Game Sprite Generator Using a Multi Discriminator GAN

Seungjin Hong[1], **Sookyun Kim**[2], **Shinjin Kang**[1*]
[1] School of Games, Hongik University
2639 Sejong-ro, Jochiwon, Sejong, Korea
[e-mail: directx@hongik.ac.kr]
[2] Department of Game Engineering, Paichai University
155-40 Baejae-ro, Seo-Gu, Daejeon, Korea
[e-mail: nicesk@gmail.com]
*Corresponding author: Shinjin Kang

*Received February 25, 2019; revised May 27, 2019; accepted June 6, 2019;
published August 31, 2019*

Abstract

This paper proposes an image generation method using a Multi Discriminator Generative Adversarial Net (MDGAN) as a next generation 2D game sprite creation technique. The proposed GAN is an Autoencoder-based model that receives three areas of information—color, shape, and animation, and combines them into new images. This model consists of two encoders that extract color and shape from each image, and a decoder that takes all the values of each encoder and generates an animated image. We also suggest an image processing technique during the learning process to remove the noise of the generated images. The resulting images show that 2D sprites in games can be generated by independently learning the three image attributes of shape, color, and animation. The proposed system can increase the productivity of massive 2D image modification work during the game development process. The experimental results demonstrate that our MDGAN can be used for 2D image sprite generation and modification work with little manual cost.

Keywords: Generative Adversarial Nets (GANs), 2D Game Sprite, Deep Learning, Game Development, Neural Network

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.NRF-2019R1A2C1002525)
A preliminary version of this paper was presented at ICONI 2018, and was selected as an outstanding paper.

1. Introduction

An important goal of game companies is the optimization of the artistic process. The game industry has been constantly trying to improve artistic processes over the years by adopting middleware. However, 2D game art resource development work still entails high costs due to intensive pixel modification using a large amount of resources. A graphics resource development technique that is often used in 2D games is the 2D sprite generation technique. This technique is a method of expressing movement while continuously changing a large number of images manually. It requires time-consuming pixel-by-pixel color correction and color palette designation for single motion expression, which leads to an increase in development costs. In the sprite generation process, artists usually create one 2D character in 8 to 16 directions, with 10 to 30 consecutive images per second. Artists create all pixels by hand within a specified resolution. In a game, the number of such 2D characters ranges from dozens to hundreds, and producing this amount of 2D sprites is a burden for game developers. To make 2D sprite generation more efficient, we propose a new image generation technique based on a Multi Discriminator Generative Adversarial Net (MDGAN) model that references color, shape, and animation independently. Our system uses three objective functions, and it has a structure that competes with three different networks. Using our technique, artists can create a new composite of 2D sprite sets with different shapes, colors, and animations, based on a pre-made 2D sprite set. Our results show that MDGAN can be applied effectively in 2D game images by easily copying and modifying a large number of sprite sets.

2. Related Work

Generating sequential synthetic data that mimic real conditions is an important problem, and plenty of literature discusses this [1][2][3]. Generative Adversarial Networks (GANs) [4][5] have achieved impressive results in image generation [6][7], image editing [8], and representation learning [9][10]. Recent methods adopt similar ideas for conditional image generation applications, such as text2image [11], image inpainting [12], and future prediction [13], as well as for other domains such as videos [14] and 3D models [15].

The task of image generation involves changing a particular aspect of a input image to another. Many GAN extensions have been proposed in this field. Deep Convolutional Generative Adversarial Networks (DCGANs) [16] introduce a class of Convolution Neural Networks (CNNs) that have certain architectural constraints, and demonstrate an unsupervised learning method that learns a hierarchy of object representations in both the generator and discriminator. Additionally, such a network uses learned features for novel tasks, demonstrating its applicability as general image representations. Auto-encoder-based GANs (AE-GANs) [17], which use an auto-encoder to encourage a model to better represent all the data they are trained with, discourage mode-collapse of GANs. Pix2Pix [18] proposes a learning method using a pair of data with a supervised learning approach. Pix2Pix is a method of learning to convert data from two domains into data in a domain that is paired in a specific domain. Instead of the auto-encoder method, Pix2Pix applies the GAN learning method to generate more realistic images. In addition, U-Net [19] is applied to the generator to improve noise and image quality. Bidirectional GAN (BiGAN) [20] has been suggested as a means of learning bidirectional mapping, as it demonstrates that the resulting learned feature representation is useful for auxiliary supervised discrimination tasks, making it competitive with contemporary approaches to feature learning. Recently, Cycle-GAN [21] and DiscoGAN [22] were proposed for image-to-image translation. These networks use the underlying cycle consistency principle, whereby image-to-code translation reveals a meaningful connection. These methods have become popular for image-to-image translation. StarGAN [23] is a GAN that learns the mappings among multiple domains using only a single generator and a discriminator, training effectively from images of all domains. FusionGAN [24] generates a fusion image with the identity of input image x and the shape of input image y. This network can simultaneously train on more than two image datasets in an unsupervised manner. The Boundary Equilibrium GAN (BEGAN) [25] is a further evolution of the GAN network. This model improves the unbalanced learning of the generator and the discriminator during GAN training. In the existing GAN, it was difficult to balance the learning between the generator and the discriminator. BEGAN solves this problem by giving coefficients to the objective function of the generator and the discriminator.

In this study, we show how a GAN technique can be used to effectively generate 2D images. In particular, this paper proposes a new Multiple Discriminator GAN (MDGAN) structure that extracts shape, color, and animation independently from multiple images. We visualized our work and compared our MDGAN with AE-GAN, BEGAN, and PIX2PIX results.

There have been many attempts to apply GANs to game content creation. Jain et al. learned the pattern of the game map using an auto-encoder, and automatically generated a new game map [26]. Xue et al. proposed a new convolution artificial neural network model, and combined new behaviors with objects in images [27]. Reed et al. proposed a model using an auto-encoder, a rotated game sprite, and 3D car model images, and applied animation to other images [28]. Summerville et al. suggested a machine-learned technique to train generators on Super Mario Bros. videos, generating levels based on latent play styles learned from the YouTube video. They demonstrated the process for extracting the path from video and how the information feeds into an LSTM/RNN [29]. Snodgrass et al. proposed Markov models to generate content for multiple games. They applied the Markov models to three game genres in order to determine how well their models perform in terms of the playability of the generated content [30]. Horsley et al. suggested that CNNs can be useed for game sprite generation even with few input datasets. They utilized a DCGAN for learning and generation of sprites [31]. Kim proposed an automatic character portrait generation system using Variable Auto-Encoder (VAE) [32]. Beckham et al. proposed terraion generation system by leveraging extremely high-resolution terrain and heightmap data provided by the NASA project. They used GANs to create a two-stage pipeline in which heightmaps can be randomly generated as well as a texture map that is inferred from the heightmap [33]. Giacomello et al. applied GANs to learn a model of DOOM levels from human-designed content. They trained two GANs: one using plain level images, one using both the images and some of the features. Their results showed that GANs could generate structure of DOOM levels in first person shooter games [34].

According to our knowledge, our approach is the first to apply the GAN model to game sprite generation for the automation of 2D game art production using color, shape, and animation properties. Our research aims to partially automate 2D art resource work, which is traditionally all done by hand, by using a GAN algorithm.

3. 2D Sprite Set in Games

2D sprite animation is a technique used to create the illusion of movement using static 2D images. An animation consists of multiple frames that are shown in a sequence at set intervals. An animation of someone running can be achieved by taking pictures of the person while running and playing those images back sequentially in a loop. **Fig. 1** shows a "sprite set" image with a complete cycle of a male character's behavior. Each line contains a frame of animation. When these frames are shown sequentially over a period of time, they appear as an animated image. The frame rate is how often the frame is changed per second. If a character is to complete a cycle in one second, 30 frames must be shown per second, so the frame rate is 30 FPS. An animation is a very simple state machine. The running character in our illustration has 30 states as per the sprite sheet. The numbered frames represent the states a running character goes through one at a time. The current state is determined by the amount of time since the animation began. If the animation is looping, it returns to the first frame after all frames have been shown.

This paper proposes a method to convert a set of sprites into a set of sprites in a different style using the image-to-image translation technique. This can reduce the manual cost of creating a different but similar-style character sprite set. To do this, an artist needs to be able to change specific elements in the sprite set. In order to meet these conditions, image attributes need to be selected independently when image-to-image translation technology is applied.

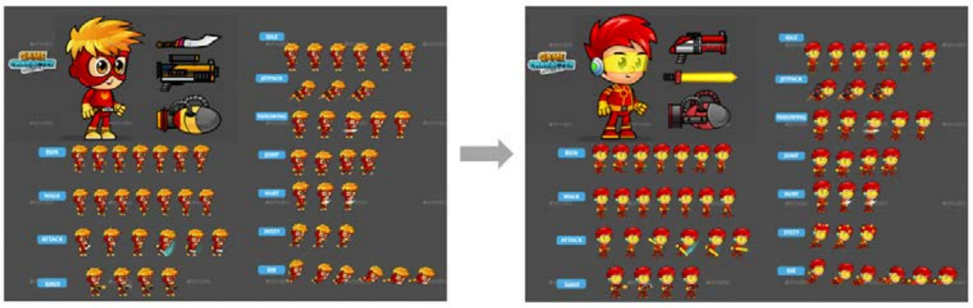

Fig. 1. Example of original sprite set (left) and manually modified sprite set by artist (right).

4. Generative Adversarial Nets

A GAN uses a training method different from general artificial neural networks. There are two networks in a GAN, the generator and the discriminator, and these two networks compete and train. The generator generates an image by receiving the latent random variable as input, and the discriminator tries to find the actual image by receiving the image generated by the generator and the actual image. The quality of the images produced by the generator as a result of learning from each other through the confrontation of the two networks is better than using the learning method over the existing auto-encoders. The existing learning method of the auto-encoder learns with the goal of reducing the error between the correct image and the pixel of the generated image. In this process, the model learns to reduce only the value of the average error, so it does not produce a neat image. However, in the GAN, the generator learns to generate an image very similar to the actual image through a learning method that must deceive the discriminator model, which distinguishes between the actual image and the generated image. As a result, it is possible to improve the performance of generators through a contradiction structure that makes comparisons in a learning method that reduces a simple total error.

The generator takes a one-dimensional random vector called a latent random variable as an input, and generates the same dimension image as the actual image set. The discriminator receives both the image generated by the generator and the image of the actual image set at the same time, and learns whether the image received is an actual image or a generated image. Since the performance of the generator improves according to the performance of the discriminator, the learning rate of the two models is important. As a result, when the learning is completed, the generator generates various images according to the input vector. By changing the input vector, a particular feature can be represented in the generated image such as eyeglasses, hair color, or gender. However, the problem is that it is not known what characteristics the input vector has, and these must be confirmed manually by hand. The black box attribute of the GAN structure is a major obstacle to the utilization of an industrial resource production process. For artists to be able to create resources using image-to-image translation, the network structure must be classified by image attributes. In this way, if the GAN learning process is not sufficient, the artist should be able to obtain information about the network weakness. Our MDGAN structure aims to meet this requirement.

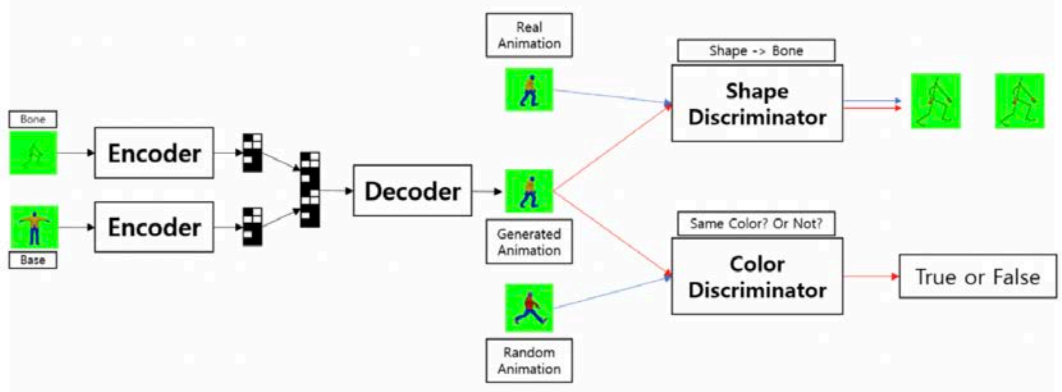

Fig. 2. Overview of proposed MDGAN.

5. Multiple Discriminant Generative Adversarial Nets

5.1 Generator

We propose a new GAN structure that extracts shape, color, and animation independently from multiple images. **Fig. 2** shows the process of the overall network. The aim is to confirm the possibility of separating shape, color, and animation when a large number of sprites need to be changed collectively. Our goal is to train a generator, G, that learn mappings among multiple domains. To achieve this, we can train G to translate input image x1 (animation information with bone graph) and image x2 (shape and color information) into an output image y. By using this image, the proposed model is synthesizing an animated sprite image from random noise. Figure 3 shows the structure of the generator. All window filter sizes were fixed to (3, 3). The size of stride was (2, 2). The number of filters doubled as the size of the image shrank. Because two encoders were used, the decoder took as input the combined result of the two encoders, and each encoder did not share parameters. The resizing of the decoder was used to expand the size of the image, and it extended the image by interpolating the values of surrounding pixels. The resulting layers from the two encoders can be merged into a single layer and entered into the input of the decoder. The decoder, which receives the compressed information of the two encoders as inputs, trained them to produce a mixture of images. We also applied a U-Net that reduced information loss of input image to the generator. Since two encoders were used for image synthesis, both outputs of the two encoders were used as inputs to the decoder.

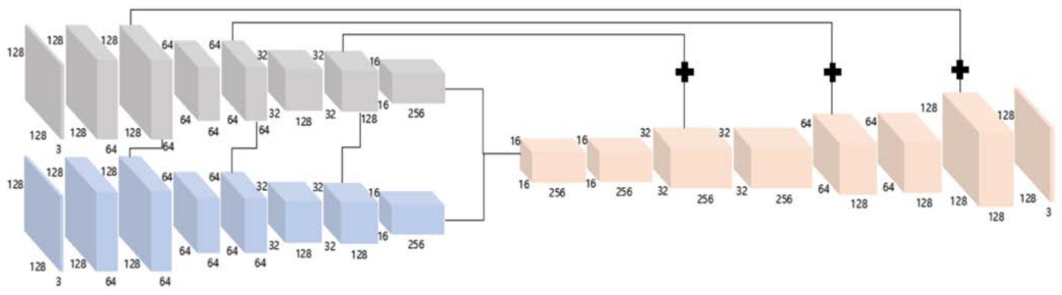

Fig. 3. Architecture of animation generator in proposed model.

5.2 Discriminators

The GAN's adversarial structure and the two discriminators were used differently from conventional methods. Also, similar to the Pix2Pix method, the objective function was used with the auto-encoder objective function. The first discriminator is a shape discriminator that learns to create a pair of skeletons in the character's animation when the character image is received as input. The second discriminator is a color discriminator that receives two character images as inputs and classifies whether the two characters are of the same color structure. In contrast to each discriminator, what the generator wants to learn is the relationship between the character and the skeleton through the shape discriminator, as well as to learn the position and shape of the color through the color discriminator. The training trains each discriminator, then uses the adversarial objective function to train the generator, and the generator also trains the objective function of the auto-encoder. **Table. 1** shows the detailed parameters of each discriminator. We used the following loss functions and Pseudocode code for MDGAN.

```
**for** number of training iterations **do**

1) Sample minibatch of $m$ character samples $\{c_1{}^{(1)}, \ldots, c_1{}^{(m)}\}$ from data distribution $P_{character}$

2) Sample minibatch of $m$ character samples $\{c_2{}^{(1)}, \ldots, c_2{}^{(m)}\}$ from data distribution $P_{character}$

3) Sample minibatch of $m$ skeleton samples $\{s^{(1)}, \ldots, s^{(m)}\}$ from data distribution $P_{skeleton}$

4) Sample minibatch of $m$ base samples $\{b^{(1)}, \ldots, b^{(m)}\}$ from data distribution $P_{base}$

5) Update the shape discriminator ($D_1$) by ascending its stochastic gradient:
```

$$\nabla\theta_{d1}\frac{1}{m}\sum_{i=1}^{m}\left\|s^{(i)} - D_1\left(c_1{}^{(i)}\right)\right\|$$

6) Update the color discriminator (D_2) by ascending its stochastic gradient:

$$\nabla\theta_{d2}\frac{1}{m}\sum_{i=1}^{m}\log D_2\left(c_1{}^{(i)},c_2{}^{(i)}\right)$$

7) Update the shape discriminator (D_1) by ascending its stochastic gradient:

$$\nabla\theta_{d1}\frac{1}{m}\sum_{i=1}^{m}\left(\left\|s^{(i)}-D_1\left(c_1{}^{(i)}\right)\right\|-\left\|s^{(i)}-D_1\left(G\left(s^{(i)},b^{(i)}\right)\right)\right\|\right)$$

8) Update the color discriminator (D_2) by ascending its stochastic gradient:

$$\nabla\theta_{d2}\frac{1}{m}\sum_{i=1}^{m}\log D_2\left(c_1{}^{(i)},c_2{}^{(i)}\right)+\log\left(1-D_2\left(c_1{}^{(i)},G\left(s^{(i)},b^{(i)}\right)\right)\right)$$

9) Update the generator by ascending its stochastic gradient:

$$\nabla\theta_g\frac{1}{m}\sum_{i=1}^{m}\left(\left\|s^{(i)}-D_1\left(G\left(s^{(i)},b^{(i)}\right)\right)\right\|+\log D_2\left(c_1{}^{(i)},G\left(s^{(i)},b^{(i)}\right)\right)\right)$$

end for

Table. 1. Layer Parameters (cl = Convolution Layer, fcl = Fully Connected Layer)

Layer	**Shape Discriminator**		
	kernel_size	stride_size	filter_size
Convolution Layer 1	[3,3]	[1,1]	64
	[3,3]	[2,2]	64
	[3,3]	[1,1]	64
	[3,3]	[2,2]	64
	[3,3]	[1,1]	128
	[3,3]	[2,2]	128
	[1,1]	[1,1]	1024
	[3,3]	[1,1]	256
	[3,3]	[1,1]	256
	Resize : upscale x 2		
Convolution Layer 2	[3,3]	[1,1]	128
	[3,3]	[1,1]	128
	Resize : upscale x 2		
Convolution Layer 3	[3,3]	[1,1]	128
	[3,3]	[1,1]	128
	Resize : upscale x 2		
Convolution Layer 4	[3,3]	[1,1]	64
	[3,3]	[1,1]	64
	[3,3]	[1,1]	3

	Color Discriminator (Encoder)		
Convolution Layer 5	[3,3]	[1,1]	64
	[3,3]	[2,2]	64
	[3,3]	[1,1]	128
	[3,3]	[2,2]	128
	[3,3]	[1,1]	128
	[1,1]	[1,1]	128
	Color Discriminator (Classifier)		
Convolution Layer 6	[3,3]	[1,1]	256
	[3,3]	[2,2]	256
	[3,3]	[1,1]	512
	[3,3]	[2,2]	512
	[3,3]	[1,1]	1024
Convolution Layer 7	[3,3]	[1,1]	1024
	[1,1]	[1,1]	512
	Reshape: Unit Size = 32768		
Fully Connected Layer	Unit Size = 2		

5.3 Post Image Processing

The resultant image of the model generates noise in the process of decoding information, which is lost when encoding is performed. The loss function used in the training of the model is trained to reduce the loss value according to the difference between the generated composite image and the target image. In this process, a composite image with noise increases the unnecessary loss value.

To overcome this problem, we applied the post-processing applied image to the loss function instead of the synthetic image used for training. Therefore, there is no loss value that can occur in an unnecessary background image. Applying the image without noise to the loss function instead of applying the image containing the noise value can reduce unnecessary learning cost in 2D sprite set generation. Consequently, faster and more accurate training is possible. In order to reduce the influence of the learning speed, the post-processing task is implemented during the learning stage so that it can be executed simultaneously in the learning process using the GPU rather than the processing using the CPU.

6. Experiment

We used a Linux operating system and one GTX1080 GPU. The libraries and code used in the implementation were Python3 and Tensorflow. Three experiments were conducted. The first experiment showed color and shape transition between different character images. The second experiment showed color and shape transition between character and object images. The third experiment showed the color, shape, and animation transition result.

6.1 Color & Shape Transition Between Character Images

The data used in the first experiment were 1,530 images with three uniformly distributed RGB values. By combining these images with color and shape combinations, about 380,000 combinations of images were created, and tbese made up for the amount needed for training.

Fig. 4 shows the result. The images in the top row of the figure show the training process, and the lower row images show the generation process. The generated images that received images of colors were not used for training. The generated image (the bottom right Result image in Figure 4) showed an accurate combination of results similar to the composite images created during the training process. This result means that our methodology can be used to easily change the color of a sprite set by using a reference image.

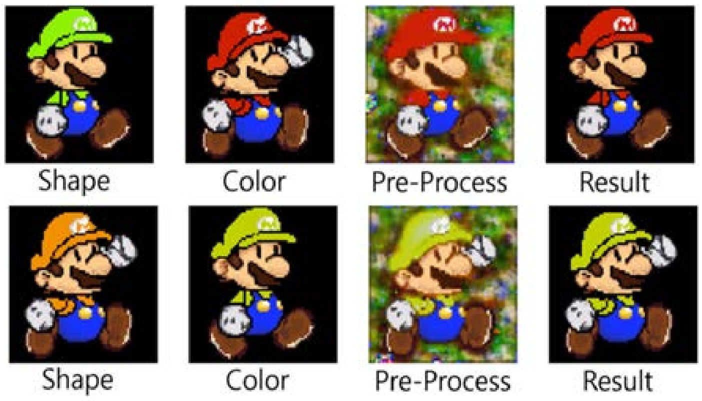

Fig. 4. Training result image (above), generation result (below).

6.2 Color & Shape Transition Between Character and Object Image

The second experiment was a synthetic experiment between character images and various object images. Unlike the dataset of the first experiment with RGB distribution, the dataset of the second experiment was a synthetic experiment with images of various objects and animals present. The total number of colors used in the experiment was 24. Of the 24 colors, up to 21 colors were used for training, and the remaining three colors, Lime, Green, and Orange, were used for color image synthesis tests not used in training. In addition, the synthesized test was performed on images that had not been used for learning about colors already learned.
A separate web image crawler was implemented for the purpose of constructing the dataset to be used as a color image. Data were collected through a combination of four keywords and image names. The words used as keywords consisted of “Fruit,” “Animal,” “Can,” “Bottle,” and so on, which can all contain various colors. An example of a combination of colors and keywords is “Red Color Can,” and a combination of up to 20 images can be collected for each keyword.
Fig. 5 shows the result of the synthesized image, the color used in the training was not used in evaluation. The trained model shows that the key colors were well extracted from the object images received as inputs and were used to produce results for composite images. In the synthesized image, the second composite image extracted one color successfully from a black background and a flame image mixed with several colors. The fourth composite image extracted key reds from a beverage can image containing white letters. This result shows that our methodology has high extraction stability for color attribute images.

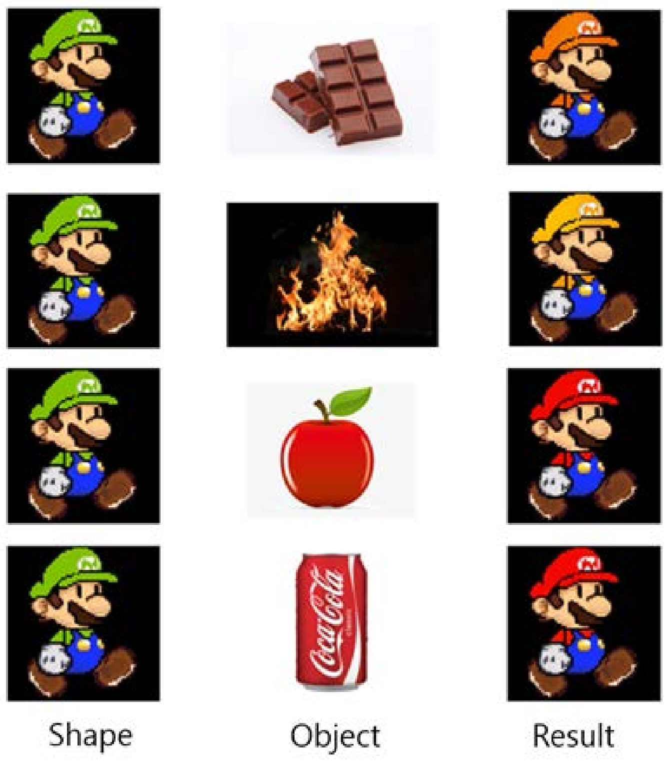

Fig. 5. Generation results between character and object image.

6.3 Animation Generation

The data needed for the experiment were three types of images: skeleton, color character, and animated character image. **Fig. 6(a)** shows an image used as a skeleton image. All skeleton animation images consisted of a total of 16 image data sets. **Fig. 6(b)** is part of the color character image. The character image consisted of 7,776 images, that is, an image of six areas changed to six colors. **Fig. 6(c)** shows an animation character image set. Each color character represents an animation of a skeleton. Each image is composed of a skeleton image and a color character image and comprises about 120,000 images. The images used in the learning were generated by dividing the characters including the animation into 16 frames and changing the color of the model.

In order to verify the performance of the proposed model, experiments were undertaken with models using the learning methods of AE-GAN, BEGAN, and Pix2Pix. To evaluate the learning result, eight combinations of images were created in the training completed model. The skeleton images used in the eight combinations used eight new skeleton images that were not used for training. The color character image consisted of four-color character images used in training and four-color character images not used in training.

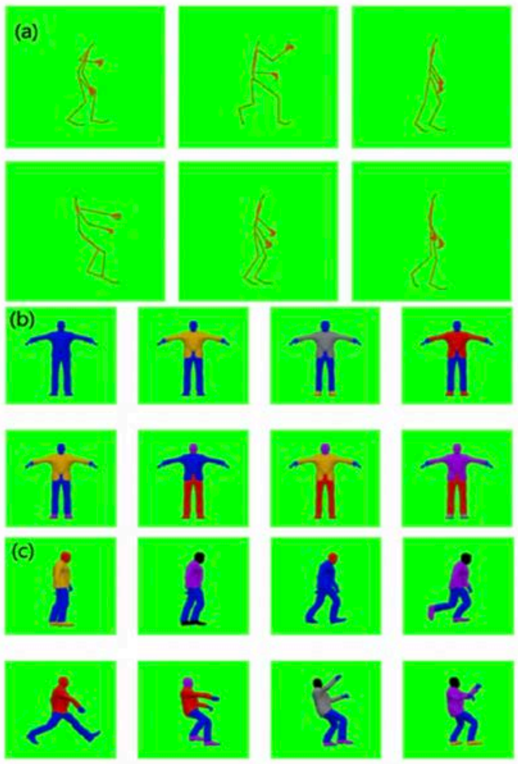

Fig. 6. Animation bone data set (a), model data set (b), and animation character data set (c).

The results of each model are shown in **Fig. 7**. In the AE-GAN model, the color character shows the result of the partial learning, in shape and animation. In the BEGAN model, all the images of the experiment result are the same shape. This means that animation learning has not occurred properly. In the Pix2Pix model, the shape of the skeleton is partially generated, and the the color is not generated overall. Our MDGAN model correctly learned the skeleton that present the behavior of the animation character. It also generated the animation character showing similar target color. The image created by MDGAN did not exactly have the same color as the target color because the color that was not used in learning was designated as the target color. In this experiment, we used six specified colors, and we expect that if we increase the number of colors, we will create a character with a more accurate color. As shown in **Fig. 8**, it can be seen that when the color character including the color of the partial pattern, which is not used in the training, is used as the input, it shows the animation character of the partial pattern color. This shows that our MDGAN model learned the location of the color of the animated character. Similar to BEGAN and several GAN papers, we used Mechanical Turk to evaluate the results. In the survey, 50 participants were asked to select the most photorealistic image out of four models. As results, 72% of the participants selected MDGAN as the model that produced the most realistic images.

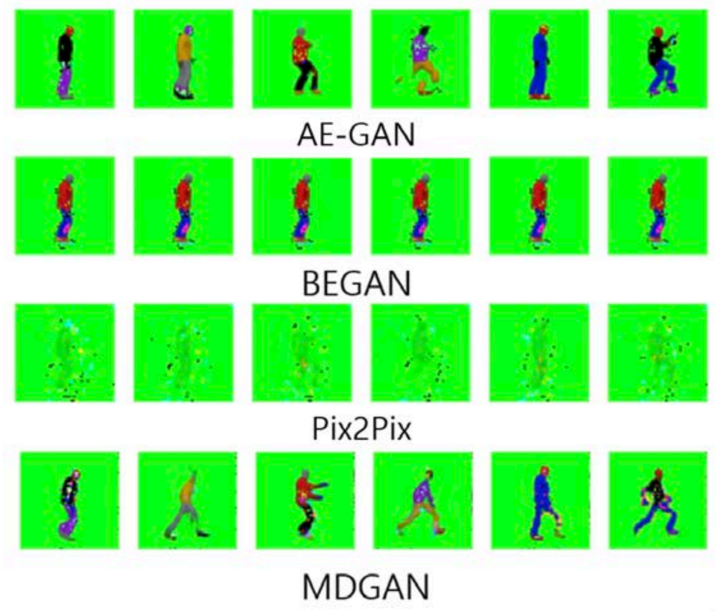

Fig. 7. Comparison results with AE-GAN, BEGAN, and Pix2Pix.

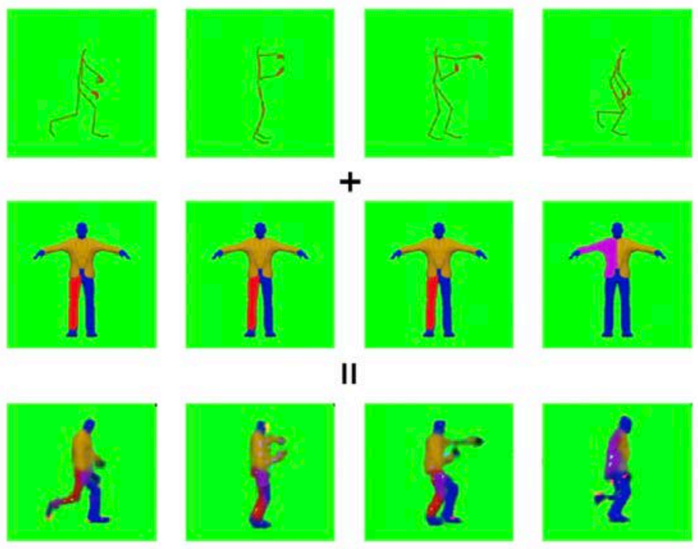

Fig. 8. MDGAN results with training data set.

7. Conclusions

In this paper, we proposed a MDGAN model to create image-to-image translation of 2D game sprite sets. The proposed network model consists of an encoder that takes shape and color from the input as well as a decoder that generates a composite image of shapes and colors combined from the compressed values of the two encoders. When we compared our MDGAN results with AE-GAN, Pix2Pix, and BEGAN, we were able to confirm that MDGAN generated new images with reliable shape and color. These image generation elements can be individually controlled by the user, demonstrating the possibility of easily creating new desired images. Recent advances in CNN/GAN algorithms have yielded remarkable results in many domains of industries. However, there are not many actual applications in the field of games. This is because the result generated by machine learning may not be able to generate the player's preferred content. Especially, the field of research related to sprite generation is very limited. This is because the number of open 2D image data available for machine learning is not enough for sufficient training. For this reason, the number of paper related to game sprite is very limited compared with that of web / video / text mining. In this situation, we propose an automatic method for generating large amount of bone and animation training image data using the existing 3D authoring tool and presented a machine learning method using it. In game development process, it is often the case that a specific sprite color is transformed collectively or a specific shape is collectively modified. Our approach can be applied to the in-house tool in that it can replace this repetition work. The proposed MDGAN model shows that the image-to-image translation technique can be applied to the in-game 2D sprite generation process, meaning that the 2D sprite creation task, which ordinarily requires a lot of manual work, can be significantly optimized.

References

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

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proc. of the IEEE conference on computer vision and pattern recognition*, pp. 2818-2826, 2016. Article (CrossRef Link)

[3] K. He, X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition,” in *Proc. of the IEEE conference on computer vision and pattern recognition*, pp. 770-778, 2016. Article (CrossRef Link)

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, and Y. Bengio, “Generative adversarial nets,” *advances in neural information processing systems*, pp. 2672-2680, 2014.

[5] Jie Li, Junjie Jia, Donglai Xu, “Unsupervised representation learning with deep convolutional generative adversarial networks,” in *Proc. of 2018 37th Chinese Control Conference (CCC)*, 2018. Article (CrossRef Link)

[6] E. L. Denton, S. Chintala, and R. Fergus, “Deep generative image models using a laplacian pyramid of adversarial networks,” *advances in neural information processing systems*, pp. 1486-1494, 2015.

[7] A. Radford, L. Metz, and S. Chintala., “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.

[8] J. Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, “Generative visual manipulation on the natural image manifold,” in *Proc. of european conference on computer vision*, pp. 597-613, 2016 Article (CrossRef Link)

[9] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” *arXiv preprint arXiv:1606.03498*, 2016.

[10] M. F. Mathieu, J. Zhao, A. Ramesh, P. Sprechmann, and Y. LeCun. “Disentangling factors of variation in deep representation using adversarial training,” *advances in neural information processing systems*, pp. 5040–5048, 2016.

[11] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative adversarial text to image synthesis,” *arXiv preprint arXiv:1605.05396*, 2016.

[12] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proc. of the IEEE conference on computer vision and pattern recognition*, 2016. Article (CrossRef Link)

[13] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multiscale video prediction beyond mean square error," in *Proc. of international conference on learning representations, arXiv preprint arXiv:1511.05440*, 2016.

[14] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," *In advances in neural information processing systems*, pp. 613–621, 2016.

[15] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," *advances in neural information processing systems*, pp. 82–90, 2016.

[16] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," in *Proc. of international conference on learning representations (ICLR),* arXiv preprint *arXiv:1511.06434*, 2016.

[17] A. Makhzani, J. Shlens, N. Jaitly, and I. Goodfellow, "Adversarial Autoencoders. In International Conference on Learning Representations," *arXiv preprint arXiv:1511.05644*, 2016.

[18] P. Isola, J. Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. of 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5967 – 5976, 2017. Article (CrossRef Link)

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. of MICCAI*, pp. 234–241, 2015. Article (CrossRef Link)

[20] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," *arXiv preprint arXiv:1605.09782*, 2016.

[21] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. of the IEEE international conference on computer vision*, pp. 2223-2232, 2017. Article (CrossRef Link)

[22] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," *arXiv:1703.05192*, 2017.

[23] Y. Choi, M. Choi, M. Kim, J. W. Ha, S. Kim, and J. Choo, "StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation," in *Proc. of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8789 – 8797, 2018. Article (CrossRef Link)

[24] D. Joo, D. Kim, and J. Kim, "Generating a Fusion Image: One's Identity and Another's Shape," in *Proc. of 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1635 – 1643, 2018. Article (CrossRef Link)

[25] D. Berthelot, T. Schumm, and L. Metz, "Began: Boundary equilibrium generative adversarial networks," *arXiv preprint arXiv:1703.10717*, 2017.

[26] R. Jain, A. Isaksen, C. Holmgård, and J. Togelius, "Autoencoders for level generation, repair, and recognition," in *Proc. of the ICCC workshop on computational creativity and games*, 2016.

[27] T. Xue, J. Wu, K. Bouman, and B. Freeman, "Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks," *advances in neural information processing systems*, pp. 91-99, 2016.

[28] S. E. Reed, Y. Zhang, Y. Zhang, and H. Lee, "Deep visual analogy-making," *advances in neural information processing systems*, pp. 1252-1260, 2015.

[29] A. Summerville, M. Guzdial, M. Mateas, and M. O. Riedl, "Learning player tailored content from observation: Platformer level generation from video traces using lstms," in *Proc. of artificial intelligence and interactive digital entertainment conference*, 2016.

[30] S. Snodgrass, and S. Ontanon, "Learning to generate video game maps using markov models," *IEEE transactions on computational intelligence and AI in games*, Vol. 9, No. 4, pp. 410-422, 2016. Article (CrossRef Link)

[31] L., Horsley and D. Perez-Liebana, "Building an automatic sprite generator with deep convolutional generative adversarial networks," in *Proc. of 2017 IEEE conference on computational intelligence and games (CIG)*, pp. 134-141, 2017. Article (CrossRef Link)

[32] H. H. Kim, “Content generation using variational auto-encoder,” in *Proc. of Nexon developer conference (NDC)*, 2017.
[33] C. Beckham, and C. Pal, “A step towards procedural terrain generation with GANs,” *arXiv preprint arXiv:1707.03383*, 2017.
[34] E. Giacomello, P. L. Lanzi, and D. Loiacono, “DOOM level generation using generative adversarial networks,” in *Proc. of 2018 IEEE games, entertainment, media conference (GEM)*, pp. 316-323, 2018. Article (CrossRef Link)

Seung Jin Hong received an MS degree at Hongik University in 2018. His research interests include deep learning algorithms for game development and autonomous agent control.

Sookyun Kim received PhD. in Computer Science & Engineering Department from Korea University, Seoul, Korea, in 2006. He joined Telecommunication R&D center at Samsung Electronics Co., Ltd., from 2006 and 2008. He is now a professor at Department of Game Engineering at Paichai University, Korea. He has published many research papers in international journals and conferences. His research interests include multimedia, pattern recognition, image processing, mobile graphics, geometric modeling, interactive computer graphics, and virtual reality.

Shinjin Kang received an MS degree at Korea University in 2003. After graduation, he joined Sony Computer Entertainment Korea (SCEK) as a video game programmer. From 2006, he has worked at NCsoft Korea as a lead game designer from 2009. He received a PhD degree in Computer Science and Engineering at Korea University in 2011. And he is now a professor at the school of games in Hongik University.