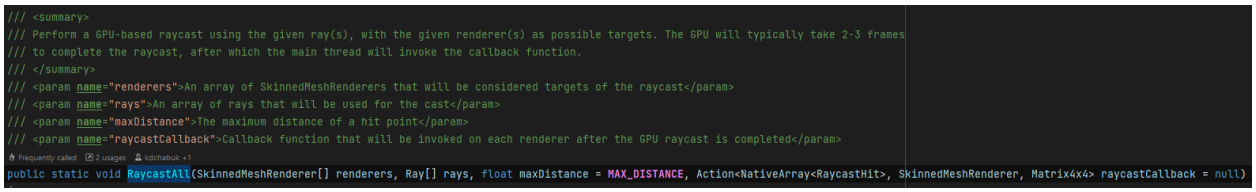
Skinned GPU Raycast

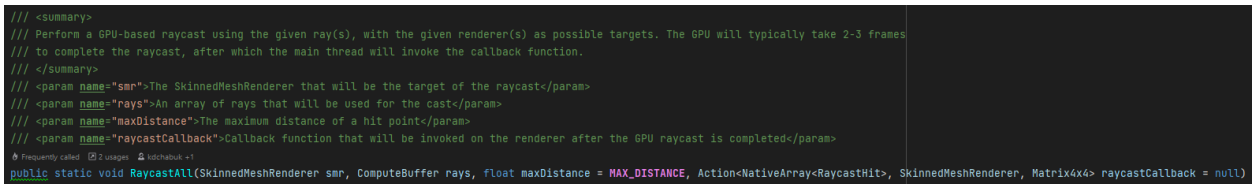
Quickstart:

Add `using RainyReignGames.SkinnedGPURaycast` to your script.
Make sure to add `asmdef` references if your script is in another assembly.

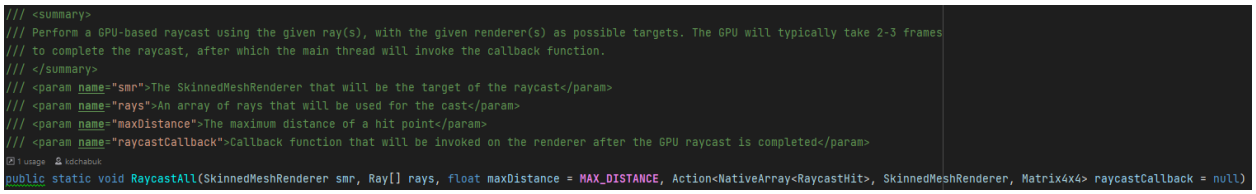
You can access the raycast functions through the static class `ShaderPhysics`. There are several overloads.



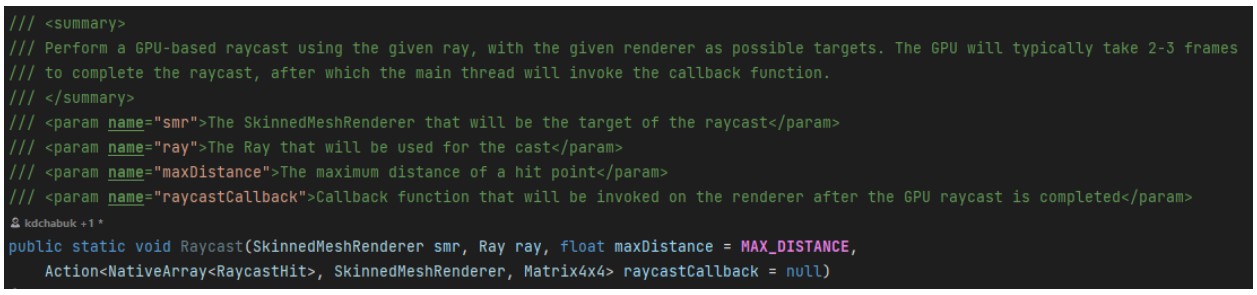
```
/// <summary>
/// Perform a GPU-based raycast using the given ray(s), with the given renderer(s) as possible targets. The GPU will typically take 2-3 frames
/// to complete the raycast, after which the main thread will invoke the callback function.
/// </summary>
/// <param name="renderers">An array of SkinnedMeshRenderers that will be considered targets of the raycast</param>
/// <param name="rays">An array of rays that will be used for the cast</param>
/// <param name="maxDistance">The maximum distance of a hit point</param>
/// <param name="raycastCallback">Callback function that will be invoked on each renderer after the GPU raycast is completed</param>
Frequently called  2 usages  kdchabuk +1
public static void RaycastAll(SkinnedMeshRenderer[] renderers, Ray[] rays, float maxDistance = MAX_DISTANCE, Action<NativeArray<RaycastHit>, SkinnedMeshRenderer, Matrix4x4> raycastCallback = null)
```



```
/// <summary>
/// Perform a GPU-based raycast using the given ray(s), with the given renderer(s) as possible targets. The GPU will typically take 2-3 frames
/// to complete the raycast, after which the main thread will invoke the callback function.
/// </summary>
/// <param name="smr">The SkinnedMeshRenderer that will be the target of the raycast</param>
/// <param name="rays">An array of rays that will be used for the cast</param>
/// <param name="maxDistance">The maximum distance of a hit point</param>
/// <param name="raycastCallback">Callback function that will be invoked on the renderer after the GPU raycast is completed</param>
Frequently called  2 usages  kdchabuk +1
public static void RaycastAll(SkinnedMeshRenderer smr, ComputeBuffer rays, float maxDistance = MAX_DISTANCE, Action<NativeArray<RaycastHit>, SkinnedMeshRenderer, Matrix4x4> raycastCallback = null)
```



```
/// <summary>
/// Perform a GPU-based raycast using the given ray(s), with the given renderer(s) as possible targets. The GPU will typically take 2-3 frames
/// to complete the raycast, after which the main thread will invoke the callback function.
/// </summary>
/// <param name="smr">The SkinnedMeshRenderer that will be the target of the raycast</param>
/// <param name="rays">An array of rays that will be used for the cast</param>
/// <param name="maxDistance">The maximum distance of a hit point</param>
/// <param name="raycastCallback">Callback function that will be invoked on the renderer after the GPU raycast is completed</param>
1 usage  kdchabuk
public static void RaycastAll(SkinnedMeshRenderer smr, Ray[] rays, float maxDistance = MAX_DISTANCE, Action<NativeArray<RaycastHit>, SkinnedMeshRenderer, Matrix4x4> raycastCallback = null)
```



```
/// <summary>
/// Perform a GPU-based raycast using the given ray, with the given renderer as possible targets. The GPU will typically take 2-3 frames
/// to complete the raycast, after which the main thread will invoke the callback function.
/// </summary>
/// <param name="smr">The SkinnedMeshRenderer that will be the target of the raycast</param>
/// <param name="ray">The Ray that will be used for the cast</param>
/// <param name="maxDistance">The maximum distance of a hit point</param>
/// <param name="raycastCallback">Callback function that will be invoked on the renderer after the GPU raycast is completed</param>
kdchabuk +1 *
public static void Raycast(SkinnedMeshRenderer smr, Ray ray, float maxDistance = MAX_DISTANCE,
    Action<NativeArray<RaycastHit>, SkinnedMeshRenderer, Matrix4x4> raycastCallback = null)
```

You will need to add a callback function to process the hits on the CPU. Note that it uses asynchronous readback to avoid stalling the GPU, so the callback will execute approximately 1-4 frames later.

The callback signature looks like:

```
void callback(NativeArray<ShaderPhysics.RaycastHit> hits, SkinnedMeshRenderer smr, Matrix4x4 localToWorldMatrix)
```

The hits are unsorted, may include multiple hits per ray, and currently have a maximum length of 4 hits times the number of rays. (For example, calling RaycastAll with 4 rays may produce a maximum of 16 hits, regardless of which rays actually hit.)

`RaycastHit` also has useful information:

```
/// <summary>
/// Hit data from the raycast.
/// All relevant properties are in object space. If transformation is required, use a callback function.
/// </summary>
[StructLayout(LayoutKind.Sequential)]
13 usages  kdchabuk +1
public struct RaycastHit
{
    /// <summary>
    /// Barycentric coordinates of the hit point.
    /// </summary>
    public float3 barycentricCoordinate;

    /// <summary>
    /// The triangle's normal vector in object space. This is computed from the cross product of the edges,
    /// so it is distinct from the vertex normals.
    /// </summary>
    public float3 faceNormal;

    /// <summary>
    /// The coordinates of the raycast hit in object space.
    /// </summary>
    public float3 hitPoint;

    // public float2 textureCoord;

    /// <summary>
    /// The distance from the ray's origin to the hit point in object space.
    /// </summary>
    public float distance;

    /// <summary>
    /// The indices into the vertex attributes for this triangle. This is the same information in the index buffer,
    /// e.g., using <c>mesh.triangles[triangleIndex]</c>.
    /// </summary>
    public uint3 triangle;

    /// <summary>
    /// The index of the hit triangle in an array of <c>int3</c>. If using an array of <c>int</c>,
    /// multiply by 3 to get the start index of the triangle.
    /// </summary>
    public uint triangleIndex;

    /// <summary>
    /// The id field of the relevant ray.
    /// </summary>
    public uint rayId;
```