Niko Kirste
Faculty of Computer Science

Real-time Lighting for Mixed Reality Applications

Final Presentation, INF-PM-FPA
Dresden, 16.10.2024

Table of Contents

Motivation

Lighting in traditional VR Applications:

- Closed environment
- No interaction with real world lighting

→ static precomputed lighting

→ low overhead

Lighting in MR Applications:

- Ongoing interaction with real world lighting
- Constantly changing
- Immersion depends on correct lighting

→ need to adapt

Research Question

How can we get lighting information from the physical world to the MR application in real-time?

- \+ **Accurate colors**
- \+ **Low overhead on the HMD**
- \+ **Preferably multi-user enabled**

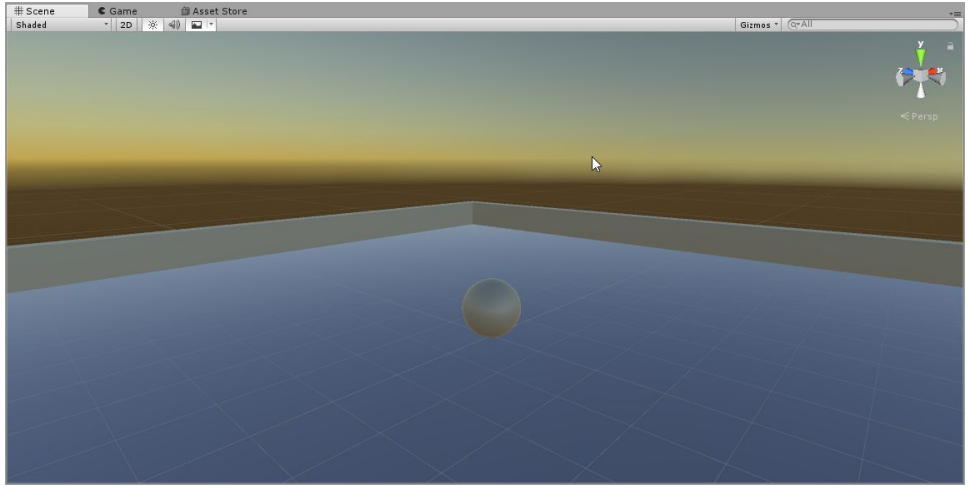
Existing Methods

Skybox:

- **Static or dynamic 360° texture**
- **Projected into infinity**
- **Contributes to single ambient light probe**

→ no spatial gradient

→ same lighting for every object



Related Work

Light Probe generation

PointAR: Efficient Lighting Estimation for Mobile Augmented Reality

Yiqin Zhao(✉) and Tian Guo(✉)

Worcester Polytechnic Institute, Worcester, USA
{yzhao11,tian}@wpi.edu

Abstract. We propose an efficient lighting estimation pipeline that is suitable to run on modern mobile devices, with comparable resource complexities to state-of-the-art mobile deep learning models. Our pipeline, PointAR, takes a single RGB-D image captured from the mobile camera and a 2D location in that image, and estimates 2nd degree spherical harmonics coefficients. This estimated spherical harmonics coefficients can be directly utilized by rendering engines for supporting spatially variant indoor lighting, in the context of augmented reality. Our key insight is to formulate the lighting estimation as a point cloud-based learning problem directly from point clouds, which is in part inspired by the Monte Carlo integration leveraged by real-time spherical harmonics lighting. While existing approaches estimate lighting information with complex deep learning pipelines, our method focuses on reducing the computational complexity. Through both quantitative and qualitative experiments, we demonstrate that PointAR achieves lower lighting estimation errors compared to state-of-the-art methods. Further, our method requires an order of magnitude lower resource, comparable to that of mobile-specific DNNs.

Keywords: Lighting estimation · Deep learning · Mobile AR

LitAR: Visually Coherent Lighting for Mobile Augmented Reality

YIQIN ZHAO, Worcester Polytechnic Institute, USA
CHONGYANG MA, Kuaishou Technology, China
HAIBIN HUANG, Kuaishou Technology, China
TIAN GUO, Worcester Polytechnic Institute, USA

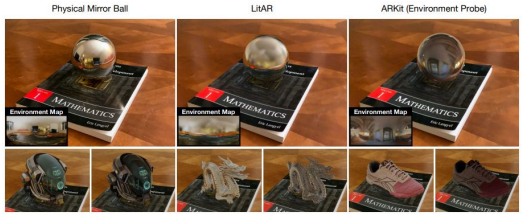


Fig. 1. Rendered objects with LitAR compared to a physical mirror ball and with ARKit [34]. *Row 1:* LitAR produces visually-coherent reflective rendering while ARKit fails; the bottom part of the ARKit-rendered ball does not faithfully reflect the book cover. The top part of the LitAR-rendered ball has an intentionally gradient blurring effect for quality-performance trade-offs (see §4.2). *Row 2:* LitAR achieves more realistic and visually coherent rendering effects than ARKit for objects with different materials.

An accurate understanding of omnidirectional environment lighting is crucial for high-quality virtual object rendering in mobile augmented reality (AR). In particular, to support reflective rendering, existing methods have leveraged deep learning models to estimate or have used physical light probes to capture physical lighting, typically represented in the form of an environment map. However, these methods often fail to provide visually coherent details or require additional setups. For

Authors' addresses: Yiqin Zhao, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA, USA, yzhao11@wpi.edu; Chongyang Ma, Kuaishou Technology, 6 Shangdi West Road Haidian, Beijing, Beijing, China, chongyangm@gmail.com; Haibin Huang, Kuaishou Technology, 6 Shangdi West Road Haidian, Beijing, Beijing, China, jackiehuanghaibin@gmail.com; Tian Guo, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA, USA, tian@wpi.edu.

Accidental Light Probes

Hong-Xing Yu[1] Samir Agarwala[1] Charles Herrmann[2] Richard Szeliski[2] Noah Snavely[2]
Jiajun Wu[1] Deqing Sun[2]
[1]Stanford University [2]Google Research

Abstract

Recovering lighting in a scene from a single image is a fundamental problem in computer vision. While a mirror ball light probe can capture omnidirectional lighting, light probes are generally unavailable in everyday images. In this work, we study recovering lighting from accidental *light probes (ALPs)—common, shiny objects like Coke cans, which often accidentally appear in daily scenes. We propose a physically-based approach to model ALPs and estimate lighting from their appearances in single images. The main idea is to model the appearance of ALPs by photogrammetrically principled shading and to invert this process via differentiable rendering to recover incidental illumination. We demonstrate that we can put an ALP into a scene to allow high-fidelity lighting estimation. Our model can also recover lighting for existing images that happen to contain an ALP*[1].

I'd rather be Shiny. — Tamatoa from Moana, 2016

Figure 1. (Left) From an image that has an accidental light probe (a Diet Pepsi can), we insert a virtual object (a Diet Coke can) with estimated lighting using the accidental light probe (Middle), and using estimated lighting from a recent state-of-the-art lighting estimation method [49] (Right). Note how our method better relights the inserted can to produce an appearance consistent with the environment (e.g., the highlight reflection and overall intensity).

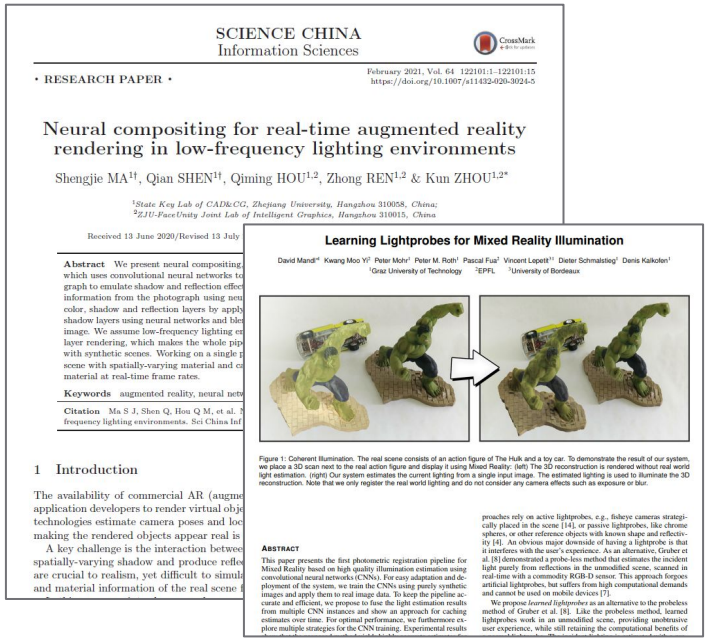
1. Introduction

Traditionally, scene lighting has been captured through the use of light probes, typically a chromium mirror ball; their shape (perfect sphere) and material (perfect mirror) allow for a perfect measurement of all light that intersects the probe. Unfortunately, perfect light probes rarely appear in everyday photos, and it is unusual for people to carry them around to place in scenes. Fortunately, many everyday objects share the desired properties of light probes: Coke cans, rings, and thermos bottles are shiny (high reflectance) and curved (have a variety of surface normals). These objects can reveal a significant amount of information about

In general, recovering scene illumination from a single view is fundamental for many computer vision applications such as virtual object insertion [9], relighting [46], and photorealistic data augmentation [51]. Yet, it remains an open problem primarily due to its highly ill-posed nature. Images are formed through a complex interaction between geometry, material, and lighting [21], and without precise prior knowledge of a scene's geometry or materials, lighting estimation is extremely under-constrained. For example, scenes that consist primarily of matte materials reveal little information about lighting, since diffuse surfaces behave like low-pass filters on lighting during the shading process [38], eliminating high-frequency lighting information. To compensate for

Related Work

Neural Network based Lighting



SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

February 2021, Vol. 64 122101:1–122101:15
https://doi.org/10.1007/s11432-020-3024-5

Neural compositing for real-time augmented reality rendering in low-frequency lighting environments

Shengjie MA[†], Qian SHEN[†], Qiming HOU[1,2], Zhong REN[1,2] & Kun ZHOU[1,2*]

[1]State Key Lab of CAD&CG, Zhejiang University, Hangzhou 310058, China;
[2]ZJU-FaceUnity Joint Lab of Intelligent Graphics, Hangzhou 310015, China

Received 13 June 2020/Revised 13 July

Abstract We present neural compositing which uses convolutional neural networks to graph to emulate shadow and reflection effect information from the photograph using neu color, shadow and reflection layers by apply shadow layers using neural networks and ble image. We assume low-frequency lighting e layer rendering, which makes the whole pip with synthetic scenes. Working on a single p scene with spatially-varying material and e material at real-time frame rates.

Keywords augmented reality, neural netw

Citation Ma S J, Shen Q, Hou Q M, et al. N frequency lighting environments. Sci China Inf

1 Introduction

The availability of commercial AR (augme application developers to render virtual obje technologies estimate camera poses and loc making the rendered objects appear real is

A key challenge is the interaction betwee spatially-varying shadow and produce reflec are crucial to realism, yet difficult to simul and material information of the real scene

Learning Lightprobes for Mixed Reality Illumination

David Mandl[a] Kwang Moo Yi[b] Peter Mohr[c] Peter M. Roth[c] Pascal Fua[b] Vincent Lepetit[b,c] Dieter Schmalstieg[c] Denis Kalkofen[c]
[c]Graz University of Technology [b]EPFL [a]University of Bordeaux

Figure 1: Coherent Illumination. The real scene consists of an action figure of The Hulk and a toy car. To demonstrate the result of our system, we place a 3D scan next to the real action figure and display it using Mixed Reality: (left) The 3D reconstruction is rendered without real world light estimation. (right) Our system estimates the current lighting from a single input image. The estimated lighting is used to illuminate the 3D reconstruction. Note that we only register the real world lighting and do not consider any camera effects such as exposure or blur.

Abstract

This paper presents the first photometric registration pipeline for Mixed Reality based on high quality illumination estimation using convolutional neural networks (CNNs). For easy adaptation and deployment of the system, we train the CNNs using purely synthetic images and apply them to real image data. To keep the pipeline accurate and efficient, we propose to fuse the light estimation results from multiple CNN instances and show an approach for caching estimates over time. For optimal performance, we furthermore explore multiple strategies for the CNN training. Experimental results

proaches rely on active lightprobes, e.g., fisheye cameras strategically placed in the scene [14], or passive lightprobes, like chrome spheres, or other reference objects with known shape and reflectivity [4]. An obvious major downside of having a lightprobe is that it interferes with the user's experience. As an alternative, Gruber et al. [8] demonstrated a probe-less method that estimates the incident light purely from reflections in the unmodified scene, scanned in real-time with a commodity RGB-D sensor. This approach forgoes artificial lightprobes, but suffers from high computational demands and cannot be used on mobile devices [7].

We propose *learned lightprobes* as an alternative to the probeless method of Gruber et al. [8]. Like the probeless method, learned lightprobes work in an unmodified scene, providing unobtrusive user experience, while still retaining the computational benefits of

Distributed Lighting

Interactive Near-Field Illumination for Photorealistic Augmented Reality on Mobile Devices

Kai Rohmer* Computational Visualistics University of Magdeburg
Wolfgang Büschel† Interactive Media Lab Technical University of Dresden
Raimund Dachselt‡ Interactive Media Lab Technical University of Dresden
Thorsten Grosch§ Computational Visualistics University of Magdeburg

Abstract

Mobile devices become more and more important today, especially for augmented reality (AR) applications in which the camera of the mobile device acts like a window into the mixed reality world. Up to now, no photorealistic augmentation is possible since the computational power of the mobile devices is still too weak. Even a streaming solution from a stationary PC would cause a latency that affects user interactions considerably. Therefore, we introduce a differential illumination method that allows for a consistent illumination of the inserted virtual objects on mobile devices, avoiding a delay. The necessary computation effort is shared between a stationary PC and the mobile devices to make use of the capacities available on both sides. The method is designed such that only a minimum amount of data has to be transferred asynchronously between the stationary PC and one or multiple mobile devices. This allows for an interactive illumination of virtual objects with a consistent appearance under both temporally and spatially varying real illumination conditions. To describe the complex near-field illumination in an indoor scenario, multiple HDR video cameras are used to capture the illumination from multiple directions. In this way, sources of illumination can be considered that are not directly visible to the mobile device because of occlusions and the limited field of view of built-in cameras.

Index Terms: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism H.5.1 [Information Interfaces and Representation]: Artificial, Augmented and Virtual Realities

1 Introduction

Mobile devices like smartphones and tablet PCs are part of our everyday life. In combination with the integrated camera, the mobile device can act like a window into an augmented real world [11]. When virtual objects are inserted, their appearance is often inconsistent with the real environment, mainly because of wrong illumination. So far, a consistent illumination that handles both dynamic scenes and dynamic lighting conditions eluded the mobile platforms. Even though many sophisticated illumination methods allow for plausible global illumination at interactive rates on non-mobile platforms, several applications would benefit from such a mobile system. For instance, applications in interior planning and architecture can be enhanced by convincing in-place visualization of the designer's vision. Bringing back virtual versions of lost or destroyed artifacts would open up new possibilities in the field of cultural heritage. Mobile AR can also be used in mobile productions or in the process of creating a theater play by providing previews of the final scene and the created mood to the director at early stages.

Furthermore, it is relevant for mobile AR games, as users expect more and more enhanced graphics.

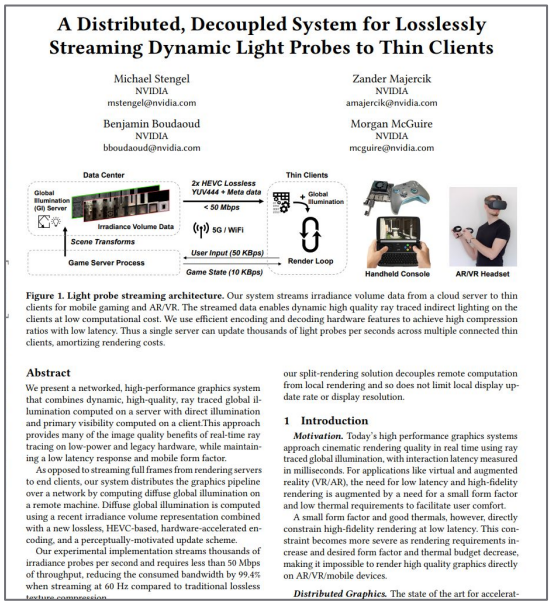
We primarily aim for applications like the augmentation of real prototypes, for which a correct illumination at any place in the scene is required and a perceptively plausible illumination is not sufficient.

To accomplish this, three problems need to be addressed:

- Mobile devices, such as tablets, do not yet have the computational resources necessary for computing interactive global illumination on their own.
- Streaming the rendered images from a powerful desktop PC causes a latency that is too high to meet the requirements for seamlessly integrated virtual objects, especially during user interactions and for multiple simultaneous views.
- Real lighting conditions in the dynamic scene need to be captured reliably and limitations caused by occlusions and the limited field of view have to be overcome.

In this paper, we present a novel, *distributed* illumination approach for AR with consistent illumination of virtual objects with direct light, indirect light (color bleeding) and shadows of primary and strong secondary lights. Due to the limitations in computational power of the mobile device and the lack of important information (see Figure 2, left), we split the illumination into two parts: In the first part, the existing radiance values are captured by a number of HDR video cameras that are placed at different locations in the scene. This acquisition process and the extraction of parameters for our lighting model is executed on a stationary PC. Based on the extracted information, we display augmentations with consistent illumination at interactive frame rate on the mobile device, as shown

*e-mail: kai.rohmer@isg.cs.ovgu.de
†e-mail: bueschel@acm.org
‡e-mail: dachselt@acm.org
§e-mail: grosch@isg.cs.ovgu.de



A Distributed, Decoupled System for Losslessly Streaming Dynamic Light Probes to Thin Clients

Michael Stengel NVIDIA mstengel@nvidia.com
Zander Majercik NVIDIA amajercik@nvidia.com
Benjamin Boudaoud NVIDIA bboudaoud@nvidia.com
Morgan McGuire NVIDIA mcguire@nvidia.com

Figure 1. Light probe streaming architecture. Our system streams irradiance volume data from a cloud server to thin clients for mobile gaming and AR/VR. The streamed data enables dynamic high quality ray traced indirect lighting on the clients at low computational cost. We use efficient encoding and decoding hardware features to achieve high compression ratios with low latency. Thus a single server can update thousands of light probes per seconds across multiple connected thin clients, amortizing rendering costs.

Abstract

We present a networked, high-performance graphics system that combines dynamic, high-quality, ray traced global illumination computed on a server with direct illumination and primary visibility computed on a client.This approach provides many of the image quality benefits of real-time ray tracing on low-power and legacy hardware, while maintaining a low latency response and mobile form factor.

As opposed to streaming full frames from rendering servers to end clients, our system distributes the graphics pipeline over a network by computing diffuse global illumination on a remote machine. Diffuse global illumination is computed using a recent irradiance volume representation combined with a new lossless, HEVC-based, hardware-accelerated encoding, and a perceptually-motivated update scheme.

Our experimental implementation streams thousands of irradiance probes per second and requires less than 50 Mbps of throughput, reducing the consumed bandwidth by 99.4% when streaming at 60 Hz compared to traditional lossless texture compression.

our split-rendering solution decouples remote computation from local rendering and so does not limit local display update rate or display resolution.

1 Introduction

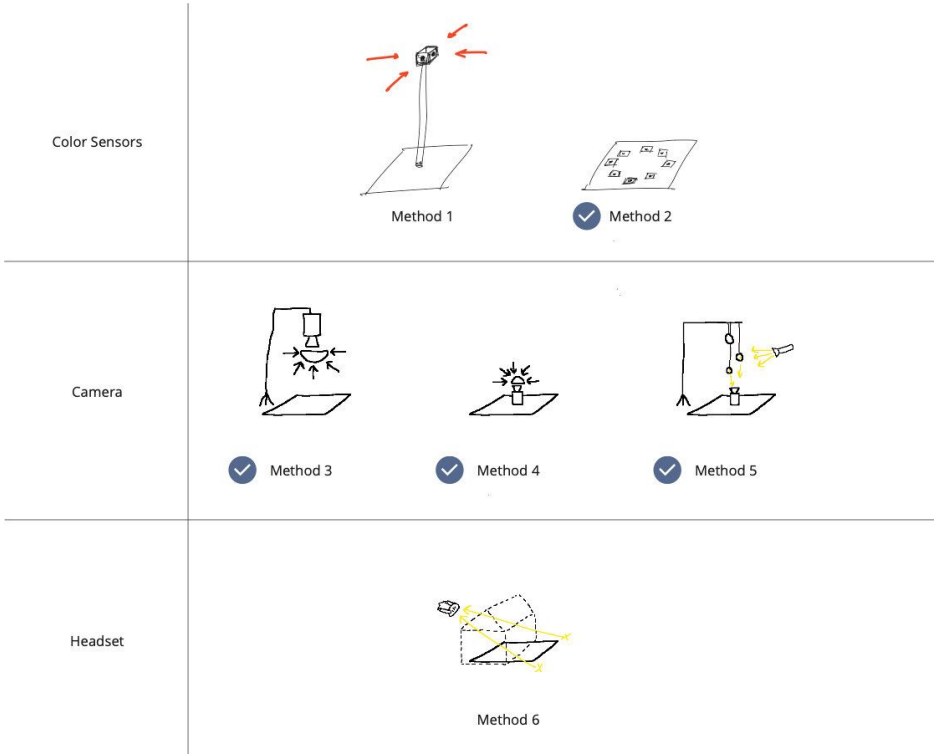
Motivation. Today's high performance graphics systems approach cinematic rendering quality in real time using ray traced global illumination, with interaction latency measured in milliseconds. For applications like virtual and augmented reality (VR/AR), the need for low latency and high-fidelity rendering is augmented by a need for a small form factor and low thermal requirements to facilitate user comfort.

A small form factor and good thermals, however, directly constrain high-fidelity rendering at low latency. This constraint becomes more severe as rendering requirements increase and desired form factor and thermal budget decrease, making it impossible to render high quality graphics directly on AR/VR/mobile devices.

Distributed Graphics. The state of the art for accelerat-

Initial Ideas

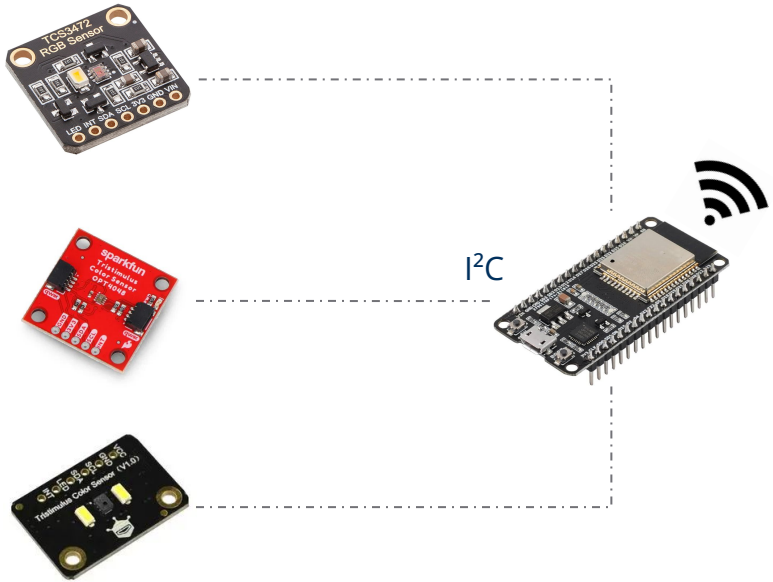
- **Come up with different methods of capturing lighting information**
- **Implement and compare some methods**
  - Feasibility
  - Visual Appearance
  - Cost
  - Complexity
- **→ Focusing on small scale / local methods for single objects**



Color Sensor Methods

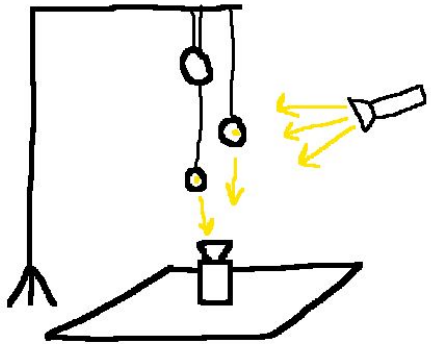
Hardware:

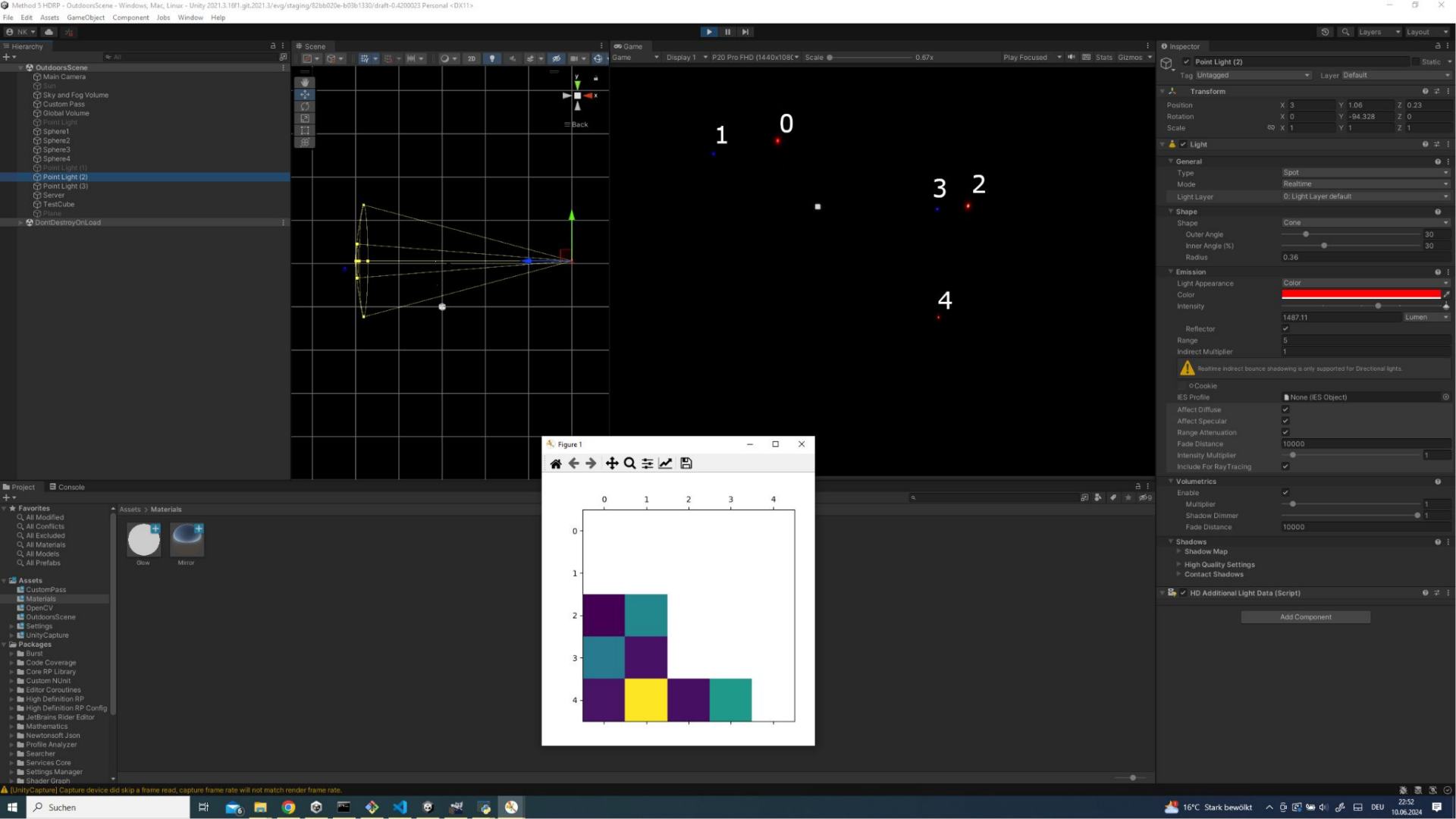
- the TCS34725:
  - RGBW sensor
  - I²C communication
  - cheap (~5€ per sensor, ~2€ from AliExpress)
- the OPT4048DTSR:
  - XYZ sensor
    -> more accurate color sensing + light intensity in lux + color temperature
  - I²C communication
  - more expensive (~10€ per sensor)
- the SEN0403:
  - XYZ sensor (like OPT4048DTSR)
  - I²C communication
  - more expensive (~13€ per sensor)

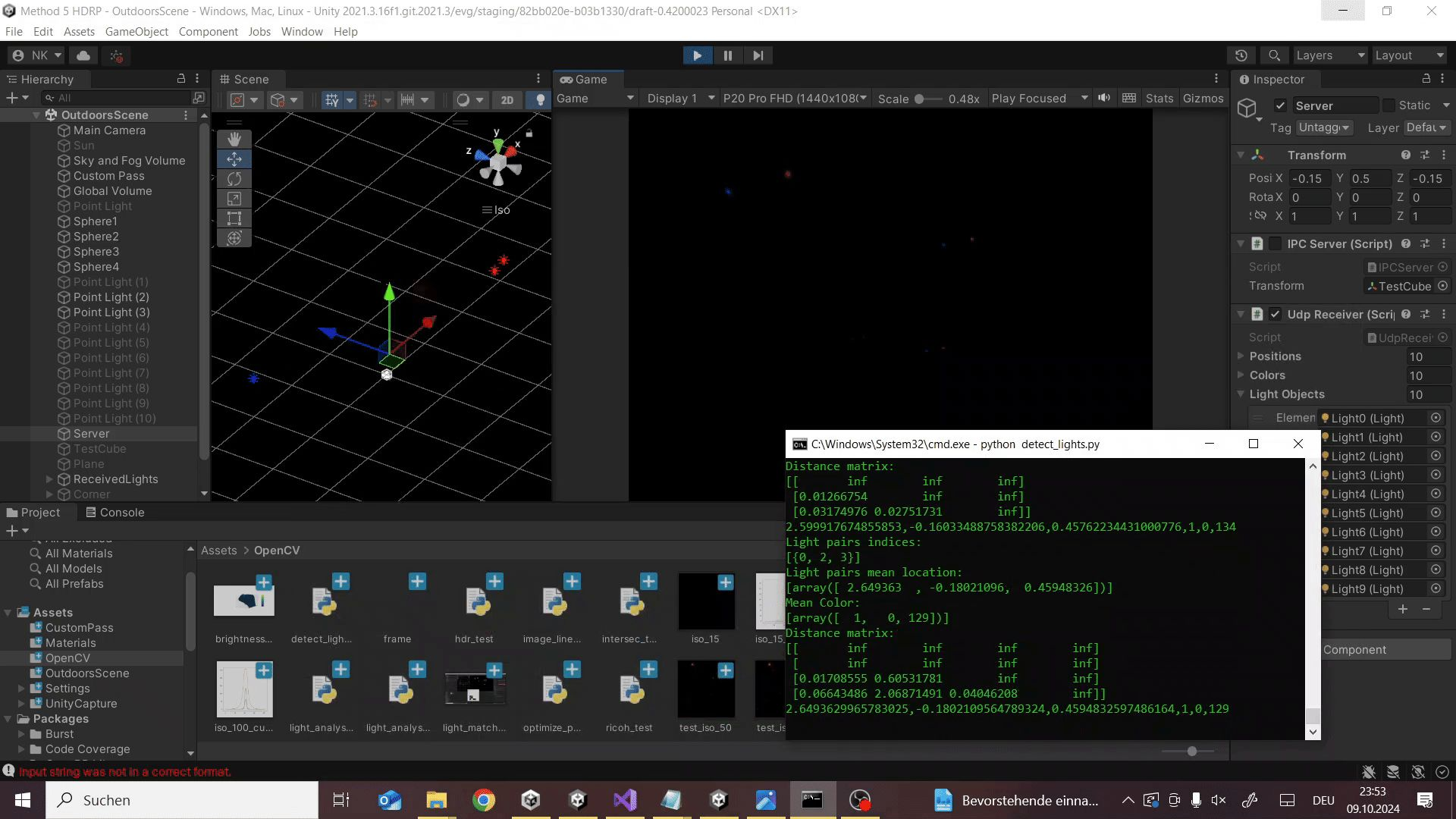


Single Camera, multiple Spheres

- **Calculate 3D position of light source**
- **Create virtual light with same properties at same position**
- **Affordable + Precise**
  - Standard camera / smartphone
  - Some mirror spheres (e.g. christmas ornaments)



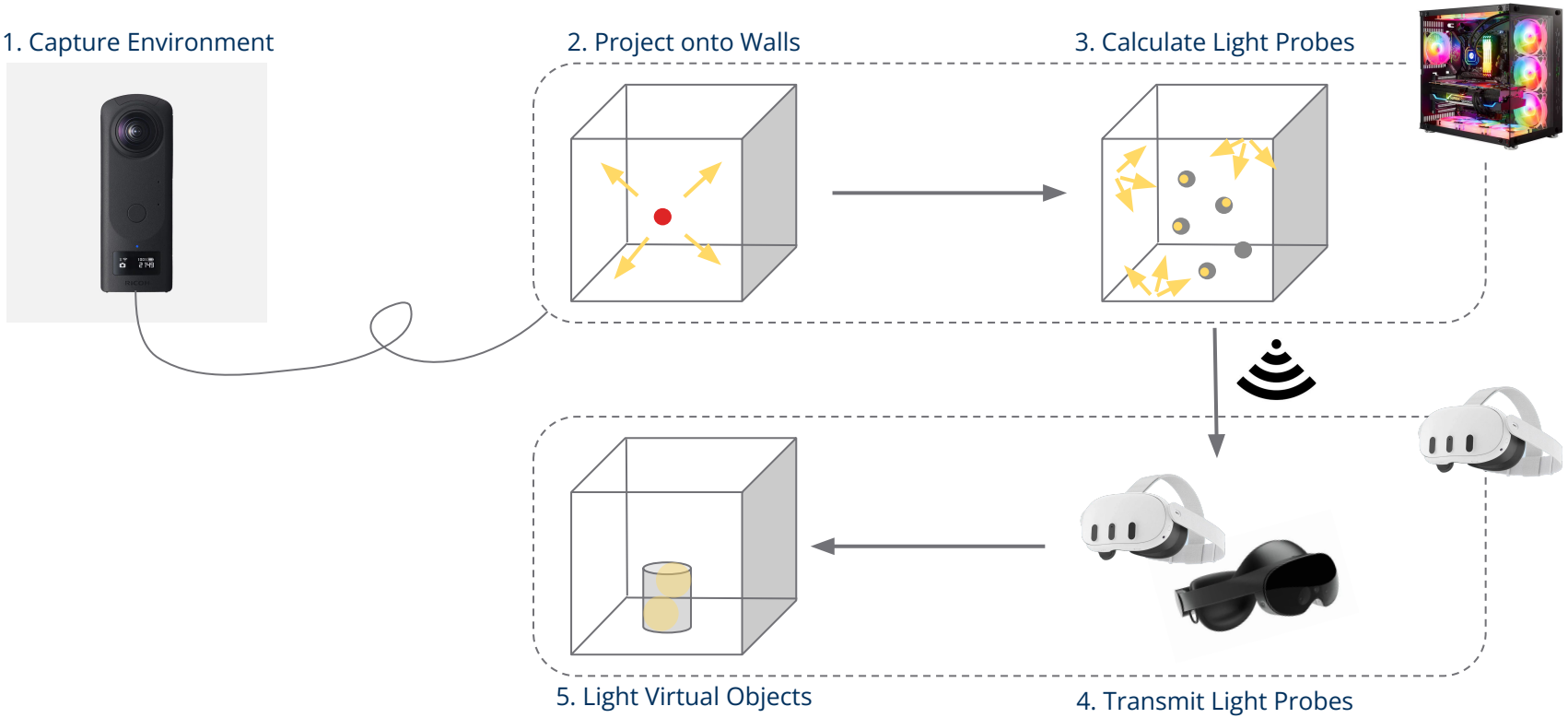

Method 5 HDRP - OutdoorsScene - Windows, Mac, Linux - Unity 2021.3.16f1.git.2021.3/evg/staging/82bb020e-b03b1330/draft-0.4200023 Personal <DX11>
File Edit Assets GameObject Component Jobs Window Help
Hierarchy
OutdoorsScene
Main Camera
Sky
Sky and Fog Volume
Custom Pass
Global Volume
Point Light
Sphere1
Sphere2
Sphere3
Sphere4
Point Light (1)
Point Light (2)
Point Light (3)
Server
TestCube
Plane
DontDestroyOnLoad
Scene
Back
Game
Display 1
Play Focused
Stats
Gizmos
0
1
2
3
4
Inspector
Point Light (2)
Static
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Light
General
Type Spot
Mode Realtime
Light Layer 0: Light Layer default
Shape
Shape Cone
Outer Angle
Inner Angle (%)
Radius 0.36
Emission
Light Appearance Color
Color
Intensity
1487.11 Lumen
Reflector
Range 5
Indirect Multiplier 1
Realtime indirect bounce shadowing is only supported for Directional lights.
Cookie
IES Profile None (IES Object)
Affect Diffuse
Affect Specular
Range Attenuation
Fade Distance 10000
Intensity Multiplier 1
Include For RayTracing
Volumetrics
Enable
Multiplier 1
Shadow Dimmer 1
Fade Distance 10000
Shadows
Shadow Map
High Quality Settings
Contact Shadows
HD Additional Light Data (Script)
Add Component
Figure 1
Project
Console
Favorites
All Modified
All Conflicts
All Excluded
All Materials
All Models
All Prefabs
Assets
CustomPass
Materials
OpenCV
OutdoorsScene
Settings
UnityCapture
Packages
Burst
Code Coverage
Core RP Library
Custom NUnit
Editor Coroutines
High Definition RP
High Definition RP Config
JetBrains Rider Editor
Mathematics
Newtonsoft Json
Profile Analyzer
Searcher
Services Core
Settings Manager
Shader Graph
Assets > Materials
Glow
Mirror
[UnityCapture] Capture device did skip a frame read, capture frame rate will not match render frame rate.
Suchen
16°C Stark bewölkt
DEU
22:52
10.06.2024


Method 5 HDRP - OutdoorsScene - Windows, Mac, Linux - Unity 2021.3.16f1.git.2021.3/evg/staging/82bb020e-b03b1330/draft-0.4200023 Personal <DX11>
File Edit Assets GameObject Component Jobs Window Help
Hierarchy
OutdoorsScene
Main Camera
Sun
Sky and Fog Volume
Custom Pass
Global Volume
Point Light
Sphere1
Sphere2
Sphere3
Sphere4
Point Light (1)
Point Light (2)
Point Light (3)
Point Light (4)
Point Light (5)
Point Light (6)
Point Light (7)
Point Light (8)
Point Light (9)
Point Light (10)
Server
TestCube
Plane
ReceivedLights
Corner
Scene
Game
Display 1
P20 Pro FHD (1440x108(
Scale 0.48x
Play Focused
Stats Gizmos
Inspector
Server
Static
Tag Untagg
Layer Defau
Transform
Posi X -0.15 Y 0.5 Z -0.15
Rota X 0 Y 0 Z 0
X 1 Y 1 Z 1
IPC Server (Script)
Script IPCServer
Transform TestCube
Udp Receiver (Scri
Script UdpRecei
Positions 10
Colors 10
Light Objects 10
Element Light0 (Light)
Light1 (Light)
Light2 (Light)
Light3 (Light)
Light4 (Light)
Light5 (Light)
Light6 (Light)
Light7 (Light)
Light8 (Light)
Light9 (Light)
Component
Project
Console
All Materials
All Models
All Prefabs
Assets
CustomPass
Materials
OpenCV
OutdoorsScene
Settings
UnityCapture
Packages
Burst
Code Coverage
Assets > OpenCV
brightness...
detect_ligh...
frame
hdr_test
image_line...
intersec_t...
iso_15
iso_15...
iso_100_cu...
light_analys...
light_analys...
light_match...
optimize_p...
ricoh_test
test_iso_50
test_is
C:\Windows\System32\cmd.exe - python detect_lights.py
Distance matrix:
[[inf inf inf]
[0.01266754 inf inf]
[0.03174976 0.02751731 inf]]
2.599917674855853,-0.16033488758382206,0.45762234431000776,1,0,134
Light pairs indices:
[{0, 2, 3}]
Light pairs mean location:
[array([2.649363 , -0.18021096, 0.45948326])]
Mean Color:
[array([1, 0, 129])]
Distance matrix:
[[inf inf inf inf]
[inf inf inf inf]
[0.01708555 0.60531781 inf inf]
[0.06643486 2.06871491 0.04046208 inf]]
2.6493629965783025,-0.1802109564789324,0.4594832597486164,1,0,129
Input string was not in a correct format.
Suchen
Bevorstehende einna...
DEU
23:53
09.10.2024

Improved Idea

- Use 360° camera instead of fish eye lens
- Use model of the room to project image onto
- Use emissive material to cast light into the scene
- Use light probes to light virtual objects

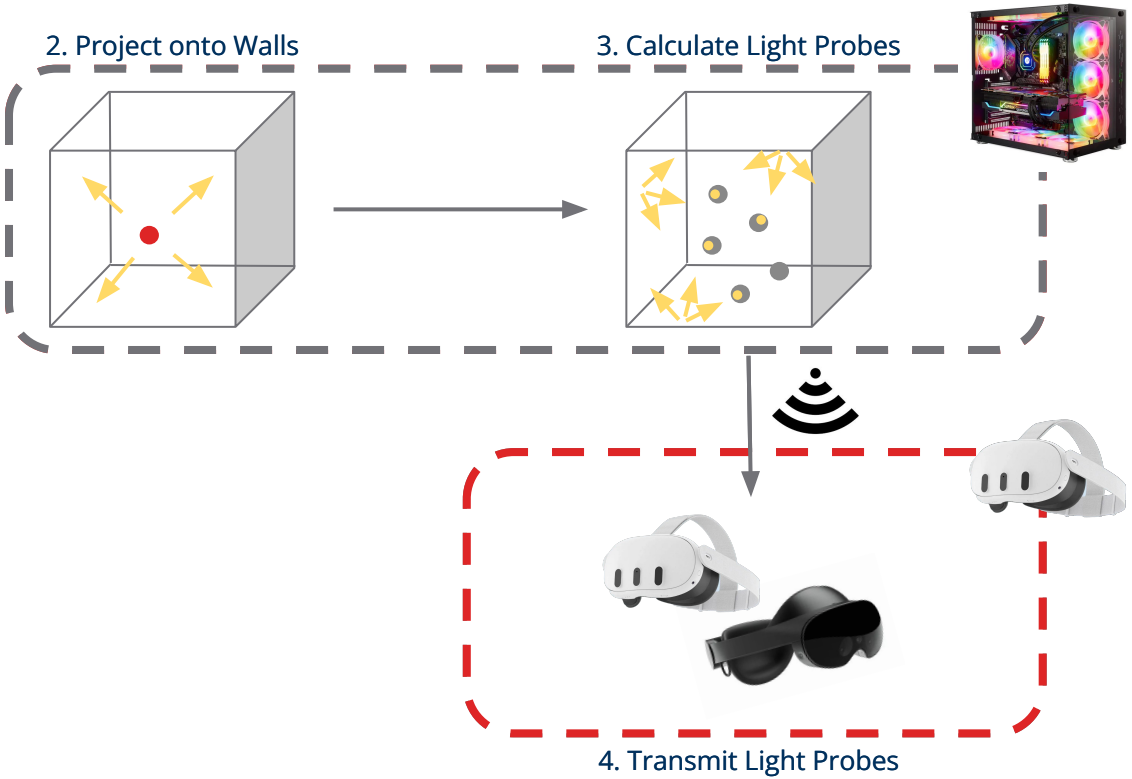
System Architecture



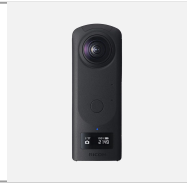
Key Features

1. Model based Lighting using emissive Materials

2. Streaming of Light Probes to multiple mobile Clients



1. Capture the Environment

	Ricoh Theta Z1 	**Vuze XR** 
Lenses	2x 180° x 180°	2x 180° x 180°
Video Resolution	Up to 4K @ 30FPS	Up to 5.7K @ 30FPS
HDR support	For photos	None
Live Streaming	**Over USB** or per App	Per App
Software Availability	Over their Website	**Discontinued**

1. Capture the Environment

Ricoh Theta Z1

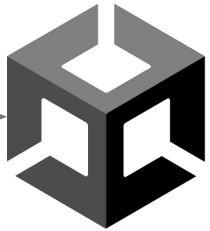
Captures Video

Ricoh UVC Driver

Enables
Live Streaming

OBS

Virtual Webcam to
convert Image format



Unity Project

Webcam to
RenderTexture

1. Capture the Environment

```
public class WebcamToRenderTexture : MonoBehaviour
{
    public RenderTexture outputTexture;
    public Texture image;

    public string cameraDeviceName = "OBS";
    private WebCamTexture webCamTexture;

    public bool enableWebcam = false;
    public bool enableImage = false;

    void Start()
    {
        if (!enableWebcam) return;

        WebCamDevice[] devices = WebCamTexture.devices;
        WebCamDevice device = devices[0];

        // Prints available webcams to the console
        for (int i = 0; i < devices.Length; i++)
        {
            print("Webcam available: " + devices[i].name);
            if (devices[i].name.Contains(cameraDeviceName))
            {
                device = devices[i];
            }
        }

        webCamTexture = new WebCamTexture(device.name);
        webCamTexture.Play();
    }

    void Update()
    {
        if (enableWebcam)
        {
            Graphics.Blit(webCamTexture, outputTexture);
        }
        if (enableImage)
        {
            Graphics.Blit(image, outputTexture);
        }

```

- **Selectable webcam device**
- **Use static images for testing**

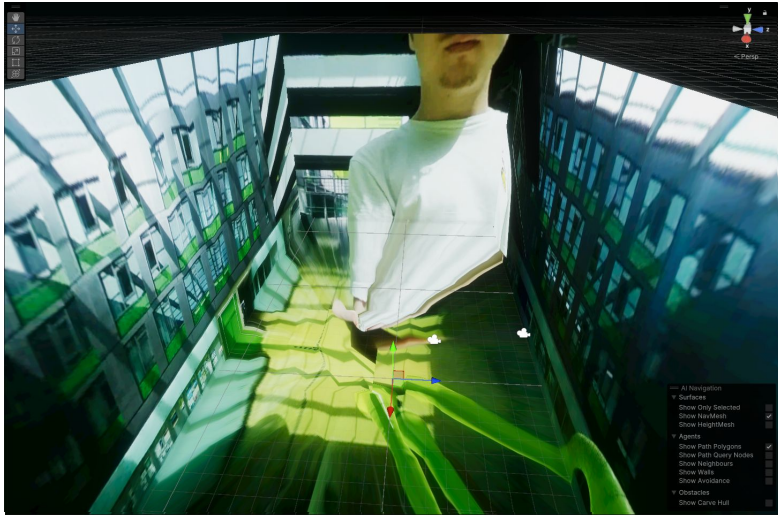
2. Projection onto Walls


```
64
65  void CreateUvMap(GameObject gameObject)
66  {
67      if (gameObject.GetComponent<MeshFilter>() == null) return;
68
69      Mesh mesh = gameObject.GetComponent<MeshFilter>().mesh;
70
71      // merge duplicate vertices from previous mappings
72      NormalizeMesh(mesh);
73
74      // calculate the UVs without post-correction
75      CalculateUVs(mesh, gameObject);
76
77      // correct the edges that span over the edge of the UV map
78      // using a vertex duplication strategy
79      FixUVSeam(mesh, gameObject);
80  }
81
82  // Calculate the UV coordinates of every vertex depending on its world position
83  // and the center of projection
84  public void CalculateUVs(Mesh mesh, GameObject gameObject)
85  {
86      Vector3[] vertices = mesh.vertices;
87      Vector2[] uvs = new Vector2[vertices.Length];
88
89      for (int i = 0; i < uvs.Length; i++)
90      {
91          Vector3 worldPosition = gameObject.transform.TransformPoint(vertices[i]);
92          Vector3 sphereCenter = ProjectionCenter.transform.position;
93
94          Vector2 uv = GetSphereUV(worldPosition, sphereCenter);
95          uv = new Vector2(uv.x, 1.0f - uv.y);
96          uvs[i] = uv;
97      }
98
99      mesh.uv = uvs;
100 }
101
102 public Vector2 GetSphereUV(Vector3 point, Vector3 sphereCenter)
103 {
104     Vector3 normalizedPoint = (point - sphereCenter).normalized;
105     normalizedPoint = Quaternion.Euler(rotationOffset) * normalizedPoint;
106
107     float u = 1.0f - (0.5f + Mathf.Atan2(normalizedPoint.z, normalizedPoint.x) / (2.0f * Mathf.PI) + xOffset);
108     float v = 0.5f - Mathf.Asin(normalizedPoint.y) / Mathf.PI + yOffset;
109     return new Vector2(u, v);
110 }
111
```

Problems

Interpolation Error

UV Seam



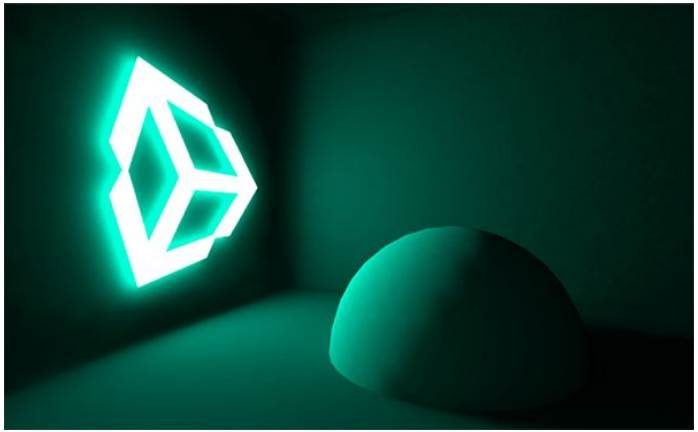
3. Calculating Light Probes

Realtime Global Illumination

- Light that changes at runtime
  (not baked)

- Direct light
  (Light Source → Object → Sensor)
- Indirect Light
  (multiple bounces; different light sources like emissive materials)

- Using Enlighten middleware



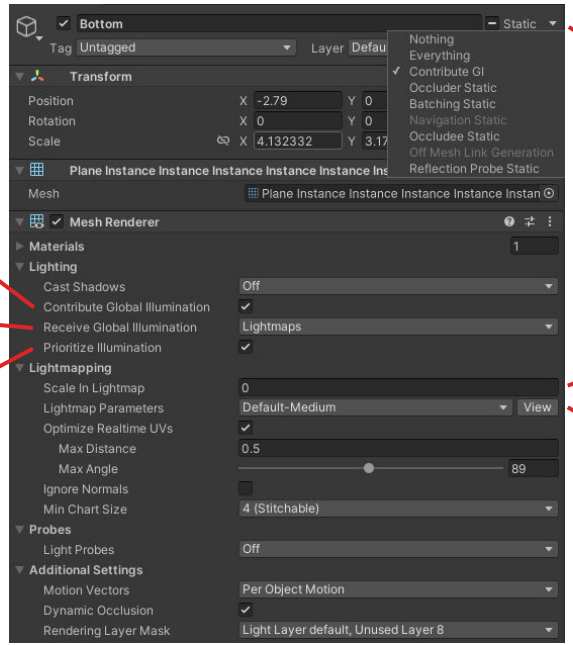
3. Calculating Light Probes using Realtime Global Illumination

Settings of the environment model

Mesh is included in GI calculations

Emission lightmap is needed to emit light

Mesh will always be included in GI calculations



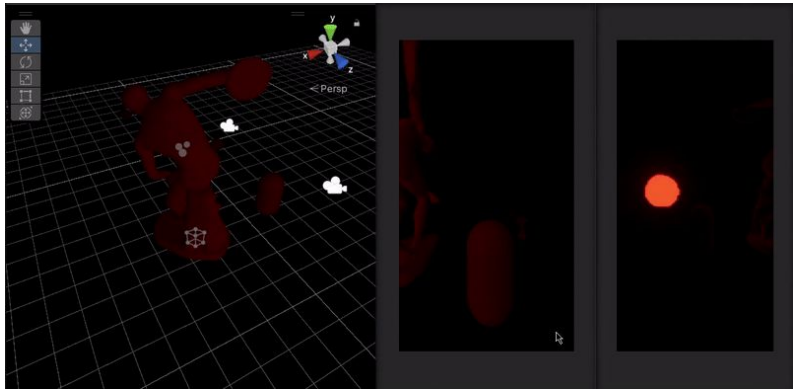
Static objects would receive lighting from other emissive objects

0 means not lightmapped but still contributes to other renderers

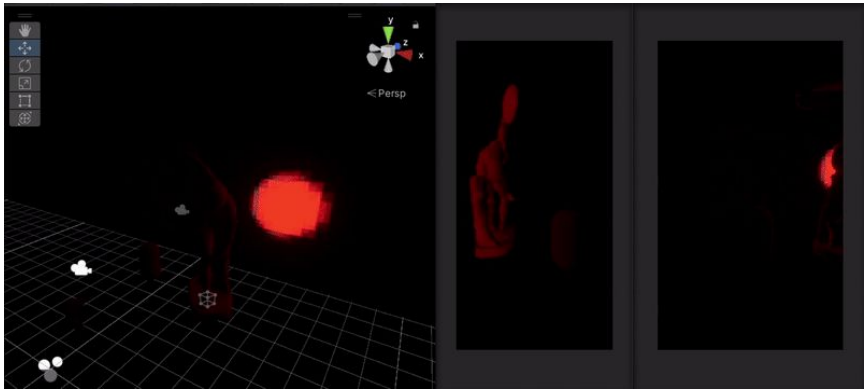
Default-Medium for better quality
Default-VeryLowResolution for higher FPS

Lightmap Parameters

Default-VeryLowResolution



Default-Medium



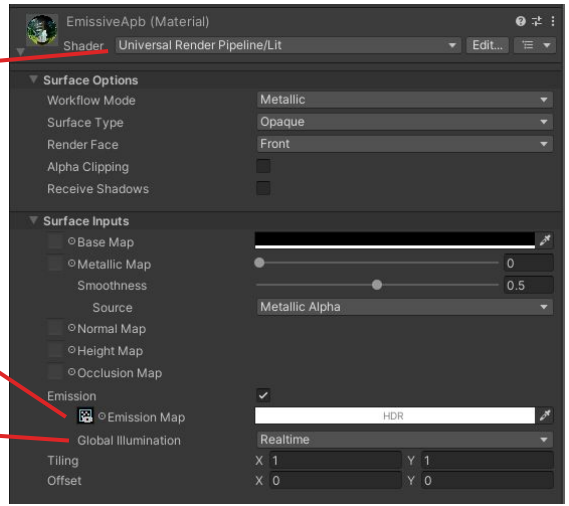
3. Calculating Light Probes using Realtime Global Illumination

Settings of the emissive material

Default shader supports emissive settings

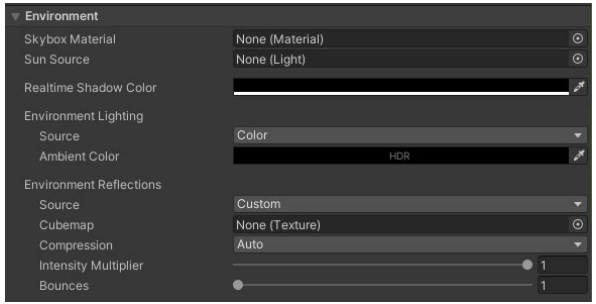
EmissionMap is set to the webcam RenderTexture

Update GI at runtime



3. Calculating Light Probes using Realtime Global Illumination

Lighting Settings



Trigger GI Update

```
60
61      void UpdateEmissiveMaterial(GameObject go)
62      {
63          Renderer renderer = go.GetComponent<Renderer>();
64          if (renderer == null) return;
65
66          RendererExtensions.UpdateGIMaterials(renderer);
67      }
68  }
69
```

Light Probes need to be baked into the scene with ‘Generate Lighting’ button

4. Transmit Light Probes

Transport Protocol

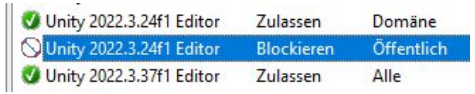
TCP	UDP
Reliable transmission	Can be unreliable
Overhead through ACKs	No Overhead
No Broadcast	Broadcast to multiple devices

perfect for realtime applications!

4. Transmit Light Probes

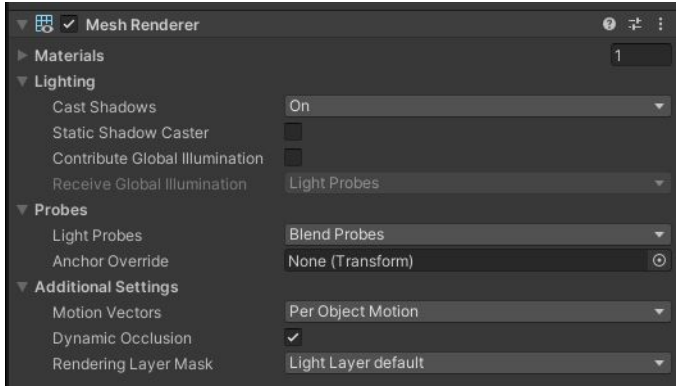
```
11
12      private UdpClient udpClient = new UdpClient();
13      public string udpIPAddress = "192.168.178.3"; // The IP address to send the data to
14      public int udpPort = 7778; // The port to send the data to
15
16      void Start()
17      {
18          udpClient.EnableBroadcast = true;
19      }
20
21      void Update()
22      {
23          SendLightProbeData(udpClient);
24      }
25
26      void SendLightProbeData(UdpClient udpClient)
27      {
28          Vector3[] probePositions = LightmapSettings.lightProbes.positions;
29          int probeCount = probePositions.Length;
30
31          // Serialize the data to a byte array
32          for (int i = 0; i < probeCount; i++)
33          {
34              SphericalHarmonicsL2 shCoefficient;
35              LightProbes.GetInterpolatedProbe(probePositions[i], null, out shCoefficient);
36              byte[] data = SerializeLightProbeData(i, shCoefficient);
37
38              // Send the data over UDP
39              udpClient.Send(data, data.Length, udpIPAddress, udpPort);
40          }
41      }
42
```

Disable Firewall for Unity!



5. Light virtual Objects

```
125
126     private void ReceiveData()
127     {
128         ReceiveClient = new UdpClient(udpPort);
129         ReceiveClient.EnableBroadcast = true;
130
131         while (true)
132         {
133             try
134             {
135                 IPEndPoint anyIP = new IPEndPoint(IPAddress.Any, 0);
136                 byte[] data = ReceiveClient.Receive(ref anyIP);
137
138                 DeserializeLightProbeData(data);
139             }
140             catch (Exception err)
141             {
142                 Debug.Log("<color=red>" + err.Message + "</color>");
143             }
144         }
145     }
146
147     void UpdateLightProbeSHCoefficients(SphericalHarmonicsL2[] shCoefficients)
148     {
149         LightProbes lightProbes = LightmapSettings.lightProbes;
150         SphericalHarmonicsL2[] existingBakedProbes = lightProbes.bakedProbes;
151
152         for (int i = 0; i < existingBakedProbes.Length; i++)
153         {
154             existingBakedProbes[i] = shCoefficients[i] * (1.0f / divider);
155         }
156
157         // Update the light probes in LightmapSettings
158         LightmapSettings.lightProbes.bakedProbes = existingBakedProbes;
159     }
160
```



Further Challenges

Consistent Camera Positioning

1. Place the Camera in the room
2. Guess the position in the virtual model
3. Project camera stream onto virtual model
4. Repeat from 2. until properly aligned

4 DoF
→ cumbersome process

→ take advantage of ViewR's calibration

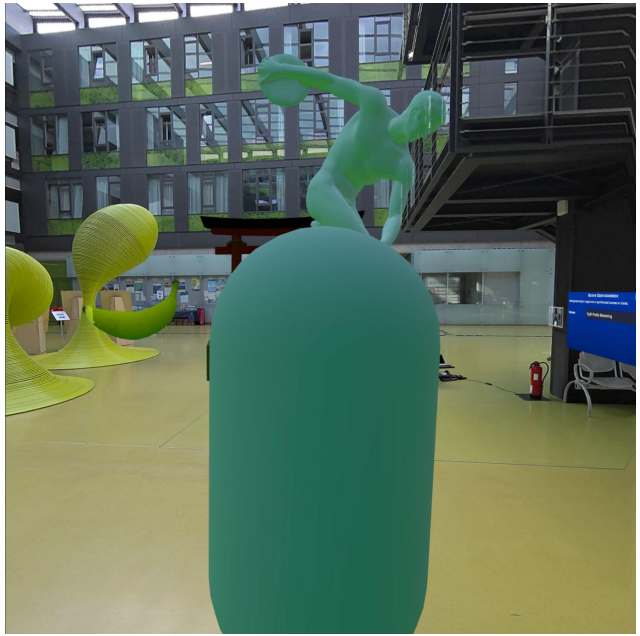
1. Place a small virtual indicator at the position of the camera
2. Place the camera at the position of the indicator using passthrough

1 DoF

Camera Calibration

Further Challenges

Blue tint on the HMD



Blue tint on HMD

Initial guess was:

- **Cold color temperature of the camera**
- **Too much influence of the blue sky**

(apparently) no tint when not lit from above

Effect only visible on computer,
On HMD still blue

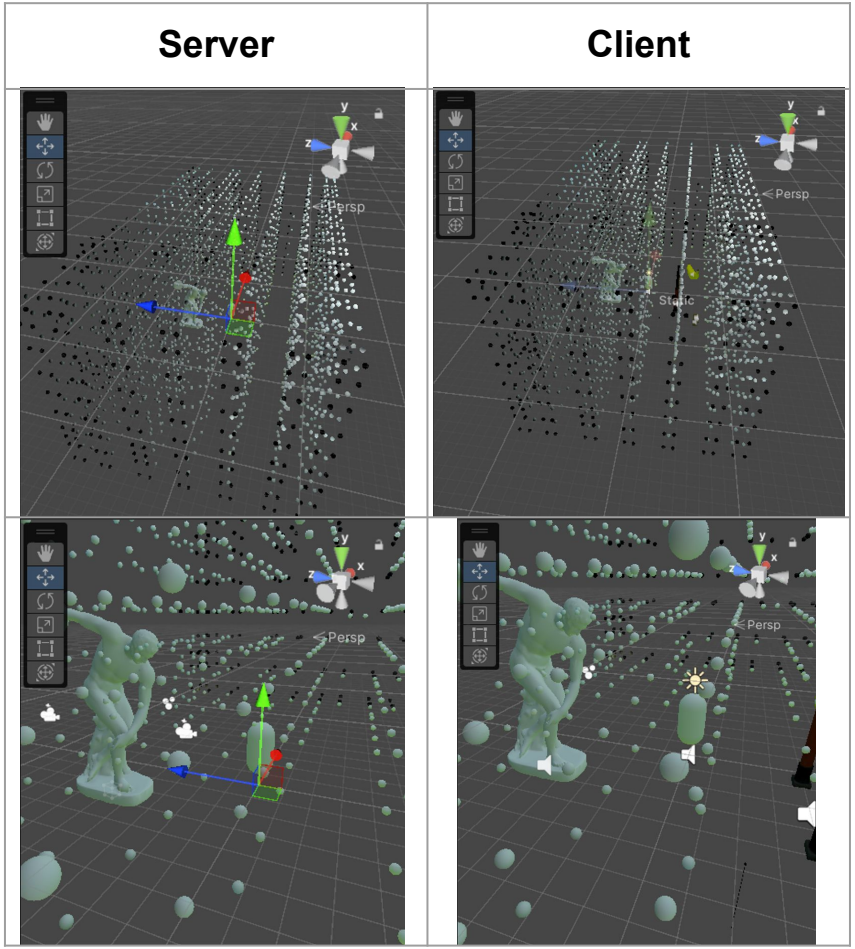
Blue tint on HMD

Next guess:

- **Error in transmission**
- **Color channels swapped**
- **Wrong Byte order**

→ Diff checker on all coefficients shows that transmission works

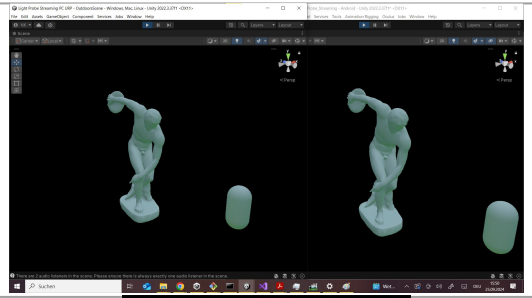
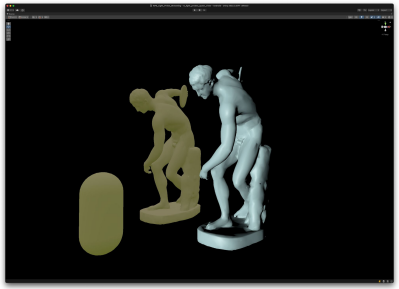
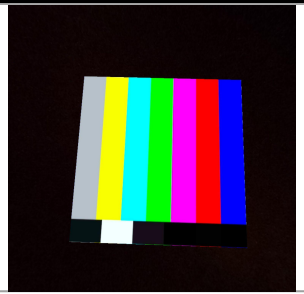
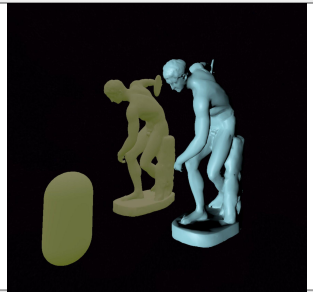
only 14 of 2359 light probes (0.6%) have discrepancies apart from rounding errors



Blue tint on HMD

Next guess:

- **Display of the HMD**
- **Color Settings**

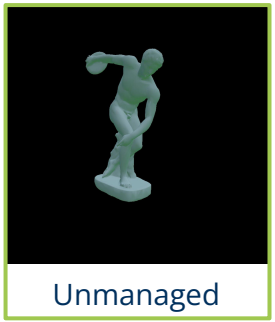
PC			
HMD			

Blue tint on HMD

- **Oculus Color Space Setting!**

PC Reference

ViewR Preset



Unmanaged

Rec2020

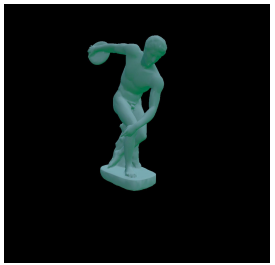
Rec709

Rift CV1

Rift S

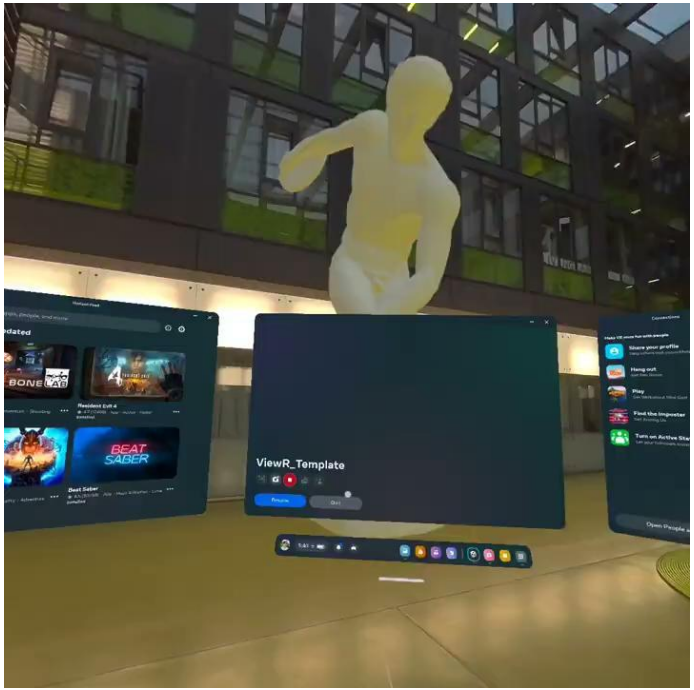
Quest 1

DCI P3

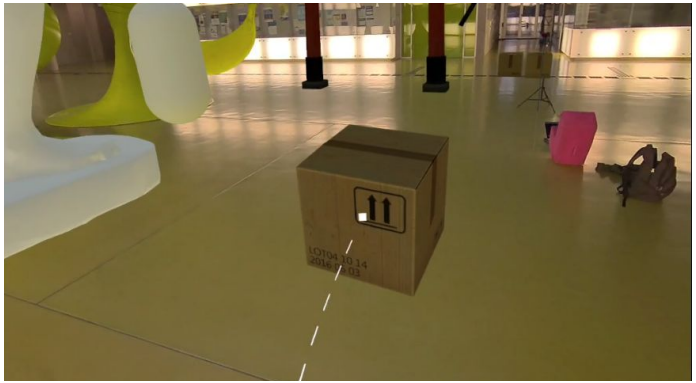


Adobe RGB

Results

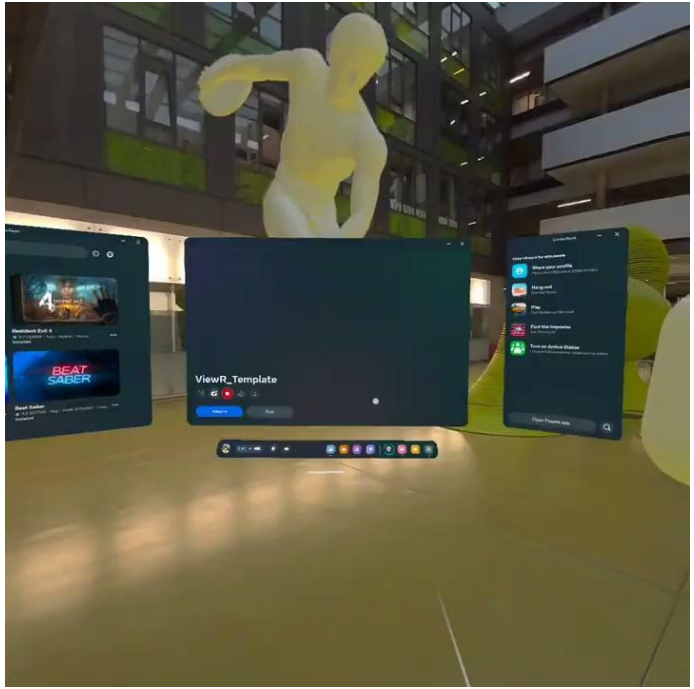


Results: Spatial Differences



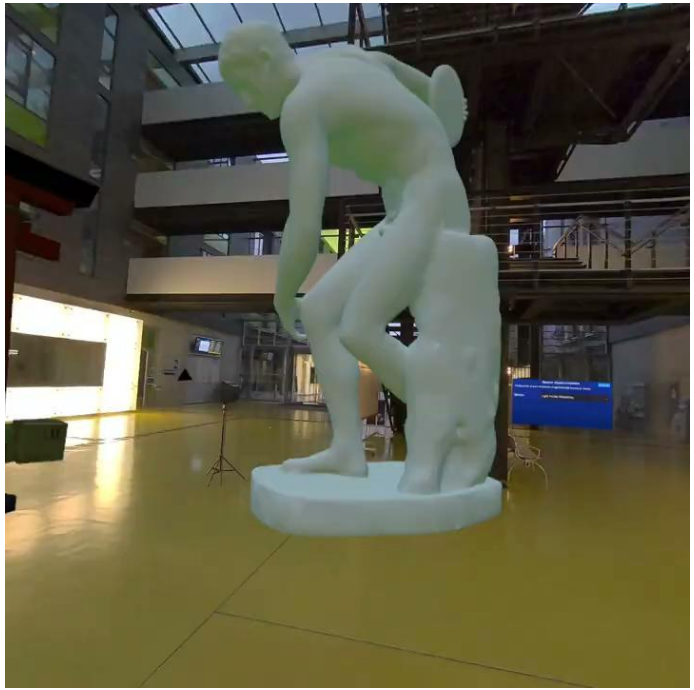
Results: Spatial Differences

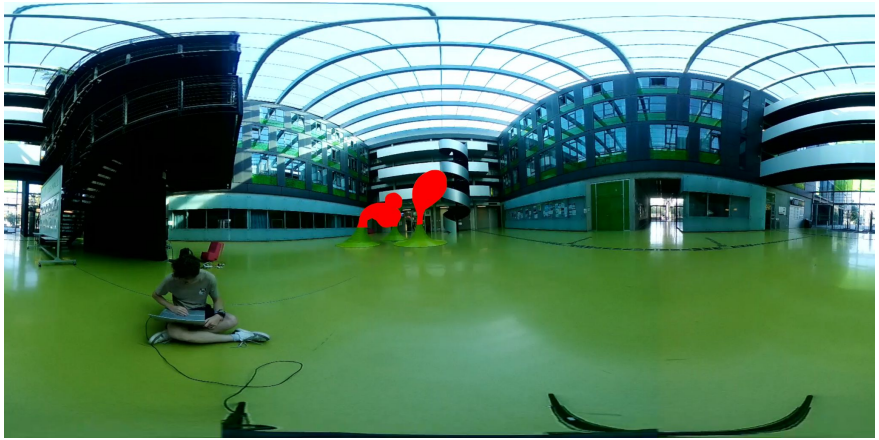
Results: Camera Occlusion



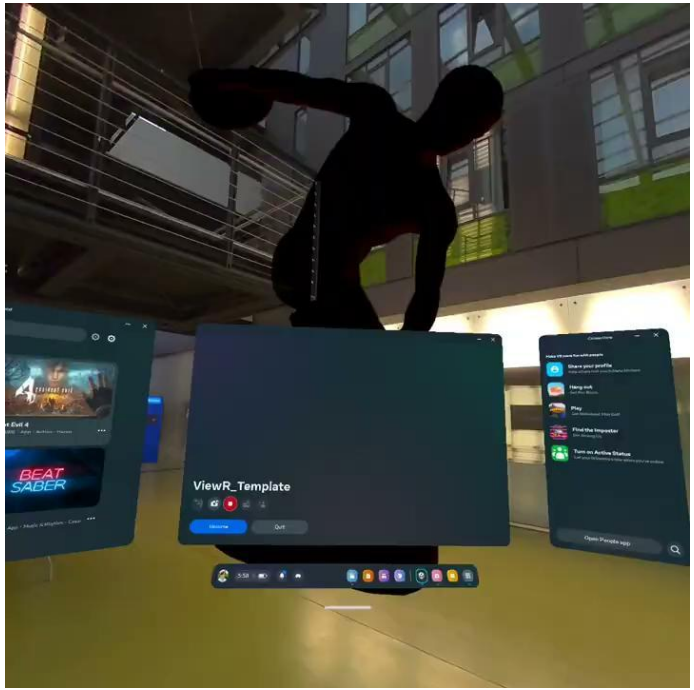
Results: Artificial Sunset

Results: Artificial Red Blobs



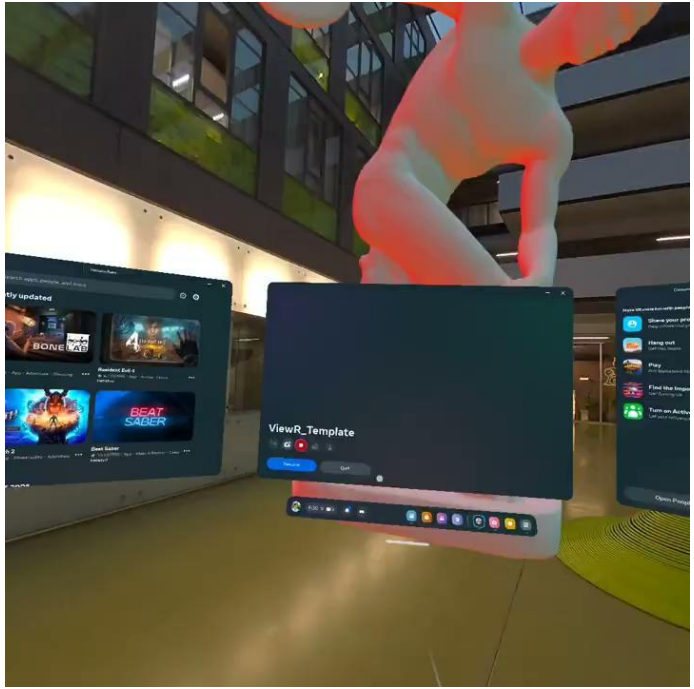


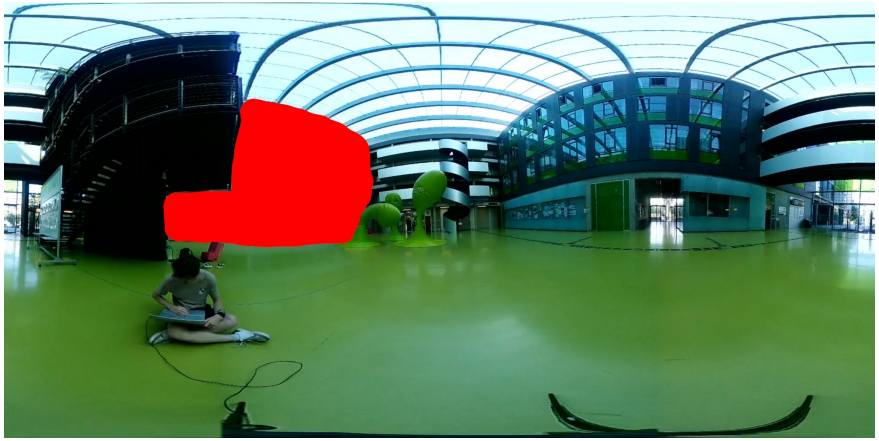
Results Artificial Red Dot

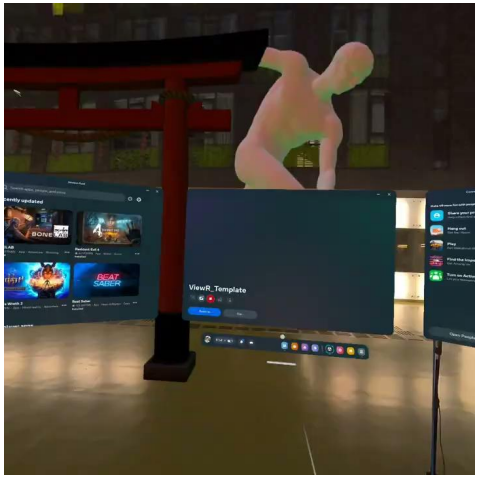


Results: Skybox comparison

Model based







Skybox

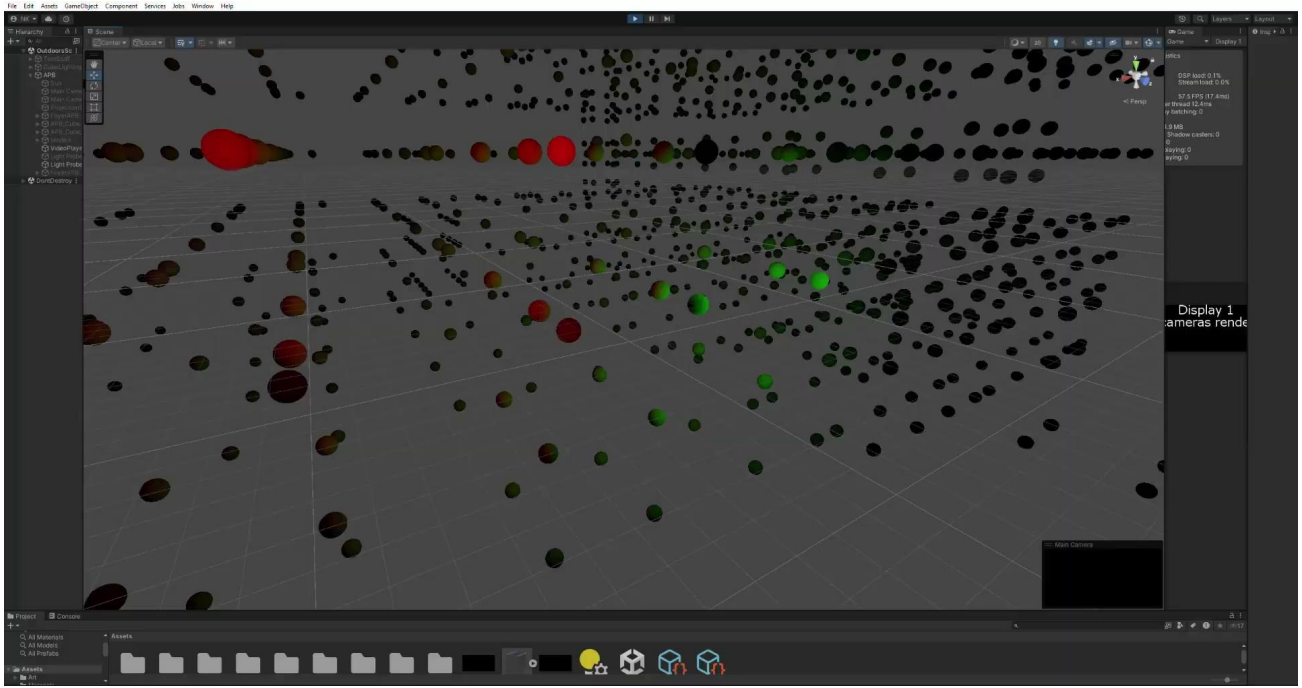
Performance

Laptop	Desktop
i5-10210U 4 Cores / 8 Threads @ 1.6GHz - 4.2GHz GeForce MX250	i7-13700K 8 P-Cores, 8 E-Cores / 16 Threads @ 2.5GHz - 5.4GHz GeForce GTX 1070
~ 10 FPS	~ 30 - 50 FPS
~ 2s delay (1s camera, 1s lighting calculation)	-

How Enlighten Realtime Global Illumination works

Enlighten Realtime Global Illumination splits the Scene into small surface patches and determines the degree to which these patches are visible to each other. At runtime, Enlighten Realtime Global Illumination uses this precomputed visibility information to approximate how Realtime Lights bounce in the Scene, saves the results in a set of lightmaps, and then uses these lightmaps to apply indirect lighting to the Scene. It is computationally intensive to update the lightmaps, and so the process is split across several frames. It takes Enlighten Realtime Global Illumination several frames to propagate changes to indirect lighting throughout the Scene.

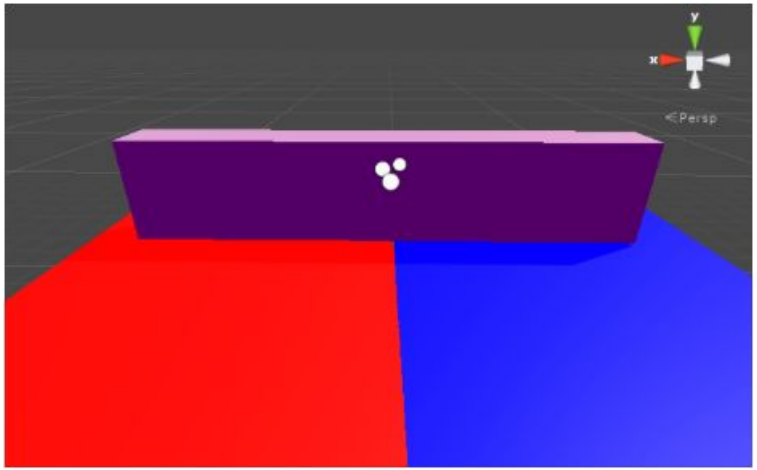
Performance: Desktop PC



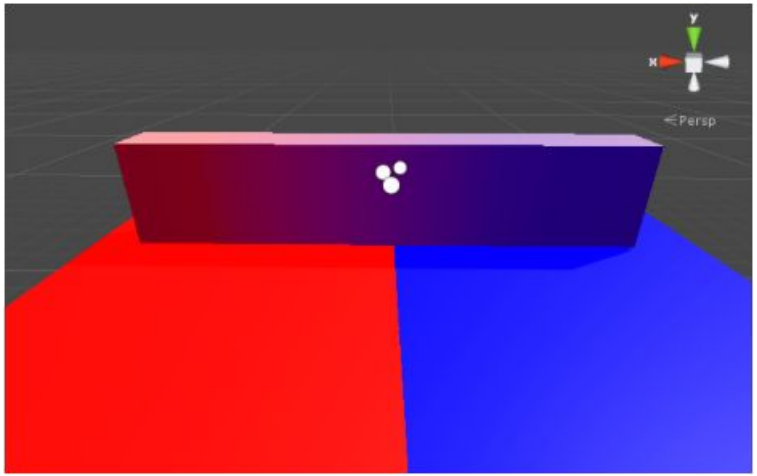
Further Considerations

Light Probe Proxy Volume

not supported in URP



without LPPV



with LPPV

Further Considerations

Adaptive Probe Volumes

only available in version 6 of Unity (preview)

without APV

with APV

Further Considerations

A proper shader based projection & emission

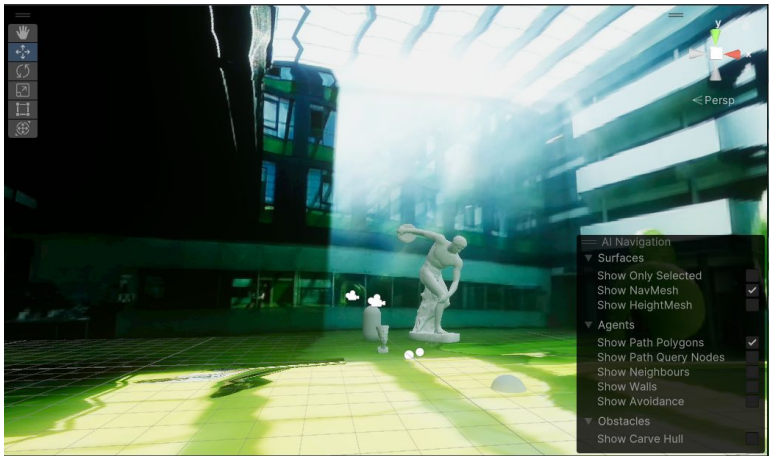
- → less interpolation error
- → more flexible regarding repositioning of the camera

Sync interactable objects with rendering scene

- → virtual objects can interact with lighting

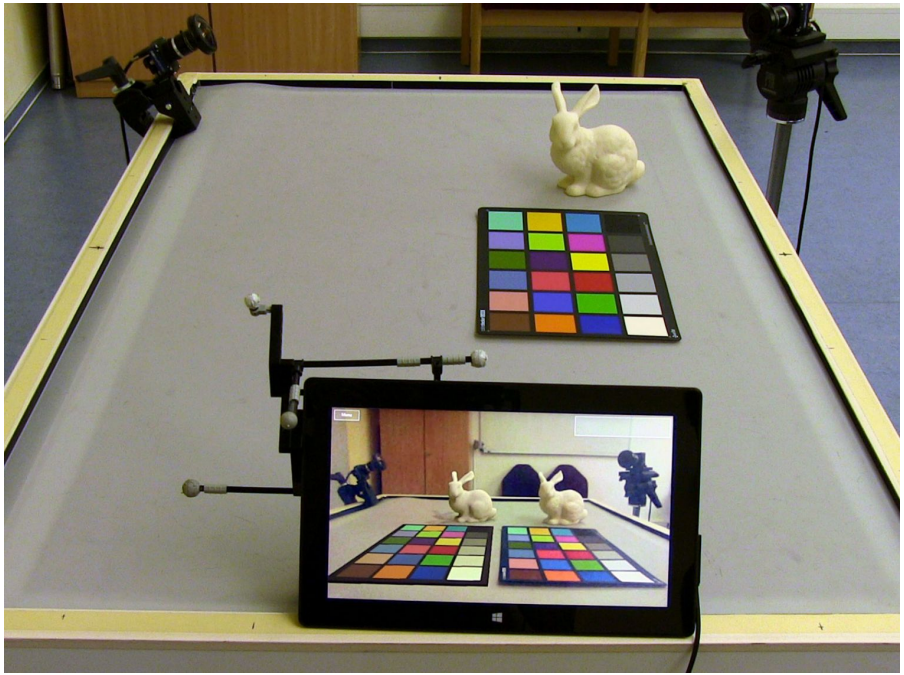
Place camera in the air + multiple cameras + HDR

- → reduce visual artifacts caused by humans
- → deal with obstruction
- → improve visual quality



Further Considerations

Compare to real objects



Conclusion

How can we get lighting information from the physical world to the MR application in real-time?

- **360° camera**
- **Room model + emissive material**
- **Streaming light probes over the network**

+ works
+ is visually appealing
+ multi-user support (tested with 2 HMDs)
+ low requirements on the HMD

- FPS & delay could be improved
- can't handle too dynamic environments
- setup process is cumbersome

→ works best for static setups with few concurrent users in a clear space (e.g. exhibitions in museums)

Q&A

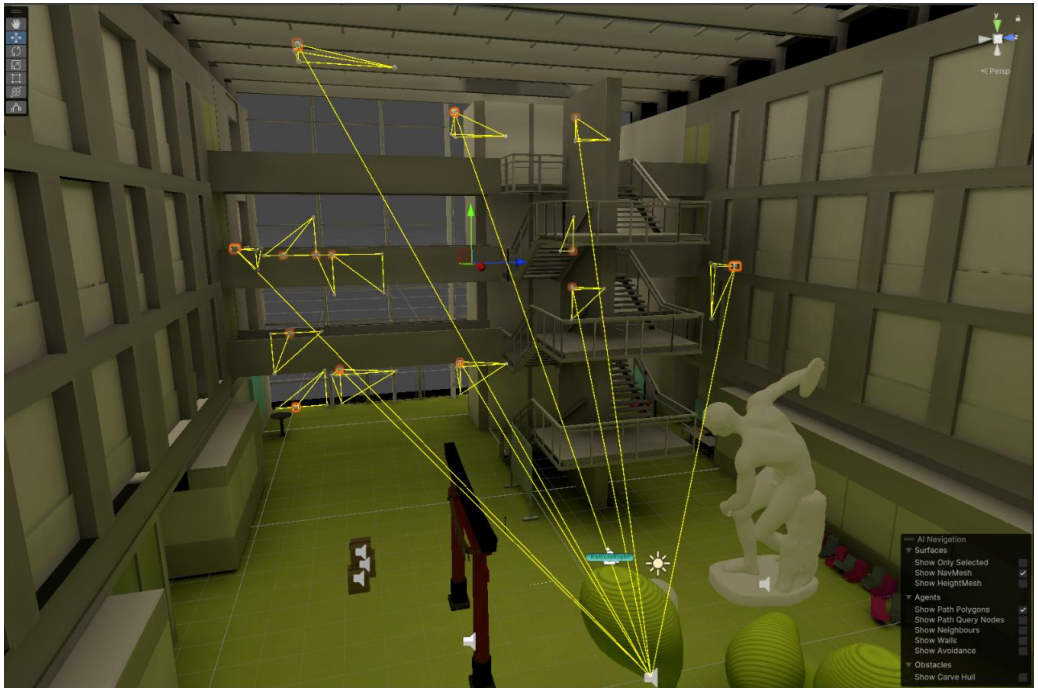
Any questions?

Image Sources

- https://scontent.oculuscdn.com/v/t64.5771-25/38974492_442085696619485_5665262356792344576_n.png?_nc_cat=107&ccb=1-7&_nc_sid=6e7a0a&_nc_ohc=T7N4iK_mO3UQ7kNvgGuZRQ7&_nc_ht=scontent.oculuscdn.com&oh=00_AYBQq080owE6bwtX055LFf7dWuuPNmJXHl-lRWaM_-OYsw&oe=66F8932B (slide 4)
- https://mixed-news.com/en/wp-content/uploads/2022/10/Demeo_Mixed_Reality-860x484.png (slide 4)
- https://i.imgur.com/Fwmkpn0.png (slide 6)
- https://www.az-delivery.de/cdn/shop/products/tcs34725-rgb-farb-sensor-mit-infrarot-filter-diy-modul-zur-farb-erkennung-kompatibel-mit-arduino-414913.jpg?v=1679399266&width=1200 (slide 10)
- https://cdn1.botland.de/123834-pdt_540/sparkfun-tristimulus-farbsensor-opt4048dtsr-qwiic-sparkfun-sen-22638.jpg (slide 10)
- https://de.farnell.com/productimages/standard/en_GB/3879713-40.jpg (slide 10)
- https://m.media-amazon.com/images/I/61o2ZUzB4XL._AC_UF894,1000_QL80_.jpg (slide 10)
- https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRAwg2N3x8ThZIaaJbuxyon9qF6mDl_pNxI_A&s (slide 15, 16)
- https://ricohtheta.eu/cdn/shop/products/Theta-Z1-front-screen-on.jpg?v=1655989855 (slide 15, 17, 18)
- https://vr-expert.de/wp-content/webp-express/webp-images/uploads/2023/08/Quest-3-diagnal-view.png.webp (slide 15, 16)
- https://haid-computers.de/cdn/shop/products/rgb_gaming_pc_talius_adf23630-328c-49ea-bc12-b6bf1bf5dc38_600x.png?v=1659005283 (slide 15, 16)
- https://heise.cloudimg.io/bound/1200x1200/q85.png-lossy-85.webp-lossy-85.foil1/_www-heise-de_/imgs/18/3/0/8/8/9/3/9/Bildschirmfoto_2021-04-15_um_13-427ab236292eb0de.png (slide 17)
- https://docs.unity3d.com/uploads/GlobalIllumination/EmissiveMaterial.png (slide 22)
- https://docs.unity3d.com/uploads/Main/LightProbeProxyVolumeExample2.png (slide 47)
- https://docs.unity3d.com/uploads/Main/LightProbeProxyVolumeExample1.png (slide 47)
- https://docs.unity3d.com/6000.0/Documentation/uploads/urp/probe-volumes/probevolumes-per-pixel.jpg (slide 48)

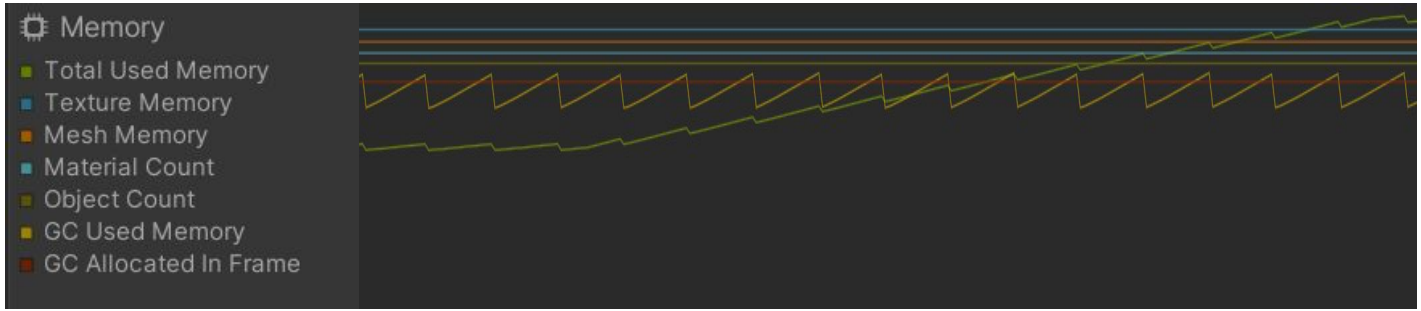
Backup Slides

14 incorrect light probes



Backup Slides

Memory Management



→ **SWAP used 30GB of disk space**

→ **degraded performance of whole system**

Backup Slides

Unity deleted all my Light Probes when saving the scene

→ had to manually edit .unity file to get them back

→ a bug in Unity?