



HAL
open science

Image-based methods for view-dependent effects in real and synthetic scenes

Simon Rodriguez

► **To cite this version:**

Simon Rodriguez. Image-based methods for view-dependent effects in real and synthetic scenes. Graphics [cs.GR]. Université Côte d'Azur, 2020. English. NNT : 2020COAZ4038 . tel-03282268v2

HAL Id: tel-03282268

<https://theses.hal.science/tel-03282268v2>

Submitted on 9 Jul 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT

Méthodes basées image pour le rendu d'effets dépendants
du point de vue dans les scènes réelles et synthétiques

Simon Rodriguez

Inria Sophia Antipolis-Méditerranée

**Présentée en vue de l'obtention du
grade de docteur en Informatique
d'Université Côte d'Azur**

Dirigée par : George Drettakis

Soutenue le : 8 Septembre 2020

Devant le jury composé de :

Pierre Alliez, Directeur de Recherche, Inria Sophia
Antipolis - Méditerranée

Tamy Boubekeur, Professeur, Telecom Paris, Adobe
3D&Immersive, et Ecole Polytechnique

George Drettakis, Directeur de Recherche, Inria
Sophia Antipolis - Méditerranée

Elmar Eisemann, Professeur, TU Delft

Michael Goesele, Research Scientist, Facebook
Reality Labs

Inria

Méthodes basées image pour le rendu d'effets dépendants du point de vue dans les scènes réelles et synthétiques

Image-based methods for view-dependent effects in real and synthetic scenes

Jury:

Président du jury / President of the jury

Pierre Alliez, Directeur de Recherche, Inria Sophia Antipolis - Méditerranée

Rapporteurs / Reviewers

Michael Goesele, Research Scientist, Facebook Reality Labs

Elmar Eisemann, Professeur, TU Delft

Examineurs / Examiners

Tamy Boubekeur, Professeur, Telecom Paris, Adobe 3D&Immersive, et Ecole Polytechnique

Directeur de thèse / Thesis supervisor

George Drettakis, Directeur de Recherche, Inria Sophia Antipolis - Méditerranée

Acknowledgements

This thesis would not exist without the trust and support of my advisor **George Drettakis**. Thank you for your never-ending optimism, your guidance in my research and for helping me find my way around academia.

I am eternally indebted to all my co-authors for the time and resources they brought and their helpful advice. I more specifically want to thank **Adrien Bousseau** for his guidance, **Peter Hedman** for his incredible enthusiasm – especially during deadlines –, **Chris Wyman** and **Peter Shirley** for welcoming me to Seattle.

I want to acknowledge the Université Côte d’Azur, Inria and the European Research Council for financially supporting this thesis. I am also grateful to the anonymous reviewers for their constructive feedback on submissions. I finally want to thank the jury of my thesis for accepting to review this manuscript and take part in the defense.

The GraphDeco team has been a formidable group of people. Thank you to everyone for the insightful discussions, whether it was around a computer screen or at a coffee break. I am grateful for the help many of you provided all along this thesis. I especially thank **Johanna Delanoy** for being my older PhD sibling along with **Théo Thonat**, who also was my partner in graphics and programming crimes. Shout out to **Bastien Wailly** for endless discussions and gaming nights. A special thanks to **Julien Philip** for facing the doctoral studies together. This also applies to **Valentin Deschaintre**, who additionally had to share an office – and jokes – with me for three years.

I am immensely grateful to my parents for always fostering my curiosity, supporting me as I grew up and more generally being awesome. To my grandparents for their interest in my studies and their support, to my uncle for welcoming me in the south of France, and to my brother for always being there. Thank you also to my friend **Gédéon Chevallier** for always lending an interested ear to my research progress.

Finally, I want to thank my partner **Claire Li** for her infinite unwavering support. Thank you for bearing with my musings and showing a genuine interest in them. Without your love and care, I could not have made it.

Résumé

La création et le rendu interactif d'environnements réalistes est un problème complexe qui requiert de nombreux réglages et ajustements manuels. Les méthodes basées-image visent à simplifier ces tâches en utilisant des vues existantes d'une scène réelle ou synthétique. Ces points de vue peuvent être capturés dans le monde réel ou générés grâce à des algorithmes de rendu hors-ligne très réalistes. Pour générer un nouveau point de vue sur la scène, la géométrie et l'apparence associées sont estimées à partir de l'information des vues existantes, permettant l'exploration de l'environnement en temps réel.

Lorsque les vues d'entrée ne couvrent la scène que partiellement, spatialement ou angulairement, des artefacts apparaissent souvent ; utiliser un faible nombre d'images d'entrée limite la qualité de la reconstruction des scènes réelles, tandis que les matériaux non-diffus compliquent le rendu des scènes tant réelles que synthétiques. Nous proposons plusieurs méthodes pour contourner ces limitations, en améliorant la façon dont l'information est stockée et agrégée depuis les vues d'entrée tout en exploitant les redondances. Nous décrivons aussi des outils géométriques afin de reprojeter les effets dépendants du point de vue pour la génération d'une nouvelle vue.

Pour les scènes tirées du monde réel, nous détectons la présence de matériaux et objets grâce à de l'information sémantique. Nous présentons tout d'abord une méthode pour reconstruire et rendre des éléments architecturaux à partir de quelques vues. Nous exploitons la nature répétitive de ces éléments, extrayant et combinant leur information d'apparence et de géométrie pour générer une représentation commune idéale qui peut par la suite être réinsérée dans la scène initiale. Cette combinaison améliore l'estimation des paramètres des vues d'entrée, la reconstruction de la géométrie de l'élément, et est utile pour détecter les régions comportant des effets spéculaires dépendants du point de vue.

Nous décrivons une deuxième méthode qui s'appuie aussi sur la sémantique de la scène pour améliorer le rendu d'environnements urbains. Dans ces scènes, les voitures présentent de nombreux effets dépendants du point de vue qui les rendent complexes à reconstruire et rendre fidèlement. Nous détectons et extrayons ces éléments grâce à

une paramétrisation spécifique à chaque instance, afin de raffiner leur géométrie, tout en construisant une représentation simplifiée des surfaces réfléchives, telles que les fenêtres. Pour rendre les effets dépendants du point de vue, nous séparons l'information spéculaire de chaque vue d'entrée et utilisons notre représentation analytique des réflecteurs pour reprojeter l'information dans la nouvelle vue, en respectant le déplacement des réflexions.

Enfin, bien que les scènes synthétiques fournissent une information exacte de la géométrie et des matériaux présents, restituer fidèlement les effets non-diffus de façon interactive reste difficile. En nous inspirant de notre deuxième méthode, nous proposons une nouvelle approche pour rendre ces effets en utilisant de l'illumination globale précalculée. Pour un ensemble de points de vue prédéfinis dans la scène, nous précalculons l'information spéculaire, stockée dans des images panoramiques. Grâce à une représentation simplifiée de la géométrie, nous pouvons estimer le déplacement des réflexions de façon robuste, accumulant dans la nouvelle vue l'information provenant des panoramas. Combinée avec une paramétrisation adaptative des images panoramiques et un filtre de reconstruction préservant les matériaux, cette méthode restitue les effets de réflexions pour différents types de matériaux, de façon interactive.

Les résultats obtenus grâce à nos méthodes montrent une amélioration de la qualité du rendu des effets dépendants du point de vue, que ce soit pour des données synthétiques ou du monde réel. Ces travaux ouvrent la voie à de futures recherches sur les techniques basées-image pour le rendu réaliste.

Mots-clés: rendu basé images - effets dépendants du point de vue - synthèse de réflexions
- reconstruction de surface

Abstract

The creation and interactive rendering of realistic environments is a time consuming problem requiring human interaction and tweaking at all steps. Image-based approaches use viewpoints of a real world or high quality synthetic scene to simplify these tasks. These viewpoints can be captured in the real world or generated with accurate offline techniques. When rendering a novel viewpoint, geometry and lighting information are inferred from the data in existing views, allowing for interactive exploration of the environment. But sparse coverage of the scene in the spatial or angular domain introduces artifacts; a small number of input images is adversarial to real world scene reconstruction, as is the presence of complex materials with view-dependent effects when rendering both real and synthetic scenes. We aim at lifting these constraints by refining the way information is stored and aggregated from the viewpoints and exploiting redundancy. We also design geometric tools to properly reproject view-dependent effects in the novel view.

For real world scenes, we rely on semantic information to infer material and object placement. We first introduce a method to perform reconstruction of architectural elements from a set of three to five photographs of a scene. We exploit the repetitive nature of such elements to extract them and aggregate their color and geometry information, generating a common ideal representation that can then be reinserted in the initial scene. Aggregation helps improve the accuracy of the viewpoint pose estimation, of the element geometry reconstruction, and is used to detect locations exhibiting view-dependent specular effects.

We describe a second method designed to similarly rely on semantic scene information to improve rendering of street-level scenery. In these scenes cars exhibit many view-dependent effects that make them hard to reconstruct and render accurately. We detect and extract them, relying on a per-object view parameterization to refine their geometry, including a simplified representation of reflective surfaces such as windows. To render view-dependent effects we perform a specular layer separation and use our analytical reflector representation to reproject the information in the novel view following the flow

of reflections.

Finally, while synthetic scenes provide completely accurate geometric and material information, rendering high quality view-dependent effects interactively is still difficult. Inspired by our second method, we propose a novel approach to render such effects using precomputed global illumination. We can precompute specular information at a set of predefined locations in the scene, stored in panoramic probes. Using a simplified geometric representation we robustly estimate the specular flow, gathering information from the probes at a novel viewpoint. Combined with an adaptive parameterization of the probes and a material-aware reconstruction filter, we render specular and glossy effects interactively.

The results obtained with our methods show an improvement in the quality of recreated view-dependent effects, using both synthetic and real world data and pave the way for further research in image-based techniques for realistic rendering.

Keywords: image-based rendering - view-dependent effects - reflections synthesis - surface reconstruction

Contents

Contents	vii
1 Introduction	1
1.1 Rendering an environment	1
1.2 Scene representation and creation	3
1.3 Importance of view-dependent effects	5
1.4 Contributions	7
1.5 Funding and publications	8
2 Related work	9
2.1 Image-based rendering	11
2.1.1 General approaches	11
2.1.2 Geometric scene representation	14
2.1.3 Blending techniques	20
2.1.4 Synthetic data	23
2.2 Extraction and rendering of view-dependent effects	27
2.2.1 Extracting view-dependent effects from real world data	28
2.2.2 Warping specular effects	30
2.2.3 Generating specular effects	34
2.2.3.1 Reprojecting reflected objects	34
2.2.3.2 Finding reflected points in the scene	36
2.3 Summary	41
3 Repetitions for Image-Based Rendering of Facades	43
3.1 Introduction	43
3.2 Related Work	45
3.3 Overview	47
3.4 Windows Extraction and Platonic Camera Calibration	50
3.4.1 Window extraction	50
3.4.2 Platonic Camera Calibration	51
3.5 Geometry Reconstruction	53
3.5.1 Platonic Window	53
3.5.2 Complete Facade	55
3.6 Data Factorization and Augmentation	57
3.6.1 View-Dependent Variations	57

3.6.2	Instance-Dependent Variations	59
3.6.3	Complete Facade	61
3.7	Implementation and results	63
3.7.1	Rendering	63
3.7.2	Implementation	63
3.7.3	Results	64
3.8	Conclusion	66
4	Image-Based Rendering of Cars with an Approximate Reflection Flow	69
4.1	Introduction	69
4.2	Related Work	71
4.3	Overview	72
4.4	Car Geometry Extraction and Refinement	74
4.4.1	Isolating Cars with Semantic Labels	74
4.4.2	Smooth Car Hull Extraction and Semantic Mesh Refinement	75
4.5	Ellipsoid Approximation for Reflection Flow Computation	78
4.5.1	Reflection flow computation	79
4.5.2	Ellipsoid Fitting for Car Windows	80
4.6	Synthesizing Reflection Layers	81
4.6.1	Reflection Layer Synthesis	82
4.7	Rendering, Results and Comparisons	83
4.7.1	Rendering and Implementation	83
4.7.2	Results	85
4.7.3	Comparisons	86
4.8	Conclusion	86
5	Glossy Probes Reprojection for Global Illumination	89
5.1	Introduction	89
5.2	Related Work	91
5.3	Overview	93
5.4	Probe generation and storage	95
5.4.1	Per-probe Data	95
5.4.2	Probe Parameterization	96
5.4.2.1	Adaptive Resolution Map Computation	97
5.4.2.2	Adaptive Parameterization	98
5.4.3	Geometric Information	99
5.5	Rendering Global Illumination	99
5.5.1	On-the-fly Reflection Position Estimation	100
5.5.2	Gathering View-dependent Color	101
5.6	Two-step Convolution for Accurate Warping of Glossy Probes	103
5.6.1	Filter Footprint Estimation	104
5.6.2	Gloss Filtering	105
5.6.3	Efficient Filter Approximation	106

5.7	Results, Evaluation and Comparisons	107
5.7.1	Test Scenes	107
5.7.2	Evaluation	108
5.7.3	Results and Comparisons	109
5.7.3.1	Comparisons with Baselines.	111
5.7.3.2	Comparisons with Prior Art.	111
5.7.3.3	Quantitative Evaluation.	112
5.7.3.4	Statistics.	112
5.8	Conclusion	114
6	Conclusion	117
6.1	Contributions	117
6.2	Insights	118
6.3	Future work	120
6.4	Impact	122
A	Transformation from Platonic Scene to Input Scene	123
B	Car Mesh Refinement and Ellipsoid Fitting	125
B.1	Mesh Refinement Minimization	125
B.2	Isolating Car Objects using Semantic Labels	125
B.3	Ellipsoid fitting algorithm	126
C	Choice of DCT for Parameterization Guiding	129

Chapter 1

Introduction

For the last fifty years, computer graphics researchers have explored ways of creating, editing and visualizing data through digital tools. The exponential democratization of computers combined with their compactness and their flexibility to display many kinds of data has deeply impacted our society. An important part of the information and entertainment we share and consume is delivered visually. Among the possibilities brought by computer graphics, the ability to explore environments has impacted many fields – even if they do not exist in reality or are unreachable. Architecture, design, healthcare, entertainment all benefit from the capability of visualizing complex objects and environments while interacting with them.

But creating and rendering such data is time-consuming, labor intensive and requires significant processing power. Creating scenes from nothing or acquiring them from the real-world are arduous tasks. Displaying a scene to the user with the proper level of detail and realism is another challenge, as humans rely constantly on the appearance and visual behavior of objects to infer their properties and function [Fle14]. In this thesis, we explore how existing or precomputed scene data can be leveraged to improve the rendering of elements with complex appearance behavior while the user explores an environment in real-time.

1.1 Rendering an environment

Rendering in computer graphics can be defined as the process of displaying existing data onto a device screen. We focus on 3D rendering, where three-dimensional objects are projected on the screen plane to generate an image. The user point of view is modeled as a simplified *camera*, and the scene data is transformed so that the image faithfully shows which objects would be visible to the user from the camera location.

Different methods exist to determine which region of which object is visible in each part of the image. *ray tracing* iterates over the image pixels; a ray originating at the viewer's

virtual location and going through the pixel is shot in the scene, until it intersects an object. Conversely, *rasterization* determines the footprint of each object on screen, and updates the content of the covered pixels accordingly while keeping track of object depth ordering.



Figure 1.1 – Left: Offline rendering from the *Spring* short movie (2019) ¹. Right: real-time rendering from the video game *Control* (2019) ².

Rendering techniques relying on ray tracing can closely model the propagation of light in a scene and support arbitrarily complex material behavior for increased physical accuracy [PJH16]. But because it requires the ability to perform visibility queries that possibly span the whole scene, memory and power requirements prevent it from being widely used in real-time. Techniques such as path tracing have thus been mainly reserved to tasks that favor accuracy over performance, such as movie visual effects [FHP⁺18] (see for instance Fig. 1.1, left) or architectural pre-visualization.

On the other hand, rasterization is the *de facto* method for interactive rendering, to the extent that graphics processing units (GPU) have been specially designed to support it. Care has to be taken to ensure that the proper object ordering and occlusions are respected, but power and memory requirement are lower, making it amenable to performance critical tasks (Fig. 1.1, right). In recent years commercial hardware support for ray tracing has begun to appear, meaning that the strengths of both methods can be combined for improved accuracy, depending on the task at hand.

¹Blender Open projects, <https://www.blender.org/about/projects/>

²Remedy Entertainment, <https://www.remedygames.com/games/control/>

1.2 Scene representation and creation

But determining visibility and object location is only the first step towards rendering an environment. Each object appearance has to be determined based on its *material* properties and the scene lighting conditions. While we focus on natural-looking environments in this thesis, rendering is not necessarily realistic *per se*; stylization and simplification can be used to great effect in animated movies or computer-assisted design for instance.

To generate a new viewpoint in a scene, multiple elements are required. The geometry of each object has to be represented, either as a surface mesh, a parameterized model, the solution of an analytic equation, a 3D volumetric discretization or other ad hoc techniques. The way light interacts with a given surface is described by a *bidirectional reflectance distribution function* (BRDF); many models have been developed to synthesize materials as diverse as skin, fabric, metal or hair in real-time. BRDF parameters can vary over an object's surface and are often stored in textures. Additional effects such as transmission or sub-surface scattering can also be modeled. Light sources can similarly be represented by analytical models or sample measurement-based profiles. Additional data such as environment panoramas can be used to help take into account all lighting contributions.

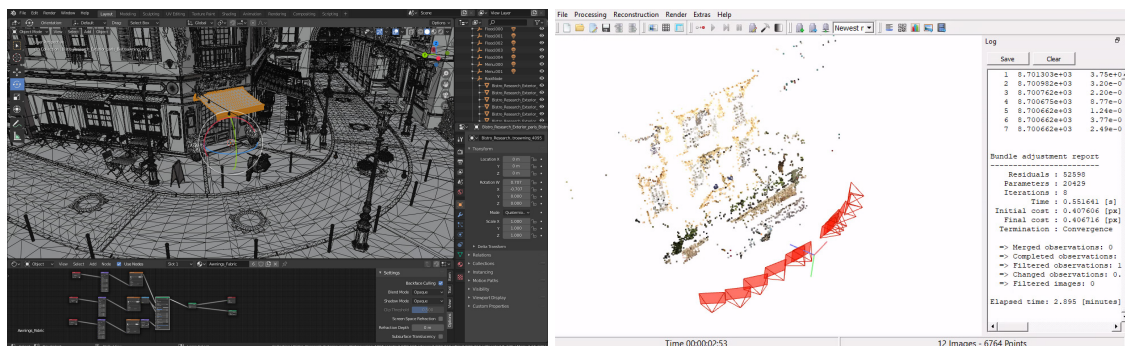


Figure 1.2 – Left: Screenshot of a synthetic scene opened in *Blender*³. Right: screenshot of the photogrammetry tool *Colmap*⁴.

All these elements can be created by artists in specialized software (Fig. 1.2, left), a task that requires a high level of skill and is extremely time consuming. This has a strong impact on creativity, the democratization of content creation and the widespread adoption of 3D graphics at large.

³Blender Foundation, <https://www.blender.org/>

⁴[SF16, SZPF16], <https://colmap.github.io/>

Capturing real-world elements through various acquisition devices is now a viable alternative used in production of films and games. BRDFs can be measured from material samples using adequate sensors or inferred from photographs. Photogrammetry can reconstruct the surface of an object from a set of pictures showing it under multiple angles (Fig. 1.2, right). But the output of these techniques requires tedious cleanup to be used in a real-time rendering engine since most solutions only provide approximate geometry and texture, with no information on materials and with lighting “baked into” the texture. Using this data for real-time rendering requires the artist to cleanup the errors in geometry, manually define material properties, and remove the lighting from textures, typically involving painstaking manual steps. Furthermore, most lighting models used for real-time have to simplify or even forego complex lighting effects to remain accessible on a wide range of hardware.

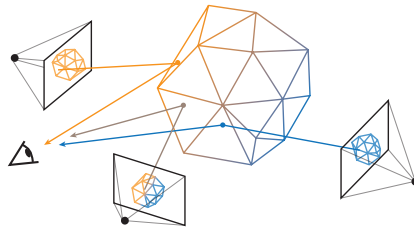


Figure 1.3 – Image-based rendering samples and reprojects information from a set of input views to the novel view, optionally using an intermediate scene representation.

Image-based rendering (IBR) pushes the idea of acquiring data from the real world to the extreme limit, by trying to capture and directly use lighting information. Instead of evaluating a simplified BRDF model, why not use the real-world shading that is visible in photographs of a scene? Multiple pictures of an object can be used to estimate not only the geometry, but also the appearance of the material under different viewpoints. This information can be aggregated and queried to generate a novel view (Fig. 1.3). Data does not have to come from the real world, and can instead be precomputed using offline techniques for synthetic content. It is thus possible to benefit from more accurate lighting than by only relying on real-time methods. Whichever data source is used, IBR replaces lighting evaluation by a sampling and reprojection task: input information has to be sampled and reprojected based on the user viewpoint. This can be a complex problem for effects such as reflections or transparency.

While the behavior of diffuse surfaces is now well handled by most rendering approaches, view-dependent effects such as reflections are more complex to synthesize accurately in real-time (Fig. 1.4). As they depend on the user's motion and involve surfaces that are potentially far from each other in the scene, they can vary at a high temporal and spatial frequency, whereas the human eye is very sensitive to them.



Figure 1.4 – Example of a synthetic scene. Left: only diffuse illumination is computed. Right: all material effects are rendered, greatly improving the scene realism.

1.3 Importance of view-dependent effects

View-dependent effects visible at the surface of non-diffuse objects are complex to render while being paramount to scene realism. Indeed, humans are extremely sensitive to the specular behavior of materials and can easily perceive inconsistencies [PFG00].

Reflections for instance are omnipresent in any real world scene but exhibit complex motion as soon as the surface or the material parameters are not simple (see Fig. 1.5). Such effects depend on the geometry and material of multiple surfaces in different parts of the scene (the reflector and reflected elements for instance). Multiple intersections have to be found and the lighting evaluated at each of these. For glossy effects, light incoming from multiple directions has to be taken into account, involving even more queries. Furthermore, compared to diffuse global contribution, these cannot be easily precomputed as they depend on the user viewpoint.

For image-based techniques, view-dependent effects have to be handled with specific sampling and reprojection approaches. As already pointed out, their motion or *flow* when



Figure 1.5 – Examples of real-world complex view-dependent behaviors in everyday life scenes: reflection, glossiness, refraction, transmission.

moving around is not the same as the one for diffuse geometry. Different flows have to be computed for different effects and their data has to be reprojected separately. For transparency effects, multiple surfaces contribute to the appearance visible at a given pixel, each with a different depth and motion. Inconsistent appearance also leads to artifacts in most capture processes that rely on a simplified diffuse assumption (Fig. 1.6). Finally the number of viewpoints required to capture high frequency effects on a given surface can increase storage requirements drastically. At the same time all materials do not exhibit effects at the same frequency, so space is wasted storing redundant information in some regions.



Figure 1.6 – Artifacts introduced by reflective and transparent surfaces: missing or erroneous geometry caused by specularly or transparency (orange), duplicated or blurry reflections caused by an erroneous flow (blue).

1.4 Contributions

The issues discussed in the previous sections motivated our research. In this thesis, we explore how to improve the rendering of view-dependent effects in scenes, both extracted from real world environments and created synthetically by artists. To reach high visual quality we leverage precomputation and thus use image-based techniques to reproject existing information in novel views. We believe that interactive visualization of realistic environments is a unique way to explore an original scene, whether it comes from the real world or an artist’s mind.

Throughout the three projects presented in this thesis, we have iteratively redefined our modeling of view-dependent effects depending on the type of scenes tackled. We thus present:

- Chapter 3: a novel technique for reconstruction and rendering of facades from a small set of input photographs. We exploit architectural *repetitions* to process the most important scene elements in isolation, even if they exhibit view-dependent behavior. By aggregating information from multiple elements, we are able to improve on the reconstruction and rendering, while detecting regions that exhibit specular effects. Additional reflection effects can then be added.
- Chapter 4: a second technique that processes reflective elements in isolation, this time focusing on street-level regular capture. Here again, working on elements separately allows us to repair geometry reconstruction artifacts caused by view-dependent effects. We build an analytic representation of reflective semi-transparent surfaces, using it to extract plausible reflection layers. This representation is also used at render time to compute the flow of reflections from the input views and sample the layers accordingly, generating plausible reflections in motion.
- Chapter 5: a third technique that, motivated by our previous reflection flow work, generates realistic mirror and glossy reflections in synthetic scenes. The perfect geometry is leveraged to warp precomputed specular information from a set of parameterized probes to the novel view. Information is gathered in the novel view, accumulating specular samples that are valid for the new location while respecting occlusions and materials. An image-space reconstruction filter ensures that glossy surfaces are properly rendered.

1.5 Funding and publications

This thesis has been funded by a PhD fellowship from the IT doctoral school of Université Côte d'Azur and the ERC Advanced Grant No. 788065 FUNGRAPH⁵. It has led to two publications in international venues ([RBDD18, RPHD20]) and a project currently under review ([RLP⁺20]).

⁵<http://fungraph.inria.fr>

Related work

As described in the introduction, this thesis focuses on the rendering of view-dependent effects in both real and synthetic environments, by relying on image-based methods, data precomputation and reprojection. We now introduce the concepts at the root of rendering and scene reconstruction. We then review the literature on image-based techniques, focusing on the real-time rendering of large scenes, and detail how view-dependent effects are extracted, reconstructed or synthesized in the literature, both for real and synthetic scenes.

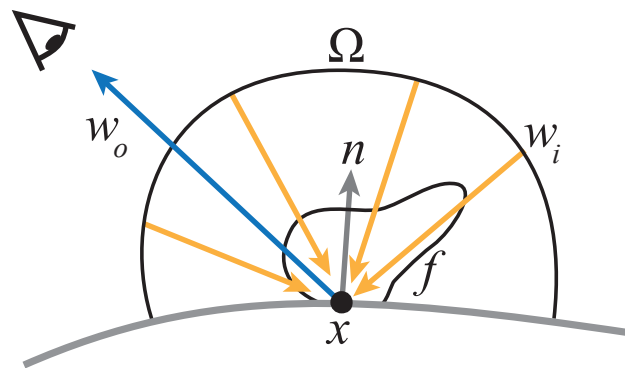


Figure 2.1 – The rendering equation estimates the radiance exiting a point on the surface of an object, taking into account the material properties and the radiance incident from all directions.

To realistically render a new viewpoint in a 3D scene, light propagation has to be simulated. This simulation should model the way light propagates from sources to the viewpoint, taking into account intermediate interactions with the materials present in the scene.

The *rendering equation* [Kaj86] describes such interaction at a point x on the surface of an object (see Fig. 2.1).

$$L_o(x, \omega_o, \lambda, t) = L_e(x, \lambda, t) + \int_{\Omega} f(x, \lambda, t, \omega_i, \omega_o) L_i(x, \omega_i, \lambda, t) (\omega \cdot n) d\omega_i$$

For a given outgoing light direction ω_o , the radiance L_o exiting the point is the sum of the emissive behavior of the surface $L_e(x)$ and the accumulated incoming radiance L_i from all directions ω_i covering the hemisphere Ω . The incoming radiance is modulated by the material based on its properties, described by a bidirectional reflectance distribution function (BRDF) [Nic65] f , and by a geometric term based on the normal n orientation. The rendering equation is given at a fixed time t and for a specific light wavelength λ . Properties of the material are often spatially-varying and stored in texture maps for convenience. Evaluating the rendering equation at a given point requires knowledge of the radiance at all points in the scene that might contribute to the incoming radiance for the current point. But these quantities might depend on the radiance we are currently evaluating. This makes solving the rendering equation for the whole visible scene intractable. Thus simplifying assumptions have to be put in place, especially for real-time rendering. The behavior of light emitters, of the BRDF and estimation of the incoming radiance to a point can be approximated using different techniques, for example analytical representations or precomputed tables.

The information required to represent and shade a scene can be created from scratch using specialized software, or be acquired in the real world. Capture relies on either specific sensors for BRDF acquisition or commercial handheld cameras. Multi-view stereo reconstruction techniques (MVS) combine information from a set of input RGB images to automatically estimate a surface mesh of the scene [GSC⁺07, SZPF16]. The input camera poses and intrinsic parameters are first estimated using structure-from-motion [SSS06]. Significant features are extracted from each input images using scale and rotation invariant descriptors [Low04, BTVG06]. They are matched between neighboring views and the depth of the corresponding pixels estimated through triangulation. This initial sparse point cloud is then used to refine camera estimation and compute dense depth maps. These are aggregated to form a dense point cloud, from which the mesh surface is reconstructed (see Fig. 2.2, left).

Erroneous feature matches introduce artifacts in the geometry, and are caused by view-dependent effects, transmissive surfaces that present multiple depths for a given pixel, or occlusion events that lead to noisy edges (Fig. 2.2, right). This adds extra constraints to the acquisition baseline, requiring redundant coverage between the input views.

For synthetic scenes, image-based techniques rely on the notion that high-quality shading

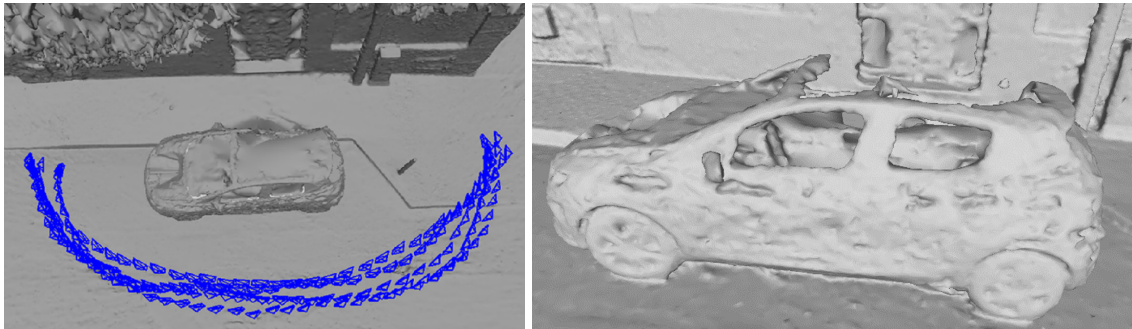


Figure 2.2 – Left: Calibrated cameras and scene geometry obtained by multi-view stereo reconstruction (using [SZPF16]). Right: The transparent window geometry is missing, as is the car roof, highly reflective at grazing angles.

for real-time rendering can be precomputed at the desired level of accuracy instead of relying on on-the-fly approximate evaluation. Image-based rendering furthermore allows for real-world lighting data to be used as-is, for increased realism at the cost of editability. Because the novel viewpoint is unknown at precomputation or acquisition time, data has to be queried and reprojected to generate the novel image. Effects that are viewpoint dependent thus require additional care as their motion can be hard to estimate. On the other hand, image-based representations of the scene can be used to synthesize more accurate visual effects in real-time, including reflection and refraction that are particularly important to achieve high levels of realism. This thesis focuses on such techniques, both in synthetic and real environments.

We now present a review of existing approaches in both image-based rendering and view-dependent effect synthesis. We focus on interactive rendering of complete scenes.

2.1 Image-based rendering

2.1.1 General approaches

Plenoptic function. Image-based techniques rely on the pre-existence of scene information such as shading and global illumination effects that would be expensive to compute at runtime. In an ideal world, we would be able to store such information for all directions, at all locations. More formally, the appearance visible from a point of view in the scene, for a given direction is given by the plenoptic function [MB95] $P(x, (\theta, \phi), \lambda, t)$ where x is the view 3D position and (θ, ϕ) the normalized direction, λ the wavelength and t time

(see Fig. 2.3).

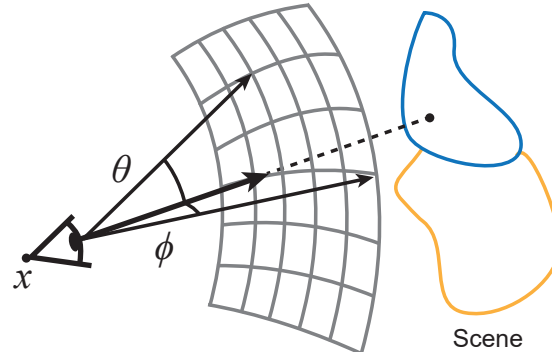


Figure 2.3 – The plenoptic function parameterizes the scene appearance observed from a given viewpoint and direction as a 5-dimensional quantity (x, θ, ϕ) – at fixed time and wavelength.

A perfect image of the scene from a given viewpoint could be generated if we had an oracle that could return P for any set of input parameters. In our applications, we consider temporally static scenes and use the standard collapse from the full spectrum to a triplet of red, green, blue values. In practice, the plenoptic function of the scene can only be sparsely sampled due to acquisition, computation and storage constraints [CTCS00]. When generating a novel viewpoint of the scene, existing samples have to be selected and blended to approximate the missing plenoptic information. Different resampling and completion approaches put different requirements on the quality and density of the initial data acquisition. One has to ensure that enough information is captured to allow for free-form exploration of the scene while limiting the storage usage for interactive applications. Furthermore, resampling might be simplified if additional scene information is known, for instance geometry.

Panoramas. Historically, one of the first examples of image-based rendering is the generation of interactive 360-degree panoramas and turntables [Che95]. In these setups, the user position is either fixed or rotating around a fixed point of interest. For this fixed location, a set of images covering all possible view angles is captured and stitched together (see Fig. 2.4). The plenoptic function is thus densely sampled in all directions (θ, ϕ) for a fixed x . The plenoptic data can originate from real world photographs or synthetic images computed with accurate non-interactive methods. Panoramas can be

created at different locations and linked together, allowing for interactive exploration of an environment by “jumping” from a panorama to another when clicking on predefined areas of interest. Shade et al. [SLS⁺96] subdivide a synthetic scene into a hierarchy of nodes; renderings of each node content can be cached for re-use while the user explores the environment, by displaying them on quads in world space.



Figure 2.4 – Multiple images taken at the same location can be combined to generate a panorama. The user can freely rotate around to visualize the scene.

Light fields. To allow for more freedom in viewer motions while still limiting the dimensionality of the problem, the Lumigraph [GGSC96] and Light field [LH96] techniques designed a custom parameterization of the plenoptic function. They define a region of the scene contained between two parallel planes with limited extent (Fig. 2.5). Rays of light that go through this region while intersecting both planes can be parameterized by their intersections locations as two sets of 2D coordinates. This parameterization has fewer dimensions than the full plenoptic function while allowing for the generation of views with different orientations.

In the case of the Light field, images are captured on a regular grid on one of the two planes, with the object of interest behind the other plane. Generating a novel view light ray is then a matter of linearly interpolating between the nearby input rays based on the two sets of planar coordinates of the new ray. The Lumigraph supports unstructured capture, but requires an additional resampling step to extract the light rays from the input images. Both methods use high-density input data, allowing for the reconstruction of complex view-dependent effects at the cost of high storage. In cases where the plenoptic function cannot be sampled as densely, existing samples have to be warped and propagated to generate novel viewpoints. For sparser setups it becomes necessary to have some additional scene information available, such as depth, defined either locally for each input view or globally for the scene.

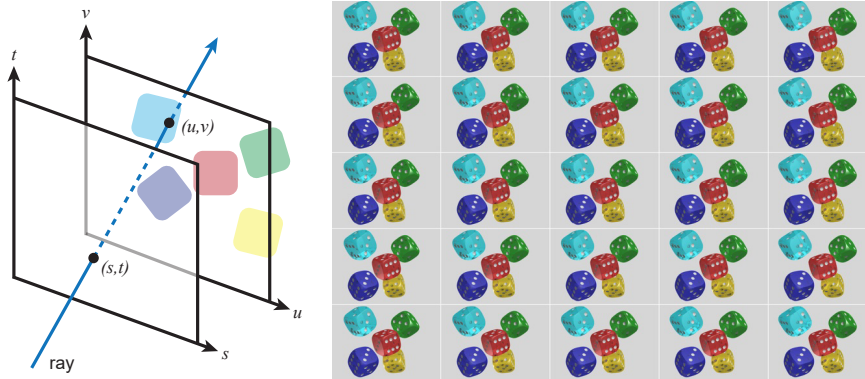


Figure 2.5 – The light field parameterization reduces the plenoptic function to a 4 dimensional quantity for all rays intersecting a focal and a sensor planes. Left: example of a regular grid light field capture.

2.1.2 Geometric scene representation

Scene geometry information can be used to reproject plenoptic samples from an input view to the novel view, while taking into account occlusions and simplifying rendering. Acquisition setups that use unstructured 2D images as input are compatible with camera pose estimation, as well as depth and surface reconstruction techniques from computer vision. Care needs to be taken when handling complex surfaces with reflective or transmissive elements, and when merging information from different input views.

Global scene geometry An approximate scene representation can be provided by the user through the use of interactive modeling software. Texture information is then reprojected from the input images onto the geometry [DTM96]. When multiple images reproject on the same region, they can either be consistent if the surface is diffuse, or contain variations caused by view-dependent effects. To preserve this information, texture data is blended between the input views closest to the novel viewpoint. Multi-directional color information can be stored on each face of the geometry [DYB98] and blended at runtime using graphics hardware, a process called view-dependent texture mapping (Fig. 2.6, left).

Heigl et al. [HKP⁺99] approximate the scene by a unique mean plane onto which input color information is reprojected and blended based on the novel view location. Buehler et al. [BBM⁺01] use a more complex scene representation. They tessellate the novel view image plane into a grid of vertices, combined with a reprojected partial scene

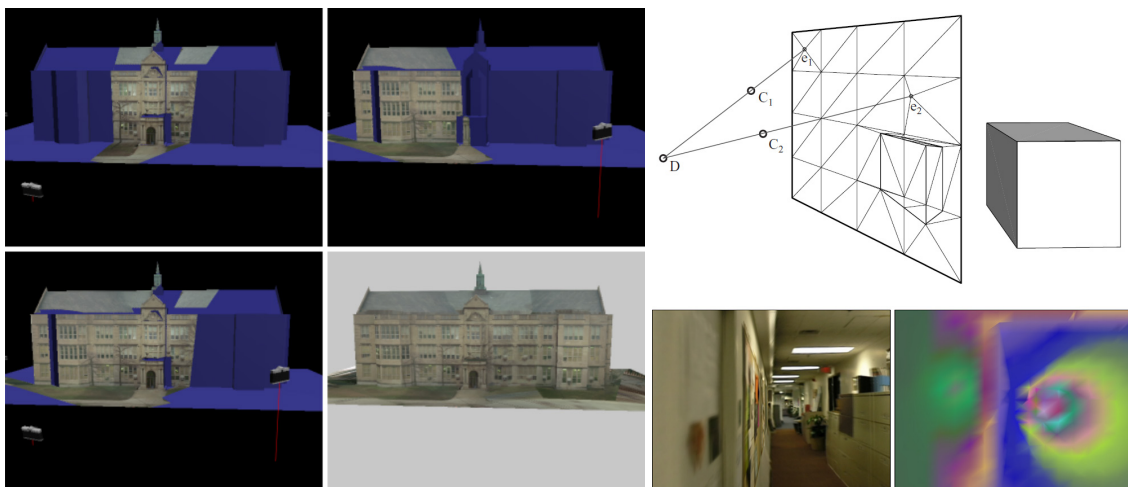


Figure 2.6 – Left: view-dependent texture mapping project the input images as textures onto the scene geometry. Right: The Unstructured Lumigraph use a screen-space scene tessellation (top right) as a support to evaluate blending weights between input images (bottom right) in the novel view (bottom left).

reconstruction (Fig. 2.6, right). At each vertex, blending weights between nearby input views can be estimated based on their relative distance and orientation with respect to the novel view currently generated. The smooth camera blending field obtained is then used to aggregate input image color information.

When the reprojection and blending process used to generate the novel view is known in advance, it is possible to guide the user during the capture process [DLD12] to ensure optimal quality of the result. Approximations in the global geometry lead to incorrect reprojections from the input images to the novel view. Floating textures [EDDM⁺08] compensate these by precomputing the optical flow error obtained when reprojecting an input image into another input view. These flow fields are queried at runtime to correct the erroneous warping introduced by the geometry.

While a global mesh simplifies the rendering process by making it amenable to regular graphics hardware, extra geometric accuracy can be gained by skipping the final aggregation step and relying on per-view depth information. Indeed, the consensus required to generate global geometry often leads to discarding per-view accurate information when it is inconsistent between views.

Per-view geometry. Because global geometry is multi-view consistent at the cost of

accuracy at each input view, several techniques have explored per-input view depth or geometric representations. Per-view depth maps can be used directly to reproject and select plenoptic samples; depending on the performance required, these depth maps can be discretized and simplified to a set of local planes [HKP⁺99].



Figure 2.7 – Left: Input image segmented in almost-constant depth super-pixels based on color information. Right: In [CDSHD13] 3D patches generated from the input views super-pixels are warped to generate the novel view.

Zitnick et al. [ZKU⁺04] segment input images extracted from videos into super-pixels of expected constant depth (example of segmentation in Fig 2.7). The depth of each region is refined and care is taken at discontinuities to separate foreground and background color information. The depth information is used to interpolate between the input images. By accumulating reprojected information from multiple super-pixels, regions occluded in one input view can be filled in by another view’s super-pixels. Additional effects such as depth of field or more complex camera motions can also be reproduced [ZCA⁺09]. Chaurasia et al. [CDSHD13] also estimate and refine depth for super-pixels in the input views, but additionally build geometric patches for each region (Fig 2.7). These patches can be warped and blended depending on the novel view location, while preserving the shape of the surfaces covered by each patch.

Inside-out [HRDB16] combines RGBD input views with reconstructed global geometry to generate refined per-view meshes (Fig. 2.8). These are then sliced on a regular grid covering the scene. At runtime, slices of the per-view meshes are selected independently for each grid region, depending on the viewpoint location. The input texture data associated to the different selected slices is blended together with a fuzzy depth test to generate the final view.

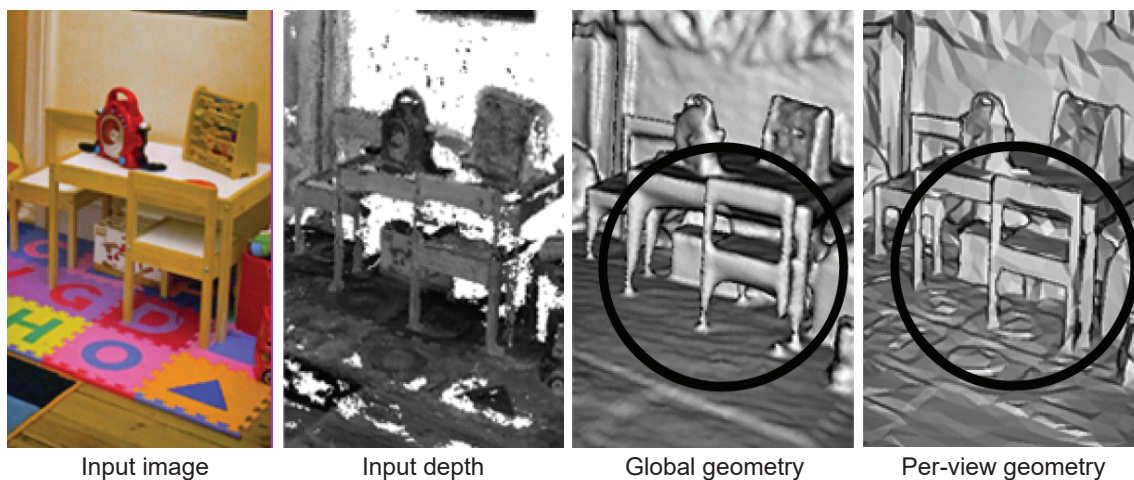


Figure 2.8 – Per-view meshes used in [HRDB16] better capture small details of the scene geometry that are lost when generating a global consensus mesh.

Such surface representations enforce a single depth per input-view pixel, which is incompatible with transparent surfaces and partial pixel coverage at discontinuities. Layered-depth images [SGHS98] decompose each input view into a set of partial images at different depths (Fig. 2.9). Each layer is warped to the novel view and alpha-blended back-to-front. No additional depth testing is required, as regions that correspond to opaque foreground objects will automatically occlude any background layer.

More recently, Tulsiani et al. [TTS18] generate layered depth images from a pair of calibrated input views using a neural network trained by comparing novel viewpoints generated from the predicted layers with a reference image. Mildenhall et al. [MSOC⁺19] apply a similar technique to a set of input views placed on a regular grid. A set of planes containing color and opacity information is extracted from each view. The planes of the neighboring input views are warped to the novel view and blended using bilinear weights.

Layered representations require more storage but fit the classical alpha compositing pipeline where successive layers are stacked and their colors blended based on the associated opacity values. For scenes with intricate geometry or complex multi-layered effects, many layers are required, and some approaches have opted for a fully volumetric representation.

Volumetric representation. Volumetric representations extend the layered multi-plane

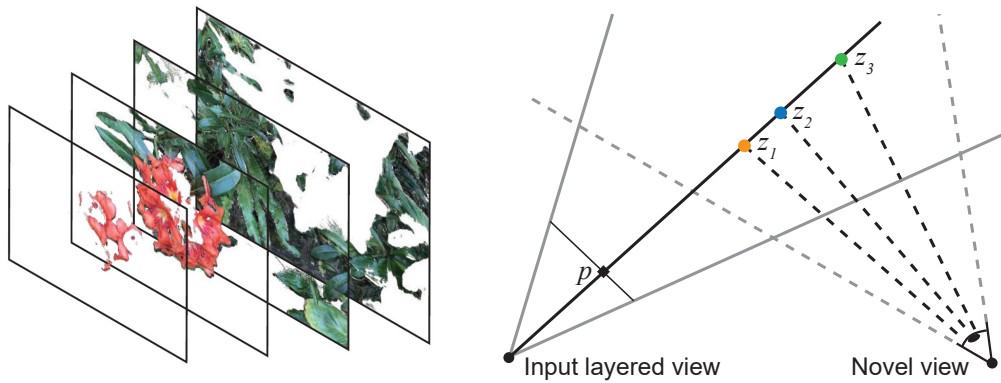


Figure 2.9 – Left: Layered depth images can store multiple (color, depth) samples for a given pixel (example from [MSOC⁺19]). Right: When rendering the novel view, layers stored for pixel p are at different depths z_i and reproject at different locations in the new image.

image approaches by slicing the scene in a 3D voxel grid, each containing information related to the corresponding region of space. They offer the possibility of storing non-binary information in a more dense fashion than discretized surface representations.

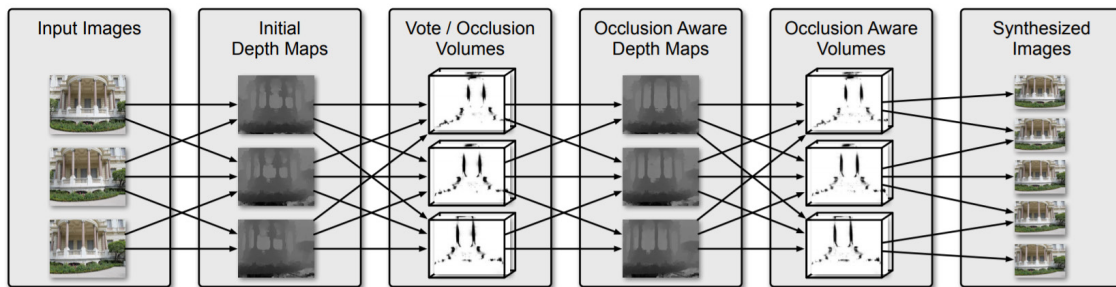


Figure 2.10 – Volume generation process described in [PZ17], taking into account occlusions and soft visibility.

Soft3D [PZ17] generate a per-input view volumetric representation on the scene, estimating color and occupancy for each voxel of a grid aligned with the input view frustum. The estimation is performed using MVS without collapsing to a final surface representation. When rendering a novel view, nearby input volumes are warped to the novel view volume and aggregated based on the relative confidence in each voxel data to build a consensus representation (Fig. 2.10). For each pixel, a ray is cast through the volumes, accumulating color information until the occupancy reaches saturation.

Instead of storing explicit information such as color and occupancy, recent techniques instead propose to store features extracted by a neural network [STH⁺19]. When generating a novel view, the features volume can be warped similarly to Soft3D, before being fed to a rendering neural network. End-to-end training ensures that the network learns to generate plausible interpolated views (Fig. 2.11a). Alternatively, learned features can be used as a compressed representation of the volumetric scene. Lombardi et al. [LSS⁺19] extract a compact feature vector from a multi-view dataset. The features are used as input to a decoder that regenerates a regular volumetric representation from the novel viewpoint (Fig. 2.11b).

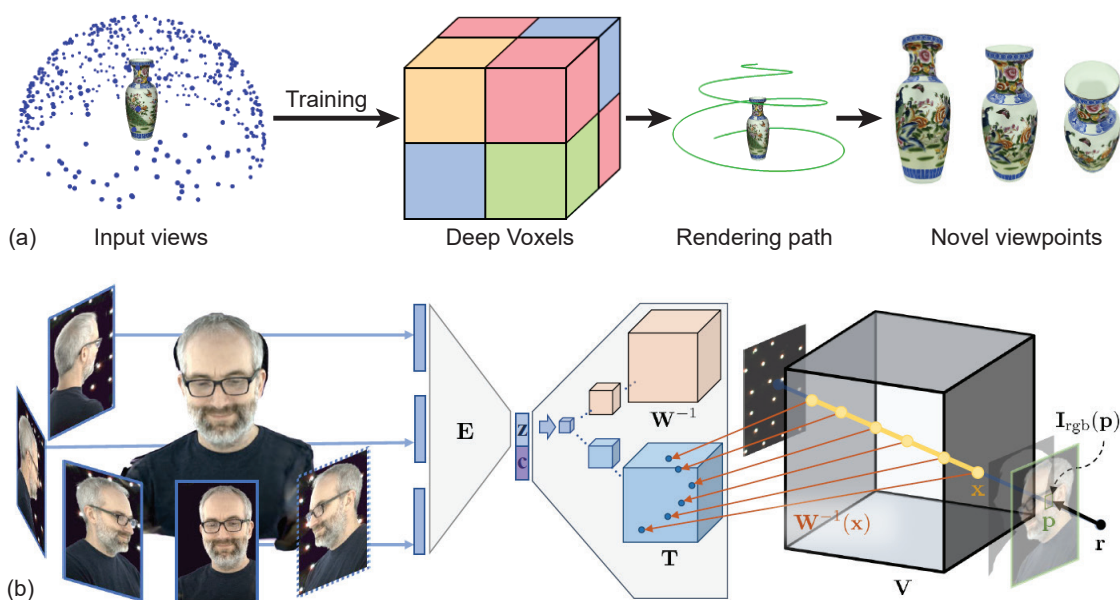


Figure 2.11 – (a) Each voxel of the volumetric representation described in [STH⁺19] stores features extracted from the corresponding scene region. The feature volume can be resampled and decoded to generate the novel view. (b) In [LSS⁺19], a compressed representation of the scene is learned and can be used to regenerate a full volumetric grid containing color and occlusion information.

As described in this section, image-based rendering techniques rely on a set of different representations with varying space/rendering trade-offs. The warping and blending applied to the plenoptic samples is usually tied to the scene representation, as some part of the reconstruction can be performed at blending time. Dense representations suffer from high storage requirement when treating large scenes, but surfaces are not always able to represent all types of objects encountered. Per-view geometry captures finer

object details at the cost of global geometric consistency. Because scene reconstruction is paramount to most image-based rendering techniques, our work also relies heavily on computer vision solutions. In the following chapters we will focus on sparse input images (chapter 3) or large scenes (chapter 4), and will try to address some of the shortcomings of surface representations when it comes to view-dependent effects.

2.1.3 Blending techniques

Blending information from multiple input views to generate a novel image while preserving view-dependent effects is a complex problem. Multiple effects need to be accounted for and the blending techniques often rely extensively on the underlying scene representation.

Geometric measures and optimization. Once plenoptic samples are selected or transferred to the novel view, it has to be blended depending on the significance/contribution of each input view for the novel view. Blending weights can be computed based on geometric criteria. The input view positions form a tessellation of the explorable space; some approaches thus resort to interpolating between the closest input views using barycentric interpolation [LH96, DLD12, MSOC⁺19]. Other methods offer more complex weighting schemes designed to measure the similarity between two views [DYB98, BBM⁺01]. For instance, the Unstructured Lumigraph [BBM⁺01] computes a per-vertex, per-camera weight that takes into account the relative distance of both input and novel views to the vertex, their relative orientations and the projection of the vertex in the image plane (Fig. 2.12).



Figure 2.12 – The ULR blending weights at a point on the surface take into account the distances from that point to the input view d_i and novel view d_n , along with their relative incidence angle θ and the location in the input image space. The best cameras are ranked and their scores normalized.

For offline rendering, optimization based techniques achieve seamless blending with temporal stability. Hyperlapse [KCS14] generates the final image using a graph-cut on the set of close input images while trying to minimize texture distortions (Fig. 2.13). A more formal framework is proposed by Pujades et al. [PDG14], where both texture deformation and the uncertainty introduced by the data resolution and geometry reconstruction are taken into account. They are able to re-derive the weights used by view-dependent texture mapping and the unstructured lumigraph.

Solving the general optimization problem remains infeasible in real-time, but these approaches provide a firm grounding upon which other blending schemes can be validated.

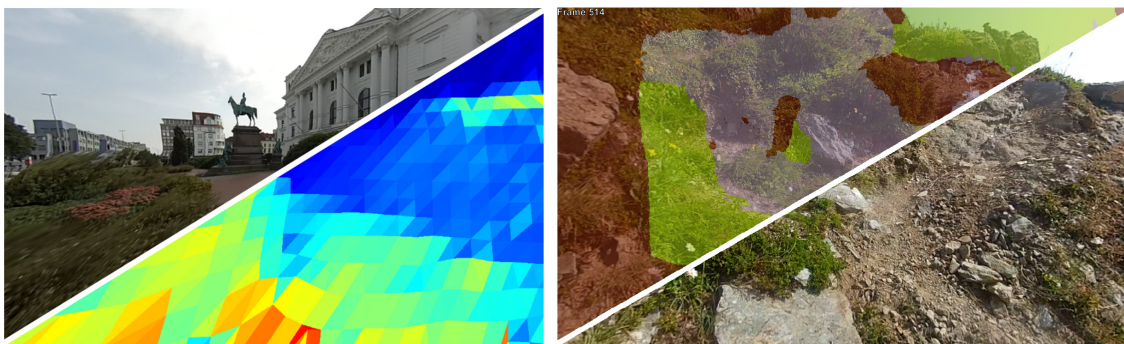


Figure 2.13 – Hyperlapse: Left: A texture deformation measure is used to compute blending weights. Right: Selected images are stitched together by solving a labeling problem offline.

Neural blending and rendering. Because the final blending is frequently performed in the novel view image space, it can be cast as a convolutional neural network task. DeepBlending [HPP⁺18] builds on InsideOut [HRDB16] by performing blending between selected candidate samples using a compact neural network (Fig. 2.14). As the network is trained on multiple scenes, it generalizes well to unseen datasets.

As mentioned in the previous section, learning-based volumetric methods usually have a strong coupling between the representation and blending. Deep Voxels [STH⁺19] generate the final image using a U-Net neural network that combines the reprojected feature volume with the output of a secondary network estimating occlusions. Thies et al. [TZN19] design a system where the scene appearance is learned as a set of features parameterized in texture space (Fig. 2.15). UV coordinates are rendered for the novel view and the *neural texture* is queried at those coordinates to obtain the final appearance.

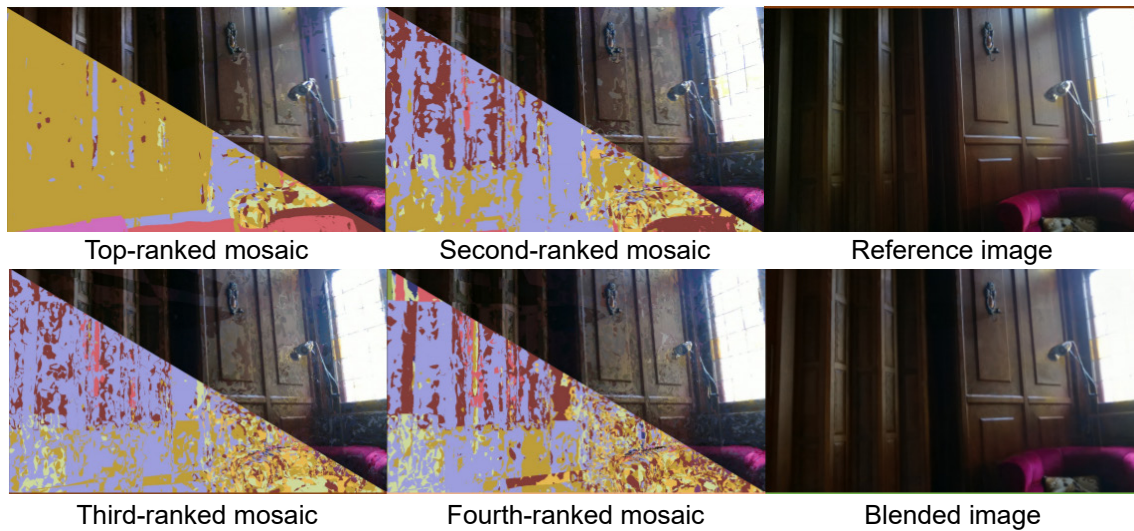


Figure 2.14 – DeepBlending [HPP⁺18] selects four candidate samples for each novel view pixel. Those colors are blended using weights predicted by a scene-agnostic network trained in a leave-one-out fashion.

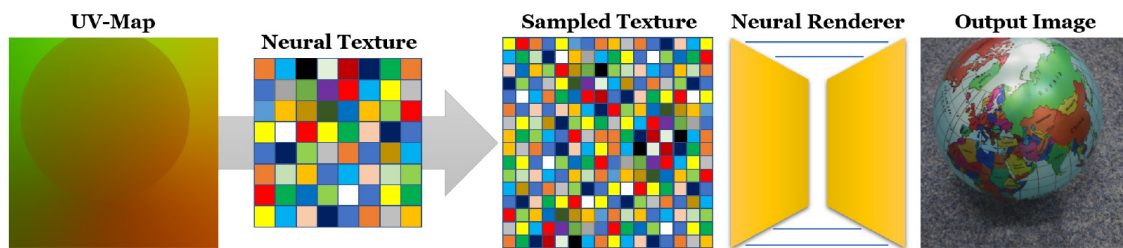


Figure 2.15 – [TZN19] build a texture space scene representation called a neural texture. Scene UV coordinates visible in the novel view are used to sample that texture and pass it to a network; the texture features are decoded to obtain the final color view.

NERF [MST⁺20] replaces the texture coordinates by a 3D space parameterization: a network is trained to predict the radiance and occupancy at any given location in the scene volume (Fig. 2.16). Radiance is queried regularly along view rays and accumulated until occupancy saturates. In practice a two-level representation is used: a coarse network is used for initial fast queries and a second, deeper network is evaluated to query finer details.

Techniques that build an internal representation specific to the scene are often limited in the size and complexity of the scene that they can capture, while requiring a high number of input images (from hundreds to thousands). In chapters 3 and 4, we focus on

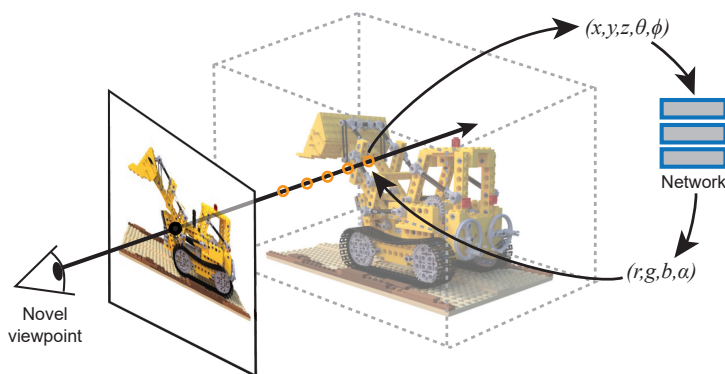


Figure 2.16 – NeRF train a network to predict the radiance for any 3D location and direction in the scene volume. The final view is generated by raymarching through this volume.

scenes that are either large scale or captured with a very sparse baseline, and we thus resort to using surface-based geometry combined with image-space blending weights.

2.1.4 Synthetic data

The possibility of generating new viewpoints in a scene or on an object by combining existing plenoptic samples also has advantages for rendering synthetic scenes created by artists. Shading computation can be decoupled from novel view generation and thus rely on more expensive approaches. Samples can also be temporally accumulated. Precomputed radiance can be queried to generate reflection effects more easily. Finally, image-based rendering enables high quality interactive rendering of complex effects or rendering on constrained hardware, as remote or precomputed information can be warped to a novel viewpoint on the fly. In chapter 5, we present a novel technique for realistic rendering of high-quality specular and glossy surfaces, motivated by the applications of image-based rendering to synthetic scenes.

Sample reuse. In early works existing radiance samples can be reused to generate a novel viewpoint on the scene. The Holodeck [WS99] renders rays carrying radiance and store them in a data structure; it is then queried to generate the novel view, by selecting the rays closest to those needed for the novel image. The Render Cache [WDP99] accumulates ray-traced samples that are reprojected in the novel view in a depth-aware fashion. Information is interpolated between samples to generate the final image (Fig. 2.17). New

samples are requested to cover empty regions of the novel view. Frameless rendering [BFMZ94, DWWL05] similarly reconstruct the final image from reprojected temporal samples. Here, the ray-traced samples are stored in a set of 2D partial images. The temporal blending parameters are adjusted based on the presence of dynamic elements in the scene.

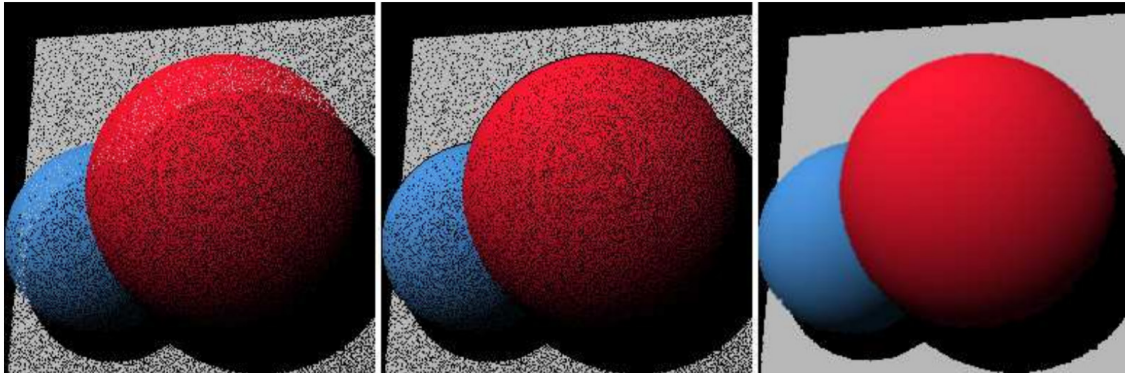


Figure 2.17 – In the Render cache[WDP99], samples are reprojected in the novel view (left). Proper depth ordering is enforced (center) and samples information is propagated in empty regions (right).

Instead of projecting samples from a cache to the novel view, the problem can be turned in a gathering task, where scene points visible in the novel view are reprojected in the previous frame rendering [NSL⁺07]. The validity of cached samples is evaluated based on disocclusions. Temporal accumulation of radiance samples from previous frames can also be used for denoising and anti-aliasing [Kar14, SKW⁺17, YLS20].

Reflection rendering. Reflections are view-dependent effects that exhibit complex non-linear behavior under motion. When shading a reflective surface, elements that are reflected onto it have to be found in the scene and their radiance estimated. This is difficult to perform accurately and interactively in rasterization-based frameworks.

Environment maps store the scene colors visible in all directions from a given location and can be fetched to obtain the radiance incoming to a reflective surface in a given direction (see next section for more details). Lischinski et al. [LR98] extend this concept by storing environment radiance as a set of layered depth images and auxiliary light fields for view-dependent effects. A similar approach replacing a regular environment cube-map by six light-fields covering the full sphere is also found in the literature [YYM05] (Fig. 2.18).

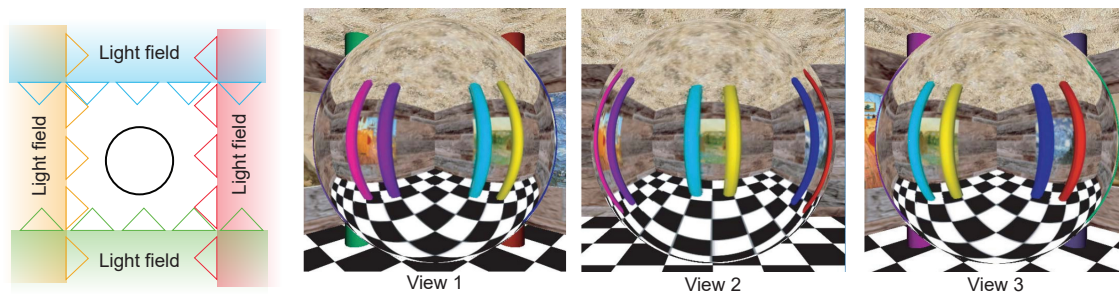


Figure 2.18 – [YYM05] surround the environment by light fields on the six faces of a cube surrounding the object. Parallax in the reflections is properly reproduced.

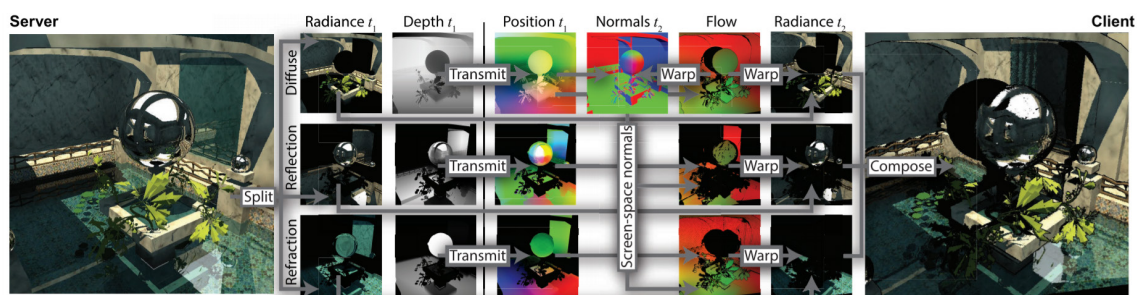


Figure 2.19 – [LRR⁺14] warps a high quality frame to a new viewpoint. Separate optical flows for diffuse, reflection and refraction effects are computed before combining them.

A temporal reuse technique accounting for reflection and refraction is described by Lochmann et al. [LRR⁺14], where previous frames information – streamed from a server – is warped separately following the flow of diffuse, reflective and refractive surfaces (Fig. 2.19). For refraction and reflection, the optical flow is estimated using an iterative search in the previous frame image space.

Image-based techniques allow for realistic rendering of reflections in synthetic scenes, by either acting as an alternate scene representation that can be queried when estimating the radiance incoming to a point, or by providing efficient ways of reprojecting information from existing views using the proper reflection flow. In chapters 4 and 5, we propose two ways of generating reflections into a novel view by relying on existing specular information and specific geometry representations that make the real-time flow estimation feasible.

High quality interactive rendering. Beyond reflections, precomputed rendering of parts of a scene can be used to lower the cost of rendering intricate shaded geometry.

Impostors [MS95, DDS03] combine one or multiple renderings of a given object with an extremely simplified supporting geometry. Many distant objects can then be rendered at a low runtime cost. This is especially useful for vegetation, clouds and buildings.

Image-based rendering has recently been applied to remote rendering, where high quality images are generated on a remote server and streamed to a device. A local two-view IBR can be performed to allow user motion even when latency arises [RKR⁺16] (Fig. 2.20). This requires the remote server to send scene geometry and an auxiliary rendered view and remains mainly limited to diffuse surfaces. Lall et al. [LBR⁺18] precompute a mainly diffuse representation of a static scene for small motion VR exploration. From a high quality path traced panorama and scene geometry, a set of floating patches is optimized to best capture the occlusion and parallax effects in the scene while minimizing overlap and storing costs. Shading information is projected onto those patches in order to render them as simple textured meshes (Fig. 2.20, second row).

Finally, high quality video light-fields can be precomputed as a set of RGBD environment maps and compressed for VR streaming [KKSM17], but the density requirement limits possible motion in the scene, and a complex decompression scheme is required due to the amount of per-frame data.

As shown in this section, image-based rendering concepts can be applied in many ways to improve the appearance of synthetic scenes. This motivates our approach described in chapter 5 to render accurate reflections in synthetic scenes, by warping precomputed glossy information from a set of probes to the novel view.

Overall, image-based techniques provide a diverse set of solutions for multiple rendering tasks on real and synthetic scenes. Different scene representations allow different types of environments and effects to be rendered. Yet complex indirect behavior such as reflection and refraction on surfaces remain difficult to both capture and synthesize for all types of scenes.

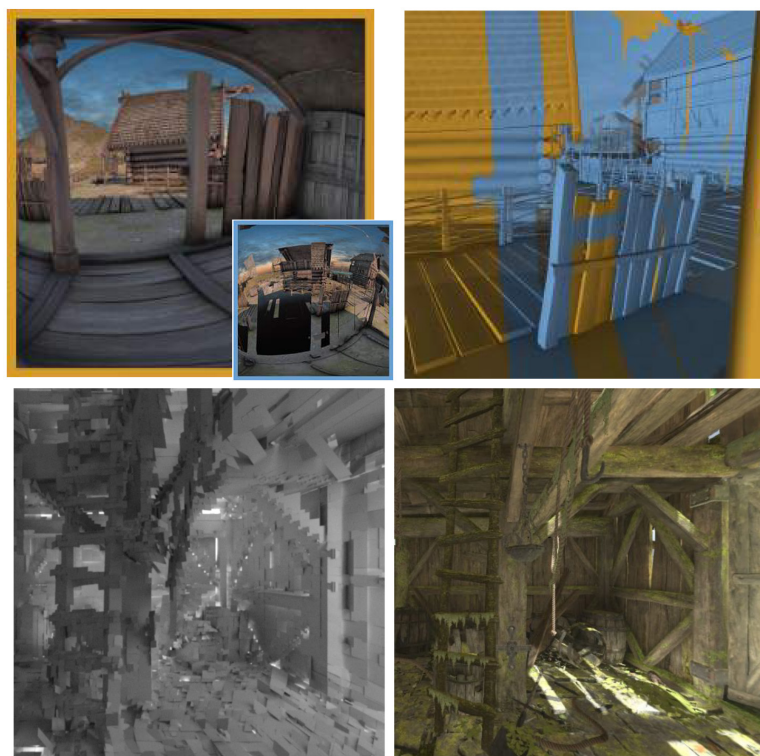


Figure 2.20 – First row: [RKR⁺16] combine a main view (orange) and an auxiliary view (blue, inset) to perform local IBR, supporting user motions even in the case of latency. Second row: [LBR⁺18] generate a patch-based representation of a high-quality panoramic rendering for VR exploration.

2.2 Extraction and rendering of view-dependent effects

As pointed out in the previous section, image-base techniques are able to reproduce complex view-dependent effects when the input capture is extremely dense. At the same time, these effects are known to be extremely adversarial to traditional capture and rendering approaches. The non-consistency of reflection and refraction when moving between viewpoints makes classical surface reconstruction often fail on specular or transparent surfaces. Furthermore, the apparent motion on such surfaces cannot be explained by a unique depth; for reflections the location of the reflected object matters, as does the surface behind the current object for refraction and transmission. Each category of effects will have its separate motion *flow*, different from the geometric flow of diffuse surfaces. These effects are also very sensitive to the geometry of the surface, where complex motions appear on non-flat objects. Finally, the appearance on such a surface is

the result of multi-layer compositing, i.e. mixing the contributions of the surface itself and the reflected and/or transmitted surfaces. In real world scenes, the contributions have to be extracted from the fused input data, while for synthetic scenes this separation is often used to apply specific rendering algorithms for each contribution.

2.2.1 Extracting view-dependent effects from real world data

The appearance of reflective and transmissive surfaces results from the composition of multiple effects, each with its own radiance and geometric information. For techniques such as the light field and lumigraph, the sheer density of the capture allows those effects to be interpolated as a whole without having to explicitly compute their flow. For sparser setups, specific approaches are required for each effect in order to compensate for the lack of information; estimating and recreating each effect separately is then a more viable solution. We now present a brief review of radiance and surface extraction techniques, and build on these to render reflective transparent surfaces in chapter 4.

Layer separation. Separating reflective and transmissive layers from the background in real world images is widely studied in computer vision and graphics [SAA00]. Because the merging of multiple sources of information at a single pixel is a lossy process, solving the problem for a unique image often requires user information [LW07] or extra capture devices such as polarized filters [WGGK18]. Multi-view setups provide more information to help disambiguate the accumulated contributions. For each layer to extract, the motion flow of the effect between input views has to be estimated so that correspondences can be established. Based on the color variations between matched pixels, the layer color information can then be separated. In practice, multiple iterations of flow estimation and color separation are required [XRLF15] (Fig. 2.21).

The flow that should be estimated for layer separation is strongly linked to the underlying geometry. This is why it can be beneficial to estimate the depth of each layer first. Sinha et al. [SKG⁺12] estimate up to two depths per pixel of each input view using a custom stereo matching technique (Fig. 2.22). The obtained depth maps are then used to build two geometric proxies of the scene, each composed of a set of local 3D planes. One proxy encompasses surfaces directly visible (reflectors and diffuse elements), the second one contains the geometry visible through transmission. Layers are then extracted using a constrained approach similar to the one described by Szeliski et al. [SAA00], but using the geometries to estimate accurate flows. An additional regularization ensures robustness

to noise and misalignments in the input views. To render the novel view, information from each layer is reprojected using the corresponding proxy. Note that for the reflection layer, the depths estimated are *virtual depths* of the reflected objects that produce the proper parallax motion when moving around.

Instead of explicitly separating layers, Kopf et al. [KLS⁺13] interpret view-dependent effects as motions of the input images gradients. Depth is only estimated for any input view pixel with non-zero color gradients. Because those are sparse, each gradient implicitly comes from a unique layer, even if it is not modeled. Gradients and their depths will be used to generate the novel view (see next subsection).

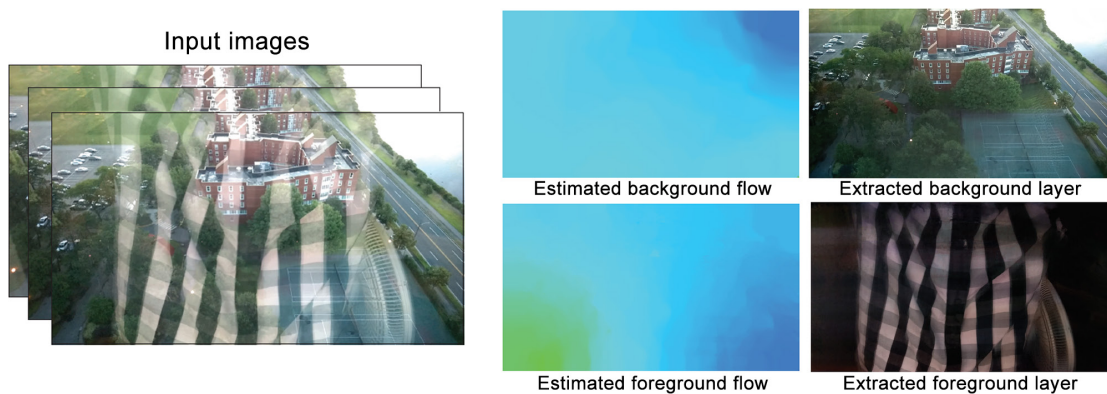


Figure 2.21 – Occlusion-free photography [XRLF15] extract a background and foreground layer by iteratively estimating their optical flows and separating their colors.

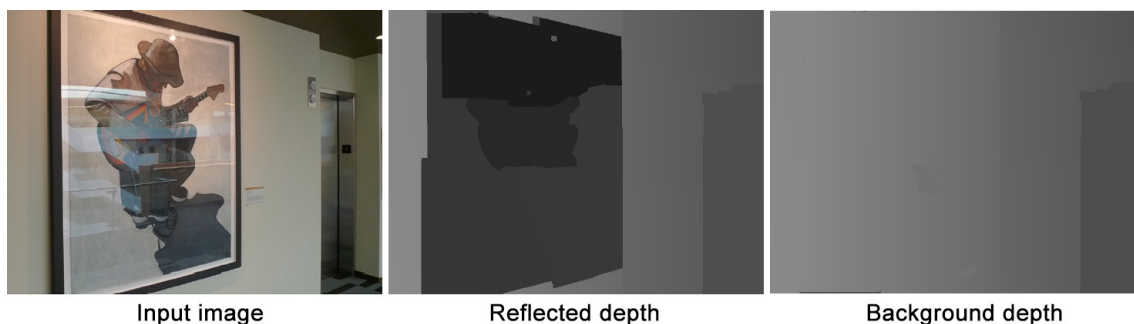


Figure 2.22 – In [SKG⁺12], up to two depths are estimated at each pixel using a custom MVS algorithm. One is associated to the directly visible elements, the other locates the reflected scenery.

Surface extraction. Alternatively, geometric reconstruction of transparent and reflective objects can be handled using specific capture setups [WGL⁺18, WZQ⁺18]. Godard et al. [GHLB15] iteratively refine the estimation of a reflective object surface. At each step, they re-generate reflections synthetically using the current estimation, and compare them to the input images. The error in the generated reflections guides the estimated surface adjustments. Volumetric approaches can also be found in the literature [IKL⁺10] to handle the multiple per-pixel depths in a more continuous fashion, but these are often limited to specific set of objects or rely on complex acquisition hardware [IKL⁺08]. Surfaces recovered by these techniques only represent the reflectors, and do not provide enough information for a full reflection reprojection.

2.2.2 Warping specular effects

If view-dependent effects have been extracted or synthesized from input data, they are typically located in the input views and have to be warped into the novel view. To properly perform this reprojection, the flow of reflection and refraction effects has to be computed. Even if the layer color and geometric information were perfect – as is the case for synthetic scenes – establishing correspondences between the novel view and the input views is a non-invertible problem for view-dependent effects in the general case. Indeed, while for planar reflectors it can be solved by reprojecting the virtual image of reflected points into the novel view, it becomes intractable for curved reflectors, even as simple as a sphere.

Blending reflection and transmission effects. If geometric layers with depth information have been extracted [SKG⁺12], they can be used to achieve reprojection. Indeed, the depth captured by each layer reproduces reflected or transmitted parallax under motion around the input views. Warping and blending is performed for each layer type separately, along with a layer containing the blending factors between other layers. A final hole-filling and compositing step generates the final image (Fig. 2.23, first row).

In Gradient image-based rendering [KLS⁺13], gradients are splatted into the novel view. Recall that the depth corresponding to each gradient has been estimated. Visibility changes are detected, and occluded gradients discarded. The projected shape of each gradient input view pixel is also taken into account during warping for accuracy. In parallel, an initial approximate color solution is generated using input colors, by applying each gradient onto the region of the image it spans when moving from the input to the

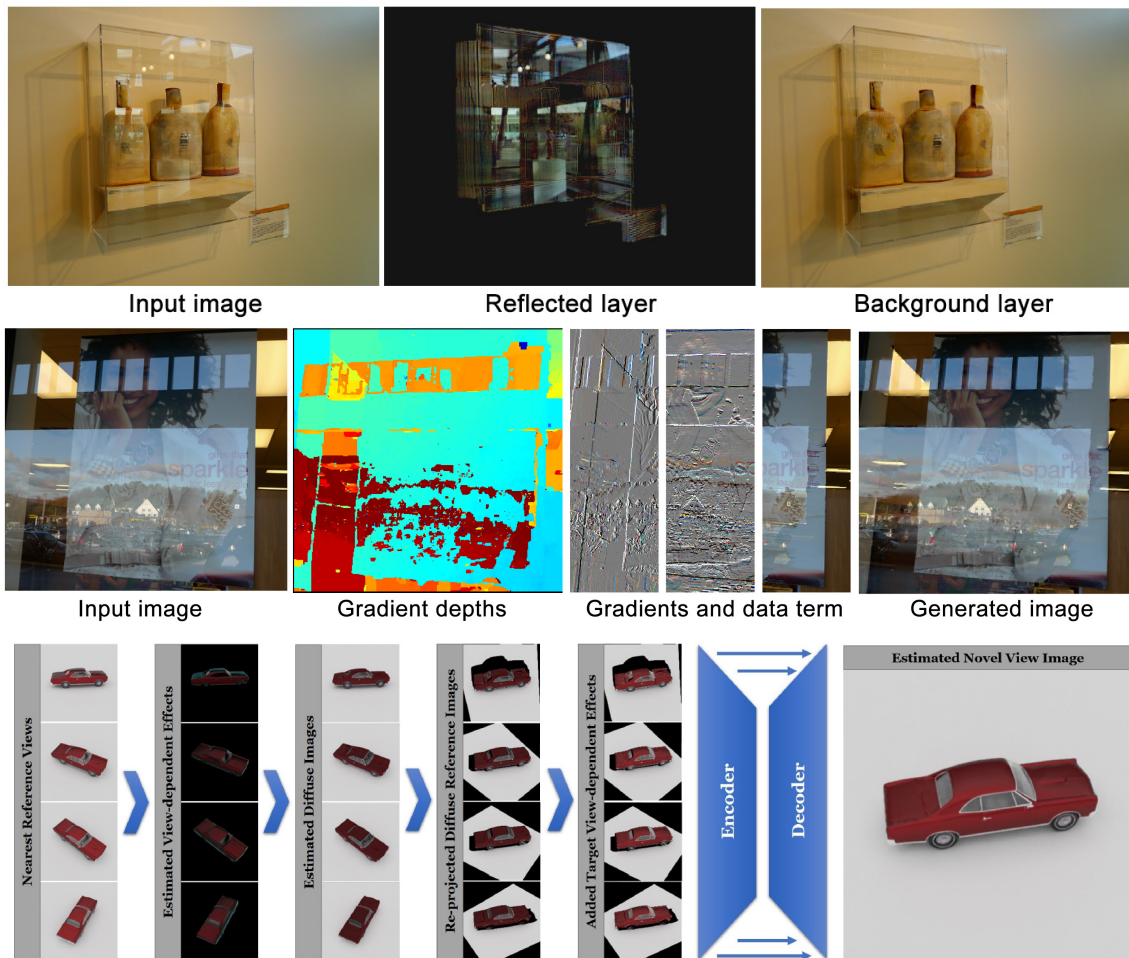


Figure 2.23 – First row: Sihna et al. [SKG⁺12] separately reproject a background and a reflection layers using distinct geometries. Second row: Kopf et al. [KLS⁺13] instead reproject color gradients using their estimated depth and integrate to obtain the final image. Thies et al. [TZT⁺20] rely on a neural network to generate a view-dependent effects layer that is subtracted from input data and added *a posteriori* to the novel view.

novel view. The final color image is obtained by Poisson integration of the generated gradient fields combined with the approximate solution as a weakly weighted data term (Fig. 2.23, second row). This method captures reflection and transparency effects well, but the assumptions used limit it to mainly planar reflective surfaces.

In IGNOR [TZT⁺20], view-dependent effects are not directly extracted from the input views. An encoder-decoder neural network is trained to predict view-dependent effects

for any viewpoint based on the corresponding depth map – obtained by reconstructing the geometry with MVS. For each selected input view, the effects layer predicted by the network is subtracted, and the resulting diffuse layers are then warped to the novel view (Fig. 2.23c). The network is evaluated again for the novel view parameters, generating a new effects layer. The reprojected diffuse and the view-dependent layers are blended using a second network. Training is performed on dense unstructured datasets; the loss compares the obtained diffuse images for pairs of views, as this layer is expected to be view-invariant.

Specular path evaluation. Specular flow can also be estimated without modeling reflected depth as a geometric layer. As previously described, efficiently finding specular paths at each visible location is an expensive task, and matching reflections between two views is at least similarly complex. Multiple works have described approaches that attempt to use sparse specular information to solve one of these two problems on-the-fly. For instance, if some light paths from a reflected point p to a viewpoint q (as shown on Fig. 2.24, left) are known, they can be extrapolated to estimate paths at other viewpoints or for other reflected locations.

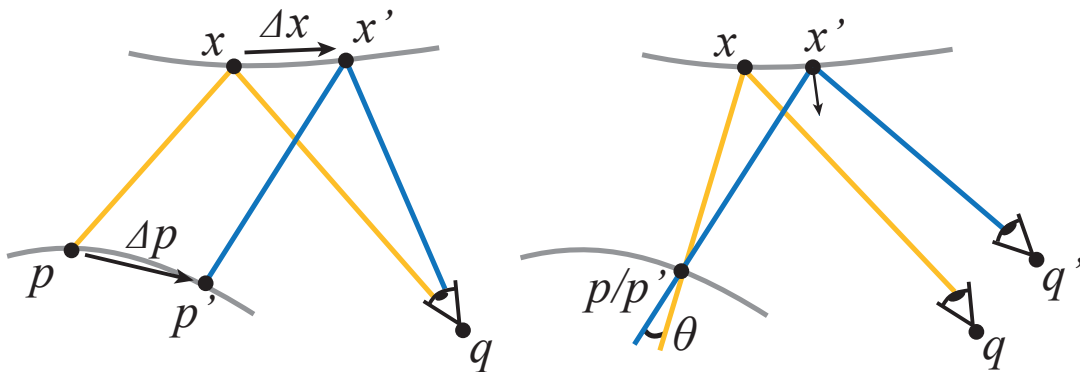


Figure 2.24 – Left: When some specular paths (p, x, q) are known, the location x' at which nearby reflected points p' reproject can be estimated using path perturbation. Right: If all specular paths (p, x, q) are known, it is possible to search for the point p/p' that is reflected at a given x' when seen from a novel viewpoint q' .

Chen and Arvo [CA00b] provide a formal framework for specular path perturbation to perform such extrapolation. Following Fermat’s principle, valid light paths between a reflected point p and a viewpoint q minimize the optical path length [MH92]. For a nearby reflected point p' , valid paths are very similar – assuming they interact on the

same reflector. These paths can thus be obtained by a perturbation of the known reflection path (p, x, q) . Let us define $\Phi(p) = x$ the path function giving the reflection point where p is visible when the viewer is at q (fixed). The following approximation can be derived: $\Phi(p') = \Phi(p) + J\Delta p + \frac{1}{2}\Delta p^T H \Delta p + \mathcal{O}(\|\Delta p\|^3)$. Here, J and H depend on the reflector surface and capture its variations when moving in the neighborhood of x . Note that these quantities can only easily be evaluated for surfaces with an analytic representation.

Instead of updating the reflection position of a known reflected point, Lochmann et al. [LRR⁺14] fix the reflection position and search for the new reflected point. In their use case, all reflection paths are known for an input viewpoint q (see Fig. 2.24, right). At a novel viewpoint q' and for a given x' , they search for the visible reflected point p' . The best p' is such that the reflection direction computed in the novel view q' using the normal at x' , and the reflection direction in the input view q using the normal at x' , coincide. If this is the case, the specular path (p', x', q') is valid and the reflected color can be used. To search for this p' , a gradient descent is performed to maximize the dot product of the input and novel view reflection directions. To avoid having to express the gradient analytically for all types of surfaces, they compute finite differences using the reflected positions map generated at q . Because the baseline between q and q' is small, the search converges quickly under the condition of a good initialization. They use a similar approach for the refraction flow.

In chapter 4, we propose a simplified flow estimation method that searches for a valid point on the surface of an ellipsoid reflector, allowing for real-time reflection reprojection from a large set of input images. Similarly to Lochmann et al. we perform a gradient descent guided by a geometric alignment criterion, but our search for the reflection point is constrained to the surface of the reflector. Because we rely on an analytic surface representation, the process is simplified and doesn't require additional position maps. In chapter 5, we build upon the framework of Chen and Arvo to fetch precomputed specular information in a set of panoramic probes. While the initial work was limited to surfaces defined by implicit functions, here we generalize the approach to triangle meshes by building a set of local implicit representations. While the goal of the techniques presented in this section is to estimate the specular flow, they address it in two very different ways, either solving for the reflection or the reflected location in the novel view. These two approaches are also used for the generation of new specular effects, as we describe in the next section.

2.2.3 Generating specular effects

When rendering synthetic scenes in real-time, specular paths are often not known in advance. In this case there is no existing color information to warp, and novel reflections have to be generated on the fly. Depending on the hardware constraints and the scene setup, locating new specular effects in the scene can be cast as a splatting problem where reflected objects have to be projected on the reflectors at the correct location, or as a gathering task where for a given point on the reflector, incoming contribution from the environment has to be evaluated.

2.2.3.1 Reprojecting reflected objects

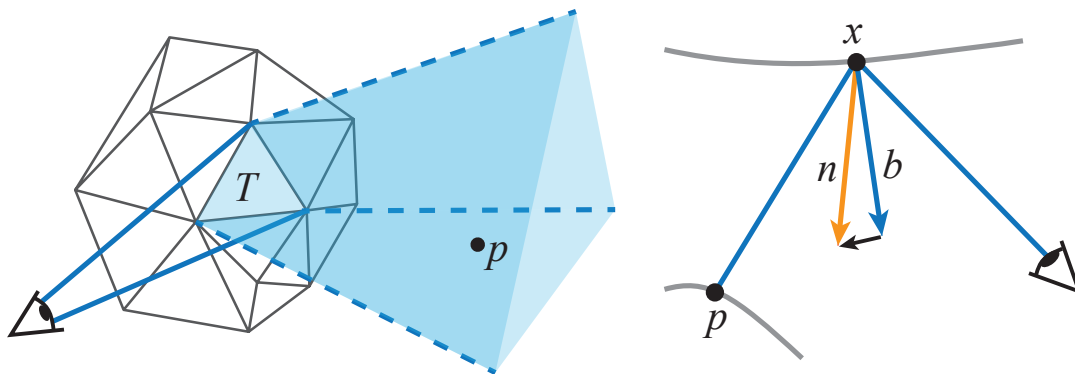


Figure 2.25 – Left: from a fixed viewpoint, each triangle of a reflector reflects a given region of the scene; this information can be precomputed and stored in an *explosion map* [OR98]. Right: for a reflection path to be valid, the bisector of the view and reflection directions b should coincide with the surface normal n . The difference between the two directions can be used to search for the proper reflection point, as shown by Estalella et al. [EMD⁺05].

Because current graphics hardware is optimized for rasterization, geometry transformation and reprojection, multiple methods attempt to project the scene objects onto the reflectors. For each reflector and for each potentially reflected object, a copy of the object geometry is created and splatted onto the reflector surface. This requires proper handling of occlusions and a good tessellation of the reflected geometry, as the projection is performed on a per-vertex level on potentially curved reflectors. Estimating the projected position is a complex problem, as described previously. If the reflector is a triangle mesh, each triangle can be treated as a small planar reflector. For a given viewpoint, it

is possible to precompute the region of the scene space that will be visible in a given triangle and store it in an *explosion map* [OR98] (Fig. 2.25, left). Reflected vertices can then query this information at runtime to determine onto which triangle they should reproject. Shading of the reflected surfaces can then be evaluated and blended with the underlying diffuse appearance of the reflector. Because the map of each reflector has to be updated when the user moves in the scene, the interactivity of the method is limited.

Maps can also be used to store geometric information of reflectors that can then be queried arbitrarily at runtime using hardware support for textures. For instance, if the reflector is entirely convex or concave, surface positions and normals can be stored in cube-maps. For a given reflected vertex, Estalella et al. [EMD⁺05] determine a valid reprojection location on the reflector by searching in such maps. The validity of a reflection point is evaluated based on the property that for a valid specular path, the bisector of the view and reflected directions coincides with the surface normal. Furthermore, when they do not coincide, following the error direction is guaranteed to lower the error at the next iteration (Fig. 2.25, right).

Specular path perturbation theory has been used to solve a similar task in real time [CA00a]. For a given scene setup a set of reflection rays between different objects are precomputed and stored in a fast query structure. When objects are moving in the scene, nearby rays can be fetched and perturbed based on the novel object vertices location. The geometry is then projected at the updated location on the reflector.

Roger et al. [RH06] also rely on Fermat's principle to reproject geometry in a similar fashion (Fig. 2.26). For a given vertex to be projected, they minimize the total optical path length by finding zeros of its gradient using the iterative secant method. Three sample points are picked on the reflector, at each one the gradient of the path length is computed and interpolated over the triangle. The location with the smallest gradient norm is found and replaces the sample with the largest gradient norm, and the process can be iterated until the desired accuracy. In practice, the evaluation of the optical length gradient can be complex for general surfaces; the reflector is thus assumed to be star-shaped to limit the number of parameters for the optical path length gradient that can be tabulated.

For reprojection of reflected vertices the main limitation is thus the complexity of the reflector surface; most techniques described above are limited to very specific types of objects. Furthermore, the need to handle a copy of each reflected object for each reflector

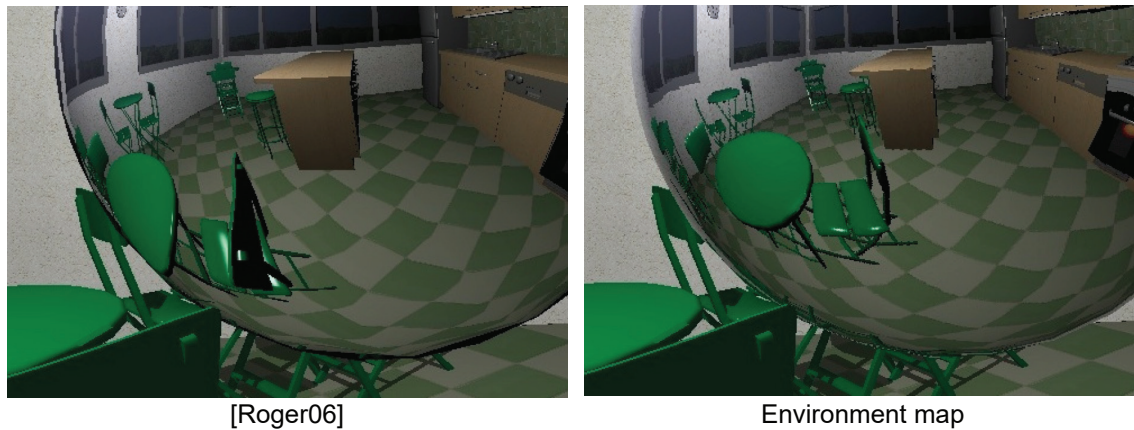


Figure 2.26 – Roger et al. [RH06] reproject reflected vertices onto the reflector, finding the proper location using the secant method. Compared to an environment placed at the center of the reflective sphere (right), their method properly displays reflections of the back of the green chair (left), invisible in the environment map.

leads to a combinatorial explosion for large scenes.

2.2.3.2 Finding reflected points in the scene

Determining the reflected point(s) that contribute to a reflection might seem like a simpler way to tackle specular effects generation. But reflected objects can be located arbitrarily far from the reflector in the scene, meaning that there is no locality to reflection contribution queries. While this is not an issue for ray tracing based approaches [Whi80] and recent hardware support has brought new options for real-time specular effects, the most commonly used rasterization pipelines struggle with the lack of locality. Many methods have been designed to counter these constraints; we review the methods closely related to our work and refer the reader to a more detailed review [SKUP⁺09] for additional techniques.

Environment maps and light probes. To avoid querying the geometric scene representation and having to shade reflected points, an extremely simplified image-based representation of the scene can be used. An environment map is a 360-degree panorama of the scene that captures incoming radiance from any direction at a specific location in the scene. They can be precomputed offline or updated on-the-fly when the scene is modified. To estimate the radiance incoming from a given direction at a reflector

point, the environment map can be queried very efficiently using hardware cube-maps or other parameterizations. Because they do not store any geometric information, environment maps are only valid at their initial location and lack parallax effects that would be observed at other positions in the scene.

To alleviate this issue, a simplified representation of the scene can be used to correct the direction queried in the map. Brennan et al. [Bre02] rely on an analytic representation of the surrounding geometry – such as a box or cylinder – to apply proper parallax correction when fetching in the environment map. The reflected ray intersection with this proxy is computed analytically, and the direction from the environment map location to this intersection is used to query the incoming radiance. For additional accuracy, and to support large scenes with different lighting conditions, multiple environment maps or probes can be generated at different locations in the scene. For each reflector point the closest ones can be selected, queried and blended to estimate the incoming radiance [SZ12]. This has strong links to the irradiance volume approach [GSHG98], even if lighting information is used there for diffuse irradiance and thus stored at a much lower resolution.

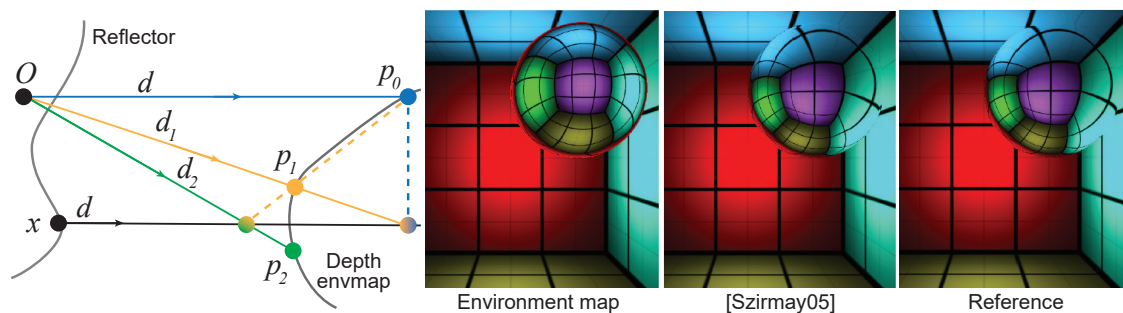


Figure 2.27 – To obtain the incoming radiance at x in direction d , [SKALP05] iteratively fetch into an environment map with depth (centered at O). At step i , the depth p_i rad in direction d_i and the previous depth p_{i-1} define a plane that is intersected with the initial ray (x, d). The intersection defines a new direction d_{i+1} to fetch into the map, and the process is iterated. The last estimated direction is finally used to fetch the environment incoming radiance.

More accurate reflection effects can be obtained by storing a depth map along with the environment map [SKALP05]. This is used for parallax correction by successively refining the direction fetched in the probe. As shown on Fig. 2.27, when the environment

map is queried in an initial direction d , the surface visible in this direction is used to approximate the scene geometry by a local plane. The ray from point x in direction d is intersected with this plane, and a new direction obtained as with the simplified analytic proxy approach. The process can be iterated for additional accuracy.

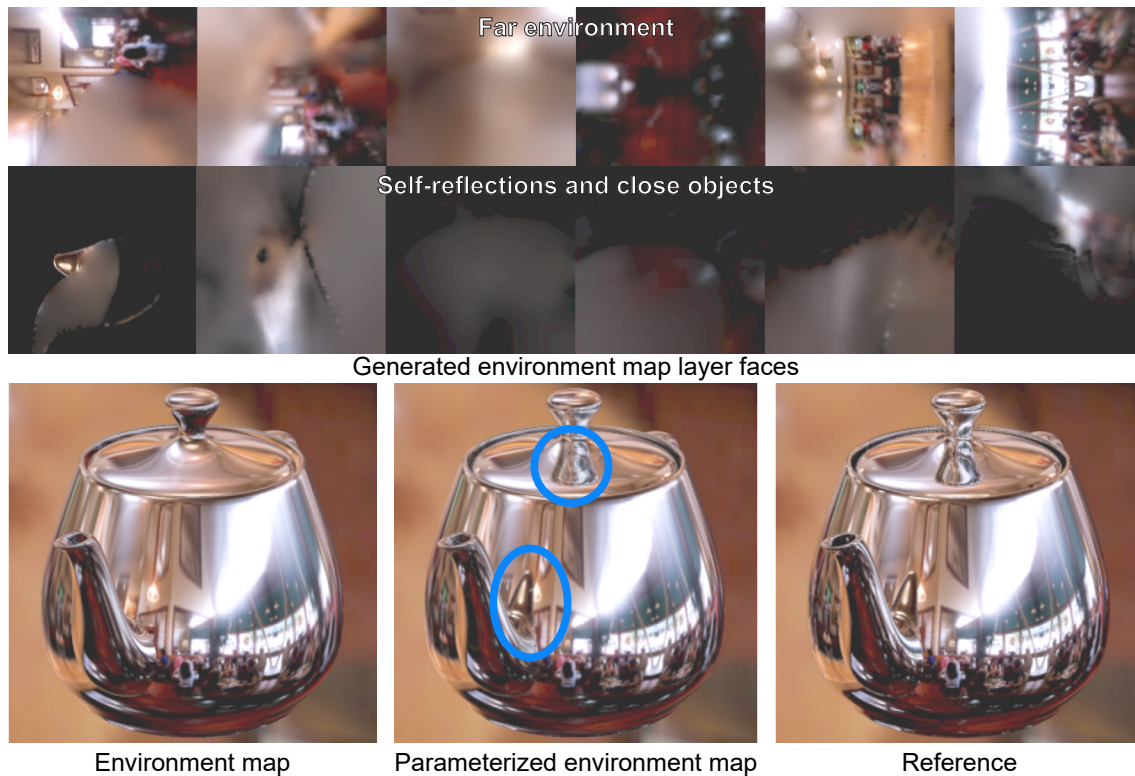


Figure 2.28 – A parameterized environment map is generated from a 2D layered rendering [HSL01]. Self-reflections and close objects are stored in a first layer, the distant environment in a second. Missing regions are filled in. Compared to a regular environment map, self-reflections – such as the reflection of the spout – and parallax effects are replicated.

Parameterized environment maps [HSL01] are a set of environment maps used to generate reflections on a given object with self-reflections (Fig.2.28). To simplify their generation, each map is inferred from a precomputed layered 2D image instead of a full panorama. Different localized environment maps are picked as the viewpoint rotates around the object. Each map contains two layers, each associated to an analytic proxy: one containing self-reflection and radiance information of close objects, and one containing the distant environment. Because each map is generated from a 2D rendering, an optimization is

applied to fill-in regions of the background layer occluded by the close objects, or regions of any layer that was not visible in the reflected image.

Environment maps have been extensively used in research and production thanks to their scalability and hardware support, even though they require a lot of storage to cover large scenes and are costly to update in real-time. Inspired by their prevalence, existing efficient parameterizations and the possibility to store additional information such as depth along with the radiance, in chapter 5 we rely on panoramic probes to directly store precomputed specular information and reproject it at runtime on reflector surfaces, producing realistic reflections on mirror and glossy surfaces. We also introduce a novel adaptive parameterization that locally maximizes the information stored in the maps based on the visible material properties.

Raymarching and ray tracing. While recent GPU hardware is now starting to allow for ray tracing in real-time [WM19], performance remains constrained for complex light paths such as glossy surface and multiple reflection bounces. Indeed rays cast following a glossy BRDF lobe can intersect any part of the scene, leading to low coherency when shading a pixel. Furthermore the shading at each intersection point has to be evaluated, introducing an extraneous evaluation cost which can be redundant for nearby reflector points. To alleviate this redundant work, [HSAS19] precompute environment maps as a G-buffer that can be shaded once per-frame at runtime, combined with ray tracing to find the proper intersections in these maps for reflection rays.

On the contrary, image space gathering [RS09] leverages this redundancy by only computing perfect mirror reflections for all reflective surfaces using real-time ray tracing. An image-space filter is then applied to create plausible glossy reflections. The image-space filter exploits the fact that rays required to evaluate a glossy BRDF at two nearby points are very similar (see Fig. 2.29). The BRDF lobe convolution can thus be approximated by an image-space blurring kernel based on the projection of the glossy lobe in the image plane.

To avoid relying on specific hardware, a simplified scene representation can be used to answer intersection queries: voxel cone tracing [CNS⁺11] builds a low-resolution volumetric representation of the scene that can be traversed to detect ray intersections and fetch back the associated stored radiance. This information can be used for perfect reflections and, through a volume mip-mapping scheme, for approximate glossy reflection



Figure 2.29 – Image-space gathering [RS09] approximates glossy reflections by using nearby perfect mirror rays (bottom left). Generated images using different roughness levels (right).

effects. McGuire et al. [MMNL17] and Wang et al. [WKKN19] compute radiance, surface normal and depth for a set of local probes, creating a light-field-like scene representation. When casting a ray from a surface for reflections, it is projected into the closest probes, performing ray-marching against stored depth (Fig. 2.30). Normal and discontinuities are taken into account to detect occlusions. Multiple nearby probes are explored until an intersection is found, the associated radiance is then fetched. Similarly to ray tracing, glossy reflections have a stark performance impact due to the lack of ray coherency, but the technique is supported on a larger range of hardware.

A G-buffer of the novel view can also be used as an approximate partial scene representation. Screen-space reflections [MM14] are generated by marching a ray against the visible scene depth buffer. When an intersection is detected, shading information can be fetched from an image buffer containing the part of the shading already evaluated for this frame, or the full shading from a previous – reprojected – frame. Disocclusions and motion have to be tracked and detected to avoid spurious reflection artifacts. Because only screen-space information is used, this technique only supports reflection of objects that are visible, but combines well with other existing techniques such as local environment maps.

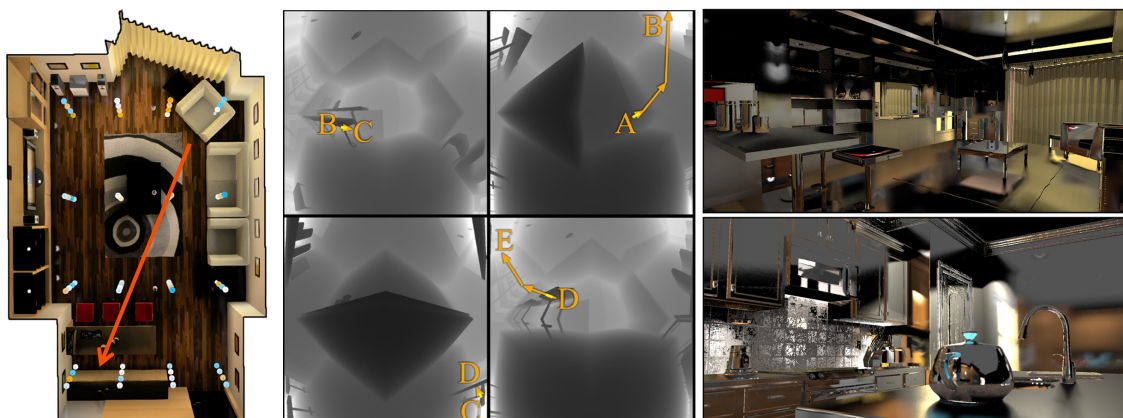


Figure 2.30 – [MMNL17] store visible surface information in a set of probes on a regular grid (left). When casting a ray, nearby probes are sampled along the ray to find the surface intersection (center). This can be used to trace reflections (right).

2.3 Summary

In this chapter, we have shown that many image-based techniques exist for rendering of real and synthetic scenes. Different representations can be used depending on the type of acquisition or generation setup, from regular grid of 2D images to individual samples generated on the fly with different capture densities. Reprojection of plenoptic data in the novel view is simplified when the input data is dense, as shown in Sec. 2.1.1. When input data is sparser, geometric scene representations can be leveraged to reproject data and generate the novel view (Sec. 2.1.2). We focus on this type of setup in the following chapters, as multi-view stereo techniques have matured and the capture process does not require specific rigs or sensors. Mesh-based approaches are additionally well-fitted to the classical real-time rendering pipeline.

Meshes and Blending. In chapter 3, we explore rendering of real-world facades from a small set of input images, improving the geometry by exploiting repetitions. Because view-dependent effects are adversarial to most reconstruction techniques, in chapter 4, we detect problematic regions in street-level scenes and repair them, while using simplified surfaces to represent thin reflectors. In both cases, we leverage existing blending techniques, based on heuristics or learning (Sec. 2.1.3) depending on the type of scene and effects. In chapter 5, we use additional material-aware sample selection heuristics when accumulating specular information. Furthermore, the use of image-based

techniques to render complex effects in synthetic scenes described in Sec. 2.1.4 inspired us to explore how precomputed data could be leveraged for rendering specular light paths in such scenes.

Reflections and Flow. Reflections and other view-dependent effects are paramount to the perception of realistic environments. For real world scenes, specular effects have to be extracted and warped to the novel view (Sec. 2.2.1). In chapter 3, we combine information from the sparse input views to detect facade elements exhibiting specular behavior. This information is then used to apply additional reflective effects. We also build on existing layer separation techniques in chapter 4, where we separate the reflected and transmitted components for rendering of car windows. In both real and synthetic scenes, specular effects have complex motion behaviors that require special treatment depending on the geometric configuration of the scene, as shown in Sec. 2.2.2. Inspired by existing work, we describe a novel flow computation approach for real world urban scenes with predefined types of reflectors in chapter 4. By using analytical surfaces we are able to solve for the flow of reflections in real-time. In chapter 5, we apply the specular path perturbation framework (Sec. 2.2.2) on general meshes to gather synthetic specular information from a set of precomputed panoramic views and generate high-quality mirror and glossy materials. Because of their indirect nature, specularities often have to be generated from scratch in synthetic scenes. But computing exact light paths on the fly is still too expensive for most current hardware. Simplified scene representations, such as environment maps with proxy geometry or depth maps (Sec. 2.2.3) can be used for approximate specular effects. Inspired by the prevalence and convenience of environment maps, in chapter 5 we store specular information in multiple panoramic views placed in the scene, along with enough geometry information to reproject specular data in the novel view in real-time. We also build on screen-space techniques to provide an accurate rendering of glossy surfaces.

Repetitions for Image-Based Rendering of Facades

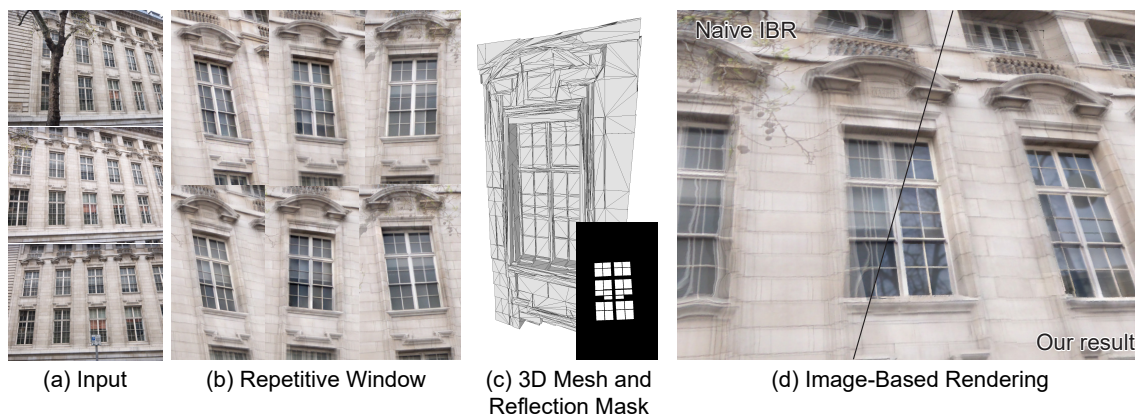


Figure 3.1 – Given a small number of pictures of a facade (a), we augment the number of views of the repetitive windows by combining the different views of each instance (b). We place the augmented set of viewpoints into a common space, allowing us to generate finer 3D geometry and to identify reflective areas (c). Compared to an image-based rendering of the input views, our solution produces sharper results and fewer popping artifacts.

3.1 Introduction

We focus on *real world* data in this chapter, and more specifically the rendering of building facades. City-wide street level captures are now widely available (e.g., Google Streetview, Microsoft Bing StreetSide), but are acquired with a very sparse baseline. As discussed in chapter 2, capture density is central in this context and such low densities hinder all components of image-based rendering: camera calibration, geometric reconstruction, and input image reprojection and blending for rendering, especially with view-dependent effects. Recent IBR algorithms that rely on geometric scene representations (Sec. 2.1.2) allow high-quality free-viewpoint navigation of cityscapes, but still require a relatively high capture density – typically a high-resolution image every meter or so. In this

chapter we propose a solution for IBR from a sparse street-level capture by leveraging the repetitive nature of facades. The key idea in our work is to extract an idealized or *platonic* object corresponding to a given repetitive element, and use its multiple instances present in the facade to perform *data augmentation* allowing us to perform high-quality free-viewpoint IBR from sparse input. In particular, we focus on repetitive windows, which exhibit rich geometric and photometric details.

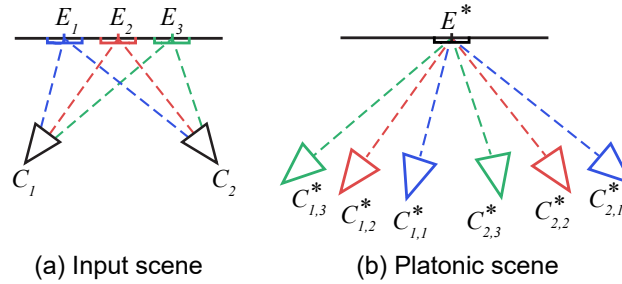


Figure 3.2 – A small number of views of similar – but physically distinct – elements E_i can be seen as a larger number of views of a unique element E^* if we align them into a common coordinate system.

We start with a small number of views of a given facade (3-4 street-level pictures), and approximate cameras calibrated using Structure-from-Motion (SfM). We then semi-automatically extract cropped instances of the repetitive windows, which we call *subviews*. Our idea is to consider that all these subviews represent the same platonic window, as illustrated in Fig. 3.2. We use 3D information in all subviews during Structure-from-Motion to provide finer camera calibration in the common space, while running multi-view stereo in this space gives us a dense, albeit noisy 3D mesh of the platonic window. We refine this mesh by decomposing it into planar polygonal regions aligned with image edges, which typically correspond to the window panels, frames and surrounding bricks.

While the multiple subviews provided by similar instances improve 3D reconstruction, they include color variations that are view-dependent – such as reflections on glass panels – and instance dependent – such as different color or shape of blinds –, which produce severe popping artifacts if used directly in IBR. We extract view-dependent variations by analyzing each instance separately, effectively removing reflections from the pictures. Aggregating the resulting reflection layers over all instances gives us a unique reflection mask for the platonic element, which we use to composite environment reflections during

rendering. We treat instance-dependent variations by re-projecting each subview of an instance into the subviews of all other instances, only mixing information between different instances in the presence of occlusions.

In summary, we describe in this chapter the idea of exploiting repetitions to augment visual data in the context of image-based rendering of facades, and show how this augmentation improves camera calibration, 3D reconstruction, and reflection segmentation. Our contributions are the following:

- A camera calibration process in platonic space that improve the accuracy of the estimated poses.
- A geometric refinement that leverages information from the multiple instances and additional geometric priors.
- A data augmentation approach to extract view-dependent effects and locate reflections while ensuring consistent rendering of the platonic elements in the final scene.

These elements combined allow us to greatly improve visual quality for IBR of facades captured with a small number of pictures.

3.2 Related Work

In this section, we discuss aspects of previous work that are more specifically relevant to the research in this chapter, notably related to image-based rendering in a sparse context using scene geometry (Sec. 2.1.2), the extraction and rendering of view-dependent effects in real world scenes (Sec. 2.2.1) and the use of repetitions in computer graphics.

Image-Based Rendering. As presented in chapter 2, image-based rendering algorithms can synthesize novel views by interpolating between photographs of a real scene. When the acquisition density is very high – for instance in the Light field [LH96] and Lumigraph [GGSC96] methods – this resampling process can be performed directly (Sec. 2.1.1). When the input images are sparser, additional information such as a 3D mesh of the scene can improve image interpolation (Sec.2.1.2). However, the baseline of the input images still needs to be small enough to obtain good angular resolution, and avoid ghosting and other rendering artifacts. Wide baselines also hinder automatic 3D reconstruction of the mesh, which is particularly problematic along object silhouettes where foreground

and background should not be mixed. Recent methods address this latter challenge by generating per-view geometric information [CDSHD13, HRDB16] that better capture occlusions and silhouettes (Sec.2.1.2). In contrast, we exploit repetitions present in facades to artificially augment the number of input views, which benefits both the 3D reconstruction and image reprojection steps of Image-Based Rendering.

View-dependent Effects. While dense IBR representations like light fields naturally handle view-dependent effects, wide-baseline methods based on 3D meshes often produce strong popping in the presence of reflections. Proper handling of such effects requires specific multi-layer representations (Sec. 2.2.1) where the reflected layer lies at a different depth than the reflective surface [SKG⁺12, KLS⁺13]. While we extract reflection layers from our input images, we cannot recover the depth of the reflected objects because they are rarely visible in multiple images due to our wide baseline. Instead, we combine the information provided by the reflection layers of all instances of a repetitive window to estimate a reflection mask for that window, which we use to render plausible reflections using image-based lighting.

Repetitions. The presence of repetitions in images has been exploited for numerous applications in Computer Graphics. In particular, several methods build on the idea that different instances of a *repetitive* element can be seen as multiple views of the same platonic element. For example, Xu et al. [XWL⁺08] take a single picture of a flock of animals to generate an animation of that animal, effectively turning spatial repetition into temporal information. Aittala et al. [AWL⁺15] use repetitions in a flash picture of a material sample to extract patches of the material under different lighting conditions, recovering rich angular information for SVBRDF acquisition. Dekel et al. [DMIF15] extract local differences between similar patches to attenuate or accentuate small variations in an image. Closer to our context is the work of Alhalawani et al. [AYLM13], who rely on user interaction to detect window-specific details, such as blinds and shutters. Since these details are present in different states in the various instances, they can be sorted and animated. We follow a similar strategy as the above approaches by turning spatial repetition into angular information, although we target the different application domain of image-based rendering.

Our approach is also inspired by 3D reconstruction methods for urban scenes. In particular, we build on the work by Wu et al. [WFP11], who detect repetitive facade elements inside

a single image to estimate a depth map, and on the work by Xiao et al. [XFT⁺08], who regularize noisy depth maps by decomposing facades into rectangular regions with constant depth. We combine and extend these two ideas to reconstruct a detailed 3D mesh from multiple images of a repetitive element. Our use of repetitions for 3D reconstruction is also related to the work of Zheng et al. [ZSW⁺10] and Demir et al. [DAB15], who aggregate information from repetitive pieces of a 3D point clouds to consolidate it. Heinly et al. [HDF14] also use repetitions to improve camera calibration by detecting conflicting observations between viewpoints.

Detecting repetitions can also be a step towards inference of procedural rules from an existing building. Some methods extract repetitions and variations from a single fronto-parallel image of a facade, as a set of tiles and rules [MRM⁺10, DRSVG13]. New facade images can then be generated by following these procedural rules and mixing the different tiles. Starting from a single planar facade image, Muller et al. [MZWVG07] factorize irreducible architectural elements. These tiles are matched to a bank of 3D models (windows, doors, etc) that are inserted in the facade plane and textured. The goal of our method is similar, but does not rely on external 3D models.

Given a set of input images and 3D model of a building, Aliaga et al. [ARB07] propose a user-assisted method to construct a procedural model of the building. The procedural grammar enables semantic edits, such as adding new floors, while the input images enable view-dependent texture mapping of each facade element. Jiang et al. [JTC09] attain similar results using a unique input image and user-drawn strokes, estimating the camera pose using the scene symmetries. Zhao et al. [ZYZQ12] also exploit repetitions to segment window elements on a facade. Jiang et al. [JTC11] extract lattice structures from repetitive facades and improve an image-based modeling process. In contrast, we focus on the components required for high-quality image based rendering of facade elements – camera calibration, geometric reconstruction, and image interpolation – rather than their use within a procedural modeling context. This remains an interesting avenue for future extensions of our work.

3.3 Overview

Figure 3.3 illustrates the main steps of our approach. Our input is a small set of images (3-4) of a building, taken at street level alongside the facade. The pictures are acquired

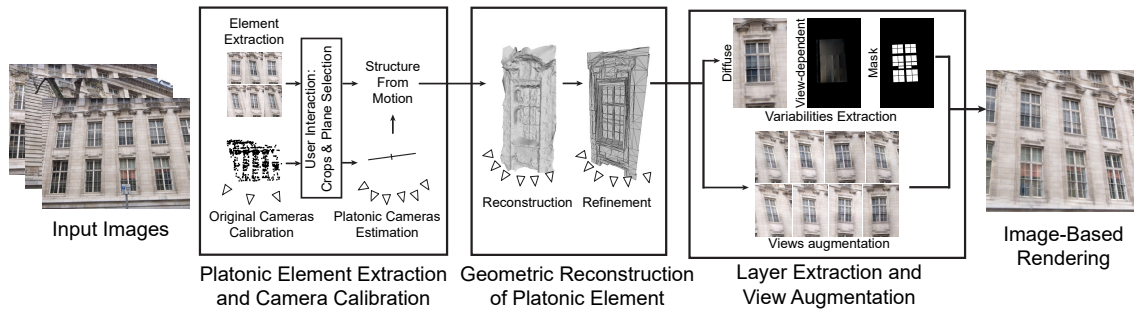


Figure 3.3 – Overview. From the input images, we first segment and select repetitive windows, followed by camera calibration of these subviews by a modified SfM algorithm. In the second step, we use the calibrated subviews to reconstruct a *platonic mesh* of the window using multi-view stereo, and subsequently refine it to obtain a piece-wise planar geometry. In the third step we extract reflections from each subview and deduce a reflection mask of the window. We additionally generate images of all instances of the window from all available viewpoints. We finally re-insert the platonic geometry and images of each instance into the complete facade for improved image-based rendering.

with a consumer camera, using fixed exposure.

We first automatically detect windows in the facade using a deep classification network, and ask the user to select the windows that are visually similar. We crop the input images around each such instance to obtain a set of images that all represent the same *platonic* window, under different viewpoints. We will refer to these crops as *subviews*.

Due to the wide baseline of the input images, existing structure-from-motion methods only produce an approximate calibration of the input cameras. We use this information to compute an initial guess of the camera of each subview in a common platonic space, which we update by running a structure-from-motion algorithm on all subviews. The initial guess allows us to reject erroneous correspondences that may occur when a subview not only shows its central window, but also part of its neighboring windows. This filtering process allows us to improve “platonic” camera calibration for each of the subviews.

After platonic camera calibration, we run a multi-view-stereo algorithm to reconstruct a 3D mesh of the platonic element. All subviews represent a similar window, but they often contain instance-specific details that disturb generic reconstruction algorithms, resulting in noisy surfaces. We regularize this geometry to obtain a piece-wise planar mesh composed of polygonal regions aligned with the dominant lines of the window.

The 3D mesh we obtain allows us to re-project any subview into any other one. We use this feature to extract view-dependent variations between subviews of the same instance, and to generate images of each instance as seen from the cameras of all subviews. While the reflections extracted from a single instance may be sparse, combining the information provided by all instances allows us to estimate a *reflection mask* that indicates the reflective areas of the platonic window.

Given these preprocessing steps, we create a complete scene representation by inserting the 3D mesh of the platonic window at the location of each of its instances over the facade mesh. We composite each view of a given instance to be consistent with the corresponding view of the entire facade, generating an enriched set of subviews by combining views and instances via reprojection. We use the resulting mesh and images in an image-based rendering algorithm, which we enhance with image-based reflections using our reflection mask. An additional use of our method output data is the generation of a multi-view textured mesh.

Terminology. In what follows, the term *input scene* refers to the original 3-4 photos of the entire facade, along with calibrated cameras and a coarse 3D mesh computed using multi-view stereo. The *platonic scene* refers to the scene containing the calibrated cameras and reconstructed geometry of the platonic element, computed from multiple subviews of that element. All elements of the platonic scene are denoted with a superscript $*$. We denote the initial images as V_i , and the input *physical instances* of the repetitive element as E_j . The cropped *subviews* $V_{i,j}$ use a double-index notation, indicating the input view i they were extracted from, and the physical instance j they contain. The associated cameras in the common platonic space are denoted $C_{i,j}^*$ and the platonic model itself E^* . We illustrate these quantities in Fig. 3.4.

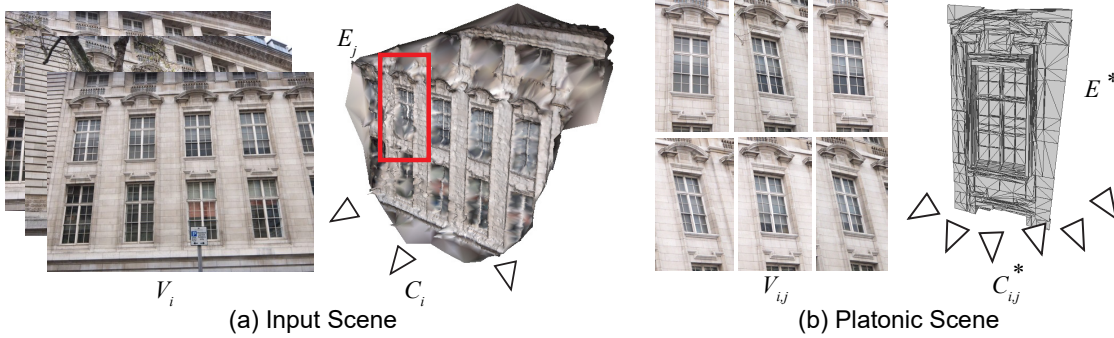


Figure 3.4 – From given input views V_i of a facade and their associated cameras C_i (a), we extract subviews $V_{i,j}$ of similar windows E_j . From these, we can reconstruct a 3D model of the *platonic element* E^* along with *platonic cameras* $C_{i,j}^*$ (b) in a common space.

3.4 Windows Extraction and Platonic Camera Calibration

The first step of our method consists in cropping repetitive windows in the facade to form a multi-view dataset of the corresponding platonic window. Calibrating the cameras of each crop within a common space then enables 3D reconstruction of the window with higher accuracy than using solely the input images.

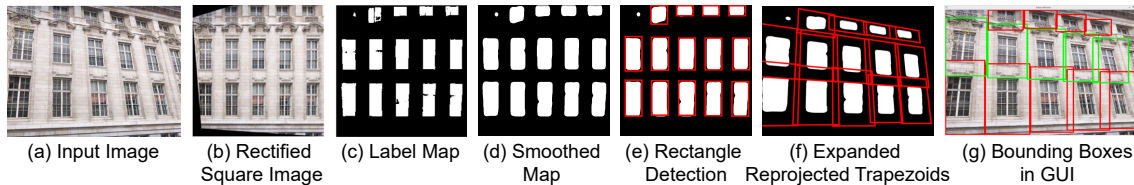


Figure 3.5 – We rectify each input image (a,b) before feeding it to a deep classification network that detects window pixels (c). We smooth the label map (d) and compute the bounding box of each connected component (e). We then expand these boxes to include the surroundings of each window and project them back into the image (f). We ask users to select the boxes corresponding to a group of similar windows (g).

3.4.1 Window extraction

We identify the repetitive windows in a semi-automatic manner, where we automatically detect candidate windows and let the user select the ones that should be considered similar (Fig. 3.5).

We generate the candidate windows by running the rectified images through a deep classification network. We use a U-Net architecture [RFB15], trained on the CMP Facade Database [RT13], containing 600 rectified images of facades with ground truth labels of architectural elements. We only predict two labels – windows and background. We obtain the rectified facade using the vanishing point method by Wu et al. [WFP10]. We then process the classification to extract its connected components, which should correspond to individual windows. We compute the bounding box of each component and scale it by a factor of 1.75 to include its surroundings, and re-project these boxes into the input image to be shown to the user.

This user interaction is very easy, and only takes a few seconds per dataset. Automating this process should be possible but would require similarity metrics robust to differences between instances of the same window while being sensitive to differences between different windows, e.g., top parts of the first and second row of windows in Fig. 3.5. We leave the exploration of robust similarity detection, for instance based on deep features, to future work.

3.4.2 Platonic Camera Calibration

We now have a list of cropped subviews, each representing an instance of the platonic window seen from a different viewpoint. However, our automatic crops sometimes contain parts of neighboring windows on their sides, as shown in Fig. 3.6a. These duplicates challenge structure-from-motion algorithms since points on the central window in one subview may be matched to points in the side window in another subview.

We filter out such erroneous matches by deriving an approximate camera for each subview, which we compute from the known position and size of the corresponding crop in the input image, as well as from the camera pose of that image. This computation also requires knowledge of the facade geometry in order to position the approximate cameras such that they all point to the same element in platonic space. We approximate this geometry as a plane, fit to the point cloud of the facade, computed by running multi-view-stereo reconstruction on the input images (Fig. 3.7). In most cases, a RANSAC fit is sufficient; when this fails we ask the user to specify the plane by selecting three points on the point cloud.

Given these cameras, we modify an SfM algorithm [MMMO] to reject matches for which

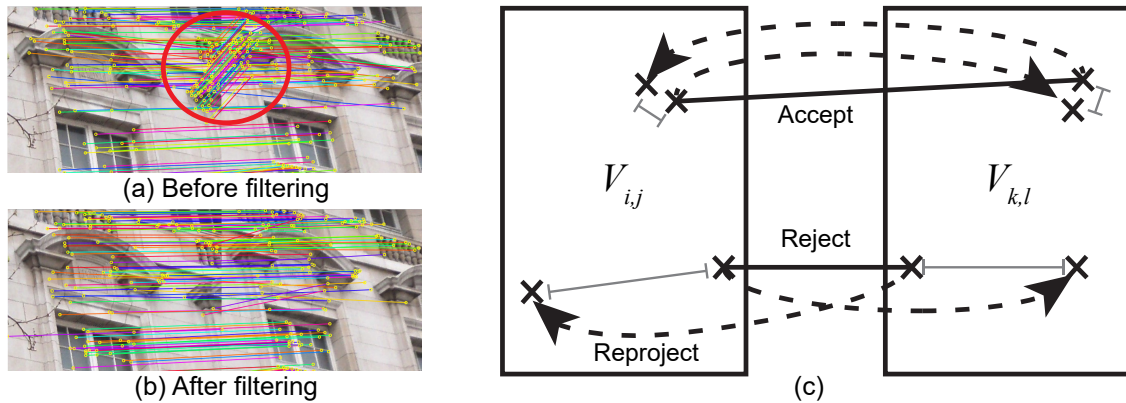


Figure 3.6 – (a) Candidate matches between two views using standard approaches. The two views represent a similar window: the first view contains parts of another window on its side, which are matched to the central window of the second view. (b) We use an approximate camera derived from the input scene to filter these outliers. (c) We reproject each pair of matched points into each other subview using the approximate cameras. If both rejections land close to their correspondences, the match is kept.

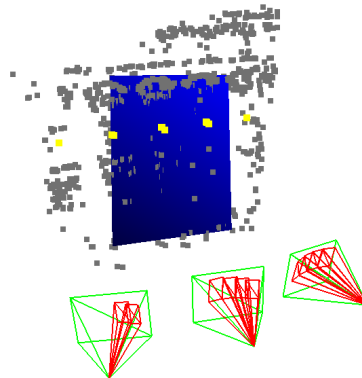


Figure 3.7 – The estimated facade plane is displayed (blue) along with the input scene sparse point cloud. Input and crop cameras are shown. The yellow points are the intersection of each crop subview with the plane.

a point, when reprojected into the other subview, lands further away than half the image dimension from its correspondence. This filtering operation greatly improves the quality of platonic camera calibration, as shown in Fig. 3.8.

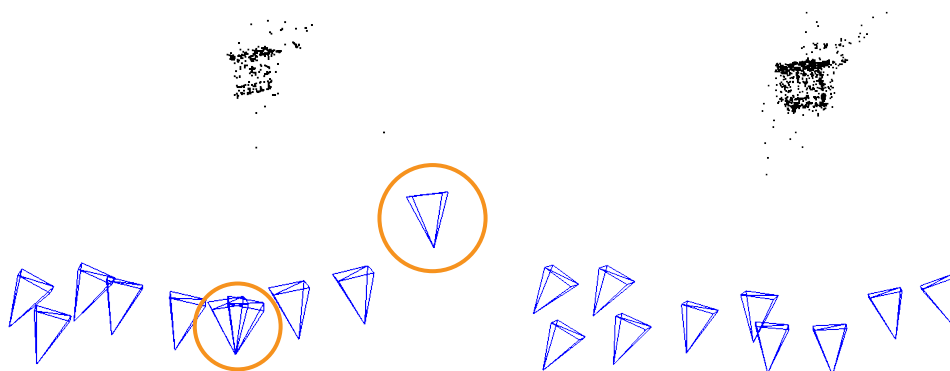


Figure 3.8 – Camera calibration using a standard approach (left), and the result with our approach (right). Note how in the standard approach, the rightmost camera does not see the window, and that three cameras are incorrectly co-located.

3.5 Geometry Reconstruction

We first describe how to reconstruct a 3D mesh of the platonic window, before explaining how we repeat it over the entire facade.

3.5.1 Platonic Window

The platonic cameras estimated by our modified SfM algorithm can now be used for dense 3D reconstruction of the platonic window. However, we found that generic multi-view stereo algorithms [Rea18] tend to produce noisy 3D point clouds on such input, possibly due to the per-instance and per-view variations present in the cropped subviews. Inspired by prior work on image-based facade modeling [XFT⁺08], we refine this noisy reconstruction by decomposing it into flat polygonal regions parallel to the facade, and aligned with image edges.

Our refinement method operates in the subview for which the view direction is the most orthogonal to the facade plane. We segment this subview into convex polygons aligned with image edges using the recent algorithm of Bauchet and Lafarge [BL18], as shown in Fig. 3.9. In a nutshell, this algorithm detects small line segments in the image and extends them until they intersect other segments to form closed regions.

We then formulate an optimization that displaces each region along the facade normal. Our formulation combines a data term, which seeks to keep the region close to the noisy

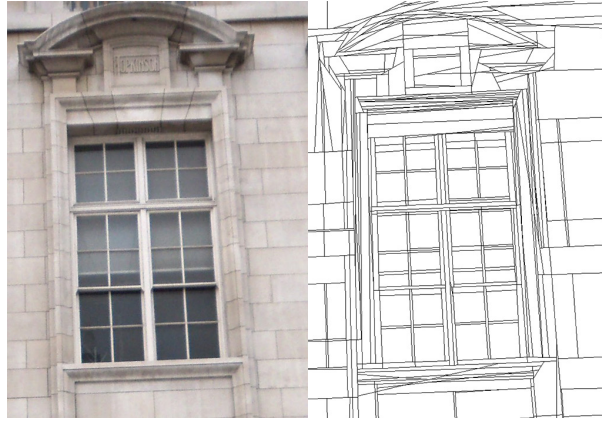


Figure 3.9 – Left: Input image. Right: Polygonal segmentation.

point cloud it covers, and a smoothness term, which encourages neighboring regions to align if they share similar colors. Denoting r a polygonal region, d_r its displacement along the facade normal ($d_r = 0$ at the facade plane), d_{ref} the average distance to the facade plane of the reconstructed points covered by r , $\mathcal{N}(r)$ the two-ring neighborhood of r , and c_r the average color of r , we want to minimize

$$E(d_r) = |d_r - d_{ref}| + \lambda \frac{1}{W} \sum_{n \in \mathcal{N}(r)} e^{-\|c_r - c_n\|^2} |d_r - d_n| \quad (3.1)$$

where $W = \sum_{n \in \mathcal{N}(r)} e^{-\|c_r - c_n\|^2}$ is a normalization factor and λ balances the two terms. We set $\lambda = 15$ in all our experiments.

Given the small number of regions, we solve this optimization problem using a simple mean field algorithm [ZC93], where we iteratively update the displacement of each region to minimize Eq. 3.1 given the displacement of its neighbors. At each iteration, we try to assign to a given region all displacements of its neighbors, as well as a regular sampling of the depth interval of the noisy reconstruction with respect to the facade plane. We initialize this optimization by setting $d_r = d_{ref}$. At the end of the optimization, we connect the displaced polygons by additional faces to handle depth discontinuities between neighbors.

Figure 3.10 illustrates the result of this optimization on two 3D meshes, the first obtained by running multi-view stereo on the input images, and the second obtained from all

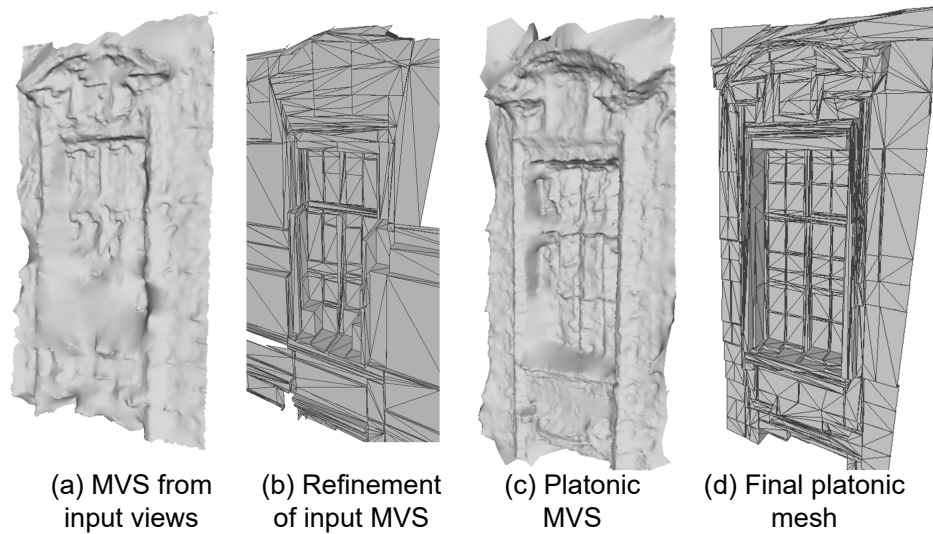


Figure 3.10 – From left to right: noisy surface obtained by applying multi-view stereo on the input images, refinement of this noisy surface, noisy surface obtained by applying multi-view stereo on the cropped subviews, refinement of this noisy surface. The cropped subviews provide additional information that yield a more detailed mesh after refinement.

subviews. The best result is achieved with the latter option, where our refinement removes bumps and captures well the flat parts of the window and wall.

3.5.2 Complete Facade

Given the reconstructed geometry of the platonic window, we generate a mesh for the entire facade by replicating the platonic mesh on the facade plane at the positions of all window instances. However, since the facade plane and the platonic mesh have been generated by separate executions of the structure-from-motion algorithm, they are not in the same coordinate system and do not have the same scale. We deduce the 3D rigid transformations from the platonic scene back to the input scene by matching the platonic cameras with the cropped cameras previously estimated in Sec. 3.4.2, as illustrated in Fig. 3.11. For each match we compute a candidate transformation and make it independent from the instance position. We average all match transformations to obtain the final one. We provide the detailed computation of this transformation in Appendix A.

While the above process results in an improved 3D mesh of the facade (see Fig.3.12), it is not perfectly aligned with the input images due to the numerous geometric trans-

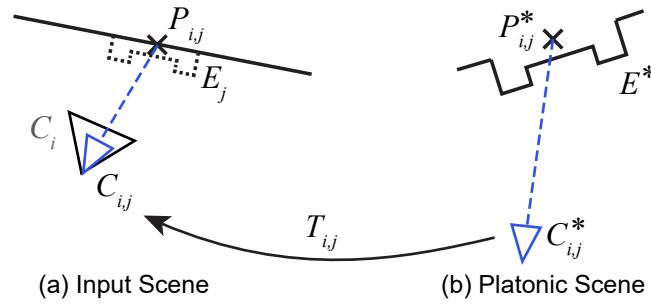


Figure 3.11 – For each platonic camera, we estimate the transformation to align it back to the corresponding estimated camera in the input scene. We use this transformation to copy the platonic mesh at each window occurrence in the facade.



Figure 3.12 – Visualization of the generated facade mesh (center) from the camera of an input view (left). We overlay the input image onto the mesh view to assess alignment. Another viewpoint of the same mesh (right).

formations involved and to the approximate calibration of the input cameras. Such misalignments produce significant ghosting if the original images are used along the refined mesh in an image-based rendering algorithm, as shown in Fig. 3.13. We next describe how to improve rendering quality by leveraging the more precisely calibrated subviews of the windows. We use these subview cameras, placed back into the input scene, along with the approximate input cameras for the final rendering. The subviews also provide complementary information gathered from different instances of the window, allowing us to better handle reflections than when using only the input images.



Figure 3.13 – Misalignments introduce ghosting when performing image-based rendering using the original images and the refined mesh.

3.6 Data Factorization and Augmentation

While the cropped subviews all represent the same platonic element, they each contain view-dependent and instance-dependent variations which would yield significant popping artifacts if used directly for image-based rendering. We treat these two sources of variations separately, as view-dependent variations mainly correspond to reflections over the window panels, while instance-dependent variations correspond to changes of shape or color of the window frame.

3.6.1 View-Dependent Variations

Each instance of a window is seen in several of our input images. While window frames are mostly diffuse, the panels are typically reflective and exhibit strong variations between these input views. However, the input camera baseline is often too wide to observe any overlap in reflections, which prevents them from being rendered with existing solutions based on 3D reconstruction of the reflected scene [KLS⁺13]. Instead, we propose to remove these view-dependent variations to later replace them by plausible reflections from an environment map.

Reflection Separation. Given all the subviews $V_{i,j}$ of a given instance, we remove the reflections in each subview by reprojecting all other subviews onto it, and computing the median gradient of the resulting image stack. We use the 3D mesh computed in Sec. 3.5 to perform this reprojection. As observed by Weiss in the context of shadow removal [Wei01], the median gradient preserves the visual content shared by the aligned images, while it discards content that only appears in one of the images. Integrating the resulting

gradient fields gives images where most reflections have been removed, as shown in Fig. 3.14. We refer to these images as *diffuse layers*, and to the removed information as *view-dependent layers*.

In practice, we only perform reflection separation for pixels identified as windows by the deep classification network (Sec. 3.4.1), since those are the most likely to be reflective. We noticed that median filtering is also effective at removing occluders present in one of the input images (trees, sign posts). Assuming that these occluders have a very different color than the scene they occlude, we also run the above process on pixels that exhibit a high variance in hue over the images.

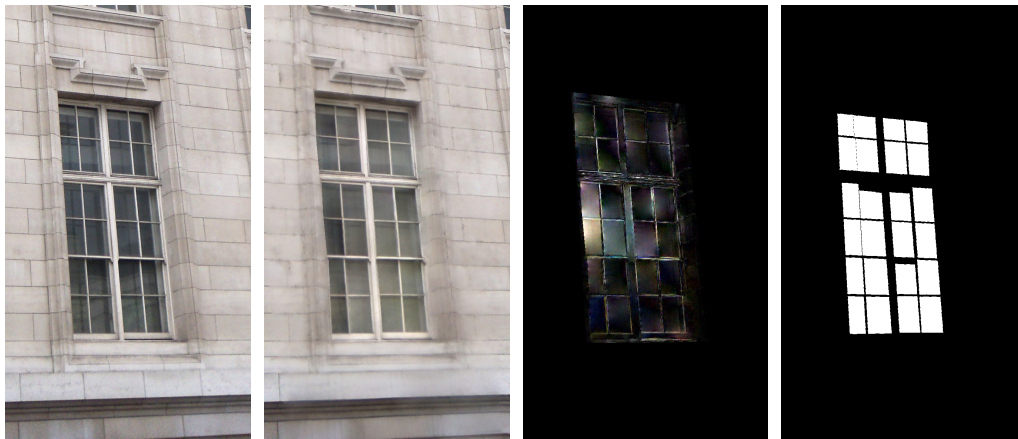


Figure 3.14 – From left to right: input cropped image, extracted diffuse layer, extracted variation layer (intensity), reprojected reflection mask.

Reflection Mask. We are now equipped with one view-dependent layer per subview, which gives us indications of which parts of the platonic window are reflective. We aggregate this information across subviews to generate a unique *reflection mask*, which we use at rendering time to composite the environment map reflections over the window. Since the extracted reflections often only cover part of the window panels, we again leverage the polygonal segmentation of the cropped subviews to obtain clean, regular masks.

We first reproject all view-dependent layers into the subview from which the polygonal segmentation has been computed (Sec. 3.5). We then count the number of pixels in each region for which the reprojected layers have non-zero intensities, and we compute

the sum of all such pixel intensities. We estimate the average value and variance of these quantities over all regions. We consider that a region is in the reflection mask if both values differ by more than 0.25 standard deviation from their average over all regions. Intuitively, this approach selects regions that are well covered by intense view-dependent effects. The resulting reflection mask (shown in Fig. 3.14) is then reprojected to all subviews. The entire view-dependent variations extraction process is illustrated in Fig. 3.15.

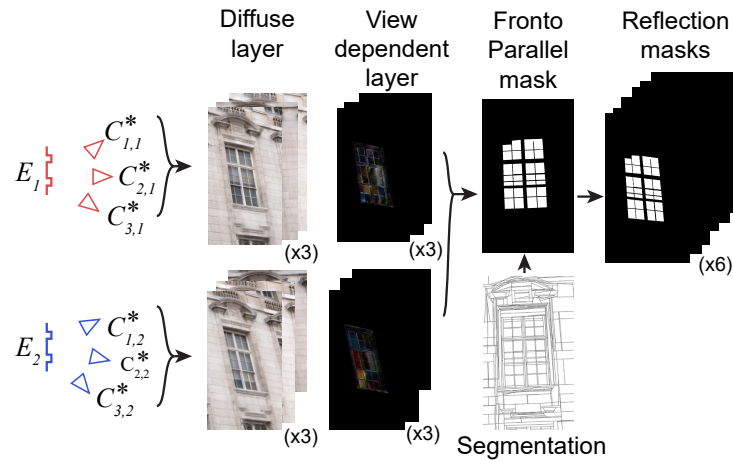


Figure 3.15 – For each instance, we decompose the associated cropped views into diffuse and view-dependent effects layers. All view-dependent layers are then combined to estimate a fronto-parallel mask guided by the segmentation. The resulting mask is then reprojected in all views.

Reflected Environment. The reflections present in the input images are too sparse to be used directly for rendering, but they provide some information about the environment surrounding the facade, see Fig. 3.16. In particular, we use the extracted reflections to construct an incomplete environment map, which we use to manually select a similar environment map from a light probe library [XEOT12].

3.6.2 Instance-Dependent Variations

The key idea behind our approach is to consider that a few views of different instances of a window can be seen as multiple views of a single platonic window. While we have shown the potential of this idea to generate a precise platonic mesh and reflection mask, using all subviews $V_{i,j}$ to render a single window yields severe popping of visual content



Figure 3.16 – Input images (left) and the resulting environment map after reprojection of the extracted reflections (center, zoom as inset). We use this incomplete environment map to select a similar one from a light probe dataset (right).

specific to each instance. For instance, the window blinds may change height or color over different instances, producing distracting animations as we rotate around the window. Each instance thus only has a subset of coherent subviews, which correspond to the crops of that instance in the original images.

Our solution consists in augmenting the coherent set of each instance by reprojecting the available subviews into their closest missing subviews. Since we only have a few available subviews per instance, parts of the window may be occluded in other subviews. We fill in these parts using content from another instance for which this particular subview is available. We introduce additional notation to indicate the *target* instance k : $V_{i,j}^k$. When the physical instance j is equal to the target instance k , all pixels are covered in the sub-view, see Fig. 3.17. When the target instance $k \neq j$ some pixels need to be reprojected from other views to complete the cropped image.

Fig. 3.17 illustrates the resulting data augmentation on two window instances, each seen in three subviews, yielding six coherent subviews for each instance. Note that this overall process is similar in spirit to the re-projection performed at runtime by the image-based rendering algorithm to generate novel virtual views. However, augmenting the subviews in an offline preprocess allows us to employ a costly gradient-domain fusion [PGB03] when combining parts from different instances at occlusions. The color coding in Fig. 3.17 indicates the source of the pixels in each subview: colored $V_{i,j}^k$ with outlined subviews are cases where target and physical instance are the same, while those in black include reprojection. Note that the colored pixels in the reprojected views correspond to the source instance used; in the general case these can come from several instances. The

effect of this augmentation is shown in Fig. 3.18.

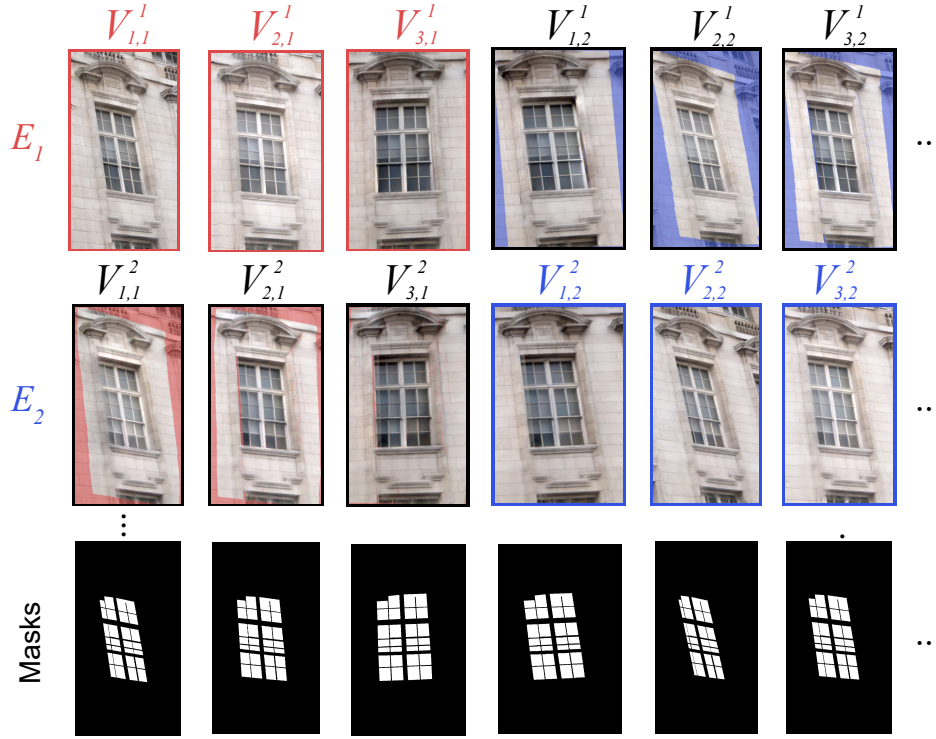


Figure 3.17 – Each instance E_i of a window is only seen in a subset of subviews (images with colored border). We augment this coherent subset by re-projecting the available subviews in other viewpoints, and fill-in occlusions and missing parts using the corresponding subviews from another instance (colored areas in images with black border). We also re-project the reflection mask in all subviews.

3.6.3 Complete Facade

We now need to create a complete model of the facade, requiring us to make the window-scale subviews compatible with the facade-scale input images. The main challenge we face at this stage is that the facade-scale images are not perfectly aligned with the subviews and refined 3D mesh, which results in visible seams if the two sets of images are combined naively during image-based rendering (see Fig. 3.19a). For each subview, our solution is to project the closest input views into it, and stitch the two images such that the subview is kept over the window, while the input image is used over the surrounding walls. We apply Digital Photomontage [ADA⁺04a] to achieve a seamless composite, using the blurred

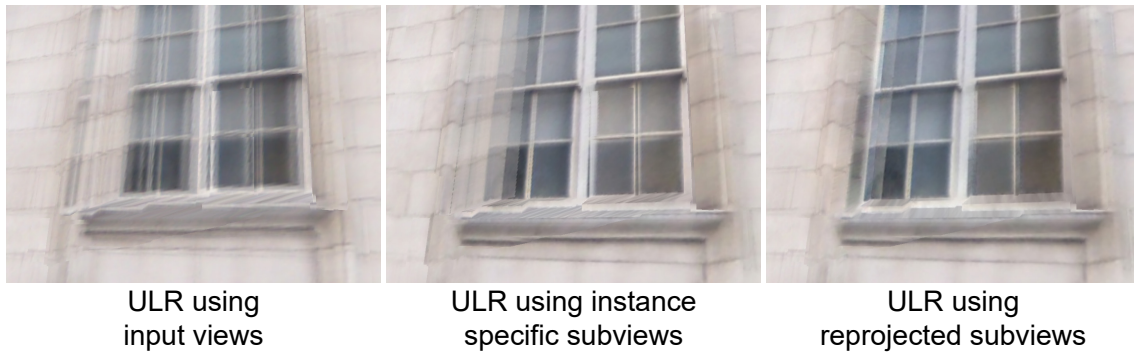


Figure 3.18 – IBR comparison on our full scene mesh, using resp. the three input views, each instance original subviews, and the reprojected subviews for each instance.

bounding box of the subview as the foreground/background unary term (see Fig. 3.19b).



Figure 3.19 – (a) If the crop is blended naively, we observe visible seams. (b) Unary terms weighting with distance from boundaries. (c) After the graph cut, visible seams are eliminated. (d) Resulting cut.



Figure 3.20 – Our method allows for instances of the platonic element to be swapped and reinserted in the final scene in a coherent manner. Left: original facade, right: novel facade.

This approach also helps with color differences between the subviews and the facade due to exposure variations in the scene. As a result, subviews are seamlessly composited and we can place the instances at different locations on the facade, creating a novel facade (see Fig. 3.20).

3.7 Implementation and results

The final augmented dataset contains the matrix of augmented views, $V_{i,j}^k$, the corresponding cameras $C_{i,j}^*$ and the augmented geometry. In addition, we have the selected environment map that will be used for synthesizing reflections.

3.7.1 Rendering

We render the scene using an extension of the Unstructured Lumigraph Rendering (ULR) method [BBM⁺01], that computes blending weights between input images for each pixel of the desired output. Our expanded set of cameras C^* and images V_{ij}^k augment the image data available for the regions in the windows. In the remaining areas of the facade plane, it is equivalent to a ULR render using solely the three input images. In the regions of the platonic elements, only the platonic cameras are used, to avoid any ghosting caused by the unprocessed input images.

To restore plausible view-dependent effects that are important to the perceived accuracy and realism of the scene, we overlay a reflection layer over the output image, using the environment map previously selected. For each pixel lying in one of the areas delimited by the reflection masks (Sec. 3.6.1), we reflect the view-to-point vector with respect to the facade plane normal and use the resulting direction to query the environment map. The resulting color is additively blended with the underlying diffuse color.

3.7.2 Implementation

We implemented our method in our custom C++/OpenGL framework. For the individual components we used the implementation of [WFP11] to rectify input images, and a Tensorflow implementation of the U-net for window segmentation. The entire preprocessing pipeline requires between 10 and 30 minutes for all our scenes; camera calibration and reconstruction requires less than 5 minutes both for input and platonic scenes. All timings are reported on an Intel Xeon 12 core 2.6GHz machine with 32Gb of memory. Rendering

is real-time.

3.7.3 Results

We validate our results on five scenes, two with facades from London (3 input images each, from Ceylan et al. [CMZP13]) and three from Paris (3 input images for Paris1 and 4 for the others). Scenes are processed as described in the previous sections. Scenes Paris2 and Paris3 required manual definition of the plane. In Fig. 3.22, for each scene we show the input augmented geometry and compare our result to the baseline result rendered with ULR rendering on the MVS reconstruction using the input images. The resulting improvement in quality is more clearly visible in videos, visible on the project web page ¹. Our contributions could also benefit rendering algorithms based on a diffuse textured mesh. In particular, the refined mesh yields better alignment of the re-projected images, which benefit multi-view texturing. We present a comparison between textured meshes generated from our data and our image-based rendering approach in Fig. 3.21. Additional comparisons, illustrating the relative advantages of each part of our method can also be found in the videos on the project page. As shown in Fig. 3.20, a novel facade can be generated from an existing scene by swapping window instances.



Figure 3.21 – Multi-view texturing in this figure was performed using the texturing module of RealityCapture [Rea18]. From left to right: textured mesh reconstructed using the 3 input views, our refined mesh textured using the 3 input views, our refined mesh textured with all our generated subviews, and our IBR solution for the same view.

¹<http://www-sop.inria.fr/reves/Basilic/2018/RBDD18/>

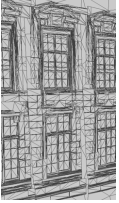



	Input images	Mesh	Masks	Viewpoint 1	Viewpoint 2
London1					
London2					
Paris1					
Paris2					
Paris3					

Figure 3.22 – Results on 2 scenes from London and 3 from Paris. Left to right: input images, augmented geometry, extracted reflection masks, baseline rendering (ULR) and our result on a first viewpoint, baseline rendering (ULR) and our result on a second viewpoint.



Figure 3.23 – On this scene where standard MVS reconstruction fails (middle row), we obtain a usable result (right).

3.8 Conclusion

Limitations. We currently rely on user intervention for subview selection. A clustering algorithm with a specific similarity metric that allows for minor differences between instances of a group could be used. Our algorithm is sensitive to the precision of the original calibration and reconstruction of the input images. In some cases, even though the MVS reconstruction fails completely we are able to produce a usable result, albeit with some artifacts (Fig. 3.23). Improved resolution street-side captures (e.g., [Eth17]) should provide sufficient quality that will allow our approach to be used. Eliminating the manual steps of subview selection and plane extraction (when needed) would allow the approach to be used at a much larger scale.

Future Work. In future work, it would be interesting to generalize our approach to sub-blocks of a given platonic element, e.g., a pediment shared by a door and a window. Such a generalization would allow our method to treat elements that are not necessarily repetitive, e.g., a door that shares sufficient sub-elements with repetitive windows. This would also allow our approach to treat elements other than windows. Another interesting avenue of future work would be the usage of our approach as a component for procedural modeling and generation. Our augmented subviews could be used as part of a procedural modeling system, allowing the generation of different combinations of the different instances. Such a method would need to include a way to generate the rest of the facade in a procedural manner, while being consistent with the requirements of IBR.

Summary. In this chapter, we have presented a novel approach to facade reconstruction by leveraging repetitions. We aggregate information from multiple instances in platonic space, performing improved camera calibration and geometry reconstruction. These

are used to extract additional scene information, ensuring visual consistency between viewpoints and locating view-dependent effects. At runtime the scene is rendered using image-based rendering and the specular masks can be leveraged to synthesize novel plausible reflections. We also made two observations that motivated the work described in the next chapter. Firstly, the use of semantic information to detect and correct specific artifacts and IBR limitations could be extended to other elements in different contexts. Secondly, the user perception of a scene is greatly enhanced when reflective effects are properly rendered, i.e. using the proper color information and warping motion flow.

In the next chapter we will describe how we leverage semantic information to detect cars and car windows in urban scenes, allowing us to extract these elements that are notoriously hard to reconstruct and render. Semantic information also helps us to synthesize reflection layers from the input views. Thanks to an analytical approximation of semi-transparent reflectors, we compute on-the-fly a reflection flow to reproject these layers, generating plausible reflection motions.

Image-Based Rendering of Cars with an Approximate Reflection Flow

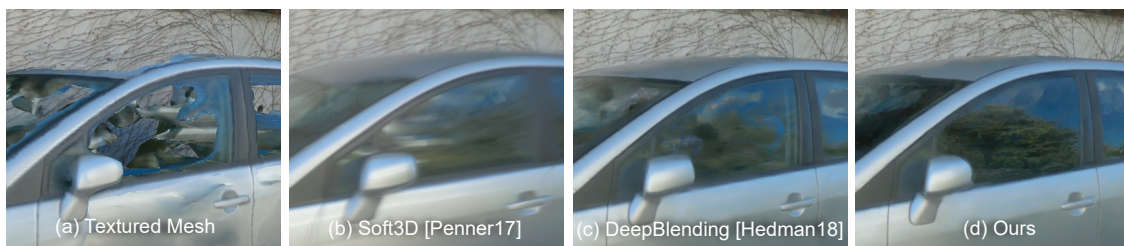


Figure 4.1 – We propose a new solution for rendering captured cars, and in particular their reflective, semi-transparent windows. A textured mesh from multi-view stereo reconstruction (a) is missing the window geometry. Recent free-viewpoint Image-Based Rendering algorithms (b, c) are not designed to handle the rendering of both the reflection of blue sky, green leaves and the transmissive content (car interior). Our method (d) handles this by computing real-time reflection flows on an ellipsoid approximation of the curved window surface, based on our estimate of a smooth hull of the car that exploit *semantic labels* in the input images.

4.1 Introduction

In this chapter, we continue to explore free-viewpoint navigation at interactive rates in cityscapes. While such scenes can be created synthetically with varying degree of realism, image-based rendering techniques provide an interesting alternative. As described in previous chapters, *real world* streets can be captured using cameras and the geometry reconstructed, allowing input image information to be reprojected to generate novel views. Yet urban scenes are complex, exhibiting different types of objects with extremely diverse materials and view-dependent behaviors. Even if the capture density is relatively dense, complex specular effects are hard to synthesize in novel views.

Among them, car and car window rendering are arguably two of the main obstacles

for using IBR for free-viewpoint street navigation. Existing solutions have difficulty with the poor reconstruction of shiny car bodies and the depth estimation ambiguity of reflections moving across curved semi-transparent windows. In this chapter, we provide a first *plausible* solution, by improving the overall rendered appearance of car bodies thanks to our estimation of smooth and filled car geometry, the believable motion of reflections on car windows and our synthesized reflection layers. We target a *lightweight* capture process with a single commodity camera (e.g., a GoPro), typically in a “street-side” fashion. In this context, recent IBR methods build on efficient Multi-View Stereo (MVS) algorithms [SZPF16, Rea18], that produce acceptable quality geometry for non-reflective/transmissive surfaces. The reconstructed geometry is used to reproject input images [BBM⁺01, HPP⁺18] in the novel view, allowing interactive, high-quality free viewpoint navigation in these cases.

However, for reflective car bodies these MVS algorithms produce inaccurate geometry due to strongly view-dependent and texture-less appearance, and window surfaces are most often missing. Most IBR algorithms (e.g., [CDSHD13, HPP⁺18]) are not designed to handle the flow of reflections on a window which is different from that of the interior or background visible through that window. Previous specific solutions for IBR with reflections (Sec. 2.2.1) have difficulty with curved surfaces of car windows, and with the novel views we target that are quite far from the input views, but quite common for street-level navigation.

We chose to take inspiration from our previous work and propose an approach where elements such as cars are detected and extracted from the scene using semantic segmentation as an input, their geometry corrected and additional reflector surfaces approximated. We use these surfaces to extract specular information from the input data, and reproject it at runtime using an approximate reflection flow. We present in this chapter three main contributions, providing a fully automatic solution for IBR of cars with view-dependent effects:

- A new algorithm to provide a complete and smooth car body reconstruction suitable for rendering in a casual capture context, as well as initial window surfaces.
- A reflection flow approximation for plausible interactive reflection rendering and an automatic ellipsoid fitting algorithm that uses the initial window surfaces.

- A reflection and background layer synthesis method building on our reflection flow.

Our solution allows interactive rendering of plausible motion of reflections in car windows, and diminishes visual artifacts due to missing and incorrect car geometry. This plausible motion greatly improves perceived visual quality compared to previous methods (Fig. 4.10, 4.11) especially when moving around the scene.

4.2 Related Work

In chapter 2, we reviewed previous work in image-based rendering of scenes with complex geometry (Sec. 2.1.2) and the rendering of view-dependent effects in that context (Sec. 2.1.3, Sec. 2.2.2). We also covered the extraction and reconstruction of specular information from real world data (Sec. 2.2.1). In this section we discuss some aspects of previous work that are specific to the work described in this chapter. We also review techniques that leverage semantic information for 3D reconstruction.

Image-based rendering. As already described in chapter 2, modern Image-based rendering techniques frequently rely on multi-view stereo reconstruction (Sec. 2.1.2), but suffer from their limitations. Despite recent progress, these methods still have difficulty with scene coverage and are not designed to handle highly reflective surface nor the depth ambiguity of transparency. Recent unstructured methods alleviate reliance on global geometry by combining refined view-dependent meshes with learning to improve blending [HPP⁺18]. We build on such ideas, and introduce a solution for semi-transparent, reflective surfaces (e.g., car windows) that were previously problematic.

Specific techniques exist for scenes with reflective and transparent surfaces as described in Sec. 2.2.1. These approaches are restricted to planar reflectors and limited motion, and are unsuitable in our context of curved windows. Volumetric approaches such as Soft3D (Sec. 2.1.2) allow multiple surfaces to be present at a given pixel; however, artifacts can appear in non-transparent regions, especially far from the input camera poses. In contrast, we provide a larger space for free-viewpoint navigation, thanks to our approximate reflection rendering and layer synthesis.

Recent work on IBR exploits neural networks to improve rendering quality (Sec. 2.2.2). Thies et al. [TZT⁺20] address the problem of moving highlights for isolated objects rather

than the full scenes we consider and do not handle semi-transparent, reflective objects. These works furthermore focus on much smaller camera baselines than ours [MSOC⁺19]. Nonetheless, these methods present many exciting ideas on using the power of deep learning to improve IBR, and we consider these very promising avenues for future work.

Layering, Reconstruction, Rendering of Semi-Transparent and Reflective Objects. Layer separation methods described in the literature (Sec. 2.2.1) require a sufficiently accurate initial flow estimate to be successful; in our context of uncertain input data, we opt to *synthesize plausible* reflection layers instead. Nonetheless, our synthesis method is inspired by the work of [SAA00], and their idea of min-composite.

Several methods address the challenge of reconstructing transparent and reflective objects, but typically require the use of custom acquisition devices [WGL⁺18, WZQ⁺18, GH15]. Volumetric approaches can also be found in the literature [IKL⁺10] to handle the multiple depths per pixel present in images of non-opaque objects. These often focus on specific object categories, or involve complex acquisition hardware setups [IKL⁺08]. We focus on a casual capture setup, using a single consumer-level camera.

Numerous methods for the rendering of reflections in real-time have been described for synthetic scenes (Sec. 2.2.3). Our IBR context is different, since we do not have access to the full geometry, but our analytical surface approximation to curved windows allows us to build on such approaches to compute reflection flow (e.g., [OR98, EMD⁺05]).

Semantic labeling. Semantic labeling has been used to detect class-specific properties, allowing specific reconstruction processing. One common approach is the use of 3D models of a given object class (e.g., cars) to train priors for reconstruction, often resulting in good quality voxel reconstructions [HSP14] or meshes [YBCLS13]. While our object-class-specific smoothing priors have similarities in spirit with e.g., the class-based normal distributions of Häne et al. [HSP14], we focus on the use of a standard Structure-from-Motion (SfM)/MVS reconstruction pipeline, without the need for training based on 3D models.

4.3 Overview

IBR for reflections on cars raises three challenges outlined in Fig 4.2a-c. We first need to provide car *geometry* that is as complete and smooth as possible, including a window surface. We then need to efficiently *compute flow* for reflections on windows taking

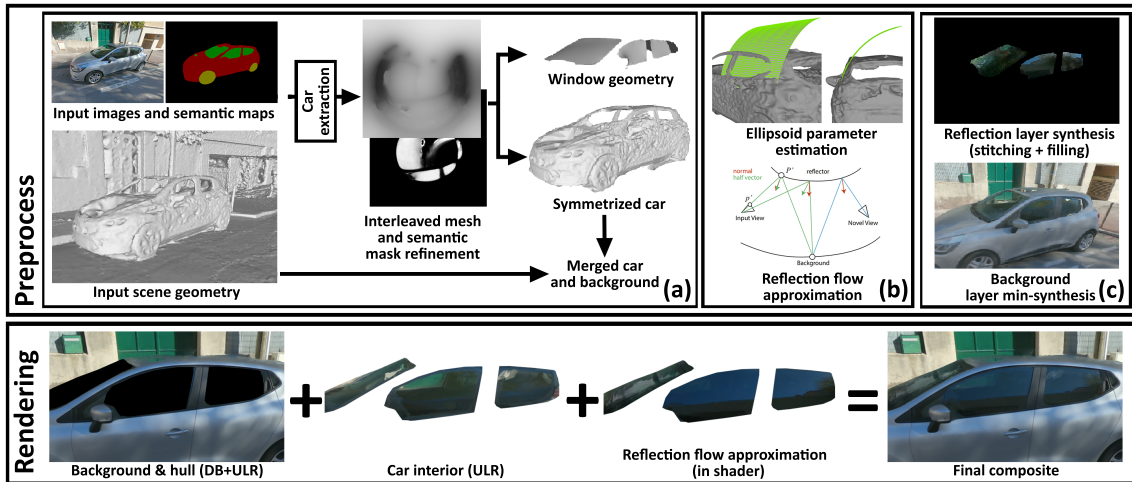


Figure 4.2 – Overview of our method. Top: (a) We isolate cars and refine the car mesh using a spherical projection, exploiting semantic labels to impose smoothing priors in window regions. Mesh smoothing is iteratively interleaved with semantic mask refinement, and approximate surfaces for windows are produced (Sec. 4.4). The second step (b) automatically fits an ellipsoid to the approximate window surface, which is used by our reflection flow computation (Sec. 4.5). The flow is also used in our final preprocessing step (c), together with the refined mesh to synthesize reflection and background layers (Sec. 4.6). Bottom: the layers and mesh are used during interactive navigation to synthesize novel viewpoints with plausible reflections, by computing reflection flow on the fly using the estimated parameters.

their curved nature into account. Finally, we need to *separate layers* for reflections and background so we can flow them separately during free-viewpoint navigation. We successively address each of these challenges in our method; each step produces the input necessary to provide a solution for the next challenge.

For the first challenge we use semantic labels to identify cars and car windows in input images, using powerful modern machine learning-based segmentation. Unfortunately, these semantic labels are inaccurate and not multi-view consistent. Our key ideas are to use a spherical projection of the car and perform interleaved mesh smoothing and multi-view consistent label refinement in this spherical space. This space fits well with multi-view consistency operations, and facilitates the use of powerful image-processing methods. This step produces a smoothed and complete car body, including a first estimation of the window surface, see Fig. 4.2a and Sec. 4.4.

For the second challenge we introduce an efficient reflection flow computation based on analytic approximation of curved windows. We fit ellipsoids to each window by exploiting the previously estimated window geometry and provide efficient flow computation by gradient descent in the shader; Fig. 4.2b and Sec. 4.5.

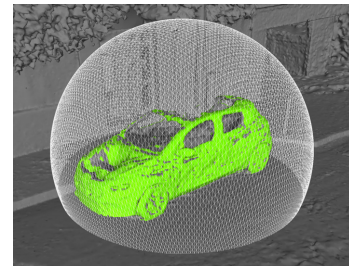
For the third challenge, traditional layer separation algorithms (Sec. 2.2.1, [SAA00, XRLF15]) are not designed for the curved window surfaces and the low quality of the lightweight capture data we acquire. Instead, we introduce a plausible reflection layer *synthesis* algorithm. We use our approximate reflection flows as an initialization, and then use image stitching to complete the synthesis; Fig. 4.2c and Sec. 4.6. We now detail each of our solutions, starting with the geometry extraction and refinement.

4.4 Car Geometry Extraction and Refinement

To perform high quality rendering for cars including window reflections, we need to isolate the cars in the scene, refine their geometry, and estimate supporting geometry for the windows.

4.4.1 Isolating Cars with Semantic Labels

We will use semantic labels to identify and isolate cars in the scene; the labels will also be used to refine car geometry. We obtain 2D label maps for each input image, using DeepLab-v2 [CPK⁺17], trained on a subset of the ADE20K dataset [ZZP⁺19, ZZP⁺17], to recognize car objects, but also *parts*, namely car body, car wheel and car window. Details of the training procedure are given in Appendix B.2. An example training image and result obtained on one of our input images are shown in Fig. 4.3c,d. We project the segmentation labels onto the geometry; vertices where at least 40% of the input semantic maps agree on the car label are considered as belonging to a car. We group these vertices in connected components using mesh and mask connectivity information. This extraction is robust due to the overlap between input images and the fact that the cars are the focus. For each remaining component we estimate a bounding box aligned with the main axes of the car using PCA, and initialize the bounding sphere used for



the spherical projection (see figure on the right).

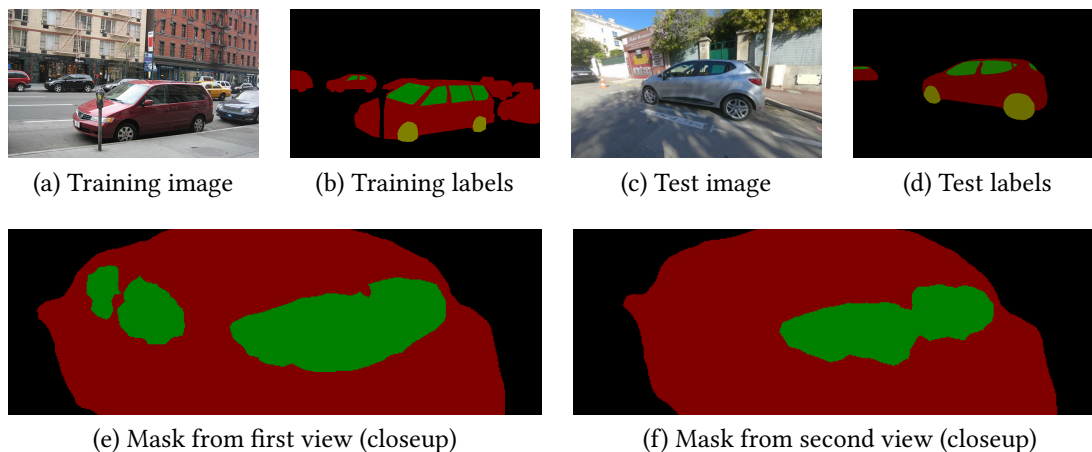


Figure 4.3 – We train a deep-learning based segmentation classifier. Example of a training image (a) and associated hand-labeled segmentation map (b). Segmentation (d) obtained for one of our input images (c). The masks of two neighboring views (e) and (f) are not consistent.

4.4.2 Smooth Car Hull Extraction and Semantic Mesh Refinement

We want to estimate a smooth version of the car body with filled holes, obtain an initial approximation of the car window surfaces and improve overall car reconstruction. Unfortunately, car bodies are badly reconstructed by state-of-the-art MVS reconstruction algorithms, and car windows are often completely missing.

Semantic labels provide an indication of the location of the windows, and could serve as a guide to refine geometry. The segmentations sometimes contain errors, possibly because our viewpoints are very different from the training set images, e.g., close-ups where only a small region of the car is visible. Another case concerns objects lacking features, e.g. a black car in shadow, or with contradictory features caused by reflections, lead to missing regions in some predicted maps. In addition, labels are not always multi-view consistent, see Fig. 4.3e,f.

A precise model of the car and windows could help improve multi-view consistency, but this model is precisely what we are trying to obtain. To solve this dependency problem, we iteratively estimate geometry, interleaving mesh smoothing with updates of *semantic mask probabilities*, i.e., the probability that a pixel has a given label.

Smoothing the mesh in object space is difficult, since semantic labels from input views reproject incorrectly on the mesh through the window holes. However, we know that cars have a spherical topology, and it is natural to use a spherical projection of each car for multi-view consistent window segmentation. We choose to enforce spherical topology as a prior in “spherical” image space, which allows the use of efficient image-processing algorithms and facilitates multi-view coherence.

We start by projecting the geometry assigned to the car onto the bounding sphere using a spherical projection and create a depth map (Fig. 4.4a). We reproject the semantic “car window” labels into the same space, estimating a semantic label probability map using the reconstructed geometry (Fig. 4.4b). We next use the semantic labels to estimate a smooth car hull, fill the window holes and repair inaccurately reconstructed regions as much as possible. The semantic map is then refined by reprojecting the input view maps using the updated geometry. Akin to an Expectation-Minimization approach, we iterate in an interleaved fashion these two steps. Finally, we refine the semantic masks in a multi-view coherent manner.

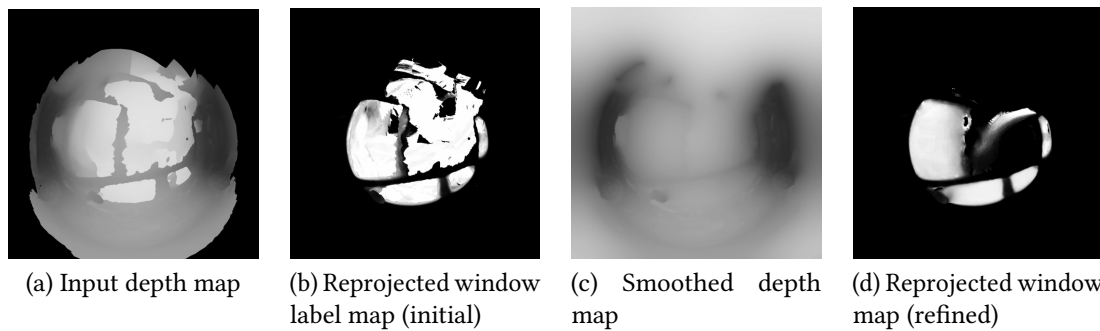


Figure 4.4 – From the input disparity map (a), we refine a smoothed and filled disparity map (c), using the reprojected semantic information as a constraint (b). Semantic labels are then reprojected again (d).

Mesh Refinement Step. To refine depth, regions of the depth map where reprojected labels agree as “car window” are considered with low confidence, while regions that are seen by a high number of cameras without this label have a strong weight. Evidently, the missing window surfaces result in incorrect label reprojected; we prefer and smooth regions that are more likely to be correctly reconstructed car body, and fill the other regions with smooth propagation of the depth. A smoothness prior is thus applied to the

entire hull of the car, and a data term is added on the Laplacian of the depth to encourage planar regions and counteract the tendency of the solver to pull the surface towards the sphere. We use a conjugate gradient method to solve this constrained optimization; details are provided in Appendix B.1. At the end of this step, we obtain a depth map with smoothed reflective surfaces and progressively filled-in window surfaces, Fig. 4.4c.

Semantic Mask Probability Update Step. We can now use this smooth car hull geometry to reproject the label maps with updated visibility. This new label probability map (Fig. 4.4d) is of much better quality than the reprojection using the original reconstructed geometry (Fig. 4.4b). The updated map is then used for the next mesh refinement step.

Final semantic mask refinement. After three interleaved iterations, we refine the label probability map in spherical space, using a Markov Random Field (MRF) guided by color similarity of input views reprojected onto the mesh and confidence, based on label probabilities and visibility. We also discourage well-reconstructed pixels to be labeled as windows. Details of the MRF are provided in Appendix B.2.

Smooth Mesh Extraction & Symmetrization. The final smooth depth map is used to regenerate a car mesh with filled windows and smoothed surfaces, Fig. 4.5b. Regions that were previously holes and deformations caused by reflections and transparent surfaces (see Fig. 4.5a) now have much smoother supporting geometry.

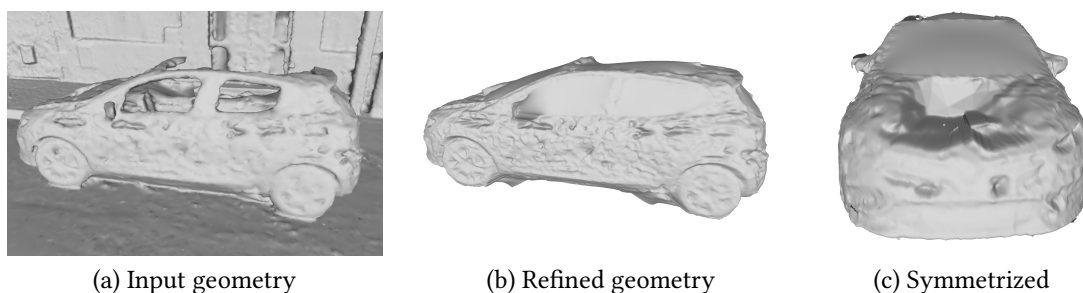


Figure 4.5 – Left to right. (a) Original input geometry from the MVS reconstruction. (b) Refined geometry after the iterative mesh smoothing step. (c) Result of symmetrization. The far side of the car not seen by input cameras in our “street-side” casual capture is reconstructed by symmetry.

In all the examples presented, we have used “street side” capture, with no photographs on the side of the car facing away from the street. We complete the missing information by

generating a symmetrized version of the refined geometry. A copy of the car geometry is reflected along its principal vertical plane, and automatically re-aligned with the initial car mesh using an Iterative Closest Point approach. Both geometries are merged based on visibility, i.e., in regions where the initial car is visible in less than 20% of the cameras, we instead use the mirrored version. The resulting refined and symmetrically completed mesh is shown in Fig. 4.5c.

The final label map (Fig. 4.6a) is then used to cut windows out of the smoothed mesh. Specifically, the parts of the mesh that are labeled as windows are extracted separately (Fig. 4.6b), and the remaining hull of the car is merged back with the initial scene geometry (Fig. 4.6c). Vertices of the initial geometry too close to the smoothed mesh are discarded, to avoid double surfaces. We refer to this windowless smoothed mesh (Fig. 4.6c) as the *refined mesh* or geometry from now on.

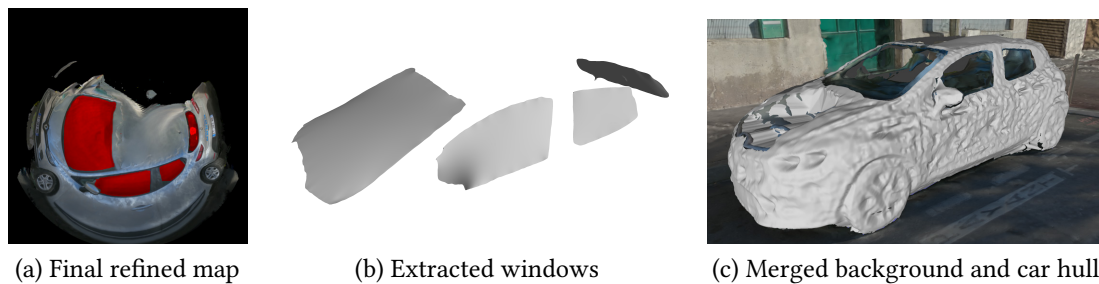


Figure 4.6 – At the end of the MRF step we obtain the final window mask (a, in red, drawn over the reprojected texture). This map is used to separate the window meshes (b) from the merged background/interior and car hull meshes (c).

4.5 Ellipsoid Approximation for Reflection Flow Computation

At a given pixel p in the novel view (Fig. 4.7a), we see a point P_r that is reflected from the background onto the reflecting window at P . We need to find the pixel in a reference input view that contains the reflection of P_r . We propose an efficient algorithm to explicitly compute reflection flow between views. This technique will be used both during preprocess and during rendering. We achieve this using two simplifying assumptions: that scene geometry is distant and that windows can be approximated by ellipsoid geometry.

4.5.1 Reflection flow computation

We assume that the scene geometry reflected in the windows can be approximated by the bounding sphere of the scene. Far-away geometry creates very small parallax when the camera moves, and convex reflectors further decrease the parallax. High quality geometry of the reflected objects is thus not required. Furthermore, each window is approximated by an ellipsoid. This representation will be key to making the problem tractable and achieving real-time performance. At pixel p , we see a point P on the surface of a window.

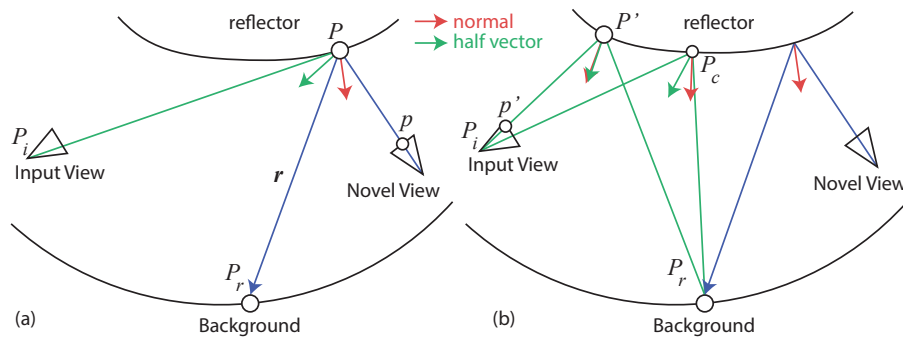


Figure 4.7 – (a) The initial configuration for the reflection flow computation. (b) Two steps of the gradient descent. When the half-vector (green) is aligned with the normal (red), we have found the point corresponding to the reflection in the input view. The value of pixel p' can be used to flow the reflection to the novel view.

We know the position, surface normal and window parameters at P . Knowing the novel view position, we can compute the reflected ray under a perfect mirror assumption, \mathbf{r} . The intersection of this ray with the background at P_r (see Fig. 4.7a), can also be derived analytically using our spherical world assumption.

We then search for the point P' on the ellipsoid reflector surface such that P_r falls inside the input view after being reflected at this point. This point has the property that the normal at the surface and the half vector between the incoming and outgoing reflected rays coincide [EMD⁺05] (see Fig. 4.7b). We find this point using an approximate gradient descent. At a given candidate point P_c , we compute the half-vector between $P_c P_r$ and $P_c P_i$ (where P_i is the location of the input view). If $P_c = P'$ this vector is equal to the normal to the ellipsoid at P_c . Else we update the normal by shifting it toward the half-vector. Thanks to the bijection between ellipsoid positions and their normals, we can easily convert this updated normal to the corresponding new P_c . We can iterate this

process until we find P' ; following this update procedure guarantees that we reach the correct solution [EMD⁺05]. In practice, the algorithm requires fewer than 30 steps to convergence, and can be performed very efficiently in a shader (Algo. 1).

Algorithm 1 Iteratively compute P' , assuming an axis-aligned ellipsoid for brevity

Input: P_i, P, P_r , ellipsoid center C and radii R_e

Output: P'

$P_c \leftarrow P$

$n \leftarrow \text{normalize}((P_c - C)/R_e)$

for $i = 1$ to 30 **do**

$h \leftarrow \text{normalize}(\text{normalize}(P_r - P_c) + \text{normalize}(P_i - P_c))$

$n \leftarrow \text{normalize}(n + 0.2(h - n))$

$P_c \leftarrow C + R_e * n$

end for

$P' \leftarrow P_c$

Once the point P' is found, it can be reprojected in the input view. If it falls inside the window label mask, the corresponding pixel p' can be used as a source for reflection (Fig. 4.7). By doing this computation for all pixels of the novel view covering a window, we can compute the reflection flow to the input view.

This reflection flow computation will be used during the window ellipsoid parameter estimation as described in the next subsection. At runtime we estimate the flow of reflections of each window for a set of input views using the same algorithm.

4.5.2 Ellipsoid Fitting for Car Windows

Each window is approximated by an ellipsoid with longitudinal and vertical radii. Due to shape and physical constraints, the range of admissible curvatures for car windows is quite limited. We start from the “cut-out” window surface (Fig. 4.8a), use the average normal of the window as the third ellipsoid axis, and the projection of the scene up vector onto the window surface as a vertical axis. The longitudinal axis is the cross product of the two previous directions.

Our method is based on feature matching and estimates the radii from the motion of reflections between close-by views of the same window. We fall back to dense image matching when such features are missing – reflections such as the sky or a uniform wall cause only a few moving lines to appear across the window. In both cases, a range

of parameters is swept and the set of radii that best explain the reflection motion is extracted. We use the same range of radii for all windows ($[1m, 40m]$) and sweep it using quadratic steps to sample small values more densely, as small changes to the radii only create noticeably different reflection motion if the radii are small (see Appendix. B.3 for an illustration).

Every reflection is mixed with transmitted colors from the car interior or the scene background, making matching more complicated – yet we only need to estimate two parameters per window. The motion only has to be correctly estimated for a few pixels ; this low dimensionality combined with the range prior make the problem tractable. A detailed description of the method is provided in Appendix B.3.

While the ellipsoid fit is an approximation – since car windows can have small imperfections and normal variations [JHKP13] – the resulting flow is sufficiently plausible in our test scenes, and allows real-time performance. See Fig. 4.8b,c for an example of fitted ellipsoid. We experimented with a planar reflector, but the field-of-view for the reflector is incorrect and results were significantly worse (see Appendix. B.3).

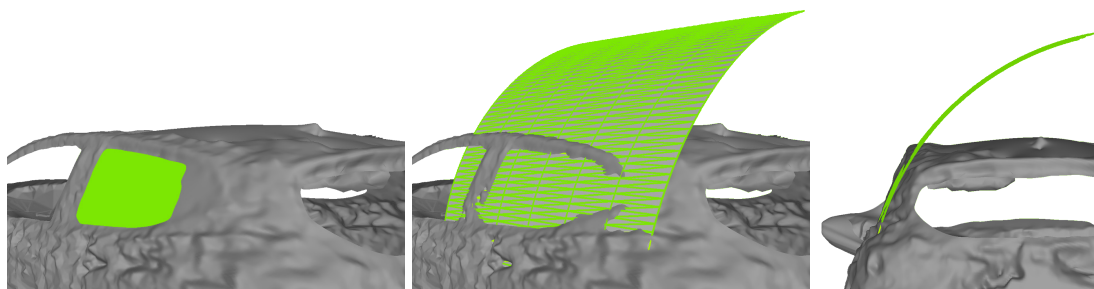


Figure 4.8 – (a) The extracted window surface (in green); (b) and (c) The result of the fitting process (in green).

4.6 Synthesizing Reflection Layers

To render car windows, we need to combine a reflection layer, and the *background* or *transmitted layer* which corresponds to the car interior and the rest of the scene visible through the window. This layer is reprojected using the reconstructed geometry in the scene. The geometry is generally of good quality for the background, and very approximate for the car interior. We use the term *background flow* to refer to the reprojection of

pixels using the refined geometry without windows. The *reflection layer* corresponds to the reflections on the windows, that move according to the flow we compute using our ellipsoid approximation.

Our data is insufficient to achieve accurate reflection layer decomposition; we thus choose to synthesize a *plausible* reflection layer that will be used at runtime. Inspired by min/max compositing [SAA00], we use the min-composite of the reflection flows as a first estimate of the reflection layer, and synthesize a plausible layer by using a variant of image stitching techniques. Consider $l = 1..L$ layers over images I_i with $i = 1..N$. For a given image I_k , the min-composite $M_{l,k}$ for layer l is given as: $\min \mathcal{W}_{i \rightarrow k}^l(I_i)$, $\forall i$, where $\mathcal{W}_{i \rightarrow k}^l$ is the warp or flow of layer l from image i to k . Since light is additive, $M_{l,k}$ is an upper bound on the value of layer l in image I_k .

For the background layer, we use the min-composite of the background flow directly, since this tends to reduce artifacts by preferring darker pixels, Fig. 4.9a. Reliably reconstructing car interiors would significantly complicate the capture process: many images near the car are needed, and the far side must most often be captured, breaking the “street-side” capture context we target.

4.6.1 Reflection Layer Synthesis

For a given input view, we use the ellipsoid approximation for each window to compute reflection flow, reprojecting pixels from 25 neighboring views into it, to create the min-composite of the reflection layer, Fig. 4.9b. This min composite has many visible artifacts: misalignment errors due to inaccurate reflection flow, the presence of moving objects (e.g., the photographer), artifacts due to errors in the mask reprojection and color harmonization discontinuities. The flow of the background layer can be even more approximate, since car interiors have very little reconstructed geometry. We thus see that the standard layer separation approach [SAA00] cannot directly be applied in our context.

Instead, we synthesize a plausible reflection layer by applying standard image stitching techniques [ADA⁺04b] on the min-composite, using a MRF formulation [KSE⁺03]. We use the seam-hiding pairwise term described by Kwatra et al. and a custom per-pixel data cost for each source image s :

$$w_s(p) = L_s(p) + \min_{s' \neq s} |C_s(p) - C_{s'}(p)| + 2 * \min(T, D(p)). \quad (4.1)$$

where $L_s(p)$ is the luminance of pixel p in image s , $C_s(p)$ its color, $D(p)$ its distance to the border and T is set to 1% of the image diagonal. The first term encourages the solver to prefer the minimum value and to remove outliers, the second measures photoconsistency as the smallest L_1 distance to colors fetched from other images (s'), and the third discourages using pixels close to image edges. The resulting stitch is shown in Fig. 4.9c. Sky reflections often have color differences between the various input views, resulting in remaining color harmonization boundaries that we remove with a final automatic Poisson editing step (Fig. 4.9d). The stitched image is used as a data term for both colors and gradients ($w_{\text{gradient}} = 1$, $w_{\text{data}} = 0.01$). However, seams are discouraged by choosing the gradient closer to zero from the source images at both sides of the boundary. The final harmonized reflection layers are then saved to be used for rendering.

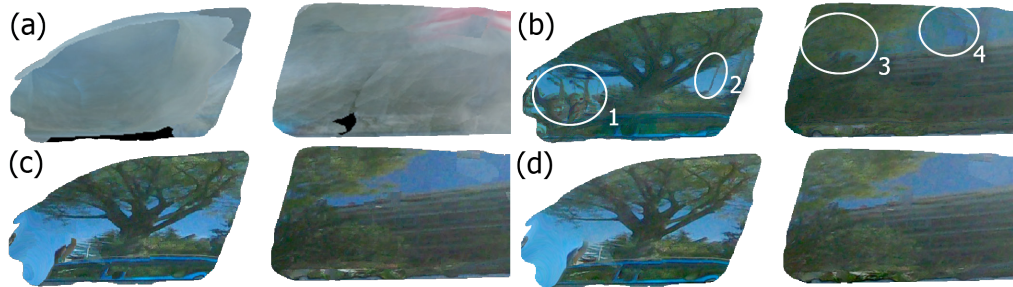


Figure 4.9 – (a) The min-composite for the background. (b) The min-composite of initial reflection flows from the 25 neighboring images to a given input view. Notice artifacts due to (1) presence of the photographer, (2) errors due to incorrect masks, (3) imprecise flows, and (4) color harmonization edges. (c) Our MRF stitching reduces alignment, motion and mask artifacts. (d) An additional Poisson image editing step reduces remaining composite and harmonization artifacts.

4.7 Rendering, Results and Comparisons

4.7.1 Rendering and Implementation

Rendering of a novel view proceeds in two steps. First we render the background of the scene, then we composite in the reflections computed using the reflection flow computation described above.

To render the background of the surrounding scene, we use Deep Blending [HPP⁺18], and a per-pixel implementation of the ULR with a standard weighting scheme [BBM⁺01],



Figure 4.10 – Our scenes, top to bottom: *Vine*, *Carpenter*, *Narrow-Street* and *Corner-Street*. Left to right: input images and semantic masks, scene geometry with the input viewpoints (in blue) and output viewpoints (pink and green), two renderings of each scene using our method. Note the distance from each novel view to the closest input viewpoint (displayed in overlay)).

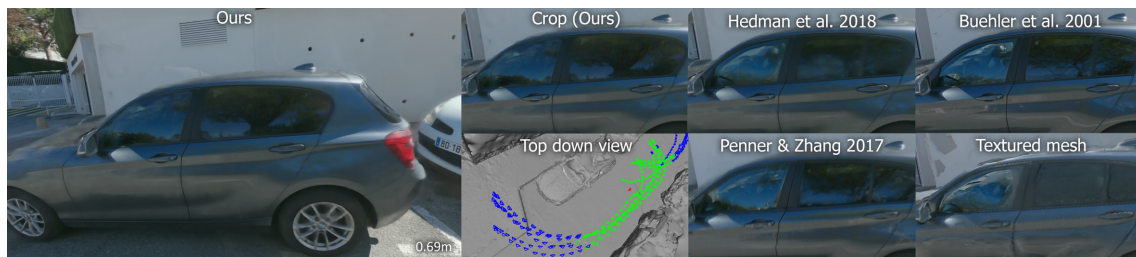


Figure 4.11 – Comparison with recent approaches. Note how our method maintains sharp and complete reflections.

reprojecting input images on the refined mesh. Since our scenes are large, we restrict the per-view mesh voxel grid for Deep Blending to encompass the cars, and use the per-pixel ULR for the rest of the scene.

We then render the interiors and background visible through the window regions, using per-pixel ULR to project the *transmission* layers onto the interior geometry, overwriting any previously rendered pixels

Finally, we render reflections by warping colors from the *reflection* layers using reflection flow. We only warp a subset of the input views, selecting the 50% views closest to the novel view, as reflection layers further away won't contribute significantly to the final reflection. For each selected view, reflection flow is computed on the fly during rendering in a shader for each novel view pixel where the supporting surface of the window is visible. The background intersection and gradient descent on the ellipsoid are used to find the corresponding pixel in the input view. We additionally check that the warped pixel falls inside the same window region in the input view (using the supporting surface again). The warped reflection information of the selected views is blended, and the result is composited with the background using an alpha value of 0.75, which we found experimentally to work well with all scenes. We apply a small blending falloff at boundaries between the two regions.

We show results of our method with our unoptimized C++ and OpenGL implementation. All tests reported here were run on an Intel Xeon 5118 (48 logical cores) with 96GB of RAM and a NVIDIA GeForce RTX 2080 Ti.

4.7.2 Results

We present results on four scenes *Vine*, *Carpenter*, *Narrow-Street*, and the *Corner-Street* (Fig. 4.10), with the latter two containing two processed cars. The number of photos for each scene is respectively 177, 200, 360, and 330, captured using a GoPro (Hero 6) in burst mode, giving 2 photos per second, while the user walks around the car 4 times at different heights. Capturing a car takes about 5 minutes. The input resolution (after camera calibration) is respectively 2720x1607, 2768x1639, 2864x1695 and 2200x1305. Car geometry extraction and ellipsoid estimation are performed using images resized to 1920px width.

Results are shown in Fig. 4.10, 4.11, and in the videos available on the project web page ¹. The effect of reflections is best perceived when moving the viewpoint. In the paths shown

¹<http://www-sop.inria.fr/reves/Basilic/2020/RPHD20/>, <https://repo-sam.inria.fr/fungraph/ibr-cars-semantic/>

in the videos, our novel camera is on average at 0.75m from the closest input camera, with a maximum at around 2.26m.

Our interactive renderer reaches 8Hz (120.0ms) at 1280x720 resolution. Rendering time is distributed as follows on average: background rendering using DeepBlending and ULR: 108.0ms, car interior rendering using ULR on the interior min-images: 3.0ms, reflection rendering and compositing: 8.5ms. Preprocessing for our method on a scene with 200 images takes 10min for mesh and segmentation refinement, 20min for the ellipsoid parameter estimation, 1h10min for reflection layer stitching and 10min for Poisson editing, in addition to standard off-the-shelf SfM/MVS (Colmap) and preprocessing for DeepBlending.

4.7.3 Comparisons

We performed comparisons with the following alternative methods (see Fig. 4.11): a textured mesh, generated by Colmap [SF16, SZPF16] and textured with RealityCapture [Rea18], a per-pixel ULR method using the same mesh, Soft3D [PZ17] and the DeepBlending [HPP⁺18] method. For fairness, we retrained DeepBlending with our scenes to achieve the best possible results. We have included Soft3D for the scenes *Carpenter*, *Corner-Street* and *Vine*, while the other methods are provided for all scenes.

For the vast majority of cases, our method provides a much cleaner and plausible result compared to previous methods. The fact that windows are filled and that reflections move in a plausible manner are key elements of realism for navigation in these scenes. Soft3D performs well when we are close to the input cameras, but degrades rapidly as we move away from the input views. DeepBlending improves the overall result compared to ULR, but still cannot completely recover from the lack of window geometry, and cannot infer reflection flow.

4.8 Conclusion

Limitations. Our method only produces *plausible* renderings of reflections and transmission for car windows, especially for the car interior.

The inaccuracy of the interior geometry is a limiting factor for the quality of rendering we can achieve. This is shown in Fig. 4.12(top row) and the video on the project web page. The feature-based and dense ellipsoid estimations both require at least some reflection

information; windows in scenes under very strong sunlight might not contain enough reflections for this step. In the specific configuration where there is a strong discontinuity in a highly transmissive area (typically dark car interior over a bright background with some reflections over the dark areas) our reflective stitching method is not very successful, resulting in rendering artifacts such as ghosting of the interior or duplicated car frames (Fig. 4.12). This is due to the ambiguity of the min composite in this configuration.



Figure 4.12 – Top left: novel view rendering with our method. Top right: crop of the view; rendering artifacts are visible. Lower left: closest input view; there is a strong discontinuity on a bright background with some reflections present. Lower right: reflection layer min-composite in this case.

The robustness of some steps (geometry refinement, ellipsoid fitting) could be improved through learning-based approaches that could extract automatic features more resilient to variability in the input data. Our method also inherits limitations from the rendering algorithms used for the background and car bodies. Those parts could benefit from future work on geometric and material priors to render broader specular effects with high fidelity. Despite these shortcomings, compared to all previous methods our plausible rendering, and the reduction of the most visible artifacts is a major step forward to allowing usable IBR in a cityscape navigation context.

Summary. We have presented a new method that allows plausible rendering of cars with view-dependent effects, in a casual capture context using a single consumer camera. Our method is based on the introduction of an efficient reflection flow computation that can be computed in real time in a shader, using an analytical approximation of

curved car window surfaces. We create a smooth car hull, filling the windows that are missing in the MVS reconstruction, efficiently enforcing spherical topology using image processing operations. The first approximation of the window surfaces is used to support the ellipsoid fit for the car windows, enabling the efficient reflection flow computation. The final component is the use of the reflection flow for a reflection layer synthesis algorithm, based on image stitching operations. To summarize, we have presented a first solution allowing plausible IBR of cars and in particular car window reflections. Our method makes a significant step forward in allowing applications requiring realistic free-viewpoint navigation in cityscapes to use IBR.

The results obtained also showed us that when the geometry is controlled – thanks to approximations or in the case of synthetic scenes –, computing the flow of reflections is a feasible task and greatly help user perception. This motivated our study of view-dependent effects in synthetic scenes and how to precompute and reproject them, that lead to the work presented in the next chapter. We will show that by taking inspiration from image-based rendering techniques, we are able to reproject precomputed complex light paths in the scene by relying on an approximate reflection flow. Combined with a reconstruction filter, this results in high quality global illumination in static scenes.

Glossy Probes Reprojection for Global Illumination

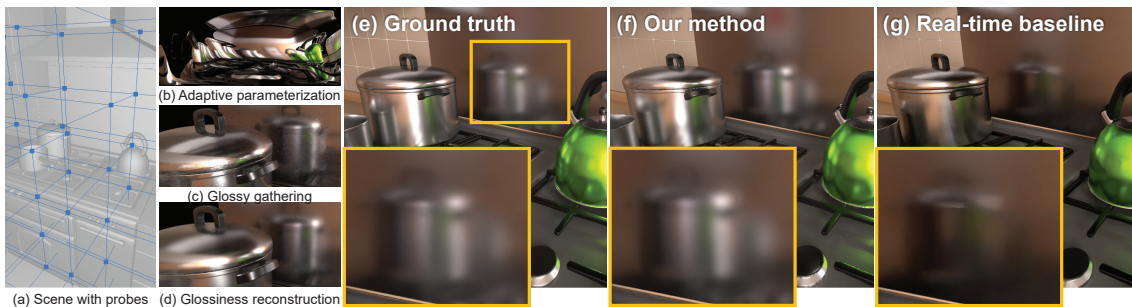


Figure 5.1 – Complex glossy light paths are hard to render interactively, even with modern GPUs. We precompute a set of probes in a 3D grid (a) storing such light paths, and reproject information in real-time to render novel views. We introduce: (b) An adaptive parameterization of probes allocating resolution to glossy materials and complex geometry; (c) A gathering algorithm based on path perturbation theory to accurately reproject glossy reflections to the novel view; (d) High quality glossiness reconstruction to correctly treat reflection occlusion boundaries. We achieve interactive walkthroughs (f) for static scenes with opaque objects, with quality close to the path-traced ground truth (e) and better than a real-time GPU ray tracing baseline (g).

5.1 Introduction

In the two previous chapters, we focused on captured scenes and treated view-dependent effects such as reflections. We now turn our attention to *synthetic scenes*, introducing a new approach for precomputing and reprojecting complex view-dependent light paths, inspired by image-based rendering.

Interactive global illumination has been a major goal of computer graphics since its inception. The introduction of GPU-accelerated ray tracing [Bur20, NV18, WM19] brings closer the prospect of real-time, physically-based global illumination. Current hardware can create a G-buffer [ST91] and trace specular paths very efficiently. However,

more complex, typically longer and glossy light paths are still very expensive: see for example the path with multiple glossy bounces in Fig. 5.1e. As describe in chapter 2 (Sec. 2.2.3), numerous techniques have been proposed to estimate the contribution of such paths in real-time. We now present a novel method where such view-dependent glossy paths are precomputed at predefined scene locations and reprojected. When augmented with view-independent paths stored in a traditional light map, our solution enables full global illumination in interactive walkthroughs of static environments containing opaque surfaces (Fig. 5.1).

Numerous real-time global illumination approximations have been explored [RDGK12], often storing direct light or irradiance in light probes [MMNL17] and approximately reconstructing a subset of light paths with various heuristics [RS09]. These methods achieve impressive results, but can be less efficient on more expensive, glossy light paths. We take a different approach inspired by image-based rendering techniques, precomputing *all* light paths and carefully handling storage and reprojection for each novel view.

Our approach relies on a simplifying assumption. We split light paths, treating view-dependent and view-independent light differently. Separating paths has a long history in graphics [CRMT91, WCG87, SSH⁺98], allowing significant acceleration of illumination computations. We focus on view-dependent paths, while for view-independent paths we use traditional light maps.

For such view-dependent paths, i.e., paths from the eye through several glossy bounces, we precompute a new kind of light probe to store them. While precomputing all light paths can enable interactive rendering of realistic lighting, *reprojecting* this data into novel views raises three main challenges. First, the dense angular sampling needed to capture view-dependent effects can impose high memory requirements, since glossiness and complex geometry imply the need for denser sample rates. Second, reprojecting glossy probe samples into a novel view can be challenging and costly. This is because complex reflector and reflected geometry/materials make it hard to find the best samples in the probes for a given novel view. Third, directly reprojecting paths can cause sharpening at glossy reflections of occlusion boundaries, as parallax changes between the probe position and the novel view can sharpen the precomputed blur.

Our work described in this chapter addresses these three challenges. Using Heckbert's

[Hec90] notation, we store $L(S|D)*DE$ in a light map and we store $L(S|D)*S^*E$ paths in light probes (here S signifies any non-diffuse reflection). Our approach has three main contributions:

- An adaptive light probe parameterization to increase resolution depending on scene geometry and material properties, reducing the overall memory footprint.
- An algorithm using efficient reflection estimation and on-the-fly search to *gather* view-dependent texels from probes, providing high-quality interactive rendering of glossy paths.
- To avoid sharpening at glossy occlusion boundaries, we introduce a new approach that splits the convolution effect of the BRDF into two steps. First, we render the probes using materials with lower roughness in precomputation, and second, during rendering we apply efficient, adaptive-footprint bilateral filtering reproducing the original material roughness.

Our algorithm plausibly approximates ground truth illumination, with complex light paths, at interactive rates for scenes with opaque objects (Fig. 5.16-5.18). We compare to modern hardware-accelerated ray tracing baselines: a light map with real-time glossy ray casting and real-time path tracing. We also compare with light-probe based illumination [MMNL17] and image-space gathering [RS09]. Overall, we show better quality compared to ground truth, when other methods run at the same framerate as ours.

5.2 Related Work

In chapter 2, we reviewed previous work on real-time reflections and probe-based rendering (Sec. 2.2.3), and the reuse of existing shaded samples in synthetic scenes (Sec. 2.1.4). We also investigated efficient ways of estimating the flow of reflections (Sec. 2.2.2); inspired by our results from previous chapters, we leverage them here once again. In this section we present the aspects of previous work that more closely relate to our novel approach described in this chapter.

Light maps are used widely in games, and recent research efforts have explored efficient light map approximations [LTH⁺13, LWS19]. Our work is largely orthogonal, and we use diffuse light maps computed offline with a modified path tracer.

Reusing Samples for Rendering. Several methods have been developed to precompute and reuse samples for synthetic scenes (Sec. 2.1.4). Shading and final images generation can be decoupled [WDP99, WS99, DWWL05], or samples from previous frames reused to improve rendering quality [Kar14, SKW⁺17]. While our work relies on sample reuse, our preprocessing is more exhaustive than these solutions.

Other methods have focused on reusing samples for view-dependent effects, under a small baseline assumption [LR98, LRR⁺14]. In contrast, our work uses light probes for specular and glossy effects, requiring a more involved approach to reproject these paths to the novel view. Nonetheless, IBR techniques inspired our combination of precomputed data feeding interactive rendering.

We additionally draw inspiration from image space gathering [RS09] (Sec. 2.2.3). to overcome the sharpening our approach introduces at reflected occlusions draws, but by precomputing all paths – not just perfect reflections – we simulate all lighting at a similar cost to the original screen-space filter.

Real-time Reflection Rendering. Real-time rendering of synthetic reflections has been extensively explored, as we described in Sec. 2.2.3. The computation of the flow of reflections on a surface is a complex problem because of the numerous indirections involved (Sec. 2.2.2). We leverage existing validity criteria for reflection paths [EMD⁺05] and the specular path perturbation framework [CA00b] when reprojecting information from our glossy light probes to the novel view.

Recent ray tracing GPUs allow fast path tracing, especially when coupled with denoising (e.g., [SKW⁺17]). Multiple-bounce glossy paths however require a high sample count to evaluate the BRDF and each path vertex, and thus result in degraded performance, as seen in our comparisons (Sec. 5.7.3.1). In addition, poor reconstruction of dynamic glossy and specular reflections from low-sample renderings is a known limitation of existing denoisers.

Rendering with Light Probes. Precomputed environment maps are often used in production to generate approximate real-time reflections (Sec. 2.2.3). But environment maps only work for mirror reflections by default, and require additional assumptions to support glossy effects. In contrast our approach handles general material properties. We remain inspired by the prevalence and convenience of environment maps and leverage them in our probe system. Light probe memory consumption can also be significant. We

minimize this with an adaptive parameterization, which relates to continuous magnification techniques to obtain spatially-varying resolution [FRS19]. Unlike our solution, they magnify according to a simple foveation pattern, independent of the scene content.

While it is now possible to use probes with geometric data as an intermediate scene representation to answer incoming light queries [MMNL17, WKKN19, HSAS19], handling glossy reflection paths in these methods also requires an increased sample count, degrading performance (see comparisons in Sec. 5.7.3.2). In contrast, thanks to our precomputed probes containing all light paths, glossy reflections are handled naturally by our solution.

5.3 Overview

We introduce a novel approach to interactive rendering of global illumination in static synthetic scenes containing opaque objects, using light maps for diffuse and probes for glossy paths. Our approach is motivated by the results obtained with flow estimation techniques on real world scenes described in the previous chapter. We address three challenges of probe-based glossy rendering: first, *reducing memory* footprint of the probes; second, efficiently and *accurately reprojecting* glossy path information to the novel view and third, *avoiding sharpening* at glossy reflection occlusion boundaries. Our method is outlined in Fig. 5.2.

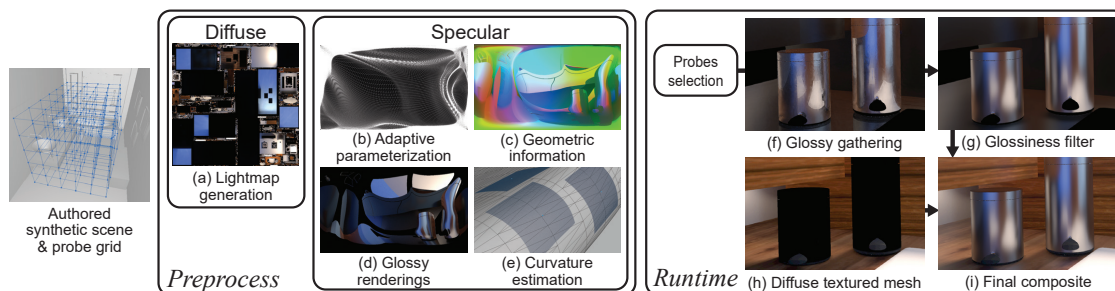


Figure 5.2 – For a given synthetic scene, both diffuse (a) and specular illumination are computed. A series of environment probes are placed on a regular grid and parameterized (b) to capture view-dependent effects. In preprocess, high quality renderings are generated (d) along with additional geometric information for each probe (c, e). At render time, the closest probes are selected and glossy information is gathered at the novel viewpoint (f); glossy effects are then reconstructed (g). The diffuse scene is rendered separately (h) and both layers are composited to generate the final image (i).

In preprocess (middle box in Fig. 5.2), we use a path-tracer to compute diffuse global

illumination stored in a light map (a), while the glossy component of radiance – i.e., $L(S|D)*S^*E$ paths – is precomputed (d) and stored in light probes placed in the scene, far left in figure.

For the first challenge, we maximize the amount of information stored where glossy surfaces are visible by computing a parameterization for each probe with more resolution assigned to shinier surfaces and objects with higher geometric complexity (Fig. 5.2b). We generate this parameterization using quasi-harmonic maps [ZRS05]. We also precompute scene geometric curvature information which is used at runtime for gathering (Fig. 5.2e). A visible geometry map is also generated along with a map containing the reflected positions visible in each direction of a probe (Fig. 5.2c).

Rendering is performed at runtime (right-hand box in Fig. 5.2). We first render the diffuse component as a surface textured by the light map.

For the second challenge, we efficiently render accurate view-dependent paths by introducing a hierarchical gathering approach. We first perform trilinear interpolation between probes. We compute the perfect mirror reflected position visible at each point of the novel view, and reproject it into each selected probe while taking specular motion into account (Fig. 5.2f). We base our approach on specular path perturbation [CA00b], but generalize it to arbitrary geometry using curvature approximation. This estimate provides an initialization for a search process performed in probe space to gather the probe texels best corresponding to the – possibly glossy – reflection. We accumulate these points from the probes and blend them according to the material properties at the reflector surface. The gather process is critical to the success of our approach, since it renders our method robust to inaccuracies in the reprojection process and finds the best available data in the probe.

The third challenge occurs because naively reprojecting glossy reflections from the probes can create a sharpening effect at occlusion boundaries in the reflections. To overcome this issue, we introduce a new approach that separates the convolution effect of BRDFs into two steps (Fig. 5.2g). We first reduce the roughness of materials in the probe precomputation, and apply bilateral filtering in screen space during rendering. Importantly, we estimate a footprint for the screen-space filter that closely reproduces the overall glossiness of the original materials.

The entire process is interactive, and reproduces all the light paths in our static scenes

made of opaque objects.

5.4 Probe generation and storage

We store glossy paths in *light probes*, and reproject them to novel views when rendering. We first describe the per-probe data and how we compute it. We then present our adaptive parameterization that concentrates probe texels in important regions, reducing memory usage at a given image quality. We also describe additional geometric information needed as part of real-time rendering.

We place probes on a regular 3D grid in the scene [MMNL17]. After exploring various adaptive probe placements [WKKN19], we found regular sampling gives the highest quality at a given probe budget. This is because our reflection estimation (Sec. 5.5.1) works best with minimal distance between probes and the novel view. Regular sampling minimizes the maximum distance, while adaptive placement naturally samples some areas sparsely. Yet, we believe that studying probe placement based on the material content in the scene is an interesting topic for future work. Further insights could be derived that might also help guide capture for reconstruction and rendering of real world scenes. Additionally, synthetic scenes are a convenient way to test and validate choices for reprojection and blending of input data in a general image-based rendering context.

5.4.1 Per-probe Data

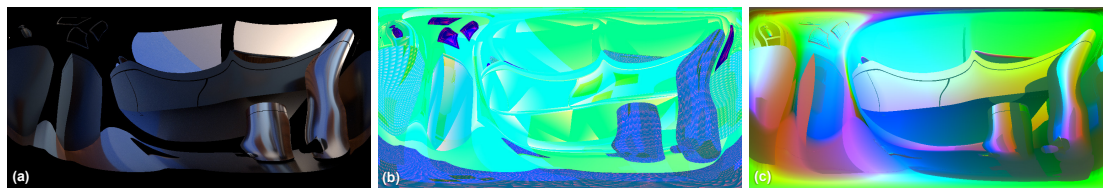


Figure 5.3 – Per-probe data: (a) glossy color, (b) reflector geometry information (triangle IDs and barycentric coordinates), (c) reflected positions (and reflected material ID in the alpha channel).

For each probe, we render a map storing surfaces visible from the probe (Fig. 5.3b); this map stores triangle ID and barycentric coordinates used to reconstruct surface attributes at runtime (e.g., position, normals or curvature). We then render an environment map containing a 360° view from the probe location, storing only the glossy color at visible

surfaces (Fig. 5.3a). We precompute this map with a modified version of the *Mitsuba* path tracer [Jak10] where we force all first-bounce rays to sample the glossy BRDF lobe and take our adaptive parameterization into account. In practice any renderer can be used, depending on desired probe quality. In a third map, we store the geometry seen with one-bounce reflection as if every surface acted as a perfect mirror (Fig. 5.3c).

5.4.2 Probe Parameterization

At a high level, we aim to allocate probe resolution preferentially to regions where it is most needed. Three requirements drive our texel allocation. (a) Smoother surfaces require higher resolution for reprojected reflections, while we can reconstruct rough materials from fewer samples due to their lower frequencies. (b) Distant surfaces or those seen at grazing angles have lower effective resolution. Probe queries may occur from novel views with different perspectives requiring higher resolution. (c) High frequency geometric content exhibits lots of variation in reflected radiance, which needs higher resolution for reliable capture.

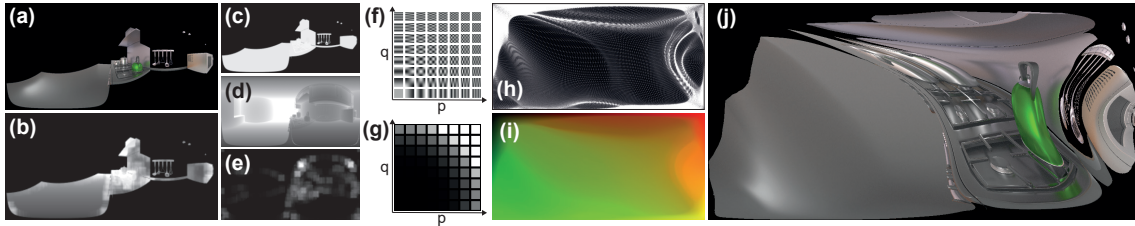


Figure 5.4 – Overview of our probe parameterization pipeline. Conventional content-agnostic spherical parameterizations (a) of a glossy rendering allocate resolution budget suboptimally. This map is never rendered in our approach and shown here only for illustrative purposes. Instead, we design an adaptive resolution map (b), combining local information on surface parameters (c), foreshortening (d), and geometric complexity (e). The latter is estimated by convolving a G-buffer with DCT basis functions (f) – here shown for $N = 8$ – which are then aggregated using a weighting scheme (g) to detect high-frequency variation. We use quasi-harmonic maps to convert the adaptive resolution map into a corresponding adaptive parameterization (h). This induces an inverse flow field (i) – here 2D lookup coordinates are visualized using red and green channels – which we use to steer rays, obtaining a probe with spatially varying resolution (j).

We start from a latitude-longitude (lat-long) parameterization (Fig. 5.4a) and modify it driven by the requirements above. We could start from other panoramic parameterizations, with few differences, but we chose lat-long as it allows easy lookups from ray directions

and the map is a single image (i.e., not a cube-map). We proceed in two steps: first we compute an *adaptive resolution map* m (Fig. 5.4b) to encode relative resolution needs of various probe directions, then we use the map to compute an adaptive parameterization (Fig. 5.4h).

5.4.2.1 Adaptive Resolution Map Computation

To construct the map, we render four low resolution buffers, via ray casting, containing: (i) material smoothness m_{mat} , (ii) depth m_{depth} , (iii) normal m_{norm} , and (iv) facing angle m_{face} , the dot product of incident ray and surface normal. We render these buffers at 256×128 pixels.

The adaptive resolution map m in Fig. 5.4b stores:

$$m = m_{\text{mat}} (m_{\text{size}} + m_{\text{complexity}}).$$

Here, m_{mat} adapts resolution based on material smoothness (Fig. 5.4c), satisfying requirement (a), above. Diffuse texels have $m_{\text{mat}} = 0$ so no space is allocated for them, increasing resolution for shiny surfaces. m_{size} (Fig. 5.4d) considers distance and orientation (b) and is computed as:

$$m_{\text{size}} = \frac{m_{\text{depth}}^2}{m_{\text{face}}} \cos(\theta_{\text{long}}).$$

This term converts probe area to actual object size, compensating for perspective and angular foreshortening akin to a form factor. θ_{long} is the longitude angle, which compensates for size variations induced by the lat-long base parameterization.

Finally, requirement (c) calls for an estimate of local geometric complexity. Simple gradient-based estimates on our buffers are not meaningful, as their response increases around any discontinuity, including (curved) edges, which we do not consider “complex” for requirement (c). Therefore, we perform a simple frequency analysis (Fig. 5.4e) as follows:

$$m_{\text{complexity}} = \frac{1}{N^2} \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} w_{p,q} \|\mathbf{b}_{p,q} * m_{\text{norm}}\|_1.$$

Here \mathbf{b} are basis functions of the 2D discrete cosine transform (DCT) [ANR74], with $[p, q]$ integer 2D frequency vectors (Fig. 5.4f). See Appendix C for more on the choice of DCT. Convolution of the basis functions with the normal buffer analyzes frequency content of local neighborhoods, and the 1-norm sums absolute responses of the three normal

map dimensions. We weight responses by $w_{p,q} = \|[p, q]\|^k$ to ensure higher frequencies contribute heavily to our geometric complexity measure (Fig. 5.4g). In practice, we set $N = 16$ and $k = 5$. We normalize both m_{size} and $m_{\text{complexity}}$ by a per-probe mean to ensure equal effective contribution. Subsequent steps behave more robustly after blurring m with a small Gaussian (i.e., $\sigma = 5$).

5.4.2.2 Adaptive Parameterization

To turn our adaptive resolution map m into a parameterization with adapted resolution, we use tensorial quasi-harmonic maps [ZRS05].

A quasi-harmonic map takes f_h from the plane to the plane, following the quasi-harmonic equation $\text{div}(C\nabla f_h) = 0$. The 2×2 matrix C is a requested first fundamental form, which can vary spatially. In our case $C = m\mathbf{I}$, where \mathbf{I} is the identity matrix. This essentially imposes scaling proportional to m .

We use a regular quad mesh to discretize the domain, where each pixel of m corresponds to a quad. We restrict motion of boundary vertices to the domain boundary. Since the lat-long base parameterization naturally wraps, we force corresponding vertices at the vertical boundaries to move in sync, ensuring parameterization smoothness. Following Zayer et al. [ZRS05] we solve the resulting quasi-harmonic equation iteratively using a sparse matrix solver.

This quasi-harmonic map induces a forward flow, telling us how to move our quad mesh vertices to obtain the desired parameterization (Fig. 5.4h). To determine where to *look up* directions in parameterized probes, we need to invert the mapping, essentially creating an inverse flow field f_h^{-1} (Fig. 5.4i). We do this by rasterizing the deformed mesh with the original vertex positions as colors. This happens at full probe resolution. To render our adaptively parameterized light probes, each pixel looks up its lat-long position using the inverse flow field and we trace a ray through the scene in the corresponding direction. This distorts the probe according to our magnification rules (Fig. 5.4j).

When rendering our glossy light probe, this inverse map specifies each texel’s correct view vector to initiate path tracing. At the first intersection, we only trace paths corresponding to the glossy BRDF lobe, exploiting standard material models’ separation of diffuse and specular components. Both the forward and inverse maps are also used at runtime (see Sec. 5.5) to render the view-dependent layer.

5.4.3 Geometric Information

When gathering glossy light probe samples at runtime (Sec. 5.5), we build on Chen and Arvo’s [CA00a] specular path perturbation. This requires implicit representations of all reflector geometry and various derivatives. To avoid limiting scenes to implicit surfaces, we need additional geometric data to support our approximation described in Sec. 5.5.1.

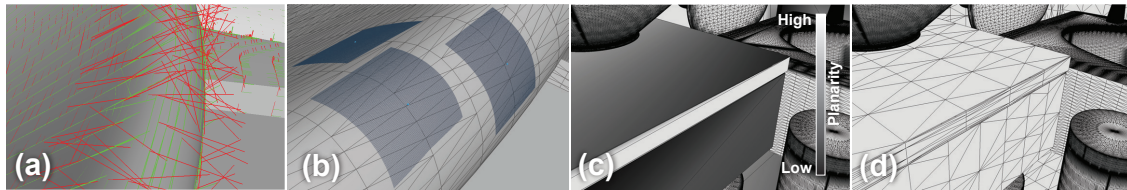


Figure 5.5 – (a) The triangle-based mesh and the estimate curvature vectors: minimal (green) and maximal (red) directions. (b) Visualization of paraboloids fitted using our estimated parameters. (c, d) Maximal planarity estimated using the initial (c) and subdivided (d) objects: the estimation improves.

For each scene vertex we first estimate principal curvature values and directions (Fig. 5.5a) based on discrete neighborhood operators [MDSB03]. We do this separately on each object. For best results, we tessellate large planar objects with only a few large triangles (Fig. 5.5c,d). To get more robust estimation, we regularize curvatures between neighboring vertices when normals deviate less than 40 degrees. We average curvatures by weighting by the normal dot product and ignoring values from boundary vertices. This reduces issues at mesh boundaries (which do not have a full cycle to estimate curvature) and filters smaller, irrelevant curvature variations.

We transfer these values back from the tessellated geometry to each scene vertex, allowing runtime interpolation of the curvature. The principal curvatures allow us to locally approximate the surface by a paraboloid (Fig. 5.5b), for which we can analytically compute our required higher-order derivatives.

5.5 Rendering Global Illumination

To render a novel view, we handle diffuse and glossy components separately, sampling two forms of precomputed lighting (see Fig. 5.6). For diffuse light, we render meshes textured with a standard light map, which contains diffuse global illumination.



Figure 5.6 – Rendering of diffuse (left) and view-dependent (right) components occurs separately. Both rely on precomputed illumination.

For glossy light paths, we reproject our light probes to the current view. To do reprojection, we first rasterize a G-buffer with position, normal, surface curvature, and material (ID and roughness). With a ray caster, we then trace perfect mirror rays at specular pixels, storing reflected hit positions and material IDs. This reflection ray is real-time on modern GPUs. We then compute per-pixel glossy lighting by gathering information from nearby probes. This gather relies on estimated specular flow, but as we gather from *glossy* probes it closely approximates a wide range of material properties. An exception are reflected occlusion boundaries, treated in Sec. 5.6.

For each output pixel containing a glossy surface, we use G-buffer and ray data above plus the probe data (see Fig. 5.3) to select relevant probe texels and merge their contributions. Let p be the current camera position. A pixel sees point x , sampled by our G-buffer, and the mirror ray from x intersects reflected point q (see Fig. 5.7a). To shade x using data from probe P' , we need to fetch the probe sample for x' that reflects point q as seen from probe origin p' (assuming such data exists in P').

5.5.1 On-the-fly Reflection Position Estimation

First, we must determine x' as the camera moves from p to p' . The theory of specular path perturbation [CA00b] allows us to approximate displacement $\Delta x = x' - x$ given displacement $\Delta p = p' - p$ (see Fig. 5.7a). For any reflector represented by implicit function f , we can derive a second order approximation of the path function from Fermat's principle and the implicit function theorem. Then, there exists a Jacobian

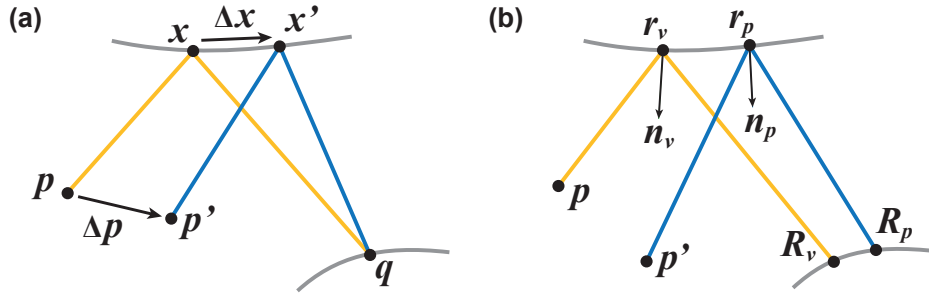


Figure 5.7 – The geometry of path perturbations. (a) From a known specular path (q, x, p) and a probe position p' , a new path (q, x', p') can be determined through local path perturbation. (b) To assess probe samples, we compute a score based on ray information (reflector and reflected positions, reflector normal) associated with the sample $(r_p, n_p$ and $R_p)$ and the novel view pixel $(r_v, n_v$ and $R_v)$

$J(p, q, x, f)$, a 3x3 matrix, and Hessian $H(p, q, x, f)$, a 3x3x3 tensor, such that

$$\Delta x = J\Delta p + [\Delta p]^T H[\Delta p],$$

where $[\Delta p]$ is a 1x1x3 tensor replicating Δp three times. We refer the reader to Chen et al. [CA00a] for a detailed discussion. As we know p, p', x , and q , this gives sufficient data to lookup probe samples.

While Chen et al. [CA00a] require first, second and third order derivatives of f , we do not want to limit scenes to implicit surfaces. We generalize to triangular meshes using our local curvature approximation stored during precomputation (see Sec. 5.4.3) and G-buffer rendering. From these curvatures we estimate a paraboloid to locally fit the surface, and use its analytical derivatives for path perturbation. We only keep points x' that share the same material as x , and reproject their probe samples.

5.5.2 Gathering View-dependent Color

Combining specular path perturbation with our estimated curvature only approximates the specular motion between the novel view and our probe. To be robust to inaccuracies, we explore a neighborhood in probe space before finalizing our sample selection.

We use a two-level search on a grid of decreasing step size and radius, looking for a texel with stored radiance valid at our novel location. The two-level search ensures

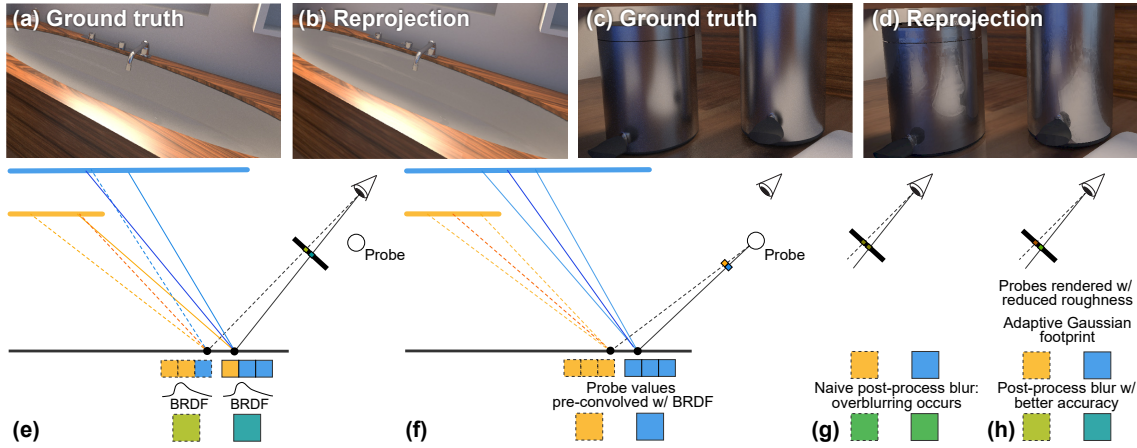


Figure 5.8 – (a) Path traced ground truth of unoccluded glossy reflection where (b) naive reprojection is accurate. (c) Reference reflection with occlusion boundaries; (d) naive reprojection sharpens these boundaries. (e) Ground truth: pixel colors integrate the BRDF over samples from both yellow and blue surfaces. (f) Naively sampling precomputed probes uses only samples from either side of the occlusion, sharpening the boundary. (g) Naive post-process filtering over blurs results. (h) By reducing roughness and estimating the footprint of the post-process filter, we improve accuracy.

thoroughness while maintaining efficiency. Samples corresponding to reflections on other materials are ignored.

We seek to favor probe samples containing information closely corresponding to the novel view surface. We achieve this by evaluating an energy function designed to favor such samples. We use the following notation (see Fig. 5.7b): reflector position r_p , normal n_p , and reflected position R_p as seen in the light probe, and corresponding values r_v , n_v , and R_v from the novel view.

Our energy function considers four criteria. First, view and probe samples (R_v and R_p) preferentially lie on the same surface; if their material IDs differ, we strongly penalize the total energy, multiplying it by $s_a = 10$. We thus use mismatched samples only if no others are available. Second, the reflected hits R_v and R_p should be close; for this we add a term $s_b = \|R_v - R_p\|$. Third, using similar surface normals n_v and n_p ensures consistent lighting; the term $s_c = 1 - (n_v \cdot n_p)$ achieves this goal. Finally, the sample should have a similar reflected ray; we use the following term:

$$s_d = 1 - \frac{(R_v - r_v) \cdot (R_p - r_p)}{\|R_v - r_v\| \|R_p - r_p\|}.$$

Combining these criteria, we use the following energy function:

$$\mathcal{E} = s_a \cdot (\min(s_b, 1) + \min(s_c, 1) + \min(s_d, 1)). \quad (5.1)$$

s_a and s_b have the most effect, but the other terms help fix issues in more uncommon cases, such as non convex reflectors.

Because we adaptively parameterize probes, searching probe neighborhoods with a constant-sized regular grid risks missing details in compressed regions or insufficiently exploring magnified areas. We thus scale the search step size by $\left| \frac{\delta f_h^{-1}}{\delta \mathbf{x}} \right|^{-1}$, where f_h^{-1} is the inverse flow field (Sec. 5.4.2.2). In our two-level neighbor search, our coarse level uses 7x7 samples at 4 texel spacing (before compensation). The fine level searches 3x3 samples with 2 texel spacing, centered at the minimal energy sample found by the coarse search.

To avoid popping during camera motion, we sample reflections in the eight probes nearest the novel view. Each probe selects its sample with minimal energy (Eq. 5.1). and the eight probe samples are combined with trilinear weights t_i into a final pixel color:

$$\mathcal{C} = \frac{1}{Z} \sum_{i=1}^8 t_i \cdot \exp(-\phi \mathcal{E}_i) \cdot \mathbf{c}_i, \quad (5.2)$$

where ϕ is a constant falloff factor we set to 8, and Z is a normalizing constant ensuring that weights sum to unity. When loading colors \mathbf{c}_i from the probes, we use bilinear interpolation when this does not blend colors from different reflectors.

For pixels with no valid sample in any of the eight probes, we temporally reproject information from last frame’s glossy layer. The lowest error \mathcal{E}_i among the eight probes is also stored for our glossy filter pass (see Sec. 5.6.2).

5.6 Two-step Convolution for Accurate Warping of Glossy Probes

For glossy materials, we can reuse samples representing any ray in the BRDF lobe, even if they are not perfect specular reflections; this is similar to Robison and Shirley [RS09]. Generally, gathered probe samples are nearly correct for the novel view, giving satisfactory results (see Fig. 5.8a,b)

However, reflected geometric occlusions often appear “sharpened” when gathered samples straddle these boundaries. To understand why, consider Fig. 5.8. Fig. 5.8e shows that

correct pixel values integrate the BRDF lobe, combining samples from both yellow and blue surfaces. Naive probe reprojection gathers precomputed samples falling entirely on either side of the occlusion, sharpening the blurry boundary (Fig. 5.8d). Simply applying a post-process filter overblurs (Fig. 5.8g), increasing apparent surface roughness. To reduce overblurring, our solution separates the convolution effect of the BRDF into two steps. We reduce the material roughnesses when precomputing probes and then adaptively blur during lookups to approximate the desired glossiness (Fig. 5.8h). We first explain how to estimate the filter footprint so that we match the original material roughness in the final image (Sec. 5.6.1). We then explain how to perform the filter operation (Sec. 5.6.2) and finally our efficient filter approximation (Sec. 5.6.3).

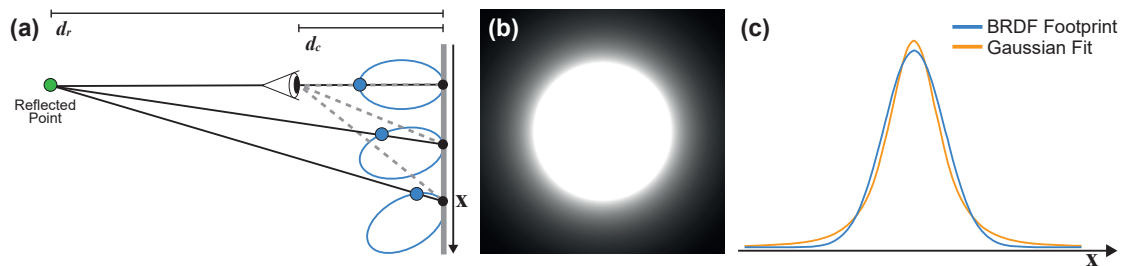


Figure 5.9 – Filter footprint estimation. (a) The setup for estimating our filter footprint. A camera observes a reflected point (green) via a planar reflector. We evaluate the BRDF lobe (blue ellipses) at various positions x and corresponding viewing angles (blue points) to obtain an image-space footprint. (b) The footprint of the GGX specular lobe, for $\rho = 0.075$, $d_c = 2m$, $d_r = 5m$. (c) A 1D slice of the specular lobe and our fit giving a close approximation.

5.6.1 Filter Footprint Estimation

To estimate our filter footprint, we use a simple configuration (see Fig. 5.9a): a camera looking at a plane of uniform isotropic roughness, with normal parallel to the view and reflected point colinear to the camera. We evaluate the GGX BRDF [WMLT07] for each point under a fixed field-of-view, shown in Fig. 5.9b. The parameters influencing the BRDF footprint are the material roughness ρ , distance to the reflector d_c , and distance to the reflected light d_r . Given this, we fit a Gaussian $\mathcal{G}_\rho(\mathbf{x}; \rho, d_c, d_r)$ with covariance matrix Σ_ρ to the image-space BRDF footprint. We can ignore the mean as the Gaussian is centered at zero. We determine Σ_ρ for specific parameters $[\rho, d_c, d_r]$ by sampling the BRDF footprint and evaluating spatial covariance numerically. Such a fit is shown in

Fig. 5.9c along one scanline; we see this approximates the BRDF quite accurately.

We precompute and tabulate covariances in $32 \times 32 \times 32$ bins with roughness varying from 0 to 0.5 and distances from 0.01 to 10m, corresponding to typical values in our scenes. We use a power sampling scheme and decrease the covariance for small d_r (≤ 0.5 m) as we observed fitting overestimated covariance in these cases.

Slanted reflectors foreshorten the BRDF footprint along the slant direction. We dynamically incorporate this when rendering: the projected surface normal in image space gives the foreshortening direction. We then update Σ_ρ by scaling the variance in this direction by the dot product between view direction and surface normal.

Using Gaussian filtering allows splitting our glossy filter into two steps, relying on properties of Gaussians. The steps include \mathcal{G}_ρ , the reflector BRDFs when precomputing glossy light probes, and \mathcal{G}_I , the runtime image filter. Final pixel values are given by the convolution $\mathcal{G}_\rho * \mathcal{G}_I$. We can reduce material roughness during precomputation, giving a new Gaussian $\mathcal{G}_{\rho'}$ with covariance matrix $\Sigma_{\rho'}$ corresponding to the new roughness ρ' . Using the property:

$$\Sigma(\mathcal{G}_1 * \mathcal{G}_2) = \Sigma(\mathcal{G}_1) + \Sigma(\mathcal{G}_2),$$

we find the covariance matrix Σ_I of the image Gaussian \mathcal{G}_I such that the operation $\mathcal{G}_{\rho'} * \mathcal{G}_I$ reproduces the effect of \mathcal{G}_ρ :

$$\Sigma_I = \Sigma_\rho - \Sigma_{\rho'}. \quad (5.3)$$

In practice, we set ρ' to $\rho/2$, and modify our preprocess probe rendering to account for this at the first glossy vertex of each path.

During interactive rendering, for each pixel, we look up the values of $\mathcal{G}_{\rho'}$ for a given initial roughness ρ and distances d_c and d_r . We compute Σ_I using Eq. 5.3, compensate for geometric foreshortening and apply the image filter as detailed in the next section.

5.6.2 Gloss Filtering

Above, we estimated the image-space filter footprint. Now, we filter guided by the geometric data used to estimate colors C in Eq. 5.2:

$$\hat{C}(\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{x}_i \in \mathcal{N}(\mathbf{x})} \mathcal{G}_I(\mathbf{x}_i - \mathbf{x}) w_r(\mathbf{x}, \mathbf{x}_i) \mathcal{E}^{-1}(\mathbf{x}_i) C(\mathbf{x}_i). \quad (5.4)$$

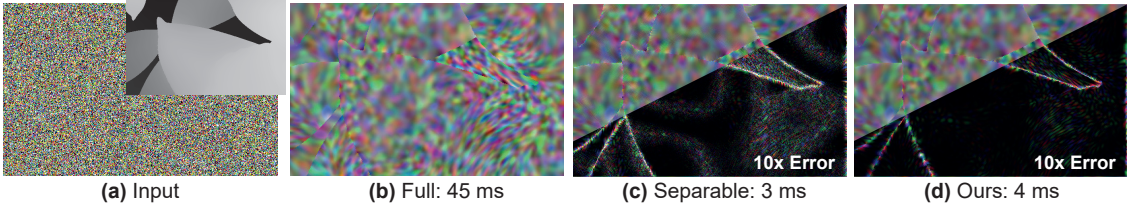


Figure 5.10 – Comparing filter alternatives. (a) Filter input is high-amplitude uniform random noise, which is adversarial for anisotropic edge-stopping filters. The guide image contains hard edges, with smoothly varying anisotropic covariance in each region. (b) Applying a full 2D filter, per Eq. 5.4, is highly inefficient for large kernels. (c) A naive separable implementation is fast, but suffers from strong artifacts. (d) Our four-pass implementation is almost as efficient as the separable version, but reduces artifacts significantly.

Here, $\mathcal{N}(\mathbf{x})$ denotes the filter footprint at \mathbf{x} . \mathcal{E} is the energy function in Eq. 5.1; the inverse acts as a confidence to limit propagation to pixels that match well [KW93]. The range weight

$$w_r(\mathbf{x}, \mathbf{x}_i) = \mathbb{1}_{\mathbf{n}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}_i) > \alpha_n} \cdot \mathbb{1}_{|d(\mathbf{x}) - d(\mathbf{x}_i)| < \alpha_d} \cdot \mathbb{1}_{m(\mathbf{x}) = m(\mathbf{x}_i)},$$

acts as a cross-bilateral term, preventing filtering across normal (\mathbf{n}) or depth (d) discontinuities or between different reflector material IDs (m). We deliberately use indicator functions ($\mathbb{1}$) instead of more canonical exponential cross-bilateral weights [TM98], since the spatial filter footprint already accounts for local geometry using \mathcal{G}_I . We set indicator thresholds to $\alpha_n = 0.8$ and $\alpha_d = 0.2$ in our experiments. Finally, Z is the normalizing partition function, ensuring filter weights sum to unity.

5.6.3 Efficient Filter Approximation

Evaluating Eq. 5.4 is costly for large footprints (Fig. 5.10b), so we employ an approximation to maintain interactivity. A two-pass separable anisotropic filter [GSVDW03] reduces complexity from quadratic to linear, but is only accurate for spatially invariant filter kernels. We observe that within regions defined by w_r the filter parameters vary smoothly, by construction. Naively implementing a separable edge-stopping filter introduces artifacts at edges that do not align with the filter directions [PVV05] (Fig. 5.10c). To mitigate this, we split the filter into two passes, effectively using four 1D separable filters [GO11]. The first and third pass filter along the first principal direction of Σ_I , whereas the others filter along the second principal direction. In all cases, covariances

must be halved to maintain the correct footprint. We observe this very closely matches the full 2D filter (Fig. 5.10d).

5.7 Results, Evaluation and Comparisons

We implemented our approach in our internal framework using C++ and OpenGL. We will release the full source code, including all preprocessing and runtime components.

Specular layers and the diffuse light map were generated with a modified version of the *Mitsuba* unidirectional path tracer at 2048 samples per-pixel. Each scene contains 252 probes placed on a regular grid, each generated at a 1024x512 resolution. We render the novel view at 1920x1080.

To highlight their relative importance, we first evaluate different aspects of our algorithm on four test scenes. We then compare our results to five different methods, including some quantitative comparisons. Results are best appreciated in the videos provided on the project web page ¹.

5.7.1 Test Scenes

We evaluated our method on the scenes shown in Fig. 5.11: *Bathroom*, *Livingroom*, *Staircase* and *Small Kitchen*, all from the Bitterli [Bit16] model repository. The first three contain the original geometry, with some added elements to showcase glossy reflections. *Small Kitchen* contains only a portion of the repository’s scene; the high object count in the original required a large diffuse light map. Solutions for handling large, complex light maps exist, but are orthogonal to our approach and we leave this as future work.

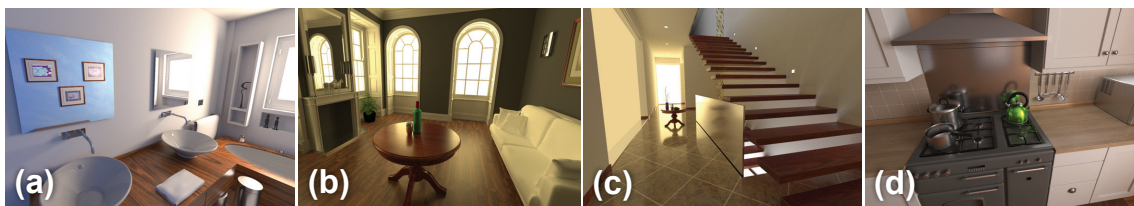


Figure 5.11 – Our four test scenes: (a) *Bathroom*, (b) *Livingroom*, (c) *Staircase* and (d) *Small Kitchen*.

¹will be available at <https://repo-sam.inria.fr/fungraph/synthetic-probes/> after publication

5.7.2 Evaluation

We evaluate four aspects of our algorithm. All figures in this subsection show only our generated glossy layer. Specifically, we explore:

1. the effect of our probe parameterization,
2. the effect of gathering via our approximate path perturbation,
3. the effect of total probe count and the subset used at runtime,
4. and the effect of our glossiness filtering.



Figure 5.12 – Comparing regular and adaptive parameterization when warping just a single probe into the novel view. Left: *Bathroom* scene, right: *Small Kitchen*. For clarity, we do not apply our glossiness filter.

Fig. 5.12 compares results using a standard lat-long probe parameterization and our adaptive method when warping just the single closest probe into a novel view. Adding resolution for reflections viewed at a grazing angle (*Bathroom*) and on high-frequency surfaces (*Small Kitchen*) clearly reduces aliasing and sub-sampling artifacts. The additional resolution improves warping, as geometric information is more accurate. For clarity, renderings in Fig. 5.12 and 5.13 do not include our glossiness filtering.

For gathering using our approximate path perturbation, Fig. 5.13 shows the effects of our paraboloid approximation and using a single level search. We see that using a planar approximation on curved objects leads to large regions where the corrective search fails, while our paraboloid representation gives much improved results (Fig. 5.13 first row). Using only the coarsest search level (Fig. 5.13 second row) we can select sub-optimal probe samples, resulting in a larger number of incorrect samples.

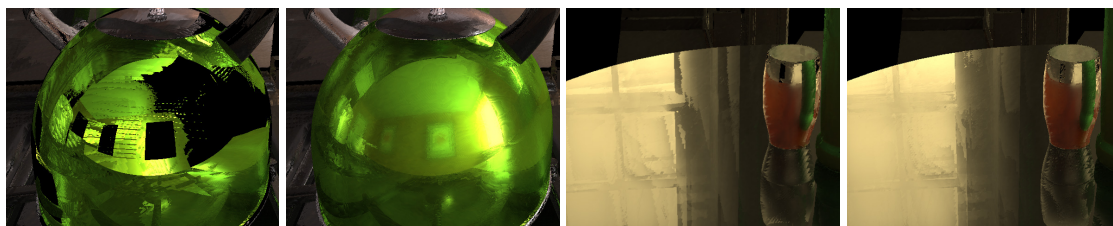


Figure 5.13 – Left using a planar approximation and center-left our paraboloid approximation. Center-right: using a one level search, right our 2-level solution. Please note that glossiness filtering is not applied here.

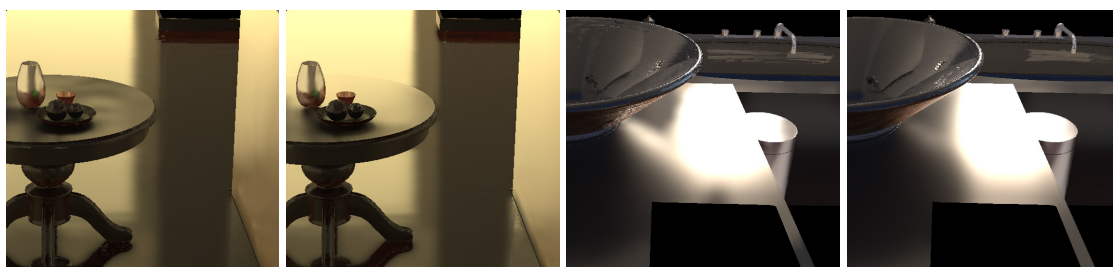


Figure 5.14 – Left, selecting 4 probes instead of 8; center-left: our solution selecting 8 probes. Center-right: total of 126 probes; right our solution with 252. Fewer available probes lead to missing information on some surfaces.

In Fig. 5.14 we show the effect of using fewer probes for reprojection and reducing the number of precomputed probes. In both cases, the reduction prevents more pixels in novel views from discovering relevant probe samples, creating discontinuities and other inconsistencies on glossy surfaces.

Using probes containing full roughness materials, discontinuities are visible at reflected occlusion boundaries. In Fig. 5.15 we illustrate the effects of our two-step glossiness convolution, using probes with halved material roughness. This reproduces the desired material glossiness without artifacts at occlusions.

5.7.3 Results and Comparisons

Fig. 5.16 shows frames from our videos camera paths, together with the corresponding ground truth. We accurately capture complex glossy light paths at interactive frames rates, including complex secondary glossy effects (e.g., the reflection of the top of the bin—row 1, left; glossy reflections of the table—row 4, right). Nonetheless, while our



Figure 5.15 – Left: glossy layered rendered by gathering from probes generated using initial material roughness. The reflections are sharper. Right: our glossiness convolution applied on halved roughness probes approximate the effect of the material full roughness.



Figure 5.16 – Results of our method. For each scene we show two viewpoints rendered with our method and the corresponding ground truth path traced image.

approximation is quite accurate overall, some differences remain (e.g., roughness levels on the floors).

We compare to three baselines and two previous methods, along with path-traced ground

truth rendered with *Mitsuba*. Note: due to issues converting and loading model and material formats, we only compare the *Bathroom* scene across all methods.

5.7.3.1 Comparisons with Baselines.

The first baseline is an image-based rendering (IBR) approach akin to an unstructured lumigraph (ULR) [BBM⁺01] rendering of our probes. We reproject the probes using the scene geometry and use standard ULR weights per-pixel. Evidently, this naive IBR cannot correctly capture reflections, see Fig. 5.18.

The second and third baselines use real-time ray tracing (RTRT) via NVIDIA’s *Falcor* framework [BYC⁺20]. We compare to a real-time path tracer, denoised with the Optix denoiser. We gave this path tracer the same compute budget as our prototype, resulting in 2 paths per pixel. Even though this captures the general structure of light paths, quality and stability are generally lower (e.g., in the reflections on the sink, Fig. 5.17, top right). We also compare to light map rendering augmented by BRDF-sampled rays. Again using the same budget allows for 3 bounces, using 4 samples for the glossy lobes (at the first path vertex) to obtain the best results. This method also provides good results, but misses secondary glossy effects from distant emitters that require a much higher sample count (Fig. 5.17, bottom left). Note, for example, the missing glossy highlight on the table. In contrast, our solution has overall good image quality, even though some small inaccuracies remain in reflections. The quality is best appreciated over the entire paths in the videos.

5.7.3.2 Comparisons with Prior Art.

We also compare to image-space gathering [RS09] and McGuire et al.’s [MMNL17] probe-based method. We reimplemented the former in our framework, using Optix [PBD⁺10] and fetching our light map to generate the perfect reflection image. For McGuire et al. [MMNL17], we adapted the publicly available implementation in the G3D framework [MMM17]. For a fair comparison, we use *Mitsuba* to path-trace 128 regular light probes using their octahedral parameterization at 1024x1024 resolution, giving the same overall pixel count our probes used. We import these probes into G3D and use them for all processing required, e.g., irradiance. We activate their glossy reflections, which trace 8 additional rays in the probes. Both methods were given the same compute budget as

ours. Again, the result is plausible, but glossy effects are missing (Fig. 5.17-5.18).

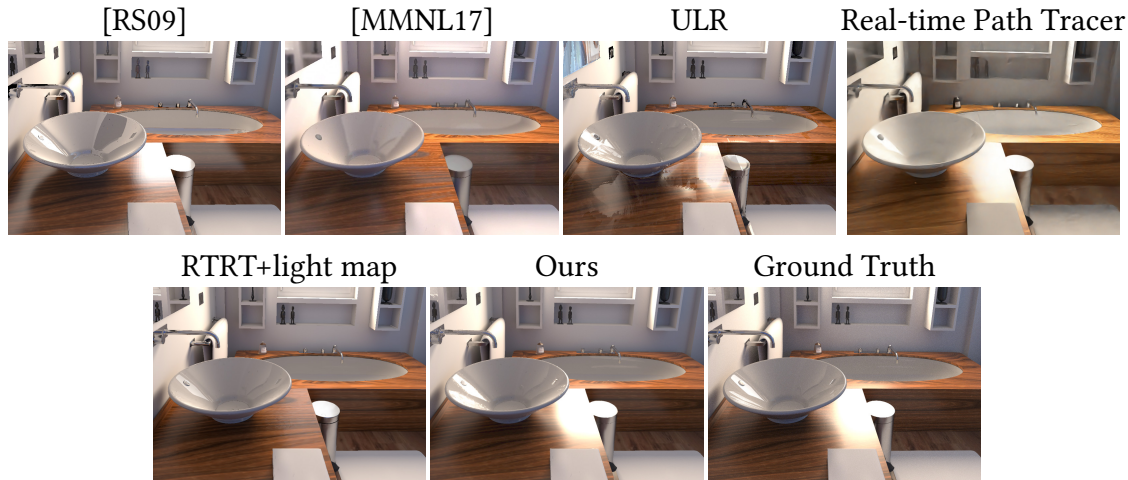


Figure 5.17 – Comparisons with same frame rate for each method. Previous methods: [RS09], [MMNL17]; baselines: unstructured lumigraph (ULR), real-time path tracing using Falcor, RTRT with light map; ours and the path-traced ground truth.

5.7.3.3 Quantitative Evaluation.

We performed a quantitative evaluation using both root mean square error (RMSE) and structural dissimilarity (DSSIM) [LMCB06]. We compute the error between the generated and ground truth glossy layers. Error is averaged over 12 frames sampled regularly along the path recorded in the *Bathroom* scene. Table 5.1 summarizes the error for Robison and Shirley [RS09], McGuire et al. [MMNL17], the RTRT with light map baseline, and our method. The error for our method is consistently much lower than the previous work and baseline.

5.7.3.4 Statistics.

The timings and memory consumption for our method, including preprocessing, are shown in Table 5.2. Rendering times are averaged over the paths shown in the videos. Interactive performance was measured on a computer with an Intel Core i7-7800X processor, 64GB of RAM, and a NVIDIA Geforce RTX 2080Ti. We currently use the *Mitsuba* renderer to precompute probes and diffuse light maps on our cluster. A typical node has a dual Intel Xeon Silver 4110 processor and 192GB of RAM. We could instead use a real-time

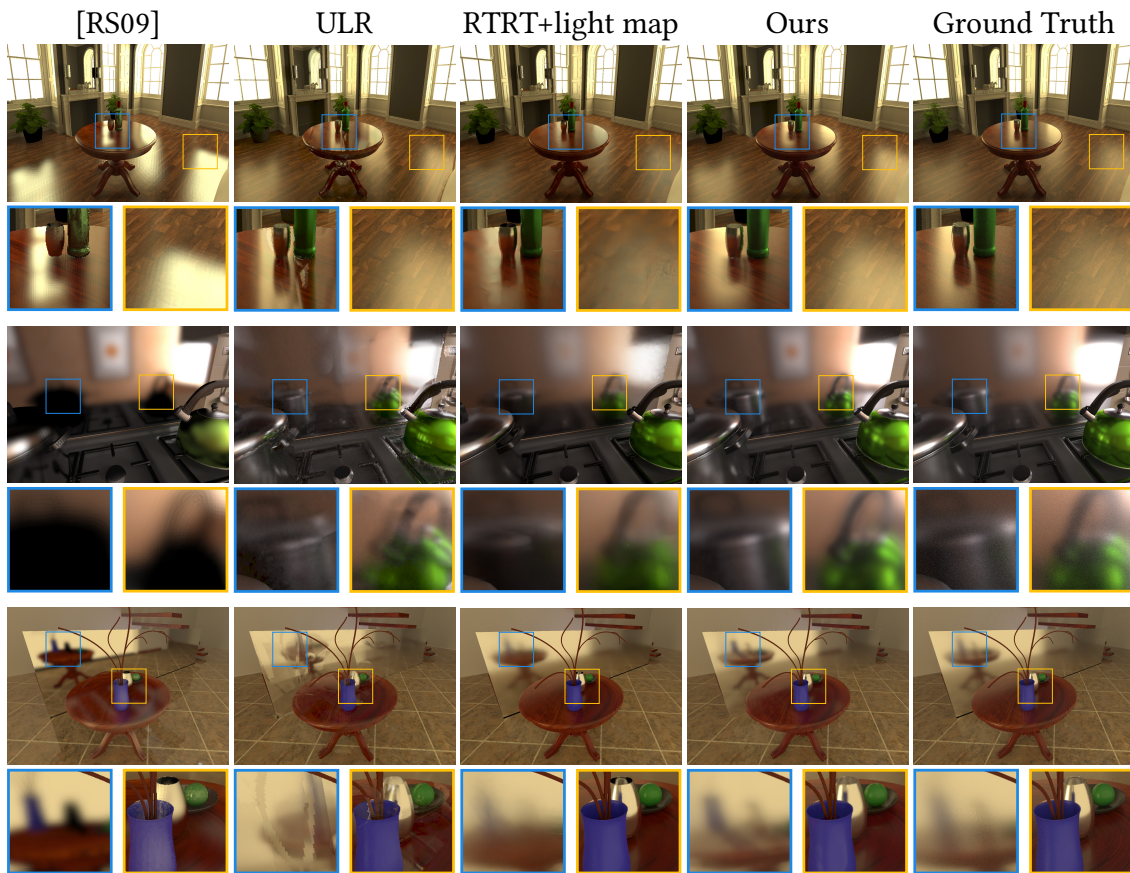


Figure 5.18 – Equal time comparisons for each method. Left to right: [RS09], ULR, RTRT with light map, ours and the path-traced ground truth.

Table 5.1 – Quantitative error metrics, using both RGB and luminance, comparing Image Space Gathering [RS09], the probe-based approach of McGuire et al. [MMNL17], the RTRT with light map baseline, and our method. Lower is better.

Method	RMSE		DSSIM	
	RGB	Lum.	RGB	Lum.
[RS09]	.086	.084	.072	.064
[MMNL17]	.093	.093	.087	.080
RTRT+light map	.050	.050	.063	.056
Ours	.027	.026	.046	.039

Table 5.2 – Timings and memory consumption of our method. Rendering timings average costs over frames in our videos.

	Bathroom	Kitchen	Livingroom	Staircase
Preprocess				
Parameterization	9 min	6 min	8 min	8 min
Geom. Data	3 min	3 min	3 min	2 min
Light map	10 h	9 h	15 h	12 h
Probes	16 h	13 h	22 h	23 h
Runtime				
Rasterization	0.9 ms	1.1 ms	0.9 ms	0.7 ms
Ray casting	6.3 ms	6.3 ms	6.3 ms	5.4 ms
Gathering	21 ms	27 ms	26 ms	18 ms
Filtering	10 ms	12 ms	11 ms	11 ms
Total	41 ms	47 ms	46 ms	36 ms
VRAM	5.6 GB	5.8 GB	5.7 GB	5.7 GB

path tracer, greatly accelerating this step, but we opted for *Mitsuba*'s mature pipeline to handle the materials in our scene repository. Comparing with our Falcor path-tracer implementation in *Bathroom* for example, for approximately the same quality one could expect a 10x speedup in preprocessing. Our adaptive parameterization minimizes overall memory use, especially when using half-precision probe textures. Our probe texels use 24 bytes: glossy color (6 bytes), reflected positions and material ID (8 bytes), triangle ID (4 bytes), barycentric coordinates (4 bytes), parameterization and its derivatives (8 bytes, stored at half resolution). Other geometric information is stored per-vertex and interpolated at runtime.

5.8 Conclusion

Limitations and Future Work. Our method achieves plausible results with a satisfactory accuracy in many cases (see Fig. 5.16-5.18); however it has some limitations.

The computational cost and memory of the precomputation is the main drawback of our approach, which also limits our solution to static scenes. An incremental approach to building light probes and maps via hardware accelerated path tracing would be an interesting way to lift this limit in future work.

We are currently limited to opaque materials. Extending our approach to transparent materials probably requires storing more information in the probes plus a new gathering approach for reprojecting transmissive surfaces. Such a gathering solution is an exciting future direction. A similar argument applies to extending the method to anisotropic materials, though a simpler solution may be possible for this case.

Finally, our two-step convolution is approximate, as seen with the small differences in glossiness in Figs. 5.16-5.18. A deeper study of error bounds and a more accurate approximation are definitely worthy goals. Nonetheless, our current results are plausible and provide convincing and quite accurate interactive renderings.

Summary. We presented a novel algorithm for real-time rendering of synthetic scenes using glossy paths dynamically reprojected from probes and diffuse lighting from a light map. Our solution builds on three main contributions: an adaptive parameterization to optimize probe memory usage, an accurate gathering algorithm for reprojecting glossy paths into novel views, and a two-step solution to avoid reflection boundary sharpening that occurs when reprojecting naively.

Our solution allows interactive walkthroughs with global illumination for opaque scenes. The path we chose is based on precomputation: the advantage is that complex light paths are precomputed at high quality and that our reprojection can accurately construct novel views. However, this comes at the price of the computational overhead of precomputation and the limitation to static scenes. On the other end of the spectrum are online methods, such as a denoised real-time ray-tracer. Our results show the feasibility of using precomputed data to render complex light paths interactively, and future methods should build on the full spectrum of methods from fully online to precomputed. Our solutions for memory optimization, accurate reprojection and occlusion-aware glossiness will hopefully be useful building blocks to such solutions, moving towards the ultimate goal of real-time global illumination for complex, dynamic scenes.

Conclusion

6.1 Contributions

In this thesis, we have presented three contributions leveraging image-based techniques for the rendering of view-dependent effects in real and synthetic scenes. All projects have tackled issues of sampling, resampling and reconstruction of different effects from the input data. Along the way, the main thread that arose linking our contributions was the study of reflective effects, how to extract them and reproject them using the corresponding flow.

By exploiting repetitions present on building facades, we were able to reconstruct the main architectural elements even if the input image set is very small. By working in an ideal space and combining information from multiple repetitive instances, we improved camera calibration, geometry reconstruction and the final rendering. By combining geometric priors with the sparse input view-dependent effects, specular regions can be detected and reflection effects re-rendered using external data.

In street-level scenes with a denser capture, we relied on semantic information to detect elements – such as car hulls and windows – that are badly reconstructed because of their materials. For each car geometry, missing and inaccurate data are filled and smoothed with the help of a custom parameterization. Through a feature-based fitting process, reflecting semi-transparent surfaces are approximated by analytic ellipsoids. From this representation, the flow of the reflections between views can be efficiently computed. This is used to extract layers containing plausible specular information from the input images. At runtime, the same flow computation is evaluated from the novel to the input views, sampling the specular layers to generate proper reflective effects.

Finally we focused on static synthetic scenes. Inspired by image-based rendering, we precompute specular effects at a set of predefined locations and store them in probes using an optimized parameterization. From a novel viewpoint, we perturb specular paths

to determine where to sample information in the probes. Specular data is gathered in the novel view and a final screen-space filter ensures that glossiness effects are properly reconstructed. We obtain high quality view-dependent effects in complex synthetic scenes that can be explored in real time.

Through these three projects, we have contributed to advancing the state of the art in rendering of view-dependent effects for both synthetic and real scenes. We have extended and built upon existing image-based rendering techniques, and tried to address multiple challenges of both reconstruction and synthesis of reflective elements.

6.2 Insights

Over the course of the presented projects, we have gained some insights related to the topics explored in our work.

Semantically guided processing. While in synthetic scenes material parameters are readily available, we have relied on semantic information as a proxy in real world scenes. This was used to detect and extract objects exhibiting reconstruction issues. We think that this proxy approach has shown to be successful in chapter 3 and 4 even if not fully explored yet. More classes of objects that are notoriously adversarial to image-based rendering techniques could be isolated and treated with ad hoc solutions before being reintegrated into the whole scene. Additionally, when semantic information is extracted from each input view, care has to be taken to ensure consistency when fusing it in a multi-view context. We faced this chicken-and-egg problem when reprojecting car semantic labels obtained by a 2D convolutional neural network to extract incomplete geometry, requiring an iterative approach. This has also been an opportunity to explore different learning-based approaches to semantic segmentation of images and meshes. Finally, we would have liked to explore how semantic could be used as an input to the rendering phase and have conducted initial promising experiments.

Changing parameterization. By isolating elements in our scenes, we were able to use per-object parameterizations to simplify processing of the available data. The platonic space in chapter 3 was used to fuse information from multiple instances while alleviating sparsity, and was at the core of the method. Per-car spherical reprojection in chapter 4 allowed us to apply image-space smoothing and filling techniques to repair geometry and refine semantic information. The probe parameterization in chapter 5 was designed

to adapt to the properties of the visible objects. In all cases we maximized the quality and amount of information stored and/or recovered in the scenes and these customized parameterizations could be leveraged for other problems.

Importance of reflections. The presence of view-dependent effects and more specifically reflections has proven to be crucial for the user perception in both real and synthetic scenes. This was initially observed when synthesizing reflections on the facade windows of chapter 3. Even in such sparse capture setup, partial input information could be used to re-apply reflections. While the background environment representation was not directly extracted from the input data, applying reflections at the proper locations provided an increase in realism that was crucial to the final result.

This motivated our study of street-level scenes where reflective and transparent surfaces are omnipresent and adversarial to both reconstruction and rendering. We thus focused on extracting specular layers and reprojecting them following an estimated reflection flow. The analytical approximation we relied on also showed that while respecting the general motion of reflections was important, the second-order behaviors – the parallax motion of the reflected background for instance – could be simplified without incurring a high visual cost.

When investigating the intersection of image-based rendering and real-time exploration of synthetic scenes, reflections came up as one of the major challenges that would be interesting to explore. Here again, we confirmed that higher-order reflections – especially on glossy surfaces – had to be present but not necessarily accurate. Quantifying the required level of accuracy for different orders of view-dependent effects would be an insightful follow-up.

Geometric quality. Because image-based rendering techniques rely on preexisting shading, the expectations put on the scene geometry are different than for classical rendering. Techniques such as the Unstructured Lumigraph reproject input image information onto the geometry, masking potential surface deformations. This is exemplified in chapter 3, where strong priors are used to obtain the refined geometry of the platonic element. While some details are lost, the general shape and sharp edges are preserved, ensuring proper occlusions when reprojecting shading from the input images. In preparatory work for chapter 4, we also observed how techniques such as Deep Blending can learn to repair geometric inconsistencies, thanks to both per-view geometry and the learned blending

network.

Our reliance on approximate surfaces in chapters 4 and 5 also proved to be insightful. The analytical representations used in both – ellipsoid and local paraboloid respectively – were efficient for the tasks at hand, although the fitting of these approximations proved to be more cumbersome than initially planned. In chapter 4, the automatic ellipsoid fitting required the combination of two different approaches to overcome the lack of reflection features in the input data. In chapter 5, synthetic geometry had to undergo a cumbersome cleanup; the fitting process furthermore had to be made extremely robust to surfaces with few triangles or incorrect normals.

6.3 Future work

While we have made progress for the rendering of view-dependent effects in multiple types of scenes, we have also observed and obtained results that call for further investigation. Furthermore, to make some of the tackled problems tractable we had to rely on assumptions or simplifications that could be lifted with further work. We now highlight the avenues for future work that appear the most interesting and promising from our point of view.

Leveraging semantic information. We have relied on semantic information obtained automatically in chapter 3 and 4, in order to detect and extract regions of interest in our scenes. We think that a more general approach exploiting multiple semantic classes for both reconstruction and rendering would be an exciting field of future work. Segmentation of scenes in the wild is now a computer vision topic that evolves quickly and highly detailed training data is now available [CZP⁺18, ZZP⁺19]. Similar approaches exist for geometry reconstruction [HZCP17] but semantic information could also be leveraged for resampling and reprojection of input data when rendering the novel view, for instance by learning to output class-specific blending weights or to synthesize per-class appearance effects [WLZ⁺18]. Going further, more precise object properties could be extracted from the input data; material acquisition of complete scenes would simplify both content extraction and the rendering of complex view-dependent effects.

Isolating and reconstructing groups of semantic elements separately could also unlock the possibility of mixing objects from different scenes as explored by Nicolet et al. [NPD20]. Entire new scenes could be generated by combining our extracted elements with proce-

dural generation rules. A few sets of components could be used to generate extremely large scenes while keeping the capture process and storage requirement under control. Modular approaches would also simplify the generation of realistic training data, for instance for automotive learning tasks.

Viewpoint placement and selection. As pointed out in chapter 5, we think that adaptive placement of input views – or probes – in a synthetic scene based on the materials is an interesting avenue for future projects, especially as existing work focused mainly on diffuse surfaces [FCOL00]. Apart from minimizing memory use or allowing for a better reconstruction of view-dependent effects, understanding placement criteria would also be an initial step towards a finer formalization of the camera selection process in real world scenes. Existing works [DLD12, MSOC⁺19, HNH18] already provide prescriptive guidelines for specific acquisition and rendering setups, but a more complete framework could be built upon frequency analysis [CTCS00] and take into account estimated material parameters. Synthetic scenes would be an important stepping stone to validate the foundations of such a framework.

Learning for extraction and blending. In chapter 4, geometry and layer extraction were performed in an ad hoc manner, and the final blending and compositing of view-dependent effects with the rest of the scene was simplified. Our custom parameterization brings the extracted car geometry in a 2D image-like map, where correction of artifacts could be performed by a convolutional neural network incorporating object priors. Similarly, layer extraction has recently been cast as a learning task [WGGK18]. As learned blending weights are already used for the background scene [HPP⁺18] in chapter 4, additional color candidates could be given to the network, coming from the reprojected specular layers or the transmitted background. Aggregating all view-dependent effects present in the scene in a unified way might allow for more complex compositing effects, resulting in a more realistic final render. For both research paths, the question of generating exploitable training data has to be addressed by relying on synthetic or heavily annotated scenes.

Precomputation for real-time rendering. We have shown in chapter 5 that reprojecting precomputed information allowed rendering of complex light paths interactively. This came at the cost of heavy preprocessing and storage requirements. By taking inspiration from recent anti-aliasing and denoising techniques, the accumulation and blending of

gathered samples could be expressed as a learning task. Results of a similar quality might then be obtained with lower sample count and smaller probes. Recent real-time ray tracing hardware could be leveraged to progressively generate shaded samples on-the-fly, inspired by the Render Cache [WDP99] and frameless rendering [DWWL05]. This could be coupled with a temporal component to support dynamic objects or lighting, reusing samples from previous frames under additional validity criteria. Refraction and other complex light paths could be supported by applying a similar perturbation framework and storing the appropriate data. Accuracy of the image-space glossiness reconstruction filter could also be improved thanks to a more accurate BRDF fit, potentially inferred using machine learning. By supporting arbitrary combinations of offline precomputation and on-the-fly ray tracing updates for all light paths, future approaches would allow for accurate real-time global illumination of dynamic scenes at an adjustable power cost.

6.4 Impact

The projects presented in this thesis have led to publications and presentations in international conferences. The content of chapter 3 has been presented to EGSR 2018, as an oral presentation and journal publication. Chapter 4 has led to a publication in the Proceedings of the ACM in Computer Graphics and Interactive Techniques and will be presented at I3D 2020. It has also motivated an internship to explore neural extraction and rendering of specular effects. The project from chapter 5 is currently under review, and has led to an internship on the formalization of camera placement in synthetic scenes. All three projects have been integrated in a common code-base shared with the implementations of other image-based rendering techniques, soon to be open sourced for reproducibility and benchmarking.

Transformation from Platonic Scene to Input Scene

In chapter 3, we need to estimate a transformation T from the platonic frame to the input scene frame, in order to place the platonic model and cameras back into the input scene frame. We evaluate a series of candidate transformations $T_{i,j}$ using each platonic camera separately (see Fig. 3.11).

For each platonic camera $C_{i,j}^*$, we estimate an approximate corresponding camera $C_{i,j}$ in the input scene frame. As in Sec. 3.4.2, we compute it from the input camera C_i and the cropped image $V_{i,j}$ location in V_i . We compute a transformation from camera $C_{i,j}^*$ to $C_{i,j}$ by aligning their positions, and their direction, up and right vectors. This transformation only characterizes a rotation and translation, leaving an unknown scaling factor between the two frames.

To lift this ambiguity, we use the distances between the camera and the facade, in the platonic and input frames (respectively $d_{i,j}^*$ and $d_{i,j}$).

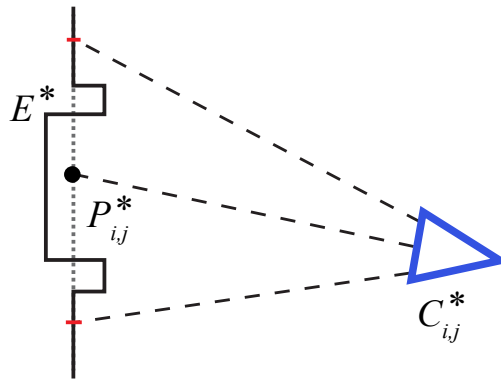


Figure A.1 – We cast rays from the borders of the view associated to $C_{i,j}^*$, intersecting the platonic model (red points). These samples can be used to estimate the facade plane (dotted gray line).

The optical center of $C_{i,j}$ intersects the known facade plane (and thus E_j) at $P_{i,j}$. Similarly the optical center of $C_{i,j}^*$ intersects the platonic facade plane at $P_{i,j}^*$ (Fig. A.1). This point does not necessarily belong to the platonic mesh, due to geometric details such as window recess. We estimate the plane of the facade in the platonic frame by sampling pixels near the borders of $V_{i,j}$, and casting rays towards the mesh. The motivation is that such intersection points will belong to the facade wall. From these points we can estimate the facade plane. We intersect a central ray with it to obtain $P_{i,j}^*$.

Then, $d_{i,j}^*$ is the distance between $P_{i,j}^*$ and $C_{i,j}^*$, $d_{i,j}$ the distance between $P_{i,j}$ and $C_{i,j}$. We estimate the scaling factor s from the ratio of these distances, taking into account the field of view values of $C_{i,j}^*$ and $C_{i,j}$.

$$s = \frac{d_{i,j} \tan(\text{fov}_{C_{i,j}}/2)}{d_{i,j}^* \tan(\text{fov}_{C_{i,j}^*}/2)}$$

In practice, we express the transformation from platonic scene space to $C_{i,j}^*$ image space, then estimate the scaling factor; we compose this result with the transformation from $C_{i,j}$ image space to input scene space. The candidate ($T_{i,j}$) transformations are centered (from E_j to the origin) and averaged to obtain the final transformation T . This transformation is used to place the platonic mesh and cameras back into the input scene frame, before duplicating and translating them at each E_j position using $P_{i,j}$.

Car Mesh Refinement and Ellipsoid Fitting

B.1 Mesh Refinement Minimization

For mesh refinement (Sec. 4.4.2), we use a conjugate gradient solver to minimize the following penalty function:

$$E(\mathbf{d}) = \sum_p \left(w_u(p) \left(\mathbf{d}(p) - \mathbf{d}'(p) \right)^2 + w_b \sum_{q \in N(p)} \left(\mathbf{d}(p) - \mathbf{d}(q) \right)^2 + w_l \left(\mathcal{L}_d(p) - \alpha_l \right)^2 \right) \quad (\text{B.1})$$

where $\mathbf{d}(p)$ is the estimated depth at pixel p , $\mathbf{d}'(p)$ the initial depth at p (both expressed in $[0, 1]$), $N(p)$ is the set of pixels around p , \mathcal{L}_d is the discrete Laplace operator on the depth. We set the Laplacian prior $\alpha_l = \cos(3^\circ)$, and $w_b = w_l = 0.33$. We initialize the unary weight with $w_u(p) = \max(0, \mathcal{P}_{car}(p) - \mathcal{P}_{win}(p))$, where the probabilities \mathcal{P} for car and window respectively are extracted from the segmentation map. In subsequent iterations we set $w_u = 0$ for outlier pixels, defined as pixels that have moved from their initial 3D position more than 2% of the sphere radius (i.e. a few centimeters).

B.2 Isolating Car Objects using Semantic Labels

Semantic segmentation network training. We start with 2D label maps for each input image, obtained with the DeepLab-v2 (Resnet-101) architecture [CPK⁺17]. We train it on a subset of the ADE20K dataset [ZZP⁺19, ZZP⁺17], only selecting labels that correspond to object categories that both exhibit the regions we want to detect and are present in cityscapes. We select both object-level and part-level labels among the available ADE20K labels: car, car wheel, car window. We merge all other car related labels into a single “car” label. All other labels are considered as background. We filter images of this dataset to only keep examples containing instances of those labels. We obtain a training set of 4000 images and a validation set of 500 images. We train our

network for 300K epochs ¹.

Semantic Labeling MRF. For the final semantic mask refinement, (end of Sec. 4.4.2), we solve a labeling problem with graph cut, using the constant data cost 0.5 for “car”, and an adaptive data cost for “window”. Pixels inside the window regions and outliers get cost 0.0 (i.e., likely windows), while pixels outside dilated windows or pixels with high photo-consistency are given cost 1.0. All remaining pixels are given cost 0.52 to encourage smaller window regions. The smoothness term is a color-gradient weighted Potts term, for neighboring pixels with different labels.

B.3 Ellipsoid fitting algorithm

In this appendix, we present the details of the ellipsoid fitting algorithm for windows (Sec. 4.5.2). For each window, we choose 10 reference images where the window is visible and as fronto-parallel as possible. We also ensure that each reference image has neighboring images on all 4 sides. We compare each reference image to its 10 closest neighbors. We compute ORB features [RRKB11] for those images ; we use best buddy matching [VLS⁺06] and Lowe thresholding [Low04] ($th = 0.95$) to establish correspondences. We also discard correspondences whose motion can be explained by the reconstructed window geometry, such as feature points on stickers or scratches on the windows. If a feature point in the reference image is reprojected in a neighbor view such that the distance to the matched point is smaller than 1.5 % of the image dimensions (10px for our typical 4Mpixel images), the correspondence is discarded. We fall back to dense image matching when the average number of successful matches is smaller than 0.1% of the window area (in pixels).

Feature point matching. We use a RANSAC inspired algorithm which computes the total number of inlier feature point correspondences as our score. That is, for each pair of radii (r_x, r_y) , we count how many matches can be explained by the predicted reflection flow. A match between the reference and a neighbor image is an inlier if the flow predicts the location of the feature point in the neighbor view with no more than a 10px error for our typical 4Mpixel input images. We observed that most side windows are very anisotropic, with a larger curvature radius – nearly planar shape – around the vertical

¹The following hyper-parameters were used for training: batch size=4, learning rate $2.5e^{-4}$, momentum=0.9, learning rate decay=0.9 and weight decay=0.0005.

axis. We encode this with a prior on the ratio between r_y and r_x :

$$p(r_x, r_y) = \mathcal{N}_{\mu=8, \sigma=3.5}\left(\frac{r_y}{r_x}\right) \quad (\text{B.2})$$

For windshields and rear windows, there is less anisotropic behavior and we use the constant prior $p(r_x, r_y) = 1$. We pick the pair of radii which maximizes $s(r_x, r_y) = p(r_x, r_y) \# \text{inliers}(r_x, r_y)$.

Dense image matching. We only use the top ranked reference image to compute our score as this matching is slower. Images are also downscaled at 720p for speed and robustness to outliers. For each (r_x, r_y) , we compute the reflection flows and warp 25 neighboring images into the reference view. The number of neighbors is increased as median consistency is sensitive to noise. We use the median-based photo-consistency from [VLS⁺06] to compute an error map for each warped image. A per-pixel median error is extracted, and its mean value is computed over all pixels in the window mask. The parameter pair with the lowest error is selected.

Effect of radii variations. Each window is approximated by an ellipsoid with longitudinal and vertical radii. Due to shape and physical constraints, the range of admissible curvatures for car windows is quite limited. We use the same range of radii for all windows ($[1m, 40m]$) and sweep it using quadratic steps to sample small values more densely, as small changes to the radii only create noticeably different reflection motion if the radii are small. This is illustrated in Fig. B.1, using both axes. A side window is almost planar along an horizontal line, but quite curved along a vertical line. The radius *around the vertical axis* is thus quite large, and even large variations only cause minor shifts. On the other hand, the radius *around the longitudinal axis* is small, and even small variations can lead to shifts in the reflections. This motivates our choice of a quadratic sweep schedule, while showing the accuracy required when estimating the radius around the longitudinal axis.

Comparison with a planar reflector. Instead of using the ellipsoid approximation, it is possible to solve the reflection flow estimation problem directly for the case of a planar reflector. Each window is then represented by a plane going through the centroid of the window mesh and oriented by the window mean normal. The intersection with the background sphere is mirrored with respect to the plane before being reprojected in the input view. The planar surface does not manage to capture the specific reflection flow

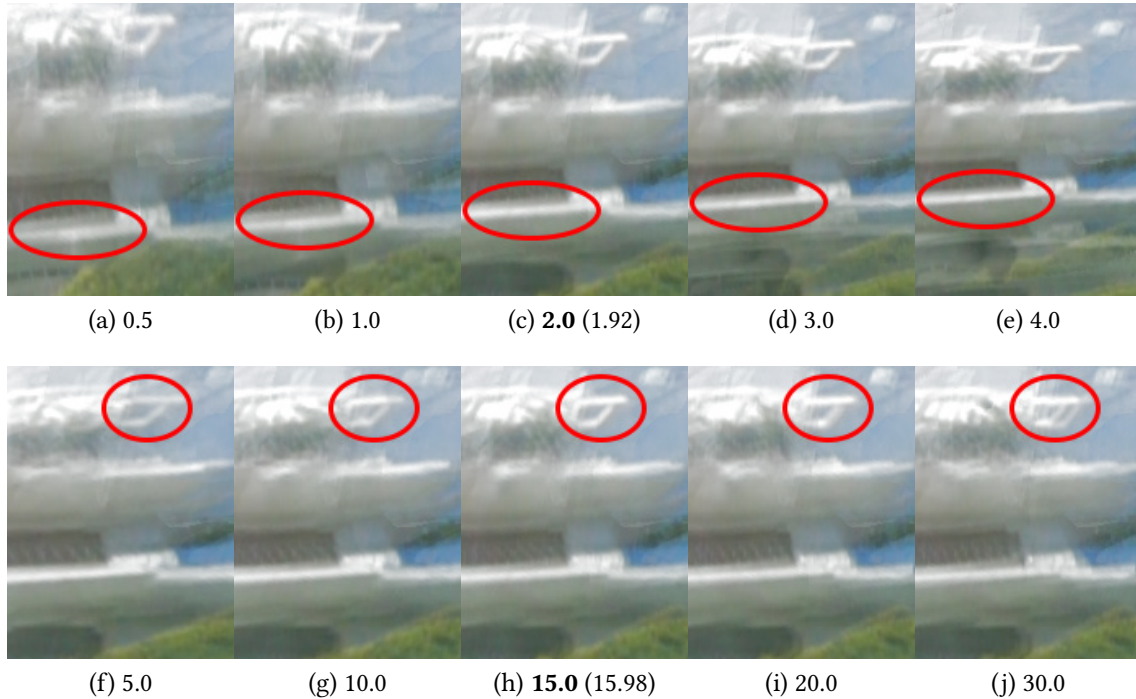


Figure B.1 – Evolution of the generated reflection when sweeping the horizontal axis radius from 0.5 to 4.0 (top) and the radius around the vertical axis from 5.0 to 30.0 (bottom). The parameters closest to the radii obtained using our automatic fitting are in bold, the estimated values in parenthesis. Details such as the white horizontal line are sharper when the radii are properly estimated.

exhibited by slightly curved windows, leading to erroneous alignment between warped specular layers (see Fig. B.2).



Figure B.2 – A comparison between a planar (left) and ellipsoid (right) representation for each window when computing the reflection flow. The planar simplification leads to strong alignment artifacts and duplications.

Choice of DCT for Parameterization Guiding

In Sec. 5.4.2.1, we use the DCT to identify probe regions requiring increased resolution due to small features or high geometric complexity. While any frequency decomposition could be used, we used the DCT for simplicity and its real-valued coefficients. We could have used image depths, rather than normals, as a representation of geometry. However, due to the linearity of the convolution, larger depth differences naturally produce stronger responses. To avoid this, we use the normal buffer, which contains normalized values by construction. A canonical DCT application subdivides the image into $N \times N$ blocks and treats blocks individually, resulting in one response per block. In contrast, we convolve the image with the corresponding $N \times N$ DCT basis functions, yielding an individual response per pixel. Note that, therefore, our approach gives the same responses as the block-based method, just at an N^2 higher spatial resolution.

Bibliography

- [ADA⁺04a] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Transactions on Graphics (ToG)*, 23(3):294–302, 2004.
- [ADA⁺04b] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 294–302. ACM, 2004.
- [ANR74] Nasir Ahmed, T. Natarajan, and Kamisetty R Rao. Discrete cosine transform. *IEEE Transactions on Computers*, 100(1):90–93, 1974.
- [ARB07] Daniel G Aliaga, Paul A Rosen, and Daniel R Bekins. Style grammars for interactive visualization of architecture. *IEEE transactions on visualization and computer graphics*, 13(4), 2007.
- [AWL⁺15] Miika Aittala, Tim Weyrich, Jaakko Lehtinen, et al. Two-shot svbrdf capture for stationary materials. *ACM Trans. Graph.*, 34(4):110–1, 2015.
- [AYLM13] Sawsan AlHalawani, Yong-Liang Yang, Han Liu, and Niloy J Mitra. Interactive facades analysis and synthesis of semi-regular facades. In *Computer Graphics Forum*, volume 32, pages 215–224. Wiley Online Library, 2013.
- [BBM⁺01] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432. ACM, 2001.
- [BFMZ94] Gary Bishop, Henry Fuchs, Leonard McMillan, and Ellen J Scher Zagier. Frameless rendering: Double buffering considered harmful. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 175–176, 1994.

- [Bit16] Benedikt Bitterli. Rendering resources, 2016. <https://benedikt-bitterli.me/resources/>.
- [BL18] Jean-Philippe Bauchet and Florent Lafarge. KIPPI: KInetic Polygonal Partitioning of Images. In *Proc. of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, US, 2018.
- [Bre02] Chris Brennan. Accurate environment mapped reflections and refractions by adjusting for object distance. *Shader X*, 2002.
- [BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.
- [Bur20] J. Burgess. Rtx on the nvidia turing gpu. *IEEE Micro*, 40(2):36–44, 2020.
- [BYC⁺20] Nir Benty, Kai-Hwa Yao, Petrik Clarberg, Lucy Chen, Simon Kallweit, Tim Foley, Matthew Oakes, Conor Lavelle, and Chris Wyman. The Falcor rendering framework, 03 2020.
- [CA00a] Min Chen and James Arvo. Perturbation methods for interactive specular reflections. *IEEE Transactions on Visualization and Computer Graphics*, 6(3):253–264, 2000.
- [CA00b] Min Chen and James Arvo. Theory and application of specular path perturbation. *ACM Transactions on Graphics (TOG)*, 19(4):246–278, 2000.
- [CDSHD13] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics (TOG)*, 32(3):30, 2013.
- [Che95] Shenchang Eric Chen. Quicktime vr: An image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 29–38, 1995.
- [CMZP13] Duygu Ceylan, Niloy J. Mitra, Youyi Zheng, and Mark Pauly. Coupled structure-from-motion and 3d symmetry detection for urban facades. *ACM Transactions on Graphics*, 2013.

- [CNS⁺11] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, volume 30, pages 1921–1930. Wiley Online Library, 2011.
- [CPK⁺17] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017.
- [CRMT91] Shenchang Eric Chen, Holly E Rushmeier, Gavin Miller, and Douglass Turner. A progressive multi-pass method for global illumination. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 165–174. ACM, 1991.
- [CTCS00] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum. Plenoptic sampling. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 307–318, 2000.
- [CZP⁺18] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.
- [DAB15] Ilke Demir, Daniel G Aliaga, and Bedrich Benes. Procedural editing of 3d building point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2147–2155, 2015.
- [DDSD03] Xavier Décoret, Frédo Durand, François X Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In *ACM SIGGRAPH 2003 Papers*, pages 689–696. 2003.
- [DLD12] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured light fields. In *Computer Graphics Forum*, volume 31, pages 305–314. Wiley Online Library, 2012.

- [DMIF15] Tali Dekel, Tomer Michaeli, Michal Irani, and William T Freeman. Revealing and modifying non-local variations in a single image. *ACM Transactions on Graphics (TOG)*, 34(6):227, 2015.
- [DRSVG13] Dengxin Dai, Hayko Riemenschneider, Gerhard Schmitt, and Luc Van Gool. Example-based facade texture synthesis. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1065–1072, 2013.
- [DTM96] Paul E Debevec, Camillo J Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20. ACM, 1996.
- [DWWL05] Abhinav Dayal, Cliff Woolley, Benjamin Watson, and David Luebke. Adaptive frameless rendering. In *ACM SIGGRAPH 2005 Courses*, page 24. ACM, 2005.
- [DYB98] Paul Debevec, Yizhou Yu, and George Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Rendering Techniques '98*, pages 105–116. Springer, 1998.
- [EDDM⁺08] Martin Eisemann, Bert De Decker, Marcus Magnor, Philippe Bekaert, Edilson De Aguiar, Naveed Ahmed, Christian Theobalt, and Anita Sellent. Floating textures. In *Computer graphics forum*, volume 27, pages 409–418. Wiley Online Library, 2008.
- [EMD⁺05] Pau Estalella, Ignacio Martin, George Drettakis, Dani Tost, Olivier Devillers, and Frédéric Cazals. Accurate interactive specular reflections on curved objects. In *Proceedings of Vision, Modeling and Visualization*. Eurographics Association, 2005.
- [Eth17] Darrel Etherington. Google's street view cameras get a high-res update focused on ai. <https://techcrunch.com/2017/09/05/googles-street-view-cameras-get-a-high-res-update-focused-on-ai/>, 2017.
- [FCOL00] Shachar Fleishman, Daniel Cohen-Or, and Dani Lischinski. Automatic camera placement for image-based modeling. In *Computer Graphics Forum*, volume 19, pages 101–110. Wiley Online Library, 2000.

- [FHP⁺18] Luca Fascione, Johannes Hanika, Rob Pieké, Ryusuke Villemin, Christophe Hery, Manuel Gamito, Luke Emrose, and André Mazzone. Path tracing in production. In *ACM SIGGRAPH 2018 Courses*, pages 1–79. 2018.
- [Fle14] Roland W Fleming. Visual perception of materials and their properties. *Vision research*, 94:62–75, 2014.
- [FRS19] Sebastian Friston, Tobias Ritschel, and Anthony Steed. Perceptual rasterization for head-mounted display image synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH 2019)*, 38(4), 2019.
- [GGSC96] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1996.
- [GHLB15] Clement Godard, Peter Hedman, Wenbin Li, and Gabriel J. Brostow. Multi-view reconstruction of highly specular surfaces in uncontrolled environments. In *International Conference on 3D Vision*, pages 19–27. IEEE, 2015.
- [GO11] Eduardo SL Gastal and Manuel M Oliveira. Domain transform for edge-aware image and video processing. In *ACM SIGGRAPH 2011 papers*, pages 1–12. 2011.
- [GSC⁺07] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M. Seitz. Multi-view stereo for community photo collections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–8. IEEE, 2007.
- [GSHG98] Gene Greger, Peter Shirley, Philip M Hubbard, and Donald P Greenberg. The irradiance volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, 1998.
- [GSVDW03] J-M Geusebroek, Arnold WM Smeulders, and Joost Van De Weijer. Fast anisotropic gauss filtering. *IEEE transactions on image processing*, 12(8):938–943, 2003.
- [HDF14] Jared Heinly, Enrique Dunn, and Jan-Michael Frahm. Correcting for duplicate scene structure in sparse 3d reconstruction. In *European Conference on Computer Vision*, pages 780–795. Springer, 2014.

- [Hec90] Paul S Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *ACM SIGGRAPH Computer Graphics*, 24(4):145–154, 1990.
- [HKP⁺99] Benno Heigl, Reinhard Koch, Marc Pollefeys, Joachim Denzler, and Luc Van Gool. Plenoptic modeling and rendering from image sequences taken by a hand-held camera. In *Mustererkennung 1999*, pages 94–101. Springer, 1999.
- [HNH18] Benjamin Hepp, Matthias Nießner, and Otmar Hilliges. Plan3d: Viewpoint and trajectory optimization for aerial multi-view stereo reconstruction. *ACM Transactions on Graphics (TOG)*, 38(1):1–17, 2018.
- [HPP⁺18] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (TOG)*, 37(6), November 2018.
- [HRDB16] Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. Scalable inside-out image-based rendering. *ACM Transactions on Graphics (TOG)*, 35(6):231, 2016.
- [HSAS19] Antti Hirvonen, Atte Seppälä, Maksim Aizenshtein, and Niklas Smal. Accurate real-time specular reflections with radiance caching. In *Ray Tracing Gems*, pages 571–607. Springer, 2019.
- [HSL01] Ziyad S Hakura, John M Snyder, and Jerome E Lengyel. Parameterized environment maps. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 203–208, 2001.
- [HSP14] Christian Hane, Nikolay Savinov, and Marc Pollefeys. Class specific 3d object shape priors using surface normals. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–659. IEEE, 2014.
- [HZCP17] Christian Haene, Christopher Zach, Andrea Cohen, and Marc Pollefeys. Dense semantic 3d reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1730–1743, 2017.

- [IKL⁺08] Ivo Ihrke, Kiriakos N. Kutulakos, Hendrik P.A. Lensch, Marcus Magnor, and Wolfgang Heidrich. State of the art in transparent and specular object reconstruction. In *Eurographics State Of The Art Report (STAR)*. Eurographics Association, 2008.
- [IKL⁺10] Ivo Ihrke, Kiriakos N. Kutulakos, Hendrik P.A. Lensch, Marcus Magnor, and Wolfgang Heidrich. Transparent and specular object reconstruction. In *Computer Graphics Forum*, volume 29, pages 2400–2426. Wiley Online Library, 2010.
- [Jak10] Wenzel Jakob. Mitsuba renderer, 2010.
- [JHKP13] Bastien Jacquet, Christian Hane, Kevin Koser, and Marc Pollefeys. Real-world normal map capture for nearly flat reflective surfaces. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 713–720. IEEE, 2013.
- [JTC09] Nianjuan Jiang, Ping Tan, and Loong-Fah Cheong. Symmetric architecture modeling with a single image. In *ACM Transactions on Graphics (TOG)*, volume 28, page 113. ACM, 2009.
- [JTC11] Nianjuan Jiang, Ping Tan, and Loong-Fah Cheong. Multi-view repetitive structure detection. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 535–542. IEEE, 2011.
- [Kaj86] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986.
- [Kar14] Brian Karis. High-quality temporal supersampling. *Advances in Real-Time Rendering in Games, SIGGRAPH Courses*, 1:1–55, 2014.
- [KCS14] Johannes Kopf, Michael F Cohen, and Richard Szeliski. First-person hyper-lapse videos. *ACM Transactions on Graphics (TOG)*, 33(4):1–10, 2014.
- [KKSM17] Babis Koniaris, Maggie Kosek, David Sinclair, and Kenny Mitchell. Real-time rendering with compressed animated light fields. In *Graphics Interface*, pages 33–40, 2017.

- [KLS⁺13] Johannes Kopf, Fabian Langguth, Daniel Scharstein, Richard Szeliski, and Michael Goesele. Image-based rendering in the gradient domain. *ACM Transactions on Graphics (TOG)*, 32(6):199, 2013.
- [KSE⁺03] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics (TOG)*, 22(3):277–286, 2003.
- [KW93] Hans Knutsson and C-F Westin. Normalized and differential convolution. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 515–523. IEEE, 1993.
- [LBR⁺18] Puneet Lall, Silviu Borac, Dave Richardson, Matt Pharr, and Manfred Ernst. View-region optimized image-based scene simplification. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(2):26, 2018.
- [LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42. ACM, 1996.
- [LMCB06] Artur Loza, Lyudmila Mihaylova, Nishan Canagarajah, and David Bull. Structural similarity-based object tracking in video sequences. In *2006 9th International Conference on Information Fusion*, pages 1–6. IEEE, 2006.
- [Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
- [LR98] Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Rendering Techniques' 98*, pages 301–314. Springer, 1998.
- [LRR⁺14] Gerrit Lochmann, Bernhard Reinert, Tobias Ritschel, Stefan Müller, and Hans-Peter Seidel. Real-time reflective and refractive novel-view synthesis. In *VMV*, pages 9–16, 2014.
- [LSS⁺19] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic

- renderable volumes from images. *ACM Transactions on Graphics (TOG)*, 38(4):65, 2019.
- [LTH⁺13] Christian Luksch, Robert F Tobler, Ralf Habel, Michael Schwärzler, and Michael Wimmer. Fast light-map computation with virtual polygon lights. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 87–94. ACM, 2013.
- [LW07] Anat Levin and Yair Weiss. User assisted separation of reflections from a single image using a sparsity prior. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(9):1647–1654, 2007.
- [LWS19] Christan Luksch, Michael Wimmer, and Michael Schwärzler. Incrementally baked global illumination. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, page 4. ACM, 2019.
- [MB95] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46, 1995.
- [MDSB03] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III*, pages 35–57. Springer, 2003.
- [MH92] Don Mitchell and Pat Hanrahan. Illumination from curved reflectors. In *SIGGRAPH*, volume 92, pages 283–291. Citeseer, 1992.
- [MM14] Morgan McGuire and Michael Mara. Efficient gpu screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)*, 3(4):73–85, 2014.
- [MMM17] Morgan McGuire, Michael Mara, and Zander Majercik. The G3D innovation engine, 01 2017. <https://casual-effects.com/g3d>.
- [MMMO] Pierre Moulon, Pascal Monasse, Renaud Marlet, and Others. Openmvg. an open multiple view geometry library. <https://github.com/openMVG/openMVG>.

- [MMNL17] Morgan McGuire, Mike Mara, Derek Nowrouzezahrai, and David Luebke. Real-time global illumination using precomputed light field probes. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, page 2. ACM, 2017.
- [MRM⁺10] Przemyslaw Musialski, Meinrad Recheis, Stefan Maierhofer, Peter Wonka, and Werner Purgathofer. Tiling of ortho-rectified facade images. In *Proceedings of the 26th Spring Conference on Computer Graphics*, pages 117–126. ACM, 2010.
- [MS95] Paulo WC Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 95–ff, 1995.
- [MSOC⁺19] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 34(4), 2019.
- [MST⁺20] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*, 2020.
- [MZWVG07] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. *ACM Transactions on Graphics (TOG)*, 26(3):85, 2007.
- [Nic65] Fred E Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied optics*, 4(7):767–775, 1965.
- [NPD20] Baptiste Nicolet, Julien Philip, and George Drettakis. Repurposing a relighting network for realistic compositions of captured scenes. In *Symposium on Interactive 3D Graphics and Games*, pages 1–9, 2020.
- [NSL⁺07] Diego Nehab, Pedro V Sander, Jason Lawrence, Natalya Tatarchuk, and John R Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Graphics hardware*, volume 41, pages 61–62, 2007.

- [NVI18] NVIDIA. Nvidia turing gpu architecture: Graphics reinvented, 2018.
- [OR98] Eyal Ofek and Ari Rappoport. Interactive reflections on curved objects. In *Proceedings of the 25th annual conference on Computer Graphics and Interactive Techniques*, pages 333–342. ACM, 1998.
- [PBD⁺10] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: a general purpose ray tracing engine. *Acm transactions on Graphics (TOG)*, 29(4):1–13, 2010.
- [PDG14] Sergi Pujades, Frédéric Devernay, and Bastian Goldluecke. Bayesian view synthesis and image-based rendering principles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3906–3913, 2014.
- [PFG00] Fabio Pellacini, James A Ferwerda, and Donald P Greenberg. Toward a psychophysically-based light reflection model for image synthesis. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 55–64, 2000.
- [PGB03] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM Transactions on graphics (TOG)*, volume 22, pages 313–318. ACM, 2003.
- [PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.
- [PVV05] Tuan Q Pham and Lucas J Van Vliet. Separable bilateral filtering for fast video preprocessing. In *2005 IEEE International Conference on Multimedia and Expo*, pages 4–7. IEEE, 2005.
- [PZ17] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics (TOG)*, 36(6):235, 2017.
- [RBDD18] Simon Rodriguez, Adrien Bousseau, Fredo Durand, and George Drettakis. Exploiting repetitions for image-based rendering of facades. In *Computer Graphics Forum*, volume 37, pages 119–131. Wiley Online Library, 2018.

- [RDGK12] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. In *Computer Graphics Forum*, volume 31, pages 160–188. Wiley Online Library, 2012.
- [Rea18] Capturing Reality. Realitycapture reconstruction software. <https://www.capturingreality.com/Product>, 2018.
- [RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [RH06] David Roger and Nicolas Holzschuch. Accurate specular reflections in real-time. In *Computer Graphics Forum*, volume 25, pages 293–302. Wiley Online Library, 2006.
- [RKR⁺16] Bernhard Reinert, Johannes Kopf, Tobias Ritschel, Eduardo Cuervo, David Chu, and Hans-Peter Seidel. Proxy-guided image-based rendering for mobile devices. In *Computer Graphics Forum*, volume 35, pages 353–362. Wiley Online Library, 2016.
- [RLP⁺20] Simon Rodriguez, Thomas Leimkhuler, Siddhant Prakash, Peter Shirley, Chris Wyman, and George Drettakis. Accurate warping of adaptive probes for interactive global illumination. Currently under review, 2020.
- [RPHD20] Simon Rodriguez, Siddhant Prakash, Peter Hedman, and George Drettakis. Image-based rendering of cars using semantic labels and approximate reflection flow. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3, 2020.
- [RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 11, pages 2564–2571. IEEE, 2011.
- [RS09] Austin Robison and Peter Shirley. Image space gathering. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 91–98. ACM, 2009.

- [RT13] Radim Šára Radim Tyleček. Spatial pattern templates for recognition of objects with regular structure. In *Proc. GCPR*, Saarbrücken, Germany, 2013.
- [SAA00] Richard Szeliski, Shai Avidan, and P Anandan. Layer extraction from multiple images containing reflections and transparency. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 246–253. IEEE, 2000.
- [SF16] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [SGHS98] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242, 1998.
- [SKALP05] László Szirmay-Kalos, Barnabás Aszódi, István Lazányi, and Mátyás Premecz. Approximate ray-tracing on the gpu with distance impostors. In *Computer graphics forum*, volume 24, pages 695–704. Wiley Online Library, 2005.
- [SKG⁺12] Sudipta N Sinha, Johannes Kopf, Michael Goesele, Daniel Scharstein, and Richard Szeliski. Image-based rendering for scenes with reflections. *ACM Trans. Graph.*, 31(4):100–1, 2012.
- [SKUP⁺09] László Szirmay-Kalos, Tamás Umenhoffer, Gustavo Patow, László Szécsi, and Mateu Sbert. Specular effects on the gpu: State of the art. In *Computer Graphics Forum*, volume 28, pages 1586–1617. Wiley Online Library, 2009.
- [SKW⁺17] Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*, pages 1–12. ACM, 2017.
- [SLS⁺96] Jonathan Shade, Dani Lischinski, David H Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs

- of complex environments. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 75–82, 1996.
- [SSH⁺98] Philipp Slusallek, Marc Stamminger, Wolfgang Heidrich, J-C Popp, and H-P Seidel. Composite lighting simulations with lighting networks. *IEEE Computer Graphics and Applications*, 18(2):22–31, 1998.
- [SSS06] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM Siggraph 2006 Papers*, pages 835–846. 2006.
- [ST91] Takafumi Saito and Tokiichiro Takahashi. Nc machining with g-buffer method. In *ACM SIGGRAPH Computer Graphics*, volume 25, pages 207–216. ACM, 1991.
- [STH⁺19] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2019.
- [SZ12] Lagarde Sébastien and Antoine Zanuttini. Local image-based lighting with parallax-corrected cubemaps. In *ACM SIGGRAPH 2012 Talks*, page 36. ACM, 2012.
- [SZPF16] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.
- [TM98] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*, pages 839–846. IEEE, 1998.
- [TTS18] Shubham Tulsiani, Richard Tucker, and Noah Snavely. Layer-structured 3d scene inference via view synthesis. In *Proceedings of the European Conference on Computer Vision*, pages 302–317. Springer, 2018.
- [TZN19] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

- [TZT⁺20] Justus Thies, Michael Zollhöfer, Christian Theobalt, Marc Stamminger, and Matthias Nießner. Image-guided neural object rendering. In *International Conference on Learning Representations. ICLR*. <https://openreview.net/forum>, 2020.
- [VLS⁺06] Vaibhav Vaish, Marc Levoy, Richard Szeliski, C. Lawrence Zitnick, and Sing Bing Kang. Reconstructing occluded surfaces using synthetic apertures: stereo, focus and robust measures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2331–2338. IEEE, 2006.
- [WCG87] John R Wallace, Michael F Cohen, and Donald P Greenberg. *A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods*, volume 21. ACM, 1987.
- [WDP99] Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In *Rendering techniques' 99*, pages 19–30. Springer, 1999.
- [Wei01] Yair Weiss. Deriving intrinsic images from image sequences. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 68–75. IEEE, 2001.
- [WFP10] Changchang Wu, Jan-Michael Frahm, and Marc Pollefeys. Detecting large repetitive structures with salient boundaries. *Computer Vision–ECCV 2010*, pages 142–155, 2010.
- [WFP11] Changchang Wu, Jan-Michael Frahm, and Marc Pollefeys. Repetition-based dense single-view reconstruction. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3113–3120. IEEE, 2011.
- [WGGK18] Patrick Wieschollek, Orazio Gallo, Jinwei Gu, and Jan Kautz. Separating reflection and transmission images in the wild. In *Proceedings of the European Conference on Computer Vision*, pages 89–104. Springer, 2018.
- [WGL⁺18] Thomas Whelan, Michael Goesele, Steven J. Lovegrove, Julian Straub, Simon Green, Richard Szeliski, Steven Butterfield, Shobhit Verma, and

- Richard Newcombe. Reconstructing scenes with mirror and glass surfaces. *ACM Transactions on Graphics (TOG)*, 37(4):102, 2018.
- [Whi80] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.
- [WKKN19] Yue Wang, Soufiane Khiat, Paul G Kry, and Derek Nowrouzezahrai. Fast non-uniform radiance probe placement and tracing. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, page 5. ACM, 2019.
- [WLZ⁺18] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [WM19] Chris Wyman and Adam Marrs. Introduction to directx raytracing. In *Ray Tracing Gems*, pages 21–47. Springer, 2019.
- [WMLT07] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. *Rendering techniques*, 2007:18th, 2007.
- [WS99] Gregory Ward and Maryann Simmons. The holodeck ray cache: an interactive rendering system for global illumination in nondiffuse environments. *ACM Transactions on Graphics (TOG)*, 18(4):361–368, 1999.
- [WZQ⁺18] Bojian Wu, Yang Zhou, Yiming Qian, Minglun Cong, and Hui Huang. Full 3d reconstruction of transparent objects. *ACM Transactions on Graphics (TOG)*, 37(4):1–11, 2018.
- [XEOT12] Jianxiong Xiao, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Recognizing scene viewpoint using panoramic place representation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2695–2702. IEEE, 2012.
- [XFT⁺08] Jianxiong Xiao, Tian Fang, Ping Tan, Peng Zhao, Eyal Ofek, and Long Quan. Image-based façade modeling. In *ACM transactions on graphics (TOG)*, volume 27, page 161. ACM, 2008.

- [XRLF15] Tianfan Xue, Michael Rubinstein, Ce Liu, and William T Freeman. A computational approach for obstruction-free photography. *ACM Transactions on Graphics (TOG)*, 34(4):79, 2015.
- [XWL⁺08] Xuemiao Xu, Liang Wan, Xiaopei Liu, Tien-Tsin Wong, Liansheng Wang, and Chi-Sing Leung. Animating animal motion from still. In *ACM Transactions on Graphics (TOG)*, volume 27, page 117. ACM, 2008.
- [YBCLS13] Sid Yingze Bao, Manmohan Chandraker, Yuanqing Lin, and Silvio Savarese. Dense object reconstruction with semantic priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1264–1271. IEEE, 2013.
- [YLS20] Lei Yang, Shiqiu Liu, and Marco Salvi. A survey of temporal antialiasing techniques. *STAR*, 39(2), 2020.
- [YYM05] Jingyi Yu, Jason Yang, and Leonard McMillan. Real-time reflection mapping with parallax. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 133–138. ACM, 2005.
- [ZC93] Josiane Zerubia and Rama Chellappa. Mean field annealing using compound gauss-markov random fields for edge detection and image estimation. *IEEE Transactions on Neural Networks*, 4(4):703–709, 1993.
- [ZCA⁺09] Ke Colin Zheng, Alex Colburn, Aseem Agarwala, Maneesh Agrawala, David Salesin, Brian Curless, and Michael F Cohen. Parallax photography: creating 3d cinematic effects from stills. In *Proceedings of Graphics Interface 2009*, pages 111–118. 2009.
- [ZKU⁺04] C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *ACM transactions on graphics (TOG)*, 23(3):600–608, 2004.
- [ZRS05] Rhaleb Zayer, Christian Rossl, and H-P Seidel. Discrete tensorial quasi-harmonic maps. In *International Conference on Shape Modeling and Applications 2005 (SMI'05)*, pages 276–285. IEEE, 2005.

- [ZSW⁺10] Qian Zheng, Andrei Sharf, Guowei Wan, Yangyan Li, Niloy J. Mitra, Daniel Cohen-Or, and Baoquan Chen. Non-local scan consolidation for 3d urban scenes. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 29(4):94:1–94:9, 2010.
- [ZYZQ12] Peng Zhao, Lei Yang, Honghui Zhang, and Long Quan. Per-pixel translational symmetry detection, optimization, and segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 526–533. IEEE, 2012.
- [ZZP⁺17] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 633–641. IEEE, 2017.
- [ZZP⁺19] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.