Table of Contents

1. Overview
2. Class Declaration
3. Fields and Properties
   3.1 Main References and Initialization Flags
   3.2 Mesh Deformation
   3.3 Wheel Deformation
   3.4 Light Deformation
   3.5 Part Deformation
   3.6 Internal Mesh/Vertex Tracking
   3.7 Octree References
4. Key Methods
   4.1 Initialization (`Initialize`)
   4.2 Data Collection
   4.3 Repair Logic (`UpdateRepair`)
   4.4 Damage Logic (`UpdateDamage`, `DamageMesh`, etc.)
   4.5 Collision Handling (`OnCollision`, `OnCollisionWithRay`)
   4.6 Wheel Detachment (`DetachWheel`)
   4.7 Octree Usage (`NearestVertexWithOctree`)
5. Usage Notes and Best Practices
6. Summary

---

1. Overview

The **`RCC_Damage`** class is a component responsible for managing and applying damage to a vehicle's mesh, wheels, lights, and detachable parts in Realistic Car Controller (RCC). It allows for **mesh deformation** on collisions, **detaching wheels**, **breaking lights**, and repairing all damage through an internal data structure that tracks both the original and deformed vertex positions.

Key Features:

- Deforms vehicle meshes at collision points based on impact force.
- Supports wheel damage, potentially detaching a wheel above a certain threshold.
- Damages lights and detachable parts (such as bumpers, doors) with configurable radii and multipliers.
- Provides a `repairNow` mechanism to gradually restore all vertices to their original positions.
- Uses **octrees** for efficient nearest-vertex lookups in mesh deformation.

---

2. Class Declaration

```
[System.Serializable]
public class RCC_Damage {
    // Implementation details...
}
```

This class is `[System.Serializable]`, making it editable in a custom inspector or included as a serializable field within another component (like `RCC_CarControllerV4`).

---

3. Fields and Properties

The `RCC_Damage` class contains numerous serialized fields for configuring its behavior. This section breaks them down by category.

3.1 Main References and Initialization Flags

- **`public RCC_CarControllerV4 carController`**
  The main RCC car controller that owns this damage component.

- **`public bool automaticInstallation = true;`**
  If true, automatically gathers mesh filters, wheels, lights, and detachable parts from the vehicle children during initialization.

- **`private bool initialized = false;`**
  Internal flag to check if the class has completed its initialization routine.

- **`public bool repairNow = false;`**
  Triggers the repair process. When true, the system gradually restores meshes/wheels to their original, undamaged state.

- **`public bool repaired = true;`**
  Indicates whether the vehicle is fully repaired (no differences between original and current vertex positions).

- **`private bool deformingNow = false;`**
  True while the mesh is currently being deformed (following a collision).

- **`private bool deformed = true;`**
  Indicates whether the vehicle has finished deforming its meshes to the damaged

positions.

3.2 Mesh Deformation

- `public bool meshDeformation = true;`
  Master toggle for whether mesh deformation is active.

- `public enum DeformationMode { Accurate, Fast }`

  - **Accurate**: Smoothly transitions vertices over time to target positions.
  - **Fast**: Immediately snaps vertices to their final positions.
- `public DeformationMode deformationMode = DeformationMode.Fast;`
  Determines how quickly or smoothly deformation occurs.

- `[Range(1, 100)] public int damageResolution = 100;`
  Influences how finely damage is applied, though the direct usage can depend on the internal approach.

- `public LayerMask damageFilter = -1;`
  Only collisions with these layers will register damage.

- `public float damageRadius = .75f;`
  Radius around the collision point in which vertices are affected.

- `public float damageMultiplier = 1f;`
  Global multiplier for mesh damage severity.

- `public float maximumDamage = .5f;`
  Maximum distance (from the original position) that any vertex can be displaced. `0f` disables the limit.

- `private readonly float minimumCollisionImpulse = .5f;`
  Collisions with impulses below this threshold do not apply any deformation.

- `public bool recalculateNormals = true;`
  Recalculate mesh normals after deformation (can be CPU-intensive).

- `public bool recalculateBounds = true;`
  Recalculate mesh bounds after deformation.

3.3 Wheel Deformation

- `public bool wheelDamage = true;`
  Toggle for wheel damage (e.g., bending or shifting wheel position).

- `public float wheelDamageRadius = 1f;`
  Radius around the collision point within which wheels can be affected.

- `public float wheelDamageMultiplier = 1f;`
  Global multiplier for wheel damage severity.

- `public bool wheelDetachment = true;`
  If true, wheels can be fully detached when damage surpasses a set threshold.

3.4 Light Deformation

- `public bool lightDamage = true;`
  Toggle for light damage (e.g., headlights, brake lights).

- `public float lightDamageRadius = .5f;`
  Collision radius for determining if lights are hit.

- `public float lightDamageMultiplier = 1f;`
  Damage multiplier for lights.

3.5 Part Deformation

- `public bool partDamage = true;`
  Toggle for detachable or breakable parts (bumper, doors, etc.).

- `public float partDamageRadius = 1f;`
  Collision radius for part damage.

- `public float partDamageMultiplier = 1f;`
  Damage multiplier for these parts.

3.6 Internal Mesh/Vertex Tracking

- `public MeshFilter[] meshFilters;`
  The collection of meshes subject to deformation.

- `public RCC_DetachablePart[] detachableParts;`
  Breakable parts that can detach on collision.

- **`public RCC_Light[] lights;`**
  Vehicle lights (headlights, tail lights) that can be disabled or broken.

- **`public RCC_WheelCollider[] wheels;`**
  The vehicle's wheel colliders for handling possible damage or detachment.

- **`public struct OriginalMeshVerts { public Vector3[] meshVerts; }`**
  Stores original mesh vertex positions.

- **`public OriginalMeshVerts[] originalMeshData, damagedMeshData;`**
  - `originalMeshData`: Snapshot of the mesh in its pristine state.
  - `damagedMeshData`: Snapshot of the mesh with any applied deformations.
- **`public struct OriginalWheelPos { public Vector3 wheelPosition; public Quaternion wheelRotation; }`**
  Maintains each wheel's original local position/rotation.

- **`public OriginalWheelPos[] originalWheelData, damagedWheelData;`**
  - `originalWheelData`: The wheels' default positions.
  - `damagedWheelData`: The wheels' new positions after collisions.

3.7 Octree References

- **`public RCC_Octree[] octrees;`**
  An array of octrees for each mesh filter. This speeds up the process of finding the nearest vertex to a collision point.

- **`public Vector3 NearestVertexWithOctree(...)`**
  Helper function that queries the octree to find the nearest vertex in local space.

---

4. Key Methods

4.1 Initialization (`Initialize`)

```
public void Initialize(RCC_CarControllerV4 _carController) {
    carController = _carController;

    if (automaticInstallation) {
        if (meshDeformation)
            CollectProperMeshFilters();
```

```
        if (lightDamage)
            GetLights(carController.GetComponentsInChildren<RCC_Light>());
        if (partDamage)
            GetParts(carController.GetComponentsInChildren<RCC_DetachablePart>());
        if (wheelDamage)
            GetWheels(carController.GetComponentsInChildren<RCC_WheelCollider>());
    }

    initialized = true;
}
```

- **Purpose**: Configures references to the main `carController` and optionally collects all relevant meshes, wheels, lights, and parts if `automaticInstallation` is enabled.
- **Usage**: Typically called once from `RCC_CarControllerV4` upon vehicle initialization.

4.2 Data Collection

`CollectProperMeshFilters()`

- Gathers all **readable** (Read/Write enabled) `MeshFilter` components, excluding wheel meshes.
- Stores them in `meshFilters` to apply damage.

`GetMeshes(MeshFilter[] allMeshFilters)`

- Assigns `meshFilters` directly if you prefer a manual approach.

`GetLights(RCC_Light[] allLights)`

- Assigns the `lights` array for breakable light objects.

`GetParts(RCC_DetachablePart[] allParts)`

- Assigns the `detachableParts` array for dynamic breakable components.

`GetWheels(RCC_WheelCollider[] allWheels)`

- Assigns the `wheels` array for wheel damage/detachment.

4.3 Repair Logic (`UpdateRepair`)

```
public void UpdateRepair() {
    if (!initialized || repaired || !repairNow)
```

```
        return;

    // Iterates all meshFilters, wheels, lights, etc. and gradually moves them
    // from their damaged positions to their original positions.
    // ...
}
```

- **Purpose**: Called every frame (or fixed frame) to gradually restore deformed meshes or wheels to their original states (`originalMeshData` / `originalWheelData`).
- **Mechanics**:
    - If `deformationMode == DeformationMode.Accurate`, uses a smooth lerp approach over time.
    - If `deformationMode == DeformationMode.Fast`, snaps vertices directly to their original positions.
    - If all vertices are within a small threshold (`minimumVertDistanceForDamagedMesh`), sets `repaired = true`.

4.4 Damage Logic (`UpdateDamage`, `DamageMesh`, etc.)

`UpdateDamage()`

- Smoothly applies final adjustments to vertex positions if `deformingNow` is still true.
- Similar approach to `UpdateRepair` but moves vertices to their *damaged* positions.

`DamageMesh(float impulse)`

- Modifies the mesh's vertex array based on collision `impulse` and `contactPoint`.
- Employs an octree search to find the nearest vertices around the collision radius, applying offsets.

`DamageWheel(float impulse)`

- Shifts wheels' local positions if they are within `wheelDamageRadius` of the collision point.
- May trigger detachment if damage surpasses `maximumDamage` and `wheelDetachment` is enabled.

`DamageLight(float impulse)`

- Invokes `lights[i].OnCollision(damage)` if a light is within `lightDamageRadius`.
- Typically breaks or disables the light.

`DamagePart(float impulse)`

- Invokes `detachableParts[i].OnCollision(damage)` if a detachable part is within `partDamageRadius`.
- Can break or detach the specified part.

4.5 Collision Handling (`OnCollision`, `OnCollisionWithRay`)

```
public void OnCollision(Collision collision) {
    if (!carController || !initialized || !carController.useDamage)
        return;

    // Check layer mask, calculate impulse,
    // gather contact points, apply damage to meshes, wheels, parts, lights...
}
```

- **`OnCollision(Collision collision)`**
    - Called externally (from `RCC_CarControllerV4` or similar) when Unity's `OnCollisionEnter` triggers.
    - Collects contact points, sums their positions, and calls the appropriate `Damage*()` methods if impulse > `minimumCollisionImpulse`.
- **`OnCollisionWithRay(RaycastHit hit, float impulse)`**
    - Variant that handles damage from a raycast hit rather than a full physics `Collision`.
    - Useful for special effects like bullet hits or environment hazards.

4.6 Wheel Detachment (`DetachWheel`)

```
public void DetachWheel(RCC_WheelCollider wheelCollider) {
    // Deactivates the RCC_WheelCollider, spawns a clone with
    // its own Rigidbody & MeshCollider, allowing it to break away physically.
}
```

- **Purpose**: Completely removes the wheel from the active vehicle, spawns a **loose wheel** object with its own `Rigidbody`.
- **Scenario**: If the local offset of the wheel surpasses `maximumDamage`, the wheel can be forcibly detached for a more dramatic collision effect.

4.7 Octree Usage (`NearestVertexWithOctree`)

```
public Vector3 NearestVertexWithOctree(int meshIndex, Vector3 contactPoint, MeshFilter meshFilter) {
    // Queries the stored RCC_Octree to find the nearest vertex in local space,
    // improving performance over naive vertex-by-vertex searches.
}
```

- **Purpose**: Rapidly locates the closest vertex in the `meshFilter` near a collision point, enabling targeted mesh deformation.
- **Note**: An `RCC_Octree` is automatically constructed the first time a mesh is damaged if one does not already exist.

---

5. Usage Notes and Best Practices

1. **Read/Write Enabled Meshes**
    - Ensure your vehicle's meshes have **Read/Write** enabled in the Unity Import Settings. Without this, the script cannot modify vertices.
    - The script logs an error for any non-readable meshes.
2. **Performance Considerations**
    - **Accurate** deformation (smooth lerp) can be CPU-heavy, especially with large meshes and real-time normal/bounds recalculations.
    - If performance is an issue, consider using **Fast** mode or reducing the frequency of normal recalculation.
3. **Limiting Damage**
    - `maximumDamage` ensures no vertex strays too far from its original position. Setting `0f` disables this constraint, allowing unbounded deformations.
4. **Wheel Detachment**
    - The wheel object is cloned and turned into a loose `Rigidbody`. Make sure your physics layers and constraints can handle the new free-floating wheel.
    - If detachment is not desired, either disable `wheelDetachment` or add logic to clamp the wheel displacement.
5. **Repair System**
    - Setting `repairNow = true` will gradually restore the vehicle over time. This is handy for repair stations or game resets.
    - Once fully repaired, `repaired = true`, and the script stops the process.
6. **Collision Source**
    - Typically called by `RCC_CarControllerV4` or similar scripts. Ensure `OnCollision` or `OnCollisionWithRay` is invoked with correct impulses.

    - `impulse` scaling is crucial for realistic damage. Tweak the `damageMultiplier` or `impulse` calculation to balance realism.
7. **Octree**
    - The `RCC_Octree` structure is best for high-polygon meshes to optimize nearest-vertex lookups.
    - If a mesh is extremely simple, the overhead of building an octree might not provide a big advantage, but is generally beneficial for complex vehicles.

---

6. Summary

The **`RCC_Damage`** class handles **advanced vehicle deformation**, **wheel detachment**, and **light/part damage** within the Realistic Car Controller framework. By tracking original vs. damaged vertex arrays, it smoothly transitions between states on collision or when repairing. Configurable parameters such as `meshDeformation`, `wheelDamage`, and `lightDamage` allow for granular control over which subsystems can be deformed or detached.

Octree optimization helps accelerate the process of finding collision-affected vertices, especially for complex meshes. Integrating this with proper collision logic, correct impulses, and well-configured mesh import settings ensures a robust, visually impressive damage system for RCC vehicles.