Expert Systems With Applications 211 (2023) 118491

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Game effect sprite generation with minimal data via conditional GAN

JaeWon Kim [a], KyoHoon Jin [a], SooJin Jang [a], ShinJin Kang [b,*], YoungBin Kim [a,*]

[a] Department of Image Science and Arts, Chung-Ang University, Dongjak, Seoul 06974, South Korea
[b] School of Games, Hongik University, Sejong-si 30016, South Korea

ARTICLE INFO

Keywords:
Deep learning
Conditional GAN
Game effect
Game sprite
Generative adversarial networks
Style transfer

ABSTRACT

Image generation using convolutional neural networks has gained popularity in various industries in the field of computer vision over the last decade. However, thus far, generative adversarial networks (GAN) have not been utilized to their maximum potential to generate computer game sprites, especially game effects. This is because typically the amount of available open-source game sprite data is considerably less than that of other problem domains such as image classification or object detection tasks, which use hundreds of thousands of images to train classification models. This study demonstrates that GAN can be utilized to generate game effect sprites adequately with a small set of input data, thus increasing the productivity of large-scale 2D image modification work in game development. In this study, we propose a simple 2D game effect sprite generation technique called a Game Effect Sprite Generative Adversarial Network (GESGAN). The proposed model generates a new style-translated image based on the structure of an effect image and a style label. The model consists of an encoder designed to extract the feature representation of an input image and a label; a decoder to generate a synthetic image; and a discriminator that learns to determine the authenticity of images with an auxiliary classifier for the label. Experimental results show that GESGAN is capable of reliably generating style-translated images for various shapes of object images and drawing styles and can perform 2D image sprite generation and modification tasks in near real-time, thereby reducing game development costs.

1. Introduction

Optimizing artistic creation and its associated processes is an important goal of media companies, especially those focusing on games. Companies invest considerable effort to improve the artistic creation process over time by adopting various software design tool applications. However, the development of 2D game art resources and sprites involves many resources, as designers manually modify graphics, pixels, and other media. 2D sprite generation techniques are used in 2D games for graphics resource development, generally as a method of expressing motion through animations comprising many sequential images. Game sprites are conventionally created through a generation process performed by a game designer. For 2D games, sprites are usually produced according to the desired direction and frame rate, typically 10 to 30 or more consecutive images per second. For this, motion expression, color correction, and color palette designation are required for each pixel, which increases development costs. Game designers commonly process all pixels manually within a specified resolution. Games require generating a variety of 2D sprites ranging from dozens to hundreds, which imposed a heavy burden on game designers (Serpa & Rodrigues, 2019). To efficiently develop game sprites and game themselves, procedural content generation via machine learning (Summerville et al., 2018) have been studied mainly in the fields of game map generation (Guzdial et al., 2018; Jain et al., 2016; Snodgrass & Ontañón, 2017), game character generation (Chelliah et al., 2019; Hong et al., 2019; Kim, 2017), and major object generation (Horsley & Liebana, 2017; Reed et al., 2015) in games. However, in the field of game effect, the research conducted remains insufficient.

The style of characters or objects in games changes frequently due to the nature of development processes and design changes in the size and shape of images requiring real-time processing. In addition, in traditional 2D sprite creation processes, designers manually perform tasks such as creation, evaluation, and modification, resulting in high operational costs (Aleem et al., 2016; Almeida & da Silva, 2013; Schell, 2008). To transform the style of game effect sprites, color, size, and texture can be easily transformed in a given image, as shown in Fig. 1; however, to transform an image effect naturally according to a given specific game condition, the shape and color of the sprite must be manually modified. Therefore, several studies (Hong et al., 2019; Horsley & Liebana, 2017; Serpa & Rodrigues, 2019) have been conducted to generate game sprites with GAN. These studies

* Corresponding authors.
E-mail addresses: mangocc@cau.ac.kr (J. Kim), fhzh123@cau.ac.kr (K. Jin), sujin0110@cau.ac.kr (S. Jang), directx@hongik.ac.kr (S. Kang), ybkim85@cau.ac.kr (Y. Kim).

https://doi.org/10.1016/j.eswa.2022.118491
Received 20 October 2021; Received in revised form 1 August 2022; Accepted 7 August 2022
Available online 17 August 2022

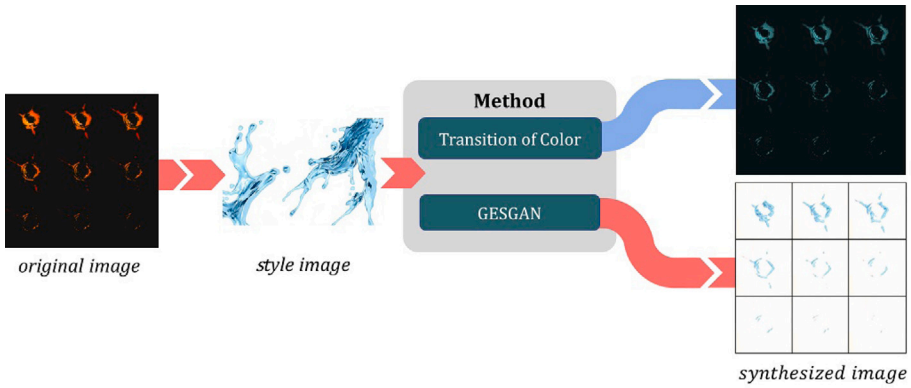

Fig. 1. An example of a game effect sprite transformation process. A result of color transition of an original image is shown above, whereas a synthesized image generated by GESGAN is shown below.

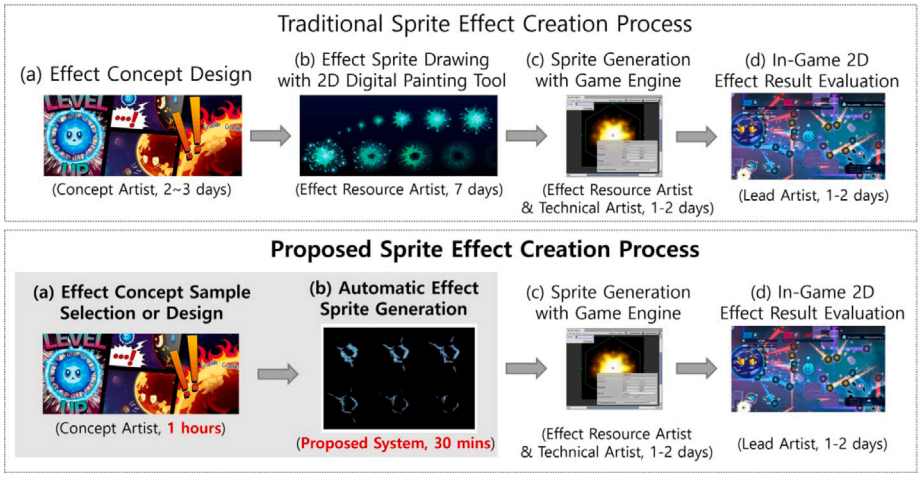

Fig. 2. Illustration of the game effect sprite creation process. Above is traditional sprite creation process, below is proposed sprite effect creation process.

have mainly focused on the colorization of characters or objects while prioritizing the preservation of the given structure mostly using their own game data. However, existing methods have limited applications in synthesizing game effects because the synthesis of effect styles is important and requires maintaining an adequate object structure. The scalability of these models is often limited to self-produced data. The rapid and simple creation of game effect sprites with intended styles and structures as a development process can significantly reduce the cost of game production. Nonetheless, relatively little research has been conducted on this topic, despite its efficiency and necessity.

To further illustrate the sprite production process, Fig. 2 shows the existing game effect production process and the effect production process when utilizing the proposed technique. The traditional sprite effect production method consists of four stages. The first step is shown in Fig. 2(a) as the concept design step. At this stage, the concept artist draws an initial original picture of the effect suitable for the game's visual concept. Then, the visual consistency is often confirmed by the lead artist. This step depends on the level of key animation, but for simple effects, it takes 2–3 days to produce. The next step is effect sprite production step, as shown in Fig. 2(b). In this stage, a 2D sprite creator usually draws a continuous sequence of images using digital painting tools such as Photoshop. This step may take up to a week of working days depending on the graphical intense of the sprite. Fig. 2(c) shows the sprite generation step on the game engine. In this stage, the series of images are loaded on game engines such as Unity and Unreal, which are used to produce real games, to generate a sprite(asset) used in real games. Finally, it leads to in-game effect evaluation step, shown in Fig. 2(d), which comprises of assessing the effects produced on the actual game engine. In details, Effect Sprite Drawing with 2D Digital Painting Tool, shown in Fig. 2(b), is usually performed in Photoshop. In Photoshop, a designer draws a sprite on one layer, and then draws several sheets to ensure inter-frame continuity. Typically, a game renders 30–60 frames per second. In 2D creation process, however, a designer draws approximately 5–15 frames per second because it requires high cost to manually create pixel art. Thus, a designer has to draw approximately 10 effects for each effect by hand or create particle, then retouch them. It is usually done in sketch (or particle creation), coloring, retouching, and check continuity process. Each step takes at least one day per step, and one week is required when these tasks are done perfectly in time. Depending on the situation of each team, this step is often repeated 2–3 times with feedback from an art director to improve completeness. Therefore, one effect sprite production goes through these steps sequentially and requires about two weeks per effect. If the final result does not meet the art director's evaluation criteria, it is re-produced from the beginning or modified at an intermediate step. The proposed process replaces steps shown in Fig. 2(a), (b). It quickly produces sprites that concept artists aim for at a low cost. Although additional post-processing may occur for images generated by GESGAN, the sprite generation period of more than one week that was previously consumed can be shortened to up to one day, and design change feedback can be quickly replaced. In general, the game production process requires the production of dozens to hundreds of effect sprites, and this development cost can be greatly reduced if the proposed tool is utilized.

In this study, we propose a method to improve the efficiency of 2D game effect sprite production processes by applying GAN-based style

transfer (image-to-image translation) with style labels without using design tools. Using labels is advantageous because when they are fed into the model, the mapping of a given condition becomes intuitively and easily applicable in training, and the model becomes relatively lightweight (Isola et al., 2017; Mirza & Osindero, 2014; Odena et al., 2017). Previous studies have shown that feeding labels to the models can generate conditioned MNIST digits (Mirza & Osindero, 2014), and using one-bit stroke encoding was able to preserve the key modes of Chinese characters (Zeng et al., 2021). Inspired by these studies, we adopted direct use of label information.

The proposed method is a supervised learning model which learns to generate a new 2D game effect sprite based on two conditions, including the target shape and label of the target style. Trained on a given structure of images through a target label, the model can obtain effects of creating a corresponding image for each label by mixing labels and images using convolutional neural networks (CNNs) in GAN. In order to evaluate the proposed method, we show experimental results compared to baseline models and further evaluate the visual quality of the generated images using BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator) (Mittal et al., 2012), PIQE (Perception based Image Quality Evaluator) (Venkatanath et al., 2015), and NIQE (Natural Image Quality Evaluator) (Mittal et al., 2013). Compared to the baseline models FastNeuralStyle (Johnson et al., 2016), Pix2pixcGAN (Isola et al., 2017), and TET-GAN (Yang et al., 2019a), on an average the proposed method scored 9.101 lower in BRISQUE and exhibited higher PIQE and NIQE scores of 11.7 and 8.53, respectively, generating better visual quality. Our model performed well in the corresponding evaluation indicators, which indicate how realistic the image is, allowing game designers to cut and paste game effect sprites generated through our model directly into the game.

- We introduce GESGAN, which can transform images according to multiple given styles using only minimal data. The experimental training dataset included a total of 16 images, with four images for each style.
- To the best of our knowledge, this study is the first to investigate the game effect sprites rather than the naive colorization of sprites.
- The GESGAN synthesizes the given structure image and label almost in real time. In reference, the generator in GESGAN result in approximately 8.93 GFLOPs (Giga Floating Point Operations Per second) and can generate images in 104.99 FPS on GTX RTX 3090.

2. Related works

2.1. Style-transfer with GAN on images

Generative adversarial networks (GAN) (Goodfellow et al., 2014) are a type of artificial intelligence algorithm which has been dominating the field of realistic image generation through unsupervised learning since 2014. Furthermore, additional approaches have been studied not only with unsupervised learning, but also with methods such as semi-supervised learning (Salimans et al., 2016), supervised learning (Isola et al., 2017), and reinforcement learning (Ho & Ermon, 2016). Among these studies on variations of GAN, areas such as style transfer, image inpainting (Nazeri et al., 2019; Ulyanov et al., 2018), conditional image generation (Isola et al., 2017; Mirza & Osindero, 2014), and face generation (Karras et al., 2018; Tolosana et al., 2020) have been widely studied. Style transfer began to be actively studied with the development of deep learning. Traditional style transfer methods were algorithms such as image analogies (Hertzmann et al., 2001) or image quilting (Efros & Freeman, 2001), which originated from the field of non-photorealistic rendering. NST, a deep learning-based style transfer model, was proposed by Gatys (Gatys et al., 2015). Gatys' work in 2015 provides a histogram-based texture synthesis algorithm that identifies the features of images using a VGGNet (Simonyan & Zisserman, 2015) structure, which utilized a CNN for image analogy problems and synthesized textures based on histograms of an input image. Google AI (Gatys et al., 2015) proposed a method for a single style transfer network based on a CNN to train multiple styles simultaneously. Subsequently, additional studies were conducted using feedforward neural networks for rapid tile transformation (Li et al., 2017a, 2017b; Wang et al., 2017) and other studies used local patches of feature maps (Li & Wand, 2016a, 2016b) or semantic segmentation of objects for more realistic style transformation (Murez et al., 2018).

Generally, style transfer methods have difficulty generating data or preserving the structure of original data. Therefore, style transfer methods based on GAN attempt to provide a flexible method of guidance and unsupervised learning by proposing cycle consistency loss (Karras et al., 2020), such as CycleGAN (Choi et al., 2018), and StyleGAN (Karras et al., 2019), or by utilizing or learning a mapping between different image domains. However, further studies are required on extending such models to new domains. Therefore, the proposed method in this study uses conditional generative adversarial networks, which either generate synthetic images by feeding conditions to the model or transform them into a realistic image based on data and conditions (Chen et al., 2016; Mirza & Osindero, 2014; Odena et al., 2017). While the conditional image generation idea has been successfully applied to super-resolution imaging (Ledig et al., 2017) and photo editing (Brock et al., 2017; Shu et al., 2017) and it has also shown excellent performance in image domain (style) transformation (Choi et al., 2018; Kim et al., 2017; Taigman et al., 2017).

2.2. Text style-transfer

Various style transfer GAN models have been developed. However, as game effect sprite transformation requires minimal loss of the original structure, even if the structure of the original image cannot be preserved perfectly, sufficient style synthesis is still required. A text style transfer model capable of producing these results can produce results which are similar and effective to the purpose of game effect sprite transformation. Style transfer methods such as the statistical approach used by Yang (Yang et al., 2017), which focuses on style rather than font transformation has been conducted. Additional studies were conducted focusing on the development of deep learning methods such as MC-GAN (Azadi et al., 2018), TET-GAN (Ho & Ermon, 2016), and UT Effect (Yang et al., 2019b) and others were studied for glyph transformation. In addition, recent works have explored modifying the strength and amount of texture styles (Gatys et al., 2016) by utilizing hyperparameters to balance content loss and style loss (Yang et al., 2019). ShapeMatchingGAN (Yang et al., 2019) generates images in three stages, include a sketch module, structure transfer model, and texture transfer model. To control the amount of texture, the method applied a controllable ResBlock to structure the transfer to learn a texture transfer model from the results learned by the structure transfer model. Although this model showed reasonable performance, characteristically the structural transfer model and the texture transfer model can synthesize only one style each and are trained based on a total of three models.

2.3. Generating game sprites

The transformation of styles based on GAN has been actively studied in specific areas such as comics (Chen et al., 2018; Seo & Seo, 2021), painting (Odena et al., 2017), makeup (Chang et al., 2018), etc. With this trend, there have also been attempts to introduce GAN-based techniques in the game development field for sprite generation. Typically, MDGAN (Hong et al., 2019) studied transforming characters into target styles by referring independently to colors, shapes, and animators, with two encoders in a GAN generator and two discriminators which distinguish the generated outputs. In addition, a study was

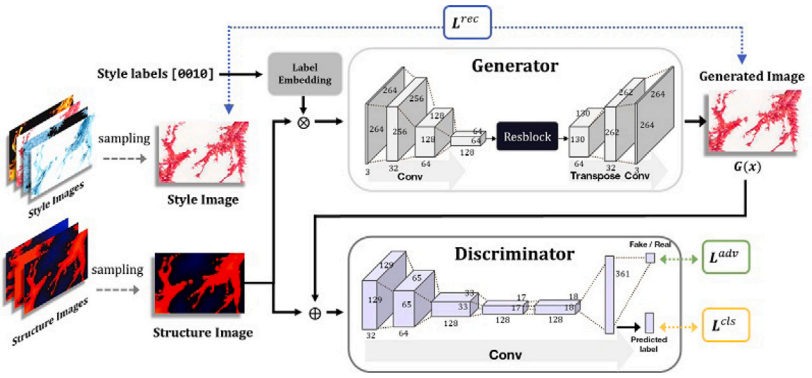

Fig. 3. Model structure of GESGAN in training. Black line refers to the transmission of data; dotted lines refer to the loss functions within GESGAN. ⊗ denotes element-wise multiplication of the structure image and the style label. And ⊕ indicates the concatenation of the generated images with the ground truth style image.

conducted to color the characters variedly during the sketch stage of the character sprite (Seo & Seo, 2021). The modification and development of characters or objects by applying effects often involves high costs, yet the generation of game effect sprites is not studied actively. This study proposes a GAN that generates 2D game effect sprites based on two conditions: a given specific structure image and target style. GESGAN learns various styles through one model to generate more realistic images and can be used immediately in game development. The proposed generator of GESGAN utilizes a modified form of DCGAN (Radford et al., 2016), which uses a CNN. PatchGAN Discriminator (Isola et al., 2017), as it exhibits relatively small computational costs and decent performance for large images, is a method to distinguish whether images are created in patches. This way, learning strong feature representations of shape images is possible and the generator effectively generates style-transferred images.

3. Proposed method

In this study, we propose a Game Effect Sprite GAN (GESGAN) that effectively creates a given image to the desired style based on the GAN. The proposed method consists of two main structures. First, it is divided into the generator which produces the image and the discriminator which determines whether the generated images are genuine. The input images include a structure image and an original image which is used as a style image. These original images are transformed into each style to train the multiple target style images. The loss function of the training GAN consists of reconstruction, adversarial, and classification loss. Unlike the existing GAN methods, it is possible to apply multiple styles using our single model. Moreover, the generator and discriminator were modified to stabilize learning and performance improvement. The overall model structure is shown in Fig. 3.

3.1. Generator

First, we assume that x and y represent the structure image and the style image, respectively. Style image includes original images and style transferred images. In this study, the models learn the mapping between different styles with a single generator G based on a one-hot encoded style label c corresponding to a style image y, given an input image x, to output a synthesized image $\hat{y}$. This problem is expressed as follows:

$$\hat{y}_c = G(x, c). \tag{1}$$

The style label c is embedded in vector dimensions of the same size as the input structure image size and fed into the model by multiplying the input image element-wise.

When training to transform a structure image into a particular style through the generator, the model learns a one-to-one mapping form that utilizes one ground truth for one structure image. However, although there is only a single structure image corresponding to each style image, the style is a multi-to-one mapping form with all styles intermixed according to each structure. Therefore, the network should not simply target the target style image y, but rather ensure that the generator does not fall into a mode collapsing that always produces the same output during the training process.

To address this problem, we designed a generator using three techniques, including target-style labeling, data augmentation, and ResBlock (He et al., 2016). First, the labels for styles, rather than the style images themselves as a condition, were fed to train the mapping of labels for each style. The addition of these conditional labeling to standard GAN structures yields excellent results and stabilizes the training process. Also, it allows the model to learn the representations of each structure image x through the loss functions that confronts the discriminator D, regardless of the label (Odena et al., 2017). This objective function applies only to the generator and is expressed as below.

$$\mathbb{E}_{x,y,c}[\log(1 - D(x, G(x, c)))]. \tag{2}$$

Second, to train the generator, we extract the structure image x, a randomly corresponding style image y, and the label c corresponding to y. We then randomly crop x and y into equal pieces and construct the training data with x, y of the sub-image pair. The input x is modified by adding Gaussian noise with a normal distribution to solve the overfitting problem so the model is not limited to the training data. Third, we adopted a method utilizing the encoder–decoder structure of the generator of FastNeuralStyle (Johnson et al., 2016). This is a prominent modified DCGAN (Radford et al., 2016) structure, which shows satisfactory performance in a short period of training time. We added ResBlock to the generator, which helps solve vanishing and exploding gradient problems that occur as the network depth deepens (He et al., 2016). Through these techniques, the generator of GESGAN is designed to show competent performance for a given style label in near real-time.

3.2. Discriminator

GAN learns through a confrontation between the generator and the discriminator, both of which learn to overcome the other. However, in practice, only the trained generator part of the GAN is used in the testing or implementation stage in general. In the network training phase, the generator utilizes input data to generate images that are as genuine as possible. In contrast, the discriminator learns to distinguish between generated images from the generator and the ground truth images. Thus, like the generator, the better the discriminator, the more realistic the images is generated. Therefore, the discriminator also requires a network that can effectively extract and distinguish feature representations. We adopted PatchGAN (Isola et al., 2017) as a classifier in GESGAN. PatchGAN refers to the technique of using a

CNN for the discriminator. The authenticity of the image is determined by patches of a certain size, i.e., the receptive field. The patch size is a hyperparameter as it must include the appropriate associated range between pixels across the image depending on the task. PatchGAN is applied effectively because the generator tends to exaggerate some features of the data to deceive the discriminator that determines the authenticity of the image, without considering the quality of the image, based on local features of the image.

GESGAN generates an image by reflecting the style label c on the structure image x. Thus, GESGAN's discriminator also plays a role in identifying images and predicting style labels. The goal of this model is to map multiple labels with a single discriminator. To this end, we added auxiliary classifiers such as ACGAN (Odena et al., 2017) and StarGAN (Choi et al., 2018). In the training phase, images are synthesized by randomly extracting the style label c including the style of the original.

3.3. Loss functions of GESGAN

The loss function of GESGAN consists of adversarial, style classification, and reconstruction loss.

3.3.1. Adversarial loss

To make the generated images indistinguishable from the real images, we utilized a condition-based adversarial loss function in which the generator produces images $G(x, c)$ conditioned on both the input image x and the target style label c, and then the discriminator tries to distinguish between the real image y and the synthesized image $\hat{y}$.

$$\mathcal{L}_{adv} = \mathbb{E}_{x,y,c}[\log(D(x, y))] + \mathbb{E}_{x,y,c}[\log(1 - D(x, G(x, c)))]. \tag{3}$$

The adversarial loss function of GESGAN, as shown in Eq. (3), attempts to minimize this goal, whereas the discriminator aims to maximize it to deceive the generator.

3.3.2. Style classification loss

The goal of GESGAN is to properly transform image x into the target style label, as in the actual target style image y, for a given input image x and target style label c. To achieve these conditions, an auxiliary classifier is added within the discriminator and the style classification loss was designed for optimization of discriminator and generator. That is, it performs two functions, including loss of style classification of real images used to train the discriminator and loss of style classification of fake images used to train the generator. Specifically, the former, D_{cls}, is defined as in Eq. (4) as the probability distribution for the style label calculated by the discriminator.

$$\mathcal{L}^{D}_{cls} = \mathbb{E}_{x,y,c'}[\log(D_{cls}(c'|x, y))], \tag{4}$$

by minimizing this loss function, the discriminator D learns to classify the real image x into the original style label c'. In contrast, the latter loss function for style classification of the synthesized fake image is defined by Eq. (5).

$$\mathcal{L}^{G}_{cls} = \mathbb{E}_{x,y,c}[\log(D_{cls}(c|G(x, c)))]. \tag{5}$$

Thus, the generator tries to minimize this goal to generate an image that can be classified as the target style c.

3.3.3. Reconstruction loss

The generator is trained to generate images classified as realistic and accurate style classes by minimizing the adversarial loss and the style classification loss. However, minimizing the losses (Eqs. (3), and (5)) does not guarantee that the transformed image preserves the structure of the input image object while only changing the style-related portion of the input. To solve this, the generator receives the converted image $G(x, c)$ and the actual style c of structure image x as input and applies a consistency loss (Kim et al., 2017; Zhu et al., 2017) as in Eq. (6) to reconstruct the original structure image x. We used L1 norm regularization for the reconstruction loss. We optimized the generator by designing a reconstruction loss function to obtain the difference from the ground truth image by training. This function trains through one generator twice, subsequently synthesizes the original structure image x into the target style c and the original style c' once each.

$$\mathcal{L}_{rec} = \mathbb{E}_{x,y,c}[\|y - G(x, c)\|_1] \tag{6}$$

3.3.4. GESGAN's objective function

The objective function of the final GESGAN can be defined as Eq. (7), where the λ value is a hyperparameter. The model learns the style transform problem in the minimax game, where the generator is optimized to reduce losses and the discriminator is optimized to increase losses.

$$\min_G \max_D \lambda_{adv}\mathcal{L}_{adv} + \lambda_{rec}\mathcal{L}_{rec} + \lambda_{cls}\mathcal{L}_{cls} \tag{7}$$

4. Experiments

4.1. Experimental setup

We used four images, one each of fire, water, smoke, and maple (tree/leaves) images for training the multi-label style transfer game sprite generation model. These images were provided by Yang et al. (2019), and were used as structure images and the ground truth style images. Then, each style was transformed into different styles using ShapeMatchingGAN (Yang et al., 2019). The image dataset for training with four style labels is elaborated in Fig. 4. In other words, the structure image dataset X consists of four original images and the target style image dataset Y consists of a total of 16 images including the original images and 12 images converted by each style. Typically, a large dataset is required to train models. However, GESGAN's training utilized data augmentation techniques, as mentioned above, to randomly crop the same (x, y) pair of images into a specific size and obtain a sufficient amount for training. Furthermore, we resized each cropped input to 256×256 and added Gaussian noise to the structure image x as a preprocessing set, which enables images to be synthesized into a wider variety of outcomes and prevents over-fitting. The game effect sprites used in the experiment were downloaded directly from itch.io,[1] which sells and shares independently produced games and game assets. Image background was removed from the sprites with transparency. The background and the object were emphasized with a Euclidean distance function when the background and object texture was unclear. The preprocessing of the experimental images is shown in Fig. 5.

GAN often does not improve performance even if the batch size increases. Heuristically, batch size 16 was identified as the most ideal image style conversion. The number of training images was set to 2000 per epoch and the best output was obtained at around 120 epochs.

The model was trained using a single RTX-3080 GPU. The generator was modified by adding style label conditions based on the model structure of FastNeuralStyle with ResBlock. The discriminator utilizes the structure of PatchGAN with an added auxiliary classifier. The number of dimensions of the embedding vector to feed the style label into the model was set to 256, which was the same size as the input image. Adam (Adaptive Moment Estimation) (Kingma & Ba, 2015) was used as the optimization function. In our experiment, the model adopted hyperparameters of the loss functions $\lambda_{adv} = 1, \lambda_{rec} = 100, \lambda_{cls} = 0.1$.

[1] https://itch.io/

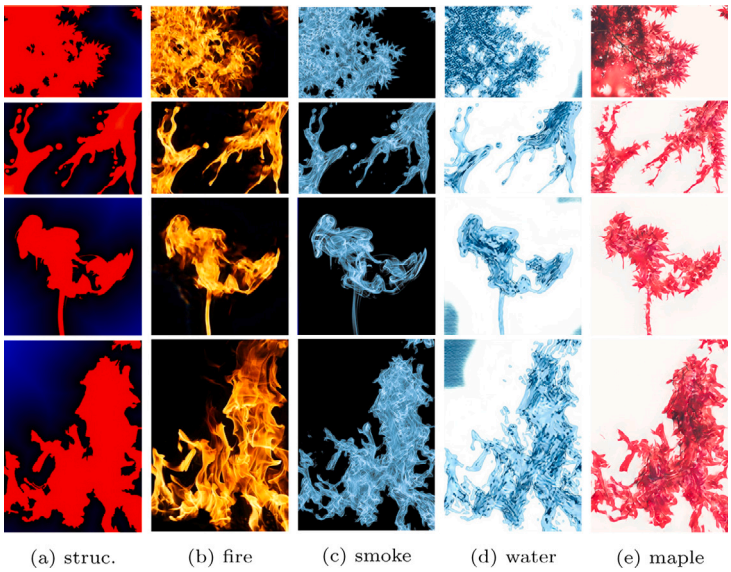

Fig. 4. Training dataset in GESGAN. (a) structure image of the original set, (b) fire-styled images, (c) smoke-styled images, (d) water-styled images, (e) maple-styled images; and diagonal images represent the original images of each style. Artifacts in water-styled images have been generated from synthesizing to the each style.

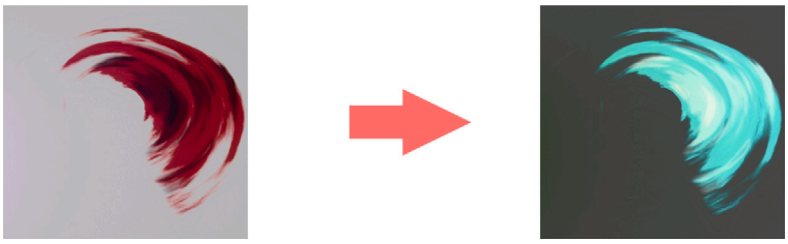

Fig. 5. Example of preprocessing of object and background emphasizing images.

4.2. Experiments on sequential sprites

Game effect sprites are often composed of sequential images for motion effects. These images can be divided into frames or fed in a mini-batch like manner in the training. The experiment demonstrates satisfactory results for the synthesis of all style labels overall. Figs. 6, 7, and 8 show synthesized output of fire in the upper left, water in the upper right, smoke in the lower left, and maple style in the lower right. The results in Figs. 6, and 7 indicate insufficient synthesis of thin lines, but when the sequential images were put in a video, natural and reasonable result was obtained. Fig. 8 shows the synthesized result of a sequential game effect sprite of a single image. The result shows that even if the object is small, details were successfully synthesized. However, as the size of each frame of the image became smaller, the characteristics of each style were not richly expressed. Additionally, Fig. 6 is an example of a transformation of sequential images, which are the effect of slashing action, to each style label. Based on the synthesized results, it can be said that the proposed model synthesized images insufficiently on the part where objects or boundaries fades. This result appears due to the presence of shapes in large chunks and not due to the distinct form of input images. We created an animation to clarify the results of sequential sprites, seen in Fig. A1 of Appendix.

4.3. Experiments on various shapes

This experiment was conducted on the shapes that are used in actual game effects and the results were based on a single image rather than a sequential game sprite. Fig. 10 illustrates this experiment, and slash, shield, and electric ball shapes showed sufficient style and shape synthesis and good results in sequential output. In contrast, the bottom two rows of Fig. 10 show unsatisfactory synthesis of the input image. This is because, as shown in Fig. 9, the image objects are not clearly distinguished from the backgrounds.

4.4. Experiments on unusual shapes

In addition to the general game effect sprite, the experiment was conducted on the transformation of special shapes. The experiment was conducted with a relatively common sword-shaped sprite, flower-like shape, gear-like, a portal similar to the letter ‘O’, a black hole like a spread portal shape, and a star-shaped galaxy around the center. The results are shown in Fig. 11. The plain-patterned sword shape was synthesized into a distinctive image, although the restoration of the style was minimal. Flowers, gears, portals, black holes, and galaxies are images of transparent backgrounds. These transparencies were removed from the transparent sprites; however, the result demonstrated fair performance, whereas the galaxy shape was synthesized inadequately. This was because the size of the surrounding stars was small (additionally explained in the next subsection), and it was anticipated that the training model will have difficulty in synthesizing images with green colors that were not present in the training dataset.

4.5. Experiments on painting styles

There are many genres and types of games, and consequently, many different styles of painting. Therefore, this study also tested the ability to restore images and synthesize several painting styles. Fig. 12 shows an experiment on painting style. The cartoon style elucidates the results of unclear restoration of the thin lines in the body. In case of pixel art, even small objects such as the torso, legs, and weapons could

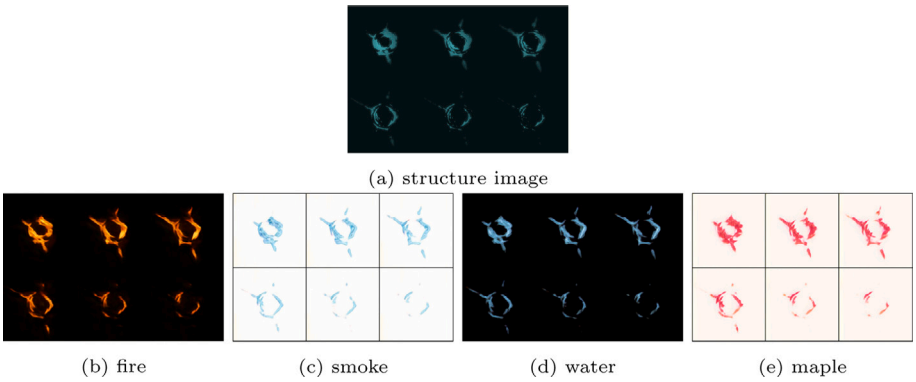

Fig. 6. Synthesized output per label of sequential game effect sprites. (a) is an example of the input structure image, (b)–(e) represent the synthesized outputs by each style. (b) fire-styled images, (c) smoke-styled images, (d) water-styled images, (e) maple-styled images.

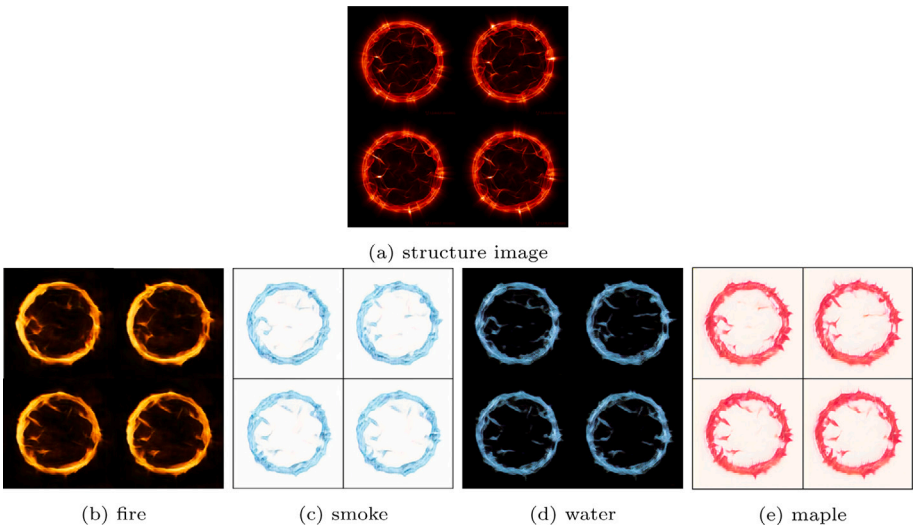

Fig. 7. Four frames of synthesized output per label of sequential game effect sprites. (a) is an example of the input structure image, (b)–(e) represent the synthesized outputs by each style. (b) fire-styled images, (c) smoke-styled images, (d) water-styled images, (e) maple-styled images. A sequential image of the sprite can check at Fig. A1 of Appendix.

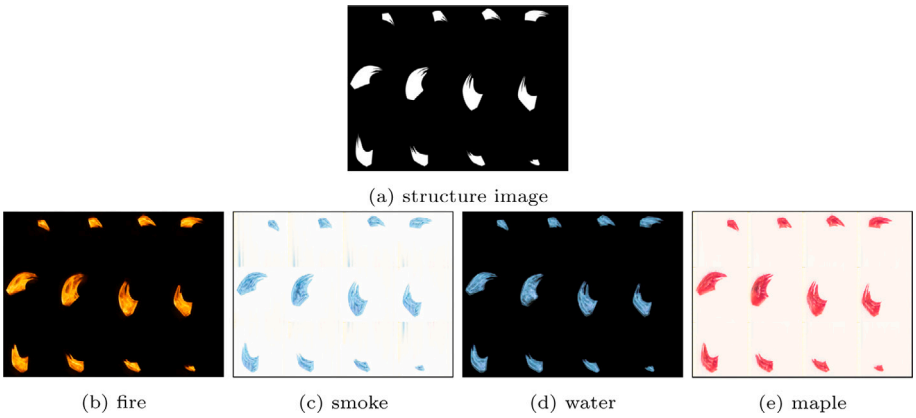

Fig. 8. Synthesized output per label of a single sequential game effect sprite. (a) is an example of the input structure image, (b)–(e) represent the synthesized outputs by each style. (b) fire-styled images, (c) smoke-styled images, (d) water-styled images, (e) maple-styled images.

be restored, but the large pixel units (bold) of pixel art could not be restored to the pixel art style. This is because the data being learned comprised wild photographs. For the vintage illustration, the model was expected to restore and synthesize very small object compared to its image size. GESGAN restored the small objects but did not synthesize the style adequately. The 'vintage * 4' shows the experimental result of an image that increased the object of the vintage picture approximately four times. Compared to the result of vintage image, which was a basic form, the style could be restored richly. The synthesis based on wild photographs restored the desired object and the floor parts together. For objects, except for the style of water, they have been restored sufficiently and synthesized the style richly. That is, when the object or line of the image to be transformed is not too small and based on colors contrasting with the background, the structure image can

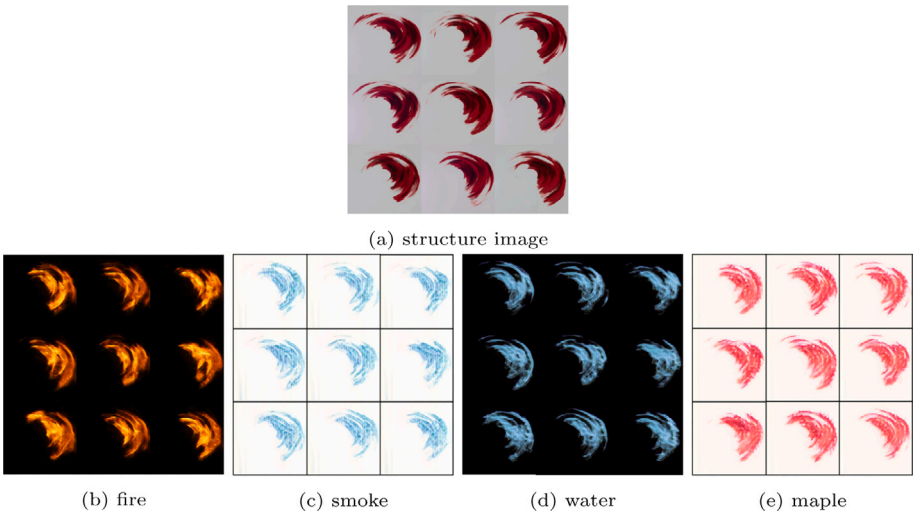

Fig. 9. Poor synthesized output per label of sequential game effect sprites. A sequential image of the sprite can be found at Fig. A1 of Appendix.

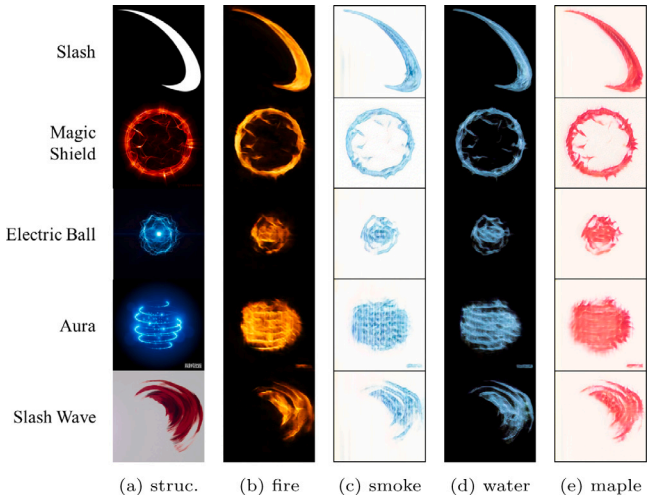

Fig. 10. Synthesized output per label of various shapes. Top three rows represent the successful synthesis result. Bottom two rows represent the poor synthesized output per label.

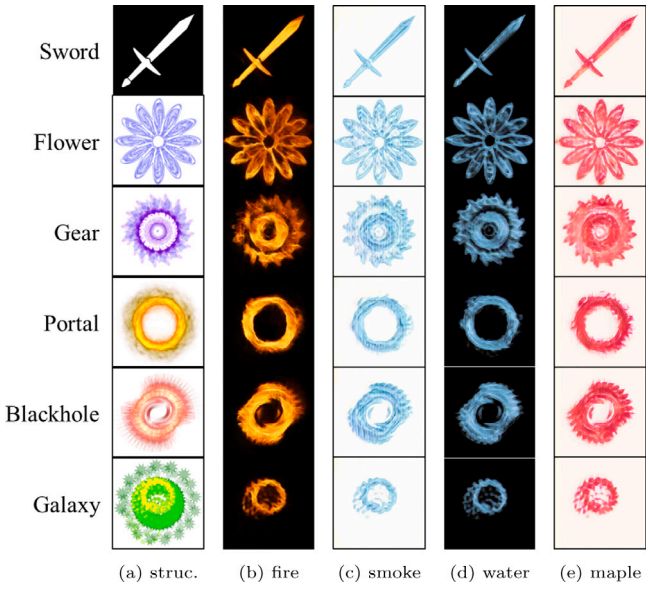

Fig. 11. Synthesized output per label of unusual shapes.

be synthesized well in harmony with the desired style. However, the results in terms of game effects of pixel art painting styles show the limitations of not restoring the painting style.

4.6. In comparison with baseline models

It is difficult to evaluate how GESGAN utilizes the image distribution for each label because it acquires data by augmenting images for one style through data augmentation and then trains them as training datasets. Furthermore, it is inappropriate to evaluate the diversity of the generated image or how much the synthesized image represents the object (Heusel et al., 2017; Salimans et al., 2016) because it generates the image based on the structure image. Therefore, in comparison experiments, we compare quantitatively with baseline models the quality of the generated images, similar to that in mGANprior (Gu et al., 2020) and SinGAN (Shaham et al., 2019).

The baseline models used included FastNeuralStyle (Johnson et al., 2016), pix2pixcGAN (Isola et al., 2017), and TET-GAN (Yang et al., 2019a). These models are condition-based and capable of supervised training models that utilize deep learning for a fair comparison. The baseline models were trained by transforming styles into models based

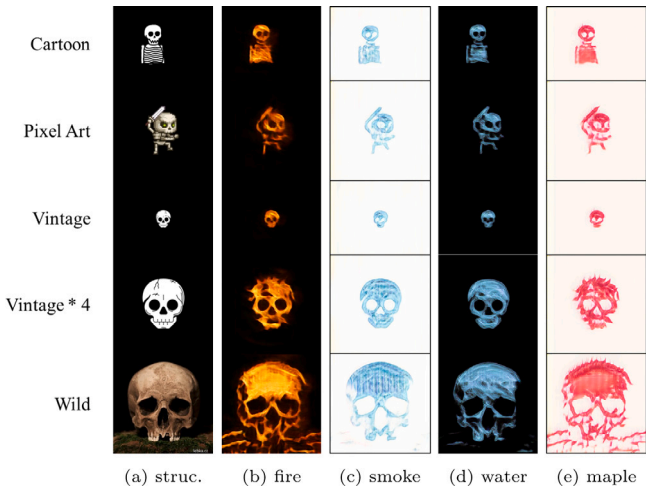

Fig. 12. Synthesized output per label of painting shapes.

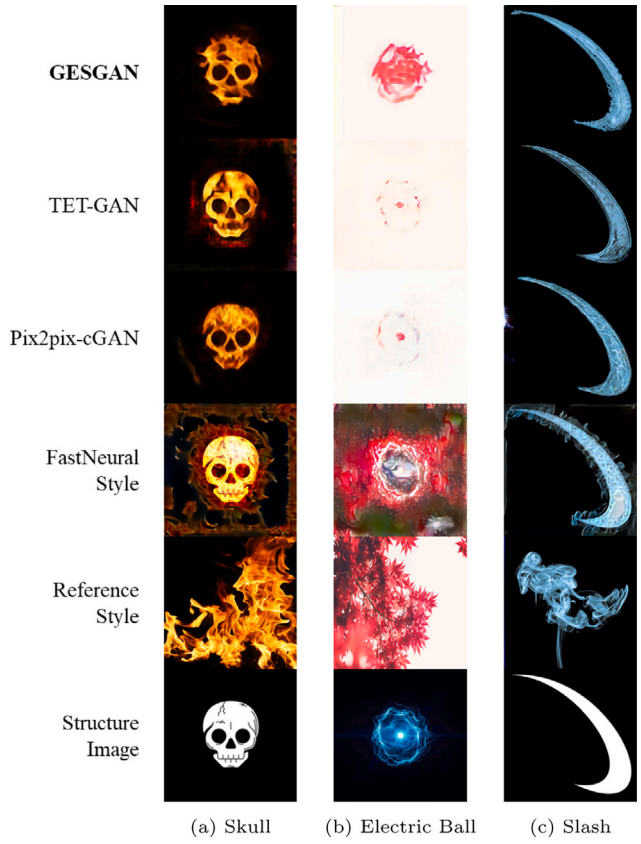

Fig. 13. Qualitative comparison with baseline models.

on structure images and with style labels. Each model is trained by implicitly representing the characteristics of the data as a potential feature by inputting a style label together through supervised learning, similar as in GESGAN's training method. Each baseline model was trained by adjusting hyperparameters to obtain optimal results.

For qualitative evaluation, Fig. 13 depicts the comparison of the synthesized images of each model with the results of GESGAN. Overall, reasonable images were generated when entering the slash image in all models. However, in the case of FastNeuralStyle, it may be observed that a reasonable image was synthesized from an artistic aspect. The electric ball results showed that FastNeuralStyle did not synthesize styles according to the structure of the input image. Pix2pixcGAN and TET-GAN generated results that could not recognize thin lines. However, GESGAN had some loss of thin lines, but it moderately synthesized the style and the structure image. FastNeuralStyle's synthesized result of skeleton and fire image was difficult to see because of structure-conscious synthesis. Although TET-GAN preserved its structure and details well, there was considerable noise. Pix2pixcGAN and GESGAN generated images creditably, but GESGAN was able to synthesize the styles.

Next, the evaluation metrics of BRISQUE (Mittal et al., 2012), PIQE (Venkatanath et al., 2015), and NIQE (Mittal et al., 2013) were used to evaluate the model indirectly. We used these metrics because our goal, as mentioned earlier, is to create realistic images that game designers can use simply in their games. Each evaluation was based on the outputs shown in Fig. 13. For BRISQUE, a spatial quality evaluator calculates a given image's non-reference image quality score. It compares the image to a pre-trained model calculated in a natural scene image with similar distortion. A lower score represents a better perceptual quality. PIQE is a metric for estimating the distortion of a patch and then calculating masks for artificially distorted blocks present in images. PIQE has an image quality score between 0 and 100, with 0 denoting a distinct and 100 denoting a blurred image. In addition, NIQE is a non-reference evaluation method that can be evaluated using only measurable deviations from statistical regularity observed in natural images. The lower the score, the better the perceptual quality of the image.

Table 1
Result of quantitative comparison with baseline models.

Model	FastNeuralStyle	Pix2pixcGAN	TET-GAN	GESGAN
Method	BRISQUE			
Slash	42.6805	59.2561	86.6202	66.2468
Electric ball	28.6019	47.9935	68.6012	47.5499
Skull	12.8179	67.8201	45.4962	46.7929
Mean	28.0334	58.3565	66.9058	53.5298
Method	PIQE			
Slash	3.7155	15.622	11.7643	8.6925
Electric ball	29.445	15.622	48.7212	9.2677
Skull	2.6036	30.1756	14.4845	16.7324
Mean	11.9213	20.4732	24.99	11.5642
Method	NIQE			
Slash	6.5793	31.9486	26.4823	12.1294
Electric ball	7.0133	24.3549	35.3101	26.1399
Skull	6.0119	21.425	24.6298	18.2301
Mean	6.5348	25.9095	28.8074	18.8331

The quantitative evaluation of each model is represented in Table 1. GESGAN was evaluated with the lowest score on all indicators except for the evaluation of the FastNeuralStyle. The average BRISQUE score was 9.101 lower, the PIQE score was 11.17, and the NIQE score was 8.53 lower. However, the FastNeuralStyle model is numerically better than our proposed method. Accordingly, we directly compared the generated images, which can be seen in Fig. 13. As you can see in Fig. 13, the FastNeuralStyle did not restore the input structure image well. Comparing the generated results with other models, a lot of noise is included not only in our model but also in other models. Even if you look at (c) Slash in Fig. 13, other models have clear boundaries, so we can crop them to suit our purpose. However, although FastNeuralStyle has the best numerical performance, the generated image has a lot of noise, limiting its use in actual game development. When looking at these quantitative and qualitative evaluation results in combination, the method we propose creates a more realistic image than the existing methods.

5. Conclusion

In this study, we proposed a conditional style transfer model, GESGAN (Game Effect Sprite Generative Adverse Network), which synthesizes a new sprite from a 2D game effect sprite or image. The proposed network learns a synthesis of the structure and the style labels from only 4 main images and can be converted into multiple styles with a single GAN. Comparing the output of GESGAN to FastNeuralStyle, pix2pixcGAN, GESGAN vividly generated new images with reliable structure and rich styles. In quantitative evaluation, the BRISQUE score was 9.101, the PIQE score was 11.17, and the NIQE score was 8.53 lower compared to other models except FastNeuralStyle. These results demonstrate that GESGAN generated a reliable image with relatively distinct and realistic appearance compared to the other models.

During game development, specific structures are often transformed or deformed not only in certain effect sprites but also in groups. The approach presented in this study can replace these iterations as design tools for pilot testing or effect sprite generation. The proposed GESGAN shows that style transformation techniques can be applied to the 2D game sprite generation processes, which means that they can significantly optimize a 2D sprite generation task that typically requires considerable manual work.

Our approach could be improved to generate higher quality game effect sprites through solutions in several key areas. First, improved scalability of the model can be achieved by applying the style image rather than the one-hot encoding of style label based on conditions, which can be regarded as a limitation of GESGAN. In addition, the model can be improved to restore more details of the object, to transform the style more fruitfully, and to control the degree of the style transfer.

Attempts to solve real industrial problems by using deep learning models are continuing and our model is also a deep learning model specialized in the game field. If a large amount of data is used for training such a deep learning model, better performance can be achieved. However, there is not much data reflecting the specificity of the industry. Accordingly, in future work, we intend to study a model that can be effectively learned even if the number of data is insufficient or a model that can be helpful to many game designers by conducting research using data augmentation techniques. Moreover, our model cannot completely replace the efforts of game sprite designers. This is because the GAN often generates blurry images and the model does not reflect the alpha values for the image background. Therefore, it is difficult to naturally put the created game sprite into the game engine. We intend to study the game sprite production algorithm that can be further fine-tuned in future. We intend to develop a model that can completely replace the repetitive tasks of game sprite designers by further developing our research.

CRediT authorship contribution statement

JaeWon Kim: Conceptualization, Methodology, Software, Data curation, Validation, Writing – original draft, Writing – review & editing. **KyoHoon Jin:** Software, Writing – review & editing, Visualization. **SooJin Jang:** Software, Writing – review & editing. **ShinJin Kang:** Conceptualization, Writing – review & editing. **YoungBin Kim:** Conceptualization, Methodology, Writing – original draft, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This research was supported in part by Institute for Information & communications Technology Planning & Evaluation (IITP) through the Korea government (MSIT) under Grant No. 2021-0-01341 (Artificial Intelligence Graduate School Program (Chung-Ang University), Contribution Rate: 25%), the National Research Foundation of Korea (NRF) through the Korea government (MSIT) under Grant No. NRF-2022R1C1C1008534 (Contribution Rate: 25%), Ministry of Culture, Sports and Tourism and Korea Creative Content Agency (Project Number: R2020040186, Contribution Rate: 25%), and Culture Technology R&D Program through the Korea Creative Content Agency grant funded by Ministry of Culture, Sports and Tourism in 2021 (Project Name: A Specialist Training of Content R&D based on Virtual Production, Project Number: R2021040044, Contribution Rate: 25%).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2022.118491.

References

Aleem, S., Capretz, L. F., & Ahmed, F. (2016). Game development software engineering process life cycle: a systematic review. *Journal of Software Engineering Research and Development*, *4*(1), 1–30.

Almeida, M. S. O., & da Silva, F. S. C. (2013). A systematic review of game design methods and tools. In *International conference on entertainment computing* (pp. 17–29). Springer.

Azadi, S., Fisher, M., Kim, V. G., Wang, Z., Shechtman, E., & Darrell, T. (2018). Multi-content GAN for few-shot font style transfer. In *2018 IEEE/CVF conference on computer vision and pattern recognition* (pp. 7564–7573).

Brock, A., Lim, T., Ritchie, J., & Weston, N. (2017). Neural photo editing with introspective adversarial networks. arXiv arXiv:1609.07093.

Chang, H., Lu, J., Yu, F., & Finkelstein, A. (2018). PairedCycleGAN: Asymmetric style transfer for applying and removing makeup. In *2018 IEEE/CVF conference on computer vision and pattern recognition* (pp. 40–48).

Chelliah, B., Vallabhaneni, V., Lenkala, S., Mithran, J., & Reddy, M. (2019). 3D character generation using PCGML. *International Journal of Innovative Technology and Exploring Engineering*, *8*, 105–109.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. (2016). InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*.

Chen, Y., Lai, Y.-K., & Liu, Y. (2018). CartoonGAN: Generative adversarial networks for photo cartoonization. In *2018 IEEE/CVF conference on computer vision and pattern recognition* (pp. 9465–9474).

Choi, Y., Choi, M.-J., Kim, M., Ha, J.-W., Kim, S., & Choo, J. (2018). StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation. In *2018 IEEE/CVF conference on computer vision and pattern recognition* (pp. 8789–8797).

Efros, A. A., & Freeman, W. (2001). Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on computer graphics and interactive techniques*.

Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A neural algorithm of artistic style. arXiv arXiv:1508.06576.

Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image style transfer using convolutional neural networks. In *2016 IEEE conference on computer vision and pattern recognition* (pp. 2414–2423).

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., & Bengio, Y. (2014). Generative adversarial nets. In *NIPS*.

Gu, J., Shen, Y., & Zhou, B. (2020). Image processing using multi-code GAN prior. In *2020 IEEE/CVF conference on computer vision and pattern recognition* (pp. 3009–3018).

Guzdial, M. J., Reno, J., Chen, J., Smith, G., & Riedl, M. O. (2018). Explainable PCGML via game design patterns. arXiv, arXiv:1809.09419.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Hertzmann, A., Jacobs, C., Oliver, N., Curless, B., & Salesin, D. (2001). Image analogies. In *Proceedings of the 28th annual conference on computer graphics and interactive techniques*.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2017). GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *NIPS*.

Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. In *NIPS*.

Hong, S., Kim, S.-K., & Kang, S.-J. (2019). Game sprite generator using a multi discriminator GAN. *KSII Transactions on Internet and Information Systems*, *13*, 4255–4269.

Horsley, L., & Liebana, D. P. (2017). Building an automatic sprite generator with deep convolutional generative adversarial networks. In *2017 IEEE conference on computational intelligence and games* (pp. 134–141).

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *2017 IEEE conference on computer vision and pattern recognition* (pp. 5967–5976).

Jain, R., Isaksen, A., Holmgård, C., & Togelius, J. (2016). Autoencoders for level generation, repair, and recognition. In *Proceedings of the ICCC workshop on computational creativity and games* (p. 9).

Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. arXiv, arXiv:1603.08155.

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2018). Progressive growing of GANs for improved quality, stability, and variation. arXiv, arXiv:1710.10196.

Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF conference on computer vision and pattern recognition* (pp. 4396–4405).

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2020). Analyzing and improving the image quality of StyleGAN. In *2020 IEEE/CVF conference on computer vision and pattern recognition* (pp. 8107–8116).

Kim, H. (2017). Content generation using variational auto-encoder. In *Proc. of nexon developer conference*.

Kim, T., Cha, M., Kim, H., Lee, J. K., & Kim, J. (2017). Learning to discover cross-domain relations with generative adversarial networks. In *ICML*.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. CoRR, arXiv:1412.6980.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Aitken, A. P., Tejani, A., Totz, J., Wang, Z., & Shi, W. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *2017 IEEE conference on computer vision and pattern recognition* (pp. 105–114).

Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., & Yang, M.-H. (2017a). Diversified texture synthesis with feed-forward networks. In *2017 IEEE conference on computer vision and pattern recognition* (pp. 266–274).

Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., & Yang, M.-H. (2017b). Universal style transfer via feature transforms. In *NIPS*.

Li, C., & Wand, M. (2016a). Combining Markov random fields and convolutional neural networks for image synthesis. In *2016 IEEE conference on computer vision and pattern recognition* (pp. 2479–2486).

Li, C., & Wand, M. (2016b). Precomputed real-time texture synthesis with Markovian generative adversarial networks. arXiv, arXiv:1604.04382.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. arXiv, arXiv:1411.1784.

Mittal, A., Moorthy, A. K., & Bovik, A. (2012). No-reference image quality assessment in the spatial domain. *IEEE Transactions on Image Processing*, *21*, 4695–4708.

Mittal, A., Soundararajan, R., & Bovik, A. (2013). Making a "completely blind" image quality analyzer. *IEEE Signal Processing Letters*, *20*, 209–212.

Murez, Z., Kolouri, S., Kriegman, D., Ramamoorthi, R., & Kim, K. (2018). Image to image translation for domain adaptation. In *2018 IEEE/CVF conference on computer vision and pattern recognition* (pp. 4500–4509).

Nazeri, K., Ng, E., Joseph, T., Qureshi, F., & Ebrahimi, M. (2019). EdgeConnect: Generative image inpainting with adversarial edge learning. arXiv, arXiv:1901.00212.

Odena, A., Olah, C., & Shlens, J. (2017). Conditional image synthesis with auxiliary classifier GANs. In *ICML*.

Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. CoRR, arXiv:1511.06434.

Reed, S. E., Zhang, Y., Zhang, Y., & Lee, H. (2015). Deep visual analogy-making. In *NIPS*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training GANs. In *NIPS*.

Schell, J. (2008). *The art of game design: a book of lenses*. CRC Press.

Seo, C. W., & Seo, Y. (2021). Seg2pix: Few shot training line art colorization with segmented image data. *Applied Sciences*, *11*(4), 1464.

Serpa, Y. R., & Rodrigues, M. A. F. (2019). Towards machine-learning assisted asset generation for games: A study on pixel art sprite sheets. In *2019 18th Brazilian symposium on computer games and digital entertainment* (pp. 182–191).

Shaham, T. R., Dekel, T., & Michaeli, T. (2019). SinGAN: Learning a generative model from a single natural image. In *2019 IEEE/CVF international conference on computer vision* (pp. 4569–4579).

Shu, Z., Yumer, E., Hadap, S., Sunkavalli, K., Shechtman, E., & Samaras, D. (2017). Neural face editing with intrinsic image disentangling. In *2017 IEEE conference on computer vision and pattern recognition* (pp. 5444–5453).

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. CoRR, arXiv:1409.1556.

Snodgrass, S., & Ontañón, S. (2017). Learning to generate video game maps using Markov models. *IEEE Transactions on Computational Intelligence and AI in Games*, *9*, 410–422.

Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A. K., Isaksen, A., Nealen, A., & Togelius, J. (2018). Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games*, *10*(3), 257–270.

Taigman, Y., Polyak, A., & Wolf, L. (2017). Unsupervised cross-domain image generation. arXiv, arXiv:1611.02200.

Tolosana, R., Vera-Rodríguez, R., Fierrez, J., Morales, A., & Ortega-Garcia, J. (2020). DeepFakes and beyond: A survey of face manipulation and fake detection. *Information Fusion*, *64*, 131–148.

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2018). Deep image prior. In *2018 IEEE/CVF conference on computer vision and pattern recognition* (pp. 9446–9454).

Venkatanath, N., Praneeth, D., Bh., M. C., Channappayya, S. S., & Medasani, S. (2015). Blind image quality evaluation using perception based features. In *2015 Twenty first national conference on communications* (pp. 1–6).

Wang, X. E., Oxholm, G., Zhang, D., & Wang, Y. (2017). Multimodal transfer: A hierarchical deep convolutional neural network for fast artistic style transfer. In *2017 IEEE conference on computer vision and pattern recognition* (pp. 7178–7186).

Yang, S., Liu, J., Lian, Z., & Guo, Z. (2017). Awesome typography: Statistics-based text effects transfer. In *2017 IEEE conference on computer vision and pattern recognition* (pp. 2886–2895).

Yang, S., Liu, J., Wang, W., & Guo, Z. (2019a). TET-GAN: Text effects transfer via stylization and destylization. In *AAAI*.

Yang, S., Liu, J., Yang, W., & Guo, Z. (2019b). Context-aware text-based binary image stylization and synthesis. *IEEE Transactions on Image Processing*, *28*, 952–964.

Yang, S., Wang, Z., Wang, Z., Xu, N., Liu, J., & Guo, Z. (2019). Controllable artistic text style transfer via shape-matching GAN. In *2019 IEEE/CVF international conference on computer vision* (pp. 4441–4450).

Zeng, J., Chen, Q., Liu, Y., Wang, M., & Yao, Y. (2021). StrokeGAN: Reducing mode collapse in Chinese font generation via stroke encoding. In *AAAI*.

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE international conference on computer vision* (pp. 2242–2251).