Towards Machine-Learning Assisted Asset Generation for Games: A Study on Pixel Art Sprite Sheets

Ygor Rebouças Serpa and Maria Andréia Formico Rodrigues
Programa de Pós-Graduação em Informática Aplicada (PPGIA)
Universidade de Fortaleza (UNIFOR)
Fortaleza-CE, Brazil

***Abstract*—Game development is simultaneously a technical and an artistic challenge. The past two decades have brought many improvements to general-purpose game engines, reducing the new games development effort considerably. However, the amount of artistic work per title has continuously grown ever since, as a result of increased audience's expectations. The cost of asset-making is further increased based on the aesthetics chosen by the design team and the availability of professionals capable of understanding the nuances of the specific visual language chosen. In this paper, we dig into the topic of deep-learning assets generation to reduce the costs of the asset making pipeline, a major concern for game development teams. More specifically, we tackle the challenge of generating pixel art sprites from line art sketches using state-of-the-art image translation techniques. We set this work within the pipeline of *Trajes Fatais: Suits of Fate*, a 2D pixel-art fighting game inspired by the late nineties classics of the fighting genre. The results show that our deep-learning assets generation technique is able to generate sprites that look similar to those created by the artists' team. Moreover, by means of qualitative and quantitative analyses, as well as character designers evaluation, we demonstrate the similarity of the generated results to the ground truth.**

***Keywords*-Deep Learning, Generative Adversarial Networks, Asset Generation, Procedural Content Generation, Qualitative and Quantitative Analyses, Character Designers Evaluation**

I. Introduction

Game development is simultaneously a technical and an artistic challenge [1]. Modern AAA games feature millions of lines of code and thousands of assets, such as models, textures, rigs, animations, sounds, shaders and others [2]. The past two decades have brought many improvements to general-purpose game engines, reducing the new games development effort considerably [3]. However, the amount of artistic work per title has continuously grown ever since, as a result of the increased audience's expectations in terms of quality and visual art variability. This trend is shifting resources towards tools to substantially help artists bring their visions to life.

The overall cost of asset-making is further increased based on the visual language chosen by the design team. Some aesthetics, such as pixel-art [4], need a nearly prohibitive number of hand-crafted assets. In addition, only a small group of artists specialize on this art form, which further increases the associated costs of producing such games. Recent advances on computer generated imagery foster the question of whether the current state-of-the-art can be leveraged to automatically generate art content in place of human artists or, at least, to automate tedious manual labour.

In this paper, we dig into the topic of deep-learning assets generation to reduce the costs of the asset making pipeline, a major concern for gaming teams of character designers & animators. More specifically, we tackle the challenge of generating pixel art sprites from line art sketches using state-of-the-art image translation techniques. We set this work within the pipeline of *Trajes Fatais: Suits of Fate*, a 2D pixel-art fighting game inspired by the late nineties classics of the fighting genre. On average, for a new character release, 450 sprites are needed, each taking 10 minutes to be sketched and one hour to be completed. For more complex and visually appealing characters, these timings might become even greater. Therefore, using deep learning asset generation, our goal is to reduce the amount of time needed to release a new character by turning sketches into semi-final sprites that can be finalized in a shorter time. Using a variant of the *Pix2Pix* architecture [5], we show that our deep-learning assets generation technique is able to generate sprites that look similar to those created by the artists' team. Moreover, by means of qualitative and quantitative analyses, as well as character designers evaluation, we demonstrate the similarity of the generated results to the ground truth.

II. Related Work

The deep learning boom has triggered a series of innovations on several key areas of computer vision [6], [7]. Among the several proposed techniques, generative adversarial networks (GANs) [8] and variational auto-encoders (VAEs) [9] dominate the recent image generation literature, with works mainly spanning themes such as face generation [10], image interpolation [11], image segmentation [12], style-transfer [13] and image translation [5], [14]–[16]. The problem of mapping sketches to completed sprites can be understood as a case of the paired image translation problem [5]. Formally, it corresponds to the task of converting images from domain A to domain B. Additionally, it is a paired problem, as the two domains have a strong relationship. In

2159-6662/19/$31.00 ©2019 IEEE
DOI 10.1109/SBGames.2019.00032

this work, the input sketch and the generated sprite share the same pose and contour. Beyond the scope of this work, the unpaired problem is more difficult and features domains that have only semantic relationships [14].

The first generally applicable framework for paired image translation was developed by Isola *et al* [5] and uses a GAN strategy. The GAN model relies on two competing networks: the generator, which produces the model's output and, the discriminator, which learns to differentiate between real and generated images. During training, the generator strives to produce results that are indistinguishable from real images to the discriminator network [8]. The work of Isola *et al* particularly features a U-Net based generator [12], a Patch-based discriminator and the L_1 loss function. Nowadays, this work is commonly known as the *Pix2Pix* architecture and its main ideas have been influencing many other works, which develop on the paired [15], unpaired [14] and multi-domain [16] settings.

In contrast, the literature on automated asset generation is notably scarce. To the best of our knowledge, a few authors have tackled the automatic sprite generation problem. Two examples are Horsley *et al* [17] and Xue *et al* [18]. The former attempt to generate 8-bit sprites from random noise using a simple GAN model, achieving limited results; whereas the latter seek to generate plausible next-frame sprites for a given animation sequence, using a probabilistic model with convolutional neural networks. These authors report interesting results using several datasets, including a dataset of 8-bit animated sprites. Beyond the sprite generation literature, most of the deep learning field on games concentrates on scene generation [19], [20] and game playing [21], [22]. On the general topic of asset generation, for example, most works deal with the generation of terrain, trees, props and textures [23], [24].

We attribute the scarcity of published works dealing with automatic sprite generation and, more specifically, character generation, to the amount of detail involved in those assets and to the level of protagonism they have within games. Poorly generated character sprites would be readily noticed by users and this would negatively impact their overall perception of quality. This reasoning explains why most procedurally generated content is more focused on the background, *e.g.,* the terrain, trees, props, grass, etc [24]. Contrastingly, in this work we investigate the generation of main character sprites using image translation techniques. However, instead of attempting to produce the final quality asset directly, we aim to aid character artists and designers by creating semi-final assets which, in turn, are suitable to be finished manually.

III. The Sprite Creation Pipeline

The concept of a sprite starts with a very rough sketch, made by the art director. This sketch is used to validate animation ideas and, if the design team is satisfied with the resulting animation, each sprite is re-drawn as a *line* sprite and then is handed over to be polished and finalized. Figure 1 shows two sprites in their early sketch stage. This pipeline centers most of the concept process around the art director and most of the creative drawing around the other artists. On average, sketches take less than two minutes to be drawn and their respective line art sprites take at most ten minutes to be done.

Figure 1: Two sprites in their early sketch stages.

To replicate the late nineties look, the *final* sprites are based on a 252 color palette scheme. For this, each *line* sprite has to be completed twice: first, as a *gray* sprite and then as a *color* one. The former uses up to 6 gray tones to convey the shading information and, the latter, up to 42 different colors to mark regions of the sprite, such as hair, skin, clothes, accessories, etc. Combined, they identify up to $6 * 42 = 252$ unique colors. Therefore, the *final* sprite is generated by encoding the *gray* and *color* sprites together. From left to right, Figure 2 illustrates the entire life cycle of a sprite. This procedure streamlines the artists work as it relaxes the 256 colors limitation into two distinct and intuitive problems. Also, it greatly simplifies the creation of skins, as all shading and coloring data are kept separate. We exemplify the system flexibility to create skins in Figure 3.

Out of the entire life cycle, painting the *gray* entire sprite is the most time consuming step, taking around 30 to 40 minutes to complete. Actually, this step can easily spend more time than all the others combined. The task complexity also varies with the character, some have more intricate shading patterns (due to more fashionable clothing, hair styles and muscles), while others have large and smooth surfaces, which are considerably easier to draw and colorize. Inversely, color sprites are considerably easier to generate as most of them are formed by a handful of colors, that is, being this process equivalent to flood fill the individual areas of the sprite (legs, arms, hair, etc.), than actually drawing the entire shapes. For example, a color sprite may take from 5 to 15 minutes. Some characters, however, have many small regions to be painted and can take longer to generate. Figure 4 shows easy and hard to drawn characters, respectively, in their *gray* and *color* variants. Unfortunately, pin-pointing the exact times taken for each step and sprite is a difficult task, as each artist has its own work-flow and, very often, they work on several sprites simultaneously.

Figure 2: From left to right, the rough sketch, *line* sprite, *gray* sprite, *color* sprite and *final* sprite, composed of grayscale lookup indices.

Figure 3: Several skins of the same character using the *index* sprite.

Figure 4: Several characters in their *gray* and *color* variants. The two characters on top are harder to shade and colorize due to their complex clothing and muscles. On the other hand, the two on bottom are more easier cases, since they have many smooth surfaces and, particularly, the character located in the leftmost, which has large areas with same colors.

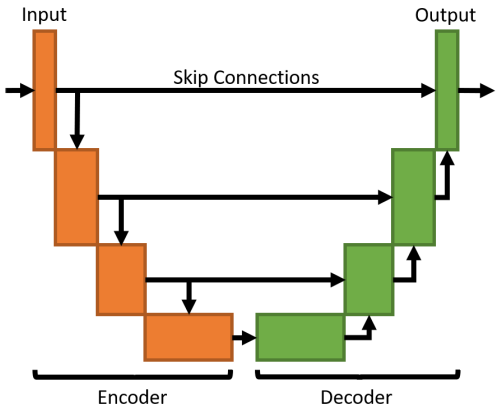

Figure 5: Overview of the U-Net architecture. The orange colored encoder network (on the left) and the green colored decoder network (on the right) are two components linked through standard and skip connections.

IV. Our Deep-Learning Assets Generation

Within the sprite painting pipeline, we frame our problem as two paired image translation sub-problems: converting line art sprites into gray sprites and into color sprites. This division mimics how the artists work and simplifies the learning task into two pixel-wise categorical classification problems, with 6 and 42 classes, respectively. In this work, we call these problems the **gray** and **color** problems.

We build our solution on the *Pix2Pix* architecture [5], which is a GAN model based on a U-Net generator, a patch-based discriminator and the L_1 loss function. These elements are detailed as follows:

- **U-Net Generator:** This network consists of a encoder-decoder model that has skip connections between the corresponding encoder and decoder layers. The reasoning behind this architecture is that the encoder-decoder model is concerned with understanding the input domain and translating it to the output domain in a semantic way, while the skip connections provide the necessary low-level information from the previous

layers to reconstruct details, such as the background or object contours. This network, shown in Figure 5, is a common basis for image-to-image models.

- **Patch-based Discriminator:** While traditional discriminators learn to classify the whole image as real or fake, the patch-based discriminator classifies small patches separately. This change allows the training process to focus more on the fake-looking areas, and leave the remaining ones unchanged. This can be accomplished by casting the discriminator as a segmentation network, which outputs a lower-resolution image of per-pixel classifications. Usually, the patch size is chosen to be $1/8^{th}$ of the image size.
- L_1 **Loss Function:** Equation 1 shows the loss function which is computed as the mean ($\mathbb{E}$) of the absolute differences between the correct (y_{true}) and generated ($y_{generated}$) outputs [25].

$$L_1 = \mathbb{E}(|y_{true} - y_{generated}|) \tag{1}$$

This loss function is preferred over the more common L_2 loss on generative networks as it encourages less blurring of the output. The importance of this loss function to the final model resides in capturing low frequency information from the output domain.

Combined, these elements compose a complete image translation framework in which the L_1 loss and the patch-based discriminator are used to train the U-Net generator. The complete training procedure follows the Conditional GAN formulation and consists in: (1) training the discriminator to classify real images as real and generated images as fake, and (2) training the generator over the L_1 loss and over the results of the discriminator classification. These steps are performed in alternation and, after enough training, they tend to converge to a scenario where the generator results are good enough to both minimize the L_1 loss and maximize the discriminator confusion. In other words, the generator results converge to a state in which the discriminator is no longer capable of distinguishing real from fake images.

A. Changes to the Pix2Pix architecture

In contrast to the original *Pix2Pix* architecture, we have two image translation problems to solve: the *gray* and *color* problems, the former in the grayscale color-space, and the latter in the RGB color-space. To more efficiently solve them, we change the U-Net architecture to have two decoder-discriminator pairs, one for each problem. This way, we have a single network to solve both *gray* and *color* problems. We found that this strategy improves our results, as it helps the encoder network to learn more semantic rich features. For instance, both decoders have to identify the character's arms to colorize and shade them. With the shared encoder, there are two distinct training signals for the encoder to learn to identify the arms and embed this information within its features. In general, this multi-objective principle is widely regarded as beneficial to neural networks.

Regarding the architecture, for the down sampling steps, we have included an additional convolution to further elaborate on the down sampled features, and have used the *ELU* activation function, which yielded better results than the originally used LeakyReLU [26]. With regard to the training procedure, we have used the default Adam optimizer with a learning rate of 0.001 and the L_2 loss function shown in Equation 2, which is defined as the mean ($\mathbb{E}$) of the squared differences between the correct (y_{true}) and generated ($y_{generated}$) outputs [25].

$$L_2 = \mathbb{E}((y_{true} - y_{generated})^2) \tag{2}$$

Experimentally, we obtained a faster convergence and better results when using the L_2 loss function.

B. Technical Details

All trained models in this work were implemented with the Keras framework [27] on top of the TensorFlow library [28]. The encoder module has seven downsample steps and each decoder module has seven upsampling steps. Each downsample step is a 3x3 convolution with stride 1, followed by a 4x4 convolution with stride 2, and a batch normalization step. The reasoning behind using a 4x4 convolution with stride 2, instead of a max-pooling layer, is that this allows the network to learn a custom downsampling function instead of using a fixed one. Empirically, this has been found to yield improved results. For the upsampling steps, a 2x nearest-neighbour resize is applied to the input, followed by a 4x4 convolution with stride 1, the ReLU activation, a 10% dropout regularization, a batch normalization step and finally, the resulting features are concatenated with the features from the respective downsampling step. At the end of the network, after the last upsampling steps, a 4x4 convolution with the $tanh$ activation is used to generate the final *gray* and *color* results. With regard to the network size, $f*32$ filters were used on every convolution operation, being f equals to $1, 2, 4, 8, 8, 8, 8$ on the seven downsampling steps and f equals to $8, 8, 8, 8, 4, 2, 1$ during the upsampling steps. In total, the generator has $20,592,100$ trainable parameters.

With regard to the dataset itself, all sprites may occupy up to 640x480 pixels. However, for efficiency reasons, we have cropped all sprites to a 256x256 box, which is enough to accommodate most sprites. To increase the amount of data available, we have performed limited data augmentation on our dataset. Namely, we have included horizontally flipped versions of all training sprites into our training data. Other data augmentation techniques, such as vertical flipping or rotations, have not been applied to our dataset as these would not match the actual sketches used in practice. Regarding the training duration, we have run enough epochs to show at least 10^6 sprites to the network, in batches of 8. Beyond

this point, we would inspect our results to decide if they need to be trained more, based on the visual quality of the generated sprites. We have not employed early-stopping as this technique is known for not working well in practice with GANs, as the main source of improvements in such networks is the discriminator, and not the loss function. Therefore, it is not reasonable to stop the training based on the latter.

In all experiments, a Core i7 7700k processor with 32 GB of RAM and a NVidia GTX 1060 with 6 GB of RAM were used. With this setup, training took from 2 to 6 hours, depending on the character dataset size. The inference time to generate all sprites was negligeble, taking a few seconds to complete.

V. Tests and Results

In this section, we bring out and discuss our results for the test cases designed, based on qualitative and quantitative data.

A. Character Datasets

We investigate our solution effectiveness using two distinct datasets: the *Sarah* and the *Lucy* characters, which are shown, respectively, on the left and right sides at the bottom of Figure 4.

The former is a tiny girl with a big platypus suit, and the latter is an adult woman wearing collant clothing. The *Sarah* character is useful as a benchmark as it has a large smooth, well behaved surface (*i.e.*, the platypus suit), and a highly detailed character, which occupies a small portion of the entire sprite. In particular, this dataset is incomplete, as this character has not been finished completely by the design team at the time of this writing. On the other hand, the *Lucy* character has been entirely completed and has mostly smooth features, thus, being easier to study both on terms of drawing complexity and amount of available data. These datasets have provided us with a perspective on the usefulness of the system to finish incomplete characters and a notion of the upper bound that the algorithm can reach, when there is enough data, and an easy and simple to draw character.

B. Qualitative Results

We start presenting results regarding the *Sarah* character, followed by the *Lucy* one.

Sarah character has *gray* and *color* schemes, whose sprites vary widely on level of difficulty within each pose. In total, we have 85 fully drawn *Sarah* sprites that can be used to the training in deep-learning network which, in turn, we have doubled applying horizontal flipping. Of these, 10% were selected for validation purposes. These sprites come mostly from "neutral" animations, such as idling, walking, running, jumping and ducking. At the time of this writing, the complete character had 292 *line* sprites, which left us with 207 sprites that might be drawn by our tool and used by the design team to fast track the character development. In Figure 6, we present our results on the validation set after 800 epochs of training. On the left, we show sprites that have more common poses, therefore, having many similar sprites on the training set, whereas on the right, we show more distinct sprites.

As a whole, the *gray* problem was well handled by our solution, having only minor shading issues (*e.g.*, on shadowed regions or around the platypus armpits). In turn, the *color* problem presented interesting results: the algorithm was unable to capture the correct color palette used, but was able to segment most of the sprite regions into distinct colors. We have found this result even more intriguing and inspiring, as it demonstrates that the algorithm could learn to solve the given task, but not on the given conditions. Therefore, in an unsupervised way, it learned its own set of colors to use. Also, unlike the *gray* sprites, some noise is present, particularly, on the platypus legs and girl's clothes. In some generated sprites, the amount of color noise is significant, making the sprite useless to the character designers.

Applying this solution to the remainder 207 unfinished sprites, we have obtained the results shown in Figure 7. On the first row, we show sprites that are very similar to the existing ones for training; on the second, sprites that have new-unseen poses and, finally, on the third, sprites that are completely different from the ones seen during testing time. In short, from top to bottom, we show how well our algorithm generalizes to unseen data. More specifically, for the *gray* problem, plausible shadings have been generated, even for extreme cases, such as the one shown in the third row of Figure 7. For the *color* problem, most sprites from the second and third rows are too noisy to be usable in any way.

To deeply investigate whether *color* sprites are feasible, we have conducted the following study: given the fact that the algorithm found its own set of colors, we have changed the colors of the sprites to match those colors and then retrained the algorithm. The idea behind this approach is very simple, these colors might be more suitable for the learning task as they are semantically richer. For instance, the learned colors for the platypus head and beak have a high amount of blue and varying amounts of red, while the colors used for the arms and body are mostly green, which has no similarity with the head colors. In addition, to reduce the amount of high frequency details on the color sprite, we have removed 8 colors used to mark the facial features and some clothing details, replacing them with the predominant surrounding color.

Figure 8 shows the visual outputs of the algorithm retrained for 600 epochs on the new color setup. Overall, similar results are observed: the sprites are reasonably segmented, but intense color noise exists on several key areas, such as in the girl's body and in the platypus legs. However, in this experiment, minimal deviation from the new palette occurred, which suggests that these colors are more stable

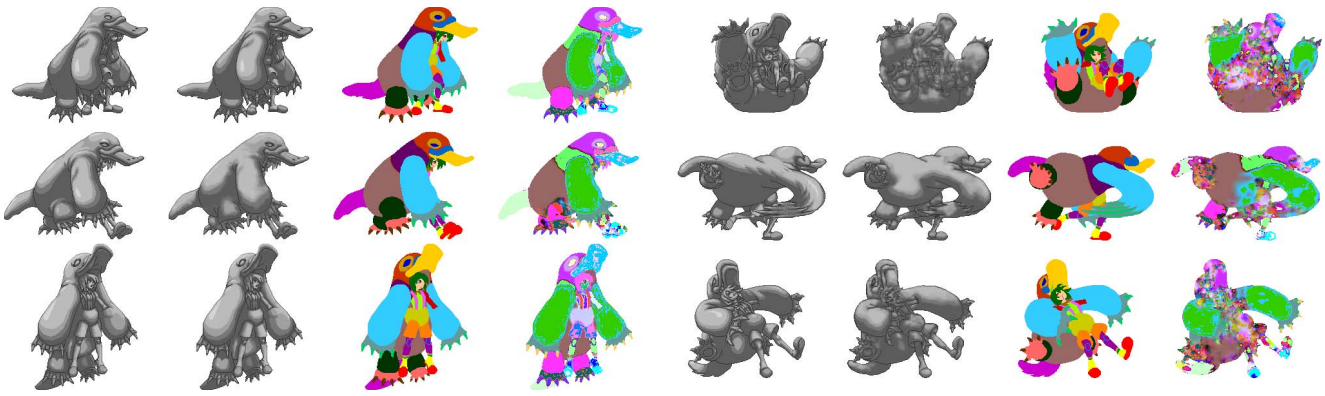

Figure 6: Results for six different sprites taken from *Sarah* animations. From top to bottom and left to right, each sprite is shown in its respective ground truth *gray*, generated *gray*, ground truth *color* and generated *color* forms.

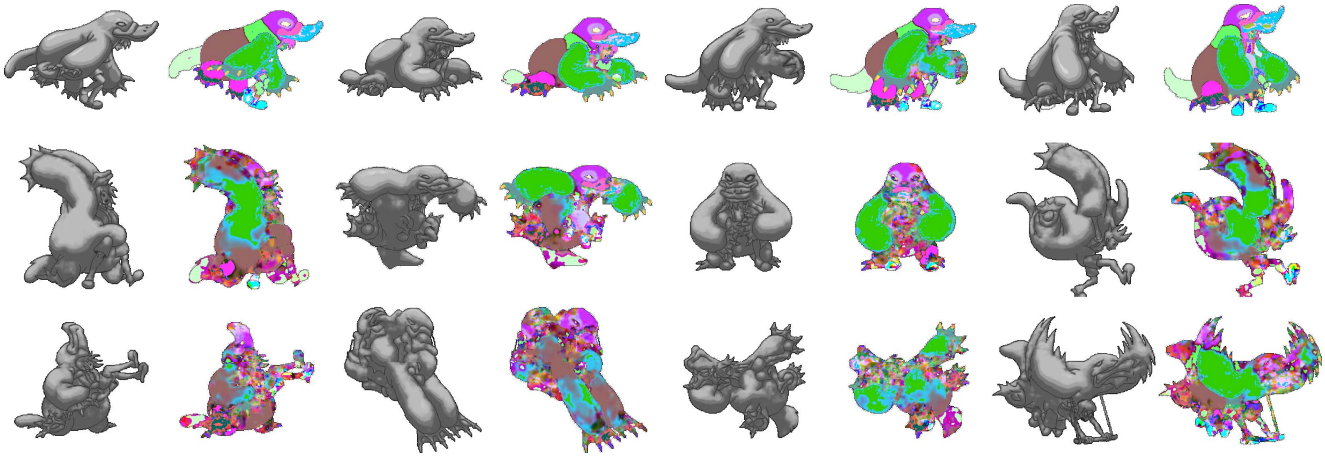

Figure 7: A sample of the generated sprites to be handed to the design team. From top to bottom row, similar poses to the ones on the training set, absent poses on the training set, and radically different poses from all others.

Figure 8: Study on the *color* problem using the algorithm's learned colors, instead of the human chosen ones.

for training. This study suggests that the *Pix2Pix* architecture for this semantic segmentation problem is unfit and that other architectures should be investigated. Also, given that changing the color palette resulted in a more stable training, we further conjecture that a study on optimal palettes might yield improved results.

To understand if a larger data base and an easier to draw character would significantly impact the effectiveness of our algorithm, we have trained it for the *Lucy* character, which has been fully drawn by the design team. The *Lucy* character has *gray* sprites which are considerably smooth, but have many curved shapes, while her *color* sprites are asymmetrical, having different colors for the left and right arms and legs. Her shading and coloring schemes are shown in Figure 9, on the bottom left corner. The complete dataset features 530 distinct sprites, which we have doubled using horizontal flipping, and then took 10% for validation purposes.

Training for 250 epochs on the *Lucy* dataset generated the results shown in Figure 9. From top to bottom, left to right, we have arranged sprites in order of pose complexity, with the last row featuring unique poses that have few similar sprites. As for the *Sarah* dataset, *gray* sprites are

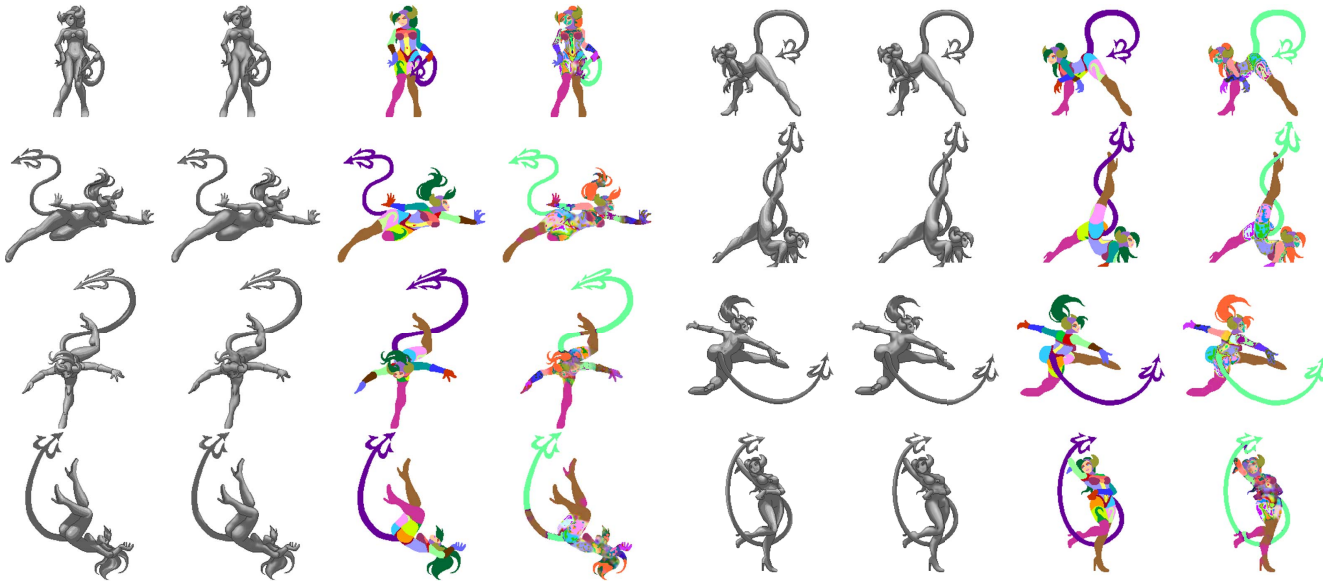

Figure 9: Results for six different sprites taken from *Lucy* animations. From top to bottom and left to right, each sprite is shown in its respective ground truth *gray*, generated *gray*, ground truth *color* and generated *color* forms.

significantly close to the ground truth, exhibiting only minor shading issues around shadowed zones, and *color* sprites yielded moderate success, being able to segment most regions, but having noticeable color noise on some problematic zones, while also not adhering to the human selected colors. It should be noted, however, that, in this case, only a handful of colors have changed, notably the hair, tail and arms, as opposed to the *Sarah* character, which had mostly all colors changed. The overall quality of the generated *gray* sprites indicate the usefulness of the solution and its capacity to learn to shade a complete character. This is an impressive result and should be noted as a milestone reached on our goal direction. With this experiment, we show that it is possible to generate high quality semi-finalized images given enough data. In other words, the upper bound on the quality of generated sprites is very close to the ground truth. Comparing this result to the *Sarah* dataset, it can be observed that with 16% of the number of sprites we had for this experiment, good results could already be obtained.

C. Quantitative Analysis

To objectively compare and evaluate the level of similarity between our simulated outputs to their respective ground truth images, we have applied the following three commonly used metrics: (1) Root-Mean Squared-Error (RMSE), (2) Mean Absolute Error (MAE) and (3) Structural SIMilarity (SSIM). These metrics measure the similarity between pairs of images, and are reported descriptively through the mean, standard deviation and quartilles information. These metrics have been calculated using the raw grayscale and RGB data and, for the RMSE and MAE scores, the average of all color channels was considered.

More specifically, the RMSE and MAE metrics are two distance metrics, that is, typical loss functions for deep learning algorithms. In particular, RMSE is the square root of the L_2 norm, the loss function used on our network, and MAE is the L_1 norm used on the original *Pix2Pix* architecture. The SSIM metric is a widely used benchmark on the image processing literature and was designed to measure the perceptual similarity of two images, thus, having a higher correlation to the human perception of similarity. A SSIM score of zero means that both images are completely different and a value of one means they are equal.

Table I shows the numerical results obtained with the three selected similarity metrics. These values were derived considering all generated images and their respective ground truth images. One can see that our quantitative analysis is consistent with our qualitative results, reinforcing that the generated *gray* sprites are visually more similar to the ground truth and performed significantly better than the *color* ones.

Also in the qualitative analysis, we compared the *color* problem using the original and the algorithm's preferred colors, obtaining worse results, but with significantly less color deviation. In Table I, we included the alternative colors results as the *Color** column. Comparing these to their original colors, we obtained worse RMSE and MAE mean errors (from 8.65 to 12.32 and from 2.89 to 3.81. respectively), but a reduced error standard deviation (from 10.23 to 7.01 and 3.37 to 2.65, respectively). This supports that the results were indeed worse, but were more stable with regard to the color deviation problem. When looking at

Table I: RMSE, MAE and SSIM metrics for the tested datasets. For the *Sarah* dataset, we include data from the original *color* problem and from the alternative *color* configuration, included as the *Color** column.

	Sarah			*Lucy*	
	Gray	**Color**	**Color***	**Gray**	**Color**
			RMSE		
μ	4.81	8.65	12.32	3.73	7.07
σ	4.91	10.23	7.01	1.68	5.02
Min	2.34	3.50	7.41	2.32	3.77
25%	2.65	4.18	8.92	2.94	5.03
50%	2.83	4.39	9.90	3.15	5.35
75%	3.99	5.21	12.42	3.50	5.74
Max	39.73	57.82	61.81	11.23	30.77
			MAE		
μ	1.78	2.89	3.81	1.01	2.01
σ	1.86	3.37	2.65	0.39	1.11
Min	0.90	1.40	2.35	0.62	1.34
25%	1.01	1.61	2.78	0.81	1.59
50%	1.10	1.69	3.04	0.88	1.65
75%	1.52	1.93	3.59	0.99	1.72
Max	17.18	22.23	24.16	2.90	8.01
			SSIM		
μ	0.97	0.84	0.88	0.99	0.94
σ	0.05	0.03	0.03	0.01	0.01
Min	0.65	0.66	0.66	0.96	0.89
25%	0.98	0.83	0.87	0.99	0.93
50%	0.99	0.84	0.89	0.99	0.94
75%	0.99	0.85	0.90	1.00	0.94
Max	0.99	0.88	0.92	1.00	0.96

the SSIM scores, we see the opposite, the alternative colors outperformed the original ones. This shows that, although the results were worse, the perceptual similarity was higher, as the color correspondence was more strictly followed on the alternative coloring scheme.

When comparing the two datasets, it can be seen that the *Lucy* dataset was superior on all observed metrics, with better average results and significantly lower standard deviations. This numerically demonstrates the degree of improvement that is possible given approximately six times more data and using an easier to draw character. More specifically, the better average mean results indicate that the overall quality can be much higher, whereas the lower standard deviations point to a more consistent quality. Also, when looking at the quartilles information, it can be noted that the minimum and maximum values reported on the *Lucy* dataset are considerably closer to the median value, whereas *Sarah* results have significant outliers.

D. Character Designers Evaluation

Throughout the development of our presented solution, the design team was regularly consulted for insights on the drawing process and for their thoughts on the artificially generated sprites. At first, the team was skeptical about the idea, but as the quality of the produced material improved, a naturally increasing interest was manifested. Inspecting the 207 generated *Sarah* sprites, there was a consensus that most of the *gray* sprites are usable and potentially time saving,

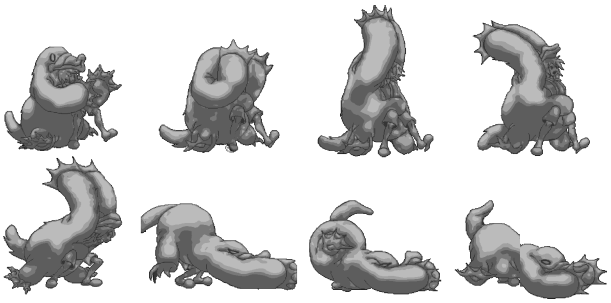

Figure 10: Example of the inconsistent shading issue for a generated sprite sequence. This problem is evident on the platypus body. Over the course of the animation, this region is often shaded either darkly or brightly.

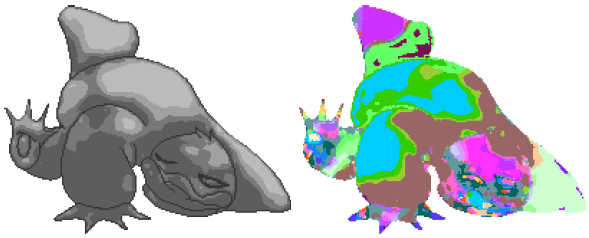

Figure 11: Example of poses that are too different from the ones present on the training set. For these cases, discontinuities on the shading of smooth surfaces and severe color noise and bleeding are common issues.

but that *color* sprites still need further improvements to be considered truly useful. Regarding quality, their feedback can be summarized into three major topics, as follows:

1) **Inconsistent Animations:** When considering animations with several frames, the shading consistency across frames is paramount for a smooth animation. Currently, we do not encode any form of animation awareness and, thus, we only optimize for consistency with the overall shading patterns of the character. In Figure 10 are shown eight sequential frames of an attack animation. In these frames, although the gener-

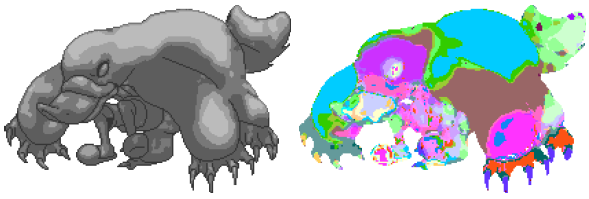

Figure 12: Example of color noise evidenced after the color rounding post processing. On the *gray* sprite, subtle noise can be seen in the sprite contour, as well as around transitions from bright to darker areas, in the arms and legs, for example. In the *color* sprite, the noise is very noticeable in the beak, girl, feet and tail.

ated shading is plausible, it changes significantly from one frame to the other. This is especially noticeable at the bottom half of the platypus. For artists, the time needed to correct these consistency errors could be significant. Also, this error is proportional to the amount of deviation from the expected shading.

2) **Difficulty on Distinct Poses:** Due to the small dataset, most of the generated sprites feature poses are significantly different than the neutral poses used for training. In some cases, this issue can lead to unexpected behaviours, such as weird highlights, severely shadowed regions and, for *color* sprites, heavy color noise. Examples of these problems are shown in Figure 11. In most cases, the artists believe that correcting these sprites would not lead to any performance improvement.
3) **Rounding Noise:** in most sprites, considering their original tones, the shading and coloring can be greatly smooth. However, when we quantize sprites to the 6 *gray* tones and to the 42 *color* tones, some color noise may appear. While this is a minor issue for *gray* sprites, this is a significant problem for the *color* ones, mainly because there are many similar colors, which can be confused when rounding, and because the generated sprites do not follow the original palette, which impairs the rounding process. To alleviate these, we have used the alternative color palette for the rounding process. However, a considerable amount of noise still remained. Examples of rounding errors are shown in Figure 12. For artists, fixing noisy images is a tedious and time consuming task.

Considering our results with the *Lucy* character, we believe that most of these problems could be solved given sufficient data. However, we conjecture that we could reduce these issues with minimal extra data by carefully selecting which sprites will be finished by the artists firstly. For instance, consider the animation shown in Figure 10. If the first and fifth sprites had been finished, they could be fed to the algorithm to potentially improve the results for the other six sprites. The same idea goes for the distinct poses problem. Given that at least one sprite is finished, the complete pose would not be as foreign as before, which would improve the algorithm's performance on similar data. Finally, for the rounding problem, we believe that more sophisticated rounding or image denoising algorithms could reduce the generated noise. Possible candidates include using neighbouring data to sort the most predominant color, using morphological operators, such as dilation and erosion, to remove single pixel noise or using a blur filter before rounding to soften the amount of color variation.

VI. Discussion and Future Work

In this work, we investigated the use of deep learning algorithms to create pixel art sprites from line art sketches, with the goal of reducing the artists workload. More specifically, we set this work within the *Trajes Fatais: Suits of Fate* pipeline, which consists of drawing a *gray* and a *color* sprite from a line art sketch. We cast our problem as an image translation problem and, for this, we have designed a variant of the *Pix2Pix* algorithm to handle the *line*-to-*gray* and *line*-to-*color* problems with a single network. In our qualitative analysis we show sprites from the validation set and how the generated sprites relate to the ground truth visually. In addition, we used different well-known objective image quality metrics to quantify the similarity of our results to the artist drawn sprites, reinforcing our qualitative results. Finally, our design team evaluated the 207 sprites generated for the *Sarah* character and provided valuable feedback on the generated material. Considering our entire analysis, we conclude that our system can generate useful sprites, most of them from the *gray* problem, and that, given more sprites, a significant improvement can be achieved.

More specifically, the study on *Lucy* sprites showed that the current architecture is able to learn how to shade a complete character, which is an impressive result, given the complexity of the task. With only 16% as many sprites, the *Sarah* dataset could still train a reasonably good generator and yield usable results. Among our key problems, the hardest is the distinct poses problem, as it requires a human level of generalization to be solved. This fact leads us to believe that the current system can be largely improved by carefully curating a dataset with enough pose variety. A possible path would be to prioritize the design team work to draw a single sprite from each animation sequence, improving the diversity of poses seen during training. Finally, it would also be worthwhile to investigate whether both datasets could be leveraged using transfer learning techniques.

As future work, several key areas of our solution could be improved to generate higher quality assets. Mainly, new architectures and techniques could be introduced in our model to improve its efficiency. Residual blocks, inception blocks, spectral normalization and custom loss functions are a few of the many techniques that could be incorporated into our model. Additionally, we believe that different settings to our problem could yield interesting results. One possible option would be to recast our problem as a pixel-wise classification, instead of being a regression model. Other idea would be to include neighbouring frames as inputs to the network to improve its intra-animation consistency. To improve the algorithms performance, we could use a weighted loss on less frequent tones or to weight animations equally, despite the amount of frames used. Finally, image processing techniques could also be investigated to improve even more the quality of generated sprites.

Acknowledgment

Ygor Rebouças Serpa and Maria Andréia Formico Rodrigues would like to thank the Brazilian Agencies

CAPES/FUNCAP and CNPq for their financial support, under grants 88887.176617/2018-00 and 439067/2018-9, respectively.

REFERENCES

[1] P. Zackariasson and T. L. Wilson, *The video game industry: Formation, present state, and future*. Routledge, 2012.

[2] E. Folmer, "Component based game development–a solution to escalating costs and expanding deadlines?" in *International Symposium on Component-Based Software Engineering*. Springer, 2007, pp. 66–73.

[3] J. Gregory, *Game engine architecture*. AK Peters/CRC Press, 2017.

[4] C. Baker, M. Schleser, and K. Molga, "Aesthetics of mobile media art," *Journal of Media Practice*, vol. 10, no. 2-3, pp. 101–122, 2009.

[5] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[7] Ygor R. Serpa, L. A. Pires, and M. A. F. Rodrigues, "Milestones and new frontiers in deep learning (accepted for publication)," in *Tutorials Track of the 32^{nd} Conference on Graphics, Patterns and Images (SIBGRAPI 2019)*. IEEE, 2019.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[9] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[10] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4401–4410.

[11] G. Lample, N. Zeghidour, N. Usunier, A. Bordes, L. Denoyer *et al.*, "Fader networks: Manipulating images by sliding attributes," in *Advances in Neural Information Processing Systems*, 2017, pp. 5967–5976.

[12] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

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

[14] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[15] H.-Y. Lee, H.-Y. Tseng, J.-B. Huang, M. Singh, and M.-H. Yang, "Diverse image-to-image translation via disentangled representations," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 35–51.

[16] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8789–8797.

[17] L. Horsley and D. Perez-Liebana, "Building an automatic sprite generator with deep convolutional generative adversarial networks," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 134–141.

[18] T. Xue, J. Wu, K. Bouman, and B. Freeman, "Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 91–99.

[19] M. Guzdial and M. Riedl, "Toward game level generation from gameplay videos," *arXiv preprint arXiv:1602.07721*, 2016.

[20] S. Snodgrass and S. Ontanón, "Controllable procedural content generation via constrained multi-dimensional markov chain sampling." in *IJCAI*, 2016, pp. 780–786.

[21] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep learning for video game playing," *IEEE Transactions on Games*, 2019.

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[23] A. Fadaeddini, B. Majidi, and M. Eshghi, "A case study of generative adversarial networks for procedural synthesis of original textures in video games," in *2018 2nd National and 1st International Digital Games Research Conference: Trends, Technologies, and Applications (DGRC)*. IEEE, 2018, pp. 118–122.

[24] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural content generation in games*. Springer, 2016.

[25] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 47–57, 2016.

[26] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[27] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[28] TensorFlow Development Team, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/