

User Guide
v1.0.0-pre.1

© 2018–2026 Apolloid Games. All Rights Reserved.

Table of Contents

Welcome to Snapshot Evaluator 3

- Need Support? 3
- What's New in 1.0 3

Installation 4

A Journey to Sphearth (Demo Scene) 4

UI Overview 6

- Snapshot Camera 6
- Photo Subject 7
- Subject Definition 7
- Scoring Profile 8
- Photo Goal 9
- Saved Snapshots 9

Basic Setup 11

- Initial Setup 11
- Creating a Photographable Object 11
- Key Points 11
- Key States 12
- Viewing Snapshot Information 12

How Scoring Works 13

- How It Works 13

Quick Start to Interfacing with Snapshot Evaluator 14

- Basic Setup 14
- Photo Goals 15
- Saving Photos (SnapshotStore) 15
- Custom Scoring Rules 15
- Re-scoring 16

Supported Render Pipelines 17

- Limitations 17

Complete API Documentation 18

© 2018–2026 Apolloid Games. All Rights Reserved.

Welcome to Snapshot Evaluator

Snapshot Evaluator is a photo scoring system built for Unity. It analyzes in-game photos and determines how many points they should be worth depending on several customizable parameters set by the game designer.

Each time a photo is taken:

- All “photographable” objects (Photo Subjects) within the photo are recorded and grouped by type, leaving out whatever is hidden behind scenery or other objects, pixel by pixel.
- Objects in the center of the photograph are determined.
- Objects facing the camera are determined.
- The proportion of the frame occupied by each object is determined.
- Key points on each object (a face, a tail) are checked to see if they are visible in the photo.
- The action (animation state) of each object at the time of the photo is recorded.
- A “best” subject is determined, and the photo is scored accordingly, with a full breakdown of where the points came from.
- Photo goals (“a Sphlion doing a handstand”) can be checked against the photo.
- The photo can be saved as a .png or .jpg file, with its scores, at any time.

Snapshot Evaluator is designed to be minimally intrusive, and easy to plug into any existing project. Your objects keep their own materials and shaders: nothing in your scene needs to be modified for it to be photographed. The asset is a product of love, now rebuilt from the ground up for Unity 6.

Need Support?

Please don’t hesitate to contact me! If you need help for any reason, or would like to request a future feature, please email me: support@apolloidgames.com. Please also check out the YouTube channel for tutorial videos: youtube.com/@snapshotevaluator7262. The latest version of this guide and other news are at apolloidgames.com.

What’s New in 1.0

Version 1.0 is a rewrite for Unity 6. If you used the 0.9 beta, these are the main differences:

- **No shader changes.** The `_ColorID` shader property and the *SnapshotEvaluator/Standard* shader are gone. Any material, from any shader, works as it is.
- **No second camera.** The *ScoreCam* is gone. Add a **Snapshot Camera** component to the camera that takes the photos; the photo and its analysis come from the same render.
- **Components and assets instead of one window.** The Window > Snapshot Evaluator tabs are now a Scoring Profile asset, Subject Definition assets and the Photo Subject component.
- **New:** URP support, photo goals, custom scoring rules, score breakdowns, asynchronous capture, and file storage with a Saved Snapshots window.

© 2018–2026 Apolloid Games. All Rights Reserved.

Installation

Requirements: Unity 6 (6000.0 or later), with the Universal Render Pipeline (URP) or the Built-in render pipeline. The High Definition Render Pipeline (HDRP) is not supported yet.

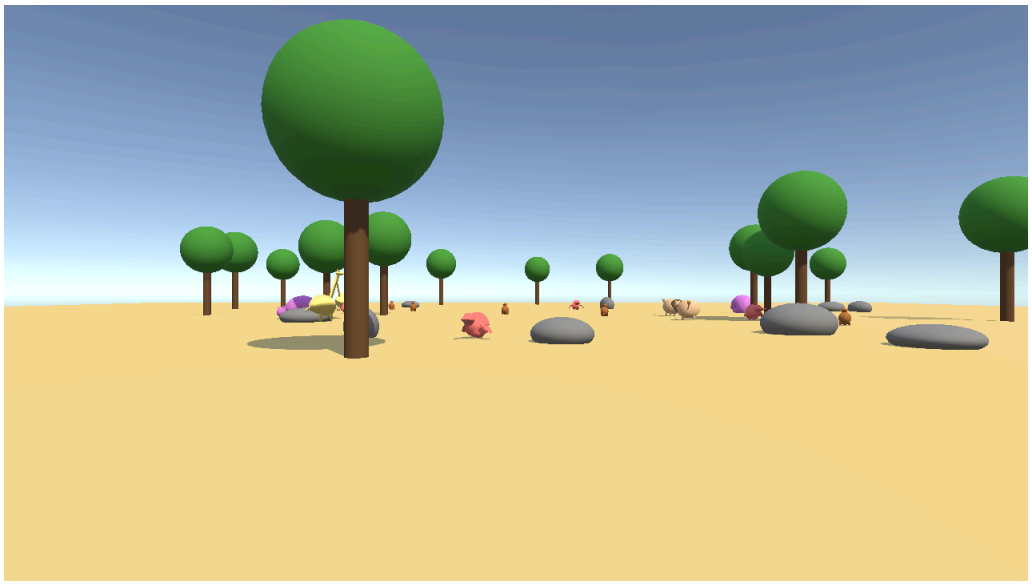
Snapshot Evaluator installs as a package, **com.apolloidgames.snapshotevaluator**. After importing it from the Asset Store, it appears in the **Packages** folder of the Project window, laid out as follows:

Packages/Snapshot Evaluator – The package root.

> **/Runtime –** Main source code: Snapshot Camera, Photo Subject, scoring and storage.
>
> **/Runtime/Shaders –** The hidden shader used to analyze photos. You never assign it yourself.
>
> **/Editor –** Inspectors and the Saved Snapshots window.
>
> **/Samples~/Sphearth Demo –** The demo scene, imported from the Package Manager (below).
>
> **/Documentation~ –** This guide.

Your own scripts reference the `ApolloidGames.SnapshotEvaluator` namespace. If your scripts live in an assembly definition, add a reference to the **ApolloidGames.SnapshotEvaluator** assembly.

A Journey to Sphearth (Demo Scene)



Sphearth is the demo scene where your adventure begins. The savanna has been set up with five kinds of spherical animals (Sphanimals) ready to have their picture taken: Sphlions, Spheerkats, Spheraffes, Spherinos and Spherds. To get it, open **Window > Package Manager**,

© 2018–2026 Apolloid Games. All Rights Reserved.

select **Snapshot Evaluator**, open the **Samples** tab and click **Import** next to **Sphearth Demo**. Then open *Assets/Samples/Snapshot Evaluator/<version>/Sphearth Demo/Sphearth Demo.unity* and press Play. The demo works with both URP and the Built-in render pipeline.

You have a roll of 24 photos. Each time you take a picture, the corner of the screen shows its score and what was found in it. Keep on the lookout for Sphanimals doing silly things! Their actions are tracked by the asset, and a list of photo goals shows which shots you still need. When the film runs out, or when you press Enter, the outing ends with a photo report: your best photo of each species and its score.

WASD / arrow keys – Walk.

Mouse – Look around (click the view first to capture the cursor).

Space / left click – Take a photo.

Enter – Finish the outing and see the photo report.

Esc – Free the cursor.

Photos are also saved to disk, so they appear in **Window > Snapshot Evaluator > Saved Snapshots** (the demo keeps the latest 30). The demo's scripts (*DemoSession*, *DemoHud*) are a good example of using the API in a game.

© 2018–2026 Apolloid Games. All Rights Reserved.

UI Overview

Snapshot Evaluator is set up through a few components and assets, each with its own inspector. Create the assets from **Assets > Create > Snapshot Evaluator**. The following sections describe each one.

Snapshot Camera

Screenshot to come: The Snapshot Camera inspector in Play mode, after **Take Test Photo**
`images/inspector-snapshot-camera.png`

Add it to the camera that takes the photos (**Add Component > Snapshot Evaluator > Snapshot Camera**).

1. **Photo**

   Photo Width / Photo Height – Size of the photo in pixels. It does not have to match the screen, so you can take photos at a higher resolution than the game runs at.

2. **Scoring**

   Profile – The Scoring Profile asset used to score photos. Built-in defaults are used when it is empty.

3. **Analysis**

   ID Buffer Width – Width in pixels of the small image in which subjects are counted (see *How It Works*). Higher values find smaller and thinner subjects, at a small cost in performance. Default 128.

   Relative / Absolute Depth Bias – How much nearer scene geometry must be before it hides a subject pixel: distance × the relative bias, plus the absolute bias in world units. Raise them if subjects standing on or touching the ground lose pixels; lower them if subjects show through thin walls.

4. **Take Test Photo** – In Play mode, takes a photo and shows it with a full score breakdown for every subject type in it. This is the quickest way to tune a Scoring Profile.

When the camera is selected, the Scene view shows the photo frame and its center box (yellow).

© 2018–2026 Apolloid Games. All Rights Reserved.

Photo Subject

Screenshot to come: The Photo Subject inspector, and its key point handles in the Scene view
`images/inspector-photo-subject.png`

Add it to the root of anything that can be photographed (**Add Component > Snapshot Evaluator > Photo Subject**). Every renderer under it counts as part of the subject.

Definition – The Subject Definition asset saying what kind of thing this is. Subjects sharing a definition are scored together as one type.

Subject Type – Used to group subjects when no definition is assigned. Defaults to the GameObject's name. Without a definition a subject has no rarity and no key states.

Local Forward – The direction, in the object's local space, that counts as its front. Shown as a blue "Front" arrow in the Scene view.

Key Points – Spots that earn points when visible (see *Key Points*). Each has a **Name**, a **Point** transform, a **Score** and a **Radius**.

Add Key Point – Creates an empty child object and adds it as a key point. Move it with the handle in the Scene view.

Subject Definition

Screenshot to come: A Subject Definition with two key states
`images/inspector-subject-definition.png`

Display Name – User-friendly name, typically used to tell the player what they took a picture of. Defaults to the asset name.

Rarity – How rare the subject is. The lower the value, the rarer the subject (1 is common, 0.01 is extremely rare). Think of it like a percentage chance of seeing this subject. Rarer subjects score higher.

Category – Free-form grouping for your own use, e.g. "Mammal".

Key States – Actions that earn points when photographed (see *Key States*). Each has a **State Name**, a **Display Name**, a **Score** and an Animator **Layer**.

© 2018–2026 Apolloid Games. All Rights Reserved.

Scoring Profile

Screenshot to come: The Scoring Profile inspector, with its default values
`images/inspector-scoring-profile.png`

All the numbers that turn what is in a photo into a score. *How Scoring Works* shows how they combine.

1. **Framing**
    a. **Centering**

        Center Box Size – The asset calculates centering by checking for pixels of a subject located within a centering box. This sets the size of the box relative to the size of the frame. Default 0.275.

        Center Fill Threshold – Fraction of the center box the subject must fill. If the subject fills at least this much of the box, it is centered. Default 0.43.

        Subject In Box Threshold – Fraction of the subject's visible pixels that must lie inside the center box. If at least this much of the subject is in the box, it is centered. Default 0.43.

    b. **Facing**

        Max Facing Angle – The maximum angle, in degrees, between the subject's front and the direction to the camera for it to be called "facing" the camera. Default 30.

2. **Points**

    Centered Score – Score added if the subject is centered. Default 1000.

    Facing Score – Score added if the subject is facing the camera. Default 500.

    Size Multiplier – Multiplied by the proportion of the frame taken up by the subject, then added to the score: a subject filling the whole frame earns this much. Default 500.

    Rarity Multiplier – Divided by the subject's rarity, then added to the score. Default 50, so a common subject earns 50 and a subject of rarity 0.1 earns 500.

3. **Multipliers**

    Centered And Facing Multiplier – The score is multiplied by this factor if one subject is both centered and facing the camera (the old *Focus Multiplier*). Default 2.

    Extra Instance Multiplier – Each extra subject of the same type in the photo raises that type's score by this fraction. Default 0.25: three Spheerkats score ×1.5.

© 2018–2026 Apolloid Games. All Rights Reserved.

4. **Photo Total**

   Secondary Subject Weight – The best subject type in the photo counts in full; every other type counts at this weight. Default 0.15.

   Min Screen Share – Subjects covering less of the frame than this cannot earn centered or facing points, so specks in the distance do not count as the photo's focus. Default 0.005.

5. **Custom Rules** – Optional Scoring Rule assets that add points of your own (see *Custom Scoring Rules*).

Photo Goal

Screenshot to come: A Photo Goal from the demo: a Sphlion doing a handstand
`images/inspector-photo-goal.png`

A pass/fail requirement for a photo, for quests and checklists where a numeric score is not what the game needs. Check one with `goal.IsMetBy(snapshot)`.

Description – What the player is asked to photograph, for showing in your own UI.

Subject – The Subject Definition that must be in the photo. **Subject Type Name** is used instead when no definition is assigned.

Min Instances – How many of the subject must be in the photo.

Min Screen Share – Combined share of the frame the subjects must cover.

Require Centered / Require Facing – The subject must be centered / facing the camera.

Required State – Key state name the subject must be in. Leave empty for any.

Required Key Points – Names of key points that must all be visible.

Saved Snapshots

Screenshot to come: Window > Snapshot Evaluator > Saved Snapshots
`images/window-saved-snapshots.png`

1. **Snapshot list** – Photos saved with `SnapshotStore` in its default folder, newest first, with their scores.

© 2018–2026 Apolloid Games. All Rights Reserved.

2. **Details** – The selected photo and everything that was recorded about it: each subject type, its score, screen share, whether it was centered and facing, its key points and its state.
3. **Toolbar** – **Refresh** after taking more photos, **Open Folder** to see the files, and **Delete** to remove the selected photo.

© 2018–2026 Apolloid Games. All Rights Reserved.

Basic Setup

Initial Setup

1. Import Snapshot Evaluator into your existing project. The imported file structure is covered in the Installation section above.
2. Decide which camera will be used to take photos. In the demo scene, I used the camera attached to the first-person player. Select it and click **Add Component > Snapshot Evaluator > Snapshot Camera**. No second camera is needed: photos are taken through this camera.
3. Create a Scoring Profile (**Assets > Create > Snapshot Evaluator > Scoring Profile**) and drag it into the camera's **Profile** field. You can skip this step at first; built-in defaults are used.

Creating a Photographable Object

4. Create a Subject Definition (**Assets > Create > Snapshot Evaluator > Subject Definition**) for each kind of thing that can be photographed: one for all your Sphlions, one for all your Spheerkats. Fill out its fields (see *UI Overview*).
5. Select the GameObject or prefab, and click **Add Component > Snapshot Evaluator > Photo Subject**. Drag the Subject Definition into its **Definition** field.
6. Check the blue **Front** arrow in the Scene view. If your model's front is not its local +Z axis, change **Local Forward** until the arrow points out of its face.
7. Your object is now ready to have its picture taken! There is no need to change its materials or shaders. Enter Play mode, select the Snapshot Camera and click **Take Test Photo**, or take photos from your own script (see *Quick Start to Interfacing with Snapshot Evaluator*).

> Subjects created at runtime with `Instantiate` need no extra setup: a Photo Subject registers itself while it is enabled. Disable the component, or the GameObject, to make something unphotographable.

Key Points

Key points are spots on a subject that earn extra points when they are visible: a face, a tail, a hat. A key point is visible when the subject's own surface is drawn at that spot in the photo, so a face turned away from the camera, or hidden behind a tree, does not count.

1. Select the subject and click **Add Key Point**. An empty child object called *KeyPoint-1* is created.
2. Parent it to the bone or part it marks (for example the head bone), so that it follows animations.
3. Move it onto the surface of the model with the handle in the Scene view. The yellow circle shows its radius.

© 2018–2026 Apolloid Games. All Rights Reserved.

4. Give it a **Name** (Photo Goals refer to key points by name) and a **Score**.
5. Set its **Radius**: how far inside the subject's surface the point may sit and still count as visible, in world units. It must be smaller than the part's depth: a face radius larger than the head would also count the back of the head as the face. For thin parts such as tails, place the point at the tip and use a generous radius.

Only the key points of the largest instance of each type in the photo are scored.

Key States

Key states are actions that earn points when a subject is photographed doing them: eating, sleeping, a handstand. They are listed on the Subject Definition, and read from the subject's Animator.

1. On the Subject Definition, add an entry to **Key States**.
2. Set **State Name** to the name of the state in the Animator Controller, exactly as it appears there (for example *handstand*), and **Layer** to its Animator layer (0 for the base layer).
3. Give it a **Display Name** for the player ("Doing a handstand") and a **Score**.

The Animator is found on the Photo Subject's object or its children. If your actions are not Animator states, set them from code instead: `subject.ManualState = "Eating";` takes priority over the Animator while it is not empty. Set it back to `null` when the action ends.

Viewing Snapshot Information

8. In Play mode, select the Snapshot Camera and click **Take Test Photo**. The photo appears in the inspector with a breakdown of its score: size, rarity, centered, facing, key points, state, custom rules and multipliers for every subject type in it.
9. Photos saved with `SnapshotStore` (as the demo does) appear in **Window > Snapshot Evaluator > Saved Snapshots**. Click any of them to reveal everything that the asset calculated and stored about it.

So far, you've created a photographable object and learned how to view what information was stored when you took a picture. For a typical workflow, you would take a test photo in game, look at its breakdown, and tweak the Scoring Profile as you wish. Some basic interfacing with the asset's API is then needed to let the player take photos in a build.

© 2018–2026 Apolloid Games. All Rights Reserved.

How Scoring Works

Every photo is scored per subject *type*: all the Sphlions in a photo share one score, all the Spheerkats another. For each type:

```
score = focus multiplier × instance multiplier ×
        (size + rarity + centered + facing + key points + key state + custom rules)
```

Part	Points
Size	Combined share of the frame covered by every instance of the type × **Size Multiplier**. Only visible pixels count: the parts hidden behind scenery do not.
Rarity	**Rarity Multiplier** ÷ the subject's rarity.
Centered	**Centered Score**, if any instance is centered.
Facing	**Facing Score**, if any instance faces the camera.
Key points	The scores of the visible key points of the largest instance.
Key state	The score of the key state the largest instance is in.
Custom rules	The sum of the profile's custom rules for this type.
Focus multiplier	**Centered And Facing Multiplier**, if one instance is both centered and facing; otherwise 1.
Instance multiplier	1 + **Extra Instance Multiplier** × (instances − 1).

Instances covering less than **Min Screen Share** of the frame cannot make their type centered or facing.

The type with the highest score is the photo's **dominant** subject. The photo's total score is the dominant type's score plus every other type's score × **Secondary Subject Weight**. A photo with no subjects scores 0.

How It Works

When a photo is taken, the same render that produces the photo also draws every enabled Photo Subject into a small, hidden image, the **ID buffer**, in which each pixel records which subject is there. It is compared with the scene's depth, so subjects hidden behind scenery or behind each other are left out pixel by pixel. Counting pixels in this image gives each subject's share of the frame, whether it is centered, and whether its key points can be seen. Its width is the Snapshot Camera's **ID Buffer Width**; its height follows the photo's aspect ratio.

The ID buffer is read back from the GPU asynchronously, so `Capture` does not stall the game. The result arrives a few frames later. Everything that can change in those frames (positions, facing, animation states, key point positions) is recorded at the moment of the photo.

© 2018–2026 Apolloid Games. All Rights Reserved.

Quick Start to Interfacing with Snapshot Evaluator

It is highly recommended to view the *DemoSession* script included in the demo scene to see an example of interfacing with the asset.

Basic Setup

Snapshot Evaluator records a lot of data for every picture taken. Diving directly into the code may be overwhelming, so it is recommended to follow this basic setup to understand the basics of the asset before trying to do everything at once.

1. Create a new script which will be used to take photos, and place it on any GameObject in your scene. In the demo scene, this is the *DemoSession* script on the Player.
2. In the script, declare a `SnapshotCamera` field and assign the camera in the inspector. The SnapshotCamera is the engine behind the asset that takes and analyzes photos.

```
using ApolloidGames.SnapshotEvaluator;

public SnapshotCamera snapshotCamera;
```

3. Call `Capture` to take a photo. The result arrives a few frames later as a `Snapshot`:

```
void TakePhoto()
{
    snapshotCamera.Capture(snapshot =>
    {
        Debug.Log("Score: " + snapshot.TotalScore);
        if (snapshot.Dominant != null)
            Debug.Log("A photo of a " + snapshot.Dominant.DisplayName);
    });
}
```

   `Capture` returns false (and logs why) if the photo could not be taken. In editor tools and tests, `CaptureImmediate()` returns the Snapshot straight away, but stalls until the GPU catches up.
4. To handle every photo in one place, whichever script took it, subscribe to the `SnapshotCaptured` event instead:

```
void OnEnable()  { snapshotCamera.SnapshotCaptured += OnPhoto; }
void OnDisable() { snapshotCamera.SnapshotCaptured -= OnPhoto; }
```

5. Once you have a Snapshot, you will have access to everything recorded about it:

```
foreach (SubjectScore type in snapshot.Scores)          // best first
{
    Debug.Log(type.DisplayName + " x" + type.InstanceCount + ": " + type.Total);
    if (type.State != null)
        Debug.Log("  " + type.State.DisplayName);
    foreach (KeyPoint point in type.KeyPoints)
        Debug.Log("  showing its " + point.name);
}
```

© 2018–2026 Apolloid Games. All Rights Reserved.

`snapshot.Find(definition)` returns the score of one subject type, or null if it is not in the photo. `snapshot.Subjects` lists every visible instance, with its own screen share, centering, facing angle, key points and state.

6. A Snapshot owns its photo, a RenderTexture. Show it in a `RawImage` while you need it, and call `snapshot.Dispose()` when you are done with it to free the memory.

Photo Goals

```
public List<PhotoGoal> goals;

void OnPhoto(Snapshot snapshot)
{
    foreach (PhotoGoal goal in goals)
        if (goal.IsMetBy(snapshot))
            Debug.Log("Goal complete: " + goal.description);
}
```

Saving Photos (SnapshotStore)

`SnapshotStore` saves photos as image files in a folder, with a small JSON index of their scores. By default the folder is *Snapshots* under `Application.persistentDataPath`, which is also the folder the Saved Snapshots window shows.

```
SnapshotStore store = new SnapshotStore();     // or new SnapshotStore(folder)
SnapshotRecord record = store.Save(snapshot, SnapshotImageFormat.Jpg);

foreach (SnapshotRecord saved in store.Records)    // oldest first
{
    Texture2D photo = store.LoadPhoto(saved);       // you own it: Destroy it when done
    Debug.Log(saved.capturedUtc + ": " + saved.totalScore);
}
store.Delete(store.Records[0]);
```

A record keeps the photo's file name, size, capture time, total score, and for each subject type its name, instance count, screen share, centered and facing flags, state and visible key points. To save the image only, use `snapshot.EncodePhoto(SnapshotImageFormat.Png)` and write the bytes yourself.

Custom Scoring Rules

To add points the built-in parts do not cover, subclass `ScoringRule`, create an asset from it, and add it to the profile's **Custom Rules**. Its points are added to each subject type's score before the multipliers, and appear in the score breakdown under its `Label`.

© 2018–2026 Apolloid Games. All Rights Reserved.

```
using ApolloidGames.SnapshotEvaluator;
using UnityEngine;

[CreateAssetMenu(menuName = "Snapshot Evaluator/Rules/Close Up Bonus")]
public class CloseUpBonus : ScoringRule
{
    public float minScreenShare = 0.3f;
    public float bonus = 300f;

    public override string Label => "Close up";

    public override float Evaluate(SubjectScore subject, Snapshot snapshot)
    {
        return subject.ScreenShare >= minScreenShare ? bonus : 0f;
    }
}
```

A rule may return negative points, for example to penalize a blurry or crowded photo.

Re-scoring

A Snapshot keeps what was found in the photo separately from its score, so it can be scored again with a different profile, for example to compare tunings or to judge the same photo for a different contest:

```
SnapshotScorer.Score(snapshot, otherProfile);
```

© 2018–2026 Apolloid Games. All Rights Reserved.

Supported Render Pipelines

Pipeline	Support
Universal (URP)	Supported. The analysis is added to the photo's own render; no Renderer Feature needs to be added to your URP renderer.
Built-in	Supported. Tested with the Forward rendering path; the Deferred path is handled but not tested yet.
High Definition (HDRP)	Not supported yet. The Snapshot Camera logs an error and its inspector shows a warning.

The demo scene works under URP and Built-in. Its materials come in both versions, and its *DemoMaterials* component switches the scene to the set for the active pipeline. If you drag a demo prefab into another scene in a Built-in project, use the materials in *Materials/Built-in*.

Limitations

- **Small and thin parts.** Subjects are counted in the ID buffer, which is much smaller than the photo (128 pixels wide by default). Distant subjects and thin parts such as tails may cover only a pixel or two. Raise **ID Buffer Width**, or for key points on thin parts use a generous radius.
- **Key point radii need tuning** for each model. A good start is half the depth of the part it marks.
- **Transparent materials** do not hide subjects behind them, as they do not write to the scene's depth. Subjects seen through glass or water count as visible.
- **Facing** is measured from the Photo Subject's transform, not its head: a subject whose head turns independently of its body counts as facing where its body faces. Use a Face key point to reward the face itself.
- **Tested platforms:** Windows with Direct3D 12, Direct3D 11 and Vulkan; WebGL 2 in Chrome; and Android (Pixel 9 Pro) with both Vulkan and OpenGL ES 3. Where `AsyncGPUReadback` is unavailable, the analysis is read back at once instead, so `Capture` stalls briefly and calls back before it returns. Where the GPU cannot render to two-channel float textures, the analysis is packed into an ordinary 8-bit texture; distances are then accurate to about 1/65,000 of the camera's far plane. Both fallbacks are chosen automatically. Other platforms, including macOS and iOS (Metal), are expected to work, but have not been tested yet.
- Up to 16,777,216 subjects can be told apart in one photo, or 65,535 where the 8-bit fallback is in use.

© 2018–2026 Apolloid Games. All Rights Reserved.

Complete API Documentation

Every public type and member is documented in the source code with XML comments, which your IDE shows as you type. The main types are:

Type	What it is
`SnapshotCamera`	Component. Takes and analyzes photos: `Capture`, `CaptureImmediate`, `SnapshotCaptured`, `ActiveProfile`.
`PhotoSubject`	Component. Marks something photographable: `Definition`, `KeyPoints`, `ManualState`, `PhotoSubject.Active`.
`SubjectDefinition`	Asset. A kind of subject: display name, rarity, category, `KeyState` list.
`KeyPoint`, `KeyState`	A spot on a subject, and an action, that earn points.
`ScoringProfile`	Asset. Every scoring number, and `FramingSettings` for centering and facing.
`ScoringRule`	Base class for custom scoring rules.
`PhotoGoal`	Asset. A pass/fail requirement: `IsMetBy(snapshot)`.
`Snapshot`	A photo and what was found in it: `Photo`, `Subjects`, `Scores`, `Dominant`, `TotalScore`, `Find`, `EncodePhoto`, `Dispose`.
`SubjectResult`	One subject instance in a photo: screen share, centered, facing angle, visible key points, state.
`SubjectScore`	One subject type's score in a photo, with every part of it.
`SnapshotScorer`	Scores a Snapshot with a profile.
`SnapshotStore`, `SnapshotRecord`	Saving photos to disk with their scores.
`IdBufferAnalyzer`	The pixel-counting functions, for advanced use.

All types are in the `ApolloidGames.SnapshotEvaluator` namespace.

© 2018–2026 Apolloid Games. All Rights Reserved.