Turn Me Into A Sprite

CS39440 Major Project Report

Author: Dave Davies (dad58@aber.ac.uk)
Supervisor: Dr Hannah Dee (hmd1@aber.ac.uk)

3rd May 2024
Version: 1.0 (Release)

This report was submitted as partial fulfilment of a BSc degree in Computer Graphics, Vision and Games (G450)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, U.K.

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name ...Dave.Davies.....

Date ...03/05/2024.....

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name ...Dave.Davies.....

Date ...03/05/2024.....

Acknowledgements

I am grateful to my supervisor Hannah Dee for her patience and guidance throughout this project. Her project suggestion as well as supervision style were perfect for me.

The support from the Aberystwyth computer science department has been outstanding throughout this degree.

I would also like to thank Milly ODonnell for being by my side in every lecture, group meeting, and mental breakdown.

In addition, I would like to thank Bob for everything.

I am also grateful for the emotional support I received from Amy Potter, Lily & James, Luke & Jeanne, and the Aberystwyth Taskmaster Society.

Finally, thank you to the Aberystwyth Computer Science Society and the residents of B59 for testing my code and listening to me talk about it for hours on end.

Abstract

Scratch is a website that allows children and programming beginners to easily write code. It has a visual, user-friendly interface and is built for innovation and fun.

This report aims to describe the process in which a program that works with scratch was created. It allows a scratch user to generate a sprite of themselves automatically. The only necessary tools are their camera's live video input and a machine running windows 11.

Python was used to create the entire project. The work produced involved use of computer vision techniques and file manipulation. Open source computer vision (Open-cv) was the most useful external tool. It allowing for simple implementation of video input, data extraction, and image analysis.

Research into motion detection optimisation was done after the project reached its minimum requirements. An exhaustive search was performed on every combination of parameters within a Mixture of Gaussians algorithm. It was specifically designed to optimise the amount and consistency of sprite images saved when running the program.

Contents

List of Figures

List of Tables

Chapter 1

Background & Objectives

1.1 Background

”Scratch is the world’s largest coding community for children and a coding language with a simple visual interface that allows young people to create digital stories, games, and animations” [1]. It is used by people all over the world to help young people better understand the principles of programming. It is common for students to immediately recognise the language, and often invokes nostalgia. Due to the language being aimed at children, user friendliness is a focus of the language. Using scratch is supposed to feel both fun and easy.

Leaning into the theme of both fun and accessibility, if children could see their own face in their scratch projects, as a video game character or being animated, it could greatly enhance the experience of learning to program. The project is to obtain images of users in a live, easy to use way. In the end product, they should be able to pose in front of a live camera and an image of them should save easily and intuitively. The program will detect a lack of motion and respond accordingly, saving the pose that the user is striking.

These files of images that a user can implement as a sprite are in a dedicated file format. A scratch sprite is composed of image files and a json file that contains data pertaining to the assets included in it. It is then zipped and renamed to have a suffix of *.sprite3*. If images can be detected, saved, and processed into this format using code, this project is possible.

Scratch is able to process transparency, so the two common image types found were vector images and pngs. Given the nature of camera images being composed of pixels, pngs were the chosen file format.

The next part of the problem is generating the sprites from a video. This may be done with the press of a button, but would ideally be done with motion detection.

Given experience in a computer vision module, this seemed like a fun task. Computer vision was the preferred topic for a dissertation, and implementing it in such a unique project was the perfect way to experiment with it. From past knowledge, open source computer vision libraries used in python seemed the best way to go.

Planning for the project involved splitting each week of the dissertation into one specific task.

1.2 Analysis

The problem was divided into distinct sections of work: planning, manually saving a sprite, generating images from a video feed, saving a sprite with those images, generating images automatically from a video feed, background subtraction, getting it on a website, improving UI/UX, testing.

The work was organised in such a way that python file handling and other libraries could be learned as the project progressed. The difficulty of each task increased, but so did the use of computer vision concepts, which was the initial appeal of this project.

Initially, measuring the performance of the code seemed very simple. The quality and consistency of the sprite generator would be very obvious; would the sprite look like what is expected? This would entail a sprite produced with not too many unwanted pixels around the outsides, actually removing the background effectively, and reasonable lighting adjustments.

The sections of work clearly translate into a list of requirements. Later in the project, when the requirements were completed to a satisfactory level, the question became how the motion detection algorithm could be optimised.

1.3 Process

Agile iterative development was used as this seemed to be the most intuitive method. All of the steps of this program were naturally able to be broken down into about a week's worth of work, including research and planning. The most intimidating part of this project was the amount of time it would take to complete the work, not inherently the difficulty of the work presented.

The first half of the project was completed as planned. However after unsuccessful testing, it became clear that the project could not be accessed via the web. The code was functioning to a satisfactory level, but was often saving images when motion was above the intended threshold, or was not saving images when designed to.

For the research method, primary quantitative data was used. An exhaustive search of all feasible motion detection algorithms were performed in order to find which one performed best.

Chapter 2

Design

The basis of the timeline for the project was split into each week's work. Each task was planned from a high level, with the intention of completing research on each following task as the project progressed.

1. Planning phase
2. Create a sprite manually using python
3. Create sprites automatically with python
4. Save images from a video feed
5. Background subtraction
6. Getting it on the web
7. Add instructions and tidy UI/UX
8. Final tidying of code and project as a whole

The planning and final phases were deemed to need more than one week of work each, so those topics were given extra time. The final plan for the project is displayed in table B.1.

The first tool researched was media pipe [2]. This is a google tool that detects people and objects using computer vision, as well as interpreting poses. An issue arose when typescript could not easily be used in a web development environment. It seemed inconvenient to learn to use typescript as a replacement for javascript in a web page. Given python was already going to be learned for this project, learning a second language on top of this was deemed illogical.

The end product was supposed to be a web page, so media pipe lent itself to this, but using a background removal algorithm as opposed to an object detection algorithm seemed more appealing after brief tests. The versatility and usability of opencv made it clear that python opencv is what would be best to use.

	Planned work
29/01/2024	Planning phase
05/02/2024	Planning phase
12/02/2024	Create a sprite manually using python
19/02/2024	Create a sprite automatically using python
26/02/2024	Save images from a video feed
04/03/2024	Background subtraction
11/03/2024	Getting it on the web
18/03/2024	Add instructions and tidy UI/UX
25/03/2024	Easter
01/04/2024	Easter
08/04/2024	Easter
15/04/2024	Final tidying of code and project as a whole
22/04/2024	Final tidying of code and project as a whole
29/04/2024	Final tidying of code and project as a whole

Wider project work
Software engineering
Rest / Other work

Table 2.1: The initial plan of work

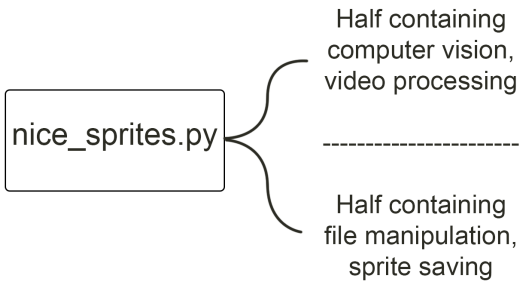

Figure 2.1: Structure of the main python file

The structure of the python code was intended to be a single file, so as to make the project easier to run. In order to test the functionality of the downloaded scratch sprites, the directory ”testing and sprite noodling” was created. Within this is many directories and files that have no impact on the final product, but were very valuable in understanding the infrastructure of external code.

No complex persistent data storage was required or even researched for this project. All file storage is done locally and client side. Any file handling necessary would be written with basic python.

The code of the project itself is split into two separate parts: saving the images from live video, and the processing of these videos including turning it into a sprite. These sections would be tested and developed independently, and implemented together.

The queue ”frames” contains all of the information needed about each individual frame.

A queue was the most logical data structure, as each frame would need to be stored for a length of time, then eventually replaced in a cyclical structure. It is composed of lists, each containing data about that frame.

The five pieces of information stored within that list are:

- frame_num: the number frame this is from the beginning
- frame: the image itself
- fgMask: the frame mask as created by the background subtraction
- white_pixels: the count of white pixels (denoting movement) in this frame
- the standard deviation of the change in mean over the last n frames

The reason for inconsistent camel case and snake case is that some code was taken from websites or tutorials. Refactoring this would have improved readability and consistency.

Chapter 3

Implementation

The implementation largely followed the structure of the planning phase. Most of the early features were completed within the expected time frame. Occasionally there would be a delay, but this was normally corrected by the day after each section's deadline. The use of sentient "rubber ducks" at the computer science society meetups on Tuesdays fixed most of the delays in the timeline.

	Actual work
29/01/2024	Planning phase
05/02/2024	Planning phase
12/02/2024	Create a sprite manually using python
19/02/2024	Create a sprite automatically using python
26/02/2024	Save images from a video feed
04/03/2024	Motion detection
11/03/2024	Background subtraction
18/03/2024	Gather data for motion detection improvement
25/03/2024	Easter
01/04/2024	Easter
08/04/2024	Process data
15/04/2024	Process and display data
22/04/2024	Final tidying of code and project as a whole
29/04/2024	Final tidying of code and project as a whole

Wider project work
Software engineering
Research
Rest / Other work

Table 3.1: The end work flow

3.1 Creating a sprite using python

The first challenge in the implementation was in deconstructing the scratch sprite. A scratch file can be unzipped. It contains all of the assets for the sprite, alongside a json file composed of information about the assets. The assets are the images contained as well as any code or audio associated with the sprite. For the purposes of this project, only images will be needed.

Deconstructing the json file and then recreating it using python was the first step. This was completed with the use of three python json manipulation tutorials [3] [4] [5]. Within the sprite json file, the md5 hash of each image is required. This was the only non obvious variable needed after looking inside the file. The variables had clear, descriptive names that alluded to their function.

A json file directly extracted from a downloaded sprite file contained the following:


```
{"isStage": false,
"name": "test",
"variables": {},
"lists": {},
"broadcasts": {},
"blocks": {},
"comments": {},
"currentCostume": 1,
"costumes":
    [{"name": "bob3.png",
    "bitmapResolution": 2,
    "dataFormat": ".png",
    "assetId": "d8c269eaeb83d982101cf887a50a8b97",
    "md5ext": "d8c269eaeb83d982101cf887a50a8b97.png",
    "rotationCenterX": 256,
    "rotationCenterY": 256},

    {"name": "269c8d076d7760d20081e4ac52ced703.png",
    "bitmapResolution": 2,
    "dataFormat": ".png",
    "assetId": "269c8d076d7760d20081e4ac52ced703",
    "md5ext": "269c8d076d7760d20081e4ac52ced703.png",
    "rotationCenterX": 256,
    "rotationCenterY": 256}],
"sounds": [],
"volume": 100,
"visible": true,
"x": -30,
"y": 5,
"size": 100,
"direction": 90,
"draggable": false,
"rotationStyle": "all around"}
```

Most of the information about the images, such as their file type and position on the scene, was easy to generate either statically and consistently or using a single python function. Eventually it was found that the md5 hash could also be generated with a single python function [6] [7]. This involved importing the hashlib library.

A few days of fixation was spent on the names of the images in the sprites. The name of each image does not make a difference to scratch's interpretation of the sprite. In a similar level of fixation, zipping the files took great care and difficulty. Many ways of zipping files were looked into [8] [9] [10] [11] [12], only to find that the first method [8] worked. The issue was that the wrong compiler was selected. Windows' in-built compiler interpreted the imports, whereas the python compiler did not.

There was also an issue surrounding the location of the images. At one point, the code would correctly create the folders, correctly generate the json file, but would pull all of the files up a directory when completed. This was due to naming multiple file variables "f" in one script.

Placing files in easy to access directories was surprisingly difficult. Understanding the structure of file storage and how to utilise it was a challenge, and required research [13] [14] [15]. These three sites all describe how to utilise python's directory management functions. In order to not have overwriting sprites, sprite files are saved as the name of the year + month + day + hour + minute + second. This will ensure that each sprite title will be unique. It also automatically sorts sprites by time, making the most recent sprite at the bottom of the list, easy to access. The datetime library was used for this [16] [17].

All of the relevant code for creating a sprite was made in a file of its own. The intention was to move this specific part into a function so that it could be imported, but given the length of the final file, it was still readable when placed into a single script with all other code. This should make running the final project more convenient.

3.2 Save images from a video feed

The minimum working requirement did not specify how the images are saved from a video feed. So the first solution was to simply save a frame when a key is pressed while displaying a video. The majority of this code can be taken directly from opencv (Open Source Computer Vision) documentation [18]. An example of the outputted image is shown in figure 3.1.

3.3 Motion detection

Early on in the project, a mixture of Gaussians (MOG) model was selected to be used as the motion detection algorithm.

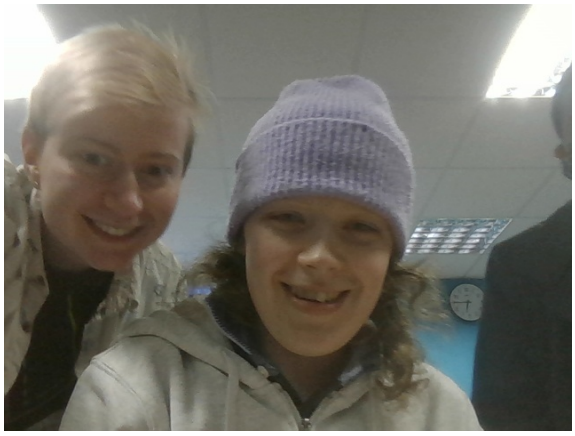

Figure 3.1: Initial testing of saving an image when pressing a spacebar

3.3.1 Mixture of Gaussians

If a background were to remain entirely static, the foreground could very easily be removed from the background. It would only involve detecting whether a pixel had changed from frame to frame or not. Real video input, however, has many inconsistencies causing the background to change colour. Sensor noise, monitor flicker, and shadows would all cause a false positive foreground. The first improvement for this is to apply a threshold, so that only change above a certain level is considered. A gaussian filter would also improve the detection, by comparing each pixel to the mean and standard deviation of its history. If a pixel is similar enough to the stored history of itself, it is considered static background.

The issue then becomes if a background pixel is consistently more than one colour. For example, if a leaf blowing in the wind was in the background, it may have a light side and a dark side. Both dark green and light green would be showing frequently, but only one of these colours could be considered foreground. Whenever the leaf flips over, it would give a false positive of foreground.

This problem can be fixed by applying multiple gaussians. If there are two gaussians, the colour of a pixel can be compared to two seperate means. For example, the leaf pixel could be compared to the mean of dark green *and* compared to the mean of light green. If it is similar enough to *either* of these colours, it is considered background.

The default amount of gaussians was used in this project, which is five.

The camera would be in a static position, and the adaptive nature of MOG should be able to scale to the amount of motion [19]. A simple way of implementing this is available on opencv documentation [20]. The specific function for mixture of gaussians is referred to as MOG2 within opencv.

This displays a mask that showed the movement detected in white pixels. Anything representing the background is black pixels. This is shown in figure 3.2.

Figure 3.2: The foreground mask created by MOG2

Figure 3.3: An image generated with MOG2 background subtraction

3.4 Background subtraction

Background subtraction was the most difficult implementable part of the project. Initially, MOG2 was intended to perform the background subtraction as well as the motion detection. The reason for this is that MOG2 motion detection can efficiently remove the background when there is a consistent background over a long period of time. If only small objects are occasionally passing in front of the camera, the objects themselves are very distinct. This makes it perfect for a security camera or showing a single boat passing by on the ocean. But as shown in figure 3.2, this does not intuitively map when processing a close-up video of one person. It only selects the outline of the person as the centre of the person is too consistent, and becomes considered background too quickly.

In attempting to produce an image with the background subtraction, the foreground mask was blended onto the image taken from the camera. This resulted in an outline of the subject on a transparent background. Given how unique and interesting this looked, the code to generate these images was kept as a file called ”scary_monsters.py”. The output of this is displayed in figure 3.3.

After one week of attempting to use MOG2 as a background subtraction algorithm, new methods of implementing it were looked into. Rembg [21], an external library was used. It is a background subtraction tool that can very easily be applied to images. This involves a pip install to utilise. It works reasonably consistently, with backgrounds effectively removed, the main issue being the occasional low opacity on clothes and bodies. When the background subtraction does produce unexpected outputs, this can often be humorous, such as decapitating the subject. This may be beneficial to the end product, especially as the target audience is children. The end goal is for this product to be engaging and fun, not necessarily perfect.

3.5 Getting it on the web

After two days of looking into how to use this program using a web page, it was deemed either impossible or not worth doing with the time left in the project. PyScript [22], a framework for implementing python within HTML on a web page, was the main method researched. The issue arose in not being able to use opencv or rembg within the web page. These required pip installs to use them, and installing them could not be done server side. Using a python based web framework may have made this possible, but that would have involved completely overhauling the structure of the project.

It was decided that without moving to the web, the project could be considered to be at its finished state. This meant the rest of the time could be dedicated to optimising whatever part of the program was performing worst. This was the motion detection.

3.6 Final tidying of code

At the very end of the project, the topic of user friendly code execution was revisited. If a website was not feasible, then the user would have to download the python file, install python, install opencv and rembg using pip, and ensure they are using the correct compiler. This is too complicated for the intended use of the project to be realistic. Running a single executable file without the need for imports and code handling would make it much easier to use.

This will only work on windows machines, but the likelihood of schools using linux is low enough that this is not a concern. It may make testing on university or friend's machines more difficult, however.

The first library looked into for completing this was cx-freeze [23]. This was installed and attempted to run. Some of the instructions were tried but produced an error.

After trying and failing with multiple support websites, a different library was used following advice on stack overflow [24]. This did not initially work and was met with the error "*The term 'pyinstaller' is not recognized as the name of a cmdlet, function, script file, or operable program.*". This was easily fixed with advice from a separate stack overflow page [25].

This compiled all code and their relevant imports and dependencies into a single executable file. The project can now be easily and accessibly ran on windows 11 machines.

3.6.1 Final structure of files

The structure of the final product is now split into two runnable files and two directories containing previous work and testing. nice_sprites.exe is the file to run for convenience, the easier file that will only take a double click to run. nice_sprites.py is a file to run on a non-windows machine, but will include installation of rembg, opencv, and python itself. Both are easily accessible so that both methods of using the software are possible.

The directory ”information gathering” contains the key information and files used in my research. The file used for data manipulation is ”datainator.py” and the file used to record the videos of subjects producing an appropriate amount of motion is ”giveMeYourFace.py”. Videos contains some of the most useful information gathered, but if one wanted to reproduce the initial data, running some functions in datainator will be needed.

”testing and sprite noodling” contains the very earliest work in the project. It is largely composed of deconstructed sprite files.

Chapter 4

Testing & Experiment Methods

4.1 Testing

Only informal testing occurred earlier in the project. The sprite generator was frequently shown to interested parties, and verbal feedback received. The biggest issue reported after background subtraction was completed was that images were not being saved at the expected moment. Sometimes people would pause for longer than the predicted five seconds and no image would be created. At other times, and more often, images would be saved when people clearly were not pausing. Given the amount of time left on the project, a test table was deemed unnecessary until research had been done into how to optimise motion detection. The tests of the overall product were conducted at the very end of the project.

Multiple different machines were used in testing. The results from these user tests are from attempting to run the executable file in its final version. Only windows machines were able to be used, as the project is a .exe file. When testers were trialling this software, thorough instructions were given as to how to correctly use it. Verbal instructions were given as well as a written message when opening the program.

The complicated nature of running and installing all of the necessary components meant that testers were reluctant to try the python file. The mysterious nature of the executable file also caused people to be apprehensive, however.

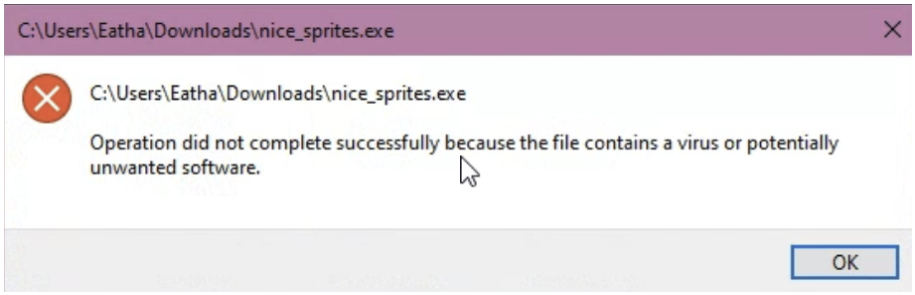

Figure 4.1: Windows preventing the project from opening

Test ID	Description	Steps	Inputs	Expected result	Pass Fail
TC1	The program executable runs when opened	Open the file	Double click the exe file in a file explorer on windows 10+	A python file is opened, with text instructions displayed	Pass, or windows prevents, due to expected virus (4.1)
TC2	Displays direct camera feed window	Open the file and wait	NONE	A window shows the live input from a camera	Pass, or windows cannot detect a camera (if there is none)
TC3	Creates directory to store images when ran, with title of current datetime	Open the file and wait	NONE	In the same directory as the exe, there is a new directory of the datetime when the program started	Pass
TC4	Image is saved when space pressed	Select the window displaying	(Click window if not selected) Press space key	The stream pauses, text at the bottom right displays ”saving one image...” and an image is saved to relevant directory	Pass, but testers often forgot to select window
TC5	Image is saved when motion detected below threshold	Have the file open and wait	Move around a lot, then move around much less for five seconds	The stream pauses, text at the bottom right displays ”saving one image...” and an image is saved to relevant directory	Pass, normally
TC6	Program is closed when quit	Press q to quit	Press q key	All windows close and program ceases	Pass, if q is pressed and held for one second. Some testers interpret as quick press

Table 4.1: Test table page 1

Test ID	Description	Steps	Inputs	Expected result	Pass Fail
TC7	Created directory is turned into a sprite when program closed	Press q to quit	Press q key	In the same directory as the exe, there is a new sprite3 file named the datetime when the program started	Pass
TC8	The sprite functions as expected when uploaded to scratch	Go to the scratch editor, upload a sprite, select sprite	Type in web address, hover mouse over choose a sprite, click upload a sprite, select sprite from directory in which it was saved	The sprite is now visible on the scratch canvas, with costume 0 diaplayed	Pass
TC9	The background is subtracted from the resulting images	Open the sprite and wait, or look in the directory of the datetime	Type in web address, hover mouse over choose a sprite, click upload a sprite, select sprite from directory in which it was saved	The sprite displayed has no background, only the subject in the image	Pass, normally - less reliable in low levels of lighting or moving camera

Table 4.2: Test table page 2

TC2 no camera error:

```
[ERROR:0@2.018] global obsensor\_uvc\_stream\_channel.cpp:159
cv::obsensor::getStreamChannelGroup Camera index out of range
Cannot open camera
Traceback (most recent call last):
File "nice\_sprites.py", line 335, in <module>
File "nice\_sprites.py", line 91, in spritify
NameError: name 'exit' is not defined
[14012] Failed to execute script 'nice\_sprites'
due to unhandled exception!
```

The compiled executable was distributed to testers twice; the first version was found to have a major bug in it. Within the section of code that turns directories into sprites, there was no test to ensure only folders generated by the program itself were turned into sprites. The code iterated through every folder where it was installed. Despite the testers being told they should save the file in a directory with nothing else in it, none of them did this. If testers cannot follow this instruction, school children should not be expected to. One tester had their recycle bin turned into a sprite, another had their project figures turned into a sprite.

These sprites still functioned well on scratch. If there had been larger folders in the directories, however, it would duplicate many files and could be considered malware due to the amount of storage and processing it would take up.

The released version includes a line checking that the title of the directory conforms to the same format that I have saved the sprite directories in. This includes checking that it is composed only of numerical digits, and is fourteen characters long. All of the sprites are exactly this format due to the titles of them being created from year (4 digits) + month (2 digits) + day (2 digits) + hour (2 digits) + minute (2 digits) + second (2 digits). There is a possibility that other directories are saved with this title format, given how logical it is, but it should filter out the majority of other directories.

4.2 Experiment Methods

4.2.1 Planning

When the direction of the project veered away from being a website, it started to become a research project in nature. The overall question was:

> **How can the motion detection algorithm be optimised so that it only saves images at the correct point?**

This can then be broken down into two smaller problems:

1. What produces the most accurate number of sprite images?

2. What makes the images be saved in the most accurate position?

These questions would need to be answered with a variety of data that could be easily compared with one another. The solution to this was saving a series of videos that analysis could be performed on collectively. Test data was gathered of a subject moving in accordance to an instruction given. Each instruction was given for five seconds at a time. The instructions were displayed on the visual of the participant's face so that they could create the sprite while looking at themselves, in the same way that a user of the end product would interact with the software.

Each test video was recorded with the intention of comparing it to other videos. For example, different lighting levels were used with the same model doing the same poses for three videos. Additionally, there were videos that contained people moving in the background, videos of stuffed toys in case a user wanted props as their sprites, and a very wide variety of amount of movement.

The three instruction strings given were:

- ”move around a lot!”
- ”bit of movement”
- ”stay still”

Three variations of movement were chosen so that data would be gathered in relation to an increase or decrease in motion at various levels. This variety should be enough to show how the program works with each movement. Hypothetically, when this level of background change is shown on a graph, there should be three distinct tiers of movement detected over time. These instructions are vague as the program needs to be able to communicate amount of movement to children very easily. It should also be able to scale to respond to how much people are able to move. For example, if someone has mobility issues and cannot move much, the code should be able to save images just as well. Conversely, if someone cannot stay still, perhaps for a health reason, it should work just as consistently.

This was tested with a list of instructions that contained every combination of consecutive movements. The order of the instructions given is shown in table 4.3.

Each of these instructions were displayed for 5 seconds. When the minute (12*5 seconds of instructions) of recording was done, the frames were saved in an avi file. These did not have a consistent frame rate, so the amount of frames each video took per instruction is estimated. Most of the videos had similar enough frame rates that this made no difference.

A signed consent form was collected from every person that took part in the data collection, shown in figure B.1.

Number	Instruction	Time started (seconds)	Frame started (approx.)
1.	”bit of movement”	0	0
2.	”bit of movement”	5	
3.	move around a lot!	10	293
4.	”stay still”	15	443
5.	move around a lot!	20	593
6.	”bit of movement”	25	744
7.	”stay still”	30	893
8.	”stay still”	35	
9.	”bit of movement”	40	1193
10.	move around a lot!	45	1343
11.	move around a lot!	50	
12.	”stay still”	55	1643

Table 4.3: Instructions for movement

4.2.2 Implementation

”Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.” [26] It was used to create graphs which showed how the motion changed over time. This helped to understand the data and where sprites were being generated in relation to the instruction given.

The vast majority of the data generation took place in the file ”datainator.py”. This is a file comprising of many functions, each with a different method of manipulating information. All of these functions require a path for the avi files so that each video can be placed into a directory of its own. Once given this path, it can recreate all of the data generated for these findings.

The function ”*spritify_videos*” iterates trough all of the directories of videos and creates a folder for the algorithm running it. Within each of these folders, four separate items are created.

1. The sprite itself
2. The unzipped sprite - a directory containing the json file and the images so that they can be easily accessed to check which video this data is from
3. A csv (Comma Separated Value) file containing the count of white pixels, the moving average, the moving standard deviation, and a value of either 0 or 300000 denoting whether an image is saved on that frame or not (Using the csv python library)
4. A graph representing the data in the csv file

Using this file, data was gathered on every feasible motion detection algorithm to find which would perform best.

Two motion detection algorithms were considered, each because they are convenient to use with open cv [19]. KNN is a more basic version of MOG2 so it was assumed that

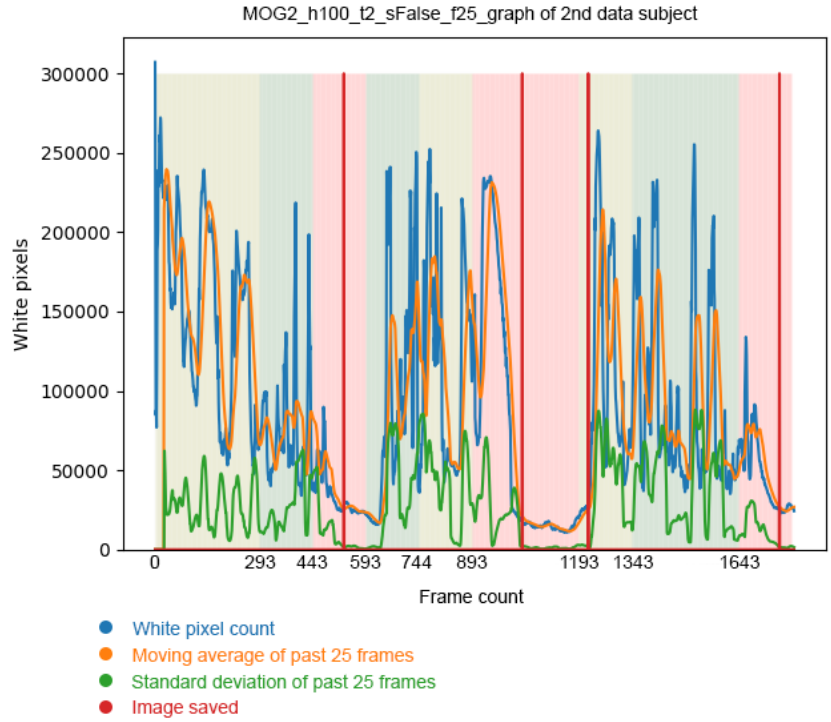

Figure 4.2: Graph showing white pixel count, moving average, and moving standard deviation of a subject's motion over the instructions given.

Type	Parameter	Value
Int	History	500
double	varThreshold	16
bool	detectShadows	True

Table 4.4: MOG2 default parameters

MOG2 would perform better. Given the example parameters in the documentation, lists of feasible parameters were chosen to be tried within the algorithm. The parameters given are shown in table 4.4.

Values above and below these were selected to be tried out against all other combinations of parameters. I realised that I could also add in the history of frames stored in the queue of previous frames in the actual spritify function itself, so the parameters tested are shown in table 4.5.

An exhaustive search was performed to check every single combination of parameters. This ended up being 4 * 5 * 2 * 4 possibilities, and ended up as 160 combinations. Each input took approximately half an hour to run on all videos combined, so this function finished after around three days of running.

The code used to run this was very simple. A function was created to iterate through every variation of *spritify_videos*, and produce a directory containing all of this information for each algorithm.

Type	Parameter	Value
Int	History	100
		200
		400
		800
double	varThreshold	2
		4
		8
		16
		32
bool	detectShadows	True
		False
Int	frame lengths	25
		50
		75
		100

Table 4.5: MOG2 given parameters

```
histories = [100, 200, 400, 800]
var_thresholds = [2, 4, 8, 16, 32]
shadows = [True, False]
frame_lengthses = [25, 50, 75, 100]

def big_spritify():
    for history in histories:
        for varThreshold in var_thresholds:
            for detectShadows in shadows:
                for frames_length in frame_lengthses:
                    spritify_videos(history,
                        varThreshold,
                        detectShadows,
                        frames_length,
                        "MOG2_h"+str(history)+
                        "_t"+str(varThreshold)+
                        "_s"+str(detectShadows)+
                        "_f"+str(frames_length))
```

Listing 4.1: big_spritify

This produced a directory for each of the 160 parameter combinations. The names for all of the algorithms used in this report are generated from this function.

In order to measure the performance of the algorithm variations quickly and efficiently, it was concluded that the ideal number of images generated would be two. The images should be created when the amount of motion drops from a lot of movement to none at all, so at 15-20 seconds and 55-60 seconds. An important part of the selection of an algorithm is its versatility and ability to work reasonably well with any of the training data, therefore a mean value of images generated was calculated for each algorithm.

The purpose for the fourth value in each csv file was to clearly display on a graph when an image is saved. Instead of using a boolean value, which would be intuitive, a

MOG2 h100 t4 sFalse f75	2.05
MOG2 h100 t4 sTrue f75	2.05
MOG2 h200 t2 sFalse f100	0.66
MOG2 h200 t2 sFalse f50	2.05
MOG2 h200 t2 sTrue f100	0.66
MOG2 h200 t2 sTrue f50	2.0
MOG2 h400 t16 sFalse f100	2.0
MOG2 h800 t32 sTrue f25	6.05

Table 4.6: Best and worst performing algorithms, according to amount of images generated per sprite

numerical value of either 0 (not save) or 300000 (save) was given for each frame. Calculating a mean was therefore counting all of the 300000 values in the csv files and dividing it by the total number of videos. 160 ”mean” csv files were created at the end of each run of video processing, each containing only one number that represented the mean of its variation.

The first shortlist of well performing algorithms initially gathered algorithms that had an average frame count of more than 1.9 and less than 2.3. Generating slightly too many frames was favoured over too little, as the change from little to double length no movement between 25 seconds and 40 seconds could generate a sprite and this would not be violating the sprite generation criteria too badly. A list of these results as well as worst performing algorithms gave 11 results. The performance criteria were then updated to more than or equal to 2 sprites, and less than 2.2. The results of this are shown in table 4.6.

Using this list of notable algorithms, a new directory of only these sprites was generated for ease of comprehension. Comparing each individual graph with one another was still a challenge, so a graph containing all of the data for each algorithm overlaid on each other was created to more accurately show how each algorithm performs.

MOG2 h200 t2 sTrue f100 was ignored as it yielded practically identical results to the same algorithm with shadow detection set to False. Only one graph was needed to show the minimum number of images generated. Both MOG2 h100 t4 sFalse f75 and MOG2 h100 t4 sTrue f75 were also ignored as, while they performed very well, they were not the top performing algorithms.

Chapter 5

Results and Conclusions

The two measures of suitability for each algorithm were the amount of sprite images produced and the distribution of those images along the timeline. These graphs are the visual indicator of both of these measures.

The two best algorithms both have an average image output of exactly 2, and therefore have the amount of images saved overall on one graph. Initially this involved 18 test videos, but this reduced to 16, so there are 32 black lines indicating saved image on both MOG2 h200 t2 sTrue f50 [5.1] and MOG2 h400 t16 sFalse f100 [5.2].

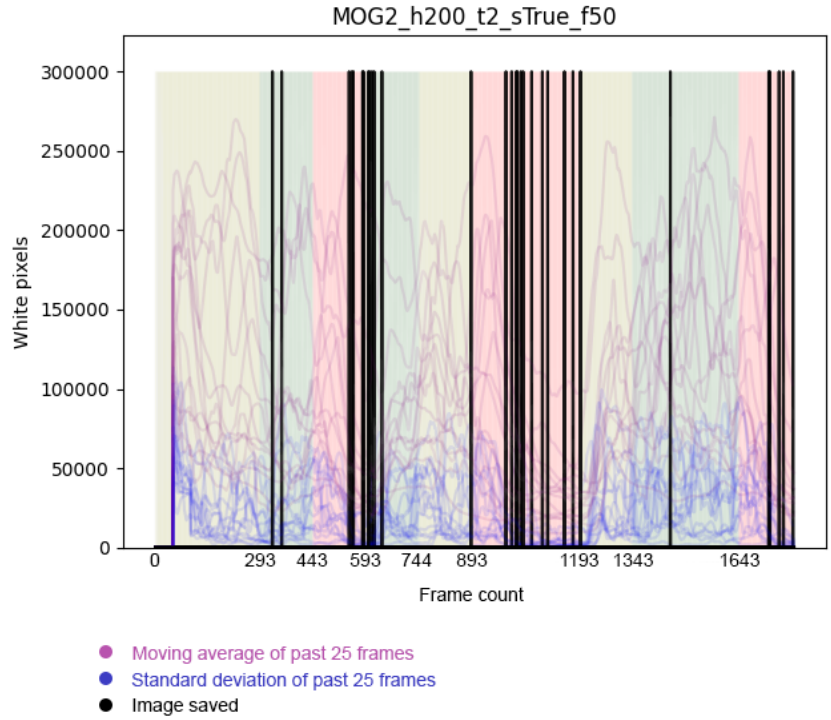

Figure 5.1: MOG2 h200 t2 sTrue f50 full graph

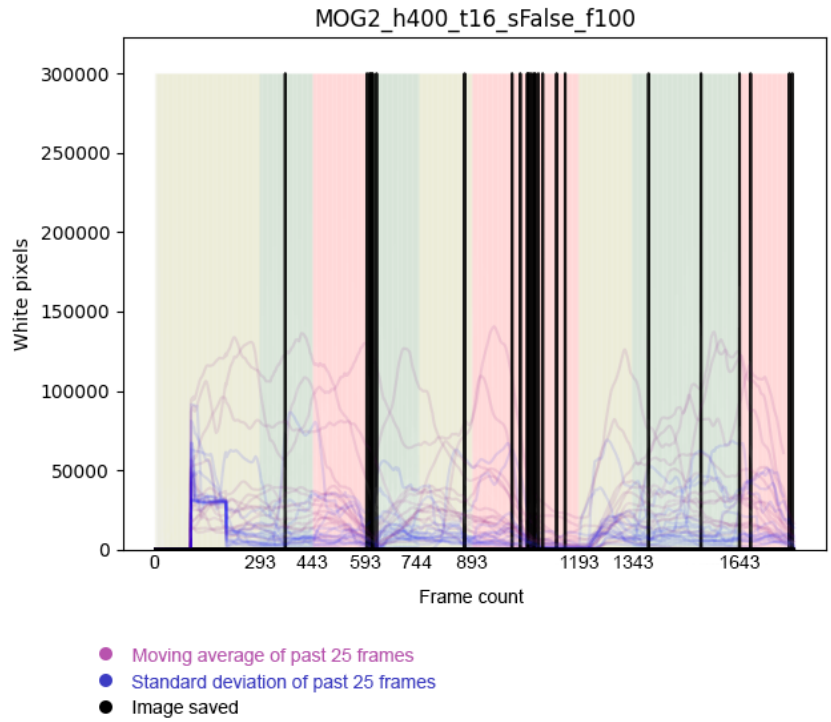

Figure 5.2: MOG2 h400 t16 sFalse f100 full graph

Each graph displays four separate pieces of information:

1. The instruction given to the model in the test video
   - Green background = ”move around a lot!”
   - Yellow background = ”bit of movement”
   - Red background = ”stay still”
2. The moving average of how many white pixels are displayed, denoting motion (purple lines)
3. The standard deviation of the change of white pixels (blue lines)
4. When the images are being saved (black vertical lines)

The reason for selecting blue, purple, and black lines was that they are clearly visible over the chosen background colours. They are completely different hues and values. This makes them easy to see even when displayed with low opacity. Low opacity was chosen so that the background colour, indicating instruction, is still visible.

The amount of previous frames to take into consideration for the moving average and standard deviation is the history that I used as a parameter in the algorithms. Images are saved when the standard deviation is below a threshold of 2000.

Test videos 20240319171001 and 20240319181355 were removed from the visualisation. They were the two low lighting level tests. In the collection of the test data, no fixed frame rate was set, so it adjusted the amount of frames taken to the lighting level. All of the other algorithms ran over approximately 1793 frames (1 frame of variation), whereas these two were composed of 1467 and 1500 respectively. When overlaid onto the rest of the graphs, these produced incorrect information as to when the images were saved in respect to time.

Altering the distribution of this data so that it could be plot on the same graph or making a new graph for these scenarios in particular was considered, but given the use of this software will largely taking place in schools or other well-lit areas, this was deemed unnecessary. This information will likely not benefit the output of the project.

The best algorithm seems to be MOG2 h400 t16 sFalse f100 [5.2]. The frames in which the images are saved are more tightly packed together on that graph, meaning images are saved at more consistent times. The ideal location for images to be saved is at the end of the first and last red bar, having detected five seconds of being stationary. Every variation, however, saved images between halfway and the end of the middle larger pause section.

Choosing the criteria of saving the images to be the standard deviation of the moving average may not be the best way to save images when detecting the difference between a lot of motion and none. What this criteria gives us, is detection of lack of motion regardless of previous respective quantity of motion.

This results in taking images after a long pause, but not necessarily after when people are altering their movement for an image. With this information, there are two theories as to how to improve the performance of the final product.

The first is considering three images to be the ideal number of detected pauses. Then finding the algorithm which most accurately created three images, one for each pause regardless of motion that proceeded it. If the current algorithm acts based off of lack of motion alone, the instructions can be consistently five seconds of pause to make an assured sprite. This would not be too difficult to implement and would only require running the tests again with different parameters.

Alternatively, instead of the measurement of motion being standard deviation of previous n frames, it could be the mean average of all previous frames. This may require much more storage and potentially processing power, and would not work well initially. But the longer a person was in front of their camera, the more accurately it would be able to determine the amount of motion in comparison to the amount they normally move. If someone was trying to make sprites very quickly however, they may end up in a state of pausing for too long for the mean to be able to represent a threshold between the two amounts of motion.

Perhaps a clustering algorithm that sorted movement into two clusters (lot and none) would be able to more efficiently distinguish between them. There was not enough time to implement this for this project, however.

The worst performing algorithms were MOG2 h800 t32 sTrue f25 [5.3] and MOG2 h200 t2 fTrue/False f100 [5.4].

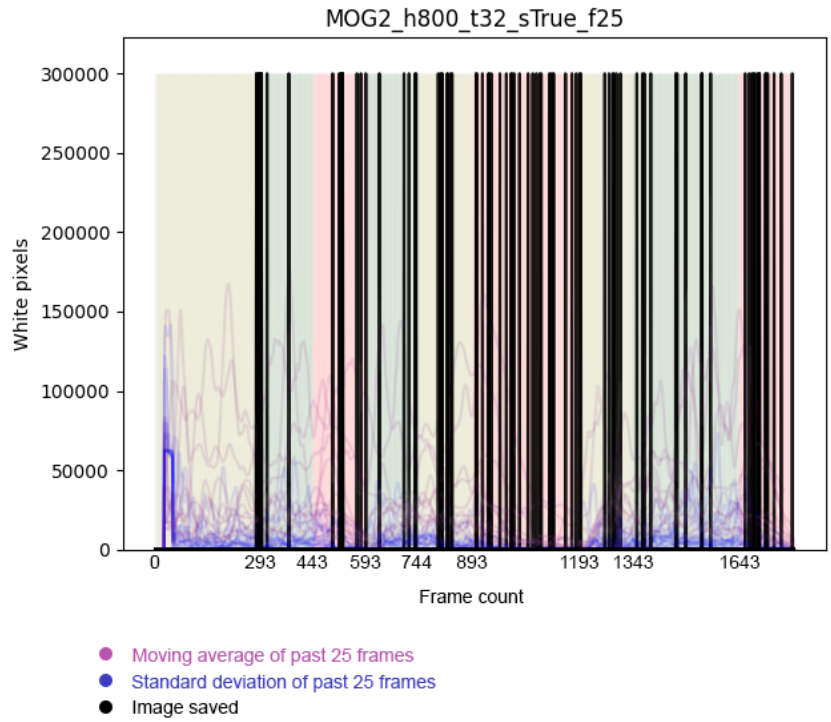

Figure 5.3: MOG2 h800 t32 sTrue f25 full graph

The former is very logical, as it was the variation with every parameter that would increase the amount of images saved. This was the largest amount of sprites saved, at 109 images saved over all test videos. It also saves images throughout the video, with little correlation to movement.

The latter two algorithms were the two that produced the least number of sprites. They are identical bar the shadow detection boolean value. Only the False graph was produced, as they are virtually identical. Only 12 images were generated over 18 test videos per algorithm, and of those 12 only 9 were in a suitable location to save an image. It is slightly odd that a history of 200 as opposed to 100 created this outcome, as it would be expected that the minimum values would create the most extreme data.

5.1 Future Work

There are many aspects of this data collection and testing that could be improved.

Seaborn was looked into, in the hopes that the visual representation of the data could be made more clear and aesthetically pleasing, but this was deemed not worth the time. Seaborn is a library used to improve the visualisation of graphs generated by matplotlib. It allows for more control over colours and layout of the data. Overlaying the various moving averages and standard deviations was a quick solution to be able to see all of the data together. The ideal solution was a single line graph with error bars denoting these.

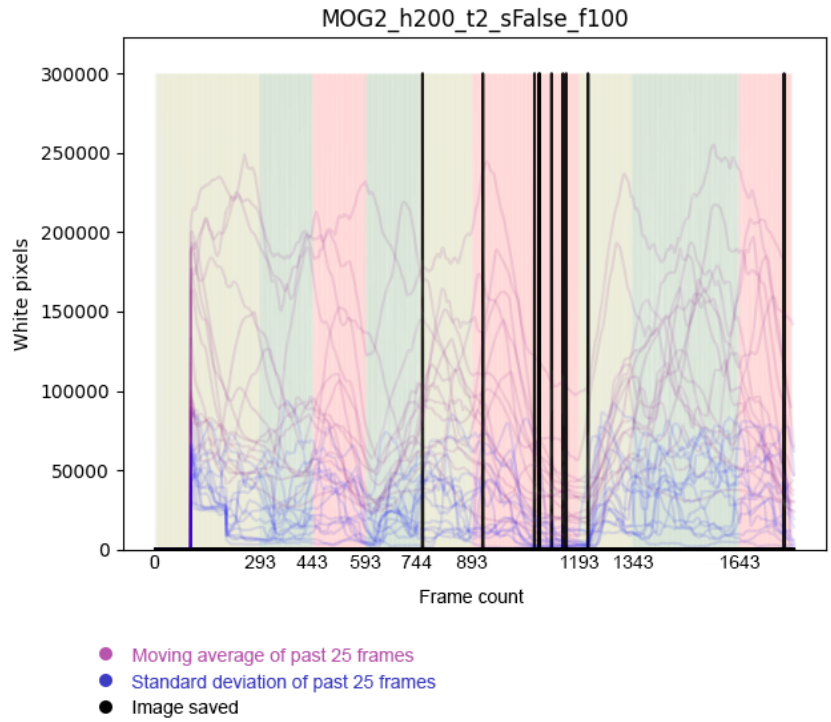

Figure 5.4: MOG2 h200 t2 sFalse f100 full graph

In the testing of algorithm parameters, only the MOG2 algorithm was tested. It was assumed that it would perform better than KNN. It would be interesting to see if KNN actually performs better than the best selected MOG2 algorithm.

More diverse test data would also improve the project. In the test videos, there was a lack of different sizes of bodies, with most models being approximately average size. There was also only one person of colour in the test data. Additionally, the majority of test data was of people standing, with no data representing people in wheelchairs or sat quite far from the camera.

Parameterising the threshold at which images are saved may also yield interesting results. Currently only standard deviations below 2000 are considered to be a lack of movement, a number conceived in early tests. There was little reason for this threshold, just that it seemed to perform well with the default MOG2 parameters.

When generating the json file for a sprite, there is a section available for any code blocks included in the sprite itself. In theory, this means that when a sprite is generated, there could be an option to generate it with a fully fledged game or animation attached to that sprite. This could be very useful for outreach fairs, in which children may be interacting with the code for only a very short period of time, ranging from a few seconds to a couple of minutes.

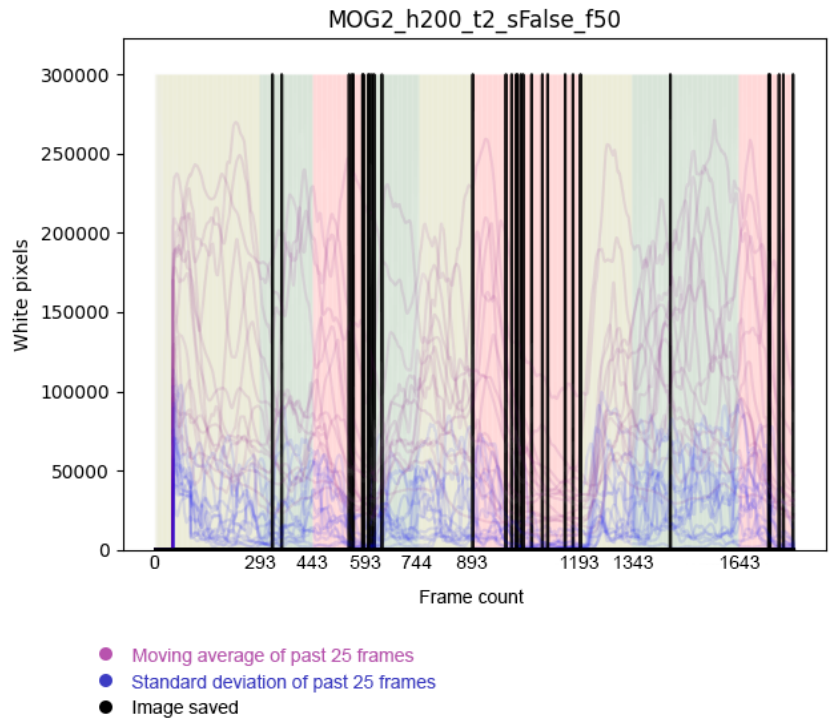

Figure 5.5: MOG2 h200 t2 sFalse f50 full graph, practically identical to MOG2 h200 t2 sTrue f50 full graph, but generated one more sprite when shadows set to False, so was considered to perform worse

Chapter 6

Evaluation

While this product did not fulfil all of its initial planned requirements, I am still very proud of the outcome of this major project. I applied data science principles in the end of the project and software engineering principles in the beginning, despite not having much confidence in either of these fields beforehand.

I chose to use python for the entire project, my first programming language, but one I had not used in years. Over the course of this degree, I have been wanting to use python again, and I am happy to find that I am more comfortable using it than ever. The code itself is neatly formatted and well commented. Although I used many external libraries, there is still a good amount of my own code making the project work as intended.

Implementation of both sections of the project were reasonably seamless. I came up against various issues, but whenever I did, I put in extra time before each meeting to either fix the problem, or have a good enough question to learn how to fix the problem quickly. Time management has traditionally been a struggle for me, but this project was split into small sections, so completing each week did not feel insurmountable.

If more thought had been put into the project as a whole at the beginning, it would have been more obvious from the start of the project that using python on the web would not be possible within the time frame. Seeing as I was desperately trying to improve usability wherever possible, I had hoped the project would be accessible from my website at dave.fish/sprites. Before the project began, I worked on making sure I could correctly push to my website while keeping the information on it secure. It was a shame that this was never used, but the project is still reasonably accessible. I intend on including the downloadable executable file on that web page in future, so it may end up on the web in one sense anyway.

There was not much tidying of the code at the end as there was not too much code to tidy. Through the duration of the project I organised the code into what was immediately helpful and what had been tested but was no longer useful by itself. This made it very easy to see what I was actively working on, and kept improving the readability of code as I moved it from file to file. While this uses the traditionally frowned upon practise of copying and pasting code, it forced me to think about which lines were helpful as it evolved.

I used some good programming practises throughout, such as consistently giving variables clear names and dividing the work into sections visually. Unfortunately, I did not use many functions in the code. I only used one in the main file so that it could be imported. And the functions used in datainator.py were largely just calling a single piece of code that may have been better organised as individual files. It could be argued that the use of functions would not be necessary due to the lack of repeated code within nice_sprites.py, but it still could have improved readability.

When the direction of the project changed, I struggled with re-planning the rest of the dissertation. I knew it would take longer than one week to gather and interpret data on motion detection, so I knew I would have to change multiple weeks of the plan. Removing one week from Easter was not difficult, but the time left for final touches did not feel quite enough. As the plan changed, so much code was created with little to no forethought. Given the time restraints, just making the data work however I needed it to was prioritised over neat code. Given more time, I would like to improve this, perhaps so that I can write similar code in the future.

The methodology following was poor. I did not take the agile module, nor did I do any programming in the second year group project explaining methodologies. I attempted to follow agile iterative development during the project, but struggled with sticking to it and the terminology associated with it. The work felt incremental and well divided, but this may not have worked had the scale been bigger and the project not as easily decomposed at the start.

Overall, the project produces a sprite given only a live video feed. That is what the project set out to do. I am so pleased with how far it has come, the speed at which I achieved this, and the research I completed to ensure it performs well. I believe the final product will be able to be used in outreach so that children and adults can feel more involved in the code they write. This project was fun to create, and will create fun in future, maybe even helping new people to fall in love with programming.

References

[1] Various, https://scratch.mit.edu/about, 2011.

[2] Google, https://developers.google.com/mediapipe, 2024.

[3] Various, https://www.geeksforgeeks.org/reading-and-writing-json-to-a-file-in-python/, 2024.

[4] ——, https://www.w3schools.com/python/python_file_write.asp, 2024.

[5] ——, https://www.w3schools.com/python/python_json.asp, 2024.

[6] ——, https://www.quickprogrammingtips.com/python/how-to-calculate-md5-hash-of-a-file-in-python.html, 2024.

[7] ——, https://www.geeksforgeeks.org/md5-hash-python/, 2024.

[8] ——, https://stackoverflow.com/questions/1855095/how-to-create-a-zip-archive-of-a-directory, 2024.

[9] ——, https://practicaldatascience.co.uk/data-science/how-to-zip-files-and-directories-with-python, 2024.

[10] ——, https://www.askpython.com/python-modules/zipfile-module, 2024.

[11] ——, https://note.nkmk.me/en/python-zipfile/#add-files-with-the-write-method, 2024.

[12] ——, https://www.w3schools.com/python/python_file_write.asp, 2024.

[13] ——, https://stackoverflow.com/questions/3207219/how-do-i-list-all-files-of-a-directory, 2024.

[14] ——, https://www.pythoncheatsheet.org/cheatsheet/file-directory-path, 2024.

[15] ——, https://www.programiz.com/python-programming/examples/file-name-from-file-path, 2024.

[16] ——, https://docs.python.org/3/library/datetime.html, 2024.

[17] ——, https://www.programiz.com/python-programming/datetime/strftime, 2024.

[18] ——, https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html, 2024.

[19] ——, https://docs.opencv.org/4.x/de/de1/group__video__motion.html#ga818a6d66b725549d3709aa4cfda3f301, 2024.

[20] ——, https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html, 2024.

[21] ——, https://github.com/danielgatis/rembg, 2024.

[22] ——, https://pyscript.net/, 2024.

[23] Vaious, https://cx-freeze.readthedocs.io/en/stable/, 2024.

[24] Various, https://stackoverflow.com/questions/5458048/how-can-i-make-a-python-script-standalone-executable-to-run-without-any-dependen, 2024.

[25] ——, https://stackoverflow.com/questions/45951964/pyinstaller-is-not-recognized-as-internal-or-external-command, 2024.

[26] ——, https://matplotlib.org/, 2024.

[27] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 2, 2004, pp. 28–31 Vol.2.

Appendices

Appendix A

Use of Third-Party Code, Libraries and Generative AI

1.1 Third Party Code and Software Libraries

1.1.1 In-built python imports used:

argparse, datetime, csv, hashlib, imghdr, json, math, matplotlib.pyplot, os, pandas, shutil, statistics

1.1.2 Pip installed python imports used:

cv2 - Open source computer vision library used for video input, processing, and motion detection `https://opencv.org/`

rembg - Tool for automatically and easily removing the background from images `https://github.com/danielgatis/rembg`

numpy - Additional numerical and scientific processing functions `https://numpy.org/`

1.2 Generative AI

No generative AI was used in this project.

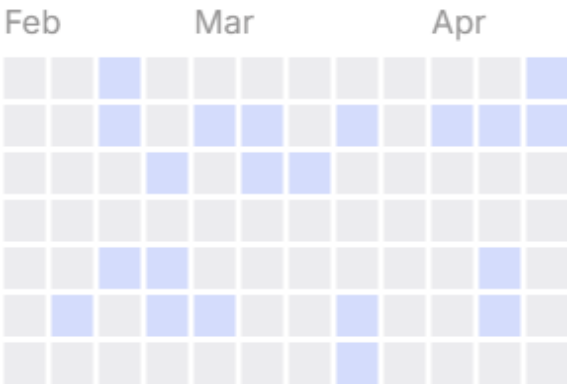

Figure A.1: Gitlab pushes

Appendix B

Ethics

The consent form used when collecting video data of people was taken from the MMP website. The form template was originally made by Neil Taylor.

2.1 Work direct comparison

Usability Study Participation Agreement: FORM B

Introduction

Thank you for agreeing to participate in my user evaluation study for my undergraduate dissertation project: **Turn me into a sprite.**

This is for the module CS39440 Major Project as part of my **Computer Graphics, Vision, and Games** degree at Aberystwyth University.

The aim of this study is **to turn people into a sprite, usable by scratch. It involves taking a live video feed of a person and automatically extracting relevant frames to be a sprite costume and removing the background from them.**

Confidentiality

I, **Dave,** will keep all responses securely. I will retain the signed consent forms. These will not be shared with the University unless I am required to do so to demonstrate that appropriate consent was gathered.

The data that is gathered during this study will be analysed and used in relevant sections of my dissertation. Copies of the raw response, without any analysis, will be provided with the technical work for my project, as evidence that I have conducted this study. However, the data will be checked and anonymised before it is included in the report.

The reports are read by members of staff in the University as part of the assessment process. Typically, this will mean approximately 3 members of staff. An external examiner from another University might also read the report.

The original data completed by participants will be deleted after the module result is released, and no later than the end of September 2024.

Withdrawal of Consent

You are entitled to withdraw your consent to use your evaluation as part of the study. The analysis will be completed and submitted as part of my work by **Friday 3rd May 2024**. If you wish to withdraw your consent, please contact me by **12noon on Tuesday 30th April 2024**.

The department keeps reports for approximately 1 year and might be shared with other students and members of staff as an example of work from this year. If you wish to withdraw your consent after the document has been sent for binding and submission, it would be possible to request that the project is not used for any purpose other than assessment, therefore limiting the number of readers.

Viewing the Analysis

As a participant, you are entitled to see a copy of the summary results and analysis from the report. You can request to see a PDF copy of the relevant sections of the report. Please email me at: **dad58@aber.ac.uk** if you would like to see a copy.

Agreement

Please note that any data you provide does not enjoy legal privilege, i.e. cannot be used in a court of law.

In signing below, you agree to the above terms about the storage and processing of the data.

Signed: _____

Date: _____

Version February 20, 2023

Figure B.1: Consent form

	Planned work	Actual work
29/01/2024	Planning phase	Planning phase
05/02/2024	Planning phase	Planning phase
12/02/2024	Create a sprite manually using python	Create a sprite manually using python
19/02/2024	Create a sprite automatically using python	Create a sprite automatically using python
26/02/2024	Save images from a video feed	Save images from a video feed
04/03/2024	Background subtraction	Motion detection
11/03/2024	Getting it on the web	Background subtraction
18/03/2024	Add instructions and tidy UI/UX	Gather data for motion detection improvement
25/03/2024	Easter	Easter
01/04/2024	Easter	Easter
08/04/2024	Easter	Process data
15/04/2024	Final tidying of code and project as a whole	Process and display data
22/04/2024	Final tidying of code and project as a whole	Final tidying of code and project as a whole
29/04/2024	Final tidying of code and project as a whole	Final tidying of code and project as a whole

Wider project work	Wider project work
Software engineering	Software engineering
Rest / Other work	Research
	Rest / Other work

Table B.1: The initial plan of work vs the end work completed