EG UK Theory and Practice of Computer Graphics (2006)
M. McDerby, L. Lever (Editors)

Visualisation of DNA - Does the Soul Have a Picture?

M. Middleton and Y. Yue

University of Luton, UK

Abstract

Although there has been a great deal of research on DNA, DNA visualisation has been long overlooked. This paper studies current techniques for 2-dimensional and 3-dimensional DNA visualisation and other relevant areas such as music and fractals. The paper then presents a new approach to displaying DNA data. An algorithm is designed and implemented with OpenGL to manipulate and display unique DNA strings. Unique colourings are applied to each part of the string. The output is customised and offers the option to animate the DNA string over time. The results are examined and future areas of work recommended.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Display Algorithms

1. Introduction

Computer visualisation is a growing area of research, with advances in higher resolutions, real-time rendering and application specific simulations. DNA visualisation is an exciting and complicated part of computer visualisation - from the recognisable two-dimensional printouts seen in courtrooms and television programs, to the more common three-dimensional representation of a corkscrew type entity. Whilst both of these forms provide acceptable effects of visualisation and represent the required data well, little work has been under taking in creating a more aesthetic visualisation of DNA. As we know, the amount of information held on human DNA is massive, and if applied to the correct algorithm, could provide an image, or set of images, uniquely from the DNA data.

This paper reviews current work on DNA visualisation and other relevant areas such as music and fractals. The paper then presents a new approach to displaying DNA data. An algorithm is designed and implemented with TAO OpenGL [TAO05] to manipulate and display unique DNA strings. Unique colourings are applied to each part of the string. The output is customised and offers the option to animate the DNA string over time. The results are examined and future areas of work recommended.

2. A literature review

This review examines the background information required for DNA visualisation and related research.

2.1 DNA

DNA (Deoxyribonucleic acid) which was first discovered in 1869 by a German chemist named Friedrich Miescher, is a substance contained in the nucleus of all cells within a living organism. Miescher believed that he had discovered a new biological substance, which he called 'nuclein', later becoming 'nucleic acid' because of its acidic properties. It was 50 years later that another biochemist, P.A. Levene, determined that DNA consisted of substances known as 'bases'. These bases can be split into two groups – purines (adenine (**A**) and guanine (**G**)) and pyramidines (thymine (**T**) and cytosine (**C**)) [RJ05].

In DNA, the purines and pyramidines are connected with hydrogen bonds. It is important to note that a purine must be bonded with its associated pyramidine – more specifically, **A** must be bonded with **T** and **G** with **C**. These bonds are referred to as base-pairs (bp). The fact that each purine must be bonded with its associated pyramidine means that should the DNA be damaged, replication could repair the damage. The DNA substance obtained from an organism can contain millions of base pairs (the DNA of the bacterium *N. meningitides* contains approximately 2.2 million bp, human DNA in excess of 3 billion bp [WWFT03]), which when gathered together make up what is known as a 'genome'. This sequence is the 'blueprint' for the organism in which it is contained. Ridley describes the enormousness of DNA in a more appealing way in his book "Genome":

"*This is a gigantic document, an immense book, a recipe of extravagant length, and it all fits inside the microscopic nucleus of a tiny cell that fits easily upon the head of a pin*" [Rid00].

The genome for any organism can also be broken down further into chromosomes, a collection of base-pairs which can vary in length, from a few hundred bp to several thousand or million bp. An organism is made up of a set number of chromosomes (human DNA, for example, contains exactly 23 chromosomes), and one extra or one less chromosome can lead to mutations and diseases – indeed, it is now known that downs syndrome is caused by the manifestation of an extra chromosome [Rid00].

The Human Genome Project (HGP) which aims to identify all the genes contained within the human genome, came about from work carried out by Alfred Sturtevant in 1911. Sturtevant realised that he was able to note down and map the locations of certain genes of the fruit fly [Hum05]. It is hoped that all the genes in the human genome may be mapped out – their lengths, locations within the DNA string and function - and have been able to identify that

there are somewhere between 30,000 and 40,000 separate genes.

2.2 DNA sequencing

DNA sequencing is a technique used to gather the information contained within a DNA string. The first and most basic technique for DNA sequencing was developed by Frederick Sanger in 1975 [Pot96]. Sanger's technique involved attaching fluorescent markers to nucleotides, allowing replication to take place, and running the sample through electrophoresis gel which sorted the sample and displayed the nucleotide positions. This gel can then be analysed with a computer, and the fluorescent marker positions can be stored. The sequence of marker positions gives us the raw DNA string sequence. The sequence would look to the human eye as just a random *series of letters* ('GTCCGATACGTTA…'), but as an entire *word*, this sequence contains an immense amount of information. For example, imagining the human genome to contain exactly 3 billion bp, which is to be stored as a simple ASCII text file on a computer hard disk, the file would be approximately 2.79 Gb in size. This sequence is only *one side* of the genome, or half of the base pairs, so a full raw DNA data string for the human genome will double this figure, taking it to well over 5.5 Gb of space.

2.3 DNA visualisation

A notable amount of DNA visualisation work has been used in areas trying to compare DNA strings. For example, DNA pattern matching is used in forensic science labs for police forces across the world. One such way is that developed by Gibbs and McIntyre [GM70], a 2-dimensional scatter graph like view. It works by comparing two sequences, and if the sequences match, a dot is placed at the predefined coordinates on the plot area. If the final output produces an unbroken diagonal line, it can be considered a match by the user. Although much work has been done through the years to improve the accuracy of this sequence matching, little has been done to make the output more pleasing to the eye. The sequence of dots can be confusing at first sight and in the case of forensic evidence used in court, any doubt must be avoided.

One 3-dimensional visualisation of DNA is the common double helix view. It displays the base pairs along a line which resembles a stretched and twisted ladder. This view is in fact what the structure of a DNA molecule looks like. The British chemist Rosalind Franklin carried out work using X-ray diffraction methods to analyse the structure of a DNA molecule. She suggested the structure to be that of a helix or corkscrew, and further work by James Watson and Francis Crick in 1953 confirmed this suggestion [RJ05]. If there are over 3 billion base pairs in human DNA [WWFT03], this 'ladder' is immensely long and twisted. Work carried out by Herisson and Gherbi [HG01], looked into predicting the twists and turns made by the DNA helix, and analysing the spatial properties of a DNA sequence. They built on Bolshoy et al's figures of 'wedge angles' (each base pair in the sequence being a wedge) [HG01]. By analysing the raw string sequence of letters, 2 letters at a time, the 16 combinations of letters lead to 16 various angles of trajectory, which the wedge points toward. Herisson and Gherbi [HG01] created a software tool, ADN-Viewer, which could display the 3-dimensional representation of a DNA sequence at various detail levels – the view can consist of any finite number of base-pairs, and provides a realistic and accurate view of the physical structure of the DNA sequence. However, when a large sequence is displayed as a whole, comparisons between data sets would be more than difficult, given the volume of information contained in the view.

Another, more recent, and perhaps more relevant example of DNA visualisation, is the work carried out by Wong et al [WWFT03]. Their approach works by feeding in raw genomic data, and displaying the string on screen as a sequence of pixels. Each letter representing the purines and pyramidines is allocated a colour and the string is literally printed on screen. The result was a flat, almost random display of pixels of various colours. They then applied a number of digital imaging techniques to the images, such as applying a Gaussian blur, and adjusting the various colour component values. By applying these techniques, they were able to produce some rather eye pleasing images – certainly much more pleasing than the common 2-dimensional dot representation described above. However, these results did not provide the spatial information obtained from Herisson and Gherbi's method – indeed, Herisson writes: "… it appears essential to design software tools focused on the representation, visualisation and interactive exploration of the three-dimensional information of DNA" [HG01].

It is worth pointing out, that all of the above described techniques are static images. We have seen that DNA can be displayed as a 2-dimensional representation as a dot scatter plot, or coloured pixel array, and the 3-dimensional visualisation of DNA which resembles its physical structure. As we know, DNA is made up of an immense amount of information, so why not add a fourth dimension, time, based on this information. As DNA data describes how organisms are created, grow and evolve over time, can we not create, grow and evolve the visualisations that we get from the data?

2.4 Other related visualisation

Another relevant area about 2-dimensional and 3-dimensional visualisations is in the realm of music visualisation. As online Internet use expands and the downloading of music becomes more popular, more and more people are starting to use their computers as media players. There are several popular media players for PCs: MS MediaPlayer, RealPlayer and Winamp [Win06]. They each have their own features, some supporting different or exclusive file types, and all offer the ability to display visualisations. The player with the most flexibility towards these visualisations is Winamp.

Winamp was created by Justin Frankel in 1997 [Win06], starting off as a simple MIDI and MP3 file player, which has continued to grow by adding more and more features and options. One such feature is the ability to program plugins, allowing not only the real-time rendering of 2-dimensional and 3-dimensional visualisations, which move and react in time to the current playing media file, but also the ability to change other areas of the core program, in a truly object-oriented way. By downloading

the small software development kit from the homepage, users can program their own plugins to change the look and operation of Winamp using C++.

An early 2-dimensional visualisation plugin made available, was 'Geiss', programmed by Ryan Geiss [Rya06]. It displayed the image of a single 2-dimensional line, which bounced and reacted to the various frequencies of the playing music. The effect was similar to an oscillator with a high release time (or 'slowed' effect), which when viewed over time, seemed to be moving in time with the sound. By analysing the output frequencies of the playing sound, the line could be moved, as well as changing colour or applying a warped effect, all in real-time. The plugin proved to be popular with Winamp users, with over 3.2 million downloads of its current iteration 'Geiss 4.24' to date, and spawned a number of similar visualisation plugins by other programmers. Ryan Geiss has continued to work on a number of other visualisation plugins: 'Monkey', which generates an adjustable, moving 3-dimensional cave based on the sound played; 'Smoke', which generates 3-dimensional smoke, based on the sound played; 'Drempels', a project to allow more customisation of a Windows desktop; and 'Milkdrop', another Winamp plugin which has a similar function to Geiss. All of these visualisations display their images based on the current media file; it could be seen as displaying a moving image, uniquely based on that media file.

2.5 Fractals

When discussing computer visualisations in relation to their visual appeal, one of the most fascinating areas is that of fractals. In terms of computer graphics, the various geometric methods used to describe an object become a factor. Euclidean geometry for example describes cuboids, pyramids, etc. but does not offer an option for natural objects. "… natural objects, such as mountains and clouds, have irregular or fragmented features, and Euclidean methods do not provide realistic representations for such objects" [HB04].

DNA in its very essence is a natural object and one of the alternative ways of displaying or applying its data might be through the use of fractal-geometry methods. A good way to describe how fractals work is to use the example of a mountain. When viewed from a distance, a mountain can be seen to have a jagged edge. If we were to *zoom in* on the mountain, a closer view would reveal that the mountain has a more detailed jagged edge. This zooming can continue indefinitely, each zoom revealing more detail, whilst still retaining the detail obtained from previous zooms. This all seems rather complicated, but the visual results obtained from fractals, are quite simply stunning.

Fractals can be obtained by applying a function on an initial point. We then repeat the function on that point, each iteration revealing more detail. Depending on the function used, a number of types of fractal can be obtained; *self-similar fractals*, fractals which are made up of varying sizes of the original object, often used to model vegetation, such as fern leaves or trees; *self-affine* fractals which apply various scaling factors to various parts of the image with the ability to add random factors, often used to model terrain, mountains, clouds and so on; and *invariant fractals*, created using self-squaring functions, often giving the most artistic, 'eye-pleasing' results [HB04].

The well-known fractal image is the one identified by Benoit Mandelbrot after researching the effect of a modified, self-squaring, z^2 transformation on a set of points [HB04]. A series of points were found to not be affected by the function; these points and their relative positions are now known as the Mandelbrot set. By applying various colouring techniques and animation to these points, it is possible to zoom in on part of the detail of the set and keep zooming ad infinitum – no detail will be lost and after a continued zoom period, the user gets the sensation that the image is simply repeating itself.

2.6 Summary of review

Having reviewed the literature available on the above topics, a number of facts have become apparent. Whilst the area of DNA visualisation has been explored to some extent, there are still a range of options and possibilities which have not been investigated. It is time that the area of DNA visualisation was merged with some other visualisation areas, in order to create newer ways to view complex data. By merging some techniques from above – the 2-dimensional scatter graph view for example, with the theory behind music visualisation and fractals, we could create an ever-changing image, representing the information in the DNA. This could be a new and exciting way of viewing DNA, and offer the chance to show interesting results.

3. Development of the DNA visualisation algorithm

An algorithm, called DndisplA, has been designed and implemented. In order to understand how the DNdisplA algorithm works, it is first necessary to understand how each DNA string is manipulated.

Within the algorithm, a DNA string is given a width and height value when it is created. As well as defining the output resolution of the final visual display, these values also define how the DNA string will be stored. The DNA string itself is read and split into individual characters (a DNAChar). The individual DNAChars are added to a 1-dimensional array, representing a line of the string (a DNALine). Each array has a maximum dimension equal to that of the string width. Finally, an array is declared to hold a number of DNALines, with a maximum dimension equal to the height of the DNA string (a DNAString).

Each DNAChar is stored with a number of properties. Each character is given a colour value, pre-determined by the character which the DNAChar represents ("A" = Red, "G" = Green, and so on). Each character is also given a colour weight value, again, pre-determined according to the character represented. Finally, each DNAChar is assigned a position. Once a DNAChar is created, it is added to a DNALine, and its position updated to reflect its location within the DNALine array.

DNAChars also house several methods which can be used to affect changes on the characters colour, or return an integer value. An important method, is that to apply the colour weighting to the DNAChar. As previously stated, each DNAChar is defined with a colour weight; by passing

this colour weight to another DNAChar, we can discreetly adjust the colour value by adding the colour weight value to it. It is also possible to return an integer value which represents the character itself ("A" = 1, "G" = 2, and so on).

The appropriate number of DNAChars is added to a single DNALine. Each DNALine has a number of properties and methods, which, once again, can be used to affect changes on the line. The first, and perhaps most interesting method, is that to generate a search string from the line. As you may recall, each DNAChar can invoke a method to return an integer value. By using, for example, the first five DNAChars in each DNALine, we can return a string of variable length from a position within that line, based on the integer equivalents of the first five DNAChars. This string of a variable length can then be stored within the DNALine object for later use. Obviously, different DNALines will generate different strings, depending on the characters and their integer values used to generate it.

Another interesting method within the DNALine object, is the method to translate the DNALine data, using which we can translate the individual DNAChars along the DNALine, wrapping the DNAChars to the opposite end of the array as required. When DNAChars are translated, their position values are updated to reflect their new position within the DNALine. Finally, each DNALine can also invoke a method to find an instance of a string within the DNALine. All of these methods play a great role in the sorting method, which is found in the DNAString object explained below.

The DNAString object is used to house the DNALines which make up the string. This object houses the main sorting method used to adjust the string. Each time the method is called, it is passed an iteration value, which is equal to the number of times the method has been called before. The process carried out is quite complicated, but subsequent iterations of the algorithm will always apply new changes to the string. Every time the algorithm is applied, the positions of the DNAChars within the string are changed. This in turn will ensure that the next time the algorithm is applied, new search strings will be generated by each DNALine, in turn resulting in a different resultant string. The process the algorithm carries out is outlined as follows (Figure 1).

```
for each (DNALine in DNAString)
        //First we search the current line for the search string from the next line
        //If we find a match, translate data so the matching strings line up
        get current_line
        get next_line  //based on iteration value
        get next_line.search_string
        look for next_line.search_string on current_line
        if(matchfound)
                current_line.translate_data()

        //Next apply colour weighting to each DNAChar. Use the colour weights from
        //next_line's DNAChars
        for each (DNAChar in current_line)
                get current_char
                get next_line_char
                apply current_char.colour_weight using next_line_char
                store result
```

Figure 1: *Pseudo code of the algorithm for DNA visualisation*

For example, assume we are given a small DNA string of 500 characters. We allocate the DNA string a width and height value (100x5), before reading the individual characters into DNAChars and adding them to a total of 5 DNALines of length 100. These DNALines are then added to the final DNAString object. An example DNALine may be as follows.

DNAChar: G T T A C A A C T G G T A G A G C C
T G G G A A G C T T C G A …

Position: 1 2 3 4 5 6 7 8 9 …

We can now generate a search string using this DNALine. For the purpose of this example, we can assume that the character values are as follows: A = 1, C = 2, G = 3 and T = 4. By using, for example, the first three characters, we can generate a position within the line

A + C + T = 1 + 2 + 4 = 7

We then use, for example, the next 4 characters to generate a length for the search string

A + G + T + T = 1 + 3 + 4 + 4 = 12

We can now extract the search string from the DNA line using the position value along with the length value (starting from position 7, with length 12)

search_string = "A C T G G T A G A G C C"

This search string can now be used to search through other DNALines for appropriate matches. Where matches are found, the data can be translated to line up with other instances of the string. Once the search and translation has been applied, the colour weights can be adjusted throughout the string, before re-applying the algorithm on the next line. As each line is searched and translated, the characters are constantly switching positions. This results in a new search string being generated each time the search string method is applied.

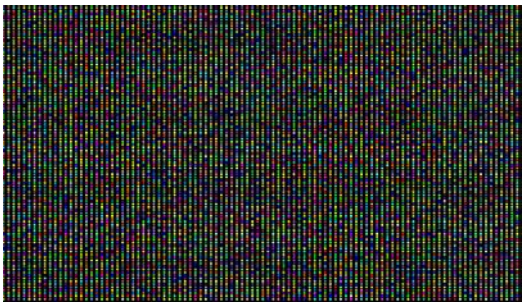
4. Testing of the DNA visualisation program

Due to the inherent randomness of DNA, it would be impossible to run all test cases of DNA with the program. Also, as little work has been done in this area, there are no test cases with results can be compared. A number of tests are run to ensure that the program runs correctly and produces images unique to a particular string. The same images should also be produced when the string is reloaded and the algorithm reapplied. Testing is also carried out to look for interesting behaviours from the algorithm with certain preset files.

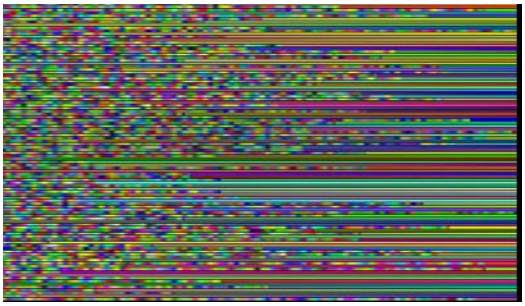
The tests are conducted on a Pentium 2.6 GHz PC with 1 Gb RAM and an NVIDIA GeForce 5200FX Graphics card. The test data sets are generated applying certain rules. Test files are created and used to validate the program. The test runs and their results are presented below.

4.1 Test on DNA string with random data (100x100)

A 100x100 random data set (TEST_100x100_Randon.dna) is created by assigning the values (e.g. 1, 2, 3, 4 for A, C, G, T respectively) randomly to the data items in the DNA string. In this test, the same DNA string is loaded into the application twice, and the same number of algorithm iterations are pre-applied. In each case, four images are generated - the first two images are represented in point and line after 100 iterations, and the last two in polygon after 100 and 217 iterations, respectively (Figure 2). The images can be compared with their counterparts to ensure that the same string produces the same output each time.

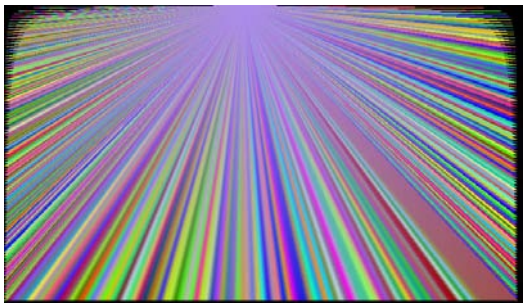


a) *After 100 iterations: With points as the draw style, no noticeable patterns or areas of interest are apparent*

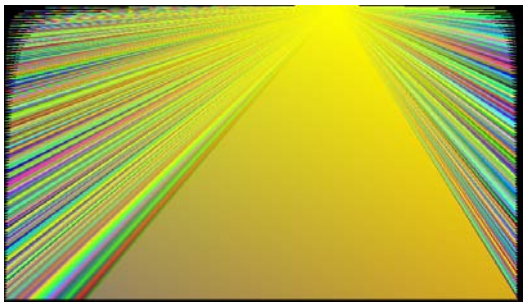


b) *After 100 iterations: With linestrip as the draw style, prominent bands of colour are seen on the right hand side*

This test has proved successful; the same image was produced by the application in all cases. In these tests, the program is closed and reopened before the second run, ensuring that it is initialised before each run, showing that the same file can be reloaded to produce the same output.



c) *After 100 iterations: Polygon draw style displays a bright wheel of colours, with a slight purple tint*

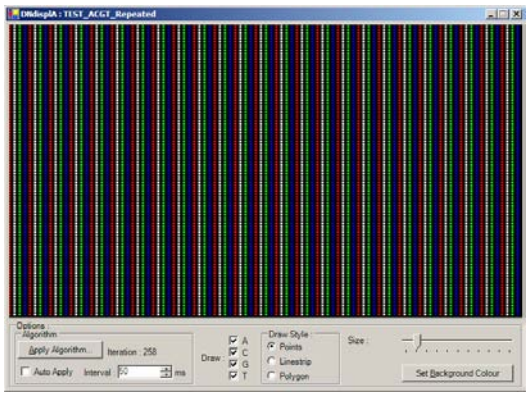


d) *After 217 iterations: Image with a large yellow triangle*

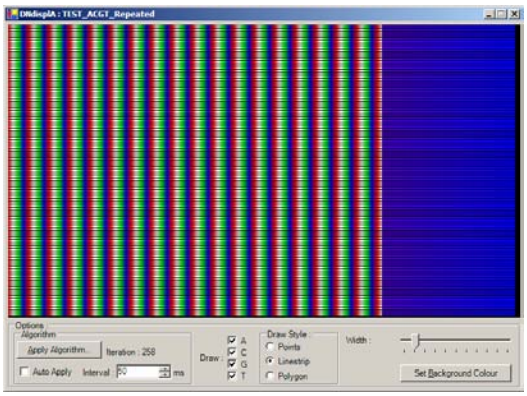
Figure 2: *Test on DNA string with random data (100x100)*

4.2 Test on DNA string with repeated 'ACGT' data

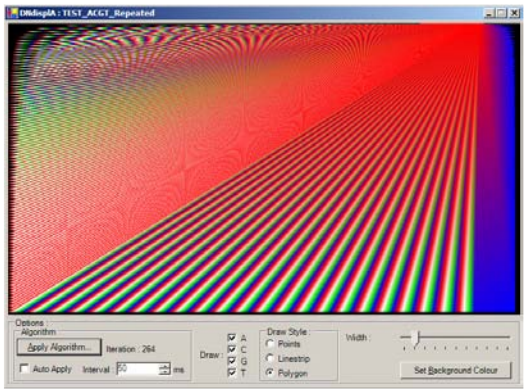
The file (TEST_100x100_ACGT_Repeated.dna) consists of the DNA string 'ACGT' repeated. The test produced predictable output. Four colours are displayed, which change during the first 15 iterations, then remain the same throughout any other iterations. These colours seem to move across the linestrip (Figure 3a) and polygon (Figure 3b) displays, but no animation is apparent in the point display (Figure 3c).



a) *Point draw*



b) *Linestrip draw*



c) *Polygon draw*

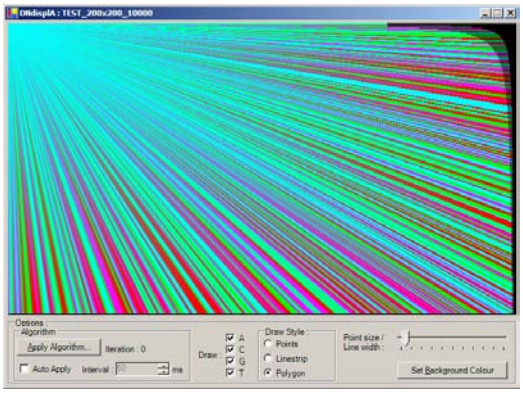
Figure 3: *Test on DNA string with random data (100x100)*

4.3 Test on DNA string with random data (200x200)

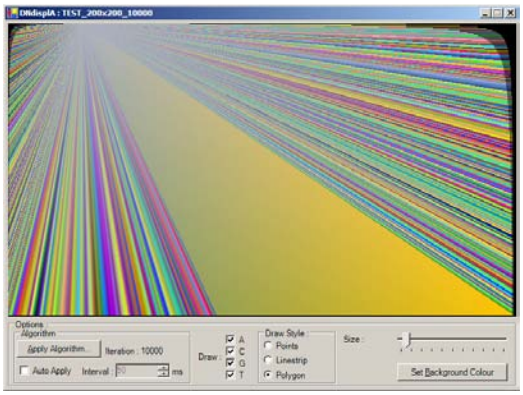
This test generates a random DNA string, with a length of 40000 (TEST_200x200_40000.dna). The algorithm is applied on the string 10000 times, and the display saved. Then another 10000 iterations is applied and the results compared with those of the first 10000 iterations. This test has proved interesting. With the polygon draw style selected, after no iterations, a simple colour wheel is displayed (Figure 4a). After the first run of 10000 iterations, an image is produced with a noticeable yellow triangle towards the centre of the image (Figure 4b). After 20000 iterations, exactly the same image is produced (Figure 4c). It would appear that the algorithm can only be run a certain number of times before the same images will be displayed.

As can be seen in Figure 5, running the algorithm and adjusting the display settings (such as hiding bases or changing the background colour) can produce some interesting results. By hiding bases (Figures 5a and 5b), areas of interest may be noted – this gives the notion that the software could be used to compare two strings. As has been proved, the algorithm gives unique results for every unique string – assuming we have two identical strings, we can use display adjustments to help in identifying possible matches. It is also possible to adjust the background colour (Figure 5c), giving the user an alternative to the default black background. Tests are also carried out to see how the algorithm works as an animation. The results for the linestrip and polygon animations are less attractive than the points drawing, though this is expected. As the algorithm searches through the string for patterns to match, it is invariably moving some data with each iteration. This movement is enough to cause a 'jumping' effect – there is just too much movement for the animation to be considered fluid. Perhaps some form of tweening (morphing between images) between algorithm iterations could allow these two drawing styles to work as an animation. However, in juxtaposition to this, the points drawing style has proved to be quite acceptable as an animation. Although the data in the DNA string is still being moved around by the algorithm, only the colour values are changing (using the colour weight variables), so the data does not appear to be moving, rather, the colours seem to blend between each other.

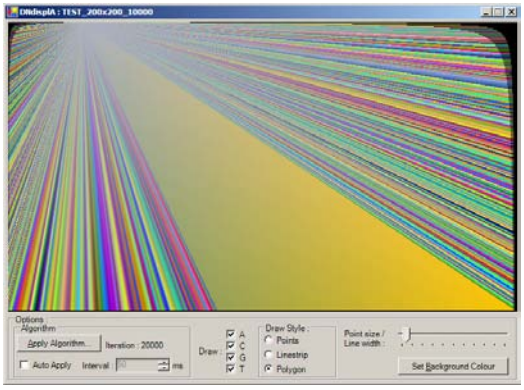
These tests of the animation also lead to the discovery of what is perhaps a potential problem. When dealing with a small resolution string, say 100x100 pixels, the program is able to process the algorithm quick enough to display a smoothly moving image. However, as the resolution increases, the processing time increases exponentially. A string with a resolution of 200x200 pixels takes considerably longer to process, and on the computer used, is not able to provide a smooth animation.



a) *Prior to iterations*



b) *After 10000 iterations*



c) *After 20000 iterations*

Figure 4: *Test on DNA string with random data (200x200)*

Whilst these tests are only carried out on a number of randomly generated or preset DNA files, it would be more interesting to open a genuine DNA string to see any effect the algorithm may have on the display. It is also important to remember that the still figures shown above are static images, captured during the running of the algorithm. By running the program, it is possibleto appreciate the effect of the algorithm over time, particularly the point draw animation, which produces pleasing displays with flowing colours.

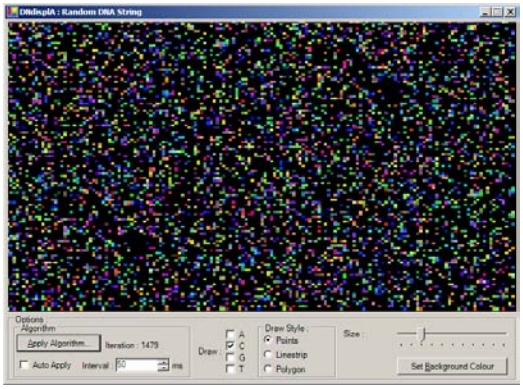
5. Concluding remarks

The area of DNA visualisation is an interesting and growing field. In the 50 or so years since Watson and Crick completed Franklins work on the structure of DNA, technology has advanced in such a way that the structure of DNA should be further examined, in *every* way possible. As time goes on and more and more is learned about the various chromosomes and positions of base pairs within DNA, is it not possible that eventually a program could be developed to examine some DNA and report any illnesses or features which may be present? Could a program take a DNA string, provide a realistic interpretation of the organism, age the organism and display the result as a dynamic 3-dimensional model? This research has made an initial attempt and achieved some results.

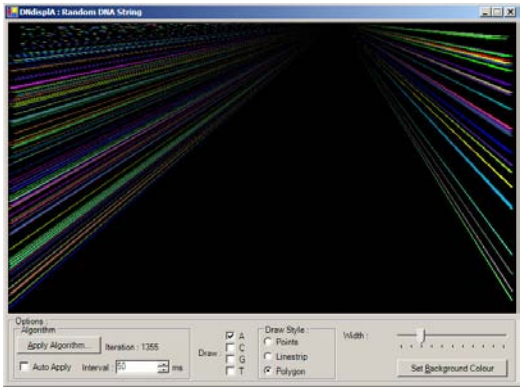
It would be inappropriate to compare the results of the program with other DNA visualisation techniques since DNdisplA has been designed to create aesthetic images from DNA data, with the option of viewing the results as a dynamic moving image, and other techniques focus on creating a static image.

Whilst working on this research, it has become apparent that there are a number of aspects could be extended or worked on further.

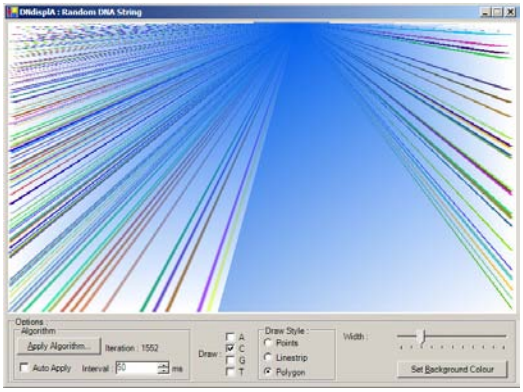
The algorithm that has been implemented, focuses on the colour and colour weighting of each character, but does not make any changes to a characters position. If the algorithm were to apply a weight to each character position, we could see the characters move across the screen. As well as moving characters around the display, it could be possible to implement a 3-dimensional display, where certain characters can be moved closer to or further from the viewer. The work carried out by Herrison and Gherbi [HG01] investigated the 3-dimensional trajectory of DNA sequences; perhaps by combining this work with the algorithm, a display could be created which will show the DNA sequence in a new manner.



a) *Point draw with hidden bases: certain features noted - the hole towards the top right*



b) *Polygon draw by hiding bases*



c) *Polygon draw: Changing the background colour resulting in different image*

Figure 5: *Tests for animation of DNA string*

The program also leads itself to a number of further developments. The three drawing styles, point, line and polygon, all make use of the DNA as a 2-dimensional array, displayed on screen from left to right and top to bottom. Perhaps the display routine could be changed to draw the DNA characters in a different order, depending on the iteration. Whilst the point draw style would exhibit no changes, the line and polygon draw styles could produce unique images.

Another area which could be addressed is the initial colours given to characters – for example, the 'G' character is given a green colour. Perhaps the program could incorporate the entire DNA string when deciding on a particular character colour (say, its position in the string, or its relation to another character). Whilst the algorithm does examine the DNA string to generate a search string, it only uses the first five characters of each line; a possible change could be to incorporate the entire line, or the entire string when generating a search strings.

Other drawing styles could be introduced in addition to the three styles used in the program, such as 'quads' or a 'triangle strip'. The final display could also be adjusted – drawing with points leads to a sharp and crisp image, a sort of anti-aliasing method could be introduced to try and blend the colours together across the whole display.

References

[GM70] Gibbs A. J., McIntyre G. A.: The diagram, a Method for comparing sequences. Its use with amino acid and nucleotide sequences. *Eur. J. Biochemistry* (Sept. 1970), vol. 16, no. 1, pp. 1-11.

[HB04] Hearn D., Baker M.: *Computer Graphics with OpenGL.* 3rd Edition, Prentice Hall (2004).

[HG01] Herisson J., Gherbi R.: Model based prediction of the 3D trajectory of huge DNA sequences. In *Proc. 2nd IEEE International Symposium on Bioinformatics and Bioengineering (BIBE'01)*, March 2001, pp. 263-270.

[Hum05] The Human Genome Project: http://www.genome.gov/10001772/

[Pot96] Potel M. J.: Computer Graphics and DNA sequencing. *IEEE Computer Graphics and Applications* (Nov 1996), vol. 16, no. 6, pp. 14-19.

[RJ05] Raven P. H., Johnson G. B.: *Biology*. 7th Edition, McGraw Hill (2005), pp. 284-287.

[Rid00] Ridley M.: *Genome*. Harper Collins (2000), p. 6.

[Rya05] Ryan Geiss homepage: http://www.geisswerks.com/

[TAO05] The TAO .NET Framework: http://www.taoframework.com/

[Win05] Winamp Media Player: http://www.winamp.com/

[WWFT03] Wong P. C., Wong K. K., Foote H., Thomas J.: Global visualization and alignments of whole bacterial Genomes *IEEE Transactions on Visualization and Computer Graphics* (July 2003), pp. 365,