

2D Destruction Pro

User Guide | ERAGANX (Charith Eranga)

A Unity 2D toolkit for authoring pre-sliced sprite fragments and cutting supported sprites during gameplay, then sending generated pieces through a shared mesh, collider, physics, pooling and reassembly workflow.

What you can build

- Breakable props and crates using Auto Grid or authored Manual Slice layouts.
- Custom cuttable objects with straight cuts or artist-authored polygon regions.
- Fruit-Ninja-style gameplay using runtime line, segment and ordered path cuts.
- Gameplay rules that score, filter or react to fragments using area ratios and ancestry.
- Reusable fragment PNG Sprites and a combined prefab containing all authored pieces.



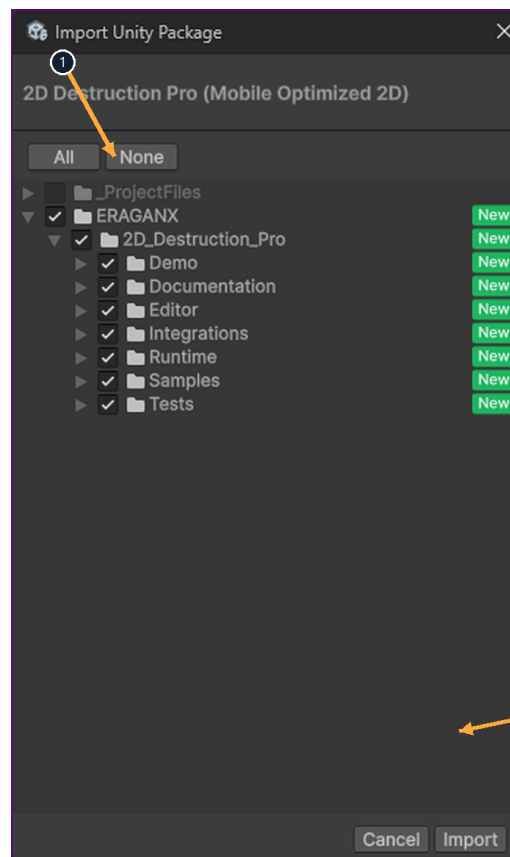
Installation and project setup

01 Installation

For the Asset Store edition, open Unity Package Manager > My Assets, download 2D Destruction Pro, then import it into your Unity project. If using a source checkout, copy the complete package folder with its .meta files; do not copy only the C# files.

02 Opening and importing the package

Open the project in Unity, wait for package import and script compilation to finish, and resolve any project-level package errors before using the tools. In the import dialog, keep the package Runtime, Editor, Samples, Documentation, Tests and Integrations folders that you intend to use.



Package import dialog supplied with the project; it shows the package tree and Import action.

1

Import list: review the selected package folders and files.

2

Import: adds the selected package content to the project. Include the complete package structure and metadata.

03 Required Unity version

The validated Editor baseline is Unity 6000.3.13f1. The current source uses Unity 6 APIs; other Unity versions have not been validated by this project.

04 Dependencies

Core slicing/destruction does not require URP, the Input System, Burst or Collections. Jobs/Burst is optional and requires compatible Burst 1.8+, Collections 2.x and Mathematics packages. Without the optional integration, HighPerformance mode falls back to Managed C#. The old click tutorial optionally uses Input System; Runtime Swipe does not. Unity Test Framework is needed only to run the package test assemblies.

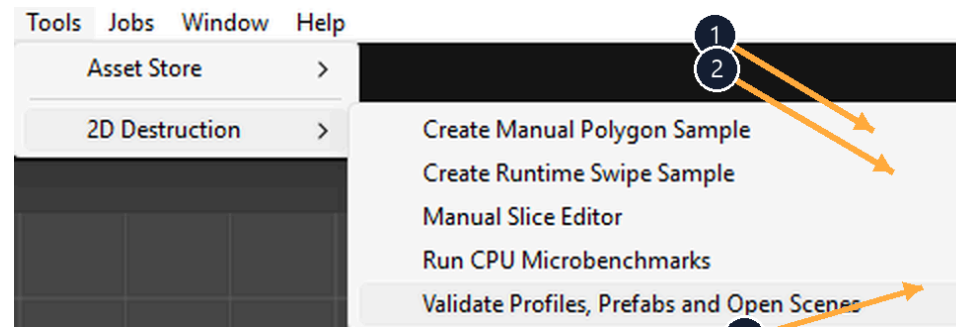
Find the package and tools

05 Where the package appears in the Project window

After import, locate `Assets/ERAGANX/2D\_Destruction\_Pro` in the Project window. Samples are in `Samples/ManualPolygon` and `Samples/RuntimeSwipe`. Legacy tutorial scenes are under `Demo/Scenes`.

06 Find the main tools

Open the Unity top menu `Tools > 2D Destruction` for the Manual Slice Editor, sample creation commands, CPU microbenchmarks and project validator. The Destructible 2D Inspector also has an `Open Manual Slice Editor` button when a suitable scene object is selected.



The supplied Unity menu screenshot with the package commands.

- 1 Sample creation commands for the Manual Polygon and Runtime Swipe examples.
- 2 Manual Slice Editor command.
- 3 Run CPU Microbenchmarks and Validate Profiles, Prefabs and Open Scenes. The validator scans; it does not change project assets.

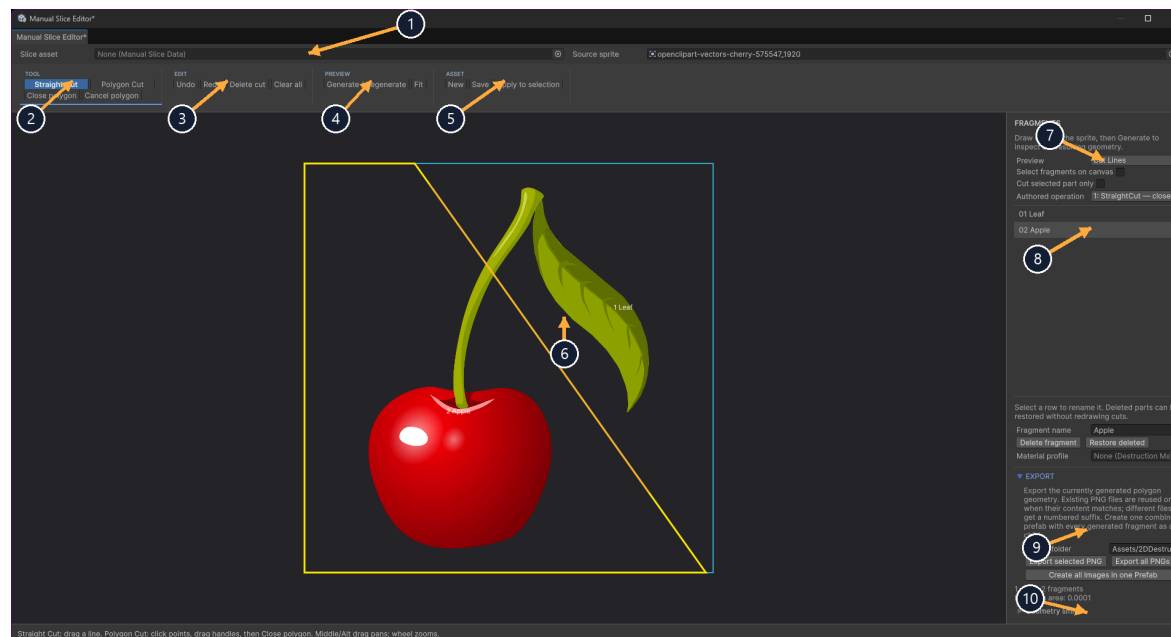
07 Open the Manual Slice Editor

Select a Sprite asset, then choose `Tools > 2D Destruction > Manual Slice Editor`. You can also select a scene object with the Destructible 2D component and use `Open Manual Slice Editor`.

Use the Manual Slice Editor

08 Editor workflow and panels

Assign a `Source sprite` to start authoring, or load an existing `Slice asset`. Create operations, generate geometry, inspect fragments, name them, then Save. The editor is an Edit Mode authoring tool; it does not perform the gameplay Fruit-Ninja cut.



Manual Slice Editor overview. The numbered callouts match the legend below.

- 1 Slice asset and Source sprite: load existing ManualSliceData or choose the geometry source.
- 2 TOOL: Straight Cut and Polygon Cut. Close polygon commits a valid draft; Cancel polygon discards the current draft.
- 3 EDIT: Undo, Redo, Delete cut and Clear all. Deleting a cut removes an authored operation; deleting a fragment is separate.
- 4 PREVIEW: Generate / Regenerate rebuilds geometry; Fit restores the canvas framing.
- 5 ASSET: New starts a blank configuration; Save stores ManualSliceData; Apply to selection assigns saved data to the selected scene SpriteRenderer.
- 6 Canvas: draw cuts, edit vertices and inspect generated geometry. Middle or Alt drag pans; mouse wheel zooms.

- | | |
|----|--|
| 7 | Preview choices: Original, CutLines, Fragments or Colliders. |
| 8 | Fragments panel: preview settings, canvas selection, Cut selected part only, Authored operation and the fragment list. |
| 9 | Fragment name, Delete fragment/Restore deleted, Material profile and EXPORT controls. |
| 10 | Geometry limits: manual generation limits such as Maximum fragments, Maximum vertices, Maximum cuts, Maximum depth and Minimum area. These are separate from runtime limits. |

Author straight and polygon cuts

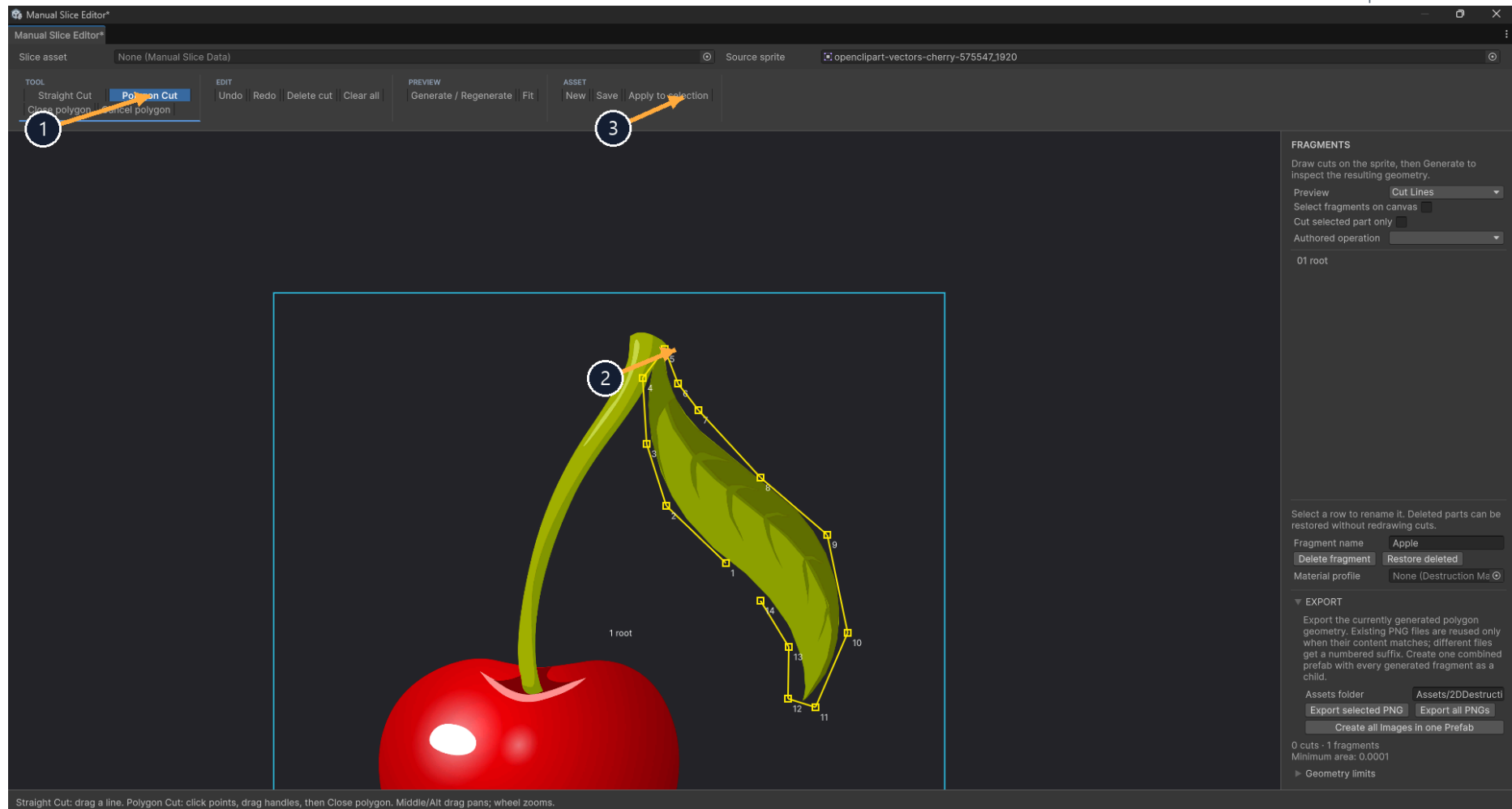
09 Create Straight Cuts

Choose `Straight Cut`, press and drag from a start point to an end point across the sprite. Each drag adds a straight authoring operation across relevant geometry. You can add multiple straight cuts to build the desired layout.

10 Create Polygon/Region Cuts

Choose `Polygon Cut`, click to add three or more vertices, and inspect the connecting edges. Drag a vertex handle to adjust the draft. Choose `Close polygon` to validate and commit, or `Cancel polygon` to discard it. Self-intersecting or zero-area outlines are rejected with a status message.

The geometry inside a closed region becomes a separate logical fragment; the surrounding geometry remains as the remainder. Concave polygons are supported and may be decomposed into convex mesh/collider pieces. A polygon fully inside the sprite leaves a hole in the remainder collider. Disconnected remainder islands may remain grouped as one logical fragment.



Polygon Cut draft from the supplied Editor screenshot; callouts show the selected mode and vertex editing area.

- 1 Polygon Cut and close/cancel controls.
- 2 Numbered vertex handles and connecting edges; drag a handle before closing to edit the boundary.
- 3 The fragment/preview panel used to inspect the generated result.

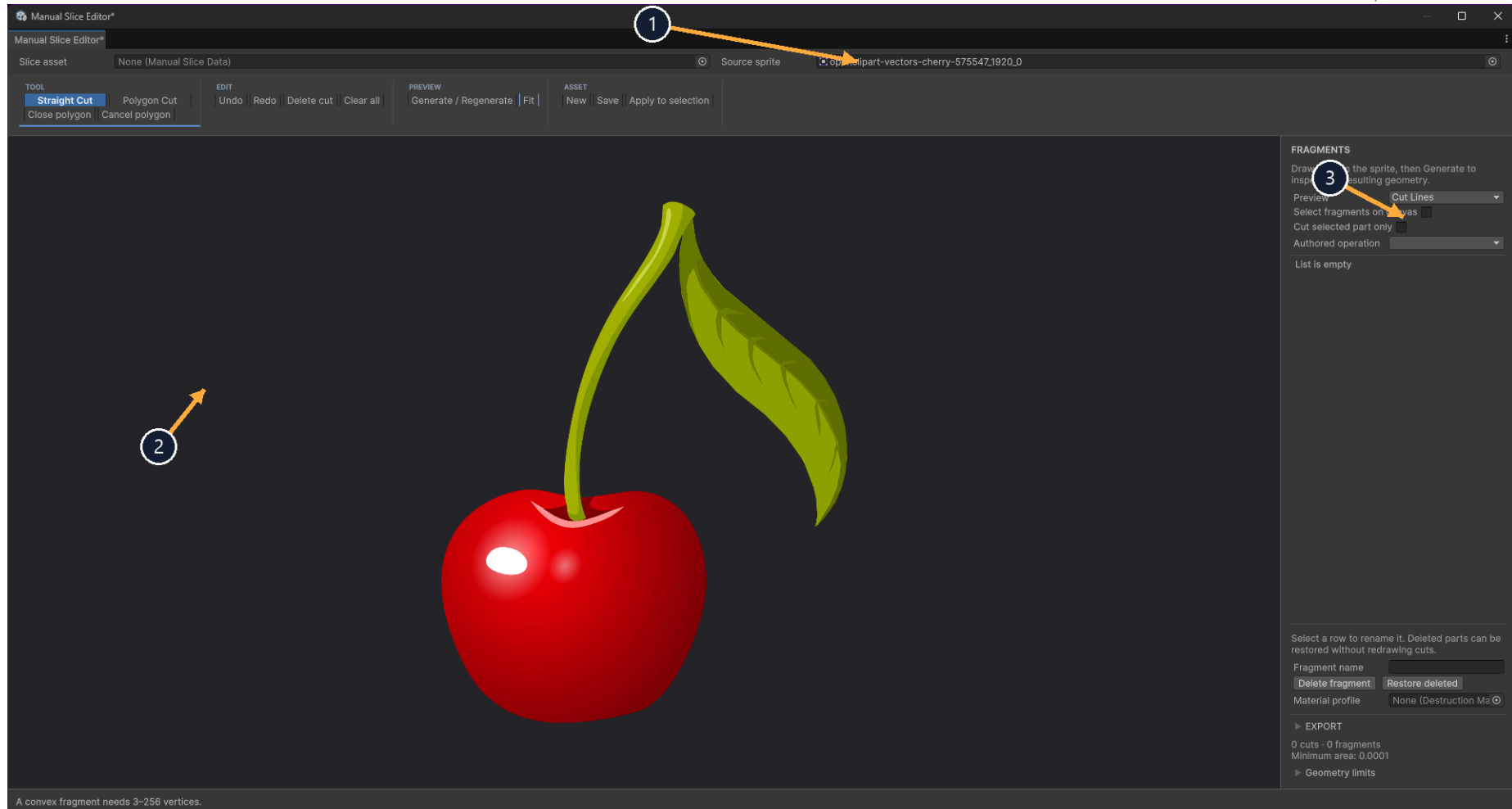
Select, edit and preview fragments

11 Select and edit fragments

After generation, select a row in the Fragments list or enable `Select fragments on canvas` and click a region. The `Fragment name` field edits the selected logical fragment name. `Delete fragment` excludes that region; `Restore deleted` re-enables it. Use `Authored operation` to select an operation for `Delete cut`. `Cut selected part only` limits the next authored cut to the selected piece.

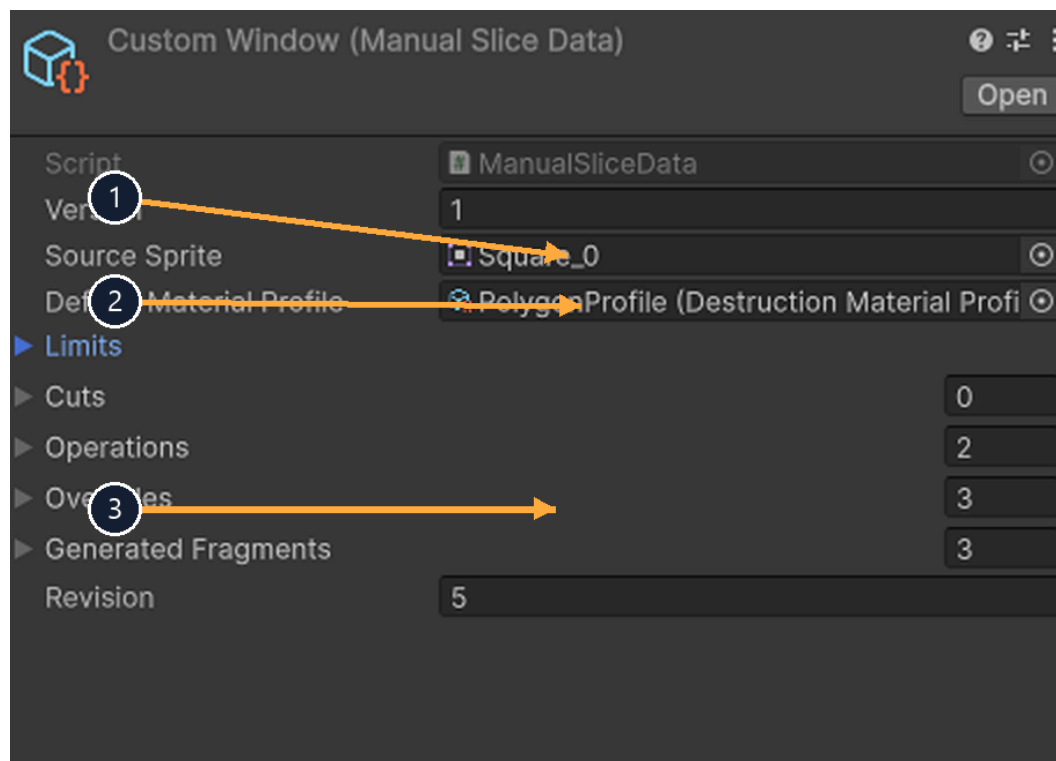
12 Preview and regenerate fragments

Choose `Original`, `CutLines`, `Fragments` or `Colliders` in the Preview selector. `Generate / Regenerate` deterministically rebuilds pieces from the source and authored operations. Inspect invalid-geometry status and fragment count before saving. The Colliders view helps compare collision paths with the artwork.



Uncut source preview from the supplied Editor screenshot.

- 1 Source sprite field.
- 2 Original artwork displayed on the authoring canvas before a cut.
- 3 Preview settings and fragment controls; generated results appear after Generate / Regenerate.



Saved ManualSliceData asset

ManualSliceData asset Inspector showing saved source/profile and data groups.

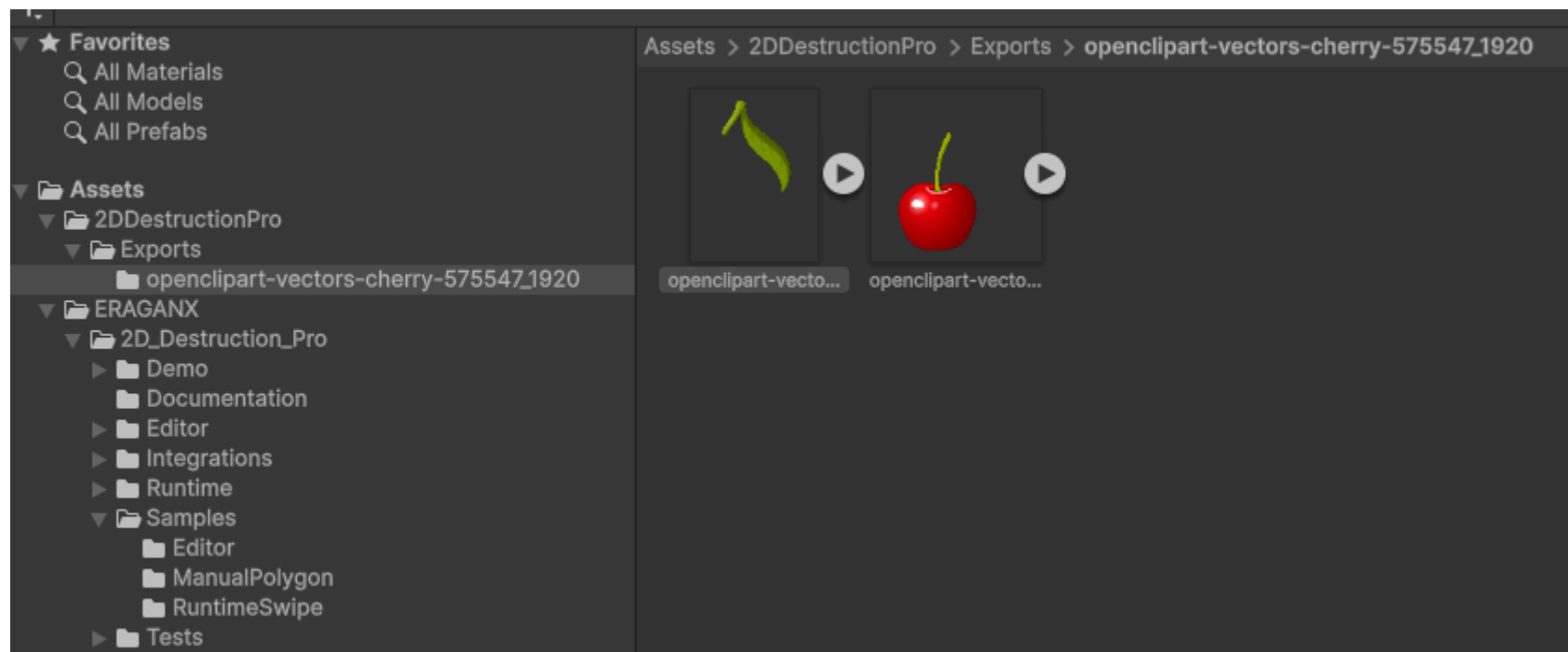
- 1 Source Sprite and optional Default Material Profile.
- 2 Limits, Cuts, Operations, Overrides and Generated Fragments are stored as separate groups.

Export fragment images and prefab

13 Export PNG

Select a generated fragment and click 'Export selected PNG', or click 'Export all PNGs' to export all active generated fragments. 'Assets folder' must be a project-relative folder inside Assets. The default is 'Assets/2DDestructionPro/Exports/<SourceSprite>'. Filenames use the source and fragment index/name. Existing different image content is preserved and a numbered suffix is used; identical content may be reused.

The exporter rasterizes the fragment mesh triangles and UV mapping into a tightly cropped transparent PNG Sprite. It preserves source pixels-per-unit and applicable filtering/wrap/import settings. Export requires a graphics device for Unity render/readback; a headless run may fail.



The supplied screenshot shows generated PNG Sprite assets in the Project window.

- 1 Output folder organized under the source sprite.
- 2 Exported fragment Sprite assets imported by Unity.

Create and use the combined prefab

14 Create a Fragment Prefab

Click `Create all Images in one Prefab`. The editor creates or reuses a PNG Sprite for each generated fragment, then saves one `\_Sliced.prefab`. If the prefab filename already exists, it adds a numbered suffix rather than silently replacing the existing asset.

The root GameObject is named after the source with a sliced suffix. Each authored fragment is a child at its source-local pivot/geometry position. All child images together reproduce the original arrangement. Every child receives a SpriteRenderer, polygon collider matching the fragment geometry, Rigidbody2D and the existing destruction/pool components required by the package. The saved default Material Profile is assigned when available. Sorting layer/order is not stored in ManualSliceData; set it on the prefab if needed.

The supplied screenshots show the export controls and PNG assets, but do not show a generated prefab hierarchy. The structure described here follows the current export implementation.

15 Use the generated prefab

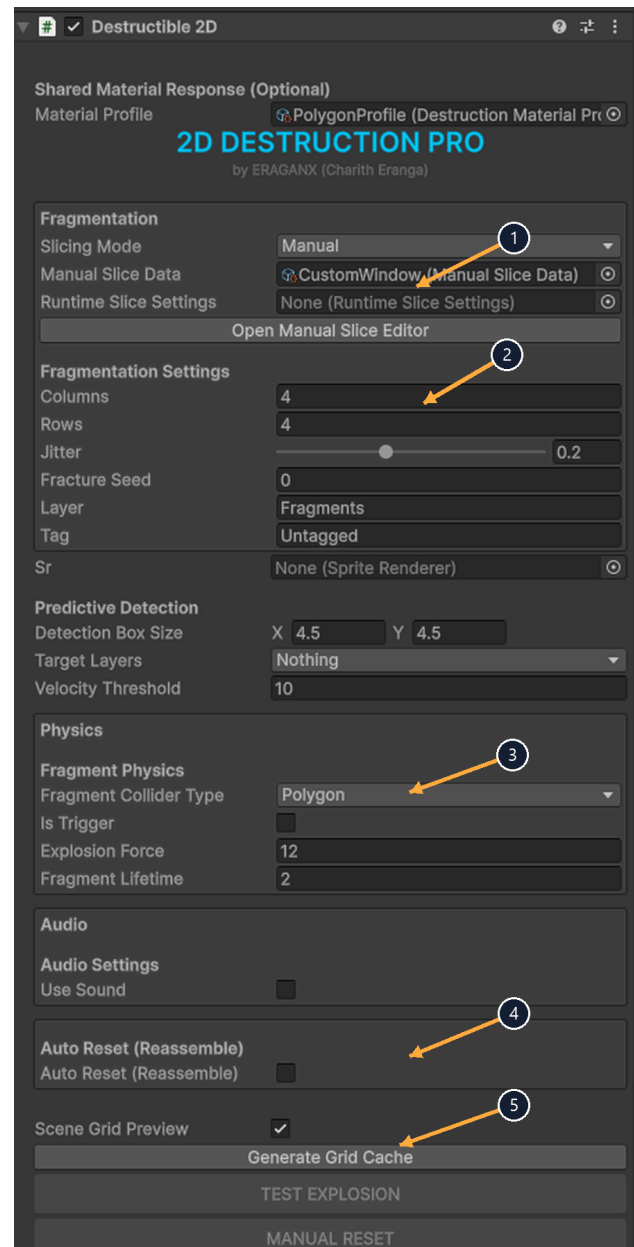
Drag the prefab into a scene or instantiate it from gameplay. Each child is an independent destructible object and can use the existing Explode/ApplyImpact and Reassemble entry points. Reassemble restores that child's exported fragment sprite; it does not reconnect siblings or reconstruct the unsliced source. Children with holes or disconnected regions may be valid authored fragments but are outside current Runtime Slice source geometry support.

```
GameObject sliced = Instantiate(slicedPrefab, position, rotation);  
// Select a child Destructible 2D component, then call Explode()  
// or ApplyImpact(...); call Reassemble() to restore that child.
```

Configure Runtime Slice and the Interactor

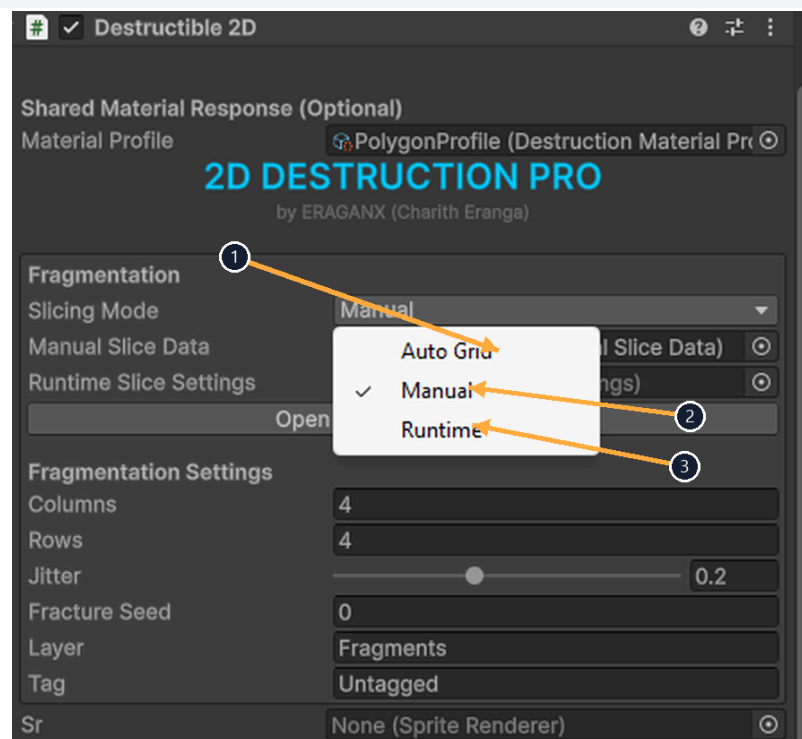
16 Configure Runtime Slice

Select a suitable Sprite GameObject with `SpriteRenderer`, add the package's `Destructible 2D` component, and set `Slicing Mode` to `Runtime`. Create `Runtime Slice Settings` from `Create > ERAGANX > 2D Destruction > Runtime Slice Settings` and assign it. Choose a Material Profile and physics mode. Polygon colliders follow fragment shapes; Box colliders approximate bounds; None disables fragment collider contacts.



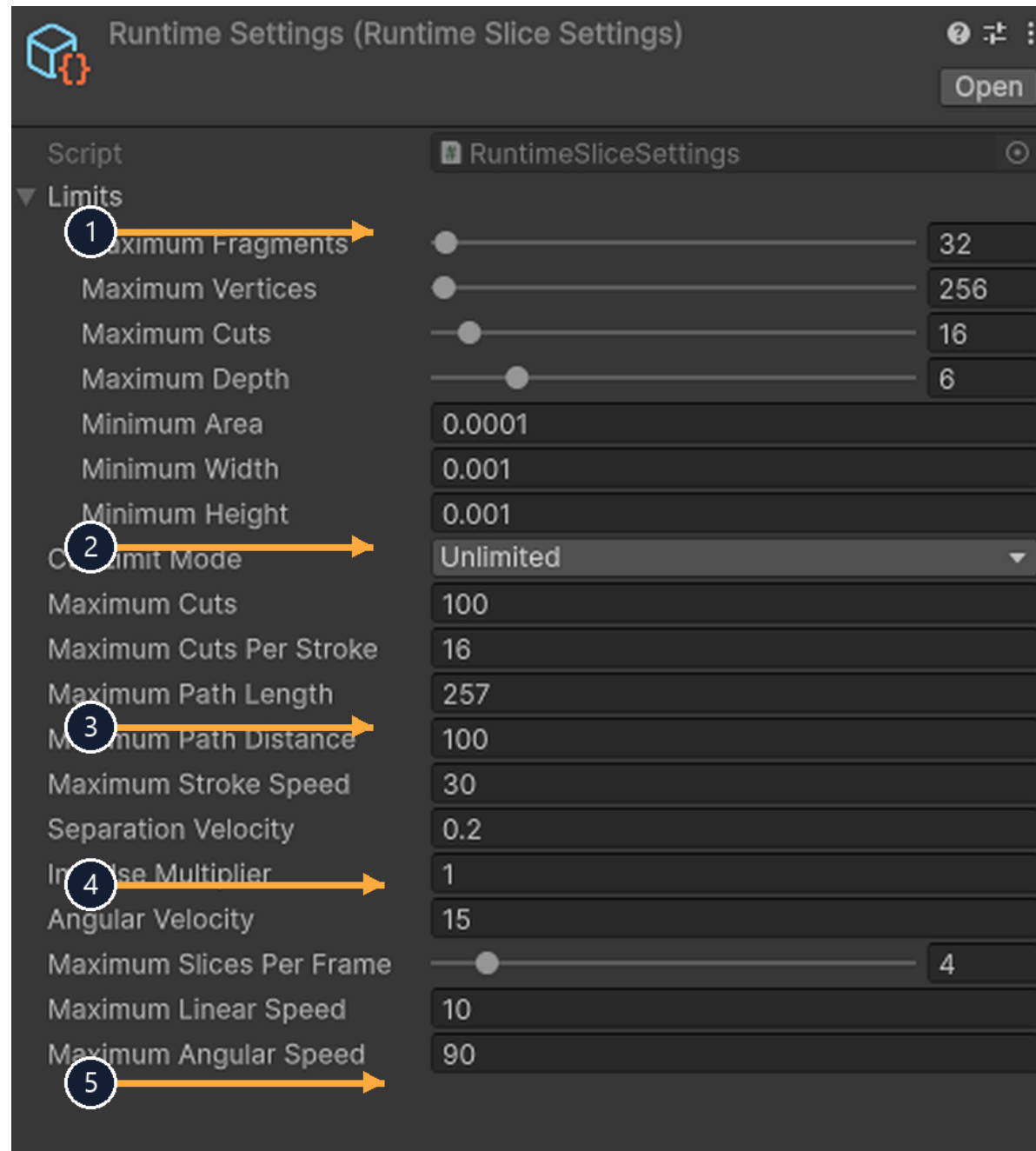
Destructible 2D Inspector. Values in a screenshot are examples, not universal recommended settings.

- 1 Material Profile assignment.
- 2 Slicing Mode, Manual Slice Data, Runtime Slice Settings and Manual Slice Editor access.
- 3 Columns/Rows/Jitter/Fracture Seed apply to Auto Grid.
- 4 Fragment Collider Type and physics-related settings.
- 5 Auto Reset, Scene Grid Preview and cache/play test controls.



Available slicing modes in the Inspector.

- 1 Auto Grid.
- 2 Manual.
- 3 Runtime.



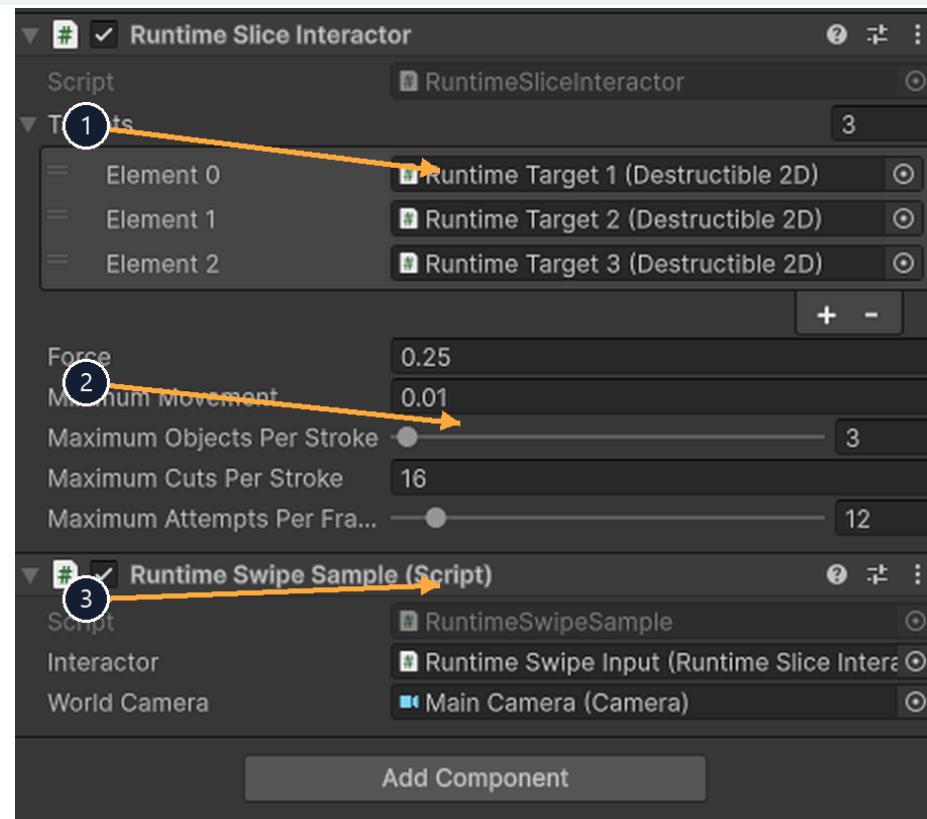
Runtime Slice Settings fields from the supplied screenshot.

- 1 Maximum Fragments and Maximum Vertices bound concurrent fragments and geometry.
- 2 Maximum Cuts and Maximum Depth bound gameplay cuts and recursion.
- 3 Minimum Area/Width/Height reject tiny results.
- 4 Cut Limit Mode selects Unlimited or a finite cut policy.
- 5 Path, speed, impulse and per-frame controls bound input and motion.

17 Use the Runtime Slice Interactor

Add `Runtime Slice Interactor`, add destructible objects to its `Targets` list, and configure stroke force and per-stroke budgets. Convert mouse/touch/weapon screen input into world positions before sending it. The interactor does not search the scene automatically.

```
interactor.BeginStroke(worldPressPosition);
interactor.UpdateStroke(worldDragPosition); // repeat while moving
interactor.EndStroke(worldReleasePosition);
// CancelStroke() if input or focus is interrupted.
```



Runtime Slice Interactor and Runtime Swipe sample component settings.

- 1 Explicit target list.
- 2 Force, movement threshold and per-stroke object/cut budgets.
- 3 Per-frame attempt budget and sample references to its interactor/world camera.

Slice limits and Unlimited mode

18 How slice limits work

- `MaximumTotalCuts` counts split operations across destructibles sharing a `RuntimeSliceSettings` asset during the play session.
- `MaximumCutsPerOriginalObject` counts the original object and descendants until reassembly; the safe default is finite (16).
- `MaximumCutsPerFragment` counts the cut lineage for a fragment and its descendants. `MaximumDepth` is a separate generation limit.
- `MaximumCutsPerStroke` bounds successful splits in a stroke. `MaximumObjectsPerStroke` counts target objects; `MaximumAttemptsPerFrame` counts geometry requests per frame.
- `MaximumFragments` is the active fragment cap. `MaximumVertices` bounds generated geometry. `FragmentPool.MaximumSize` caps total leased plus available pool entries. These values do not mean the same thing.
- `MaximumPathLength/MaximumPathDistance` bound `SlicePath` or swipe distance; `MaximumStrokeSpeed` limits interactor movement. `Minimum Area/Width/Height` reject tiny pieces.
- `RuntimeSliceSettings.maximumCutsPerStroke` and interactor `maximumCutsPerStroke` are stroke budgets. `ManualSliceData.MaximumCuts` is only an authored-operation limit, not the runtime policy.

19 How Unlimited mode works

Unlimited disables configurable gameplay cut-count policies. It does not disable catastrophic protection: a hard per-settings session safeguard of 65,536 splits remains, along with configured active fragment, vertex, depth, area, path, speed, per-frame and pool limits. Pool capacity will not grow beyond its maximum. If preflight fails, the source object must remain unchanged and the result reports a useful failure status/reason.

Fragment ratios and original/parent lineage

20 How fragment ratios work

Use the generated polygon solid area, not a bounding-box estimate. `Area` is computed from polygon geometry in source-sprite local space. Texture alpha is not sampled: transparent pixels inside a Full Rect sprite mesh count as geometry. A tight/custom sprite outline excludes its empty exterior.

```
RuntimeSliceResult result = target.SliceSegmentDetailed(worldStart, worldEnd, force: 2f);
if (result.Success)
{
    foreach (RuntimeFragmentInfo fragment in result.Fragments)
        Debug.Log($"Original {fragment.OriginalRatio:P1}; parent {fragment.ParentRatio:P1}");
}
```

21 How original and parent lineage work

- OriginalRatio = fragment area / original source polygon area: “How much of the original sprite am I?”
- ParentRatio = fragment area / immediate parent polygon area: “What share of the fragment that was just cut am I?”

Example: original = 100%. First cut yields A = 30%, B = 70%. Cutting B again yields B1 = 25% and B2 = 45% of the original. B1 reports OriginalRatio = 0.25 and ParentRatio = $25/70 \approx 0.357$. B2 reports OriginalRatio = 0.45 and ParentRatio = $45/70 \approx 0.643$.

Each result exposes FragmentId, ParentFragmentId, OriginalSourceId, Generation, Area and Bounds. Detailed slice results also expose world CutPosition, source-local LocalPosition, NormalizedSpritePosition, CutDirection and line endpoints for gameplay feedback, scoring, sound or analytics.

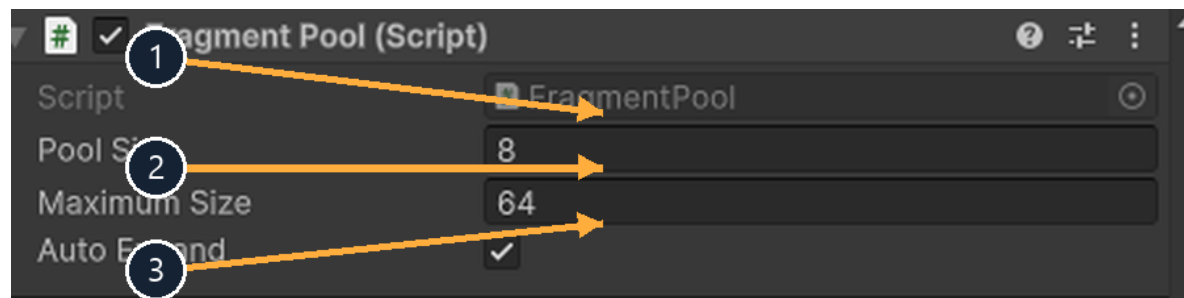
22 How reassembly works

`Reassemble()` restores the original source object and invalidates active runtime fragment handles. It is restoration, not recorded time-reverse playback. Expired fragments cannot animate back; the source still restores. After reassembly the prior ancestry tree is cleared, so discard old runtime fragment references.

Pooling, Jobs and Burst

23 How pooling works

'FragmentPool' prewarms reusable fragment objects, can expand when Auto Expand is enabled, and stops at Maximum Size. Pool capacity is total available plus leased entries. It is different from the active-fragment cap and cut counters. Configure a finite capacity that supports expected simultaneous fragments. A pool exhaustion rejection should happen before the source is changed.

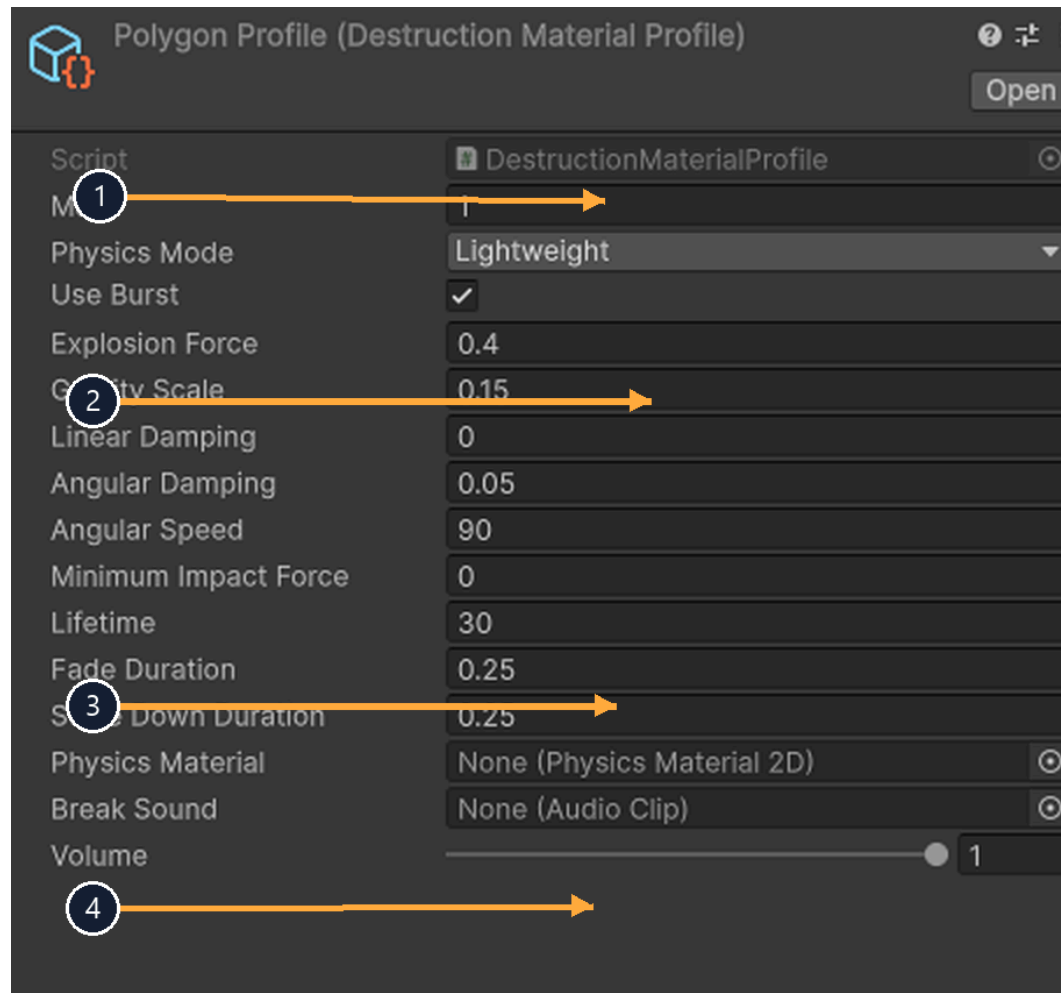


FragmentPool Inspector fields.

- 1 Pool Size is the prewarm count.
- 2 Maximum Size is the hard pool capacity.
- 3 Auto Expand allows growth up to that capacity.

24 Jobs/Burst and fallback

Rigidbody2D uses Unity collision/contact response. Lightweight uses Managed C# ballistic movement and does not resolve fragment collisions. HighPerformance requests the optional Jobs backend; if it is unavailable or not registered, the system falls back to Managed C#. Use Burst is relevant only when Jobs integration and compatible Burst dependencies are available.



Material Profile physics and lifetime fields.

- 1 Mass and Physics Mode.
- 2 Burst toggle and motion settings.
- 3 Lifetime, fade and scale-down values.
- 4 Optional physics material, break sound and volume.

Jobs/Burst applies to fragment movement data. Runtime polygon splitting, mesh generation, collider construction and Unity object/pool updates remain in the common managed/main-thread pipeline. Managed and Jobs movement is ballistic, unlike Rigidbody2D contacts. Do not infer guaranteed FPS, zero GC or mobile performance from Editor tests.

Developer API and events

25 Use the developer API

The component provides ``SliceLine``, ``SliceSegment`` and ``SlicePath`` for simple status/split counts, and ``SliceLineDetailed``, ``SliceSegmentDetailed`` and ``SlicePathDetailed`` when gameplay needs metadata. Line uses an infinite straight line; Segment must cross both sides of the piece; Path processes an ordered series of straight segments, not a curved boolean cut.

```
RuntimeSliceResult result = target.SliceSegmentDetailed(start, end, force: 2f);
if (!result.Success)
{
    Debug.Log(result.Failure + ": " + result.Reason);
    return;
}
foreach (RuntimeFragmentInfo f in result.Fragments)
{
    if (f.OriginalRatio < 0.25f) { /* game-specific rule */ }
    Debug.Log($"{f.FragmentId} parent={f.ParentFragmentId}, generation={f.Generation}");
}
```

``RuntimeSliceResult`` includes `Success`, `SourceObject`, `OriginalSourceId`, cut positions and direction, `LineStart/LineEnd`, `IsFiniteSegment`, `Failure`, `Reason` and the new `Fragments`. ``RuntimeFragmentInfo`` exposes `Fragment`, `IDs`, `OriginalRatio`, `ParentRatio`, `Area`, `Bounds` and `Generation`. `Fragment` handles are valid only while that pooled piece remains active. ``TryGetRuntimeFragment`` and ``TryGetParent`` can resolve current-cycle ancestry.

26 Subscribe to slice and reassembly events

Detailed events are synchronous on the main thread. Subscribe in `OnEnable` and unsubscribe in `OnDisable`. Keep handlers brief. Exceptions are isolated/logged; recursive same-object slicing from a slice callback is rejected.

```
private void OnEnable()
{
    target.OnSliceCompleted += HandleSlice;
    target.OnReassembleRequested += HandleReassemble;
}
private void OnDisable()
{
    target.OnSliceCompleted -= HandleSlice;
    target.OnReassembleRequested -= HandleReassemble;
}
```

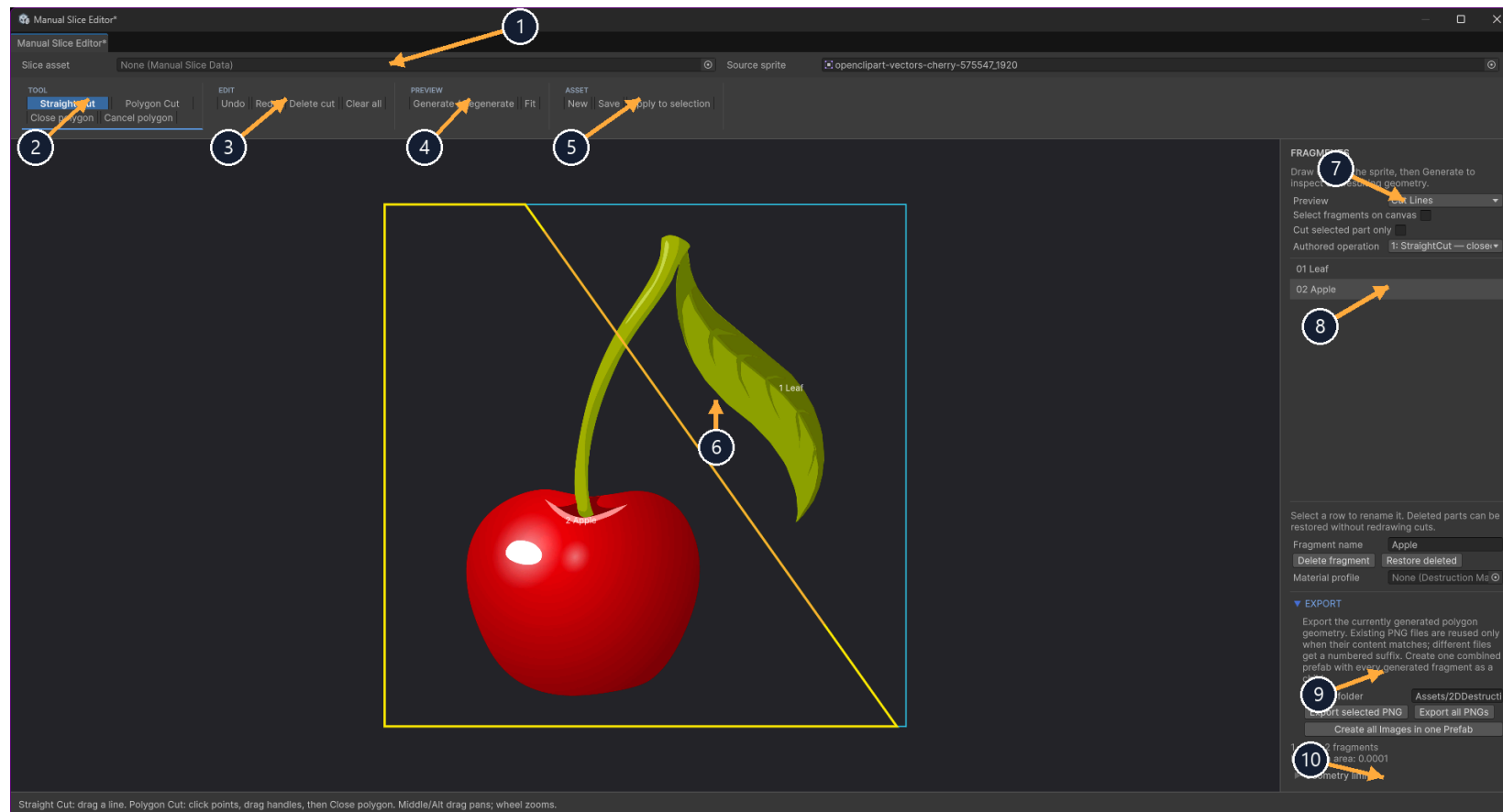
- `OnSliceStarted`: request processing begins. `OnSliceValidated`: geometry, limits and pool preflight passed; live state is about to change. `OnSliceCompleted`: result/metadata committed. `OnFragmentCreated`: each new child after completion. `OnSliceFailed`: rejected request. `OnReassembleRequested`: restoration begins before handles retire.

Existing whole-object lifecycle events include `Destroyed` and `Restored`. Slice events carry a `RuntimeSliceEvent` with `Kind`, `Result`, and, for fragment creation, `Fragment`. `OnSliceValidated` signals successful preflight, not that unrelated application callbacks cannot subsequently interrupt game logic.

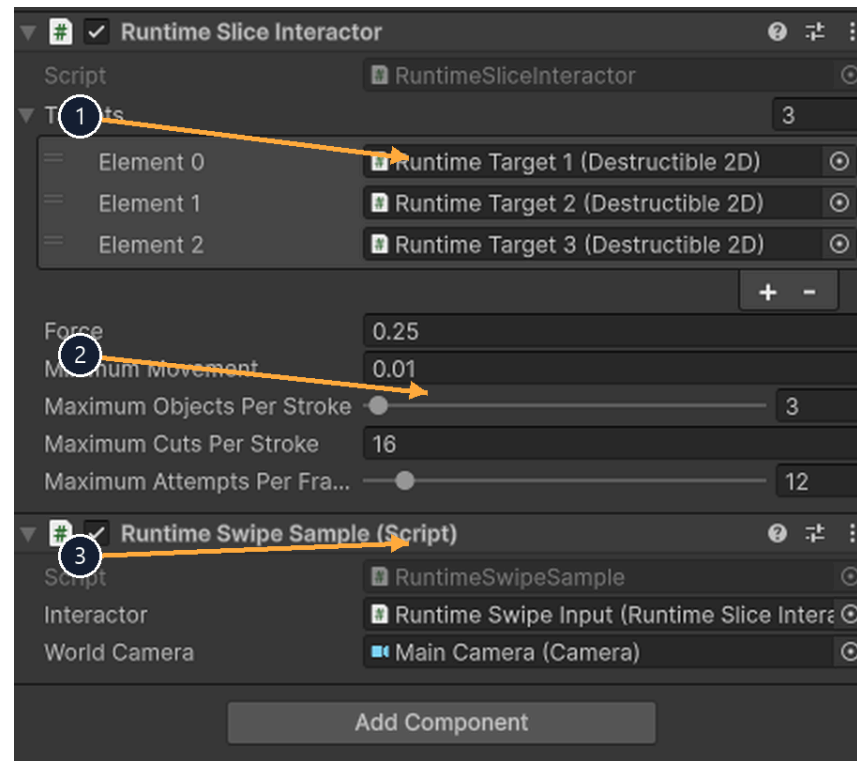
Samples, common mistakes and troubleshooting

27 Use the included samples

- `Samples/ManualPolygon/ManualPolygon.unity`: saved polygon/region authoring, named fragments and Reassemble. The `CustomWindow.asset` can be opened in the Manual Slice Editor.
- `Samples/RuntimeSwipe/RuntimeSwipe.unity`: mouse-driven swipes against three targets, repeated cuts, fragment physics, cut/readout metadata and Reassemble all. Drag from outside one edge to outside the other.
- `Demo/Scenes/Tute 1 - Click to Destroy`, `Tute 2 - Destroy Obstacles`, and `Tute 3 - Player Respawn` are legacy lifecycle tutorials; they are not current Manual/Runtime Slice workflows. Tute 1 click input is optional and requires Input System for interaction.



The supplied Manual Slice Editor screenshot can be compared with the Manual Polygon sample workflow.



The supplied Runtime Swipe Inspector capture shows the explicit targets and sample references.

28 Common mistakes

- A finite SliceSegment starts or ends inside the object. Use endpoints that cross both polygon boundaries; use SliceLine for an infinite line.
- Runtime Slice has no targets. Assign the objects explicitly to the Interactor Targets list.
- Assuming ManualSliceData makes a concave source valid for Runtime Slice. Runtime source geometry still requires a supported single convex contour.
- Expecting Full Rect geometry to ignore transparent pixels. Ratios follow mesh geometry, not texture alpha.
- Treating pool capacity, active fragments, vertices and cut limits as one budget. They are independent.
- Expecting Lightweight/Jobs fragments to collide like Rigidbody2D. These backends are ballistic.
- Saving while a Polygon Cut draft is still open. Close or cancel it first.

Troubleshooting

29 Troubleshooting

Package does not compile

No runtime intersection

LimitReached / PoolExhausted

Invalid polygon

PNG export fails

Prefab export fails

Jobs/Burst unavailable

Collider differs from image

Editor UI resources missing

Runtime Slice supports straight lines, finite segments and ordered straight segment paths. It does not currently provide general curved cutting, blade thickness, arbitrary concave/holey source boundaries, or full Lightweight collision response. For more details, open the package Documentation folder and consult Runtime-Slicing.md, Manual-Slice-Editor.md, Slicing-Modes.md and Jobs-and-Burst.md.

Check Unity 6000.3.13f1 baseline, complete package folder and .meta files. If using Jobs, confirm compatible optional dependencies.

Check world-space endpoints; a finite segment must cross both sides. Verify object transform and source bounds.

Read LastRuntimeSliceResult Failure/Reason. Check RuntimeSliceSettings limits, active fragments, vertices and FragmentPool.MaximumSize.

Remove crossing, touching, duplicate-adjacent or collinear points; make a non-zero-area closed outline.

Generate fragments first, select an Assets/ destination, and use an Editor with a graphics device. Read the per-fragment status message.

Confirm valid generated fragments and Assets/ output path. Sorting settings may need adjustment after export.

Check Burst, Collections and Mathematics versions and optional Jobs assembly registration; Managed fallback remains available.

Choose Preview = Colliders and Fragment Collider Type = Polygon. Box intentionally approximates bounds.

Reimport the complete Editor/Slicing folder, including UXML, USS and .meta files.