Advanced Light Detection System

Table of Contents

Introduction

This is the advanced light detection system for unity. This system is also included in the Ultimate Stealth System package. If you own this package, you should get a discount on the Ultimate Stealth System. This system supports 4 different, well-optimized light detection methods. Each detection method has its own pros and cons which I'll go over in another section.

Setup

1. Download the Asset from the Unity asset store.
2. Import the package.
3. For an example of the light detection in action, go into the demo scene or head to the load testing scene to see where I got the figures for the performance breakdown.
4. Simply drag the LightDetectionActor prefab into your scene or add the LightLevelDetection script to one of your actors.
5. Adding the script to your actor will work out the box but you probably want to do some configuration. The biggest thing you'll want to look into is the light detection method you want to use. I'll do a full breakdown of every method later in the readme.
6. If using Raycast or Mixed method, you must add the corresponding stealth light script to your lights you want to add detection. Supported types include directional, spot, and point. If you need another type (area or custom), you can either try to implement them yourself or use floor rendering method as that should work with any lighting system.
7. There are prefab lights in the LIghtDetection/Lights folder which show how to set them up.
8. There is also a sample of a moving spot light to look at for reference.

Lighting Detection Methods

As mentioned before, this system implements 4 methods. Each has their own pros and cons which I will go over now.

Raycasts

Implementation:

For each effecting light, we do a raycast from the light to each light detection spot on the actor. If the raycast hits something, that spot is considered in the shade, otherwise, it does some math to find how much light that light should be affecting the actor.

Pros:

- Super well optimized for huge groups. With low settings, each actor can be set to be doing only one raycast per light per call. Meaning you could easily have hundreds or even thousands of actors running this method.
- Very accurate. Because there are no mip maps blending pixels or possibility of unorthodox lighting, camera, or shadowing settings interfering, this is a very robust method since it is just checking if there is anything blocking the light.

Cons:

- Does not work with colliders that don't match the shadowing models. So, if you have a tree that has a simple capsule collider despite having leaves all over the place, this will go right through the leaves and will be inaccurate. This is a big bottleneck for player characters especially in stealth games as we need to be able to optimize our game with simple colliders and still be accurate. You could get around this by changing the raycasts to only hit a secondary collider only used for this and that would work.
- Does not account for transparency at all. Assuming they have collision, windows, fences with holes in them, and any other transparent things will completely block the raycast. Again, you could get around this a bit by using a multicast (a raycast that can hit multiple things). You'd then have to sample the material of the hit object and check if it is transparent. This is possible but not very clean.

Use Cases:

- Large numbers of actors that need to detect light. Think of a horde of vampires or zombies.
- Games where geometry is simple or low poly enough that colliders can be accurate without tanking performance and windows or other transparent or semi-transparent objects either don't exist or because they are high enough up, far enough away, or small enough to not need collision.

Floor Render Texture

Implementation:

Using a secondary, orthographic camera that renders shadows, we look directly at the ground. We use a mip map to compress the ground pixels into one, average pixel. We then sample the color of that pixel and compare the brightness of the pixel to manually set shadowed and lit ranges.

Pros:

- Does not need Stealth Lights to be set up since they simply look at the ground. No calculations with the lights, no transparency reading.
- Accounts for transparent blockers like windows since light will be hitting the floor and affecting the color of the pixels when it goes through a transparent object.
- As long as detections per second are kept relatively low (under 20), floor rendering is well optimized. More explained in the Performance section.
- Unlike the raycast method, the collision accuracy of shadow casters is completely irrelevant.

Cons:

- Needs additional setup. Because it gets the color of the floor to determine brightness, if say a light detector goes from standing on a white floor to standing on a darker floor, the shadowed and lit ranges need to be adjusted. This can be consistently implemented in many ways including changing on scene changes, trigger colliders, etc.
- Not well optimized for huge groups that detect light many times per second.
- Uses a secondary camera which can prove to be a problem in some games.

Use Cases:

- Player Characters that need precise light detection no matter what the colliders look like.
- You want full customizability for your detection system in each scene.
- Simple floor textures allowing for a low resolution and render size making it extremely performant especially for a single player character.
- Top-down games where what should matter is where they are standing more than anything else.

Sky Render Texture

Implementation:

Very similar to the Floor Render Texture. Now, our secondary, orthographic camera with shadows off (unless using built in pipeline as that is not clean to do for a single camera) looks at the sky in the direction of the sun. We still use a mip map to compress the render into a single pixel but now we get the transparency of the average pixel. We then use some math to find the intensity the sun should be casting on the player given the sun's intensity, position in the sky (noon is bright, sunset is not as bright), and the transparency of the pixel.

Pros:

- Specializes in transparency.
- Unlike the floor render texture method, there is very little setup. The sun simply needs to be added to the lights to check array as the first and only element with a basic stealth light script.
- Because shadows are not rendered, the GPU can render the secondary camera quite a bit quicker than the floor render camera. This means that higher detections per second affect the performance less.
- Unlike the raycast method, the collision accuracy of shadow casters is completely irrelevant.
- Captures ambient effects like fog.

Cons:

- Does not look at any light but the sun. Point, spot, and any additional directional lights besides the primary one will not be accounted for in the light level detection.
- Often a bit slower than the floor rendering system at low detections per second since we're sampling transparency instead of simple color and the calculations we do every call. More about that in performance section.
- Has potential to be inconsistent or noisy if you have a very dynamic or complex sky system.

Use Cases:

- Player Characters that need precise light detection no matter what the colliders look like.
- The only light in your scene that matters for detection is the directional light (the sun).
- You want fog to be automatically factored into the detection.

Mixed

Implementation:

Mixed method combines the Raycasts and Sky Render methods. For the first light in the lights to check array, it will do the sky render method and it will do raycasts for all other lights in the array.

Pros:

- All the pros of the Sky Render Texture method but without the con of not accounting for other lights besides the sun.
- Actually usually more performant than sky render texture because the raycasts are done first and the sky render is not done if the raycasts get the actor to fully lit up.

Cons:

- Same cons as the sky render texture method (besides the one about only seeing the sun) for the sun.
- Same cons as the raycasts method for the lights besides the sun.

Use Case:

- Your sun is the primary light that matters for detection but there are also other lights like spotlights or room lights (point lights) that don’t usually have complex geometry or transparent objects obstructing them.

Lighting System Customizable Settings

I'll go over the lights first since they only have a few settings:

Stealth Lights

1. **Lit – Light:**

Link to the light component on this actor. Just a quick optimization, it will work if this is null.

2. **Overrides Base Intensity for Detection – Bool:**

If true, this light will override the corresponding light detection setting in the light level detection script for this light's type (direction, spot, or point).

3. **Intensity for Detection – Float:**

If the above bool is true, this is the intensity level that will override.

4. **Light Collider – Collider:**

The collider that represents when an actor might be getting hit by this light. When an actor with the light level detection script enters this collider, it will automatically add this light to the light level detection's lights to check array. This is only used for spot and point lights since directional lights should always be in the array.

The Light Level Detection script is the bulk of the system. I'll split up the settings into the methods the settings effect.

General

1. **Light Detection Method:**

Which method does this actor use to detect light?

2. **Current Light Level – Float:**

The current light detected by the script. This is only shown in the inspector to help with testing and debugging.

3. **Light Detections a Second – Float:**

How many times we call to detect light a second. 5-15 is perfect for most cases. If set to a decimal like 0.1, it will be called every 10 seconds.

4. **-Debug- Debug Light Img – Image:**

The image that visualizes the current light level. Optional

Raycasts

1. **Raycast Points - List of Transforms:**

Where should raycasts come out of the actor to lights to detect light. The more points, the more accurate the light detection will be. This can be set to be bones of a armature and that way when the head moves, the spot the raycast is coming out of changes.

2. **Lights to Check – List of Stealth Lights:**

A list of stealth lights to raycast to. Automatically updated by stealth lights with trigger colliders

3. **Directional Light Cast Distance – Float:**

How far we should cast a ray to check if anything blocks the sun. If you have a big world, this might need to be pretty large. It's a pretty small optimization to make this as low as it can be so don't worry too much about making it perfect

4. **Light Falloff Factor – Float:**

How quickly spot and point lights fall off the further the player is from them.

5. **Directional, Point, and Spot Intensity for Detection – Float:**

These 3 variables set at what intensity a light detector considers itself fully detected (currentlightlevel == 1) for each light type. These are all general and can be overrode by each individual stealth light.

6. **-Debug- Draw Debug Lines – Bool:**

If true, the raycasts will be drawn for you to see. Great for debugging weird issues

Floor Render Texture

1. **Light Detection Cam – Camera:**

The Secondary camera used for rendering making the render texture

2. **Render Size – Float:**

How big the secondary camera's viewport should be. Larger objects need a bigger render size. Too small of a render size is less accurate.

3. **Floor Fully Shadowed Max – Float:**

The max brightness of the floor at which the actor is still considered fully in the shadow. Finding the right value for this variable is extremely important to the floor method working correctly.

4. **Floor Fully Lit Min – Float:**

The min brightness of the floor at which the actor is still considered fully lit up. Finding the right value for this variable is extremely important to the floor method working correctly.

5. **Texture Rendering Res:**

Accuracy of the floor rendering. Make this higher if you have a complex floor material. It has a slight performance impact.

6. **-Debug- Show Floor Color Level – Bool:**

If true, display the current floor color level in the inspector and print it in the console. This is helpful for finding out what your fully shadowed and fully lit values should be on your current floor material.

7. **-Debug- Current Floor Color Level – Float:**

The current floor color level. Used to figure out what your “floor fully shadowed max” and “floor fully lit min” variables should be.

Sky Render Texture

1. **Light Detection Cam – Camera:**

The Secondary camera used for rendering making the render texture.

2. **Render Size – Float:**

How big the secondary camera’s viewport should be. Larger objects need a bigger render size. Too small of a render size is less accurate.

3. **Lights To Check – Array of Stealth Lights:**

In the case of the sky render texture method, this should just be the one sun in it’s first element and nothing else.

4. **Directional Light Cast Distance - Float:**

How far should the camera look for things that can block the sun. If you have a really big scene, you’ll need to increase this.

5. **Directional Intensity for Detection - Float:**

What intensity is necessary to hit the actor from the directional light for it to be considered full lit (currentlightlevel == 1).

6. **Texture Rendering Res:**

The resolution of the camera's rendering texture. The higher it is, the smoother the current light level will change with movement. Slight performance cost at higher levels.

7. **Daylight Factor – Float:**

How much the intensity of the sun is affected by the time of day. If you already handle changing the intensity of the sun with the time of day, you should set this to 0.

Mixed

1. **Light Detection Cam – Camera:**

The Secondary camera used for rendering making the render texture.

2. **Render Size – Float:**

How big the secondary camera's viewport should be. Larger objects need a bigger render size. Too small of a render size is less accurate.

3. **Raycast Points - List of Transforms:**

Where should raycasts come out of the actor to lights to detect light. The more points, the more accurate the light detection will be. This can be set to be bones of a armature and that way when the head moves, the spot the raycast is coming out of changes.

4. **Lights to Check – List of Stealth Lights:**

A list of stealth lights to raycast to. Automatically updated by stealth lights with trigger colliders

5. **Directional Light Cast Distance – Float:**

How far we should cast a ray to check if anything blocks the sun. If you have a big world, this might need to be pretty large. It's a pretty small optimization to make this as low as it can be so don't worry too much about making it perfect

6. **Light Falloff Factor – Float:**

How quickly spot and point lights fall off the further the player is from them.

7. **Daylight Factor – Float:**

How much the intensity of the sun is affected by the time of day. If you already handle changing the intensity of the sun with the time of day, you should set this to 0.

8. **Texture Rendering Res:**

The resolution of the camera's rendering texture. The higher it is, the smoother the current light level will change with movement. Slight performance cost at higher levels.

9. **Directional, Point, and Spot Intensity for Detection – Float:**

These 3 variables set at what intensity a light detector considers itself fully detected (currentlightlevel == 1) for each light type. These are all general and can be overrode by each individual stealth light.

10. **-Debug- Draw Debug Lines – Bool:**

If true, the raycasts will be drawn for you to see. Great for debugging weird issues

Detection Method Performance Breakdown

General

In general, there are some general things to keep in mind to help performance:

- Keep Detections per second (d/s) as low as you can. For single player actors detecting lights, you can get away with being 20-30+ d/s but for large numbers of detectors, keeping it low can help a lot.
- As with all unity projects, the more lights you have close together, the more complex your shadow casting is going to be. Try to either keep it to as few lights in one place as possible or turn down your shadow quality.
- Realtime lights are generally bad for performance in unity. Use mixed or better yet baked lights where you can.
- Using hard shadows instead of soft shadows can also help a bit as well.

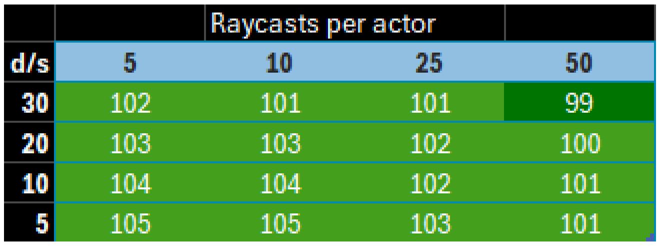
I'm going to briefly show off the performance of each method and give some insights for each one. Each method was tested with 33 animating humanoid characters all being affected by 5 lights (1 Realtime directional light and 4 mixed lights). The FPS without any light detection at all was around 105.

Key:

d/s – Detections per second

100+	High performance for large numbers of actors
60+	High performance for medium numbers of actors
30+	Usable for medium numbers of actors
20+	Usable for small groups of actors
10+	Only to be used for player character(s)

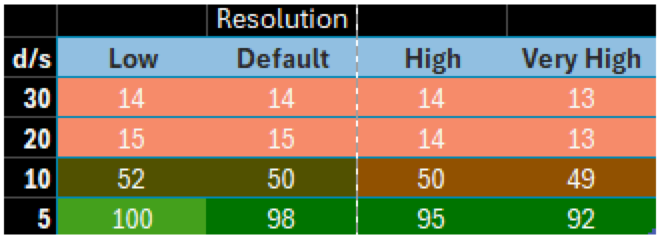
Raycasts



	Raycasts per actor			
d/s	5	10	25	50
30	102	101	101	99
20	103	103	102	100
10	104	104	102	101
5	105	105	103	101

The raycast method is really well optimized. Even with 1650 (50x33) raycasts 30 times a second, the fps was only reduced by about 5-7. Absolutely fantastic for having a horde of vampires.

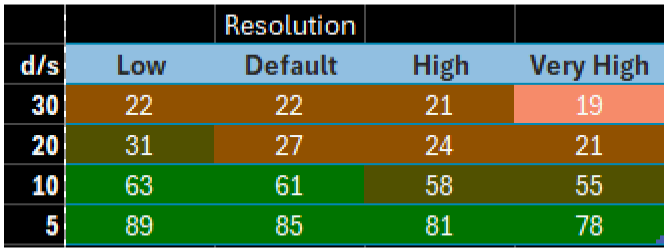
Floor Render Texture



	Resolution			
d/s	Low	Default	High	Very High
30	14	14	14	13
20	15	15	14	13
10	52	50	50	49
5	100	98	95	92

The floor render texture method actually has really good performance at a low d/s. Unfortunately, because the render texture is pulled from a secondary camera with an async call, if we have much higher than 200 or so total calls a second, the gpu does not return the first texture before the second is requested. This builds up and slows down the game a lot which is why we see such a stark drop in fps at higher d/s. This is not a problem if you aren't doing a large group of detectors. A single player using the floor render texture can comfortably run it at a high d/s (although again I see no reason to ever go over 20 d/s).

Sky Render Texture

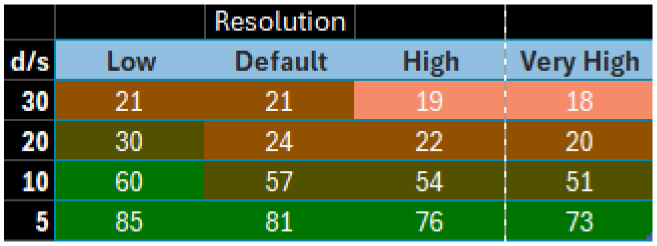


	Resolution			
d/s	Low	Default	High	Very High
30	22	22	21	19
20	31	27	24	21
10	63	61	58	55
5	89	85	81	78

The sky render texture method has slightly worse initial fps but drops down a lot less than the floor variant. I believe the worse initial fps is due to the transparency calculation which is slightly harder to do than the floor's simple color. The drop is much less intense at high d/s because the secondary camera in the sky render texture method does not need to render shadows. This speeds up the GPU's response to the async render request a lot so the requests don't build up as much.

Mixed

All tests here were done with 20 raycasts per actor (5 raycast points on each actor x 4 lights that aren't the directional light).



		Resolution		
d/s	Low	Default	High	Very High
30	21	21	19	18
20	30	24	22	20
10	60	57	54	51
5	85	81	76	73

The test here showed that the mixed method is just ever so slightly worse performance wise than the sky render texture method. This is because on top of doing the same thing the sky render texture method is doing, it also does the same thing the raycast method is doing.

However, in practice, I would actually expect it to have a slightly higher average performance than the sky render texture method because the implementation is set up as follows:

1. Raycast to all the lights (except the directional light) from all the raycast points.
2. Check if the current light from just step 1 is greater than or equal to 1. If it is greater than 1, skip to step 5, otherwise move on.
3. Do the sky render texture method to the sun.
4. Add the sun's light detected in step 3 to the raycasts light detected in step 1.
5. Set the current light level.

As you may have noticed, it is common for the sky render texture step to be completely skipped. That is, by far, the step that impacts performance the most. This means that any time the actor detects enough light from any lights besides the sun, the performance will be about as high as the raycast method's performance.

Frequently Asked Questions

Which detection method should I use?

This comes down to a lot of factors and I would urge you to read the rest of this readme for more info. Especially the method breakdown section. but as a quick TL; DR:

- Pick Raycasts if you have a huge number of detectors (a horde of vampires for example). Doesn't work with colliders that don't match the shadow casters!
- Pick Floor Render Texture if you don't mind doing some setup and don't care about ambient effects like fog and you are confident you made your lighting system consistent for detected areas being consistently lit up.
- Pick Sky Render Texture if you only care about the sun. No other lights will contribute to the light level detected by the actor. This also works with ambient effects like fog and colliders are irrelevant.
- Pick Mixed if you want the ambient effects and collider workaround that sky render texture has but need other lights to work as well.

Don't forget you can mix and match. If the player is sneaking around a horde of vampires maybe use floor or sky render texture for your player and raycasts for the vampires.

I'm overwhelmed by all the settings. Help?

I know there is a lot of customizability. Luckily the default settings are pretty good. If you don't know what something does, each setting has a tooltip. If that doesn't help, refer to the settings section of this readme. If you're still confused, send me an email.

I used an editor script to only display the settings that are used for method you selected. So if you are looking for a setting you saw in here that isn't in the inspector, chances are you selected the wrong method for that setting to exist. If you'd like to see all settings no matter what, you can do that by deleting the editor script entirely.

Why isn't the Raycast method working?

- Make sure you have at least 1 raycast point in the raycast points array.
- Your directional light should be in the lights to check array in the editor. All others should be added dynamically.
- Make sure the direction light cast distance is high enough so rays can hit far away things that block out the sun.
- Make sure the light falloff factor isn't too high.
- Make sure to configure the directional, point and spot intensities for detection correctly.

Why isn't the floor render texture method working?

- Make sure you have a second camera set to light detection cam. An example of what the camera should look like is in the Light Detection Actor prefab.
- Make sure the render size variable isn't too small or too big. 0.25 is pretty normal for a human sized detector.
- Make sure you set up the Floor fully shadowed max and floor fully lit min variables correctly.

How do I set up the floor render texture method?

The easiest way in my opinion is as follows:

1. Turn on the debug setting: Show floor color level.
2. Click play and walk to the brightest spot you want to still be considered fully shadowed (you want currentlightlevel == 0).
3. Look at the current floor color level variable. This is around what you should set Floor Fully Shadowed Max to for this scene/section.
4. Walk to the darkest area you want to still be considered fully lit up (you want currentlightlevel == 1).
5. Look at the current floor count level variable. This is around what you should set Floor fully Lit Min to for this scene/section.

Why isn't sky render texture method working?

- Make sure you have a second camera set to light detection cam. An example of what the camera should look like is in the Light Detection Actor prefab.
- Check to make sure you've added your sun (directional light) to the lights to check array. It should be the first and only element if you are using the sky render texture method.
- Make sure the render size variable isn't too small or too big. 0.25 is pretty normal for a human sized detector.
- Make sure the direction light cast distance is high enough to see all objects that could block the sun.
- Check the intensity of your directional light. Make sure it is high enough.

Why isn't mixed method working?

Refer to the steps in both the raycast and sky render texture not working sections. Most common one for mixed is that the directional light is not set to the first element of the lights to check array or that there are no transforms in the raycast points array.

Why aren’t my lights getting raycasted to?

- You probably forgot to add a stealth light component to your light. Make sure the type of stealth light script you add matches the type of light you’re using. StealthLight for directional, StealthPointLight for point lights, and SteathSpotLight for spotlights.
- If you want to save some performance from adding a component dynamically, you should give all StealthPointLights a sphere collider and SteathSpotLight a meshcollider.
- Make sure spotlights have the colliderConeMesh set to the cone mesh I provided!

How do I add a new detection method?

There are a few steps to achieve this:

1. Make sure whatever you are trying to add isn’t already covered by one of the implemented methods
2. If not, add the name of your method to the Light Detection Method enum.
3. In the update function, add a new “else if” for your method.
4. Do any setup in the start function.
5. Make your function. If you do any secondary camera rendering, I suggest using async request.
6. Good luck!

I have another question…

No Problem! Email me your question at marasciagames@gmail.com and I’ll get back to you as fast as I can.

I found a bug!

Thanks! You can email me at marasciagames@gmail.com and I’ll get that fixed ASAP. I usually respond within a day.

Links:

This tool’s Unity Asset Store page: https://assetstore.unity.com/packages/slug/330808

This tool’s parent asset (if you bought this asset from the unity asset store, you should get a discount): https://assetstore.unity.com/packages/slug/325788

My Unity Asset Store publisher page: https://assetstore.unity.com/publishers/113743