Solving light parameters using RadiosityGS

Eetu Pakkanen – 12507307

August 2026

AI disclosure

We utilized LLMs such as Anthropic Claude and OpenAI Codex in writing and debugging code.

ChatGPT was used for language editing, including correcting grammar, improving sentence structure, and enhancing clarity. All arguments, interpretations, and conclusions are our own.

1 Introduction

Realistic representations of both virtual and real-world scenes are needed for many applications, such as VR/AR, robotics and digital content creation. A good 3D representation of real-world scenes consists of many different aspects like

- the ability to render images from novel views,
- accurate geometry, and
- modelling of lighting and the shadows it induces.

Previous work such as Neural Radiance Fields (NeRFs) [1] and 3D Gaussian Splatting (3DGS) [2] tackle novel view synthesis simultaneously with view-dependent effects such as specular highlights. Meshes are one of the most established explicit representations of 3D scene geometry. There is no universally adopted scene representation that simultaneously provides high-quality novel-view synthesis, accurate explicit geometry, and physically meaningful light transport without significant trade-offs.

RadiosityGS [3] aims to bridge this gap by utilizing *surfels* [4] with precomputed light transport. The method models the exchange of illumination between the surfels using a radiosity formulation. The resulting light transport model allows illumination to be efficiently computed and modified under different lighting conditions. This enables the framework to jointly support novel-view synthesis,

geometry reconstruction, and relighting while accounting for indirect illumination.

While RadiosityGS provides a differentiable framework for modeling light transport and enables relighting of reconstructed scenes, its primary focus is not on optimizing the configuration of light sources. We extend RadiosityGS to allow the parameters of a light source (e.g., intensity and 3D position) to be optimized directly. Using the differentiable light-transport formulation, we search for a light configuration that minimizes a given image-space loss.

We extend RadiosityGS with a workflow for inverse light estimation and controllable relighting. First, we introduce optimization of lights on top of frozen, pre-trained RadiosityGS scenes. Second, we implement directional spotlights. We also provide tooling to help in, e.g., creating synthetic datasets from Blender models.

2 Related Work

Albedo describes the fraction of incident light reflected by a surface and is therefore an intrinsic material property rather than a property of the illumination. In RGB rendering, it is typically represented as a three-channel value $\boldsymbol{\rho} = (\rho_R, \rho_G, \rho_B)$, which determines the diffuse color of the surface. The observed brightness of a surface therefore depends jointly on its albedo and on the amount of incoming illumination. This feature makes the problem of albedo estimation from images *ill-posed*; a surface might appear dark due either to its material properties or to shading. Thus, it might be beneficial to constrain the problem with *a priori* information. Deep learning-based approaches are particularly promising in this regard, as they can learn semantic priors about scene structure, object appearance, and material properties from large datasets.

`Intrinsic Image Diffusion` [5] is a recent deep learning-based approach for material and albedo estimation. Its objective is to "predict albedo and BRDF properties given a single input image". The authors use the extensive synthetic `InteriorVerse` dataset [6] to fine-tune a conditional diffusion model based on Stable Diffusion [7]. In addition, a custom trainable encoder $\mathcal{E}$ is used to extract spatially relevant features from the input image, while `CLIP` [8] provides semantic conditioning features through cross-attention. After training the diffusion model, material predictions are generated through the standard iterative reverse diffusion process, producing albedo, roughness, and metallic maps.

Our initial approach was to use `Intrinsic Image Diffusion` to generate albedo maps for the input images and subsequently train a 3D Gaussian Splatting model (with zero-order SH) on these estimated albedo images. The resulting 3DGS representation could then be used for lighting optimization and relighting. However, `Intrinsic Image Diffusion` processes each view independently and therefore does not explicitly enforce multi-view consistency between the predicted albedo maps.

Training a 3DGS representation nevertheless introduces some implicit consistency across views, since the color of each Gaussian is jointly optimized to explain all observations in which that Gaussian is visible. Consequently, an erroneous albedo prediction in a single view does not necessarily dominate the reconstructed appearance if the same surface is correctly predicted and visible from several other viewpoints. However, it does not constitute true multi-view-consistent albedo estimation, as inconsistencies are only reconciled indirectly rather than being explicitly constrained during the albedo estimation itself.

As our end objective is to recover the lighting of the scene, the problem can be formulated as *inverse rendering*. A practical example of a forward rendering pipeline is the synthesis of an image I_i using, for example, Blender [9]. To generate the image, one needs a description of the scene geometry $\mathcal{G}$ and its material properties $\mathcal{M}$, together with the scene illumination $\mathcal{L}$ and the camera parameters $\mathcal{K}_i$. The forward rendering process can therefore be written as

$$I_i = r(\mathcal{G}, \mathcal{M}, \mathcal{L}, \mathcal{K}_i),$$

where $r(\cdot)$ denotes an arbitrary rendering function. In forward rendering, the parameters of $r(\cdot)$ are known and the objective is to synthesize the corresponding image I_i. In inverse rendering, the setting is reversed: an observed image I_i is given and one or more of the underlying scene parameters are estimated. In our case, the primary unknown is the illumination $\mathcal{L}$.

Inverse rendering is generally an ill-posed problem, since multiple combinations of scene parameters may produce identical or very similar rendered observations. In other words, the set of feasible solutions $\mathcal{S}$ may contain several parameter configurations $(\mathcal{G}, \mathcal{M}, \mathcal{L}, \mathcal{K})$ that explain the same image. Additional observations, prior information, or constraints are therefore required to reduce the solution space and recover physically meaningful scene parameters.

One way to solve the problem of inverse rendering is to start from an initial guess for the unknown parameter and iteratively adjust it toward the correct value with gradient-based methods such as `Adam` [10]. Gradients can be obtained either numerically using finite differences or analytically through differentiable rendering. In finite-difference methods, each degree of freedom of the optimized parameter vector is perturbed by a small amount ϵ, after which the scene is rendered again to observe the resulting change in the objective function. For a parameter θ_j, the corresponding derivative can be approximated, for example, using the forward difference

$$\frac{\partial \mathcal{L}}{\partial \theta_j} \approx \frac{\mathcal{L}(\theta_j + \epsilon) - \mathcal{L}(\theta_j)}{\epsilon}.$$

Repeating this procedure for every degree of freedom yields an approximate gradient of the objective with respect to the optimized parameters. The main

disadvantage is computational cost; For n_v views, estimating a gradient for D parameters[1] requires approximately $n_v(D+1)$ forward renders per optimization step when using forward differences. Since rendering can itself be expensive, finite-difference optimization quickly becomes impractical for high-dimensional inverse-rendering problems.

Analytical gradients can instead be obtained using differentiable rendering frameworks. Examples include `nvdiffrast` [11], `PyTorch3D` [12], and `Mitsuba 3` [13], as well as its earlier versions. For our use case, `Mitsuba 3` is the most relevant, as it supports differentiable Monte Carlo path tracing and can therefore model global illumination.

While `Mitsuba 3` offers physically accurate differentiable rendering, path tracing is computationally expensive, as each viewpoint requires tracing new camera rays and differentiating through the rendering process. `RadiosityGS` instead separates view-independent light transport from image formation. It solves outgoing radiance in scene space once, then reuses it to efficiently render novel views with Gaussian splatting, achieving hundreds of frames per second while retaining global illumination effects.

The radiosity formulation used in `RadiosityGS` is closely related in spirit to finite-element methods. Instead of solving the continuous light-transport equation directly over all surface points, the scene is discretized into a finite set of elements and the unknown outgoing radiance is represented using coefficients associated with those elements. In classical radiosity, these elements are typically surface patches, whereas in `RadiosityGS` the Gaussian surfels themselves act as the discrete elements of the formulation.

Radiosity becomes cumbersome for conventional mesh-based scenes. A sufficiently accurate mesh may contain a very large number of surface patches, and evaluating visibility and energy transfer between all pairs of patches can lead to prohibitive computational and memory costs. A naive radiosity solver would nevertheless remain expensive, since computing the contribution of every Gaussian surfel to every other Gaussian would require considering approximately $\mathcal{O}(n^2)$ pairwise interactions for n surfels. `RadiosityGS` therefore introduces a Monte Carlo solver that estimates light-transport by sampling only a subset of surfels. Instead of evaluating every possible interaction, the method assigns higher sampling probability to surfels that are expected to contribute strongly based on quantities such as their outgoing radiance and geometric coupling to the receiving surfel.

This differs from conventional Monte Carlo path tracing, where random light paths are typically sampled from camera rays in order to estimate individual pixel values. In `RadiosityGS`, Monte Carlo sampling is instead used to approximate the transport between the discrete Gaussian elements themselves. The resulting outgoing radiance is stored on the surfels and can subsequently be

[1] i.e., degrees of freedom

reused across camera viewpoints, allowing novel views to be rendered efficiently using the fast rasterization inherent to 3D Gaussian Splatting.

`RadiosityGS` also accounts for *global illumination*. The global-illumination formulation of `RadiosityGS` is recursive, since the outgoing radiance B_i^c of Gaussian i in color channel c depends on the outgoing radiance B_j^c of other Gaussians j. Consequently, radiance received by one surfel can be reflected further to another surfel in subsequent iterations, allowing illumination to propagate through multiple surface interactions. Repeatedly solving this recursive light-transport equation therefore accounts for multiple-bounce indirect illumination. The Monte Carlo solver is similar in nature to TD(0), as current estimates of the outgoing radiance are used to update other radiance estimates without explicitly evaluating complete light paths.[2]

3 Method

3.1 Light optimization

We freeze the reconstructed Gaussian surfel scene, including its geometry and material parameters, and optimize only a rig of local lights through the differentiable `RadiosityGS` renderer. For each point light k, the trainable parameters are its position $\mathbf{x}_k \in \mathbb{R}^3$ and log-intensity $\mathbf{a}_k \in \mathbb{R}^3$. The physical RGB intensity is parameterized as

$$\mathbf{i}_k = e^{\mathbf{a}_k}, \text{where } \mathbf{a}_k \in [-30, 30]^3.$$

Optimizing intensity in log space makes Adam updates multiplicative, which provides a more suitable parameterization for HDR light values. The position–intensity ambiguity caused by inverse-square falloff is handled separately by constraining the light position. For spotlights, we additionally optimize the light orientation; the cone cutoff angle and Gaussian falloff width remain fixed, while the directional emission profile is represented using spherical-harmonic coefficients.

Given a rendered image I_{render}, the corresponding reference image I_{gt}, and its alpha mask $\mathbf{M}$, we minimize the following masked photometric objective for a single view:

$$\mathcal{L}_{\text{light}} = (1 - \lambda_{\text{DSSIM}}) \left\| \mathbf{M} \odot (I_{render} - I_{gt}) \right\|_1 + \lambda_{\text{DSSIM}} \left[1 - \text{SSIM} \left(\mathbf{M} \odot I_{render}, \mathbf{M} \odot I_{gt} \right) \right].$$

Here, $\lambda_{DSSIM} \in [0, 1]$ controls the relative weighting of the DSSIM and L_1 terms. and The alpha mask $\mathbf{M} \in [0, 1]^{H \times W}$ denotes the foreground opacity of the reference image. It restricts the photometric loss to the object region, so that background pixels do not dominate light estimation. When no alpha

[2] TD(0), or temporal-difference learning with zero-step lookahead, is a reinforcement-learning method that updates a current value estimate using an immediate observation and the current estimate of the next state, rather than waiting for a complete trajectory to finish.

channel is available, we use $\mathbf{M} = \mathbf{1}$. For multi-view optimization, we average the single-view photometric losses over all n_v views:

$$\mathcal{L}_{\text{light}} = \frac{1}{n_v} \sum_{v=1}^{n_v} \mathcal{L}_{\text{light},v}.$$

Light positions are optimized directly in Cartesian coordinates $\mathbf{x}_k \in \mathbb{R}^3$. To prevent the inverse-square position–intensity ambiguity from sending a light far from the scene, we constrain each position to the closed ball

$$\mathcal{B}(\mathbf{c}, r_{\max}) = \{\mathbf{x} \in \mathbb{R}^3 : \|\mathbf{x} - \mathbf{c}\|_2 \le r_{\max}\},$$

where $\mathbf{c}$ is the centre of the reconstructed surfel cloud and $r_{\max} > 0$ is the maximum allowed distance of a light from this centre. After each Adam update, positions outside this ball are radially projected back to its boundary:

$$\mathbf{x}_k \leftarrow \mathbf{c} + \min\left(1, \frac{r_{\max}}{\|\mathbf{x}_k - \mathbf{c}\|_2}\right)(\mathbf{x}_k - \mathbf{c}).$$

This constraint prevents degenerate solutions in which the optimizer moves a light arbitrarily far away to compensate for its intensity. For OLAT data, where illumination differs between views, optimization is performed against one selected target view rather than fitting a single global rig to mutually inconsistent observations.

3.2 Spotlights

We implemented spotlights without modifying the `RadiosityGS` light-transport solver. Each spotlight is represented as a small emissive Gaussian surfel with position $\mathbf{x}_k \in \mathbb{R}^3$, RGB intensity $\mathbf{i}_k$, and a rotation that aligns its local $+z$ axis with the optimized unit direction $\mathbf{d}_k$. Its angular emission is encoded using spherical-harmonic (SH) coefficients, allowing the existing directional-emission evaluation in the renderer to produce a cone-shaped light distribution.

For an angle θ from the spotlight axis, the desired normalized emission profile is

$$f(\theta) = \begin{cases} 1, & \theta \le \theta_c, \\ \exp\left(-\dfrac{(\theta - \theta_c)^2}{2\sigma^2}\right), & \theta > \theta_c, \end{cases}$$

where θ_c is the fixed inner-cone half-angle and σ controls the smooth Gaussian falloff. We fit this profile once to the renderer's SH basis; during optimization, only the spotlight position, orientation, and log-intensity are updated. The light

position is constrained to the same closed Cartesian ball $\mathcal{B}(\mathbf{c}, r_{\max}) \subset \mathbb{R}^3$ used for point-light optimization.

Because the spotlight profile is band-limited by the finite SH degree, narrow requested cones are rendered with a somewhat softer boundary.

3.3 Blender scenes

We generate a GS^3-compatible Cornell-box dataset in Blender. The script removes existing cameras and lights, normalizes the scene to the unit cube, and renders linear HDR OpenEXR images with Cycles. Each frame uses a sampled camera aimed at the internal objects and a known point light sampled inside the room. We store the image, camera pose $\mathbf{T}_v$, light position $\mathbf{l}_v$, and RGB intensity $\mathbf{i}_v$ as

$$\mathcal{D} = \{(I_v, \mathbf{T}_v, \mathbf{l}_v, \mathbf{i}_v)\}_{v=1}^{N},$$

where N is the number of rendered views. We also render depth maps for the training views and back-project valid depth pixels into a depth-initialized point cloud:

$$\mathbf{p} = \mathbf{o}_v + d\,\mathbf{R}_v\mathbf{r}(u_1, u_2),$$

where $\mathbf{o}_v$ and $\mathbf{R}_v$ are the camera origin and rotation, d is the rendered depth, (u_1, u_2) are the image coordinates, and $\mathbf{r}(u_1, u_2)$ is the camera-space ray. This point cloud initializes the Gaussian surfel means.

4 Experiments

4.1 Light optimization

Our light optimization framework allows one to solve the location $\mathbf{x}_k \in \mathbb{R}^3$ and (log-)intensity $\mathbf{i}_k$ of point light k. This is visualized in Figure 1.

4.2 Multiple lights

Our method supports relighting the scene with an arbitrary number of point lights or spotlights. Figure 2 visualizes our custom Cornell box scene in different lighting configurations.

4.3 Spotlights

Our codebase allows both relighting and optimizing spotlights. Figure 3 showcases relighting with different setups.

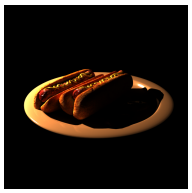
Epoch 0
random init.

Epoch 10

Epoch 20

Epoch 30

Epoch 50



Ground truth

Figure 1: Progression of point-light optimization on the Hotdog scene. The optimized rendering gradually approaches the ground-truth reference.

(a) One point light

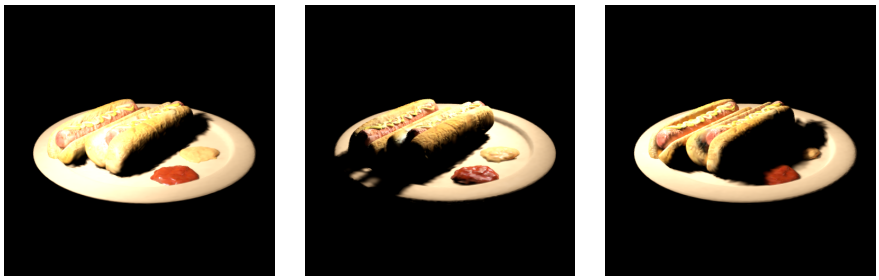
(b) Two point lights

Figure 2: Rendering results with different numbers of point lights.

4.4 Occluded optimization

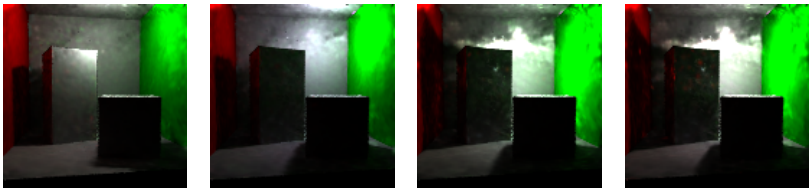
Optimization of occluded light sources remains especially interesting as it answers one of our main motivations, i.e., the possibility to "see around corners." This configuration is showcased in Figure 4.

(a) Aim sweep: the spotlight remains fixed while its direction changes.

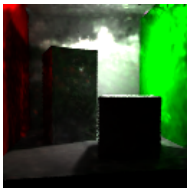


(b) Orbit sweep: the spotlight moves around the scene while remaining aimed at the object.

Figure 3: Spotlight relighting using the `aim` and `orbit` command-line sweep modes. The selected frames illustrate the changing illumination and shadows.



Epoch 0 Epoch 1 Epoch 2 Epoch 10

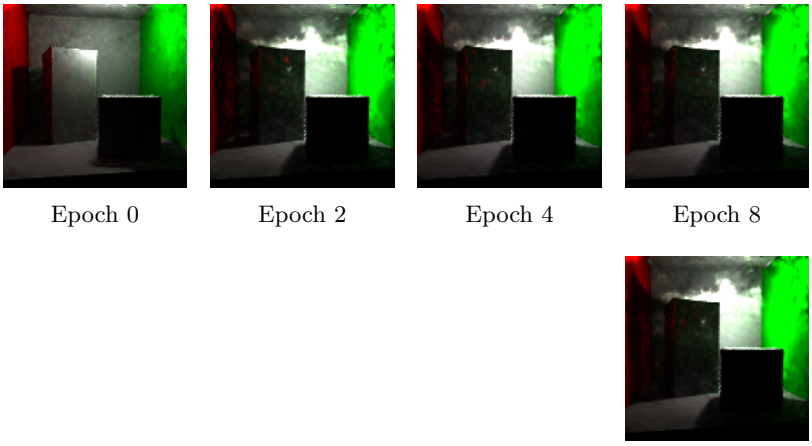


Ground truth

Figure 4: Light optimization when the ground truth light source is occluded. At epoch 0, the light source is initialized at a visible location. Optimization was run with lower-resolution images for speed.

4.5 Single vs. multi-view

Lastly, we demonstrate how utilizing information from multiple views instead of a single one helps the optimization. This is visualized in Figure 5.



Epoch 0 Epoch 2 Epoch 4 Epoch 8

Ground truth

Figure 5: Multi-view light optimization using three views visualized from view 2. The initialization and hyperparameters are the same as in Figure 4. One can see that multi-view optimization allows the solution to be found in fewer epochs.

5 Conclusion

In this project, we developed a framework for light optimization using `RadiosityGS`. Given a frozen scene in which the geometry, material properties, and camera parameters are assumed to be known, the framework estimates the parameters of an unknown light source from rendered observations. In particular, we optimize the three-dimensional position and intensity of the light source and, when applicable, its orientation. To enable directional illumination, we additionally extended `RadiosityGS` with support for spotlights whose orientation can be optimized jointly with the other lighting parameters.

The proposed framework was evaluated using two synthetic datasets: a custom Cornell box scene and a scene from the TensoIR synthetic dataset [14, 15]. These experiments demonstrate that `RadiosityGS` can be used not only for forward relighting, but also as part of an inverse-rendering pipeline in which illumination parameters are recovered from image observations.

Several directions remain for future work. First, the method should be evaluated on real-world data. Both the original `RadiosityGS` work and the experiments presented in this project rely on synthetic scenes, where geometry, material

properties, camera calibration, and illumination can be controlled accurately. Real-world data would introduce additional challenges such as imperfect geometry, inaccurate material estimates, sensor noise, and lighting effects that are not fully represented by the current model. It would also be beneficial to evaluate the method on larger, room-scale environments rather than only on object-centric scenes. The `InteriorVerse` dataset would provide a suitable synthetic benchmark for this purpose.

Second, a more systematic comparison between single-view and multi-view light optimization should be conducted. Multiple viewpoints provide additional constraints on the unknown illumination and may reduce ambiguities that cannot be resolved from a single image. Quantifying this effect would help determine how many observations are required for reliable light estimation under different scene configurations.

A potential application of the proposed approach is illumination-aware perception in autonomous driving. A vehicle may already estimate scene geometry and camera poses using, e.g., SLAM, while material properties could be estimated as part of an inverse-rendering pipeline. Observed illumination could then provide information about objects that are not directly visible. For example, headlights from a vehicle approaching around a corner may illuminate a visible wall. By estimating the three-dimensional position of the corresponding light source, an autonomous vehicle could potentially infer the presence and motion of the otherwise occluded vehicle before it enters the camera's direct line of sight. Such an application illustrates how inverse light estimation could provide additional scene information beyond conventional image-based perception.

References

[1] Ben Mildenhall et al. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *ECCV*. 2020.

[2] Bernhard Kerbl et al. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Transactions on Graphics* 42.4 (July 2023). URL: `https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/`.

[3] Kaiwen Jiang et al. "Differentiable Light Transport with Gaussian Surfels via Adapted Radiosity for Efficient Relighting and Geometry Reconstruction". In: *ACM Transactions on Graphics (TOG)* 44.6 (Dec. 2025).

[4] Binbin Huang et al. "2D Gaussian Splatting for Geometrically Accurate Radiance Fields". In: *SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, 2024. DOI: `10.1145/3641519.3657428`.

[5] Peter Kocsis, Vincent Sitzmann, and Nie{ssner, familyi=N., given=Matthias, giveni=M. "Intrinsic Image Diffusion for Indoor Single-view Material Estimation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024.

[6] Jingsen Zhu et al. "Learning-Based Inverse Rendering of Complex Indoor Scenes with Differentiable Monte Carlo Raytracing". In: *SIGGRAPH Asia 2022 Conference Papers*. ACM, 2022. URL: https://doi.org/10.1145/3550469.3555407.

[7] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2021. arXiv: 2112.10752 [cs.CV].

[8] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV]. URL: https://arxiv.org/abs/2103.00020.

[9] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Blender Institute, Amsterdam. URL: http://www.blender.org.

[10] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[11] Samuli Laine et al. "Modular Primitives for High-Performance Differentiable Rendering". In: *ACM Transactions on Graphics* 39.6 (2020).

[12] Nikhila Ravi et al. "Accelerating 3D Deep Learning with PyTorch3D". In: *arXiv:2007.08501* (2020).

[13] Wenzel Jakob et al. *Mitsuba 3 renderer*. Version 3.9.1. https://mitsuba-renderer.org. 2022.

[14] Haian Jin et al. "TensoIR: Tensorial Inverse Rendering". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023.

[15] Zoubin Bi et al. "GS^3: Efficient Relighting with Triple Gaussian Splatting". In: *SIGGRAPH Asia 2024 Conference Papers*. 2024.