EarClipper – Polygon Triangulation for Unity

EarClipper is a lightweight and efficient triangulation algorithm for simple 2D or projected 3D polygons using the ear clipping method. Designed specifically for Unity, it converts vertex lists into triangle indices suitable for mesh generation in both runtime and editor tools.

📚 In this document:

🚀 Features

- Triangulates simple, non-self-intersecting polygons
- Supports 3D input – auto-projects to 2D for processing
- Handles polygons with holes via basic merging
- Outputs triangles in Unity-friendly `int[]` format
- Optional triangle flipping for winding order control
- Minimal and fast – no external dependencies

🛠️ How To Use

Using **EarClipper** in your Unity project is simple. Here's a quick example:

1. **Define your polygon** – Create an array of `Vector3` points in **clockwise** order.
2. **Create an EarClipper instance** – Pass the points to the constructor.
3. **Generate triangles** – Call `.Triangulate()` to get an array of triangle indices.
4. **Build your mesh** – Use Unity's `Mesh` class to create a mesh with the vertices and triangles.
5. **Assign to MeshFilter** – Attach the mesh to any GameObject with a `MeshFilter`.

🛠️ Create Meshes with Holes

To create a mesh with one or more holes, you simply provide both the outer boundary and the holes as input. The **outer polygon** must be ordered **clockwise**, while each **hole** must be ordered **counter-clockwise** for correct merging and triangulation.

The `EarClipper` constructor will automatically merge all holes into the outer shape using a simple visibility-based method, returning a merged vertex array that can be used directly in Unity's `Mesh` system.

Examples

In this example we create a simple plane and assign it to the object's MeshFilter

```
1 reference
private void CreateMesh()
{
    // Define the vertices of a simple polygon.
    // IMPORTANT: The points must be ordered in **clockwise** order for proper triangulation.
    Vector3[] points = new Vector3[] {
        new Vector3(0, 0, 0),
        new Vector3(-1f, 0, 0),
        new Vector3(-1f, 1f, 0),
        new Vector3(0, 1f, 0)
    };

    // Create a new instance of the EarClipper triangulator with the defined points.
    EarClipper earClipper = new(points);

    // Generate triangle indices using the ear clipping algorithm.
    int[] triangles = earClipper.Triangulate();

    // Create a new Unity mesh and assign the vertices and triangles.
    Mesh mesh = new();
    mesh.vertices = points;
    mesh.triangles = triangles;

    // Recalculate normals and bounds to ensure correct rendering and physics interaction.
    mesh.RecalculateNormals();
    mesh.RecalculateBounds();

    // Assign the generated mesh to this object's MeshFilter component.
    GetComponent<MeshFilter>().mesh = mesh;
}
```

In this example we create a mesh with one or more holes

```
1 reference
private void CreateMeshWithHoles()
{
    // Define the outer boundary of the polygon.
    // Must be ordered in **clockwise** direction.
    Vector3[] points = myBaseMesh.points;

    // Define the holes inside the polygon.
    // Each hole must be defined in **counter-clockwise** order.
    List<Vector3[]> holes = new();
    myHoles.ForEach(h => holes.Add(h.points));

    // Create an EarClipper instance that will automatically merge the holes with the outer shape.
    // The merged result is returned as a new array of points.
    EarClipper earClipper = new(points, holes, out var merged);

    // Create a Unity Mesh and assign the merged vertices.
    Mesh mesh = new();
    mesh.vertices = merged;

    // Generate triangle indices from the merged polygon structure.
    // Set 'flipFaces' to true if you want to invert the triangle winding order (If you want to flip the face of your mesh).
    mesh.triangles = earClipper.Triangulate(flipFaces);

    // Generate UVs for the mesh (implementation of GenerateUV is user-defined).
    mesh.uv = GenerateUV(merged);

    // Recalculate normals and bounds for correct lighting and rendering.
    mesh.RecalculateNormals();
    mesh.RecalculateBounds();

    // Assign the mesh to the MeshFilter component of this GameObject.
    GetComponent<MeshFilter>().mesh = mesh;
}
```