Towards Real-time G-buffer-Guided Style Transfer in Computer Games

Eleftherios Ioannou and Steve Maddock

***Abstract*—Artistic Neural Style Transfer (NST) has achieved remarkable success for images. However, this is not the case for dynamic 3D environments, such as computer games, where temporal coherence remains a challenge. Our paper presents an approach that uses the G-buffer information available in a game pipeline to generate robust and temporally consistent in-game artistic stylizations based on a style reference image. We use a synthetic dataset created from open-source computer games and demonstrate that the utilization of depth, normals, and edge information enables the stylization process to be more aware of the geometric and semantic aspects of a game scene. The proposed approach builds on previous work by injecting style transfer in the rendering pipeline, while also utilizing G-buffer information during inference time to improve upon the stability of the stylizations, offering a controllable way to stylize computer games in terms of temporal coherence and content preservation. Qualitative and quantitative evaluations of our in-game stylization network demonstrate significantly higher temporal stability compared to existing style transfer approaches when stylizing 3D computer games.**

***Index Terms*—Neural style transfer, computer games, G-buffer, convolutional neural network (CNN), graphics pipeline.**

I. Introduction

Neural Style Transfer (NST) is the process of transferring the style of an artwork onto an input content image. The objective of Artistic Image NST methods is to synthesize an output image that seamlessly blends the style elements, such as colors and textures, from the reference style image with the content information present in the input content image. Since the seminal work by Gatys et al. [1], NST research has shown remarkable advances with a variety of approaches [2]–[5] capable of producing visually impressive results.

Despite the great progress of NST, and its applicability to different data spaces (e.g., image, video, 3D, radiance fields), its utilization in 3D computer games remains relatively unexplored. Stylizing computer games could be treated similarly to the problem of stylizing videos or image sequences, as temporal consistency is the main challenge. However, our experiments have shown that utilizing image [2], [6] or video [7]–[9] NST methods to stylize each rendered frame sequentially does not result in appealing stylizations that are also temporally consistent.

Whilst 3D computer games encapsulate more difficulties, such as the synthetic nature of the data, they offer additional three-dimensional information that can be exploited [10]–[12]. Previous work has shown that stylizations of game scenes can be significantly improved when employing G-buffer data [10], [11]. Nevertheless, temporal consistency is not addressed. An attempt was made to integrate NST in the game's pipeline using color buffer data [12], yet G-buffer data is underutilized and temporal stability remains unsolved. In addition to improving stylization quality utilizing G-buffer data and integrating NST in the rendering process [12], here, we demonstrate more ways in which a game pipeline's intermediate data can be used to generate enhanced and temporally stable in-game artistic stylizations.

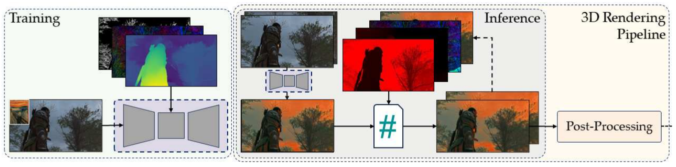

Fig. 1. Our stylization network is trained on G-buffer data. It is then integrated into the 3D rendering pipeline, while G-buffer information is also used in a novel algorithm to improve the quality and temporal coherence of stylized game scenes.

In this paper, we propose a solution that not only considers G-buffer data during training but also embeds an NST module in the rendering pipeline and makes use of depth, normal and motion vector data during inference for improved temporal stability (Figure 1). Our stylization framework is specifically trained on data extracted from open-source games. The trained stylization network is injected in the 3D rendering pipeline [12], while G-buffer information is employed to enforce consistency across subsequent frames. The contributions of the proposed in-game stylization approach can be summarized as follows:

- We develop a novel G-buffer-guided stylization algorithm seamlessly integrated into a graphics pipeline. As G-buffer data is available during rendering, motion vectors, depth and normal data are used to significantly improve the quality and stability of the stylizations. Motion vectors track the color of the stylized frame's pixels to the previous stylized frame. Linear interpolation is employed to calculate the final color. Depth information fine-tunes this interpolation, while the inclusion of depth and normals ensures the preservation of global structures.
- We design a simple, yet effective stylization network whose composition allows for utilization in a computer games pipeline. A synthetic dataset is used to train the network with G-buffer fusion. Concatenated depth, normal, and edge features are fused into the stylization network making it more aware of the geometric and semantic aspects of a game scene. The stylization network is injected into the 3D rendering pipeline before the post-process stage.
- Synthetic datasets that encapsulate a wide variety of lighting and environment characteristics are notably limited.

We create a synthetic dataset from open-source computer games. A small dataset of $\sim$10k images is compiled from 3D scenes with randomized environment conditions, such as lighting and post-process effects.

- Qualitative and quantitative experiments validate the effectiveness of our approach which showcases the potential of integrating NST in computer games.

II. Related Work

A. Image style transfer

Initially, Gatys et al. [1], [13] suggested an image-optimization technique based on content and style features extracted from Convolutional Neural Networks (CNNs), which is capable of faithfully reproducing artistic qualities of artworks on photographs. To alleviate the computational cost required to generate one stylized image, NST research progressed towards offline model optimization methods [14]. These approaches train a model that can be used to produce a stylized result with a single forward pass [15]–[18]. Instead of capturing and transferring the style of one particular artwork, methods for transferring the style of one artist (e.g., Cezanne, Van Gogh) have been developed [19], [20]. Additionally, depth maps and image edges have been used to generate stylized results of high quality and retained structural information [21]–[23]. Arbitrary-style-per-model methods have also emerged [24], garnering considerable attention within NST research in recent years [4]–[6], [8], [25]–[34].

B. Video style transfer

Inconsistent stylizations across subsequent frames, visual artefacts and undesired flickering effects are the challenges in artistic video style transfer. To mitigate these issues, Ruder et al. [35], [36] introduced a temporal constraint leveraging optical flow information and considering disoccluded regions and motion boundaries. Motion field data has also been utilized for the synthesis of stable example-based stylized videos [37], [38]. Other video NST studies attempt to improve stylization quality [19] or computation speed [39], [40]. Similarly to image NST, video NST methods focused on retaining depth and structural information were also proposed [41], [42], as well as multiple-style-per-model [7] and arbitrary-style-per-model techniques [8], [9], [43]–[45]. Some image style transfer approaches have also shown suitability for video style transfer when trained with additional temporal considerations [2], [29], [46].

C. Style transfer for Computer Games

Style transfer approaches tailored for computer game applications primarily focus on bridging the gap between synthetic and realistic worlds [10], [11]. Richter et al. [10] leveraged real-world datasets to improve the photorealism of the Grand Theft Auto V game. Taking into consideration G-buffer information, such as normal, depth, albedo, and glossiness, their suggested image enhancement network intercepts a rendered frame and outputs an enhanced photorealistic image. Later, Mittermueller et al. [11] validated the effectiveness of using intermediate data from a game engine's rendering process in an approach that reduces the gap between real-world and synthetic domains for the image-to-image translation task.

There is only limited work for artistic style transfer applied in a computer game setting. Employing the method of Ghiasi et al. [6], Unity's implementation [47] showcases the integration of an artistic stylization network at the end of the rendering pipeline. Despite achieving multi-style style transfer in a game, the approach is agnostic to any three-dimensional information. This also shows that it is possible to achieve artistic in-game stylizations by attaching any image or video style transfer approach (e.g., [2], [15], [24], [28]) at the end of the post-process stage of a game's rendering pipeline, yet it can be ineffective in terms of temporal stability and structural preservation. To alleviate this issue, an approach to inject style transfer in the rendering pipeline has been suggested [12]. Although integrating a stylization network earlier in the rendering process results in improved artistic stylizations, the method does not take into account G-buffer information but only trains the stylization network on a combination of real-world and synthetic images.

Here, we propose a stylization framework that not only integrates a trained stylization network in the game's rendering pipeline but also takes advantage of the available G-buffer data both in training and during inference. Experiments demonstrate that this leads to significant improvements in the quality and stability of the stylized gameplay.

III. Our Approach

A. Dataset Generation

The available synthetic/game datasets are only restricted to one particular game [48] or animation style [49]. Previous photorealistic style transfer approaches [10], [11] utilize datasets that capture a specific look-and-feel from a real-world dataset [50] or a specific game (Red-Dead-Redemption 2). For a more generalizable approach, we utilize four available open-source computer games [51]–[54] to generate a small-scale dataset using the Unity game engine. Employing the Unity Perception package [55] we are able to capture approximately 10 thousand frames with their corresponding depth and normal maps. We configure multiple scenes in each of the games, with cameras wandering in the synthetic worlds and capturing frames at particular time intervals. To introduce variation in the generated frames, we randomize environment parameters related to the lighting and the post-process effects applied in the game scene. Dataset examples are included in the supplementary material.

B. Stylization Network & G-buffer Fusion

Our stylization network is adopted from the transformation networks suggested by state-of-the-art approaches [15], [21], learning to reproduce a reference style artwork. This design of convolutional layers followed by instance normalization has been shown to produce aesthetically pleasing stylizations that preserve content structures, necessary for an in-game setting. Utilizing simple convolution operations also offers the advantage of exporting the trained model to a format (ONNX

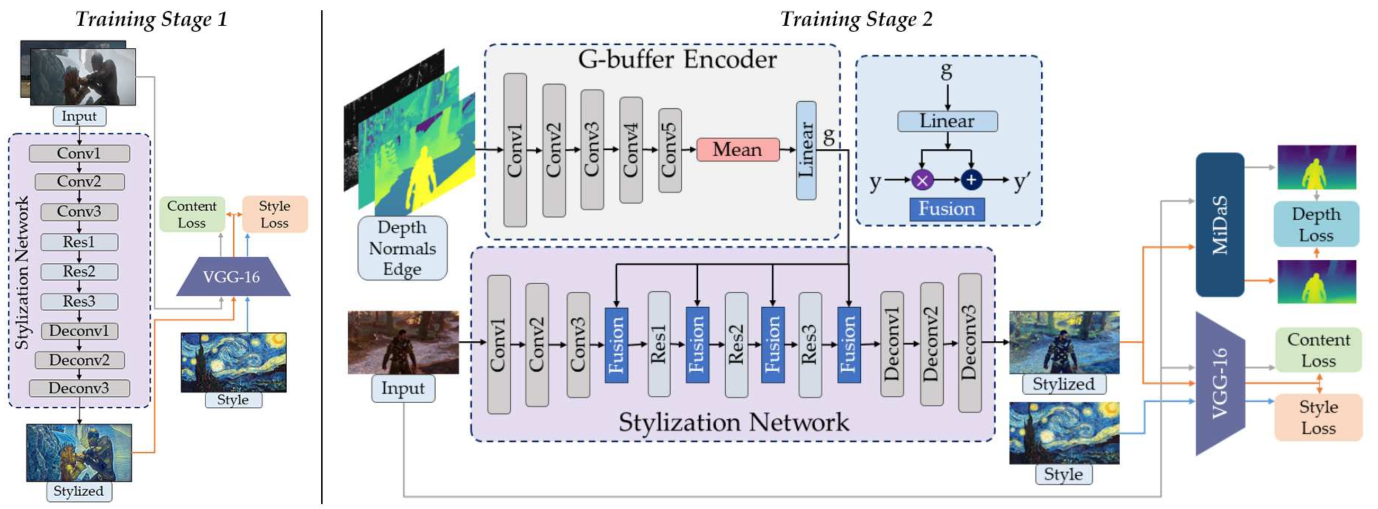

Fig. 2. System Architecture. At Training Stage 1, the stylization network is pre-trained on the MS COCO dataset mixed with MPI Sintel frames. At Training Stage 2, the stylization network is trained on the created synthetic dataset with G-buffer features fusion.

[56]) recognizable by a conventional game engine. Here, we propose some important modifications to the suggested CNN architecture [15], [21], [23] in order to create a G-buffer-aware stylization framework.

Figure 2 displays the overall system architecture. A two-stage progressive training strategy is used [7]. The stylization network is pre-trained on conventional datasets [49], [57] without G-buffer data (*Training Stage 1*). The incorporation of the MS COCO [57] dataset facilitates the generalization of the model to photorealistic scenes. Preliminary experiments have indicated that integrating real-world data contributes to reducing artifacts. Simultaneously, this integration extends the synthetic dataset generated that includes approximately $\sim 10k$ frames. For *Training Stage 2*, firstly, we design a G-buffer encoder, a simple CNN architecture that is used to create feature representations of input concatenated G-buffer data. In addition to depth and normals, the Sobel operator [58] is used to generate an edge map of the input RGB frame. Depth, normal, and edge maps are then concatenated and used as input to the G-buffer encoder. To intercept the learnt G-buffer features (g), we introduce four *fusion* layers, embedded before and after each of the three residual layers. The input y of each *fusion* layer is then transformed to account for the G-buffer information:

$$y' = y \cdot (alpha \cdot (f^l_W(g)) + ((1 - alpha) \cdot f^l_W(g)) \qquad (1)$$

where f^l denotes a linear transformation layer (l) of the trained transformation network (f_W) applied to the input G-buffer features (g), and $alpha$ is a scalar value, set to 0.9. The G-buffer encoder implementation and the setting of the $alpha$ value are adapted from the implementation of Richter et al. [10]. Inspired by this work that aims to enhance the photorealism of game scenes [10], we also resort to a CNN architecture for the G-buffer encoder and demonstrate that global structures and geometry can be maintained.

1) Loss functions: The content and style losses are adopted from previous work on image stylization [15], [23]. A pre-trained *VGG-16* [59] (denoted by ϕ) is used to extract feature representations of the original input frames x and the transformed images $\hat{y}$. Feature representations are extracted from $j = relu2\_2$ layer to define the content loss as:

$$\mathcal{L}^{\phi_0}_{content}(\hat{y}, x) = \frac{1}{C_j H_j W_j} \|\phi^j_0(\hat{y}) - \phi^j_0(x)\|^2_2 \qquad (2)$$

where $H \times W \times C$ is the shape of the processed image. To capture the style information, features are extracted from multiple layers of the image recognition network. Style loss is then defined using the calculated Gram-based style representations (G) that give feature correlations:

$$\mathcal{L}^{\phi_0,j}_{style}(\hat{y}, y) = \|G^{\phi_0}_j(\hat{y}) - G^{\phi_0}_j(y)\|^2_F \qquad (3)$$

and it is summed up for all layers j in $J = \{relu1\_2, relu2\_2, relu3\_3, relu4\_3\}$.

To enforce depth preservation and retain the global structure of the game frames, we utilize *MiDaS* [60] and train the system to minimize a depth reconstruction loss defined as:

$$\mathcal{L}^{MiDaS}_{depth}(\hat{y}, x) = \|MiDaS_1(\hat{y}) - MiDaS_1(x)\|^2_2 \qquad (4)$$

This has been shown to effectively preserve depth in the output images while improving the overall quality [21], [23].

C. Style Transfer in the rendering process

Following previous work for in-game artistic stylization [12], we implement a *Custom Pass* in the Unity High Definition Rendering Pipeline (HDRP) [61]. We inject the trained NST network before the Post-Process stage. This leads to improved temporal coherence. It is an efficient way to generate artistic stylizations for game frames while preventing the post-process effects (e.g., Depth-of-Field, Bloom) from being diminished [12].

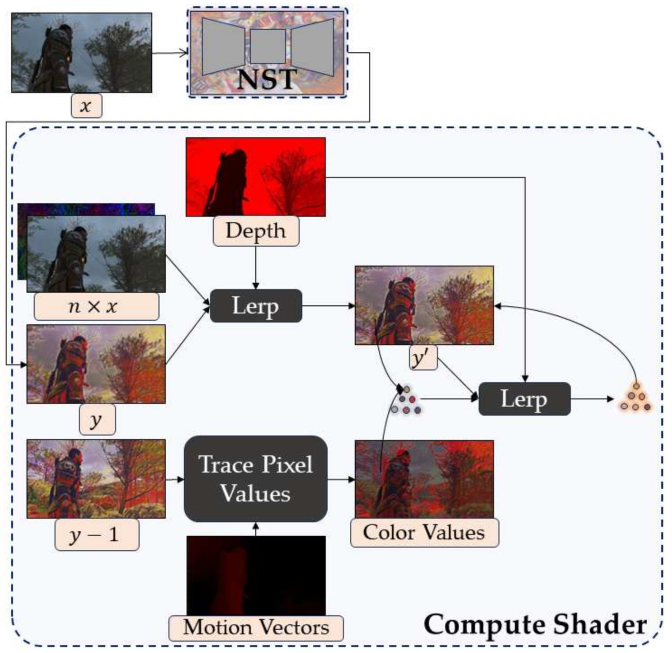

Fig. 3. Implementation of the Compute Shader that is used for in-game G-buffer-guided stylization. The shader intercepts the stylized frame and the previous stylized frame, along with the original input frame, the normal map, the depth map and the motion vector texture. The computation outputs a stylized result that is then passed through the game's pipeline post-process stage before being rendered. Motion Vectors are used to trace the pixel color values of the stylized frame on the previous stylized frame. *Lerp* is used to denote linear interpolation.

1) G-buffer-Guided real-time stylization: In addition, our stylization framework takes advantage of the intermediate data that is available through the rendering process. Figure 3 gives an overview of the implemented shader's algorithm that intercepts the stylized frame along with G-buffer data. The shader computes the new pixel values and outputs the final stylized frame that is rendered after the post-processing stage. Initially, a normal intensity is calculated based on the input normal map:

$$n = max(normal \cdot viewDirection, 0) + 1.0 \tag{5}$$

where $viewDirection = (0, 0, 1)$ is the camera's facing direction. The original input frame (x) is multiplied by the computed normal intensity and a new stylized output is calculated based on the frame's depth map. Using the normal map (and calculated normal intensity), the color frame's pixel values are modulated based on the orientation of the surfaces – the original frame's pixel values are fine-tuned according to the geometric characteristics of the scene. Applying stylization in a game frame often results in a loss of structure and blurring of the edges between the objects. Using normal intensity allows us to apply stylization based on the surface normals and thus emphasize (low-level) features when applying the stylistic effect. Also, using the combination of depth and the original frame (modulated using normal intensity), we can distribute the stylization effect on the input frame in a way that allows for more depth preservation and structure enhancement. The new version of the stylization is calculated as:

$$y' = Lerp(y, n \cdot x, a \times d) \tag{6}$$

where $Lerp$ is used to denote a linear interpolation between the pixels of the stylized frame y and the pixels of the input frame multiplied by the normal intensity, based on the depth value d. The intensity of the depth value is adjusted with a scalar value a. The closer $a \times d$ is to 0, the closer the final y' value is to y; the closer it is to 1, the closer y' is to $n \cdot x$.

During the execution of the Custom Pass, the previous stylized frame is stored at each time step. This is then passed into the implemented shader. Using the motion vectors, we find the pixel color values of the stylized frame on the previous stylized frame, denoted by z. The color of the pixels of the current stylized frame is then computed as an interpolation between the current color of the pixels (of the stylized current frame) and the color of the pixels of the stylized previous frame, controlled by a scaled depth value. The final frame is then computed as:

$$y' = Lerp(y', z, b \times d) \tag{7}$$

where b is a scale factor that can be freely adjusted to control the amount of the weight of the depth value d on the linear interpolation.

Our proposed system uses G-buffer data not only during training but also during inference in order to guide the stylization process. The suggested shader algorithm provides a robust way to control the trade-off between temporal coherence and content preservation. By adjusting the a and b values, we can control the intensity of the depth-based stylization and the intensity of pixel changes between sequential frames respectively.

IV. Experiments

A. Training Details

As the compiled synthetic dataset is not large, we pre-train the stylization network on the MS COCO dataset [57] combined with frames from the MPI Sintel dataset [49], for 2 epochs, and batch size of 2. At this stage, the G-buffer Encoder is not trained and the linear layer of the transformation network is frozen. We then train the entire stylization network and the G-buffer Encoder using the synthetic dataset. The G-buffer features are encoded and fused into the stylization network as described in Section III-B. We train for 2 epochs, a batch size of 2 and the training images are resized to 360×360. The stylization network and the G-buffer encoder are jointly optimized using the Adam optimizer [62] and a learning rate of 1×10^{-3}. The weights for content, style and depth loss are adopted from previous implementations [12], [15], [23]. Similarly to [12], we train a single network for each style.

B. G-buffer-Guided In-Game Style Transfer

Our proposed in-game stylization system is composed of a stylization network pre-trained on a combination of real-world and synthetic frames and then trained on a synthetic dataset with G-buffer information. In addition, G-buffer data

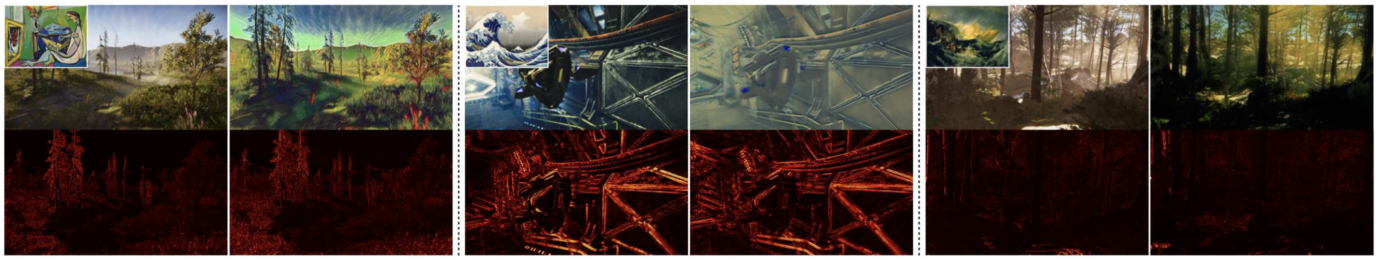
Fig. 4. Our results on different game scenes. The odd columns show the original input frames; the even columns show the corresponding stylizations using our approach. The bottom row shows the temporal error heatmap. Our in-game G-buffer-Guided Stylization framework produces temporally coherent results while capturing the style of the reference artwork.

that is available during the 3D rendering process is utilized to guide the stylization process and produce temporally consistent artistically stylized game scenes. Figure 4 demonstrates examples of our method running on different game scenes and for different styles. To demonstrate the effectiveness of our approach in producing temporally coherent in-game stylizations, heatmaps of the temporal error (difference between the current and previous frame) are provided (bottom row). For each input frame, the difference from the previous frame is calculated. We do the same for the frames generated with our stylization framework. It is noticeable that the heatmaps generated from the stylized frames are very close to the input's temporal error heatmaps – visually coherent results are produced while the style of the reference artwork is captured.

C. *Comparisons with State-of-the-Art Methods*

For an in-depth evaluation of our approach, we compare against state-of-the-art image (AdaAttN [2]), video (CSBNet [9], MCCNet [8], FVMST [7]) and in-game (NSTFCG [12]) artistic style transfer approaches. As we optimize for stable and temporally coherent in-game stylizations we opt to compare our approach with methods that are also optimized for temporal consistency. Evaluation is performed using frames in Full HD (1920×1080) resolution). For fairer comparisons, for the training of our models, the images are kept at a lower resolution, similar to the training practice of the in-comparison methods.

D. *Qualitative Results*

Figure 5 demonstrates that our approach is better than the state-of-the-art approaches in terms of visual quality and temporal stability. As shown in the zoomed-in cut-outs in cyan, some approaches alter the stylization of the same pixels in two sequential frames [2], [7], [8] or introduce artefacts that do not exist in the previous frame [9], [12]. In contrast, our method preserves the fine details and generates clean stylizations without undesired flickering. The zoomed-in cut-outs in pink show that our system reduces the halo effects around objects that are apparent in other approaches [7]–[9]. Although the stylization effect is not clear in the approach of Liu et al. (AdaAttN [2]), it still produces an intense light blue line around the tree's edge. For the method that applies the stylization as part of the rendering process [12], the halo effect is reduced but it is still visible. Our approach eliminates this. To more adequately visualize temporal stability comparisons, Figure 6 plots the heatmap of temporal error between the demonstrated frame and its previous frame. Our method is closer in appearance to the input temporal error heatmap. More qualitative results are included in the supplementary material.

E. *Quantitative Results*

We use a range of different computational metrics to gauge the performance of our approach quantitatively. From the four open-source games used to create the training dataset, a test set is also compiled composed of 2100 frames (9 gameplays $\times$ 200 frames and 3 gameplays $\times$ 100 frames). The same dataset was used in the evaluation of [12]. The gameplays encompass a wide range of environment characteristics, complicated lighting scenes and moving objects. Evaluation is performed using 10 different styles and results are reported in Table I.

Warping error and Learnt Perceptual Image Patch Similarity (LPIPS) Error [63] are used to measure temporal coherence performance. Warping error is computed as the difference between a warped next frame (using optic flow) and the original next frame. Optical flow information of the original gameplay videos is computed using *FlowNetS* [64]. The LPIPS metric is calculated as the average perceptual distances between consecutive frames, which is an additional measure of the stability of the generated sequential stylizations. As depicted in Table I, our method significantly outperforms state-of-the-art methods and produces the most consistent stylizations of the input game sequences.

To measure content preservation and perceptual similarity to the original input frames, we employ the LPIPS metric [63], SSIM [65], and the content loss ($\mathcal{L}_c$) [1] with a pre-trained VGG-16 network [59]. Style loss ($\mathcal{L}_s$) [1] is also used to assess style fidelity, along with the SIFID metric [66]. In addition, the *MiDaS* [60] depth estimation network is used to measure the difference in depth maps between the input and stylized frames. The Depth Error gives an approximation of how well depth is retained. Our system performs competently in perceptual and style performance metrics. When embedding the trained stylization network as part of the rendering process, some of the perceptual information is lost, as post-process effects are applied over. Our method applied as a post-effect (image) outperforms state-of-the-art approaches in some metrics and it illustrates the trade-off between temporal stability and content-style performance degradation. We

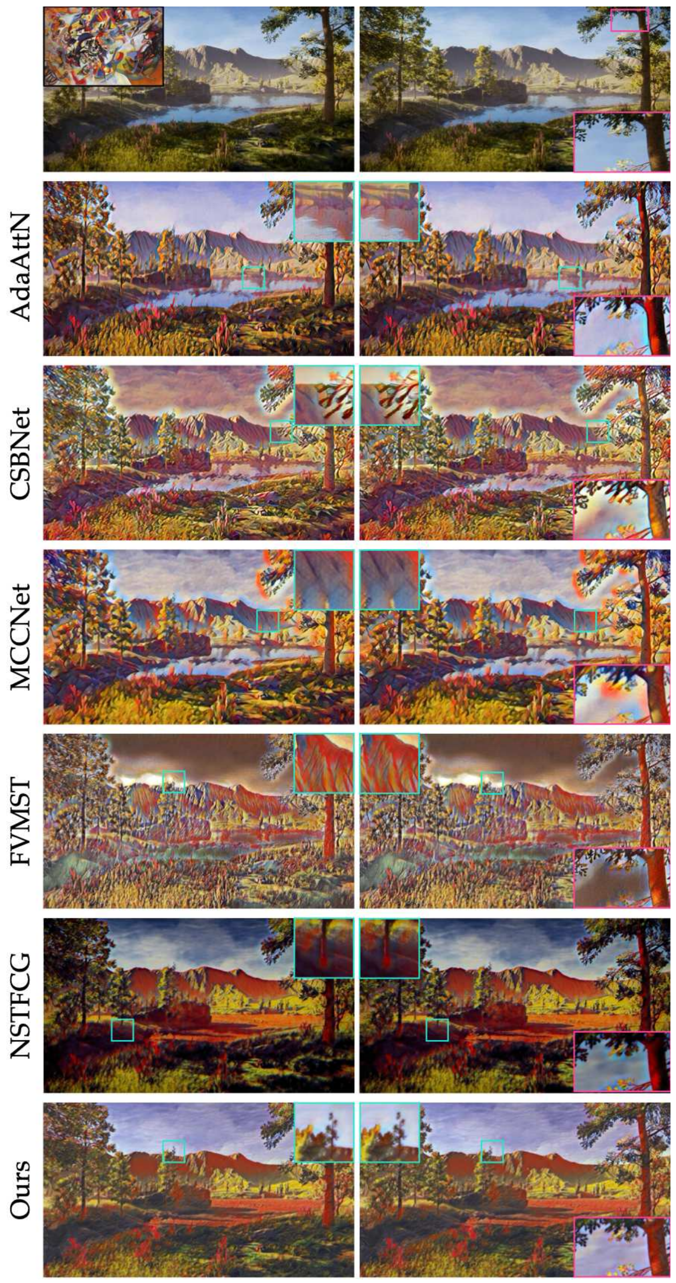

Fig. 5. Comparison against state-of-the-art methods. The first row shows the original input consecutive frames with the style image provided at the top-left. Zoomed-in cut-outs illustrate the effectiveness of our method in preserving fine details, avoiding undesired artefacts, and eliminating the halo effect around objects.

tailor our in-game method for optimized temporal coherence which is desirable in a game context. As seen in Table I, our algorithm's performance is comparable to state-of-the-art methods while it surpasses most of the approaches in content and depth preservation. This is achieved due to the geometry and depth information fused into the stylization network and also due to the depth reconstruction loss optimized during training. It is important to notice that although AdaAttN [2] performs favorably in perceptual metrics, it does not produce stylizations with sufficient style intensity (Figure 5).

F. Ablation Study

1) Synthetic Dataset: From Table I, the best-performing method, after our approach, in terms of temporal coherence that is not game-specific is the method of Liu et al. [2]. To assess the effectiveness of the utilization of the created dataset during training, we fine-tune the trained *AdaAttN* model using the synthetic game-specific dataset (Section III-A) for 2 epochs. The performance of the *AdaAttN* is significantly degraded. Our stylization network's implementation is straightforward, yet it can capture the synthetic nature of the dataset and generate effective stylizations. Table II captures the performance of the *AdaAttN* model [2] trained on the synthetic dataset; for reference, the performance of the original *AdaAttN* model is also included.

2) G-buffer Fusion: The consideration of G-buffer information during training has been shown to improve the quality of generated imagery [10], [11]. To illustrate the effectiveness of our G-buffer Encoder network and its utilization for the injection of geometry and structure-related features in the stylization network, we train the stylization network on the generated synthetic dataset but without G-buffer fusion. Figure 7 demonstrates the degraded performance of these models. It is noticeable that when using G-buffer fusion, the objects' boundaries are respected and the halo effect around objects is reduced. As shown in the third row of Figure 7, the lighting and geometry of the input scene are captured and preserved more effectively than without this component.

3) G-buffer-Guided In-Game Stylization: To assess the effectiveness of the proposed Compute Shader (Figure 3), we remove the shader from the implementation of the in-game stylization framework. Stylization is still injected in the pipeline before the post-process stage, but G-buffer information such as depth, and motion vectors are not considered during inference. We evaluate the results of the modified framework on a random scene of 200 frames and 10 different styles. Table III shows that the warping error performance decreases, as well as the efficiency in preserving the content information of the input frame (LPIPS is calculated between the original input frames and the corresponding stylized frames). The implemented shader considers the color of the pixels of the previous stylized frame before calculating the final frame that is passed through the post-process stage. It also offers a controllable way to preserve structural and depth information. This is illustrated in Figure 8; depth is better retained when the shader is not removed. Removing the shader that performs G-buffer-Guided stylization from the pipeline, the system is similar to the approach described in [12]; nevertheless this has a significant impact on the performance.

G. Limitations

Our in-game G-buffer-guided stylization shader implementation is suitable for any style transfer model applied in a game setting. The developed shader is efficient in improving the temporal stability of the stylized game frames by intercepting stylized frames alongside G-buffer information. Experiments have shown its effectiveness that comes without any time constraints as it does not introduce further delays in the

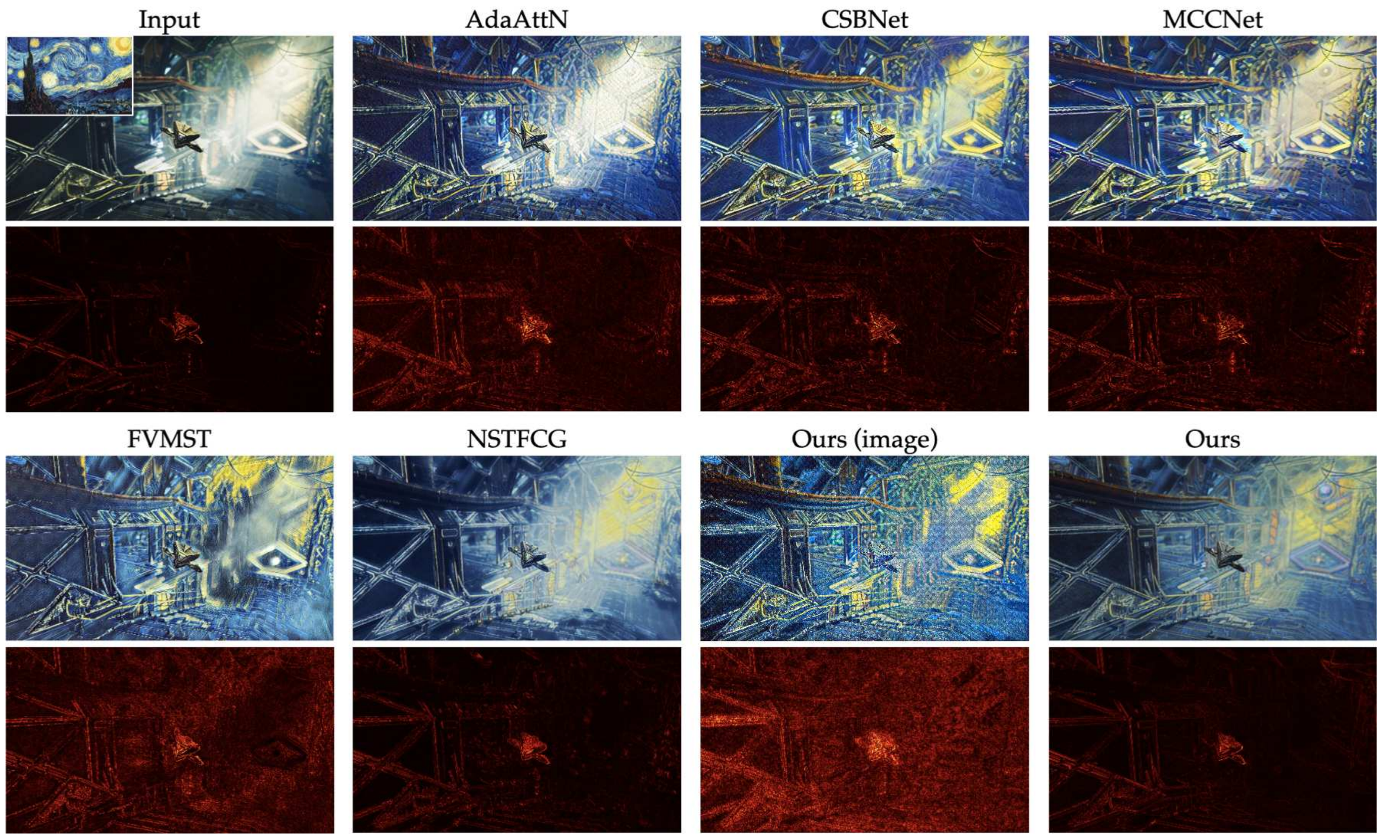

Fig. 6. Visual comparisons with state-of-the-art methods. The bottom rows show heatmap of temporal error between the current and previous frame. Our approach produces appealing stylizations while the temporal error heatmap is the closest to the input's heatmap.

TABLE I
QUANTITATIVE RESULTS. WARPING ERROR AND LPIPS ERROR ARE BOTH IN THE FORM $\times 10$. SSIM AND $\mathcal{L}_c$ RELATE TO CONTENT PRESERVATION, AND SIFID AND $\mathcal{L}_s$ QUANTIFY THE STYLE PERFORMANCE. DEPTH ERROR MEASURES THE RETAINMENT OF DEPTH IN THE STYLIZED FRAMES. WE PROVIDE RESULTS FOR OUR SYSTEM INJECTED IN THE PIPELINE PERFORMING G-BUFFER-GUIDED STYLIZATION (IN-GAME) AND FOR THE TRAINED STYLIZATION NETWORK APPLIED AS A POST-PROCESS EFFECT (IMAGE).

Method	Warping Error ↓	LPIPS Error ↓	SSIM ↑	LPIPS ↓	SIFID ↓	$\mathcal{L}_c$ ↓	$\mathcal{L}_s$ ↓	Depth Error ↓
AdaAttN [2]	1.6477	0.3217	**0.7820**	**0.2692**	1.6115	**0.4945**	1.0391	**5.1716**
CSBNet [9]	1.7458	0.3908	0.6370	0.3378	2.2468	0.8674	1.0053	11.6661
MCCNet [8]	1.6519	0.3547	0.6637	0.3468	1.5555	0.8065	1.0042	13.6036
FVMST [7]	1.8524	0.3215	0.5855	0.3806	2.2529	0.7834	1.0077	19.1459
NSTFCG [12]	1.5798	0.2930	0.6057	0.3879	1.8679	0.7830	1.0612	13.9274
Ours (image)	1.6039	0.3611	0.7107	0.3132	**1.0973**	0.6205	**0.9796**	8.13575
Ours (in-game)	**1.2984**	**0.2515**	0.5914	0.3494	1.9401	0.7514	1.0674	8.8524

TABLE II
AdaAttN [2] PERFORMANCE WHEN TRAINED ON THE SYNTHETIC DATASET IN COMPARISON WITH THE ORIGINAL TRAINED *AdaAttN* MODEL.

Method	Warping Er. ↓	LPIPS Er. ↓	LPIPS ↓
AdaAttN [2]			
Original	1.6477	0.3217	0.2692
+ Synthetic	1.7742	0.5322	0.3301

TABLE III
ABLATION STUDY ON THE EFFECT OF THE SHADER DURING INFERENCE. RESULTS ARE AVERAGED FOR A RANDOM SCENE OF 200 FRAMES AND FOR 10 DIFFERENT STYLE IMAGES.

Configuration	Warping Error ↓	LPIPS ↓
w/ Shader	0.1578	0.3315
w/o Shader	0.1803	0.3652

rendering pipeline. In Unity HDRP, a game that is stylized using our approach in Full HD resolution has a frame rate of ∼10fps, similar to previous work on computer games [12]. To improve upon the game's frame rate, future work could consider model compression approaches [67], [68] to reduce the complexity and memory size of the stylization network or explore the utilization of temporal upsampling [47].

In addition, for this work, we have trained a model that captures intercepted G-buffer features, yet is capable of reproducing one style image. As such methods can offer unlimited possibilities within the game community, arbitrary-style-per-models can be explored. Creating more diverse datasets that

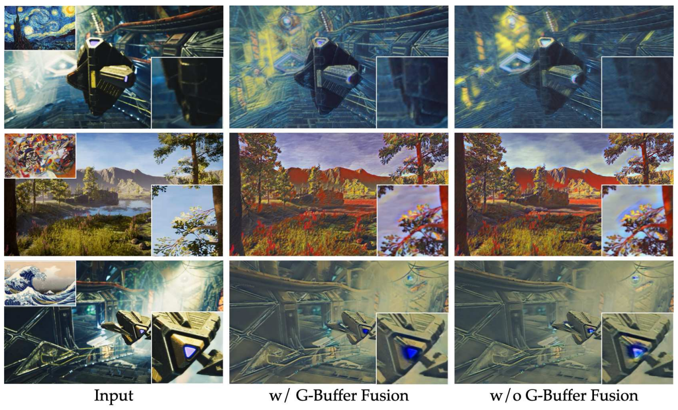

Fig. 7. Ablation study on the effect of G-buffer feature fusion in the Stylization network. Learning G-buffer features and fusing this knowledge in the stylization network has a visible benefit to the resulting stylizations.

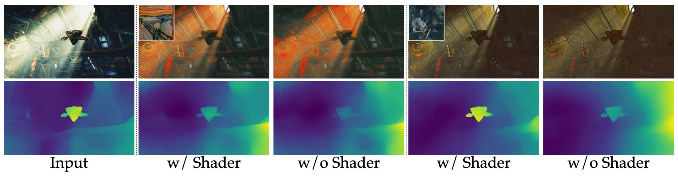

Fig. 8. Ablation Study on the effect of the implemented shader during stylization. Results are shown for two different styles. Utilizing the shader during inference results in higher quality and better preservation of depth information.

encompass a wider variety of game genres would also enhance the effectiveness of the algorithm and ensure broader applicability. This would allow users to upload any artwork to be immersed in any game stylized according to their personal preferences.

V. Conclusion

In this work, we study the problem of artistic style transfer for computer games. We have proposed a G-buffer-guided stylization framework capable of significantly improving the temporal consistency of stylized game scenes compared to state-of-the-art methods. Our stylization network is trained on a synthetic dataset that includes depth and normal data. Learnt G-buffer features are injected into the stylization network that becomes more aware of the geometric and semantic aspects of the game scene. The trained network is injected into the 3D rendering pipeline avoiding diminished post-process effects. A novel algorithm is developed that utilizes G-buffer information during inference time to further improve temporal coherence. Our shader implementation also offers a controllable way to stylize computer games in terms of tuning the trade-off between temporal stability and content preservation. Qualitative and quantitative experiments have shown that our method achieves significantly higher temporal stability with comparable perceptual and stylization performance when compared with state-of-the-art approaches.

Acknowledgments

This research was funded by the EPSRC. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2414–2423.

[2] S. Liu, T. Lin, D. He, F. Li, M. Wang, X. Li, Z. Sun, Q. Li, and E. Ding, "Adaattn: Revisit attention mechanism in arbitrary neural style transfer," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 6649–6658.

[3] Y. Deng, F. Tang, W. Dong, C. Ma, X. Pan, L. Wang, and C. Xu, "Stytr2: Image style transfer with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 326–11 336.

[4] S. Huang, J. An, D. Wei, J. Luo, and H. Pfister, "Quantart: Quantizing image style transfer towards high visual fidelity," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 5947–5956.

[5] W. Xu, C. Long, and Y. Nie, "Learning dynamic style kernels for artistic style transfer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 10 083–10 092.

[6] G. Ghiasi, H. Lee, M. Kudlur, V. Dumoulin, and J. Shlens, "Exploring the structure of a real-time, arbitrary neural artistic stylization network," *arXiv preprint arXiv:1705.06830*, 2017.

[7] W. Gao, Y. Li, Y. Yin, and M.-H. Yang, "Fast video multi-style transfer," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2020, pp. 3222–3230.

[8] Y. Deng, F. Tang, W. Dong, H. Huang, C. Ma, and C. Xu, "Arbitrary video style transfer via multi-channel correlation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, pp. 1210–1217, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16208

[9] H. Lu and Z. Wang, "Universal video style transfer via crystallization, separation, and blending," in *Proc. Int. Joint Conf. on Artif. Intell.(IJCAI)*, vol. 36, 2022, pp. 4957–4965.

[10] S. R. Richter, H. A. AlHaija, and V. Koltun, "Enhancing photorealism enhancement," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 2, pp. 1700–1715, 2022.

[11] M. Mittermueller, Z. Ye, and H. Hlavacs, "EST-GAN: Enhancing style transfer gans with intermediate game render passes," in *2022 IEEE Conference on Games (CoG)*, 2022, pp. 25–32.

[12] E. Ioannou and S. Maddock, "Neural style transfer for computer games," in *British Machine Vision Conference 2023, BMVC 2023*, 2023.

[13] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[14] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural style transfer: A review," *IEEE transactions on visualization and computer graphics*, 2019.

[15] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European conference on computer vision*. Springer, 2016, pp. 694–711.

[16] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images." in *ICML*, vol. 1, 2016, p. 4.

[17] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6924–6932.

[18] C. Li and M. Wand, "Precomputed real-time texture synthesis with markovian generative adversarial networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*. Springer, 2016, pp. 702–716.

[19] A. Sanakoyeu, D. Kotovenko, S. Lang, and B. Ommer, "A style-aware content loss for real-time hd style transfer," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 698–714.

[20] D. Kotovenko, A. Sanakoyeu, S. Lang, and B. Ommer, "Content and style disentanglement for artistic style transfer," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 4422–4431.

[21] X.-C. Liu, M.-M. Cheng, Y.-K. Lai, and P. L. Rosin, "Depth-aware neural style transfer," in *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, 2017, pp. 1–10.

[22] M.-M. Cheng, X.-C. Liu, J. Wang, S.-P. Lu, Y.-K. Lai, and P. L. Rosin, "Structure-preserving neural style transfer," *IEEE Transactions on Image Processing*, vol. 29, pp. 909–920, 2019.

[23] E. Ioannou and S. Maddock, "Depth-aware neural style transfer using instance normalization," in *Computer Graphics & Visual Computing (CGVC) 2022*. Eurographics Digital Library, 2022.

[24] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1501–1510.

[25] S. Gu, C. Chen, J. Liao, and L. Yuan, "Arbitrary style transfer with deep feature reshuffle," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8222–8231.

[26] F. Shen, S. Yan, and G. Zeng, "Neural style transfer via meta networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8061–8069.

[27] J. Svoboda, A. Anoosheh, C. Osendorfer, and J. Masci, "Two-stage peer-regularized feature recombination for arbitrary image style transfer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 816–13 825.

[28] J. An, S. Huang, Y. Song, D. Dou, W. Liu, and J. Luo, "Artflow: Unbiased image style transfer via reversible neural flows," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 862–871.

[29] X.-C. Liu, Y.-L. Yang, and P. Hall, "Learning to warp for style transfer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3702–3711.

[30] J. Huo, S. Jin, W. Li, J. Wu, Y.-K. Lai, Y. Shi, and Y. Gao, "Manifold alignment for semantically aligned style transfer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 14 861–14 869.

[31] H. Tang, S. Liu, T. Lin, S. Huang, F. Li, D. He, and X. Wang, "Master: Meta style transformer for controllable zero-shot and few-shot artistic style transfer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 18 329–18 338.

[32] K. Hong, S. Jeon, J. Lee, N. Ahn, K. Kim, P. Lee, D. Kim, Y. Uh, and H. Byun, "AesPA-Net: Aesthetic pattern-aware style transfer networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 22 758–22 767.

[33] B. Gu, H. Fan, and L. Zhang, "Two birds, one stone: A unified framework for joint learning of image and video style transfers," *arXiv preprint arXiv:2304.11335*, 2023.

[34] Y. Ma, C. Zhao, X. Li, and A. Basu, "RAST: Restorable arbitrary style transfer via multi-restoration," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 331–340.

[35] M. Ruder, A. Dosovitskiy, and T. Brox, "Artistic style transfer for videos," in *Pattern Recognition*, B. Rosenhahn and B. Andres, Eds. Cham: Springer International Publishing, 2016, pp. 26–36.

[36] ——, "Artistic style transfer for videos and spherical images," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1708.04538

[37] J. Fišer, O. Jamriška, D. Simons, E. Shechtman, J. Lu, P. Asente, M. Lukáč, and D. Sỳkora, "Example-based synthesis of stylized facial animations," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–11, 2017.

[38] O. Jamriška, Šárka Sochorová, O. Texler, M. Lukáč, J. Fišer, J. Lu, E. Shechtman, and D. Sýkora, "Stylizing video by example," *ACM Transactions on Graphics*, vol. 38, no. 4, 2019.

[39] H. Huang, H. Wang, W. Luo, L. Ma, W. Jiang, X. Zhu, Z. Li, and W. Liu, "Real-time neural style transfer for videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 783–791.

[40] C. Gao, D. Gu, F. Zhang, and Y. Yu, "ReCoNet: Real-time coherent video style transfer network," in *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part VI 14*. Springer, 2019, pp. 637–653.

[41] S. Liu and T. Zhu, "Structure-guided arbitrary style transfer for artistic image and video," *IEEE Transactions on Multimedia*, 2021.

[42] E. Ioannou and S. Maddock, "Depth-aware neural style transfer for videos," *Computers*, vol. 12, no. 4, p. 69, 2023.

[43] W. Wang, S. Yang, J. Xu, and J. Liu, "Consistent video style transfer via relaxation and regularization," *IEEE Transactions on Image Processing*, vol. 29, pp. 9125–9139, 2020.

[44] X. Luo, Z. Han, L. Yang, and L. Zhang, "Consistent style transfer," *arXiv preprint arXiv:2201.02233*, 2022.

[45] Z. Wu, Z. Zhu, J. Du, and X. Bai, "CCPL: Contrastive coherence preserving loss for versatile style transfer," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVI*. Springer, 2022, pp. 189–206.

[46] X. Li, S. Liu, J. Kautz, and M.-H. Yang, "Learning linear transformations for fast image and video style transfer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3809–3817.

[47] T. Deliot, F. Guinier, and K. Vanhoey, "Real-time style transfer in unity using deep neural networks," 2020. [Online]. Available: https://blog.unity.com/engine-platform/real-time-style-transfer-in-unity-using-deep-neural-networks

[48] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*. Springer, 2016, pp. 102–118.

[49] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *European Conf. on Computer Vision (ECCV)*, ser. Part IV, LNCS 7577, A. Fitzgibbon et al. (Eds.), Ed. Springer-Verlag, Oct. 2012, pp. 611–625.

[50] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[51] Unity Technologies, "Unity Terrain - HDRP demo scene," 2022. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/unity-terrain-hdrp-demo-scene-213198

[52] ——, "Fontainebleau Demo," 2022. [Online]. Available: https://github.com/Unity-Technologies/FontainebleauDemo

[53] POLYGONAUTIC, "Seed hunter," 2020. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/seed-hunter-143414

[54] Unity Technologies, "Book of the dead: Environment: HDRP: Tutorial projects," 2023. [Online]. Available: https://assetstore.unity.com/packages/essentials/tutorial-projects/book-of-the-dead-environment-hdrp-121175

[55] ——, "Unity Perception package," 2020. [Online]. Available: https://github.com/Unity-Technologies/com.unity.perception

[56] ONNX, "Open neural network exchange," 2019. [Online]. Available: https://onnx.ai/

[57] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[58] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of solid-state circuits*, vol. 23, no. 2, pp. 358–367, 1988.

[59] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[60] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[61] Unity Technologies, "High definition Render Pipeline: 12.1.12," 2021. [Online]. Available: https://docs.unity.cn/Packages/com.unity.render-pipelines.high-definition@12.1/manual/index.html

[62] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[63] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 586–595.

[64] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462–2470.

[65] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[66] T. R. Shaham, T. Dekel, and T. Michaeli, "SinGAN: Learning a generative model from a single natural image," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 4570–4580.

[67] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[68] H. Wang, Y. Li, Y. Wang, H. Hu, and M.-H. Yang, "Collaborative distillation for ultra-resolution universal style transfer," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1860–1869.