

Editor Display	Public Variable	Description
On?	`onM`	This is an object that is destroyed immediately as opposed to the delay that you can set via laser system. This is particularly useful for lights.
Use Global	`useGlobal`	Set whether this object uses local or global variables
Enable Block	`blockType`	Make this laser a block type
Enable Charge	`chargeType`	Make this laser a charge type
Local Variables		These are the localy assigned variables, enabled when user global is off. All variables that are local require the tick-box/toggle to the left of each to be checked.
Laser GameObject	`localLaserObj`	The laser gameobject, instantiated during reflection.
Length	`length`	The length of the laser.
Intersect	`intersectE`	Allows lasers to intersect with this laser
Reflection Intersect	`reflectE`	Makes this laser reflect upon intersect/hit
Reflect Remaining Length	`reflLenRemE`	When reflecting, use the remaining length of the laser before the point of intersect or hit
Hit	`hitE`	Allow this laser to hit with colliders
Hit Mask	`hitLayerM`	This layermask on raycashits for this laser
Hit Reflect	`hitReflectE`	Reflect when this laser hits.

Hit Scan Mode	`hitScanMode`	Sets whether to use tags or materials during the hit scan, applies to reflection.

Hit Reflect Materials	`hitRefMatE`
	
	`reflMatsE`
	`reflMatObj`
	`reflMatInc`
		Allows this laser to hit reflect by material, click add new to add a new entry and delete to delete existing entries. Each entry has three values, the 1st is the material, the 2nd is the power increment and the 3rd is the hit fx object that's used when hitting this material.

Hit Reflect Tags	`reflTag`
	`allowedTagObj`
	`reflTagInc`
		As above, but with tags. This is recommended over material due to increasing performance and reducing the amount of garbage.

Passthrough Material	`passRefMatE`
	
	`passedMatsE`
	`pasMatInc`
		(Disabled in 2D or tag mode, recommend you use a hitmask generally) Allows this laser to passthrough hit colliders when colliding with a certain material. Click add new to add a new entry and delete to delete existing entries. Each entry has two values, the first is the material and the second is the power increment.

Set Line Renderer Width	`lasLineREff`	Sets wether this laser will modify it's line renderer width. This is calculated by line renderer base width multiplied by power.
Line Renderer Base Width	`lrWidth`	The base width used for the line renderer.
Starting Power	`powerStart`	The starting power of this laser.
Power Cap	`powerCap`	The maximum power of this laser.
Reflection Power Increase	`reflInc`	The increment that the power will increase by when reflecting.
Charge Power Increase	`chargeInc`	The increment the other lasers power will increase by when charging.
Start Cap	`localStartCapObj`	The start fx object for this laser.
End Cap	`localEndCapObj`	The end fx object for this laser.
Hit FX	`localHitObj`	The hit fx object for this laser.
Intersection FX	`localIntSctObj`	The intersection fx object for this laser.
Passed Values		These are values passed to the laser that the laser system has calculated for this laser.
End Position	`intPos`	The position this laser ends at. Effected by hits and intersections.
Is Child?	`isChild`	Is this laser a child of another laser?
Group	`groupId`	The group this laser is in, defined by the ids of each object contributing to the creation of this laser.
Intersection Laser	`intSctGo`	The laser object that has intersected with this laser.
Hit GameObject	`hitGO`	The object this laser has hit.
Hit Distance	`hitDist`	The distance to the hit.
Child	`directChild`	The child of this laser object.
Parent 1	`parent1`	The first parent of this laser (if a reflected laser).
Parent 2	`parent2`	The second parent of this laser (if a reflected laser).
Origin 1	`origin1`	The first origin object of the laser, this will never be a child laser.

Origin 2	`origin2`	The second origin object of the laser, see above.
Current Power Level	`curPower`	The current power level.
Other Variables		
	`intSctA`	All possible intersection points for this laser.
	`sPos`	This lasers position in intSctA arrays.
	`hitMat`	The current hit material.
	`hitNorm`	The current hit normal.
	`shrtA`	The shortest possible intersection in the intersection array.
	`infL`	Is this laser part of an infinite loop?
	`hasHit`	Has this laser hit something?
	`hasInt`	Has this laser intersected?

Laser System

This is the laser system script, responsible for processing all laser objects and providing/setting global variables.

Editor Display	Public Variable	Description
Global Options		Options used by lasers when set to global.
System On?	`onSys`	Controls whether the system is on or off.
Enable Object Pooling?	`enablePool`	Enable basic pooling, increases performance
Cleanup Interval	`nextSc`	(Seconds) Interval the cleanup script runs, useful to remove orphaned FX objects or lasers. The larger the value, the less GC overhead.

Infinite Loop Protection Mode	`infLM (int)`
	`0 = Flicker`
	`1 = Specify`
	`2 = Last`
	`3 = Ignore`
		Specify what the infinite loop protection will do.
		Flicker - Flicker between the first and last intersect.
		Specify - (see below)
		Last - Last viable intersection
		Ignore - Sets all infinite looping lasers to their max length.

Specify Position		Set the maximum position of the infinite loop protection (max position referring to the furthest intersect from origin). The max position will always be used unless the number of viable intersects is less than this value. It will then revert to the last viable intersect. Set to 0 for the first viable intersect.
Self Infinite Loop Protection	`infProt`	If checked, this protects the system from self infinite loops. When a laser intersects with one of it's origin lasers, the system ignores it. Recommended on unless levels are specifically designed to avoid this.
Control Laser On/ Off	`controlSwitch`	If checked, allows the system to control the lasers on/ off values. Helps when lasers are created and they are intiially facing the wrong direction.
2d Mode?	`mode2D`	If checked, system will execute code for 2D games. If unchecked, system will execute code for 3D games.
2D Hit Dist Spawn	`hit2DDistance`	Spawn spacing of new lasers reflecting from a 2D hit, required to be over 0 or bugs will occur.
Laser GameObject	`laserObj`	The global laser gameobject, instantiated during reflection
Intersect Distance	`minIntDist`	The distance between closest points of two lines before an intersection is logged. Recommended to be between 0.08 and 0.2
Update Line Renderers	`updateLrPos`	Set the position of line renderers attached to laser objects, based up start and calculated end positions.
Set Line Renderer Width	`lasLineREff`	Sets wether this laser will modify it's line renderer width. This is calculated by line renderer base width multiplied by power.
Line Renderer Base Width	`lrWidth`	The base width used for the line renderer.
Length	`rayL`	The global length of laser(s).
Intersect	`intersect`	Allows lasers to intersect with this other lasers.

Reflect	`reflect`	Makes lasers reflect when intersecting
Reflect Remaining Length	`reflectLRemain-der`	When reflecting, use the remaining length of laser(s) before the point of intersect or hit
Hit	`canHit`	Allow laser(s) to hit with colliders
Hit Mask	`hitLayerM`	This layermask on raycashits for this laser(s)
Hit Reflect	`hitRefl`	Reflect when laser(s) hit colliders.
Hit Scan Mode	`hitScanMode`	Sets whether to use tags or materials during the hit scan, applies to reflection.

Hit Reflect Materials	`hitReflMat`
	
	`allowedMat`
	`allowedMatObj`
	`reflMatInc`
		Allows laser(s) to hit reflect by material, click add new to add a new entry and delete to delete existing entries. Each entry has three values, the first is the material, the second is the power increment and the third is the hit fx object that is used when hitting this material.

Hit Reflect Tags	`reflTag`
	`allowedTagObj`
	`reflTagInc`
		As above, but with tags. **This is recommended over material due to increasing performance and reducing the amount of garbage.**

Hit Passthrough Materials	`passReflMat`
	
	`passMat`
	`pasMatInc`
		(Disabled in 2D or tag mode, recommend you use a hitmask generally) Allows this laser to passthrough hit colliders when colliding with a certain material. Click add new to add a new entry and delete to delete existing entries. Each entry has two values, the first is the material and the second is the power increment.

Starting Power	`powerStart`	The starting power of laser(s).
Power Cap	`powerCap`	The maximum power of laser(s).
Reflection Power Increase	`reflInc`	The increment that the power will increase by when reflecting.
Charge Power Increase	`chargeInc`	The increment that power will increase by when charging.
Cap Objects		These are fx objects and the relevant options.
Destroy Delay	`desSpObj`	The destroy delay when removing fx objects.
Enable Intersection FX	`disalbePartObj`	Enable the creation of intersection fx, gameobjects that appear at and follow intersection locations
Start Cap	`startCapObj`	The global start fx object.
End Cap	`endCapObj`	The global end fx object.
Hit FX	`hitObj`	The global hit fx object.
Intersection FX	`intSctObj`	The global intersection fx object.

simpleGOPool

This is a simple script used to facilitate a pooling system, kindly provided via invulse in the Unity forums.

In order to use object pooling, you must enable it on the ‘Laser System’ script and make sure the script is attached to a game object within your scene.

To add order to add your own objects to the pool, make sure to place your gameobject in the ‘Object Prefabs’ field.
In the array below ‘Amount to buffer’, you can place a corresponding entry here to state the amount of pooling objects you want spawning on start/maximum amount allowed (depending upon how you use the system).
In the bool array below ‘Only Pool’, you can place a corresponding entry here to state if this object should only draw upon the pool when spawning new ones. Useful if you want to limit the number of a certain object (FX objects are a good fit for this).

When using this in your own scripts, first make sure you have the lasers namespace included.

To spawn an object via pooling, use the following command:

```
[storedgameobject] = simpleGOPool.instance.GetObjectForType([gameobject].name, [onlyusepooled]);
```

The first variable is the name of the gameobject prefab being spawned, this is used to identify the gameobject.
The second variable is the ‘only use pooled’ feature, which allows you to not spawn new objects if the currently spawned objects are all used up. Useful if you want to limit the number of a certain object.

In order to destroy an object in the pooling system, use this command:

```
simpleGOPool.instance.PoolObject([gameobject]);
```

The gameobject value being the one destroyed.

If you want to replicate the delay function of Unity’s default Destroy() method, we recommend placing this in a Coroutine/IEnumerator.

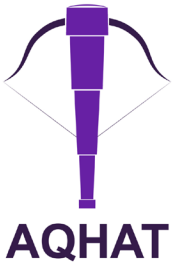
SOCIAL MEDIA/PRODUCT UPDATES

When we release an update to our products we will announce it on our social media via facebook and twitter.

Facebook - https://www.facebook.com/EagerAmoeba/
Twitter - https://twitter.com/eageramoeba

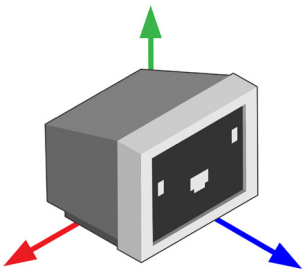
If you publish a game using any of our assets, please get in touch!
We may feature your product in future promotional posts.

OTHER ASSETS



AQHAT, $8.00

AQHAT stands for 'Auto Quality Hardware Assessment Tool', it is a simple script used to detect the potential performance of a device and score it. This score is then used in a second script to set the quality level. All done immediately upon launch.
The scripts are not packaged in a dll, so can be freely edited for integration into your own quality systems/solutions.
http://u3d.as/CdY



Easy Screen Align, Free

'Easy Screen Align' does what it's title suggests. It's a simple script for aligning elements to the screen, essentially creating a similar effect to rect transforms but dynamically with 3D objects.
http://u3d.as/DjQ

DONATIONS AND CONTRIBUTIONS

While our license allows you to use this is any many projects as you like, we would appreciate those that have found success in multiple projects to contribute a small amount to help with the upkeep of Beamtime.
We want to keep updating this software, our main aim is to add multi-player support in the form of an update or extension.
If you find any bugs or ways to optimize our script, please get in touch and let us know.

Our contact form can be found here - http://eageramoeba.co.uk/Contact/
Lastly, if you feel like contributing a small amount of money, you can do this via PayPal here - http://paypal.me/EagerAmoeba

Thanks for purchasing Beamtime, we hope it serves you well!
A review/rating in the Unity Asset Store is much appreciated and only takes a minute.