ChatGPT

Real-Time Sunlight-Responsive Lighting in Mixed Reality

Technical Methods for Sunlight-Responsive Lighting

Realistically blending virtual content with real sunlight requires a combination of sophisticated rendering techniques and sensor data. **Light probes and environment maps** are commonly used to capture surrounding illumination: by sampling the real environment (often via a cube map or spherical panorama), engines derive ambient lighting and reflections for virtual objects [1] [2]. These environment maps can be converted into **spherical harmonics** coefficients representing low-frequency ambient light from all directions, which provide a form of real-time *global illumination* for diffuse lighting [3] [4]. To handle direct sunlight, algorithms perform **directional light estimation** – extracting the brightest light source direction and intensity (usually corresponding to the sun or a dominant lamp) from the environment data [2]. For example, one method involves blurring the captured cubemap to find the predominant light direction and color; the brightest region in the low-resolution image indicates the sun's orientation and hue, which is then applied to a virtual directional light in the scene [2].

Real-time **shaders** in MR applications leverage this data by dynamically adjusting lighting parameters. Physically-Based Rendering (PBR) shaders use the estimated **ambient intensity and color** to modulate their diffuse lighting, and apply the derived directional light to cast highlights and shadows consistent with the real world [5] [6]. **Shadow mapping** techniques are employed so virtual objects cast shadows in alignment with the real sun's direction, enhancing realism. In video-see-through AR (e.g. on mobile devices), these shadows can be composited onto the camera feed via shadow-catching surfaces, darkening the video where virtual shadows fall. (In optical see-through displays like HoloLens, true shadow projection onto real surfaces is not possible due to additive display technology – a limitation discussed later.) Meanwhile, **real-time global illumination (GI)** systems can simulate indirect light bounce among virtual objects, so that sunlight can bounce off a virtual surface and illuminate other virtual elements. In practice, fully dynamic GI in AR is challenging on mobile hardware, but high-end solutions like Unreal Engine's Lumen provide multi-bounce diffuse reflections and color bleeding at real-time speeds on capable devices [7] [8]. Lumen's approach uses software or hardware ray tracing to continuously update indirect lighting as the sun moves or the environment changes, enabling effects like realistic color bleed from sunlit surfaces and darkening of interiors when the sun is occluded [7] [8].

Sensor-driven approaches also play a role. Modern devices include ambient light sensors that measure overall brightness of the environment. These can calibrate the exposure and intensity of virtual lights – for example, a phone's ambient lux reading can scale the intensity of the virtual sun so that on a very bright day the virtual highlights appear correspondingly intense. MR headsets like HoloLens and Magic Leap have dedicated sensors for ambient light [9] [10] and even color (to adjust display white balance), which feed into the rendering pipeline to globally brighten or dim virtual content to match real-world conditions. Beyond simple sensors, **computer vision and AI techniques** are emerging for light estimation. These analyze camera imagery to infer lighting; for instance, machine learning models can predict an approximate environment map or sun direction from a single camera frame. Academic work in AR often explores

photometric registration – using known objects or markers in the scene to estimate lighting. A classic approach is to place a reflective sphere in view (or use the user's face or eyes as proxies); by observing the highlight on a curved surface, the system can deduce the light source direction and intensity [11] [12]. More recent research employs deep learning to estimate illumination from arbitrary scenes, bridging computer vision with rendering. These techniques complement traditional image-based lighting by handling cases where direct analysis (e.g. finding the brightest pixel) might fail due to limited camera view or multiple light sources. In summary, MR systems combine **image-based lighting, light probes, directional light extraction algorithms, and sensor/AI inputs** to achieve lighting that dynamically responds to sunlight and other real-world illumination [13] [5].

Engines and Frameworks Supporting Dynamic Sunlight

Modern game engines have built-in support for these lighting techniques, often via AR-specific frameworks. **Unity** supports real-time light estimation through its AR Foundation framework, which abstracts ARKit (iOS) and ARCore (Android) capabilities. Unity's Universal Render Pipeline (URP) and High Definition Render Pipeline (HDRP) both handle physically-based lighting, but HDRP offers more advanced features (like screen-space global illumination and refined reflections) at the cost of performance. On mobile AR, URP is commonly used for its efficiency, whereas HDRP might be employed on powerful AR devices or PCs for superior visual fidelity. Unity's AR Foundation provides an `ARCameraManager` that can feed lighting estimates into the scene. In practice, developers link these estimates to a Unity **Directional Light**. For example, Unity's **HDR Light Estimation** mode populates values for ambient intensity, ambient color, main light direction, main light color and intensity, and even spherical harmonics coefficients when the platform supports them [14] [15]. Unity then allows these to drive its lighting system: one can script the scene's Directional Light to update direction and color every frame based on the AR camera's light estimation data [16] [3]. In Unity's standard shader (or URP/Lit shader), the ambient spherical harmonics contribute to diffuse shading of objects, while the main light affects direct shading and shadow casting. Unity also supports **environment probes** – essentially automated reflection probes – in AR scenes. With ARKit, enabling environment probes causes the engine to capture cube maps of the real surroundings which Unity can use for realistic reflections on shiny virtual surfaces [17] [18]. These reflections will update as the user moves, mirroring changes in real lighting (for instance, a virtual chrome ball will show the bright outline of the real sun at the correct relative position).

Unreal Engine provides similar capabilities through its AR plugins and the powerful rendering features of Unreal Engine 5. For AR on iOS and Android, Unreal offers an `ARLightEstimate` accessible via blueprint or C++ which (depending on the platform) yields ambient brightness and color temperature. However, support for a **key directional light** estimate in Unreal's AR framework has been limited – as of UE 5.2, the engine defines an enum for *directional light estimation* but notes it is "currently not supported" in general AR sessions [19] [20]. This means that out-of-the-box, UE might only use ambient light intensity/color from ARKit/ARCore. Developers have often implemented custom solutions to integrate ARCore's main light direction data or ARKit's face-light data. On the high end, Unreal's **Lumen** GI system can dramatically improve realism for mixed reality by allowing fully dynamic global illumination and reflections [21] [7]. If an MR experience is running on hardware that supports Lumen (high-end PC or possibly future AR devices with strong GPUs), the virtual scene can have real-time bounce lighting: e.g. sunlight entering a virtual window, bouncing off a virtual floor, and illuminating a virtual character – all updating as the sun's angle changes [7] [8]. Lumen also provides ray-traced reflections, so glossy virtual objects could reflect bright real lights without needing manual reflection captures [22] [23]. That said, on mobile-class hardware, Lumen may be too heavy, and Unreal would fall back to simpler techniques (like ambient light and static reflection

captures). Unreal's forward-rendered AR mode can still use a *Sky Light* actor set to capture the environment, which functions similarly to a light probe capturing ambient lighting. Additionally, Unreal developers working with ARKit have access to **AREnvironmentProbe** support (as of ARKit 2.0 integration) to get environment textures for reflections [17].

Beyond Unity and Unreal, other engines and SDKs also support sunlight-responsive lighting. **Godot Engine** (with ARCore via plugins) can retrieve light estimates from ARCore on Android, though it's less robust than Unity's solution. **WebXR** in browsers now has a Light Estimation API (in proposals/experiments) that provides spherical harmonics and a primary light estimate to web AR content [24]. For instance, 8th Wall's WebAR framework introduced real-time reflections using environment lighting data from the camera [25]. **Niantic Lightship (ARDK)**, a toolkit for AR experiences, offers environmental HDR lighting on supported devices similar to ARCore's, ensuring that developers can get ambient intensity and a main light direction to light their objects. And on the proprietary side, **Meta's Spark AR** and Snap's Lens Studio (for social AR effects) both include basic ambient light estimation (to slightly adjust filters based on scene brightness). These lightweight engines don't do full GI, but they at least ensure a face filter or AR sticker appears correctly lit (for example, adding a virtual shadow on one side of a face if that side is darker in the real environment).

To summarize engine support, Unity and Unreal lead with integrated pipelines for real-world lighting: Unity's AR Foundation exposes the underlying AR platform's light data in an easy way [5] [26], and Unreal, while needing a bit more manual work for directional lights, provides cutting-edge GI via Lumen for higher-end MR. Other engines and frameworks are rapidly catching up, leveraging platform APIs to achieve realism in MR lighting.

(See comparison table below for a snapshot of features across AR platforms and engines.)

Platform/ Engine	Ambient Light	Directional Sunlight	Ambient GI / Reflections
ARKit (iOS) – world tracking via AR Foundation or SceneKit	**Yes** – Ambient intensity (lux) and color temperature available [27].	**No** – No direct sun direction in world-tracking (only in face AR) [28] [29].	**Partial** – No spherical harmonics by default in world mode, but environment probes provide reflection cubemaps [30] [31].
ARKit (Face Tracking) – e.g. ARFaceAnchor	**Yes** – Ambient intensity & color [27].	**Yes** – Provides `ARDirectionalLightEstimate` with primary light direction & intensity [32] [33].	**Yes** – Spherical harmonics for full environment lighting on the face [33].
ARCore (Android) – via AR Foundation or ARCore SDK	**Yes** – Ambient intensity and color in basic mode [34].	**Yes** – In *Environmental HDR* mode, gives main light direction & intensity (key light) [16] [3].	**Yes** – Environmental HDR also provides ambient spherical harmonics for image-based lighting [16] [3].

Platform/ Engine	Ambient Light	Directional Sunlight	Ambient GI / Reflections
Magic Leap 2 – (Lumin OS, OpenXR)	**Yes** – Ambient light sensors for overall brightness [10].	**Yes** – ML2's Light Estimation API yields a primary light direction vector and color [4] [35].	**Yes** – Captures an HDR cubemap of the environment for reflections/ambient light [36] [4].
HoloLens 2 (MRTK/Unity)	**Yes** – Has ambient light sensor (used for auto-brightness) [9]; Unity can also get average brightness from camera.	**No** – No built-in directional light estimation for world; must derive via custom camera analysis or user input.	**Partial** – No native spherical harmonics; third-party tools capture environment cubemaps for ambient light [1] [2].
Unity (AR Foundation) – using above AR platforms	**Yes** – Exposes ARKit/ ARCore ambient intensity as `averageBrightness` [26] [37].	**Yes** – If underlying AR platform supports it (ARCore world or ARKit face) it provides `mainLightDirection`, `mainLightIntensity` [6] [38].	**Yes** – Provides `sphericalHarmonics` coefficients when available (ARCore HDR, ARKit face) [6] [39]; supports environment probes for reflections.
Unreal Engine (AR plugins)	**Yes** – Ambient light estimation for ARKit/ ARCore (blueprint `Get AR Ambient Light Estimate`) gives intensity & color.	**Limited** – No official support for directional light in world-facing AR as of UE5 [19] (developers can manually integrate ARCore's data).	**Partial** – Can use SkyLight to capture environment for ambient diffuse; Lumen GI/reflections available on high-end platforms [7] [8].

Toolkits and APIs for Ambient & Directional Lighting

Several AR toolkits expose lighting estimation in accessible ways:

- **Apple ARKit**: ARKit's API provides an `ARLightEstimate` on each ARFrame for world tracking, containing `ambientIntensity` and `ambientColorTemperature` [27]. These correspond to the estimated total light in the scene (normalized such that ~1000 is average indoor lighting) and the scene's color temperature in Kelvin (e.g. ~6500 is neutral white) [27]. Developers using SceneKit or RealityKit can simply enable light estimation; SceneKit will automatically apply the intensity and a temperature-based color correction to its lighting environment. ARKit does **not** compute the sun's direction for world tracking sessions – it assumes lighting is diffuse. However, for **face-tracking sessions** (using the TrueDepth front camera), ARKit provides `ARDirectionalLightEstimate` which extends the base estimate with a `primaryLightDirection` (a unit vector in world coordinates), a `primaryLightIntensity` (in lumens), and even a set of spherical harmonics coefficients for ambient lighting [40] [41]. This unique face-tracking feature treats the user's face as a probe; ARKit analyses the lighting on the face to infer where the strongest light comes from [42]. For

example, an AR face mask can be lit from the same direction as the real key light on the user's face. ARKit also introduced **environment texturing** (AREnvironmentTexturing) which, when enabled, will automatically generate **environment probe anchors** with cube map textures of the surroundings [30] [31]. Developers can use these environment textures for reflections or even sample them to do custom lighting calculations (like finding a bright spot for sun direction, mimicking what ARCore does).

- **Google ARCore**: ARCore's **Lighting Estimation API** has evolved to offer multiple modes [43]. The basic *Ambient Intensity* mode provides a single `ambientIntensity` (as a proportional value or lux) and `ambientColor` (a color correction to adjust white balance) [16] [34]. This roughly tells you how bright and what tint the overall lighting is. ARCore's more advanced *Environmental HDR* mode goes further – it computes a directional light and ambient spherical harmonics [16] [3]. Specifically, ARCore will give a vector for the **main light direction** (in world coordinates relative to the device), a **main light intensity** in lux, and a set of **ambient spherical harmonic** coefficients (usually 9 coefficients for RGB) that represent the diffuse lighting from all directions [16] [3]. ARCore achieves this by analyzing the camera image – essentially using computer vision to estimate a low dynamic range environment map and then boosting it to HDR [44] [3]. Internally it might use a machine learning model or image analysis to find the brightest region (sun or dominant light) and overall color balance of the scene. The ARCore Unity SDK (and AR Foundation) exposes these as `ARLightEstimationData`: for instance, `mainLightDirection`, `mainLightIntensityLumens`, and an array of spherical harmonic values [6] [39]. In practice, ARCore's directional light can be used to cast **real-time shadows** from virtual objects that align with real shadows – developers often attach a Unity `Light` and set its direction each frame to ARCore's `mainLightDirection`, enabling virtual shadows to fall in the same direction as the sun [14] [29]. ARCore's API also provides a **color correction** value (an RGB scale) intended to correct the virtual content's colors to match camera auto-exposure and white balance settings. Using this can make the virtual objects blend more naturally (for example, if the camera image is slightly warmed by auto white-balance, ARCore supplies a color correction to similarly warm the virtual object rendering).

- **Mixed Reality Toolkit (MRTK)**: MRTK is Microsoft's toolkit for HoloLens and other MR devices in Unity. Out of the box, MRTK provides utilities for scene lighting but does not automatically estimate directional lighting from the real world (since HoloLens's API doesn't provide that). However, MRTK includes a **Scene Lighting Adjuster** that can take the HoloLens ambient light sensor reading to adjust RenderSettings ambient light in Unity. Additionally, Microsoft released an **MR Lighting Tools** extension for Unity [45] [46] (now archived but available) which captures lighting using the cameras. This tool uses the HoloLens's PV (photo/video) camera to periodically take fisheye-like snapshots as the user looks around, constructing a cubemap of the environment [1] [47]. It then computes spherical harmonics and even a dominant light direction from that cubemap (by filtering down to low mips and finding the bright spot) [2]. When enabled, it will set Unity's lighting accordingly – updating a directional light to the computed direction and color, and feeding the generated cubemap into a Unity **Reflection Probe** for real-time reflections [1] [2]. Essentially, MRTK's solution mimics ARCore's capabilities via a custom approach since HoloLens doesn't natively do it. HoloLens 2 developers can also use **Windows APIs** in Unity to read the raw ambient sensor for brightness (in nits or lux) and then manually adjust light intensity.

- **Magic Leap**: The Magic Leap 2, via OpenXR, offers a **Light Estimation feature** in its SDK. ML2's Unity API (MagicLeapLightEstimationFeature) allows creating a Light Estimation instance which will

start capturing lighting in the user's environment [36] [48]. It produces an `EstimateData` structure containing a `DirectionalLight` (with a direction vector and color for the strongest light) and an environment **cubemap** for the ambient lighting [4] [35]. This is very similar to ARCore's HDR mode. Under the hood, Magic Leap uses its camera(s) to build an HDR cubemap of the surroundings (developers can specify the resolution, e.g. 256×256 per face) [49] [50]. It likely continuously updates this by sampling different directions as you look around. The primary light direction is derived from that cubemap data. Using the ML API, Unity or Unreal can then assign the estimated directional light to a Light component and apply the cubemap to a Sky Light or reflection probe. Magic Leap's platform also has **ambient light sensors** for overall illumination and a unique **dynamic dimming** technology in the ML2 headset, which can electronically dim the real world background to make virtual content appear more solid in bright conditions [51] [52]. While dynamic dimming is about display hardware rather than rendering, it underscores Magic Leap's holistic approach: they not only estimate lighting to light the virtual content correctly, but also adjust the optical environment to help virtual content keep contrast even under direct sunlight.

In summary, these toolkits (ARKit, ARCore, MRTK, Magic Leap's SDK) give developers high-level access to real-world lighting info. ARKit and ARCore handle it behind the scenes with a mix of sensor data and CV algorithms, exposing simple properties like *brightness* or *mainLightDirection*. Toolkits like MRTK and Magic Leap go further on headworn devices, actively capturing environment maps. By using these APIs, developers can ensure virtual elements have the correct lighting cues: the right shading, color tone, and shadows consistent with the ever-changing real world.

Hardware Considerations for Sunlight Estimation

The capabilities and limitations of MR lighting are heavily influenced by device hardware. **Cameras** are the primary sensors for capturing environmental light. Most AR devices use their RGB camera feed not just for rendering the background but also for analyzing light. For example, ARCore and ARKit use the same images from the rear camera (on a phone or tablet) to estimate lighting. This means the **camera's dynamic range** and auto-exposure behavior affect light estimation. Under very bright sunlight, a phone's camera might stop down exposure, causing the captured image to appear darker – the AR system must compensate for that to avoid underestimating brightness. High-end devices may use **HDR cameras or multiple exposure sampling** to better gauge intense lighting (some AR frameworks take exposure values into account – ARKit's ambient intensity is scaled based on camera exposure settings [27]).

Beyond cameras, many devices include dedicated **ambient light sensors**. Smartphones have them for auto-brightness; HoloLens has one on its sensor bar [9]; Magic Leap 2 has several. These typically are photodiodes that measure overall illumination in lux. They have a wide field of view and react quickly, which is useful for detecting sudden brightness changes (like walking outdoors into sunlight). However, ambient sensors lack directional information and color data. They might be used to bias or validate the camera-based estimates – e.g. if the camera is mostly seeing a dark corner but the ambient sensor on top of the device senses sunlight, the system might infer there's a bright light just outside the camera view. Some research prototypes consider additional sensors: for instance, devices could use **UV sensors** or multiple light sensors around the headset to triangulate a light source, but this is not common in current commercial MR devices.

Device orientation and location can also be leveraged. If an AR headset knows its geolocation and the current time, it can compute the sun's expected position in the sky. Indeed, architecture apps use this

approach: by tying into GPS and clock, an AR app can visualize where the sun *should* be and cast virtual shadows accordingly for design simulations. However, most real-time engines don't do this automatically for lighting; it could be an added feature for outdoor AR – effectively using an *astronomical model* as a backup to sensor-based estimation, especially if the sun is not directly visible to sensors (e.g. behind clouds, but one might still want to simulate approximate sunlight).

Different classes of devices handle **sunlight tracking** uniquely: - **Optical see-through headsets** (HoloLens 2, Magic Leap) have transparent displays. They face a major hardware challenge: making virtual content visible under bright sunlight. HoloLens 2's holograms can appear washed out outdoors because the display can't output the same luminance as the sunlit world. Magic Leap 2 addressed this with **dynamic dimming** – essentially an electrochromic layer that darkens the real-world view like sunglasses when needed [51] [53]. This doesn't affect the lighting estimation per se, but it improves visual coherence by letting virtual lighting *look* right relative to a dimmer real backdrop. These headsets also often incorporate **inertial sensors** and multiple cameras for environment understanding; the cameras can be fish-eye for mapping but low-resolution for lighting. HoloLens's environment understanding cameras are grayscale and primarily for tracking, not useful for color light estimation. Therefore, HoloLens relies on its color camera (which is typically used for photos/videos) for any color lighting capture. The Microsoft Research Mode API allows accessing that camera and even IR cameras (though IR can measure patterns of illumination in infrared, which might be used for some lighting feedback in dark conditions). On HoloLens, if developers want a full environment cubemap, they must take over the color camera feed at intervals (as the MR Lighting Tool does) [1] [47]. The **impact of sunlight on sensors** is another factor: direct sun into the cameras can cause lens flares or clip sensor readings, which can confuse algorithms. Some devices might detect when the sun is in the camera view (using gyroscope orientation plus known sun position) and handle that case specially (for example, ARCore could disregard pixel data that's blown out and instead rely on overall brightness).

- **AR handheld devices (phones/tablets)** have the advantage of robust camera modules with good auto-settings and sometimes even multiple cameras. Some newer phones feature **wide-angle cameras** that could serve as enviromental light catchers – for instance, a phone might use its ultrawide camera to get a nearly 180° view for lighting estimation while using the main camera for rendering the AR scene. This isn't explicitly done by ARCore/ARKit yet, but it's a possibility for future hardware integration. Phones also can leverage their **front-facing sensors**; interestingly, iPhones with TrueDepth (for FaceID) use an IR dot projector and IR camera – those could measure how bright the IR illumination is on your face to infer ambient IR light, but that's more for indoors and not directly used in ARKit lighting. Generally, mobile AR frameworks stick to the main RGB camera feed.

- **Specialized AR hardware**: Some niche MR devices or prototypes have taken unique approaches. For example, an AR headset could include a **built-in fisheye lens** pointed upward, solely to capture the sky and lighting (there have been research papers mounting a skyward fisheye on AR headsets for constant environment mapping). While not in mainstream products, such ideas indicate how hardware might evolve to better capture sunlight in all conditions (imagine a headset with tiny dome light sensors giving a full 360° light map continuously).

In all cases, **power and performance** are considerations: continuously capturing images for lighting or running heavy vision algorithms can tax battery and CPU/GPU. Hardware constraints often mean that lighting estimation is updated at a lower rate (e.g. ARCore might update the light estimate only a few times per second, not every frame) to save resources. Some devices could offload this to a dedicated ISP (Image Signal Processor) or AI accelerator – for instance, Qualcomm Snapdragon chips have an ISP that could run

an efficient HDR analysis for AR without burdening the CPU [54] [55]. We see hardware-software co-design: AR platforms are starting to utilize those chip features to make light estimation both fast and low-power.

Summary: Hardware defines what data is available for MR lighting. Cameras provide the rich data for direction and color; ambient sensors give quick brightness cues; specialized solutions like Magic Leap's dimming improve visual result under sun. Knowing these hardware aspects helps in understanding the limitations – e.g., why an AR app might sometimes misjudge lighting (the camera auto-exposure fooled it, or the sensor was saturated by sunlight) – and in designing MR experiences that remain robust (perhaps by smoothing out sensor data or using additional cues like time/location when possible).

Research and Emerging Techniques

The pursuit of *photorealistic lighting* in MR has been a hot topic in academic and industry research. A substantial body of research addresses how to estimate and render lighting so that virtual and real scenes blend seamlessly. A 2020 survey by Alhakamy and Tuceryan categorized the pipeline into **light source acquisition, lighting estimation, and global illumination composition** [13]. Key techniques and papers from the last decades include:

- **Photometric Registration (Classic Approach)**: Early AR research (e.g., by Debevec et al. and later researchers in the 2000s) introduced the idea of using a **mirror ball or calibrated reference** in the scene to capture lighting [11] [56]. By taking a high-dynamic-range photo of a chrome sphere, one can obtain an environment map of the scene's lighting. In AR, this was often simplified (since you can't assume a chrome sphere is present in user's environment), but it set a framework: if any known geometry with known material is in view (even part of the user's hand, or the glossy reflection off an eye), it can be leveraged as a light probe.

- **Realtime Illumination from Camera Image**: Researchers have developed algorithms to estimate a scene's illumination directly from images. One notable thread is using **spherical harmonics from a single image**. For example, a method might detect the sky region (bright, blue) versus ground (darker) in the camera frame and assume a typical outdoor lighting model (sun + sky). Another method uses the shading on arbitrary objects in view: if AR can identify a planar surface and its reflectance (perhaps using ARKit's plane and texture data), it might infer how light falls on that plane to get the light direction. There are papers on **shadow-based light direction estimation** – if the camera sees a shadow of a real object, computer vision can extract its direction to set the virtual sun accordingly [57]. One 2018 paper presented a deep neural network that inputs an LDR image and outputs an HDR lighting environment (often represented by coefficients or a low-res environment map). Such AI-driven approaches have been shown to handle complex indoor scenes where multiple lights exist – the network can output several basis lights or an HDR map that, while not exact, gives plausible illumination for virtual objects. The AR community has been integrating some of these: for example, ARCore's lighting may incorporate learned models from a large dataset of indoor images with known lighting.

- **Augmented Reality *Relighting***: There is emerging research not only to light virtual objects correctly, but also to *relight the real scene* in some cases. While generally MR doesn't alter real lighting, one concept is *coherent illumination*: if a virtual object is very bright or emissive (like a virtual fire), some experiments have tried to cast that light into the real world via projection or by influencing the camera feed (for video AR). A notable challenge is two-way illumination – how virtual and real lights

affect each other. This remains mostly in research (e.g., using a projector in the loop to actually shine light). For MR headsets, researchers explored using additional optical elements to dim or tint real light in selective areas, but that's beyond current products.

- **Global Illumination Solutions**: On the rendering side, techniques from computer graphics like **Precomputed Radiance Transfer** (PRT) and **Instant Radiosity** have been tested for AR. If the environment's geometry is known (e.g., via 3D scan), one can simulate how light bounces in that environment and precompute some form of response to dynamic light. For example, one could precompute how sunlight entering a particular room (with known layout) illuminates it at different angles, then at runtime just pick the closest match for the current sun angle. Some AR academic projects pre-scan the real room and then allow virtual objects to be illuminated by *both* the real sun and secondary bounce light computed from the room's model – yielding very convincing results but requiring an upfront scan of the scene. With devices like HoloLens 2, which can map rooms with spatial mapping meshes, there's potential to combine that with lighting algorithms: a research prototype might take the mesh and compute in real-time how a virtual light (like the sun) bounces off those meshes onto virtual objects or even modulate virtual shadows where real walls are (for consistency).

- **Use of Reflective Objects in Scene**: The GLEAM system (2020) by Prakash et al. proposed collaborative sensing for lighting, where multiple devices or multiple viewpoints build a more complete light map [58] [12]. GLEAM also demonstrated using a known **specular object in the scene** (like a reflective toy) as a target: by observing the reflections on it from different angles, the system reconstructs an accurate environment lighting (essentially solving an inverse lighting problem) [11] [12]. Such techniques are not in mainstream use (they need a known object), but they point toward hybrid approaches where the AR system might actively use objects in your environment as light probes. Imagine an AR app recognizing a metallic fridge in view and using its highlights as a cue for lighting – not far-fetched if object recognition and lighting estimation are combined.

- **Industry Whitepapers and SDK innovations**: Companies occasionally publish insights – for instance, Google's blog on ARCore's Environmental HDR explained how they approximate a **high dynamic range lighting** from a normal camera image [44]. Unity and Unreal's forums are full of tricks by developers to get better AR lighting (like capturing a panoramic shot at app start for static lighting, or using light probes around an AR model for better GI). An interesting industry note is from 8th Wall (WebAR) – they introduced real-time reflections by using WebXR lighting estimation plus some clever environment mapping on the web, showing that even in browser-based AR one can achieve dynamic sunlight reflections [25].

- **Emerging hardware for lighting**: Research isn't limited to software – some works propose hardware like **light field sensors** or specialized HDR fisheye cameras on AR headsets. One example is a paper that put a small omnidirectional photodiode array on a headset to continuously measure incident light from every direction (effectively capturing the 4π steradian light probe in hardware). While not commercial, it shows how far one can go: a future AR headset might have a built-in "chrome ball" sensor capturing the lighting at all times, feeding that to the renderer for ultimate accuracy.

The overarching trend in research is to reduce the gap between *real* and *virtual* lighting. Many techniques from offline VFX (where one has time to manually calibrate lights or do HDR photography of the set) are

being translated into automated, real-time methods suitable for AR. **Machine learning, fisheye environment capture, collaborative sensing, and advanced GI algorithms** are all being actively explored to make MR lighting more adaptive and realistic [13] [11]. As these mature, we can expect MR experiences where virtual objects are nearly indistinguishable from real ones even as the sun goes behind a cloud or as you move from room to room.

Commercial Applications and Use Cases

Dynamic sunlight-responsive lighting in MR isn't just a technical showcase – it unlocks practical and creative applications across industries:

- **Architecture and Construction**: Architects use AR to visualize buildings and additions on site, and realistic lighting is crucial. For example, an AR app can place a 3D model of a proposed building on its real future location and show **shadows** cast by the virtual structure in sync with the actual sun. This helps in evaluating how the building will impact natural light at different times of day [59] [60]. Conversely, one can simulate how sunlight will penetrate a space: an interior design AR app might let a user place virtual furniture and see *real-time* how morning sun from a window would light a sofa. Without dynamic lighting, these AR models look flat; with sunlight response, an architect can "walk" a client through a building at sunset, with the AR model glowing warm as it would in reality. Firms are indeed leveraging AR for **site analysis**, checking solar exposure and shadow studies through AR visualization [61] [62]. There are apps (like Sun Seeker, Sun Locator, etc.) that use AR to show the sun's path and cast virtual shadows of real objects for planning purposes [60] [63]. These might not insert new virtual objects, but they demonstrate the value of aligning with real sunlight – architects can stand on-site and see where shadows will fall at any date/time through AR overlays. When virtual objects are involved, ensuring they are lit by the actual sun makes them far more credible for decision-making (e.g., how a new extension might shade the neighbor's property is quickly understood when the AR model's shadow lines up with the real sun angle).

- **Training and Simulation**: In sectors like military, aviation, or first-responder training, MR is used to add virtual elements to real environments. Lighting coherence can be critical for realism and even for efficacy of training. Consider a firefighting training scenario in an actual training ground with AR overlays of virtual fire: if it's daytime, the virtual fire and smoke should appear lit by the same sunlight as the surroundings for a believable experience. Likewise in military exercises, virtual vehicles or targets inserted in a real landscape need proper shading; a virtual tank in shadow vs. sunlight might change how trainees react. MR training systems thus try to respect environmental lighting to avoid visual discrepancies that could reduce immersion or give away the "fakeness" of a target. **Industrial maintenance AR** is another use case: a technician wearing an AR headset outdoors to overlay service info on a solar panel array will benefit from sunlight-responsive rendering – for instance, virtual arrows or highlights might adjust brightness/contrast when under harsh sun versus under a cloud to remain visible. Some enterprise AR applications explicitly mention adaptive rendering to ensure visibility in varying light conditions (e.g., **dynamic contrast UIs** and shading cues that match real lighting so that virtual instructions don't get lost in glare).

- **Immersive Art and Entertainment**: Artists are exploring AR installations that interact with the environment. One example: an AR art piece that "blooms" with the sun. Imagine an AR sculpture placed in a park – at noon it shines brightly and casts long virtual shadows that align with real ones, while in the evening it dims in color temperature to match the golden hour. Artists use these

techniques to create a stronger connection between the digital art and the physical world's diurnal cycle. There have been AR exhibits where the virtual content explicitly responds to weather and lighting – e.g., virtual creatures that seek shade on a sunny day (using the device's light estimate to know it's bright). Another artistic use is in storytelling: an AR narrative app might have a scene that only appears at a certain time of day, blending with the real light. For instance, an app could render a virtual portal that is only visible when the real sunlight from a window falls on a certain wall – this requires precise alignment of virtual light (the portal might glow only when the sun angle is right). Without accurate sunlight tracking, these creative ideas wouldn't be convincing.

- **Gaming and Consumer AR**: Popular mobile AR games and apps benefit from realistic lighting too. Niantic (makers of *Pokémon GO* and *AR outdoor experiences*) have demonstrated improved lighting as part of their Real World Platform – for example, their demo showed virtual Pokémon casting believable shadows based on the sun, making them feel present in the environment. While many mobile AR games still use simple fixed lighting (for performance reasons), we see a push toward using ARCore's Environmental HDR so that if you catch a monster at sunset, its coloring and shadow direction reflect that setting. Snapchat Lenses have begun to incorporate environmental color tone; a filter might, for instance, add a virtual object on your desk that takes on the cool hue of your office's fluorescent lights, or the warm tint of a sunset in the background – all driven by ARKit/ARCore light estimation. **E-commerce AR** (placing products in your space) is another commercial area: when a user tries an AR sofa in their living room, the app can adjust the virtual sofa's material appearance to the room's lighting – making it not only look more realistic but also giving the user a better sense of the fabric color under their home's lighting conditions. Companies like IKEA (with IKEA Place app) have noted that realistic lighting and shadows significantly improve users' trust in the AR visualization of furniture. Automotive companies showcasing cars in AR will ensure the car has accurate reflections of the sky and sun glints in the correct place; for example, a virtual car viewed through your phone in your driveway at noon could show a bright specular highlight where the real sun hits the hood, and a soft shadow under the car matching the real sun's angle. These details make the AR car appear as if it's truly there, aiding marketing and user engagement.

- **Film/TV Pre-visualization**: MR is also used on set in filmmaking (using tablets or see-through devices to line up VFX). Matching real sun with virtual elements is crucial so directors can pre-visualize composite shots. If a director sees through an iPad that a CGI creature will be added to a scene, the AR preview can show the creature lit by the current sunlight on location. This helps decide camera exposure and placement. Companies working on **mixed reality virtual production** use techniques to sync stage lighting with virtual lighting – sometimes even having IoT-connected lights that adjust to simulate certain times of day in AR for training or entertainment events.

Each of these applications gains value from dynamic lighting: it's not just about looking pretty, but about **functional integration** of virtual content. In architecture, it answers questions about light and shadow for real designs [59] [64]; in training, it ensures the virtual cues don't break immersion; in art, it creates a poetic connection to nature; in retail, it builds confidence that "what you see is what you get." Achieving this requires the methods discussed – real-time estimates and responsive rendering – which is why engines and platforms have invested in these features. As hardware improves and techniques mature, we'll likely see even wider adoption: more apps will turn on environmental HDR by default, and new creative uses will emerge (like AR characters that "sunbathe" when it's clear or cast longer shadows in winter vs summer, etc., using the real sun as an interactive element of the experience).

Challenges and Limitations

Despite the advancements, several challenges remain in achieving perfectly coherent real-time lighting in MR:

- **Accuracy and Stability**: Light estimation algorithms can sometimes be wrong or noisy. A common issue is *flicker* or *jumping* in estimated lighting. For example, as a user moves their phone, the auto-exposure might fluctuate, causing ARCore's reported ambient intensity to oscillate frame to frame. This can make a virtual object's brightness pulsate unnaturally. To mitigate this, developers often smooth or lerp the lighting values over time, sacrificing responsiveness for stability. Directional estimation can also flip unpredictably if the scene lacks a clear light source – ARCore might suddenly change the mainLightDirection if you point the camera from the sky to a darker area. ARKit world tracking doesn't even attempt directional light for this reason – it's hard to do robustly with a single camera without context, so Apple avoids potentially wrong directions (they leave it to environment probes or the developer's logic) [65]. Another accuracy issue arises with **multiple light sources**: indoors you might have the sun through a window and a lamp on; current systems typically provide only *one* directional light (the strongest). This means the virtual object won't perfectly match a scene lit by complex lighting. It will align with the strongest source, but might miss the secondary illumination (leading to slightly off shading or missing a second shadow).

- **Latency**: There's a time cost to estimating lighting. If using camera images, the pipeline might introduce a few frames of delay. If you walk from a dark indoor hallway out into bright sun, the real world changes instantly, but the AR lighting might lag for a second, during which the virtual content looks wrongly lit (too dark) until the sensors catch up and the algorithms update the estimates. This is especially true if heavy processing or multiple frames averaging is used to reduce noise. Also, on headsets like HoloLens using the lighting capture approach, the cubemap might only update when you move your head significantly (to stamp a new view) [47], so rapid changes in lighting (sun going behind a cloud) might not reflect on the virtual objects until the user looks around enough to update the cubemap.

- **Performance Constraints**: Mobile hardware can't afford very sophisticated lighting computations each frame. Real-time global illumination (like multi-bounce ray tracing) is usually beyond the reach of battery-powered AR glasses or phones, so approximations are used. Even spherical harmonics, while efficient, cover only diffuse lighting; specular highlights from the sun require either a proper reflection computation or a high-resolution environment map. Engines often use a **reflection probe texture** for specular, but updating a probe texture frequently is expensive (capturing six faces of a cube at decent resolution). ARCore avoids this by not providing an actual cube map, instead giving already-processed SH and directional data. Tools like MR Lighting on HoloLens capture cubemaps gradually for this reason (one face at a time as you turn, to spread cost) [47] [66]. There's also the cost of **additional draw calls or shader complexity**: enabling dynamic lighting means you can't bake lighting into textures (which is a common performance trick for static scenes). Everything virtual in AR is lit per-pixel each frame, which can tax the GPU if the scene is heavy. This is one reason AR apps tend to have relatively simple virtual content – to leave headroom for dynamic lighting and other AR processes.

- **Visual Coherence Issues**: Even with good estimates, achieving perfect blending can be hard. Consider **color calibration**: the real-world view (either via camera feed on mobile or through one's

eyes on optical see-through) has certain color characteristics. If the AR content's lighting color doesn't exactly match, the virtual object might appear slightly off (too bluish or too warm). ARKit giving color temperature helps, but matching human perception of white balance is tricky. The display technology further complicates it: phone screens have a certain gamut and might be showing the camera feed with some tonemapping, so the virtual object rendered in HDR then composited might not undergo the exact same tonemapping. Discrepancies can lead to the virtual object looking a tad more saturated or contrasty than the background video. On optical see-through, coherence is even tougher: since holograms are essentially additively overlaid, they can't display true black or cast real darkness. A virtual shadow on HoloLens cannot *darken* the real ground – at best, one could render a semi-transparent gray, but that just tints the light coming from the display, not actually blocking real light. So in bright conditions, virtual shadows are practically invisible on optical see-through; thus the user sees real objects with dark shadows but the virtual object might oddly lack a corresponding dark shadow on the ground (it can only produce a faint overlay). Magic Leap's dimming helps a bit by lowering overall real light, but still, fully dark shadows remain a challenge. **Contrast** is another coherence issue: if the real scene has very high contrast lighting (harsh sun), a virtual object might require very bright highlights and very dark self-shadows to match. However, limited dynamic range of displays might clip those, resulting in a somewhat flatter look for the virtual element compared to reality.

- **Environmental Understanding Limits**: Without a perfect 3D model of the real world, virtual lighting can also break in terms of interaction with real surfaces. For instance, a virtual character might be correctly lit by the sun, but ideally it should also drop a shadow *onto real ground* or walls. In mobile AR, a common cheat is a "shadow catcher" plane placed at the estimated ground, so the character casts a shadow onto that invisible plane which is then composited on the video. This works if the ground is flat and one-colored, but if the ground is uneven or cluttered, the fake shadow may look wrong (floating or not conforming to objects). Moreover, if the virtual object moves behind a real object, ideally its lighting should be partially occluded (like if it goes under a tree, it should enter a shadow). Without knowing the tree's shape, AR can't dim the virtual lighting appropriately. Advanced AR might use the device's depth sensor to identify real shadowing geometry – e.g., HoloLens could detect a real wall casting a shadow and then reduce light on virtual objects in that region. Some experimental systems combine segmentation (to know where real shadows are) with lighting – but it's complex and not mainstream yet. This limitation means virtual content sometimes looks "too lit" in places it shouldn't, or vice versa.

- **Cross-Platform and Consistency**: Because ARKit and ARCore have different lighting systems (one using more physical units like lumens, the other somewhat relative intensities), developers making cross-platform MR apps often find the need to tune things per platform. An object in an ARCore experience might appear brighter or differently colored than in ARKit under the same real conditions due to how each calculates and applies lighting. Achieving consistency so that an iPhone and Android user see similar results under sunlight is challenging. There's ongoing work in standards (like OpenXR extensions for lighting estimation) to unify this [67], but as of now, differences remain.

- **User Perception and Device Limitations**: Even if the lighting is physically correct, human perception might expect certain cues that are hard to reproduce. For example, *glare* – in a bright sunny scene, real objects might have lens flares or bloom in our vision. Virtual objects rendered normally won't have the camera artefacts or eye response that real bright objects do. Some AR apps intentionally add a slight bloom or glare to virtual highlights under sunlight to mimic this. However,

too much can look fake. Additionally, bright daylight can wash out phone screens, meaning even if lighting calc is right, the user might not *see* the virtual object clearly. That's not a rendering issue per se, but it limits effectiveness. Magic Leap-like dimming is one hardware fix, but on phones or HoloLens, often the virtual content is just tough to see under direct sun regardless of lighting correctness, simply due to brightness competition.

In essence, while current MR technology can do an impressive job responding to sunlight, **it's not perfect**. Developers must account for and mitigate these issues: smoothing inputs to avoid flicker, designing content that remains visible (e.g., adding outlines or adjusting color contrast dynamically), and guiding users (some AR apps literally prompt the user to "scan the environment for lighting" by moving the device around at start). It's also an active area of improvement. Each generation of AR SDKs refines light estimation (e.g., ARCore gradually improved its color accuracy and stability from version to version), and devices get better sensors. The ultimate limitation is that we're trying to recreate the full complexity of real-world illumination with limited data and display capabilities. So long as that gap exists, there will occasionally be moments where the illusion breaks – a virtual object might look a bit too dim or a shade off-color compared to its surroundings. Recognizing these limitations, developers and researchers continue to iterate on solutions, inching closer to seamless visual coherence in any lighting, from a dim indoors to blinding noon sun. Each challenge overcome brings MR closer to truly blending realities under the same sun.

Sources:

- Unity AR Foundation documentation – Light Estimation (ARCore HDR mode and properties) [16] [3] [14] [29]
- Google ARCore Developer Guide – Environmental HDR Light Estimation [16] [3]
- Andy Jazz (2020) *ARKit Light Estimation* – ARKit ambient and directional light details [27] [40]
- Apple Developer Documentation – ARKit Lighting (via ARDirectionalLightEstimate for face tracking) [68] [33]
- Microsoft MRTK Lighting Tools – Environment cubemap and directional light extraction [1] [2]
- Magic Leap 2 Developer Guide – Light Estimation API (directional light and cubemap) [4] [35]
- Tom's Hardware – HoloLens sensors (ambient light sensor presence) [9]
- ActLight industry blog – Overview of AR light sensor usage (mentions ARCore, ARKit ambient light integration) [69] [70]
- Epic Games Tech Blog – Unreal Engine 5 Lumen for dynamic global illumination [7] [8]
- Sun Locator (2025) – AR for sun path and shadow simulation in architecture [59] [60]
- AUGmentecture (2025) – AR in site planning (sunlight and orientation analysis) [61] [62]
- Alhakamy & Tuceryan (2020) – *Real-time Illumination for Photorealistic AR/MR* (ACM Computing Surveys) [13]
- Prakash et al. (2020) – *GLEAM: Generating Light Estimates Across MR devices* (concept of using reflective objects) [11] [56]
- TutorialsForAR (2021) – *Using Light Estimation in AR (Unity)* – list of ARKit/ARCore light properties [5] [6] .

---

[1] [2] [45] [46] [47] [66] GitHub - microsoft/MRLightingTools-Unity: A Unity library and MRTK extension for estimating and replicating the current environment's lighting on Mixed Reality devices.
https://github.com/microsoft/MRLightingTools-Unity

3 16 34 43 44 Realistically light virtual objects in a scene | ARCore | Google for Developers
https://developers.google.com/ar/develop/unity-arf/lighting-estimation/developer-guide

4 35 36 48 49 50 Light Estimation API Overview | MagicLeap Developer Documentation
https://developer-docs.magicleap.cloud/docs/guides/unity-openxr/light-estimation/unity-light-estimation-api-overview/

5 6 26 37 38 39 Day 16 : Using Light Estimation in AR using ARKit and ARCore with Unity. - Tutorials For AR Tutorials For AR
https://tutorialsforar.com/using-light-estimation-in-ar-using-arkit-and-arcore-with-unity/

7 8 21 22 23 Unreal Engine 5 goes all-in on dynamic global illumination with Lumen
https://www.unrealengine.com/en-US/tech-blog/unreal-engine-5-goes-all-in-on-dynamic-global-illumination-with-lumen

9 What's Inside Microsoft's HoloLens And How It Works | Tom's Hardware
https://www.tomshardware.com/news/microsoft-hololens-components-hpu-28nm,32546.html

10 Magic Leap 2 Devices
https://www.magicleap.com/legal/devices-ml2

11 12 56 58 prakashsidd18.github.io
https://prakashsidd18.github.io/papers/prakash-thesis.pdf

13 Real-time Illumination and Visual Coherence for Photorealistic Augmented/Mixed Reality
https://scholarworks.indianapolis.iu.edu/items/c8cb7930-e8fc-4d37-8710-cfcbb4d4d5fe

14 15 29 Camera samples | AR Foundation | 6.1.1
https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@6.1/manual/samples/features/camera.html

17 18 31 Realistic reflections and environment textures in ARKit - Krootl Agency
https://www.krootl.com/blog/realistic-reflections-and-environment-textures-in-arkit

19 20 unreal.ARLightEstimationMode — Unreal Python 5.2 (Experimental) documentation
http://dev.epicgames.com/documentation/en-us/unreal-engine/python-api/class/ARLightEstimationMode?application_version=5.2

24 Get the lighting right | ARCore
https://developers.google.com/ar/develop/lighting-estimation

25 Introducing 8th Wall Realtime Reflections: create Realistic WebAR ...
https://www.8thwall.com/blog/post/41425530761/introducing-8th-wall-realtime-reflections-create-realistic-webar-experiences-that-win-customers

27 32 33 40 41 68 ARKit 911 — Light estimation. In ARKit 4.0 to get a well-tracked... | by Andy Jazz | Mac O'Clock | Medium
https://medium.com/macoclock/arkit-911-light-estimation-e10adf49ab32

28 42 ARCore User face tracking and mainLightDirection estimation
https://discussions.unity.com/t/arcore-user-face-tracking-and-mainlightdirection-estimation/821761

30 environmentTexturing | Apple Developer Documentation
https://developer.apple.com/documentation/arkit/argeotrackingconfiguration/environmenttexturing

51 [PDF] Magic Leap 2's Advanced AR Platform and Revolutionary Optics
https://www.tdsynnex.com/na/us/magic-leap/wp-content/uploads/sites/85/2023/10/Whitepaper-Magic-Leap-2-Optics.pdf

52 Sensor Data | MagicLeap Developer Documentation
https://developer-docs.magicleap.cloud/docs/guides/features/sensor-data/

53 Magic Leap 2- Base Model | My Site - Avrio Analytics
https://www.avrioanalytics.com/product-page/magic-leap-2-base-model

54 55 69 70 A Dive into the Top 10 Leading AR Companies and Their Light Sensor Technologies | ActLight
https://act-light.com/a-dive-into-the-top-10-leading-ar-companies-and-their-light-sensor-technologies/

57 [PDF] Realtime Estimation of Illumination Direction for Augmented Reality ...
https://library.imaging.org/admin/apis/public/api/ist/website/downloadArticle/cic/20/1/art00020

59 60 63 64 Shadow simulation and sun path planning: essential tools for architects – Sun Locator
https://sunlocator.app/blog/shadow-simulation-and-sun-path-planning-essential-tools-for-architects/

61 62 AR in Site Analysis and Planning for Architects
https://www.augmentecture.com/blog/ar-in-site-analysis-and-planning-for-architects/

65 Light Estimation: Light Direction etc. unavailable - Unity Discussions
https://discussions.unity.com/t/light-estimation-light-direction-etc-unavailable/808276

67 Light Estimation Overview | MagicLeap Developer Documentation
https://developer-docs.magicleap.cloud/docs/guides/unity-openxr/light-estimation/unity-light-estimation-overview/