MeshSplats: Mesh-Based Rendering with Gaussian Splatting Initialization

Rafał Tobiasz*[1,2,3][0009−0002−9265−6148], Grzegorz Wilczyński*[1,2,3][0009−0002−6053−4410], Marcin Mazur[1][0000−0002−3440−8173], Sławomir Tadeja[2][0000−0003−0455−4062], Weronika Smolak-Dyżewska[1,2][0009−0009−1454−3157], and Przemysław Spurek[1,3][0000−0003−0097−5521]

[1] Faculty of Mathematics and Computer Science, Jagiellonian University, Poland
[2] Doctoral School of Exact and Natural Sciences, Jagiellonian University, Kraków, Poland
[3] Department of Engineering, University of Cambridge, United Kingdom
[4] IDEAS Research Institute, Poland

Abstract. Gaussian Splatting (GS) is an emerging, pivotal technique in 3D computer graphics. Most GS-based algorithms exclude the use of classical methods such as ray tracing, which offer numerous inherent advantages in rendering. For example, ray tracing can handle incoherent rays for advanced lighting effects, including shadows and reflections. To address this issue, we introduce MeshSplats, a method which converts GS to a mesh-like format. Following the completion of training, MeshSplats transforms Gaussian elements into mesh faces, enabling rendering using ray tracing methods with all their associated benefits. Our model can be used immediately after transformation, yielding a mesh without additional training, with only a slight reduction in reconstruction quality. We can enhance the quality by applying a dedicated optimization algorithm that operates on mesh faces rather than Gaussian components. Importantly, MeshSplats acts as a wrapper, converting pre-trained GS models into a ray-traceable format. The efficacy of our method is substantiated by experimental results, underscoring its extensive applications in computer graphics and image processing.

Keywords: Gaussian Splatting · Ray Tracing · Mesh Generation · Novel View Synthesis · 3D Computer Graphics

1 Introduction

Classical meshes enable rapid rendering [2], which can be coupled with ray tracing to handle incoherent rays for secondary lighting effects, such as shadows and reflections [13]. Unfortunately, training meshes directly on 2D images is challenging. In contrast, Gaussian Splatting (GS) [8] offers high-quality, real-time 3D reconstruction, but its reliance on rasterization complicates the addition of

* Equal contribution

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_32

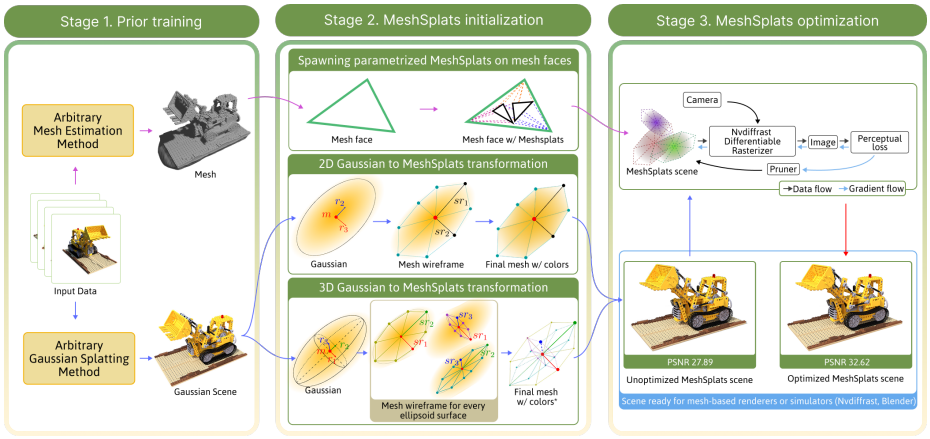

Fig. 1: MeshSplats features two modes: (1) a GS prior, converting trained GS ellipses into fan-type meshes (with optional fine-tuning), and (2) a Mesh prior, training triangles directly on external meshes [14, 17].

lighting or reflections [8, 10]. Consequently, recent methods combine GS with ray tracing to achieve such effects [10, 1, 9], but they require dedicated renderers.

We address these limitations with MeshSplats, which converts GS into a disjoint mesh-like representation natively renderable in standard pipelines like Blender[5] or Nvdiffrast[6]. Rather than approximating macro-geometry, MeshSplats acts as a wrapper, transforming individual Gaussians into independent, semi-transparent colored polygons. This enables out-of-the-box ray tracing with complex lighting. We then fine-tune these meshes to mitigate artifacts from ellipse-to-polygon conversion.

While applicable to all GS models (Fig. 1), MeshSplats is particularly effective for flat Gaussians, as their planar morphology simplifies conversion into mesh faces compared to volumetric 3D Gaussians. MeshSplats supports two scenarios (Fig. 1). Using a *GS prior*, it wraps models into disjoint fan-type meshes with minimal tuning, prioritizing render quality over editability. Alternatively, using a *mesh prior* from external tools [14, 17], MeshSplats overlays semi-transparent triangle faces onto a classical geometry mesh. This two-level setup requires more fine-tuning but permits direct manual shape editing. Both seamlessly integrate complex lighting in standard pipelines.

Our contributions are as follows:

- MeshSplats, a novel method converting flat Gaussians into mesh-based formats for direct integration with classical 3D tools;
- native support for traditional rendering environments, eliminating the need for specialized GS renderers;

[5] https://www.blender.org
[6] https://nvlabs.github.io/nvdiffrast

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_32

(a) Qualitative comparison (b) Classical rendering tools

Fig. 2: **(a)** Qualitative results: GT, MeshSplats, and 3DGS (w/o SH). Rows 1-2: Mip-NeRF360 (res 4). Row 3: "Truck" scene. **(b)** MeshSplats supports simulations and rendering in classical tools (e.g., Blender, Nvdiffrast).

– an optimization pipeline that reduces conversion artifacts, refining geometric details for photorealistic rendering.

2 Related Works

3D Gaussian Splatting (3DGS) [8] represented scenes as anisotropic Gaussians optimized by differentiable rasterization. Extensions like Mip-Splatting [20] reduced aliasing, while 2DGS [6] improved surface alignment. However, these methods rely on spherical harmonics to approximate view-dependent effects, resulting in blurry reflections under complex lighting. Works like GaussianShader [7] and 3DGS-DR [19] incorporated environment maps for distant reflections. 3iGS [15] introduced illumination fields via tensorial factorization, but bounded scene assumptions limit real-world use. Crucially, all these methods rely on rasterization, which struggles with incoherent rays needed for effects like shadows and inter-reflections.

Various works sought to overcome rasterization issues by integrating GS with ray tracing. 3D Gaussian Ray Tracing (3DGRT) [10] yields excellent visual effects by encompassing each Gaussian in bounding primitives for efficient ray tracing. Similarly, RaySplats [1] used ellipses as an approximation instead of bounding primitives, and LinPrim [9] employed linear primitives (e.g. octahedra and tetrahedra) for differentiable volumetric rendering. Other methods, such as EnvGS [18], introduced environment Gaussians to model reflections, while IRGS [4] proposed differentiable 2D Gaussian ray tracing for inverse rendering.

While these methods highlight the massive potential of ray tracing and enable advanced lighting effects, they all inherit the structural limitations of GS: they

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_32

require complex Monte Carlo sampling or necessitate highly specialized, custom environments for both training and rendering. In contrast, our approach directly outputs universally compatible meshes.

3 MeshSplats with Gaussian Splatting Prior

The core of our approach consists of three straightforward steps. First, we use classical GS to create a collection of Gaussians. Next, we apply MeshSplats to transform Gaussian elements into separate mesh faces that approximate their shapes while preserving color and opacity. This step results in a high-quality reconstruction (Fig. 2), although minor artifacts may still be visible. Finally, we refine this representation as in GS, using separate mesh faces instead of Gaussians. Fig. 1 demonstrates the scheme of MeshSplats transformation.

Gaussian Splatting. The GS reconstructs a 3D scene using 3D Gaussians, each defined as $(\mathcal{N}(\mathbf{m}, \Sigma), \sigma, c)$, where $\mathbf{m}$ is the mean (position), $\mathbf{\Sigma}$ is the covariance matrix, σ is the opacity, and c is the color. Colors are typically represented with Spherical Harmonics (SH) [3, 11], but can be replaced with standard RGB at a slight cost to view-dependent quality. In MeshSplats, we use RGB colors for mesh compatibility, enabling external lighting with a minor loss in quality.

MeshSplats Transformation for Flat Gaussians. While several approaches use flat Gaussians [5, 16, 6], we adopt the GaMeS representation [16], defined as $\mathcal{N}(\mathbf{m}, R, S)$, where $S = \mathrm{diag}(\varepsilon, s_2, s_3)$ enforces flatness and $R = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]$. Discarding the negligible s_1 dimension, we approximate each 2D Gaussian as a triangle fan inscribed in an ellipse (Fig. 1).

This mapping uses three hyperparameters: $scale\_mul = 2.7$, no_triag, and $opac\_mul$. The ellipse axes are defined as $\text{scaled\_rot}_k = scale\_mul \cdot \exp(s_{k+1}) \cdot \mathbf{r}_{k+1}$ for $k \in \{1, 2\}$. We construct the triangle fan by sampling no_triag boundary vertices v_i via evenly distributed angles $\theta_i \in [-\pi, \pi]$:

$$\mathrm{v}_i = \mathbf{m} + \cos(\theta_i) \cdot \text{scaled\_rot}_1 + \sin(\theta_i) \cdot \text{scaled\_rot}_2. \tag{1}$$

Vertices inherit the Gaussian's RGB color. To simulate the Gaussian fall-off, opacity is linearly interpolated: the center ($\mathbf{m}$) keeps the exact opacity, while boundary vertices (v_i) are scaled by $opac\_mul$.

MeshSplats Transformation for 3D Gaussians. The 3D Gaussians transformation into meshes builds upon the 2D methodology but extends it to three dimensions. Instead of discarding the smallest scale, all three scale constants (s_1, s_2, s_3) and rotation vectors $(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)$ are retained to compute three orthogonal principal axes. Using the same hyperparameters, no_triag points are generated along the boundaries of the three orthogonal surfaces spanned by these axes. Shared points at surface intersections are merged to avoid redundant vertices and ensure a seamless mesh. Triangle fans are assigned to each of the three surfaces, resulting in $3 \cdot no\_triag$ triangles per Gaussian. Color and opacity are assigned exactly as in the 2D case. While this captures the full 3D spatial extent, it results in higher vertex counts, increasing memory usage and rendering time.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_32

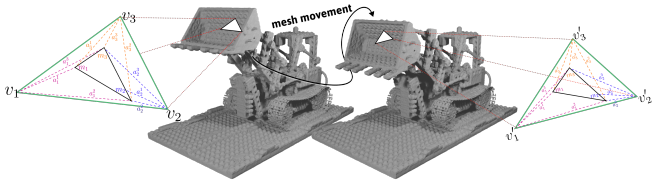

Fig. 3: Using an external mesh prior, MeshSplats employs a base geometry mesh for shape and a parameterized MeshSplats-generated mesh of disjoint triangles, allowing manual editing.

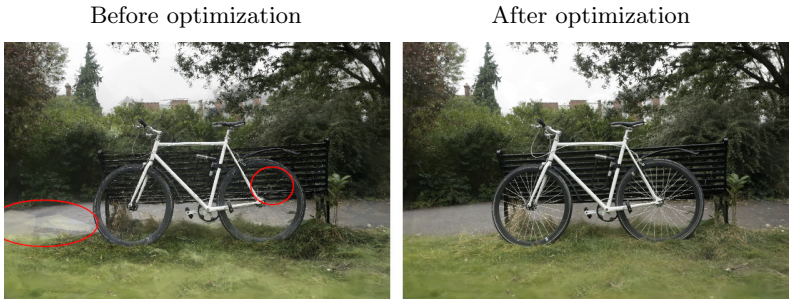

Fig. 4: Before (left) and after (right) MeshSplats optimization in Nvdiffrast. The optimized result eliminates floaters and reconstructs missing fine structures (red), like bicycle spokes.

MeshSplats Rendering and Fine-tuning. To refine the mesh soup and eliminate conversion artifacts, we fine-tune the vertices against ground-truth target scenes. The mesh is rasterized using Nvdiffrast [12], which employs a standard model-view-projection (MVP) matrix and depth-peeling to accurately composite overlapping transparent geometry. We optimize the vertex colors and opacities (learning rate lr_{color}) as well as their positions ($lr_{\text{verts}} = \exp(-3) \cdot lr_{\text{color}}$) using a loss function that blends L_1 and structural similarity: $\mathcal{L} = \lambda \cdot L_1(\hat{y}, y) + (1-\lambda) \cdot \text{SSIM}(\hat{y}, y)$, where $\lambda = 0.6$. To ensure efficiency, a pruning step is executed every 10 epochs. Faces are deleted if all their vertices fall below an opacity threshold of $\exp(-4)$, and orphaned vertices are subsequently removed.

4 MeshSplats with Mesh Prior

When using MeshSplats with a GS prior, we achieve high-quality renderings via a simple, efficient transformation. However, the resulting mesh is optimized solely for rendering and lacks precise geometric accuracy, making it unsuitable for detailed shape approximation. Editing these meshes in Blender resembles manipulating point clouds, akin to GaMeS [16]. To overcome this, we integrate

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_32

external mesh estimation tools as priors [5]. MeshSplats utilizes the estimated geometry mesh as a base, and constructs a secondary layer composed of disjoint, colored triangles with associated opacity attributes directly on top of it. This approach uses standard triangular elements instead of the fan-type meshes typical of the GS prior, significantly reducing the total face count and enabling precise control over rendering quality and manual editing (Fig. 3). Training is efficiently constrained to the geometry mesh surface, and the initialization of secondary triangles is directly based on the faces of this mesh.

Consider a single triangular face of the geometry mesh with vertices: $M = \{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3\} \subset \mathbb{R}^3$. Each vertex $\mathbf{v}_i$ of a new triangle $V = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ is expressed as a convex combination of the geometry face vertices: $\mathbf{v}_i(\alpha_1, \alpha_2, \alpha_3) = \alpha_1\mathbf{m}_1 + \alpha_2\mathbf{m}_2 + \alpha_3\mathbf{m}_3$, where the trainable parameters $\alpha_1, \alpha_2, \alpha_3$ satisfy $\alpha_1 + \alpha_2 + \alpha_3 = 1$ and $\alpha_1, \alpha_2, \alpha_3 \geq 0$, see Fig. 3. This ensures that all generated triangles remain within the bounds of the original geometry mesh and are therefore optimally positioned for rendering.

To maintain consistency and efficiency, MeshSplats employs a fixed number of triangles per mesh face. MeshSplats is fine-tuned using Nvdiffrast, following a process analogous to that used for the GS prior. This hierarchical and parameterized structure facilitates seamless integration with standard graphics pipelines and manual mesh-editing workflows, enabling advanced editing capabilities.

5 Experiments

In this section, we present the outcomes of our experimental study. For all quantitative comparisons (Tab. 1), we employed MeshSplats with a GS prior to achieve optimal visual quality, as our mesh-prior mode prioritizes editability and geometric control.

Furthermore, we elucidate the behavior of our algorithm after parametrization and training.

Quantitative Results. We evaluated the performance of MeshSplats across three benchmark datasets (Tab. 1). On the Mip-NeRF360 and Tanks and Temples datasets, results were highly comparable to existing methods, aligning closely with state-of-the-art approaches like 3DGS [8] and RaySplats [1]. Despite its mesh-based representation, our method achieved scores matching standard rasterization-based techniques. Noteworthy is MeshSplats's superior performance on the Deep Blending dataset among RGB-based methods, suggesting high efficacy in managing intricate indoor geometries and transparency.

Qualitative Results. Nvdiffrast renders of MeshSplats visually match 3DGS without spherical harmonics (Fig. 2). Both methods exhibit similar artifacts when approximating reflections on transparent surfaces like glass. Although MeshSplats yields marginally coarser reconstructions in high-frequency regions (e.g., dense grass), its overall fidelity remains comparable to 3DGS. Furthermore, Blender's EEVEE engine (Fig. 2) seamlessly renders MeshSplats with coherent geometry, accurate colors, and functional physical light interactions.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_32

Table 1: Quantitative comparison. Despite being mesh-based, MeshSplats matches rasterization methods on Mip-NeRF360 and Tanks and Temples, and excels on Deep Blending (indoor scenes). Dashes indicate that baseline results are unavailable for specific datasets.

		Mip-NeRF360			Tanks and Temples			Deep Blending		
		SSIM ↑	PSNR ↑	LPIPS ↓	SSIM ↑	PSNR ↑	LPIPS ↓	SSIM ↑	PSNR ↑	LPIPS ↓
Spherical Harmonics	MobileNeRF	0.527	23.06	0.430	-	-	-	-	-	-
	NeRF2Mesh	0.523	22.74	0.460	-	-	-	-	-	-
	Plenoxels	0.670	23.63	0.440	0.379	21.08	0.795	0.510	23.06	0.510
	INGP-Base	0.725	26.43	-	0.723	21.72	0.330	0.797	23.62	0.423
	INGP-Big	0.751	26.75	0.300	0.745	21.92	0.305	0.817	24.96	0.390
	M-NeRF360	0.844	**29.23**	-	0.759	22.22	0.257	0.901	29.40	0.245
	3DGS-30K	**0.87**	28.69	0.220	**0.841**	23.14	**0.183**	**0.903**	**29.41**	**0.243**
	3DGRT	0.854	28.71	0.250	0.830	**23.20**	0.222	0.900	29.23	0.315
	GS-IR	0.812	26.57	0.238	-	-	-	-	-	-
	LinPrim	0.803	26.63	0.221	-	-	-	-	-	-
	RadiantFoam	0.830	28.47	**0.210**	-	-	-	0.890	28.95	0.260
RGB	RaySplats	**0.846**	27.31	0.237	**0.829**	**22.20**	**0.202**	**0.900**	**29.57**	0.320
	MeshSplats (our)	0.817	**28.08**	**0.229**	0.766	21.71	0.248	0.890	29.50	**0.254**

6 Conclusions

In this paper, we present MeshSplats, a method that addresses the limitations of GS by transforming it into a mesh-like structure compatible with ray tracing. This enables enhanced rendering with improved lighting, shadows, and reflections. MeshSplats provides an efficient and practical solution that can be further refined through our optimization algorithm. Extensive experiments confirm its effectiveness and versatility across diverse datasets. Despite using a mesh-based representation, MeshSplats achieves photorealistic quality that is comparable to or better than GS, eliminating floaters and refining structural details. These results highlight MeshSplats as a strong alternative for high-quality rendering in computer graphics.

The main limitation of MeshSplats is that artifacts such as fragmented geometry and inconsistent opacity can appear in large, low-texture areas because mesh-based interpolation struggles to mimic the smooth falloff of Gaussians.

Acknowledgments. The project "Effective rendering of 3D objects using Gaussian Splatting in an Augmented Reality environment" (FENG.02.02-IP.05-0114/23) is carried out within the First Team programme of the Foundation for Polish Science co-financed by the European Union under the European Funds for Smart Economy 2021-2027 (FENG).

References

[1] Byrski, K., Mazur, M., Tabor, J., Dziarmaga, T., Kądziołka, M., Baran, D., Spurek, P.: Raysplats: Ray tracing based gaussian splatting. arXiv preprint arXiv:2501.19196 (2025)

[2] Foley, J.D., Van Dam, A., Feiner, S.K., Hughes, J.F., Phillips, R.L.: Introduction to computer graphics, vol. 55. Addison-Wesley Reading (1994)

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_32

[3] Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance fields without neural networks. In: CVPR. pp. 5501–5510 (2022)
[4] Gu, C., Wei, X., Zeng, Z., Yao, Y., Zhang, L.: Irgs: Inter-reflective gaussian splatting with 2d gaussian ray tracing. arXiv preprint arXiv:2412.15867 (2024)
[5] Guédon, A., Lepetit, V.: Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. CVPR (2024)
[6] Huang, B., Yu, Z., Chen, A., Geiger, A., Gao, S.: 2d gaussian splatting for geometrically accurate radiance fields. In: ACM SIGGRAPH 2024 conference papers. pp. 1–11 (2024)
[7] Jiang, Y., Tu, J., Liu, Y., Gao, X., Long, X., Wang, W., Ma, Y.: Gaussianshader: 3d gaussian splatting with shading functions for reflective surfaces. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5322–5332 (2024)
[8] Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 3d gaussian splatting for real-time radiance field rendering. ACM Trans. Graph. **42**(4), 139–1 (2023)
[9] von Lützow, N., Nießner, M.: Linprim: Linear primitives for differentiable volumetric rendering. arXiv preprint arXiv:2501.16312 (2025)
[10] Moenne-Loccoz, N., Mirzaei, A., Perel, O., de Lutio, R., Martinez Esturo, J., State, G., Fidler, S., Sharp, N., Gojcic, Z.: 3d gaussian ray tracing: Fast tracing of particle scenes. ACM Transactions on Graphics (TOG) **43**(6), 1–19 (2024)
[11] Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics (ToG) **41**(4), 1–15 (2022)
[12] Munkberg, J., Hasselgren, J., Shen, T., Gao, J., Chen, W., Evans, A., Müller, T., Fidler, S.: Extracting triangular 3d models, materials, and lighting from images. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8280–8290 (2022)
[13] Peddie, J.: Ray Tracing: A Tool for All. Springer Publishing Company, Incorporated, 1st edn. (2019)
[14] Rosu, R.A., Behnke, S.: Permutosdf: Fast multi-view reconstruction with implicit surfaces using permutohedral lattices. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8466–8475 (2023)
[15] Tang, Z.J., Cham, T.J.: 3igs: Factorised tensorial illumination for 3d gaussian splatting. In: European Conference on Computer Vision. pp. 143–159. Springer (2025)
[16] Waczyńska, J., Borycki, P., Tadeja, S., Tabor, J., Spurek, P.: Games: Mesh-based adapting and modification of gaussian splatting. arXiv preprint arXiv:2402.01459 (2024)
[17] Wang, P., Liu, L., Liu, Y., Theobalt, C., Komura, T., Wang, W.: Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. arXiv preprint arXiv:2106.10689 (2021)
[18] Xie, T., Chen, X., Xu, Z., Xie, Y., Jin, Y., Shen, Y., Peng, S., Bao, H., Zhou, X.: Envgs: Modeling view-dependent appearance with environment gaussian. arXiv preprint arXiv:2412.15215 (2024)
[19] Ye, K., Hou, Q., Zhou, K.: 3d gaussian splatting with deferred reflection. In: ACM SIGGRAPH 2024 Conference Papers. pp. 1–10 (2024)
[20] Yu, Z., Chen, A., Huang, B., Sattler, T., Geiger, A.: Mip-splatting: Alias-free 3d gaussian splatting. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 19447–19456 (2024)

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29924-6_32