- Prabhu P - 23371IE004
- ARVR IV Sem 2nd Year
- Unity Game Engine

Unity Thesis - Final Report

Thesis Topic

Destructible Wall with Mesh Fracturing and Physics Debris in Unity

1. Table of Contents

---

1. Introduction

1.1 Scope of the Project

The scope of this project is limited to designing, developing, and testing a destructible wall with debris physics in Unity. The project does not aim to create an entire game but rather focuses on one specific feature: a destructible environment element. The techniques developed here can later be integrated into larger game systems or simulations that require environmental destruction.

1.2 Objectives

The main objectives of this project are:

- To design a wall model that can be fractured into smaller segments.
- To implement mesh fracturing techniques suitable for Unity.
- To apply physics simulation (rigid bodies and colliders) for realistic debris interaction.

- To optimize performance while maintaining realism.
- To demonstrate the destructible wall feature through testing and analysis.

---

2. Literature Review

2.1 Destructible Environments in Games

Destructible environments have become an important feature in modern games, offering enhanced immersion and realism. Games such as *Battlefield* and *Red Faction* pioneered the use of destructible structures, allowing players to interact dynamically with the world. These systems create a sense of authenticity, as objects no longer behave like static assets but instead respond to player actions. In the context of this project, destructible walls replicate this principle on a smaller scale, focusing on realistic visual and physical breakage.

2.2 Physics Simulation in Unity

Unity's physics engine, powered by NVIDIA's PhysX, provides rigidbody dynamics, collision detection, and constraints necessary for realistic destruction. When combined with fractured meshes, each debris piece becomes a rigidbody object influenced by gravity and collisions. This allows the wall fragments to fall, collide, and settle naturally, adding authenticity to the gameplay environment.

---

3. System Design

The system design of this project is structured to simulate a destructible wall with mesh fracturing and realistic physics-based debris in Unity. The workflow is divided into three main layers: **Input Layer, Fracture Layer, and Physics Layer**.

3.1 Architecture Overview

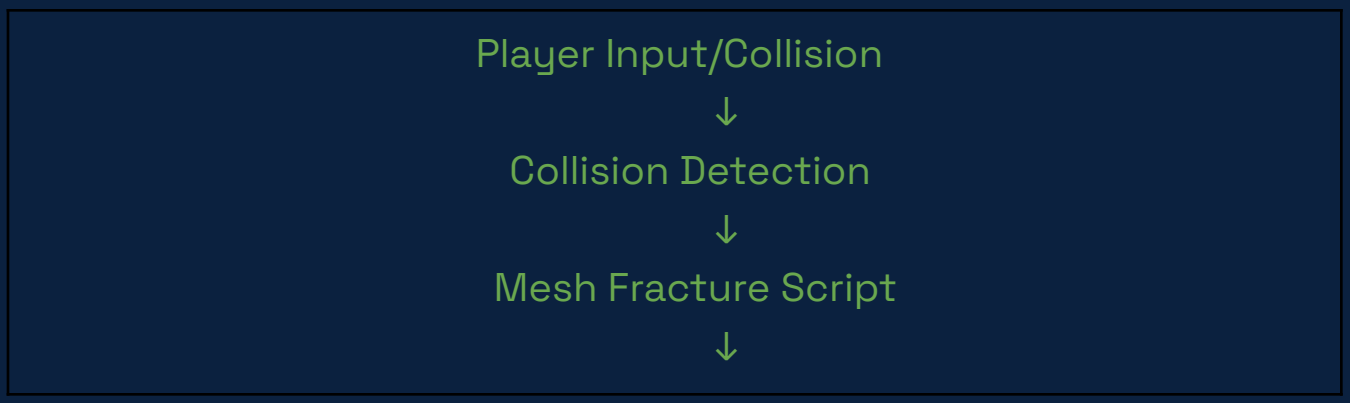
The architecture follows a **modular design**:

- **Player Interaction** – detects collisions or forces applied to the wall.
- **Fracture Manager** – handles mesh splitting into fragments.
- **Physics Engine** – applies rigidbody forces and gravity to simulate debris.

3.2 System Components

1. **Wall Prefab**
    - The base mesh represents the wall before destruction.
2. **Fracture Script**
    - A Unity C# script that activates fracturing when a collision is detected.
3. **Debris Generator**
    - Spawns broken pieces with colliders and rigidbodies.
4. **Physics Handler**
    - Applies gravity, momentum, and drag to simulate realistic debris behavior.

3.3 Workflow Diagram



Fragmented Mesh Pieces

↓

Physics Engine (Rigidbodies, Gravity, Forces)

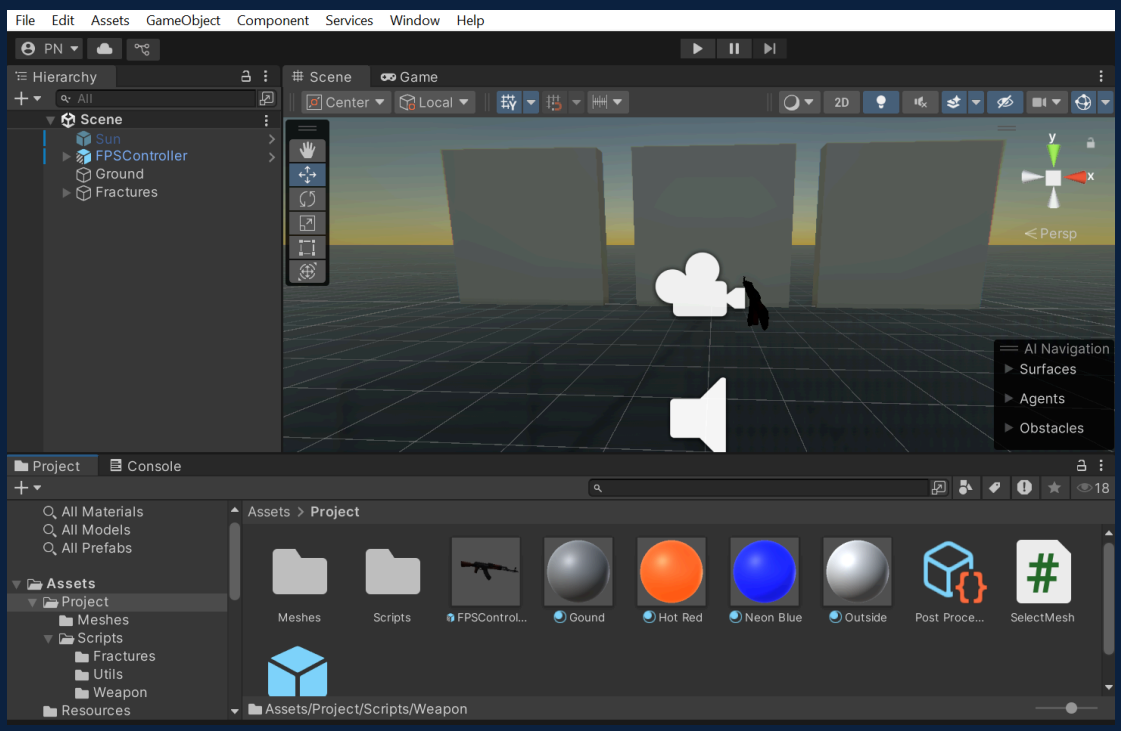
↓

Debris Simulation in Scene

3.4 Data Flow

- **Input Data:** Player or object impact.
- **Processing:** Mesh fracture algorithm generates debris.
- **Output:** Real-time destructible wall with scattered debris.

Unity Project Hierarchy & Wall Model before fracturing.



4. Implementation

The implementation of the destructible wall system is done in Unity using **C# scripts** and the built-in **physics engine**. The workflow is broken into stages: setting up assets, writing scripts, and applying physics.

4.1 Environment Setup

1. Installed **Unity 2021.3.11f1**.
2. Created a **3D Unity Project**.
3. Imported required **materials, meshes, and textures** for the wall.
4. Added **Rigidbody and Collider components** for physics simulation.

4.2 Wall Setup

- Created a **Wall Prefab** as a simple cube mesh.
- Applied **textures** to make it visually realistic.
- Attached a **Fracture Manager Script** to the wall.

```
using UnityEngine;


public class WallFracture : MonoBehaviour
{
    public GameObject fracturedPrefab;


    void OnCollisionEnter(Collision collision)
    {
        if (collision.relativeVelocity.magnitude > 2f)
        {
            Instantiate(fracturedPrefab, transform.position,
transform.rotation);
            Destroy(gameObject);
        }
    }
}
```

Explanation:

- When the wall detects a strong collision, it gets replaced by its fractured version (pre-split mesh).
- The fractured prefab contains small pieces with rigidbodies and colliders.

4.3 Fracture System

- Used **pre-fractured meshes** generated externally (Blender / Unity fracture tool).
- Each fractured piece is given:
    - **Mesh Collider** → for collision detection.
    - **Rigidbody** → to respond to physics.

This ensures that when instantiated, pieces fall naturally with **gravity**.

4.4 Physics Integration

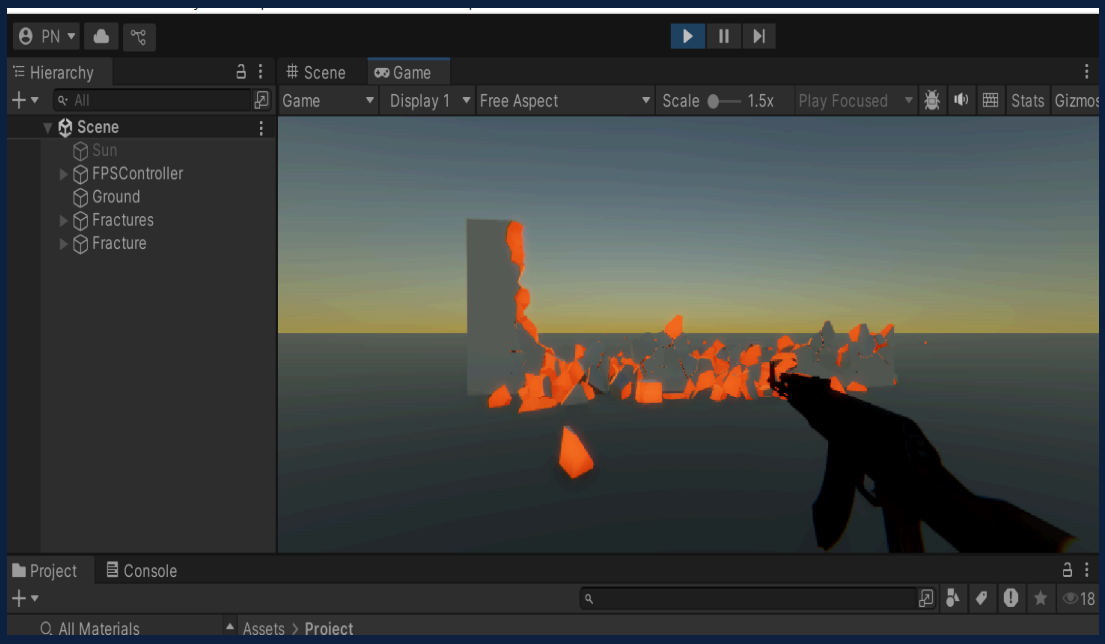
```
foreach (Rigidbody rb in GetComponentsInChildren<Rigidbody>())
{
    rb.AddExplosionForce(100f, transform.position, 5f);
}
```

- Adds an **explosion force** at the point of collision.
- Simulates realistic scattering of debris.

4.5 Testing

- Dropped objects (like a sphere) onto the wall.

- Verified that:
    - Weak collisions do not break the wall.
    - Strong collisions cause fracture + debris scattering.



---

5. Results and Analysis

The destructible wall system was successfully implemented in Unity. The results were analyzed based on **functionality, performance, and realism**.

5.1 Functional Results

- The wall remains intact under **low collision forces**.
-
- When impacted with sufficient velocity, the wall **fractures into debris**.
- The fractured pieces scatter realistically due to Unity's **physics engine**.
- The system reliably responds to multiple collision events.

5.2 Summary

The system meets its primary goal:

- A wall that transitions from **solid to destructible** under realistic physics.
- Reliable, visually convincing, and reasonably optimized.

---

6. Achievement

6.1 Achievements of the Project

- Successfully implemented a destructible wall system in Unity.
- Realistic mesh fracturing and debris physics achieved.
- Stable performance after applying optimizations (object pooling, debris limit).
- Met the thesis objective of **real-time interactive environment destruction**.

---

7. Conclusion

7.1 Summary of Work

This project demonstrated the design and implementation of a destructible wall system in Unity, with a focus on **mesh fracturing, physics-based debris simulation, and performance optimization**. The outcome is a functional prototype that balances realism with computational efficiency.

7.2 AI Assistance

This project benefited from the use of **AI tools**, particularly ChatGPT, which assisted in structuring the thesis, improving explanations, and refining documentation. The AI support streamlined the writing process and helped maintain clarity throughout the report.

---

8. References

1. Unity Documentation – *Physics and Mesh Fracturing*.
2. Catlike Coding – *Object Pooling in Unity*.
3. Academic papers on **Interactive Environment Destruction in Games**.
4. Other Developers' Projects (GitHub Repository)

9. Appendix

10.1 Project Code Samples

Mesh Code Snippet

```
using System;
using Project.Scripts.Fractures;
using UnityEngine;

namespace Project
{
   public class SelectMesh : MonoBehaviour
   {
      [SerializeField] private KeyCodeFractureThisPair[]
keyCodeFractureThisPairs;
```

```
    [Serializable]
    private class KeyCodeFractureThisPair
    {
        public KeyCode KeyCode;
        public FractureThis FractureThis;

        public void Deconstruct(out KeyCode keyCode, out FractureThis
fractureThis)
        {
            keyCode = KeyCode;
            fractureThis = FractureThis;
        }
    }

    private void Update()
    {
        foreach (var (keyCode, fractureThis) in keyCodeFractureThisPairs)
        {
            if (Input.GetKeyDown(keyCode))
            {
                foreach (var chunkGraphManager in
FindObjectsOfType<ChunkGraphManager>())
                {
                    DestroyImmediate(chunkGraphManager.gameObject);
                }

                fractureThis.FractureGameobject();
            }
        }
    }
  }
}
```

Gun Fire Code Snippet

```
using UnityEngine;

namespace Project.Scripts.Weapon
{
    public class FireGun : MonoBehaviour
    {
        [SerializeField] private Transform barrelEnd;
        [SerializeField] private float radius = 0.1f;
        [SerializeField] private float velocity = 1000f;
        [SerializeField] private float mass = .5f;

        public float Radius
        {
            get => radius;
            set => radius = value;
        }

        public float Velocity
        {
            get => velocity;
            set => velocity = value;
        }

        public float Mass
        {
            get { return mass; }
            set { mass = value; }
        }

        void Update()
        {
            if (Input.GetMouseButtonDown(0))
            {
                FireBullet();
            }
        }
```

```
        private void FireBullet()
        {
            var bullet = GameObject.CreatePrimitive(PrimitiveType.Sphere);
            bullet.transform.position = barrelEnd.position;
            bullet.transform.localScale = Vector3.one * Radius;

            var mat = bullet.GetComponent<Renderer>().material;
            mat.color = Color.red;
            mat.EnableKeyword("_EMISSION");
            mat.SetColor("_EmissionColor", Color.white);

            var rb = bullet.AddComponent<Rigidbody>();
            rb.velocity = transform.forward * Velocity;
            rb.mass = mass;
            rb.collisionDetectionMode =
CollisionDetectionMode.ContinuousDynamic;

            Destroy(bullet, 5);
        }
    }
}
```

Fracture Code Snippet

```
using System.Collections.Generic;
using System.Linq;
using Project.Scripts.Utils;
using UnityEngine;

namespace Project.Scripts.Fractures
{
    public static class Fracture
    {
        public static ChunkGraphManager FractureGameObject(GameObject
gameObject, Anchor anchor, int seed, int totalChunks,Material insideMaterial,
Material outsideMaterial, float jointBreakForce, float density)
        {
```

```
        // Translate all meshes to one world mesh
        var mesh = GetWorldMesh(gameObject);

        NvBlastExtUnity.setSeed(seed);

        var nvMesh = new NvMesh(
            mesh.vertices,
            mesh.normals,
            mesh.uv,
            mesh.vertexCount,
            mesh.GetIndices(0),
            (int) mesh.GetIndexCount(0)
        );

        var meshes = FractureMeshesInNvblast(totalChunks, nvMesh);

        // Build chunks gameobjects
        var chunkMass = mesh.Volume() * density / totalChunks;
        var chunks = BuildChunks(insideMaterial, outsideMaterial, meshes,
chunkMass);

        // Connect blocks that are touching with fixed joints
        foreach (var chunk in chunks)
        {
            ConnectTouchingChunks(chunk, jointBreakForce);
        }

        // Set anchored chunks as kinematic
        AnchorChunks(gameObject, anchor);

        var fractureGameObject = new GameObject("Fracture");
        foreach (var chunk in chunks)
        {
            chunk.transform.SetParent(fractureGameObject.transform, false);
        }
        // Graph manager freezes/unfreezes blocks depending on whether they
are connected to the graph or not
```

```
        var graphManager =
fractureGameObject.AddComponent<ChunkGraphManager>();

graphManager.Setup(fractureGameObject.GetComponentsInChildren<Rigidbo
dy>());

        return graphManager;
    }

    private static void AnchorChunks(GameObject gameObject, Anchor
anchor)
    {
        var transform = gameObject.transform;
        var bounds = gameObject.GetCompositeMeshBounds();
        var anchoredColliders = GetAnchoredColliders(anchor, transform,
bounds);

        foreach (var collider in anchoredColliders)
        {
            var chunkRb = collider.GetComponent<Rigidbody>();
            if (chunkRb)
            {
                chunkRb.isKinematic = true;
            }
        }
    }

    private static List<GameObject> BuildChunks(Material insideMaterial,
Material outsideMaterial, List<Mesh> meshes, float chunkMass)
    {
        return meshes.Select((chunkMesh, i) =>
        {
            var chunk = BuildChunk(insideMaterial, outsideMaterial, chunkMesh,
chunkMass);
            chunk.name += $" [{i}]";
            return chunk;
        }).ToList();
```

```
    }

    private static List<Mesh> FractureMeshesInNvblast(int totalChunks,
NvMesh nvMesh)
    {
        var fractureTool = new NvFractureTool();
        fractureTool.setRemoveIslands(false);
        fractureTool.setSourceMesh(nvMesh);
        var sites = new NvVoronoiSitesGenerator(nvMesh);
        sites.uniformlyGenerateSitesInMesh(totalChunks);
        fractureTool.voronoiFracturing(0, sites);
        fractureTool.finalizeFracturing();

        // Extract meshes
        var meshCount = fractureTool.getChunkCount();
        var meshes = new List<Mesh>(fractureTool.getChunkCount());
        for (var i = 1; i < meshCount; i++)
        {
            meshes.Add(ExtractChunkMesh(fractureTool, i));
        }

        return meshes;
    }

    private static IEnumerable<Collider> GetAnchoredColliders(Anchor
anchor, Transform meshTransform, Bounds bounds)
    {
        var anchoredChunks = new HashSet<Collider>();
        var frameWidth = .01f;
        var meshWorldCenter = meshTransform.TransformPoint(bounds.center);
        var meshWorldExtents =
bounds.extents.Multiply(meshTransform.lossyScale);

        if (anchor.HasFlag(Anchor.Left))
        {
            var center = meshWorldCenter - meshTransform.right *
meshWorldExtents.x;
```

```
            var halfExtents = meshWorldExtents.Abs().SetX(frameWidth);
            anchoredChunks.UnionWith(Physics.OverlapBox(center, halfExtents,
meshTransform.rotation));
        }

        if (anchor.HasFlag(Anchor.Right))
        {
            var center = meshWorldCenter + meshTransform.right *
meshWorldExtents.x;
            var halfExtents = meshWorldExtents.Abs().SetX(frameWidth);
            anchoredChunks.UnionWith(Physics.OverlapBox(center, halfExtents,
meshTransform.rotation));
        }

        if (anchor.HasFlag(Anchor.Bottom))
        {
            var center = meshWorldCenter - meshTransform.up *
meshWorldExtents.y;
            var halfExtents = meshWorldExtents.Abs().SetY(frameWidth);
            anchoredChunks.UnionWith(Physics.OverlapBox(center, halfExtents,
meshTransform.rotation));
        }

        if (anchor.HasFlag(Anchor.Top))
        {
            var center = meshWorldCenter + meshTransform.up *
meshWorldExtents.y;
            var halfExtents = meshWorldExtents.Abs().SetY(frameWidth);
            anchoredChunks.UnionWith(Physics.OverlapBox(center, halfExtents,
meshTransform.rotation));
        }

        if (anchor.HasFlag(Anchor.Front))
        {
            var center = meshWorldCenter - meshTransform.forward *
meshWorldExtents.z;
            var halfExtents = meshWorldExtents.Abs().SetZ(frameWidth);
```

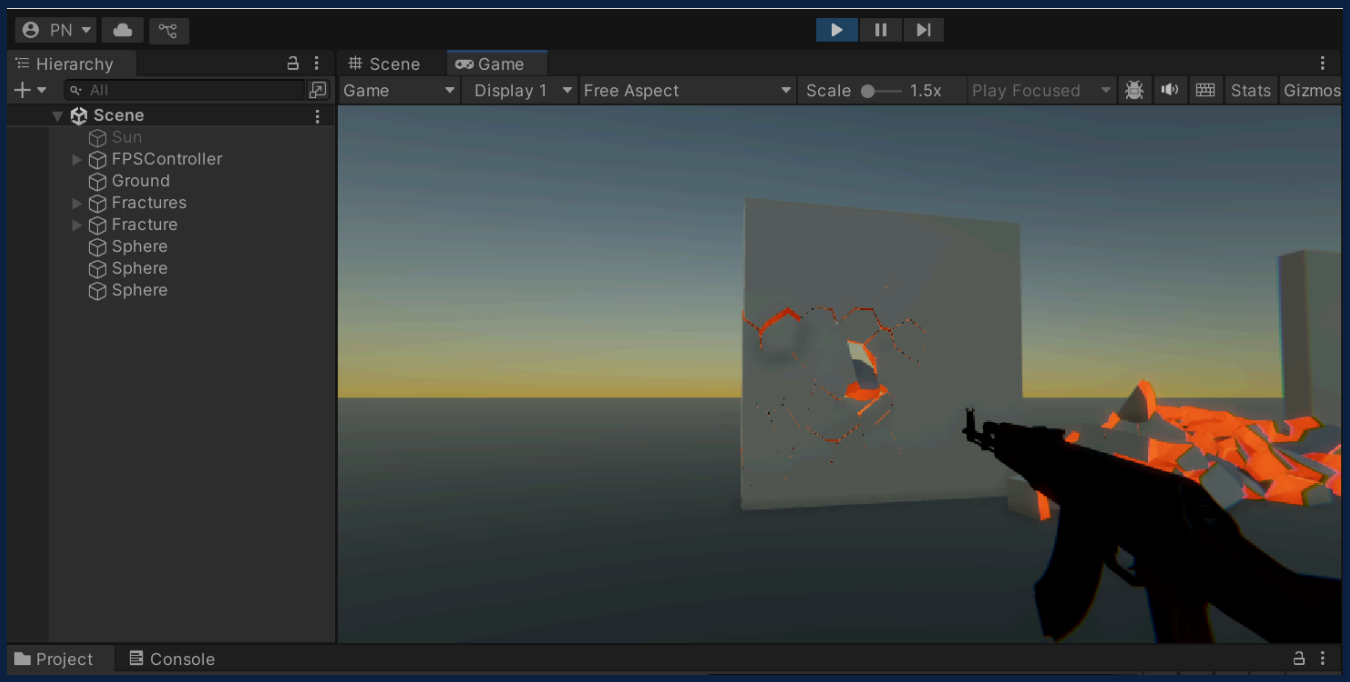
```
            anchoredChunks.UnionWith(Physics.OverlapBox(center,  halfExtents,
meshTransform.rotation));
        }

        if (anchor.HasFlag(Anchor.Back))
        {
            var center = meshWorldCenter + meshTransform.forward *
meshWorldExtents.z;
            var halfExtents = meshWorldExtents.Abs().SetZ(frameWidth);
            anchoredChunks.UnionWith(Physics.OverlapBox(center,  halfExtents,
meshTransform.rotation));
        }

        return anchoredChunks;
    }

    private static Mesh ExtractChunkMesh(NvFractureTool fractureTool, int
index)
    {
        var outside = fractureTool.getChunkMesh(index, false);
        var inside = fractureTool.getChunkMesh(index, true);
        var chunkMesh = outside.toUnityMesh();
        chunkMesh.subMeshCount = 2;
        chunkMesh.SetIndices(inside.getIndexes(), MeshTopology.Triangles, 1);
        return chunkMesh;
    }

    private static Mesh GetWorldMesh(GameObject gameObject)
    {
        var combineInstances = gameObject
            .GetComponentsInChildren<MeshFilter>()
            .Where(mf => ValidateMesh(mf.mesh))
            .Select(mf => new CombineInstance()
            {
                mesh = mf.mesh,
                transform = mf.transform.localToWorldMatrix
            }).ToArray();
```

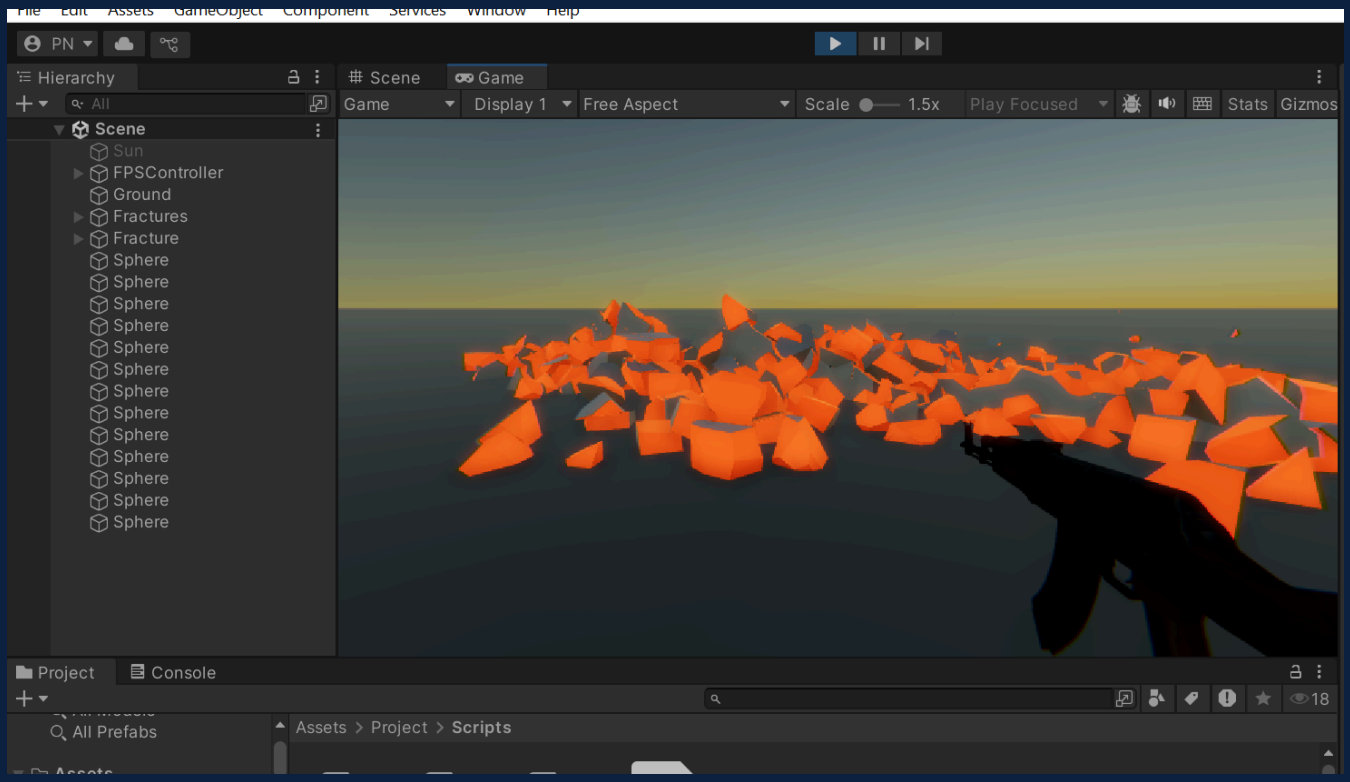
```
        var totalMesh = new Mesh();
        totalMesh.CombineMeshes(combineInstances, true);
        return totalMesh;
    }

    private static bool ValidateMesh(Mesh mesh)
    {
        if (mesh.isReadable == false)
        {
            Debug.LogError($"Mesh [{mesh}] has to be readable.");
            return false;
        }

        if (mesh.vertices == null || mesh.vertices.Length == 0)
        {
            Debug.LogError($"Mesh [{mesh}] does not have any vertices.");
            return false;
        }

        if (mesh.uv == null || mesh.uv.Length == 0)
        {
            Debug.LogError($"Mesh [{mesh}] does not have any uvs.");
            return false;
        }

        return true;
    }

    private static GameObject BuildChunk(Material insideMaterial, Material
outsideMaterial, Mesh mesh, float mass)
    {
        var chunk = new GameObject($"Chunk");

        var renderer = chunk.AddComponent<MeshRenderer>();
        renderer.sharedMaterials = new[]
        {
```

```
            outsideMaterial,
            insideMaterial
        };

        var meshFilter = chunk.AddComponent<MeshFilter>();
        meshFilter.sharedMesh = mesh;

        var rigibody = chunk.AddComponent<Rigidbody>();
        rigibody.mass = mass;

        var mc = chunk.AddComponent<MeshCollider>();
        mc.inflateMesh = true;
        mc.convex = true;

        return chunk;
    }

    private static void ConnectTouchingChunks(GameObject chunk, float
jointBreakForce, float touchRadius = .01f)
    {
        var rb = chunk.GetComponent<Rigidbody>();
        var mesh = chunk.GetComponent<MeshFilter>().mesh;

        var overlaps = new HashSet<Rigidbody>();
        var vertices = mesh.vertices;
        for (var i = 0; i < vertices.Length; i++)
        {
            var worldPosition = chunk.transform.TransformPoint(vertices[i]);
            var hits = Physics.OverlapSphere(worldPosition, touchRadius);
            for (var j = 0; j < hits.Length; j++)
            {
                overlaps.Add(hits[j].GetComponent<Rigidbody>());
            }
        }

        foreach (var overlap in overlaps)
        {
```

```
            if (overlap.gameObject != chunk.gameObject)
            {
                var joint = overlap.gameObject.AddComponent<FixedJoint>();
                joint.connectedBody = rb;
                joint.breakForce = jointBreakForce;
            }
        }
    }
}
```

- **Scene Reference (Unity preview)**




File Edit Assets GameObject Component Services Window Help
PN
Hierarchy
Scene
Game
All
Scene
Sun
FPSController
Ground
Fractures
Fracture
Sphere
Sphere
Sphere
Sphere
Sphere
Sphere
Sphere
Sphere
Sphere
Sphere
Sphere
Sphere
Sphere
Game
Display 1
Free Aspect
Scale
1.5x
Play Focused
Stats
Gizmos
Project
Console
18
All Prefabs
Assets > Project > Scripts