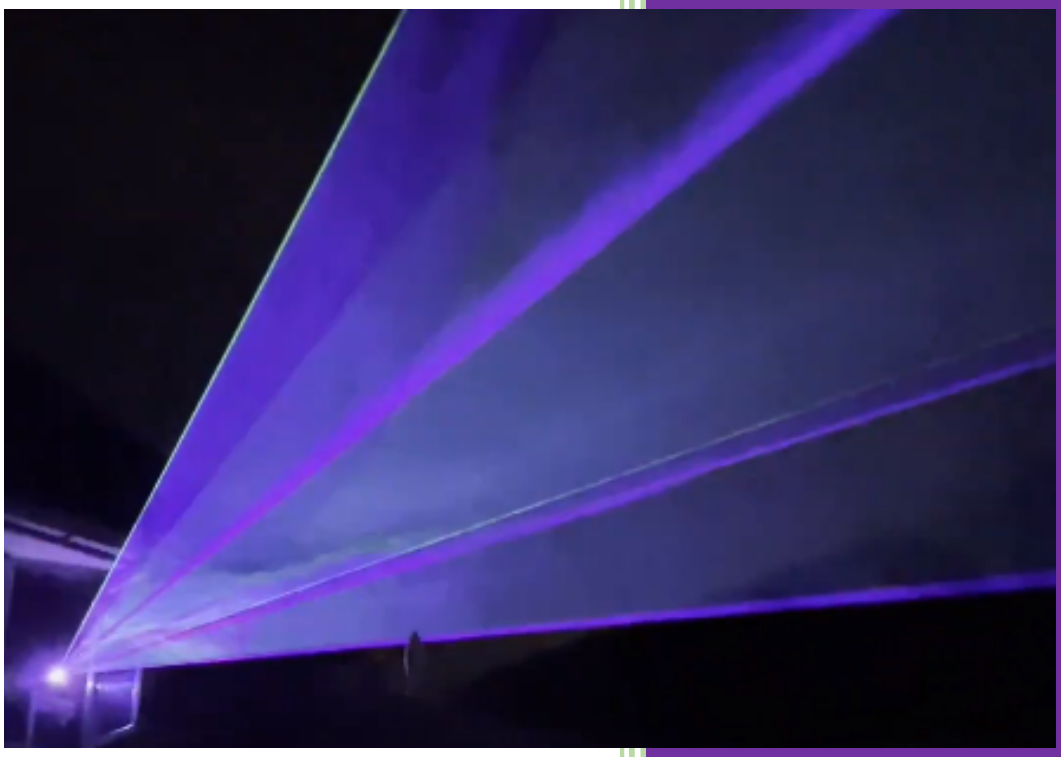
xxxxx :

xxxx

Path – A laser system



Leo Hammett

Xxxxx xxxxxx School

xxxxx : xxxx

Contents

1.Analysis

1.1 The Problem

Since the invention of the blue LED in 1994, high efficiency lighting has slowly entered all our homes. With this, stage-lighting has become more affordable and consumer friendly, with some households beginning to rent these devices for their home events.

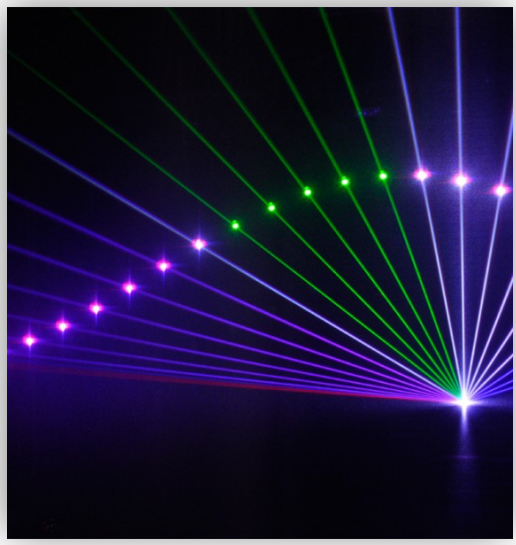


Figure 1: A 3D output from a laser

Along with the LED came the laser diode, which is a high efficiency way of making laser beams. Laser enabled devices have grown in accessibility over the past decade, primarily due to lower costs. When I refer to lasers from now on, I specifically mean stage lasers which can create multiple laser beams in different directions with the aid of two mirrors. Like the image to the right.

The problem is people cannot control lasers effectively unless they program them directly or use an alternative solution. The alternatives (which I will outline below), do solve this problem but mine will be made specifically with the intention of allowing people with little understanding of lasers to use them.

In summary, my aim is to make software that can allow users with little to no knowledge prepare a personalised light show.

1.2 Research

My first piece of research is into how laser systems physically work:

1.2.1 How do these laser systems physically work?

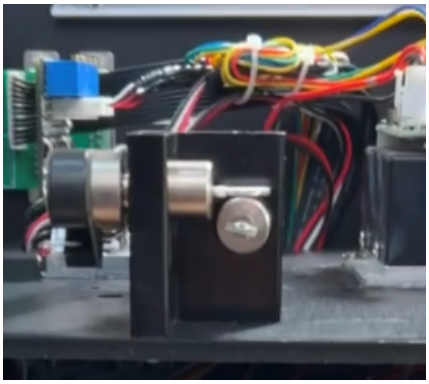


Figure 2: Mirrors inside of my laser

The stage laser has two parts to it. The first part is the laser which is comprised of three (or more) sub-lasers: R, G & B which each have varying levels of brightness to allow for all the different colours, it's also worth mentioning that the laser can change colours and levels of brightness for all intents and purposes, instantly.

The second part of the laser is made up of two mirrors. These mirrors change rotation thousands of times per second, which reflect the inputted beam into a specific direction.

By moving the beam around you can form what look like 2D planes but are simply lines moving very rapidly. This is the same way that the effect of multiple beams is made.

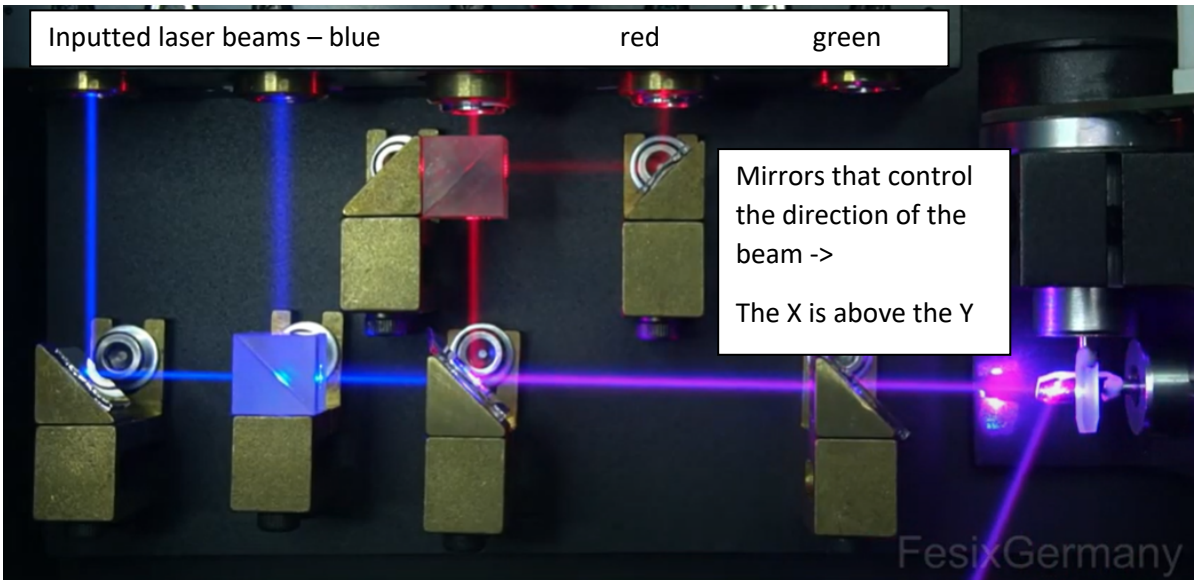


Figure 3: A birds eye view of a laser interior

The image above shows how the mirrors change the direction of the beam precisely. The video I used to understand this further was https://www.youtube.com/watch?v=57RMK1En_yc.

Most people understand it by comparing it to a CRT.

1.2.2 Research into the software, protocols, drivers & safety:

To find out my next steps for building a system like this, I looked on several websites. Some of the most useful ones were:

https://github.com/sebleedelisle/ofxLaser

This code by Seb Lee Delisle was practically my inspiration, it essentially converted commands into a script for lasers. I contemplated about building a GUI on top of this but felt a C# system would be a good addition to the laser ecosystem.

https://github.com/Grix/helios_dac

This repo was what ran the driver I ended up buying, I spent a lot of time looking into this github as well as: https://github.com/Grix/ildagen which is the code for the alternative solution LaserShowGen

https://www.youtube.com/@seblee

This was probably where I got my inspiration for my whole project

https://bitlasers.com/helios-laser-dac/

This was more information on laser DAC's

https://www.ilda.com/

If you hear the word ILDA, essentially Ilda is the laser protocol and main foundation, made 37 years ago.

https://pangolin.com/

Another alternative solution.

https://www.youtube.com/watch?v=17Q6My9OT70

This was where I learnt about safety considerations for a Laser.

I also emailed the creator of LaserShowGen and SebLeeDelisle. I was really happy to receive a reply from Gitle at LaserShowGen, I have shared the email chain in Appendix 1. I had hoped to learn how they drafted plans for their software, what their thought processes were to get them from the ideas to some form of realised implementation (I was optimistic they may share as DAC is open source). Whilst this was not possible, they did signpost me to further material that I could access to learn more.

1.2.3 How computers interface with the lasers

For the computer to control the laser they need to be able to change:

- The voltage applied to the X and Y galvanometers (the mirrors attach to these)
- The voltage given to the R G and B diodes.
- Any safety shutters or emergency features

All the information is sent over an ILDA (International Laser Display Association) protocol. The protocol manages all this information and is over 20 years old.

1.2.3.4 Systems safety

An aspect of this project is safety. Maintaining appropriate safety warnings is somewhat an objective of this software however this isn't a priority as no warnings can replace safety training. Any end user would be given short training upon delivery of the laser.

The main safety guidelines (although official training is necessary) are:

- No mirrors or reflective surfaces (apart from the in-device mirrors) in the exposure zone. This would include household items like reflective light switches.
- Sufficient warnings around the exposure zone such as labelling on the internal doors or external entrance/exit points.
- No people in the exposure zone, unless the operator understands the safety requirements behind audience scanning.
- Safety zone for the user to operate from out of exposure zone
- This safety warning extends to humans and animals who may both be at risk of blindness.

To ensure safety, one of my objectives for measuring the success of my project is how effective the warning messages are to people (Whilst this isn't technically difficult, it is essential and therefore quantifies my software's effectivity).

The safety regulations in the UK are very simple: under no circumstances can you point a laser in the eyes of any person driving a vehicle and if anyone gets injured (blinded in this case) due to negligence, the operator of the laser is liable.

1.2.4 Purchasing a stage laser projector

When I first found out about lasers, I was wondering if I would be able to afford one. I began my search not knowing what to look for in the systems. I found out that the three things I would be looking for are:

- Brightness
- Speed
- Range

After a while of looking, I found the Cameo Luke range of lasers, they had a reasonable brightness (although 700mW was quite low for a stage laser), the speed was 30 Kpps, which is the professional standard, and it did have an ILDA connection. However, it wasn't particularly bright and it was £531 before delivery. Now I needed to find a cheaper laser.

The cheapest option was to have it shipped over from China directly via Ali Express, a 3W laser with 40Kpps scanners costs much less. The compromises for this cheaper price were: No built-in safety systems; The laser beam had a much higher divergence, and it only has about 10 different colour combinations because of how its dimmers are super low quality.

If price were out of the option in hindsight, I would have brought a high-power Pangolin laser as these are of a high quality. However, I am a 17-year-old in possession of a laser system so I'm not going to complain.

1.2.5 Buying a DAC

To interface via ILDA, you need to output to a DB25 port, this is an ILDA port and isn't common on modern desktops. To output to this, you need to purchase a DAC which converts digital signal, in my case USB to analogue ILDA signals. I decided to purchase a Helios DAC for two reasons:

- They were open source meaning that there would be more compatibility with software's.
- It was the cheapest and I was massively over budget.

1.3 Investigation into Alternative Systems

For my analysis I need some inspiration into how current systems function, the main alternatives are LaserShowGen and Pangolin QuickShow. There are several alternative solutions, but the rest seem to be expensive versions of LaserShowGen. I also wanted feedback into existing systems both from myself and my end user.

1.3.1 Alternative System 1 – LaserShowGen

To help aid my understanding of the challenge, I did research into already present solutions. LaserShowGen by BitLasers is a freemium piece of software that allows the user to program lasers by drawing vector shapes in a coordinate grid and then selecting colours for them. It then uses a pre-built script that converts these "frames" into points which then get sent to the DAC.

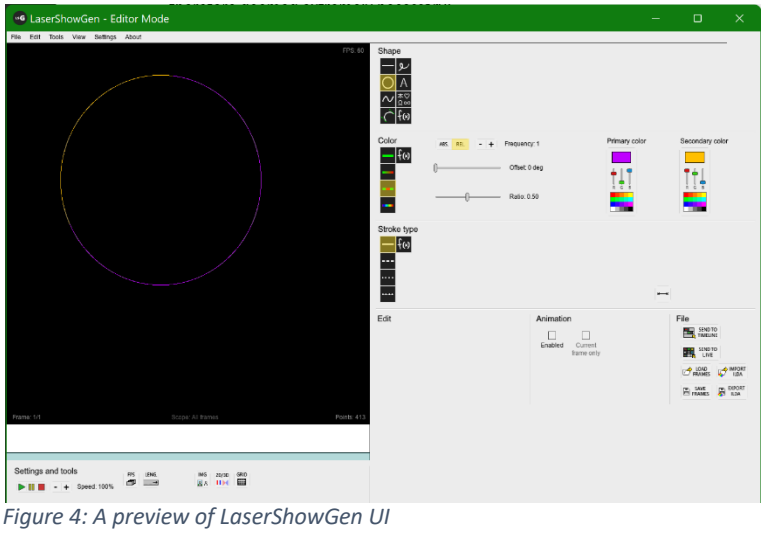


Figure 4: A preview of LaserShowGen UI

Pro's:

- It is intuitive for basic tasks (gets confusing around animations), and it can project basic frames in a few clicks.
- The coordinate grid style of input felt ordered and intuitive, leading to a simple experience.

Con's:

- It is slow to create light shows:
  - Each animation takes at least a minute to create.
  - Its challenging to modify already present animations.
  - The software doesn't autosave and therefore has hundreds of save prompts.
- It is challenging to make lines parallel or connect at ends etc due to poor snapping to grid feature.
- It is difficult to understand the settings page which is quite necessary for calibrating the laser with more complicated graphics. This is important if I am designing the page for beginners.

1.3.1.1 How it Works

LaserShowGen works off an old game engine called GameMaker, the software has various tools to add items to a canvas, the canvas then uses a pre-built script to convert the vector imagery into a path for the laser to travel along. The software uses control objects for each mode of projection. This makes it easier to manage by a long way. Beyond this each UI element was an object and then a proprietary script managed the projections.

1.3.1.2 User Feedback from Software

Because the system is designed to work alongside music performances, the system has no audio feedback, however one piece of feedback from the system that I aim to replicate is the indication of selected tools via different appearances of buttons.

1.3.2 Alternative System – 2 Pangolin Quick show

Pangolin Quick Show is a similar piece of software to LaserShowGen.

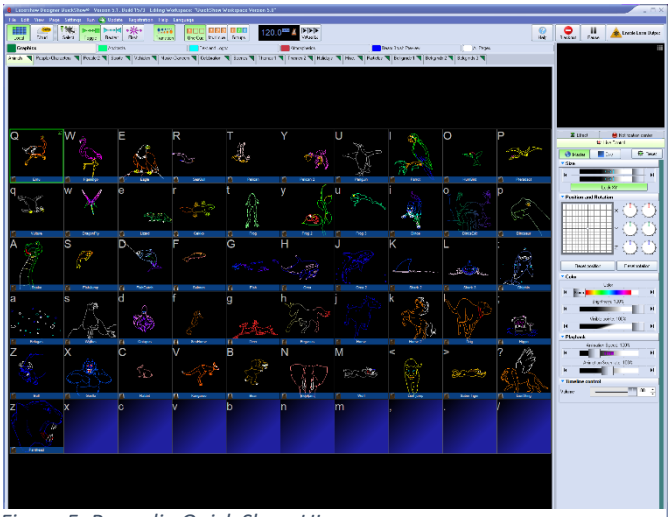


Figure 5: Pangolin Quick Show UI

Pros:

- The software comes with pre-built animations, allowing you to run high quality shows off the bat.
- The software is high quality, so I guess you would get more for spending more.
- Super advanced controls with very high accuracy.

Cons:

- The software looks and feels incredibly old.
- The software makes no sense to me at all.
- The software only works for pangolin lasers or DAC's (which cost more than £500)
- The software is better at showing pre prepared animations than really making them.

1.3.2.1 Further alternatives

The other available alternatives had one of three downsides:

- Outdated, sometimes too outdated to run
- Expensive and thus end up being exclusively for professionals.
- Designed only for one type of laser setup.

1.3.3 Summary of Alternative Software

1.3.3.1 User feedback

The main thing in common between all software's is that they had no sound, this is because they are designed to operate in a musical environment where interruptions wouldn't be heard or would interrupt the music.

To test my solution and alternative solutions effectiveness, we are going to make a demo scenario and have a typical end user attempt to use the solutions.

1.3.3.2 The typical scenario with an end user

The scenario used to test the efficiency of the system is a parent renting a system and making a display to project on the front of their house a happy birthday message. In this case, the software should in theory:

- Give the user ideas of what to display via icons representative of the tools and their functions
- Give the user the ability to change the settings given to the laser to improve resolution
- Give the user the ability to input their display in an intuitive manner
- Project the display using the settings inputted in an efficient way (i.e., making the laser travel an efficient path.)
- The user should be able to use the software without any previous knowledge and I shouldn't have to give them any hints as to what to do.

1.3.4 My end user's response when I showed them alternative solutions.

My end user is intended to be someone with average IT user skills (e.g. mouse, keyboard and experience with standard GUI's), but with no knowledge of coding and no previous experience of using lasers. However, it is important that the user should be educated on the dangers of lasers or should be supervised by someone who is educated on them.

1.3.4.1 Pangolin Laser

Because I didn't want to spend £500 on the software, I used the demo and only got a virtual output. I did however still challenge them to make a display of their choice as well.

1.3.4.2 LaserShowGen

LaserShowGen was much more user friendly; my end user was able to download, install and run the software. This observation gave me a lot of insight which is listed below:

- Once the software was running the UI was a bit overwhelming to begin with and the user asked if there were any help buttons, the only help they could find was a 40-page instruction manual PDF which they didn't want to read. The trick here I think is a less advanced but more intuitive approach with potentially advanced tools hidden further into the software.
- When the user first began, they tried adding text using a feature that required a font package to be installed. I learnt that the font would need to either work immediately, or not be shown. I'm not sure if this makes sense here, but I'm trying to emphasise the importance of a UI that naturally makes sense.
- They then found the symbols tool and started adding in symbols, by far the biggest lesson for me is how similar the editor needs to be to something like PowerPoint so that users feel they naturally understand it. When designing their layout, the user only needed help when the design didn't work like the Office or Apple productivity suite work.
- Another observation I had with this system is that when they tried to hit run and there was an error, the user didn't read the help - instead asking what went wrong and assuming the software was broken. The lesson here being that the software needs to automatically sort out errors.
- I also think the points counter needs to be more explanatory and obvious as users won't understand why the display is flickering unless this is more obvious.

- I also think some settings should be changeable in the main layout and it should make sense what they do.

Afterwards I interviewed the user and here are my notes below:

Q) What did you like about using the LaserShowGen software?

A) I liked the fact that it worked straight 'out of the box', I didn't have to load it. I liked being able to draw my designs with a mouse and see in real time the image on the display in front of me. I liked that it worked reasonably fast. I liked having the options of different colours for my designs.

Q) What did you dislike about using the LaserShowGen software?

A) I didn't like having a second cursor on the screen, as it made it impossible for me to join my lines properly to create images in a way that I wanted to. *(Interviewer note: user was describing an issue where a second cursor was appearing at a constant offset to the actual mouse).*

I didn't find it easy to select different colours, and found the RGB dials quite intimidating. I didn't know how to create an animation, or moving image. I couldn't work out how to edit my drawing retrospectively. Finally and maybe most importantly, I wanted to see my laser display on the wall, but when I clicked 'laser on', I then got a message box (which to me looked like an error message), saying that DACs weren't there which made me think it was broken. I didn't know if I should (or could) resolve this.

In the end the user ended up producing the following.

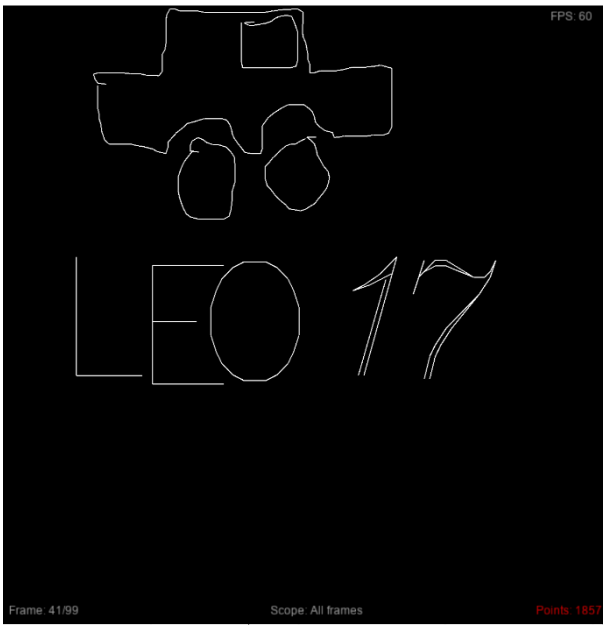


Figure 6: LaserShowGen Preview of end user's drawing

Figure 7: The output of the end user's drawing project

This would be projected on a garage door at the front of a house for example, however we spent a while testing and I live by a busy road so it could be a distraction to have flashing lights for drivers, so we opted for testing the software indoors.

1.4 Project Objectives: Performance Criteria

1.4.1 Required Features

I have broken down the objectives for this work into four areas:

- Safety
- Affordability
- Useability
- Customisability

Safety

- Ability by the user to turn off the laser using the software at any time.
- In person warnings given to alert users how to treat these devices with appropriate caution.
    1. These must explain that usage can cause blindness if eyes are exposed to the laser radiation
    2. These must explain that usage can cause fire
    3. These must be shown to the user before they hit play
- Appropriate algorithms to prevent hardware damaging requests to the device.
    1. These must block the laser from leaving the range 0&4096 as either side further will damage it.
    2. These must have a limiting maximum velocity that can be set in the code.
    3. These must have a limiting maximum acceleration that can be set in the code.
- Dark Mode so it can be used in the dark, and allow the user to see the surroundings without adjusting eyes to allow for situational awareness. To ensure brightness of the GUI isn't preventing users from being aware of anyone within the laser exposure zone.
    1. The software must have mainly black backgrounds
    2. There must be contrast between the text and background so it remains legible / useable when it is dark.
- File verification
    1. The software must be able to handle invalid files.
    2. The software must have a warning if the JSON has been edited but can still be opened (so that any damaging requests are at the users responsibility).

Affordability

- Laser hardware needs to <£1000
- Laser must be bright.
    1. Beam must be visible at night with smoke.
- Efficient path generation algorithms to prevent the need for more unnecessarily costly hardware.
    1. The algorithms must be able to run more than 1 frame per second.
    2. The algorithms must be able to distribute spacing based on detail.
- No paid for code required.
    1. The code file must be able to run without paying for any other software (drivers auto install and are free).
- Works with a Helios Dac that can project 30000 kps.

1. The dac should be able to plug into a computer and connect like any peripheral would.

Useability

- Intuitive GUI to allow any clients with IT skills to generate images on their own.
    1. Intuitive meaning that it doesn't require any previous training to use.
    2. The end user should ideally be able to understand the GUI when shown to them.
    3. Single graphics layout so there's no hidden functionality.
    4. Lines drawn from the first click.
- Responsive software, that runs "out the box".
    1. The laser can run as soon as you click toggle laser, no setup required (other than setting up the hardware physically).
- No console usage, all in a display.
    1. The system should be able to run purely via a graphics based interface.
- GUI that can handle variable amounts of detail.
    1. The system should be able to draw a simple frame as well as a frame with large amounts of shapes and changes over time.
    2. The system should have guidance (i.e. snap into place)
- GUI that can modify lines graphically post drawing.
    1. The frame should be changeable without deleting shapes.
- GUI that can allow for mistake correction.
    1. There should be the option to delete specific shapes if they weren't necessary etc.
- A reliable GUI
    1. Consistent output given input
    2. Doesn't crash if invalid data is inputted

Customisability

- Choice of colours, minimum of 6 colours.
    1. The shapes should be able to have many different colours, ideally at different times.
- Ability to project dynamic animations.
    1. The system should be able to loop a changing projection animation that isn't being modified by a user at the time.
- Variable animation times.
    1. The length of the transitions must be variable, i.e. one be fast one be slow.
- Variable animation positions.
    1. The lines should be able to change positions during animations.

Optional extra features

- The ability to draw more than straight lines (i.e. Bezier curves, drawn lines).
- The ability to modify the laser settings & timeline settings without editing the code.
    1. This would be done via a settings page.
- The ability to swap which point is shown in the line properties using the selection tool.
    1. This is a niche thing that would be nice but could cause many bugs.

1.4.2 Limitations and Constraints

- The driver is in C++. This means I will need to convert the driver to C#.
- The user is not expected to have an advanced level of ICT knowledge so the software needs to be intuitive.
- I have no contacts who have used this type of hardware before.
- I have limited ability to test the system as I do not want to disturb neighbours at night.
- DAC can only run on Windows or Mac devices.
- The laser has to be delivered before I can even begin programming as I need to work out how to project with it. This will use up a large portion of time.
- My end user was hard to meet up with for regular feedback.
- Lasers are usually business proprietary and the software is expensive because of it, only recently have prices dropped making it commercial. i.e. unexplored territory for private use.
- The lasers that are going to be used will not be the highest quality lasers (I am a student), the software will need to account for this by:
    1. Creating efficient pathways
    2. Warning users when the shapes aren't projectable – or would require too high a definition.
    3. Finding a detail – frame rate compromise to hide the complex information from users.

There are no constraints concerning accessibility or data protection as the system needs a fully aware operator and the system stores no personal data.

2. Documented Design

(Please reference analysis for selecting and purchasing hardware requirements).

Before planning, I began building my GUI as a large amount of this code is around making an effective UI.

The first part of the documented design process was to make a list of more specific requirements:

2.0.1 List of tasks to work on (my way of breaking the objectives into more specific goals)

- Scripts that convert vector shapes into points.
- Scripts that send these points to a HeliosDac (if the software was more professional it would work with other DAC's, but I lack the financial ability to create and test this)
- A GUI that:
    1. Allows for the input of lines.
    2. Allows the modification of shapes via a text-based editor.
    3. Uses a timeline system:
        - That allows for different shapes over different times.
    4. Allows for the selection of the following tools:
        - Select shapes and modify them via the GUI.
        - Create lines
        - Change the selected colour.

2.1 How I designed my code

My approach to convert this into a more detailed plan was to create a mock up GUI, I then thought about the code of several days.

2.1.1 Layout & User Interface (UI)

The first iteration of the design process was drawing out a preview of what I wanted the system to look like, the image attached is the first example.

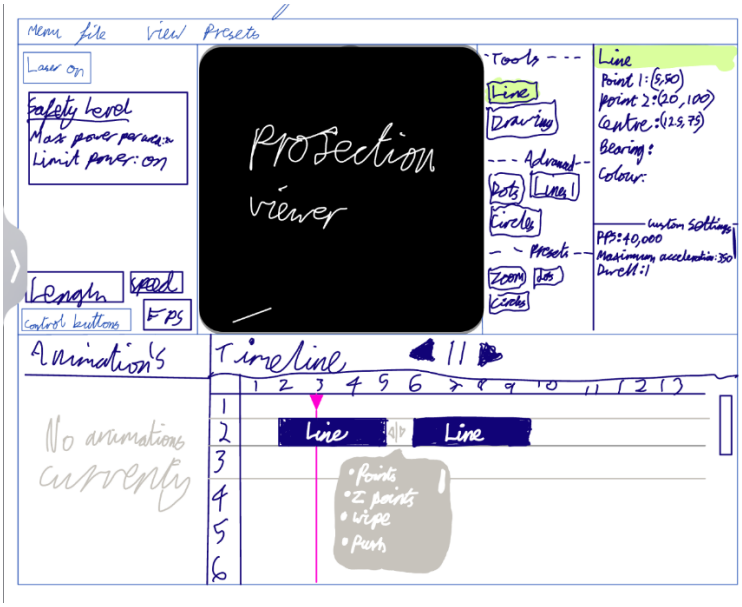


Figure 8: The first sketch of a GUI

I am keen to create an application containing most of the features shown above as this would effectively meet my criteria. This desktop layout requires quite a complicated set of planning otherwise implementing features or debugging later will be incredibly difficult. That is why the planning will take multiple prototypes and learning before the final code.

After making this UI sketch, I made a prototype system, and by doing so, I quickly realised certain features would work more efficiently in other ways.

The main changes to the application were:

- how the timeline will work
- animations

These animations would be controlled in separate shape objects with start points and end points, thus allowing the animations to be embedded later.

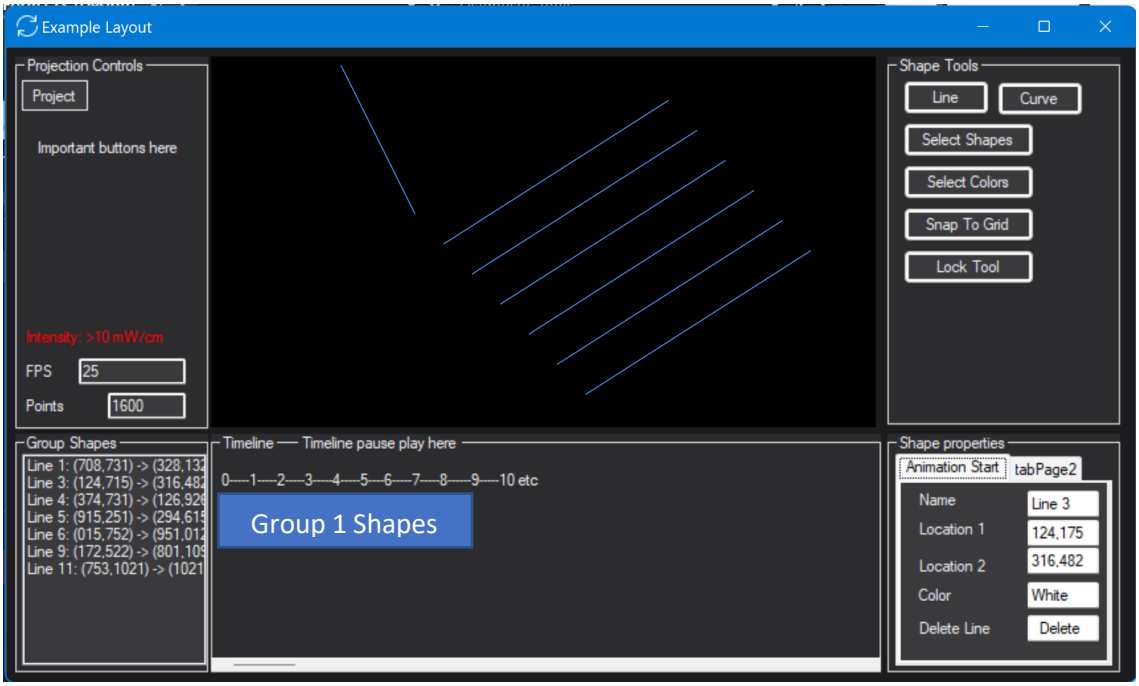


Figure 9: First visual studio design of the code

The main changes in this further iteration of the GUI are:

- I removed the rotation features (as these can be accomplished 90% effectively by moving the points around.)
- I removed the concept of pre-set's, as this is something which is less technically impressive but requires a significant investment of time. The timeline is now more suited to visual studio's graphic tools.

2.1.2 The GUI above fulfils the following objectives.

- Ability by the user to turn off the laser using the software at any time.
- Dark Mode so it can be used in the dark, and allow the user to see the surroundings without adjusting eyes to allow for situational awareness. To ensure brightness of the GUI isn't preventing users from being aware of anyone within the laser exposure zone.
    1. The software must have mainly black backgrounds
    2. There must be contrast between the text and background so it remains legible / useable when it is dark.
- Intuitive GUI to allow any clients with IT skills to generate images on their own.
    1. Intuitive meaning that it doesn't require any previous training to use.
    2. The end user should ideally be able to understand the GUI when shown to them.
    3. Single graphics layout so there's no hidden functionality.
    4. Lines drawn from the first click.
- Responsive software, that runs "out the box".
    1. The laser can run as soon as you click toggle laser, no setup required (other than setting up the hardware physically).
- No console usage, all in a display.
    1. The system should be able to run purely via a graphics based interface.
- GUI that can handle variable amounts of detail.
    1. The system should be able to draw a simple frame as well as a frame with large amounts of shapes and changes over time.
    2. The system should have guidance (i.e. snap into place)
- GUI that can modify lines graphically post drawing.
    1. The frame should be changeable without deleting shapes.
- GUI that can allow for mistake correction.
- There should be the option to delete specific shapes if they weren't necessary etc.

2.2 High level diagrams of the code

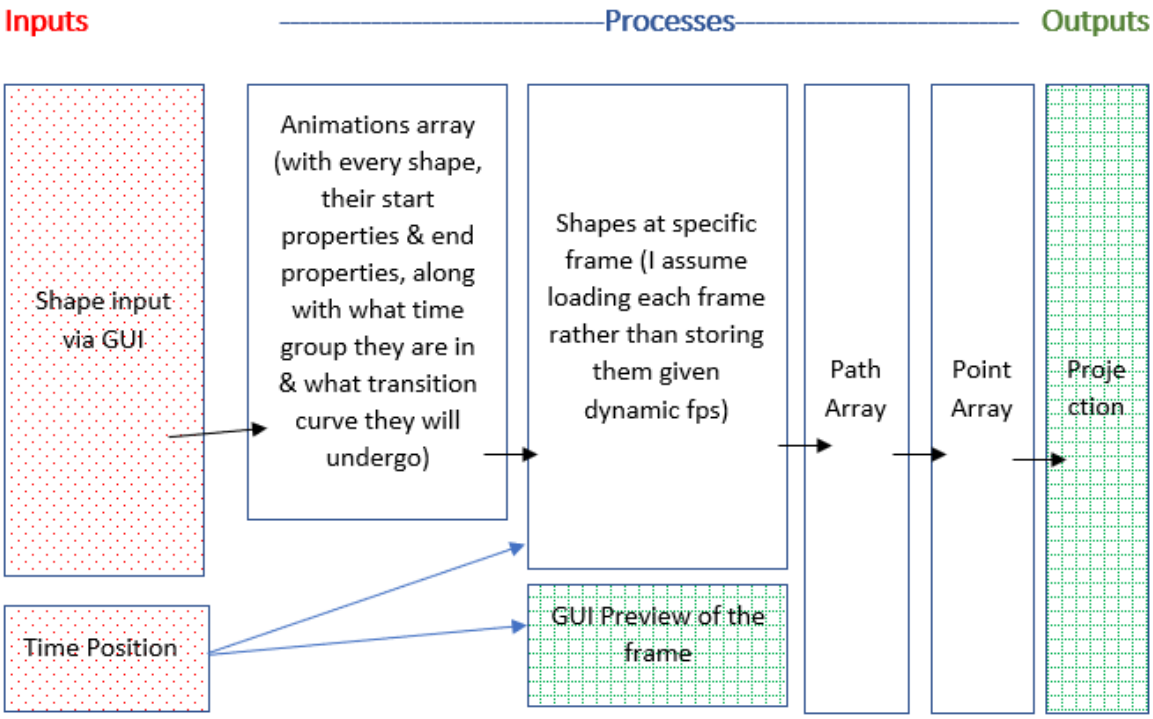
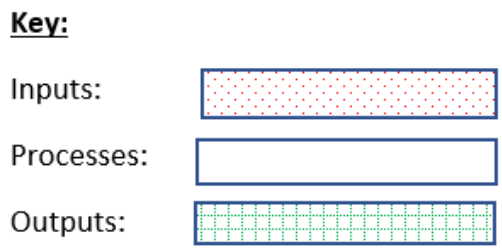


Figure 10: The above diagram was the first diagram of the code, to summarise how I would use objects and methods to convert the inputs into outputs.



Using the diagrams above and on the next page, I began working out how I would lay out the classes. Once that was worked out, the problem became a lot more simple.

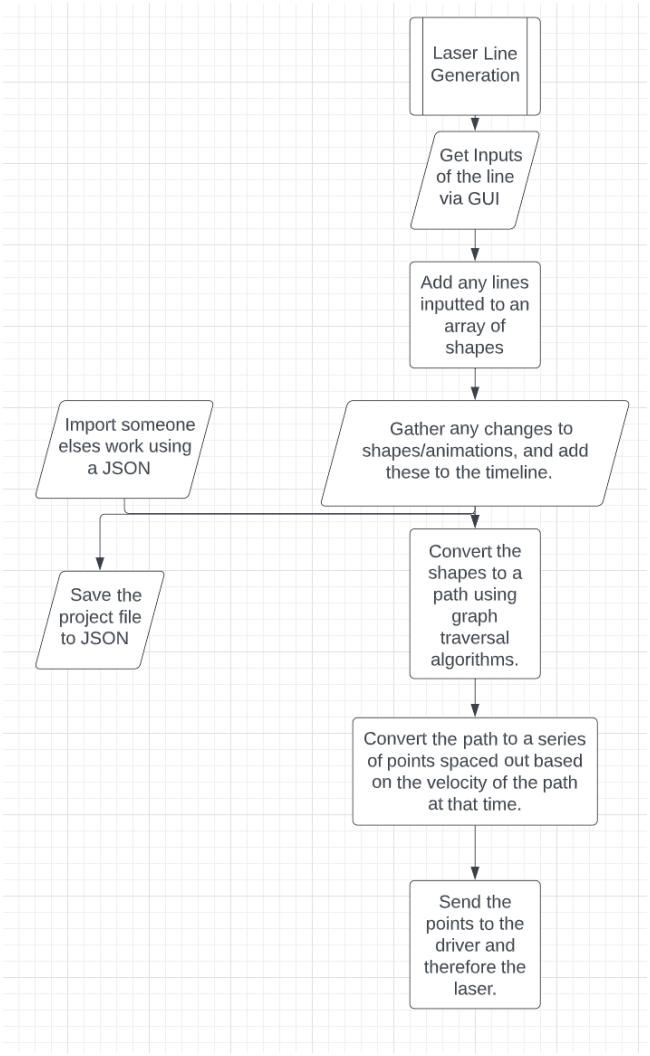


Figure 11: High Level Flowchart of my code

To the left, you can see a second diagram in flowchart form that included the file saving features.

This is an aid used when explaining my system.

Using these, I could then get to work designing each of the components.

2.2.1 Designing the shape classes

After planning out the high level diagrams, I designed the class to contain all the shapes.

A single PathProject class will be used that contains all the PathLine shapes that contain all its PathLineFrames that contain PathLinePoints.

The relationship can be seen in this class relationship diagram.

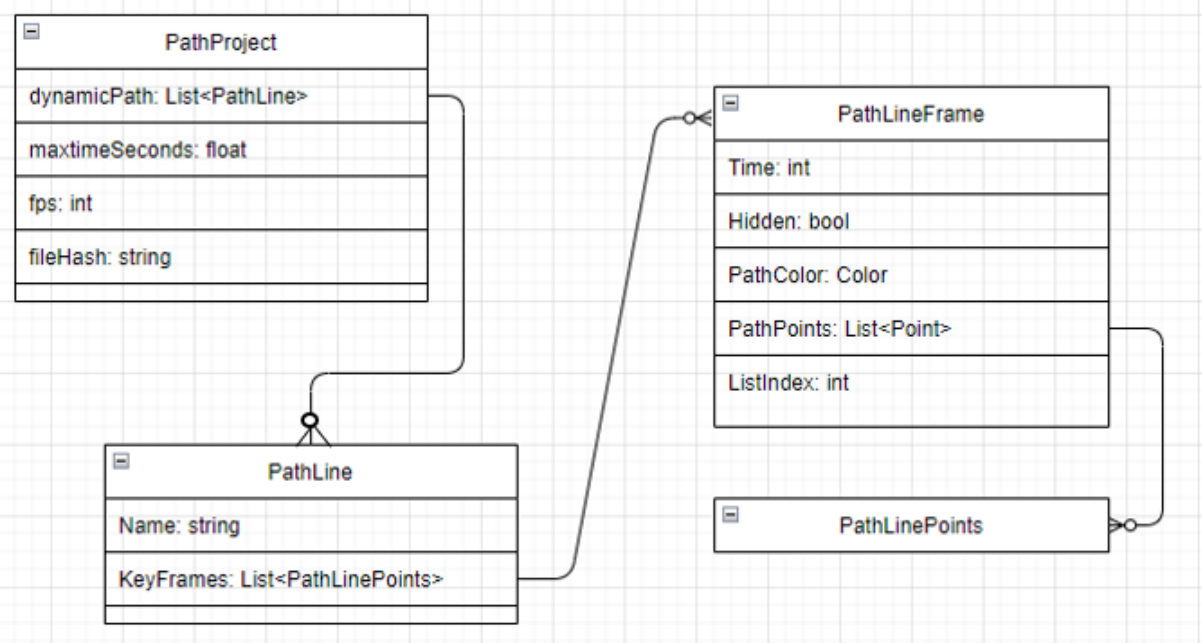


Figure 12: Class relationship diagram to show how a single object can store a whole session

2.2.2 Designing the line class

2.2.2.1 PathLine

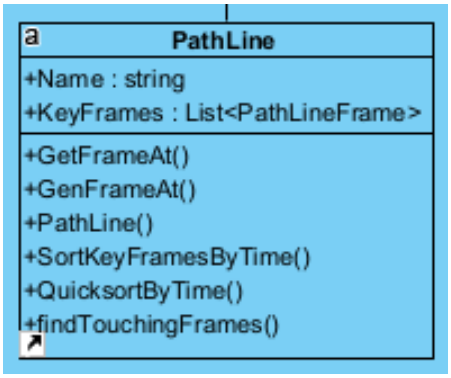


Figure 13: UML Diagram of the PathLine class

This class is very important to get right, it needs to contain the ability to change throughout time. The system will essentially become a list of keyframes, each of these keyframes will contain subframes.

The Path Line property contains a name and then a list of keyframes, the main uses come from the methods.

The methods GetFrameAt and GenFrameAt look similar but the big difference is that GetFrameAt gets a reference to a keyframe of the time inputted, this is either an existing frame or one it has added that has been generated via GenFrameAt, which is a disposable copy. Both are useful.

The PathLine constructor is required for importing the JSON files. SortKeyFramesByTime uses a quicksort algorithm to make sure the keyframes are in order, this will likely be needed for find touching frames & QuicksortByTime is a part of sortkeyframesbytime.

2.2.2.2 PathLineFrame

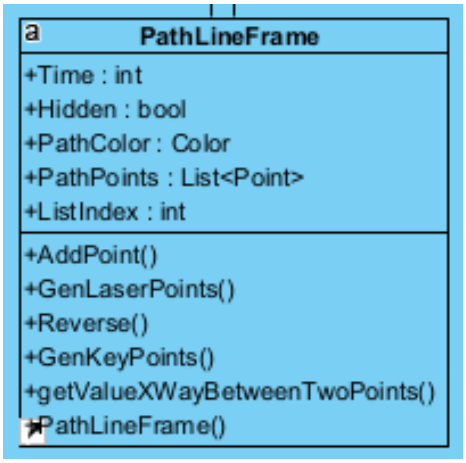


Figure 14: UML Diagram of the PathLineFrame class

This class (shown to the left), is the other half of the storage system, the pathLineFrame. This is used to store keyframes as well as any frames of the current time. All methods are explained in the technical solution and below:

- Time: is the time of the frame in the animations
- Hidden: whether the line is a display line (shown) or a transition line (hidden). Reference to GenLaserPath (explained later)
- PathColor: colour of the line
- PathPoints: the locations of the ends of the line
- ListIndex: the list of the PathLine object that contains the frame

The Points are a list in case I ever decided to implement shapes. However, the GenLaserPoints isn't compatible with multiple points so you will need to update it first.

The methods here are all clearly named so I will only explain them briefly:

- AddPoint adds another point to pathpoints
- Reverse swaps the order of the points (useful for generating a path as if the points are generated the wrong way round it's useless).
- GenKeyPoints converts the pathpoints into a list of KeyLinePoints, these are a special class that I will talk about later.
- The getValueXWayBetweenTwoPoints() is used for generating animations and is needed for working how close to either frame it should be (if it's right next to one frame and a larger distance from another it needs to look a lot like the one it's next to). Currently, this is linear as I didn't intend to make an animation progress editor in this version, however, I do plan to add this feature which will look similar to the one in Adobe After Effects.
- The GenLaserPoints method is extremely challenging, I originally wanted to have a system with variable acceleration but this is of no value unless travelling between points, the options for optimising this are quite limitless. I opted for using SUVAT to calculate the varying distances as the point moved across. The system would then reach maximum speed at a point and then would slow down when it had the required distance left.

2.2.2.3 PathProject

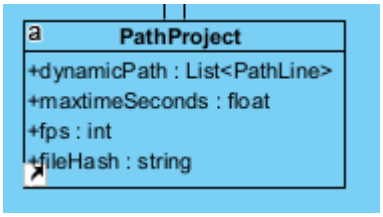


Figure 15: UML Diagram of the PathProject class

The whole list of shapes (as there would be multiple lines) would then be stored in the single PathProject class, this also had the FPS and time of the project, the fileHash is used later.

2.3 Other classes

2.3.1 Settings

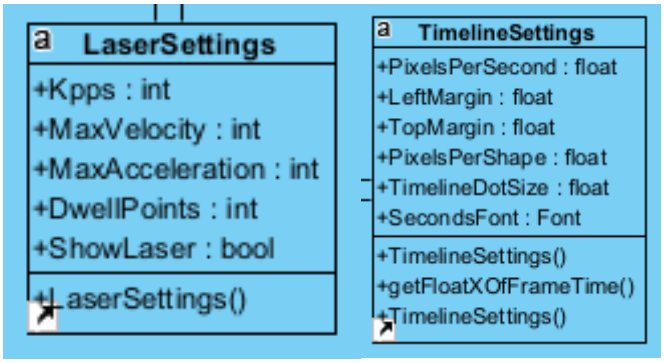


Figure 16: UML Diagram of the LaserSettings class

Figure 17: UML Diagram of the TimelineSettings class

These two classes are made such that eventually they can be edited via a settings page, for now the settings have to be edited via code as I don't know how to send objects between form windows. (Something for the summer holidays! 😊)

2.3.2 LinePoint

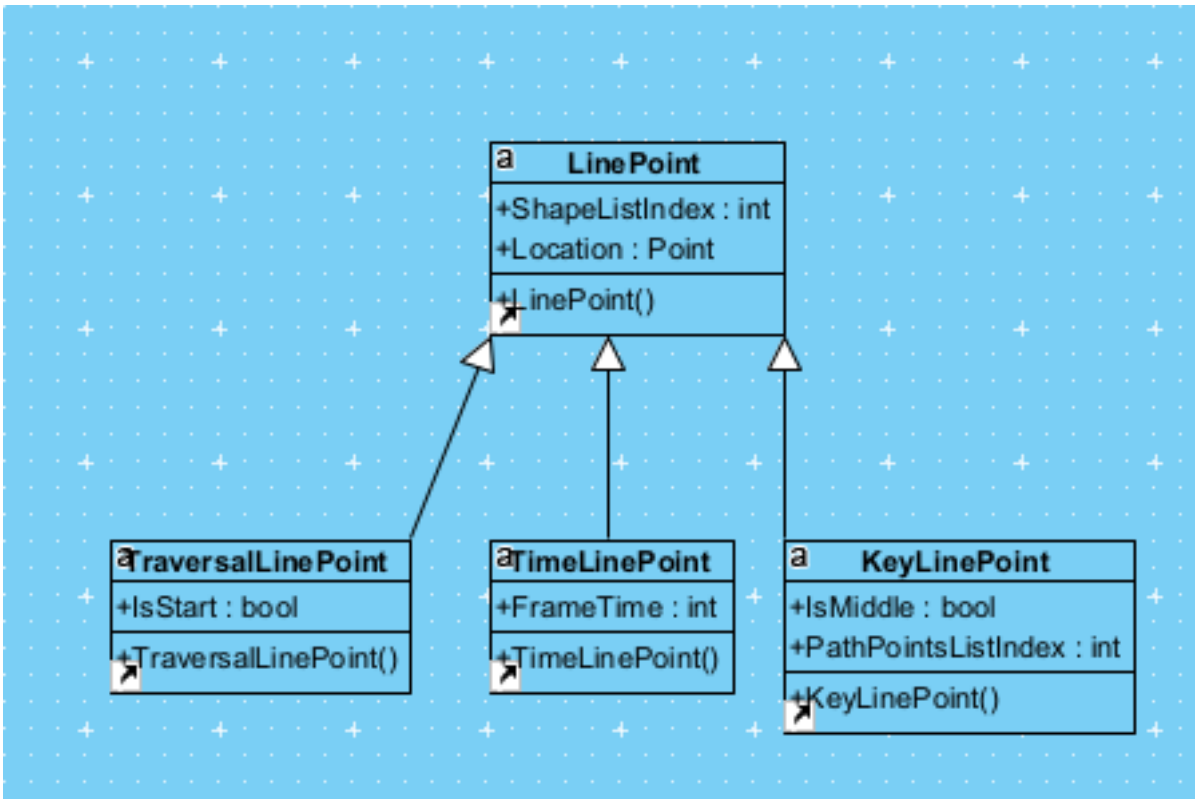


Figure 18: UML Diagram of the LinePoint class and its subclasses

The other object I would need is LinePoint. I added in specific objects based on the properties they need, as most properties weren't overlapping. Each one was used in its specific methods.

2.3.3 Visual studio generated class diagrams

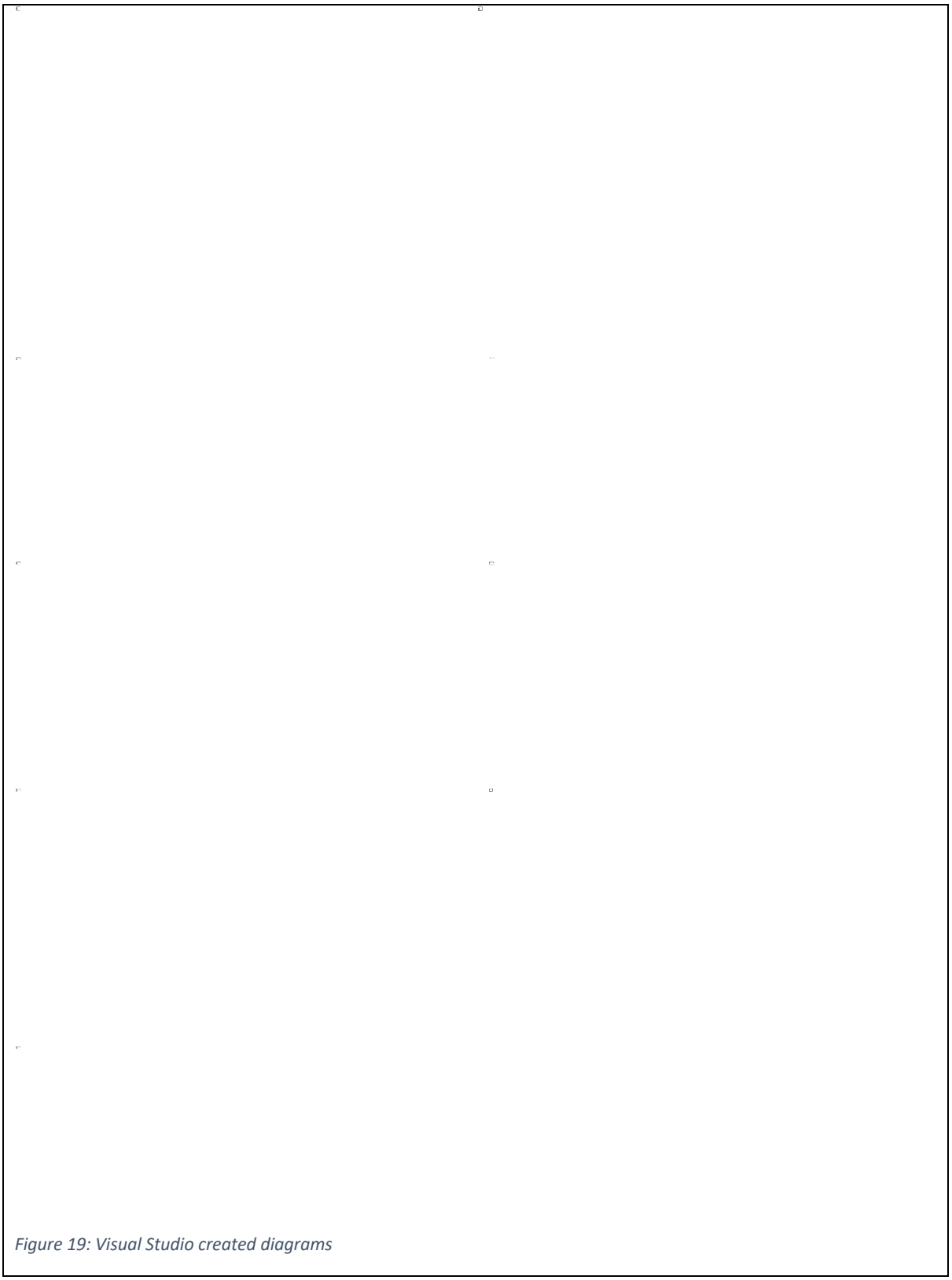


Figure 19: Visual Studio created diagrams

2.3.4 How does the application become data?

The application itself can be fully used without leaving a trace due to the nature of its output. However, one feature of the system is the ability to save and open files. I am going to explain how this work alongside **explaining what part of the code isn't mine**. I have also clearly commented on which code is not mine. Most of this is about using hashing to verify the file is not corrupted.

2.3.4.1 Saving to file

Steps:

1. Show the file explorer dialog.
2. Set the projects fileHash property to ""
3. Get the hash of the project file using the get hash method and set the fileHash property of projects to it
4. Write the project object (of class PathProject) to JSON file using someone else's method.

2.3.4.2 Opening the file

Steps:

1. Turn the laser off because the projection is about to change
2. Begin a try catch loop to catch any invalid files
3. Open a file dialog so the user can select the location
4. Set the project to the json
5. Make a copy of the projects new filehash
6. Set the projects fileHash property to ""
7. Check if the hash generated from project object is the same as the one set to it
8. If they don't match show a warning
9. Set the time to 0 (make a method to do this)
10. Set any selected points and lines to 0 as there might not be the selected items anymore.

2.4 Methods that need planning/ones I am most proud of

2.4.1 Find Touching Frames

I wanted to use a modified binary sort for this and felt that a recursive method would be the best decision here. My final code had notation O(logn) which is helpful when you are running this code for every animation.

2.4.1.1 Pseudocode

```
SUBROUTINE findTouchingFrames(timeOfFrame, touchingPoints, startingOccurance = False)

    IF startingOccurance THEN

        touchingPoints ← {emptyLineAtTime-1}

        touchingPoints ← touchingPoints + keyframes

        touchingPoints ← touchingPoints + emptyLineAtLargestPossibleTime
```

```
        ENDIF   #This bit is so that the first iteration can be setup. The key thing is that there must
                #be a frame greater and less than.

        midFrameIndex ← touchingPoints.Count() / 2;   # We always check the middle of the array
                                                      # in binary sort

        IF(touchingPoints[midFrameIndex].Time = timeOfFrame) THEN

                Return touchingPoints[midFrameIndex]

        ELSE

                IF(timeOfFrame > touchingPoints[midFrameIndex].Time) THEN

                        IF (timeOfFrame < touchingPoints[midFrameIndex + 1]) THEN

                                return (touchingPoints[midFrameIndex],
                        touchingPoints[midFrameIndex + 1])

                #If the one below is less than the time and the one above greater than return these
                # touching frames (also sorry about the formatting)

                        ENDIF

                        return findTouchingFrames(timeOfFrame, touchingPoints.
GetRange(midFrameIndex - 1, 1 + touchingPoints.Count() - midFrameIndex),false)

                ENDIF

                IF(timeOfFrame < touchingPoints[midFrameIndex].Time) THEN

                        return findTouchingFrames(timeOfFrame, touchingPoints.
GetRange(midFrameIndex - 1, 1 + touchingPoints.Count() - midFrameIndex),false)

                ENDIF

        ENDIF

        Return {}


ENDSBROUTINE
```

2.4.2 SortKeyFramesByTime

Here I implement quicksort, which I have learned partially from the following video and trial and error: https://www.youtube.com/watch?v=SLauY6PpjW4&t=15s.

I used a recursive routine with Keyframe's as the initial variable. Essentially the algorithm selects a pivot and makes sure everything on the left is less than and everything on the right greater than. Then you quicksort each side and due to its recursive nature it keeps breaking it down until there's nothing left! The main benefit here is that this now takes O(logn) instead of O(n)

Also if there are incorrect casings in my pseudocode it's probably autocorrect.

2.4.2.1 QuicksortByTime Pseudocode

```
SUBROUTINE QuicksortByTime(list)
        IF (list.COUNT = 0) THEN
                return {}
        ENDIF
        left = {}
        right = {}
        pivot = list[list.COUNT / 2];
        FOR i ← 0 TO list.COUNT
                IF (list[i].Time < pivot.Time)
                        left.ADD(list[i])
                ELSE
                        Right.ADD(list[i])
                ENDIF
        ENDFOR          #Split the list into two lists, one of all the less than's and one  #of the rest
        leftSorted = QuicksortByTime(left)
        rightSorted = QuicksortByTime(right)
        list.CLEAR ()
        list.ADDRANGE(leftSorted)
        list.ADD(pivot)
        list.ADDRANGE(rightSorted)
        return list
ENDSUBROUTINE
```

2.4.3 GenLaserPoints

For this we are using the SUVAT equation S = ut + 1/2at^2. This is because I need to calculate the varying distances. This equation is derived in my physics and mechanics classes. The equation is used to calculate displacement of an object given constant acceleration and an amount of time. Which is why I am using It here. The only other mathematical consideration here is that I cannot let the line accelerate to a higher velocity than the maximum permitted velocity because it could break the laser. I work out when to stop accelerating by calculating the time it will take to accelerate to the highest velocity. I also check to see if the velocity would be higher anyway. Once the acceleration has reached its highest velocity I have to maintain a constant velocity and reverse this once I am halfway across the line.

We are essentially teaching the laser how to accelerate.

2.4.3.1 Pseudocode

I am going to shorten currentLaserSettings to CLS

```
SUBROUTINE GenLaserPoints()  ##This is inside of a pathLineFrame object with colour pathline
                             ##points etc

    IF PathPoints.Count > 0 THEN

        beginningDisplacements ← QUEUE

        endingDisplacements ← STACK

        FOR i IN RANGE(0,CLS.DwellPoints)

                beginningDisplacements.Enqueue(0)

        ENDFOR

        acceleratingTime ← CLS.velocity/CLS.MaxAcceleration   #little rearrangement of v

        currentDisplacement ← 0

        halfDisplacement ← getDistance(this.pathPoints[0],this.pathPoints[1])

        t = 0

        WHILE t < acceleratingTime  && (0.5*t^2*CLS.MaxAcceleration) < halfDisplacement

                beginningDisplacements.ENQUEUE(0.5*t^2*CLS.MaxAcceleration)

                endingDisplacements.PUSH (0.5*t^2*CLS.MaxAcceleration)

                currentDisplacement ← (0.5*t^2*CLS.MaxAcceleration)

        ENDWHILE

        IF currentDisplacement < halfDisplacement + (CLS.MaxVelocity / 2) THEN

                WHILE currentDisplacement + CLS.MaxVelocity < halfDisplacement DO

                        currentDisplacement += CLS.MaxVelocity

                        beginningDisplacements.ENQUEUE(currentDisplacement)

                        endingDisplacements.PUSH (CurrentDisplacement)

                ENDWHILE

        ENDIF ##IF the accelerating didn't get both sides to the middle

        currentPoint ← new HeliosPoint()

        currentDisplacement ←  0

        horizontalScaleValue ← (PathPoints[1].X - PathPoints[0].X) / getDistance(this.PathPoints[0],
this.PathPoints[1])
```

```
        verticalScaleValue ← (PathPoints[1].X - PathPoints[0].X) / getDistance(this.PathPoints[0],
this.PathPoints[1])
        WHILE beginningDisplacements.Count > 0 DO
                currentDisplacement = beginningDisplacements.DEQUEUE()
                currentPoint.X ←  PathPoints[0].X + currentDisplacement * horizontaleScaleValue
                currentPoint.Y ←  PathPoints[0].Y + currentDisplacement * verticalScaleValue
                currentPoint.Color ← this.Color
                points.ADD(currentPoint)
        ENDWHILE
        WHILE endingDisplacements.Count > 0 DO
                currentDisplacement = endingDisplacements.POP()
                currentPoint.X ←  PathPoints[0].X + currentDisplacement * horizontaleScaleValue
                currentPoint.Y ←  PathPoints[0].Y + currentDisplacement * verticalScaleValue
                currentPoint.Color ← this.Color
                points.ADD(currentPoint)

        ENDWHILE
    ELSE
                Return points {}
     ENDIF
ENDSUBROUTINE
```

2.4.4 GenLaserPath

2.4.4.1 Pseudo-pseudocode (even higher level code for planning the pseudo code)

Declaring variables as we go

Generate a list of points to traverse

Skip if we have an odd number of points

Travel the first line in the list.

Remove the first and second points as they are traversed now

While there are points left

Find the closest point using a for and if loop.

Travel to the closest point using a hidden black line

If there are points left in the list travel across the line to the other point

Move to the next point

Remove the points we have just travelled

2.4.4.2 Pseudocode

In some of this code it's got explanations instead of variable names.

```
SUBROUTINE genShapeLaserPath(framePath)
    pointsToTraverse ← {}
    FOREACH line IN framePath
        pointsToTraverse.ADD (TraversalLinePoint(First in the path line, isfirstpoint = true))
        pointsToTraverse.ADD (TraversalLinePoint(Last point in the path line))
    ENDFOR
    QUEUE linesToGenPointsFor ←  {}
    IF (pointsToTraverse.COUNT > 1 && pointsToTraverse.COUNT % 2 ==0) #if greater than 1
    and #even
    THEN
        currentPoint ←  pointsToTraverse[0]
        #Traverse the first line vvv
        linesToGenPointsFor.ENQUEUE(framePath)
        nextIndex ← pointsToTraverse.INDEXOF(currentPoint)
        pointsToTraverse.REMOVE(currentPoint)
        currentPoint ← pointsToTraverse[nextIndex]
        pointsToTraverse.REMOVE (currentPoint)
        WHILE (pointsToTraverse.COUNT > 0) DO
          nextPoint ← NEW POINT REALLY FAR AWAY
          FOREACH point IN pointsToTraverse
              IF (GETDISTANCE(point,currentPoint) <
GETDISTANCE(nextPoint,currentPoint) THEN
                  nextPoint ← point
              ENDIF
```

```
                ENDFOR
                #Got the next point as it will be the closest point
                linesToGenPointsFor.ENQUEUE(new hiddenline from current point to next point)
                nextIndex ← pointsToTraverse.INDEXOF(nextPoint)
                pointsToTraverse.REMOVE(nextPoint)
                currentPoint ←nextPoint
                #Travel across the lines now
               IF(pointsToTraverse.COUNT > 0) THEN
                    IF(currentPoint.IsStart) THEN
                            linesToGenPointsFor.ENQUEUE(
                                    framePath[currentpoint.ShapeListIndex])
                    ELSE            #Add in the line
                            frame ← framePath[currentpoint.ShapeListIndex]
                            frame.reverse()
                            linesToGenPointsFor.ENQUEUE(frame)#Add in the line
                            nextIndex = nextIndex -1
                    ENDIF
                ENDIF
            ENDWHILE
            linesToGenPointsFor.ENQUEUE(line from current point to start)
        ENDIF
        Return linesToGenPointsFor
ENDSUBROUTINE
```

2.4.5 In my code I am also most proud of:

- The shapes storage and the flexibility of the animations, this should also be easily scalable. This took a lot of thinking to plan out an animation system, although it may look simple.
- The projection efficiency, in many cases the algorithm traverses efficiently with SUVAT systems. Ideally after learning more calculus, I want to design a system with variable acceleration to travel between points faster and with a final velocity in the right direction.
- The hardest part of this project was planning how I was going to interact with the driver, I discuss this below but honestly this was a lot of learning by trial and error. This was

incredibly hard because I had to use interop services and learn about a language I didn't know about.

2.5 Making my prototypes (plus additional design notes from the beginning)

2.5.1 My first prototype

After hours and hours of learning the basics of Interop Services, I finally made a piece of software that would run in c#. This software consisted of the following functions (these are not documented below using the actual function names):

- Start up the DAC (and therefore open the connection to the laser).
- Shut down the DAC.
- Ask how many DACs are connected (if this number is ever more than one I've made a mistake or are now using multiple lasers).
- Write a frame.

2.5.2 Making my second prototype

To help me plan my program, my approach was to draw a basic idea of what the software would look like as then I can plan how to implement the code for all of it. A screenshot of my prototype is shown below (I have also attached the prototypes code at the end):

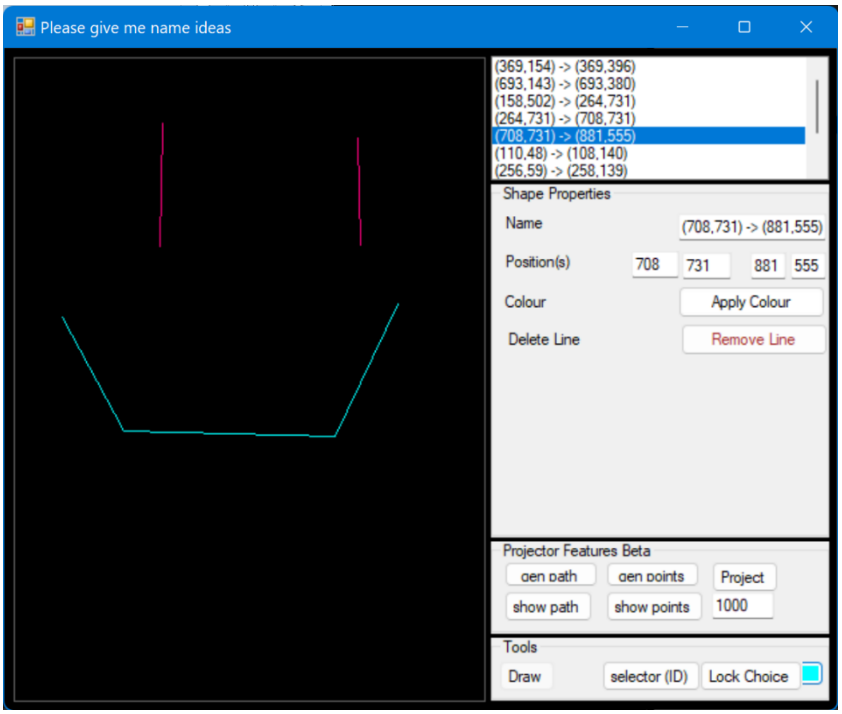


Figure 20: The first working version of my code, including a preview panel programmed entirely by myself

2.5.3 How I will run the timeline

The timeline will be an object in itself, that when invalidated will call a paint function, this will grab all the objects in the animations array, the objects in the animation array will be groups of shapes. These groups will have start times and end times and will be laid out in the timeline as rectangles.

Since windows has no pre built forms that seem to fit the requirements for this, I will create the timeline using the windows graphics feature.

2.6 Parts of the program that need designing:

These are parts that don't need working on much but still need some work (perhaps even just some thought whilst programming)

- Class charts - completed
- The Select Feature
    1. Work out which shape is closest – use a similar method to find closest point in traversal.
    2. How to draw the select GUI adjustments & transformations – will select the closest point and any adjustments will update the line directly. Any drawing was also treated as modifying a selected line (the selected line will be created on mouse down)
- How the scrolling will work on the timeline – Will resize the timeline inside a panel and visual studios auto scrollbars can fit it in.
- How the program will work with the UI – make flowcharts for any complicated elements. – I have discussed this.
- Laser conversion
    1. Animations to frames – A GenFrameAt feature will use findtouchingframes and then bias either frame depending on distance
    2. Frame to path – using genPath method mentioned earlier
    3. Path to points – using genPoints from earlier
    4. Sending points – this was done in my prototyping and uses a DLL
- Easy driver functionality – Windows manages drivers
- Settings page – This is an optional feature, not sure how to communicate across windows
- Software storage of settings – optional but could use save to file

2.6.1 High level plan of the software

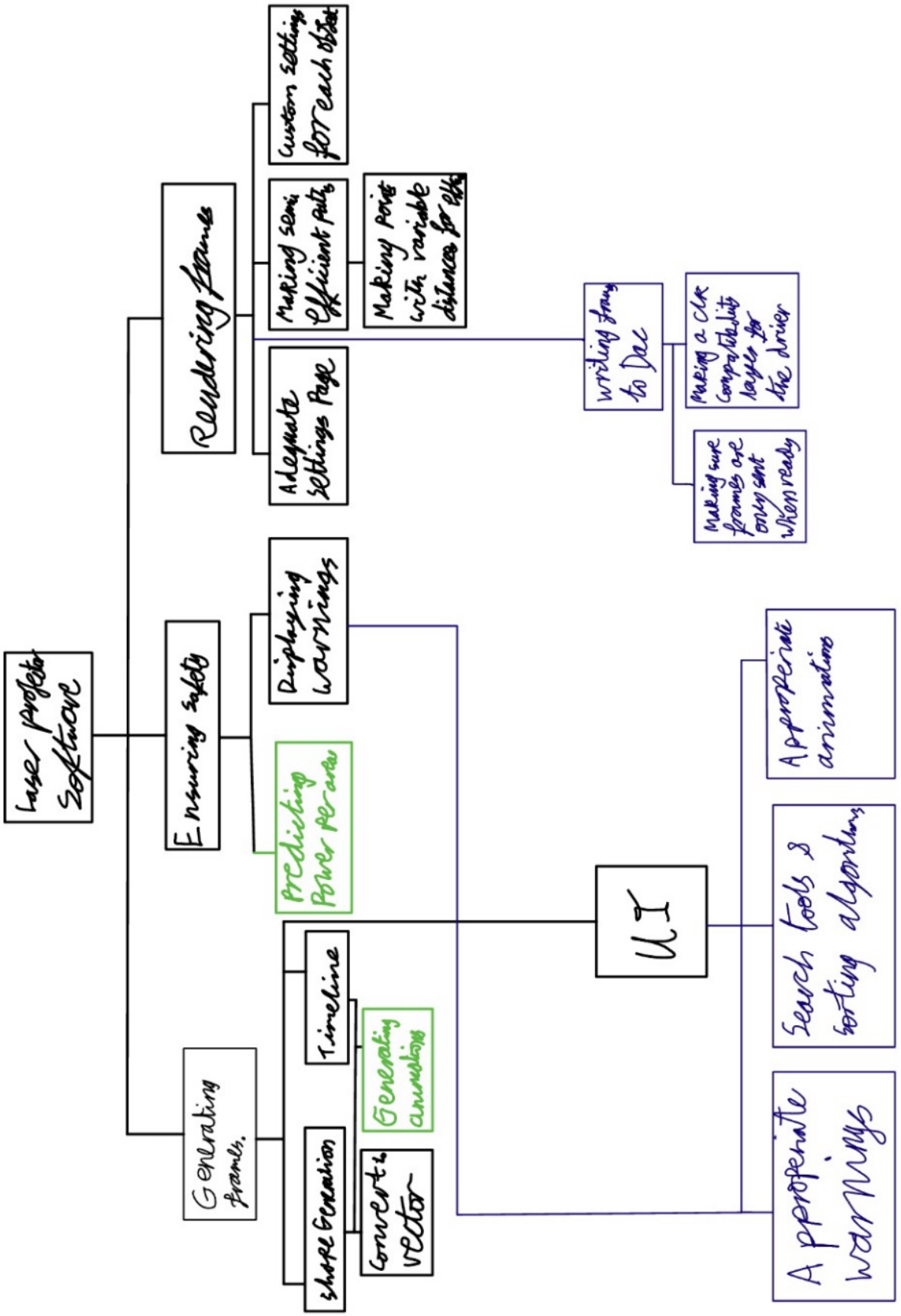


Figure 21:Early plan of the software from high level, pasted in so you can see how I've iterated over time

After looking at this I then revisited my hierarchy.

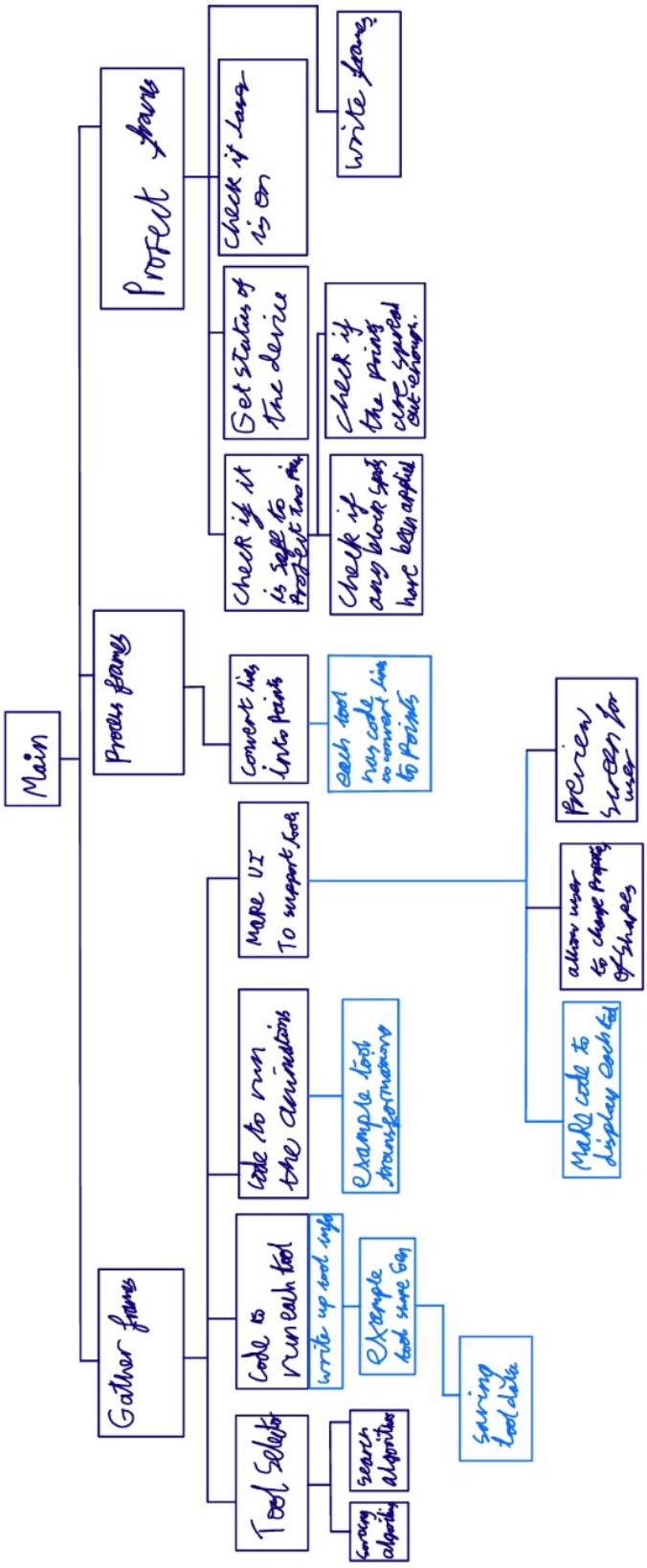


Figure 22: Another high level design of the code from earlier on

2.7 Writing frames

The first step in my code needs to be the writing frame's function. The sdk for this was built for C++, so my first step was to convert the functions into callable C# functions as this is the language my code will be using C#.

To write frames to my laser my first step (which has taken weeks and weeks to do), is to wrap my code in such a way that the frames can be written through C#.

The way I had to wrap the code is via interop services. I also had to learn about pinning variables etc. but the majority of the code was importing functions.

2.7.1 Managing shapes

For my code the most important part is managing shapes. Once I have found out how to do the following, the code will be much easier to produce:

- Store the shapes (short term)
- Preview the shapes
- Project the shapes
- Save the shapes (long term)

https://docs.microsoft.com/en-us/dotnet/api/system.windows.shapes.shape?view=windowsdesktop-6.0

https://docs.microsoft.com/en-us/dotnet/desktop/wpf/graphics-multimedia/graphics?view=netframeworkdesktop-4.8

2.7.2 More on how I prototyped my own system

To analyse more about the problem, I thought I would research how to convert the shapes into points.

My most effective method of learning was by doing, so I wanted to run the example program which came with the driver. This program was intended on projecting a horizontal line that travelled up and down for 150 frames or five seconds. This turned out to be rather difficult as the code wouldn't run for the following reasons:

- Libusb (a library needed for interfacing via usb) wasn't working with my USB ports
- The code itself was written for VS 2015 so I needed to download the iso for that just to convert the code to a format that VS 2022 could convert into something it could run.
- The driver was throwing up errors saying it needed to run in 32 bit.

The key reason for my struggles (apart from the code being written 7 years ago), was not knowing the programming language, C++. Once I ran the driver, I did some basic tweaking, such as changing the colour variables and the movement of the line etc. (this was the most rewarding bit of programming I had done as no errors appeared for a whole 5 minutes).

I decided to make an end goal for this prototype that would point the laser in the real world where my mouse hovered over a form. If my mouse hovered top left, the laser would point to the top left of its range. This would be a sufficient proof of concept that I could control the laser via a GUI based

application. The code would also be relatively simple as I wouldn't need to do anything to convert the point, I wanted the laser to shine on, to a point the DAC could understand. In the end I did this using a CLR (.NET) project.

One of my biggest constraints will be that currently I don't fully understand this area. I tried to extend my demo further but as I didn't understand C++ very well, I struggled with basic tasks such as lists due to requiring unmanaged and managed classes that I didn't understand. I learnt that I need to make the driver work in C# somehow.

From my prototype I learnt that the converting of shapes to points is going to be difficult but possible. Let's say, I wanted the laser to project a square, I wouldn't simply send four (one for each corner) points to the DAC, I would send up to 40,000 (at least for the laser I'm using I intend to make the software flexible) points to make it as square as possible as the laser doesn't travel point to point.

It is beneficial to have lower points per frame as the laser can do 40,000 points per second (most lasers can do 30,000) so if you could have 1,000 points per frame, you'd be able to have 40 frames per second.

3.Technical Solution

Some images of the solution throughout development.

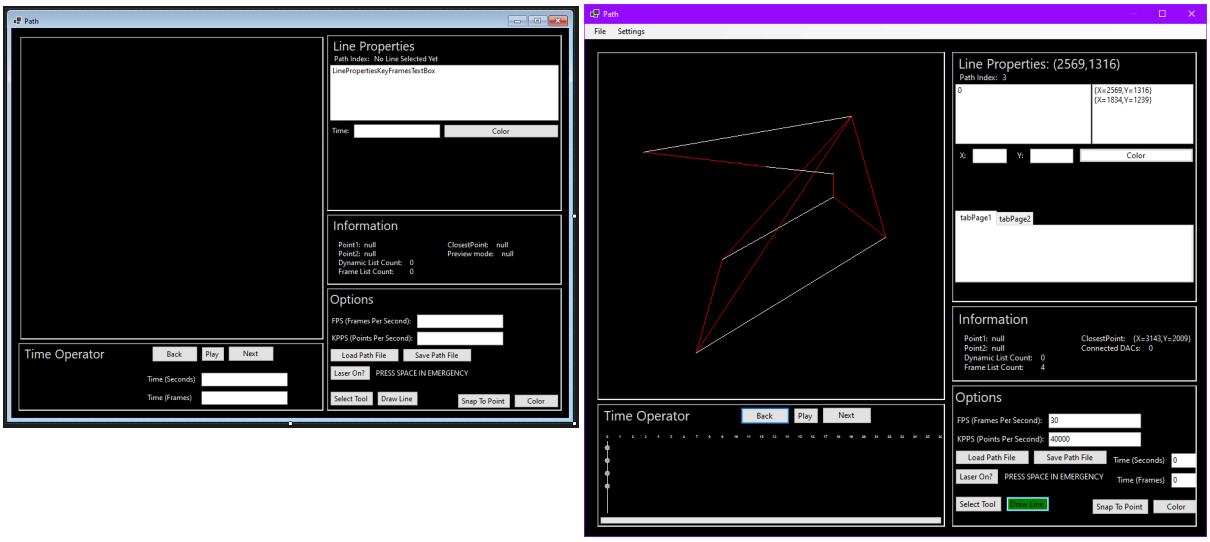


Figure 233: GUI throughout development. Showing without the timeline and whilst testing the path traversal

3.1 Whole solution overview

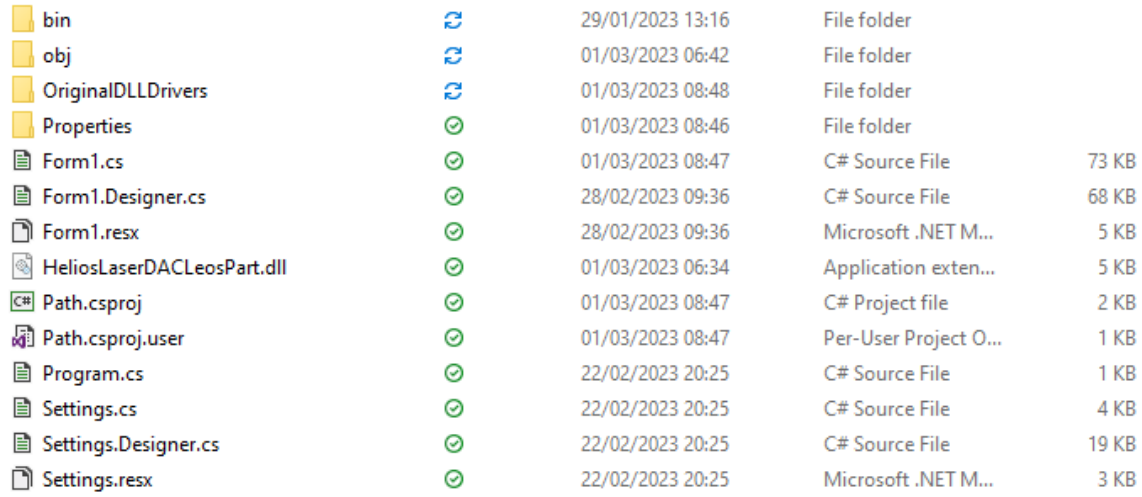


Figure 24: The solution file (explanations below)

Form1.cs contains my code shown in the appendix.

Form1.Designer.cs has all the autogenerated code from when you create the form in the visual studio forms designer. (For the sake of showing everything I will paste this in as well

Form1.resx is an automatically generated code used to identify resources in code.

The HeliosLaserDACLeosPart.dll is something I made so that I can treat the driver as c#. The original driver was in C# (and needs to be placed in the debug file) but this layer makes programming the laser relatively seamless. The code for the dll will be placed in the appendix.

Path.csproj & path.csproj.user is autogenerated

Program.cs contains little code as I placed all my code & classes with the form1.cs methods however it still runs the project.

Settings.cs, settings.designer.cs and settings.resx are part of a second window I began building, it isn't complete but it is designed to make the code more customizable. I didn't want to spend too much time on this as I could use the program fine without it however it is close to working. I am struggling with sending information between windows (likely due to all my code being in form1.cs)

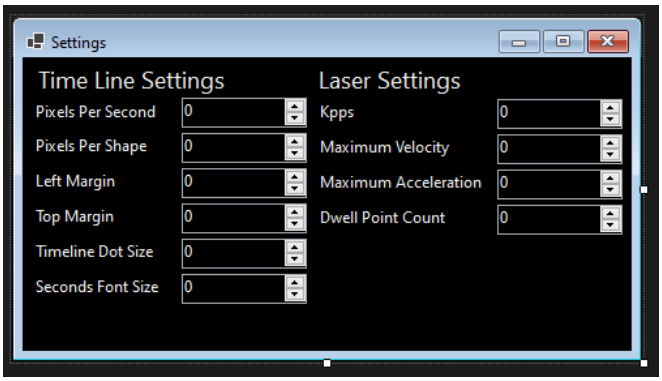


Figure 25: This is my settings window, this isn't working fully yet

For the sake of transparency it looks like this. It is in dark mode again because the system runs in the dark.

3.1.2 Subroutine annotation of Form1.cs

PathMainWindow() – Initialises the form & the code

New LaserSettings() – Initialises the laser settings object, with settings added in.

ConvertToHeliosCoords() – Every stored point should be in "helios" format, essentially a 4095 by 4095 plane. So this converts points from the graphics window into the consistent Helios format. It also handles the range checks.

getDistance() – does basic Pythagoras to work out how far two points are from each other.

//travelBetweenPoints() – Isnt working at the moment and is commented out to avoid confusion

New LinePoint() – base constructor for any linepoint subclasses

New TraversalLinePoint() – Constructor for points used in traversal, requires items like is start so it knows weather to reverse lines or not.

New KeyLinePoint() – Constructor for selection points.

New TimeLinePoint() – Constructor for timeline points, these points are used in the timeline to store frames etc.

genShapeLaserPath() – Generates a queue of lines for the laser to traverse in order. This has been discussed earlier.

PathLineFrame.GenLaserPoints() – Given a frame can generate a list of points for the laser to traverse across (this is essentially how ILDA works)

PathLineFrame.Reverse() – uses the built in function to reverse pathpoints, not sure its really needed given that its only used once.

PathLineFrame.GenKeyPoints() – returns key points for traversal using the PathPoints list.

PathLineFrame.GetValueXWayBetweenPoints() – used for animations (hence why it's in the path line frame.)

New PathLineFrame() – there are several constructors for this including one for jsons with one parameters and one with 5 parameters.

PathLine.GetFrameAt() – Gives you the reference to a frame at time in the items list.

PathLine.GenFrameAt() – Gives you a copy/mockup of what it would be at that time.

PathLine.FindTouchingFrames() – uses binary sort to efficiently get the frames either side of a given time.

New PathLine() – several constructors

PathLine.SortKeyFramesByTime – Calls the QuickSortByTime method

PathLine.QuickSortByTime – Efficiently sorts the frames into order.

PreviewGraphics_Paint() – Creates the graphics for the UI based off of stored objects and also updates laser projections (as laser and graphics should be in sync).

PreviewGraphics_MouseDown() – Begins drawings or selections based on what is selected

GetSnapOrNoSnap() – returns a point, either snapped or not based on how close it is to the closest keypoints.

PreviewGraphics_MouseMove() – Updates the line we are selecting or drawing.

PreviewGraphics_MouseUp() – If line being drawn is the same point both ends, delete it and other tidying up from generating or selecting.

OptionsColorSelecterOpener_Click() – I spelt selector wrong, this opens the DrawerColorDialog which stores the color new lines are drawn with. Also if I spell color wrong its because its spelt American in visual studio which is very confusing.

PreviewGraphics_Resize() – Used for making the graphics panel resize in a square fashion.

OptionsDrawLineMode_CheckedChanged() – change the color and makes sere the select mode isn't on when drawing is and vice versa.

OptionsSelectModeButtonCheckedChanged() - change the color and makes sere the dra mode isn't on when selecting is on and vice versa.

OptionsSnapToPoint_CheckedChanged() – Green when on red when off

TimeLineSecondsInput_TextChanged() – Set the time to the input in seconds

TimeLineFramesInput_TextChanged() – Set the time to the input in frames

UpdateLineProperties() – Make the details of the information the same as the details of the lines

GetSelectedFrameWrite() – return the frame you are modifying or creating

ChangeTime() – Change the time to the specified time and update everything

LinePropertiesKeyFramesTextBox_SelectedIndexChanged() – Update the selected time and frame.

PathLinePointsListBox_SelectedIndexChanged() – Update the selected shape and call updatelineproperties().

New TimelineSettings() – Constructors

timelineSettings.getFloatXOfFrameTime() – converts the seconds to frames.

timeline_GUI_Updater() – draws out the labels and points of timeline

timelineGUI_MouseMove() – Changes the time to wherever the mouse clicked in the timeline and updates everything to work with it.

saveToolStripMenuItem_Click() – saves the file

GetHash<>() - converts any item into a hash

openToolStripMenuItem_click() – opens a file

connectToDACToolStripMenuItem_Click() – Connects to the DAC

DisconnectDACToolStripMenuItem_Click() – Disconnects the DAC

backgroundWorker1_DoWork() – runs the Laser

linePropertiesXOrYCoordinate_ValueChanged() – updates the line via the properties panel

deleteShape_click() – deletes the selected shape

Form1_KeyPress() – checks to see if the laser needs shutting off etc

OptionsToggleProject_CheckedChanged() – Turn Laser On or off

LinePropertiesChangeColor_Click() - Open dialog and let user change color

ToolStripMenuItem_Click() – open settings

BackgroundWorker2_DoWork() – MovesTimeline

projectMaxTimeSelector_ValueChanged() – Change maximum time

TimeLinePlay_CheckedChanged() – Play or pause the projection

JsonSerialization.WriteToJsonFile() – write to file

JsonSerialization.ReadFromJsonFile()- read to file

GraphicsExtensions.DrawCircle() – draws circles

GraphicsExtensions.FillCircle() – draws filled circles

3.2 Skilled programming methods (use of GROUP A & B techniques):

GROUP A Techniques

Whilst making the technical solution some of the parts of the code (See the Appendix) that demonstrate the most skill is:

3.2.1 Graph Traversal (group A technique):

I use this to traverse all the required lines (essentially, I work out the closest point and traverse to the point and then draw the line and repeat from our new point).

The code can be found around lines: 211 to 283

The code itself is:

```
Queue<PathLineFrame> genShapeLaserPath(List<PathLineFrame> framePath)
{
        List<TraversalLinePoint> pointsToTraverse = new List<TraversalLinePoint>();

        // Gets all the points required in order to traverse.
    foreach (PathLineFrame line in framePath)
    {
        pointsToTraverse.Add(new TraversalLinePoint(framePath.IndexOf(line), line.PathPoints[0], true));
        pointsToTraverse.Add(new TraversalLinePoint(framePath.IndexOf(line), line.PathPoints[^1], false));
    }

        // Setting up variables.
    Queue<PathLineFrame> linesToGenPointsFor = new Queue<PathLineFrame>();
    TraversalLinePoint currentPoint;  // Where we are.
    TraversalLinePoint nextPoint;  // Where we want to go.

        // If we are in a position where there is a need to project.
    if (pointsToTraverse.Count > 1 && (pointsToTraverse.Count % 2 == 0))
        {
        currentPoint = pointsToTraverse[0];

                // Travels to the next line.
                        linesToGenPointsFor.Enqueue(framePath[currentPoint.ShapeListIndex]);
            int nextIndex = pointsToTraverse.IndexOf(currentPoint);  // Not add one because current point is
removed.
        pointsToTraverse.Remove(currentPoint);
        currentPoint = pointsToTraverse[nextIndex];

                // This if statement basically travels the line either forwards or backwards.
        pointsToTraverse.Remove(currentPoint);  // Removed because it's already been traversed.

        // Start the while loop after drawing the first line.
        while (pointsToTraverse.Count > 0)
        {
```

```
            nextPoint = new TraversalLinePoint(-1, new Point(0x7FFFFFFF, 0x7FFFFFFF), false);  // Reset next point to be infinitely far away. Here we are setting the ideal closest point.

                foreach (TraversalLinePoint point in pointsToTraverse)
                {
                    // This point closer to the goal than the other point.
                    if (getDistance(point.Location, currentPoint.Location) < getDistance(nextPoint.Location, currentPoint.Location))
                    {
                        nextPoint = point;  // Closest point = the closer point.
                    }
                }

                // Find where to travel to (yes, it's the most basic path traversal heuristic, but it works quite well when things are simple).
                // I would do a more complicated one, but the points need to be met in certain ways.
                linesToGenPointsFor.Enqueue(new PathLineFrame(true, currentPoint.Location, nextPoint.Location));
                nextIndex = pointsToTraverse.IndexOf(nextPoint);  // Needs it to remove the other part of the line. Could have done a for loop for each line to be honest, might be a piece of improvement for the future.
                pointsToTraverse.Remove(nextPoint);  // And remove the point you've just run away from. You need to remove two points per line/while loop. It's next point that gets removed as this has just been traversed.
                currentPoint = nextPoint;

                // Travel between two lines (the laser has to do this it cannot simply teleport).
                if (pointsToTraverse.Count > 0)
                {
                    if (currentPoint.IsStart)  // If yes, travel the line then remove the points.
                    {
                        linesToGenPointsFor.Enqueue(framePath[currentPoint.ShapeListIndex]);  // Index already set above.
                    }
                    else
                    {
                        PathLineFrame frame = new PathLineFrame(framePath[currentPoint.ShapeListIndex]);
                        frame.Reverse();
                        linesToGenPointsFor.Enqueue(frame);
                        nextIndex = nextIndex - 1;  //Subtract one because the index below isnt affected. Like because the line is reversed yk
                    }
                    currentPoint = pointsToTraverse[nextIndex];
                    pointsToTraverse.Remove(pointsToTraverse[nextIndex]);
                }
            }
            linesToGenPointsFor.Enqueue(new PathLineFrame(true, currentPoint.Location, framePath[0].PathPoints[0]));
        }
            return linesToGenPointsFor;
    }
```

I'm sorry if the tabbage gave you a headache.

3.2.2 List, Stack and Queue Operations (group A technique):

In my code I utilize lists for all of my points and shapes as this can be indefinitely scalable.

I declare the list of pathpoints on line 289 and refer to it in several places such as my method to convert these frames to laser points which work with lists lots of times, I also use queues and stacks in this to take advantage of their lifo and fifo feature set on lines 295-405:

```
public List<HeliosPoint> GenLaserPoints(LaserSettings currentLaserSettings)
        {
            List<HeliosPoint> points = new List<HeliosPoint>();
            //If we have constant acceleration life is hella easy thanks to SUVAT equations
            //S = ut + 1/2 a t^2 (hold on one sec we got to the good bit in my playlist I needa jam for a sec)
            if (PathPoints.Count > 0)
            {
                Queue<double> beginningDisplacements = new Queue<double>();
                Stack<double> endingDisplacements = new Stack<double>();
                for (int i = 0; i < currentLaserSettings.DwellPoints; i++)
                {
                    beginningDisplacements.Enqueue(0);
                }
                int acceleratingTime = currentLaserSettings.MaxVelocity /
currentLaserSettings.MaxAcceleration;
                double currentDisplacement = 0;
                double halfDisplacement = getDistance(this.PathPoints[0], this.PathPoints[1]) / 2;//I realise that
I havent added multi point compatibility this is a relatively easy fix for before summer :)
                for (double t = 0; t < acceleratingTime && (0.5 * t * t * currentLaserSettings.MaxAcceleration) <
halfDisplacement; t++)
                {
                    beginningDisplacements.Enqueue(0.5 * t * t * currentLaserSettings.MaxAcceleration);
                    endingDisplacements.Push(0.5 * t * t * currentLaserSettings.MaxAcceleration);
                    currentDisplacement = 0.5 * t * t * currentLaserSettings.MaxAcceleration;
                }
                if (currentDisplacement < halfDisplacement + (currentLaserSettings.MaxVelocity / 2))//If
distance left to cover will take more than a point.
                {
                    for (int i = 0; currentDisplacement + currentLaserSettings.MaxVelocity < halfDisplacement;
i++)
                    {
                        currentDisplacement += currentLaserSettings.MaxVelocity;
                        beginningDisplacements.Enqueue(currentDisplacement +
currentLaserSettings.MaxVelocity);
                        endingDisplacements.Push(currentDisplacement + currentLaserSettings.MaxVelocity);
//Should make it so the middle points are evenly spaced.0

                    }

                    // This bit fills in the middle so the velocity isnt bigger than the max velocity. The rest uses
the max acceleration.
                }
                HeliosPoint currentPoint = new HeliosPoint();
                currentDisplacement = 0;
                double horizontaleScaleValue = (PathPoints[1].X - PathPoints[0].X) /
getDistance(this.PathPoints[0], this.PathPoints[1]);
```

```
            double verticalScaleValue = (PathPoints[1].Y - PathPoints[0].Y) / getDistance(this.PathPoints[0],
this.PathPoints[1]);
            while (beginningDisplacements.Count > 0)
            {
                currentDisplacement = beginningDisplacements.Dequeue();
                currentPoint.x = (ushort)(PathPoints[0].X + currentDisplacement * horizontaleScaleValue);
                currentPoint.y = (ushort)(PathPoints[0].Y + currentDisplacement * verticalScaleValue);
                currentPoint.r = (byte)((this.PathColor.R * this.PathColor.A) / 0xFF);
                currentPoint.g = (byte)((this.PathColor.G * this.PathColor.A) / 0xFF);
                currentPoint.b = (byte)((this.PathColor.B * this.PathColor.A) / 0xFF);
                currentPoint.i = (byte)(this.PathColor.A);
                points.Add(currentPoint);
            }
            while (endingDisplacements.Count > 0)
            {
                currentDisplacement = endingDisplacements.Pop();
                currentPoint.x = (ushort)(PathPoints[1].X - currentDisplacement * horizontaleScaleValue);
                currentPoint.y = (ushort)(PathPoints[1].Y - currentDisplacement * verticalScaleValue);
                currentPoint.r = (byte)((this.PathColor.R * this.PathColor.A) / 0xFF);
                currentPoint.g = (byte)((this.PathColor.G * this.PathColor.A) / 0xFF);
                currentPoint.b = (byte)((this.PathColor.B * this.PathColor.A) / 0xFF);
                currentPoint.i = (byte)(this.PathColor.A);
                points.Add(currentPoint);
            }
        }
        else
        {
            points = new List<HeliosPoint>();
        }
        return points;
    }
```

3.2.3 Hashing (group A technique):

In this I have a hashing algorithm that converts an object (of any type which I am proud) into a set of bytes which then gets hashed using standard md5 so other software can work with it and converted back into a string.

The algorithm is lines 1167 to 1179 and is shown below:

```
    public string GetHash<T>(T currentObject)   //The capital T is the Type and is worked out per call of
the method.
    {
        // This was quite confusing to me as I had to find a way to convert the object into a single string
        // Calculate the MD5 hash of the project object
        // Convert the object to a byte array
        byte[] data = Encoding.UTF8.GetBytes(Newtonsoft.Json.JsonConvert.SerializeObject(currentObject));
// Convert object to a byte array

        // Generate the MD5 hash
        MD5 md5 = MD5.Create();
        byte[] hash = md5.ComputeHash(data);
        string hashString = BitConverter.ToString(hash);
        return Convert.ToBase64String(hash);
```

```
    }
```

It is used to verify the file wasn't corrupted between saving and loading in the methods 1158-1165:

```
    private void saveToolStripMenuItem_Click(object sender, EventArgs e)
    {
        saveFileDialog1.ShowDialog();
        project.fileHash = "";
        // Calculate the MD5 hash of the project object
        project.fileHash = GetHash(project);
        JsonSerialization.WriteToJsonFile<PathProject>(saveFileDialog1.FileName, project);
    }
```

and 1180-1203:

```
    private void openToolStripMenuItem_Click(object sender, EventArgs e)
    {
        currentLaserSettings.ShowLaser = false;
        try
        {
            openFileDialog1.ShowDialog();
            project = JsonSerialization.ReadFromJsonFile<PathProject>(openFileDialog1.FileName);
            String projectsFileHash = project.fileHash;
            project.fileHash = "";
            if (GetHash(project) != projectsFileHash)
            {
                MessageBox.Show("File modified outside of software or corrupted." +
                    " Be careful when using the file as it may be dangerous or damage device. ", "Warning");
            }
            selectedLineDynamicIndex = 0;
            selectedPointDynamicIndex = 0;
            ChangeTime(0);
        }
        catch
        {
            MessageBox.Show("File incorrect or corrupt, try again or check the JSON to see if its formatted
correctly.");
        }

    }
```

3.2.4 Recursive algorithms (group A technique):

3.2.4.1 Find touching frames 530-561:

```
        public List<PathLineFrame> findTouchingFrames(int timeOfFrame, List<PathLineFrame>
touchingPoints, bool startingOccurance = true)
        {
            if (startingOccurance)
            {
                touchingPoints = new List<PathLineFrame> { new PathLineFrame(-1, Color.Black, new
List<Point>(), -1) };
                //Added fake beginning abnd ending frames so theres always a frame above and below or the
code would j break
```

```
                touchingPoints.AddRange(KeyFrames);
                touchingPoints.Add(new PathLineFrame((int)0x7FFFFFFF, Color.Black, new List<Point>(), -1));
            }
            int midFrameIndex = touchingPoints.Count() / 2;
            if (touchingPoints[midFrameIndex].Time == timeOfFrame)  //If the mid frame matches the time
            {
                return new List<PathLineFrame> { touchingPoints[midFrameIndex] };
            }
            else if (timeOfFrame > touchingPoints[midFrameIndex].Time)
            {
                if (timeOfFrame < touchingPoints[midFrameIndex + 1].Time)
                {
                    return new List<PathLineFrame> { touchingPoints[midFrameIndex],
touchingPoints[midFrameIndex + 1] };
                }
                return findTouchingFrames(timeOfFrame, touchingPoints.GetRange(midFrameIndex - 1, 1 +
touchingPoints.Count() - midFrameIndex), false);
            }
            else if (timeOfFrame < touchingPoints[midFrameIndex].Time)
            {
                return findTouchingFrames(timeOfFrame, touchingPoints.GetRange(0, midFrameIndex + 1),
false);
            }
            else
            {
                return new List<PathLineFrame>();
            }

        }
```

3.2.4.2 QuickSortByTime 607-641

```
        public List<PathLineFrame> QuicksortByTime(List<PathLineFrame> list)
        {
            //Here I implement quicksort partially explained to me here:
https://www.youtube.com/watch?v=SLauY6PpjW4&t=15s
            //I used a recursive routine
            if (list.Count() == 0)
            {
                return new List<PathLineFrame>();      //Gotta have something end the recursive routine
            }

            List<PathLineFrame> left = new List<PathLineFrame>();
            List<PathLineFrame> right = new List<PathLineFrame>();
            PathLineFrame pivot = list[list.Count / 2];
            //Basically the way pivot works is you keep making lists of numbers greater than the pivot and
less than the pivot until the whole thing is sorted
            //This makes it O(nlogn) because you keep splitting the array in half each time
            //You learn something new everyday!
            for (int i = 0; i < list.Count(); i++)
            {
                if (list[i].Time < pivot.Time)
                {
                    left.Add(list[i]);
```

```
            }
            else if (list[i].Time > pivot.Time)
            {
                right.Add(list[i]);
            }
        }
        List<PathLineFrame> leftSorted = QuicksortByTime(left);
        List<PathLineFrame> rightSorted = QuicksortByTime(right);
        list.Clear();
        list.AddRange(leftSorted);
        list.Add(pivot);
        list.AddRange(rightSorted);
        return list;
    }
}
```

3.2.4.3 GetFrameAt 519-529 (only recursive up too once but still...)

```
    public PathLineFrame GetFrameAt(int FrameTime)  //Literally a recursive algorithm. Didnt even make it on purpose.
    {
        List<PathLineFrame> touchingFrames = findTouchingFrames(FrameTime, KeyFrames);
        if (touchingFrames.Count == 1)
        {
            return touchingFrames[0];
        }
        KeyFrames.Add(new PathLineFrame(GenFrameAt(FrameTime)));
        this.SortKeyFramesByTime();
        return GetFrameAt(FrameTime);
    }
```

3.2.5 Complex user defined algorithms (group A technique):

The algorithm forms points spread out based on the velocity at the time, which changes as that way you can get faster velocities at points with less detail, this allows for much smoother looking animations.

3.2.5.1 GenLaserPoints

This method is from lines 295-405:

```
public List<HeliosPoint> GenLaserPoints(LaserSettings currentLaserSettings)
    {
        List<HeliosPoint> points = new List<HeliosPoint>();
        //If we have constant acceleration life is hella easy thanks to SUVAT equations
        //S = ut + 1/2 a t^2 (hold on one sec we got to the good bit in my playlist I needa jam for a sec)
        if (PathPoints.Count > 0)
        {
            Queue<double> beginningDisplacements = new Queue<double>();
            Stack<double> endingDisplacements = new Stack<double>();
            for (int i = 0; i < currentLaserSettings.DwellPoints; i++)
            {
                beginningDisplacements.Enqueue(0);
            }
            int acceleratingTime = currentLaserSettings.MaxVelocity / currentLaserSettings.MaxAcceleration;
            double currentDisplacement = 0;
```

```
            double halfDisplacement = getDistance(this.PathPoints[0], this.PathPoints[1]) / 2;//I realise that
I havent added multi point compatibility this is a relatively easy fix for before summer :)
            for (double t = 0; t < acceleratingTime && (0.5 * t * t * currentLaserSettings.MaxAcceleration) <
halfDisplacement; t++)
            {
                beginningDisplacements.Enqueue(0.5 * t * t * currentLaserSettings.MaxAcceleration);
                endingDisplacements.Push(0.5 * t * t * currentLaserSettings.MaxAcceleration);
                currentDisplacement = 0.5 * t * t * currentLaserSettings.MaxAcceleration;
            }
            if (currentDisplacement < halfDisplacement + (currentLaserSettings.MaxVelocity / 2))//If
distance left to cover will take more than a point.
            {
                for (int i = 0; currentDisplacement + currentLaserSettings.MaxVelocity < halfDisplacement;
i++)
                {
                    currentDisplacement += currentLaserSettings.MaxVelocity;
                    beginningDisplacements.Enqueue(currentDisplacement +
currentLaserSettings.MaxVelocity);
                    endingDisplacements.Push(currentDisplacement + currentLaserSettings.MaxVelocity);
//Should make it so the middle points are evenly spaced.0

                }

                // This bit fills in the middle so the velocity isnt bigger than the max velocity. The rest uses
the max acceleration.
            }
            HeliosPoint currentPoint = new HeliosPoint();
            currentDisplacement = 0;
            double horizontaleScaleValue = (PathPoints[1].X - PathPoints[0].X) /
getDistance(this.PathPoints[0], this.PathPoints[1]);
            double verticalScaleValue = (PathPoints[1].Y - PathPoints[0].Y) / getDistance(this.PathPoints[0],
this.PathPoints[1]);
            while (beginningDisplacements.Count > 0)
            {
                currentDisplacement = beginningDisplacements.Dequeue();
                currentPoint.x = (ushort)(PathPoints[0].X + currentDisplacement * horizontaleScaleValue);
                currentPoint.y = (ushort)(PathPoints[0].Y + currentDisplacement * verticalScaleValue);
                currentPoint.r = (byte)((this.PathColor.R * this.PathColor.A) / 0xFF);
                currentPoint.g = (byte)((this.PathColor.G * this.PathColor.A) / 0xFF);
                currentPoint.b = (byte)((this.PathColor.B * this.PathColor.A) / 0xFF);
                currentPoint.i = (byte)(this.PathColor.A);
                points.Add(currentPoint);
            }
            while (endingDisplacements.Count > 0)
            {
                currentDisplacement = endingDisplacements.Pop();
                currentPoint.x = (ushort)(PathPoints[1].X - currentDisplacement * horizontaleScaleValue);
                currentPoint.y = (ushort)(PathPoints[1].Y - currentDisplacement * verticalScaleValue);
                currentPoint.r = (byte)((this.PathColor.R * this.PathColor.A) / 0xFF);
                currentPoint.g = (byte)((this.PathColor.G * this.PathColor.A) / 0xFF);
                currentPoint.b = (byte)((this.PathColor.B * this.PathColor.A) / 0xFF);
                currentPoint.i = (byte)(this.PathColor.A);
                points.Add(currentPoint);
```

```
            }
        }
        else
        {
            points = new List<HeliosPoint>();
        }
        return points;
    }
```

3.2.5.2 GenShapeLaserPath

The other complex optimisation algorithm is my path generation algorithm, instead of travelling through the frames as they were listed, I use code to travel to the closest point, this way it reduces the amount of unnecessary distance travelled, this also has graph traversal involved so has been shown earlier.

The code can be found around lines: 211 to 283

The code itself is:

```
Queue<PathLineFrame> genShapeLaserPath(List<PathLineFrame> framePath)
{
        List<TraversalLinePoint> pointsToTraverse = new List<TraversalLinePoint>();

        // Gets all the points required in order to traverse.
    foreach (PathLineFrame line in framePath)
    {
        pointsToTraverse.Add(new TraversalLinePoint(framePath.IndexOf(line), line.PathPoints[0], true));
        pointsToTraverse.Add(new TraversalLinePoint(framePath.IndexOf(line), line.PathPoints[^1], false));
    }

        // Setting up variables.
    Queue<PathLineFrame> linesToGenPointsFor = new Queue<PathLineFrame>();
    TraversalLinePoint currentPoint;  // Where we are.
    TraversalLinePoint nextPoint;  // Where we want to go.

        // If we are in a position where there is a need to project.
    if (pointsToTraverse.Count > 1 && (pointsToTraverse.Count % 2 == 0))
        {
        currentPoint = pointsToTraverse[0];

                // Travels to the next line.
                        linesToGenPointsFor.Enqueue(framePath[currentPoint.ShapeListIndex]);
        int nextIndex = pointsToTraverse.IndexOf(currentPoint);  // Not add one because current point is
removed.
        pointsToTraverse.Remove(currentPoint);
        currentPoint = pointsToTraverse[nextIndex];

                // This if statement basically travels the line either forwards or backwards.
        pointsToTraverse.Remove(currentPoint);  // Removed because it's already been traversed.

        // Start the while loop after drawing the first line.
        while (pointsToTraverse.Count > 0)
        {
```

```
            nextPoint = new TraversalLinePoint(-1, new Point(0x7FFFFFFF, 0x7FFFFFFF), false);  // Reset next point to be infinitely far away. Here we are setting the ideal closest point.

                foreach (TraversalLinePoint point in pointsToTraverse)
                {
                    // This point closer to the goal than the other point.
                    if (getDistance(point.Location, currentPoint.Location) < getDistance(nextPoint.Location, currentPoint.Location))
                    {
                        nextPoint = point;  // Closest point = the closer point.
                    }
                }

                // Find where to travel to (yes, it's the most basic path traversal heuristic, but it works quite well when things are simple).
                // I would do a more complicated one, but the points need to be met in certain ways.
                linesToGenPointsFor.Enqueue(new PathLineFrame(true, currentPoint.Location, nextPoint.Location));
                nextIndex = pointsToTraverse.IndexOf(nextPoint);  // Needs it to remove the other part of the line. Could have done a for loop for each line to be honest, might be a piece of improvement for the future.
                pointsToTraverse.Remove(nextPoint);  // And remove the point you've just run away from. You need to remove two points per line/while loop. It's next point that gets removed as this has just been traversed.
                currentPoint = nextPoint;

                // Travel between two lines (the laser has to do this it cannot simply teleport).
                if (pointsToTraverse.Count > 0)
                {
                    if (currentPoint.IsStart)  // If yes, travel the line then remove the points.
                    {
                        linesToGenPointsFor.Enqueue(framePath[currentPoint.ShapeListIndex]);  // Index already set above.
                    }
                    else
                    {
                        PathLineFrame frame = new PathLineFrame(framePath[currentPoint.ShapeListIndex]);
                        frame.Reverse();
                        linesToGenPointsFor.Enqueue(frame);
                        nextIndex = nextIndex - 1;  //Subtract one because the index below isnt affected. Like because the line is reversed yk
                    }
                    currentPoint = pointsToTraverse[nextIndex];
                    pointsToTraverse.Remove(pointsToTraverse[nextIndex]);
                }
            }
            linesToGenPointsFor.Enqueue(new PathLineFrame(true, currentPoint.Location, framePath[0].PathPoints[0]));
        }
            return linesToGenPointsFor;
    }
```

3.2.6 Quicksort (group A technique):

From lines 607 to 641, sorts each item specifically by its time:

```
        public List<PathLineFrame> QuicksortByTime(List<PathLineFrame> list)
        {
            //Here I impliment quicksort partially explained to me here:
https://www.youtube.com/watch?v=SLauY6PpjW4&t=15s
            //I used a recursive routine
            if (list.Count() == 0)
            {
                return new List<PathLineFrame>();       //Gotta have something end the recursive routine
            }

            List<PathLineFrame> left = new List<PathLineFrame>();
            List<PathLineFrame> right = new List<PathLineFrame>();
            PathLineFrame pivot = list[list.Count / 2];
            //Basically the way pivot works is you keep making lists of numbers greater than the pivot and
less than the pivot until the whole thing is sorted
            //This makes it O(nlogn) because you keep splitting the array in half each time
            //You learn something new everyday!
            for (int i = 0; i < list.Count(); i++)
            {
                if (list[i].Time < pivot.Time)
                {
                    left.Add(list[i]);
                }
                else if (list[i].Time > pivot.Time)
                {
                    right.Add(list[i]);
                }
            }
            List<PathLineFrame> leftSorted = QuicksortByTime(left);
            List<PathLineFrame> rightSorted = QuicksortByTime(right);
            list.Clear();
            list.AddRange(leftSorted);
            list.Add(pivot);
            list.AddRange(rightSorted);
            return list;
        }
    }
```

3.2.7 Dynamic generation of objects based on complex user defined use of OOP model (group A technique):

The best example of this is my GenFrameAt method that creates frames between two points, this is constructing really niche classes that would be really difficult to create with typical variables.

Lines 562-590

```
        public PathLineFrame GenFrameAt(int FrameTime)
        {
            PathLineFrame newFrame;
```

```
            List<PathLineFrame> touchingFrames = findTouchingFrames(FrameTime, new
List<PathLineFrame>());  //This algorithm should be much more efficient :0

            if (touchingFrames.Count == 1)
            {
                return touchingFrames[0];
            }
            else if (touchingFrames[0].Time == -1)               //If before all frames then newframe is the same
as the frame after or first frame
            {
                newFrame = new PathLineFrame(touchingFrames[1]);    //Sets it to the not fake frame
                newFrame.Time = FrameTime;
                return newFrame;
            }
            else if (touchingFrames[1].Time == (int)0x7FFFFFFF)                //Literally the same situation as
the above if
            {
                newFrame = new PathLineFrame(touchingFrames[0]);       //Sets it to the not fake frame
                newFrame.Time = FrameTime;
                return newFrame;
            }
            else
            {
                //SET ALL PROPERTIES TO PROPERTIES IN BETWEEN THEM BOTH
                float animationProgress = ((float)(FrameTime - touchingFrames[0].Time) /
(float)(touchingFrames[1].Time - touchingFrames[0].Time)); //If you set each property to beforeFrame value
plus (afterFrame - beforeFrame) * difference  -- This assumes a linear animation hence the constant
progress as time moves forward.
                return new PathLineFrame(animationProgress, touchingFrames[0], touchingFrames[1]);
            }
        }
```

3.2.8 Inheritance (group A technique):

As shown in my documented design, I used many complex objects but I used inheritance for LinePoints as shown from 171-210:

```
    abstract class LinePoint
    {
        public int ShapeListIndex { get; set; }
        public Point Location { get; set; }

        public LinePoint(int shapeListIndex, Point location)
        {
            ShapeListIndex = shapeListIndex;
            Location = location;
        }
    }
    class TraversalLinePoint : LinePoint
    {
        public bool IsStart { get; set; }
        public TraversalLinePoint(int shapeListIndex, Point location, bool isStart) : base(shapeListIndex,
location)
        {
```

```
            ShapeListIndex = shapeListIndex;
            Location = location;
            IsStart = isStart;
        }
    }
    class KeyLinePoint : LinePoint
    {
        public bool IsMiddle { get; set; }
        public int PathPointsListIndex { get; set; }
        public KeyLinePoint(int shapeListIndex, int pathPointsListIndex, Point location, bool isMiddle) :
base(shapeListIndex, location)
        {
            IsMiddle = isMiddle;
            PathPointsListIndex = pathPointsListIndex;
        }
    }
    class TimeLinePoint : LinePoint
    {
        public int FrameTime { get; set; }
        public TimeLinePoint(int shapeListIndex, int frameTime, Point location) : base(shapeListIndex,
location)
        {
            FrameTime = frameTime;
        }
    }
```

Group B

3.2.9 Binary search (group B technique):

530-561:

```
        public List<PathLineFrame> findTouchingFrames(int timeOfFrame, List<PathLineFrame>
touchingPoints, bool startingOccurance = true)
        {
            if (startingOccurance)
            {
                touchingPoints = new List<PathLineFrame> { new PathLineFrame(-1, Color.Black, new
List<Point>(), -1) };
                //Added fake beginning abnd ending frames so theres always a frame above and below or the
code would j break
                touchingPoints.AddRange(KeyFrames);
                touchingPoints.Add(new PathLineFrame((int)0x7FFFFFFF, Color.Black, new List<Point>(), -1));
            }
            int midFrameIndex = touchingPoints.Count() / 2;
            if (touchingPoints[midFrameIndex].Time == timeOfFrame)  //If the mid frame matches the time
            {
                return new List<PathLineFrame> { touchingPoints[midFrameIndex] };
            }
            else if (timeOfFrame > touchingPoints[midFrameIndex].Time)
            {
                if (timeOfFrame < touchingPoints[midFrameIndex + 1].Time)
                {
                    return new List<PathLineFrame> { touchingPoints[midFrameIndex],
touchingPoints[midFrameIndex + 1] };
                }
```

```
                return findTouchingFrames(timeOfFrame, touchingPoints.GetRange(midFrameIndex - 1, 1 +
touchingPoints.Count() - midFrameIndex), false);
            }
            else if (timeOfFrame < touchingPoints[midFrameIndex].Time)
            {
                return findTouchingFrames(timeOfFrame, touchingPoints.GetRange(0, midFrameIndex + 1),
false);
            }
            else
            {
                return new List<PathLineFrame>();
            }

        }
```

3.2.10 Reading and writing from files

I used someone elses method for part of this feature. I borrowed it because I ended up learning about it by seeing the code, I could probably recreate it but because they taught it line for line I didn't want to remake it and take credit. The first part is my code then I declare when the borrowed code begins.

My code is here 1158-1203:

```
    private void saveToolStripMenuItem_Click(object sender, EventArgs e)
    {
        saveFileDialog1.ShowDialog();
        project.fileHash = "";
        // Calculate the MD5 hash of the project object
        project.fileHash = GetHash(project);
        JsonSerialization.WriteToJsonFile<PathProject>(saveFileDialog1.FileName, project);
    }

    public string GetHash<T>(T currentObject)   //The capital T is the Type and is worked out per call of
the method.
    {
        // This was quite confusing to me as I had to find a way to convert the object into a single string
        // Calculate the MD5 hash of the project object
        // Convert the object to a byte array
        byte[] data = Encoding.UTF8.GetBytes(Newtonsoft.Json.JsonConvert.SerializeObject(currentObject));
// Convert object to a byte array

        // Generate the MD5 hash
        MD5 md5 = MD5.Create();
        byte[] hash = md5.ComputeHash(data);
        string hashString = BitConverter.ToString(hash);
        return Convert.ToBase64String(hash);
    }
    private void openToolStripMenuItem_Click(object sender, EventArgs e)
    {
        currentLaserSettings.ShowLaser = false;
        try
        {
            openFileDialog1.ShowDialog();
```

```
            project = JsonSerialization.ReadFromJsonFile<PathProject>(openFileDialog1.FileName);
            String projectsFileHash = project.fileHash;
            project.fileHash = "";
            if (GetHash(project) != projectsFileHash)
            {
                MessageBox.Show("File modified outside of software or corrupted." +
                    " Be careful when using the file as it may be dangerous or damage device. ", "Warning");
            }
            selectedLineDynamicIndex = 0;
            selectedPointDynamicIndex = 0;
            ChangeTime(0);
        }
        catch
        {
            MessageBox.Show("File incorrect or corrupt, try again or check the JSON to see if its formatted
correctly.");
        }

    }
```

END OF MY CODE IS HERE

Borrowed code from stack overflow 1397+:

```
  public static class JsonSerialization
  {
      /// <summary>
      /// Writes the given object instance to a Json file.
      /// <para>Object type must have a parameterless constructor.</para>
      /// <para>Only Public properties and variables will be written to the file. These can be any type
though, even other classes.</para>
      /// <para>If there are public properties/variables that you do not want written to the file, decorate
them with the [JsonIgnore] attribute.</para>
      /// </summary>
      /// <typeparam name="T">The type of object being written to the file.</typeparam>
      /// <param name="filePath">The file path to write the object instance to.</param>
      /// <param name="objectToWrite">The object instance to write to the file.</param>
      /// <param name="append">If false the file will be overwritten if it already exists. If true the contents
will be appended to the file.</param>
      public static void WriteToJsonFile<T>(string filePath, T objectToWrite, bool append = false) where T :
new()
      {
          TextWriter writer = null;
          try
          {
              var contentsToWriteToFile = Newtonsoft.Json.JsonConvert.SerializeObject(objectToWrite);
              try
              {
                  writer = new StreamWriter(filePath, append);
                  writer.Write(contentsToWriteToFile);
              }
              catch
              {
```

```
            }
        }
        finally
        {
            if (writer != null)
                writer.Close();
        }
    }

    /// <summary>
    /// Reads an object instance from an Json file.
    /// <para>Object type must have a parameterless constructor.</para>
    /// </summary>
    /// <typeparam name="T">The type of object to read from the file.</typeparam>
    /// <param name="filePath">The file path to read the object instance from.</param>
    /// <returns>Returns a new instance of the object read from the Json file.</returns>
    public static T ReadFromJsonFile<T>(string filePath) where T : new()
    {
        TextReader reader = null;
        try
        {
            reader = new StreamReader(filePath);
            var fileContents = reader.ReadToEnd();
            return Newtonsoft.Json.JsonConvert.DeserializeObject<T>(fileContents);
        }
        finally
        {
            if (reader != null)
                reader.Close();
        }
    }

}
```

I also borrowed code to create points which I will disclose here:

```
    public static class GraphicsExtensions      //This code (the graphics extensions) is borrowed code. IT works really well so I am keeping it
    {
        public static void DrawCircle(this Graphics g, Pen pen,
                                      float centerX, float centerY, float radius)
        {
            g.DrawEllipse(pen, centerX - radius, centerY - radius,
                          radius + radius, radius + radius);
        }

        public static void FillCircle(this Graphics g, Brush brush,
                                      float centerX, float centerY, float radius)
        {
            g.FillEllipse(brush, centerX - radius, centerY - radius,
                          radius + radius, radius + radius);
        }//End of borrowed code.
```

Whilst I could have made the code myself it was someone elses Idea and I want to give them credit and be safe.

I modified their code to make:

```
public static void FillCircle(this Graphics g, Brush brush,
                              Point center, float radius)
{
    g.FillEllipse(brush, center.X - radius, center.Y - radius,
                  radius + radius, radius + radius);
}
```

And that's all the code I borrowed in form.cs!

3.3 My Technical code (Form1.cs)

Here is where I did my programming, I will put as much of my solution file as possible in appendix 6 a lot of which is autogenerated but I will make that clear.

```
using System.Collections.Generic;

using System.Linq;

using Newtonsoft;

using HeliosLaserDACLeosBit;

using System.Drawing;

using System.Text;

using System.Text.Json;

using System.Security.Cryptography;

using System.Linq.Expressions;

//project.dynamicPath = the file & whole animation

//framePath = the PathLineFrame array for the individual frame


//REMEMBER THAT ALL SHAPES SHOULD BE STORED IN A 4095 CO-OORDINATE RESOLUTION NOT THE RESOLUTION OF THE SCREEN.


namespace Path

{

   public partial class PathMainWindow : Form

   {

      PathProject project = new PathProject();

      List<PathLineFrame> framePath;  //What individual frames should look like. Should hopefully be generateable from a for loop and get line at project.dynamicPath[i]

      int framePathTime = -1;  //Gives the time the framePath is generated for


      int mainTime = 0;  //The time in frames that the system is at


      Point mouseLastLocation;  //If set to -1,-1 no line preview should be made - Saving where the mouse went down

      Point timelineMouseLastLocation;
```

```
List<KeyLinePoint> closestPoints = new List<KeyLinePoint>();
List<KeyLinePoint> closestPointsFrozen = new List<KeyLinePoint>();

Point closestPoint;

bool currentlyPlaying = false;
bool middleSelected = false;

Point tempIntermediateFramePoint = new Point(-1, -1);

int selectedLineDynamicIndex = 0;
PathLineFrame selectedFrameReadOnly;
int selectedPointDynamicIndex = -1;

Queue<PathLineFrame> laserTraversalPath;  //Used for converting frames into a path we can make projections for.
List<HeliosPoint> laserPoints = new List<HeliosPoint>();

bool showCircles = false;
bool showLines = false;
bool mouseDown = false;

//Timeline GUI variables
public TimelineSettings currentTimelineSettings;
TimeLinePoint timelineClosestPoint;
List<TimeLinePoint> TimelineDots = new List<TimeLinePoint>();

//Helios Laser management variables
public HeliosDac helios = new HeliosDac();
List<HeliosPoint> heliosLaserPoints = new List<HeliosPoint>();
```

```
    public LaserSettings currentLaserSettings = new LaserSettings();



    public PathMainWindow()
    {
       InitializeComponent();
       //Initialising Variables
       project.dynamicPath = new List<PathLine>();
       HeliosDac helios = new HeliosDac();
       framePath = new List<PathLineFrame>();
       mainTime = 0;
       DrawerColorDialog.Color = Color.White;
       OptionsDrawLineMode.Checked = true;
       currentTimelineSettings = new TimelineSettings();
       InformationPreviewModeData.Text = helios.openDevices().ToString();
       backgroundWorker1.RunWorkerAsync();
    }
    class PathProject
    {
       public List<PathLine> dynamicPath { get; set; }  = new List<PathLine>(); //The whole
animation, all of the shapes etc.


       public float maxtimeSeconds { get; set; }  = 25;


       public int fps { get; set; } = 30; //The number of frames per second (to obtain time in seconds
divide time by fps)
       public string fileHash { get; set; }    //Used to see if file has been modified outside of the
software.
    }
    public class LaserSettings
    {
```

```
        public int Kpps { get; set; }  //The amount of points per second the laser can travel across.
        public int MaxVelocity { get; set; }  //The fastest speed the laser can travel.
        public int MaxAcceleration { get; set; }   //The fastest rate of change the laser can travel.
        public int DwellPoints { get; set; } //The amount of time the laser waits at the end of a line. This is a quality improver.
        public bool ShowLaser  { get; set; }  //Important variable for safety measures. When 'true' the laser is on which is a danger to eyes.
        public LaserSettings()  //JSON constructor.
        {
            Kpps = 30000;  //The amount of points per second the laser can travel across.
            MaxVelocity = 40;  //The fastest speed the laser can travel.
            MaxAcceleration = 2;  //The fastest rate of change the laser can travel.
            DwellPoints = 10;  //The amount of time the laser waits at the end of a line. This is a quality improver.
            ShowLaser = false; //Important variable for safety measures. When 'true' the laser is on which is a danger to eyes.
        }
    }
    Point ConvertToHeliosCoords(Point Original, bool backwards = false)
    {
        Point returnConverted;  //This translates all the laser coordinates onto a single map to ensure consistency when resizing images.
        float Scale = 4095f / (float)(PreviewGraphics.Size.Width);  //This assumes preview graphics is square, I intend to make this bit such that the projection range is resizeable.
        if (!backwards)
        {
            returnConverted = new Point((int)(Original.X * Scale), (int)(Original.Y * Scale));
            if (returnConverted.X > 4095)  //Checks to make sure no points are outside the laser area.
            {
                returnConverted.X = 4095;
            }
            else if (returnConverted.X < 0)
```

```
            {
                returnConverted.X = 0;
            }
            if (returnConverted.Y > 4095)
            {
                returnConverted.Y = 4095;
            }
            else if (returnConverted.Y < 0)
            {
                returnConverted.Y = 0;
            }
        }
        else
        {
            returnConverted = new Point((int)(Original.X / Scale), (int)(Original.Y / Scale));
        }
        return returnConverted;
    }
    static double getDistance(Point point1, Point point2)
    {
        return Math.Sqrt(Math.Pow(point1.X - point2.X, 2) + Math.Pow(point1.Y - point2.Y, 2));
    }
    //not working vvv
    /*List<HeliosPoint> travelBetweenPoints(PointF zeroPoint, PointF lastPoint, PointF
currentVelocity, PointF endVelocity, int MaxVelocity, int MaxAcceleration)
    {
        HeliosPoint currentPoint;
        List<HeliosPoint> returnPoints = new List<HeliosPoint>();

        double timeToReachFrameX = (lastPoint.X - zeroPoint.X) / (((3 / 2) * currentVelocity.X) - ((1 /
2) * endVelocity.X));
```

```
        double timeToReachFrameY = (lastPoint.Y - zeroPoint.Y) / (((3 / 2) * currentVelocity.Y) - ((1 /
2) * endVelocity.Y));

        double timeScalingX;

        double timeScalingY;

        double totalTime;


        if ((endVelocity.X - currentVelocity.X) > (endVelocity.X - currentVelocity.X))

        {

            timeScalingX = (endVelocity.X - currentVelocity.X) / (MaxAcceleration *
timeToReachFrameX);

            timeScalingY = timeToReachFrameX / timeToReachFrameY;

            totalTime = timeToReachFrameX * timeScalingX;

        }

        else

        {

            timeScalingY = (endVelocity.Y - currentVelocity.Y) / (MaxAcceleration *
timeToReachFrameY);

            timeScalingX = timeToReachFrameY / timeToReachFrameX;

            totalTime = timeToReachFrameX * timeScalingX;

        }


        for (float time = 0; time < totalTime; time++)

        {

            currentPoint.x = (ushort)(1.5 * currentVelocity.X * time - 0.5 * endVelocity.X * time +
zeroPoint.X);

            currentPoint.y = (ushort)(1.5 * currentVelocity.Y * time - 0.5 * endVelocity.Y * time +
zeroPoint.Y);

            currentPoint.r = (Byte)0;

            currentPoint.g = (Byte)0;

            currentPoint.b = (Byte)0;

            currentPoint.i = (Byte)0;

            returnPoints.Add(currentPoint);
```

```
            }


            return returnPoints;
        }       //Beginning of my optimising pathalgorithm*/
        abstract class LinePoint
        {
            public int ShapeListIndex { get; set; }
            public Point Location { get; set; }


            public LinePoint(int shapeListIndex, Point location)
            {
                ShapeListIndex = shapeListIndex;
                Location = location;
            }
        }
        class TraversalLinePoint : LinePoint
        {
            public bool IsStart { get; set; }
            public TraversalLinePoint(int shapeListIndex, Point location, bool isStart) :
base(shapeListIndex, location)
            {
                ShapeListIndex = shapeListIndex;
                Location = location;
                IsStart = isStart;
            }
        }
        class KeyLinePoint : LinePoint
        {
            public bool IsMiddle { get; set; }
            public int PathPointsListIndex { get; set; }
```

```
        public KeyLinePoint(int shapeListIndex, int pathPointsListIndex, Point location, bool isMiddle) :
base(shapeListIndex, location)
        {
            IsMiddle = isMiddle;
            PathPointsListIndex = pathPointsListIndex;
        }
    }
    class TimeLinePoint : LinePoint
    {
        public int FrameTime { get; set; }
        public TimeLinePoint(int shapeListIndex, int frameTime, Point location) : base(shapeListIndex,
location)
        {
            FrameTime = frameTime;
        }
    }

    Queue<PathLineFrame> GenShapeLaserPath(List<PathLineFrame> framePath)
    {
        List<TraversalLinePoint> pointsToTraverse = new List<TraversalLinePoint>();

        // Gets all the points required in order to traverse.
        foreach (PathLineFrame line in framePath)
        {
            pointsToTraverse.Add(new TraversalLinePoint(framePath.IndexOf(line), line.PathPoints[0],
true));
            pointsToTraverse.Add(new TraversalLinePoint(framePath.IndexOf(line), line.PathPoints[^1],
false));
        }

        // Setting up variables.
        Queue<PathLineFrame> linesToGenPointsFor = new Queue<PathLineFrame>();
```

```
        TraversalLinePoint currentPoint;  // Where we are.
        TraversalLinePoint nextPoint;  // Where we want to go.

        // If we are in a position where there is a need to project.
        if (pointsToTraverse.Count > 1 && (pointsToTraverse.Count % 2 == 0))
        {
            currentPoint = pointsToTraverse[0];

            // Travels to the next line.
            linesToGenPointsFor.Enqueue(framePath[currentPoint.ShapeListIndex]);
            int nextIndex = pointsToTraverse.IndexOf(currentPoint);  // Not add one because current point is removed.
            pointsToTraverse.Remove(currentPoint);
            currentPoint = pointsToTraverse[nextIndex];

            // This if statement basically travels the line either forwards or backwards.
            pointsToTraverse.Remove(currentPoint);  // Removed because it's already been traversed.

            // Start the while loop after drawing the first line.
            while (pointsToTraverse.Count > 0)
            {
                nextPoint = new TraversalLinePoint(-1, new Point(0x7FFFFFFF, 0x7FFFFFFF), false);  // Reset next point to be infinitely far away. Here we are setting the ideal closest point.

                foreach (TraversalLinePoint point in pointsToTraverse)
                {
                    // This point closer to the goal than the other point.
                    if (getDistance(point.Location, currentPoint.Location) < getDistance(nextPoint.Location, currentPoint.Location))
                    {
                        nextPoint = point;  // Closest point = the closer point.
```

```
            }
        }

        // Find where to travel to (yes, it's the most basic path traversal heuristic, but it works quite well when things are simple).
        // I would do a more complicated one, but the points need to be met in certain ways.
        linesToGenPointsFor.Enqueue(new PathLineFrame(true, currentPoint.Location, nextPoint.Location));
        nextIndex = pointsToTraverse.IndexOf(nextPoint);  // Needs it to remove the other part of the line. Could have done a for loop for each line to be honest, might be a piece of improvement for the future.
        pointsToTraverse.Remove(nextPoint);  // And remove the point you've just run away from. You need to remove two points per line/while loop. It's next point that gets removed as this has just been traversed.
        currentPoint = nextPoint;

        // Travel between two lines (the laser has to do this it cannot simply teleport).
        if (pointsToTraverse.Count > 0)
        {
            if (currentPoint.IsStart)  // If yes, travel the line then remove the points.
            {
                linesToGenPointsFor.Enqueue(framePath[currentPoint.ShapeListIndex]);  // Index already set above.
            }
            else
            {
                PathLineFrame frame = new PathLineFrame(framePath[currentPoint.ShapeListIndex]);
                frame.Reverse();
                linesToGenPointsFor.Enqueue(frame);
                nextIndex = nextIndex - 1;  //Subtract one because the index below isnt affected. Like because the line is reversed yk
            }
```

```
                currentPoint = pointsToTraverse[nextIndex];
                pointsToTraverse.Remove(pointsToTraverse[nextIndex]);
            }
        }
        linesToGenPointsFor.Enqueue(new PathLineFrame(true, currentPoint.Location,
framePath[0].PathPoints[0]));
      }
      return linesToGenPointsFor;
    }
    class PathLineFrame //SOMETIMES IS A KEYFRAME SOMETIMES ISNT
    {
      public int Time { get; set; }//Note the time is in frames not seconds, i'd recommend 30 frames
per second unless you have expensive gear
      public bool Hidden { get; set; }
      public Color PathColor { get; set; }
      public List<Point> PathPoints { get; set; } = new List<Point>();
      public void AddPoint(Point newPoint)
      {
        PathPoints.Add(newPoint);
      }
      public int ListIndex { get; set; }          //THE INDEX OF THE LIST VARIES DEPENDING ON IF
THE OBJECT IS A KEYFRAME OR NOT
      public List<HeliosPoint> GenLaserPoints(LaserSettings currentLaserSettings)
      {
        List<HeliosPoint> points = new List<HeliosPoint>();
        #region higher order maths that i swear is very close to working but isnt quite working.
Think it might have made curves anyway.
        /* DOESNT WORK AS I SO VERY WISHED IT WOULD
        for(int i = 0; i < PathPoints.Count-1; i++)
        {
          double lineFrameLength = getDistance(PathPoints[i], PathPoints[i+1]);
```

```
            double lineFrameProcessed = (double)(currentLaserSettings.MaxAcceleration -
currentLaserSettings.bufferLength);
            HeliosPoint currentPoint;
            double deltaY = PathPoints[i+1].Y - PathPoints[i].Y;
            double deltaX = PathPoints[i + 1].X - PathPoints[i].X;
            int velocity = currentLaserSettings.MaxAcceleration;
            while (lineFrameProcessed < (lineFrameLength / 2))
            {
               currentPoint.x = (ushort) (PathPoints[i].X + (lineFrameProcessed * deltaX));
               currentPoint.y = (ushort) (PathPoints[i].Y + (lineFrameProcessed * deltaY));
               currentPoint.r = PathColor.R;
               currentPoint.g = PathColor.G;
               currentPoint.b = PathColor.B;
               currentPoint.i = PathColor.A;
               points.Add(currentPoint);
               if (velocity + currentLaserSettings.MaxAcceleration < currentLaserSettings.MaxVelocity)
               {
                  velocity += currentLaserSettings.MaxAcceleration;
               }
               lineFrameProcessed += velocity;
            }
            lineFrameProcessed -= velocity;
            lineFrameProcessed = lineFrameLength - lineFrameProcessed;
            while (lineFrameProcessed < (lineFrameLength + currentLaserSettings.bufferLength))
            {
               currentPoint.x = (ushort)(PathPoints[i].X + (lineFrameProcessed * deltaX));
               currentPoint.y = (ushort)(PathPoints[i].Y + (lineFrameProcessed * deltaY));
               currentPoint.r = PathColor.R;
               currentPoint.g = PathColor.G;
               currentPoint.b = PathColor.B;
               currentPoint.i = PathColor.A;
```

```
            points.Add(currentPoint);
            if (velocity + currentLaserSettings.MaxAcceleration < currentLaserSettings.MaxVelocity)
            {
                velocity += currentLaserSettings.MaxAcceleration;
            }
            lineFrameProcessed += velocity;
        }
    }*/
    #endregion
    //If we have constant acceleration life is hella easy thanks to SUVAT equations
    //S = ut + 1/2 a t^2 (hold on one sec we got to the good bit in my playlist I needa jam for a sec)
    if (PathPoints.Count > 0)
    {
        Queue<double> beginningDisplacements = new Queue<double>();
        Stack<double> endingDisplacements = new Stack<double>();
        for (int i = 0; i < currentLaserSettings.DwellPoints; i++)
        {
            beginningDisplacements.Enqueue(0);
        }
        int acceleratingTime = currentLaserSettings.MaxVelocity / currentLaserSettings.MaxAcceleration;
        double currentDisplacement = 0;
        double halfDisplacement = getDistance(this.PathPoints[0], this.PathPoints[1]) / 2;//I realise that I havent added multi point compatibility this is a relatively easy fix for before summer :)
        for (double t = 0; t < acceleratingTime && (0.5 * t * t * currentLaserSettings.MaxAcceleration) < halfDisplacement; t++)
        {
            beginningDisplacements.Enqueue(0.5 * t * t * currentLaserSettings.MaxAcceleration);
            endingDisplacements.Push(0.5 * t * t * currentLaserSettings.MaxAcceleration);
            currentDisplacement = 0.5 * t * t * currentLaserSettings.MaxAcceleration;
        }
```

```
            if (currentDisplacement < halfDisplacement + (currentLaserSettings.MaxVelocity / 2))//If distance left to cover will take more than a point.
            {
                while( currentDisplacement + currentLaserSettings.MaxVelocity < halfDisplacement)
                {
                    currentDisplacement += currentLaserSettings.MaxVelocity;
                    beginningDisplacements.Enqueue(currentDisplacement + currentLaserSettings.MaxVelocity);
                    endingDisplacements.Push(currentDisplacement + currentLaserSettings.MaxVelocity); //Should make it so the middle points are evenly spaced.0

                }

                // This bit fills in the middle so the velocity isnt bigger than the max velocity. The rest uses the max acceleration.
            }
            HeliosPoint currentPoint = new HeliosPoint();
            currentDisplacement = 0;
            double horizontaleScaleValue = (PathPoints[1].X - PathPoints[0].X) / getDistance(this.PathPoints[0], this.PathPoints[1]);
            double verticalScaleValue = (PathPoints[1].Y - PathPoints[0].Y) / getDistance(this.PathPoints[0], this.PathPoints[1]);
            while (beginningDisplacements.Count > 0)
            {
                currentDisplacement = beginningDisplacements.Dequeue();
                currentPoint.x = (ushort)(PathPoints[0].X + currentDisplacement * horizontaleScaleValue);
                currentPoint.y = (ushort)(PathPoints[0].Y + currentDisplacement * verticalScaleValue);
                currentPoint.r = (byte)((this.PathColor.R * this.PathColor.A) / 0xFF);
                currentPoint.g = (byte)((this.PathColor.G * this.PathColor.A) / 0xFF);
                currentPoint.b = (byte)((this.PathColor.B * this.PathColor.A) / 0xFF);
                currentPoint.i = (byte)(this.PathColor.A);
                points.Add(currentPoint);
```

```
            }
            while (endingDisplacements.Count > 0)
            {
                currentDisplacement = endingDisplacements.Pop();
                currentPoint.x = (ushort)(PathPoints[1].X - currentDisplacement *
horizontaleScaleValue);
                currentPoint.y = (ushort)(PathPoints[1].Y - currentDisplacement * verticalScaleValue);
                currentPoint.r = (byte)((this.PathColor.R * this.PathColor.A) / 0xFF);
                currentPoint.g = (byte)((this.PathColor.G * this.PathColor.A) / 0xFF);
                currentPoint.b = (byte)((this.PathColor.B * this.PathColor.A) / 0xFF);
                currentPoint.i = (byte)(this.PathColor.A);
                points.Add(currentPoint);
            }
        }
        else
        {
            points = new List<HeliosPoint>();
        }
        return points;
    }
    public void Reverse()
    {
        PathPoints.Reverse();
    }
    public List<KeyLinePoint> GenKeyPoints(bool middle = false)
    {
        List<KeyLinePoint> KeyPoints = new List<KeyLinePoint>();

        for (int i = 0; i < PathPoints.Count - 1; i++)
        {
            KeyPoints.Add(new KeyLinePoint(ListIndex, i, PathPoints[i], false));
```

```
            if (middle)
            {
                KeyPoints.Add(new KeyLinePoint(ListIndex, i, new Point(
                        Convert.ToInt32((PathPoints[i].X + PathPoints[i + 1].X) / 2),
                        Convert.ToInt32((PathPoints[i].Y + PathPoints[i + 1].Y) / 2))
                        , true));
            }
        }

        KeyPoints.Add(new KeyLinePoint(ListIndex, PathPoints.Count - 1, PathPoints.Last(), false));
        return KeyPoints;
    }
    /*So basically this function generates keypoints, keypoints are any points with details that we might want to adjust
     * This selection tool should work well until there are a lot of points to select*/
    public int getValueXWayBetweenTwoPoints(int value1, int value2, float multiplier)
    {
        return Convert.ToInt32(value1 + (value2 - value1) * multiplier);
    } //This is a function to work out the properties of a nonkeyframe. Iz guud.
    public PathLineFrame(int time, Color pathColor, List<Point> pathPoints, int listIndex)
    {
        this.Time = time;
        this.PathColor = pathColor;
        this.PathPoints = pathPoints;
        this.ListIndex = listIndex;
    }  //Basic constructor
    public PathLineFrame(int time, Color pathColor, Point point1, Point point2, int listIndex)
    {
        Time = time;
        PathColor = pathColor;
        PathPoints = new List<Point>();
```

```
        PathPoints.Add(point1);
        PathPoints.Add(point2);
        ListIndex = listIndex;
    }  //Mid Constructor
    public PathLineFrame()  //Json Constructor
    {
        //Just dont use this
    }
    public PathLineFrame(bool hidden, Point point1, Point point2)
    {
        if (hidden)
        {
            this.Time = -1;
            this.PathColor = Color.Black;
            this.PathPoints = new List<Point>();
            this.PathPoints.Add(point1);
            this.PathPoints.Add(point2);
            this.Hidden = hidden;
            this.ListIndex = -1;
        }
    }
    public PathLineFrame(float animationProgress, PathLineFrame frameBefore, PathLineFrame
frameAfter)
    {
        Time = getValueXWayBetweenTwoPoints(frameBefore.Time, frameAfter.Time,
animationProgress);
        PathColor = Color.FromArgb(
            getValueXWayBetweenTwoPoints(frameBefore.PathColor.A, frameAfter.PathColor.A,
animationProgress),
            getValueXWayBetweenTwoPoints(frameBefore.PathColor.R, frameAfter.PathColor.R,
animationProgress),
```

```
                getValueXWayBetweenTwoPoints(frameBefore.PathColor.G, frameAfter.PathColor.G,
animationProgress),
                getValueXWayBetweenTwoPoints(frameBefore.PathColor.B, frameAfter.PathColor.B,
animationProgress)); //I turned it into a function because I was bound to make a mistake. And
because I need to use this function a LOT.
            while (frameBefore.PathPoints.Count != frameAfter.PathPoints.Count)
            {
               if (frameBefore.PathPoints.Count < frameAfter.PathPoints.Count)
               {
                  frameAfter.PathPoints.Add(frameBefore.PathPoints.Last());
               }
               else
               {
                  frameBefore.PathPoints.Add(frameAfter.PathPoints.Last());
               }
            }  //Makes all extra detail grow out the end of the line --im proud of myself for the
attention to detail, might be buggy tho
            for (int i = 0; i < frameAfter.PathPoints.Count; i++)
            {
               PathPoints.Add(new Point(
                  getValueXWayBetweenTwoPoints(frameBefore.PathPoints[i].X,
frameAfter.PathPoints[i].X, animationProgress),
                  getValueXWayBetweenTwoPoints(frameBefore.PathPoints[i].Y,
frameAfter.PathPoints[i].Y, animationProgress)
                  ));
               }
            }
            /*public PathLineFrame(int newTime, PathLineFrame frame)
            {
               time = newTime;
               PathColor = frame.PathColor;
               PathPoints = frame.PathPoints;
```

```
    }*/
    public PathLineFrame(PathLineFrame frame)
    {
        Time = frame.Time;
        ListIndex = frame.ListIndex;
        PathColor = frame.PathColor;
        PathPoints = new List<Point>();
        foreach (Point point in frame.PathPoints)
        {
            PathPoints.Add(point);
        }
    }
}
class PathLine
{
    public string Name { get; set; }
    public List<PathLineFrame> KeyFrames { get; set; } //A list of all keyframes sorted by time

    public PathLineFrame GetFrameAt(int FrameTime)  //Sort of a recursive algorithm?.
    {
        List<PathLineFrame> touchingFrames = findTouchingFrames(FrameTime, KeyFrames);
        if (touchingFrames.Count == 1)
        {
            return touchingFrames[0];
        }
        KeyFrames.Add(new PathLineFrame(GenFrameAt(FrameTime)));
        this.SortKeyFramesByTime();
        return GetFrameAt(FrameTime);
    }
```

```
        public List<PathLineFrame> findTouchingFrames(int timeOfFrame, List<PathLineFrame>
touchingPoints, bool startingOccurance = true)
        {
            if (startingOccurance)
            {
                touchingPoints = new List<PathLineFrame> { new PathLineFrame(-1, Color.Black, new
List<Point>(), -1) };
                //Added fake beginning abnd ending frames so theres always a frame above and below
or the code would j break
                touchingPoints.AddRange(KeyFrames);
                touchingPoints.Add(new PathLineFrame((int)0x7FFFFFFF, Color.Black, new List<Point>(), -
1));
            }
            int midFrameIndex = touchingPoints.Count() / 2;
            if (touchingPoints[midFrameIndex].Time == timeOfFrame)  //If the mid frame matches the
time
            {
                return new List<PathLineFrame> { touchingPoints[midFrameIndex] };
            }
            else if (timeOfFrame > touchingPoints[midFrameIndex].Time)
            {
                if (timeOfFrame < touchingPoints[midFrameIndex + 1].Time)
                {
                    return new List<PathLineFrame> { touchingPoints[midFrameIndex],
touchingPoints[midFrameIndex + 1] };
                }
                return findTouchingFrames(timeOfFrame, touchingPoints.GetRange(midFrameIndex - 1,
1 + touchingPoints.Count() - midFrameIndex), false);
            }
            else if (timeOfFrame < touchingPoints[midFrameIndex].Time)
            {
                return findTouchingFrames(timeOfFrame, touchingPoints.GetRange(0, midFrameIndex +
1), false);
```

```
            }
            else
            {
                return new List<PathLineFrame>();
            }

        }
        public PathLineFrame GenFrameAt(int FrameTime)
        {
            PathLineFrame newFrame;

            List<PathLineFrame> touchingFrames = findTouchingFrames(FrameTime, new
List<PathLineFrame>());  //This algorithm should be much more efficient :0

            if (touchingFrames.Count == 1)
            {
                return touchingFrames[0];
            }
            else if (touchingFrames[0].Time == -1)              //If before all frames then newframe is the
same as the frame after or first frame
            {
                newFrame = new PathLineFrame(touchingFrames[1]);    //Sets it to the not fake frame
                newFrame.Time = FrameTime;
                return newFrame;
            }
            else if (touchingFrames[1].Time == (int)0x7FFFFFFF)              //Literally the same situation
as the above if
            {
                newFrame = new PathLineFrame(touchingFrames[0]);       //Sets it to the not fake frame
                newFrame.Time = FrameTime;
                return newFrame;
```

```
            }
            else
            {
                //SET ALL PROPERTIES TO PROPERTIES IN BETWEEN THEM BOTH
                float animationProgress = ((float)(FrameTime - touchingFrames[0].Time) / (float)(touchingFrames[1].Time - touchingFrames[0].Time)); //If you set each property to beforeFrame value plus (afterFrame - beforeFrame) * difference  -- This assumes a linear animation hence the constant progress as time moves forward.
                return new PathLineFrame(animationProgress, touchingFrames[0], touchingFrames[1]);
            }
        }
        public PathLine(string name, PathLineFrame keyFrame, int dynamicPathIndex)
        { //For ListIndex just do Path
            Name = name;
            KeyFrames = new List<PathLineFrame>();
            KeyFrames.Add(keyFrame);
        }
        public PathLine()
        {
            //Needed for JSON Conversion IDK why its just how the extension works
        }
        public void SortKeyFramesByTime()
        {
            //Here I impliment quicksort partially explained to me here: https://www.youtube.com/watch?v=SLauY6PpjW4&t=15s
            //I used a recursive routine
            QuicksortByTime(KeyFrames);
        }
        public List<PathLineFrame> QuicksortByTime(List<PathLineFrame> list)
        {
            //Here I impliment quicksort partially explained to me here: https://www.youtube.com/watch?v=SLauY6PpjW4&t=15s
```

```
        //I used a recursive routine
        if (list.Count() == 0)
        {
            return new List<PathLineFrame>();      //Gotta have something end the recursive routine
        }

        List<PathLineFrame> left = new List<PathLineFrame>();
        List<PathLineFrame> right = new List<PathLineFrame>();
        PathLineFrame pivot = list[list.Count / 2];
        //Basically the way pivot works is you keep making lists of numbers greater than the pivot
and less than the pivot until the whole thing is sorted
        //This makes it O(nlogn) because you keep splitting the array in half each time
        //You learn something new everyday!
        for (int i = 0; i < list.Count(); i++)
        {
            if (list[i].Time < pivot.Time)
            {
                left.Add(list[i]);
            }
            else if (list[i].Time > pivot.Time)
            {
                right.Add(list[i]);
            }
        }
        List<PathLineFrame> leftSorted = QuicksortByTime(left);
        List<PathLineFrame> rightSorted = QuicksortByTime(right);
        list.Clear();
        list.AddRange(leftSorted);
        list.Add(pivot);
        list.AddRange(rightSorted);
        return list;
```

```
    }
  }

  //THE GRAPHICS PANEL
  private void PreviewGraphics_Paint(object sender, PaintEventArgs e)
  {
    //Gen framePath (useful for lazer as well.)
    if (framePathTime != mainTime)
    {
      framePathTime = mainTime;
      framePath = new List<PathLineFrame>();
      for (int i = 0; i < project.dynamicPath.Count(); i++)
      {
        PathLineFrame newFrame = project.dynamicPath[i].GenFrameAt(framePathTime);
        newFrame.ListIndex = framePath.Count;
        framePath.Add(newFrame);
      }
    }
    InformationFrameListCountInfo.Text = framePath.Count().ToString();

    laserTraversalPath = GenShapeLaserPath(framePath);  //Literally generates the laser Path
    laserPoints.Clear();
    foreach (PathLineFrame laserLine in laserTraversalPath)
    {
      if (laserPathToolStripMenuItem.Checked)
      {
        if (laserLine.Hidden)
        {
          e.Graphics.DrawLine(new Pen(Color.Blue),
          ConvertToHeliosCoords(laserLine.PathPoints[0], true),
ConvertToHeliosCoords(laserLine.PathPoints[1], true));
```

```
            }
            else
            {
                e.Graphics.DrawLine(new Pen(Color.Green, 2),
                    ConvertToHeliosCoords(laserLine.PathPoints[0], true),
ConvertToHeliosCoords(laserLine.PathPoints[1], true));
            }
          }

          laserPoints.AddRange(laserLine.GenLaserPoints(currentLaserSettings));

        }
        if (laserPointsToolStripMenuItem.Checked)
        {
          foreach (HeliosPoint point in laserPoints)
          {
            e.Graphics.FillCircle(new SolidBrush(Color.Pink), ConvertToHeliosCoords(new
Point(point.x, point.y), true), 2);
          }
        }
        //Draw lines, dots and selection bits.
        Pen linePen;
        double closestPointDistance = 1000000;

        for (int i = 0; i < framePath.Count(); i++) //Goes through all the lines
        {
          //This bit draws all the points out, and the lines if needed, and also shows the points..
          linePen = new Pen(framePath[i].PathColor);
          for (int j = 0; j < framePath[i].PathPoints.Count() - 1; j++)    //REMEMBER TO UNCOMMENT
THIS
          {
```

```
            if (previewShapesToolStripMenuItem.Checked)
            {
                e.Graphics.DrawLine(linePen, ConvertToHeliosCoords(framePath[i].PathPoints[j], true),
ConvertToHeliosCoords(framePath[i].PathPoints[j + 1], true));
            }
        }

        //Im pretty sure this is the function that provides closest points.
        List<KeyLinePoint> keyPoints = new List<KeyLinePoint>();
        keyPoints = framePath[i].GenKeyPoints(OptionsSelectModeButton.Checked);
        for (int j = 0; j < keyPoints.Count(); j++)
        {
            Point pointLocation = keyPoints[j].Location;
            Point pointLocationGraphics = ConvertToHeliosCoords(pointLocation, true);
            if (showCircles)    //Also finds closest point to mouse
            {
                e.Graphics.FillCircle(new SolidBrush(Color.LightGray), pointLocationGraphics.X,
pointLocationGraphics.Y, 4);
                e.Graphics.FillCircle(new SolidBrush(Color.DarkGray), pointLocationGraphics.X,
pointLocationGraphics.Y, 3);
            }
            if (getDistance(pointLocation, mouseLastLocation) < closestPointDistance) //Finds closest
point
            {
                closestPoint = keyPoints[j].Location;
                closestPoints.Clear();
                closestPoints.Add(keyPoints[j]);
                InformationClosestPointData.Text = closestPoint.ToString();
                closestPointDistance = getDistance(pointLocation, mouseLastLocation);
            }
            else if (getDistance(pointLocation, mouseLastLocation) == closestPointDistance)  //Adds
all lines to the closest point, pretty nifty if you ask me :)
```

```
            {
                closestPoints.Add(keyPoints[j]);
            }
        }
    }
    if (showLines)
    {
        try
        {
            if (closestPoints[0].IsMiddle)
            {
                e.Graphics.DrawLine(new Pen(Color.Red, 2),
ConvertToHeliosCoords(project.dynamicPath[closestPoints[0].ShapeListIndex].GenFrameAt(mainTim
e).PathPoints[0], true),

ConvertToHeliosCoords(project.dynamicPath[closestPoints[0].ShapeListIndex].GenFrameAt(mainTim
e).PathPoints[1], true));
                e.Graphics.DrawLine(new
Pen(project.dynamicPath[closestPoints[0].ShapeListIndex].GenFrameAt(mainTime).PathColor),
ConvertToHeliosCoords(project.dynamicPath[closestPoints[0].ShapeListIndex].GenFrameAt(mainTim
e).PathPoints[0], true),

ConvertToHeliosCoords(project.dynamicPath[closestPoints[0].ShapeListIndex].GenFrameAt(mainTim
e).PathPoints[1], true));
            }
            else
            {
                e.Graphics.FillCircle(new SolidBrush(Color.Red), ConvertToHeliosCoords(closestPoint,
true), 3);
            }
        }
        catch
        {
```

```
            }
        }
    }
    private void PreviewGraphics_MouseDown(object sender, MouseEventArgs e)
//MOUSE MOVEMENT
    {
        mouseDown = true;
        //If mouse down, create a new line or frame.
        if (OptionsDrawLineMode.Checked)
        {
            project.dynamicPath.Add(new PathLine("(" + ConvertToHeliosCoords(e.Location).X + "," +
ConvertToHeliosCoords(e.Location).Y + ")", new PathLineFrame(mainTime, DrawerColorDialog.Color,
new List<Point>(), project.dynamicPath.Count), project.dynamicPath.Count));
            selectedLineDynamicIndex = project.dynamicPath.Count - 1;

project.dynamicPath[selectedLineDynamicIndex].GetFrameAt(mainTime).AddPoint(GetSnapOrNoSn
ap(e.Location, closestPoints, OptionsSnapToPoint.Checked));

project.dynamicPath[selectedLineDynamicIndex].GetFrameAt(mainTime).AddPoint(GetSnapOrNoSn
ap(e.Location, closestPoints, OptionsSnapToPoint.Checked));
        }
        else if (OptionsSelectModeButton.Checked && project.dynamicPath.Count > 0)
        {
            if (closestPoints[0].IsMiddle)
            {
                selectedLineDynamicIndex = closestPoints[0].ShapeListIndex;
                middleSelected = true;
                if(project.dynamicPath.Count == 0)
                {
                    mouseDown = false;
                }
            }
            else
```

```
            {
                selectedLineDynamicIndex = closestPoints[0].ShapeListIndex;
                closestPointsFrozen = new List<KeyLinePoint>(closestPoints);
            }

        }
        framePathTime = -1;
        PreviewGraphics.Invalidate();
        UpdateLineProperties();
    }
    Point GetSnapOrNoSnap(Point mouseLocation, List<KeyLinePoint> closestPointsList, bool
snapToPoint)
    {// Remember mouseLocation is converted in this method.
        if (OptionsSnapToPoint.Checked && closestPointsList.Count > 0)
        {
            if(getDistance(closestPointsList[0].Location, ConvertToHeliosCoords(mouseLocation)) <
100)
            {
                return closestPointsList[0].Location;
            }
            else
            {
                return ConvertToHeliosCoords(mouseLocation);
            }
        }
        else
        {
            return ConvertToHeliosCoords(mouseLocation);
        }
    }
```

```
    private void PreviewGraphics_MouseMove(object sender, MouseEventArgs e)
//MOUSE ACTIVITY
    {
        if (mouseDown)
        {
            if (OptionsDrawLineMode.Checked)
            {
                try
                {
                    GetSelectedFrameWrite().PathPoints[1] = GetSnapOrNoSnap(e.Location, closestPoints,
OptionsSnapToPoint.Checked);
                }
                catch
                {

                }
            }
            else if (OptionsSelectModeButton.Checked && middleSelected == false &&
project.dynamicPath.Count > 0)
            {
                GetSelectedFrameWrite().PathPoints[closestPointsFrozen[0].PathPointsListIndex] =
GetSnapOrNoSnap(e.Location, closestPoints, OptionsSnapToPoint.Checked);
            }
        }
        mouseLastLocation = ConvertToHeliosCoords(e.Location);
        PreviewGraphics.Invalidate();
        framePathTime = -1;
        UpdateLineProperties();
    }
    private void PreviewGraphics_MouseUp(object sender, MouseEventArgs e)
//MOUSE SHENINAGINS
    {
```

```
        if(project.dynamicPath.Count > 0)
        {
            mouseDown = false;
            if (project.dynamicPath[selectedLineDynamicIndex].GetFrameAt(mainTime).PathPoints[0]
== e.Location)
            {

project.dynamicPath[selectedLineDynamicIndex].KeyFrames.Remove(project.dynamicPath[selectedL
ineDynamicIndex].GetFrameAt(mainTime));
                if (project.dynamicPath[selectedLineDynamicIndex].KeyFrames.Count == 0)
                {
                    project.dynamicPath.RemoveAt(selectedLineDynamicIndex);
                }
            }
            middleSelected = false;
            closestPointsFrozen.Clear();
            framePathTime = -1;
            PreviewGraphics.Invalidate();
            UpdateLineProperties();
            mouseLastLocation = ConvertToHeliosCoords(e.Location);
        }

    }
    private void OptionsColorSelecterOpener_Click(object sender, EventArgs e)
    {
        DrawerColorDialog.ShowDialog();
    }

    private void PreviewGraphics_Resize(object sender, EventArgs e)
    {
        int maxHorizontalSpace = splitContainer1.Panel1.Width - PreviewGraphics.Margin.Horizontal;
//300 for the rhs panels
```

```
        int maxVerticalSpace = splitContainer1.Panel1.Height - (splitContainer1.Panel1.Height / 5) -
PreviewGraphics.Margin.Vertical - TimeLinePanel.Margin.Vertical;   //150 for the bottom panel
        if (maxHorizontalSpace < maxVerticalSpace)
        {
            maxVerticalSpace = maxHorizontalSpace;
        }
        else
        {
            maxHorizontalSpace = maxVerticalSpace;  //The actual max size is the smallest of the two.
        }
        PreviewGraphics.Size = new Size(maxHorizontalSpace - PreviewGraphics.Margin.Horizontal,
maxVerticalSpace - 2 * PreviewGraphics.Margin.Vertical);
        TimeLinePanel.Size = new Size(splitContainer1.Panel1.Width -
TimeLinePanel.Margin.Horizontal, splitContainer1.Panel1.Height - maxVerticalSpace);
        TimeLinePanel.Location = new Point(TimeLinePanel.Margin.Left,
splitContainer1.Panel1.Height - (TimeLinePanel.Height + TimeLinePanel.Margin.Bottom));
    }

    private void OptionsDrawLineMode_CheckedChanged(object sender, EventArgs e)
    {
        OptionsSelectModeButton.Checked = !OptionsDrawLineMode.Checked;
        if (OptionsDrawLineMode.Checked)
        {
            OptionsDrawLineMode.BackColor = Color.Green;
        }
        else
        {
            OptionsDrawLineMode.BackColor = Color.Red;
        }
    }

    private void OptionsSelectModeButton_CheckedChanged(object sender, EventArgs e)
```

```
    {
        OptionsDrawLineMode.Checked = !OptionsSelectModeButton.Checked;
        showCircles = OptionsSelectModeButton.Checked;
        showLines = OptionsSelectModeButton.Checked;
        if (OptionsSelectModeButton.Checked)
        {
            OptionsSelectModeButton.BackColor = Color.Green;
        }
        else
        {
            OptionsSelectModeButton.BackColor = Color.Red;
        }
    }

    private void OptionsSnapToPoint_CheckedChanged(object sender, EventArgs e)
    {
        if (OptionsSnapToPoint.Checked)
        {
            OptionsSnapToPoint.BackColor = Color.Green;
        }
        else
        {
            OptionsSnapToPoint.BackColor = Color.Red;
        }
    }

    private void TimeLineSecondsInput_TextChanged(object sender, EventArgs e)
    {
        try
        {
```

```
            if (TimeLineFramesInput.Text != "")
            {
                mainTime = Convert.ToInt32(TimeLineSecondsInput.Text) * project.fps;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Error: cannot convert text to number: " +
TimeLineFramesInput.Text);
        }
    }
    private void TimeLineFramesInput_TextChanged(object sender, EventArgs e)
    {
        try
        {
            if (TimeLineFramesInput.Text != "")
            {
                ChangeTime(Convert.ToInt32(TimeLineFramesInput.Text));
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Error: cannot convert text to number: " +
TimeLineFramesInput.Text);
        }
        UpdateLineProperties();
        PreviewGraphics.Invalidate();
    }

    void UpdateLineProperties()
    {
```

```
        // If currently playing, dont bother to update line properties
        if (currentlyPlaying)
        {
            return;
        }

        // Invalidate timeline GUI
        timelineGUI.Invalidate();

        // If dynamic path has no points dont run this code.
        if (project.dynamicPath.Count == 0)
        {
            return;
        }

        // Ensure selected dynamic line index is within bounds
        if (selectedLineDynamicIndex > project.dynamicPath.Count)
        {
            // If out of bounds, reset to 0
            selectedLineDynamicIndex = 0;
            selectedPointDynamicIndex = 0;
        }

        // Set selected frame to frame generated by selected dynamic line at current mainTime
        // This frame is self-updating
        selectedFrameReadOnly =
project.dynamicPath[selectedLineDynamicIndex].GenFrameAt(mainTime);

        //Ensures the item is within the selected index by changing it to the closest allowed value.
        selectedPointDynamicIndex = Math.Clamp(selectedPointDynamicIndex, 0,
selectedFrameReadOnly.PathPoints.Count - 1);
```

```
        // Clear items in PathLinePointsListBox and add each point in selected frame
        PathLinePointsListBox.BeginUpdate();    //Ag
        PathLinePointsListBox.Items.Clear();
        foreach (var pathPoint in selectedFrameReadOnly.PathPoints)
        {
            PathLinePointsListBox.Items.Add(pathPoint.ToString());
        }
        PathLinePointsListBox.EndUpdate();

        // Set LinePropertiesTitle label to display name of selected dynamic line
        LinePropertiesTitle.Text = "Line Properties: " +
project.dynamicPath[selectedLineDynamicIndex].Name;

        // Set LinePropertiesPathIndexData label to display index of selected dynamic line
        LinePropertiesPathIndexData.Text = selectedLineDynamicIndex.ToString();

        // If there is a selected point, set X and Y value of LinePropertiesXCoordinate and
LinePropertiesYCoordinate
        if (selectedPointDynamicIndex != -1)
        {
            Point tempPoint = selectedFrameReadOnly.PathPoints[selectedPointDynamicIndex];
            tempIntermediateFramePoint.X = tempPoint.X; //I don't know if there is a better way to do
this it feels a bit tedious.
            LinePropertiesXCoordinate.Value = tempPoint.X;
            tempIntermediateFramePoint = tempPoint;
            LinePropertiesYCoordinate.Value = tempPoint.Y;
        }

        // Clear items in LinePropertiesKeyFramesTextBox and add each keyframe time in selected
dynamic line
```

```
        LinePropertiesKeyFramesTextBox.BeginUpdate();   //This line and endupdate allow the list to update without flickering.
        LinePropertiesKeyFramesTextBox.Items.Clear();
        foreach (var keyFrame in project.dynamicPath[selectedLineDynamicIndex].KeyFrames)
        {
            LinePropertiesKeyFramesTextBox.Items.Add(keyFrame.Time);
        }
        LinePropertiesKeyFramesTextBox.EndUpdate();

        // Set color of LinePropertiesChangeColor label to color of selected dynamic line at current mainTime
        LinePropertiesChangeColor.ForeColor = selectedFrameReadOnly.PathColor;
    }

    PathLineFrame GetSelectedFrameWrite()
    {
        return project.dynamicPath[selectedLineDynamicIndex].GetFrameAt(mainTime);
    }
    void ChangeTime(int newTime)
    {
        selectedLineDynamicIndex = 0;
        selectedPointDynamicIndex = -1;
        mainTime = newTime;
        UpdateLineProperties();
        timelineGUI.Invalidate();
        PreviewGraphics.Invalidate();
    }
    private void LinePropertiesKeyFramesTextBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (LinePropertiesKeyFramesTextBox.SelectedIndex != -1 && project.dynamicPath.Count > 0)
```

```
        {
            mainTime =
project.dynamicPath[selectedLineDynamicIndex].KeyFrames[LinePropertiesKeyFramesTextBox.Selec
tedIndex].Time;
            UpdateLineProperties();
        }
    }

    private void PathLinePointsListBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (PathLinePointsListBox.SelectedIndex != -1)
        {
            selectedPointDynamicIndex = PathLinePointsListBox.SelectedIndex;
            UpdateLineProperties();
        }
    }
    public class TimelineSettings
    {
        public float PixelsPerSecond { get; set; } = 20;
        public float LeftMargin { get; set; }  = 10;
        public float TopMargin { get; set; }  = 10;
        public float PixelsPerShape { get; set; } = 20;
        public float TimelineDotSize { get; set; }  = 5;
        public Font SecondsFont { get; set; } = new Font("Arial", 8, FontStyle.Bold);
        public TimelineSettings(int pixelsPerSecond = 20, int pixelsPerShape = 20, Font secondsFont =
null)
        {
            this.PixelsPerSecond = pixelsPerSecond;
            this.PixelsPerShape = pixelsPerShape;
            if (secondsFont != null)
            {
```

```
            this.SecondsFont = secondsFont;
        }
        this.LeftMargin = pixelsPerSecond / 2;
        this.TopMargin = pixelsPerShape / 2;
    }
    public float getFloatXOfFrameTime(int frameTime, int fps)
    {
        return this.LeftMargin + (frameTime) * (this.PixelsPerSecond / fps) + this.PixelsPerSecond;
    }
    public TimelineSettings()
    {
        //JSON fies
    }
}


private void timeline_GUI_updater(object sender, PaintEventArgs e)
{
    timelineGUI.Size = new Size(Convert.ToInt32((float)project.maxtimeSeconds *
currentTimelineSettings.PixelsPerSecond + 2 * currentTimelineSettings.LeftMargin),
        Convert.ToInt32(currentTimelineSettings.TopMargin * 2 +
currentTimelineSettings.PixelsPerShape * (project.dynamicPath.Count)) + 10);
    int seconds = 0;
    Pen thinWhitePen = new Pen(Color.White);
    float currentTimeX = currentTimelineSettings.LeftMargin + (mainTime + project.fps) *
(currentTimelineSettings.PixelsPerSecond / project.fps);
    e.Graphics.DrawLine(thinWhitePen, new PointF(currentTimeX,
currentTimelineSettings.PixelsPerShape), new PointF(currentTimeX, timelineGUI.Size.Height));


    // Draw seconds markers
    for (float horizontalPixelsUsed = currentTimelineSettings.LeftMargin; horizontalPixelsUsed <
timelineGUI.Size.Width; horizontalPixelsUsed += currentTimelineSettings.PixelsPerSecond)
    {
```

```
            e.Graphics.DrawString((seconds - 1).ToString(), currentTimelineSettings.SecondsFont, new
SolidBrush(Color.White),
                new PointF(horizontalPixelsUsed - ((int)((currentTimelineSettings.SecondsFont.Size *
seconds.ToString().Count()) / 2)),
                    currentTimelineSettings.TopMargin + timelineGUIHugger.VerticalScroll.Value));
            seconds++;
        }

        // Draw shape number markers
        int shapeNumber = 1;
        for (float verticalPixelsUsed = currentTimelineSettings.TopMargin +
currentTimelineSettings.PixelsPerShape; verticalPixelsUsed < timelineGUI.Size.Height;
verticalPixelsUsed += currentTimelineSettings.PixelsPerShape)
        {
            e.Graphics.DrawString(shapeNumber.ToString(), currentTimelineSettings.SecondsFont, new
SolidBrush(Color.White),
                new PointF(currentTimelineSettings.LeftMargin +
timelineGUIHugger.HorizontalScroll.Value - ((int)((currentTimelineSettings.SecondsFont.Size *
shapeNumber.ToString().Count()) / 2)),
                    verticalPixelsUsed));
            shapeNumber++;
        }

        // Draw keyframes for each dynamic path
        int dynamicPathTempIndex = 0;
        TimelineDots.Clear();
        for (float verticalSpaceUsed = currentTimelineSettings.TopMargin; verticalSpaceUsed <
Convert.ToInt32(currentTimelineSettings.TopMargin * 2 + currentTimelineSettings.PixelsPerShape *
(project.dynamicPath.Count)); verticalSpaceUsed += currentTimelineSettings.PixelsPerShape)
        {
            if (project.dynamicPath.Count != 0 && project.dynamicPath.Count >
dynamicPathTempIndex)
            {
```

```
            foreach (PathLineFrame frame in
project.dynamicPath[dynamicPathTempIndex].KeyFrames)
            {
                TimelineDots.Add(new TimeLinePoint(dynamicPathTempIndex, frame.Time, new
Point((int)currentTimelineSettings.getFloatXOfFrameTime(frame.Time, project.fps), (int)(
                    currentTimelineSettings.TopMargin + (dynamicPathTempIndex + 1) *
currentTimelineSettings.PixelsPerShape))));

                e.Graphics.FillCircle(new SolidBrush(Color.LightGray),
(int)currentTimelineSettings.getFloatXOfFrameTime(frame.Time, project.fps), (int)(
                    currentTimelineSettings.TopMargin + (dynamicPathTempIndex + 1) *
currentTimelineSettings.PixelsPerShape), 4);

                e.Graphics.FillCircle(new SolidBrush(Color.DarkGray),
(int)currentTimelineSettings.getFloatXOfFrameTime(frame.Time, project.fps), (int)(
                    currentTimelineSettings.TopMargin + (dynamicPathTempIndex + 1) *
currentTimelineSettings.PixelsPerShape), 3);
            }
            dynamicPathTempIndex++;
          }
        }

        // Draw closest point marker if it's actually close
        if (timelineClosestPoint != null)
        {
          if (getDistance(timelineClosestPoint.Location, timelineMouseLastLocation) < 5)
          {
            e.Graphics.FillCircle(new SolidBrush(Color.Red), timelineClosestPoint.Location, 3);
          }
        }
      }

      private void timelineGUI_MouseMove(object sender, MouseEventArgs e)
```

```
    {
        if (e.Button == MouseButtons.Left)
        {
            int newTimeFrame = (int)((e.Location.X - currentTimelineSettings.LeftMargin -
currentTimelineSettings.PixelsPerSecond
                ) / (currentTimelineSettings.PixelsPerSecond / project.fps));
            if (newTimeFrame > 0)
            {
                ChangeTime(newTimeFrame);
            }
        }
        double closestPointDistance = 100000;
        for (int j = 0; j < TimelineDots.Count(); j++)
        {
            Point pointLocation = TimelineDots[j].Location;
            if (getDistance(pointLocation, e.Location) < closestPointDistance) //Finds closest point
            {
                timelineClosestPoint = TimelineDots[j];
                closestPointDistance = getDistance(timelineClosestPoint.Location, e.Location);
            }
        }
        timelineMouseLastLocation = e.Location;
        timelineGUI.Invalidate();
    }

    private void timelineGUI_MouseDown(object sender, MouseEventArgs e)
    {
        if (timelineClosestPoint != null)
        {
            if (getDistance(timelineClosestPoint.Location, timelineMouseLastLocation) < 5)
            {
```

```
            selectedLineDynamicIndex = timelineClosestPoint.ShapeListIndex;
            ChangeTime(timelineClosestPoint.FrameTime);
          }
        }
      }

      private void saveToolStripMenuItem_Click(object sender, EventArgs e)
      {
        saveFileDialog1.ShowDialog();
        project.fileHash = "";
        // Calculate the MD5 hash of the project object
        project.fileHash = GetHash(project);
        JsonSerialization.WriteToJsonFile<PathProject>(saveFileDialog1.FileName, project);
      }

      public string GetHash<T>(T currentObject)   //The capital T is the Type and is worked out per call of the method.
      {
        // This was quite confusing to me as I had to find a way to convert the object into a single string
        // Calculate the MD5 hash of the project object
        // Convert the object to a byte array
        byte[] data = Encoding.UTF8.GetBytes(Newtonsoft.Json.JsonConvert.SerializeObject(currentObject)); // Convert object to a byte array

        // Generate the MD5 hash
        MD5 md5 = MD5.Create();
        byte[] hash = md5.ComputeHash(data);
        string hashString = BitConverter.ToString(hash);
        return Convert.ToBase64String(hash);
      }
```

```
    private void openToolStripMenuItem_Click(object sender, EventArgs e)
    {
        currentLaserSettings.ShowLaser = false;
        try
        {
            openFileDialog1.ShowDialog();
            project = JsonSerialization.ReadFromJsonFile<PathProject>(openFileDialog1.FileName);
            String projectsFileHash = project.fileHash;
            project.fileHash = "";
            if (GetHash(project) != projectsFileHash)
            {
                MessageBox.Show("File modified outside of software or corrupted." +
                    " Be careful when using the file as it may be dangerous or damage device. ",
"Warning");
            }
            selectedLineDynamicIndex = 0;
            selectedPointDynamicIndex = 0;
            ChangeTime(0);
        }
        catch
        {
            MessageBox.Show("File incorrect or corrupt, try again or check the JSON to see if its
formatted correctly.");
        }

    }

    private void connectToDACToolStripMenuItem_Click(object sender, EventArgs e)
    {
        InformationPreviewModeData.Text = helios.openDevices().ToString();
    }
```

```
    private void disconnectDACToolStripMenuItem_Click(object sender, EventArgs e)
    {
        InformationPreviewModeData.Text = "Disconnected";
        helios.closeDevices();
    }

    private void backgroundWorker1_DoWork(object sender,
System.ComponentModel.DoWorkEventArgs e)
    {
        while (true)
        {
            if (currentLaserSettings.ShowLaser)
            {
                while (helios.getStatus(0) == 0)
                {
                    Thread.Sleep(1);
                }
                helios.writeFrame(0, currentLaserSettings.Kpps, 0, laserPoints.ToArray(),
laserPoints.Count());
            }
            else
            {
                Thread.Sleep(100);
            }
        }
    }

    private void splitContainer1_Panel1_Paint(object sender, PaintEventArgs e)
    {
```

```
    }

    private void linePropertiesXOrYCoordinate_ValueChanged(object sender, EventArgs e)
    {
        if (tempIntermediateFramePoint.X == -1 || tempIntermediateFramePoint.Y == -1)
        {
            return;
        }
        if (tempIntermediateFramePoint.X == LinePropertiesXCoordinate.Value &&
tempIntermediateFramePoint.Y == LinePropertiesYCoordinate.Value)
        {
            return; //If this was set to show the value of a midway frame, dont generate a new
keypoint, only if user wanted new keyframe.
        }
        if (project.dynamicPath.Count > 0 && GetSelectedFrameWrite().PathPoints.Count >
(selectedPointDynamicIndex))
        {
            GetSelectedFrameWrite().PathPoints[selectedPointDynamicIndex] = new
Point(Convert.ToInt32(LinePropertiesXCoordinate.Value),
Convert.ToInt32(LinePropertiesYCoordinate.Value));
            this.PreviewGraphics.Invalidate();
            UpdateLineProperties();
        }
    }

    private void deleteShape_Click(object sender, EventArgs e)
    {
        if (selectedLineDynamicIndex != -1 && project.dynamicPath.Count > 0)
        {
            project.dynamicPath.RemoveAt(selectedLineDynamicIndex);
            ChangeTime(mainTime);
            PreviewGraphics.Invalidate();
```

```
            UpdateLineProperties();
            selectedLineDynamicIndex = 0;
            selectedPointDynamicIndex = 0;
        }
    }

    private void Form1_KeyPress(object sender, KeyPressEventArgs e)
    {
        if (e.KeyChar == ' ')
        {

            currentLaserSettings.ShowLaser = false;
            HeliosPoint[] blackFrame = { new HeliosPoint() };
            blackFrame[0].x = (ushort)(0x000);
            blackFrame[0].y = (ushort)(0x000);
            blackFrame[0].r = (byte)(0x00);
            blackFrame[0].g = (byte)(0x00);
            blackFrame[0].b = (byte)(0x00);
            blackFrame[0].i = (byte)(0x00);
            helios.writeFrame(0, currentLaserSettings.Kpps, 1, blackFrame, 1);
            OptionsToggleProject.Checked = false;
        }
        if (e.KeyChar == 'p')
        {
            PreviewGraphics.Invalidate();
            currentLaserSettings.ShowLaser = true;
        }
    }

    private void OptionsToggleProject_CheckedChanged(object sender, EventArgs e)
```

```
{
    if (OptionsToggleProject.Checked)
    {
        PreviewGraphics.Invalidate();
        currentLaserSettings.ShowLaser = true;
    }
    else
    {
        currentLaserSettings.ShowLaser = false;
        HeliosPoint[] blackFrame = { new HeliosPoint() };
        blackFrame[0].x = (ushort)(0x000);
        blackFrame[0].y = (ushort)(0x000);
        blackFrame[0].r = (byte)(0x00);
        blackFrame[0].g = (byte)(0x00);
        blackFrame[0].b = (byte)(0x00);
        blackFrame[0].i = (byte)(0x00);
        helios.writeFrame(0, currentLaserSettings.Kpps, 1, blackFrame, 1);
    }
}

private void LinePropertiesChangeColor_Click(object sender, EventArgs e)
{
    if(project.dynamicPath.Count > 0)
    {
        LinePropertiesColorDialog.ShowDialog(this);
        GetSelectedFrameWrite().PathColor = LinePropertiesColorDialog.Color;
    }
}

private void ToolStripMenuItem_Click(object sender, EventArgs e)
```

```
    {
        Settings settingsWindow = new Settings();
        settingsWindow.Show();
        try
        {
            currentLaserSettings =
JsonSerialization.ReadFromJsonFile<LaserSettings>(@"//CurrentLaserSettings.Config");
            currentTimelineSettings =
JsonSerialization.ReadFromJsonFile<TimelineSettings>(@"//CurrentTimelineSettings.Config");
        }
        catch
        {


        }
    }


    private void backgroundWorker2_DoWork(object sender,
System.ComponentModel.DoWorkEventArgs e)
    {
        int sleepTime = 10;
        float timeIncrement = (sleepTime / 1000f) * project.fps;
        float preciseTime = mainTime;
        while (currentlyPlaying)
        {
            Thread.Sleep(sleepTime);
            preciseTime += timeIncrement;
            //Change time method without updating line properties
            selectedPointDynamicIndex = -1;
            mainTime = Convert.ToInt32(preciseTime);
            timelineGUI.Invalidate();
            PreviewGraphics.Invalidate();
```

```
            if (mainTime > project.maxtimeSeconds * project.fps)
            {
                selectedPointDynamicIndex = -1;
                mainTime = Convert.ToInt32(0);
                preciseTime = 0;    //This gave me hella headaches
                timelineGUI.Invalidate();
                PreviewGraphics.Invalidate();
            }
        }
        if (!currentlyPlaying)
        {
            backgroundWorker2.Dispose();
        }
        return;
    }

    private void projectMaxTimeSelector_ValueChanged(object sender, EventArgs e)
    {
        project.maxtimeSeconds = (float)projectMaxTimeSelector.Value;
    }

    private void TimeLinePlay_CheckedChanged(object sender, EventArgs e)
    {
        currentlyPlaying = TimeLinePlay.Checked;
        if (!currentlyPlaying)
        {
            backgroundWorker2.Dispose();
        }
        else
        {
```

```
            backgroundWorker2.RunWorkerAsync();
        }
    }



}
#region Borrowed code used for saving files in a human readable format & serialising files. I do understand it but because I had so much help I wont take credit.
/// <summary>
/// Functions for performing common Json Serialization operations.
/// <para>Requires the Newtonsoft.Json assembly (Json.Net package in NuGet Gallery) to be referenced in your project.</para>
/// <para>Only public properties and variables will be serialized.</para>
/// <para>Use the [JsonIgnore] attribute to ignore specific public properties or variables.</para>
/// <para>Object to be serialized must have a parameterless constructor.</para>
/// </summary>
public static class JsonSerialization
{
    /// <summary>
    /// Writes the given object instance to a Json file.
    /// <para>Object type must have a parameterless constructor.</para>
    /// <para>Only Public properties and variables will be written to the file. These can be any type though, even other classes.</para>
    /// <para>If there are public properties/variables that you do not want written to the file, decorate them with the [JsonIgnore] attribute.</para>
    /// </summary>
    /// <typeparam name="T">The type of object being written to the file.</typeparam>
    /// <param name="filePath">The file path to write the object instance to.</param>
    /// <param name="objectToWrite">The object instance to write to the file.</param>
    /// <param name="append">If false the file will be overwritten if it already exists. If true the contents will be appended to the file.</param>
```

```
    public static void WriteToJsonFile<T>(string filePath, T objectToWrite, bool append = false)
where T : new()
    {
        TextWriter writer = null;
        try
        {
            var contentsToWriteToFile = Newtonsoft.Json.JsonConvert.SerializeObject(objectToWrite);
            try
            {
                writer = new StreamWriter(filePath, append);
                writer.Write(contentsToWriteToFile);
            }
            catch
            {


            }
        }
        finally
        {
            if (writer != null)
                writer.Close();
        }
    }


    /// <summary>
    /// Reads an object instance from an Json file.
    /// <para>Object type must have a parameterless constructor.</para>
    /// </summary>
    /// <typeparam name="T">The type of object to read from the file.</typeparam>
    /// <param name="filePath">The file path to read the object instance from.</param>
    /// <returns>Returns a new instance of the object read from the Json file.</returns>
```

```
    public static T ReadFromJsonFile<T>(string filePath) where T : new()
    {
        TextReader reader = null;
        try
        {
            reader = new StreamReader(filePath);
            var fileContents = reader.ReadToEnd();
            return Newtonsoft.Json.JsonConvert.DeserializeObject<T>(fileContents);
        }
        finally
        {
            if (reader != null)
                reader.Close();
        }
    }

}

#endregion
public static class GraphicsExtensions      //This code (the graphics extensions) is borrowed code. IT works really well so I am keeping it
{
    public static void DrawCircle(this Graphics g, Pen pen,
                                  float centerX, float centerY, float radius)
    {
        g.DrawEllipse(pen, centerX - radius, centerY - radius,
                      radius + radius, radius + radius);
    }

    public static void FillCircle(this Graphics g, Brush brush,
                                  float centerX, float centerY, float radius)
```

```
        {
            g.FillEllipse(brush, centerX - radius, centerY - radius,
                          radius + radius, radius + radius);
        }//End of borrowed code.
        public static void FillCircle(this Graphics g, Brush brush,
                                      Point center, float radius)
        {
            g.FillEllipse(brush, center.X - radius, center.Y - radius,
                          radius + radius, radius + radius);
        }
    }
}
```

3.4 My Technical code (HeliosDACLeosPart.Dll)

```
4   using System;
5   using System.Runtime.InteropServices;
6
7   namespace HeliosLaserDAC
8   {
9
10      //point data structure
11      [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode, Pack = 8,Size = 7)]
12      public struct HeliosPoint
13      {
14          public UInt16 x; //12 bit (from 0 to 0xFFF)
15          public UInt16 y; //12 bit (from 0 to 0xFFF)
16          public byte r; //8 bit      (from 0 to 0xFF)
17          public byte g; //8 bit (from 0 to 0xFF)
18          public byte b; //8 bit (from 0 to 0xFF)
19          public byte i; //8 bit (from 0 to 0xFF)
20      }
21
22      public class HeliosDac
23      {
24          [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
25          static extern int OpenDevices();
26          [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl,
    CharSet = CharSet.Unicode, ExactSpelling = false)]
27          static unsafe extern int WriteFrame(uint devNum, int pps, byte flags, HeliosPoint* frame, uint
    numOfPoints);
28          [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
29          static extern int CloseDevices();
```

```
30          [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
31          static extern string GetName(uint devNum);
32          [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
33          static extern UInt16 GetFirmwareVersion(uint devNum);
34          [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
35          static extern int GetStatus(uint devNum);
36
37          public int openDevices()
38          {
39              return OpenDevices();
40          }
41
42          public unsafe int writeFrame(int devNum, int pps, byte flags, HeliosPoint[] frame, int
    numOfPoints)
43          {
44              fixed (HeliosPoint* frameAdd = frame)
45              {
46                  int result = WriteFrame(Convert.ToUInt32(devNum), pps, flags, frameAdd,
    Convert.ToUInt32(numOfPoints));
47                  return result;
48              }
49          }
50
51          public HeliosDac()
52          {
53              openDevices();
54          }
55          public int getStatus(int deviceNumber)
56          {
57              return GetStatus(Convert.ToByte(deviceNumber));
58          }
59          public void closeDevices()
60          {
61              CloseDevices();
62          }
63      }
64  }
```

4 Testing

During my testing the first problem I incurred was a mis-selection of points, wherein on a very specific occasion my lines wouldn't select properly. After lots of testing and eventual duplication, I found that the lists were not consistent, therefore when referencing them the problem began to show.

I managed to fix this problem by making sure the index was stored properly in the code. Each objective has a number and this number is matched to a number in the test intention, each test is designed to test the success of one or more objectives.

Note: test video contains lasers, DO NOT WATCH IF YOU CANNOT HANDLE FLASHING LIGHTING.

4.1 Objectives being tested

Safety

- Ability by the user to turn off the laser using the software at any time.
- In person warnings given to alert users how to treat these devices with appropriate caution.
    1. These must explain that usage can cause blindness if eyes are exposed to the laser radiation
    2. These must explain that usage can cause fire
    3. These must be shown to the user before they hit play
- Appropriate algorithms to prevent hardware damaging requests to the device. **Test: 3,4, 30**
    1. These must block the laser from leaving the range 0&4096 as either side further will damage it. **Test: 3, 4,**
    2. These must have a limiting maximum velocity that can be set in the code. **Test: 30**
    3. These must have a limiting maximum acceleration that can be set in the code. **Test: 30**
- Dark Mode so it can be used in the dark, and allow the user to see the surroundings without adjusting eyes to allow for situational awareness. To ensure brightness of the GUI isn't preventing users from being aware of anyone within the laser exposure zone.
    1. The software must have mainly black backgrounds
    2. There must be contrast between the text and background so it remains legible / useable when it is dark.
- File verification **Test: 24,25,26,27 – note the bug is still safe as no laser code runs and safety is the most important albeit the bug is annoying.**
    1. The software must be able to handle invalid files. **Test: 27**
    2. The software must have a warning if the JSON has been edited but can still be opened (so that any damaging requests are at the users responsibility). **Test: 26**

Affordability

- Laser hardware needs to <£1000
- Laser must be bright.
    1. Beam must be visible at night with smoke. **Test: lines visible in tests 31-35**
- Efficient path generation algorithms to prevent the need for more unnecessarily costly hardware. **Test: 29,30,34**

    1. The algorithms must be able to run more than 1 frame per second. **Test: 34,**
    2. The algorithms must be able to distribute spacing based on detail. **Test: 29,30**
- No paid for code required.
    1. The code file must be able to run without paying for any other software (drivers auto install and are free).
- Works with a Helios Dac that can project 30000 kps. **Test: 28**
    1. The dac should be able to plug into a computer and connect like any peripheral would. **Test: 28**

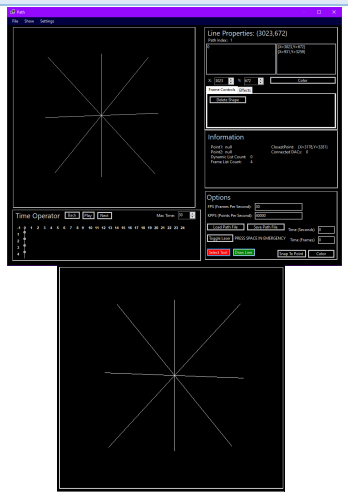
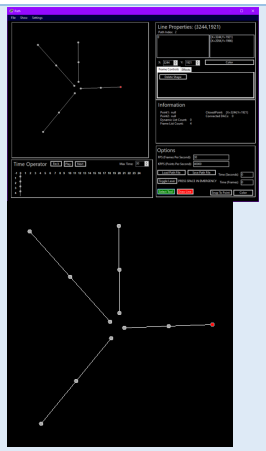
Useability

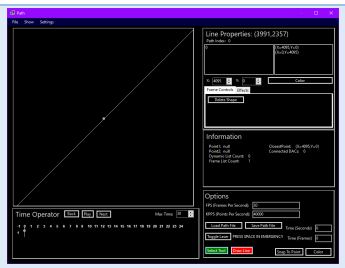
- Intuitive GUI to allow any clients with IT skills to generate images on their own.
    1. Intuitive meaning that it doesn't require any previous training to use.
    2. The end user should ideally be able to understand the GUI when shown to them.
    3. Single graphics layout so there's no hidden functionality.
    4. Lines drawn from the first click. **Test 1,**
- Responsive software, that runs "out the box". **Test: 31**
    1. The laser can run as soon as you click toggle laser, no setup required (other than setting up the hardware physically). **Test: 31**
- No console usage, all in a display.
    1. The system should be able to run purely via a graphics based interface.
- GUI that can handle variable amounts of detail. **Test: 1,7, 13, 14, 15,**
    1. The system should be able to draw a simple frame as well as a frame with large amounts of shapes and changes over time. **Test: 1, 13, 14, 15**
    2. The system should have guidance (i.e. snap into place) **Test: 7, 8**
- GUI that can modify lines graphically post drawing. **Test: 2, 8~~, 18~~**
    1. The frame should be changeable without deleting shapes.
- GUI that can allow for mistake correction. **Test: 11,**
    1. There should be the option to delete specific shapes if they weren't necessary etc.
- A reliable GUI **Test: 1-6,**
    1. Consistent output given input **Test: 6**
    2. Doesn't crash if invalid data is inputted **Test: 9, 10, 12, 17, 23**

Customisability

- Choice of colours, minimum of 6 colours. **Test: 5, 16**
    1. The shapes should be able to have many different colours, ideally at different times.
- Ability to project dynamic animations. **Test: 19,20,21,22,23**
    1. The system should be able to loop a changing projection animation that isn't being modified by a user at the time. **Test: 19,20,21,22,23**
- Variable animation times. **Test 22 (shown during play as one animation takes two seconds the other a fraction of a second)**
    1. The length of the transitions must be variable, ie one be fast one be slow. **Test 22**
- Variable animation positions. **Test: 19,22**
    1. The lines should be able to change positions during animations. **Test: 19**
- The ability to swap which point is shown in the line properties using the selection tool.
    1. This is a niche thing that would be nice but could cause many bugs.

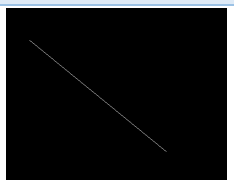
4.2 Test tables

GUI – test basic line input function. No snap to point.					
Test No.	**Test Intention**	**Test Data**	**Expected Output**	**Actual Output**	**Pass/Fail**
1	Can lines be drawn? Check multiple angles and positions to make sure gradients work as intended.	Draw lines of multiple angles within the boundaries.	The lines with the starting point and end point shown	 Testing Video Timestamp: 0:08	Pass
2	Ensure the select tool can accurately modify lines.	Aim to modify the lines into many positions, moving both sides of each line and making sure you try all gradients.	The lines move with intuitive indicators of which points were selected. The movement is shown in my video.	 Testing Video Timestamp: 0:22	Pass
3	Test to determine if the lines can go outside of the range of the laser by selecting and moving lines.	Drag the points of the lines into the borders and see if they get stopped.	They cannot go out of the 0<x<4095 boundary	 Video Timestamp: 0:41	Pass – except the box doesn't quite cover the whole 0<x<4095 so they can go out on the GUI but the laser itself is safe.

4	Can lines be drawn outside of the laser?	Draw a line from one corner to the other, both ways and see if they are in bounds.	They cannot go outside the laser boundary.	 Video Timestamp: 1:13	Pass – again the box isn't quite covering the GUI but importantly the Laser is protected.
5	Can lines be drawn in alternate colours?	Use the colour selector and draw many lines in different colours.	They show up as intended. (The video shows less contrast that I would have liked but it works and you can see it in the video if you look closely)	 Video Timestamp: 1:40	Pass – Don't my artistic skills look great :)
6	Can the GUI be resized and image scaled correctly?	Resize the window and see if the lines scale properly	They resize properly.	They resize properly: 1:54	Pass

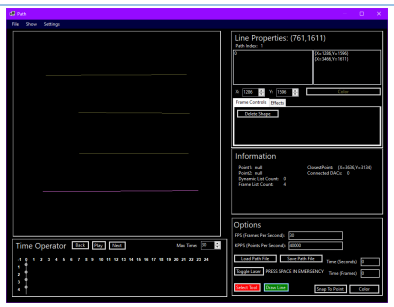
GUI – testing invalid options, such as snap to point or delete line when there's no point or line to delete.					
Test No.	**Test Intention**	**Test Data**	**Expected Output**	**Actual Output**	**Pass/Fail**
n/a	Make sure the snap to draw works with lines present	Draw a line and then on top of that try using the snap to point to draw other lines.	The first line works as usual then any other lines work as usual unless they are close to the end of another line in which case they join up with the line.	It did not work, for some reason all the points are going to 3977 & 4062. I will debug this and will test again. The bug was that it was converting points already stored in the Helios format. This double scaled them. 	Fail

n/a	Make sure the snap to select works	Draw a line then select other lines to draw off of.	The selected line gets dragged and when close enough perfectly aligns with the closest keypoint.	One second let me debug and retest. I left this table in to show that I have found most if not all my bugs through testing (either whilst coding or here).	Fail
n/a	**Make sure the select tool works with no lines**	Drag the select tool on an empty frame	Ideally nothing happens	The program crashed	Fail

Tests continue below as the above test failed.					
Test No.	**Test Intention**	**Test Data**	**Expected Output**	**Actual Output**	**Pass/Fail**
7	Make sure the snap to draw works with lines present	Draw a line and then on top of that try using the snap to point to draw other lines.	The first line works as usual than any other lines work as usual unless they are close to the end of another line in which case they will join up with the line.	It works now! Assuming the lines are within the pre-set radius the points will indeed "snap" into place. Video timestamp: 2:14	Pass
8	Make sure the snap to select works	Draw a line then select other lines to draw off of.	The selected line gets dragged and when close enough perfectly aligns with the closest key point.	This also works. Video Timestamp: 2:53	Pass
9	Make sure they can draw with snap to point and no lines present.	Draw a line in an empty canvas and the snap to point.	The line draws like normal with no errors.	 Video Timestamp: 3:00	Pass.
10	Make sure they can select without snap to point and no lines present.	Use the select tool on a blank canvas with	Nothing happens	Video Timestamp: **5:36**	Pass

	(should check reliability of the select tool)	the snap to point.			
11	Test if the delete shape functionality works.	Draw a couple of lines and then try to delete one of them.	The line deletes immediately (and then once the mouse hovers over the graphics panel the line disappears).	Video Timestamp: 3:08	Pass – code works
12	Test if the delete shape button can be clicked without crashing the code. (If there's nothing to delete).	Click the button when no shapes are there. Then add in a shape and click the delete button twice.	Nothing happens ideally, then we are still able to add a line, then the line deletes, and then nothing happens the second time	No crashes Video Timestamp: 3:29 (I said "that's always good when that works in the video referring to any sort of invalid data, my code for this is stable and anything that could go wrong did in the video)	Pass.

Testing the line properties panel					
Test No.	**Test Intention**	**Test Data**	**Expected Output**	**Actual Output**	**Pass/Fail**
13	Test to see if we can change the points in X & Y	Change the x point to another value and if that works try the y value.	They must change in the GUI as well (we can assume any GUI changes change on the lasers).	OMG it works as expected – for some reason I was convinced it would fail. Timestamp: 3:33 (In the testing video I misread my table after this but am working on getting glasses, usually I guess what I'm reading which isn't great)	Pass
14	Test if you can select and change the other points location via the list.	Test if you can select another line and edit this. We select the point using the list section.	The line should adjust to the points selected.	Worked as expected. Timestamp: 4:30	Pass
15	Test if you can change the	Change the colours of all the lines to	They change (but for some reason the	Worked as expected.	Pass

	lines colour retrospectively.	white / a different colour.	feedback isn't instant. You have to move the mouse over the graphics panel before it updates.)	 Timestamp: 5:05	
16	Test no errors arise if anything is modified after deletion.	Draw a line, delete it & try changing properties.	Nothing happens.	Worked as expected. Timestamp: 5:48	Pass
18	Make sure people can change selected points by selecting either end of a line.	Click either end of a line and make sure the line point properties change appropriately.	The line properties change appropriately.	Timestamp: 6:40 & 3:45 – I didn't want to cut the testing video as in my mind transparency is better than a polished testing video, the system works really well and the end of the video hopefully shows it off.	Fail – I accidentally included this feature, the feature was removed due to it not helping any objectives and causing many bugs.
19	Make sure you can jump to different times via the line properties panel.	Click different frames to make sure it switches to the correct place.	The time changes to the points in time where the selected keyframes are.	Timestamp: 6:58	Pass

Testing the time operator panel					
Test No.	**Test Intention**	**Test Data**	**Expected Output**	**Actual Output**	**Pass/Fail**
20	Make sure you can change the time properly	Click at several points on the timeline.	The timeline changes to that time (indicated by a vertical line).	This works unless you click more than about 4 times per second in which case the program crashes due to a stack overflow. Timestamp: 7:58	Pass
21	Make sure the timeline can	Click on frames in	The time should change to exactly the frame and the	It works as expected. Timestamp: 8:06	Pass

	select specific frames	the timeline	selected shape should be the shape you clicked on		
22	Make sure the timeline plays when play is clicked.	Hit play	The timeline should play and then loop past max time	It works as expected. Timestamp: 8:24	Pass
23	Make sure you can't set the time to negative times	Try dragging the timeline to the left.	The line cannot go left of zero	It works as expected. Timestamp: 8:39	Pass

Testing the file import features:					
Test No.	**Test Intention**	**Test Data**	**Expected Output**	**Actual Output**	**Pass/Fail**
24	Make sure you can save files	Save a file with multiple lines, frames, and colours.	The file is stored in my computer in human readable JSON format	It works as expected most of the time. This is something to bring up in the evaluation Timestamp: 9:08	Fail – unreliable and haven't been able to successfully replicate off testing
25	Make sure you can open the file properly	Open the same file without modifying the JSON. Make sure you have changed the canvas so you can see it change back.	The file should be the same one you just saved with no corrupt file warnings.	It works as expected. Timestamp: 10:37	Pass
26	Make sure the system can detect externally changed files and warn the user	Modify the JSON using a text editor. Then open the file.	The file should open but with a warning.	It caught the file. Timestamp: 11:43	Pass
27	Make sure the system can handle	Open up a random word document	The software should show an error.	It worked as expected. Timestamp: 12:25	Pass

	incorrect files

Testing the Laser Features					
Test No.	**Test Intention**	**Test Data**	**Expected Output**	**Actual Output**	**Pass/Fail**
28	Make sure the computer can connect to the dac	Plug in your dac to the computer and click connect to dac under settings, laser settings.	The connected dac's goes to 1 in the informatio n panel	It works as expected	Pass
29	Make sure paths can be generated	Draw in shapes and click the show path value.	As we draw out the lines the system should show the traversal in between.	It works as expected Timestamp: 13:18	Pass
30	Make sure points can be generated efficiently	Draw in shapes and click the show point value Set the velocity high and acceleration low to make it easier to see.	As the lines are drawn the system should generate points with appropriat ely varying distance.	It works as expected. Timestamp: 13:42	Pass
31	Make sure the laser can project something By doing this we are simultaneously testing various sub components such as: - Path traversal, finding a way to traversal the whole system - Dot generation through acceleration - Shape properties being read properly.	MAKE SURE YOU'RE SAFE NOW. Load any project with a line (not something tiny or the lasers safety might kick in (because a beam can set fire to things)) & click toggle laser.	Something is projected	It projects as intended with a slight minor spoken about in the improveme nts section of the evaluation. Timestamp: 14:50	Pass
32	Make sure the lines can be in all colours and angles.	Try a nice animated	What the GUI shows	What the GUI shows	Pass

		frame with many colours and angles and speeds.		and its amazing Timestamp: 16:36	
33	Quality assessments below	**Quality assessments below**	**Quality assessments below**	**Quality assessments below**	**Quality assessments below**
34	See how much the laser can project smoothly	Keep adding in more lines until the laser starts noticeably flickering at night (This is when the FPS is below about 10)	Hopefully it works with about three lines	Wow it can handle about 6 or so very easily. 16:36	Pass
35	See if the lines have much trail (basically I'm testing how effective my settings are here)	Any frames	The lines end where they should and you can't see where the lasers travelled to next.	Only noticeable if you're being pedantic I'm really happy with it. 16:36	Pass

Final miscellaneous tests – not necessarily in order (sorry for any inconvenience)

Test No.	Test Intention	Test Data	Expected Output	Actual Output	Pass/Fail
36	Check if the system was less than a thousand pounds.	Work out how much the laser and dac cost.	The system costs hopefully less than a thousand	It was around £600 (paid for using scholarship money with a hard limit)	Pass
37	Ability to turn the laser on and off at any time	Click the toggle laser button (with some frame content) and make sure the laser toggles appropriately	The laser should turn on and off as the toggle laser button is clicked	It turned on and off as planned. Timestamp On: 16:54 (I was so excited I forgot to film it turning off but It does do this)	Pass
38	Make sure software is dark with good contrast	Run the software	Throughout using make sure the system is adequately dark	It was dark. This can be seen throughout all screenshots and videos	Pass

	and black backgroun ds				
39	Ensure no paid for code is required	Run the software	Make sure on running the code doesn't check for any other code or require any further fees	The system doesn't require any external paid for software (checked manually as well) Timestamp: 1:55	Pass
40	Ensure the graphics are kept mostly to a single window	Use the software	Throughout using make sure all projections can be done on a single screen	All the projections made as well as testing were done on a single screen. Seen throughout testing and screenshots	Pass
41	Ensure everything required could be done via a GUI	Use the software	Make sure no command prompts or terminals are required for any features	All functionalities could be done via the GUI, this can be seen throughout my testing video.	Pass

Whilst the combinations aren't exhaustive, it was a thorough and systematic approach. Whilst there is a possibility there could be a bug/s remaining I am confident that the majority of testing was robust to ensure an effective outcome.

As I was testing, the objectives were numbered with each test such that we can see how fulfilling my code was of the objectives, I will show the objectives here:

Safety

- Ability by the user to turn off the laser using the software at any time.
- In person warnings given to alert users how to treat these devices with appropriate caution.
    1. These must explain that usage can cause blindness if eyes are exposed to the laser radiation
    2. These must explain that usage can cause fire
    3. These must be shown to the user before they hit play
- Appropriate algorithms to prevent hardware damaging requests to the device. **Test: 3,4, 30**
    1. These must block the laser from leaving the range 0&4096 as either side further will damage it. **Test: 3, 4,**
    2. These must have a limiting maximum velocity that can be set in the code. **Test: 30**
    3. These must have a limiting maximum acceleration that can be set in the code. **Test: 30**
- Dark Mode so it can be used in the dark, and allow the user to see the surroundings without adjusting eyes to allow for situational awareness. To ensure brightness of the GUI isn't preventing users from being aware of anyone within the laser exposure zone.
    1. The software must have mainly black backgrounds

2. There must be contrast between the text and background so it remains legible / useable when it is dark.

- File verification **Test: 24,25,26,27 – note the bug is still safe as no laser code runs and that is the most important albeit the bug is annoying.**
    1. The software must be able to handle invalid files. **Test: 27**
    2. The software must have a warning if the JSON has been edited but can still be opened (so that any damaging requests are at the users responsibility). **Test: 26**

Affordability

- Laser hardware needs to <£1000
- Laser must be bright.
    1. Beam must be visible at night with smoke. **Test: lines visible in tests 31-35**
- Efficient path generation algorithms to prevent the need for more unnecessarily costly hardware. **Test: 29,30,34**
    1. The algorithms must be able to run more than 1 frame per second. **Test: 34,**
    2. The algorithms must be able to distribute spacing based on detail. **Test: 29,30**
- No paid for code required.
    1. The code file must be able to run without paying for any other software (drivers auto install and are free).
- Works with a Helios Dac that can project 30000 kps. **Test: 28**
    1. The dac should be able to plug into a computer and connect like any peripheral would. **Test: 28**

Useability

- Intuitive GUI to allow any clients with IT skills to generate images on their own.
    1. Intuitive meaning that it doesn't require any previous training to use.
    2. The end user should ideally be able to understand the GUI when shown to them.
    3. Single graphics layout so there's no hidden functionality.
    4. Lines drawn from the first click. **Test 1,**
- Responsive software, that runs "out the box". **Test: 31**
    1. The laser can run as soon as you click toggle laser, no setup required (other than setting up the hardware physically). **Test: 31**
- No console usage, all in a display.
    1. The system should be able to run purely via a graphics based interface.
- GUI that can handle variable amounts of detail. **Test: 1,7, 13, 14, 15,**
    1. The system should be able to draw a simple frame as well as a frame with large amounts of shapes and changes over time. **Test: 1, 13, 14, 15**
    2. The system should have guidance (i.e. snap into place) **Test: 7, 8**
- GUI that can modify lines graphically post drawing. **Test: 2, 8~~, 18~~**
    1. The frame should be changeable without deleting shapes.
- GUI that can allow for mistake correction. **Test: 11,**
    1. There should be the option to delete specific shapes if they weren't necessary etc.
- A reliable GUI **Test: 1-6,**
    1. Consistent output given input **Test: 6**
    2. Doesn't crash if invalid data is inputted **Test: 9, 10, 12, 17, 23**

Customisability

- Choice of colours, minimum of 6 colours. **Test: 5, 16**
    1. The shapes should be able to have many different colours, ideally at different times.
- Ability to project dynamic animations. **Test: 19,20,21,22,23**
    1. The system should be able to loop a changing projection animation that isn't being modified by a user at the time. **Test: 19,20,21,22,23**
- Variable animation times. **Test 22 (shown during play as one animation takes two seconds the other a fraction of a second)**
    1. The length of the transitions must be variable, i.e. one be fast one be slow. **Test 22**
- Variable animation positions. **Test: 19,22**
    1. The lines should be able to change positions during animations. **Test: 19**

Optional extra features

- The ability to draw more than straight lines (i.e. Bezier curves, drawn lines).
- The ability to modify the laser settings & timeline settings without editing the code.
    1. This would be done via a settings page.
    2. Doesn't crash if invalid data is inputted **Test: 9, 10, 12, 17, 23**

4.3 Testing video

Due to the interlinked nature of my code, it would require an unreasonable investment of time to change all the properties from all ways of selection. The code is surprisingly durable from the tests, and you can find my testing video here: https://youtu.be/N66aweARqmc

https://youtu.be/N66aweARqmc

https://youtu.be/N66aweARqmc

Figure 26: A working image of my laser - it's very impressive and even more impressive in person

5 Evaluation

5.1 Reflections

I cannot believe I have made and own a working Laser System. It's so cool! The system looks visually very impressive and can be used at parties. The end user said it was so much easier to understand, being able to make animations with little to no help and being able to retrospectively edit lines and change colours!

We know the biggest room in the world is the room for improvement and the amazing thing about this system is that we can continue to iterate and improve it almost indefinitely. So, I have many improvements later. Nonetheless this has been an amazing project.

5.2 Objectives Evaluations

Safety Objectives			
Met / Not met	**Objective**	**Measurement**	**Evaluation and critique**
✓	Ability to turn off the laser via software at any time		
✓	In person warnings given to alert users how to treat these devices with appropriate caution.	1. These must explain that usage can cause blindness if eyes are exposed to the laser radiation 2. These must explain that usage can cause fire 3. These must be shown to the user before they hit play	Whilst I could have added them into the code, as I was testing with my end user they mentioned that warnings kept making them think the system was broken, reducing the satisfaction of the system whilst ignoring the warnings anyway. My objective works well to have the user warned by a human being upon delivery and given a physical flyer with the summary of safety.
✓	Appropriate algorithms to prevent hardware damaging requests to the device	1. These must block the laser from leaving the range 0&4096 as either side further will damage it. 2. These must have a limiting maximum velocity that can be set in the code.	These algorithms worked great. It would be good to add in a settings page so the limits can be more specific and therefore higher and safer.

		3. These must have a limiting maximum acceleration that can be set in the code.	
✓	Dark Mode so it can be used and allow the user to see the surroundings without adjusting eyes to allow for situational awareness. To ensure brightness of the GUI isn't preventing users from seeing people within the Laser exposure zone	1. The software must have mainly black backgrounds 2. There must be contrast between the text and background so it remains legible / useable when it is dark.	The dark mode worked very effectively, the slight challenge we had was colour accuracy in the GUI, this was because the lines were very thin they didn't always show as expected. This could perhaps be improved by making the line thickness adjustable.
✓	File verification	1. The software must be able to handle invalid files. 2. The software must have a warning if the JSON has been edited but can still be opened (so that any damaging requests are at the user's responsibility).	Worked very well however sometimes it loaded files that would crash, this was still safe however as the laser wasn't ever damageable without warnings.

Affordability Objectives			
Met / Not met	**Objective**	**Measurement**	**Evaluation and critique**
✓	Hardware must be under £1000		This was well met as the system was probably closer to £600, with extra bulk or making my own drivers, the price could go down further.
✓	Laser brightness	1. Beam must be visible at night with smoke	The laser was definitely bright enough to be viewed, brighter lasers can be bought for brighter areas.
✓	Efficient path generation algorithms to prevent the need for more unnecessarily costly hardware.	1. The algorithms must be able to run more than 1 frame per second. 2. The algorithms must be able to distribute spacing based on detail.	These objectives were very effectively met. I did notice a cluster of points in the centre of lines, whilst these guarantee the mirrors don't move too fast, they aren't as efficient as they could be.
✓	No "paid for" code required	1. The code file must be able to run without paying for any other	This is the case!

		software (drivers auto install and are free).	
✓	Works with a Helios Dac that can project 30000 kps	The dac should be able to plug into a computer and connect like any peripheral would.	This was automatic

Useability Objectives			
Met / Not met	**Objective**	**Measurement**	**Evaluation and critique**
✓	Intuitive GUI to allow users with IT skills to generate images on their own	1. Intuitive meaning that it doesn't require any previous training to use. 2. The end user should ideally be able to understand the GUI when shown to them. 3. Single graphics layout so there's no hidden functionality 4. Lines drawn from the first click.	The GUI could be more responsive, IE instead of all points being red being the colour of the shape at the time etc. Or the border of the graphics input being the colour of the next line.
✓	Responsive software, that runs "out the box"	The laser can run as soon as you click toggle laser, no setup required (other than setting up the hardware physically).	This works!
✓	No console usage, all in a display	The system should be able to run purely via a graphics based interface.	
✓	GUI that can handle variable amounts of detail	1. The system should be able to draw a simple frame as well as a frame with large amounts of shapes and changes over time. 2. The system should have guidance	
✓	GUI that can modify lines graphically post drawing	The frame should be changeable without deleting shapes.	It would be good to be able to modify the selected line property using the GUI and vice versa

✓	GUI that can allow for mistake correction	There should be the option to delete specific shapes if they weren't necessary etc.	You can delete many shapes but only one at once, I also want to add a delete frame option later.
✓	A reliable GUI	1. Consistent output given input 2. Doesn't crash if invalid data is inputted	This works for the most part as the only crashes came on opening specific files.

Customizability Objectives			
Met / Not met	**Objective**	**Measurement**	**Evaluation and critique**
✓	Choice of colours, up to 6 or more colours	1. The shapes should be able to have many different colours, ideally at different times.	This worked very well. We have already mentioned that contrast wasn't great but otherwise it was brilliant.
✓	Ability to project dynamic animations	The system should be able to loop a changing projection animation that isn't being modified by a user at the time.	This was really effective, it would be nice to make non linear animations at some point.
✓	Variable animation times	The length of the transitions must be variable, ie one be fast one be slow.	Using the timeline system meant this worked very effectively, maybe a future feature is changing the lengths by dragging keyframes.
✓	Variable animation positions	The lines should be able to change positions during animations.	All positions work!

Optional Extra Features		
Met / Not met	**Objective**	**Evaluation and critique**
X	The ability to draw more than straight lines (i.e. Bezier curves, drawn lines)	I needed to work with our time constraint for this project and had the main struggle of not understanding Bezier curves, with time I will implement this however I wanted at least an MVP.
X	The ability to modify the laser settings & timeline settings without editing the code.	This wasn't possible as easily as I thought as I struggled to pass the settings between windows, with a day this will be possible.

5.3 User's opinions

Likes:

- Clear large area that I could create my images on
- One cursor on the screen to ensure accuracy of where I was drawing
- I liked being able to select the lines that I wanted to edit, delete or update in an intuitive way
- I liked working in dark mode for programming outside
- The colour field was straightforward and I liked being able to click on it and access the colour options
- I liked having a wide variety of colours to incorporate into my designs and not having to adjust RGB levels to select new colours
- I liked being able to select different colours for each part of my image, and seeing this happen in real time on the display

Dislikes:

- I was not sure what the fields for the line properties were, and wondered if I should use these to adjust the size (like I can do with image size on a computer)
- I wasn't clear on how to use the Time Operator field without instruction, once it was explained, that each row was an independent animation and that I moved the icon to select the speed, I found this very simple
- I wanted to be able to draw circles and was limited to lines

Suggested Improvements by the user:

- Could a user help option be offered in each section, for example a 'question mark' in the top right of each field that the user can select to explain the options available
- Could the fields for user use be a different colour, than the fields which were giving system detail (does the user need to see the frame control details?)
- Could I import gifs and have these moving in the display?
- Can the user select light or dark mode depending on whether they are programming inside during the day, or using outside at night time?

5.3.1 My comments on the discussion

Essentially key feedback from the user is how to further make the UI more understandable, this may be through better icons or better labels and help guides. However, once taught the ways, the user did like the actual methods to modify the systems.

The other main dislike was the limited options for now. I want to find ways to increase options without overwhelming the user, I would really love to learn how to automatically generate animations or create very impressive animations easily.

The user seems to agree with me on the improvement of more colour, and the help question marks are a great idea. Gifs won't be possible due to their raster nature but the idea of animated shapes or frames might be really nice to import.

The user also decided that they wanted a dark and light mode so they can program indoors (isn't it lovely when people change their mind on what they want?) Whilst time consuming, this is easily achievable, all that would be required is individually changing each property in a method.

5.4 Improvements:

- One interesting problem was that when each frame was generated, the frame would start at the position of point 0 of line 0, however this meant that the start point of the next frame wasn't always where the next frame ended, on a slight occasion this meant the laser would be slightly unpredictable for a split second. This wasn't dangerous however it isn't ideal and isn't good for user confidence. The solution in my head is to send a frame to transfer the point of the laser from where it is to where it should be.
- The selected line feedback could be a bit more obvious
- The options and information panels in the GUI are a bit debugging centric. I feel like a bit more abstraction would be good there.

Would have liked to improve the graph traversal algorithms such that the first line is a bigger part of the loop.

To avoid this problem:

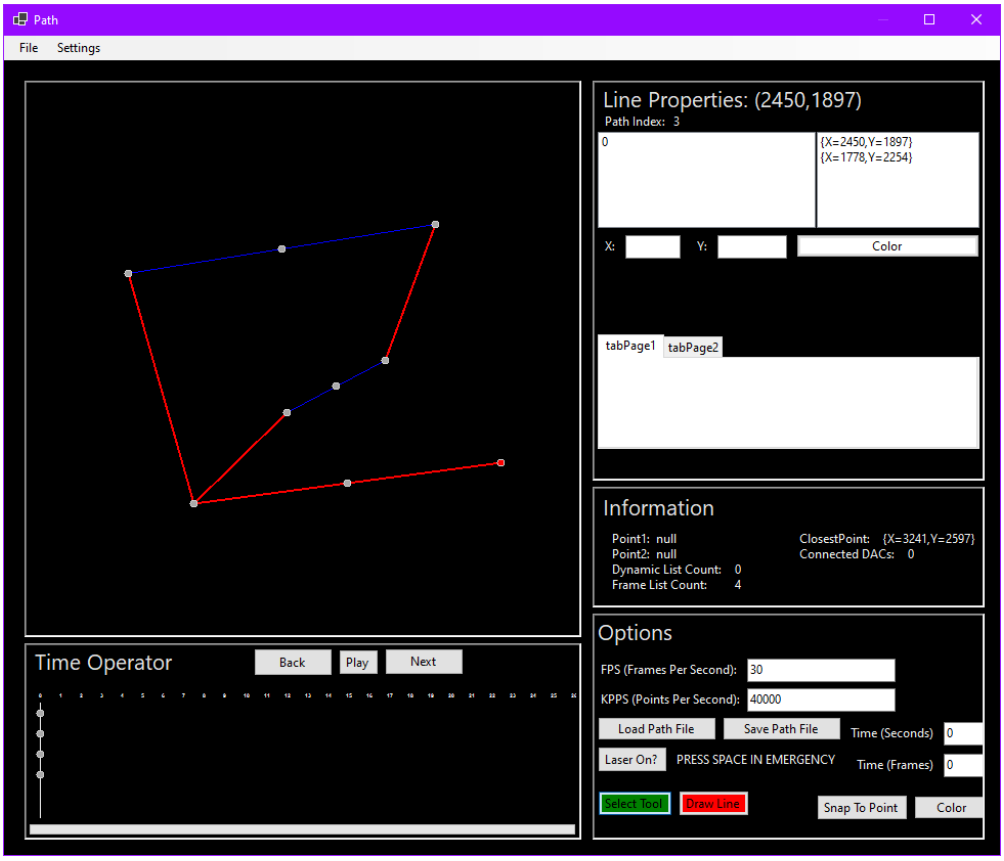


Figure 27: A demonstration of an inefficiency of the path traversal algorithm I use

5.5 What else would I improve with time?

- I really like the animations, right now everything is manual however it would be nice to allow symmetric or repeated options, this would allow more complex effects.
- I would like to add effects such as strobe and multiple colours per line and patterns on lines.
- The option to add rotation effects in better than the alternative solutions did.
- I would like to implement Bezier curves, I feel like this is much more possible than I first thought.
- I would like to make the UI a bit more information efficient, keyframes on the timeline could be the colour of their shape and the lines colours weren't always obvious during testing, maybe make the outline of the graphics preview the colour of the selected shape.
- I would like a settings window, as briefly discussed I couldn't get the two windows to talk but I probably can with enough time to figure it out.
- I would like able to add in freehand drawing and circles.
- I would like to add in non linear animations, these look so natural when implemented properly, like in after effects.
- I would like to make the path traversal more efficient, potentially considering not just the closest point in distance but the closest point in time.
- I would like to add in variable acceleration when travelling between points as this is more efficient than a straight line in most cases.
- I would like to add in some sort of AI mode to convert music into displays
- I would like to add in more features to select a specific projection area (so you cannot accidentally project in the wrong place.

5.6 Long term goals

- In the future I really want real time inputs through something like an iPad, (I feel like we have accomplished something sort of like this with audio and Djying but not lighting).
- I want to also develop the ability to control multiple lasers and other devices like moving heads and LED strips which are now often added to stages.
- Due to the large number of lights, I would like to find a way to intuitively control many at once, however here I think the problem is much wider.
- Maybe for my university project, I would be enthusiastic to see if swarm technologies can merge with a system like this to exponentially improve effects (ideally safely though).
- I also want to experiment with 3D tracking, the software might not be suitable for this however projection mapping is.

5.7 Final comments

This project has taught me a myriad of practical skills and the outcome is very flashy and entertaining, I am immensely proud of my work and cannot wait to see where I take it next.

I would really like to thank my teachers for teaching me the skills needed to work on projects like this.

6 Appendix

Appendix 1 – Research correspondence

Hi Gitle,

I am a student in education (17 years old) and am studying how to program laser systems, as an alternative to your software (don't worry mine isn't planning to be commercial and isn't going to compete with yours at all).

It would be really helpful for my project if I could have any draft plans of the LaserShowGen software (high level diagrams, flow charts, notes on the back of an envelope etc) because part of my qualification requires thorough analysis of software.

I have really appreciated the DAC being open source and so easy to learn for a beginner and have been inspired by the software and am saving up for the full version as it would be really fun to learn about (and to enjoy cool laser shows)!

I cannot express how much help it would be if you have any plans of your software that could be sent, this wouldn't be to copy but to demonstrate my research into alternative solutions already present. The specification of my project assessment is here: AQA | Computer Science | Subject content – A-level | Non-exam assessment - the computing practical project

Thanks for your time & look forward to hearing from you soon,

Leo

Reply:

Hi Leo,

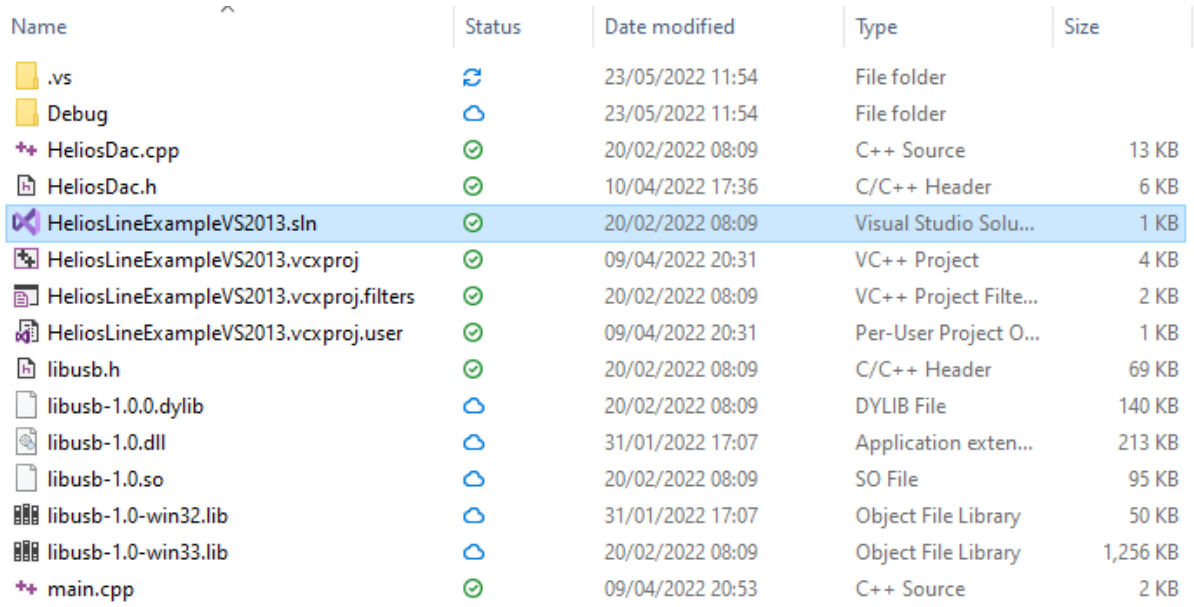
Thanks for the interest in the software, but to be honest I'm not sure it's something you'd want to use as a reference, I started building this program almost a decade ago when I was far less experienced in programming and design, and thus it is somewhat of a mess. I would structure the program quite differently if I were to start over now. But if you are still interested, a very rough overview is that there are controller objects for each of the main views (editor, timeline, live) which keeps track of variables like current frame number, and data structures with all the frame data, and settings. All the UI elements are separate objects that can write or read to these controller objects. And there are many times of helper scripts with functions that is reused several times throughout the software like loading and writing ILDA and project files, assembling laser frames including optimizing points, etc. The whole thing is built in a proprietary commercial engine called GameMaker.

Unfortunately I don't have any visual aids like diagrams and flowcharts and documenting the code to a more detailed level than the above would be too big of a task. But as you may know the code is available here: https://github.com/Grix/ildagen

Mvh / Regards,
Gitle Mikkelsen, Bitlasers

Appendix 2 – Prototype 1 code

File overview:



Name	Status	Date modified	Type	Size
.vs		23/05/2022 11:54	File folder	
Debug		23/05/2022 11:54	File folder	
HeliosDac.cpp		20/02/2022 08:09	C++ Source	13 KB
HeliosDac.h		10/04/2022 17:36	C/C++ Header	6 KB
HeliosLineExampleVS2013.sln		20/02/2022 08:09	Visual Studio Solu...	1 KB
HeliosLineExampleVS2013.vcxproj		09/04/2022 20:31	VC++ Project	4 KB
HeliosLineExampleVS2013.vcxproj.filters		20/02/2022 08:09	VC++ Project Filte...	2 KB
HeliosLineExampleVS2013.vcxproj.user		09/04/2022 20:31	Per-User Project O...	1 KB
libusb.h		20/02/2022 08:09	C/C++ Header	69 KB
libusb-1.0.0.dylib		20/02/2022 08:09	DYLIB File	140 KB
libusb-1.0.dll		31/01/2022 17:07	Application exten...	213 KB
libusb-1.0.so		20/02/2022 08:09	SO File	95 KB
libusb-1.0-win32.lib		31/01/2022 17:07	Object File Library	50 KB
libusb-1.0-win33.lib		20/02/2022 08:09	Object File Library	1,256 KB
main.cpp		09/04/2022 20:53	C++ Source	2 KB

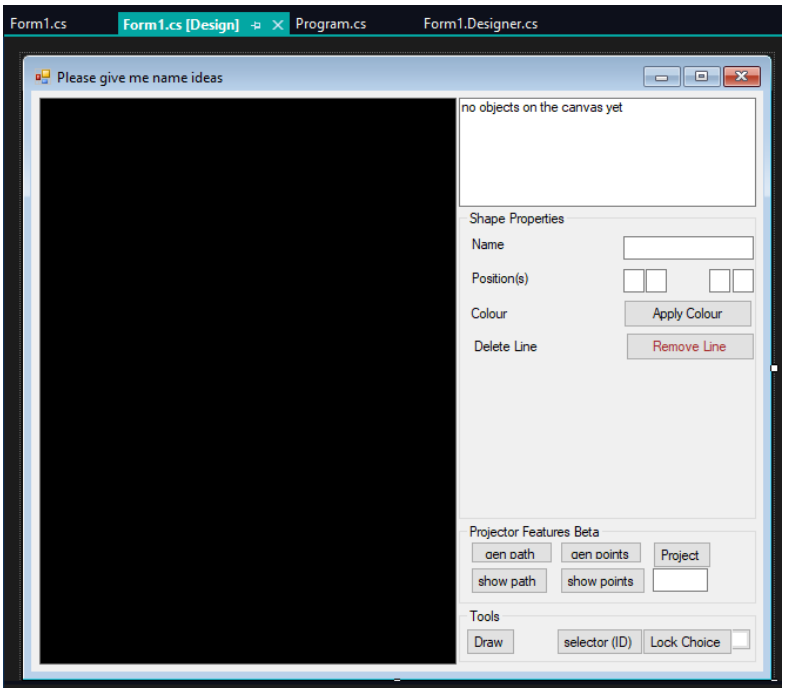
main.cpp code, example code by Gitle that took ages to get working (I also modified this to take drawing inputs) as usb files were outdated NOT PURELY MY OWN WORK:

```
//Example program scanning a line from top to bottom on the Helios

#include "HeliosDac.h"

int main(void)
{
	//make frames
	HeliosPoint frame[30][1000];
	int x = 0;
	int y = 0;
	for (int i = 0; i < 30; i++)
	{
		y = i * 0xFFF / 30;
		for (int j = 0; j < 1000; j++)
		{
			if (j < 500)
				x = j * 0xFFF / 500;
			else
				x = 0xFFF - ((j - 500) * 0xFFF / 500);

			frame[i][j].x = x;
			frame[i][j].y = 0xFFF;
			frame[i][j].r = 0x00;
			frame[i][j].g = 0x3F;
			frame[i][j].b = 0x00;
			frame[i][j].i = 0xFF;
		}
	}
```

```
	//connect to DACs and output frames
	HeliosDac helios;
	int numDevs = helios.OpenDevices();

	int i = 0;
	while (1)
	{
		i++;
		//if (i > 150) //cancel after 5 cycles, 30 frames each
			//break;

		for (int j = 0; j < numDevs; j++)
		{
			//wait for ready status
			for (unsigned int k = 0; k < 512; k++)
			{
				if (helios.GetStatus(j) == 1)
					break;
			}
			helios.WriteFrame(j, 40000, HELIOS_FLAGS_DEFAULT, &frame[i % 30][0], 1000);
//send the next frame
		}
	}

	//freeing connection
	helios.CloseDevices();
}
```

Appendix 3 - Prototype 2 – First version in c# with a GUI

I refer to prototype 2 as prototype 1 in my files, apologies for the confusion.



p applications › Creating the interactive gui › Open day version

Name	Status	Date modified	Type	Size
.git	✓	12/11/2022 22:46	File folder	
.vs	✓	30/01/2023 19:40	File folder	
packages	✓	05/10/2022 19:00	File folder	
Prototype 1	☁	14/11/2022 20:05	File folder	
.gitattributes	✓	10/08/2022 10:19	Text Document	3 KB
.gitignore	✓	10/08/2022 10:19	Text Document	7 KB
Prototype 1.sln	✓	08/11/2022 09:11	Visual Studio Solu...	2 KB

›p applications › Creating the interactive gui › Open day version › Prototype 1 ›

Name	Status	Date modified	Type	Size
bin	✓	05/10/2022 19:00	File folder	
obj	☁	05/10/2022 19:00	File folder	
Properties	✓	08/11/2022 09:25	File folder	
Resources	✓	10/11/2022 10:42	File folder	
App.config	✓	09/08/2022 15:42	XML Configuratio...	1 KB
Form1.cs	✓	14/11/2022 20:05	C# Source File	20 KB
Form1.Designer.cs	✓	08/11/2022 09:11	C# Source File	24 KB
Form1.resx	✓	08/11/2022 09:11	Microsoft .NET M...	7 KB
HeliosLaserDAC.dll	✓	20/05/2022 17:41	Application exten...	5 KB
packages.config	✓	09/08/2022 15:42	XML Configuratio...	1 KB
Program.cs	✓	09/08/2022 13:52	C# Source File	1 KB
Prototype 1.csproj	✓	08/11/2022 09:11	C# Project file	5 KB

Main Prototype code:

```
using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using HeliosLaserDAC;

using System.Threading;

using System.Windows.Forms;


namespace Prototype_1

{


   public partial class Form1 : Form

   {

      public Form1()

      {

         InitializeComponent();

         this.colorPicker.Color = Color.White;

         this.toggleDraw.Checked = true;

         this.pointsCount.Text = "2000";

         Console.WriteLine(helios.openDevices());

         listBoxFiller = this.listBox1.Items[0];     //Gather the filler (non line item before lines are
added) to remove once not needed


      }
```

```
    public HeliosDac helios = new HeliosDac();

    public HeliosPoint[] heliosLaserPoints;

    public List<Line> lines = new List<Line>();     //Create the list of all the lines (in the final piece this will be all shapes? maybe arranged differently, I think this way works best)0+

    public bool mouseDown = false;                  //Gathers if the mouse is down, possibly an inbuilt feature for this

    public Point downLocation = new Point(-1, 0);   //This is to store where the last point was that the user clicked down

    public Point upLocation = new Point(-1, 0);     //Use your brain dummy (Sorry for the mean language)

    object listBoxFiller;                           //Only used cuz I cant think of an easier way

    List<Line> path = new List<Line>();             //Create the path variable for the projector

    List<LaserPoint> laserPoints = new List<LaserPoint>();

    public bool debugViewer = false;

    private void pictureBox1_MouseDown(object sender, MouseEventArgs e)

    {

      if (!mouseDown)                               //Dont really know why I have this if statement, I guess I dont trust forms

      {

        downLocation = new Point(e.X, e.Y);     //Updating the down location

        mouseDown = true;                       //The mouse is now down

      }

    }


    private void pictureBox1_MouseMove(object sender, MouseEventArgs e)

    {

      upLocation = new Point(e.X, e.Y);         //Ok this was a really cheap work around

      this.pictureBox1.Invalidate();            //Ask the graphics to update themselves when they next ready

    }


    private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
```

```
    {
        if (toggleDraw.Checked)
        {
            lines.Add(new Line(downLocation, new Point(e.X, e.Y), this.colorPicker.Color)); //Create
new line and add it too list
            if (listBoxFiller == this.listBox1.Items[0])//Check if the default text is still in the list
            {
                this.listBox1.Items.Remove(listBoxFiller);      //Removes the default text if it is indeed
present
            }
            this.listBox1.Items.Add(lines.Last<Line>().Name);   //Adds the new line to the list
        }
        mouseDown = false;                      //The mouse is now up
        this.pictureBox1.Invalidate();          //Ask the graphics to refresh themselves
    }

    private void pictureBox1_Paint(object sender, PaintEventArgs e)
    {
        e.Graphics.Clear(Color.White);          //Reset and redraw the whole thing (im not resource
efficient).
        foreach (Line line in lines)            //Draws all the stored lines
        {
            e.Graphics.DrawLine(new Pen(line.Color1, 1), line.Point1, line.Point2);
        }
        if (this.toggleShowPath.Checked)
        {
            Brush brush1 = new SolidBrush(Color.Red);
            Brush brush2 = new SolidBrush(Color.Purple);
            foreach (Line line in path)             //Draws all the stored lines
            {
                if (line.Color1 != Color.Transparent)
```

```
            {
                e.Graphics.DrawLine(new Pen(line.Color1, 1), line.Point1, line.Point2);
            }
            else
            {
                e.Graphics.DrawLine(new Pen(Color.Red, 1), line.Point1, line.Point2);
            }
            if (debugViewer)
            {
                e.Graphics.DrawString(path.IndexOf(line).ToString(), SystemFonts.DefaultFont, brush1,
new Point(line.Point1.X, line.Point1.Y + 25));
                e.Graphics.DrawString(path.IndexOf(line).ToString(), SystemFonts.DefaultFont, brush2,
new Point(line.Point2.X, line.Point2.Y - 15));
            }
        }
    }
    if (this.toggleShowPoints.Checked)
    {
        Brush brush = new SolidBrush(Color.Black);
        foreach (LaserPoint point in laserPoints)            //Draws all the stored lines
        {
            e.Graphics.DrawLine(new Pen(Color.Black, 2), new Point(point.Location.X - 1,
point.Location.Y - 1), new Point(point.Location.X + 1, point.Location.Y + 1));
            e.Graphics.DrawLine(new Pen(Color.Black, 2), new Point(point.Location.X - 1,
point.Location.Y + 1), new Point(point.Location.X + 1, point.Location.Y - 1));
            e.Graphics.DrawString(laserPoints.IndexOf(point).ToString(), SystemFonts.DefaultFont,
brush, new Point(point.Location.X, point.Location.Y + 5));
        }
    }
    if (mouseDown && toggleDraw.Checked)      //Draws a bonus preview line
    {
        e.Graphics.DrawLine(new Pen(this.colorPicker.Color, 1), downLocation, upLocation);
```

```
    }
  }

  private void listBox1_SelectedIndexChanged(object sender, EventArgs e)  //Update properties box to that of the selected line
  {
    if (this.listBox1.Items.Count != 0)               //Work around for list glitch
    {
      if (listBoxFiller != this.listBox1.Items[0])    //Check if the default text is still in the list
      {
        this.nameProperty.Text = lines[this.listBox1.SelectedIndex].Name;
        this.pos1xProperty.Text = lines[this.listBox1.SelectedIndex].Point1.X.ToString();
        this.pos1yProperty.Text = lines[this.listBox1.SelectedIndex].Point1.Y.ToString();
        this.pos2xProperty.Text = lines[this.listBox1.SelectedIndex].Point2.X.ToString();
        this.pos2yProperty.Text = lines[this.listBox1.SelectedIndex].Point2.Y.ToString();
        this.applyColorButton.BackColor = lines[this.listBox1.SelectedIndex].Color1;
      }
    }
  }

  private void textBox1_TextChanged(object sender, EventArgs e)
  {
    this.listBox1.SelectedValue = nameProperty.Text;
    lines[this.listBox1.SelectedIndex].Name = nameProperty.Text;
  }

  private void applyColor(object sender, EventArgs e)
  {
    if (this.listBox1.SelectedIndex == -1)
    {
      MessageBox.Show("No line to selected to change the color of.");
```

```
    }
    else if(listBoxFiller == this.listBox1.Items[0])
    {
        MessageBox.Show("No lines have been drawn to change the color of.");
    }
    else
    {
        lines[this.listBox1.SelectedIndex].Color1 = this.colorPicker.Color;
        this.applyColorButton.BackColor = lines[this.listBox1.SelectedIndex].Color1;
        pictureBox1.Invalidate();
    }
}

private void colorPickerButtonClicked(object sender, EventArgs e)
{
    colorPicker.ShowDialog();
    this.colorPickerButton.BackColor = this.colorPicker.Color;
}

private void toggleDraw_CheckedChanged(object sender, EventArgs e)
{
    if(!toggleDraw.Checked)
    {
        this.toggleSelector.Checked = true;
    }
}

private void genPath_Click(object sender, EventArgs e)
{
    path = new List<Line>();
```

```
        List<TargetPoint> targetPoints = new List<TargetPoint>();
        int lineIndex = 0;
        bool nextLineSideOne = true;
        TargetPoint firstPoint;
        TargetPoint secondPoint;
        foreach(Line line in lines)
        {
            targetPoints.Add(new TargetPoint(line.Point1, lines.IndexOf(line), true));
            targetPoints.Add(new TargetPoint(line.Point2, lines.IndexOf(line), false));
            Console.WriteLine("Current index of line: " + lines.IndexOf(line).ToString());
            Console.WriteLine("Length of target points: " + targetPoints.Count());
        }
        //path.Add(new Line(lines[0].Point1, lines[0].Point2, lines[0].Color1,false));
        TargetPoint targetPoint = targetPoints[0];
        while (true)
        {
            if (nextLineSideOne)
            {
                firstPoint = targetPoint;
                secondPoint = targetPoints[targetPoints.IndexOf(targetPoint) + 1];
            }
            else
            {
                firstPoint = targetPoint;
                secondPoint = targetPoints[targetPoints.IndexOf(targetPoint) - 1];
            }
            path.Add(new Line(firstPoint.Point1, secondPoint.Point1, lines[lineIndex].Color1,
!nextLineSideOne));
            targetPoints.Remove(firstPoint);
            targetPoints.Remove(secondPoint);
            Console.WriteLine(targetPoints.Count);
```

```
            if(targetPoints.Count == 0)
            {
                path.Add(new Line(secondPoint.Point1, lines[0].Point1, Color.Transparent, false));
                this.toggleShowPath.Checked = true;
                this.pictureBox1.Invalidate();
                break;
            }
            targetPoint = getClosestPoint(targetPoints, lines[lineIndex].Point2);
            lineIndex = targetPoint.LineIndex;
            nextLineSideOne = targetPoint.SideOneFirst;
            path.Add(new Line(secondPoint.Point1, targetPoint.Point1, Color.Transparent,
!nextLineSideOne));
        }
    }
    public TargetPoint getClosestPoint(List<TargetPoint> targetPoints, Point currentPoint)
    {
        TargetPoint closestPoint = targetPoints[0];
        double closestDistance = getDistanceBetweenPoints(targetPoints[0], currentPoint);
        foreach (TargetPoint targetPoint in targetPoints)
        {
            if (getDistanceBetweenPoints(targetPoint, currentPoint) < closestDistance)
            {
                closestPoint = targetPoint;
                closestDistance = getDistanceBetweenPoints(targetPoint, currentPoint);
            }
        }
        return closestPoint;
    }
    public double getDistanceBetweenPoints(TargetPoint targetPoint, Point currentPoint)
    {
        Point targetPointPoint = targetPoint.Point1;
```

```
        double hypotinuse = (double) Math.Pow((targetPointPoint.X - currentPoint.X), 2) +
Math.Pow((targetPointPoint.Y - currentPoint.Y), 2);
        return(Math.Sqrt(hypotinuse));
    }
    public double getDistanceBetweenPoints(Point targetPointPoint, Point currentPoint)
    {
        double hypotinuse = (double) ((targetPointPoint.X - currentPoint.X) * (targetPointPoint.X -
currentPoint.X)) + ((targetPointPoint.Y - currentPoint.Y)*(targetPointPoint.Y - currentPoint.Y));
        return (Math.Sqrt(hypotinuse));
    }

    private void genPoints_Click(object sender, EventArgs e)
    {
        double distance = 0;
        laserPoints.Clear();
        double pointCount = (double) Convert.ToInt32(this.pointsCount.Text);
        LaserPoint currentPoint = new LaserPoint(path[0].Point1, path[0].Color1,0);
        foreach(Line line in path)      //Calculating the overall distance to cover
        {
            distance += getDistanceBetweenPoints(line.Point1, line.Point2);
            Console.WriteLine("Distance: " + distance.ToString() + " Distance between points: " +
getDistanceBetweenPoints(line.Point1, line.Point2).ToString());
        }
        double spacing = distance / pointCount;
        double remaindingSpacing = spacing;
        Point corner = path[0].Point2;
        double xPortion;
        double yPortion;
        double distanceToCorner;
        currentPoint.LocationX = path[0].Point1.X;
        currentPoint.LocationY = path[0].Point1.Y;
```

```
        laserPoints.Add(new LaserPoint(path[0].Point1, path[0].Color1, currentPoint.LineIndex));
        List<HeliosPoint> heliosLaserPointsTemp = new List<HeliosPoint>();
        HeliosPoint pointTemp;
        while (true)
        {
           distanceToCorner = getDistanceBetweenPoints(currentPoint.Location,
path[currentPoint.LineIndex].Point2);
           if (distanceToCorner < remaindingSpacing)
           {
              remaindingSpacing = remaindingSpacing - distanceToCorner;      //Subtracting the line
travelled from remaining spacing
              currentPoint.LineIndex = currentPoint.LineIndex + 1;          //Moving onto the next line in
path
              if (currentPoint.LineIndex == path.Count)
              {
                 foreach(LaserPoint point in laserPoints)
                 {
                    pointTemp = new HeliosPoint();
                    pointTemp.r = (byte) ((point.Color1.R * point.Color1.A) / 0xFF);
                    pointTemp.g = (byte) ((point.Color1.G * point.Color1.A) / 0xFF);
                    pointTemp.b = (byte) ((point.Color1.B * point.Color1.A) / 0xFF);
                    pointTemp.i = (byte) (point.Color1.A);
                    pointTemp.x = (UInt16)(0xFFF - (point.Location.X * 0xFFF) / pictureBox1.Width);
                    pointTemp.y = (UInt16)(0xFFF - (point.Location.Y * 0xFFF )/ pictureBox1.Height);
                    heliosLaserPointsTemp.Add(pointTemp);
                 }
                 heliosLaserPoints = heliosLaserPointsTemp.ToArray();
                 this.toggleShowPoints.Checked = true;
                 this.pictureBox1.Invalidate();
                 break;
              }
```

```
                currentPoint.Color1 = path[currentPoint.LineIndex].Color1;      //Setting the new settings for currentPoint
                corner = path[currentPoint.LineIndex].Point2;
                currentPoint.Location = path[currentPoint.LineIndex].Point1;    //Setting the new location of currentPoint to the start of the next line
                currentPoint.LocationX = path[currentPoint.LineIndex].Point1.X;
                currentPoint.LocationY = path[currentPoint.LineIndex].Point1.Y;
            }
            else
            {
                xPortion = (remaindingSpacing / distanceToCorner) * (corner.X - currentPoint.Location.X);
                yPortion = (remaindingSpacing / distanceToCorner) * (corner.Y - currentPoint.Location.Y);
                currentPoint.LocationX = (currentPoint.LocationX + xPortion);
                currentPoint.LocationY = (currentPoint.LocationY + yPortion);
                //Creating the next point - only change is location as its still on the line.
                currentPoint.Location = new Point(Convert.ToInt32(currentPoint.LocationX), Convert.ToInt32(currentPoint.LocationY));
                remaindingSpacing = spacing;
                laserPoints.Add(new LaserPoint(currentPoint.Location, currentPoint.Color1, currentPoint.LineIndex));
                //Adding the current point
            }
        }
    }

    private void toggleShowPoints_CheckedChanged(object sender, EventArgs e)
    {
        this.pictureBox1.Invalidate();
    }

    private void pos1xProperty_TextChanged(object sender, EventArgs e)
    {
```

```
        lines[this.listBox1.SelectedIndex].Point1.X = Convert.ToInt32(this.pos1xProperty.Text);
    }

    private void button1_Click(object sender, EventArgs e)
    {
        try
        {
            lines.RemoveAt(this.listBox1.SelectedIndex);
            this.listBox1.Items.Remove(this.listBox1.SelectedIndex);
            this.pictureBox1.Invalidate();
        }
        catch
        {
            lines.Clear();
            this.listBox1.Items.Clear();
            this.pictureBox1.Invalidate();
        }
    }

    private void label4_Click(object sender, EventArgs e)
    {

    }

    private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
    {
        while (true)
        {
            if (this.toggleProject.Checked)
            {
```

```
            while (helios.getStatus(0) == 0)
            {
                Thread.Sleep(1);
            }
            helios.writeFrame(0, 40000, 0, heliosLaserPoints, heliosLaserPoints.Count());
        }
        else
        {
            Thread.Sleep(100);
            helios.closeDevices();
        }
    }
}

private void toggleProject_CheckedChanged(object sender, EventArgs e)
{
    if (!backgroundWorker1.IsBusy)
    {
        this.backgroundWorker1.RunWorkerAsync();
    }
}
}
public class Line
{
    public Line(Point point1, Point point2, Color color1, bool reverse = false)
    {
        if(!reverse)
        {
            Point1 = point1;
            Point2 = point2;
```

```
        }
        else
        {
            Point1 = point2;
            Point2 = point1;
        }
        Color1 = color1;
        Reverse = reverse;
        Name = "(" + point1.X + "," + point1.Y + ") -> (" + point2.X + "," + point2.Y + ")";

    }
    public Point Point1;
    public Point Point2;
    public String Name;
    public Color Color1;
    public bool Reverse;
}
public class TargetPoint
{
    public TargetPoint(Point point, int lineIndex, bool sideOneFirst)
    {
        Point1 = point;
        LineIndex = lineIndex;
        SideOneFirst = sideOneFirst;
    }
    public Point Point1;
    public int LineIndex;
    public bool SideOneFirst;
}
public class LaserPoint
```

```
    {
        public LaserPoint(Point location, Color color, int lineIndex)
        {
            Location = location;
            Color1 = color;
            LineIndex = lineIndex;
        }
        public Point Location;
        public Color Color1;
        public int LineIndex;
        public double LocationX;
        public double LocationY;
    }
}
```

Appendix 4 – My DLL script (also shown in technical solution)

```
using System;
using System.Runtime.InteropServices;

namespace HeliosLaserDAC
{

    //point data structure
    [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode, Pack = 8,Size = 7)]
    public struct HeliosPoint
    {
        public UInt16 x; //12 bit (from 0 to 0xFFF)
        public UInt16 y; //12 bit (from 0 to 0xFFF)
        public byte r; //8 bit  (from 0 to 0xFF)
        public byte g; //8 bit (from 0 to 0xFF)
        public byte b; //8 bit (from 0 to 0xFF)
        public byte i; //8 bit (from 0 to 0xFF)
    }

    public class HeliosDac
    {
        [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
        static extern int OpenDevices();
        [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl, CharSet =
CharSet.Unicode, ExactSpelling = false)]
        static unsafe extern int WriteFrame(uint devNum, int pps, byte flags, HeliosPoint* frame, uint
numOfPoints);
        [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
        static extern int CloseDevices();
        [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
        static extern string GetName(uint devNum);
        [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
        static extern UInt16 GetFirmwareVersion(uint devNum);
        [DllImport(@"\DLL\libHeliosDACAPI-x86.dll", CallingConvention = CallingConvention.Cdecl)]
        static extern int GetStatus(uint devNum);

        public int openDevices()
        {
            return OpenDevices();
        }

        public unsafe int writeFrame(int devNum, int pps, byte flags, HeliosPoint[] frame, int numOfPoints)
        {
            fixed (HeliosPoint* frameAdd = frame)
            {
                int result = WriteFrame(Convert.ToUInt32(devNum), pps, flags, frameAdd,
Convert.ToUInt32(numOfPoints));
                return result;
            }
        }

        public HeliosDac()
        {
```

```
            openDevices();
        }
        public int getStatus(int deviceNumber)
        {
            return GetStatus(Convert.ToByte(deviceNumber));
        }
        public void closeDevices()
        {
            CloseDevices();
        }
    }
}
```

This is then compiled into HeliosLaserDACLeosPart.dll and added to my code in the library.

Appendix 5 - Code that isn't fully mine + references.

The libHeliosDACAPI-x86.dll file that I reference in my code was made by the DAC manufacturer, this isn't my work.

Lines 1400 + in form1.cs weren't my code in my technical solution.

References will be dotted throughout my work or clearly highlighted as borrowed code.

Most of the references are:

https://github.com/sebleedelisle/ofxLaser

This code by Seb Lee Delisle was practically my inspiration, it essentially converted commands into a script for lasers. I contemplated about building a GUI on top of this but felt a C# system would be a good addition to the laser ecosystem.

https://github.com/Grix/helios_dac

This repo was what ran the driver I ended up buying, I spent a lot of time looking into this github as well as: https://github.com/Grix/ildagen which is the code for the alternative solution LaserShowGen

https://www.youtube.com/@seblee

This was probably where I got my inspiration for my whole project

https://bitlasers.com/helios-laser-dac/

This was more information on laser DAC's

https://www.ilda.com/

If you hear the word ILDA, essentially Ilda is the laser protocol and main foundation, made 37 years ago.

https://pangolin.com/

Another alternative solution.

https://www.youtube.com/watch?v=17Q6My9OT70

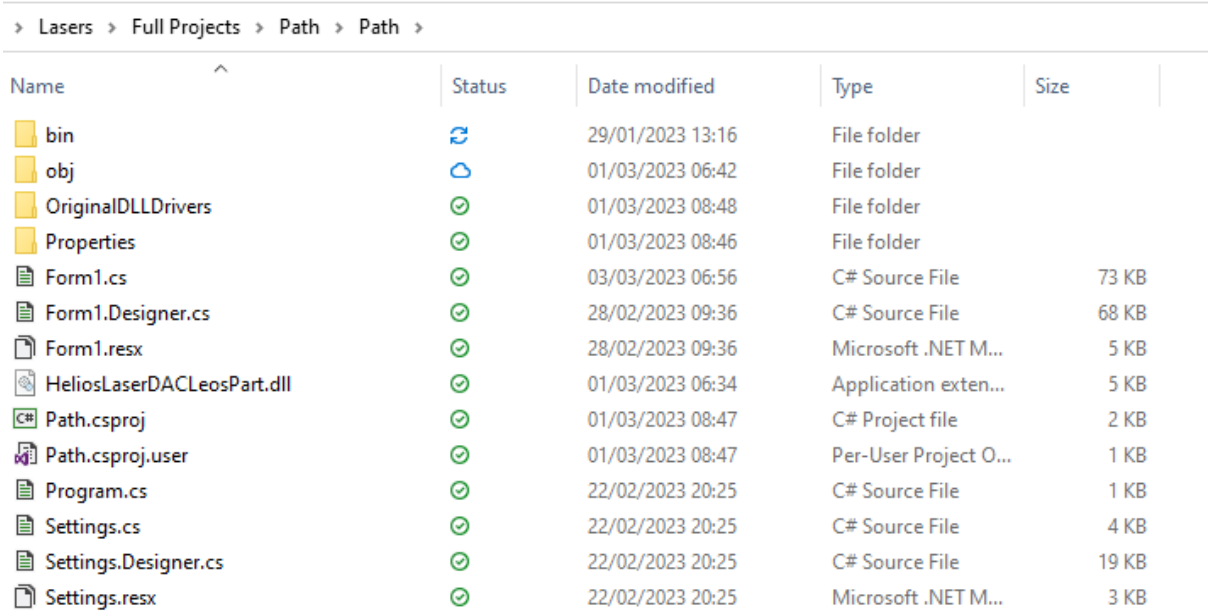
This was where I learnt about safety considerations for a Laser.

https://www.youtube.com/watch?v=SLauY6PpjW4&t=15s.

As well as code from kind members of stack overflow that I have unfortunately not got the usernames of.

Appendix 6 – My Technical solution files

See Technical solution for Form1.cs which has my written code in it.



Program.cs:

```
namespace Path
{
   internal static class Program
   {
      /// <summary>
      ///  The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         // To customize application configuration such as set high DPI settings or default font,
         // see https://aka.ms/applicationconfiguration.
         ApplicationConfiguration.Initialize();
         PathMainWindow form1 = new PathMainWindow();
         Application.Run(form1);
      }

   }
}
```

Form1.Designer.cs

```
namespace Path

{

   partial class PathMainWindow

   {
```

```
        /// <summary>
        ///  Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        ///  Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        ///  Required method for Designer support - do not modify
        ///  the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.PreviewGraphics = new System.Windows.Forms.PictureBox();
            this.OptionsPanel = new System.Windows.Forms.Panel();
            this.TimeLineFramesInput = new System.Windows.Forms.TextBox();
            this.OptionsSnapToPoint = new System.Windows.Forms.CheckBox();
```

```
this.TimeLineFramesLabel = new System.Windows.Forms.Label();
this.TimeLineSecondsInput = new System.Windows.Forms.TextBox();
this.OptionsColorSelecterOpener = new System.Windows.Forms.Button();
this.OptionsDrawLineMode = new System.Windows.Forms.CheckBox();
this.TimeLineSecondsLabel = new System.Windows.Forms.Label();
this.OptionsSelectModeButton = new System.Windows.Forms.CheckBox();
this.OptionsEmergencyLabel = new System.Windows.Forms.Label();
this.OptionsToggleProject = new System.Windows.Forms.CheckBox();
this.OptionsSavePathFile = new System.Windows.Forms.Button();
this.OptionsLoadPathFile = new System.Windows.Forms.Button();
this.OptionsKPPSTextBox = new System.Windows.Forms.TextBox();
this.OptionsFPSTextBox = new System.Windows.Forms.TextBox();
this.OptionsKPPSLabel = new System.Windows.Forms.Label();
this.OptionsTitleLabel = new System.Windows.Forms.Label();
this.OptionsFPSLabel = new System.Windows.Forms.Label();
this.TimeLinePanel = new System.Windows.Forms.Panel();
this.TimeLineMaxTimeLabel = new System.Windows.Forms.Label();
this.projectMaxTimeSelector = new System.Windows.Forms.NumericUpDown();
this.timelineGUIHugger = new System.Windows.Forms.Panel();
this.timelineGUI = new System.Windows.Forms.PictureBox();
this.TimeLinePlay = new System.Windows.Forms.CheckBox();
this.TimeLineNextFrame = new System.Windows.Forms.Button();
this.TimeLineBackFrame = new System.Windows.Forms.Button();
this.TimeLineTitleLabel = new System.Windows.Forms.Label();
this.openFileDialog1 = new System.Windows.Forms.OpenFileDialog();
this.saveFileDialog1 = new System.Windows.Forms.SaveFileDialog();
this.LinePropertiesPanel = new System.Windows.Forms.Panel();
this.LinePropertiesYCoordinate = new System.Windows.Forms.NumericUpDown();
this.LinePropertiesXCoordinate = new System.Windows.Forms.NumericUpDown();
this.linePropertiesStrobeSettings = new System.Windows.Forms.TabControl();
```

```
this.linePropertiesFrameControlsPanel = new System.Windows.Forms.TabPage();
this.deleteShape = new System.Windows.Forms.Button();
this.tabPage2 = new System.Windows.Forms.TabPage();
this.PathLinePointsListBox = new System.Windows.Forms.ListBox();
this.LinePropertiesChangeColor = new System.Windows.Forms.Button();
this.LinePropertiesYCoordinateLabel = new System.Windows.Forms.Label();
this.LinePropertiesPathIndexData = new System.Windows.Forms.Label();
this.LinePropertiesXCoordinateLabel = new System.Windows.Forms.Label();
this.LinePropertiesPathIndexLabel = new System.Windows.Forms.Label();
this.LinePropertiesKeyFramesTextBox = new System.Windows.Forms.ListBox();
this.LinePropertiesTitle = new System.Windows.Forms.Label();
this.InformationPanel = new System.Windows.Forms.Panel();
this.InformationPreviewModeData = new System.Windows.Forms.Label();
this.InformationConnectedDACCount = new System.Windows.Forms.Label();
this.InformationClosestPointData = new System.Windows.Forms.Label();
this.InformationClosestPointLabel = new System.Windows.Forms.Label();
this.InformationFrameListCountInfo = new System.Windows.Forms.Label();
this.InformationDynamicListCountInfo = new System.Windows.Forms.Label();
this.InformationFrameListCountLabel = new System.Windows.Forms.Label();
this.InformationDynamicListCountLabel = new System.Windows.Forms.Label();
this.InformationPoint2Info = new System.Windows.Forms.Label();
this.InformationPoint1Info = new System.Windows.Forms.Label();
this.InformationPoint2TitleLabel = new System.Windows.Forms.Label();
this.InformationPoint1TitleLablel = new System.Windows.Forms.Label();
this.InformationTitleLabel = new System.Windows.Forms.Label();
this.DrawerColorDialog = new System.Windows.Forms.ColorDialog();
this.LinePropertiesColorDialog = new System.Windows.Forms.ColorDialog();
this.menuStrip1 = new System.Windows.Forms.MenuStrip();
this.fileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.newPathProjectToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
```

```
this.openToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.saveToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.showToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.laserPathToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.laserPointsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.previewShapesToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.settingsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.sToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.dacSettingsToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.connectToDACToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.disconnectDACToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
this.splitContainer1 = new System.Windows.Forms.SplitContainer();
this.backgroundWorker1 = new System.ComponentModel.BackgroundWorker();
this.backgroundWorker2 = new System.ComponentModel.BackgroundWorker();
((System.ComponentModel.ISupportInitialize)(this.PreviewGraphics)).BeginInit();
this.OptionsPanel.SuspendLayout();
this.TimeLinePanel.SuspendLayout();
((System.ComponentModel.ISupportInitialize)(this.projectMaxTimeSelector)).BeginInit();
this.timelineGUIHugger.SuspendLayout();
((System.ComponentModel.ISupportInitialize)(this.timelineGUI)).BeginInit();
this.LinePropertiesPanel.SuspendLayout();
((System.ComponentModel.ISupportInitialize)(this.LinePropertiesYCoordinate)).BeginInit();
((System.ComponentModel.ISupportInitialize)(this.LinePropertiesXCoordinate)).BeginInit();
this.linePropertiesStrobeSettings.SuspendLayout();
this.linePropertiesFrameControlsPanel.SuspendLayout();
this.InformationPanel.SuspendLayout();
this.menuStrip1.SuspendLayout();
((System.ComponentModel.ISupportInitialize)(this.splitContainer1)).BeginInit();
this.splitContainer1.Panel1.SuspendLayout();
this.splitContainer1.Panel2.SuspendLayout();
```

```
        this.splitContainer1.SuspendLayout();
        this.SuspendLayout();
        //
        // PreviewGraphics
        //
        this.PreviewGraphics.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
        this.PreviewGraphics.Location = new System.Drawing.Point(5, 3);
        this.PreviewGraphics.Name = "PreviewGraphics";
        this.PreviewGraphics.Size = new System.Drawing.Size(569, 490);
        this.PreviewGraphics.TabIndex = 0;
        this.PreviewGraphics.TabStop = false;
        this.PreviewGraphics.Paint += new
System.Windows.Forms.PaintEventHandler(this.PreviewGraphics_Paint);
        this.PreviewGraphics.MouseDown += new
System.Windows.Forms.MouseEventHandler(this.PreviewGraphics_MouseDown);
        this.PreviewGraphics.MouseMove += new
System.Windows.Forms.MouseEventHandler(this.PreviewGraphics_MouseMove);
        this.PreviewGraphics.MouseUp += new
System.Windows.Forms.MouseEventHandler(this.PreviewGraphics_MouseUp);
        this.PreviewGraphics.Resize += new System.EventHandler(this.PreviewGraphics_Resize);
        //
        // OptionsPanel
        //
        this.OptionsPanel.Anchor =
((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Left)
        | System.Windows.Forms.AnchorStyles.Right)));
        this.OptionsPanel.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;
        this.OptionsPanel.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
        this.OptionsPanel.Controls.Add(this.TimeLineFramesInput);
        this.OptionsPanel.Controls.Add(this.OptionsSnapToPoint);
        this.OptionsPanel.Controls.Add(this.TimeLineFramesLabel);
```

```
        this.OptionsPanel.Controls.Add(this.TimeLineSecondsInput);
        this.OptionsPanel.Controls.Add(this.OptionsColorSelecterOpener);
        this.OptionsPanel.Controls.Add(this.OptionsDrawLineMode);
        this.OptionsPanel.Controls.Add(this.TimeLineSecondsLabel);
        this.OptionsPanel.Controls.Add(this.OptionsSelectModeButton);
        this.OptionsPanel.Controls.Add(this.OptionsEmergencyLabel);
        this.OptionsPanel.Controls.Add(this.OptionsToggleProject);
        this.OptionsPanel.Controls.Add(this.OptionsSavePathFile);
        this.OptionsPanel.Controls.Add(this.OptionsLoadPathFile);
        this.OptionsPanel.Controls.Add(this.OptionsKPPSTextBox);
        this.OptionsPanel.Controls.Add(this.OptionsFPSTextBox);
        this.OptionsPanel.Controls.Add(this.OptionsKPPSLabel);
        this.OptionsPanel.Controls.Add(this.OptionsTitleLabel);
        this.OptionsPanel.Controls.Add(this.OptionsFPSLabel);
        this.OptionsPanel.Location = new System.Drawing.Point(5, 481);
        this.OptionsPanel.Margin = new System.Windows.Forms.Padding(5, 6, 5, 6);
        this.OptionsPanel.Name = "OptionsPanel";
        this.OptionsPanel.Size = new System.Drawing.Size(402, 208);
        this.OptionsPanel.TabIndex = 1;
        //
        // TimeLineFramesInput
        //
        this.TimeLineFramesInput.BackColor = System.Drawing.SystemColors.ControlText;
        this.TimeLineFramesInput.ForeColor = System.Drawing.SystemColors.Control;
        this.TimeLineFramesInput.Location = new System.Drawing.Point(341, 135);
        this.TimeLineFramesInput.Name = "TimeLineFramesInput";
        this.TimeLineFramesInput.Size = new System.Drawing.Size(51, 23);
        this.TimeLineFramesInput.TabIndex = 8;
        this.TimeLineFramesInput.Text = "0";
        this.TimeLineFramesInput.TextChanged += new
System.EventHandler(this.TimeLineFramesInput_TextChanged);
```

```
        //
        // OptionsSnapToPoint
        //
        this.OptionsSnapToPoint.Appearance = System.Windows.Forms.Appearance.Button;
        this.OptionsSnapToPoint.AutoSize = true;
        this.OptionsSnapToPoint.BackColor = System.Drawing.SystemColors.ControlText;
        this.OptionsSnapToPoint.ForeColor = System.Drawing.SystemColors.Control;
        this.OptionsSnapToPoint.Location = new System.Drawing.Point(217, 175);
        this.OptionsSnapToPoint.Name = "OptionsSnapToPoint";
        this.OptionsSnapToPoint.Size = new System.Drawing.Size(89, 25);
        this.OptionsSnapToPoint.TabIndex = 19;
        this.OptionsSnapToPoint.Text = "Snap To Point";
        this.OptionsSnapToPoint.UseVisualStyleBackColor = false;
        this.OptionsSnapToPoint.CheckedChanged += new
System.EventHandler(this.OptionsSnapToPoint_CheckedChanged);
        //
        // TimeLineFramesLabel
        //
        this.TimeLineFramesLabel.AutoSize = true;
        this.TimeLineFramesLabel.ForeColor = System.Drawing.SystemColors.Control;
        this.TimeLineFramesLabel.Location = new System.Drawing.Point(253, 138);
        this.TimeLineFramesLabel.Name = "TimeLineFramesLabel";
        this.TimeLineFramesLabel.Size = new System.Drawing.Size(82, 15);
        this.TimeLineFramesLabel.TabIndex = 6;
        this.TimeLineFramesLabel.Text = "Time (Frames)";
        //
        // TimeLineSecondsInput
        //
        this.TimeLineSecondsInput.BackColor = System.Drawing.SystemColors.ControlText;
        this.TimeLineSecondsInput.ForeColor = System.Drawing.SystemColors.Control;
        this.TimeLineSecondsInput.Location = new System.Drawing.Point(341, 104);
```

```
        this.TimeLineSecondsInput.Name = "TimeLineSecondsInput";
        this.TimeLineSecondsInput.Size = new System.Drawing.Size(51, 23);
        this.TimeLineSecondsInput.TabIndex = 7;
        this.TimeLineSecondsInput.Text = "0";
        this.TimeLineSecondsInput.TextChanged += new
System.EventHandler(this.TimeLineSecondsInput_TextChanged);
        //
        // OptionsColorSelecterOpener
        //
        this.OptionsColorSelecterOpener.BackColor = System.Drawing.SystemColors.ControlText;
        this.OptionsColorSelecterOpener.ForeColor = System.Drawing.SystemColors.HighlightText;
        this.OptionsColorSelecterOpener.Location = new System.Drawing.Point(310, 175);
        this.OptionsColorSelecterOpener.Name = "OptionsColorSelecterOpener";
        this.OptionsColorSelecterOpener.Size = new System.Drawing.Size(82, 26);
        this.OptionsColorSelecterOpener.TabIndex = 18;
        this.OptionsColorSelecterOpener.Text = "Color";
        this.OptionsColorSelecterOpener.UseVisualStyleBackColor = false;
        this.OptionsColorSelecterOpener.Click += new
System.EventHandler(this.OptionsColorSelecterOpener_Click);
        //
        // OptionsDrawLineMode
        //
        this.OptionsDrawLineMode.Appearance = System.Windows.Forms.Appearance.Button;
        this.OptionsDrawLineMode.AutoSize = true;
        this.OptionsDrawLineMode.BackColor = System.Drawing.SystemColors.ControlText;
        this.OptionsDrawLineMode.ForeColor = System.Drawing.SystemColors.Control;
        this.OptionsDrawLineMode.Location = new System.Drawing.Point(82, 171);
        this.OptionsDrawLineMode.Name = "OptionsDrawLineMode";
        this.OptionsDrawLineMode.Size = new System.Drawing.Size(69, 25);
        this.OptionsDrawLineMode.TabIndex = 17;
        this.OptionsDrawLineMode.Text = "Draw Line";
```

```
        this.OptionsDrawLineMode.UseVisualStyleBackColor = false;
        this.OptionsDrawLineMode.CheckedChanged += new
System.EventHandler(this.OptionsDrawLineMode_CheckedChanged);
        //
        // TimeLineSecondsLabel
        //
        this.TimeLineSecondsLabel.AutoSize = true;
        this.TimeLineSecondsLabel.ForeColor = System.Drawing.SystemColors.Control;
        this.TimeLineSecondsLabel.Location = new System.Drawing.Point(247, 107);
        this.TimeLineSecondsLabel.Name = "TimeLineSecondsLabel";
        this.TimeLineSecondsLabel.Size = new System.Drawing.Size(88, 15);
        this.TimeLineSecondsLabel.TabIndex = 5;
        this.TimeLineSecondsLabel.Text = "Time (Seconds)";
        //
        // OptionsSelectModeButton
        //
        this.OptionsSelectModeButton.Appearance = System.Windows.Forms.Appearance.Button;
        this.OptionsSelectModeButton.AutoSize = true;
        this.OptionsSelectModeButton.BackColor = System.Drawing.SystemColors.ControlText;
        this.OptionsSelectModeButton.ForeColor = System.Drawing.SystemColors.Control;
        this.OptionsSelectModeButton.Location = new System.Drawing.Point(3, 171);
        this.OptionsSelectModeButton.Name = "OptionsSelectModeButton";
        this.OptionsSelectModeButton.Size = new System.Drawing.Size(73, 25);
        this.OptionsSelectModeButton.TabIndex = 16;
        this.OptionsSelectModeButton.Text = "Select Tool";
        this.OptionsSelectModeButton.UseVisualStyleBackColor = false;
        this.OptionsSelectModeButton.CheckedChanged += new
System.EventHandler(this.OptionsSelectModeButton_CheckedChanged);
        //
        // OptionsEmergencyLabel
        //
```

```
        this.OptionsEmergencyLabel.AutoSize = true;
        this.OptionsEmergencyLabel.ForeColor = System.Drawing.SystemColors.Control;
        this.OptionsEmergencyLabel.Location = new System.Drawing.Point(85, 133);
        this.OptionsEmergencyLabel.Name = "OptionsEmergencyLabel";
        this.OptionsEmergencyLabel.Size = new System.Drawing.Size(162, 15);
        this.OptionsEmergencyLabel.TabIndex = 15;
        this.OptionsEmergencyLabel.Text = "PRESS SPACE IN EMERGENCY";
        //
        // OptionsToggleProject
        //
        this.OptionsToggleProject.Appearance = System.Windows.Forms.Appearance.Button;
        this.OptionsToggleProject.AutoSize = true;
        this.OptionsToggleProject.BackColor = System.Drawing.SystemColors.ControlText;
        this.OptionsToggleProject.ForeColor = System.Drawing.SystemColors.Control;
        this.OptionsToggleProject.Location = new System.Drawing.Point(3, 128);
        this.OptionsToggleProject.Name = "OptionsToggleProject";
        this.OptionsToggleProject.Size = new System.Drawing.Size(82, 25);
        this.OptionsToggleProject.TabIndex = 14;
        this.OptionsToggleProject.Text = "Toggle Laser";
        this.OptionsToggleProject.UseVisualStyleBackColor = false;
        this.OptionsToggleProject.CheckedChanged += new
System.EventHandler(this.OptionsToggleProject_CheckedChanged);
        //
        // OptionsSavePathFile
        //
        this.OptionsSavePathFile.BackColor = System.Drawing.SystemColors.ControlText;
        this.OptionsSavePathFile.ForeColor = System.Drawing.SystemColors.Control;
        this.OptionsSavePathFile.Location = new System.Drawing.Point(125, 95);
        this.OptionsSavePathFile.Name = "OptionsSavePathFile";
        this.OptionsSavePathFile.Size = new System.Drawing.Size(116, 23);
        this.OptionsSavePathFile.TabIndex = 13;
```

```
this.OptionsSavePathFile.Text = "Save Path File";
this.OptionsSavePathFile.UseVisualStyleBackColor = false;
//
// OptionsLoadPathFile
//
this.OptionsLoadPathFile.BackColor = System.Drawing.SystemColors.ControlText;
this.OptionsLoadPathFile.ForeColor = System.Drawing.SystemColors.Control;
this.OptionsLoadPathFile.Location = new System.Drawing.Point(3, 95);
this.OptionsLoadPathFile.Name = "OptionsLoadPathFile";
this.OptionsLoadPathFile.Size = new System.Drawing.Size(116, 23);
this.OptionsLoadPathFile.TabIndex = 12;
this.OptionsLoadPathFile.Text = "Load Path File";
this.OptionsLoadPathFile.UseVisualStyleBackColor = false;
//
// OptionsKPPSTextBox
//
this.OptionsKPPSTextBox.BackColor = System.Drawing.SystemColors.ControlText;
this.OptionsKPPSTextBox.ForeColor = System.Drawing.SystemColors.Control;
this.OptionsKPPSTextBox.Location = new System.Drawing.Point(149, 61);
this.OptionsKPPSTextBox.Name = "OptionsKPPSTextBox";
this.OptionsKPPSTextBox.Size = new System.Drawing.Size(145, 23);
this.OptionsKPPSTextBox.TabIndex = 11;
this.OptionsKPPSTextBox.Text = "40000";
//
// OptionsFPSTextBox
//
this.OptionsFPSTextBox.BackColor = System.Drawing.SystemColors.ControlText;
this.OptionsFPSTextBox.ForeColor = System.Drawing.SystemColors.Control;
this.OptionsFPSTextBox.Location = new System.Drawing.Point(149, 34);
this.OptionsFPSTextBox.Name = "OptionsFPSTextBox";
```

```
        this.OptionsFPSTextBox.Size = new System.Drawing.Size(145, 23);
        this.OptionsFPSTextBox.TabIndex = 9;
        this.OptionsFPSTextBox.Text = "30";
        //
        // OptionsKPPSLabel
        //
        this.OptionsKPPSLabel.AutoSize = true;
        this.OptionsKPPSLabel.ForeColor = System.Drawing.SystemColors.Control;
        this.OptionsKPPSLabel.Location = new System.Drawing.Point(3, 64);
        this.OptionsKPPSLabel.Name = "OptionsKPPSLabel";
        this.OptionsKPPSLabel.Size = new System.Drawing.Size(143, 15);
        this.OptionsKPPSLabel.TabIndex = 10;
        this.OptionsKPPSLabel.Text = "KPPS (Points Per Second):";
        //
        // OptionsTitleLabel
        //
        this.OptionsTitleLabel.AccessibleName = "Time Manager";
        this.OptionsTitleLabel.AutoSize = true;
        this.OptionsTitleLabel.Font = new System.Drawing.Font("Segoe UI", 15.75F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
        this.OptionsTitleLabel.ForeColor = System.Drawing.SystemColors.ControlLight;
        this.OptionsTitleLabel.Location = new System.Drawing.Point(-2, 0);
        this.OptionsTitleLabel.Name = "OptionsTitleLabel";
        this.OptionsTitleLabel.Size = new System.Drawing.Size(86, 30);
        this.OptionsTitleLabel.TabIndex = 9;
        this.OptionsTitleLabel.Text = "Options";
        //
        // OptionsFPSLabel
        //
        this.OptionsFPSLabel.AutoSize = true;
        this.OptionsFPSLabel.ForeColor = System.Drawing.SystemColors.Control;
```

```
this.OptionsFPSLabel.Location = new System.Drawing.Point(3, 37);
this.OptionsFPSLabel.Name = "OptionsFPSLabel";
this.OptionsFPSLabel.Size = new System.Drawing.Size(140, 15);
this.OptionsFPSLabel.TabIndex = 9;
this.OptionsFPSLabel.Text = "FPS (Frames Per Second):";
//
// TimeLinePanel
//
this.TimeLinePanel.Anchor = System.Windows.Forms.AnchorStyles.Bottom;
this.TimeLinePanel.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;
this.TimeLinePanel.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
this.TimeLinePanel.Controls.Add(this.TimeLineMaxTimeLabel);
this.TimeLinePanel.Controls.Add(this.projectMaxTimeSelector);
this.TimeLinePanel.Controls.Add(this.timelineGUIHugger);
this.TimeLinePanel.Controls.Add(this.TimeLinePlay);
this.TimeLinePanel.Controls.Add(this.TimeLineNextFrame);
this.TimeLinePanel.Controls.Add(this.TimeLineBackFrame);
this.TimeLinePanel.Controls.Add(this.TimeLineTitleLabel);
this.TimeLinePanel.Location = new System.Drawing.Point(5, 499);
this.TimeLinePanel.Name = "TimeLinePanel";
this.TimeLinePanel.Size = new System.Drawing.Size(569, 192);
this.TimeLinePanel.TabIndex = 2;
//
// TimeLineMaxTimeLabel
//
this.TimeLineMaxTimeLabel.AutoSize = true;
this.TimeLineMaxTimeLabel.ForeColor = System.Drawing.SystemColors.Control;
this.TimeLineMaxTimeLabel.Location = new System.Drawing.Point(433, 8);
this.TimeLineMaxTimeLabel.Name = "TimeLineMaxTimeLabel";
this.TimeLineMaxTimeLabel.Size = new System.Drawing.Size(62, 15);
```

```
        this.TimeLineMaxTimeLabel.TabIndex = 7;
        this.TimeLineMaxTimeLabel.Text = "Max Time:";
        //
        // projectMaxTimeSelector
        //
        this.projectMaxTimeSelector.BackColor = System.Drawing.SystemColors.ControlText;
        this.projectMaxTimeSelector.ForeColor = System.Drawing.SystemColors.Control;
        this.projectMaxTimeSelector.Location = new System.Drawing.Point(501, 4);
        this.projectMaxTimeSelector.Maximum = new decimal(new int[] {
        -727379968,
        232,
        0,
        0});
        this.projectMaxTimeSelector.Name = "projectMaxTimeSelector";
        this.projectMaxTimeSelector.Size = new System.Drawing.Size(53, 23);
        this.projectMaxTimeSelector.TabIndex = 6;
        this.projectMaxTimeSelector.Value = new decimal(new int[] {
        30,
        0,
        0,
        0});
        this.projectMaxTimeSelector.ValueChanged += new
System.EventHandler(this.projectMaxTimeSelector_ValueChanged);
        //
        // timelineGUIHugger
        //
        this.timelineGUIHugger.Anchor =
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
        | System.Windows.Forms.AnchorStyles.Left)
        | System.Windows.Forms.AnchorStyles.Right)));
```

```
        this.timelineGUIHugger.AutoScroll = true;
        this.timelineGUIHugger.AutoSizeMode =
System.Windows.Forms.AutoSizeMode.GrowAndShrink;
        this.timelineGUIHugger.Controls.Add(this.timelineGUI);
        this.timelineGUIHugger.Location = new System.Drawing.Point(5, 36);
        this.timelineGUIHugger.Name = "timelineGUIHugger";
        this.timelineGUIHugger.Size = new System.Drawing.Size(556, 149);
        this.timelineGUIHugger.TabIndex = 5;
        //
        // timelineGUI
        //
        this.timelineGUI.Location = new System.Drawing.Point(0, 0);
        this.timelineGUI.Name = "timelineGUI";
        this.timelineGUI.Size = new System.Drawing.Size(556, 149);
        this.timelineGUI.TabIndex = 5;
        this.timelineGUI.TabStop = false;
        this.timelineGUI.Paint += new
System.Windows.Forms.PaintEventHandler(this.timeline_GUI_updater);
        this.timelineGUI.MouseDown += new
System.Windows.Forms.MouseEventHandler(this.timelineGUI_MouseDown);
        this.timelineGUI.MouseMove += new
System.Windows.Forms.MouseEventHandler(this.timelineGUI_MouseMove);
        //
        // TimeLinePlay
        //
        this.TimeLinePlay.Appearance = System.Windows.Forms.Appearance.Button;
        this.TimeLinePlay.AutoSize = true;
        this.TimeLinePlay.BackColor = System.Drawing.SystemColors.ControlText;
        this.TimeLinePlay.ForeColor = System.Drawing.SystemColors.Control;
        this.TimeLinePlay.Location = new System.Drawing.Point(210, 3);
        this.TimeLinePlay.Name = "TimeLinePlay";
        this.TimeLinePlay.Size = new System.Drawing.Size(39, 25);
```

```
        this.TimeLinePlay.TabIndex = 4;
        this.TimeLinePlay.Text = "Play";
        this.TimeLinePlay.UseVisualStyleBackColor = false;
        this.TimeLinePlay.CheckedChanged += new
System.EventHandler(this.TimeLinePlay_CheckedChanged);
        //
        // TimeLineNextFrame
        //
        this.TimeLineNextFrame.BackColor = System.Drawing.SystemColors.ControlText;
        this.TimeLineNextFrame.ForeColor = System.Drawing.SystemColors.Control;
        this.TimeLineNextFrame.Location = new System.Drawing.Point(255, 4);
        this.TimeLineNextFrame.Name = "TimeLineNextFrame";
        this.TimeLineNextFrame.Size = new System.Drawing.Size(47, 24);
        this.TimeLineNextFrame.TabIndex = 3;
        this.TimeLineNextFrame.Text = "Next";
        this.TimeLineNextFrame.UseVisualStyleBackColor = false;
        //
        // TimeLineBackFrame
        //
        this.TimeLineBackFrame.BackColor = System.Drawing.SystemColors.ControlText;
        this.TimeLineBackFrame.ForeColor = System.Drawing.Color.FromArgb(((int)(((byte)(224)))),
((int)(((byte)(224)))), ((int)(((byte)(224)))));
        this.TimeLineBackFrame.Location = new System.Drawing.Point(156, 3);
        this.TimeLineBackFrame.Name = "TimeLineBackFrame";
        this.TimeLineBackFrame.Size = new System.Drawing.Size(48, 24);
        this.TimeLineBackFrame.TabIndex = 2;
        this.TimeLineBackFrame.Text = "Back";
        this.TimeLineBackFrame.UseVisualStyleBackColor = false;
        //
        // TimeLineTitleLabel
        //
```

```
        this.TimeLineTitleLabel.AccessibleName = "Time Manager";
        this.TimeLineTitleLabel.AutoSize = true;
        this.TimeLineTitleLabel.Font = new System.Drawing.Font("Segoe UI", 15.75F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
        this.TimeLineTitleLabel.ForeColor = System.Drawing.SystemColors.ControlLight;
        this.TimeLineTitleLabel.Location = new System.Drawing.Point(3, 0);
        this.TimeLineTitleLabel.Name = "TimeLineTitleLabel";
        this.TimeLineTitleLabel.Size = new System.Drawing.Size(147, 30);
        this.TimeLineTitleLabel.TabIndex = 0;
        this.TimeLineTitleLabel.Text = "Time Operator";
        //
        // openFileDialog1
        //
        this.openFileDialog1.DefaultExt = "dyPth";
        this.openFileDialog1.FileName = "openFileDialog1";
        //
        // saveFileDialog1
        //
        this.saveFileDialog1.DefaultExt = "dyPth";
        //
        // LinePropertiesPanel
        //
        this.LinePropertiesPanel.Anchor =
((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Left)
        | System.Windows.Forms.AnchorStyles.Right)));
        this.LinePropertiesPanel.AutoScroll = true;
        this.LinePropertiesPanel.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
        this.LinePropertiesPanel.Controls.Add(this.LinePropertiesYCoordinate);
        this.LinePropertiesPanel.Controls.Add(this.LinePropertiesXCoordinate);
        this.LinePropertiesPanel.Controls.Add(this.linePropertiesStrobeSettings);
```

```
        this.LinePropertiesPanel.Controls.Add(this.PathLinePointsListBox);
        this.LinePropertiesPanel.Controls.Add(this.LinePropertiesChangeColor);
        this.LinePropertiesPanel.Controls.Add(this.LinePropertiesYCoordinateLabel);
        this.LinePropertiesPanel.Controls.Add(this.LinePropertiesPathIndexData);
        this.LinePropertiesPanel.Controls.Add(this.LinePropertiesXCoordinateLabel);
        this.LinePropertiesPanel.Controls.Add(this.LinePropertiesPathIndexLabel);
        this.LinePropertiesPanel.Controls.Add(this.LinePropertiesKeyFramesTextBox);
        this.LinePropertiesPanel.Controls.Add(this.LinePropertiesTitle);
        this.LinePropertiesPanel.Location = new System.Drawing.Point(3, 3);
        this.LinePropertiesPanel.Name = "LinePropertiesPanel";
        this.LinePropertiesPanel.Size = new System.Drawing.Size(405, 312);
        this.LinePropertiesPanel.TabIndex = 3;
        //
        // LinePropertiesYCoordinate
        //
        this.LinePropertiesYCoordinate.BackColor = System.Drawing.SystemColors.ControlText;
        this.LinePropertiesYCoordinate.ForeColor = System.Drawing.SystemColors.HighlightText;
        this.LinePropertiesYCoordinate.Location = new System.Drawing.Point(119, 151);
        this.LinePropertiesYCoordinate.Margin = new System.Windows.Forms.Padding(2);
        this.LinePropertiesYCoordinate.Maximum = new decimal(new int[] {
        4096,
        0,
        0,
        0});
        this.LinePropertiesYCoordinate.Name = "LinePropertiesYCoordinate";
        this.LinePropertiesYCoordinate.Size = new System.Drawing.Size(59, 23);
        this.LinePropertiesYCoordinate.TabIndex = 34;
        this.LinePropertiesYCoordinate.ValueChanged += new
System.EventHandler(this.linePropertiesXOrYCoordinate_ValueChanged);
        //
        // LinePropertiesXCoordinate
```

```
        //
        this.LinePropertiesXCoordinate.BackColor = System.Drawing.SystemColors.Desktop;
        this.LinePropertiesXCoordinate.ForeColor = System.Drawing.SystemColors.HighlightText;
        this.LinePropertiesXCoordinate.Location = new System.Drawing.Point(29, 151);
        this.LinePropertiesXCoordinate.Margin = new System.Windows.Forms.Padding(2);
        this.LinePropertiesXCoordinate.Maximum = new decimal(new int[] {
        4096,
        0,
        0,
        0});
        this.LinePropertiesXCoordinate.Name = "LinePropertiesXCoordinate";
        this.LinePropertiesXCoordinate.Size = new System.Drawing.Size(59, 23);
        this.LinePropertiesXCoordinate.TabIndex = 33;
        this.LinePropertiesXCoordinate.ValueChanged += new
System.EventHandler(this.linePropertiesXOrYCoordinate_ValueChanged);
        //
        // linePropertiesStrobeSettings
        //
        this.linePropertiesStrobeSettings.Controls.Add(this.linePropertiesFrameControlsPanel);
        this.linePropertiesStrobeSettings.Controls.Add(this.tabPage2);
        this.linePropertiesStrobeSettings.Location = new System.Drawing.Point(3, 177);
        this.linePropertiesStrobeSettings.Name = "linePropertiesStrobeSettings";
        this.linePropertiesStrobeSettings.SelectedIndex = 0;
        this.linePropertiesStrobeSettings.Size = new System.Drawing.Size(395, 128);
        this.linePropertiesStrobeSettings.TabIndex = 32;
        //
        // linePropertiesFrameControlsPanel
        //
        this.linePropertiesFrameControlsPanel.BackColor =
System.Drawing.SystemColors.ControlText;
        this.linePropertiesFrameControlsPanel.Controls.Add(this.deleteShape);
```

```
this.linePropertiesFrameControlsPanel.ForeColor = System.Drawing.SystemColors.Control;
this.linePropertiesFrameControlsPanel.Location = new System.Drawing.Point(4, 24);
this.linePropertiesFrameControlsPanel.Name = "linePropertiesFrameControlsPanel";
this.linePropertiesFrameControlsPanel.Padding = new System.Windows.Forms.Padding(3);
this.linePropertiesFrameControlsPanel.Size = new System.Drawing.Size(387, 100);
this.linePropertiesFrameControlsPanel.TabIndex = 0;
this.linePropertiesFrameControlsPanel.Text = "Frame Controls";
//
// deleteShape
//
this.deleteShape.BackColor = System.Drawing.SystemColors.ControlText;
this.deleteShape.Location = new System.Drawing.Point(6, 6);
this.deleteShape.Name = "deleteShape";
this.deleteShape.Size = new System.Drawing.Size(113, 23);
this.deleteShape.TabIndex = 0;
this.deleteShape.Text = "Delete Shape";
this.deleteShape.UseVisualStyleBackColor = false;
this.deleteShape.Click += new System.EventHandler(this.deleteShape_Click);
//
// tabPage2
//
this.tabPage2.BackColor = System.Drawing.Color.Black;
this.tabPage2.ForeColor = System.Drawing.SystemColors.Control;
this.tabPage2.Location = new System.Drawing.Point(4, 24);
this.tabPage2.Name = "tabPage2";
this.tabPage2.Padding = new System.Windows.Forms.Padding(3);
this.tabPage2.Size = new System.Drawing.Size(387, 100);
this.tabPage2.TabIndex = 1;
this.tabPage2.Text = "Effects";
//
```

```
        // PathLinePointsListBox
        //
        this.PathLinePointsListBox.BackColor = System.Drawing.SystemColors.Desktop;
        this.PathLinePointsListBox.ForeColor = System.Drawing.SystemColors.HighlightText;
        this.PathLinePointsListBox.FormattingEnabled = true;
        this.PathLinePointsListBox.ItemHeight = 15;
        this.PathLinePointsListBox.Location = new System.Drawing.Point(217, 48);
        this.PathLinePointsListBox.Name = "PathLinePointsListBox";
        this.PathLinePointsListBox.Size = new System.Drawing.Size(181, 94);
        this.PathLinePointsListBox.TabIndex = 31;
        this.PathLinePointsListBox.SelectedIndexChanged += new
System.EventHandler(this.PathLinePointsListBox_SelectedIndexChanged);
        //
        // LinePropertiesChangeColor
        //
        this.LinePropertiesChangeColor.BackColor = System.Drawing.Color.Black;
        this.LinePropertiesChangeColor.ForeColor = System.Drawing.SystemColors.Control;
        this.LinePropertiesChangeColor.Location = new System.Drawing.Point(197, 148);
        this.LinePropertiesChangeColor.Name = "LinePropertiesChangeColor";
        this.LinePropertiesChangeColor.Size = new System.Drawing.Size(201, 23);
        this.LinePropertiesChangeColor.TabIndex = 30;
        this.LinePropertiesChangeColor.Text = "Color";
        this.LinePropertiesChangeColor.UseVisualStyleBackColor = false;
        this.LinePropertiesChangeColor.Click += new
System.EventHandler(this.LinePropertiesChangeColor_Click);
        //
        // LinePropertiesYCoordinateLabel
        //
        this.LinePropertiesYCoordinateLabel.AutoSize = true;
        this.LinePropertiesYCoordinateLabel.ForeColor = System.Drawing.SystemColors.Control;
        this.LinePropertiesYCoordinateLabel.Location = new System.Drawing.Point(97, 152);
```

```
this.LinePropertiesYCoordinateLabel.Name = "LinePropertiesYCoordinateLabel";
this.LinePropertiesYCoordinateLabel.Size = new System.Drawing.Size(17, 15);
this.LinePropertiesYCoordinateLabel.TabIndex = 22;
this.LinePropertiesYCoordinateLabel.Text = "Y:";
//
// LinePropertiesPathIndexData
//
this.LinePropertiesPathIndexData.AutoSize = true;
this.LinePropertiesPathIndexData.ForeColor = System.Drawing.SystemColors.Control;
this.LinePropertiesPathIndexData.Location = new System.Drawing.Point(74, 30);
this.LinePropertiesPathIndexData.Name = "LinePropertiesPathIndexData";
this.LinePropertiesPathIndexData.Size = new System.Drawing.Size(114, 15);
this.LinePropertiesPathIndexData.TabIndex = 28;
this.LinePropertiesPathIndexData.Text = "No Line Selected Yet";
//
// LinePropertiesXCoordinateLabel
//
this.LinePropertiesXCoordinateLabel.AutoSize = true;
this.LinePropertiesXCoordinateLabel.ForeColor = System.Drawing.SystemColors.Control;
this.LinePropertiesXCoordinateLabel.Location = new System.Drawing.Point(7, 152);
this.LinePropertiesXCoordinateLabel.Name = "LinePropertiesXCoordinateLabel";
this.LinePropertiesXCoordinateLabel.Size = new System.Drawing.Size(17, 15);
this.LinePropertiesXCoordinateLabel.TabIndex = 21;
this.LinePropertiesXCoordinateLabel.Text = "X:";
//
// LinePropertiesPathIndexLabel
//
this.LinePropertiesPathIndexLabel.AutoSize = true;
this.LinePropertiesPathIndexLabel.ForeColor = System.Drawing.SystemColors.Control;
this.LinePropertiesPathIndexLabel.Location = new System.Drawing.Point(7, 30);
```

```
        this.LinePropertiesPathIndexLabel.Name = "LinePropertiesPathIndexLabel";
        this.LinePropertiesPathIndexLabel.Size = new System.Drawing.Size(69, 15);
        this.LinePropertiesPathIndexLabel.TabIndex = 27;
        this.LinePropertiesPathIndexLabel.Text = "Path Index: ";
        //
        // LinePropertiesKeyFramesTextBox
        //
        this.LinePropertiesKeyFramesTextBox.BackColor = System.Drawing.SystemColors.Desktop;
        this.LinePropertiesKeyFramesTextBox.ForeColor =
System.Drawing.SystemColors.HighlightText;
        this.LinePropertiesKeyFramesTextBox.FormattingEnabled = true;
        this.LinePropertiesKeyFramesTextBox.ItemHeight = 15;
        this.LinePropertiesKeyFramesTextBox.Location = new System.Drawing.Point(3, 48);
        this.LinePropertiesKeyFramesTextBox.Name = "LinePropertiesKeyFramesTextBox";
        this.LinePropertiesKeyFramesTextBox.Size = new System.Drawing.Size(213, 94);
        this.LinePropertiesKeyFramesTextBox.TabIndex = 19;
        this.LinePropertiesKeyFramesTextBox.SelectedIndexChanged += new
System.EventHandler(this.LinePropertiesKeyFramesTextBox_SelectedIndexChanged);
        //
        // LinePropertiesTitle
        //
        this.LinePropertiesTitle.AccessibleName = "Time Manager";
        this.LinePropertiesTitle.AutoSize = true;
        this.LinePropertiesTitle.Font = new System.Drawing.Font("Segoe UI", 15.75F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
        this.LinePropertiesTitle.ForeColor = System.Drawing.SystemColors.ControlLight;
        this.LinePropertiesTitle.Location = new System.Drawing.Point(3, 0);
        this.LinePropertiesTitle.Name = "LinePropertiesTitle";
        this.LinePropertiesTitle.Size = new System.Drawing.Size(150, 30);
        this.LinePropertiesTitle.TabIndex = 18;
        this.LinePropertiesTitle.Text = "Line Properties";
```

```
        //
        // InformationPanel
        //
        this.InformationPanel.Anchor =
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
        | System.Windows.Forms.AnchorStyles.Left)
        | System.Windows.Forms.AnchorStyles.Right)));
        this.InformationPanel.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
        this.InformationPanel.Controls.Add(this.InformationPreviewModeData);
        this.InformationPanel.Controls.Add(this.InformationConnectedDACCount);
        this.InformationPanel.Controls.Add(this.InformationClosestPointData);
        this.InformationPanel.Controls.Add(this.InformationClosestPointLabel);
        this.InformationPanel.Controls.Add(this.InformationFrameListCountInfo);
        this.InformationPanel.Controls.Add(this.InformationDynamicListCountInfo);
        this.InformationPanel.Controls.Add(this.InformationFrameListCountLabel);
        this.InformationPanel.Controls.Add(this.InformationDynamicListCountLabel);
        this.InformationPanel.Controls.Add(this.InformationPoint2Info);
        this.InformationPanel.Controls.Add(this.InformationPoint1Info);
        this.InformationPanel.Controls.Add(this.InformationPoint2TitleLabel);
        this.InformationPanel.Controls.Add(this.InformationPoint1TitleLablel);
        this.InformationPanel.Controls.Add(this.InformationTitleLabel);
        this.InformationPanel.Location = new System.Drawing.Point(4, 316);
        this.InformationPanel.Name = "InformationPanel";
        this.InformationPanel.Size = new System.Drawing.Size(404, 156);
        this.InformationPanel.TabIndex = 19;
        //
        // InformationPreviewModeData
        //
        this.InformationPreviewModeData.AutoSize = true;
        this.InformationPreviewModeData.ForeColor = System.Drawing.SystemColors.Control;
```

```
this.InformationPreviewModeData.Location = new System.Drawing.Point(303, 56);
this.InformationPreviewModeData.Name = "InformationPreviewModeData";
this.InformationPreviewModeData.Size = new System.Drawing.Size(27, 15);
this.InformationPreviewModeData.TabIndex = 30;
this.InformationPreviewModeData.Text = "null";
//
// InformationConnectedDACCount
//
this.InformationConnectedDACCount.AutoSize = true;
this.InformationConnectedDACCount.ForeColor = System.Drawing.SystemColors.Control;
this.InformationConnectedDACCount.Location = new System.Drawing.Point(197, 56);
this.InformationConnectedDACCount.Name = "InformationConnectedDACCount";
this.InformationConnectedDACCount.Size = new System.Drawing.Size(100, 15);
this.InformationConnectedDACCount.TabIndex = 29;
this.InformationConnectedDACCount.Text = "Connected DACs:";
//
// InformationClosestPointData
//
this.InformationClosestPointData.AutoSize = true;
this.InformationClosestPointData.ForeColor = System.Drawing.SystemColors.Control;
this.InformationClosestPointData.Location = new System.Drawing.Point(279, 41);
this.InformationClosestPointData.Name = "InformationClosestPointData";
this.InformationClosestPointData.Size = new System.Drawing.Size(27, 15);
this.InformationClosestPointData.TabIndex = 28;
this.InformationClosestPointData.Text = "null";
//
// InformationClosestPointLabel
//
this.InformationClosestPointLabel.AutoSize = true;
this.InformationClosestPointLabel.ForeColor = System.Drawing.SystemColors.Control;
```

```
this.InformationClosestPointLabel.Location = new System.Drawing.Point(197, 41);
this.InformationClosestPointLabel.Name = "InformationClosestPointLabel";
this.InformationClosestPointLabel.Size = new System.Drawing.Size(76, 15);
this.InformationClosestPointLabel.TabIndex = 27;
this.InformationClosestPointLabel.Text = "ClosestPoint:";
//
// InformationFrameListCountInfo
//
this.InformationFrameListCountInfo.AutoSize = true;
this.InformationFrameListCountInfo.ForeColor = System.Drawing.SystemColors.Control;
this.InformationFrameListCountInfo.Location = new System.Drawing.Point(134, 86);
this.InformationFrameListCountInfo.Name = "InformationFrameListCountInfo";
this.InformationFrameListCountInfo.Size = new System.Drawing.Size(13, 15);
this.InformationFrameListCountInfo.TabIndex = 26;
this.InformationFrameListCountInfo.Text = "0";
//
// InformationDynamicListCountInfo
//
this.InformationDynamicListCountInfo.AutoSize = true;
this.InformationDynamicListCountInfo.ForeColor = System.Drawing.SystemColors.Control;
this.InformationDynamicListCountInfo.Location = new System.Drawing.Point(134, 71);
this.InformationDynamicListCountInfo.Name = "InformationDynamicListCountInfo";
this.InformationDynamicListCountInfo.Size = new System.Drawing.Size(13, 15);
this.InformationDynamicListCountInfo.TabIndex = 25;
this.InformationDynamicListCountInfo.Text = "0";
//
// InformationFrameListCountLabel
//
this.InformationFrameListCountLabel.AutoSize = true;
this.InformationFrameListCountLabel.ForeColor = System.Drawing.SystemColors.Control;
```

```
this.InformationFrameListCountLabel.Location = new System.Drawing.Point(14, 86);
this.InformationFrameListCountLabel.Name = "InformationFrameListCountLabel";
this.InformationFrameListCountLabel.Size = new System.Drawing.Size(100, 15);
this.InformationFrameListCountLabel.TabIndex = 24;
this.InformationFrameListCountLabel.Text = "Frame List Count:";
//
// InformationDynamicListCountLabel
//
this.InformationDynamicListCountLabel.AutoSize = true;
this.InformationDynamicListCountLabel.ForeColor = System.Drawing.SystemColors.Control;
this.InformationDynamicListCountLabel.Location = new System.Drawing.Point(14, 71);
this.InformationDynamicListCountLabel.Name = "InformationDynamicListCountLabel";
this.InformationDynamicListCountLabel.Size = new System.Drawing.Size(114, 15);
this.InformationDynamicListCountLabel.TabIndex = 23;
this.InformationDynamicListCountLabel.Text = "Dynamic List Count:";
//
// InformationPoint2Info
//
this.InformationPoint2Info.AutoSize = true;
this.InformationPoint2Info.ForeColor = System.Drawing.SystemColors.Control;
this.InformationPoint2Info.Location = new System.Drawing.Point(57, 56);
this.InformationPoint2Info.Name = "InformationPoint2Info";
this.InformationPoint2Info.Size = new System.Drawing.Size(27, 15);
this.InformationPoint2Info.TabIndex = 22;
this.InformationPoint2Info.Text = "null";
//
// InformationPoint1Info
//
this.InformationPoint1Info.AutoSize = true;
this.InformationPoint1Info.ForeColor = System.Drawing.SystemColors.Control;
```

```
this.InformationPoint1Info.Location = new System.Drawing.Point(57, 41);
this.InformationPoint1Info.Name = "InformationPoint1Info";
this.InformationPoint1Info.Size = new System.Drawing.Size(27, 15);
this.InformationPoint1Info.TabIndex = 21;
this.InformationPoint1Info.Text = "null";
//
// InformationPoint2TitleLabel
//
this.InformationPoint2TitleLabel.AutoSize = true;
this.InformationPoint2TitleLabel.ForeColor = System.Drawing.SystemColors.Control;
this.InformationPoint2TitleLabel.Location = new System.Drawing.Point(14, 56);
this.InformationPoint2TitleLabel.Name = "InformationPoint2TitleLabel";
this.InformationPoint2TitleLabel.Size = new System.Drawing.Size(47, 15);
this.InformationPoint2TitleLabel.TabIndex = 20;
this.InformationPoint2TitleLabel.Text = "Point2: ";
//
// InformationPoint1TitleLablel
//
this.InformationPoint1TitleLablel.AutoSize = true;
this.InformationPoint1TitleLablel.ForeColor = System.Drawing.SystemColors.Control;
this.InformationPoint1TitleLablel.Location = new System.Drawing.Point(14, 41);
this.InformationPoint1TitleLablel.Name = "InformationPoint1TitleLablel";
this.InformationPoint1TitleLablel.Size = new System.Drawing.Size(47, 15);
this.InformationPoint1TitleLablel.TabIndex = 19;
this.InformationPoint1TitleLablel.Text = "Point1: ";
//
// InformationTitleLabel
//
this.InformationTitleLabel.AccessibleName = "Time Manager";
this.InformationTitleLabel.AutoSize = true;
```

```
        this.InformationTitleLabel.Font = new System.Drawing.Font("Segoe UI", 15.75F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
        this.InformationTitleLabel.ForeColor = System.Drawing.SystemColors.ControlLight;
        this.InformationTitleLabel.Location = new System.Drawing.Point(3, 2);
        this.InformationTitleLabel.Name = "InformationTitleLabel";
        this.InformationTitleLabel.Size = new System.Drawing.Size(122, 30);
        this.InformationTitleLabel.TabIndex = 18;
        this.InformationTitleLabel.Text = "Information";
        //
        // DrawerColorDialog
        //
        this.DrawerColorDialog.Color = System.Drawing.Color.White;
        //
        // LinePropertiesColorDialog
        //
        this.LinePropertiesColorDialog.Color = System.Drawing.Color.White;
        //
        // menuStrip1
        //
        this.menuStrip1.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(0)))),
((int)(((byte)(0)))), ((int)(((byte)(64)))));
        this.menuStrip1.ImageScalingSize = new System.Drawing.Size(28, 28);
        this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
        this.fileToolStripMenuItem,
        this.showToolStripMenuItem,
        this.settingsToolStripMenuItem});
        this.menuStrip1.Location = new System.Drawing.Point(0, 0);
        this.menuStrip1.Name = "menuStrip1";
        this.menuStrip1.Size = new System.Drawing.Size(1015, 24);
        this.menuStrip1.TabIndex = 20;
        this.menuStrip1.Text = "menuStrip1";
```

```
        //
        // fileToolStripMenuItem
        //
        this.fileToolStripMenuItem.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(0)))),
((int)(((byte)(0)))), ((int)(((byte)(64)))));
        this.fileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.newPathProjectToolStripMenuItem,
        this.openToolStripMenuItem,
        this.saveToolStripMenuItem});
        this.fileToolStripMenuItem.ForeColor = System.Drawing.Color.White;
        this.fileToolStripMenuItem.Name = "fileToolStripMenuItem";
        this.fileToolStripMenuItem.Size = new System.Drawing.Size(37, 20);
        this.fileToolStripMenuItem.Text = "File";
        //
        // newPathProjectToolStripMenuItem
        //
        this.newPathProjectToolStripMenuItem.BackColor = System.Drawing.SystemColors.Control;
        this.newPathProjectToolStripMenuItem.ForeColor =
System.Drawing.SystemColors.ControlText;
        this.newPathProjectToolStripMenuItem.Name = "newPathProjectToolStripMenuItem";
        this.newPathProjectToolStripMenuItem.Size = new System.Drawing.Size(165, 22);
        this.newPathProjectToolStripMenuItem.Text = "New Path Project";
        //
        // openToolStripMenuItem
        //
        this.openToolStripMenuItem.Name = "openToolStripMenuItem";
        this.openToolStripMenuItem.Size = new System.Drawing.Size(165, 22);
        this.openToolStripMenuItem.Text = "Open Path File";
        this.openToolStripMenuItem.Click += new
System.EventHandler(this.openToolStripMenuItem_Click);
        //
```

```
        // saveToolStripMenuItem
        //
        this.saveToolStripMenuItem.Name = "saveToolStripMenuItem";
        this.saveToolStripMenuItem.Size = new System.Drawing.Size(165, 22);
        this.saveToolStripMenuItem.Text = "Save Path File As";
        this.saveToolStripMenuItem.Click += new
System.EventHandler(this.saveToolStripMenuItem_Click);
        //
        // showToolStripMenuItem
        //
        this.showToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.laserPathToolStripMenuItem,
        this.laserPointsToolStripMenuItem,
        this.previewShapesToolStripMenuItem});
        this.showToolStripMenuItem.ForeColor = System.Drawing.SystemColors.Control;
        this.showToolStripMenuItem.Name = "showToolStripMenuItem";
        this.showToolStripMenuItem.Size = new System.Drawing.Size(48, 20);
        this.showToolStripMenuItem.Text = "Show";
        //
        // laserPathToolStripMenuItem
        //
        this.laserPathToolStripMenuItem.CheckOnClick = true;
        this.laserPathToolStripMenuItem.Name = "laserPathToolStripMenuItem";
        this.laserPathToolStripMenuItem.Size = new System.Drawing.Size(155, 22);
        this.laserPathToolStripMenuItem.Text = "Laser Path";
        //
        // laserPointsToolStripMenuItem
        //
        this.laserPointsToolStripMenuItem.CheckOnClick = true;
        this.laserPointsToolStripMenuItem.Name = "laserPointsToolStripMenuItem";
```

```
        this.laserPointsToolStripMenuItem.Size = new System.Drawing.Size(155, 22);
        this.laserPointsToolStripMenuItem.Text = "Laser Points";
        //
        // previewShapesToolStripMenuItem
        //
        this.previewShapesToolStripMenuItem.Checked = true;
        this.previewShapesToolStripMenuItem.CheckOnClick = true;
        this.previewShapesToolStripMenuItem.CheckState =
System.Windows.Forms.CheckState.Checked;
        this.previewShapesToolStripMenuItem.Name = "previewShapesToolStripMenuItem";
        this.previewShapesToolStripMenuItem.Size = new System.Drawing.Size(155, 22);
        this.previewShapesToolStripMenuItem.Text = "Preview Shapes";
        //
        // settingsToolStripMenuItem
        //
        this.settingsToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.sToolStripMenuItem,
        this.dacSettingsToolStripMenuItem});
        this.settingsToolStripMenuItem.ForeColor = System.Drawing.SystemColors.Control;
        this.settingsToolStripMenuItem.Name = "settingsToolStripMenuItem";
        this.settingsToolStripMenuItem.Size = new System.Drawing.Size(61, 20);
        this.settingsToolStripMenuItem.Text = "Settings";
        //
        // sToolStripMenuItem
        //
        this.sToolStripMenuItem.Name = "sToolStripMenuItem";
        this.sToolStripMenuItem.Size = new System.Drawing.Size(180, 22);
        this.sToolStripMenuItem.Text = "Laser Settings";
        this.sToolStripMenuItem.Click += new System.EventHandler(this.ToolStripMenuItem_Click);
        //
```

```
        // dacSettingsToolStripMenuItem
        //
        this.dacSettingsToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
        this.connectToDACToolStripMenuItem,
        this.disconnectDACToolStripMenuItem});
        this.dacSettingsToolStripMenuItem.Name = "dacSettingsToolStripMenuItem";
        this.dacSettingsToolStripMenuItem.Size = new System.Drawing.Size(180, 22);
        this.dacSettingsToolStripMenuItem.Text = "DAC Settings";
        //
        // connectToDACToolStripMenuItem
        //
        this.connectToDACToolStripMenuItem.Name = "connectToDACToolStripMenuItem";
        this.connectToDACToolStripMenuItem.Size = new System.Drawing.Size(180, 22);
        this.connectToDACToolStripMenuItem.Text = "Connect To DAC";
        this.connectToDACToolStripMenuItem.Click += new
System.EventHandler(this.connectToDACToolStripMenuItem_Click);
        //
        // disconnectDACToolStripMenuItem
        //
        this.disconnectDACToolStripMenuItem.Name = "disconnectDACToolStripMenuItem";
        this.disconnectDACToolStripMenuItem.Size = new System.Drawing.Size(180, 22);
        this.disconnectDACToolStripMenuItem.Text = "Disconnect DAC";
        this.disconnectDACToolStripMenuItem.Click += new
System.EventHandler(this.disconnectDACToolStripMenuItem_Click);
        //
        // splitContainer1
        //
        this.splitContainer1.Anchor =
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
        | System.Windows.Forms.AnchorStyles.Left)
```

```
        | System.Windows.Forms.AnchorStyles.Right)));
        this.splitContainer1.Location = new System.Drawing.Point(12, 27);
        this.splitContainer1.Name = "splitContainer1";
        //
        // splitContainer1.Panel1
        //
        this.splitContainer1.Panel1.Controls.Add(this.PreviewGraphics);
        this.splitContainer1.Panel1.Controls.Add(this.TimeLinePanel);
        this.splitContainer1.Panel1.Paint += new
System.Windows.Forms.PaintEventHandler(this.splitContainer1_Panel1_Paint);
        this.splitContainer1.Panel1.Resize += new
System.EventHandler(this.PreviewGraphics_Resize);
        //
        // splitContainer1.Panel2
        //
        this.splitContainer1.Panel2.Controls.Add(this.LinePropertiesPanel);
        this.splitContainer1.Panel2.Controls.Add(this.InformationPanel);
        this.splitContainer1.Panel2.Controls.Add(this.OptionsPanel);
        this.splitContainer1.Size = new System.Drawing.Size(995, 695);
        this.splitContainer1.SplitterDistance = 579;
        this.splitContainer1.TabIndex = 21;
        this.splitContainer1.Resize += new System.EventHandler(this.PreviewGraphics_Resize);
        //
        // backgroundWorker1
        //
        this.backgroundWorker1.DoWork += new
System.ComponentModel.DoWorkEventHandler(this.backgroundWorker1_DoWork);
        //
        // backgroundWorker2
        //
        this.backgroundWorker2.DoWork += new
System.ComponentModel.DoWorkEventHandler(this.backgroundWorker2_DoWork);
```

```
//
// PathMainWindow
//
this.AccessibleName = "Path Window";
this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 15F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.BackColor = System.Drawing.SystemColors.Desktop;
this.ClientSize = new System.Drawing.Size(1015, 729);
this.Controls.Add(this.splitContainer1);
this.Controls.Add(this.menuStrip1);
this.MainMenuStrip = this.menuStrip1;
this.MinimizeBox = false;
this.Name = "PathMainWindow";
this.Text = "Path";
this.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.Form1_KeyPress);
((System.ComponentModel.ISupportInitialize)(this.PreviewGraphics)).EndInit();
this.OptionsPanel.ResumeLayout(false);
this.OptionsPanel.PerformLayout();
this.TimeLinePanel.ResumeLayout(false);
this.TimeLinePanel.PerformLayout();
((System.ComponentModel.ISupportInitialize)(this.projectMaxTimeSelector)).EndInit();
this.timelineGUIHugger.ResumeLayout(false);
((System.ComponentModel.ISupportInitialize)(this.timelineGUI)).EndInit();
this.LinePropertiesPanel.ResumeLayout(false);
this.LinePropertiesPanel.PerformLayout();
((System.ComponentModel.ISupportInitialize)(this.LinePropertiesYCoordinate)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.LinePropertiesXCoordinate)).EndInit();
this.linePropertiesStrobeSettings.ResumeLayout(false);
this.linePropertiesFrameControlsPanel.ResumeLayout(false);
this.InformationPanel.ResumeLayout(false);
```

```
        this.InformationPanel.PerformLayout();
        this.menuStrip1.ResumeLayout(false);
        this.menuStrip1.PerformLayout();
        this.splitContainer1.Panel1.ResumeLayout(false);
        this.splitContainer1.Panel2.ResumeLayout(false);
        ((System.ComponentModel.ISupportInitialize)(this.splitContainer1)).EndInit();
        this.splitContainer1.ResumeLayout(false);
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion

    private PictureBox PreviewGraphics;
    private Panel OptionsPanel;
    private Panel TimeLinePanel;
    private Button TimeLineNextFrame;
    private Button TimeLineBackFrame;
    private Label TimeLineTitleLabel;
    private CheckBox TimeLinePlay;
    private TextBox OptionsKPPSTextBox;
    private TextBox OptionsFPSTextBox;
    private Label OptionsKPPSLabel;
    private Label OptionsTitleLabel;
    private Label OptionsFPSLabel;
    private TextBox TimeLineFramesInput;
    private TextBox TimeLineSecondsInput;
    private Label TimeLineFramesLabel;
    private Label TimeLineSecondsLabel;
```

```
private Button OptionsLoadPathFile;
private Button OptionsSavePathFile;
private CheckBox OptionsDrawLineMode;
private CheckBox OptionsSelectModeButton;
private Label OptionsEmergencyLabel;
private CheckBox OptionsToggleProject;
private OpenFileDialog openFileDialog1;
private SaveFileDialog saveFileDialog1;
private Panel LinePropertiesPanel;
private Label LinePropertiesTitle;
private Panel InformationPanel;
private Label InformationTitleLabel;
private Label InformationPoint2Info;
private Label InformationPoint1Info;
private Label InformationPoint2TitleLabel;
private Label InformationPoint1TitleLablel;
private Button OptionsColorSelecterOpener;
private ColorDialog DrawerColorDialog;
private Label InformationFrameListCountInfo;
private Label InformationDynamicListCountInfo;
private Label InformationFrameListCountLabel;
private Label InformationDynamicListCountLabel;
private ListBox LinePropertiesKeyFramesTextBox;
private Label LinePropertiesPathIndexData;
private Label LinePropertiesPathIndexLabel;
private Button LinePropertiesChangeColor;
private ColorDialog LinePropertiesColorDialog;
private CheckBox OptionsSnapToPoint;
private Label InformationClosestPointLabel;
private Label InformationClosestPointData;
```

```
private Label InformationPreviewModeData;
private Label InformationConnectedDACCount;
private ListBox PathLinePointsListBox;
private TabControl linePropertiesStrobeSettings;
private TabPage linePropertiesFrameControlsPanel;
private TabPage tabPage2;
private Label LinePropertiesYCoordinateLabel;
private Label LinePropertiesXCoordinateLabel;
private PictureBox timelineGUI;
private MenuStrip menuStrip1;
private ToolStripMenuItem fileToolStripMenuItem;
private ToolStripMenuItem saveToolStripMenuItem;
private ToolStripMenuItem openToolStripMenuItem;
private ToolStripMenuItem newPathProjectToolStripMenuItem;
private ToolStripMenuItem settingsToolStripMenuItem;
private ToolStripMenuItem sToolStripMenuItem;
private SplitContainer splitContainer1;
private ToolStripMenuItem dacSettingsToolStripMenuItem;
private ToolStripMenuItem connectToDACToolStripMenuItem;
private ToolStripMenuItem disconnectDACToolStripMenuItem;
private Panel timelineGUIHugger;
private NumericUpDown projectMaxTimeSelector;
private ToolStripMenuItem showToolStripMenuItem;
private ToolStripMenuItem laserPathToolStripMenuItem;
private ToolStripMenuItem laserPointsToolStripMenuItem;
private ToolStripMenuItem previewShapesToolStripMenuItem;
private System.ComponentModel.BackgroundWorker backgroundWorker1;
private NumericUpDown LinePropertiesYCoordinate;
private NumericUpDown LinePropertiesXCoordinate;
private Button deleteShape;
```

```
        private System.ComponentModel.BackgroundWorker backgroundWorker2;
        private Label TimeLineMaxTimeLabel;
    }
}
```

Settings.Designer.cs

```
namespace Path
{
    partial class Settings
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
```

```
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
   this.label1 = new System.Windows.Forms.Label();
   this.label2 = new System.Windows.Forms.Label();
   this.label3 = new System.Windows.Forms.Label();
   this.label4 = new System.Windows.Forms.Label();
   this.label5 = new System.Windows.Forms.Label();
   this.label6 = new System.Windows.Forms.Label();
   this.label7 = new System.Windows.Forms.Label();
   this.label8 = new System.Windows.Forms.Label();
   this.numericUpDown1 = new System.Windows.Forms.NumericUpDown();
   this.numericUpDown2 = new System.Windows.Forms.NumericUpDown();
   this.numericUpDown3 = new System.Windows.Forms.NumericUpDown();
   this.numericUpDown4 = new System.Windows.Forms.NumericUpDown();
   this.numericUpDown5 = new System.Windows.Forms.NumericUpDown();
   this.numericUpDown6 = new System.Windows.Forms.NumericUpDown();
   this.label9 = new System.Windows.Forms.Label();
   this.label10 = new System.Windows.Forms.Label();
   this.label11 = new System.Windows.Forms.Label();
   this.label12 = new System.Windows.Forms.Label();
   this.numericUpDown7 = new System.Windows.Forms.NumericUpDown();
   this.numericUpDown8 = new System.Windows.Forms.NumericUpDown();
   this.numericUpDown9 = new System.Windows.Forms.NumericUpDown();
   this.numericUpDown10 = new System.Windows.Forms.NumericUpDown();
   ((System.ComponentModel.ISupportInitialize)(this.numericUpDown1)).BeginInit();
   ((System.ComponentModel.ISupportInitialize)(this.numericUpDown2)).BeginInit();
   ((System.ComponentModel.ISupportInitialize)(this.numericUpDown3)).BeginInit();
```

```
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown4)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown5)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown6)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown7)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown8)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown9)).BeginInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown10)).BeginInit();
        this.SuspendLayout();
        //
        // label1
        //
        this.label1.AutoSize = true;
        this.label1.Font = new System.Drawing.Font("Segoe UI", 13F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
        this.label1.Location = new System.Drawing.Point(7, 4);
        this.label1.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(155, 25);
        this.label1.TabIndex = 0;
        this.label1.Text = "Time Line Settings";
        //
        // label2
        //
        this.label2.AutoSize = true;
        this.label2.Font = new System.Drawing.Font("Segoe UI", 13F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
        this.label2.Location = new System.Drawing.Point(222, 4);
        this.label2.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
        this.label2.Name = "label2";
        this.label2.Size = new System.Drawing.Size(121, 25);
        this.label2.TabIndex = 1;
```

```
this.label2.Text = "Laser Settings";
//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(7, 34);
this.label3.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(99, 15);
this.label3.TabIndex = 2;
this.label3.Text = "Pixels Per Second";
//
// label4
//
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(7, 60);
this.label4.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(92, 15);
this.label4.TabIndex = 3;
this.label4.Text = "Pixels Per Shape";
//
// label5
//
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(7, 87);
this.label5.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(68, 15);
this.label5.TabIndex = 4;
```

```
this.label5.Text = "Left Margin";
this.label5.Click += new System.EventHandler(this.label5_Click);
//
// label6
//
this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(7, 114);
this.label6.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(67, 15);
this.label6.TabIndex = 5;
this.label6.Text = "Top Margin";
//
// label7
//
this.label7.AutoSize = true;
this.label7.Location = new System.Drawing.Point(7, 141);
this.label7.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(97, 15);
this.label7.TabIndex = 6;
this.label7.Text = "Timeline Dot Size";
//
// label8
//
this.label8.AutoSize = true;
this.label8.Location = new System.Drawing.Point(7, 168);
this.label8.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(101, 15);
```

```
        this.label8.TabIndex = 7;
        this.label8.Text = "Seconds Font Size";
        //
        // numericUpDown1
        //
        this.numericUpDown1.BackColor = System.Drawing.SystemColors.WindowText;
        this.numericUpDown1.ForeColor = System.Drawing.SystemColors.Window;
        this.numericUpDown1.Location = new System.Drawing.Point(122, 31);
        this.numericUpDown1.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
        this.numericUpDown1.Name = "numericUpDown1";
        this.numericUpDown1.Size = new System.Drawing.Size(96, 23);
        this.numericUpDown1.TabIndex = 8;
        this.numericUpDown1.ValueChanged += new
System.EventHandler(this.numericUpDown1_ValueChanged);
        //
        // numericUpDown2
        //
        this.numericUpDown2.BackColor = System.Drawing.SystemColors.WindowText;
        this.numericUpDown2.ForeColor = System.Drawing.SystemColors.Window;
        this.numericUpDown2.Location = new System.Drawing.Point(122, 58);
        this.numericUpDown2.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
        this.numericUpDown2.Name = "numericUpDown2";
        this.numericUpDown2.Size = new System.Drawing.Size(96, 23);
        this.numericUpDown2.TabIndex = 9;
        //
        // numericUpDown3
        //
        this.numericUpDown3.BackColor = System.Drawing.SystemColors.WindowText;
        this.numericUpDown3.ForeColor = System.Drawing.SystemColors.Window;
        this.numericUpDown3.Location = new System.Drawing.Point(122, 85);
        this.numericUpDown3.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
```

```
this.numericUpDown3.Name = "numericUpDown3";
this.numericUpDown3.Size = new System.Drawing.Size(96, 23);
this.numericUpDown3.TabIndex = 10;
//
// numericUpDown4
//
this.numericUpDown4.BackColor = System.Drawing.SystemColors.WindowText;
this.numericUpDown4.ForeColor = System.Drawing.SystemColors.Window;
this.numericUpDown4.Location = new System.Drawing.Point(122, 112);
this.numericUpDown4.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.numericUpDown4.Name = "numericUpDown4";
this.numericUpDown4.Size = new System.Drawing.Size(96, 23);
this.numericUpDown4.TabIndex = 11;
//
// numericUpDown5
//
this.numericUpDown5.BackColor = System.Drawing.SystemColors.WindowText;
this.numericUpDown5.ForeColor = System.Drawing.SystemColors.Window;
this.numericUpDown5.Location = new System.Drawing.Point(122, 139);
this.numericUpDown5.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.numericUpDown5.Name = "numericUpDown5";
this.numericUpDown5.Size = new System.Drawing.Size(96, 23);
this.numericUpDown5.TabIndex = 12;
//
// numericUpDown6
//
this.numericUpDown6.BackColor = System.Drawing.SystemColors.WindowText;
this.numericUpDown6.ForeColor = System.Drawing.SystemColors.Window;
this.numericUpDown6.Location = new System.Drawing.Point(122, 166);
this.numericUpDown6.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
```

```
this.numericUpDown6.Name = "numericUpDown6";
this.numericUpDown6.Size = new System.Drawing.Size(96, 23);
this.numericUpDown6.TabIndex = 13;
//
// label9
//
this.label9.AutoSize = true;
this.label9.Location = new System.Drawing.Point(225, 34);
this.label9.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(33, 15);
this.label9.TabIndex = 14;
this.label9.Text = "Kpps";
//
// label10
//
this.label10.AutoSize = true;
this.label10.Location = new System.Drawing.Point(225, 61);
this.label10.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label10.Name = "label10";
this.label10.Size = new System.Drawing.Size(106, 15);
this.label10.TabIndex = 15;
this.label10.Text = "Maximum Velocity";
//
// label11
//
this.label11.AutoSize = true;
this.label11.Location = new System.Drawing.Point(225, 88);
this.label11.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label11.Name = "label11";
```

```
this.label11.Size = new System.Drawing.Size(131, 15);
this.label11.TabIndex = 16;
this.label11.Text = "Maximum Acceleration";
//
// label12
//
this.label12.AutoSize = true;
this.label12.Location = new System.Drawing.Point(225, 115);
this.label12.Margin = new System.Windows.Forms.Padding(2, 0, 2, 0);
this.label12.Name = "label12";
this.label12.Size = new System.Drawing.Size(103, 15);
this.label12.TabIndex = 17;
this.label12.Text = "Dwell Point Count";
//
// numericUpDown7
//
this.numericUpDown7.BackColor = System.Drawing.SystemColors.WindowText;
this.numericUpDown7.ForeColor = System.Drawing.SystemColors.Window;
this.numericUpDown7.Location = new System.Drawing.Point(364, 32);
this.numericUpDown7.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.numericUpDown7.Name = "numericUpDown7";
this.numericUpDown7.Size = new System.Drawing.Size(96, 23);
this.numericUpDown7.TabIndex = 18;
//
// numericUpDown8
//
this.numericUpDown8.BackColor = System.Drawing.SystemColors.WindowText;
this.numericUpDown8.ForeColor = System.Drawing.SystemColors.Window;
this.numericUpDown8.Location = new System.Drawing.Point(364, 59);
this.numericUpDown8.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
```

```
this.numericUpDown8.Name = "numericUpDown8";
this.numericUpDown8.Size = new System.Drawing.Size(96, 23);
this.numericUpDown8.TabIndex = 19;
//
// numericUpDown9
//
this.numericUpDown9.BackColor = System.Drawing.SystemColors.WindowText;
this.numericUpDown9.ForeColor = System.Drawing.SystemColors.Window;
this.numericUpDown9.Location = new System.Drawing.Point(364, 86);
this.numericUpDown9.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.numericUpDown9.Name = "numericUpDown9";
this.numericUpDown9.Size = new System.Drawing.Size(96, 23);
this.numericUpDown9.TabIndex = 20;
//
// numericUpDown10
//
this.numericUpDown10.BackColor = System.Drawing.SystemColors.WindowText;
this.numericUpDown10.ForeColor = System.Drawing.SystemColors.Window;
this.numericUpDown10.Location = new System.Drawing.Point(364, 113);
this.numericUpDown10.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.numericUpDown10.Name = "numericUpDown10";
this.numericUpDown10.Size = new System.Drawing.Size(96, 23);
this.numericUpDown10.TabIndex = 21;
//
// Settings
//
this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 15F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.BackColor = System.Drawing.SystemColors.ControlText;
this.ClientSize = new System.Drawing.Size(467, 225);
```

```
this.Controls.Add(this.numericUpDown10);
this.Controls.Add(this.numericUpDown9);
this.Controls.Add(this.numericUpDown8);
this.Controls.Add(this.numericUpDown7);
this.Controls.Add(this.label12);
this.Controls.Add(this.label11);
this.Controls.Add(this.label10);
this.Controls.Add(this.label9);
this.Controls.Add(this.numericUpDown6);
this.Controls.Add(this.numericUpDown5);
this.Controls.Add(this.numericUpDown4);
this.Controls.Add(this.numericUpDown3);
this.Controls.Add(this.numericUpDown2);
this.Controls.Add(this.numericUpDown1);
this.Controls.Add(this.label8);
this.Controls.Add(this.label7);
this.Controls.Add(this.label6);
this.Controls.Add(this.label5);
this.Controls.Add(this.label4);
this.Controls.Add(this.label3);
this.Controls.Add(this.label2);
this.Controls.Add(this.label1);
this.ForeColor = System.Drawing.SystemColors.Control;
this.Margin = new System.Windows.Forms.Padding(2, 2, 2, 2);
this.Name = "Settings";
this.Text = "Settings";
((System.ComponentModel.ISupportInitialize)(this.numericUpDown1)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.numericUpDown2)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.numericUpDown3)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.numericUpDown4)).EndInit();
```

```
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown5)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown6)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown7)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown8)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown9)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericUpDown10)).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion

    private Label label1;
    private Label label2;
    private Label label3;
    private Label label4;
    private Label label5;
    private Label label6;
    private Label label7;
    private Label label8;
    private NumericUpDown numericUpDown1;
    private NumericUpDown numericUpDown2;
    private NumericUpDown numericUpDown3;
    private NumericUpDown numericUpDown4;
    private NumericUpDown numericUpDown5;
    private NumericUpDown numericUpDown6;
    private Label label9;
    private Label label10;
    private Label label11;
```

```
        private Label label12;
        private NumericUpDown numericUpDown7;
        private NumericUpDown numericUpDown8;
        private NumericUpDown numericUpDown9;
        private NumericUpDown numericUpDown10;
    }
}
```

Path.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net6.0-windows</TargetFramework>
    <Nullable>enable</Nullable>
    <UseWindowsForms>true</UseWindowsForms>
    <ImplicitUsings>enable</ImplicitUsings>
    <Platforms>AnyCPU;x64;ARM64;x86</Platforms>
  </PropertyGroup>

  <ItemGroup>
    <Compile Remove="Form1.Designer-Leos-PC.cs" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.CodeAnalysis.CSharp" Version="4.3.1" />
    <PackageReference Include="Newtonsoft.Json" Version="13.0.2" />
  </ItemGroup>

  <ItemGroup>
    <Reference Include="HeliosLaserDACLeosPart">
      <HintPath>\1Prototype \versionday Open HeliosLaserDAC.dll</HintPath>
      <EmbedInteropTypes>False</EmbedInteropTypes>
    </Reference>
  </ItemGroup>

  <ItemGroup>
    <Compile Update="Properties\Resources.Designer.cs">
      <DesignTime>True</DesignTime>
      <AutoGen>True</AutoGen>
      <DependentUpon>Resources.resx</DependentUpon>
    </Compile>
    <Compile Update="Properties\Settings.Designer.cs">
      <DesignTimeSharedInput>True</DesignTimeSharedInput>
      <AutoGen>True</AutoGen>
      <DependentUpon>Settings.settings</DependentUpon>
    </Compile>
  </ItemGroup>
```

```
  <ItemGroup>
    <EmbeddedResource Update="Properties\Resources.resx">
      <Generator>ResXFileCodeGenerator</Generator>
      <LastGenOutput>Resources.Designer.cs</LastGenOutput>
    </EmbeddedResource>
  </ItemGroup>

  <ItemGroup>
    <None Update="Properties\Settings.settings">
      <Generator>SettingsSingleFileGenerator</Generator>
      <LastGenOutput>Settings.Designer.cs</LastGenOutput>
    </None>
  </ItemGroup>

</Project>
```