Bolt3D: Generating 3D Scenes in Seconds

Supplementary Material

1. More experimental results

1.1. Videos and interactive results

The project website contains interactive results and video results, which we encourage the reader to explore.

1.2. Comparison to video models

Video models have emerged as a powerful tool for novel view systhesis. However, using them to reconstruct a 3D asset from a generated video requires distillation [12], similarly to CAT3D, which significantly increases the runtime of such approaches when applied to 3D reconstruction. Nonetheless, we evaluate the quality of novel views generated by one representative approach, MVSplat360 [2]. We evaluate Bolt3D and MVSplat360 on 2-view and 4-view reconstruction on scenes from DL3DV at 512 × 512 resolution. We take care to input appropriately-sized renders to MVSplat360's diffusion model to match its training resolution, and crop images appropriately for evaluation. In Tab. 1 we observe that renders from Bolt3D's 3D scenes are more accurate than the novel views generated by MVSplat360. In Fig. 1 we find that the conditioning often needed for video models (e.g., ViewCrafter [22], MVSplat360, ReconX [12]) can be brittle, resulting in poor accuracy of generated views. In addition, video models are also slow (5.8 minutes for 56 frames) while Bolt3D reconstructs a 3D asset in 7 **seconds** and renders in real-time.

	Method	PSNR ↑	SSIM ↑	LPIPS ↓	FID ↓
2-view DL3DV	MVSplat360	13.97	0.400	0.575	104.83
	Ours	**17.75**	**0.550**	**0.392**	**64.53**
4-view DL3DV	MVSplat360	15.42	0.422	0.507	76.91
	Ours	**20.64**	**0.652**	**0.311**	**48.28**

Table 1. **MVSplat360**'s video model renderer is less accurate and more than 200× slower than Bolt3D for 2- and 4-view DL3DV.

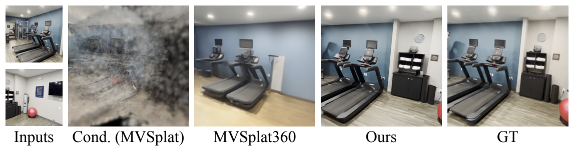

Figure 1. MVSplat360 uses MVSplat for conditioning, which works poorly when there is little or no input view overlap.

1.3. Ablations

Ablation—Geometry VAE. We observe in Tab. 2 that training the encoder (rather than using a frozen, pre-trained one) is important for high autoencoding precision, likely due to pointmaps being outside of the value range on which the encoder was pre-trained. Removing the weighting on distant points (Eq. (2)) or the gradient loss (Eq. 7 main paper) reduces performance of our system.

	Rel. ↓	$\delta_{1.01}$ ↑	Δuv(px) ↓
Full model	**0.99**	**73.0**	**2.56**
- encoder training	1.63	58.9	3.79
- $\mathcal{L}_{\text{rec}}$ re-weighting	1.68	56.7	5.43
- $\mathcal{L}_{\text{grad}}$	1.16	69.8	2.96

Table 2. **Geometry ablation.** Removing encoder training or geometry-specific losses leads to worse performance.

Ablation—Gaussian Head. In Tab. 3 we illustrate that the design choices in the Gaussian Head are important for high-quality rendering. Using fewer (8, rather than 16) views reduces scene coverage and thus incurs a bigger rendering error. Cross attention is important because it allows modulating opacity in splatter images depending on visibility from other views. Using constant opacity and scale parameters reduces rendering quality. Interestingly, allowing the means to be modified by the Gaussian Head also drops performance, suggesting that explicit geometry losses give a stronger supervisory signal, consistent with comparisons to Wonderland [10] in the main paper. Lastly, forcing the Gaussians to lie on rays is advantageous for rendering quality.

Low-resolution comparisons Most methods evaluate performance at lower resolution than our method can handle, and sometimes train at a different resolution, making apples-to-apples comparison challenging. We make the best effort at comprehensive testing at different possible input resolutions to present baseline methods in the most favorable chance. First, we verify that the best way to evaluate baseline methods at high resolution is to feed in a high resolution image Tab. 4, rather than upscaling outputs from lower-resolution input. This is the method we use for all baseline methods in the main paper.

Next, we evaluate performance at a lower resolution, closer to the training setup of baseline methods. In Tab. 5 we illustrate that at a lower 256 × 256 resolution, our model still outperforms Depthsplat. Finally, we give Depthsplat an advantage by evaluating its performance when receiving wide field-of-view (fov) 256 × 448 images. In this setting, the wider field-of-view provides more scene coverage by a factor of ×1.75 and more cues for matching across different views. Only in this setting do we find that Depthsplat achieves similar performance to our method.

	PSNR ↑	SSIM ↑	LPIPS ↓
Ours	**24.72**	**0.831**	**0.209**
Fewer views at inference	24.30	0.823	0.221
No cross-attention	23.80	0.804	0.239
No Gaussian Head	21.94	0.734	0.343
XYZ learnt from rendering	21.88	0.755	0.311
No ray-clipping	20.78	0.642	0.288

Table 3. **Appearance ablation.** All components of architecture, training and inference are important for high-quality appearance.

Resolution input to network	PSNR ↑	SSIM ↑	LPIPS ↓
256×256	23.49	0.845	0.203
512×512	**24.17**	**0.875**	**0.183**

Table 4. **Evaluating Depthsplat at high resolution.** We verify that the most advantageous setting for Depthsplat when evaluating it on high 512×512 resolution is to use a high-resolution input.

	Method	Input Res.	PSNR ↑	SSIM ↑	LPIPS ↓
1-view RE10K	Flash3D	256×256	17.70	0.616	0.393
	Ours	256×256	**21.62**	**0.804**	**0.202**
3-view RE10K	Depthsplat	256×256	24.69	0.873	0.126
	Ours	256×256	**27.39**	**0.916**	**0.103**
2-view DL3DV	Depthsplat	256×448	18.09	0.549	0.323
	Depthsplat	256×256	16.16	0.467	0.388
	Ours	256×256	**18.01**	**0.556**	**0.320**
4-view DL3DV	Depthsplat	256×448	21.20	0.697	0.208
	Depthsplat	256×256	19.64	0.633	0.254
	Ours	256×256	**21.16**	**0.695**	**0.231**
6-view DL3DV	Depthsplat	256×448	21.93	0.730	0.184
	Depthsplat	256×256	20.64	0.680	0.225
	Ours	256×256	**22.18**	**0.733**	**0.206**

Table 5. **Low-resolution 256×256 comparisons.** Our method outperforms competitors at low resolution when receiving the same input information. Only when DepthSplat [21] receives 1.75 times more input information than our method by ingesting wide-fov images, does its performance become similar to ours.

2. Implementation details

2.1. XYZ normalization

Relativization. The supervising (pseudo-ground truth) data used to train our diffusion model is reconstructed using off-the-shelf 3D reconstruction algorithms (MASt3R) [9]. We transform this 3D reconstruction to the view-space of the first camera, such that all point coordinates and all cameras are relative to this coordinate frame: $\mathbf{\Pi}, {}^{xyz} := \mathbf{\Pi}\mathbf{\Pi}_0^{-1}, {}^{xyz}\mathbf{\Pi}_0^{-1}$, where $\mathbf{\Pi}_0$ denotes the camera-to-world rigid body transform of the first camera.

Scaling. We normalize the 3D scale of the reconstructed scenes by applying a per-scene scaling factor α to the camera poses and point coordinates: $\mathbf{\Pi}, {}^{xyz} = \alpha\mathbf{\Pi}, \alpha^{xyz}$. This scale factor is chosen such that the mean depth value from the first camera is the same across every scene in our dataset: $\alpha_0^{xyz}[z] = 1$.

Re-weighting Points in VAE Reconstruction Loss. In our VAE reconstruction loss (Eq 5. main paper), we introduce a re-weighting scheme for two reasons: 1) ground truth points far from the scene center are more likely to be incorrect, and 2) points with high magnitude would make up a large proportion of an equal weighting loss.

For each scene, the point maps are defined in the coordinate system of the "first camera." When computing the reconstruction loss, we first transform each point $\mathbf{x} \in \mathbf{P}$ to the local camera coordinate system:

$$\mathbf{x}_{\text{local}} = [R \mid T]_{\text{w2c}}\, \mathbf{x}, \tag{1}$$

where $[R \mid T]_{\text{w2c}}$ is the world-to-camera transformation matrix. Because scenes are scaled such that the mean depth to the first camera is 1, we can think of $[0, 0, 1]^\top$ as the look-at point or center of the scene, so $d = \left\|\mathbf{x}_{\text{local}} - [0, 0, 1]^\top\right\|$ is the distance of the point to the center of the scene of the local camera. Thus, we compute:

$$\mathcal{L}_{\text{rec}} = \frac{2\sqrt{w} - 1}{w} \left\|\hat{\mathbf{x}}_{\text{local}} - \mathbf{x}_{\text{local, gt}}\right\|^2 \tag{2}$$

where $w = \max(1, d^2)$ is the bounded squared distance to the local scene center and $\frac{2\sqrt{w}-1}{w}$ is the Jacobian of the contraction function defined in MipNerf-360 [1].

2.2. Architecture and training details.

Diffusion model. We use a U-Net with full 3D attention on all feature maps up to 32×32, as in CAT3D [6]. Unlike CAT3D, our diffusion model is trained to model the joint distribution of latent appearance and geometry. To this end, we increase the number of channels in the input and output layers of CAT3D's architecture by 8 to additionally accept geometry latents. The input to our network thus has 8-dimensions for the geometry latents, 8-dimensions for the image latent, 6-dimensions for the camera pose raymaps, and a 1-dimensional mask indicating which views are given as conditioning, yielding an input dimension of 64×64×23. We train with the same optimization hyperparameters as [6], except we additionally finetune on 16 input views with a lower learning rate of $1e-5$.

Autoencoder. We use a pre-trained and frozen image autoencoder similar to that of Stable Diffusion [14]. The geometry encoder has the same architecture, except we increase the channel dimension to additionally accept a 6-dimensional camera pose representation. The decoder is a transformer-based network. We patchify the $64 \times 64 \times 8$ latent with patch size 2, thus using a token length 1024. We use the ViT-B architecture hyperparameters: 12 layers with channel size 768, with the fully-connected layer consisting of 2 dense layers with GeLU activation function and a hidden MLP dimension 3072. The linear projection head projects each token to a 16×16 patch. We optimize the parameters of the Autoencoder with the Adam [5] optimizer using constant learning rate $1e-4$, batch size 512, Adam parameters $(\beta_1, \beta_2) = (0.0, 0.99)$. We first train for 3M iterations at 256×256 resolution, followed by fine-tuning at 512×512 for 250k iterations. We use loss weight parameters $\lambda_1 = 3e-9$ and $\lambda_2 = 0.033$.

Foreground masking for synthetic data. In synthetic data, we apply loss $\mathcal{L}_{rec}$ only on foreground pixels and train the model to additionally output a foreground alpha mask, supervised with binary cross-entropy loss.

Gaussian head. We detail the Gaussian head architecture in Fig. 2. The Gaussian Head receives as input the 3-dimensional image, 3-dimensional pointmap, and 6-dimensional raymap encoding the camera pose. Each view is first passed through 3 residual convolutional blocks with the Swish activation function [13] and channel size 128, followed by 4×4 patchification to token dimension 128 and full cross attention. We use 3 transformer layers with hidden dimension 128 and 8 heads, MLP dimension 512. The tokens are then unpatchified to original resolution and 128 channel dimension. Following that, there are another 3 residual convolutional blocks, and a final, unactivated 3×3 convolutional layer that outputs channel size 11 (3 for color, 3 for size, 4 for rotation and 1 for opacity). The outputs are activated with an exponential function for scale and a sigmoid function for opacity. To facilitate accurate scale prediction, the size output by the network is then multiplied by the z-distance of the gaussian from the camera. The means of Gaussians are not predicted by the Gaussian head, as they are already available from the VAE. The Gaussian head is trained with 8 input views, leading to a sequence length $131k$. We manage this computational complexity by using FlashAttention [3, 4] and rematerializing gradients on the dot product operation. For losses, we use L2 photometric loss with weight $\lambda = 1$ and LPIPS loss weight $\lambda_{LPIPS} = 0.05$. We train with learning rate $1e-4$ and batch size 8. When training the Gaussian Head, the Geometry and Image autoencoders are frozen.

Gaussian head for viewer assets To enhance rendering performance and reduce the memory footprint of assets for the viewer on our project website, we add an L1 regularizer term to encourage completely transparent Gaussians when they are not necessary, similar to LongLRM [23]. Gaussians with low opacity are then culled before saving the asset. To further reduce file sizes, the model data is quantized in chunks of 256 gaussians (`https://github.com/playcanvas/splat-transform`).

2.3. Inference.

Sampling details. We train our diffusion model [7] using v-parameterization [15] with T=1000 timesteps. At inference time, we use DDIM [17] to speed up inference to 50 steps using the same noise schedule as CAT3D [6] except with zero terminal SNR [11].

Camera path sampling. We use the same camera path heuristics as CAT3D – sampling circular paths, forward-facing paths and splines. We use much fewer views than CAT3D (16 vs their 800), so we sample camera paths on only one path, typically with the median radius and height of cameras in the training set, without offsets or scaling [6].

3. Limitations, discussion and future work

Limitations. While our method can produce a wide range of geometries, it still struggles on thin structures, especially those that are fewer than 8 pixels wide (the spatial downsampling ratio of our geometry VAE). Our method also struggles with scenes that have large amounts of transparent or highly non-lambertian surfaces, for which geometry reconstruction in Structure-from-Motion frameworks is typically inaccurate.

Our model is also sensitive to the distribution of the target cameras, in particular the up-vector chosen to generate the camera path as well as the scene scale. Perhaps these could ameliorated in future work with better data augmentation.

Discussion and future work. To the best of our knowledge, Bolt3D is the first work to explore the architecture and training recipe of a Geometry VAE, and there remain several design choices to be explored. In particular, we chose compress pointmaps over depth due to their resounding success in multi-view reconstruction (Dust3r [19], Mast3r [9]), but concurrent work shows that inferring depth can be complimentary [8] or even advantageous [18]. We leave exploration of these findings in context of 3D generation to future works.

Next, despite making a significant step in feed-forward 3D generation, the quality of 3D generation, Bolt3D lacks in quality compared to optimization-based methods such as CAT3D. We hypothesize this is due to CAT3D generating much more views (≈800, compared to our 16), resulting in more complete scene coverage. Generating more views, perhaps through an anchoring strategy [16], could improve the quality of results, though it would result in a large number of 3D Gaussians.

Finally, Bolt3D generates exclusively static scenes. Perhaps future work could combine multi-view video diffusion models [20] with Bolt3D's direct geometry generation to generate dynamic 3D scenes in a feed-forward manner.

4. Experimental details

DL3DV scenes. We ran evaluation on the intersection of our test set and the public test benchmark. The scenes used for evaluation were:

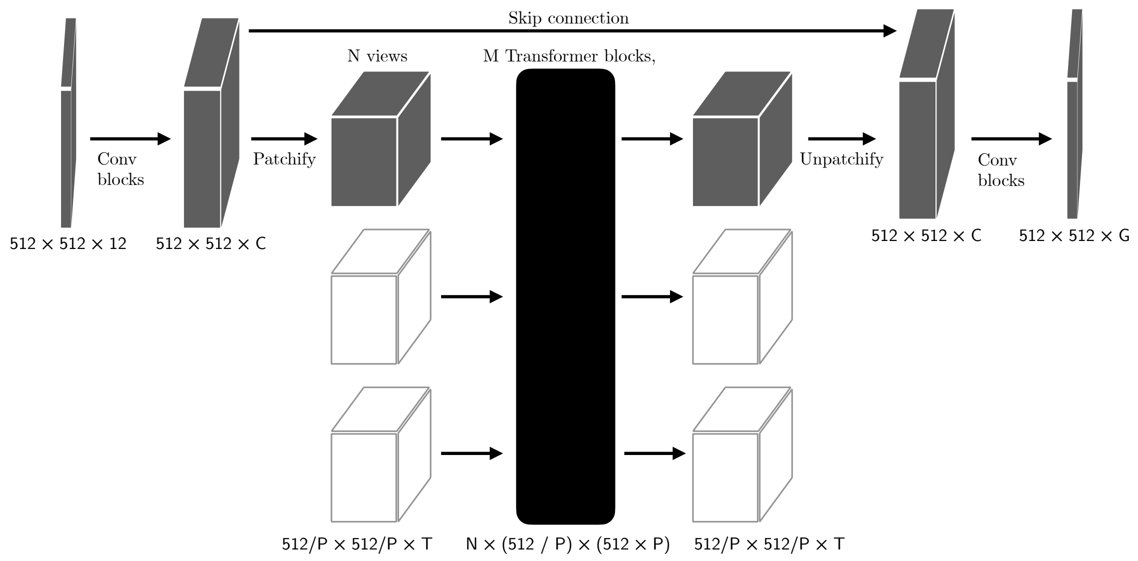

Figure 2. Our Gaussian head architecture consists of convolutional and transformer blocks, with patchification to manage the cross-attention sequence length.

- `0569e83fdc248a51fc0ab082ce5e2baff15755c53c207f545e6d02d91f01d166`,
- `073f5a9b983ced6fb28b23051260558b165f328a16b2d33fe20585b7ee4ad561`,
- `183dd248f6a86e07c5adf9de8ee2d0abe45b1216331c03678e89634c2e9b1c7f`,
- `1ba74c22670ad047981441581d00f26f4a148d1010bcac7468c615adf5fa4d5d`,
- `389a460ca1995e0658e85fe8e6b520b4e88b370cd6710dfe728b1564bba31aee`,
- `493816813d2d6d248eb3c2b0b77b63e54235266e9a06e270fd0d282f13960493`,
- `50c46cf8b8b22c8d2ffdef8964b05ddbceaef312c9a9ff331d1ecebfd223f72a`,
- `4ae797d07b6d1644c9db6919c8cc8c0d28d72be45108ac7a3abf8dc21b599d83`,
- `565553aa894be621e8b4773cac288e60ad0c2cf7edb621be62b348c9a0f78380`,
- `599ca3e04cae3ec83affc426af7d0d7ab36eb91cd8e539edbc13070a4d455792`,
- `5c8dafad7d782c76ffad8c14e9e1244ce2b83aa12324c54a3cc10176964acf04`,
- `63798f5c6fbfcb4eb686268248b8ecbc8d87d920b2bcce967eeaedfd3b3b6d82`,
- `946f49be73928469000baa5ca04d2573137c5ee6a66362bcf8d130354dca8924`,
- `9e9a89ae6fed06d6e2f4749b4b0059f35ca97f848cedc4a14345999e746f7884`,
- `cd9c981eeb4a9091547af19181b382698e9d9eee0a838c7c9783a8a268af6aee`,
- `d4fbeba0168af8fddb2fc695881787aedcd62f477c7dcec9ebca7b8594bbd95b`.

References

[1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR*, 2022. 2

[2] Yuedong Chen, Chuanxia Zheng, Haofei Xu, Bohan Zhuang, Andrea Vedaldi, Tat-Jen Cham, and Jianfei Cai. MVSplat360: Feed-forward 360 Scene Synthesis from Sparse Views. *NeurIPS*, 2024. 1

[3] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024. 3

[4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, 2022. 3

[5] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *ICLR*, 2015. 2

[6] Ruiqi Gao, Aleksander Holynski, Philipp Henzler, Arthur Brussee, Ricardo Martin-Brualla, Pratul P. Srinivasan, Jonathan T. Barron, and Ben Poole. CAT3D: Create Anything in 3D with Multi-View Diffusion Models. *NeurIPS*, 2024. 2, 3

[7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. *NeurIPS*, 2020. 3

[8] Zeren Jiang, Chuanxia Zheng, Iro Laina, Diane Larlus, and Andrea Vedaldi. Geo4d: Leveraging video generators for geometric 4d scene reconstruction. *ICCV*, 2025. 3

[9] Vincent Leroy, Yohann Cabon, and Jérôme Revaud. Grounding Image Matching in 3D with MAST3R. *ECCV*, 2024. 2, 3

[10] Hanwen Liang, Junli Cao, Vidit Goel, Guocheng Qian, Sergei Korolev, Demetri Terzopoulos, Konstantinos N Plataniotis, Sergey Tulyakov, and Jian Ren. Wonderland: Navigating 3D Scenes from a Single Image. *arXiv preprint arXiv:2412.12091*, 2024. 1

[11] Shanchuan Lin, Bingchen Liu, Jiashi Li, and Xiao Yang. Common Diffusion Noise Schedules and Sample Steps are Flawed. In *WACV*, 2024. 3

[12] Fangfu Liu, Wenqiang Sun, Hanyang Wang, Yikai Wang, Haowen Sun, Junliang Ye, Jun Zhang, and Yueqi Duan. Reconx: Reconstruct any scene from sparse views with video diffusion model. *arXiv:2408.16767*, 2024. 1

[13] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for Activation Functions. *arXiv preprint arXiv:1710.05941*, 2017. 3

[14] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *CVPR*, 2022. 2

[15] Tim Salimans and Jonathan Ho. Progressive Distillation for Fast Sampling of Diffusion Models. *ICLR*, 2022. 3

[16] Kyle Sargent, Zizhang Li, Tanmay Shah, Charles Herrmann, Hong-Xing Yu, Yunzhi Zhang, Eric Ryan Chan, Dmitry Lagun, Li Fei-Fei, Deqing Sun, et al. ZeroNVS: Zero-Shot 360-Degree View Synthesis from a Single Image. *CVPR*, 2024. 3

[17] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *ICLR*, 2021. 3

[18] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. 2025. 3

[19] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. DUSt3R: Geometric 3D Vision Made Easy. *CVPR*, 2024. 3

[20] Rundi Wu, Ruiqi Gao, Ben Poole, Alex Trevithick, Changxi Zheng, Jonathan T. Barron, and Aleksander Holynski. CAT4D: Create Anything in 4D with Multi-View Video Diffusion Models. 2025. 3

[21] Haofei Xu, Songyou Peng, Fangjinhua Wang, Hermann Blum, Daniel Barath, Andreas Geiger, and Marc Pollefeys. DepthSplat: Connecting Gaussian Splatting and Depth. *CVPR*, 2025. 2

[22] Wangbo Yu, Jinbo Xing, Li Yuan, Wenbo Hu, Xiaoyu Li, Zhipeng Huang, Xiangjun Gao, Tien-Tsin Wong, Ying Shan, and Yonghong Tian. ViewCrafter: Taming Video Diffusion Models for High-fidelity Novel View Synthesis. *arXiv:2409.02048*, 2024. 1

[23] Chen Ziwen, Hao Tan, Kai Zhang, Sai Bi, Fujun Luan, Yicong Hong, Li Fuxin, and Zexiang Xu. Long-LRM: Long-sequence Large Reconstruction Model for Wide-coverage Gaussian Splats. *ICCV*, 2025. 3