**RayFire Rigid**

It's main purpose is **Advanced Physics Control** and **Runtime Demolitions**. Using Rigid component you can demolish 3d objects into thousands of fragments and every fragment can be demolished further deeper and deeper.

**Initialization**: Defines when an object will be initialized. Initialization adds all necessary components to the object, setup them accordingly to properties and start necessary coroutines.

[**https://youtu.be/-wW9OfnhIh0**](https://youtu.be/-wW9OfnhIh0)

∙ **By Method**: Object will be initialized by ***Initialize()***method. This is the default initialization type. In this way you can add a Rigid component to objects in your code and they will not be initialized unless you will Initialize them as well. Object with **ByMethod** type can be initialized in Editor using the Initialize button on top of the Rigid component.

∙ **At Start**: Object will be initialized at Start.

Object can be initialized using method:

public void Initialize()

**Main**

**Simulation Type**: Defines behavior of object during simulation.

[**https://youtu.be/KZ8h9F45lkM**](https://youtu.be/KZ8h9F45lkM)

∙ **Dynamic**: Object will be affected by gravity, will start to fall down and it will be possible to affect on it by other simulated objects.

∙ **Sleeping**: Object will freeze in air until first collision with any other object, then it will start to behave as a Dynamic object.

∙ **Inactive**: Object will freeze in air and will not be affected by gravity, it can be affected by other simulated objects but it will not start to fall down until it will be activated. After activation the object will start to behave like a Dynamic object.

∙ **Kinematic**: Object will use it's animation to affect other objects but it will not be affected anyhow by any other object. Can use actual mesh as a collider.

∙ **Static**: Object will not be moved from it's position, it will interact with Dynamic objects but it will not be affected anyhow by any other object. Can use actual mesh as a collider.

**\*\*\***

**Object Type**: Rigid component can simulate a single object using it's mesh for simulation, but also it can simulate multiple objects as one concave object using all children meshes.

[**https://youtu.be/fbVG1Y0RoBY**](https://youtu.be/fbVG1Y0RoBY)

∙ **Mesh**: Simulate object using its Meshfilter's mesh in Mesh Collider.

∙ **Mesh Root**: Should be added to the empty root with children with meshes. Add the same Rigid component to all children and simulate them as **Mesh** type. Root itself is not going to be simulated. Using this type you don't need to select every child to add Rigid components or to edit them later, you only need to edit all properties for Root.

∙ **Skinned Mesh**: Demolishes Skinned mesh object. Skinned Mesh objects can be demolished only by manually using the public method Demolish().

∙ **Nested Cluster**: Simulated object using all its children Meshfilter's meshes as one solid concave object. In this case root will get a Mesh Collider for every child with it's mesh. To Create such an object you need to add a Rigid component to the root which has all objects which you want to simulate as solid object as its children. In case of demolition Cluster will detach every child and start simulating them on their own. If a child is also just a root for other children with meshes after demolition it will be considered as a Nested Cluster as well. In this way you can demolish object in your own way.

∙ **Connected Cluster**: Simulated object using all its children Meshfilter's meshes as one solid concave object like Nested Cluster. It uses only children from the first layer, nesting is not allowed. During demolition it will detach fragments at contact point while the rest of the fragments will stay as a solid cluster. At every demolition cluster checks itself for connectivity and if it detects that some groups of fragments are not connected together anymore they will start to simulate as separate Connected Clusters.

**\*\*\***

**Demolition Type**: Define when and how object will be demolished during Play mode.

[**https://youtu.be/\_aO08dcqBOQ**](https://youtu.be/_aO08dcqBOQ)

∙ **None**: Object will not be demolished.

∙ **Runtime**: Object will be demolished in Runtime. Fragments will be created at collision contact moment. It is better to use this type for low poly objects with low Fragments **Amount** value. Slowest speed among other types.

∙ **Awake Precache**: Object will precached all fragments mesh data in Awake and store all mesh data info in Rigid component. At collision contact gameobjects with all necessary components will be created and mesh will be assigned.

∙ **Awake Prefragment**: Object will be prefragmented in Awake. All fragments will be disabled and waiting for demolition.

∙ **Reference Demolition**: Allows to swap demolished object to predefined reference and demolish instance of this reference instead. Reference object could be FBX asset, prefab or other scene object.

**Simulation**

Physics

[**https://youtu.be/MxMaPZLpLto**](https://youtu.be/MxMaPZLpLto)

**Physic Material**

**Material Type**: Material preset with predefined density, friction, elasticity and solidity. Can be edited in the Rayfire Man component.

**Material**: Allows to define own **Physic Material.** Keep in mind that it still will define **Material Type** for **Solidity** and **Mass**.

**Mass**

**Mass By**: Allows to choose the way Mass will be applied to the RigidBody component.

∙ **Material Density**: Object will get mass according to defined **Material Type** and mesh volume.

∙ **Mass Property**: Object will get mass by Mass property

**Mass**: RigidBody component will get this value if **Mass By** set to **Mass Property.**

**Collider**

**Collider Type**: Allows to choose automatic collider type for object if the object has no custom collider applied.

∙ **Mesh**: Object will get Mesh Collider.

∙ **Sphere**: Object will Sphere Collider.

∙ **Box**: Object will get Box Collider.

∙ **None**: Object will not get an automatic collider.

**Planar Check**: Do not add Mesh Collider to object with very small and/or planar mesh. Mesh Collider can't add the correct mesh collider to such meshes so applied mesh collider is always bigger than it should be and when such an object is positioned very close to other objects such collider will cause explosive simulations. Enable this property to avoid such simulation behaviour.

**Ignore Near**: Allows to ignore all colliders around object if their bounding boxes overlap with each other. Enable if you have a lot of objects positioned close to each other so their convex mesh colliders overlap with each other which causes unstable simulation**.**

**Other**

**Use Gravity**: Object will be affected by Gravity force.

**Solver Iterations**: Defines Solver Iterations for objects Rigid Body component.

**Fragments**

**Dampening**: Decrease velocity of demolished fragments.

*Activation*

[**https://youtu.be/t8JbxKyP91Y**](https://youtu.be/t8JbxKyP91Y)

Inactive object can be activated using public method:

public void Activate()

**By Velocity**: Inactive object will be activated when it's velocity will be higher than **By Velocity** value when it was pushed by other dynamic objects. Turned Off when set to 0.

**By Offset**: Inactive object will be activated if pushed from its original position farther than **By Offset** value. Turned Off when set to 0.

**By Damage**: Inactive object will be activated if it's Current Damage will be higher than **By Damage** value. Turned Off when set to 0.

**By Activator**: Inactive object will be activated by overlapping with object with **RayFire Activator** component.

**By Impact**: Inactive object will be activated when it will be shot by **RayFire Gun** component.

**By Connectivity**: Inactive object will be activated if it won't be able to reach any Inactive object with Unyielding property On through other Inactive objects. **RayFire Connectivity** components have to be used to establish connectivity between Inactive objects and activate them if they lose connection with other Inactive objects.

**Connectivity**

**Unyielding**: Allows to define Inactive/Kinematic object as Unyielding to check for connection with other Inactive/Kinematic objects with enabled By Connectivity activation type.

**Activatable**: Unyielding object can not be activated by default. When On allows to activate Unyielding objects as well.

**Post Activation**

**Layer**: Allows to set different Layer for activated object.

**Demolition**

Demolition can be initiated using method:

public void DemolishObject()

Slicing by planes can be initiated by method:

public void AddSlicePlane(Vector3[] slicePlane)

Vector3[] slicePlane is array with slicing planes info:

(plane position, plane normal, plane position, plane normal, etc...)

*Limitations*

[**https://youtu.be/ITYShtaF5cQ**](https://youtu.be/ITYShtaF5cQ)

**Collision**

**By Collision**: Enables demolition by collision.

**Solidity**: Local Object solidity multiplier for object. Low Solidity makes object more fragile. Keep in mind that every **Material Type** in **Physics** group also has its own Solidity. Set to 0 if you want an object to be demolished at first contact.

**Tag**: Allows to set specific Tag so object with Rigid will be demolished by collision only when collides with object with defined Tag.

**Other**

**Depth**: Defines how deep object can be demolished. Depth is limitless if set to 0.

*Example*: **Max Depth** set to 1, it means that object can be demolished one time and new fragments (Depth level 1 fragments) will not be possible to demolish anymore.

If **Max Depth** set to 2, it means that object can be demolished one time and new fragments (Depth level 1 fragments) will possible to demolish one time, but new fragments (Depth level 2 fragments) will not be possible to demolish anymore. Etc

**Time**: Safe time. Measures in seconds and allows to prevent fragments from being demolished right after they were just initialized.

*Example*: **Time** set to 3 seconds, it means that object can be demolished and new fragments will not be demolished for sure during the next 3 seconds.

**Size**: Prevent objects with bounding box size less than defined value to be demolished. Measures in units.

**Visible**: Prevent object from being demolished if they are not visible by the main camera..

**Slice by Blade**: Allows object to be sliced by object with **RayFire Blade** component.

*Mesh Demolition*

**Fragments**

[**https://youtu.be/TpTHstzzvRU**](https://youtu.be/TpTHstzzvRU)

**Amount**: Defines amount of new fragments after demolition.

**Variation**: Defines additional amount variation for object in percents. Final amount will always be higher than **Amount** value.

*Example*: Amount set to 50, Variation set to 10 %. Final amount will be randomly picked in the range from 50 to 55.

**Depth Fade**: Amount multiplier for next Depth level. Allows to decrease fragments amount of every next demolition level.

*Example*: **Amount** set to 100. **Depth Fade** set to 0.5. Object demolishes to 100 fragments. For every fragment it's **Amount** for next demolition will be set to 50 (100 \* 0.5). Let's say some fragment demolishes to 50 fragments. For every fragment among these 50 fragments it's **Amount** for next demolition will be set to 25 (50 \* 0.5).

**Contact Bias**: Higher value allows to create more tiny fragments closer to collision contact point and bigger fragments far from it. This is the property which makes the player understand that Runtime demolition happened right now and nothing was prefragmented before.

**Seed**: Defines Seed for fragmentation algorithm. Same Seed will produce the same fragments for the same object every time.

**Use Shatter**: Allows to use RayFire Shatter properties for fragmentation. Works only if the object has a RayFire Shatter component. This is the only way to create other Fragmentation types in runtime because default demolition fragmentation type produce only regular Voronoi fragments.

**Advanced**

**Clusterize**: Allows to turn demolition fragments to Connected Cluster.

**Mesh Input**: Allows to choose when demolished mesh should be processed for fragmentation. Processing takes about 10% - 15% of total fragmentation time.

∙ **At Start**: Mesh will be processed at Start.

∙ **At Initialization**: Mesh will be processed at Initialization.

∙ **At Demolition**: Mesh will be processed at Demolition.

*Properties*

**Collider**

**Collider Type**: Defines Collider type for mesh fragments if their bounding box size is less than **Size Filter** value. Useful in case you want to apply simple colliders for a lot of small fragments.

**Size Filter**: Defines Bounding Box size value for Collider Type property. Turned Off if set to 0.

**Mesh Ops**

**Decompose**: Decompose output fragment's mesh to several meshes if they are not connected with each other.

**Remove Collinear**: Remove collinear vertices on output fragment's mesh.

**Custom Layer**

**Layer**: Set custom Layer to fragments.

*Runtime Caching*

[**https://youtu.be/RF-srUHcy0g**](https://youtu.be/RF-srUHcy0g)

**Type**: Defines Runtime Caching type. Disabled by default.

∙ **By Frames**: Cache fragment's meshes over defined amount of frames.

∙ **By Fragments per Frame**: Cache defined amount of fragment's meshes every frame.

**Frames**: Defines amount of frames for **By Frames** type.

**Fragments**: Defines amount of fragments for **By Fragments per Frame** type.

**Skip First Demolition**: Do not demolish object when all meshes were cached and wait for the next demolition. Allows to use first Demolition to initiate caching and demolish object only at second demolition.

*Cluster Demolition*

[**https://youtu.be/fNFG4SGWuQQ**](https://youtu.be/fNFG4SGWuQQ)

[**https://youtu.be/ZuHq1OFylnM**](https://youtu.be/ZuHq1OFylnM)

**Properties**

**Connectivity**: Define the way connections among Shards will be calculated.

∙ **By Bounding Box**: Shards will be considered as connected if their bound boxes interpenetrate with each other. This is the fastest way to establish connectivity and can be used in Runtime, but calculated shared area value will not be precise and will be based on shard's size.

∙ **By Mesh**: Shards will be considered as connected if the triangle of one Shard fits the triangle of neighbor shard. This is the slowest way to establish connectivity and should be used only in Editor with Setup Cluster button, but it precisely calculates shared area value which provides realistic Collapse By Area.

∙ **By Bounding Box and Mesh**: Shards will be considered as connected if their bound boxes interpenetrate with each other. If Shards also share triangles then the Shared area will be calculated using By Mesh method, if shards do not share triangles then By Bounding Box method will be used.

**Filters**

Filters allow you to decrease the amount of connections. There are three filters and they are turned off by default.

**Minimum Area**: Two shards will have connection if their shared area is bigger than this value.

**Minimum Size**: Two shards will have connection if their size is bigger than this value.

**Percentage**: Random percentage of connections will be discarded.

**Seed**: Seed for random percentage filter and for Random Collapse.

**Demolition Distance**

When a Connected Cluster collides with something or gets Impact from Gun and should be demolished at contact point it has to decide how much Shards should be detached into separate fragments, all Shards in **Demolition Distance** radius from contact point will be detached.

Then the rest of the Shards will check for connectivity with each other and if there are several groups of Shards which are connected only with each other then such groups will be simulated as separate Connected Clusters.

**Type**: Defines the way Demolition Distance will be measured.

∙ **Ratio To Size**: Measured in percentage relative to initial cluster size.

For instance, if cluster size is 50 units and Ratio To Size value is 20% then Demolition Distance radius will be 10 units and all Shards in 10 Units radius around contact point will be detached from cluster.

∙ **World Units**: Measured in world Units.

**Ratio**: Percentage value for Demolition Distance.

**Units**: World Units value for Demolition Distance.

**Shards**

All detached Shards in the Demolition Distance area will get RayFire Rigid Component with Mesh object type and will be simulated separately. All such Shards will inherit Mesh Demolition properties from the initial Connected Cluster Rigid. But it is possible to clusterize these detached Shards into small Connected Clusters, thanks to this you can create a smooth transition between Initial demolished Connected Cluster to separate detached SHards so between them there will be small groups of Connected Clusters.

**Shard Area**: Defines percentage relative to Demolition Distance.

For instance, Cluster size is 50 units, Demolition Distance is 20 units, so all Shards in 20 units radius around contact point will be detached from demolished connected cluster. If Shard Area value is 100% then all Shards will be simulated separately as Rigid with Mesh object type, if Shards Area value is 25% then only Shards in 5 units radius around contact point will be simulated separately and the rest of the Shards from 5 to 20 units radius will be divided into groups and every group will be simulated as small Connected Cluster. Amount of such groups can be defined using Min and Max Amount properties in the Cluster section.

**Shard Demolition**: Set Runtime demolition type for detached Shards with Mesh object type.

**Clusters**

**Min Amount**: Defines minimum amount of Connected Clusters which will be created inside Demolition Distance radius.

**Max Amount**: Defines maximum amount of Connected Clusters which will be created inside Demolition Distance radius. Final value will be picked randomly between Min and Max range.

**Demolishable**: Set Runtime demolition type for created Connected Clusters so it will be possible to demolish them as well.

**Collapse**

**Type**: There are three ways to break connections among Shards. You should try in Editor all of them to find the one which looks better for your structure and your goal.

∙ **By Area**: Shard will lose its connections if it's shared area surface is less than defined value.

∙ **By Size**: Shard will lose its connections if it's Size is less then defined value.

∙ **Random**: Shard will lose its connections if it's random value in range from 0 to 100 is less than the defined value.

**Start**: Defines start value in percentage relative to the whole range of picked type. For instance, Type set to By Size, you have 200 shards with the smallest shard size equal to 0.1 unit and biggest shard size equal to 50 units. If you set **Start** value to 50% it will start to collapse by setting By Size threshold value to 25 units and all shards with size less than 25 units will lose their connections. Increase this value if you want to Start Collapse quickly so a lot of shards will lose connections and will be activated right from the beginning. Set it to 0 if you want to start collapse slowly activating shards one by one.

**End**: Defines end value in percentage relative to the whole range of picked type. Set tp 100 if you want to break all connections or set it lower if you want to keep some connections.

**Steps**: Amount of times when defined threshold value will be set during the Durationperiod.

**Duration**: Time which will take **Start** value to be increased to **End** value**.**

*Reference Demolition*

Using Reference Demolition type you can swap demolished object to another Gameobject in scene, Prefab asset, FBX asset or even custom Particle System with your own scripts. This demolition type can be used for fragments which you just created using the Shatter component or for FBX asset with fragments which you imported from another 3d package.

If Reference is just one object then at demolition it will be demolished instead of original object.

If Reference is an empty root with children then these children will be simulated as fragments.

It is also possible to keep the Reference field empty so the original demolished object will disappear without leaving anything instead.

[**https://youtu.be/CtY4w16357Q**](https://youtu.be/CtY4w16357Q)

**Source**

**Reference**: Defines reference for demolition. Reference will be instantiated and demolished instead of the original object.

**Random List**: Allows to define several references and pick for demolition one random.

**Properties**

**Action**: Defines how source object should be used.

* **Instantiate**: Instantiate Source objects and use instance.
* **Set Active**: Activate Source object and use it for demolition.

**Add Rigid**: All references without Rigid Component will get it automatically. Single Object will get Mesh Object Type, Roots with a group of mesh objects will get Mesh Root object type.

**Inherit Scale**: Instantiated reference will inherit original object scale.

*Materials*

[**https://youtu.be/2Aqdzt58xw0**](https://youtu.be/2Aqdzt58xw0)

**Inner surface**

**Inner material**: Defines material for fragment's inner surface. If not applied RayFire will use original object material for the inner surface.

**Mapping Scale**: Defines mapping scale for inner surface. Using this property you can increase or decrease the size of texture for the inner surface.

**Outer surface**

**Outer material**: Defines material for fragment's.

*Damage*

[**https://youtu.be/WfoaziBQiGU**](https://youtu.be/WfoaziBQiGU)

**Enable**: Allows to demolish object by its own floating Damage value. Object will be demolished when Current Damage value will be more or equal Max Damage value. Current Damage value can be increased by RayFireBomb and RayFireGun components, by Collision or by public method:

public bool ApplyDamage(float damageValue, Vector3 damagePosition)

*damagePosition: P*osition in world space. In this case the damage position will be used by Contact Bias to create more tiny fragments at the damaged area.

True boolean return value means the object was demolished because of applied damage.

**Max Damage**: Defines maximum allowed damage for an object before it will be demolished.

**Current Damage**: Shows current damage value.

**Collisions**

**Collect** : Allows to accumulate damage value by collisions during dynamic simulation.

**Multiplier**: Multiply collected collision damage.

*Fading*

[**https://youtu.be/wYFPii-pAuY**](https://youtu.be/wYFPii-pAuY)

**Initiation**

**On Demolition**: Initiate Fading for demolished object fragments.

**On Activation**: Initiate Fading when objects Activated (change Inactive simulation type to Dynamic)

**By Offset**: Initiate Fading when objects moved far enough from it's Initialise position.

**Life**

**Life Type**: Defines time while objects will be simulated before object will start to fade.

* **By Life Time**: Life time timer starts right after Fading was initiated and equal to **Life Time** property value.
* **By Life Time and Simulation**: Life time timer starts when objects stop simulation and do not move for a few seconds, then Life Time timer starts.

**Life Time**: Minimum period of time in seconds during which fragments created by demolition or slicing will be simulated.

**Life Variation**: Add random period of time in seconds to final **Life Time.**

**Fade**

**Fade Type**: Fading lets you keep your scene clean and optimize dynamic simulation by destroying fragments or excluding them from simulation.

* **None**: Fragments will stay in the scene with no limitation and they always will be simulated.
* **Sim Exclude**: Fragments will be simulated for **Life Time** period after their birth, then their Colliders and RigidBody components will be destroyed.
* **Fall Down**: Fragments will be simulated for **Life Time** period after their birth, then they will fall through the ground and will be destroyed after **Fade Time** period.
* **Scale Down**: Fragments will be simulated for **Life Time** period after their birth, then they will start slowly scaling to nothing during the Fade **Time** period and be destroyed after all.
* **Move Down**: Fragments will be simulated for **Life Time** period after their birth, then they will move through the ground to the distance equal to their size and will be destroyed after **Fade Time** period.
* **Destroy:** Fragments will be simulated for **Life Time** period after their birth and destroyed instantly.
* **Set Static**: Fragments will be simulated for **Life Time** period after their birth, then Destroy Rigid Body component and leave Collider.
* **Set Kinematic**: Fragments will be simulated for **Life Time** period after their birth, then Rigid Body component will be set to Kinematic.

**Fade Time**: Period of time in seconds during which an object slowly moves through the ground or scales down to nothing.

**Size FIlter**: Prevent object for fading if it's size is bigger than defined value. Turned Off if 0.

*Reset WIP*

[**https://www.youtube.com/watch?v=6h94gd\_gDrY**](https://www.youtube.com/watch?v=6h94gd_gDrY)

*Events*

[**https://youtu.be/lRlqcHGA1sI**](https://youtu.be/lRlqcHGA1sI)

RayFire Rigid component provides subscription to **Demolition** and **Activation** events.

Subscription to **Global** event will invoke subscribed method when any demolition happens.

**Global Demolition** event subscription reference**:**

RFDemolitionEvent.GlobalEvent += MyMethod;

**Global Activation** event subscription reference**:**

RFActivationEvent.GlobalEvent += MyMethod;

Subscription to **Local** event will invoke subscribed method when demolition of specific Rigid script happens and because of this you need to have reference to Rigid component you want to track.

**Local Demolition** event subscription reference**:**

GameObject myGameObject = GameObject.Find("MyObjectWithRigidComponent");

RayfireRigid rigid = myGameObject.**GetComponent**<RayfireRigid>();  
 rigid.demolitionEvent.LocalEvent += MyMethod;

**Local Activation** event subscription reference**:**

GameObject myGameObject = GameObject.Find("MyObjectWithRigidComponent");

RayfireRigid rigid = myGameObject.**GetComponent**<RayfireRigid>();  
 rigid.activationEvent.LocalEvent += MyMethod;

Keep in mind that MyMethod should have RayFireRigid component as input, in this way you can get access to all public variables of Rigid component, for instance:

void MyMethod(RayfireRigid rigid)  
**{** Debug.Log(rigid.*fragments*.*Count*);  
}

will show in console the amount of fragments which were created as a result of demolition.