3D to 2D: Using Image Segmentation to Automate Rotoscoping

Group 10 - Logan White, Sari Pagurek van Mossel, Zeph Van Iterson

Motivation

Rotoscoping is a widely used technique in animation that involves tracing frame-by-frame over a video to create a 2D sequence. A frequent first step in this rotoscoping process, as well as in many artistic practices, is to break down the 3D figure into its key shapes before then moving on to further detail. One can see how in a larger animation project that is minutes or even hours long at the standard of 30 frames per second, this frame-by-frame drawing process can become exceptionally tedious and time consuming. There exist some tools within the field that ease this manual process, however it remains an extremely time-consuming task. The motivation for this project is to explore how a trained neural network can streamline this process and reduce the manual effort required from animators. Successfully automating this step would mark a significant advancement in the rotoscoping process, providing animators with a more efficient tool to create animation from live-action or 3D sequences while allowing them to focus on creative refinement.

Problem Description

This project works towards the problem of automating rotoscoping within the field of animation, by transforming a 3D visual into a 2D representation. To achieve this, we propose using semantic image segmentation techniques. Given a frame of a 3D scene, our goal is to segment the image into a simplified 2D visualization by extracting distinct tonal areas including highlight, material tone, diffuse shadow, cast shadow, and background. This simplified output would serve as a starting point for further refinement and drawing by the artist, minimizing or even eliminating the need for repetitive tracing.

Contributions

Group Member	Contributions
Sari Pagurek van Mossel	Data creation and preprocessing Model set up Training and testing
Logan White	Model refining

	Fixing image labels Training and testing
Zeph Van Iterson	Literature research Training and testing

Related Work

The first paper we reviewed was named “Rotoscope Automation with Deep Learning” and focused on automating the process of rotoscoping in animation. The model aimed to separate people from the background of images, and it achieved significant success in this area. However, the goal for our model was more complex: rather than merely distinguish a subject from a background, our model segments an image into 5 distinct components (highlight, material tone, cast shadow, diffuse shadow, and background). Additionally, while their model employed Google’s Xception architecture, our model uses the U-Net architecture. While their goal was similar to ours, the dataset they used for training did not capture all the information we wanted and was therefore not sufficient to train our model.

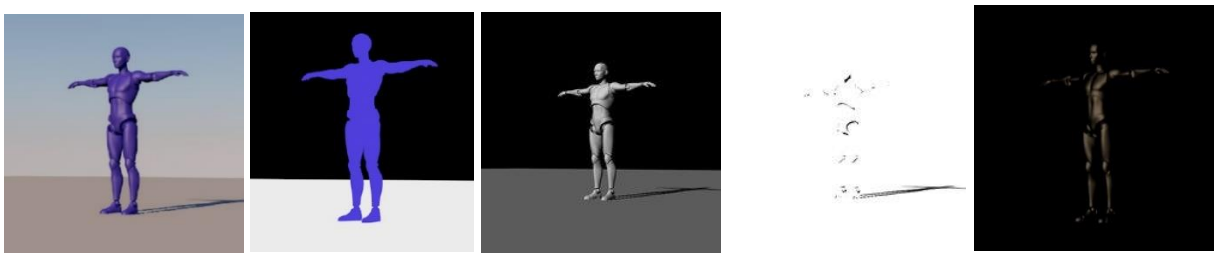
We then compared our model to several other U-Net implementations, but since the U-Net was developed for medical uses, most of its applicable research is attempting to segment medical images to find and diagnose illness using instance segmentation. One example was a research paper named “Image Segmentation Based on Improved Unet” which was attempting to segment images into two classes, tumour and non tumour (Li, Qian, Xu, & Liu, 2020). Yet again, this is much simpler than our model, since it is segmenting images into only two classes and their dataset was comprised of medical images which are not useful for us. Finally, there was one paper named “In-between frame generation for 2D using generative adversarial networks” that used U-Net for animation purposes, like our model does (Pazos, 2024). They used U-Net from in-between frame generation, so while this shows that U-Net have been used in the field of animation, their specific application and their dataset were not very applicable to ours. So, while there are many U-Net implementations out there which we used to inspire our design, we were unable to find any existing research or dataset for our specific application.

Dataset & Preprocessing

Given our prior research, we determined that no existing dataset fully meets the requirements of this project. Thus, we created our own through 3D modelling in Cinema4D, multi-pass rendering, and a preprocessing pipeline. To begin, we created several different 3D scenes within the modeling software which could be output as 90 frame video

sequences. We then used Cinema4D's multi-pass renderer to output individual layers of the scene: material colour, illumination, specular, and shadow.

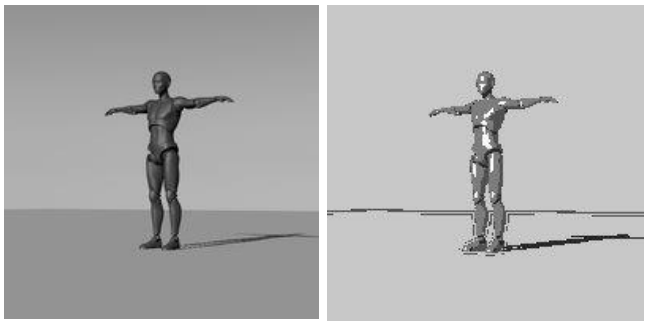
Examples of the frame exported from Cinema4D, followed by its individual multi-pass layers:



To reduce data size, we compressed and cropped the images to work on a smaller 200 x 200-pixel scale. We also converted the image formats from RGBA to greyscale. Next, we developed a threshold system to preprocess these multi-pass layers. This involved checking each layer separately, setting a colour value threshold to classify each off the 4 layers, labelling each pixel, converting each layer to a transparent png with only it's specified class, and then compositing them all back together with an additional background as a separate class. For example, the specular render only contains pixels where light is reflecting off material in the scene, with the rest of the image pure black. With the pixel-by-pixel analysis, we can separate out the sections of the image above a certain brightness and declare it as the highlight.

Below you can see an example of each preprocessed layer, followed by the final composited image:

With several video sequences of different 3D scenes, we ended up with approximately 1000 datapoints to train our model on. For each input greyscale image, we used our preprocessed segmented image as the label.



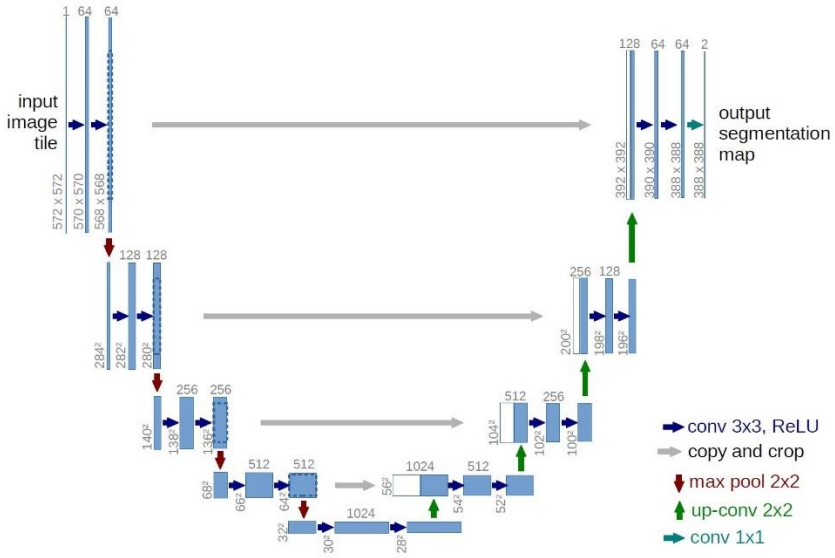
Overall dataflow of preprocessing:

1. Cinema4D 3D Scene
2. Multi-pass rendering of frames
3. Cropping, resizing
4. Convert RGBA to greyscale
5. Pixel-by-pixel preprocessing

Implementation

Experimental Setup

For this project, we selected the neural network model the U-Net due to its suitability for image segmentation. Additionally, it is notably effective when working with smaller datasets, a crucial factor for our project since we were generating our own custom dataset from scratch. During implementation, we followed a typical U-Net architecture, incorporating its encoder-decoder structure with skip connections. To implement our model, we used the Python programming language with Pytorch libraries in VSCode, and Github for filesharing. The data creation was done using Cinema4D.



The U-Net is designed with a symmetrical encoder-decoder architecture to facilitate semantic image segmentation. The encoder progressively down samples the input image, reducing its spatial dimensions while increasing the feature channels. Conversely, the decoder upscales the feature maps back to the original image dimensions, producing image segmentation maps. Both sections use double convolution blocks implemented with ReLU activation functions.

Encoder (Down)

The encoder comprises of four down sampling blocks, each including the double convolution followed by a max-pooling operation to reduce the spatial dimensions. As the resolution decreases, the model captures increasingly abstract features, culminating in a bottleneck layer with a maximum of 1024 channels.

Block	Operation	Channels In → Out
Down Block 1	DoubleConv(1→64)	1→64
	MaxPool(64)	64→64
Down Block 2	DoubleConv(64→128)	64→128
	MaxPool(128)	128→128
Down Block 3	DoubleConv(128→256)	128→256
	MaxPool(256)	256→256
Down Block 4	DoubleConv(256→512)	256→512
	MaxPool(512)	512→512
Bottleneck	DoubleConv(512→1024)	512→1024

Decoder (Up)

The decoder mirrors the encoder, consisting of four up sampling blocks to increase the number of feature channels. Each block includes an up-sampling operation to halve the number of channels, a concatenation step to integrate features from the corresponding encoder block (doubling the number of channels), and a double convolution that halves the out channel again. This process ensures the preservation of spatial information while refining feature representations. By the final decoder block, the feature maps are reduced to 64 channels, which is then further compressed to 5, to correspond with the model's 5 segmentation classes. These outputs can then be passed through a softmax function to compute the class-wise confidence scores, facilitating accurate pixel-level predictions.

Block	Operation	Channels In → Out
Up Block 1	Upsample(1024→512)	1024→512
	Concat(512+512→1024)	1024→1024
	DoubleConv(1024→512)	1024→512
Up Block 2	Upsample(512→256)	512→256
	Concat(256+256→512)	512→512
	DoubleConv(512→256)	512→256
Up Block 3	Upsample(256→128)	256→128
	Concat(128+128→256)	256→256
	DoubleConv(256→128)	256→128
Up Block 4	Upsample(128→64)	128→64
	Concat(64+64→128)	128→128
	DoubleConv(128→64)	128→64
Final	Conv2D(64→5)	64→5

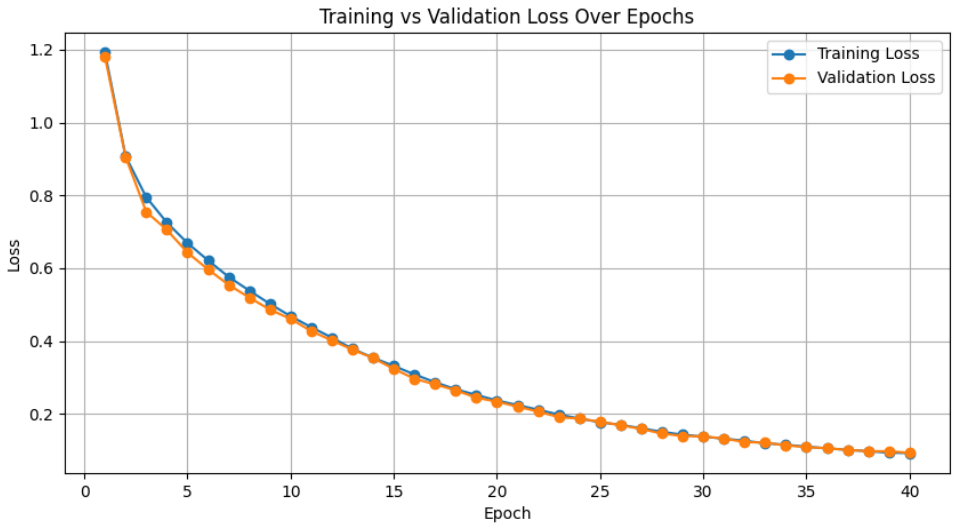
Training, Testing, and Validation

Before training, the dataset was divided into training, validation, and testing partitions. Since the dataset contained 14 unique scenes of 90 images, 3 of the scenes were used as the test set to make sure that the model did not have access to very similar data in both the training and testing sets. The remaining images from the other 11 scenes were randomly divided into training and validation partitions using an 80/20 split.

Training of the model was performed over 40 epochs using the Adam optimizer with an initial learning rate of 10^{-4}. 40 epochs was chosen as the stopping point based on an analysis of the loss plot and to avoid overfitting to the training data. Since this can be

described as a multiclass classification problem, the cross-entropy loss function was used to determine the loss of the network. All training was performed on an NVIDIA RTX 4060 GPU with a batch size of 32 to speed up training times.

Using the hyperparameters described above, the training produced a smooth, steadily decreasing loss curve which flattened out around 40 epochs at a loss value of approximately 0.09. The validation loss curve also followed a similar pattern. The loss plot for the training can be seen below.



Challenges of Training

One challenge that we encountered during training was how to process the labels for the loss function. Our first iteration of the code was incorrectly using grayscale intensities to convert the label images into classes instead of the dictionary that mapped classes to values from the preprocessing script. This initially caused our model to train using incorrect ground truth values and severely hampered performance. Once this was fixed, the model trained properly and achieved much better cross-entropy loss.

Testing

The model was tested using both the cross-entropy loss function and by visually inspecting the outputs of the model. To help with the visual inspection, each scene was stitched into gifs to match the original camera movement through the scene. Examining the gifs allowed us to make sure that there were no visible jumps between frames and that the highlights and shadows moved through the scene as expected.

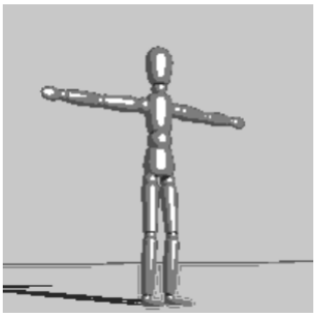
Results and Discussion

Testing results

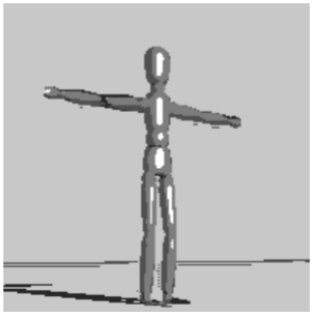
Dataset	Cross Entropy Loss
Train	0.0918
Validation	0.0905
Test	0.1328

As seen in the table, the overall loss of the model across all datasets was very small. However, an even more important metric was using the "eye-test" to determine if the semantic outputs of the model would be useful to a human animator. After comparing the model output with the input side-by-side, we determined that our model also passed this test and produced accurate and useful outputs.

Input sequence



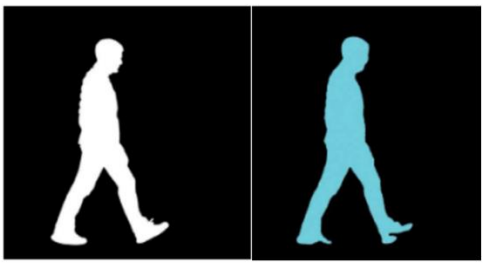
Sequence from label data



Sequence from model predictions

Comparing to State-of-the-art

Since we were unable to find anyone else who had made a model for this type of image segmentation, there is no perfect comparison for our results. The closest we could find was the paper we discussed earlier on automatic rotoscoping (Torrejon, Peretti, & Figueroa, 2020), which only separates the image into the object and the background instead of 5 classes. Their final model was able to separate the person very well, with some minor errors. An example of their results can be seen below, comparing the label on the left with the model prediction on the right:



Our model was also quite accurate based on visual comparison of our predictions with our labels and was able to separate the object from the background like their model. Overall, their model was more consistently accurate, but this is expected given the simplicity of their segmentation classes, larger dataset, and overall higher resolution images with the ability to yield more intricate and accurate shapes. Given additional resources, we are confident our model could compete with these results.

Conclusion and Future Work

In conclusion, we believe our project successfully achieved the goals we set at the outset. Using a U-Net neural network, we developed a model capable of segmenting unseen frames into five distinct classes: highlight, material tone, diffuse shadow, cast shadow, and background; with sufficient accuracy to reassemble smooth video sequences. These results are particularly promising given the constraints of limited training time and a small dataset, and they demonstrate performance comparable to existing automated rotoscoping techniques. To transition from a research proof-of-concept to a practical tool for animators, future work could focus on incorporating higher-resolution datasets, training on more complex scenes and backgrounds, and refining the segmentation process to handle a broader range of animation styles.

References

Krithika, M., & Suganthi, K. (2022). Review of Semantic Segmentation of Medical Images Using Modified Architectures of UNET. *Diagonstics, 12*.

Li, X., Qian, W., Xu, D., & Liu, C. (2020). Image Segmentation Based on Improved Unet. *Journal of Physics: Conference Series, 1815*.

Pazos, F. A. (2024). In-between frame generation for 2D using generative adversarial networks. *Open Access Theses & Dissertations*.

Torrejon, O. E., Peretti, N., & Figueroa, R. (2020). Rotoscope Automation With Deep Learning. *SMPTE Motion Imaging Journal*.