GAN-Assisted YUV Pixel Art Generation

Zhouyang Jiang and Penny Sweetser

The Australian National University, Canberra ACT 2601, Australia

Abstract. Procedural Content Generation (PCG) in games has grown in popularity in recent years, with Generative Adversarial Networks (GANs) providing a promising option for applying PCG for game artistic asset generation. In this paper, we introduce a model that uses GANs and the YUV colour encoding system for automatic colouring of game assets. In this model, conditional GANs in Pix2Pix architecture are chosen as the main structure and the YUV colour encoding system is used for data preprocessing and result visualisation. We experimented with parameter settings (number of epochs, activation functions, optimisers) to optimise output. Our experimental results show that the proposed model can generate evenly coloured outputs for both small and larger datasets.

Keywords: Generative Adversarial Networks · Art Generation · Video Games · Procedural Content Generation · Pixel Art.

1 Introduction

Over the last few decades, the scale and complexity of video games have significantly increased in both technical and artistic terms. Due to advances in video game and computing technology, such as central and graphics processing, the cost of game development has rapidly grown in order to provide players with a high-quality game experience and graphics [6]. Many modern games include thousands of art assets for models, actions, rigs, and so on [2]. Moreover, the demand for amount and quality of artistic assets in games will continue to grow, driven by consumer demand and further advances in technology. To reduce the cost of development while maintaining (or improving) the quality, some of the work of generating artistic assets can be completed by artificial intelligence. In order to solve the problems of exponential growth in demand for game assets and to automate some of the tedious manual labour of asset creation, the concept of Procedural Content Generation (PCG) for games has gained increasing attention [7].

When applying PCG for game artistic asset generation, Generative Adversarial Networks (GANs) are a good option. As the assets for a specific game usually have similar features, with the help of the generator and discriminator in GANs competing with each other, the model can learn from a set of training data and generate new data with the same characteristics as the training data. Among all the procedures for producing game assets, image colourisation for game sketches

is a typical tedious manual task. Using *Trajes Fatais: Suits of Fate*, a 2D pixel art fighting game [13] as an example, the time involved in making the sketch for a character is only 10 minutes on average. However, it takes an hour to complete the colourisation task for the sketch.

In this paper, we propose a new PCG model for automatically colouring game assets with GANs. With the help of deep learning, the aim of this project is to develop a model with GANs and Convolutional Neural Networks (CNNs) to generate 3-Channel game assets from YUV grayscale pixel art that can fit a given game theme. Specifically, the model is based on a modified version of Pix2Pix architecture [9]. The YUV colour transfer algorithm is used in data preprocessing to improve performance. After combining these two algorithms, we found that the generated outputs satisfy the requirements of generating assets that have similar characteristics and can fit the chosen game theme. Furthermore, with the help of qualitative and quantitative analysis, we optimised the parameters and model structures to provide improved performance.

2 Background

The focus of this project is on developing a model with GANs and CNNs to generate 3-Channel game assets from YUV grayscale pixel art that can fit the whole game theme. In this section, we introduce YUV colour space, GANs and Pix2Pix architecture. Additionally, a variant of the Pix2Pix architecture is also considered as a reference.

2.1 YUV colour encoding system (YUV colour space) with GANs

YUV colour space is a colour encoding system that focuses on human perception. Compared with the original RGB colour encoding system, it can reduce transmission errors [14]. It is not the first time that YUV colour space has been used in GANs. The YUV colour encoding system has already been added to GANs for the thin cloud removal task [15] and to transform LR SDR videos into UHD HDR format [17]. In contrast, the literature on using YUV colour space with GANs for automatic colourisation is notably scarce. There is one previous study about using DCGAN in the YUV colour encoding system for image colourisation for remote sensing images [16]. However, remote sensing images are different from game artistic assets, which are more scattered and irregular. Additionally, compared with DCGAN, Pix2Pix architecture can provide more detailed information. Therefore, investigating the use of the YUV colour encoding system in Pix2Pix GANs architecture is worthwhile.

2.2 Pix2Pix architecture

Pix2Pix is a GAN network, developed by Isola et al [9], which has a U-Net generator, a patch-based discriminator, and a combined objective function. Because the Pix2Pix architecture uses U-Net to improve the details of the output

and a patch-based discriminator to classify small patches separately, it has been widely used for PCG of images. Lewis [8] used Pix2Pix to build an automatic pixel-art sprite generator, which generates the whole pixel-art character, combining sketch-making and colourisation together. Although this approach is more efficient for generating the game assets, the edges and shapes of assets will be blurred. There will be noise in outputs and details will be missing since there are no sketches of edges. Therefore, in our research, we focus on the colourisation task from the sketches. Compared with Lewis's research, the goal of our model is to assist game artists in performing a tedious manual task, instead of replacing the role of the artist.

2.3 Variants of the Pix2Pix architecture

There are several variants of the Pix2Pix architecture. For colourisation tasks, Ygor et al. [13] built a modified version of the Pix2Pix architecture. To adjust the model to be suitable for colourisation tasks, they added another decoder-discriminator pair to the U-Net architecture, so that it can learn more semantic-rich features. Additionally, the original activation function for downsampling steps, LeakyReLU was changed to ELU and the original combined objective function was changed to L2 loss to converge faster and get better results. However, as Ygor's model is used to automatically colour the whole action sprites, it focused on the same characters with different actions and some modifications of that model are not suitable for our single image task. Additionally, there are also better options for parameters for our tasks, which we describe in our results and analysis section. Furthermore, among all the variants of the Pix2Pix architecture used for colourisation tasks, models that specifically colour pixel art are notably scarce. However, as there is a U-Net architecture with skip-connections in the Pix2Pix architecture, it can help to pass the location information of pixels and produce pixel-art with a clearer edge.

3 Method

Our proposed model is based on GANs and the YUV colour encoding system. We chose a conditional GAN as the main structure, as illustrated in Figure 1. Conditional GANs [10] map grayscale inputs to coloured outputs. The discriminator, D, tries to classify between the fake outputs generated by the generator and the real original images. The generator, G, learns to synthesise outputs that are close to the real images and cannot be distinguished from the originals by the discriminator, D. The difference between unconditional GANs [3] and conditional GANs is that, for conditional GANs, both the generator and discriminator are able to receive the grayscale inputs, while, in unconditional GANs, the discriminator cannot observe the inputs. Specifically, the conditional GAN solution is built on a modified Pix2Pix architecture. The YUV colour encoding system is used for data preprocessing and results visualisation. In this section, we describe the details of our conditional GAN (Pix2Pix architecture) and the YUV colour encoding system.

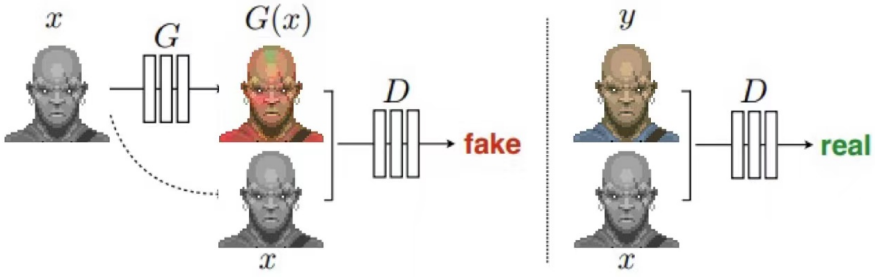

Fig. 1. Structure of conditional GAN (image based on Isola et al.[9]). Both the generator, G, and discriminator, D, are able to receive the grayscale inputs.

3.1 Datasets

In all experiments, two datasets were used: Ultimate Fantasy Sprites from Oryx Design Labs and Pokémon Images dataset from Kaggle.

Ultimate Fantasy Sprites from Oryx Design Labs Ultimate Fantasy Sprites contain 36 PNG pixel-art portraits, which are in the same theme and suitable for RPG game making. The pixel size for each image is 48x48. This is a fairly small dataset that can train the model efficiently.

Pokémon Images dataset from Kaggle Pokémon Images dataset originally contains 1634 PNG images. However, only 870 images of the dataset are the actual characters in the Pokémon games. The remaining 764 images are gathered from fan-made games. Therefore, to ensure all the pixel-art images are in the same theme, only the actual characters were kept, while others have been removed. Each image is a front-facing battle PNG pixel art in 160x160 pixel size. Since this dataset is larger, the details of the outputs and the difference of using each parameter and structure can be shown more clearly.

3.2 Data Preprocessing and Results Visualisation

During data preprocessing, the YUV colour encoding system is used to convert all RGB raw image data into YUV images. Then, the Y channel is separated as the single-channel grayscale image inputs from YUV images. For results visualisation, the YUV colour encoding system is used to turn the U and V values combined with Y inputs back to RGB values that can be shown as images.

Data Preprocessing During data preprocessing, the method of converting all RGB images into YUV images can be expressed as:

$$Y = W_R R + W_G G + W_B B = 0.299R + 0.587G + 0.114B \tag{1}$$

$$U = U_{max}\frac{B-Y}{1-W_B} \approx 0.492(B-Y) \tag{2}$$

$$V = V_{max}\frac{R-Y}{1-W_R} \approx 0.877(R-Y) \tag{3}$$

Where Y stands for the luma component, U is the blue projection, and V is the red projection. Y is used to indicate the value of brightness and U and V are the chrominance (colour) components. The reason why green is weighted most heavily in the Y channel is that people are more sensitive to green than other colours. After getting YUV values, the next step of data preprocessing is separating the Y channel as the single-channel grayscale image inputs from YUV images. Figure 2 shows several samples of original images and Y channel grayscale images after data preprocessing.

Fig. 2. Samples of original images and Y channel grayscale images after preprocessing.

Results Visualisation The YUV colour encoding system is also used for results visualisation, turning YUV 3-Channel images back to RGB images. The method of converting all YUV images to RGB images can be expressed as:

$$R = Y + V\frac{1-W_R}{V_{max}} = Y + 1.14V \tag{4}$$

$$G = Y - U\frac{W_B(1-W_B)}{U_{max}W_G} - V\frac{W_R(1-W_R)}{V_{max}W_G} = Y - 0.395U - 0.581V \tag{5}$$

$$B = Y + U\frac{1-W_B}{U_{max}} = Y + 2.033U \tag{6}$$

Another task during results visualisation is to show the single Y-channel grayscale image and RGB 3-Channel image in the same whole image. The way to visualise the single-channel grayscale image in a 3-Channel structure is by copying the single Y value to all three RGB channels. Figure 3 shows several samples of visualised results.

3.3 Modified Pix2Pix Architecture

Pix2Pix is a GAN network that has a U-Net generator, a patch-based discriminator, and a combined objective function. These three parts are described in the following sections.

Fig. 3. Samples of visualised results (left-to-right: input, target, predicted).

U-Net Generator The U-Net generator has the standard encoder-decoder architecture, except that it has skip connections between mirrored layers in the encoder and decoder stacks (see Figure 4). The skip connection concatenates channels at the encoder layer and decoder layer. The reason why the skip connection is used is that it can provide the pixel-level information for decoder layers, from the encoder layers to reconstruct details. Specifically, in our pixel-art colourisation task, the pixel-level information that skip connections share from encoder layers to decoder layers is the location information of every edge, contour, and area in the grayscale input image.

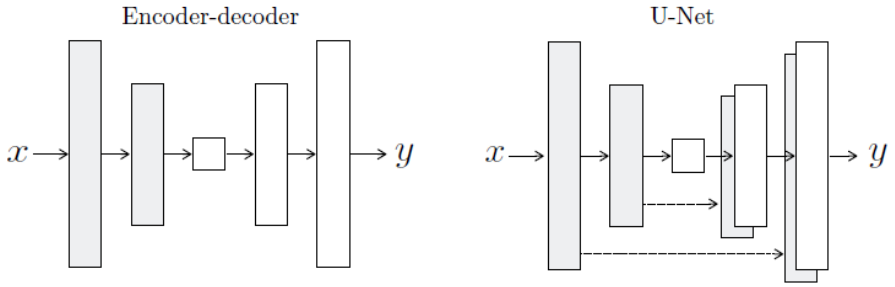

Fig. 4. Difference between the normal encoder-decoder architecture and U-Net architecture: U-Net has skip connections (image from [9]).

Patch-based Discriminator A patch-based discriminator is used for this modified Pix2Pix architecture. In contrast to traditional discriminators that classify the whole image as real or fake directly, the patch-based discriminator first separates the whole image into small N x N patches. Then, it tries to classify whether each patch is real or fake separately. After running this discriminator convolutionally, it will sum and average all responses to provide the output. The reason why this patch-based discriminator is beneficial is that small patches can let the training process focus on the interesting parts of the image, instead of the whole image.

Combined Objective Function For a conditional GAN network, the objective function can be described as:

$$L_{CGAN}(G, D) = E_{x,y}[logD(x, y)] + E_{x,z}[log(1 - D(x, G(x, z)))] \quad (7)$$

However, for this modified Pix2Pix architecture, adding a traditional L1 loss to the original objective function can improve the performance [12], which can be described as:

$$L_{L1}(G) = E_{x,y,z}[||y - G(x, z)||_1] \quad (8)$$

Therefore, the final objective function can be expressed as:

$$G^* = \arg\min_G \max_D L_{CGAN}(G, D) + \lambda L_{L1}(G) \quad (9)$$

Technical Details and Parameters To build this project in Python, the whole model was implemented with the Keras [5] framework, based on the TensorFlow [1] library. For the data preprocessing, NumPy [11] and OpenCV [4] libraries were used as the two main tools.

In our U-Net module, there are eight down sampling steps and seven up sampling steps. From previous research [9][13], ReLU and ELU are used as the activation functions for the down sampling steps. However, when testing different activation functions in our model (shown in results and analysis section), LeakyReLU provided the best performance, instead of ReLU and ELU. Therefore, LeakyReLU is chosen as the activation function for down sampling steps. As for up sampling steps, ReLU is chosen as the activation function. For the patch-based discriminator, the activation function is also LeakyReLU.

For optimisation when training the model, both the default Adam optimiser and SGD optimiser can provide good performance. The difference between the output from the two optimisers will be discussed in the results section. To get the best results, the number of epochs when using Adam as the optimiser is set to be 50, while it is 200 using SGD.

All the experiments were done on a PC with a GeForce RTX 2080 Ti GPU. With this setup, it takes around 2 to 5 hours to finish training each model. The time of training is mainly related to the number of epochs and the size of the dataset.

4 Results and Analysis

Before comparing different parameters and structures of the model, Figure 5 shows several examples of results from our model with optimised parameters for reference. We found that with increased number of epochs, the output will gain more detailed colours. Although some colours are different from the original images, the generated images can still fit the theme most of the time.

Fig. 5. Examples of results with optimised parameters, trained after after 10 (top-left), 25 (lower-left), 50 (top-right) and 200 (lower-right) epochs (left-to-right: input, target, predicted).

Fig. 6. From left to right, after and before using RGB colour encoding systems, after and before using YUV colour encoding systems to do the data preprocessing.

4.1 Colour encoding systems in data preprocessing

As discussed, different colour encoding systems (YUV, RGB) will lead to different data preprocessing results. In Figure 8, the first two images are the items after and before using RGB colour encoding systems for data preprocessing. The second two images are the items after and before using YUV colour encoding systems for data preprocessing.

Compared with RGB colour encoding, we found that the results using YUV colour encoding for data preprocessing contained more detail and had a larger contrast ratio. This result is due to the Y channel combining all three RGB channels' information.

Fig. 7. Sample results of using RGB colour encoding systems to inputs and outputs (left-to-right: input, target, predicted).

Fig. 8. Sample results of using YUV colour encoding systems to inputs and outputs (left-to-right: input, target, predicted).

4.2 Colour encoding system formats as inputs and outputs

We found that compared with RGB colour encoding, the YUV outputs were more detailed and evenly coloured, while some light parts in the RGB results were the same as the white background. There are two reasons that can cause this difference. First, as mentioned previously, the Y channel combines all three RGB channels' information. In the colourisation task, more information on the actual colours will provide better performance. On the other hand, when using YUV images as inputs and outputs, only U, V need to be predicted instead of the three RGB channels. Because the Y channel is always the same, which means, when using YUV colour encoding for both input and output, the colourisation task will be simplified from predicting the three RGB channels to predicting only the two U and V channels.

Fig. 9. Sample results of using ELU as activation function in down sampling steps (left-to-right: input, target, predicted).

Fig. 10. Sample results of using LeakyReLU as activation function in down sampling steps (left-to-right: input, target, predicted).

4.3 Activation functions in down sampling steps

We found that using ELU as the activation function in down sampling led to similar colours for different inputs (see Figures 9 and 10). Specifically, all the results of using ELU as activation function in down sampling steps use orange as their main colour. However, each character has its own settings. Therefore, all the different outputs with the same colours are not acceptable. Compared with ELU, when the activation function in down sampling steps was set to be LeakyReLU, the results were improved. With LeakyReLU, the outputs were in different colours, instead of the same orange colour. The reason why ELU provided results with similar colours might be the vanishing gradient problem. Therefore, LeakyReLU was chosen as the activation function in down sampling steps.

Fig. 11. Results for Adam as optimiser, training after 50 (top) and 200 epochs.

Fig. 12. Results for SGD as optimiser, training after 50 (top) and 200 epochs.

4.4 Optimisers

We found that when using Adam as the optimiser (see Figure 11), after 50 epochs of training, the character was given the correct blue colour as the target image. However, when using SGD as the optimiser (see Figure 12), after 50 epochs the

performance was not as good as with Adam, with very limited colour in the ouptput. However, when we continued training the model with Adam and SGD for 200 epochs, things turn out differently. For Adam, the results after 50 epochs and results after 200 epochs were almost the same (top parts were given the correct colours). As for SGD, after 200 epochs the model gave the correct body colour for the output, which was even better than the results after 200 epochs with Adam. In summary, according to these experimental results, we concluded that both Adam and SGD were good options for this model, each with different advantages. The Adam optimiser converged faster and provided results more efficiently, while SGD needed more time to get the results, but gave more details and provide better final outputs.

5 Conclusions and Future Work

In this paper, we proposed a model using GANs and CNNs to generate 3-Channel game assets from YUV grayscale pixel art. This model uses the YUV colour encoding system for data preprocessing and result visualisation, and the Pix2Pix structure as the main GAN architecture. Based on our experiments, our model achieved the best performance when the YUV colour encoding system was applied and LeakyReLU was the activation function in down sampling steps. Both Adam and SGD can be used as the optimiser for the model and can achieve good outputs.

Although the proposed model can automatically colour game assets, it sometimes misidentified the character settings. Using Pokémon Images dataset as an example, as shown in Figure 13, the model sometimes misidentifies the types of the characters. Specifically, the first character is a dual-type Fire/Ground character. However, the model fails to identify that correctly. On the contrary, it colours it in blue as water or ice type. The same applies to the other character. This will influence the accuracy of the final outputs. Therefore, as future work, it would be interesting and beneficial to extend this model to be able to receive the character setting information with the grayscale inputs. One of the possible solutions is to adjust the model to consider the labeled data so that the model can not only observe the image, but also gain more information from the labels.

Fig. 13. Results with misidentified types (left-to-right: input, target, predicted).

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: OSDI. vol. 16, pp. 265–283 (2016)
2. Baker, C., Schleser, M., Molga, K.: Aesthetics of mobile media art. Journal of Media Practice **10**(2-3), 101–122 (2009)
3. Biswas, S., Rohdin, J., Drahanskỳ, M.: Synthetic retinal images from unconditional gans. In: 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). pp. 2736–2739. IEEE (2019)
4. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
5. Chollet, F., et al.: Keras (2015), https://github.com/fchollet/keras
6. Folmer, E.: Component based game development–a solution to escalating costs and expanding deadlines? In: International Symposium on Component-Based Software Engineering. pp. 66–73. Springer (2007)
7. Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A.: Procedural content generation for games: A survey. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) **9**(1), 1–22 (2013)
8. Horsley, L., Perez-Liebana, D.: Building an automatic sprite generator with deep convolutional generative adversarial networks. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG). pp. 134–141. IEEE (2017)
9. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1125–1134 (2017)
10. Lan, H., Initiative, A.D.N., Toga, A.W., Sepehrband, F.: Three-dimensional self-attention conditional gan with spectral normalization for multimodal neuroimaging synthesis. Magnetic Resonance in Medicine (2021)
11. Oliphant, T.: NumPy: A guide to NumPy. USA: Trelgol Publishing (2006–), http://www.numpy.org/, [Online; accessed ¡today¿]
12. Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A.A.: Context encoders: Feature learning by inpainting. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2536–2544 (2016)
13. Serpa, Y.R., Rodrigues, M.A.F.: 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 (SBGames). pp. 182–191. IEEE (2019)
14. Wang, G., Fu, R., Sun, B., Lv, J., Sheng, T., Tan, Y.: Comparison of two types of color transfer algorithms in yuv and lab color spaces. In: AOPC 2017: Optical Sensing and Imaging Technology and Applications. vol. 10462, p. 104622V. International Society for Optics and Photonics (2017)
15. Wen, X., Pan, Z., Hu, Y., Liu, J.: Generative adversarial learning in yuv color space for thin cloud removal on satellite imagery. Remote Sensing **13**(6), 1079 (2021)
16. Wu, M., Jin, X., Jiang, Q., Lee, S.j., Liang, W., Lin, G., Yao, S.: Remote sensing image colorization using symmetrical multi-scale dcgan in yuv color space. The Visual Computer pp. 1–23 (2020)
17. Zeng, H., Zhang, X., Yu, Z., Wang, Y.: Sr-itm-gan: Learning 4k uhd hdr with a generative adversarial network. IEEE Access **8**, 182815–182827 (2020)